PROCESO DE GESTIÓN DE FORMACIÓN PROFESIONAL INTEGRAL

FORMATO GUÍA DE APRENDIZAJE

1. IDENTIFICACIÓN DE LA GUIA DE APRENDIZAJE

- Denominación del Programa de Formación: Análisis y desarrollo de software
- Código del Programa de Formación: 228118
- Nombre del Proyecto Formativo : construcción de software integrador de tecnologías orientadas a servicios.
- Fase del Proyecto: Ejecución
- Actividad de Proyecto Formativo: Codificar los módulos del software
- Competencia: Desarrollar la solución de software de acuerdo con el diseño y metodologías de desarrollo
- Resultados de Aprendizaje: Codificar el software de acuerdo con el diseño establecido
- Duración de la Guía de Aprendizaje (horas): 340 Horas

2. PRESENTACIÓN

Estimado aprendiz, el SENA extiende una cordial bienvenida a la octava guía de aprendizaje que comprende la competencia técnica de: desarrollar la solución de software de acuerdo con el diseño y metodologías de desarrollo.

En palabras de Avison y Fitzgerald (1995), una metodología es una serie de procedimientos y herramientas que ayudan a los desarrolladores a implementar nuevos sistemas de información y están formadas por fases que guían a los desarrolladores a elegir las técnicas más apropiadas en cada momento del proyecto.

Para el desarrollo de las actividades planteadas en esta guía, contará con el acompañamiento de los instructores asignados al programa, los cuales de forma continua y permanente lo orientarán con las pautas necesarias para el logro de las actividades de aprendizaje, brindando herramientas básicas de tipo conceptual y metodológico. Los instructores programarán encuentros de asesoría virtual, para brindar orientaciones específicas relacionadas con las temáticas a desarrollar en las actividades. Es importante que organice su tiempo, dada la exigencia que demanda la realización de esta guía de aprendizaje. No olvide revisar y explorar los materiales de estudio del programa.

Por consiguiente, se presentan cada una de las acciones de aprendizaje que le permitirán desarrollar lo anteriormente mencionado.

3. FORMULACIÓN DE LAS ACTIVIDADES DE APRENDIZAJE

En este apartado se describirán las actividades de aprendizaje para cada una de las competencias que plantea la fase de ejecución del proyecto formativo: construcción de software integrador de tecnologías orientadas a servicios.

3.1 Actividades de reflexión inicial:

En grupos de 5 personas genero una discusión respondiendo a las preguntas ¿Por qué las aplicaciones móviles son importantes para la sociedad?, ¿Qué beneficio puedo obtener al convertirme en desarrollador de apps?, ¿Cuáles son las aplicaciones móviles más conocidas y que más se usan en la actualidad?, ¿Qué tipo de aplicación movil mejoraría mi vida? Elaboro una presentación en Presentación en PowerPoint con las respuestas seleccionadas y las comparto con los demás grupos.

- Ambiente Requerido: ambiente de formación con computadores
- Materiales: Guía de aprendizaje, equipos con sistema operativo Windows y conectividad a internet.
- Duracion: 2 horas

3.2 Actividades de contextualización e identificación de conocimientos necesarios para el aprendizaje:

Actividad Individual:

- Leo detenidamente el documento “Casos de Éxito Aplicaciones moviles.pdf” este documento contiene una serie de artículos sobre generalidades de las apps para dispositivos móviles, cuáles serían las características de una aplicación exitosa, y la descripción de dos casos de éxito, Angry Birds y whatsaap. El documento se encuentra en la carpeta de la fase Ejecución en la plataforma, además buscar en internet información que profundice o brinde mayor información sobre los temas leídos”

- Respondo las siguientes preguntas en el Foro que se encuentra en la plataforma.

    - ¿Qué fue lo que más le llamó la atención de los diferentes artículos leidos?
    - ¿Cuales son las características fundamentales que debe tener una aplicación exitosa?

- ¿Por qué es importante incursionar eln el mundo del desarrollo de aplicaciones y videojuegos para dispositivos móviles?
- ¿Qué aplicación podría crear que solucione un problema real actual?
- ¿Qué puede aprender del caso de Rovio?
- ¿Qué puedo aprender del caso de Whatsapp?

- Participo activamente en la lluvia de ideas dirigida por el instructor.
- Leo el documento “Temario Desarrollo de Aplicaciones en la plataforma Android.pdf”
- Participo en la explicación del instructor y en la sesión de preguntas sobre el temario.

Ambiente requerido: ambiente de formación con computadores

Materiales de formación: Guía de aprendizaje, equipos con sistema operativo Windows y conectividad a internet.

Material de apoyo: Documento Temario Desarrollo de Aplicaciones en la plataforma Android.pdf”

Duración de la actividad: 2 horas.

3.3 Actividades de apropiación:

Esta actividad se propone para el desarrollo de los diferentes aspectos claves acerca de la elaboración de aplicaciones móviles para Android.

Elementos para tener en el desarrollo del taller:

Taller

Elabore una investigación corta usando los materiales disponibles en la biblioteca o internet acerca del desarrollo de aplicaciones móviles para Android.

● ¿Qué es Android?

● Definición del concepto de APK.

● ¿Qué es el Android SDK?

● ¿Cuál es el lenguaje utilizado para desarrollar aplicaciones para Android?

● ¿Qué IDEs de desarrollo existen para codificar?

- Definir el concepto de Android multiusuario realizar una breve descripción de este concepto
- Definir el concepto de mínimo privilegio
- ¿Cuáles son los componentes esenciales de una aplicación Android?

Ambiente requerido: ambiente de formación con computadores

Materiales de formación: Guía de aprendizaje, equipos con sistema operativo Windows y conectividad a internet.

Material de apoyo: Documento Temario Desarrollo de Aplicaciones en la plataforma Android.pdf"

Duración de la actividad: 6 horas.

3.4 Actividades de Transferencia el Conocimiento:

3.4.1 Concepotos básicos Kotlin

- En esta actividad los aprendices realizaran los codelabs planteados en el material "Concepotos Basicos Kotlin".

Ambiente requerido: ambiente de formación con computadores

Materiales de formación: Guía de aprendizaje, equipos con sistema operativo Windows y conectividad a internet

Material de apoyo: Link "Concepotos Basicos Kotlin"

Duración de la actividad: 18 horas.

3.4.2 Actividad: Activities e Interfaces

• En esta actividad los aprendices realizarán ejercicios prácticos sobre el empleo y uso de las activities o interfaces de usuario. Atiendo a la demostración de método realizada por el instructor sobre cada uno de los temas identificados en el temario de la actividad.Para tal actividad consulte los medios didácticos entregados por el instructor.

Para tal actividad consulte el siguiente enlace:

https://developer.android.com/guide/components/activities/?hl=es-419

• Ambiente Requerido: ambiente de formación con computadores

• Materiales: Guía de aprendizaje No 1, documento de apoyo: Activities e Interfaces en formato PDF

(Carpeta de material de apoyo), equipos con sistema operativo Windows y conectividad a internet.

3.4.2 Actividad: Activities e Intents

• En esta actividad los aprendices realizarán ejercicios prácticos sobre el empleo y uso de las activities e intents. Atiendo a la demostración de método realizada por el instructor sobre cada uno de los temas identificados en el temario de la actividad.Para tal actividad consulte los medios didácticos entregados por el instructor.Para tal actividad consulte el siguiente enlace: https://developer.android.com/guide/components/intents- filters?hl=es-419

• Ambiente Requerido: ambiente de formación con computadores

• Materiales: Guía de aprendizaje No 1, documento de apoyo: Intents en formato PDF (Carpeta de material de apoyo), equipos con sistema operativo Windows y conectividad a internet.

3.4.3 Actividad: Componentes Interfaz Gráfica

• En esta actividad los aprendices realizarán ejercicios prácticos sobre el uso de los diferentes componentes de la interfaz gráfica empleados en Android. Atiendo a la demostración de método realizada por el instructor sobre cada uno de los temas identificados en el temario de la actividad.Para tal actividad consulte los medios didácticos entregados por el instructor.Para tal actividad consulte los medios didácticos entregados por el instructor. Para tal actividad consulte el siguiente enlace: https://developer.android.com/guide/topics/ui/?hl=es-419

• Materiales: Guía de aprendizaje No 1, documento de apoyo: Componentes Interfaz Gráfica formato PDF (Carpeta de material de apoyo), equipos con sistema operativo Windows y conectividad Internet

3.4.4 Actividad: Activities – Views

• En esta actividad los aprendices realizarán ejercicios prácticos sobre el uso de los diferentes componentes de la interfaz gráfica empleados en Android. Atiendo a la demostración de método realizada por el instructor sobre cada uno de los temas identificados en el temario de la actividad.Para tal actividad consulte los medios didácticos entregados por el instructor.Para tal actividad consulte los medios didácticos entregados por el instructor. Para tal actividad consulte el siguiente enlace:
https://www.movilzona.es/tutoriales/android/desarrollo/curso-de-desarrollo-android-tema-views-tipos-de-views-mas-habituales-y-sus-propiedades/

• Materiales: Guía de aprendizaje No 1, documento de apoyo: Views en Android en formato PDF (Carpeta de material de apoyo), equipos con sistema operativo Windows y conectividad a internet.

3.4.5 Actividad: Layouts - ImageViews – OnClickListener

• En esta actividad los aprendices realizarán ejercicios prácticos sobre el uso de los diferentes tipos de layouts en Android. Atiendo a la demostración de método realizada por el instructor sobre cada uno de los temas identificados en el temario de la actividad.Para tal actividad consulte los medios didácticos entregados por el instructor. Para tal actividad consulte los medios didácticos entregados por el instructor. Para tal actividad consulte el siguiente enlace: http://www.hermosaprogramacion.com/2014/09/android-layouts-views/

• Materiales: Guía de aprendizaje No 1, documento de apoyo: Layouts en Android en formato PDF (Carpeta de material de apoyo), equipos con sistema operativo Windows y conectividad a internet.

3.4.6 Actividad: Proyecto Final

En esta actividad los aprendices realizarán un proyecto final empleando todos los conceptos vistos durante el desarrollo de la formación del lenguaje de programación en Android. Para tal actividad consulte los medios didácticos entregados por el instructor Tomando como referencia la planeación pedagógica y las orientaciones para elaborar guías de aprendizaje citado en la guía de desarrollo curricular

Materiales de formación: : Guía de aprendizaje, equipos con sistema operativo Windows y conectividad a internet.

Material de apoyo:

Tutorial 1: Como hacer un App en android desde cero

Tutorial 2: Como actualizar el SDK de android (SDK Manager)

Tutorial 3: Como crear un emulador en Android (AVD Manager)

Tutorial 4 Como Instalar y Configurar los emuladores de Genymotion

Tutorial 5: Como Utilizar mi Celular para emular mis aplicaciones Android

Tutorial 6: Como crear la interfaz gráfica de mi aplicación Android (LinearLayout)

Tutorial 7: Propiedad Weight en LinearLayout

Tutorial 8: Como crear la interfaz grafica de la aplicacion android - RelativeLayout

Tutorial 9: Como agregar un scroll a la interfaz de mi aplicacion Android - ScrollView

Tutorial 10: Como hacer una App en Android -onClickListener funcion del boton

Tutorial 11: Como usar RadioButton y CheckBox en Android

Tutorial 12 Como usar un Spinner en mi aplicacion Android

Tutorial 13 Como usar un ListView en mi Aplicacion Android

Tutorial 13 1 ListView con ArrayList

Tutorial 14: Como cambiar entre pantallas en Android (Navegacion entre Activities)

Tutorial 15: Como pasar Informacion de un Activity a otro en Android

Tutorial 16: Como Animar un boton personalizado en android

Tutorial 17: Como usar fragments en Android

Tutorial 18: Como usar Fragments Dinámicos

Tutorial 19 Como agregar sonido a un boton en Android

Tutorial 20: Como hacer un SplashScreen o pantalla de inicio en Android

Tutorial 21: Swipe Views Como Hacer Un ViewPager En Android

Tutorial 22: Como implementar un Navigation Drawer en Android Studio

Tutorial 23: Como Guardar datos en Android - SharedPreferences

Tutorial 24: SQLite en Android- Interfaz de Usuario

Tutorial 25: SQLite en Android 2- Como crear la tabla e insertar datos

Tutorial 26: SQLite en Android 3 -Como buscar, eliminar y actualizar datos en la tabla

Tutorial 27: Como conectarse a una base de datos MySQL con la Aplicacion Android

Tutorial 28: Como conectar a una base de datos MySQL con la aplicación Android

Tutorial 29: Como conectar a una base de datos MySQL con la aplicación Android

Tutorial 30: Google Maps

Tutorial 31: Volley en Android Studio

Evidencias de aprendizaje: APK de la app del proyectyo con el producto mínimo viable definid por el instructor.

Duración de la actividad: 120 horas.

4. PLANTEAMIENTO DE EVIDENCIAS DE APRENDIZAJE PARA LA EVALUACIÓN EN EL PROCESO FORMATIVO.

Fase del proyecto formativo	Actividad del proyecto formativo	Actividad de Aprendizaje	Evidencias de Aprendizaje	Criterios de Evaluación	Técnicas e Instrumentos de Evaluación
				Los criterios de evaluación se encuentran en el formato instrumento de evaluación para consultar estos criterios viste la plataforma Territorium Contenido del Curso/ III Trimestre 2024/ Programación Dispositivos Móviles / Instrumentos de Evaluación Por cada una de las actividades propuestas hay un instrumento de evaluación	

5. GLOSARIO DE TÉRMINOS

ADB y ADB shell, Android Depuration Bridge o Puente de Depuración de Android. Es la

herramienta utilizada para enviar comandos desde un ordenador a un dispositivo con este sistema. El shell es un programa Terminal que hace de traductor de nuestras órdenes, actúa sobre una línea de comandos.

• AOSP, Android Open Source Project o Proyecto de Código Abierto Android. Las propias siglas ya lo indican, Android es un sistema de código abierto, lo que quiere decir que cualquiera puede descargar su código fuente y modificarlo a su gusto. Este detalle precisamente es el que hace que Android sea un grande entre los grandes, consiguiendo diferenciarse así de su competidor más directo, iOS de Apple, que se trata de un sistema cerrado.

• APK, Android Package Archive o Paquete de Archivo de Android, es la extensión de los paquetes de los programas que operan en este sistema. Pueden ser abiertos y modificados fácilmente.

• APN, Access Point Name o Nombre del Punto de Acceso, son los datos para poder acceder a las redes 2G, 3G para el uso de datos e internet, y además hacer uso de mensajes multimedia MMS desde el móvil que tiene cada operadora de forma única. En la mayoría de los casos estos datos vienen ya configurados.

• APP, abreviatura de la palabra inglesa application, que quiere decir aplicación.

• ARM, Advanced RISC Machines o Máquinas RISC Avanzadas, es una familia de microprocesadores RISC diseñados por la empresa Acom Computers. Son microprocesadores muy usados en los dispositivos móviles gracias a su gran característica, su bajo consumo.

• OTA, Over The Air o Sobre El Aire, es la acción de descargar datos a través del aire. Las actualizaciones oficiales de Android normalmente se realizan mediante OTA, al igual que la instalación de aplicaciones y sus actualizaciones.

• SDK, Software Development Kit o Kit de Desarrollo de Software, es un conjunto de herramientas de desarrollo que permite al programador crear aplicaciones para un sistema, en este caso Android.

• BOOT, es el proceso que se realiza en el inicio de un ordenador o dispositivo computarizado como puede ser un móvil Android. Su función es la de cargar el programa de arranque que carga toda la configuración del hardware para un correcto funcionamiento.

• Bootloader, es el boot encargado del arranque en los dispositivos móviles.

• Brick o semibrick, también conocida como pantalla de la muerte, es la consecuencia de cargar una ROM de forma errónea. En casi la totalidad de los casos es recuperable.

• Class 2, 4, 6, 10, son las velocidades de las tarjetas SD usadas por los dispositivos móviles, las clases 2 trabajan a 2Mb, la clase 4 a 4Mb, y así sucesivamente.

• Dalvik, es una máquina virtual encargada de la ejecución de las aplicaciones en la versión 2.1 y anteriores de Android.

• Fastboot, es un bootloader con el que podemos hacer flasheos de bajo nivel.

• Hard Reset o Factory Reset, es el proceso de dejar el teléfono tal y como estaba justo al momento de comprarlo como de fábrica, o bien para recuperarlo a como esta ajusto después de haber instalado una rom. HardSPL, otro bootloader, en este caso ha sido modificado para dejarnos cargar una ROM a nuestra preferencia.

• Kernel, es el núcleo de los sistemas que se encarga de realizar las operaciones relativas a la gestión del sistema operativo (gestión de memoria, planificador, etc.)

• Launcher, es la parte encargada de la interfaz gráfica de usuario de Android, es totalmente personalizable como ya hemos visto.

• Wipe, del verbo limpiar en inglés, es lo que comúnmente llamamos a "formatear" en el ámbito de los ordenadores, hacer wipe implica la pérdida de los datos almacenados

6. REFERENTES BILBIOGRÁFICOS

- **https://desarrolloweb.com/articulos/introduccion-android.html**
- **http://androidzone.org/android-para-principiantes/**
- **http://androidzone.org/diccionario-android/**
- **https://www.youtube.com/watch?v=Be5yKKIWVrE**
- **http://www.sgoliver.net/blog/curso-de-programacion-android/indice-de-contenidos/**
- **https://www.tutorialspoint.com/android/**
- **http://www.hermosaprogramacion.com/android/**

7. CONTROL DEL DOCUMENTO

	Nombre	Cargo	Dependencia	Fecha
Autor (es)				

8. CONTROL DE CAMBIOS (diligenciar únicamente si realiza ajustes a la guía)

	Nombre	Cargo	Dependencia	Fecha	Razón del Cambio
Autor (es)					



PROCESO DE GESTIÓN DE FORMACIÓN PROFESIONAL INTEGRAL

FORMATO GUÍA DE APRENDIZAJE

1. IDENTIFICACIÓN DE LA GUIA DE APRENDIZAJE

- Denominación del Programa de Formación: Análisis y desarrollo de software.
- Código del Programa de Formación: 228118
- Nombre del Proyecto: Construcción de software integrador de tecnologías orientadas a servicios.
- Fase del Proyecto: Ejecución
- Actividad de Proyecto: Codificar los módulos del software.
- Competencia: Evaluar requisitos de la solución de software de acuerdo con metodologías de análisis y estándares.
- Resultados de Aprendizaje Alcanzar: Desarrollar procesos lógicos a través de la implementación de algoritmos.
- Duración de la Guía: 80 horas

  Formación directa: 60 horas

  Trabajo Autónomo: 20 Horas

2. PRESENTACIÓN

En el mundo actual, donde la información y la tecnología son fundamentales, la capacidad de desarrollar procesos lógicos eficientes es crucial. Los algoritmos, como herramientas para resolver problemas, juegan un papel fundamental en este proceso. En esta presentación, exploraremos cómo desarrollar procesos lógicos a través de la implementación de algoritmos y cómo esta habilidad puede ser beneficiosa en diversos campos.

Esta guía le permitirá al aprendiz introducirse en el mundo de la programación a través de una serie de conocimientos que les darán las herramientas necesarias para obtener los procesos mentales adecuados para aprender a programar. Es muy importante que los aprendices realicen a conciencia las actividades propuestas para que puedan adquirir los conocimientos y la lógica necesaria.

Por consiguiente, se presentan cada una de las acciones de aprendizaje que le permitirán desarrollar lo anteriormente mencionado.

3. FORMULACIÓN DE LAS ACTIVIDADES DE APRENDIZAJE

En este apartado se describirán las actividades de aprendizaje para cada una de las competencias que plantea la fase de ejecución del proyecto formativo: construcción de software integrador de tecnologías orientadas a servicios.

3.1 3.1 Actividades de reflexión inicial: Analizar los elementos básicos de la lógica y la construcción de algoritmos

Antes de iniciar las actividades propuestas, conforme grupos, máximo de 3 personas.

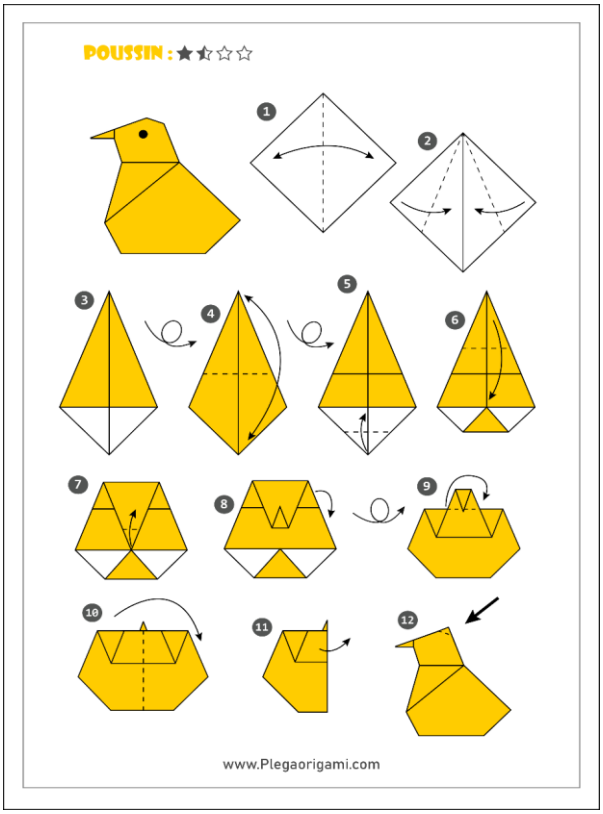
Materiales: Hojas de papel, lápices, esferos, tablero, marcadores

Detengamos el tiempo por un segundo y recordemos cómo hacemos figuras de origami. Para ello se sugiere elaborar una pollito. Sigan los pasos que se les indican a continuación:

Paso 1. Con la hoja de papel realice la figura propuesta

Paso 2. Junto a sus compañeros de grupo escriba en una hoja como sería el paso a paso para elaborar la figura de un animal en papel.

Paso 3. ¡Ahora reflexionemos! ¿Qué pasaría si cambiamos los materiales o la forma del animal? ¿Cambiaría el paso a paso



Escriban sus conclusiones:
Algo que para algunos es sencillo, puede que para otros sea difícil. En la vida diaria encontraremos una serie de actividades que requieren de pasos para su solución. ¿Lo has notado?
descrito?

3.2 Actividades de contextualización e identificación de conocimientos necesarios para el aprendizaje:

Duración: 2 horas.

Después de la explicación del instructor, me reúno en parejas y consulto información para las siguientes actividades de investigación:

Historia de la Lógica Proposicional:

Investigar y presentar una breve historia de la lógica proposicional, destacando sus orígenes y su evolución a lo largo del tiempo.

- Aplicaciones de la Lógica Proposicional en la Vida Real:

Identificar ejemplos de aplicación de la lógica proposicional en la vida cotidiana, como sistemas de seguridad, electrónica digital, inteligencia artificial, entre otros.

- Importancia de los Algoritmos:

Discutir la importancia de los algoritmos en la resolución de problemas, desde aplicaciones simples hasta complejas, y su impacto en diferentes áreas, como la informática, las finanzas y la medicina.

- **Evidencia de conocimiento:**

Elementos para tener en cuenta en el documento:

- Se deben seguir las normas básicas de presentación de un documento escrito, es decir el documento debe tener como mínimo una portada, introducción, objetivos.

Lineamientos generales para la entrega de la evidencia:

- Productos para entregar: documento tipo informe
- Extensión: Libre.
- Formato: PDF.

3.3 Actividades de apropiación:

En esta actividad usted encontrará 4 actividades de Aprendizaje, desglosadas cada una de ellas en subactividades. Desarrolle cada una de ellas, evidenciando lo propuesto.

3.3.1 Actividad de Aprendizaje 1: Identificar los conocimientos necesarios para elaborar algoritmos básicos

Siga las indicaciones que se presentan a continuación:

1. Conforme un equipo de trabajo como máximo de 4 personas.

2. Elegir a quien llevará la venda, es decir, el que hará el papel de gallinita ciega.

3. El compañero(a), una vez elegido debe ponerse un pañuelo en los ojos, de forma que no pueda ver nada.

4. Los compañeros(as) que pueden ver deberán guiar al compañero a algún lugar de la sede, indicado por el instructor.

Nota: Tenga en cuenta que su compañero no podrá ver por ende deberá guiarlo paso a paso hasta el lugar de destino.

5. Comparta con sus compañeros la experiencia de guiar y ser guiado.

6. A continuación, observe el siguiente video: que_es_un_algoritmo.mp4 https://acortar.link/ez1ieR contenido en la carpeta de material de apoyo.

7. Comparta y socialice con sus compañeros de clase si lo que hicieron al guiar al compañero de la venda, tiene algo que ver con lo visto en el video.

3.3.2 Actividad de Aprendizaje 2: Identificar los conocimientos básicos necesarios para elaborar proposiciones simples y compuestas

1. Observen el video proposiciones y operaciones básicas.mp4, que se encuentra en el material de apoyo.
2. En grupos de dos personas realicen la lectura del documento denominado **lectura_introducción_a_la_lógica**, que se encuentra en el material de apoyo.
3. Elaboren en su cuaderno 10 enunciados diferentes y traten de observar si son o no proposiciones. Clasifíquenlas en compuestas o simples al frente de cada enunciado.
4. Socialicen con los demás grupos los enunciados trabajados y los resultados obtenidos.

3.3.3 Actividad de Aprendizaje 3: Conceptos claves para resolver problemas lógicos proposicionales.

En un curso de lógica proposicional, los estudiantes enfrentan dificultades para resolver problemas complejos que involucran la construcción de tablas de verdad, la simplificación de expresiones

lógicas y la demostración de teoremas. En grupo de trabajo realizaremos un foro con las siguientes preguntas:

1. Define qué es una proposición y proporciona un ejemplo.
2. Explica la diferencia entre una proposición simple y una proposición compuesta.
3. ¿Qué es una tabla de verdad y cómo se utiliza para evaluar proposiciones compuestas?
4. Describe qué es una tautología y da un ejemplo.
5. ¿Qué es una contradicción en el contexto de la lógica proposicional? Proporciona un ejemplo.
6. Define qué es una implicación lógica y proporciona un ejemplo.
7. ¿Qué son las leyes de De Morgan en lógica proposicional? Proporciona las expresiones equivalentes.
8. ¿Cuál es la diferencia entre una implicación material y una equivalencia lógica?
9. Explica qué significa que dos proposiciones sean equivalentes en el contexto de la lógica proposicional.
10. ¿Qué son las formas normales en lógica proposicional y cuáles son sus tipos principales

3.3.4 Actividad de Aprendizaje 4: Problemas lógicos proposicionales Simboliza los siguientes enunciados

Dadas las siguientes proposiciones:

a) No vi la película, pero leí la novela
b) Ni vi la película ni leí la novela
c) No es cierto que viese la película y leyese la novela
d) Vi la película, aunque no leí la novela
e) No me gusta trasnochar ni madrugar
f) tú estás equivocado o es falsa la noticia que has leído
g) Si no estuvieras loca, no habrías venido aquí
h) Llueve y o bien nieva o sopla el viento
i) está lloviendo y nevando o está soplando el viento
j) Si hay verdadera democracia, entonces no hay detenciones arbitrarias ni otras violaciones de los derechos civiles.
k) Roberto hará el doctorado cuando y solamente cuando obtenga la licenciatura
l) Si viene en tren, llegará antes de las seis. Si viene en coche, llegará antes de las seis. Luego, tanto si viene en tren como si viene en coche, llegará antes de las seis.

Enlaza cada proposición con su formalización

“Llueve” = p, “Hace sol” = q

1	Llueve y hace sol	5A	¬ p
2	Llueve y no hace sol	3B	p ∨ q
3	Llueve o hace sol	1C	p ∧ q
4	Si no llueve, hace sol	2D	p ∧ ¬ q
5	No es cierto que llueva	6E	¬ ¬ q
6	No es cierto que no llueva	7F	q ↔ ¬p
7	Hará sol si y solo si llueve	4G	¬p → q

Construye las tablas de verdad de:

a. ¬p ∧ q
b. ¬p ∧ ¬q
c. (p ∨ ¬q) ∨ p
d. (p → q) ∧ p
e. [¬(p → q) ∨ (p ↔ q)] ∧ [(¬p → q) ∨ ¬p]

3.3.5 Actividad Aprendizaje 5: Identificar los conocimientos necesarios para elaborar algoritmos básicos

Para elaborar esta actividad, deben estar atentos a la explicación de su instructor y seguir los siguientes pasos:

• Observar atentos el video que es un algoritmo y lógica de programación estructura selectiva

• Luego, según lo visto en los videos, identifiquen cada una de las figuras y describan su utilidad o funcionamiento, señalen en qué parte de un diagrama de flujo se pueden utilizar. Escriban en la misma hoja.

• A continuación, con el grupo completo y haciendo una lluvia de ideas, piensen en una acción de la vida cotidiana. Ejemplo, pasos para cambiar un foco (bombillo) y utilizan las figuras dibujadas en el papel.

3.3.6 Actividad de Aprendizaje 6: Problemas Algoritmos

Realice la actividad de manera individual propuesta en los documentos de la unidad en el siguiente orden

- 6.1 Ejercicios Pseudocódigo-diagramas de flujo.pdf
- 6.2 Estructuras de control flowgorithm.pdf
- 6.3 Ciclos flowgorithm.pdf
- 6.4 Arrays flowgorithm.pdf
- 6.5 Funciones y Procedimientos NetBeans.pdf

3.4 Actividades de Transferencia el Conocimiento:

Actividad de Aprendizaje: Cuestionario y problemas de Lógica de Programación y Algoritmos para validar los conocimientos adquiridos por los aprendices.

4. PLANTEAMIENTO DE EVIDENCIAS DE APRENDIZAJE PARA LA EVALUACIÓN EN EL PROCESO FORMATIVO.

Fase del proyecto formativo	Actividad del proyecto formativo	Actividad de Aprendizaje	Evidencias de Aprendizaje	Criterios de Evaluación	Técnicas e Instrumentos de Evaluación
			Evidencia de producto: Taller de estructuras de control en la solución de algoritmos. GA3-220501096-AA1-EV01	Emplea lenguaje de programación para la solución de problemas de lógica proposicional según el análisis realizado.	Lista de chequeo. IE-GA3-220501096-AA1-EV01.
			Evidencia de producto: Taller de	Emplea lenguaje de programación para la solución de problemas de lógica proposicional	Lista de chequeo. IE- GA3-2205010

			funciones y procedimientos en la solución de algoritmos. GA3-220501096-AA1-EV02	según el análisis realizado.	96-AA1-EV02

5. GLOSARIO DE TÉRMINOS

Lógica Proposicional: Parte de la lógica que estudia las proposiciones simples y las operaciones que se pueden realizar sobre ellas (como la conjunción, la disyunción y la negación).

Algoritmo: Conjunto finito de instrucciones precisas y ordenadas que permiten llevar a cabo una actividad a resolver un problema.

Variable: Un símbolo que puede representar cualquier valor o elemento dentro de un conjunto dado.

Condición: Una expresión lógica cuyo valor determina la ejecución de ciertas partes de un algoritmo.

Ciclo/ Bucle: Una estructura de control que permite repetir un conjunto de instrucciones hasta que se cumpla una condición específica.

Condición IF: Una estructura de control que ejecuta cierto bloque de código si se cumple una condición determinada.

Ciclo While: Un bucle que repite un bloque de código mientras una condición específica sea verdadera.

Ciclo For: Un bucle que itera sobre una secuencia de elementos (como una lista o rango) y ejecuta un bloque de código para cada elemento.

Recursión: Un proceso en el que una función se llama a sí misma directa o indirectamente.

Estructura de datos: Forma de organizar y almacenar datos para que puedan ser accedidos y utilizados de manera eficiente.

Pila (Stack): Una estructura de datos que sigue el principio de "último en entrar, primero en salir" (LIFO).

Cola (Queue): Una estructura de datos que sigue el principio de "primero en entrar, primero en salir" (FIFO).

Árbol: Una estructura de datos jerárquica compuesta por nodos conectados entre sí mediante aristas.

Grafo: Una estructura de datos compuesta por un conjunto de nodos (vértices) y un conjunto de aristas que conectan esos nodos.

Complejidad algorítmica: Medida del tiempo y/o espacio necesario para ejecutar un algoritmo en función del tamaño de la entrada.

Orden de complejidad: Una función que describe cómo aumenta el tiempo y/o el espacio requerido por un algoritmo a medida que aumenta el tamaño de la entrada.

Big O Notation: Una notación utilizada para describir el orden de complejidad de un algoritmo en términos asintóticos.

Programación Dinámica: Una técnica para resolver problemas computacionales dividiéndolos en subproblemas más pequeños y resolviéndolos de manera recursiva.

Divide y Vencerás: Un paradigma de diseño de algoritmos que consiste en resolver un problema dividiéndolo en subproblemas más pequeños, resolviendo estos subproblemas de manera independiente y luego combinando sus soluciones.

Heurística: Una técnica que proporciona una solución aproximada a un problema cuando no es posible encontrar una solución óptima en un tiempo razonable.

6. REFERENTES BILBIOGRÁFICOS

Java en Castellano. (2021, Dic 01). Servlets y JSP.

https://programacion.net/articulo/servlets_y_jsp_82 Maida, E. G., & Pacienzia, J. (2015). Metodologías de desarrollo de software. Quintas, A.

F. (2000). *Java 2: manual de usuario y tutorial*. Grupo Editorial RA-MA. Sun. (2021, Dic 01). Java. http://java.sun.com/

Vall Mainou, A. (2010). Desarrollo para internet con tecnología JAVA.

7. CONTROL DEL DOCUMENTO

	Nombre	Cargo	Dependencia	Fecha
Autor (es)				

8. CONTROL DE CAMBIOS (diligenciar únicamente si realiza ajustes a la guía)

	Nombre	Cargo	Dependencia	Fecha	Razón del Cambio
Autor (es)					

Práctica: Conceptos básicos de Kotlin

1. Notificaciones móviles

Por lo general, el teléfono te proporciona un resumen de las notificaciones.

En el código inicial que se proporciona en el siguiente fragmento de código, escribe un programa que imprima el mensaje de resumen según la cantidad de notificaciones que recibiste. El mensaje debe incluir lo siguiente:

- La cantidad exacta de notificaciones cuando haya menos de 100
- `99+` como cantidad de notificaciones cuando haya 100 o más

```
fun main() {
    val morningNotification = 51
    val eveningNotification = 135

    printNotificationSummary(morningNotification)
    printNotificationSummary(eveningNotification)
}

fun printNotificationSummary(numberOfMessages: Int) {
    // Completa el código
}
```

Completa la función `printNotificationSummary()` para que el programa imprima estas líneas:

You have 51 notifications.
Your phone is blowing up! You have 99+ notifications.

2. Precio de la entrada de cine

Las entradas de cine suelen tener un precio diferente según la edad de los espectadores.

En el código inicial que se proporciona en el siguiente fragmento de código, escribe un programa que calcule los precios de estas entradas basados en la edad:

- Un precio de entrada infantil de USD 15 para personas de 12 años o menos.
- Un precio de entrada estándar de USD 30 para personas de entre 13 y 60 años. Los lunes, un precio estándar con descuento de USD 25 para el mismo grupo etario
- Un precio para adultos mayores de USD 20 para personas de 61 años o más (asumimos que la edad máxima de un espectador es de 100 años)

- Un valor de `-1` para indicar que el precio no es válido cuando un usuario ingresa una edad fuera de las especificaciones

```
fun main() {
    val child = 5
    val adult = 28
    val senior = 87

    val isMonday = true

    println("The movie ticket price for a person aged $child is \$${ticketPrice(child, isMonday)}.")
    println("The movie ticket price for a person aged $adult is \$${ticketPrice(adult, isMonday)}.")
    println("The movie ticket price for a person aged $senior is \$${ticketPrice(senior,
isMonday)}.")
}

fun ticketPrice(age: Int, isMonday: Boolean): Int {
    //Completa el codigo
}
```

Completa la función `ticketPrice()` para que el programa imprima estas líneas:

```
The movie ticket price for a person aged 5 is $15.
The movie ticket price for a person aged 28 is $25.
The movie ticket price for a person aged 87 is $20.
```

3. Conversor de temperatura

En el mundo, se usan tres escalas de temperatura principales: Celsius, Fahrenheit y Kelvin.

En el código inicial que se proporciona en el siguiente fragmento de código, escribe un programa que convierta una temperatura de una escala a otra con estas fórmulas:

- De grados Celsius a Fahrenheit: $°F = 9/5\ (°C) + 32$
- Kelvin a Celsius: $°C = K - 273.15$
- De Fahrenheit a Kelvin: $K = 5/9\ (°F - 32) + 273.15$

Ten en cuenta que el método `String.format("%.2f", /* measurement */ )` se usa para convertir un número en un tipo `String` con 2 decimales.

```
fun main() {
  //Completa el código
}
```

```
fun printFinalTemperature(
    initialMeasurement: Double,
    initialUnit: String,
    finalUnit: String,
    conversionFormula: (Double) -> Double
) {
    val finalMeasurement = String.format("%.2f", conversionFormula(initialMeasurement)) // two
decimal places
    println("$initialMeasurement degrees $initialUnit is $finalMeasurement degrees $finalUnit.")
}
```

Completa la función `main()` para que llame a la función `printFinalTemperature()` e imprima las siguientes líneas. Debes pasar argumentos para la fórmula de conversión y temperatura. *Sugerencia:* Te recomendamos usar valores `Double` para evitar el truncamiento de `Integer` durante las operaciones de división.

```
27.0 degrees Celsius is 80.60 degrees Fahrenheit.
350.0 degrees Kelvin is 76.85 degrees Celsius.
10.0 degrees Fahrenheit is 260.93 degrees Kelvin.
```

4. Perfil de Internet

A menudo, debes completar los perfiles de sitios web en línea que contienen campos obligatorios y no obligatorios. Por ejemplo, puedes agregar tu información personal y un vínculo a otras personas que te refirieron para que registraras tu perfil.

En el código inicial que se proporciona en el siguiente fragmento de código, escribe un programa que imprima los detalles del perfil de una persona.

```
fun main() {
    val amanda = Person("Amanda", 33, "play tennis", null)
    val atiqah = Person("Atiqah", 28, "climb", amanda)

    amanda.showProfile()
    atiqah.showProfile()
}

class Person(val name: String, val age: Int, val hobby: String?, val referrer: Person?) {
    fun showProfile() {
      //Completa le codigo
    }
}
```

Completa la función `showProfile()` para que el programa imprima estas líneas:

Name: Amanda
Age: 33
Likes to play tennis. Doesn't have a referrer.

Name: Atiqah
Age: 28
Likes to climb. Has a referrer named Amanda, who likes to play tennis.

5. Teléfonos plegables

Por lo general, la pantalla del teléfono se enciende y se apaga cuando se presiona el botón de encendido. En cambio, si un teléfono plegable está plegado, su pantalla interna principal no se enciende cuando se presiona el botón de encendido.

En el código inicial que se proporciona en el siguiente fragmento de código, escribe una clase `FoldablePhone` que se herede de la clase `Phone`. Debe contener lo siguiente:

- Una propiedad que indique si el teléfono está plegado
- Un comportamiento de función `switchOn()` diferente del de la clase `Phone` para que solo encienda la pantalla cuando el teléfono no esté plegado
- Métodos para cambiar el estado de plegado

```
class Phone(var isScreenLightOn: Boolean = false){
    fun switchOn() {
        isScreenLightOn = true
    }

    fun switchOff() {
        isScreenLightOn = false
    }

    fun checkPhoneScreenLight() {
        val phoneScreenLight = if (isScreenLightOn) "on" else "off"
        println("The phone screen's light is $phoneScreenLight.")
    }
}
```

6. Subasta especial

Por lo general, en una subasta, el ofertante que ofrece el importe más alto determina el precio de un artículo. En esta subasta especial, si nadie oferta por un artículo, este se vende automáticamente a la casa de subastas al precio mínimo.

En el código inicial que se proporciona en el siguiente fragmento de código, se te proporciona una función `auctionPrice()` que acepta un tipo `Bid?` anulable como argumento:

```
fun main() {
    val winningBid = Bid(5000, "Private Collector")

    println("Item A is sold at ${auctionPrice(winningBid, 2000)}.")
    println("Item B is sold at ${auctionPrice(null, 3000)}.")
}

class Bid(val amount: Int, val bidder: String)

fun auctionPrice(bid: Bid?, minimumPrice: Int): Int {
   //Completa el código
}
```

Completa la función `auctionPrice()` para que el programa imprima estas líneas:

```
Item A is sold at 5000.
Item B is sold at 3000.
```

Usa MVVM en tus apps con Kotlin



Kevin Morales ·
5 min read · Jan 13, 2024

¡Hola amig@! Mi nombre es Kevin Morales y soy desarrollador mobile por más de 7 años. Hoy te explicaré cómo organizar tu proyecto con la arquitectura por excelencia recomendada por Google implementando MVVM en Android y que sea más escalable a futuro.



Bueno, entrando en materia, MVVM (Model View ViewModel o Modelo Vista VistaModel) es una arquitectura muy usada actualmente por desarrolladores en el mundo del software. Cuenta con una curva de aprendizaje rápida y la cual se usa en conjunto con paradigmas de programación actuales como por ejemplo Jetpack Compose (Android) y SwiftUI (iOS).



Ahora veamos algunos conceptos antes de mostrar un ejemplo con código en Kotlin explicando este patrón.

Model View ViewModel

Es una arquitectura de diseño de software que se ha vuelto popular en el desarrollo de aplicaciones Android. En MVVM, la lógica de presentación se separa de la interfaz de usuario y del modelo de datos, lo que facilita la prueba, la reutilización y el mantenimiento del código. A continuación te muestro una descripción básica de cada componente en MVVM.

Modelo o Model

- Representa los datos y la lógica de negocio de la aplicación.

- Puede incluir fuentes de datos como bases de datos, servicios web o cualquier otra fuente de datos.

Vista o View:

- Representa la interfaz de usuario.
- Muestra los datos al usuario y captura las interacciones del usuario.
- No debe contener lógica de negocio directamente.

Model Vista o ViewModel

- Actúa como un intermediario entre el Modelo y la Vista.
- Contiene la lógica de presentación y se encarga de preparar los datos para que la Vista los muestre.
- La Vista se suscribe a cambios en el ViewModel y se actualiza automáticamente cuando los datos cambian.

El MVVM es una arquitectura que al tener una separación de las capas facilita la escritura de pruebas unitarias. Puedes probar el ViewModel de manera aislada, lo que permite una cobertura de prueba más efectiva y la identificación temprana de errores. Siendo una arquitectura escalable que se adapta bien a aplicaciones de diferentes tamaños. A medida que la aplicación crece, la separación de capas permite gestionar la complejidad de manera más efectiva.

Entonces, centrándonos en Android, la implementación de MVVM generalmente se realiza con las siguientes bibliotecas y componentes.

ViewModel

Esto es una clase diseñada para contener y administrar datos relacionados con la interfaz de usuario de una manera consciente del ciclo de vida. Se utiliza para almacenar y gestionar datos necesarios para la interfaz de usuario, incluso durante cambios de configuración de hardware, como por ejemplo rotación de la pantalla en tiempo de ejecución.

LiveData

Esto es una clase observable (si quieres saber más acerca del patrón **Observer/ Observable** te recomiendo este artículo Patrones de diseño en Kotlin) que se utiliza en conjunto con ViewModel para proporcionar actualizaciones automáticas a la interfaz de usuario cuando los datos de la aplicación cambian.

Data Binding

Esto es una biblioteca que permite vincular componentes de la interfaz de usuario en el diseño directamente a los datos, eliminando así gran parte del código necesario para actualizar la interfaz de usuario.

Una vez entendido los conceptos básicos para poder aplicar MVVM en Android, vamos a ver un ejemplo orientado con el lenguaje de programación Kotlin.



Model

En primer lugar, generaremos una *data class* en donde crearemos dos constantes, *nombre* y *edad*. Esto será nuestro modelo en el ejemplo que estaremos realizando.

```
data class Usuario(
        val nombre: String,
        val edad: Int
)
```

View

En segundo lugar, generaremos la vista, básicamente un Activity en el que crearemos una variable *usuarioViewModel* la cual se comunicará con nuestro ViewModel.

```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import androidx.activity.viewModels
```

```
import androidx.lifecycle.Observer
import com.example.mvvm.databinding.ActivityMainBinding

class MainActivity: AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding
    private val usuarioViewModel: UsuarioViewModel by viewModels()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        // Observar cambios en el usuario y actualizar la interfaz
de usuario
        usuarioViewModel.usuario.observe(this,
Observer                { usuario ->
            binding.nombreId.text = usuario.nombre
            binding.edadId.text = usuario.edad.toString()
        })

                updateUser()
    }

        fun updateUser() {

    }
}
```

XML

En tercer lugar, apreciamos una variable *binding* la cual es la forma de enlazar los objetos de la vista XML con el Activity, esto es un buen tema para hablarlo en otro artículo. En este ejemplo, generemos dos *TextView,* uno para el nombre y otro para la edad

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/nombre_id"
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:text="Aquí irá el nombre"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_marginTop="8dp"
        android:textSize="14sp" />

    <TextView
        android:id="@+id/edad_id"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Aquí irá la edad"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_marginTop="8dp"
        android:textSize="14sp" />

</LinearLayout>
```

ViewModel

Por último y en cuarto lugar estará nuestro ViewModel, el cual llevará la lógica de negocio. En este caso actualizará el usuario e inmediatamente se conectará a la vista para realizar la actualización del *TextView* nombre y *TextView* edad respectivamente.

```
import androidx.lifecycle.ViewModel
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData

class UsuarioViewModel: ViewModel() {
    private val _usuario = MutableLiveData<Usuario>()
    val usuario: LiveData<Usuario> get() = _usuario

    // Método para actualizar el usuario
    fun actualizarUsuario(nombre: String, edad: Int) {
        val nuevoUsuario = Usuario(nombre, edad)
        _usuario.value = nuevoUsuario
    }
}
```

Una vez generada todas las clases necesarias, agregamos esta línea de código en la función *updateUser()* la cual simulará una acción de cambio de nombre y edad. La interfaz de usuario se actualiza automáticamente cuando cambian los datos del usuario gracias a LiveData y el patrón Observer en el ViewModel.

```
fun updateUser() {
```

```
        // Actualiza el usuario
        usuarioViewModel.actualizarUsuario("Juan", 25)
}
```

Conclusión

En resumen, la arquitectura Model View ViewModel o Modelo Vista VistaModelo proporciona una estructura organizada y eficiente que mejora la claridad del código, facilita las pruebas y reduce la complejidad del desarrollo de aplicaciones. Estos beneficios contribuyen a un desarrollo más rápido, mantenible y robusto.

Recuerda

> *Aprende cada día y serás una nueva persona*

		Versión: 01
		Código: GFPI-F-132
GESTIÓN DE LA FORMACIÓN PROFESIONAL INTEGRAL		
TALLER DE APRENDIZAJE		
Denominación del programa o módulo de formación:	ANALISIS, DISEÑO Y DESARROLLO DE SOFTWARE	
Actividad técnica de aprendizaje - ATA:	Identificar los requerimientos del informe técnicos.	
Nombre del taller Introducción a las aplicaciones móviles	Descripción:	N° 1
Objetivo de aprendizaje del taller:	Conocer la importancia de las aplicaciones móviles y como impactas en nuestra vida cotidiana.	
Descripción del taller:		
Preparación (Recursos y medios):	Computador. Acceso a internet. Biblioteca SENA. Google scholar. Microsoft Power Point o similar	
Tiempo de ejecución	3 horas	
Bibliografía y webgrafía:		
DESARROLLO DEL TALLER		

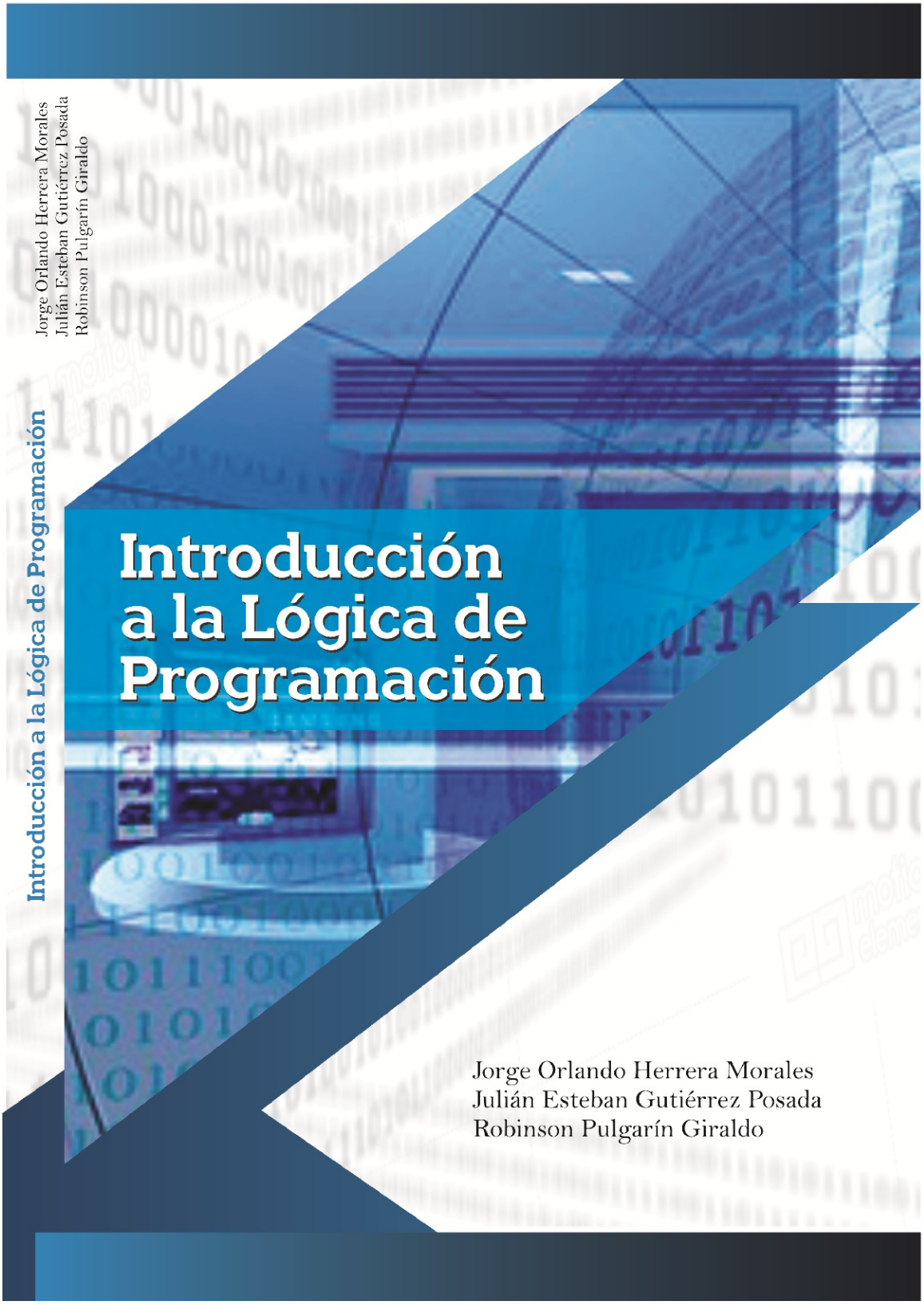
Actividad Grupal:

- En grupos de 3 Personas genero una discusión respondiendo a las preguntas:

1. ¿Por qué las aplicaciones móviles son importantes para la sociedad?
2. ¿Qué beneficio puedo obtener al aprender a desarrollar aplicaciones para dispositivos móviles?
3. ¿Cuáles son las aplicaciones móviles más conocidas, y que más se usan en la actualidad (mencione al menos 5 app)?
4. ¿Qué tipo de aplicación móvil mejoraría mi vida?

- Elaboro una presentación en Power Point con las respuestas seleccionadas y las comparto con los demás miembros de mi grupo, posteriormente y en consenso se da una respuesta a cada pregunta de manera grupal y se socializa al resto del curso a través de diapositivas hechas en Power Point.

Control de documento:	Nombre completo de quien elabora el cambio y centro de formación al que pertenece	Fecha
	Gustavo Adolfo Salazar Escobar - CCYS	22/10/2021


Introducción a la Lógica de Programación
Jorge Orlando Herrera Morales
Julián Esteban Gutiérrez Posada
Robinson Pulgarín Giraldo
Introducción a la Lógica de Programación
Jorge Orlando Herrera Morales
Julián Esteban Gutiérrez Posada
Robinson Pulgarín Giraldo

Introducción a la Lógica de Programación

Jorge O. Herrera M., Julián E. Gutiérrez P., Robinson Pulgarín G.

Armenia - Quindío - Colombia
2017

Primera edición: Colombia, diciembre 2017

Introducción a la Lógica de Programación
ISBN 978-958-8801-65-0

Editorial: ElIZCOM S.A.S
http://www.liber-book.com
ventas@elizcom.com
+57 311 334 97 48

Arte de los diagramas de flujo: María Alejandra Martos Díaz

Editado en: LaTeX 2ε

Diciembre de 2017



Este libro se distribuye bajo la licencia Creative Commons: Atribución-No Comercial-Sin Derivadas **CC BY-NC-ND**. Usted puede utilizar este archivo de conformidad con la Licencia. Usted puede obtener una copia de la Licencia en http://creativecommons.org/licenses/by-nc-nd/4.0/. En particular, esta licencia permite copiar y distribuir de forma gratuita, pero no permite venta ni modificaciones de este material.

Índice general

1 Capítulo 1 Fundamentos

1.1. Algo de historia 13

1.2. Dato 22

1.2.1. Tipos de datos 22

1.3. Identificadores 24

1.4. Variables 25

1.5. Constantes 30

1.6. Operadores y expresiones 30

1.6.1. Operadores aritméticos 31

1.6.2. Operadores relacionales 32

1.6.3. Operadores lógicos 33

1.6.4. Expresiones aritméticas 34

1.6.5. Conversión de fórmulas a notación algorítmica . . . 38

1.6.6. Expresiones relacionales 40

1.6.7. Expresiones lógicas 40

1.6.8. Prioridad de operación 43

1.7. Algoritmo 45
1.7.1. Clasificación de los algoritmos 46
1.7.2. Representación de un algoritmo 47
1.8. Cómo solucionar un problema por computador 66
1.9. Ejercicios propuestos 73

2 Capítulo 2
Estructura secuencial

2.1. Estructura básica de un algoritmo secuencial 81
2.2. Pruebas de escritorio 113
2.2.1. Ejemplos 114
2.3. Ejercicios propuestos 115

3 Capítulo 3
Estructuras de decisión

3.1. Decisiones compuestas 121
3.2. Decisiones anidadas 142
3.3. Decisiones múltiples 160
3.4. Ejercicios propuestos 174

4 Capítulo 4
Estructuras de repetición

4.1. Conceptos básicos 179
4.1.1. Contador 179
4.1.2. Acumulador 180
4.1.3. Bandera 181
4.2. Estructura `Mientras - FinMientras` 182

4.2.1. Prueba de escritorio 222
4.3. Estructura `Haga - MientrasQue` 228
4.3.1. Prueba de escritorio 266
4.4. Estructura de repetición `Para-FinPara` 270
4.4.1. Prueba de escritorio 312
4.5. Ejercicios propuestos 316

5 Capítulo 5
Procedimientos y funciones

5.1. Procedimiento 325
5.2. Funciones 338
5.3. Ejercicios propuestos 347

6 Capítulo 6
Vectores y matrices

6.1. Vectores 351
6.1.1. Declaración de un vector 352
6.1.2. Almacenamiento de datos en un vector 356
6.1.3. Recuperación de datos almacenados en un vector . 358
6.2. Matrices 408
6.2.1. Declaración de una matriz 409
6.2.2. Almacenamiento de datos en una matriz 410
6.2.3. Recorrido de una matriz 411
6.3. Ejercicios propuestos 440

Presentación

La importancia que ha tomado el mundo de la informática y la computación durante los últimos años ha sido tal que ha llegado a permear todas las áreas del saber humano. Siendo la programación de computadores uno de los aspectos relevantes dentro del mundo de la computación y el manejo de procesos algorítmicos una tarea cotidiana dentro del mundo organizacional, se hace necesario para quienes trabajan y quienes aspiran a trabajar en los sistemas empresariales tener conocimientos y desarrollar habilidades lógicas y algorítmicas que les permitan tener un mejor desempeño.

Esta obra pretende explorar la lógica de programación, los algoritmos y los diagramas de flujo con el propósito de ofrecer una alternativa actualizada para el aprendizaje de estos temas a quienes se inician en el estudio de esta interesante área. En ella, los autores han expuesto sus conocimientos y han plasmado sus diversas experiencias obtenidas como profesores durante varios años en instituciones de educación superior en las carreras donde se hace imprescindible el aprendizaje de la algoritmia y la lógica de programación.

Este texto se ha escrito buscando dar una mirada sencilla y práctica a la lógica de programación, exponiendo claramente los conceptos pero sin tratar de entrar en formalismos innecesarios que dificulten el aprendizaje. A su vez, se utilizan ejemplos de diferentes tipos para ilustrar cada uno de los tópicos estudiados en cada capítulo, desde los más sencillos hasta otros más complejos que guían paulatinamente al lector en la forma como deben resolverse los problemas de corte algorítmico, promoviendo el desarrollo de las habilidades necesarias en la escritura de algoritmos.

Para el desarrollo de los temas, la obra ha sido escrita en los siguientes seis capítulos:

Capítulo 1, Fundamentos: este capítulo introduce al lector en los conceptos generales sobre la lógica de programación y la algoritmia.

Se inicia con los aspectos más relevantes de la historia y origen de la computación; seguidamente se abordan los conceptos de datos y sus tipos, las variables y las constantes, los operadores y las expresiones, los tipos de algoritmos y cómo solucionar problemas a través de los algoritmos.

Capítulo 2, Estructura secuencial: a través de este capítulo se hacen las explicaciones generales sobre cómo escribir los primeros algoritmos, tanto en pseudocódigo como con diagramas de flujo y la forma adecuada de probarlos. El capítulo expone una buena cantidad de ejemplos de algoritmos suficientemente documentados.

Capítulo 3, Estructuras de decisión: en este apartado se exponen de forma clara las instrucciones necesarias para indicar a un algoritmo la forma de realizar un conjunto de tareas dependiendo de una condición. Se utilizan ejemplos documentados sobre las principales estructuras de decisión: simple, compuesta, anidada y múltiple.

Capítulo 4, Estructuras de repetición: se inicia definiendo los términos de contador, acumulador y bandera, utilizados durante todo el capítulo. Posteriormente se tratan cada una de las instrucciones repetitivas como son: el ciclo `Mientras-FinMientras`, el ciclo `Haga-MientrasQue` y el ciclo `Para-FinPara`. Todas estas estructuras son explicadas desde el punto de vista conceptual y práctico, con el fin de que el lector comprenda no solamente los conceptos sino que aprenda a utilizarlos en la solución de problemas algorítmicos.

Capítulo 5, Procedimientos y Funciones: se llevan a cabo las definiciones de lo que son procedimientos y funciones, su uso dentro de la algoritmia y se expone una serie de ejemplos prácticos que ilustran la utilidad de este tipo de instrucciones.

Capítulo 6, Vectores y Matrices: en este último capítulo se introduce al lector en el tema de los arreglos unidimensionales y bidimensionales con el propósito de que se comprendan los conceptos generales sobre estas estructuras de datos y se aprendan a utilizar cuando se deban resolver problemas algorítmicos que así lo requieran.

Organización de la obra.

Para abordar cada tema el libro fue organizado en capítulos, cada uno desarrolla los conceptos teóricos, llevados a la práctica con los suficientes ejemplos que ilustran su uso en la solución de problemas de tipo computacional.

Uno de los aspectos a destacar, es la forma como fueron abordadas las soluciones de los ejemplos propuestos. Para cada uno se hizo un análisis previo, tratando de tener un mayor dominio sobre la problemática, ya que es fundamental un total entendimiento para poder emprender la búsqueda de una adecuada solución. Cada ejemplo desarrollado fue cuidadosamente detallado desde su análisis hasta la concepción del algoritmo, representados a través de pseudocódigo y en la mayoría de los casos acompañados de su correspondiente diagrama de flujo; adicionalmente, se hace una descripción minuciosa de cada una de las instrucciones que fueron usadas. Esta metodología busca crear la buena práctica de realizar un análisis detallado del problema y entender cada uno de los pasos de la solución.

Adicionalmente, dentro de las diferentes páginas el lector encontrará el recuadro de Buenas prácticas y el recuadro de Aclaraciones:

Buena práctica:



En este recuadro se mostrarán recomendaciones que se espera sean seguidas por los lectores de esta obra, con el fin de que aprendan, desde sus inicios como desarrolladores, buenas prácticas para un mejor desempeño profesional.

Aclaración:



Esta anotación, es usada para resaltar algunos conceptos o dar mayor claridad sobre aspectos que son fundamentales en el desarrollo de algoritmos.

Otra característica importante, es que se aprovechó el formato digital de la obra, de tal forma que cada una de las referencias escritas tienen presente un link, permitiendo que con solo pulsar la referencia se puede dirigir al sitio correspondiente.

Por último, al finalizar cada capítulo, los autores han dejado una serie de ejercicios propuestos que van desde preguntas teóricas que pretenden afianzar en el lector los conceptos estudiados, hasta los enunciados de ejercicios prácticos para que el estudioso escriba los algoritmos que solucionen el problema propuesto utilizando la lógica y la algorítmia.

Con todo el trabajo realizado desde la concepción de esta obra, pasando por su escritura y elaboración de la gran cantidad de algoritmos que en ella se exponen, los autores esperan hacer una contribución en la difusión de la lógica de programación y la algoritmia, así como orientar y facilitar el aprendizaje de estos temas a todos aquellos lectores apasionados de la informática y la computación. Es por eso, que para lograr este propósito el libro fue concebido bajo la Licencia Creative Commons permitiendo el uso y la distribución gratuita, siempre y cuando la obra se conserve igual y se haga el respecivo reconocimiento a sus creadores.

Por último, los autores desean expresar su agradecimiento a todos aquellos que directa o indirectamente han inspirado este libro, entre los cuales se encuentran, estudiantes quienes manifiestan la necesidad de un texto que facilite el aprendizaje, docentes que resaltan la importancia de contar con un libro de referencia y guía en el salón de clases y profesionales de diferentes áreas quienes desean contar con una referencia de consulta para resolver alguna duda o inquietud.

Capítulo 1

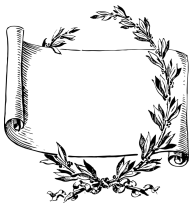
Fundamentos

Hazlo todo tan simple como sea posible, pero no más simple

Albert Einstein

Objetivos del capítulo:

- Conocer los aspectos más relevantes de la historia de la programación.
- Comprender los conceptos básicos de la Lógica de Programación.
- Expresar un algoritmo en una de las formas de representación.
- Identificar, evaluar y construir operaciones aritméticas, relacionales y lógicas.
- Aplicar los pasos para la solución de un problema por computador .



1.1. Algo de historia

Al estudiar el origen de la programación, se puede descubrir que los algoritmos y los programas para computador surgieron antes que los mismos computadores u otros dispositivos de procesamiento de datos.

Euclides en el año 300 a.C, escribió el algoritmo que lleva su nombre y que es usado en el mundo de las matemáticas y la computación para hallar el Máximo Común Divisor (MCD). Al-Khwārizmī o Al-Juarismi (750 - 850 d.C.), matemático y astrónomo persa, considerado como el padre del álgebra, escribió las reglas para realizar las operaciones básicas de la aritmética [Mancilla et al., 2016], denominadas la Reglas de al-Kharizmi; del nombre de este personaje se derivó el término algoritmo. Leonardo de Pisa, también conocido como Leonardo Fibonacci, hacia el año 1202, diseñó el algoritmo para generar la sucesión de los números de Fibonacci.

Así mismo, los orígenes e historia del computador y el procesamiento de datos se remontan a miles de años atrás, cuando el hombre empezó a registrar información. Sus manifestaciones iniciales se dieron en los muros de las cavernas cuando los primitivos hicieron sus primeros registros. Con el paso del tiempo surge la necesidad de realizar cálculos, de esta forma se introdujeron dispositivos que permitían realizarlos de manera más rápida.

A continuación, se hará un recorrido histórico por los acontecimientos y personajes más representativos que han dejado su legado en la evolución de esta temática. Aunque se visualizan como una serie de eventos aislados, muchos de ellos han convergido en la creación de nuevos desarrollos.

- El ábaco, que es el instrumento más antiguo que se conoce con el cual se hacen operaciones matemáticas sencillas, fue inventado por las antiguas civilizaciones hace miles de años [Oviedo, 2013]. En la actualidad es usado como un instrumento didáctico en muchas escuelas del mundo. El imperio Inca, hacia los años 1300 a 1500 d.C. usaba un sistema de cuentas, conocido con el nombre de Quipu, que a través de nudos en cuerdas de colores registraba la contabilidad de sus inventarios.
- Luego de muchos años, se empezaron a registrar diseños de aparatos mecánicos que cumplían tareas de calculadora. En el año 1500 de nuestra era, aproximadamente, Leonardo Da Vinci, diseñó una sumadora mecánica, que fue construida muchos años después, en 1968.

- En 1617, el matemático escocés John Napier inventa las varillas que llevan su apellido. Las varillas de Napier permitían realizar multiplicaciones, divisiones y la extracción de raíces.
- Para el año 1623, Wilhelm Schickard, alemán, esbozó una máquina que realizaba las 4 operaciones básicas, su invento fue derivado de las varillas de Napier. Este instrumento fue bautizado por su creador como el Reloj calculante. No existe evidencia de que este diseño se hubiese materializado en aquella época, pero en tiempos recientes se construyeron modelos basados en su diseño.
- El francés Blaise Pascal en 1642, construye la primera sumadora mecánica, llamada Pascalina, fue inspirada en el diseño de Da Vinci.
- Entre 1671 y 1674 el filósofo y matemático alemán Gottfried Wilheim Von Leibnitz mejora el diseño de Pascal y construye la Calculadora Universal; adicionalmente podía calcular multiplicaciones y divisiones a partir de sumas y restas sucesivas. Además, gracias a este gran genio se cuenta con el sistema binario o de base 2 (b_2) que utiliza el 0 y el 1, tal como se conoce en la actualidad. El sistema binario es el sistema numérico por naturaleza de los computadores. A este matemático también se le debe el concepto de arreglo o matriz [Mancilla et al., 2016].

Hasta este momento de la historia solamente se habían diseñado o creado instrumentos que permitían hacer cálculos, ya fuera de forma manual o mecánica. Pero a partir del año 1800, se inventan nuevos dispositivos un poco más complejos y con otros propósitos.

- Entre 1801 y 1804, en Francia, Joseph Jacquard construyó un telar para producir telas de manera automática, controlado mediante un conjunto de delgadas placas de madera perforada.
- En 1822, el matemático inglés Charles Babbage, de la Universidad de Cambridge, concibió la idea de computadores mecánicos. Diseñó y construyó un prototipo de la Máquina de Diferencias o Máquina diferencial, por diferentes inconvenientes este proyecto no se terminó. En 1834 inició un nuevo proyecto, la creación de la Máquina Analítica; fue diseñada como una máquina de propósito general, es decir, programable. Tenía memoria, procesador y dispositivos de entrada y de salida. Su funcionamiento estaba controlado a través de tarjetas perforadas. Esta máquina es considerada la antecesora de los computadores actuales y Babbage es reconocido como el

padre de la informática. Igual que su primer proyecto, esta máquina tampoco se construyó, estaba muy adelantada para aquella época y las limitaciones técnicas impidieron su desarrollo. No obstante, luego de más de 100 años de este trabajo, la máquina se construyó siguiendo los diseños originales, comprobando así que era totalmente funcional.

- Augusta Ada Byron, mejor conocida como Ada Lovelace, destacada en el área de las matemáticas, fue la primera persona que escribió un programa de computador. Trabajó al lado de Babbage y escribió el primer conjunto de instrucciones para que fueran ejecutadas por la máquina Analítica de Babbage [Boullosa, 2016]; aunque su desarrollo fue teórico le permitió ganarse el reconocimiento como la primera persona programadora de la historia. Por encargo del gobierno de Estados Unidos y como un homenaje por este mérito, se desarrolló un lenguaje de programación que lleva su nombre.
- Después de más de 100 años de los aportes de Babbage y Lovelace, se pudieron crear los primeros computadores. Pero esta creación es un conjunto de diferentes desarrollos en tecnología y nuevos conocimientos que se fueron consolidando hasta obtener los equipos, los métodos algorítmicos y los lenguajes de programación de la actualidad.
- En 1848, George Boole matemático y lógico inglés, hace una gran contribución a las ciencias de la computación y a la electrónica, creando el álgebra de Boole o álgebra booleana; con ella se construyen las expresiones lógicas que son constantemente usadas dentro de los algoritmos computacionales.
- En 1887, Herman Hollerith, diseñó la Máquina de censos, que funcionaba basada en la lógica de Boole y con tarjetas perforadas que fueron inspiradas en las placas de madera del telar de Jacquard. En 1896 fundó la compañía de Máquinas de tabulación y en 1924 se fusionó con otras empresas para fundar la reconocida y aún vigente multinacional IBM (*International Business Machines Corporation*).
- Alan Mathison Turing, nacido en Londres, fue matemático, lógico y criptógrafo. Se considera uno de los pioneros de la informática moderna. En 1936 elaboró la teoría de computador digital, imaginando una máquina llamada Máquina de Turing, capaz de resolver problemas matemáticos siguiendo algunas reglas lógicas [López, 2015].

  La Máquina de Turing, era un dispositivo con una cinta infinita, que representaba la memoria, en la cual se podían escribir y

leer instrucciones a través de un cabezal. Con esta máquina se estableció el marco teórico que generó los actuales computadores [Areitio and Areitio, 2009].

- Luego entre 1939 y 1940 un equipo de ingenieros, dirigido por Alan Turing, construyó uno de sus diseños, una máquina considerada un computador electromecánico de uso especial a la cual llamaron Bomba; con ella se descifraban los códigos de la máquina Enigma[1] de los alemanes.
- En 1941, Konrad Zuse, ingeniero alemán, construyó la Z3, un computador electromecánico que funcionaba con sistema binario. Sus programas eran almacenados en una cinta externa. Existe divergencia de conceptos en considerarla como el primer computador programable, algunos historiadores difieren de este calificativo, porque no cumplía la condición de ser un computador de propósito general[2].
- En 1943, tras el ingreso a escena de nuevas formas para cifrar mensajes por parte de los alemanes, la máquina Bomba de Turing ya no era eficaz; surge la necesidad de un nuevo dispositivo. El inglés Tommy Flowers, especialista en electrónica, junto con su equipo de trabajo crea a Colossus, una máquina de propósito específico que es considerado como uno de los primeros computadores electrónicos; contaba con 1500 válvulas o tubos de vacío y utilizaba cintas de papel perforado para leer los datos.
- Para 1944, Howard Aiken, en asocio con IBM construyen el Mark I, un computador electromecánico. En ella se materializaron los diseños de Babbage. Utilizaba cinta de papel perforado para leer sus datos y sus instrucciones.
- Entre 1943 y 1945 se construye el ENIAC (*Electronic Numerical Integrator And Computer* - Computador e Integrador Numérico Electrónico) en los Estados Unidos, fue el primer computador electrónico de propósito general. Contenía 18000 tubos al vacío. Inicialmente fue pensado para cálculos balísticos, luego se usó para solucionar problemas de tipo científico. Junto con esta máquina surge el sistema de codificación de ENIAC, utilizado para programar este

[1] Máquina para cifrar y descifrar mensajes. Fue ampliamente usada por los alemanes en la segunda guerra mundial.

[2] Un computador de propósito general es aquel que puede ser programado para ejecutar diversas tareas o aplicaciones, a diferencia de los de propósito específico que son programados para ejecutar una sola tarea o aplicación.

computador realizando ajustes en sus conexiones. Esta programación estaba a cargo de 6 mujeres que pasaron a la historia como las programadoras originales del primer computador electrónico, ellas fueron: Jean Jennings Bartik, Betty Snyder Holberton, Frances Bilas Spence, Kathleen McNulty Mauchly Antonelli, Marlyn Wescoff Meltzer y Ruth Linchterman Teitelbaum.

- En 1945 el matemático húngaro John Von Neumann, considerado el padre de los computadores electrónicos, diseñó el Computador de Programa Almacenado. Este concepto revolucionó el mundo de la computación, a partir de él se tienen los computadores que existen en la actualidad. Las instrucciones se almacenan y procesan dentro de la máquina y solamente se deben ingresar nuevas instrucciones para ejecutar diferentes instrucciones; con este nuevo sistema no se requiere cambiar las conexiones de los cables de la máquina para asignarle nuevas tareas.
- Konrad Zuse, el mismo creador del Z3, en 1946 inventó el Plankalkül (plan de cálculo); primer lenguaje de programación registrado en la historia y predecesor de los actuales lenguajes de programación algorítmicos. Era un lenguaje teórico pensado para ser usado en los Z3. Fue dado a conocer solo hasta 1972 y en el año 2000 lo implementaron en la Universidad Libre de Berlín [Szymanczyk, 2013].
- Hacia 1947 crean el transistor en los laboratorios BELL de Estados Unidos. Con este invento se inicia la miniaturización de componentes.
- En 1949 en la Universidad de Cambridge, Inglaterra, se construyó el primer computador con el concepto de Von Neumann, denominado EDSAC (*Electronic Delay Storage Automatic Calculator*). Este computador pasó a los libros de historia como el primer computador electrónico con programa almacenado. Lo sucedió un año después el EDVAC (*Electronic Discrete Variable Automatic Computer*). Estos equipos usaban tecnología de tubos de vacío y la entrada de datos se hacía usando cintas de papel perforado.
- Se crea el Lenguaje Ensamblador en 1950, que mejora la expresividad, haciendo que la programación fuera más comprensibles en este momento de la historia.
- Grace Murray Hooper, fue una destacada científica computacional y militar nacida en Nueva York. En 1951 implementó el primer compilador[3] para un lenguaje de programación. Ella introdujo el

[3]Un compilador es un programa que traduce instrucciones de un lenguaje a otro.

concepto que los lenguajes debían ser independientes de la máquina en que se ejecutan.

- En 1951 inicia operaciones el UNIVAC I (*Universal Automatic Computer*), primer computador electrónico de programa almacenado y propósito general que fue comercializada; se utilizó inicialmente con fines científicos y militares, luego se usó en aplicaciones comerciales. Estaba construido con válvulas de vacío y utilizaba unidades de cinta magnética para almacenar datos.

- Los laboratorios BELL de Estados Unidos en 1955 fabrican a Tradic, la primera calculadora que hace uso del transistor [Szymanczyk, 2013]. Esto marcó el inicio del desarrollo de una nueva generación de computadores, logrando una producción más variada que permitió la comercialización de diversos modelos.

- Entre 1957 y 1960 hacen su aparición los lenguajes FORTRAN, Algol 58, LISP, COBOL, BASIC y PL/1. FORTRAN (*Formula Translating*) utilizado para realizar cálculos de tipo científico. Algol 58 (*Algorithmic Language*) catalogado como el primer lenguaje algorítmico, muy usado como herramienta de enseñanza. LISP (*LISt Processor*), se introdujo inicialmente en el campo de la inteligencia artificial. COBOL (*COmmon Business Oriented Language*) es un lenguaje orientado a los negocios. BASIC (*Beginner's All purpose Symbolic Instruction Code* - Código simbólico de instrucciones de propósito general para principiantes), creado inicialmente para facilitar la enseñanza y el aprendizaje de la programación a aquellas personas que no eran expertas en el campo de la computación. PL/1 (*Programming Language 1*) creado para aplicaciones científicas y comerciales, que sirvió de base para la generación del Lenguaje C y C++.

- Hacia 1965 se construyen los primeros computadores haciendo uso del circuito integrado o chip, reduciendo significativamente su tamaño y aumentando su eficiencia. El chip fue inventado por Jack Kilby en 1958.

- En 1969 se establece una comunicación entre dos computadores ubicados en dos Universidades de Estados Unidos y se da origen a ARPANET, que evolucionó a lo que hoy se conoce como Internet.

- En los laboratorios BELL, en el año 1969 se crea el sistema operativo UNIX y en 1970 el Lenguaje B, orientado a la construcción de sistemas operativos y predecesor del lenguaje C.

- En 1970 se publica el Lenguaje Pascal, nombrado así en honor a Blaise Pascal, el inventor de Pascalina mencionada anteriormente. Inicialmente este lenguaje se desarrolló con fines académicos, pero su facilidad de aprendizaje permitió que se popularizara y fuera empleado para construir programas con diferentes propósitos.
- Entre 1970 y 1971, Intel crea el microprocesador, permitiendo así el desarrollo de los computadores personales.
- Hacia 1971, nace el correo electrónico, su creador fue Ray Tomlinson. Este medio de comunicación ha tomado tanta relevancia que es uno de los más utilizados tanto a nivel empresarial como personal.
- En el año 1972 se crean los lenguajes Smalltalk, uno de los primeros lenguajes orientados a objetos. Lenguaje C, creado para desarrollar sistemas operativos y aplicaciones de propósito general. PROLOG (*PROgrammation in LOGique*), es un lenguaje lógico, ampliamente utilizado en el campo de la inteligencia artificial.
- Nace, sin acogida, en 1973 el primer computador personal (PC), denominado Alto, creado por Xerox. Su fracaso obedeció a sus elevados costos.
- En 1975 se crea un nuevo computador personal, el Altair 8800; comercializado con gran éxito y reconocido como el primer computador personal, quitándole la gloria al Alto de Xerox. En este mismo año, Bill Gates y Paul Allen, crearon el Altair Basic para el Altair 8800, un intérprete del Lenguaje Basic, dándole así inicio a una de las grandes compañías del software, Microsoft Corporation.
- En 1977 sale al mercado uno de los computadores más populares de la época, el Apple II, sus ventas fueron masivas tanto para hogares como para empresas.
- La comercialización de software para los usuarios se inicia en 1978.
- ADA fue dado a conocer en 1980, creado por encargo del Departamento de Defensa de los Estados Unidos, es un lenguaje multipropósito y con aplicación en el desarrollo de sistemas de seguridad y aeronáutica.
- Hacia 1981 IBM hace historia con la creación de un nuevo PC basado en el procesador Intel 8088, su memoria RAM era de 64 KB con posibilidad de expansión a 256 KB. Venía equipado con una unidad de diskette de 5 pulgadas y sin disco duro. Su sistema operativo era el MS-DOS, *MicroSoft Disk Operating System*, Sistema operativo de disco de Microsoft que podía ser inicializado desde un disco flexible.

- Un año después se popularizan los computadores clones de IBM.
- Se publica el Lenguaje C++ en 1983, una evolución del Lenguaje C con orientación a objetos.
- En 1984, Apple lanza el Macintosh. Su software, basado en íconos y gráficos tuvo bastante éxito por la usabilidad que ofrecía.
- El sistema operativo Windows empieza a comercializarse en 1985.
- En 1989, sale al mercado el procesador 486 de Intel.
- Internet comienza su expansión hacia el año 1989. Iniciando así un desarrollo vertiginoso que aún no para de crecer.
- Para 1990 Windows evoluciona a la versión 3.0.
- Finlandia, en 1991, le entregó al mundo de la computación el sistema operativo Linux, su creador fue Linus Torvalds.
- En ese mismo año surgen los lenguajes Python y Visual Basic. Python tiene una curva de aprendizaje rápida y permite el desarrollo de diferentes tipos de aplicaciones. Visual Basic es una evolución del Basic, es de propósito general.
- Surge WWW (*World Wide Web*) en 1993. A través de múltiples hipertextos (link) se pueden enlazar, por medio de Internet, diferentes tipos de información tales como textos, imágenes, vídeos, entre otros.
- Mosaic es lanzado al mercado en 1994, como uno de los navegadores más populares en Internet. En ese mismo año, dos estudiantes de la Universidad de Stanford crean a Yahoo, uno de los motores de búsqueda más populares de ese tiempo; de igual forma Lycos, otro motor de búsqueda hace su aparición para competir con Yahoo. Estos nuevos desarrollos hacen que el uso de Internet se popularice.
- En 1995 se suceden varios acontecimientos de gran importancia. Netscape, logra convertirse en el navegador de más uso del momento. Surge Altavista, otro popular motor de búsqueda que entra a la competencia, logrando en poco tiempo convertirse en uno de los favoritos de los navegantes. Nace Windows 95, un sistema operativo con interfaz gráfica, poco a poco fue desplazando al sistema operativo MS-DOS y a Windows 3.x que era un entorno gráfico que se ejecutaba bajo el MS-DOS. Amazon, una de las tiendas virtuales más conocidas, inicia sus operaciones comerciales. En ese mismo año se crea Java, un lenguaje de propósito general que se ha convertido en uno de los más usados en el momento.

- Microsoft en 1996, crea Internet Explorer, un navegador para ser ejecutado en el sistema operativo Windows. Luego fue incorporado en los equipos de Apple.
- PHP (*Hypertext Preprocessor - Procesador de Hipertexto*), se crea en 1997 como un lenguaje para desarrollo web.
- Google es lanzado en 1998 como un motor de búsqueda muy prometedor. Actualmente es uno de los más usados para realizar consultas en Internet.
- En 1999 se lanza el MSN Messenger Service, un servicio de mensajería instantánea que hoy en día está fusionado con la aplicación Skype. En ese mismo año crece el temor por el Y2K, también conocido como el Problema del año 2000. El Y2K fue un problema que se presentó en el mundo de la computación a causa de que los programadores anteriores al año 2000 tenían la costumbre de omitir los dos primeros dígitos del año, es decir, en lugar de escribir 1999 por ejemplo, lo expresaban como 99. Al llegar el año 2000, muchos de los computadores iban a tomar el año como 00, lo cual podía ocasionar bastantes problemas. El mundo se preparó para este evento y se hicieron los ajustes pertinentes, con lo cual se logró reducir los inconvenientes previstos.
- C# (C Sharp) se publica en el año 2001. Es un lenguaje de programación visual creado por Microsoft.
- En el 2003 Apple lanza su navegador Safari.
- El término de WEB 2.0 y sus nuevos servicios es conocido por el mundo en el año 2004. La Web 2.0 es la transformación que sufre Internet de pasar de páginas web estáticas a página interactivas, donde los navegantes no son simples consumidores de información, ya pueden interactuar con otros usuarios de la red y producir sus propios contenidos. Es así como toman relevancia o nacen sitios como Wikipedia, Youtube, Flickr, My Space, Blogger, Wordpress, Facebook, LinkedIn, Twitter, Tuenti, servicios de Google, entre muchos más.

Otro aspecto fundamental en la evolución de la computación y en especial en el área de la programación y algoritmia, es el desarrollo de los actuales smartphone y tablets, dispositivos que permiten almacenar aplicaciones, mejor conocidas como apps. Son millones las implementaciones que se han hecho para estos dispositivos, las cuales involucran la creación de diferentes algoritmos para desarrollar variadas

actividades, por ejemplo, encontrar una dirección como en el caso de Waze, buscar algún dato en Google, chatear en tiempo real mediante Whatsapp, gestionar correos electrónicos con Gmail, compartir información en una red social y muchas más.

La lista de esta evolución puede continuar y son muchos los dispositivos y lenguajes que no se han nombrado en este libro, pero que también han contribuido al desarrollo de la computación y la algoritmia.

El continuo y vertiginoso desarrollo tecnológico ha hecho que la programación no sea una competencia solo de los estudiosos de la computación o la informática, sino también de muchos profesionales en diferentes áreas del conocimiento y del resto de las personas que a diario interactúan con tecnología.

1.2. Dato

Un dato es una representación simbólica (un número, letra, gráfico, entre otros) de una característica de un elemento u objeto. En este sentido, se afirma que un dato puede estar representado por una cifra, letra, palabra o conjunto de palabras que describen una característica (atributo o propiedad) del elemento. Por ejemplo, “Gabriela Herrera” y “20” pueden representar el nombre y la edad de una persona respectivamente, donde el nombre y la edad son las características de esa persona.

Es necesario aclarar: un solo dato no representa información, sino que la información está conformada por un conjunto de datos que, al ser procesados mediante algún mecanismo, constituyen un mensaje para incrementar el conocimiento de quien lo recibe. Esto significa que solo los seres humanos o sistemas de cómputo muy avanzados como los sistemas expertos procesan información.

1.2.1 Tipos de datos

Un tipo de dato corresponde a una clasificación que se hace para poder tratar cada dato de la forma más adecuada, según lo que se requiera. El tipo de dato le indica al dispositivo de procesamiento cuanto espacio de memoria debe reservar para almacenar el dato, es decir, para determinar el tamaño del espacio de memoria. Los tipos de datos más comunes y que se trabajarán en este libro son: alfanuméricos, numéricos y lógicos.

Datos alfanuméricos

Estos datos se componen de la combinación de todos los caracteres conocidos: letras del alfabeto, dígitos y caracteres especiales, incluyendo el espacio en blanco. Los datos alfanuméricos se dividen en carácter y cadena.

- `Caracter`(sin tilde): se refiere a los datos que solo tienen un carácter, que puede ser una letra del alfabeto, un dígito del 0 al 9 o un carácter especial. Los valores de tipo carácter deben encerrarse entre comillas simples.

  Por ejemplo: 'a', 'R', '2', '#', '@'

  Los dígitos que son tratados como caracteres son diferentes a los valores numéricos, por lo tanto, no se deben realizar operaciones aritméticas con ellos.

- `Cadena`: son los datos que están compuestos por un conjunto de letras del alfabeto, dígitos y caracteres especiales, incluyendo el espacio en blanco. Los datos de tipo cadena deben encerrarse entre comillas dobles.

  Son ejemplos de datos de tipo cadena los siguientes: "Carrera 17 # 12 – 65", "Gato", "Jacobo Herrera", "75478923", "01800043433".

  Los dos últimos ejemplos podrían corresponder a un número de una cédula y al número de una línea de servicio 018000, aunque estos datos están conformados por dígitos, no pueden ser utilizados para hacer cálculos numéricos.

Datos numéricos

Corresponden a los datos que están compuestos por solo números y signos (positivo y negativo), es decir, dígitos del 0 al 9, con los cuales se pueden realizar operaciones aritméticas. Estos tipos de datos se subdividen en: Enteros y Reales.

- `Entero`: son aquellos datos compuestos por números que no tienen punto decimal. Pueden ser números positivos, negativos o el cero.

  Ejemplos de este tipo de datos pueden ser: 20, -5, 200, 1500000.

- `Real`: son datos con componente decimal, pueden ser positivos, negativos o cero.

  Ejemplos: 1.75, 4.5, 1800000.00, -234.00.

  Tenga presente que las unidades de mil no se deben separar con puntos o comas. Solamente se usa el punto para indicar la parte decimal.

Datos lógicos o booleanos

Son aquellos datos que toman solo uno de los dos posibles valores booleanos[4]: `Verdadero` o `Falso`. Estos valores son equivalentes a los dígitos del sistema binario: 1 corresponde a Verdadero y 0 a Falso [Mancilla et al., 2016]. Por ejemplo, supóngase que se necesita almacenar la respuesta suministrada por un paciente en una clínica sobre si fuma o no, o sobre si es alérgico o no a un tipo de medicamento. En ambos casos, se debería utilizar este tipo de datos para clasificar el dato suministrado por el paciente.

1.3. Identificadores

Un identificador es el nombre que se le asigna a las variables, constantes, funciones, procedimientos y al algoritmo[5]; esto se hace para que el algoritmo pueda identificar claramente cada uno de estos elementos.

Para asignar nombres válidos en un algoritmo a los elementos mencionados en el párrafo anterior, existen una serie de reglas que facilitan su escritura. Es importante mencionar también que, existen reglas y recomendaciones propias, dependiendo del lenguaje de programación en el que se vaya a codificar el algoritmo; de esta manera, la forma de asignar identificadores puede ser ligeramente diferente de un lenguaje a otro.

Para el caso de este texto, se utilizarán las siguientes reglas o recomendaciones para escribir identificadores:

- Definir identificadores nemotécnicos, es decir, alusivos o relacionados con la función del elemento que se está nombrando.

[4] Un valor booleano solo admite una de dos posibles respuestas que son mutuamente excluyentes: `Verdadero` o `Falso`.

[5] Estos conceptos se trabajarán de manera amplia más adelante en este capítulo.

- El primer carácter del identificador debe ser una letra.
- No utilizar caracteres especiales dentro de los identificadores como vocales tildadas, la letra ñ, o símbolos como: $, #, !, ?, entre otros.
- No se deben dejar espacios en blanco dentro del nombre de un identificador.
- No utilizar palabras propias del lenguaje algoritmico / programación que se está utilizando "Palabras reservadas"[6].
- En un identificador se pueden utilizar varias palabras, preferiblemente unidas. También se puede usar un guion bajo entre cada una de ellas
- Evite el uso de artículos y proposiciones, tales como: el, los, la, un, unos, a, para, de, entre otros.
- Los identificadores suelen tener reglas dependiendo del lenguaje, en general, se escriben en minúscula, cuando el identificador se componga de dos o más palabras, la primera letra a partir de la segunda deberá escribirse en mayúsculas.
- El identificador para el nombre del algoritmo, comienza en mayúscula.
- Si el identificador corresponde al nombre de una constante, este debe escribirse en mayúsculas.

1.4. Variables

Para almacenar los datos en un dispositivo de procesamiento de datos o computador, se utiliza la memoria de este, la cual se puede comparar con un conjunto de cuadritos que guardan valores (Ver Figura 1.1).

Aclaración:



Una variable es una posición o espacio de memoria en el cual se almacena un dato. Su valor puede cambiar en cualquier momento de la ejecución del algoritmo, precisamente por eso recibe el nombre de variable.

[6]La lista completa de las palabras reservadas se encuentra en el apartado correspondiente a pseudocódigo - página 61.

Cada “cuadrito” o celda representa una dirección física dentro de la memoria de la máquina a la cual se le puede asignar un nombre mediante un identificador.

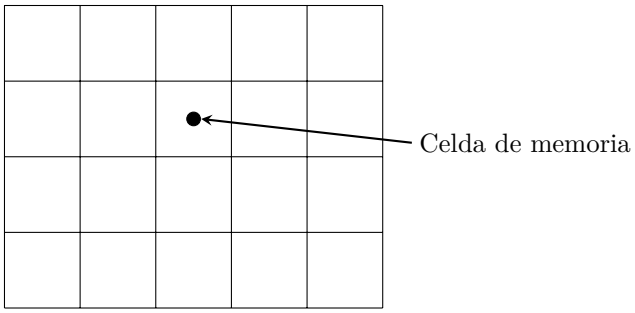


Figura 1.1: Representación gráfica de la memoria

Para trabajar con variables, se deben tener presentes los siguientes elementos:

- Tipo
- Nombre o identificador
- Contenido

El tipo se refiere al tipo de dato que va a almacenar. Puede ser uno de estos cinco: `Caracter`, `Cadena`, `Entero`, `Real` o `Logico`. Cando vaya a declarar el tipo lógico o carácter dentro de un algoritmo, no se deben usar las tildes.

El nombre o identificador de la variable, corresponde al mecanismo con el que se referencia el espacio o posición de memoria en el cual se almacenará el dato.

Contenido, hace referencia al valor que almacena, el cual depende del tipo de dato que se haya definido.

Declaración de variables

Cuando en un algoritmo se requiera utilizar una variable, esta debe ser declarada. Declarar una variable quiere decir que se va a reservar un espacio de memoria, el cual tendrá un nombre y un tipo de dato.

La forma general para declarar variables es la siguiente:

```
Tipo variable
```

Ejemplos:

```
Cadena    cedula
Cadena    telefono
Real      salario
Entero    edad
Caracter  estratoSocioeconomico
Logico    esFumador
```

Analizando las anteriores declaraciones, se puede observar y concluir lo siguiente:

- Se declararon 6 posiciones de memoria identificadas como `cedula`, `telefono`, `salario`, `edad`, `estratoSocioeconomico` y `esFumador`.
- Con estas declaraciones se le indica al equipo de procesamiento de datos o computador que debe reservar 6 espacios en su memoria y a cada uno asignarle el respectivo nombre que se definió.
- Cada uno de estos identificadores representa una variable.
- En el nombre de las variables se tuvieron en cuenta las reglas estudiadas anteriormente para la escritura de los identificadores. Por ejemplo, no se usaron tildes en las variables `cedula`, `telefono` y `estratoSocioeconomico`; en el identificador de esta última variable que está compuesta por dos palabras, la segunda inicia con letra mayúscula, al igual que no se dejó ningún espacio en blanco.
- En cada una de las declaraciones de variables se está determinando el tipo de dato que almacenarán.
- Para este libro y para lenguajes como C, el valor inicial de todas las variables no se conoce previamente y se denomina como "Información basura", es decir, toda variable declarada tiene un valor desconocido (arbitrario) y el diseñador del algoritmo debe asegurarse de asignarle un valor antes de intentar usar el contenido de la variable. Existen otros lenguajes de programación como por ejemplo Java que sí le asignan un valor por defecto a toda variable recién declarada.

Cuando en una declaración de variables se tengan varias del mismo tipo, estas pueden ser agrupadas en una sola declaración, separándolas mediante comas. Ejemplo:

```
    Cadena cedula, telefono
```

Almacenamiento de un dato en una variable

Almacenar un dato en una variable, se puede hacer de dos maneras:

La primera forma es leyendo el dato, proveniente desde el exterior del algoritmo, el cual lo proporciona el usuario; por ejemplo, cuando un cajero automático le solicita la clave de su tarjeta débito, el usuario le está asignando un valor a la variable donde se almacenará dicha clave, el valor será la clave que se digitó. Cuando se trate el tema de algoritmos, más adelante en este capítulo, se indicará la forma de leer los datos.

La segunda forma se hace a través de una expresión de asignación. Una expresión de asignación, es el mecanismo por medio del cual una variable o una constante toma un valor. Para realizar una asignación se utiliza el signo igual (=).

Su sintaxis o forma general es la siguiente:

```
    variable = valor
```

Esta forma indica que `valor` será almacenado en `variable`. En este caso, `valor` puede ser una constante, otra variable o una expresión aritmética (cálculo).

Teniendo en cuenta los ejemplos dados anteriormente en la declaración de variables, se procederá a asignar valores a cada uno de los espacios de memoria declarados:

```
    cedula    = "41459822"
    telefono  = "7454548"
    salario   = 3000000.00
    edad      = 24
    esFumador = Verdadero
    estratoSocioeconomico = '3'
```

Los valores que hay al lado derecho del signo igual (=), son asignados a cada una de las variables que están al lado izquierdo, de esta forma en cada uno de los espacios de memoria reservados se guardarán estos valores, los cuales pueden ser referenciados usando el nombre de la variable.

Observe que los valores asignados a las variables `cedula` y `teléfono` están encerrados entre comillas dobles, esto obedece a que las variables fueron definidas de tipo `Cadena`. En el caso de `estratoSocioeconomico` el valor que se le asigna está entre comillas simples, debido a que es una variable de tipo `Caracter`. No olvide que, aunque estos datos están compuestos exclusivamente por dígitos no se pueden tratar como valores numéricos, es decir, con ellos no se podrán realizar operaciones matemáticas.

Como se dijo anteriormente, en la expresión de asignación se pueden usar valores provenientes de otras variables o de expresiones aritméticas; adicionalmente recuerde que una variable puede cambiar su valor en cualquier momento. Ejemplo:

```
1   a = 5
2   b = a
3   c = 3 * b
4   a = c - b
```

- Línea 1: la variable `a` toma el valor de 5, proveniente de un valor constante.
- Línea 2: la variable `b` toma el valor de `a`, es decir, en este momento tanto `a` como `b` tienen almacenado el valor de 5. En este ejemplo, la asignación se hace tomando el valor de otra variable. Cuando se hace referencia al nombre de la variable, realmente se está referenciando su contenido.
- Línea 3: la variable `c` toma el valor de 15, al multiplicar el valor de `b` por 3. En este caso la asignación proviene de una expresión aritmética.
- Línea 4: la asignación almacena en la variable a el valor de 10, puesto que `c` tiene almacenado el valor de 15 y le es restado el valor de `b`, que en este caso es 5. Esta línea muestra una asignación realizada como resultado de una expresión aritmética. Así mismo se comprueba que una variable puede cambiar de valor; en la línea 1 la variable `a` tenía el valor de 5 y al finalizar, en la línea 4, esa misma variable termina con un valor de 10. Cada que una variable cambia su valor, el dato anterior se pierde.

1.5. Constantes

Una constante es un espacio en la memoria donde se almacena un dato que, a diferencia de los datos que se almacenan en las variables, permanece constante durante la ejecución de todo el algoritmo. Al igual que las variables, las constantes también se deben declarar.

Para declarar una constante se utiliza la siguiente forma general o sintaxis:

```
Constante Tipo IDENTIFICADOR = Valor
```

- La palabra `Constante` es una palabra reservada que indica que se va a declarar una constante y que su valor no podrá ser cambiado durante la ejecución del algoritmo.
- Tipo, refiere a uno de los cinco tipos de datos: `Caracter`, `Cadena`, `Entero`, `Real` y `Logico`.
- IDENTIFICADOR, es el nombre que se le asigna a la constante. Se recomienda que se escriba en letras mayúsculas.
- `Valor`, es el valor que tendrá la constante y debe estar acorde con el tipo de dato que se le haya asignado a la constante.

Ejemplos:

```
Constante Real      VALOR_PI = 3.1415926
Constante Real      DESCUENTO = 0.10
Constante Entero    MAXIMO = 10
Constante Real      SALARIOMINIMO = 781000.0
Constante Caracter CATEGORIAPORDEFECTO = 'B'
```

1.6. Operadores y expresiones

Un operador es un símbolo que permite realizar una operación con números o con datos que se encuentran almacenados en las variables y constantes. En lógica de programación, existen 3 tipos de operadores: aritméticos, relacionales y lógicos.

Por su parte, una expresión es una instrucción que puede estar compuesta por operadores, variables, constantes y números, que generalmente produce un resultado, ya sea numérico o lógico.

Las expresiones para ser usadas dentro de un algoritmo, deben escribirse en notación algorítmica (en una sola línea), para ello se seguirá usando la siguiente forma general:

```
    Operando1 Operador Operando2
```

`Operando1` y `Operando2` representan los datos que intervienen en la expresión y, `operador` es un símbolo que indica el tipo de operación que se va a realiza entre los operandos.

1.6.1 Operadores aritméticos

Se utilizan para realizar operaciones aritméticas entre datos de tipo entero o real, su resultado es de tipo numérico. Los operadores aritméticos son los siguientes:

- + Suma
- - Resta
- * Multiplicación o producto
- / División real o entera
- % Módulo o Resto de la división entera
- ^ Potenciación, este símbolo tiene el nombre de circunflejo.

La suma, resta, multiplicación, división y potenciación son operaciones que son conocidas desde los estudios de primaria y bachillerato. Aunque en el anterior listado la división se está mostrando como división real y división entera.

Una división real es la que el resultado arroja valores con decimales, generalmente, la que siempre se realiza. Por ejemplo, al dividir 15.0 entre 2.0, se obtendrá como resultado 7.5.

Diferente a lo anterior, la división entera no es tan usual y consiste en obtener el cociente entero, es decir, sin decimales. Por ejemplo, al dividir 15 entre 2, se obtendrá como resultado 7.

La especificación del tipo de división que se necesita realizar (real o entera), radica en el tipo de dato de los operandos. Si uno o ambos operandos son de tipo real, el resultado que se obtiene es del mismo tipo; si ambos operandos son enteros, la división arroja un resultado entero:

$15.0/2.0 = 7.5$, ambos operandos reales, resultado real.

$15/2 = 7$, ambos operandos enteros, resultado entero (sin decimales).

Otro operador que puede ser nuevo para el lector, es el % (residuo, resto o módulo de la división entera), no se debe confundir con el símbolo de porcentaje, en este caso se trata de un operador de división entre datos de tipo entero, pero no obtiene el cociente, sino el residuo o resto de la división entera:

$$\begin{array}{r|l} 15 & 2 \\ \hline 1 & 7 \end{array}$$

Para usarlo en un algoritmo, se representa así: 15 % 2, obteniendo 1 como resultado.

Aclaración:



Cuando se use el operador / o %, el operando de la derecha, no debería tener el valor de 0, ya que generaría un error, debido a que la división entre cero, matemáticamente, no está definida.

El operador % (Módulo o Resto de la división) solo se puede emplear con datos de tipo `Entero`.

El operador / puede usarse con datos enteros o reales. Si los datos son enteros, el resultado es entero, si alguno de los datos es real, el resultado será del mismo tipo (real).

1.6.2 Operadores relacionales

Estos operadores se utilizan para escribir expresiones relacionales o de comparación, las cuales producen un resultado lógico o booleano: `Verdadero` o `Falso`.

Los operadores relacionales son los siguientes:

- < Menor que.
- > Mayor que.
- <= Menor o igual a.
- >= Mayor o igual a.
- ! = Diferente de.
- == Igual a.

Aclaración:



Se usa un solo igual (=) para realizar asignación de valores en variables o constantes. Se utiliza doble igual (==) para realizar comparaciones entre dos operandos.

1.6.3 Operadores lógicos

Estos operadores se utilizan para crear expresiones lógicas o booleanas cuyo resultado es de tipo lógico: `Verdadero` o `Falso`.

Los operadores lógicos son los siguientes:

- `Y` Conjunción.
- `O` Disyunción.
- `NO` Negación.

Estos operadores funcionan con datos de tipo lógico; para obtener el resultado de la aplicación de estos operadores, es indispensable conocer cómo funcionan las tablas de verdad de cada uno de ellos.

Operador Y, denominado **Conjunción**. Es un operador binario, es decir, requiere de dos operandos para producir un resultado. El resultado es verdadero solo cuando ambos operandos son verdaderos (Ver Tabla 1.1).

*Operando*1	*Operando*2	*Operando*1 `Y` *Operando*2
`Verdadero`	`Verdadero`	`Verdadero`
`Verdadero`	`Falso`	`Falso`
`Falso`	`Verdadero`	`Falso`
`Falso`	`Falso`	`Falso`

Tabla 1.1: Tabla de verdad del operador Y

Operador O, denominado **Disyunción**. Igual que el anterior, es un operador binario. Su resultado será verdadero cuando al menos uno de los dos operandos tenga ese valor (Ver Tabla 1.2).

*Operando*1	*Operando*2	*Operando*1 O *Operando*2
Verdadero	Verdadero	Verdadero
Verdadero	Falso	Verdadero
Falso	Verdadero	Verdadero
Falso	Falso	Falso

Tabla 1.2: Tabla de verdad del operador O

Operador NO, denominado **Negación**. A diferencia de los dos anteriores, este es un operador unario, es decir, requiere de un solo operando para producir su resultado. Si el operando es Verdadero, cambia su estado a Falso, y viceversa. En conclusión, su función es cambiar el valor o estado lógico de su único operando (Ver Tabla 1.3).

*Operando*1	NO *Operando*1
Verdadero	Falso
Falso	Verdadero

Tabla 1.3: Tabla de verdad del operador NO

Aclaración:



Con el operador Y, el resultado es Verdadero cuando ambos operandos sean verdaderos.

Con el operador O, el resultado es Verdadero cuando uno de los operandos sea verdadero.

El operador NO, cambia el estado del operando, es decir, si el operando es Verdadero, el resultado será Falso y viceversa.

1.6.4 Expresiones aritméticas

En estas expresiones intervienen variables, constantes, números y operadores aritméticos, así como los paréntesis. La expresión entrega un resultado de tipo numérico luego de ser calculada.

Recuerde que estas expresiones deben ser usadas en notación algorítmica, teniendo en cuenta la siguiente forma general:

```
Operando1 Operador Operando2
```

En este caso `Operador`, es uno de los operadores aritméticos estudiados en el numeral 1.6.1. Son ejemplos de expresiones aritméticas los de la Tabla 1.4.

Ejemplo	Explicación	Resultado
3 + 5	Suma 3 con 5	8
2 - 8	Resta 8 de 2	-6
7 * 6	Multiplica 7 por 6	42
20.0 / 3.0	Divide 20.0 entre 3.0 (Real)	6.33
20 / 3	Divide 20 entre 3 (Entero)	6
2^3	Eleva 2 a la potencia 3	8
20%3	Resto de 20 entre 3	2

Tabla 1.4: Ejemplos de expresiones aritméticas

Las expresiones aritméticas también pueden ser construidas con variables o en combinación con constantes. Ejemplos Tabla 1.5.

Expresión	Explicación
$lado1 + lado2$	Se suman los valores almacenados en las variables *lado*1 y *lado*2.
`a - b`	Al valor de la variable `a` se le resta el valor de la variable `b`.
$base * altura$	Los valores almacenados en las variables *base* y *altura* son multiplicados entre sí.
$a \ / \ b$	El valor almacenado en la variable a es dividido entre el valor de la variable b. El resultado será entero, si ambas variables fueron declaradas de tipo `Entero`, en otro caso, será `Real` (con decimales).
x^2	El valor almacenado en la variable x es elevado a la 2.
25 % *divisor*	25 es dividido entre el valor de la variable *divisor*, el resultado será el resto de la división. La variable *divisor* deberá ser declarada de tipo `Entero`.

Tabla 1.5: Ejemplos de expresiones aritméticas con variables

Cuando una expresión aritmética involucra varios operadores, es necesario realizar los cálculos respetando la precedencia de los mismos, es decir, se debe tener en cuenta lo que se denomina prioridad en las operaciones. La Tabla 1.6 muestra el orden de ejecución, la cual puede ser modificada con el uso de paréntesis, los cuales serían la máxima prioridad.

Orden	**Operador**
1	( )
2	^
3	*, /, %
4	+, -

Tabla 1.6: Prioridad de los operadores

Ejemplos:

```
10 + 3 * 5
```

Esta expresión presenta dos operaciones: una suma y una multiplicación. La prioridad la tiene el operador $*$, por lo tanto, debe ser la primera operación que se realiza:

```
 10 + 15
```

Seguidamente se efectúa la suma, dando como resultado `25`.

Si en el anterior ejemplo, no se hubiese tenido en cuenta el orden de ejecución, de acuerdo a la prioridad de operación, sino que se hubiese ejecutado en el orden como está escrita la expresión, se obtendría el siguiente resultado, que obviamente es un **error**:

```
10 +  3 * 5
13 * 5
65
```

Si en una expresión se encuentran `2` o más operadores consecutivos de igual jerarquía o precedencia, las operaciones se van realizando de izquierda a derecha. Por ejemplo:

```
10 - 6 + 2
4 + 2
6
```

En la anterior expresión se encuentra una resta y una suma, ambas son de igual precedencia, por lo tanto, se ejecutan de izquierda a derecha, primero la resta y luego la suma.

Cuando se requiera cambiar el orden de ejecución de algún operador, se deben usar los paréntesis. Por ejemplo, si en la anterior expresión lo que se buscaba era que primero se ejecutara la suma y luego que el resultado de esa suma le fuera restado al `10`, la expresión tendría que haber sido escrita así:

```
10 - (6 + 2)
```

Al calcular dicha expresión se tiene entonces:

```
10 - (6 + 2)
10 - 8
2
```

Ahora observe el siguiente ejemplo, asumiendo que `a = 17` y `b = 20`.

```
a % 3 + b
```

El valor de la variable `a` es dividido entre `3`, el residuo de esta división se suma con el dato contenido en la variable b. Por lo tanto, 17 al ser dividido entre 3 dará como residuo `2`, valor que es sumado con el dato que hay en `b`, es decir `20`, obteniendo `22` como resultado final.

Ahora bien, si esta expresión se escribe de la siguiente forma:

```
a % (3 + b)
```

El resultado sería diferente. Observe:

```
17 % (3 + 20)
17 % 23
17
```

El residuo es 17:

$$\begin{array}{r|l} 17 & 23 \\ \hline 17 & 0 \end{array}$$

Cuando se tengan varias agrupaciones de paréntesis, se deben solucionar de izquierda a derecha, teniendo en cuenta que, si hay paréntesis internos, estos son de mayor prioridad. Analice la expresión que aparece enseguida:

```
((4 + 8) * 3) / (7^2 % (2 + 1))
```

En esta expresión, la máxima prioridad será resolver los paréntesis antes de llevar a cabo la división que los separa; pero como hay dos juegos de paréntesis, se deben resolver de izquierda a derecha.

```
((4 + 8) * 3) / (7^2 % (2 + 1))
```

Dentro del primer paréntesis hay una suma y un producto; la prioridad la tendría el producto, pero los paréntesis que encierran la suma deben resolverse primero, lo cual cambia el orden de operación, debiéndose desarrollar en primer lugar la suma:

```
((4 + 8) * 3) / (7^2 % (2 + 1))
(12 * 3) / (7^2 % (2 + 1))
```

Ahora si es posible llevar a cabo el producto dentro del paréntesis del lado izquierdo de la división, obteniendo el siguiente resultado:

```
36 / (7 ^ 2 % (2 + 1))
```

A continuación, se debe resolver la expresión que está entre el paréntesis del lado derecho de la división. Dentro de él se encuentra una operación de potencia, una división modular y una suma. Según la jerarquía de los operadores aritméticos (Tabla 1.6) primero debe resolverse la potencia, luego la división y por último la suma, pero los paréntesis internos cambian el orden de ejecución, por lo tanto, lo primero que se va a ejecutar es la suma:

```
36 / (7 ^ 2 % (2 + 1))
```

La expresión se reduce de la siguiente manera, donde la prioridad la tiene la potencia:

```
36 / (7 ^ 2 % 3)
```

Al resolverla, la expresión se presenta así:

```
36 / (49 % 3)
```

Aún hace falta resolver la división modular para eliminar los paréntesis:

```
36 / 1
```

Por último, se realiza la división, lo que da como resultado `36`.

1.6.5 Conversión de fórmulas aritméticas en notación algorítmica

Cuando se va a trabajar con una fórmula dentro de un algoritmo, se debe convertir a una expresión aritmética que sea interpretada por los

dispositivos de procesamiento de datos, esta conversión se le conoce como Notación algorítmica. Generalmente las fórmulas son encontradas en forma de notación matemática:

$$a = \frac{2x + y}{x - y}$$

Pero para poder ser trabajada dentro de un algoritmo debe reescribirse en una sola línea, así:

```
a = (2 * x + y) / (x - y)
```

Para llegar a esta notación algorítmica, es necesario tener en cuenta las siguientes reglas:

- **Primera regla**. Cuando la fórmula exprese una división y en alguno de sus términos (dividendo o divisor) se tenga una operación, esta debe encerrarse entre paréntesis. **Ejemplo:**

$$x = \frac{a+1}{b} \text{ ,debe expresarse así: } x = (a + 1)/b$$

- **Segunda regla**. Cuando se tenga una potencia y su exponente involucre una operación, debe encerrarse entre paréntesis. **Ejemplo:**

$$x^{3b}, \text{ debe expresarse así: } x\hat{}(3 * b)$$

- **Tercera regla**. Cuando la fórmula incluya raíces, estas deben expresarse en forma de potencias, aplicando el siguiente concepto matemático:

$$\sqrt[n]{x} = x^{1/n}$$

Ejemplo:

$$\sqrt[3]{\sqrt[5]{x}}, \text{debe expresarse así: } x\hat{}(1/5)\hat{}(1/3)$$

Expresión	Resultado	Explicación
$4 < 8$	`Verdadero`	Compara si 4 es menor que 8.
$3 > 7$	`Falso`	Compara si 3 es mayor que 7.
$6 <= 6$	`Verdadero`	Compara si 6 es menor o igual a 6.
$2 >= 9$	`Falso`	Compara si 2 es mayor o igual a 9.
$5 != 1$	`Verdadero`	Compara si 5 es diferente de 1.
$10 == 10$	`Verdadero`	Compara si 10 es igual a 10.

Tabla 1.7: Ejemplo de expresiones relacionales

1.6.6 Expresiones relacionales

En este tipo de expresiones intervienen los operadores relacionales, además de variables, números y constantes. Estas expresiones comparan el valor de sus operandos y arrojan un resultado de tipo lógico: `Verdadero` o `Falso`. Los operadores relacionales fueron estudiados en el numeral 1.6.2.

La Tabla 1.7 presenta ejemplos de expresiones relacionales.

Los operandos que hacen parte de una expresión relacional, pueden estar compuestos por constantes, variables o la combinación de ambos. Ejemplos:

```
definitiva >= 3.0
```

Aquí se está preguntando si el valor contenido en la variable `definitiva` es mayor o igual a `3.0`. La expresión retornará un resultado de `Verdadero` o `Falso`, dependiendo de lo que tenga almacenado la variable `definitiva`.

```
(2 * a + b) > (3 * c - d)
```

Para que esta expresión sea verdadera, la parte izquierda de la expresión, es decir, lo que hay del lado izquierdo del operador $>$ debe ser mayor que lo que hay del lado derecho. Nótese que en ambos lados de la expresión hay expresiones aritméticas que involucran variables y números. En estos casos, también se debe aplicar la prioridad en las operaciones.

1.6.7 Expresiones lógicas

También llamadas booleanas. Son aquellas que se utilizan para crear condiciones a partir de datos lógicos. Están compuestas por expresiones relacionales o lógicas, conectadas a través de operadores lógicos, los

cuales fueron estudiados en el numeral 1.6.3. Los resultados que arroja la evaluación de este tipo de expresiones solo pueden ser de dos formas: `Verdadero` o `Falso`.

Para ilustrar el funcionamiento de estos operadores, se analizarán los siguientes ejemplos.

Ejemplo con el operador `Y`: para poder hacer cualquier transacción en un cajero automático se requiere poseer una tarjeta (débito o crédito) y adicionalmente la clave de dicha tarjeta; actualmente existen otros medios, pero para el ejemplo se tomarán esas condiciones.

A continuación, se analizará en qué situaciones se podría hacer alguna transacción y en cuales no:

1. Posee la tarjeta y la clave de ella, podrá realizar transacciones.
2. Posee la tarjeta, pero no la clave, por lo tanto, no podrá realizar transacciones.
3. No posee la tarjeta y posee la clave, no podrá realizar transacciones.
4. No posee la tarjeta, ni la clave, por consiguiente, no podrá realizar transacciones.

Para expresar este ejemplo con el operador `Y`, primero que todo recuerde la forma general de una expresión:

```
Operando1 Operador Operando2 \\
```

De acuerdo a esta forma general, el `operando1` estará representado por la `tarjeta` de crédito y el `operando2` será la `clave` de dicha tarjeta, el `operador` será la conjunción (`Y`).

Los valores de `Falso` o `Verdadero` para cada operando estarán dados de la siguiente forma: si se posee la `tarjeta` el valor será `Verdadero`, en caso contrario será `Falso`; para la `clave` se procede de la misma manera (Ver Tabla 1.8).

tarjeta	clave	tarjeta Y clave	Explicación
Verdadero	Verdadero	Verdadero	Hay transacción
Verdadero	Falso	Falso	No hay transacción
Falso	Verdadero	Falso	No hay transacción
Falso	Falso	Falso	No hay transacción

Tabla 1.8: Tabla de verdad del operador **Y** - Ejemplo

En conclusión, solo puede hacer la transacción cuando se cumplan las dos condiciones, tener la `tarjeta` y la `clave`, lo cual dará un resultado `Verdadero`.

Ejemplo con el operador O: para el próximo cumpleaños de una sobrinita, ella quiere que le regale una mascota, dice que estaría feliz si le regalo un gato o un perro, o ambos; pero que si no le doy la mascota sería infeliz en su día.

Analizando esta situación observe en cuáles casos estaría feliz y en cuáles no:

1. Le regalo tanto el perro como el gato, estaría feliz.
2. Solamente le doy el perro, estaría feliz.
3. Solamente le regalo el gato, estaría feliz.
4. Pensándolo bien, ella está muy niña para hacerse cargo de una mascota, no le doy el perro, ni el gato, estaría triste.

Para expresar este ejemplo con el operador `O`, recuerde la forma general de una expresión:

```
Operando1 Operador Operando2
```

De acuerdo a esta forma general, el `operando1` estará representado por el `perro` y el `operando2` será el `gato`, el operador será la disyunción (`O`).

Los valores de `Falso` o `Verdadero` para cada operando estarán dados de la siguiente forma: si le regalo el `perro` el valor será `Verdadero`, en caso contrario será `Falso`; para el `gato` se procede de la misma manera (Ver Tabla 1.9).

perro	gato	perro O gato	Explicación
Verdadero	Verdadero	Verdadero	La sobrina está feliz
Verdadero	Falso	Verdadero	La sobrina está feliz
Falso	Verdadero	Verdadero	La sobrina está feliz
Falso	Falso	Falso	La sobrina no está feliz

Tabla 1.9: Tabla de verdad del operador **O** - Ejemplo

Concluyendo, con el operador **O**, el resultado será verdadero con solo cumplir una de las condiciones.

Ejemplo con el operador NO: con este operador el asunto es más sencillo; su función es cambiar el estado lógico de su único operando.

```
NO (5 < 10)
```

esta expresión se debe leer: es falso que 5 sea menor que 10.

El resultado que arroja esta expresión es Falso. La expresión relacional (5 <10) es verdadera, al ser negada con el operador NO, cambia su estado de Verdadero a Falso.

1.6.8 Prioridad de operación

Las expresiones lógicas también pueden combinar expresiones relacionales. En el momento de hacer su evaluación, al igual que con los operadores aritméticos, se debe respetar la precedencia y para ello en la Tabla 1.10 se dará el orden de ejecución de todos los operadores, incluyendo los paréntesis.

En los siguientes ejemplos se ilustrará la aplicación de la tabla. Suponga la siguiente asignación de valores:

```
a = 5
b = 10
c = 15
```

¿Cuál sería el resultado de la siguiente expresión lógica?

```
  a  <  b  Y   b <  c
  5  < 10  Y  10 <  15
Verdadero Y Verdadero
     Verdadero
```

Orden	Operador
1	( )
2	- (signo)
3	^
4	*, /, %
5	+, -
6	<, <=, >, >=
7	==, ! =
8	NO (Negación)
9	Y (Conjunción)
10	O (Disyunción)
11	= (asignación)

Tabla 1.10: Prioridad completa de los operadores

Primero se evalúan las expresiones relacionales y luego el operador lógico (Y). Ambas expresiones relacionales son verdaderas, por lo tanto la expresión lógica es verdadera.

En el caso del ejemplo anterior no se usaron paréntesis, es una buena práctica usarlos para dar mayor claridad a las expresiones, aunque de acuerdo al orden de prioridad, no son requeridos. Esta expresión también puede ser expresada así:

```
    (a < b) Y (b < c)
```

Expresiones más complejas, también se evalúan de acuerdo a la prioridad expresada en la Tabla 1.10. Ejemplo:

```
(20 > 40 Y 2 <= 10) O (32 < 50 Y 20 <= 20)
```

Obsérvese que se tienen dos juegos de paréntesis separados por el operador lógico O. Para que el resultado final sea verdadero, uno de los juegos de paréntesis deberá tener un resultado verdadero. Sin embargo, cada juego de paréntesis tiene en su interior un operador Y que implica que ambas expresiones dentro de cada paréntesis deben ser verdaderas para entregar un resultado verdadero. La evaluación, de acuerdo a la prioridad, es la siguiente:

Primero se evalúa el paréntesis de la izquierda:

```
(20 > 40 Y 2 <= 10) O (32 < 50 Y 20 <= 20)
```

Dentro de él, la prioridad la tienen los operadores relaciones:

```
(Falso Y 2 <= 10) O (32 < 50 Y 20 <= 20)
```

Teniendo en cuenta que la expresión lógica está conectada por el operador `Y`, no hace falta la evaluación de la expresión relacional del lado derecho `(2 <= 10)`, porque se sabe que el resultado será falso. Sin embargo, se realizará con fines didácticos:

```
(Falso Y Verdadero) O (32 < 50 Y 20 <= 20)
```

La primera expresión lógica arroja un resultado falso:

```
Falso O (32 < 50 Y 20 <= 20)
```

Ahora se procede con la parte derecha del operador `O`.

```
Falso O (32 < 50 Y 20 <= 20)
Falso O (Verdadero Y 20 <= 20)
Falso O (Verdadero Y Verdadero)
Falso O Verdadero
```

El resultado final de esta expresión es verdadero.

Las expresiones lógicas también son muy útiles para definir intervalos:

```
(edad >= 18) Y (edad <= 24)
```

Para que esta expresión lógica entregue un resultado verdadero, el dato almacenado en la variable `edad` deberá ser mayor o igual a `18` y menor o igual a `24`.

1.7. Algoritmo

Un algoritmo es un conjunto de acciones o pasos finitos, ordenados de forma lógica y que se utilizan para resolver un problema o para obtener un resultado.

Si se detiene unos instantes a analizar esta definición, puede concluir que el uso de algoritmos es muy común en su vida diaria. Piense en algunas de las tareas que realiza desde el momento en que se levanta hasta que se vuelve a acostar. Ese conjunto de tareas lo hace de forma mecánica y repetitiva, pero por lo general siempre ejecuta los mismos pasos. Por ejemplo, cuando se baña, cepilla sus dientes, se dirige al trabajo o la universidad, enciende su computador, hace una llamada desde su celular o busca alguna información en Google; son tan normales que pasan inadvertidas.

En este libro se explicará la forma de plasmar un conjunto de pasos que debe llevar un algoritmo para la solución de problemas que pueden

resolverse mediante el uso de un computador en el momento que se traduzcan a un lenguaje de programación.

Para lograr estas soluciones, los pasos de los algoritmos no pueden pasar inadvertidos como se mencionó anteriormente que sucedía con los que se ejecutan en sus actividades diarias. Este tipo de soluciones requieren de un análisis detallado de la información que se posee y del objetivo o resultados que se pretenden alcanzar.

Como resultado de esos análisis se diseñarán algoritmos que deben cumplir con tres de sus características fundamentales, deben ser ordenados, definidos y finitos [Pimiento, 2009] y [Joyanes A., 1996].

- **Ordenado**: el orden de ejecución de sus pasos o instrucciones debe ser riguroso, algunos tendrán que ser ejecutados antes de otros, de manera lógica, por ejemplo, no se podrá imprimir un archivo, si previamente no se ha encendido la impresora y no se podrá encender la impresora si previamente no se tiene una. Cada uno de ellos debe ser lo suficientemente claro para que determine con exactitud lo que se debe hacer.
- **Definido**: si el algoritmo se ejecuta en repetidas ocasiones, usando los mismos datos, debe producir siempre el mismo resultado.
- **Finito**: todo algoritmo posee un inicio, de igual forma debe tener un final; la ejecución de sus instrucciones debe terminar una vez procese los datos y entregue resultados.

Adicionalmente, el algoritmo debe plantear soluciones generales, es decir, que puedan ser utilizadas en varios problemas que tengan las mismas características.

1.7.1 Clasificación de los algoritmos

Existen diferentes clasificaciones de los algoritmos. Algunos autores contemplan la categoría de algoritmos matemáticos y computacionales [Echeverri and Orrego, 2012]. Otros autores, por su parte los dividen en algoritmos computacionales y no computacionales, también denominados algoritmos informales [Corona and Ancona, 2011] y [Trejos, 2017].

- **Algoritmos matemáticos**: mediante un conjunto de pasos, describen como se realiza una operación matemática.

- **Algoritmo informal**: son aquellos que son ejecutados por el ser humano. Por ejemplo, cepillarse los dientes o preparar un alimento. Aunque muchos de las actividades que en la actualidad se pueden numerar en esta categoría podrán cambiar debido a los adelantos tecnológicos [Trejos, 2017]. Hace algunos años conducir un vehículo era solamente una tarea propiamente humana, hoy en día existen vehículos autónomos que circulan por algunas grandes ciudades del mundo; de igual forma la búsqueda de una dirección no era algo que pudiera realizar un computador o un dispositivo de procesamiento de datos, en la actualidad existen los GPS o también las aplicaciones que son instaladas en los Smartphone que desarrollan esa tarea y permiten guiar a una persona hacia cualquier lugar.
- **Algoritmos computacionales**: son aquellos que se diseñan para que luego puedan ser ejecutados por un computador. Este libro está orientado al desarrollo de este tipo de algoritmos.

1.7.2 Representación de un algoritmo

Para representar algoritmos debe usarse una notación simple y lo suficientemente clara para que no se dé lugar a ambigüedades. Estas notaciones son muy útiles para entender la lógica de los problemas y poder luego traducirlas a cualquier lenguaje de programación de una manera simple y casi mecánica.

Dentro de las formas más utilizadas para representar algoritmos se encuentran las siguientes:

1. Descripción narrada.
2. Diagramas de flujo.
3. Pseudocódigo.

Descripción narrada

Los pasos o instrucciones se describen mediante un lenguaje natural, usando palabras o frases normales y corrientes. Su uso principal se da en el diseño de algoritmos informales. Son ejemplos de ellos:

- Indicar como se llega a una dirección.
- Preparar una deliciosa receta de cocina.
- Registrarse en Netflix para disfrutar de su contenido.
- Solicitar la autorización para presentar un examen supletorio en una Universidad, ejemplo que se desarrollará a continuación.

.:Ejemplo 1.1. *Examen Supletorio*

En la Universidad UQ es común la presentación de exámenes supletorios, que son aquellos que se presentan dentro de los 5 días hábiles después de la realización oficial del examen. Para presentar un supletorio se debe solicitar autorización ante el Director del programa, el cual lo autorizará o no, dependiendo de que exista una debida justificación.

Una vez consultado el Director de uno de los Programas de la Universidad UQ, se pudo establecer el siguiente conjunto de pasos, que se deben de ejecutar para poder obtener la autorización de la presentación del examen supletorio.

Los pasos fueron definidos teniendo en cuenta que no necesariamente se puede obtener la autorización; esta puede ser negada si la justificación presentada no es suficiente motivo para la no presentación. También se tiene en cuenta que si ya pasaron los 5 días reglamentarios, no se puede hacer la solicitud. En el caso de que sea autorizada la presentación, se definieron los pasos hasta el momento que el estudiante presenta el supletorio.

```
1 Algoritmo ExamenSupletorio
2   Si está dentro del plazo reglamentario
3    a. Dirigirse a la dirección del Programa.
4    b. Solicitar autorización para presentarlo.
5    c. Presentar la justificación ante el Director
6    d. Si autorizan la presentación
7       i.   Descargar el recibo de pago (portal).
8       ii.  Cancelar el recibo.
9       iii. Presentar el recibo cancelado en la Dirección.
10      iv.  Recibir formato de autorización.
11      v.   Entregar formato de autorización al Profesor.
12      vi.  Presentar examen supletorio.
13  Si no está dentro del plazo reglamentario
14   a. No hacer la solicitud
15 FinAlgoritmo
```

Antes de entrar en detalle con los diagramas de flujo y pseudocódigos, que son las formas de más uso para la representación de algoritmos computacionales, se debe establecer que este tipo de algoritmos siguen un patrón de ejecución en 3 etapas (Ver Figura 1.2).



Figura 1.2: Patrón de ejecución de un algoritmo

- **Entrada**: en esta etapa se le proporciona al algoritmo los datos que se poseen del problema y que son necesarios para su solución.
- **Proceso**: hace referencia a los pasos, actividades, instrucciones o cálculos que realiza el algoritmo para solucionar el problema o encontrar un resultado. Generalmente, en esta etapa se transforman los datos de entrada en resultados de salida.
- **Salida**: es la entrega de resultados o la respuesta dada por el algoritmo.

Ahora conocerá como es que los diagramas de flujo y el pseudocódigo se ocupan de hacer la representación de este esquema de entrada, proceso y salida.

Diagrama de flujo

Es la representación gráfica de un algoritmo. Lo conforma un conjunto de componentes que permiten representar acciones, decisiones o cálculos con los cuales se soluciona un problema determinado. Cuando el diagrama de flujo está correctamente diseñado, la concepción de un programa en un lenguaje de programación, en fácilmente codificable.

Los gráficos que se usarán en este texto, para representar los pasos o instrucciones son los presentados en las Tablas 1.11 y 1.12.

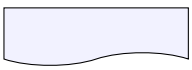
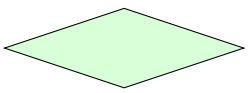
Símbolo	Nombre	Explicación
	**Terminal**	Representa el inicio y el final del algoritmo. Se rotula con la palabra Inicio o la palabra Final. En cada algoritmo solo puede estar presente un inicio y un final.
	**Entrada**	Permite la interacción con el entorno, a través de este símbolo el algoritmo recibe datos. Indica lectura de datos.
	**Proceso**	Se usa para indicar la ejecución de una acción.
	**Salida**	Permite la interacción con el entorno, a través de este símbolo muestra resultados. Indica escritura.
	**Decisión**	Indica una toma de decisiones. Se rotula con una expresión relacional o lógica, dependiendo de su resultado se toma un camino de ejecución. Se usa en decisiones, condiciones de las instrucciones `Mientras-FinMientras` y `Haga-MientrasQue`.

Tabla 1.11: Elementos en un digrama de flujo - Parte 1

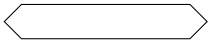
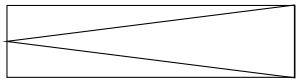
Símbolo	Nombre	Explicación
	**Decisión múltiple**	Permite decidir que paso se ejecutarán dentro de un conjunto de acciones predeterminadas.
	**Estructura Para**	Se usa para establecer procesos repetitivos.
	**Procedimiento**	Permite dividir el algoritmo en módulos independientes, que realizan una tarea indispensable en el resultado del mismo [Pimiento, 2009].
	**Conector misma página**	Permite conectar dos partes del diagrama en una misma página. Siempre se utilizan en pares, uno de salida y otro de entrada.
	**Conector otra página**	Cuando el diagrama es muy extenso, permite conectar dos partes del diagrama en páginas diferentes. Igual que el anterior, se utilizan en pares.
	**Líneas de flujo**	Indican el orden y la dirección en que las instrucciones se deben de ejecutar.

Tabla 1.12: Elementos en un digrama de flujo - Parte 2

Para el uso de estos símbolos se deben tener presentes las siguientes consideraciones:

Terminal

En un diagrama deben existir solamente dos de estos símbolos, uno rotulado con la palabra `Inicio` y otro con la palabra `Final` (Ver Figuras 1.3 y 1.4).

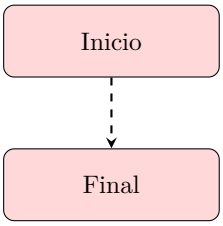


Figura 1.3: Terminal

Del que se rotula como `Inicio` solo puede salir una única línea de flujo. El que está rotulado como `Final`, recibe una única línea de flujo.

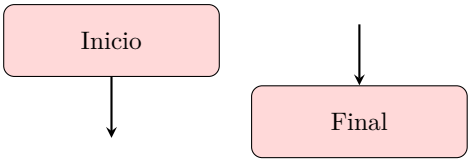


Figura 1.4: Elementos Terminal

Buena práctica:



El gráfico del `Inicio` debe quedar de primero en el diagrama, y el del `Final` debe ser la última parte del mismo.

Entrada

A este símbolo entra y sale una única línea de flujo (Ver Figura 1.5). Se rotula con el identificador de la variable que recibirá el valor que proporcione el usuario del algoritmo. La variable debe ser uno de los datos disponibles para la solución del problema.

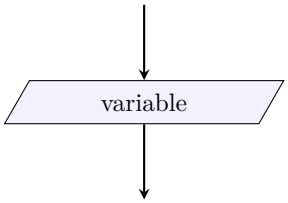


Figura 1.5: Entrada

Proceso

A este símbolo entra y sale una única línea de flujo (Ver Figura 1.6). Se rotula con la instrucción que se vaya a ejecutar, puede ser, por ejemplo, una instrucción de asignación.

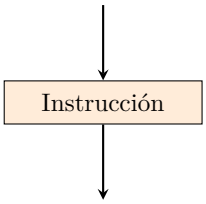


Figura 1.6: Proceso

Salida

A este símbolo entra y sale una única línea de flujo (Ver Figura 1.7). Se utiliza para mostrar resultados o mensajes. Dentro de él se puede escribir el nombre de una variable o una constante (Figura 1.7(a)), una operación matemática (Figura 1.7(b)), un mensaje con alguna variable o constante (Figura 1.7(c)) o combinar varios de estos elementos (salida Figura 1.7(d)).

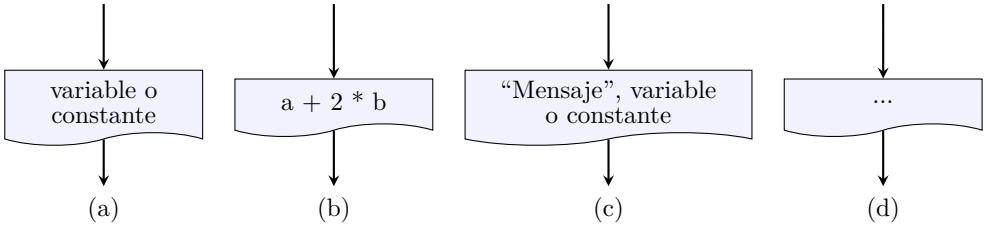


Figura 1.7: Salida

Los mensajes deben escribirse dentro de comillas dobles (“”) y se mostrarán exactamente como fueron escritos. En el caso de la variable o constante, que no lleva las comillas, se mostrará el valor que tenga almacenado. En las operaciones matemáticas, tampoco se usan las comillas y se mostrará el resultado de la operación.

Decisión o bifurcación

A este rombo debe llegar una única línea de flujo y salir dos, una que indica el camino verdadero y otra que indica el camino falso (Ver Figura 1.8).

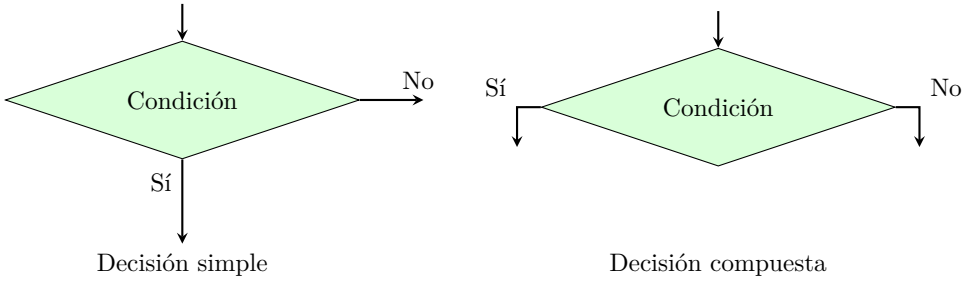


Figura 1.8: Decisiones

El símbolo se rotula con una condición que se representa mediante una expresión relacional o lógica, dependiendo de su resultado se elige uno de dos caminos. Las líneas de flujo que salen de él, deben rotularse con la palabra `Sí` para el camino a seguir cuando la condición es verdadera y con la palabra `No`, para el caso contrario.

Una decisión tiene varias configuraciones: **simple**, **compuesta** y **anidada**. Este símbolo también es usado para establecer la condición en los procesos repetitivos, los cuales se estudiarán en detalle en el capítulos posteriores.

- **Decisión simple**: cuando la condición arroja un resultado verdadero se ejecutan una o más instrucciones que estén asociadas al flujo rotulado con la palabra Si (`Instrucción-V`) . Si el resultado es falso, no se lleva a cabo ninguna acción que dependa de la condición. Ver Figura 1.9.

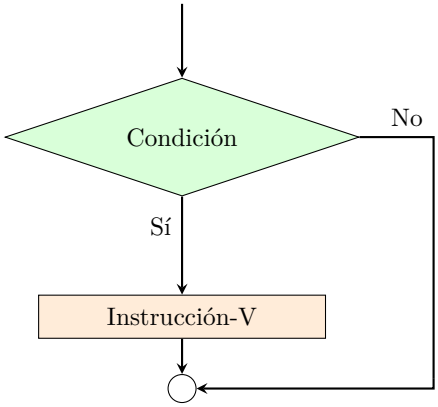


Figura 1.9: Decisión simple

- **Decisión compuesta**: si el resultado de la condición es verdadero se ejecutan una o más instrucciones asociadas al flujo rotulado con la palabra Si (`Instrucción-V`), en caso contrario se deben ejecutar una o más instrucciones asociadas al flujo rotulado con la palabra No (`Instrucción-F`). Ver Figura 1.10.

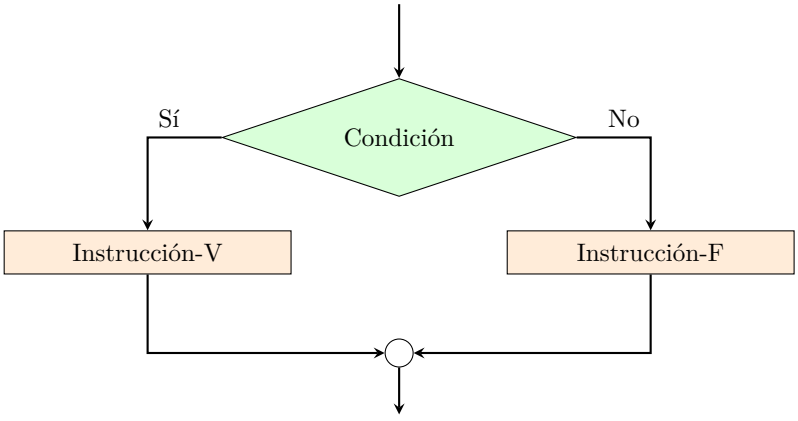


Figura 1.10: Decisión compuesta

- **Decisión anidada**: cuando se deben tomar otras decisiones, dependiendo del resultado de una decisión anterior. Ver Figuras 1.11 y 1.12.

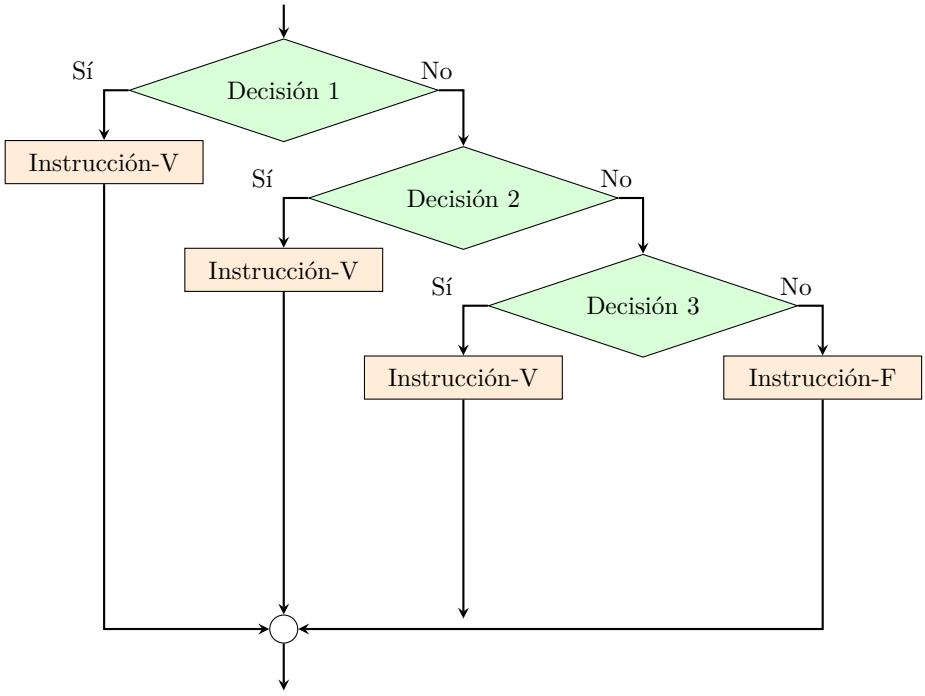


Figura 1.11: Decisión anidada - Ejemplo 1

Aclaración:



Este tipo de decisión se puede presentar en múltiples formas, todo depende del problema que se está solucionando.

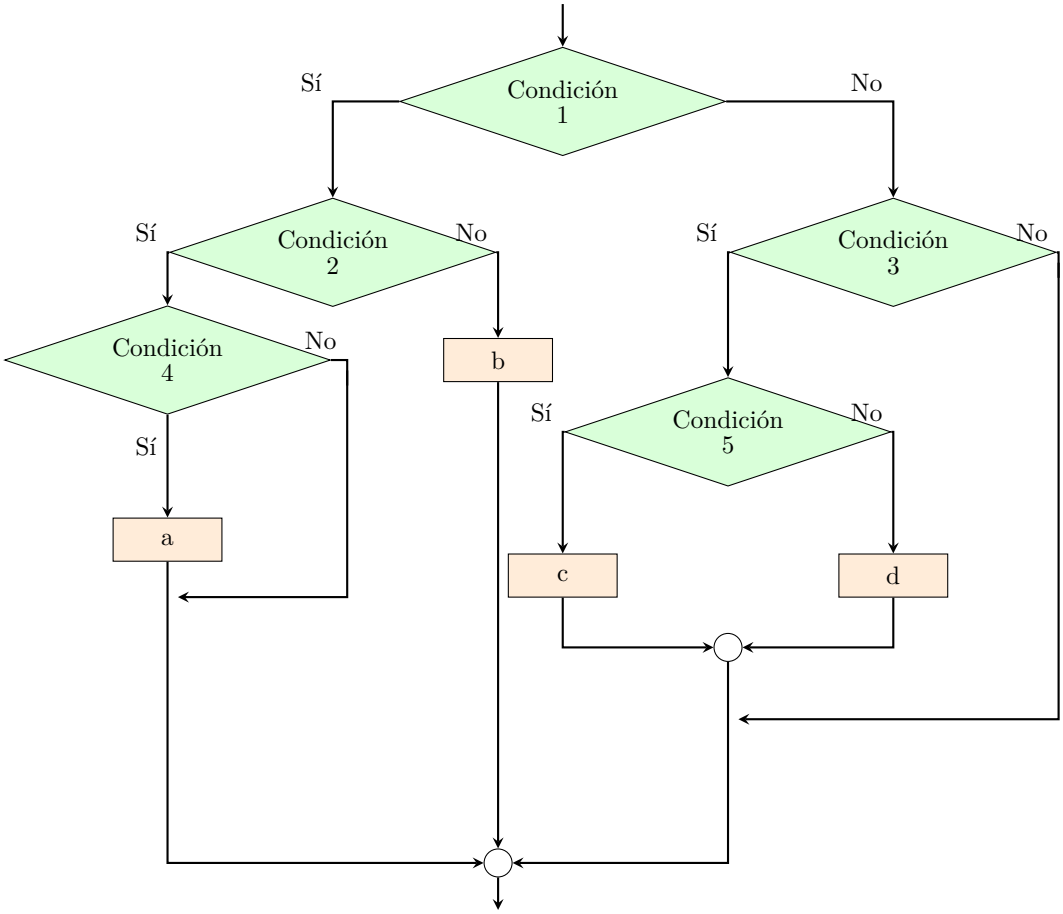


Figura 1.12: Decisión anidada - Ejemplo 2

Buena práctica:



De todo símbolo de decisión debe salir una línea de flujo por la parte verdadera (`Si`) y otra por la parte falsa (`No`)

Selector o decisión múltiple

Dependiendo del valor de una variable o expresión que se usa para rotular el símbolo, se elige uno de varios caminos posibles y se ejecutan la o las instrucciones correspondientes. [En otro caso] indica que si el valor de la

variable selector no es igual a ninguno de los valores relacionados (`valor 1`, `valor 2`,..., `valor n`) se ejecuta la instrucción por defecto. Ver Figura 1.13.

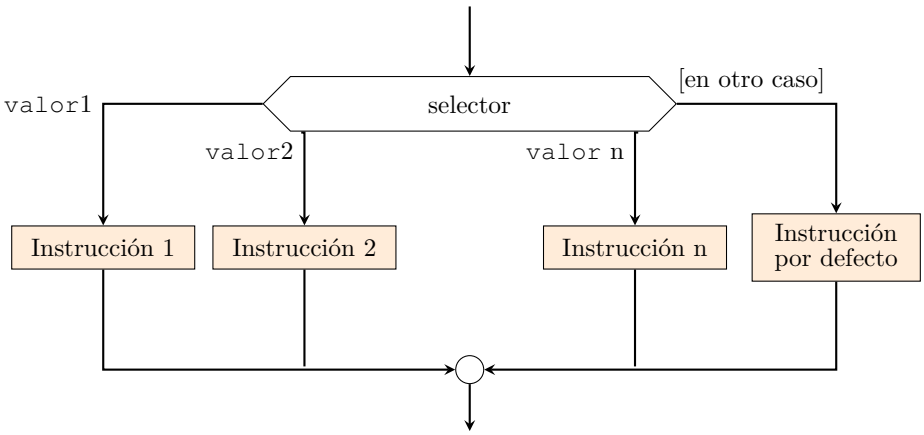


Figura 1.13: Decisión múltiple

Conector misma página

Los símbolos que reciben la línea de flujo representan la salida y deben emparejar con otro de los mismos símbolos del que sale la línea de flujo. Se deben rotular con una letra o un número, que se repetirá tanto en el conector de salida, como en el de entrada. Con el uso de los conectores se evita el cruce de líneas de flujo o dibujar líneas de flujo demasiado largas.

En un diagrama pueden existir varios conectores de salida con el mismo rótulo, pero solamente debe haber uno de entrada.

En la Figura 1.14 se observa que se pueden usar de forma vertical o de forma horizontal.

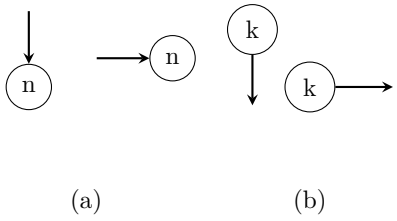


Figura 1.14: Conector misma página

Este símbolo también se puede usar como punto de concentración de varias líneas de flujo, en cuyo caso no se rotula y se usa uno solo. Pueden llegar varios flujos de entrada, pero solamente habrá un flujo de salida.

Conector otra página

Tiene las mismas características del conector anterior, la diferencia radica en que este se usa para hacer conexiones de partes del diagrama que están dibujadas en páginas diferentes. Ver Figura 1.15.

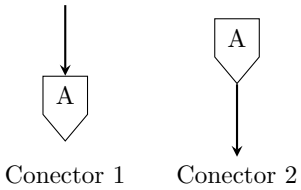


Figura 1.15: Conector otra página

El uso detallado de estos y los demás símbolos, se estudiará a lo largo de los siguientes capítulos.

Ahora que ya conoce los símbolos de un diagrama de flujo, recuerde que un algoritmo consta de 3 etapas: Entrada, Proceso y Salida. Dentro de la forma general de algoritmo expresado mediante un diagrama de flujo, estas etapas se pueden representar como se aprecia en la Figura 1.16.

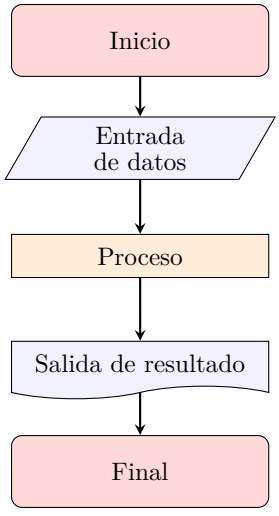


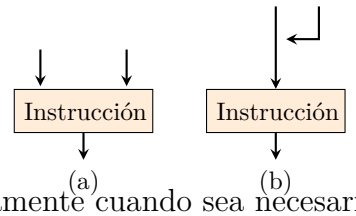
Figura 1.16: Diagrama de flujo (forma general)

Buenas prácticas:



Para la construcción de los diagramas de flujo, se deben seguir algunas reglas o recomendaciones.
- El diagrama de flujo debe tener un nombre que identifique su función.
- Los diagramas de flujo no tienen declaración de variables, ellas simplemente se usan.
- Construya el diagrama de arriba hacia abajo y de izquierda a derecha.
- Debe existir solamente un símbolo de inicio y uno de final, ambos debidamente rotulados.
- Dibuje únicamente líneas de flujo en rectas horizontales o verticales. No se deben usar líneas inclinadas.
- Toda línea de flujo debe estar conectada a uno de los símbolos o a otra línea de flujo, no pueden haber líneas sueltas, es decir, sin conexión.
- Evite pasar una línea sobre otra, en caso de que no sea posible, debe usar un conector a la misma página.
- Ningún símbolo puede recibir más de una línea, exceptuando el utilizado en la estructura repetitiva `Para` y el `conector a la misma página` cuando se usa como punto de concentración para las líneas de flujo. El funcionamiento de la estructura `Para` se estudiará de forma detallada en el Capítulo 4.

Por ejemplo, un proceso que recibe dos líneas de flujo es un error (Figura (a)); la forma correcta de diagramar este caso, es uniendo todos los flujos a una sola línea y que esta sea la que conecte al símbolo, tal como se aprecia al lado derecho de la gráfica (Figura (b)).



- Use conectores solamente cuando sea necesario.
- En el caso que el diagrama sea muy extenso, use conectores, bien sea dentro de la misma página o a página diferente. Se recomienda que los conectores a la misma página se identifiquen con números y los conectores a página diferente usen letras, o viceversa.
- A los símbolos de decisión se le deben dibujar las dos líneas de salida, especificando cual es la verdadera y cual es la falsa.

Pseudocódigo

La palabra Pseudocódigo, proviene de pseudo[7] y código, que tomado literalmente significa un falso código o dicho en otras palabras una especie de código. Se le ha dado esta denominación porque no es un lenguaje de programación, sino otra de las formas de expresar un algoritmo, usando palabras similares a las propias de los lenguajes de programación.

El pseudocódigo está compuesto por un conjunto de palabras en español, inglés o cualquier otro idioma[8] (códigos) que representan una instrucción que es entendida de una manera específica por el algoritmo en la solución de un problema. Estos códigos son fácilmente traducidos a un lenguaje de programación para que puedan ser interpretados y ejecutados por un computador.

Las palabras o códigos que se utilizan en el pseudocódigo no tienen un estándar, por lo tanto, pueden variar de una fuente de información a otra. Sin embargo, conservan mucha similitud a las características de las palabras reservadas y estructuras de los lenguajes de programación.

Palabras reservadas

Los códigos o palabras reservadas cumplen tareas específicas dentro de los algoritmos y no pueden usarse para propósitos diferentes. En este texto se trabajarán las siguientes:

- `Algoritmo` – `FinAlgoritmo`. Indica el comienzo y final del algoritmo, respectivamente. Cada algoritmo tendrá solo un inicio y un solo final.
- `Entero`, `Real`, `Cadena`, `Caracter`, `Logico`. Se usan para declarar el tipo de dato de las variables que se requieren en la solución del problema.
- `Constante`. Este código se emplea en el momento de declarar constantes.
- `Si`, `Entonces`, `SiNo`, `FinSi`. Se usan para la toma de decisiones simples y compuestas.
- `Segun`, `Caso`, `FinCaso`, `EnOtroCaso`, `FinSegun`. Utilizadas para la toma de decisiones múltiples.

[7] Del gr. *ψξνδο* - pseudo: significa "falso". http://dle.rae.es/?id=XkBx392

[8] Depende del idioma que se maneje en el país donde se enseñe. Para el caso de este texto, se usarán palabras en español.

- `Para`, `Hasta`, `Incremento`, `Decremento`, `FinPara`. Especifican una estructura repetitiva condicionada al comienzo.
- `Mientras`, `FinMientras`. Códigos para el manejo de una estructura repetitiva condicionada al comienzo.
- `Haga`, `MientrasQue`. Palabras usadas para diseñar una estructura repetitiva condicionada al final.
- `Procedimiento`, `FinProcedimiento`. Palabras reservadas empleadas para dividir el algoritmo en módulos independientes, que realizan una tarea indispensable en el resultado del mismo [Pimiento, 2009]
- `Funcion`, `FinFuncion`, `Retornar`. Permite hacer la división en bloques funcionales independientes que retornan valores.
- `dimensionar`. Se usa para declarar la dimensión en un vector o en una matriz.

Aclaración:



Los códigos o palabras reservadas cumplen tareas específicas dentro de los algoritmos y no pueden usarse para propósitos diferentes.

También existen las denominadas funciones del lenguaje algorítmico. Estas son funciones que ya están predefinidas para realizar diversas tareas, entre las más importantes se destacan las funciones matemáticas que permiten realizar diversos cálculos y, las funciones para el manejo de cadenas que se usan para el tratamiento de datos de tipo alfanumérico.

Aclaración:



Las funciones son también palabras reservadas y no pueden ser utilizadas para declarar variables, constantes, procedimientos, funciones de usuario e incluso los algoritmos.

Entre las funciones que serán trabajadas en los ejemplos y ejercicios de los siguientes capítulos se encuentran:

```
Entero longitud( Cadena )
Real   abs( Real o Entero )
Real   cos( Real )
Real   convertirANumero( Cadena )
Real   raizCuadrada( Real o Entero )
Real   ln( Real )
Real   log( Real )
Real   exp( Real )
Real   sen( Real )
Real   tan( Real )
```

Una de las palabras reservadas más utilizadas en los algoritmos en pseudocódigo, es la instrucción `imprimir`, por ello, tenga en cuenta estas consideraciones en su uso:

`imprimir`. Es una instrucción de salida, se usa para mostrar datos o información. Su forma general es la siguiente:

`imprimir` (`parámetro`)

Donde `parámetro` puede tomar cualquiera de los siguientes formatos:

- `imprimir( variable )` o `imprimir( CONSTANTE )`

  Muestra el valor almacenado en una variable o en una constante, previamente definidas y a las cuales se les asignó un valor.

- `imprimir( Operación )`

  Donde `Operación` es una expresión matemática, que puede estar conformada por valores constantes o por variables, por ejemplo:

  `imprimir (5 * 2)`, muestra como resultado de salida un 10.

- `imprimir( "Mensaje" )`

  La salida que muestra, es el `Mensaje` que se escriba entre las comillas dobles.

- Los anteriores formatos pueden combinarse de diferentes formas:

  - `imprimir( "Mensaje", variable )`
  - `imprimir( "Mensaje", variable, CONSTANTE )`
  - `imprimir( "Mensaje", operación )`
  - `imprimir( variable, "Mensaje", operación )`
  - Entre otros.

Comentarios

Es otro de los componentes de un algoritmo; se utilizan de manera opcional para documentarlo. No hacen parte de su lógica, por lo cual no afectará su ejecución.

La documentación es una buena práctica que permite que otra persona o el mismo autor tenga claridad sobre algunos aspectos que se diseñaron dentro del algoritmo. Esto permite que a futuro los ajustes o modificaciones a que haya lugar, sean mucho más sencillos o por lo menos más entendibles.

Los comentarios pueden ser de dos formas.

1. De una sola línea. Para ello se usan dos barras (`//`).

   ```
   // Esto es un comentario.
   ```

   Toda la línea, a partir de estos caracteres, no se tendrá en cuenta como parte de la lógica del algoritmo.

2. De varias líneas. En este caso se usa la pareja de caracteres `/*` para abrir y la pareja `*/` para cerrar.

   ```
   /* Este es un comentario de varias líneas. Todo lo que
      se escriba entre estas parejas de caracteres será
      ignorado. Es decir, no hará parte de la lógica del
      algoritmo.
   */
   ```

Forma general de un algoritmo en pseudocódigo

Ahora que ya se conoce todo lo que puede contener un algoritmo en pseudocódigo, el siguiente paso es determinar cómo se combinan o conjugan las palabras reservadas con las variables, constantes, expresiones y comentarios. Para ello observe la siguiente forma general:

```
Algoritmo <nombre>  // Comienzo del algoritmo

  // Comentarios

  // Declaración de variables
  TipoDato variable1, variable2, ..., variableN

  // Definición de constantes

  Constante tipoDato CONSTANTE1 = valor
```

```
    // Instrucciones a ejecutar
    <Instrucción 1>

    <Instrucción 2>
    < ... >
    < ... >
    <Instrucción n>
  FinAlgoritmo    // Fin del algoritmo
```

Antes de analizar esta forma general, recuerde que los comentarios son opcionales, se escribieron para hacer más didáctica la explicación.

```
  Algoritmo <nombre>
```

Se identifica el algoritmo con un nombre[9], que debe dar un contexto de lo que soluciona.

```
  // Comentarios.
```

Son opcionales y se usan para documentar el algoritmo, en esta parte generalmente se escribe el enunciado del problema y el nombre del autor.

```
  TipoDato variable1, variable2, ..., variableN
```

Es la declaración de las variables que se van a utilizar. Todas las del mismo tipo se pueden agrupar, en uno o varios renglones, separándose mediante comas (,).

```
  Constante tipoDato CONSTANTE1 = valor
```

Se usa para definir las constantes que se requieran dentro del algoritmo.

```
    <Instrucción 1>
    <Instrucción 2>
    < ... >
    < ... >
    <Instrucción n>
```

Se escriben en orden lógico las instrucciones que van a ser ejecutadas por el algoritmo. Cada Instrucción debe ser escrita mediante una de las

[9]Se deben eliminar los caracteres < y > una vez se asigne el nombre, Ej: `Algoritmo NumerosPares`

palabras reservadas que se estudiaron con anterioridad o puede ser una expresión de asignación o una expresión aritmética escrita en notación algorítmica.

Generalmente, las primeras instrucciones que hay en un algoritmo son las que permiten la entrada de los datos disponibles o conocidos, con los cuales se hará el proceso para hallar un resultado. Recuerde que para lograr la entrada de un dato se usa el código `leer` y que puede ir acompañado de un código `imprimir`. Dentro de las últimas instrucciones, se entregan los resultados mediante la palabra reservada `imprimir`.

```
FinAlgoritmo
```

Indica que se debe terminar la ejecución del algoritmo.

En este momento el lector ya conoce varias de las herramientas que le permiten diseñar una solución algorítmica para resolver un problema. En los siguientes párrafos de este capítulo, se explicará cómo hacer el análisis de esos problemas y cómo implementar los algoritmos que los solucionen.

1.8. Cómo solucionar un problema por computador

Para la construcción de software, independientemente de su tamaño o complejidad, se deben ejecutar un conjunto de pasos que garanticen su calidad. Esos pasos son [Jiménez et al., 2016] y [Corona and Ancona, 2011]:

1. Definición del problema.
2. Análisis del problema.
3. Diseño del algoritmo.
4. Codificación (implementación).
5. Compilación y ejecución.
6. Verificación y depuración.
7. Mantenimiento.
8. Documentación.

Para el caso del que se ocupa este libro, se trabajarán los tres primeros pasos planteados en la lista anterior.

Definición del problema. Se debe establecer claramente cuál es el problema a solucionar. Para el caso de este texto, esta etapa la cubren los enunciados que se plantean para cada uno de los ejemplos que se desarrollarán; en la práctica profesional, estará cubierta por las necesidades que expresen los clientes.

Para poder pasar a la siguiente etapa, es importante tener claro que lo que se va a resolver debe estar dentro de los conocimientos del diseñador del algoritmo y así pueda abordar la construcción de una solución. En caso de no poseer los conocimientos suficientes, se debe buscar ayuda de personas expertas en el tema, generando un trabajo interdisciplinario, que por cierto es una de las características que tiene el diseño de algoritmos.

Análisis del problema. Analizar es comprender. Para poder emprender la búsqueda de una solución se debe comprender perfectamente el problema planteado. Se debe identificar la información relevante y descartar la que no aporte a la solución. Una vez se entienda con claridad, qué es lo que se debe solucionar, se procede a realizar el análisis de la información que se posee. Esta etapa se fundamenta en encontrar la respuesta a cuatro preguntas:

- ¿Cuáles resultados se esperan?
- ¿Cuáles son los datos disponibles?
- ¿Qué procesos hay que realizar?
- ¿Qué variables se deben utilizar?

Los **resultados que se esperan**, constituyen las salidas del algoritmo y son el producto del proceso que realice. Todo algoritmo debe proporcionar algún resultado. A pesar de que, en la ejecución del algoritmo, la salida es la última etapa, dentro de este análisis será la primera, ya que con esta información se podrá tener la claridad suficiente de qué pasos se deben realizar para llegar al objetivo final.

Los **datos disponibles**, se refiere al conjunto de datos que se tiene y que se va a procesar para servir de insumo a la solución del problema. Estos datos disponibles, generalmente representan los datos de entrada al algoritmo, aunque hay que hacer claridad que habrá situaciones en donde los algoritmos no tienen entrada de datos y los datos disponibles son suministrados por alguna instrucción dentro del proceso a realizar.

En cuanto a los **procesos**, son todas las acciones a los que se someten los datos disponibles para encontrar la solución del problema. En esta etapa se deben establecer las fórmulas matemáticas necesarias para realizar los cálculos correspondientes. Igualmente se establecen las condiciones y decisiones que hacen parte del proceso. Cada una de las instrucciones que se ejecutarán, deben ser tenidas en cuenta en este punto, no solamente en su función, sino también en el orden lógico en que deben ser ejecutadas.

Con base en los puntos anteriores, se establecen las **variables requeridas** para almacenar los valores de los datos disponibles, de los resultados esperados y de las fórmulas matemáticas que se requieran, además se deben identificar todas las variables necesarias en la solución. En este punto, se hace necesario establecer el tipo de dato al que pertenecerá cada variable, de acuerdo al valor que deba almacenar.

Diseño. Esta etapa se desarrolla luego de terminado el análisis; consiste en la creación del algoritmo que soluciona el problema, para ello se utiliza alguna de las técnicas de representación (descripción narrada, diagramas de flujo o pseudocódigo).

Para convertirlos a algoritmos, los enunciados deben ser minuciosamente analizados por el diseñador. Algunos, enunciados no proporcionan explícitamente lo que se requiere, es ahí donde está la habilidad del diseñador para sacar el máximo provecho e información relevante para codificar la solución. A manera de ejemplo, se analizará el siguiente enunciado.

El profesor de Física Mecánica, desea que cada uno de sus estudiantes puedan calcular, mediante un algoritmo, la velocidad con que se desplazan de su casa a la Universidad. Todos los estudiantes deben medir la distancia que recorren y tomar el tiempo que invierten en él.

De este enunciado hay que tener en cuenta lo siguiente:

- No proporcionó la fórmula para el cálculo de la velocidad. Por lo cual, es un dato que se tendrá que investigar de forma adicional.
- El enunciado no deja explícito qué información o dato debe mostrarse, pero al solicitar únicamente el cálculo de la velocidad, se puede deducir que se debe informar ese resultado. En algunas ocasiones, no todos los cálculos deben ser informados, algunos son cálculos intermedios que serán utilizados en otras fórmulas.
- Es importante determinar cuáles son los datos relevantes para hallar la solución y cuáles no representan ningún aporte. Por ejemplo, el

hecho que sea un profesor de Física Mecánica, no es significativo para la solución.

Una vez analizadas estas características, la aplicación de los pasos para solucionar el problema planteado en el ejemplo, se ejecuta así:

Resultado esperado: la velocidad.

Al poseer la información de cuál es el resultado que el algoritmo debe informar, se procede a revisar con que datos de entrada se cuenta, los cuales serán uno de los insumos primordiales en el proceso.

Datos conocidos: distancia y tiempo invertidos en el recorrido, considerando las respectivas unidades.

Proceso: calcular la velocidad, teniendo en cuenta los datos disponibles. Para realizar este cálculo se requiere de una fórmula, la cual no fue proporcionada por el enunciado y que debe ser investigada para poder plantear la solución. En este ejemplo, si no se recuerda la fórmula que es enseñada en los cursos de física que se recibieron en el Colegio, conseguirla es un proceso simple, basta con consultar un libro de física mecánica o en un motor de búsqueda en Internet. Se presentarán situaciones, en donde el problema a resolver es un poco más complejo y se requerirá el trabajo conjunto con otros profesionales de diferentes áreas del conocimiento.

Retomando el ejercicio y teniendo en cuenta que se posee la distancia (`x`) y el tiempo (`t`), la fórmula a la que se hace referencia para hallar la velocidad (`v`) es la siguiente: $v = \frac{x}{t}$, que al escribirla como una expresión algorítmica se visualiza así: $v = x/t$

No olvide que las fórmulas en su notación matemática, deben ser convertidas en su notación algorítmica.

El siguiente paso, es definir las variables que se emplearán en el algoritmo.

Variables requeridas: en vista de que esta fórmula y sus variables son conocidas universalmente, en este caso se usarán tal cual están expresadas.

- `v`: velocidad.
- `x`: distancia.
- `t`: tiempo.

Los valores que se van a almacenar podrán tener cantidades decimales, por lo tanto, se trabajarán de tipo `Real`.

Antes de pasar a la etapa del diseño del algoritmo y con el propósito de dar mayor claridad a algunos conceptos estudiados anteriormente, se usará este ejemplo para ilustrar mediante la Figura 1.17 las 3 etapas que tiene un algoritmo.

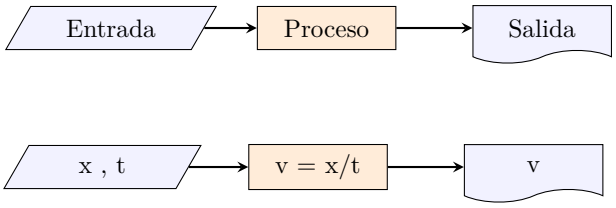


Figura 1.17: Las tres etapas de un algoritmo

Una vez realizado el análisis del problema, el siguiente paso es **diseñar el algoritmo**, para ello se utilizarán las 3 formas más comunes de representación.

Primera forma de representación: descripción narrada

```
1 Algoritmo
2   Leer la distancia recorrida.
3   Leer el tiempo empleado en recorrer esa distancia.
4   Calcular la velocidad, empleando la fórmula.
5   Informar el resultado del cálculo (la velocidad).
6 FinAlgoritmo
```

La palabra **`Algoritmo`**, marca el comienzo del algoritmo. Entre las líneas 2 y 5, se encuentra descritas las instrucciones que solucionan el problema. La palabra **`FinAlgoritmo`** de la línea 6, señala el fin de la ejecución de este algoritmo. Este tipo de solución no es fácil de traducir a un lenguaje de programación, como lo es un diagrama de flujo o un pseudocódigo.

Ahora, para diseñar este mismo algoritmo usando diagramas de flujo (Ver Figura 1.18) o pseudocódigo, es necesario tener presente que se requiere del uso de algunas palabras reservadas o símbolos para representar los pasos. En las Tablas 1.13, 1.14 y 1.15 se mostrarán los símbolos correspondientes a los diagramas de flujo y sus equivalencias en pseudocódigo.

Para la entrada o lectura de los datos conocidos:

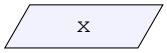
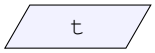
Diagrama Flujo	Pseudocódigo	Explicación
	`leer ( x )`	Captura el valor de la distancia y lo almacena en la variable `x`
	`leer ( t )`	Captura el valor del tiempo y lo almacena en la variable `t`

Tabla 1.13: Entrada de datos conocidos

El cálculo de la velocidad se expresa así (Proceso):

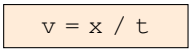
Diagrama Flujo	Pseudocódigo	Explicación
	`v = x / t`	Calcula la velocidad (`v`), dividiendo la distancia (`x`) sobre el tiempo (`t`); el resultado lo almacena en `v`.

Tabla 1.14: Cálculo (Proceso)

Para la salida o resultados esperados se procede de la siguiente manera:

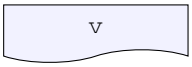
Diagrama Flujo	Pseudocódigo	Explicación
	`imprimir ( v )`	Informa o muestra el valor que esté almacenado en la variable `v`, el cual corresponde a la velocidad calculada.

Tabla 1.15: Entrega de resultados esperados

Con todas estas instrucciones, se puede obtener entonces el algoritmo que encuentra la velocidad, teniendo como datos de entrada la distancia y el tiempo. A continuación, se presentan las versiones completas en diagrama de flujo y pseudocódigo.

Segunda forma de representación: diagrama de flujo

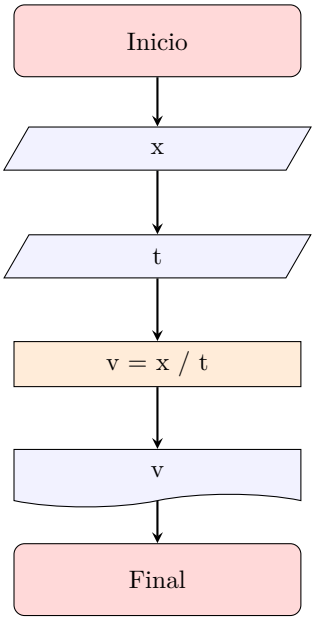


Figura 1.18: Diagrama de flujo Velocidad

A continuación, se presentan las versiones completas en pseudocódigo.

Tercera forma de representación: pseudocódigo - Versión 1

```
1 Algoritmo Velocidad
2   Real v, x, t
3   leer( x )
4   leer( t )
5
6   v = x / t
7
8   imprimir( v )
9 FinAlgoritmo
```

A diferencia de la solución que se presentó a través el diagrama de flujo, en esta versión en pseudocódigo se hizo la declaración de variables (línea 2), con esto se le especifica al algoritmo que habrán 3 posiciones de memoria de tipo `Real`, llamadas `v,x` y `t`, en las cuales se almacerán los datos que requieren para solucionar el problema planteado.

La anterior versión en pseudocódigo puede ser mejorada, incluyendo algunos mensajes con la instrucción `imprimir`, que serán de ayuda para el usuario del algoritmo. Con ellos se solicitará la distancia y el tiempo, de

igual forma la salida se acompañará de un mensaje que proporcione más información.

Tercera forma de representación: pseudocódigo - Versión 2

```
Algoritmo Velocidad
  Real v, x, t

  imprimir( "Ingrese la distancia: " )
  leer( x )
  imprimir( "Ingrese el tiempo: " )
  leer( t )

  v = x / t

  imprimir( "La velocidad es: ", v )
FinAlgoritmo
```

Al ejecutar este algoritmo, suponiendo que la distancia es de 300 metros y el tiempo empleado es de 15 minutos, se observará lo siguiente:

```
Ingrese la distancia: 300
Ingrese el tiempo: 15
La velocidad es: 20
```

En los siguientes capítulos, se profundizará en el desarrollo de algoritmos.

Aclaración:



En la versión en pseudocódigo se hace la declaración de variables, en los diagramas de flujo no se requiere.

La impresión de mensajes también se puede hacer dentro de los diagramas de flujo.

1.9. Ejercicios propuestos

1. Imagine que se desea declarar variables para almacenar datos de acuerdo con el enunciado de la primera columna de la Tabla 1.16 que aparece a continuación. Diligencie la segunda y tercera columna, declarando un identificador válido y adecuado para la variable, con su respectivo tipo de dato.

Dato a almacenar	Variable	
	Identificador	Tipo de dato
Placa de un vehículo		
Tamaño del motor en centímetros cúbicos		
Número de pasajeros		
Número de baños de una casa		
Área de la casa en metros		
Valor del alquiler		
Valor del descuento de un producto		
¿Encendió el computador?		
Número de matrícula del estudiante		
Valor de la matrícula del estudiante		

Tabla 1.16: Identificación de variables y su tipo

2. Para las variables que se declaran en la Tabla 1.17, diga si el identificador utilizado es correcto o incorrecto y justifique porqué.

Variable	¿Es válido?	Justifique
Nombre medico		
@especialidad		
generoAspirante		
Valor a pagar		
salarioEmpleado1		
#CEDULA		
Titulo_Libro		
tituloLibro		
Años_de_Experiencia		
esCasado		

Tabla 1.17: Validez de variables

3. Describa, de acuerdo a la precedencia de los operadores, en cuál orden se ejecutan las siguientes expresiones:

   a) `resultado = PI * radio^2`

   b) `resultado = 2 * a + 3 * b - c`

   c) `resultado = 2 * ( a + 3 ) * b - c`

   d) `resultado = a^2 - b * 36^(1/2)`

   e) `resultado = ( a + b ) / ( 2 * c + 1 - a%3)`

4. Resuelva paso a paso las siguientes expresiones, teniendo en cuenta la prioridad de ejecución de los operadores.

   a) `3 * (3 + 4) * (5 - 2)`

   b) `(3^(2 + 3) - 1) ^ (1.0/2.0)`

   c) `5.0/2.0 * 3 + 4 ^ 3.0 * 2/(5.0 + 2.0) - 2`

   d) `10 * (7 + 7)% (9 + 2)/10`

   e) `3% 2 - (2 + 2) / 1 * (3 + 1) + 3`

   f) `(5 + 8*2 - 3.0/5.0)/(4*1 - 2.0/3.0 + 8/2)`

   g) `(4 + 8)/(4 /(1 + 1)) - (3 + 2 + 1)/(5^2+4)`

5. Suponga que se requieren las variables: a, b, c, d de tipo real.

   a) Declare estas variables.

   b) Asígnele un número cualquiera entre 10 y 20 a cada variable.

   c) Evalúe las siguientes expresiones, teniendo en cuenta los valores asignados:

   - `a = 3 * b + d%5`
   - `d = (4 * a/2 - 3*c) / (4 + b%3)`
   - `c = 3 * b^2 - 5 * c/3`
   - `d = 2 * a + 3 * b + 4 * c/d`
   - `a = 3 * a`
   - `b = 7 + a`
   - `c = b - a`
   - `d = c + a - b`

6. Se tienen las variables: a, b, c y d de tipo entero. A partir de las asignaciones que se dan enseguida, determine si las siguientes expresiones entregan como resultado un valor verdadero o falso:

```
a = 5
b = 4
c = 7
d = 3
```

   a) `3 * a <= 15`

   b) `c - d == a + 2`

   c) `(5 * a + 3 * b) >= (c^d - 35)`

   d) `(5% c + 3 * b/d) < (c^d/3)`

e) `(32 % (4 * c) + 3 * (b + d)) < (c^(1 + a % 2))`

f) `(5 * d + b != 4^d)` **O** `(c + d >= a/b)`

g) `(c * a + 10 > b * d - 6)` **Y** `( (b + c) % 5 <a)`

h) `(c*(a + 10) > b*(8*d-6))` **Y** `((b + c) % 5 < a)`

i) `((a + b + c)/d != 12)` **O** `(4*c + 5 != 3 * a - d)`

j) `(c*a+7 >b * d - 3) Y ( (b+c)^(1/2) <a^(3/2))`

k) **NO** `(a == b)`

l) **NO** `(a * b + c != d)`

7. Escriba en notación algorítmica las siguientes expresiones matemáticas:

a) $x = \dfrac{-b + \sqrt{b^2 - 4ac}}{2a}$

b) $x = \dfrac{3a + 4b}{5 + c} + \dfrac{8c - 10}{3a + b}$

c) $x = \dfrac{3a\sqrt{4b + 8}}{\frac{4b+7}{10+\sqrt{4c}}} + \sqrt[3]{8d}$

d) $x = \dfrac{5ab^7}{3b + a} + 2ac^{3/5}$

e) $x = \dfrac{3a + \frac{2b+4}{3c4^3}}{\frac{a+b+c}{\sqrt{a+c}+7}}$

8. Preguntas sobre los conceptos vistos

a) Defina con sus propias palabras los siguientes términos, vistos en este capítulo:

- Variable
- Constante
- Dato
- Tipo de dato
- Operador
- Expresión

b) Dado que existen diferentes tipos de datos, operadores y expresiones, haga una clasificación para:

- Tipos de datos

- Operadores
- Expresiones

9. Algoritmos.

   a) Para los siguientes enunciados realice el análisis y el diseño del algoritmo (utilizando la forma de descripción narrada) que permita realizar las acciones que a continuación se listan:

   - Adquirir un libro a través de una librería virtual.
   - Descargar un vídeo de YouTube.
   - Calcular cuánto dinero se gastará el día de mañana.
   - Invitar a un amigo a desayunar en la cafetería.
   - Desplazarse desde su casa a un centro comercial.

   b) Si el lector es un estudiante universitario, también realice los siguientes ejercicios, los cuales le permitirán familiarizarse con los procesos de su institución educativa. Tenga en cuenta que, si no los conoce, deberá realizar las consultas necesarias:

   - Solicitar la homologación de un espacio académico.
   - Realizar un proceso de validación de un espacio académico.
   - Cancelar materias y semestre.
   - Realizar un préstamo de un libro en la biblioteca.
   - Realizar una consulta bibliográfica en las bases de datos de la biblioteca.
   - Solicitar una cita en el Centro Médico de Bienestar Institucional.

Capítulo 2

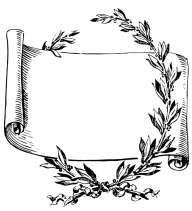
Estructura secuencial

La mayoría de los buenos programadores programan no porque esperan que les pagen o que el público los adore, sino porque programar es divertido.

Linus Torvalds

Objetivos del capítulo:

- Analizar problemas en los que se puedan aplicar estructuras algorítmicas secuenciales para su solución.
- Construtir algoritmos en pseudocódigo y diagramas de flujo para resolver problemas con la estructura algorítmica secuencial.
- Realizar pruebas de escritorio para verificar el funcionamiento de los algoritmos construidos.



En este capítulo se estudiarán algoritmos sencillos a través de los cuales se resolverán problemas, inicialmente de baja complejidad; Estos algoritmos se escribirán en pseudocódigo y luego tendrán su representación a través de diagramas de flujo. Para escribir estos algoritmos en pseudocódigo, es necesario utilizar las palabras reservadas vistas anteriormente, principalmente las palabras `Algoritmo`, `FinAlgoritmo`, `leer` e `imprimir`, así como la utilización de variables, constantes, operadores y expresiones ya expuesto en el capítulo anterior.

De acuerdo con [Joyanes A., 1996], un algoritmo secuencial es aquel en el que una acción sigue a otra acción en la secuencia de instrucciones. "La secuenciación es una estructura que permite controlar la ejecución de un conjunto de acciones en orden secuencial; esto es, ejecuta la primera acción, luego la que sigue y así sucesivamente hasta la última" [López, 2009]. Todo algoritmo secuencial consta de tres secciones: La primera de ellas corresponde a "las entradas" (de los datos disponibles), en la cual se identifican e ingresan al algoritmo los datos que se conocen y son necesarios para resolver el problema. La segunda sección es la denominada "Procesos" o cálculos, donde se implementan las operaciones que encontrarán los resultados que se espera obtendrá el algoritmo. La tercera y última sección será la de "salidas" (Resultados esperados), en la que se mostrarán los resultados que se encontraron en la sección anterior y será lo que debe entregar como respuesta el algoritmo a quien lo utilice.

2.1. Estructura básica de un algoritmo secuencial

Con el fin de tener una manera formal para expresar los algoritmos, tanto en pseudocódigo como en diagramas de flujo, se va a utilizar la notación especificada en el apartado "Forma general de un algorítmo en pseudocódigo"del capítulo anterior donde se explicaron los elementos fundamentales para escribir los algoritmos con pseudocódigo.

Recuerde que los algoritmos empiezan con un nombre que los identifica. Se recomienda que este nombre esté directamente relacionado con su funcionalidad del algoritmo, es decir, que el nombre exprese lo que este hace.

Es recomendable que, el nombre del algoritmo comience por un sustantivo en singular y con su primera letra en mayúscula. Puede estar compuesto de varias palabras que irán unidas, cada una de ellas iniciando con letra en mayúscula.

De la misma forma, se utilizará la notación para los diagramas de flujo estudiada también en el capítulo anterior en el apartado "Diagramas de flujo."

A continuación, se exponen algunos ejemplos para ilustrar la estructura algorítmica secuencial, primero su enunciado, luego un análisis del problema, su solución en pseudocódigo y, por último, el diagrama de flujo.

Aclaración:



Cuando se ingresan a un algoritmo los datos necesarios para producir los resultados esperados, es decir, se ingresan los datos conocidos en el problema, es probable que se entren datos erróneos o de otros tipos de datos diferentes a los esperados, lo que producirá un error en la ejecución del algoritmo. En los algoritmos que se tratan en este capítulo, se dará por sentado que el usuario siempre ingresa los datos adecuados y no datos erróneos. Si bien es cierto que también puede evitarse el ingreso de datos erróneos a un algoritmo, lo que se conoce como validación, este tema está fuera del alcance de los primeros capítulos de este libro.

.:Ejemplo 2.1. *Suponga que la oficina de tesorería de una empresa requiere de un algoritmo que le permita calcular el salario a pagar a un empleado. Imagine que a este empleado le pagan de acuerdo con el número de horas que haya laborado durante el periodo a razón de un valor específico cada hora.*

Análisis del problema:

- **Resultados esperados:** se espera que el algoritmo pueda determinar el valor del salario a pagar al empleado fruto de las horas que laboró y, que este resultado pueda ser mostrado a quien utiliza el algoritmo.
- **Datos disponibles:** quien vaya a utilizar el algoritmo, deberá saber el número de horas que haya laborado el empleado durante el periodo y, el valor de la hora para este empleado.
- **Proceso:** para obtener el salario a pagar, es necesario escribir una expresión en la que se multiplique el número de horas laboradas por

el valor de la hora. Este resultado deberá asignarse a una variable. Lo anterior corresponde al proceso o cálculo que debe hacerse en este algoritmo.

- **Variables requeridas:**
  - `numeroHoras:` almacena la cantidad de horas que laboró el empleado.
  - `valorHora:` valor a pagar por cada hora de trabajo.
  - `salarioPagar:` valor a pagar al empleado.

De acuerdo al análisis planteado, se propone el Algoritmo 2.1.

Algoritmo 2.1: SalarioEmpleado

```
1  Algoritmo SalarioEmpleado
2
3    // Este algoritmo permite calcular el salario de un
4    // empleado con base en las horas laboradas y el valor
5    // de la hora.
6
7    Entero numeroHoras
8    Real   valorHora, salarioPagar
9
10   imprimir( "Ingrese el número de horas laboradas: " )
11   leer( numeroHoras )
12
13   imprimir( "Ingrese el valor de la hora: " )
14   leer( valorHora )
15
16   salarioPagar = numeroHoras * valorHora
17
18   imprimir( "El salario a pagar es: ", salarioPagar )
19 FinAlgoritmo
```

Al ejecutar el algoritmo:

Primera ejecución:

```
Ingrese el número de horas laboradas: 10
Ingrese el valor de la hora: 45000
El salario a pagar es: 450000
```

Segunda ejecución:

```
Ingrese el número de horas laboradas: 48
Ingrese el valor de la hora: 22500
El salario a pagar es: 1080000
```

Aclaración:



En todos los ejemplos del libro se presenta la sección "Explicación del algoritmo", que tiene como fin ayudar a comprender el pseudocódigo propuesto. Como estrategia para facilitar la explicación, se retoman algunas partes del algoritmo y se explican paso a paso. Sin embargo, en los algoritmos que se presentan mas adelante, se omitirán aquellas explicaciones que ya se hayan hecho previamente.

Explicación del algoritmo:

La primera sección del algoritmo se utiliza para declarar las variables que se necesitan en la solución, indicando además del nombre el respectivo tipo de dato.

```
7   Entero numeroHoras
8   Real   valorHora, salarioPagar
```

La sentencia `imprimir` que aparece dos veces: al principio del algoritmo permite imprimir mensajes que se mostrarán para que el usuario sepa qué datos debe ingresar.

```
10  imprimir( "Ingrese el número de horas laboradas: " )
11  leer( numeroHoras )
12
13  imprimir( "Ingrese el valor de la hora: " )
14  leer( valorHora )
```

Luego se calcula el valor del salario a pagar mediante el producto del número de horas laboradas por el valor de la hora.

```
16  salarioPagar = numeroHoras * valorHora
```

Finalmente, será mostrado por la última sentencia `imprimir` el salario a pagar.

```
18  imprimir( "El salario a pagar es: ", salarioPagar )
```

En la Figura 2.1 se muestra la solución del Ejemplo 2.1 mediante un diagrama de flujo.

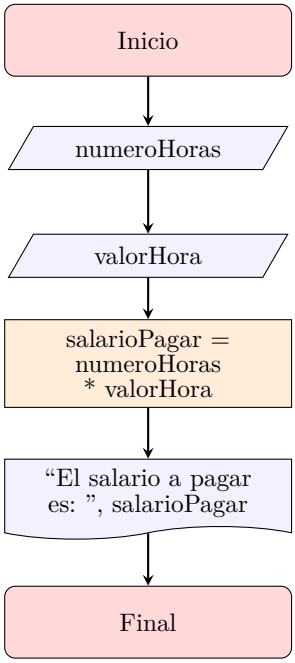


Figura 2.1: Diagrama de flujo del Algoritmo `SalarioEmpleado`

.:Ejemplo 2.2. *Elabore un algoritmo que, a partir del valor del radio de un círculo, permita calcular tanto su área como su perímetro.*

Este es un problema típico en Geometría básica que consiste en calcular tanto el área como el perímetro de un círculo aplicando las fórmulas matemáticas que existen para obtener estos resultados. Para poder aplicar las fórmulas, es necesario conocer el valor del radio del círculo al que se le va a calcular área y perímetro. También es necesario el uso de la constante PI equivalente de manera muy aproximada a 3.1416*.*

Recuerde que la fórmula para obtener el área de un círculo es $PI * radio^2$*. Por su parte, la fórmula del perímetro es* $2 * PI * radio$*.*

Análisis del problema:

- **Resultados esperados:** se pretende que, al terminar de ejecutarse el algoritmo, este haya encontrado el área y el perímetro del círculo y, que estos resultados se muestren al usuario.
- **Datos disponibles:** para resolver este problema y poder obtener el área y el perímetro de un círculo cualquiera, es necesario conocer el radio del mismo; por tanto, cuando se va a aplicar este algoritmo, se requiere saber el radio.

- **Proceso:** los procesos consistirán en las instrucciones necesarias para hallar el área y el perímetro del círculo. Tenga en cuanta que, tanto para encontrar el área como el perímetro, deberá aplicar las fórmulas matemáticas, que tendrán que ser escritas como expresiones algorítmicas.
- **Variables requeridas:**
  - `radio`: distancia del centro del círculo al exterior.
  - `area`: valor del área del círculo.
  - `perimetro`: valor de la longitud del círculo.

De acuerdo al análisis planteado, se propone el Algoritmo 2.2.

Algoritmo 2.2: Circulo

```
1  Algoritmo Circulo
2
3    // Mediante este algoritmo se puede calcular el área y
4    // el perímetro de un círculo cualquiera conociendo su
5    // radio.
6
7    Real radio, area, perimetro
8    Constante Real PI  = 3.1416
9
10   imprimir( "Ingrese el radio del círculo: " )
11   leer( radio )
12
13   area = PI * radio ^ 2
14   perimetro = 2 * PI * radio
15
16   imprimir( "El área del círculo es: ", area )
17   imprimir( "El perímetro del círculo es:", perimetro )
18 FinAlgoritmo
```

Al ejecutar el algoritmo:

```
Ingrese el radio del círculo: 4
El área del círculo es: 50.2656
El perímetro del círculo es: 25.1328
```

Explicación del algoritmo:

El algoritmo comienza con la declaración de las variables y la constante que se requieren, tanto para almacenar el `radio` de la circunferencia como para almacenar los cálculos del `area` y el `perimetro`. Estas variables son de tipo `Real`, pues almacenarán números, incluso con decimales.

```
7   Real radio, area, perimetro
8   Constante Real PI  = 3.1416
```

Con la instrucción `imprimir` se muestra el mensaje solicitando el `radio`, mientras que con `leer` se captura el dato y se asigna en la respectiva variable `radio`.

```
10  imprimir( "Ingrese el rádio del círculo: " )
11  leer( radio )
```

Posteriormente, se calculan área y perímetro a partir de las fórmulas matemáticas que involucran la constante `PI`.

```
13  area = PI * radio ^ 2
14  perimetro = 2 * PI * radio
```

Por último, se utiliza nuevamente la instrucción para mostrar los resultados.

```
16  imprimir( "El área del círculo es: ", area )
17  imprimir( "El perímetro del círculo es:", perimetro )
```

En la Figura 2.2 se muestra la solución del Ejemplo 2.2 mediante un diagrama de flujo.

Aclaración:



Note que en el diagrama de flujo que resuelve este ejercicio y que aparece a continuación, las instrucciones de salida no muestran los mensajes exactamente como están escritos en el pseudocódigo, sino que aparecen solo los nombres de las variables; mientras que en otros diagramas del libro, ambos textos coinciden (pseudocódigo y diagrama de flujo). El tener o no los mensajes no afecta en últimas la estructura de la solución, así que por simplicidad, en algunos casos se omiten.

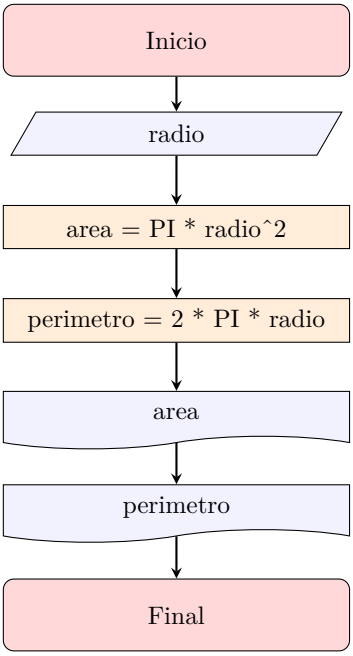


Figura 2.2: Diagrama de flujo del Algoritmo `Circulo`

.:Ejemplo 2.3. *Elabore un algoritmo que, a partir de un número ingresado, se calculen y muestren los resultados que arroajn las principales funciones del lenguaje algorítmico para ese número. Este ejercicio pretende ilustrar la operatividad de las principales funciones matemáticas disponibles en el lenguaje algorítmico y que fueron mencionadas en el apartado de las palabras reservadas. Estas funciones operan sobre un número Real que el usuario debe ingresar; las funciones que se utilizan proporcionan las respuestas que se piden.*

Análisis del problema:

- **Resultados esperados:** una vez ejecutado el algoritmo, se espera que se muestren el seno, el coseno, la tangente, la raíz cuadrada, el logaritmo natural y el logaritmo en base 10 del número ingresado por el usuario.
- **Datos disponibles:** se debe tener disponible el número al que se le van a realizar las operaciones.
- **Proceso:** luego de que se haya ingresado el número, se utilizan las funciones matemáticas, una a una, pasándoles el número como dato necesario para que puedan hacer el cálculo; este cálculo se almacena en variables que se mostrarán al final del algoritmo.

- **Variables requeridas:**
  - `numero`: valor a procesar
  - `seno`: valor de la función seno sobre el valor
  - `coseno`: valor de la función coseno sobre el valor
  - `tangente`: valor de la función tangente sobre el valor
  - `raiz`: valor de la función raíz cuadrada sobre el valor
  - `logaritmoNatural`: valor de la función logaritmo natural sobre el valor
  - `logaritmo10`: valor de la función logaritmo en base 10.

De acuerdo al análisis planteado, se propone el Algoritmo 2.3.

Algoritmo 2.3: UsoFunciones

```
1  Algoritmo UsoFunciones
2
3    /* Mediante este algoritmo se ilustra
4       el uso de varias funciones matemáticas
5       disponibles en el lenguaje algorítmico
6       y en los difentes lenguajes de programación
7    */
8
9    Real numero, seno, coseno, tangente
10   Real raiz, logaritmoNatural, logaritmo10
11
12   imprimir( "Ingrese un número: " )
13   leer( numero )
14
15   seno             = sen( numero )
16   coseno           = cos( numero )
17   tangente         = tan( numero )
18   raiz             = raizCuadrada( numero )
19   logaritmoNatural = ln( numero )
20   logaritmo10      = log( numero )
21
22   imprimir( "Del número ", numero )
23   imprimir( "Su seno es ", seno )
24   imprimir( "Su coseno es ", coseno )
25   imprimir( "Su tangente es ", tangente )
26   imprimir( "Su raíz cuadrada es ", raiz )
27   imprimir( "Su logaritmo natural es ", logaritmoNatural )
28   imprimir( "Su logaritmo en base 10 es ", logaritmo10 )
29 FinAlgoritmo
```

Al ejecutar el algoritmo:

```
Ingrese un número: 25
Del número 25
Su seno es -0.13235175009777303
Su coseno es 0.9912028118634736
Su tangente es -0.13352640702153587
Su raiz cuadrada es 5.0
Su logaritmo natural es 3.2188758248682006
Su logaritmo en base 10 es 1.3979400086720377
```

Explicación del algoritmo:

El algoritmo empieza como todos los algoritmos anteriores, declarando las variables que se necesitan.

```
6    Real numero, seno, coseno, tangente
7    Real raiz, logaritmoNatural, logaritmo10
```

Posteriormente, se solicita mediante la instrucción `imprimir` el número requerido para hacer los cálculos y se captura en la variable numero con la instrucción `leer`.

```
9    imprimir( "Ingrese un número: " )
10   leer( numero )
```

A continuación, se usan las funciones matemáticas para obtener los resultados deseados y almacenarlos en las variables destinadas para este fin.

```
12   seno             = sen( numero )
13   coseno           = cos( numero )
14   tangente         = tan( numero )
15   raiz             = raizCuadrada( numero )
16   logaritmoNatural = ln( numero )
17   logaritmo10      = log( numero )
```

Por último, se muestran los resultados obtenidos con la instrucción `imprimir`.

```
19   imprimir( "Del número ", numero )
20   imprimir( "Su seno es ", seno )
21   imprimir( "Su coseno es ", coseno )
22   imprimir( "Su tangente es ", tangente )
23   imprimir( "Su raíz cuadrada es ", raiz )
24   imprimir( "Su logaritmo natural es ", logaritmoNatural )
25   imprimir( "Su logaritmo en base 10 es ", logaritmo10 )
```

En la Figura 2.3 se muestra la solución del Ejemplo 2.3 mediante un diagrama de flujo.

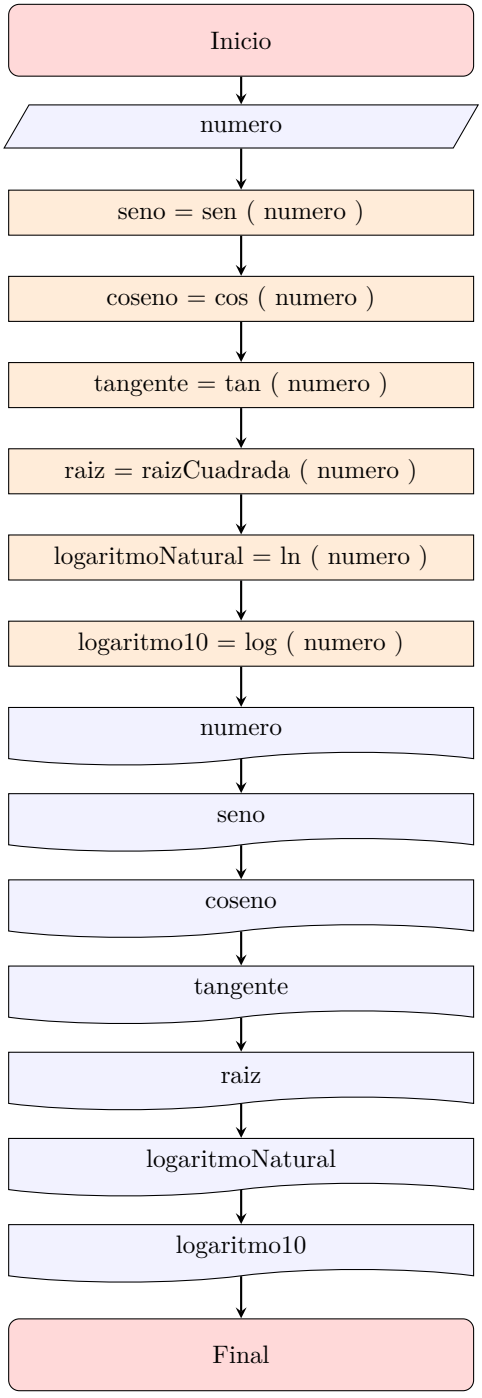


Figura 2.3: Diagrama de flujo del Algoritmo `UsoFunciones`

.:Ejemplo 2.4. *En el acuerdo al que han llegado profesor y estudiantes para la asignatura de Introducción a la lógica de programación, se determinó que se obtendrían 4 notas parciales durante el semestre y que, la nota definitiva para la asignatura sería la media aritmética de esas 4 notas. Suponiendo que un estudiante ya conoce las 4 notas, construya un algoritmo que determine cuál sería la nota definitiva de ese estudiante en la asignatura de* ***Introducción a la lógica de programación****.*

Aclaración:



Este es el problema típico que tienen los estudiantes para calcular la nota definitiva de una asignatura. Cuando la definitiva se calcula a través de la media aritmética, es necesario sumar las notas que se han obtenido y dividir esa suma en el número de notas. Para este caso en particular, se deben sumar las 4 notas y luego dividir entre 4; sin embargo, existe otra forma de calcular y es por el promedio ponderado en donde las notas tiene porcentajes diferentes.

Tenga en cuenta que, la forma en que se calcula la definitiva de los estudiantes bien puede ser distinta, no con media aritmética, sino con media ponderada; cuando se habla de media ponderada, se está asignando un grado de importancia distinta a cada nota, por ejemplo, la primera nota podría tener mayor grado de importancia que la segunda nota y, la tercera nota podría valer menos que la cuarta.

En el apartado de ejercicios propuestos, se retoma este ejercicio pero, el lector deberá resolverlo utilizando una media ponderada.

Análisis del problema:

- **Resultados esperados:** luego de ejecutar el algoritmo, se deberá obtener la nota definitiva de la asignatura, para que, tanto profesor como estudiante la puedan conocer.
- **Datos disponibles:** para poder calcular la nota definitiva, se debe conocer el valor de cada una de las 4 notas parciales que se obtuvieron durante el semestre.

- **Proceso:** el proceso para este ejercicio, consiste en calcular la media aritmética de las 4 notas parciales; esto se logra sumando primero las notas y, posteriormente, dividiendo entre el número de notas, que para este caso es 4.
- **Variables requeridas:**
  - `nota1`: valor de la primera nota.
  - `nota2`: valor de la segunda nota.
  - `nota3`: valor de la tercera nota.
  - `nota4`: valor de la cuarta nota.
  - `definitiva`: valor de la nota final (promedio de las cuatro notas).

De acuerdo al análisis planteado, se propone el Algoritmo 2.4.

Algoritmo 2.4: DefinitivaEstudiante

```
1  Algoritmo DefinitivaEstudiante
2    /* Este algoritmo calcula la nota definitiva de un
3       estudiante a partir de 4 notas parciales a través
4       de la media aritmética. */
5
6    Real nota1, nota2, nota3, nota4, definitiva
7
8    imprimir( "Ingrese la nota 1 del estudiante: " )
9    leer( nota1 )
10
11   imprimir( "Ingrese la nota 2 del estudiante: " )
12   leer( nota2 )
13
14   imprimir( "Ingrese la nota 3 del estudiante: " )
15   leer( nota3 )
16
17   imprimir( "Ingrese la nota 4 del estudiante: " )
18   leer( nota4 )
19
20   definitiva = (nota1 + nota2 + nota3 + nota4) / 4
21
22   imprimir( "La definitiva es: ", definitiva )
23 FinAlgoritmo
```

Al ejecutar el algoritmo:

```
Ingrese la nota 1 del estudiante: 3.8
Ingrese la nota 2 del estudiante: 4.4
Ingrese la nota 3 del estudiante: 5.0
Ingrese la nota 4 del estudiante: 2.0
La nota definitiva es: 3.8
```

Explicación del algoritmo:

Al principio del algoritmo se declararon las cinco variables necesarias, todas de tipo `Real`, ya que tendrán que almacenar números de esas características.

```
6    Real nota1, nota2, nota3, nota4, definitiva
```

Luego, con las instrucciones `imprimir` y `leer`, se solicitan los datos y se almacenan en las variables que representan las notas.

```
8    imprimir( "Ingrese la nota 1 del estudiante: " )
9    leer( nota1 )
10
11   imprimir( "Ingrese la nota 2 del estudiante: " )
12   leer( nota2 )
13
14   imprimir( "Ingrese la nota 3 del estudiante: " )
15   leer( nota3 )
16
17   imprimir( "Ingrese la nota 4 del estudiante: " )
18   leer( nota4 )
```

Posteriormente, se hace el cálculo de la media de las 4 notas, realizando la suma de las mismas y luego dividiendo entre la cantidad de notas (en este caso 4); se utiliza el paréntesis con el fin de garantizar que primero se realice la suma y posteriormente la división, conforme a la prioridad o jerarquía de operaciones.

```
20   definitiva = (nota1 + nota2 + nota3 + nota4) / 4
```

El resultado de este cálculo se almacena en la variable `definitiva`. Para finalizar, se muestra la nota `definitiva` calculada con la instrucción `imprimir`.

```
22   imprimir( "La definitiva es: " , definitiva )
```

En la Figura 2.4 se muestra la solución del Ejemplo 2.4 mediante un diagrama de flujo.

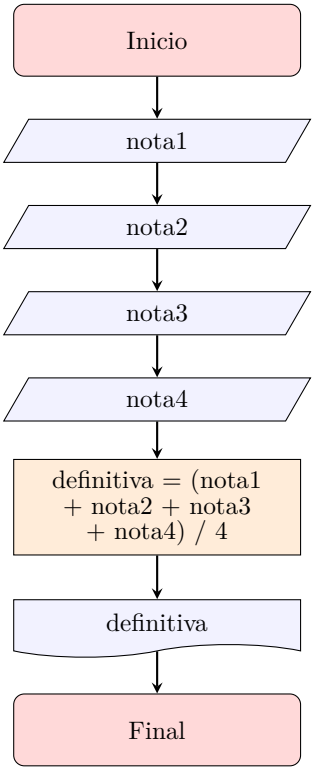


Figura 2.4: Diagrama de flujo del Algoritmo `DefinitivaEstudiante`

.:Ejemplo 2.5. *Construya un algoritmo que permita calcular el valor de los intereses ganados por una cantidad de dinero invertida en un Certificado de Depósito a Término (CDT) en un banco o entidad financiera durante un periodo de tiempo (en días), de acuerdo con la siguiente fórmula:*

$$valorIntereses = (cantidad * porcentajeInteres * periodo)/360$$

El algoritmo debe determinar también el valor total a retirar por el cliente que invirtió en el CDT al final del periodo, suponiendo que sobre los intereses ganados hay un descuento del 7 % por concepto del impuesto de retención en la fuente.

Análisis del problema:

- **Resultados esperados:** al finalizar la ejecución del algoritmo, se debe mostrar el valor de los intereses ganados por el CDT durante el periodo de tiempo, el descuento por concepto de impuesto de retención en la fuente y el valor total a retirar por el cliente.
- **Datos disponibles:** se requiere conocer: La cantidad a ingresar en

el CDT, el periodo de tiempo en días y el porcentaje de interés que se va a aplicar.

- **Proceso:** luego de que el usuario haya ingresado los datos requeridos, se debe calcular el valor de los intereses generados aplicando la fórmula dada en el enunciado del ejercicio, luego sacarle el 7 % de impuesto al valor de los intereses recién calculados y, por último, sumar la cantidad ingresada con el valor de los intereses y restar el valor de los impuestos obtenidos.
- **Variables requeridas:**
  - `cantidad`: valor total de dinero en el CDT
  - `periodo`: plazo a que se desea hacer el CDT
  - `porcentajeInteres`: tasa de interés del CDT
  - `valorIntereses`: valor real del interés calculado a partir de los datos ingresados.
  - `valorImpuesto`: valor de los impuestos a pagar
  - `netoPagar`: valor total a pagar.

Aclaración:



Note que para poder resolver los ejercicios que se plantean, es necesario tener una clara comprensión del problema; esto implica hacer un análisis que permita entender aquello de lo que se está hablando. Lo anterior es imprescindible en la resolución de cualquier tipo de problemas. Por lo anterior, a continuación se lleva a cabo una breve explicación de lo que es un CDT.

Un Certificado de Depósito a Término es un producto que ofrecen los bancos y que le permite a los clientes ahorrar de una forma diferente en una cuenta bancaria. El CDT se abre con una cantidad de dinero, por un periodo de tiempo determinado y recibe unos intereses durante ese periodo, al término del cual, el cliente recibe su dinero más el valor de los intereses ganados. Para obtener el valor de los intereses ganados, es necesario aplicar la fórmula expuesta anteriormente.

De acuerdo al análisis planteado, se propone el Algoritmo 2.5.

Algoritmo 2.5: CdtBancario

```
1  Algoritmo CdtBancario
2
3     // Algoritmo que calcula el valor a pagar en un CDT
4     // luego de un periodo de tiempo ingresado en días.
5
6     Real    cantidad, porcentajeInteres, valorIntereses,
7             valorImpuesto, netoPagar
8     Entero periodo
9
10    imprimir( "Ingrese la cantidad de dinero: " )
11    leer( cantidad )
12
13    imprimir( "Ingrese el periodo en días: " )
14    leer( periodo )
15
16    imprimir( "Ingrese el porcentaje de interés: " )
17    leer( porcentajeInteres )
18
19    valorIntereses = (cantidad * porcentajeInteres/100 *
         periodo)/360
20    valorImpuesto = valorIntereses * 0.07
21    netoPagar = cantidad + valorIntereses - valorImpuesto
22
23    imprimir( "Intereses ganados ", valorIntereses )
24    imprimir( "Valor del impuesto ", valorImpuesto )
25    imprimir( "Total a pagar al cliente ", netoPagar )
26 FinAlgoritmo
```

Al ejecutar el algoritmo:

```
Ingrese la cantidad de dinero: 1000000
Ingrese el periodo en días: 360
Ingrese el porcentaje(\%) de interés: 6
Intereses ganados: 60000.0
Valor del impuesto: 4200.0
Total a pagar al cliente  1055800.0
```

Explicación del algoritmo:

Luego de declarar las variables:

```
6     Real    cantidad, porcentajeInteres, valorIntereses,
7             valorImpuesto, netoPagar
8     Entero periodo
```

Se solicitan al usuario, con la instrucción `imprimir` los datos que requiere el algoritmo: cantidad de dinero, porcentaje[1] de interés y periodo;

```
10      imprimir( "Ingrese la cantidad de dinero: " )
11      leer( cantidad )
12
13      imprimir( "Ingrese el periodo en días: " )
14      leer( periodo )
15
16      imprimir( "Ingrese el porcentaje de interés: " )
17      leer( porcentajeInteres )
```

Con estos datos, se calcula el valor de los intereses ganados multiplicando la cantidad por el porcentaje (dividido entre 100.00) y por el periodo y, dividiendo todo esto en 360, ya que es la cantidad de días que tiene el año comercial. Al valor de los intereses obtenidos en la instrucción anterior, se le aplica el 7 % o 0.07 para determinar el valor a pagar por concepto de impuesto de retención en la fuente.

```
19      valorIntereses = (cantidad * porcentajeInteres/100 *
            periodo)/360
20      valorImpuesto = valorIntereses * 0.07
21      netoPagar = cantidad + valorIntereses - valorImpuesto
```

Por último, se suman la cantidad, el valor de los intereses y se resta el valor del impuesto con el fin de obtener el total o neto a pagarle al cliente. Los resultados obtenidos son mostrados a través de las instrucciones finales con `imprimir`.

```
23      imprimir( "Intereses ganados ", valorIntereses )
24      imprimir( "Valor del impuesto ", valorImpuesto )
25      imprimir( "Total a pagar al cliente ", netoPagar )
```

.:Ejemplo 2.6. *Se requiere construir un algoritmo al que se le ingrese un número entero de 3 cifras (Por ejemplo, 927 o 483). El algoritmo deberá determinar el valor de la suma de las 3 cifras, 18 para el primer ejemplo y 15 para el segundo ejemplo.*

Análisis del problema:

- **Resultados esperados:** luego de ejecutar el algoritmo, este debe entregar la suma de los tres dígitos del número ingresado.

[1]Note que en la ejecución, el porcentaje se ingresa como un número entre 1 y 100 y que en la fórmula es convertido a un número entre 0 y 1 dividiéndolo por 100.

- **Datos disponibles:** para este ejercicio, se debe tener un número cualquiera de 3 dígitos.
- **Proceso:** el algoritmo debe descomponer el número de 3 dígitos ingresado para obtener cada dígito por separado y luego hacer la suma de los mismos. La obtención de cada dígito se realiza haciendo una división, ya sea entera o modular. Por ejemplo, para obtener el primer dígito, se divide el número de tres cifras en 100, esto hará que se obtenga el primer número ya que será la parte entera de la división. El segundo dígito se obtendrá realizando una división entera entre 10 del número de tres cifras y luego, este resultado parcial dividiéndolo en 10 mediante el operador módulo. Por su parte, el tercer dígito se obtendrá haciendo una división modular del número de tres cifras en 10.
- **Variables requeridas:**
  - `numero`: valor numérico de tres cifras que será procesado
  - `digito1`: primera cifra del `numero`.
  - `digito2`: segunda cifra del `numero`.
  - `digito3`: tercera cifra del `numero`.
  - `suma`: valor que corresponde a la suma de los tres térmicos.

Aclaración:



Por ahora, es necesario suponer que el dato ingresado al algoritmo es un número de 3 dígitos y no uno con un número de dígitos diferente.

De acuerdo al análisis planteado, se propone el Algoritmo 2.6.

Algoritmo 2.6: SumaDigitosNumero

```
1  Algoritmo SumaDigitosNumero
2
3  // Este algoritmo obtiene la suma de los 3 dígitos que
4  // componen un número
5
6   Entero numero, digito1, digito2, digito3, suma
7
8   imprimir ( "Ingrese un número entero de tres dígitos ")
9   leer( numero )
10
11  digito1 = numero / 100
```

```
12   digito2 = (numero / 10)  % 10
13   digito3 = numero  % 10
14   suma    = digito1 + digito2 + digito3
15
16   imprimir( "La suma de los tres dígitos de " , numero,
17             " es ", suma)
18  FinAlgoritmo
```

Al ejecutar el algoritmo:

```
Ingrese un número entero de tres dígitos: 927
La suma de los tres dígitos de 927 es 18
```

Explicación del algoritmo:

Al principio del algoritmo (línea 6), se declaran las variables que se van a utilizar: una para almacenar el número; tres para los dígitos y una última para almacenar la `suma`.

```
6    Entero numero, digito1, digito2, digito3, suma
```

Se solicita el único dato disponible (`numero`).

```
8   imprimir ( "Ingrese un número entero de tres dígitos ")
9   leer( numero )
```

Luego se procede a realizar los cálculos que determinan los resultados esperados:

```
11   digito1 = numero / 100
12   digito2 = (numero / 10)  % 10
13   digito3 = numero  % 10
14   suma    = digito1 + digito2 + digito3
```

Con la instrucción `digito1 = numero / 100`, el número se divide en 100 y como resultado se obtiene la parte entera que será el primer dígito, que es almacenado en digito1. Con la instrucción digito2 = `(numero / 10) % 10`, se obtiene el segundo dígito de la siguiente forma: el numero al ser dividido entre 10 entrega como resultado los dos primeros dígitos que, luego al ser divididos en `% 10` retornará como resultado su residuo, es decir, el segundo dígito del número. La instrucción `digito3 = numero % 10`, entrega como resultado el residuo de esta división, esto es, el tercer dígito. Posteriormente, se suman los tres dígitos y su resultado se almacena en la variable suma que será mostrada con la instrucción para imprimir el mensaje.

En la Figura 2.5 se muestra la solución del Ejemplo 2.6 mediante un diagrama de flujo.

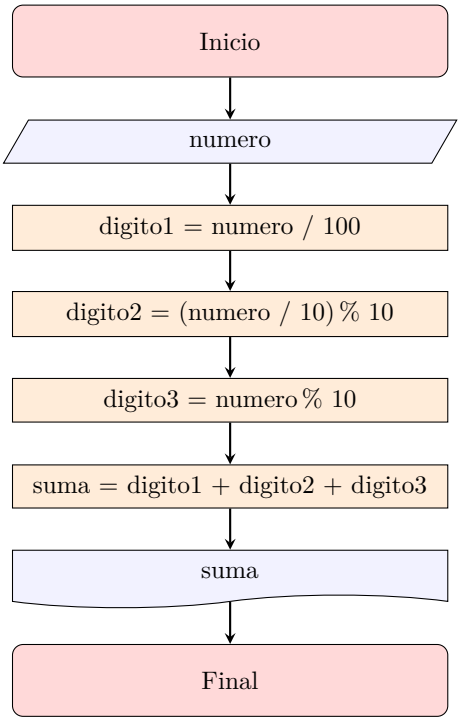


Figura 2.5: Diagrama de flujo del Algoritmo `SumaDigitosNumero`

.:Ejemplo 2.7. *Los empleados asalariados en Colombia deben realizar un aporte a seguridad social que consiste en un 4 % para la salud y un 4 % para la pensión. Diseñe un algoritmo que permita calcular el valor del aporte que debe realizar un empleado a salud y a pensión sobre el salario base, que determine el total del descuento por estos conceptos y cuál sería el valor del salario neto que recibiría el empleado luego de que le realicen los descuentos.*

Análisis del problema:

- **Resultados esperados:** cuando se haya ejecutado el algoritmo, este entregará como resultados el valor a aportar por salud, por pensión, la suma de estos y el salario neto que recibirá el empleado.
- **Datos disponibles:** se debe conocer el salario base del empleado.
- **Proceso:** se debe calcular el valor del aporte a salud y pensión, cada uno de ellos equivalente al 4 % sobre el salario base. Se debe obtener

el total que le descontarán al empleado, que será la suma del valor de los aportes calculados previamente. Por último, se debe calcular el salario neto a pagar que será la resta entre el salario base ingresado por el usuario y el descuento que le hacen al empleado.

- **Variables requeridas:**
  - `salarioBase`: valor del salario sin descuentos.
  - `aporteSalud`: valor del aporte a salud.
  - `aportePension`: valor del aporte de pensión.
  - `descuento`: valor del descuento.
  - `salarioNeto`: valor del salario con los descuentos.

Aclaración:



La seguridad social es un aporte que hacen tanto los empleadores como los empleados. A estos últimos les toca aportar un 4 % de su salario para salud y un 4 % para pensión. Estos aportes son descontados del salario base para hacer los respectivos pagos a las empresas que prestan estos servicios, por lo que el salario neto debe contemplar el descuento de estos valores. Por ejemplo, suponiendo que un Empleado recibe como salario base 1000000 (un millón de pesos), el descuento por salud corresponde a 40000, lo mismo que el descuento por pensión; así, el salario neto que recibiría el empleado sería de 920000 (novecientos veinte mil pesos).

De acuerdo al análisis planteado, se propone el Algoritmo 2.7.

Algoritmo 2.7: SeguridadSocial

```
1  Algoritmo SeguridadSocial
2    // Este algoritmo calcula el valor del aporte por salud y
3    // pensión que se hacen sobre el salario base de un
4    // empleado y determina el valor total de estos descuentos
5    // y el salario neto a pagar al empleado
6
7    Real salarioBase, aporteSalud, aportePension,
8         descuento,   salarioNeto
9
10   imprimir( "Ingrese el salario base del empleado " )
11   leer( salarioBase )
12
13   aporteSalud   = salarioBase * 0.04
14   aportePension = salarioBase * 0.04
```

```
15   descuento     = aporteSalud + aportePension
16   salarioNeto   = salarioBase - descuento
17
18   imprimir( "El aporte a salud es de " , aporteSalud )
19   imprimir( "El aporte a pensión es de " , aportePension )
20   imprimir( "El descuento es de " , descuento )
21   imprimir( "El salario neto a pagar es " , salarioNeto )
22 FinAlgoritmo
```

Al ejecutar el algoritmo:

```
Ingrese el salario base del empleado 1000000
El aporte a salud es de 40000
El aporte a pensión es de 40000
El descuento es de 80000
El salario neto a pagar es 920000
```

Explicación del algoritmo:

En principio, se declaran las variables necesarias para almacenar el dato que se va a ingresar:

```
7   Real salarioBase, aporteSalud, aportePension,
8        descuento,   salarioNeto
```

Posteriormente, se solicita el ingreso del salario base que gana el empleado,

```
10   imprimir( "Ingrese el salario base del empleado " )
11   leer( salarioBase )
```

Luego se calculan los aportes que corresponden al 4 % cada uno; note que en la operación se utiliza el valor de 0.04. Cada descuento se calcula en una línea separada, una para el aporte a salud y otra para el aporte a pensión.

```
13   aporteSalud   = salarioBase * 0.04
14   aportePension = salarioBase * 0.04
```

Enseguida, se determina el descuento que corresponde a la suma de los aportes.

```
15    descuento     = aporteSalud + aportePension
```

Por último, se obtiene el salario neto restando del salario base el descuento.

```
16    salarioNeto   = salarioBase - descuento
```

Finalmente, se imprimen los datos calculados.

```
18   imprimir( "El aporte a salud es de " , aporteSalud )
19   imprimir( "El aporte a pensión es de " , aportePension )
20   imprimir( "El descuento es de " , descuento )
21   imprimir( "El salario neto a pagar es " , salarioNeto )
```

En la Figura 2.6 se muestra la solución del Ejemplo 2.7 mediante un diagrama de flujo.

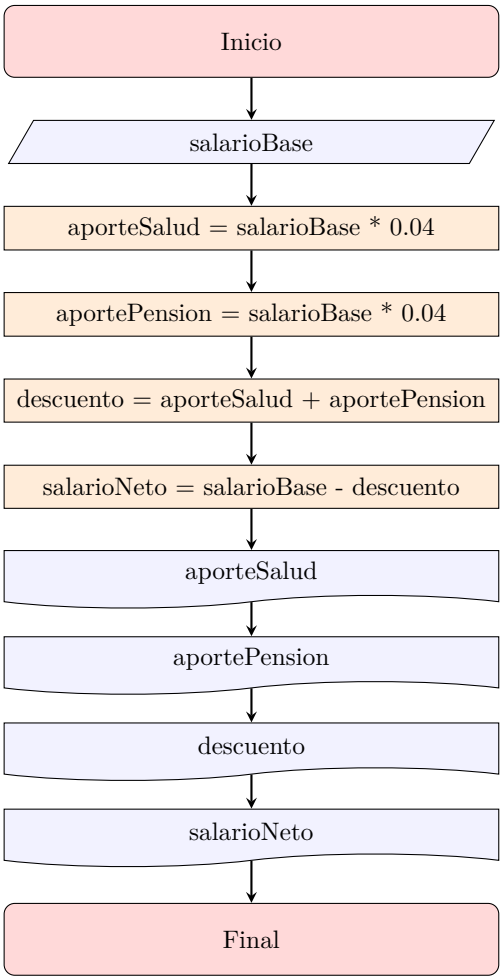


Figura 2.6: Diagrama de flujo del Algoritmo `SeguridadSocial`

.:Ejemplo 2.8. *Se requiere construir un algoritmo que, al ingresarle un número de días cualquiera, este permita saber cuántos minutos y cuántos segundos tiene la cantidad de días ingresados. Para ello, es imprescindible saber cuántos minutos tiene un día, así como también el número de segundos contenidos en cada día, esto se puede obtener al multiplicar el número de días que se ingresaron por la cantidad de horas que tiene cada día (24) y a su vez, por la cantidad de minutos que tiene cada hora (60) y por la cantidad de segundos que tiene un minuto (60).*

Análisis del problema:

- **Resultados esperados:** cuando se haya ejecutado el algoritmo, este entregará como resultados la cantidad de minutos contenidos en el número de días ingresados, así como la cantidad de segundos.
- **Datos disponibles:** se debe conocer el número de días que van a ser convertidos.
- **Proceso:** para obtener la cantidad de minutos que tienen los días ingresados, se crea una expresión aritmética que multiplique los días por 24 horas que tiene cada día y luego por 60 minutos que tiene cada hora. Para obtener la cantidad de segundos, solo hace falta multiplicar los minutos obtenidos por 60.
- **Variables requeridas:**
  - `numeroDias`: cantidad de días.
  - `numeroMinutos`: cantidad de minutos.
  - `numeroSegundos`: cantidad de segundos.

Aclaración:



Muchos problemas algorítmicos consisten en realizar algún tipo de conversión, por ejemplo, podría necesitarse convertir una cantidad de dinero de una moneda de un país a otra de otro país, convertir una longitud que se encuentra en una unidad de medida a otra, convertir una temperatura que está en una escala de medida en otra. Este ejercicio es un ejemplo prototipo de conversión. Para poder realizar un ejercicio de conversión, se hace necesario conocer el equivalente de una medida con respecto a la otra, de esta manera, será posible construir una expresión que facilite hacer la conversión.

De acuerdo al análisis planteado, se propone el Algoritmo 2.8.

Algoritmo 2.8: ConversionDias

```
1  Algoritmo ConversionDias
2
3    // Este algoritmo encuentra la cantidad de minutos
4    // contenidos en un determinado número de días,
5    // así como la cantidad de segundos.
6
7    Entero numeroDias, cantidadMinutos, cantidadSegundos
8
9    imprimir( "Ingrese el número de días: " )
10   leer( numeroDias )
11
12   cantidadMinutos  = numeroDias * 24 * 60
13   cantidadSegundos = cantidadMinutos * 60
14
15   imprimir( numeroDias, " días equivalen a: " )
16   imprimir( cantidadMinutos,  " Minutos" )
17   imprimir( cantidadsegundos, " Segundos" )
18 FinAlgoritmo
```

Al ejecutar el algoritmo:

```
Ingrese el número de días: 2
2 Días equivalen a:
2880 Minutos
172800 Segundos
```

Explicación del algoritmo:

Este algoritmo inicia como todos, declarando las variables necesarias.

```
7  Entero numeroDias, cantidadMinutos, cantidadSegundos
```

Posteriormente, se solicita el dato con el que se va a realizar la conversión, este dato es el número de días.

```
9    imprimir( "Ingrese el número de días: " )
10   leer( numeroDias )
```

Una vez se tiene el número de días, se procede a realizar el primer cálculo que consiste en determinar la cantidad de minutos; para ello, se multiplican los días por 24 (ya que cada día tiene 24 horas) y este resultado se multiplica a su vez por 60 (pues cada hora tiene 60 minutos). Para conocer la cantidad de minutos que tienen los días ingresados, basta con multiplicar la cantidad de minutos ya obtenidos en el cálculo anterior por 60 (puesto que cada minuto consta de 60 segundos).

```
12   cantidadMinutos  = numeroDias * 24 * 60
13   cantidadSegundos = cantidadMinutos * 60
```

Para finalizar el algoritmo, se muestran los resultados solicitados con la instrucción `imprimir`.

```
15   imprimir( numeroDias, " días equivalen a: " )
16   imprimir( cantidadMinutos,  " Minutos" )
17   imprimir( cantidadsegundos, " Segundos" )
```

En la Figura 2.7 se muestra la solución del Ejemplo 2.8 mediante un diagrama de flujo.

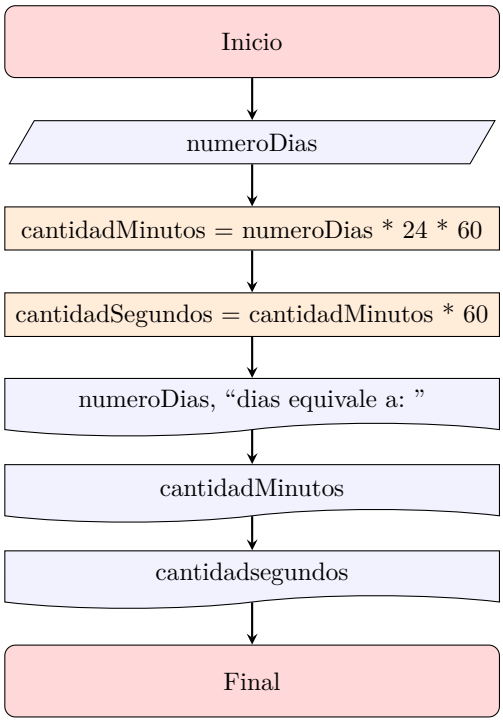


Figura 2.7: Diagrama de flujo del Algoritmo `ConversionDias`

.:Ejemplo 2.9. *Construya un algoritmo que permita encontrar las dos soluciones reales a una ecuación algebraica de segundo grado a partir de la fórmula general. Suponga por ahora que el ejercicio planteado no tiene soluciones imaginarias.*

Fórmula general:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Aclaración:



En Álgebra elemental se estudian diferentes tipos de ecuaciones; unas de ellas son las ecuaciones de segundo grado. Las ecuaciones de este tipo pueden resolverse con la fórmula general que se acaba de presentar, conociendo por supuesto, los valores de `a`, `b` y `c`, produciendo dos respuestas x_1 y x_2, una cuando se toma el signo + en la fórmula, antes del radical y la otra al tomar el signo menos.

Las ecuaciones de segundo grado, presentan la siguiente forma, de donde se obtienen los valores con los cuales trabajar:

$$ax^2 + bx + c = 0$$

Análisis del problema:

- **Resultados esperados:** se espera que el algoritmo entregue como resultados las dos respuestas a la ecuación: `x1` y `x2`, suponiendo que son reales[2].
- **Datos disponibles:** se deben conocer los valores de `a`, `b` y `c` en la ecuación dada.
- **Proceso:** luego de que se hayan ingresado los datos correspondientes a los coeficientes de `a`, `b` y `c`, debe aplicarse la fórmula general escrita en lenguaje algorítmico, una para encontrar `X1` (usando el signo + antes del radical) y otra para encontrar `X2` (usando el signo – antes del radical). Es necesario tener en cuenta que, como la fórmula implica elevar al cuadrado la variable `b` y también obtener la raíz cuadrada de una parte de la fórmula, en ella deben usarse funciones del lenguaje algorítmico.
- **Variables requeridas:**
  - `a`: valor del primer coeficiente de la ecuación.
  - `b`: valor del segundo coeficiente de la ecuación.

[2]Las ecuaciones de segundo grado también pueden dar un resultado con valores imaginarios, pero estos escapan al ejercicio que se plantea, ya que para poder obtenerlos sería necesario utilizar otras estructuras algorítmicas que se verán más adelante en el libro.

- c: valor del tercer coeficiente de la ecuación.
- x1: primera solución de la ecuación.
- x2: segunda solución de la ecuación.

De acuerdo al análisis planteado, se propone el Algoritmo 2.9.

Algoritmo 2.9: FuncionCuadratica

```
1  Algoritmo FuncionCuadratica
2    // Este algoritmo calcula las dos soluciones de una
3    // ecuación cuadrática. Teniendo en cuenta que estas
4    // deben ser reales por ahora
5
6    Real a, b, c, x1, x2
7
8    imprimir( "Ingrese el primer coeficiente: " )
9    leer( a )
10
11   imprimir( "Ingrese el segundo coeficiente: " )
12   leer( b )
13
14   imprimir( "Ingrese el tercer coeficiente: " )
15   leer( c )
16
17   x1 = (-b + raizCuadrada( b^2 - 4 * a * c))/(2 * a )
18   x2 = (-b - raizCuadrada( b^2 - 4 * a * c))/(2 * a )
19
20   imprimir( "Primera solución: ", x1)
21   imprimir( "Segunda solución: ", x2)
22 FinAlgoritmo
```

Al ejecutar el algoritmo:

```
Ingrese el primer coeficiente: 3
Ingrese el segundo coeficiente: 8
Ingrese el tercer coeficiente: 4
Primera solución -0.6666666666666666
Segunda solución -2.0
```

Explicación del algoritmo:

Para poder resolver la ecuación de segundo grado, se requieren los valores de a, b y c; razón por la cual se le solicita al usuario el ingreso de estos valores.

```
6    Real a, b, c, x1, x2
7
8    imprimir( "Ingrese el primer coeficiente: " )
9    leer( a )
```

```
10
11   imprimir( "Ingrese el segundo coeficiente: " )
12   leer( b )
13
14   imprimir( "Ingrese el tercer coeficiente: " )
15   leer( c )
```

Luego, se convierte la fórmula algebraica a expresión de computador utilizando en ella la función `raizCuadrada()`. Es necesario aplicar esta fórmula dos veces, una para encontrar el valor de `x1` y otra para hallar `x2`.

```
17   x1 = (-b + raizCuadrada( b^2 - 4 * a * c))/(2 * a )
18   x2 = (-b - raizCuadrada( b^2 - 4 * a * c))/(2 * a )
```

Cuando ya se han obtenido estos valores, se muestran con la instrucción `imprimir`.

```
20   imprimir( "Primera solución: ", x1)
21   imprimir( "Segunda solución: ", x2)
```

.:Ejemplo 2.10. *Un instalador de pisos requiere saber la cantidad de cajas de cerámica y el costo total de las mismas, que debe comprar para colocarle el piso a una casa o apartamento. El instalador toma las medidas con el fin de conocer la cantidad de metros cuadrados que tiene el inmueble. Sabe que cada caja del producto cubre un área de* 2.26 m^2 *y que, de acuerdo a las características de la cerámica que seleccione, cada caja tiene su costo particular. Construya un algoritmo que al ingresarle la cantidad de metros cuadrados de la casa o apartamento en la que se va a instalar el piso y el costo de la caja de cerámica, este permita saber cuántas cajas debe comprar y el costo total de las mismas.*

Aclaración:



Este ejercicio pretende mostrar la aplicación práctica de los algoritmos en el cálculo de cantidades de material para la construcción. De esta forma se expone la necesidad de crear algoritmos para calcular cualquier cantidad de material, incluso en la elaboración de diferentes tipos de productos. Conociendo el tamaño de la obra y las dimensiones de cada unidad de materia prima, se puede obtener la cantidad de unidades a comprar, luego, conociendo el precio de la unidad de materia prima, es posible calcular su valor.

Análisis del problema:

- **Resultados esperados:** el algoritmo debe determinar el número de cajas de cerámica a comprar y el costo total de las mismas.
- **Datos disponibles:** se debe conocer la cantidad de metros cuadrados que posee el inmueble (casa o apartamento) donde se va a instalar la cerámica y el costo de la caja de cerámica seleccionada para hacer la instalación.
- **Proceso:** con la cantidad de metros cuadrados se debe realizar una división entre `2.26`, que corresponde al número de metros cuadrados cubiertos por la cerámica que trae cada caja; con esto se obtendrá el número de cajas necesarias. Ahora, conociendo el número de cajas, se multiplica por el valor de las cajas y se obtendrá el costo de este material.
- **Variables requeridas:**
  - `tamanioPiso`: valor que representa el tamaño del piso.
  - `valorCaja`: valor de cada caja.
  - `cantidadCajas`: cantidad de cajas necesarias.
  - `valorTotal`: valor total a pagar.

De acuerdo al análisis planteado, se propone el Algoritmo 2.10.

Algoritmo 2.10: CeramicaPiso

```
1  Algoritmo CeramicaPiso
2    /* Este algoritmo determina la cantidad de cajas de
3       cerámica que se deben comprar para instalar en
4       un inmueble y su costo total. */
5
6    Real tamanioPiso, valorCaja, cantidadCajas, valorTotal
7
8    imprimir( "Ingrese la cantidad de metros cuadrados: " )
9    leer( tamanioPiso )
10
11   imprimir( "Ingrese el valor de la caja de cerámica: " )
12   leer( valorCaja )
13
14   cantidadCajas = tamanioPiso / 2.26
15   valorTotal    = cantidadCajas * valorCaja
16
17   imprimir( "Se deben comprar " )
18   imprimir(  cantidadCajas, " de cajas de cerámica " )
19   imprimir( "Cuyo valor es  ", valorTotal)
20 FinAlgoritmo
```

Al ejecutar el algoritmo:

```
Ingrese la cantidad de metros cuadrados: 84
Ingrese el valor de la caja de cerámica: 20000
Se deben comprar
37.16814159292036 de cajas de cerámica
Cuyo valor es 743362.8318584071
```

Explicación del algoritmo:

Lo primero que se lleva a cabo en el algoritmo es la declaración de las variables a utilizar:

```
5   Real tamanioPiso, valorCaja, cantidadCajas, valorTotal
```

Luego, se solicitan los datos necesarios para realizar los cálculos. Estos datos son: la cantidad de metros cuadrados a instalar y el valor de la caja de cerámica.

```
7   imprimir( "Ingrese la cantidad de metros cuadrados: " )
8   leer( tamanioPiso )
9
10  imprimir( "Ingrese el valor de la caja de cerámica: " )
11  leer( valorCaja )
```

Posteriormente el algoritmo procede a encontrar el número de cajas, esto se logra dividiendo los metros cuadrados a instalar en 2.26 que es la cantidad de metros cuadrados que trae cada caja; Más adelante se procede a determinar el costo de las cajas que hay que comprar multiplicando la cantidad de cajas obtenidas previamente por el valor de cada una de ellas.

```
13  cantidadCajas = tamanioPiso / 2.26
14  valorTotal    = cantidadCajas * valorCaja
```

Al final, se muestran los resultados con la instrucción `imprimir`.

```
16  imprimir( "Se deben comprar " )
17  imprimir(  cantidadCajas, " de cajas de cerámica " )
18  imprimir( "Cuyo valor es  ", valorTotal)
```

En la Figura 2.8 se muestra la solución del Ejemplo 2.10 mediante un diagrama de flujo.

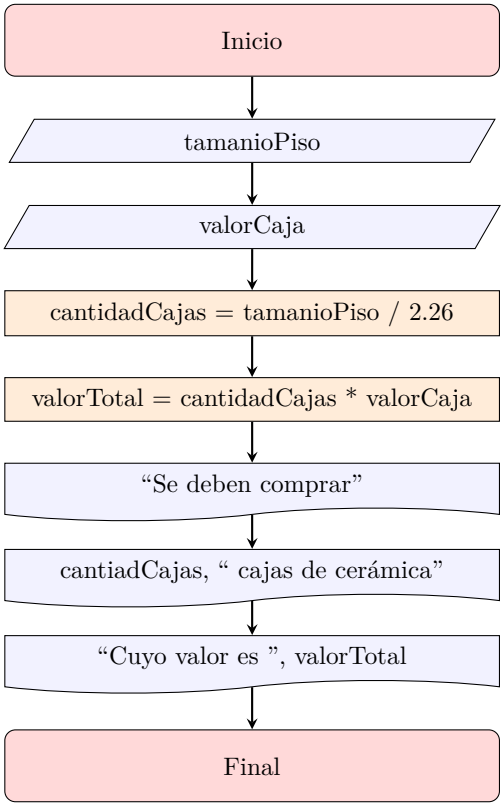


Figura 2.8: Diagrama de flujo del Algoritmo `CeramicaPiso`

2.2. Pruebas de escritorio

Estas pruebas pretenden verificar el funcionamiento del algoritmo y encontrar posibles errores en su ejecución que no dejan que el algoritmo entregue los resultados correctos o esperados. Para [Trejos, 2004], la prueba de escritorio es la prueba reina que permite saber si un algoritmo hace lo que debe hacer y si produce los resultados correctos. Posterior a la realización de las pruebas de escritorio, en el caso de encontrar alguna inconsistencia en los resultados esperados, será necesario revisar las instrucciones con las que se diseñó el algoritmo para determinar cuál o cuáles de ellas son las que están generando el error.

Una prueba de escritorio se diseña de la siguiente manera:

- Se seleccionan unos datos disponibles que servirán como entrada para el algoritmo.

- Se hacen los respectivos cálculos y se encuentran los resultados esperados.
- Posteriormente, se recorre el algoritmo, línea por línea y se va verificando si el algoritmo hace lo que debe de hacer o no.
- Se comparan los resultados que generó el algoritmo con los esperados y se concluye si el algoritmo funciona correctamente o no.
- En el caso de que el algoritmo no haya entregado los resultados correctos, será necesario revisar el código escrito tratando de encontrar la instrucción o instrucciones que están causando el error.

Aclaración:



Una forma de realizar la prueba de escritorio a un algoritmo consiste en construir una tabla (llamada **tabla de verificación**) en la que se relacionan las variables, primero las que reciben los datos ingresados, luego las que almacenan los cálculos y por último, las instrucciones que muestran los resultados esperados. Si el algoritmo incluye estructuras algorítmicas con condiciones, estas deben involucrarse en la tabla en el orden en que van apareciendo en el algoritmo.
Es recomendable ejecutar el algoritmo varias veces con diferentes datos disponibles para determinar su comportamiento en cada ejecución.

2.2.1 Ejemplos

En la Tabla 2.1 se llevará a cabo la prueba de escritorio para el algoritmo del Ejemplo 2.1, denominado `SalarioEmpleado` que tiene que ver con el salario a apagar a un empleado al que le pagan de acuerdo al número de horas laboradas y al valor de la hora.

Ejecución	`numeroHoras`	`valorHora`	`salarioAPagar`	`Respuesta`
1	10	30000	300000	300000
2	40	25000	1000000	1000000
3	20	27000	540000	540000

Tabla 2.1: Prueba de escritorio para el algoritmo del Ejemplo 2.1

En la Tabla 2.2, se puede observar la prueba de escritorio para el algoritmo del Ejemplo 2.4 denominado `definitivaEstudiante`:

Ejecución	nota1	nota2	nota3	nota4	definitiva	Resultado
1	3.4	4.2	3.0	5.0	3.9	3.9
2	2.0	3.0	2.0	3.0	2.5	2.5
3	1.0	4.0	3.0	4.0	3.0	3.0

Tabla 2.2: Prueba de escritorio para el algoritmo del Ejemplo 2.4

Y en la Tabla 2.3 la prueba de escritorio para el algoritmo del ejercicio número 2.5 denominado `CDTBancario`.

Por limitaciones de espacio, las variables que aparecen en la Tabla 2.3 fueron renombradas de la siguiente forma:

- `cantidad` por `cant`
- `porcentajeInteres` por `pInteres`
- `valorIntereses` por `vIntereses`
- `valorImpuesto` por `vImpuesto`
- `netoPagar` por `nPagar`

Ej.	cant	periodo	pInteres	vIntereses	vImpuesto	nPagar
1	3.4	4.2	3.0	5.0	3.9	3.9
2	2.0	3.0	2.0	3.0	2.5	2.5
3	1.0	4.0	3.0	4.0	3.0	3.0

Tabla 2.3: Prueba de escritorio para el algoritmo del Ejemplo 2.5

2.3. Ejercicios propuestos

Diseñe algoritmos utilizando tanto pseudocódigo como diagramas de flujo para los problemas que se enuncian a continuación. Una vez se hayan construido, elabore la prueba de escritorio con la tabla de verificación para determinar si el algoritmo se ejecuta adecuadamente y entrega los resultados esperados:

1. Suponga que, para el algoritmo del cálculo de la nota definitiva expuesto en el Ejemplo 2.4, el acuerdo al que llegaron profesor y

estudiantes consiste en darle un porcentaje a cada una de las 4 notas, de la siguiente manera:

Nota Parcial	Porcentaje
Primera	15 %
Segunda	15 %
Tercera	30 %
Cuarta	40 %

Tabla 2.4: Porcentaje de notas para el Ejercicio Propuesto 1

Reescriba el algoritmo, de tal forma que se pueda obtener la nota definitiva a partir de las notas y sus porcentajes. Compare los resultados obtenidos por este algoritmo con los que se obtienen en el algoritmo del Ejemplo 2.4.

2. Imagine que se conoce el valor del lado de un cubo. Elabore un algoritmo que permita calcular el área total de sus seis caras, el perímetro y el volumen del cubo. El área de una cara se obtiene por el producto de sus dos lados. El volumen se obtiene elevando al cubo su lado conocido y el perímetro es la suma de todos sus lados.

3. Un empleado fue contratado durante un periodo específico en días y por un salario, ambos conocidos. Construya un algoritmo que permita calcular el valor de su liquidación al terminar el contrato. La liquidación se compone de prima, cesantías, intereses a las cesantías y vacaciones. Para calcular estos valores, se usan las siguientes fórmulas:

   - **Prima**: `(salario * diasLaborados) / 360`
   - **Cesantías**: `(salario * diasLaborados) / 360`
   - **Intereses cesantías**: `cesantías * (12 % / diasLaborados)`
   - **Vacaciones**: `(salario * diasLaborados) / 720`

4. Diseñe un algoritmo que, al ingresarle el costo de un producto y la cantidad de ese producto que compra un cliente, calcule y muestre el valor a pagar por el cliente, obteniendo igualmente, el valor del impuesto al valor agregado IVA, que corresponde al 16 % del valor de la compra y que también el cliente debe pagar.

5. Construya un algoritmo al que se le ingresa el valor de un número determinado y el algoritmo realiza la suma de los números que le preceden desde 1. Para ello, utilice la siguiente fórmula:

$$suma = \frac{n * (n+1)}{2}$$

6. Un agente inmobiliario necesita un algoritmo con el que pueda saber el precio al cual debe vender una finca o terreno; para ello, el algoritmo debe conocer el número de metros cuadrados que tiene el terreno, el valor del metro cuadrado y el valor de la comisión por la venta que equivale al 2 % del valor de los metros cuadrados de los que consta el terreno. Escriba un algoritmo que permita saber el valor en el que se debe vender una finca o terreno.

7. Construya un algoritmo que reciba como entrada un ángulo expresado en grados, minutos y segundos y se entregue como resultado la medida del ángulo en radianes.

8. Un artesano se enfrenta diariamente al problema de determinar cuál será el precio de venta de un producto fabricado. Para asignar el precio, el artesano identifica el valor de la materia prima utilizada, el valor de la mano de obra y a estos les incrementa la utilidad que desea ganar, que es del 30 %. Diseñe un algoritmo al que se le ingresa el nombre de un producto, el valor de las materias primas usadas en su elaboración y el valor de la mano de obra con que se construyó y el algoritmo determina el precio de venta del producto.

9. Construir un algoritmo que permita determinar cuál es la altura de un edificio del que se deja caer una pelota que tarda 3 segundos en tocar el piso y cuál es la velocidad con la que la pelota llega al piso. Para escribir el algoritmo, utilice las siguientes fórmulas:

$$altura = velocidadInicial * tiempo + \frac{1}{2} * gravedad * tiempo^2$$

$$velocidadFinal = velocidadInicial^2 + 2 * gravedad * altura$$

   Recuerde que en la tierra, la gravedad es una constante de valor `9.8`.

10. Dada una temperatura en grados centígrados, construya un algoritmo que convierta esa temperatura tanto a grados Kelvin como a grados Fahrenheit. Use las siguientes fórmulas.

$$fahrenheit = centigrados * \frac{9}{5} + 32$$

$$kelvin = centigrados + 273$$

Capítulo 3

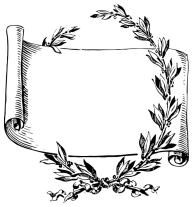
Estructuras de decisión

Cuando se está depurando, el programador novato introduce código correctivo; el experto elimina el código defectuoso

Richard Pattis

Objetivos del capítulo:

- Construir algoritmos que involucran condiciones simples, compuestas, anidadas y múltiples.
- Reconocer y utilizar los árboles de decisión en el análisis de problemas.
- Elaborar diagramas de flujo que representen los tipos de decisiones.



Una de las estructuras fundamentales en la creación de algoritmos son las estructuras de decisión. Una decisión permite que un algoritmo ejecute un conjunto u otro de instrucciones dependiendo del valor de verdad de una expresión lógica / relacional. Si dentro del conjunto de instrucciones a ejecutar no hay otras decisiones se habla entonces de decisiones simples, pero si las hubiera se denominan decisiones anidadas.

3.1. Decisiones compuestas

La forma general de esta estructura es presentada en el segmento del Algoritmo 3.1 y su diagrama de flujo fue presentado en la página 55.

Algoritmo 3.1: Forma general del `Si`

```
1  Si( condición ) Entonces
2    InstrucciónV-1
3    InstrucciónV-2
4    ...
5    InstrucciónV-n
6  SiNo
7    InstrucciónF-1
8    InstrucciónF-2
9    ...
10   InstrucciónF-m
11 FinSi
```

En esta estructura, `Instrucción-V` representa las instrucciones en la parte `Verdadera` de la decisión, es decir la parte que se ejecutan cuando la condición es `Verdadera`. Por otro lado, `Instrucción-F` representa las instrucciones en la parte `Falsa` de la decisión, es decir, las instrucciones que se ejecutan cuando la condición es `Falsa`.

En algunos casos, es posible no se requiera ejecutar instrucciones cuando la condición sea `Falsa`, así que dicha zona se puede eliminar de la estructura, quedando como la presentada en el segmento del Algoritmo 3.2, a este tipo de funciones se le denomina "**Decisión simple**" y su respectivo diagrama se puede observar en la página 54.

Algoritmo 3.2: Forma general del `Si` sin el `SiNo`

```
1  Si( condición ) Entonces
2    InstrucciónV-1
3    InstrucciónV-2
4    ...
5    InstrucciónV-n
6  FinSi
```

Para poner la definición en otros términos, imagine que desea planear las actividades para el día domingo, entonces usted reflexiona y piensa que: "*Si hace calor, me visto con ropa deportiva y voy al parque; sino, me visto informalmente y voy al cine*". Una forma para representar este tipo de decisiones es por medio de un árbol de decisión (Ver Figura 3.1).

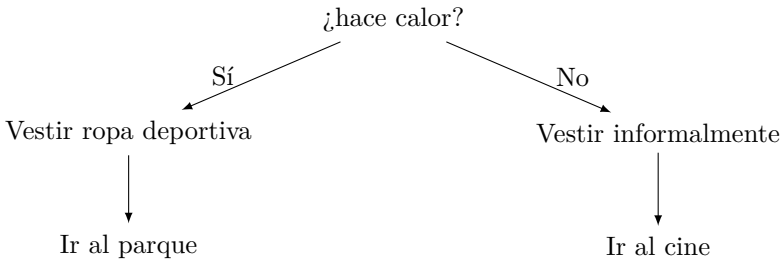


Figura 3.1: Árbol de decisión para planear el domingo

Un árbol de decisión es una representación gráfica de una o varias decisiones relacionadas, así como todas las acciones asociadas a cada una de las alternativas de la decisión. Es decir, qué sucede cuando la respuesta a la decisión es `Verdadera` (**`sí`**) o `Falsa` (**`no`**). Los árboles de decisión son una herramienta fundamental para analizar los problemas que requieran de decisiones y pueden ser usados en contexto diferentes a la programación.

Aclaración:



En la Figura 3.1 se observa que: se viste deportivamente y va parque o se viste informalmente y va al cine; pero en ningún caso, realiza las dos actividades.

Otro tipo de representación más algorítmica es mediante un diagrama de flujo (Ver Figura 3.2). La diferencia principal es que con un diagrama de flujo es posible representar todo un algoritmo, mientras que con un árbol de decisión, solo se representa la parte asociada a la estructura de decisión del algoritmo, la cual en muchos casos es suficiente para realizar el análisis de los problemas.

Es posible que una decisión posea solo la parte verdadera de la decisión, tal y como es ilustrado mediante el Ejemplo 3.1.

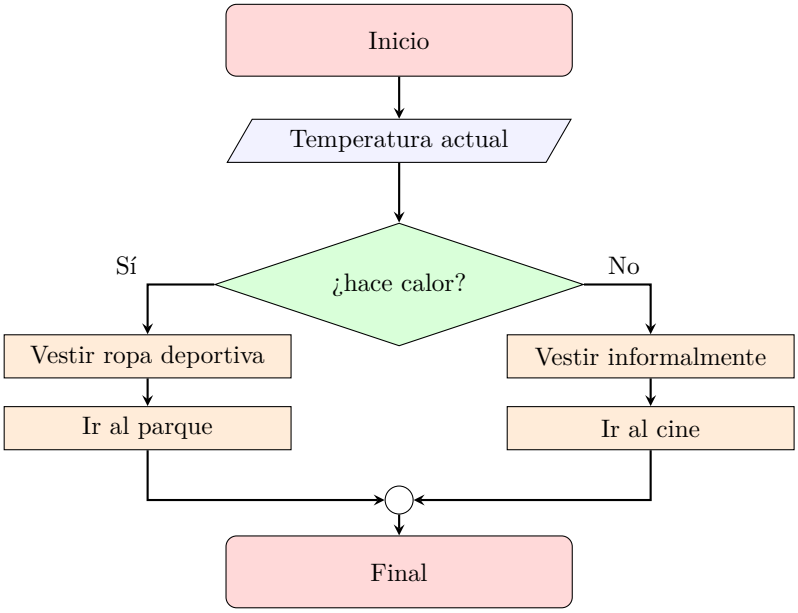


Figura 3.2: Planeando el domingo - Versión 1

.:Ejemplo 3.1. *Diseñe un algoritmo que reciba una nota definitiva entre 0.0 y 5.0. El algoritmo debe imprimir el valor ingresado, y de ser una nota mayor o igual a 4.0, deberá imprimir un mensaje de felicitaciones.*

Análisis del problema:

- **Resultados esperados:** la impresión de una nota definitiva y el mensaje de "Felicitaciones" siempre y cuando la nota definitiva sea igual o superior a 4.0.
- **Datos disponibles:** solo es necesaria la nota definitiva.
- **Proceso:** se le solicita al usuario ingresar la nota definitiva, luego se imprime dicha nota, finalmente se verifica si la nota es mayor o igual a 4.0 para determinar si es no necesario imprimir un mensaje de "Felicitaciones". (Ver el árbol de decisión, Figura 3.3).

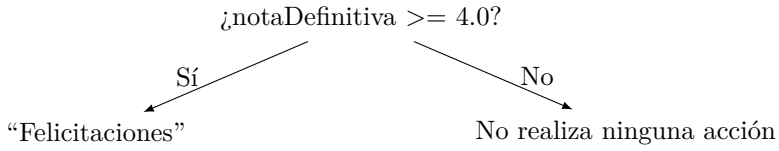


Figura 3.3: Árbol de decisión del Ejemplo 3.1

- **Variables requeridas:**
  - `notaDefinitiva`: almacena el valor de la nota definitiva del estudiante.

De acuerdo al análisis planteado, se propone el Algoritmo 3.3.

Algoritmo 3.3: Felicitaciones

```
1  Algoritmo Felicitaciones
2    /* Este algoritmo imprime la nota ingresada y si la nota
3       es mayor o igual a 4.0 se imprime un mensaje de
4       "Felicitaciones"
5     */
6
7    // Declaración de variables
8    Real notaDefinitiva
9
10   // Dato disponible
11   imprimir( "Ingrese la nota definitiva: " )
12   leer( notaDefinitiva )
13
14   // Resultados esperados
15   imprimir( "Su definitiva es: ", notaDefinitiva )
16
17   Si( notaDefinitiva >= 4.0 )
18      imprimir( " Felicitaciones" )
19   FinSi
20
21 FinAlgoritmo
```

Al ejecutar el algoritmo:
Primera ejecución

```
Ingrese la nota definitiva: 2.5
Su definitiva es: 2.5
```

Segunda ejecución

```
Ingrese la nota definitiva: 4.3
Su definitiva es: 4.3 Felicitaciones
```

Explicación del algoritmo:

Lo primero es declarar la única variable que el algoritmo requiere, en este caso (`notaDefinitiva`) de tipo `Real`. Es necesario que sea de este tipo porque las notas pueden llegar a tener una parte decimal; luego se procede a realizar su respectiva lectura, es decir, solicitarle al usuario el valor de la nota definitiva del estudiante.

```
11    imprimir( "Ingrese la nota definitiva: " )
12    leer( notaDefinitiva )
```

Observe que la impresión del mensaje “Felicitaciones” se realiza solo cuando la nota ingresada es mayor o igual 4.0; además note que es necesario dejar un espacio en blanco antes de la palabra para que ella sea impresa separadamente de la nota definitiva. Esta estructura se conoce como decisión simple.

```
17    Si( notaDefinitiva >= 4.0 )
18       imprimir( " Felicitaciones" )
19    FinSi
```

De ser una nota inferior a 4.0, el algoritmo continua su ejecución, para este caso continua a la línea 20 para terminar posteriormente el algoritmo.

En la Figura 3.4 se muestra la solución del Ejemplo 3.1 mediante un diagrama de flujo.

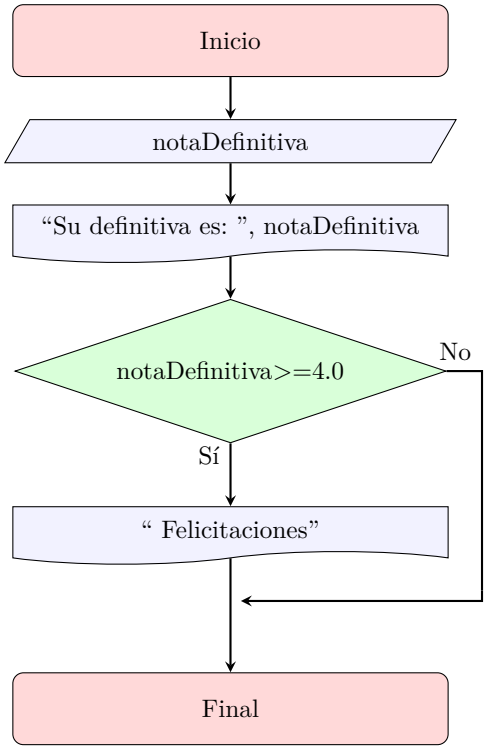


Figura 3.4: Diagrama de flujo del Algoritmo `Felicitaciones`

.:Ejemplo 3.2. *Diseñe un algoritmo que reciba una nota definitiva entre 0.0 y 5.0. El algoritmo debe imprimir el valor ingresado, y un mensaje que indique si el estudiante "Ganó el curso" o "Perdió el curso". Para el ejemplo, se gana el curso solo si la nota definitiva es mayor o igual a 3.0.*

Análisis del problema:

- **Resultados esperados:** la impresión de una nota definitiva con un mensaje que indique si el estudiante ganó o perdió el curso.
- **Datos disponibles:** solo es necesaria la nota definitiva.
- **Proceso:** se le solicita al usuario ingresar la nota definitiva, luego se imprime dicha nota, finalmente se verifica si la nota es mayor o igual a 3.0 para determinar si el estudiante ganó o perdió el curso. (Ver el árbol de decisión, Figura 3.5).

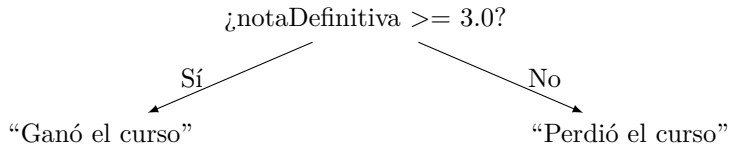


Figura 3.5: Árbol de decisión del Ejemplo 3.2

- **Variables requeridas:**
  - `notaDefinitiva`: almacena el valor de la nota definitiva del estudiante.

.

De acuerdo al análisis planteado, se propone el Algoritmo 3.4.

Algoritmo 3.4: Definitiva

```
1 Algoritmo Definitiva
2    /* Este algoritmo imprime la nota ingresada y si la nota
3       es mayor o igual a 3.0 se imprime un mensaje de
4       "Ganó el curso" o en otro caso "Perdió el curso"
5     */
6
7    // Declaración de variables
8    Real notaDefinitiva
9
```

```
10    // Dato disponible
11    imprimir( "Ingrese la nota definitiva: " )
12    leer( notaDefinitiva )
13
14    // Resultados esperados
15    imprimir( "Su definitiva es: ", notaDefinitiva )
16
17    Si( notaDefinitiva >= 3.0 ) Entonces
18       imprimir( " Ganó el curso" )
19    SiNo
20       imprimir( " Perdió el curso" )
21    FinSi
22
23 FinAlgoritmo
```

Al ejecutar el algoritmo:

Primera ejecución

```
Ingrese la nota definitiva: 2.5
Su definitiva es: 2.5 Perdió el curso
```

Segunda ejecución

```
Ingrese la nota definitiva: 4.3
Su definitiva es: 4.3 Ganó el curso
```

Explicación del algoritmo:

La diferencia con el Ejemplo 3.1 se encuentra en la decisión. En este caso se usó una decisión compuesta.

```
17    Si( notaDefinitiva >= 3.0 )
18       imprimir( " Ganó el curso" )
19    SiNo
20       imprimir( " Perdió el curso" )
21    FinSi
```

Una vez que se evalúa la expresión `notaDefinitiva >= 3.0`, su valor de verdad determinará si se imprime el mensaje “Ganó el curso” o “Perdió el curso”, observe que en ningún caso aparecerán ambos mensajes, es decir, el algoritmo, una vez ejecuta la parte verdadera o falsa de la decisión, ignora la otra parte de la decisión. Para ilustrar mejor esto, observe la Figura 3.6 que presenta el respectivo diagrama de flujo.

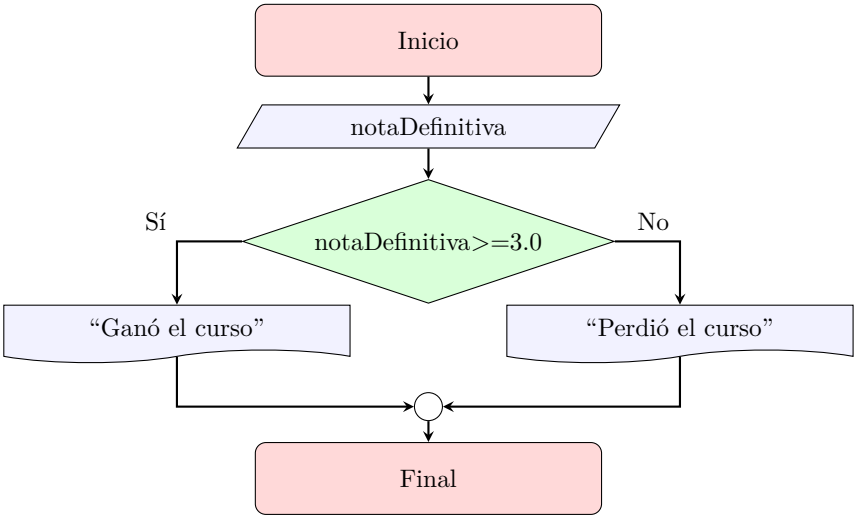


Figura 3.6: Diagrama de flujo del Algoritmo `Definitiva`

.:Ejemplo 3.3. *Diseñe un algoritmo que permita solicitar tanto el nombre como la edad de una persona y posteriormente indicar si ella es “Mayor de edad” o “Menor de edad” según la información ingresada. Para el ejemplo, una persona se considera mayor de edad si tiene 18 años o más.*

Análisis del problema:

- **Resultados esperados:** Un mensaje según la edad de usuario que indique ‘*Mayor de edad*” o “*Menor de edad*” según sea el caso.
- **Datos disponibles:** El nombre de la persona y su edad.
- **Proceso:** se le solicita al usuario que ingrese tanto el nombre como la edad, luego se determina cuál de los dos mensajes debe “*Mayor de edad*” o “*Menor de edad*” dependiendo si la persona tiene una edad mayor o igual a 18 años. (Ver el árbol de decisión, Figura 3.7).

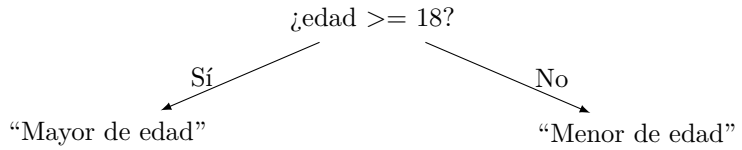


Figura 3.7: Árbol de decisión del Ejemplo 3.3

- **Variables requeridas:**
  - `nombre`: nombre de la persona.
  - `edad`: edad de la persona.

.

De acuerdo al análisis planteado, se propone el Algoritmo 3.5.

Algoritmo 3.5: MayoriaEdad

```
1  Algoritmo MayoriaEdad
2    // Este algoritmo determina si una persona es mayor o
       menor de edad
3
4    // Declaración de variables
5    Cadena nombre, mensajeEdad
6    Entero edad
7
8    // Datos disponibles
9    imprimir( "Ingrese su nombre: " )
10   leer( nombre )
11
12   imprimir( "Ingrese su edad: " )
13   leer ( edad )
14
15   // Resultados esperados
16   Si( edad >= 18 ) Entonces
17      mensajeEdad = "Mayor de edad"
18   SiNo
19      mensajeEdad = "Menor de edad"
20   FinSi
21
22   imprimir  ( "Hola ", nombre )
23   imprimir  (" usted es ", mensajeEdad )
24 FinAlgoritmo
```

Al ejecutar el algoritmo:
Primera ejecución

```
Ingrese su nombre: Juan Pablo
Ingrese su edad: 13
Hola Juan Pablo usted es Menor de edad
```

Segunda ejecución

```
Ingrese su nombre: Carmen Cilia
Ingrese su edad: 56
Hola Carmen Cilia usted es Mayor de edad
```

Explicación del algoritmo:

El Algoritmo 3.5 presenta una forma de implementar la solución a este problema. En la primera parte se declaran las variables a utilizar:

```
5   Cadena nombre, mensajeEdad
6   Entero edad
```

Luego se le solicita al usuario los datos disponibles, es decir, tanto el nombre como la edad.

```
9   imprimir( "Ingrese su nombre: " )
10  leer( nombre )
11
12  imprimir( "Ingrese su edad: " )
13  leer( edad )
```

Posteriormente y usando el valor de verdad de la expresión relacional (`edad >= 18`), se determina el respectivo mensaje, usando una decisión compuesta.

```
16    Si( edad >= 18 ) Entonces
17      mensajeEdad = "Mayor de edad"
18    SiNo
19      mensajeEdad = "Menor de edad"
20    FinSi
```

Finalmente se procede a imprimir el mensaje correspondiente

```
22    imprimir  ( "Hola ", nombre )
23    imprimir  (" usted es ", mensajeEdad )
```

Observe como en el algoritmo solo existe una única instrucción para imprimir el mensaje correspondiente, sin embargo, otra forma válida sería eliminar la variable y hacer la impresión dentro de la decisión.

```
      Si( edad >= 18 ) Entonces
        imprimir( "Hola ", nombre, " usted es Mayor de edad" )
      SiNo
        imprimir( "Hola ", nombre, " usted es Menor de edad" )
      FinSi
```

O hacer la primera parte de la impresión antes de la decisión (el mensaje es común para la parte verdadera y falsa) y luego por medio de la decisión se determina el mensaje complementario. En este caso, tampoco es necesaria la variable. De nuevo se deja en evidencia que un problema no necesariamente tiene una única solución.

```
imprimir( "Hola ", nombre, " usted es ")
Si( edad >= 18 ) Entonces
  imprimir( "Mayor de edad" )
SiNo
  imprimir( "Menor de edad" )
FinSi
```

Otra forma de escribir la estructura de decisión sería invertir la condición, es decir `edad <18`; esto implicaría invertir los mensajes, pero la estructura resolvería de igual manera el problema.

```
imprimir( "Hola ", nombre, " usted es ")
Si( edad < 18 ) Entonces
  imprimir( "Menor de edad" )
SiNo
  imprimir( "Mayor de edad" )
FinSi
```

En la Figura 3.8 se muestra la solución original del Ejemplo 3.3 mediante un diagrama de flujo.

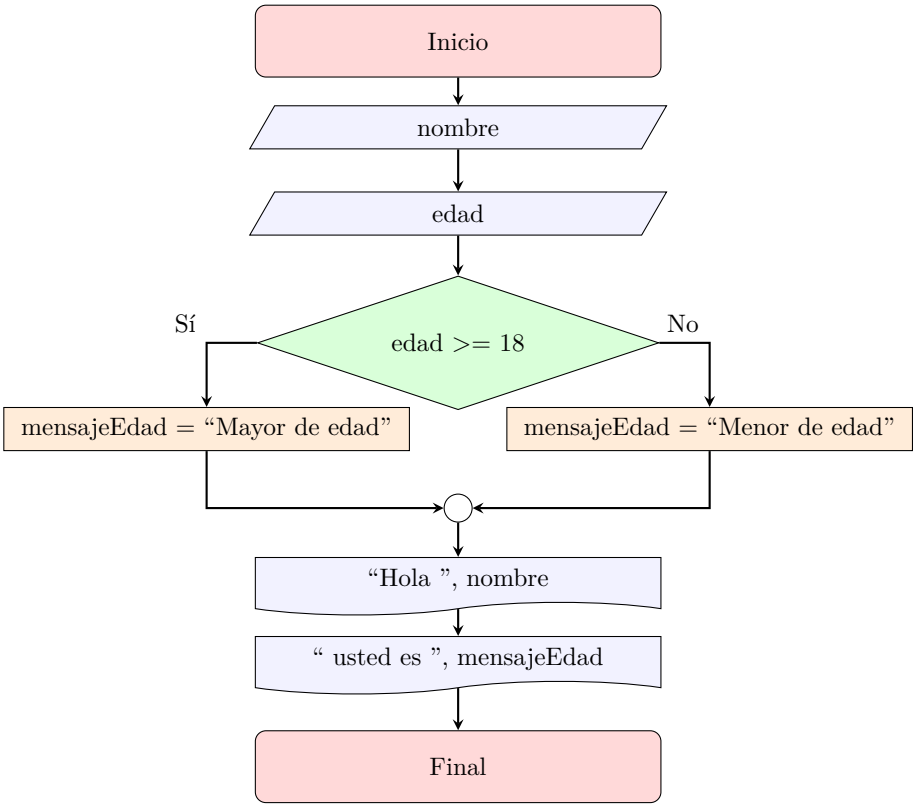


Figura 3.8: Diagrama de flujo del Algoritmo `MayoriaEdad`

.:Ejemplo 3.4. *Diseñe un algoritmo que permita determinar el mayor de dos números enteros.*

Análisis del problema:

- **Resultados esperados:** el mayor de dos números ingresados por el usuario.
- **Datos disponibles:** los dos números enteros
- **Proceso:** primero se le solicita al usuario los dos números enteros, luego por medio de una decisión se determina el mayor de ellos, finalmente se imprime el mayor. (Ver el árbol de decisión, Figura 3.9).

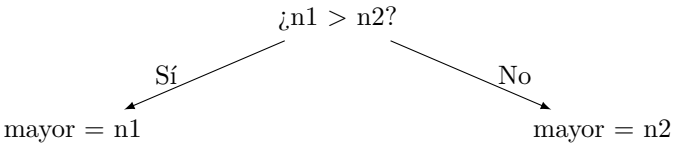


Figura 3.9: Árbol de decisión del Ejemplo 3.4

- **Variables requeridas:**
  - `n1`: primero número entero ingresado por el usuario.
  - `n2`: segundo número entero ingresado por el usuario.
  - `mayor`: mayor de los dos números.

De acuerdo al análisis planteado, se propone el Algoritmo 3.6.

Algoritmo 3.6: MayorNumero

```
1  Algoritmo MayorNumero
2    // Este algoritmo determina el mayor de dos números
3
4    // Declaración de variables
5    Entero n1, n2, mayor
6
7    // Datos disponibles
8    imprimir ( "Ingrese el primer número: " )
9    leer ( n1 )
10
11   imprimir ( "Ingrese el segundo número: " )
12   leer ( n2 )
13
14   // Resultados esperados
```

```
15   Si  n1 > n2  Entonces
16     mayor = n1
17   SiNo
18     mayor = n2
19   FinSi
20
21   imprimir( "El mayor entre ", n1, " y ", n2 )
22   imprimir( " es ", mayor )
23 FinAlgoritmo
```

Al ejecutar el algoritmo:
Primera ejecución

```
Ingrese el primer número: 4
Ingrese el segundo número: 56
El mayor número entre 4 y 56 es 56
```

Segunda ejecución

```
Ingrese el primer número: 78
Ingrese el segundo número: 12
El mayor número entre 78 y 12 es 78
```

Explicación del algoritmo:

Primero se declaran las variables requeridas:

```
5   Entero n1, n2, mayor
```

Posteriormente se le solicita al usuario que ingrese ambos números:

```
8   imprimir ( "Ingrese el primer número: " )
9   leer ( n1 )
10
11  imprimir ( "Ingrese el segundo número: " )
12  leer ( n2 )
```

Después por medio de una decisión se determina cuál es el mayor entre `n1` y `n2`, el mayor de ellos se almacena en la variable `mayor`.

```
15   Si  n1 > n2  Entonces
16     mayor = n1
17   SiNo
18     mayor = n2
19   FinSi
```

Observe que si el valor de ambas variables es igual, el algoritmo indicaría como mayor el valor de `n2`, no obstante, es un resultado tan válido como decir que el mayor es el valor de la variable `n1`, debido a que ambas variables tiene el mismo valor.

Finalmente se imprime la información solicitada, complementando el resultado con los datos suministrados por el usuario.

```
21   imprimir( "El mayor entre ", n1, " y ", n2 )
22   imprimir( " es ", mayor )
```

En la Figura 3.10 se muestra la solución del Ejemplo 3.4 mediante un diagrama de flujo.

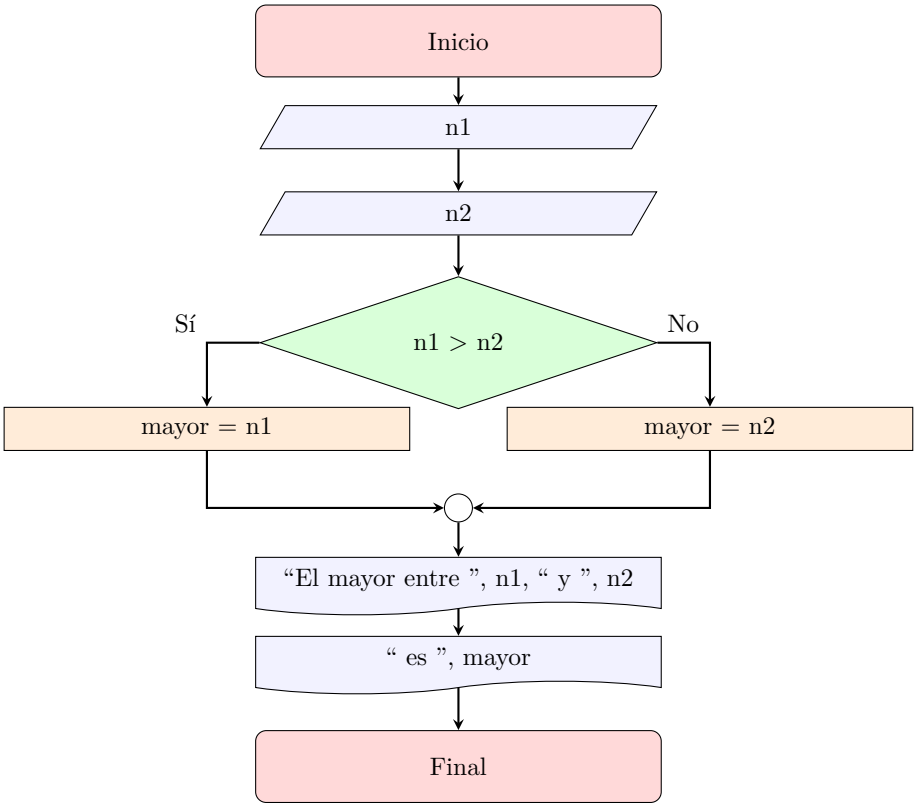


Figura 3.10: Diagrama de flujo del Algoritmo `MayorNumero`

.:Ejemplo 3.5. *Diseñe un algoritmo que determina si un número es par o impar. Recuerde que un número es par si el resto de una división entera con el número 2 es cero.*

Análisis del problema:

- **Resultados esperados:** un mensaje que indica que un número es par o impar.
- **Datos disponibles:** un número entero ingresado por el usuario.

- **Proceso:** primero se le solicita al usuario el número entero, luego por medio de una decisión se determina si es par o no usando el resto de la división entera con el número 2. (Ver el árbol de decisión, Figura 3.11).

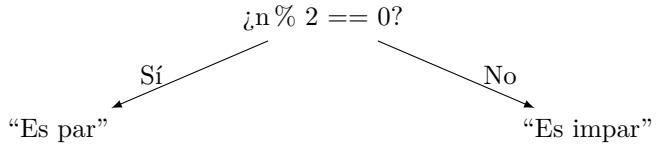


Figura 3.11: Árbol de decisión del Ejemplo 3.5

- **Variables requeridas:**
  - n: número entero que el usuario ingresa

De acuerdo al análisis planteado, se propone el Algoritmo 3.7.

Algoritmo 3.7: ParImpar

```
1  Algoritmo ParImpar
2    // Este algoritmo determina si un número es par o impar
3
4    // Declaración de la variable
5    Entero n
6
7    // Datos disponibles
8    imprimir ( "Ingrese un número: " )
9    leer ( n )
10
11   // Resultados esperados
12   Si( n % 2 == 0 )  Entonces
13     imprimir( "Es par" )
14   SiNo
15     imprimir( "Es impar" )
16   FinSi
17 FinAlgoritmo
```

Al ejecutar el algoritmo:

Primera ejecución

```
Ingrese un número: 42
Es par
```

Segunda ejecución

```
Ingrese un número: 73
Es impar
```

Explicación del algoritmo:

Lo primero es declarar la variable requerida (línea 5), luego se procede a solicitarle al usuario los datos disponibles, en este caso (n).

```
8   imprimir ( "Ingrese un número: " )
9   leer ( n )
```

Posteriormente se procede a determinar si es par o impar usando una decisión y la operación para determinar el resto de la división entera con 2.

```
12  Si( n % 2 == 0 )  Entonces
13    imprimir( "Es par" )
14  SiNo
15    imprimir( "Es impar" )
16  FinSi
```

Observe que también es posible invertir la parte verdadera y la parte falsa de la decisión al negar la comparación del valor del resto con cero.

```
12    Si( n % 2 != 0 )  Entonces
13      imprimir( "Es impar" )
14    SiNo
15      imprimir( "Es par" )
16    FinSi
```

En la Figura 3.12 se muestra la solución del Ejemplo 3.5 mediante un diagrama de flujo.

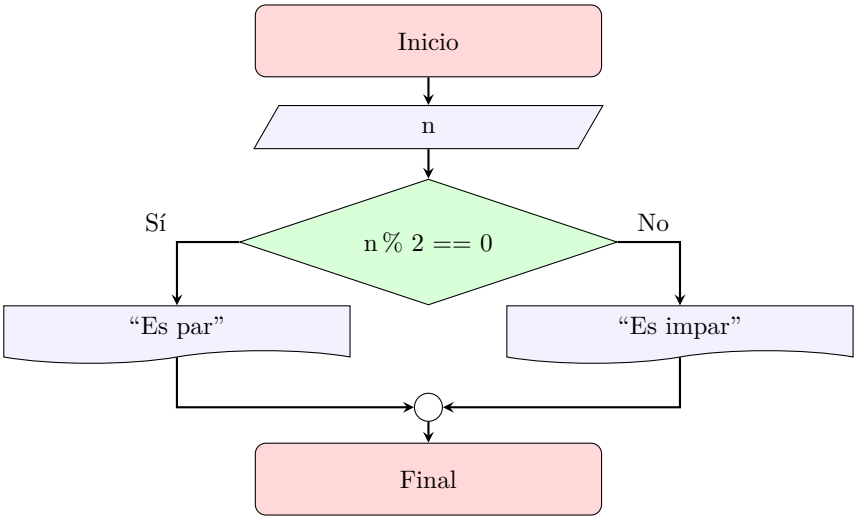


Figura 3.12: Diagrama de flujo del Algoritmo `ParImpar`

.:Ejemplo 3.6. *Diseñe un algoritmo que determine si un número real (x) se encuentra dentro del rango abierto-cerrado (3.5, 7.8].*

Recuerde que se considera abierto o cerrado, si el valor de límite en cuestión se encuentra o no incluido respectivamente. Para el ejemplo, el límite 3.5 no hace parte del rango, por ser parte del intervalo abierto, y se reconoce por tener paréntesis '('; mientras que por el contrario, el valor de 7.8 sí hace parte al ser el intervalo cerrado, y se reconoce por el corchete ']'. Es bueno tener en cuenta que intervalo puede ser abierto o cerrado en cualquiera de sus dos extremos.

Análisis del problema:

- **Resultados esperados:** indicar si un número real (x) se encuentra dentro del rango abierto-cerrado (3.5, 7.8].
- **Datos disponibles:** un único valor real.
- **Proceso:** primero se le solicita al usuario el valor de x, luego se determina si el valor se encuentra dentro del rango (Figura 3.13) o de forma análoga, se determina si está por fuera (Figura 3.14).

  Para el ejemplo se empleará la decisión ilustrada en el árbol de la Figura 3.13, sin embargo, sería totalmente válido usar la decisión ilustrada en la Figura 3.14.

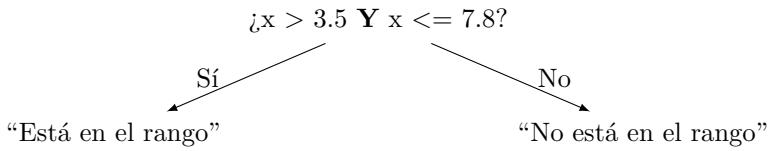


Figura 3.13: Árbol de decisión del Ejemplo 3.6 - Solución 1

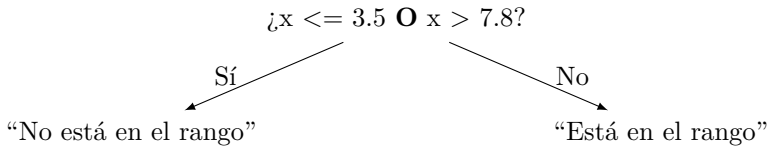


Figura 3.14: Árbol de decisión del Ejemplo 3.6 - Solución 2

Aclaración:



Existe una ley de operadores lógicos[a] que básicamente determina que la negación de una expresión lógica con el operador lógico **Y**, es equivalente a negar cada operador relacional[b] y cambiar el operador lógico por **O**. La ley también se aplica en sentido contrario.

Por ejemplo:

No (x > 3.5 **Y** x <= 7.8) equivale a (x <= 3.5 **O** x > 7.8)
No (x <= 3.5 **O** x > 7.8) equivale a (x > 3.5 **Y** x <= 7.8)

[a] Ley de De Morgan
[b] Lo contrario de: > es <=, < es >=, = es ! = y ! = es ==

- **Variables requeridas:**
  - x: valor ingresado por el usuario.

De acuerdo al análisis planteado, se propone el Algoritmo 3.8.

Algoritmo 3.8: Rango

```
1  Algoritmo Rango
2    /* Este algoritmo determina si un número real (x) está en
3       el rango abierto-cerrado (3.5, 7.8]
4    */
5
6    // Declaración de la variable
7    Real x
8
9    // Datos disponibles
10   imprimir ( "Ingrese un número: " )
11   leer ( x )
12
13   // Resultados esperados
14   Si( x > 3.5 Y x <= 7.8 ) Entonces
15     imprimir( "Está en el rango" )
16   SiNo
17     imprimir( "No está en el rango" )
18   FinSi
19 FinAlgoritmo
```

Al ejecutar el algoritmo:
Primera ejecución

```
Ingrese un número: 9.3
No está en el rango
```

Segunda ejecución

```
Ingrese un número: 2.8
No está en el rango
```

Tercera ejecución

```
Ingrese un número: 4.1
Está en el rango
```

Explicación del algoritmo:

Luego de declarar la variable x (línea 7) y solicitar su valor al usuario (líneas 10 y 11), se procede a emplear la respectiva decisión para determinar la instrucción de impresión a utilizar.

```
14  Si( x > 3.5 Y x <= 7.8 ) Entonces
15    imprimir( "Está en el rango" )
16  SiNo
17    imprimir( "No está en el rango" )
18  FinSi
```

Pero como ya fue mencionado, también es posible invertir los textos a imprimir al negar la expresión lógica

```
14  Si( No ( x > 3.5 Y x <= 7.8 ) ) Entonces
15    imprimir( "No está en el rango" )
16  SiNo
17    imprimir( "Está en el rango" )
18  FinSi
```

Y al aplicar la ley de De Morgan, también se puede usar:

```
14  Si( x <= 3.5 O x > 7.8 ) ) Entonces
15    imprimir( "No está en el rango" )
16  SiNo
17    imprimir( "Está en el rango" )
18  FinSi
```

En la Figura 3.15 se muestra la solución del Ejemplo 3.6 mediante un diagrama de flujo.

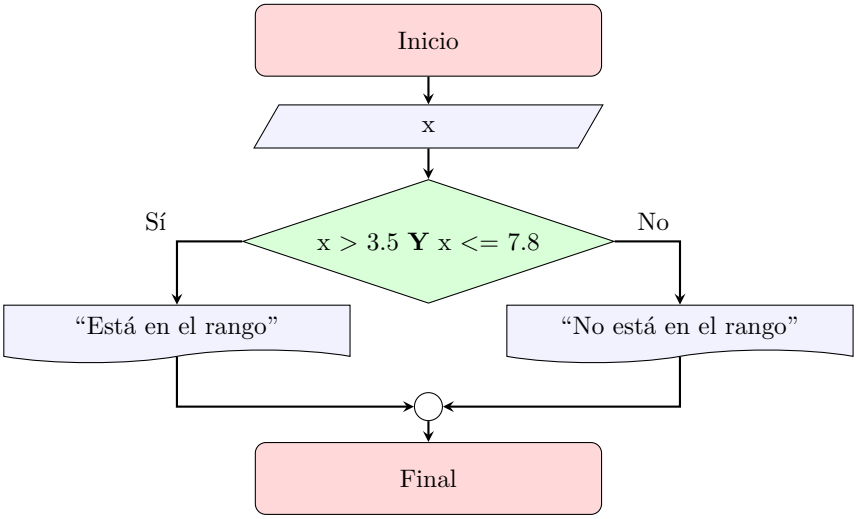


Figura 3.15: Diagrama de flujo del Algoritmo `Rango`

.:Ejemplo 3.7. *Diseñe un algoritmo que determine si un número real (x) se encuentra dentro de uno de los siguientes rangos: (3.5, 7.8], [9.3, 4.5) y [23.4, 45.3].*

Análisis del problema:

- **Resultados esperados:** indicar si un número real (`x`) se encuentra dentro de uno de los siguientes rangos: (3.5, 7.8], [9.3, 4.5) y [23.4, 45.3].
- **Datos disponibles:** un único valor real.
- **Proceso:** primero se le solicita al usuario el valor de `x`, luego se determina si el valor se encuentra dentro de uno de los rangos (Figura 3.16).

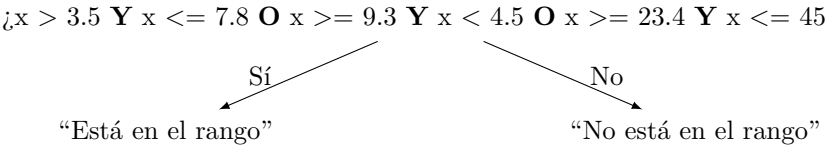


Figura 3.16: Árbol de decisión del Ejemplo 3.7

- **Variables requeridas:**
  - x: valor ingresado por el usuario.

De acuerdo al análisis planteado, se propone el Algoritmo 3.9.

Algoritmo 3.9: Rangos

```
1  Algoritmo Rangos
2    /* Este algoritmo determina si un número real (x) está en
3       el rango abierto-cerrado (3.5, 7.8]
4    */
5
6    // Declaración de la variable
7    Real x
8
9    // Datos disponibles
10   imprimir ( "Ingrese un número: " )
11   leer ( x )
12
13   // Resultados esperados
14   Si( x >  3.5  Y x <= 7.8  O
15       x >= 9.3  Y x <  4.5  O
16       x >= 23.4 Y x <= 45.3 ) Entonces
17     imprimir( "Está en el rango" )
18   SiNo
19     imprimir( "No está en el rango" )
20   FinSi
21 FinAlgoritmo
```

Explicación del algoritmo:

Observa que la expresión lógica de la decisión no requiere paréntesis para agrupar las expresiones relacionales, debido a que el operador **Y** tiene mayor precedencia que el operador **O**. No obstante por legibilidad[1] es posible adicionar los paréntesis.

```
14   Si( (x >  3.5  Y x <= 7.8)  O
15       (x >= 9.3  Y x <  4.5)  O
16       (x >= 23.4 Y x <= 45.3) ) Entonces
17     imprimir( "Está en el rango" )
18   SiNo
19     imprimir( "No está en el rango" )
20   FinSi
```

El diagrama de flujo del Ejemplo 3.7 es similar al de la Figura 3.15, solo que, en este se aumenta la expresión lógica que hay en la condición, según lo expresado anteriormente.

[1] Facilitar la comprensión

3.2. Decisiones anidadas

Las estructuras de decisión se pueden usar cuantas veces sean necesarias en un algoritmo, incluso dentro de otra estructura de decisión (ver diagrama de flujo de la página 56). Para ilustrar este caso y continuando con el ejemplo introductorio de la sección "Decisiones compuesta" (página 122), usted sigue pensando y concluye que:

Si ¿hace calor? **Entonces**
- Vestir ropa deportiva
- Ir al parque

Sino
- Vestir informalmente
- Ir al cine
- **Si** ¿tengo compañía? **Entonces**
  - Usar el carro

  Sino
  - Usar transporte público

En el diagrama de flujo de la Figura 3.17 se puede evidenciar de una manera más gráfica que la decisión "*¿tengo compañía?*" depende si en el día hace calor o no. En otras palabras, si decide ir al parque, no tiene que considerar el hecho de tener o no compañía para determinar el tipo de transporte a utilizar.

Pero y si se desea decidir el tipo de transporte independientemente del lugar a visitar, es necesario que la segunda decisión, no dependa de la primera, es decir, son dos decisiones simples y no una anidada, tal y como se aprecia a continuación y se visualiza en la Figura 3.18.

Si ¿hace calor? **Entonces**
- Vestir ropa deportiva
- Ir al parque

Sino
- Vestir informalmente
- Ir al cine

Si ¿tengo compañía? **Entonces**
- Usar el carro

Sino
- Usar transporte público

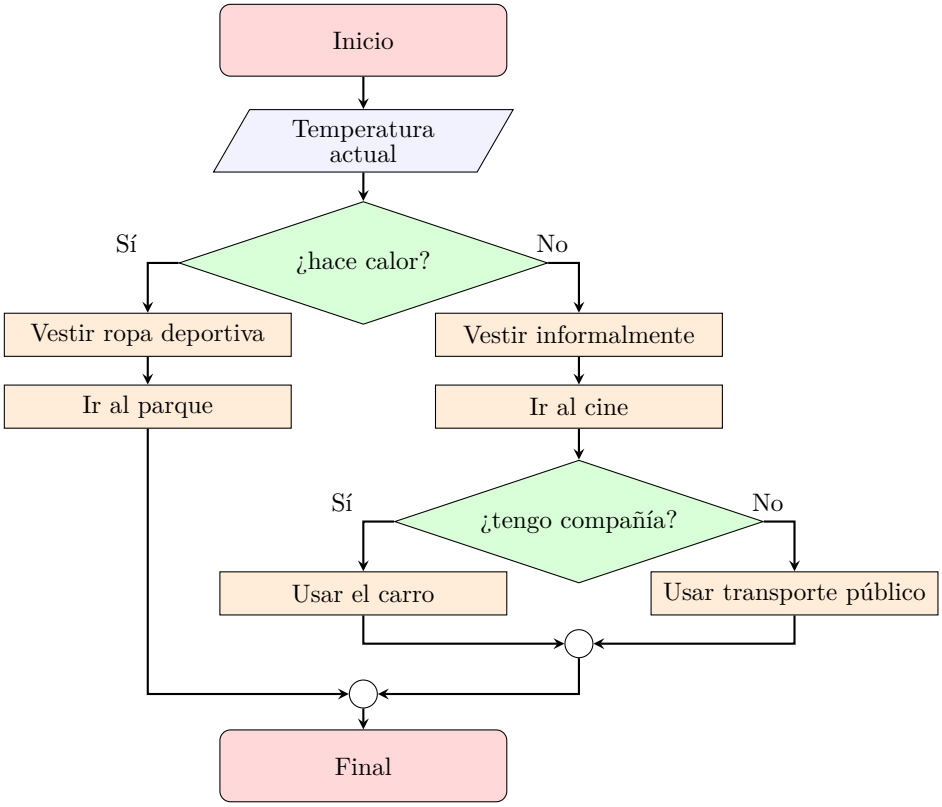


Figura 3.17: Planeando el domingo - Versión 2

Aclaración:



Las decisiones secuenciales y anidadas, en realidad no son dos conceptos diferentes, sino, que son los nombres que reciben dos formas de usar las decisiones y que dependen directamente de la solución que se está construyendo.

Se dice entonces que hay una decisión anidada, cuando existe otra decisión en la parte verdadera (**Sí**) o en la parte falsa (**No**) de una decisión; en otro caso son decisiones independientes o secuenciales.

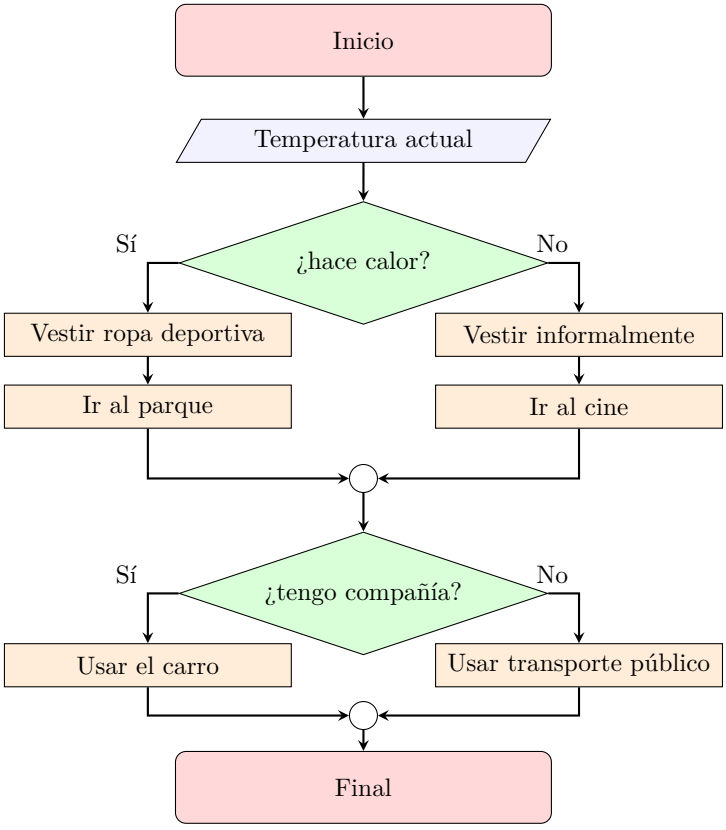


Figura 3.18: Planeando el domingo - Versión 3

.:Ejemplo 3.8. *Diseñe un algoritmo que permita imprimir un mensaje según un carácter dado por el usuario, independiente que sea ingresado en mayúscula o minúscula, según la Tabla 3.1.*

Caracter	**Mensaje a imprimir**
'a'	"Android"
'i'	"iOS"
otro	"Opción inválida"

Tabla 3.1: Opciones para el Ejemplo 3.8

Análisis del problema:

- **Resultados esperados:** el mensaje acorde al carácter dado por la Tabla 3.1.
- **Datos disponibles:** el carácter ingresado por el usuario.

- **Proceso:** solicitar al usuario el carácter, luego determinar el mensaje según el árbol de decisión de la Figura 3.19.

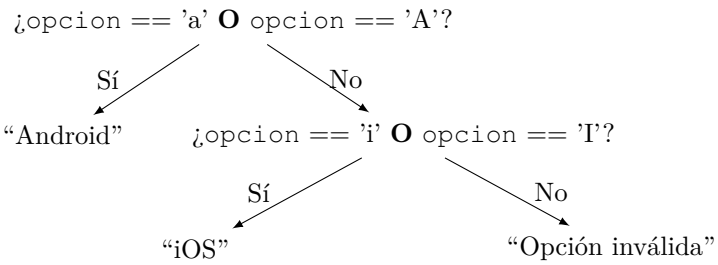


Figura 3.19: Árbol de decisión del Ejemplo 3.8

- **Variables requeridas:**
  - `opcion`: carácter que el usuario ingresa

De acuerdo al análisis planteado, se propone el Algoritmo 3.10.

Algoritmo 3.10: SistamaOperativo

```
1  Algoritmo SistamaOperativo
2    // Este algoritmo determina un mensaje según un carácter
3
4    // Declaración de la variable
5    Caracter opcion
6
7    // Datos disponibles
8    imprimir( "Ingrese un carácter: " )
9    leer( opcion )
10
11   // Resultados esperados
12   imprimir( "Su opción es " )
13   Si( opcion == 'a' O opcion == 'A' ) Entonces
14     imprimir( "Android" )
15   SiNo
16     Si( opcion == 'i' O opcion == 'I' ) Entonces
17          imprimir( "iOS" )
18        SiNo
19          imprimir( "inválida" )
20     FinSi
21   FinSi
22 FinAlgoritmo
```

Al ejecutar el algoritmo:
Primera ejecución

```
Ingrese un carácter: i
Su opción es iOS
```

Segunda ejecución

```
Ingrese un carácter: A
Su opción es Android
```

Tercera ejecución

```
Ingrese un carácter: x
Su opción es inválida
```

Explicación del algoritmo:

Lo primero es declarar la variable necesaria (línea 5), luego se solicita al usuario un carácter.

```
8     imprimir( "Ingrese un carácter: " )
9     leer( opcion )
```

Luego por medio de una decisión se determina si el carácter ingresado es la letra ('a' **O** 'A') para imprimir "Android"; si por el contrario, el carácter ingresado es otro, es necesario determinar si corresponde a la letra ('i' **O** 'I') para imprimir "iOS"; para cualquier otro carácter se imprime "inválida".

```
13    Si( opcion == 'a' O opcion == 'A' ) Entonces
14      imprimir( "Android" )
15    SiNo
16      Si( opcion == 'i' O opcion == 'I' ) Entonces
17        imprimir( "iOS" )
18      SiNo
19        imprimir( "inválida" )
20      FinSi
21    FinSi
```

En la Figura 3.20 se muestra la solución del Ejemplo 3.8 mediante un diagrama de flujo, simplificado al omitir el operador lógico **O** para aceptar la opción en mayúsculas.

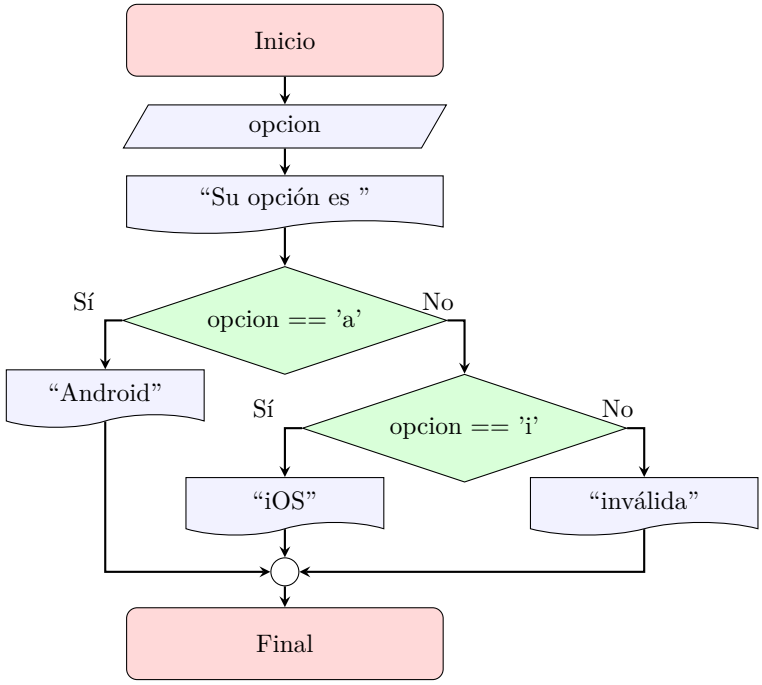


Figura 3.20: Diagrama de flujo del Algoritmo `SistamaOperativo`

.:Ejemplo 3.9. *Diseñe un algoritmo que permita imprimir un mensaje según la nota definitiva de un estudiante entre 0.0 y 5.0, de acuerdo con la Tabla 3.2.*

nota	**Mensaje a imprimir**
< 3.0	“Insuficiente”
<= 3.5	“Aceptable”
<= 4.0	“Sobresaliente”
<= 5.0	“Excelente”

Tabla 3.2: Opciones para el Ejemplo 3.9

Análisis del problema:

- **Resultados esperados:** el mensaje acorde a la nota definitiva según la Tabla 3.2.
- **Datos disponibles:** la nota definitiva ingresada por el usuario.

- **Proceso:** solicitar al usuario la nota definitiva, luego determinar el mensaje según el árbol de decisión de la Figura 3.21.

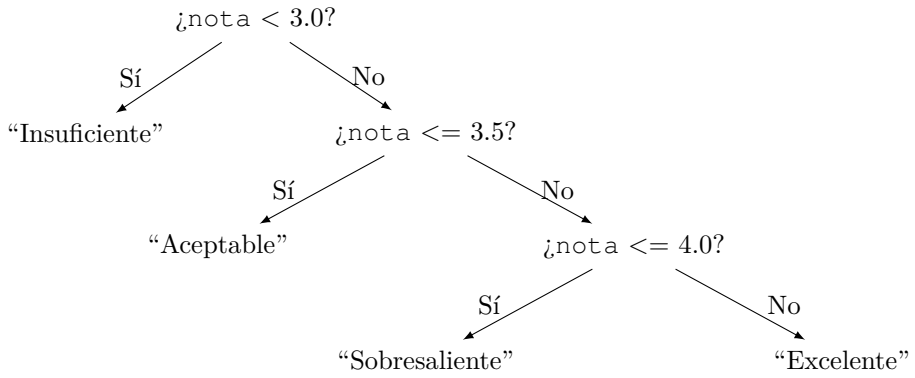


Figura 3.21: Árbol de decisión del Ejemplo 3.8

Observe que en el árbol de decisión no es necesario preguntar si la nota es menor o igual 5.0, debido a que con las otras condiciones, todas las demás posibilidades ya están cubiertas, quedando solo por fuera el rango de una nota excelente (según el ejercicio), es decir, una nota definitiva mayor a 4.0 y menor o igual a 5.0.

- **Variables requeridas:**
  - `nota`: definitiva del estudiante que el usuario ingresa

De acuerdo al análisis planteado, se propone el Algoritmo 3.11.

Algoritmo 3.11: MensajeNota

```
1  Algoritmo MensajeNota
2     // Este algoritmo determina un mensaje según una nota
3
4     // Declaración de la variable
5     Real nota
6
7     // Datos disponibles
8     imprimir( "Ingrese la nota definitiva: " )
9     leer( nota )
10
11    // Resultados esperados
12    imprimir( "Su nota es " )
```

```
13      Si( nota < 3.0 ) Entonces
14        imprimir( "Insuficiente" )
15      SiNo
16        Si( nota <= 3.5 ) Entonces
17          imprimir( "Aceptable" )
18        SiNo
19          Si( nota <= 4.0 ) Entonces
20             imprimir( "Sobresaliente" )
21          SiNo
22             imprimir( "Excelente" )
23          FinSi
24        FinSi
25      FinSi
26    FinAlgoritmo
```

Al ejecutar el algoritmo:
Primera ejecución

```
Ingrese la nota definitiva: 4.7
Su nota es Excelente
```

Segunda ejecución

```
Ingrese la nota definitiva: 3.8
Su nota es Sobresaliente
```

Tercera ejecución

```
Ingrese la nota definitiva: 3.2
Su nota es Aceptable
```

Cuarta ejecución

```
Ingrese la nota definitiva: 1.3
Su nota es Insuficiente
```

Explicación del algoritmo:

Lo primero que se realiza es la declaración y lectura de la única variable requerida (líneas de la 5 a la 9).

Luego implementa el árbol de decisión de la Figura 3.21 mediante las decisiones:

```
13      Si( nota < 3.0 ) Entonces
14        imprimir( "Insuficiente" )
15      SiNo
16        Si( nota <= 3.5 ) Entonces
17          imprimir( "Aceptable" )
18        SiNo
19          Si( nota <= 4.0 ) Entonces
20            imprimir( "Sobresaliente" )
21          SiNo
22            imprimir( "Excelente" )
23          FinSi
24        FinSi
25      FinSi
```

Es importante resaltar que se asume que la nota es siempre ingresada en el rango válido, pero de no hacerlo, el algoritmo imprimirá "`Insuficiente`" para una nota menor a cero y "`Excelente`" para notas mayores a 4.0. En la Figura 3.22 se muestra la solución del Ejemplo 3.9 mediante un diagrama de flujo.

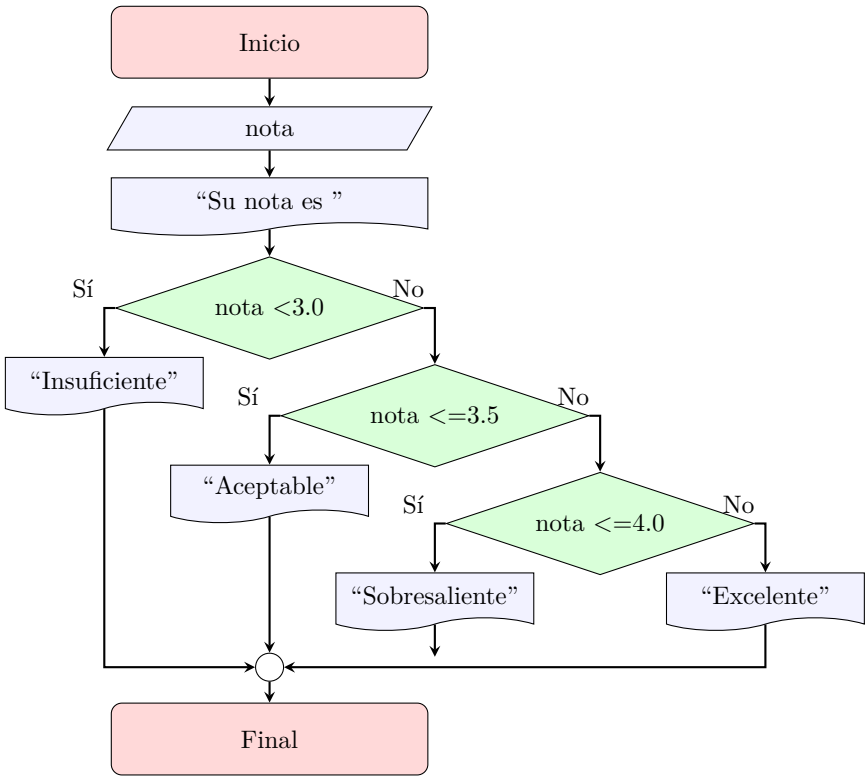


Figura 3.22: Diagrama de flujo del Algoritmo `MensajeNota`

.:Ejemplo 3.10. *Diseñe un algoritmo que determine mayor número entre cuatro posibles números*

Análisis del problema:

- **Resultados esperados:** el mayor de cuatro números ingresados por el usuario.
- **Datos disponibles:** los cuatro números (n_1, n_2, n_3 y n_4).
- **Proceso:** solicitar al usuario los cuatro números, luego por medio de decisiones (Ver árbol de decisión de las Figura 3.23 o 3.24) determinar el mayor de ellos y finalmente imprimir el resultado. Los elementos al final de cada rama del árbol (hojas) son los valores mayores de cada uno de los cuatro valores.

  - En una primera versión del árbol de decisión se puede construir sin el uso de operadores lógicos, el resultado se aprecia en la Figura 3.23.

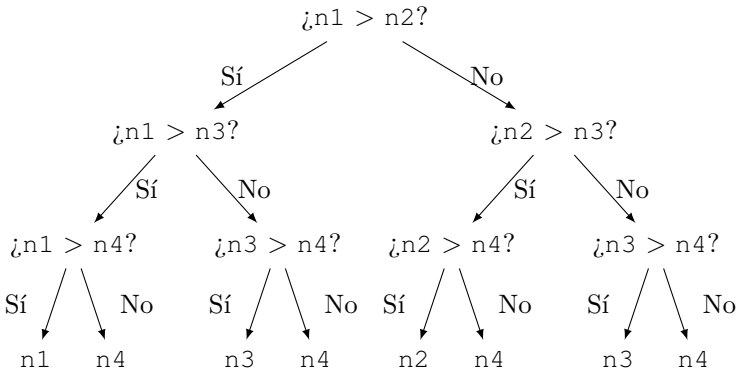


Figura 3.23: Árbol de decisión del Ejemplo 3.10 - Versión 1

  - En esta segunda versión se hace el uso del operador lógico **Y**.

Aunque a primera vista el árbol de la Figura 3.24 es más pequeño que el de la Figura 3.23, desde el punto de vista algorítmico tiene un problema, en algunas circunstancias realiza más comparaciones que el primero[2]; por ejemplo con los valores: $n1 = 4$, $n2 = 3$, $n3 = 2$ y $n4 = 10$, se realizan cuatro comparaciones, en el mejor de los casos, lo cual lo hace para ciertos casos más ineficiente.

[2]Siempre realiza tres comparaciones para llegar a un resultado.

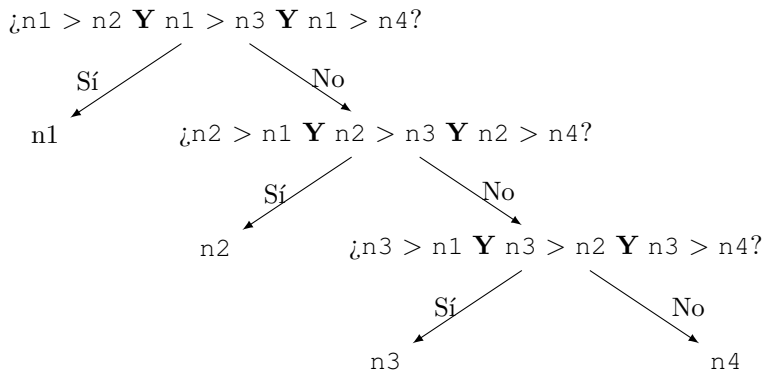


Figura 3.24: Árbol de decisión del Ejemplo 3.10 - Versión 2

Aclaración:



Muchos lenguajes de programación, al momento de evaluar una expresión lógica hacen optimizaciones para ganar velocidad, entre ellas están:

- Si una expresión lógica tiene el operador **Y** y alguna de las comparaciones da falso, entonces, se ignoran las demás, ya que sin importar el resultado ellas, el valor final siempre será falso.
- Si una expresión lógica tiene el operador **O** y alguna de las comparaciones da verdadero, entonces, se ignoran las demás, ya que sin importar el resultado ellas, el valor final siempre será verdadero.

- **Variables requeridas:**
  - `n1`: primer número ingresado.
  - `n2`: segundo número ingresado.
  - `n3`: tercer número ingresado.
  - `n4`: cuarto número ingresado.
  - `mayor`: el mayor valor de los cuatro ingresados.

.

De acuerdo al análisis planteado, se propone el Algoritmo 3.12.

Algoritmo 3.12: Mayor4 - Versión 1

```
Algoritmo Mayor4-V1
    // Este algoritmo determina el mayor de cuatro números

    // Declaración de la variable
    Real n1, n2, n3, n4, mayor

    // Datos disponibles
    imprimir( "Ingrese el primer número: " )
    leer( n1 )

    imprimir( "Ingrese el segundo número: " )
    leer( n2 )

    imprimir( "Ingrese el tercer número: " )
    leer( n3 )

    imprimir( "Ingrese el cuarto número: " )
    leer( n4 )

    // Determina el mayor de los cuatro números
    Si( n1 > n2 ) Entonces
       Si( n1 > n3 ) Entonces
          Si( n1 > n4 ) Entonces
             mayor = n1
          SiNo
             mayor = n4
          FinSi
       SiNo
          Si( n3 > n4 ) Entonces
             mayor = n3
          SiNo
             mayor = n4
          FinSi
       FinSi
    SiNo
       Si( n2 > n3 ) Entonces
          Si( n2 > n4 ) Entonces
             mayor = n2
          SiNo
             mayor = n4
          FinSi
        SiNo
          Si( n3 > n4 ) Entonces
             mayor = n3
          SiNo
             mayor = n4
```

```
47            FinSi
48          FinSi
49        FinSi
50
51        // Resultados esperados
52        imprimir( "La número mayor es: ", mayor )
53    FinAlgoritmo
```

Al ejecutar el algoritmo:
Primera ejecución

```
Ingrese el primer número: 27
Ingrese el segundo número: 11
Ingrese el tercer número: 343
Ingrese el cuarto número: 20
El mayor de todos es 343
```

Segunda ejecución

```
Ingrese el primer número: 31
Ingrese el segundo número: 5
Ingrese el tercer número: 73
Ingrese el cuarto número: 19
El mayor de todos es 73
```

Explicación del algoritmo:

En ambos casos, se declaran todas las variables y se solicitan todos los datos conocidos.

```
5       Real n1, n2, n3, n4, mayor
6
7       // Datos disponibles
8       imprimir( "Ingrese el primer número: " )
9       leer( n1 )
10
11      imprimir( "Ingrese el segundo número: " )
12      leer( n2 )
13
14      imprimir( "Ingrese el tercer número: " )
15      leer( n3 )
16
17      imprimir( "Ingrese el cuarto número: " )
18      leer( n4 )
```

Luego usando las respectivas decisiones se determina el mayor número y finalmente se imprime.

Otra posibilidad es reemplazar todas las líneas de la decisión anidada (líneas de la 21 a la 49) por el siguiente código, obteniendo exactamente el mismo resultado.

```
// Determina el mayor de los cuatro números
Si( n1 > n2 Y n1 > n3 Y n1 > n4 ) Entonces
   mayor = n1
SiNo
   Si( n2 > n1 Y n2 > n3 Y n2 > n4 ) Entonces
      mayor = n2
   SiNo
      Si( n3 > n1 Y n3 > n2 Y n3 > n4 ) Entonces
         mayor = n3
      SiNo
         mayor = n4
      FinSi
   FinSi
FinSi
```

Existen otras alternativas para resolver este problema. A continuación se presenta una tercera versión en la que se van descartando aquellos números que no son mayores y que va reduciendo el número de comparaciones a realizar.

```
// Determina el mayor de los cuatro números
Si( n1 > n2 Y n1 > n3 Y n1 > n4 ) Entonces
   mayor = n1
SiNo
   Si( n2 > n3 Y n2 > n4 ) Entonces
      mayor = n2
   SiNo
      Si( n3 > n4 ) Entonces
         mayor = n3
      SiNo
         mayor = n4
      FinSi
   FinSi
FinSi
```

Aclaración:



Este ejemplo de determinar el mayor de cuatro números, le muestra nuevamente al lector que existen diversas maneras de resolver un problema. Es una habilidad a desarrollar en el diseñador del algoritmo, la capacidad de discernir y seleccionar la "mejor" opción entre las posibles soluciones.

Sin embargo, puede observar que en la medida que el número de variables aumenta, el número de posibles alternativas también lo hace, así como la complejidad de los respectivos árboles de decisión.

Otra posibilidad sería resolver el problema mediante el uso de decisiones simples, por tanto, si se reeescribe la parte de las decisiones de la siguiente manera:

```
// Determina el mayor de los cuatro números
mayor = n1
Si( n2 > mayor ) Entonces
   mayor = n2
FinSi

Si( n3 > mayor ) Entonces
   mayor = n3
FinSi

Si( n4 > mayor ) Entonces
   mayor = n4
FinSi
```

La solución que se obtiene es fácilmente ampliable a cualquier cantidad de variables a comparar, sin hacer más complejo el algoritmo y mantiene el número de comparaciones en el ideal.

Este ejemplo ilustra la importancia que tiene realizar un buen análisis del problema antes de comenzar a crear una solución.

.:Ejemplo 3.11. *La oficina de aguas de la ciudad requiere crear un algoritmo que le permita liquidar las facturas de sus clientes durante cada mes. El cobro de cada factura se realiza de la siguiente forma: se cobra el cargo fijo, el consumo de agua en el periodo, es decir, la cantidad de metros consumidos y el servicio de recolección de basuras y alcantarillado. Todos estos cobros se llevan a cabo dependiendo del estrato socioeconómico al que pertenezca el predio, de acuerdo con la siguiente tabla:*

Estrato	Cargo Fijo	Metro^3 consumido	Basuras y alcantarillado
1	*$2500*	*$2200*	*$5500*
2	*$2800*	*$2350*	*$6200*
3	*$3000*	*$2600*	*$7400*
4	*$3300*	*$3400*	*$8600*
5	*$3700*	*$3900*	*$9700*
6	*$4400*	*$4800*	*$11000*

Construya un algoritmo que, al ingresarle es estrato socioeconómico del predio y la cantidad de metros cúbicos de agua consumidos, permita determinar el valor de la factura a pagar.

Análisis del problema:

- **Resultados esperados:** los valores a pagar en la factura de aguas, discriminando el valor del cargo fijo, el valor del consumo, el del servicio de recolección de basura y alcantarillado y que entregue el valor total a pagar.
- **Datos disponibles:** el estrato socioeconómico y la cantidad de metros consumidos.
- **Proceso:** primero será ingresar los datos disponibles, posteriormente, basado en el estrato, será necesario utilizar una estructura de decisión anidada que permita determinar el valor del cargo fijo, el valor del consumo que se obtiene multiplicando la cantidad de metros consumidos por el valor de cada metro dependiendo por supuesto, del estrato y, además, determinar también el valor de la recolección de basuras y alcantarillado. Luego de tener todos estos valores, se deben sumar para encontrar el valor total a pagar. Al finalizar, se muestran al usuario todos estos resultados encontrados.
- **Variables requeridas:**
  - `estrato`: almacena el estrato socioeconómico del predio.
  - `cantidad`: cantidad de metros consumidos por el cliente.
  - `cargoFijo`: valor del cargo fijo.
  - `valorConsumo`: valor del consumo.
  - `valorRecoleccion`: valor de la recolección de las basuras y el alcantarillado.
  - `totalPago`: valor total a pagar por el servicio.

.

De acuerdo al análisis planteado, se propone el Algoritmo 3.13.

Algoritmo 3.13: FacturaAgua

```
1  Algoritmo FacturaAgua
2    // Este algoritmo calcula el pago de una factura de agua
3    Caracter estrato
4    Real cantidad, cargoFijo, consumo, valorRecoleccion,
5         totalPago
6
7    imprimir( "Estrato socioeconómico del predio: " )
8    leer( estrato )
9
10   imprimir( "Cantidad de metros consumidos: " )
11   leer( cantidad )
12
13   Si( estrato == '1' ) Entonces
14     cargoFijo = 2500.0
15     valorConsumo = cantidad * 2200.0
16     valorRecolección = 5500.0
17   SiNo
18     Si( estrato == '2' ) Entonces
19       cargoFijo = 2800.0
20       valorConsumo = cantidad * 2350.0
21       valorRecolección = 6200.0
22     SiNo
23       Si( estrato == '3' ) Entonces
24          cargoFijo = 3000.00
25          valorConsumo = cantidad * 2600.0
26          valorRecolección = 7400.0
27       SiNo
28         Si( estrato == '4' ) Entonces
29           cargoFijo = 3300.00
30           valorConsumo = cantidad * 3400.0
31           valorRecolección = 8600.0
32         SiNo
33           Si( estrato == '5' ) Entonces
34             cargoFijo = 3700.00
35             valorConsumo = cantidad * 3900.0
36             valorRecolección = 9700.00
37           SiNo
38             cargoFijo = 2800.00
39             valorConsumo = cantidad * 2350.0
40             valorRecolección = 6200.0
41           FinSi
42         FinSi
43       FinSi
44     FinSi
45   FinSi
46
```

```
47    totalPago = cargoFijo + valorConsumo + valorRecoleccion
48
49    imprimir( "Valor del cargo fijo: ", cargoFijo )
50    imprimir( "Valor del consumo: ", cantidad )
51    imprimir( "Valor de la recolección: ", valorRecoleccion )
52    imprimir( "Total a pagar: ", totalPago )
53  FinAlgoritmo
```

Explicación del algoritmo: En este algoritmo, las primeras líneas (líneas 2 a 5) se hace ña declaración de las variables, luego (líneas 7 a 11) permiten solicitar los datos que se requieren, esto es estrato y cantidad de metros consumidos; para ello se utilizaron las instrucciones imprimir y leer.

```
2    Caracter estrato
3    Real cantidad, cargoFijo, consumo, valorRecoleccion,
4         totalPago
5
6    imprimir( "Estrato socioeconómico del predio: " )
7    leer( estrato )
8
9    imprimir( "Cantidad de metros consumidos: " )
10   leer( cantidad )
```

A continuación, el algoritmo utiliza una estructura `Si` anidada (líneas 13 a la 45) que va determinando paulatinamente cual es el estrato socioeconómico ingresado con el propósito de aplicar los valores correspondientes a ese estrato. El primer `Si`, pregunta si el `estrato` ingresado es '1', si esto ocurre, se hacen los cálculos indicados para ese estrato pero, si por el contrario, el `estrato` no es '1', dentro del `SiNo` del primer, `Si`, se ubica una segunda estructura `Si` que pregunta si el `estrato` es '2', si esto es verdadero, se realizan los cálculos pertinentes para este segundo estrato pero, si el `estrato` tampoco es '2', por el `SiNo` del segundo `Si`, se utiliza un tercer `Si` para preguntar entonces, si el `estrato` es '3'. De la misma forma, el algoritmo continúa analizando los estratos '4' y '5'. Note que, al finalizar el último `Si`, por la parte del `SiNo`, se ubican los cálculos correspondientes al estrato '6'; esto se hace de esta forma, ya que si el algoritmo no ingresó a ninguna de las partes verdaderas de los `Si` anteriores, esto indica que el estrato no es ni '1', '2', '3', '4', '5', por lo cuál solo queda por descarte, el estrato '6'.

3.3. Decisiones múltiples

Las decisiones múltiples, también conocida como "estructura selectiva", en realidad son una forma abreviada de reescribir un tipo especial de decisión anidada, la utilizada para seleccionar un opción entre un conjunto de valores; por lo anterior, toda decisión múltiple se puede reescribir como una decisión anidada, pero no toda decisión anidada se puede reescribir como una decisión múltiple.

La forma general de esta estructura es presentada en el segmento del Algoritmo 3.14.

En esta forma general se puede observar que existen un conjunto `k` de valores (`valor1` hasta la `valork`).

Cada valor a su vez tiene un conjunto independiente de instrucciones que está delimitado desde la instrucción (`Caso valor:`) hasta `FinCaso`, todas estas instrucciones se ejecutan cuando el contenido de la variable denominada `selector` coincide con el de la valor en cuestión.

Algoritmo 3.14: Forma general de las decisiones múltiples

```
1  Segun( selector )
2    Caso valor1: InstruccionA1-1
3                 InstruccionA1-2
4                 ...
5                 InstruccionA1-n1
6                 FinCaso
7
8    Caso valor2: InstruccionA2-1
9                 InstruccionA2-2
10                ...
11                InstruccionA2-n2
12                FinCaso
13   ...
14
15   Caso valork: InstruccionAk-1
16                InstruccionAk-2
17                ...
18                InstruccionAk-nk
19                FinCaso
20
21   EnOtroCaso:  InstruccionOtroCaso-1
22                InstruccionOtroCaso-2
23                ...
24                InstruccionOtroCaso-n
25                FinCaso
26 FinSegun
```

Aclaración:



Si se omite la instrucción `FinCaso`, el algoritmo continua la ejecución a las instrucciones del siguiente valor y así sucesivamente hasta encontrar un `FinCaso` o un `FinSegun`. Esta característica permite crear ciertos comportamientos deseables en un algoritmo.

Si el contenido de la variable `selector` no incide con ninguna de los valores, se ejecutan las instrucciones de la sección llamada `EnOtroCaso`, la cual se suele escribir después de todas los valores válidos.

El tipo de dato de la variable `selector` suele estar limitada a: `Entero`, `Caracter` o `Cadena`. No se utilizan selectores tipo `Real`.

El equivalente de esta estructura mediante decisiones anidadas se presenta en el Algoritmo 3.15.

Algoritmo 3.15: Forma general del `Segun` mediante `Si`

```
1  Si( selector == valor1 ) Entonces
2    InstruccionA1-1
3    InstruccionA1-2
4    ...
5    InstruccionA1-n1
6  SiNo
7    Si( selector == valor2 ) Entonces
8      InstruccionA2-1
9      InstruccionA2-2
10     ...
11     InstruccionA2-n2
12   SiNo
13     ...
14
15     Si( selector == valork ) Entonces
16       InstruccionAk-1
17       InstruccionAk-2
18       ...
19       InstruccionAk-nk
20     SiNo
21       InstruccionOtroCaso-1
22       InstruccionOtroCaso-2
23       ...
24       InstruccionOtroCaso-n
25     FinSi
26   FinSi
27 FinSi
```

La estructura general de una decisión múltiple en un diagrama de flujo se puede observar en la Figura 3.25.

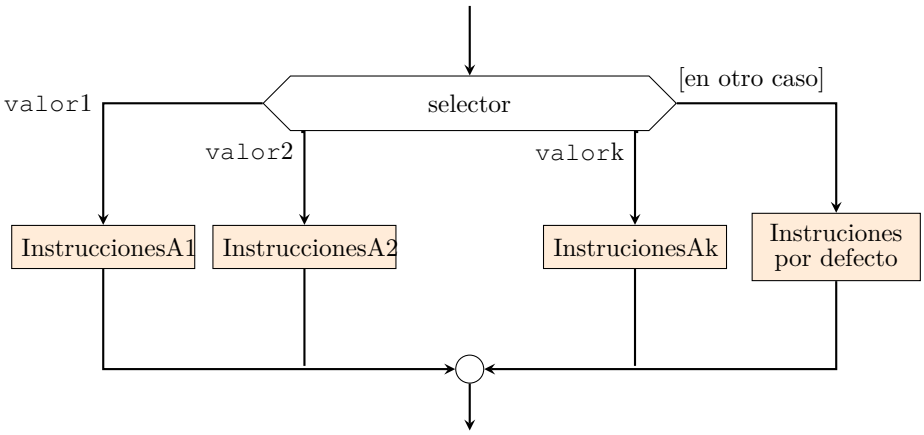


Figura 3.25: Decisiones múltiples - Estructura General

Observe que en el diagrama de flujo de la decisión múltiple (Figura 3.25) corresponde a un diagrama en donde todas las instrucciones `Caso` tienen su respectivo `FinCaso`. Si una de estas instrucciones de finalización (`FinCaso`) fuera omitida, por ejemplo la de la `valor2`, el diagrama general sufre un cambio y quedaría como el presentado en la Figura 3.26.

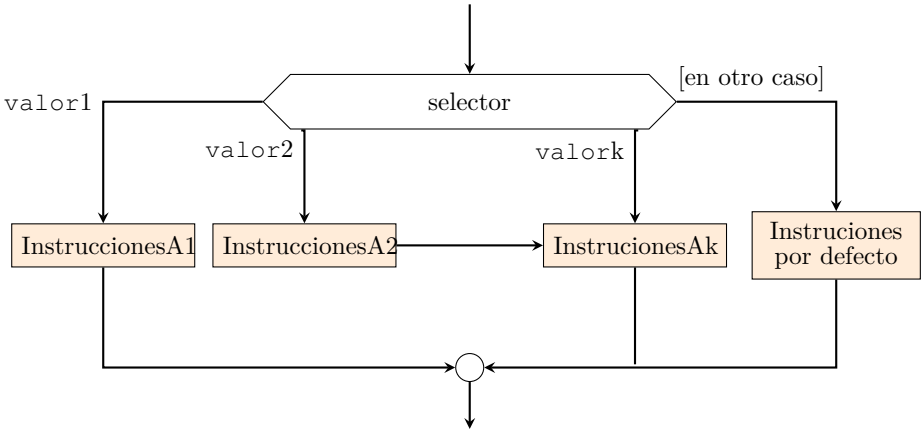


Figura 3.26: Decisiones múltiples - Sin el segundo `FinCaso`

Las decisiones múltiples son solo una forma abreviada de reescribir ciertas decisiones anidadas. Aquí se presentan algunos ejemplos que ilustran su uso.

.:Ejemplo 3.12. *Reescribir el Ejemplo 3.8 pero con decisiones múltiples.*

Considerando el análisis del Ejemplo 3.8, se propone entonces el Algoritmo 3.16.

Algoritmo 3.16: SistamaOperativo

```
1  Algoritmo SistamaOperativo
2     // Este algoritmo determina un mensaje según un carácter
3
4     // Declaración de la variable
5     Caracter opcion
6
7     // Datos disponibles
8     imprimir( "Ingrese un carácter: " )
9     leer( opcion )
10
11    // Resultados esperados
12    imprimir( "Su opción es " )
13
14    Segun( opcion )
15       Caso 'a':
16       Caso 'A':  imprimir( "Android" )
17                  FinCaso
18
19       Caso 'i':
20       Caso 'I':  imprimir( "iOS" )
21                  FinCaso
22
23       EnOtroCaso:
24                  imprimir( "inválida" )
25                  FinCaso
26     FinSegun
27  FinAlgoritmo
```

Al ejecutar el algoritmo:
Se obtiene exactamente el mismo resultado del ejemplo original.

Primera ejecución

```
Ingrese un carácter: i
Su opción es iOS
```

Segunda ejecución

```
Ingrese un carácter: A
Su opción es Android
```

Tercera ejecución

```
Ingrese un carácter: x
Su opción es inválida
```

Explicación del algoritmo:

Observe que para este ejemplo se omitió el `FinCaso` del primer y tercer valor (líneas 15 y 19), debido a que se necesita que se ejecute la misma instrucción de impresión para los casos en donde el valor es una letra mayúscula o minúscula.

Como la variable `selector` es de tipo `Caracter`, el valor de todos los valores se escribe entre comillas simples (' '). Si fuera de tipo `Cadena` se escribiría entre comillas dobles (" ") y si fuera `Entero`, solo se escribe el número.

Aclaración:



No es posible usar decisiones múltiples con operadores relacionales u operadores lógicos. Una decisión múltiple es una decisión de selección en donde cierta variable llamada `selector` almacena uno de los valores definidos y en cuyo caso se procede a ejecutar el conjunto de instrucciones asociadas a dicho valor.

.:Ejemplo 3.13. *Diseñe un algoritmo que permite conocer el valor del descuento de un artículo según su tipo. Todos los tipos y sus respectivos descuentos se pueden observar en la Tabla 3.3.*

Tipo	Descuento
1	12.5 %
2	8.3 %
3	3.2 %
Otro	0.0 %

Tabla 3.3: Opciones para el Ejemplo 3.13

Análisis del problema:

- **Resultados esperados:** el valor del descuento en el precio de un artículo según el tipo del mismo y definidos en la Tabla 3.3.
- **Datos disponibles:** el valor del artículo, así como el tipo de artículo.
- **Proceso:** se le solicita el usuario del valor del artículo y su respectivo tipo, luego según los porcentajes de descuento de la Tabla 3.3 para cada uno de los tipos se define el porcentaje, posteriormente con el porcentaje se calcula el valor del descuento y finalmente se imprime su valor.
- **Variables requeridas:**
  - `valor`: representa el valor del artículo
  - `descuento`: representa el valor del descuento.
  - `porcentaje`: valor que representa el porcentaje de descuento asignado al artículo según el tipo

De acuerdo al análisis planteado, se propone el Algoritmo 3.17.

Algoritmo 3.17: DescuentoArticulo

```
1  Algoritmo DescuentoArticulo
2     // Este algoritmo determina el descuento de un artículo
3
4     // Declaración de la variable
5     Caracter tipo
6     Real     valor, descuento, porcentaje
7
8     // Datos disponibles
9     imprimir( "Ingrese el valor del artículo: " )
10    leer( valor )
11
12    imprimir( "Ingrese el tipo de artículo: " )
13    leer( tipo )
14
15    // Se calculan los datos esperados
16    Segun( tipo )
17       Caso '1':  porcentaje = 0.125 // 12.5%
18                  FinCaso
19
20       Caso '2':  porcentaje = 0.083 // 8.3%
21                  FinCaso
22
23       Caso '3':  porcentaje = 0.032 // 3.2%
24                  FinCaso
```

```
25
26          EnOtroCaso:
27                      porcentaje = 0.0
28                      FinCaso
29      FinSegun
30
31      descuento = valor * porcentaje
32
33      // Resultados esperados
34      imprimir( "El valor del descuento es: ", descuento )
35  FinAlgoritmo
```

Al ejecutar el algoritmo:

Primera ejecución

```
Ingrese el valor del artículo: 2500
Ingrese el tipo de artículo: 2
El valor del descuento es: 207.5
```

Segunda ejecución

```
Ingrese el valor del artículo: 2500
Ingrese el tipo de artículo: 3
El valor del descuento es: 80
```

Explicación del algoritmo:

Después de declarar todas las variables (líneas 5 y 6) y solicitar los datos disponibles (líneas de la 9 a la 13), se procede a determinar el `porcentaje` de descuento usando la decisión múltiple (líneas de la 16 a la 29).

Una vez determinado el valor del porcentaje, se procede a calcular el valor del descuento al multiplicar este porcentaje por el valor del artículo (línea 31); finalmente se procede a imprimir el valor del `descuento`.

.:Ejemplo 3.14. *Diseñe un algoritmo que permita imprimir la capital de uno de los departamentos de la zona cafetera de Colombia o un mensaje que indique que el departamento especificado no hace parte de dicha zona.*

Análisis del problema:

- **Resultados esperados:** el nombre de la capital de uno de los departamentos de la zona cafetera o un mensaje que indique que el departamento indicado no hace parte de la zona cafetera de Colombia.

- **Datos disponibles:** el nombre del departamento.
- **Proceso:** solicitar al usuario el nombre de un departamento, luego por medio de una decisión múltiple se procede a determinar el nombre de la capital; finalmente se procede a imprimir la capital. Si el nombre del departamento no corresponde a un de los departamentos de la zona cafetera, se asigna (“ ”)[3] en la capital para poder posteriormente indicar que dicho departamento no hace parte de la zona cafetera.
- **Variables requeridas:**
  - `departamento`: nombre del departamento del que se desea conocer la capital
  - `capital`: nombre la capital del departamento indicado o (“”) para indicar que el departamento indicado no pertenece a la zona cafetera.

.

De acuerdo al análisis planteado, se propone el Algoritmo 3.18.

Algoritmo 3.18: Capitales

```
1  Algoritmo Capitales
2      // Este algoritmo determina la capital de los
3      // departamentos de la zona cafetera de Colombia
4
5      // Declaración de la variable
6      Cadena departamento, capital
7
8      // Datos disponibles
9      imprimir( "Ingrese el nombre del departamento: " )
10     leer( departamento )
11
12     // Se calculan los datos esperados
13     Segun( departamento )
14        Caso "Quindio":   capital = "Armenia"
15                          FinCaso
16
17        Caso "Caldas":    capital = "Manizales"
18                          FinCaso
19
20        Caso "Risaralda": capital = "Pereira"
21                          FinCaso
22        EnOtroCaso:
23                          capital = ""
24                          FinCaso
```

[3]Una cadena vacia

```
25        FinSegun
26
27        // Resultados esperados
28        Si( capital != "" ) Entonces
29           imprimir( "La capital del ", departamento )
30           imprimir( " es ", capital )
31        SiNo
32           imprimir ( "No es un departamento cafetero" )
33        FinSi
34     FinAlgoritmo
```

Al ejecutar el algoritmo:
Primera ejecución

```
Ingrese el nombre del departamento: Quindio
La capital del Quindio es Armenia
```

Segunda ejecución

```
Ingrese el nombre del departamento: Amazonas
No es un departamento cafetero
```

Explicación del algoritmo:

Después de declarar todas las variables (línea 6) y solicitar los datos disponibles (líneas 9 y 10), los datos deben ser ingresados de forma idéntica. Posteriormente se procede a determinar la `capital` del departamento usando la decisión múltiple (líneas de la 13 a la 25). Observe que el tipo de la variable `departamento` es `Cadena`, por lo tanto en los valores se escriben con comillas dobles, por ejemplo: `Caso "Quindio":`, `Caso "Caldas": ...`

Al finalizar esta estructura de decisión, el algoritmo ya habrá determinado el nombre de la capital del departamento ingresado; sin embargo, si se ingresó un departamento que no concuerda con uno del Eje cafetero, el algoritmo asignará una "cadena vacía" () a la variable `capital`.

Al final del algoritmo (líneas de la 28 a la 33), se muestra el nombre de la capital o en su defecto se un mensaje informando que el departamento ingresado no es uno del Eje cafetero.

.:Ejemplo 3.15. *La oficina de incorporación del ejército necesita un algoritmo que le pueda permitir saber si un aspirante a ingresar a la institución como soldado es apto o no para poder vincularlo. Para que una persona sea apta, debe cumplir los siguientes requisitos:*

- *Si es mujer, su estatura debe ser superior a 1.60 mts y su edad debe estar entre 20 y 25 años.*
- *Si el aspirante es hombre, se estatura debe ser superior a 1.65 mts y su edad debe estar entre los 18 y 24 años.*

Tanto mujeres como hombres deben ser solteros. Diseñe el algoritmo de tal forma que permita informar si un aspirante es apto o no para ingresar al ejército.

Aclaración:



Antes que nada, es importante recordar que un algoritmo puede implementarse de diferentes formas; la que se va a utilizar aquí es solo una de ellas.

Análisis del problema:

- **Resultados esperados:** informar si el aspirante al que se le han ingresado sus datos es "Apto" o "No es apto" para ingresar al ejército.
- **Datos disponibles:** el estado civil del aspirante, su género, su edad y su estatura.
- **Proceso:** lo primero será ingresar los datos disponibles, posteriormente, se deben usar varias estructuras de decisión que vayan permitiendo saber si el aspirante es apto o no lo es, dependiendo de si cumple los requisitos de estado civil, estatura y edad que se mencionan para cada género.
- **Variables requeridas:**
  - `genero`: representa el género o sexo del aspirante.
  - `estadoCivil`: corresponde al estado civil del aspirante.
  - `estatura`: es la estatura del aspirante.
  - `edad`: indica la edad del aspirante.
  - `salida`: variable que contendrá el mensaje a mostrar, o sea, si el aspirante es o no apto.

De acuerdo al análisis planteado, se propone el Algoritmo 3.19.

Algoritmo 3.19: AspiranteEjercito

```
1  Algoritmo  AspiranteEjercito
2    Caracter genero, estadoCivil
3    Cadena salida
4    Real estatura
5    Entero edad
6
7    imprimir( "Género del aspirante (M/F): " )
8    leer( genero )
9
10   imprimir( "Estado civil del aspirante (S/C/V/D/U): " )
11   leer( estadoCivil )
12
13   imprimir( "Estatura del aspirante: " )
14   leer( estatura )
15
16   imprimir( "Edad del aspirante: " )
17   leer( edad )
18
19   Si (estadoCivil == 'S' O estadoCivil == 's' ) Entonces
20     Segun( genero )
21       Caso 'F':
22       Caso 'f': Si( estatura > 1.60 Y
23                     edad >= 20 Y edad <= 25 ) Entonces
24                   salida = "Es Apto"
25                 SiNo
26                   salida = "No es Apto"
27                 FinSi
28
29       Caso 'M':
30       Caso 'm': Si( estatura > 1.65 Y
31                     edad>=18 Y edad<=24 ) Entonces
32                   salida = "Es Apto"
33                 SiNo
34                   salida = "No es Apto"
35                 FinSi
36               SiNo
37                 salida =  "No es Apto"
38               FinCaso
39
40       EnOtroCaso: salida = ""
41                   FinCaso
42     FinSegun
43    SiNo
44      salida =  "No es Apto"
45    FinSi
46
```

```
47    Si( salida == "" ) Entonces
48      imprimir( "Género incorrecto" )
49    SiNo
50      imprimir( "El aspirante: ", salida )
51    FinSi
52  FinAlgoritmo
```

Explicación del algoritmo:

El algoritmo inicia declarando las variables que se necesitan en las primeras líneas.

```
2   Caracter genero, estadoCivil
3   Cadena salida
4   Real estatura
5   Entero edad
```

A continuación, se solicitan los datos con la instrucción `imprimir` y se capturan con la instrucción `leer`.

```
7   imprimir( "Género del aspirante (M/F): " )
8   leer( genero )
9
10  imprimir( "Estado civil del aspirante (S/C/V/D/U): " )
11  leer( estadoCivil )
12
13  imprimir( "Estatura del aspirante: " )
14  leer( estatura )
15
16  imprimir( "Edad del aspirante: " )
17  leer( edad )
```

Acto seguido, se utiliza un decisión para determinar si el aspirante es soltero(a). Si es cumple esta decisión, se emplea una decisión múltiple para decidir sobre la edad y la estatura según el sexo. Note que cada caso considera la letra tanto en mayúscula, como en minúscula.

```
19  Si (estadoCivil == 'S' O estadoCivil == 's' ) Entonces
20    Segun( genero )
21      Caso 'F':
22      Caso 'f': Si( estatura > 1.60 Y
23                    edad >= 20 Y edad <= 25 ) Entonces
24                salida = "Es Apto"
25              SiNo
26                salida = "No es Apto"
27              FinSi
28
29      Caso 'M':
30      Caso 'm': Si( estatura > 1.65 Y
```

```
31                      edad>=18 Y edad<=24 ) Entonces
32                    salida = "Es Apto"
33                  SiNo
34                    salida = "No es Apto"
35                  FinSi
36                SiNo
37                  salida =  "No es Apto"
38                FinCaso
39
40       EnOtroCaso: salida = ""
41                   FinCaso
42      FinSegun
43    SiNo
44      salida =  "No es Apto"
45    FinSi
```

En caso que el usuario ingrese un `genero` incorrecto, el algoritmo ejecutará la instrucción `EnOtroCaso`, allí, se almacenará en la variable `salida` una cadena vacía (“”). Esta cadena vacía servirá como bandera para poder imprimir el mensaje adecuado en la parte final del algoritmo.

```
47  Si( salida == "" ) Entonces
48    imprimir( "Género incorrecto" )
49  SiNo
50    imprimir( "El aspirante: ", salida )
51  FinSi
```

Observe lo que pasaría si en lugar de asignar a la variable `salida` una cadena vacía, se asignara el mensaje `"Género incorrecto"`; en este caso, al imprimir el mensaje al final se obtendría:

```
El aspirante: Género incorrecto
```

En lugar de:

```
Género incorrecto
```

.:Ejemplo 3.16. *Reescribiendo el Ejemplo 3.11 pero con decisiones múltiples y usando la variable* `estrato` *de tipo* `Entero`*, para ilustrar su uso en decisiones múltiples.*

De acuerdo al análisis planteado, se propone el Algoritmo 3.20.

Algoritmo 3.20: FacturaAgua

```
1 Algoritmo FacturaAgua
2   // Este algoritmo calcula el pago de una factura de agua
3   Entero estrato
```

```
4   Real cantidad, cargoFijo, consumo, valorRecoleccion,
5        totalPago
6
7   imprimir( "Estrato socioeconómico del predio: " )
8   leer( estrato )
9   imprimir( "Cantidad de metros consumidos: " )
10  leer( cantidad )
11
12  Segun( estrato )
13
14    Caso 1:  cargoFijo = 2500.0
15             valorConsumo = cantidad * 2200.0
16             valorRecolección = 5500.0
17             FinCaso
18
19    Caso 2:  cargoFijo = 2800.0
20             valorConsumo = cantidad * 2350.0
21             valorRecolección = 6200.0
22             FinCaso
23
24    Caso 3:  cargoFijo = 3000.0
25             valorConsumo = cantidad * 2600.0
26             valorRecolección = 7400.0
27             FinCaso
28
29    Caso 4:  cargoFijo = 3300.00
30             valorConsumo = cantidad * 3400.00
31             valorRecolección = 8600.0
32             FinCaso
33
34    Caso 5:  cargoFijo = 3700.0
35             valorConsumo = cantidad * 3900.0
36             valorRecolección = 9700.0
37             FinCaso
38
39    EnOtroCaso:
40             cargoFijo = 2800.00
41             valorConsumo = cantidad * 2350.00
42             valorRecolección = 6200.00
43             FinCaso
44  FinSegun
45
46  totalPago = cargoFijo + valorConsumo + valorRecoleccion
47
48  imprimir( "Valor del cargo fijo: ", cargoFijo )
49  imprimir( "Valor del consumo: ", cantidad )
50  imprimir( "Valor de la recolección: ", valorRecoleccion )
51  imprimir( "Total a pagar: ", totalPago )
52 FinAlgoritmo
```

Explicación del algoritmo:

Complementando la explicación del algoritmo original (página 159), se puede observar como se reemplazó toda la decisión anidada por una decisión múltiple. Cada caso dentro de la decisión múltiple representa un estrato socioeconómico. Note que se asume (`EnOtroCaso`) que cualquier valor diferente a los valores del 1 al 5 será considerado el estrato 6.

3.4. Ejercicios propuestos

1. Crear un algoritmo que indique el valor del descuento de un artículo el cual es del 5% solo si el artículo tiene un costo superior al $150.000.

2. Crear un algoritmo que indique si la llave de un tanque de agua debe ser abierta o cerrada. El tanque debe estar siempre entre 250 y 450 litros.

3. Crear un algoritmo que dado un número entero entre 0 y 20 diga si es o no un número primo.

   Recuerde que los números primos menores o iguales a 20 son: $2, 3, 5, 7, 11, 13, 17, 19$.

4. Crear un algoritmo que indique si un estudiante ganó o perdió un curso después de presentar los cinco trabajos asociados al curso (Notas entre 0.0 y 5.0). Los trabajos tienen igual peso sobre la nota final y se gana el curso si la nota definitiva es superior a 3.5

5. Crear un algoritmo que permita saber si una ecuación cuadrática tiene o no solución.

   Recuerde que una ecuación cuadrática se define como:

   $$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

   Y se dice que tiene solución si el valor del discriminaste (que corresponde al cálculo interno de la raíz cuadrada $b^2 - 4ac$) es mayor o igual a cero y el valor de a es diferente de cero.

6. Crear un algoritmo que indique si un número entero x, ingresado por el usuario, se encuentra por dentro o por fuera del intervalo cerrado-cerrado $[minimoValor,\ maximoValor)]$ también ingresados por el usuario.

Por ejemplo: si los valores mínimos y máximo son 3 y 7 respectivamente, el valor 5 está dentro, mientras que el valor de 8 está por fuera del intervalo.

7. Crear un algoritmo que indique si un número entero x se encuentra por dentro o por fuera de tres intervalos abierto-abierto cuyo rangos no se interceptan entre sí y sus límites son ingresados por el usuario.

8. Crear un algoritmo que indique el valor del descuento de un artículo dependiendo de su tipo:

Tipo	**Porcentaje de descuento**
Textil	`0 %`
Electrodoméstico	`3.7 %`
Elementos de cocina	`4.2 %`
Video juego	`7.8 %`

Tabla 3.4: Tabla de descuento por tipo de artículo

9. Crear un algoritmo que indique el valor del descuento de un artículo dependiendo de su valor:

Rango de valores	**Porcentaje de descuento**
$0.0 hasta $100.000	`0 %`
Más de $100.000 hasta $225.000	`1.5 %`
Más de $225.000 hasta $375.000	`3.8 %`
Más de $375.000	`10.3 %`

Tabla 3.5: Tabla de descuento según el rango de valores

Capítulo 4

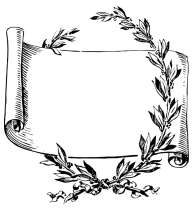
Estructuras de repetición

Si el código y los comentarios no coinciden, posiblemente ambos sean erróneos

Norm Schryer

Objetivos del capítulo:

- Identificar el tipo de variables que intervienen en el manejo de estructuras de repetición (contador, acumulador, centinela).
- Comprender el funcionamiento de las diferentes estructuras de repetición.
- Implementar algoritmos que involucran el uso de estructuras de repetición.
- Elaborar pruebas de escritorios con procesos iterativos



En la solución de algunos problemas, se requiere que el algoritmo ejecute la misma tarea de forma repetitiva; es ahí donde cobran protagonismo las estructuras de repetición. Estas estructuras también reciben el nombre de **ciclos** o de **bucles**, ellas permiten que una instrucción o un bloque de ellas se pueda ejecutar más de una vez. A cada ejecución de un ciclo se le denomina iteración.

En este libro se estudiarán las siguientes estructuras de repetición:

- `Mientras` - `FinMientras`
- `Haga` - `MientrasQue`
- `Para` - `FinPara`

4.1. Conceptos básicos

Es importante que el lector, antes de iniciar la construcción de los algoritmos que las incluyen, se familiarice con algunos conceptos básicos entorno a ellas.

4.1.1 Contador

Como su nombre lo indica, es una variable que tiene la función de llevar la cuenta de determinadas situaciones que se repiten dentro de los ciclos.

Por ejemplo, es común contar:

- Goles en un juego de un partido de fútbol.
- Cantidad de términos que tiene una serie numérica.
- Los amigos que se tienen en Facebook.
- Cantidad de votos que obtienen los candidatos en unas elecciones.
- El número de visitas a un video en YouTube.

Esta clase de variables se declaran de tipo `Entero`. Deben inicializarse antes de entrar al ciclo, es decir, se le debe asignar un valor inicial que corresponde al número desde el cual se requiere que inicie el conteo, generalmente se inicializan en 0; aunque todo depende del problema que se va a resolver.

Dentro del ciclo debe haber una instrucción que modifique el valor del contador, su forma general es la siguiente:

```
contador = contador + valorIncremento
contador = contador - valorDecremento
```

El valor a incrementar o decrementar, hace referencia a cualquier cantidad de tipo numérico, en la cual aumenta o disminuye su valor en cada iteración del ciclo. Los valores a incrementar o decrementar son cantidades constantes.

En los ejemplos anteriores, la cantidad de amigos en Facebook se incrementa en el momento que se confirme una solicitud de amistad o se decrementa cuando se elimina un contacto de la lista de amigos.

4.1.2 Acumulador

Un acumulador es una variable que funciona de forma similar a un contador, la diferencia radica en que aumentan o disminuyen en cantidades variables y no en forma constante como es el caso de los contadores. Algunos autores le dan también la denominación de totalizador [Corona and Ancona, 2011].

Un acumulador se puede usar para:

- Almacenar el puntaje acumulado en un juego.
- Calcular el saldo en una cuenta de ahorros.
- Obtener el valor de una sumatoria de notas que luego puede ser usada para calcular un promedio.
- Determinar el valor a pagar en un supermercado cuando se compran varios artículos.
- Conocer el acumulado de puntos que se tienen por compras en un almacén o establecimiento comercial que ofrezca este beneficio.

Esta clase de variables se declaran de tipo numérico, bien sea `Entero` o `Real`. Deben inicializarse antes de entrar al ciclo, es decir, se le debe asignar un valor inicial que dependerá del problema a resolver, generalmente se inicializan en 0.

Dentro del ciclo debe haber una instrucción que modifique el valor del acumulador, su forma general es la siguiente:

```
acumulador = acumulador + valorIncremento
acumulador = acumulador - valorDecremento
```

El valor a incrementar o valor a decrementar se refiere a cualquier cantidad de tipo numérico, en la cual el acumulador aumenta o disminuye su valor en cada iteración del ciclo. Los valores a incrementar o decrementar son cantidades variables.

En los ejemplos anteriores, el saldo en una cuenta de ahorros se incrementa con las consignaciones y se decrementa con los retiros. De igual forma sucede con los puntos por compras, incrementan por cada compra que se realice y decrementan en el momento en que se haga un canje por algún artículo o promoción.

Existen casos especiales donde la forma general no trabaja con operaciones de suma o resta, sino que se involucran operaciones de multiplicación o división:

```
acumulador = acumulador * valorIncremento
acumulador = acumulador / valorDecremento
```

Aclaración:



Se debe tener especial cuidado en la inicialización del acumulador cuando se trabaje con multiplicaciones, su valor inicial no puede ser 0, debido a que todas las operaciones realizadas con él darían como resultado 0. De igual manera si se trabaja con divisiones, la variable `valorDecremento` por ningún motivo podría tomar el valor de 0, la división entre 0 no está definida.

4.1.3 Bandera

Es una variable que se usa para controlar diferentes acciones dentro de un algoritmo. Algunos textos se refieren a esta variable como interruptor, conmutador o centinela [Joyanes A., 1996].

Una variable bandera puede ser definida virtualmente de cualquier tipo de dato. Generalmente estas variables toman uno de dos valores posibles, ese valor depende del tipo de dato con que fue declarada.

Si la variable es de tipo `Logico`, los únicos valores posibles son `Verdadero` o `Falso`. Si la variable es de tipo `Entero` los valores posibles

podrían ser 1 o 0, que se pueden interpretar como verdadero o encendido para el 1 y falso o apagado para el 0. Si la declaración se hizo de tipo `Caracter`, puede tomar cualquier valor; los valores más comunes son 'S'o 'N', interpretados como 'S' = Sí o 'N'= No. Sin embargo, para estos dos últimos tipos de datos, el diseñador del algoritmo le puede asignar los valores que estime convenientes.

Es fundamental darle un valor inicial a la variable bandera, el cual cambiará dependiendo de ciertas condiciones que estarán dadas por la solución del problema. De acuerdo a lo explicado en los párrafos anteriores, el valor inicial debe ser uno de los posibles que puede tomar; una vez se presente la situación esperada estos valores deberán cambiar su estado al valor contrario, es decir, si se inicializó en `Verdadero` cambiará a `Falso`, si fue en 1 cambiará a 0 y si fue en 'S'cambiará a 'N'. Luego de ejecutar las instrucciones correspondientes podrían retomar su valor inicial.

En los siguientes párrafos a lo largo de este capítulo, a medida que se estudian las diferentes estructuras de repetición, se ilustrará el uso de este tipo de variables.

4.2. Estructura `Mientras - FinMientras`

Es una estructura de repetición que permite que una instrucción o un conjunto de ellas se ejecuten una o más veces, o por el contrario que no lleguen a ejecutarse ya que todo depende del resultado de una condición que debe evaluarse al inicio del ciclo.

La forma general de esta estructura de repetición es presentada en el segmento del Algoritmo 4.1.

Algoritmo 4.1: Forma general - Mientras-FinMientras

```
1 Instrucción de inicialización
2 Mientras( condición )
3   Instrucción-1
4   Instrucción-2
5     ...                    /* Cuerpo del ciclo */
6   Instrucción-n
7   Instrucción modificadora de condición
8 FinMientras
9 Instrucción externa
```

Teniendo en cuenta que el `Mientras-FinMientras` es una instrucción repetitiva condicionada al inicio, se debe prestar especial cuidado en inicializar la variable o las variables que serán evaluadas en

la condición, ya que de esto depende que se ejecute o no el cuerpo del ciclo; es por ello que dentro de la anterior forma general se contempla una Instrucción de inicialización (línea 1). De igual manera la inicialización también se aplica a los contadores y acumuladores que serán modificados dentro del ciclo. La inicialización puede estar implícita en el mismo algoritmo o puede ser suministrada por el usuario.

Al encontrar la instrucción `Mientras` se debe evaluar la condición (línea 2), la cual estará representada por una expresión relacional o lógica. Si el resultado de la evaluación es verdadero, entonces se procede a ejecutar el cuerpo del ciclo (líneas de la 3 a la 7) hasta encontrar la instrucción `FinMientras`; luego el control del algoritmo regresa al inicio del ciclo, es decir a la instrucción `Mientras` y una vez más se evalúa la condición. Este proceso terminará en el momento que la evaluación de la condición arroje un resultado falso, en cuyo caso el control del algoritmo lo asume la Instrucción externa (línea 9), la cual no hace parte del ciclo.

Cuando se evalúe por primera vez la condición y el resultado sea falso, las instrucciones que componen el cuerpo del ciclo no se ejecutan; el control lo asume la Instrucción externa.

En conclusión, este ciclo itera mientras el valor de la condición sea verdadero.

Es importante tener presente que todo ciclo debe terminar de ejecutarse cuando cumpla con la tarea para la cual fue diseñado, para ello dentro de su cuerpo se encuentra la Instrucción modificadora de condición (línea 7) cuyo propósito es cambiar el estado de la condición. De omitir la instrucción modificadora, se obtendrá lo que se conoce como un "Ciclo infinito", debido a que su ejecución "nunca termina". Aunque en la forma general está representada de última en la secuencia de instrucciones que componen el cuerpo del ciclo, no necesariamente debe ocupar ese lugar.

Para representar la estructura de repetición `Mientras – FinMientras` mediante un diagrama de flujo, se usa la notación que se presenta en la Figura 4.1.

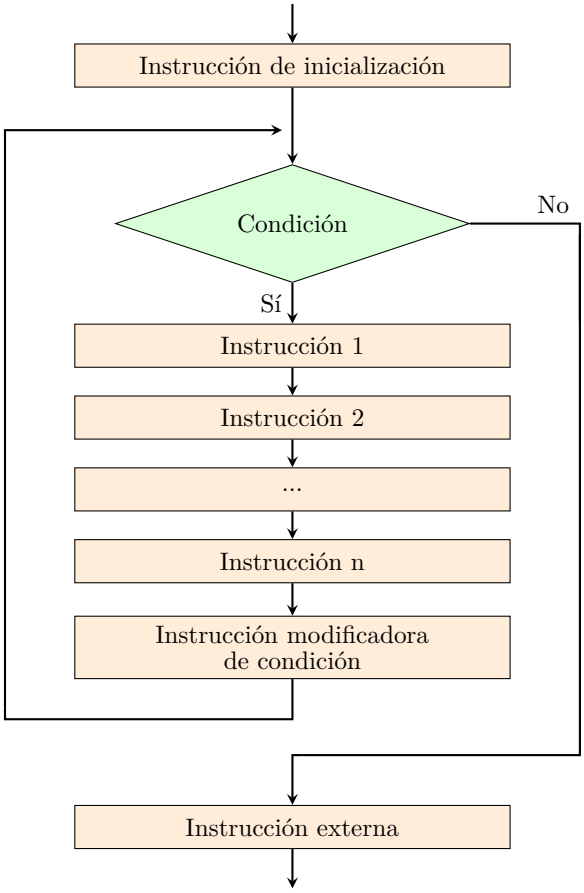


Figura 4.1: Forma general del ciclo `Mientras - FinMientras`

A continuación, se ilustrará el funcionamiento del ciclo `Mientras - FinMientras` mediante el siguiente segmento de un algoritmo que imprime los números del 1 al 10, se explicará en dos versiones (segmentos de Algoritmo 4.2 y 4.3).

Algoritmo 4.2: Imprimir los números del 1 al 10 - Primera Versión

```
1 numero = 1
2 Mientras( numero <= 10 )
3     imprimir( numero )
4     numero = numero + 1
5 FinMientras
```

Para una explicación más clara, se identificarán sus partes de acuerdo a la forma general expresada en párrafos anteriores (Ver Tabla 4.1).

Línea	Explicación
1	Instrucción de inicialización
2	Condición e inicio del ciclo
3	Cuerpo del ciclo
4	Cuerpo del ciclo e instrucción modificadora de condición
5	Fin del ciclo. Regresa el control a la línea 2.

Tabla 4.1: Explicación del Algoritmo 4.2

En la primera línea la variable numero se inicializa en 1. Pasando a la línea 2 se encuentra la instrucción `Mientras` con su respectiva condición.

El cuerpo del ciclo está conformado por dos instrucciones (líneas 3 y 4), que se ejecutan mientras que la condición de la línea 2 sea verdadera, es decir, mientras el contenido de la variable `numero` sea menor o igual a 10.

La instrucción `imprimir` se ejecuta 10 veces, imprimiendo uno a uno los números del 1 al 10.

La operación que incluye el contador `numero`: `numero = numero + 1`, además de incrementar en cada iteración en 1 el valor de la variable `numero`, hace el papel de la instrucción modificadora de condición, con el propósito de llegar a cambiar de verdadero a falso el estado de la condición. Esto lo logra en el momento que la variable tome el valor de 11, así al evaluar la condición (`numero <= 10`) se obtiene un resultado falso, con lo cual se da por terminada la ejecución del `Mientras`.

Debe tener en cuenta que cada problema puede tener múltiples soluciones, por ejemplo, el siguiente segmento de algoritmo también imprime los números del 1 al 10:

Algoritmo 4.3: Imprimir los números del 1 al 10 - Segunda Versión

```
1 numero = 0
2 Mientras( numero < 10 )
3     numero = numero + 1
4     imprimir( numero )
5 FinMientras
```

Comparando el Algoritmo 4.3 (la versión 2) con el Algoritmo 4.2 (la versión 1) se puede observar lo siguiente:

Línea 1: la inicialización es diferente, se hizo en 0.

Línea 2: el ciclo se ejecuta mientras el valor de numero sea menor a 10, en lugar de menor o igual a 10, como se planteó en la versión 1 del algoritmo. Esto obedece a que la inicialización fue en 0 en lugar de 1. Recuerde que en la versión 1 del algoritmo la variable `numero` alcanza a tomar el valor de 11, en esta nueva versión el valor llega hasta 10.

Líneas 3 y 4: se codificaron las expresiones `numero = numero + 1` e `imprimir ( numero )`, que son iguales a la planteadas en las líneas 4 y 3 del algoritmo versión 1, respectivamente. En esta solución fue necesario cambiar el orden de ejecución de las instrucciones del cuerpo del ciclo, ya que si se dejaban en el orden como habían sido escritas en la versión 1 del algoritmo, el resultado sería la impresión de los números del 0 al 9 y no del 1 al 10 como se esperaría.

Línea 5: la instrucción `FinMientras` retorna el control a la línea 2 para que sea evaluada la condición y se determine si continúan o paran las iteraciones.

Del anterior análisis se puede establecer que, de acuerdo a la inicialización de la variable que controlará la ejecución del ciclo, se debe prestar especial cuidado en elegir correctamente el operador relacional que hará parte de la condición del `Mientras`. En los segmentos de algoritmos analizados se pudo observar lo siguiente:

- `numero = 0` se utiliza el operador $<$
- `numero = 1` se utiliza el operador $<=$

También es fundamental tener en cuenta el orden de las instrucciones dentro del cuerpo del ciclo, el no ubicarlas en la secuencia lógica correspondiente podría dar resultados inesperados.

Aclaración:



Para un mismo problema se pueden plantear diferentes soluciones algorítmicas. Por lo tanto, su solución no tiene porque ser exactamente igual a las que se muestran en este libro.

Los dos anteriores segmentos de algoritmo se pueden representar gráficamente con los diagramas de flujo presentados en la Figura 4.2.

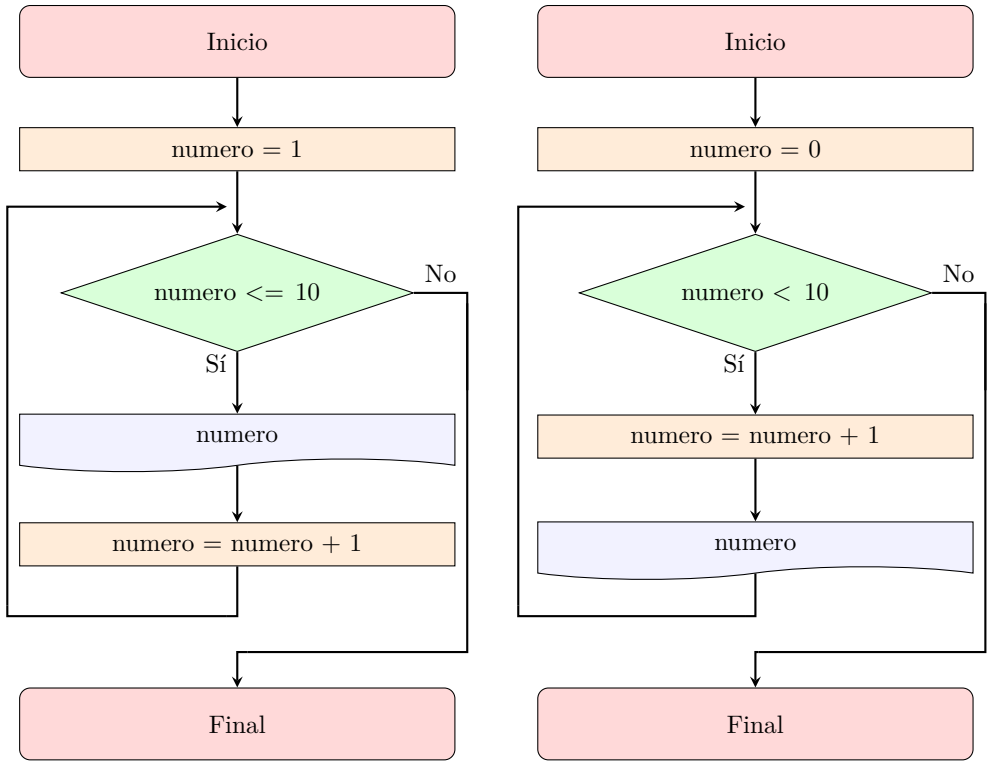


Figura 4.2: Diagrama de flujo Numeros

Aclaración:



Las variables que intervienen en la condición del ciclo deben tener un valor inicial antes de su primera evaluación.

El bloque de instrucciones o cuerpo del ciclo, se ejecutará mientras el resultado de la evaluación de la condición sea `Verdadero`. Si en la primera evaluación de la condición del `Mientras`, el resultado es `Falso`, las instrucciones del cuerpo del ciclo no se ejecutan.

La evaluación de una expresión relacional o lógica, siempre dará como resultado un valor `Verdadero` o `Falso`.

A continuación, se darán algunos enunciados como ejemplo para ser resueltos mediante la estructura repetitiva `Mientras-FinMientras`.

.:Ejemplo 4.1. *Diseñe un algoritmo que permita generar e imprimir la siguiente serie de números:* $1, 3, 5, 7, 9, 11, \ldots, n$

El algoritmo deberá recibir un número entero (n) que indicará la cantidad de términos de la serie.

Análisis del problema:

- **Resultados esperados:** este algoritmo debe generar e imprimir la siguiente serie: $1, 3, 5, 7, 9, 11, \ldots, n$

- **Datos disponibles:** Para la solución de este ejemplo se proporcionará la cantidad de términos que tendrá la serie (n).

- **Proceso:** la serie está conformada por números impares. El primer término es el número 1, los siguientes tienen un incremento de 2 en 2. Para lograr la serie, debe hacerse un proceso repetitivo que terminará en el momento que se complete la cantidad de términos especificada.

  Dentro de las instrucciones que conforman el cuerpo del ciclo se debe imprimir cada término de la serie. Así mismo, se deben ir contando los términos impresos para poder determinar el fin de las iteraciones.

- **Variables requeridas:**

  - `cantidadTerminos`: almacena el número de términos que tendrá la serie (n).
  - `contadorNumeros`: llevará el control de la cantidad de términos que se vayan imprimiendo, a la vez que se utilizará en la condición del ciclo.
  - `termino`: representará cada uno de los términos de la serie que se van generando.

De acuerdo al anterior análisis, se plateará la solución a través de un diagrama de flujo y luego mediante pseudocódigo.

En la Figura 4.3 se muestra la solución del Ejemplo 4.1 mediante un diagrama de flujo.

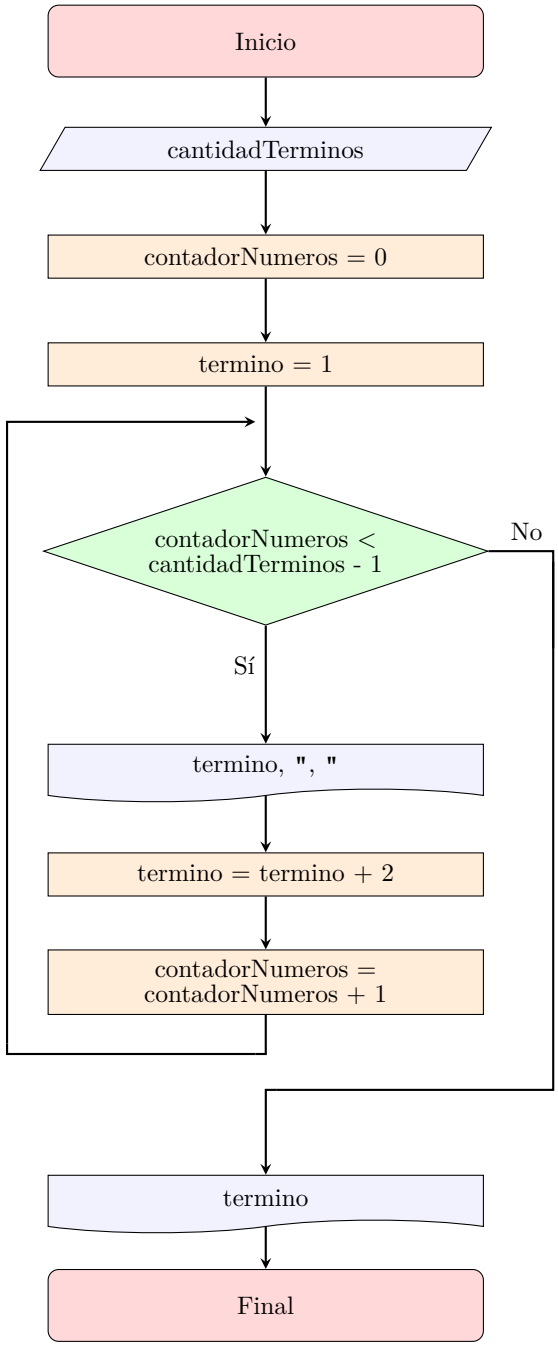


Figura 4.3: Algoritmo Serie

La solución expresada en pseudocódigo es la siguiente:

Algoritmo 4.4: Serie

```
1  Algoritmo Serie
2    /* Este algoritmo imprime la siguiente serie, de acuerdo
3       al número de términos que se le especifique:
4       1, 3, 5, 7, 9, 11, ..., n
5     */
6
7    // Declaración de variables
8    Entero cantidadTerminos, contadorNumeros, termino
9
10   // Dato disponible
11   imprimir( "Ingrese la cantidad de términos a generar: " )
12   leer( cantidadTerminos )
13
14   // Inicialización de variables
15   contadorNumeros = 0
16   termino = 1
17
18   // Generación de la serie
19   // Resultados esperados
20   Mientras( contadorNumeros < cantidadTerminos - 1 )
21     imprimir( termino, ", " )
22     termino = termino + 2
23     contadorNumeros = contadorNumeros + 1
24   FinMientras
25
26   imprimir( termino )
27 FinAlgoritmo
```

Al ejecutar el algoritmo:

```
Ingrese la cantidad de términos a generar: 15
1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29
```

Explicación del algoritmo:

En este algoritmo la Instrucción de inicialización está dada por tres instrucciones. La primera se encuentra en la línea 12, en donde se lee la cantidad de términos; la segunda corresponde a la inicialización de la variable `contadorNumeros` en 0, ubicada en la línea 15. Ambas variables hacen parte de la condición planteada en el `Mientras`. En este caso se tiene una inicialización suministrada por el usuario con la instrucción `leer` y otra implícita en el mismo algoritmo con la asignación `contadorNumeros = 0`.

La tercera inicialización, se encuentra en la línea 16, donde la variable `termino` recibe el valor de 1, el cual corresponde al primer número de la serie a generar.

Seguidamente para lograr el proceso repetitivo se implementó la estructura `Mientras`, cuya condición se estableció así:

```
Mientras( contadorNumeros < cantidadTerminos - 1 )
```

Donde `contadorNumeros` contará la cantidad de números que se van imprimiendo, esta variable controlará que el ciclo se ejecute mientras su valor sea menor al de `cantidadTerminos - 1`. Es necesario restar 1 a la cantidad de términos para que la impresión del último número no vaya acompañada de una coma. La impresión de este último término se realiza en la línea 26.

Suponga que para este ejemplo se desean imprimir los primeros 15 términos de la serie. En este caso la variable `cantidadTerminos` almacenaría dicho valor, al hacer la evaluación de la condición `contadorNumeros < cantidadTerminos - 1`, daría un resultado verdadero ya que contadorNumeros se inicializó en 0.

Mientras la condición sea verdadera se procede a ejecutar las instrucciones del cuerpo del ciclo (líneas 20 al 22). La primera instrucción, que está ubicada en la línea 20, imprime el contenido de la variable `termino`, acompañado de una coma (`,`) y un espacio en blanco con el fin de separarlo del siguiente número, de esta manera se irá mostrando al usuario la conformación de la serie. El primer número a imprimir es el 1.

Luego de la impresión se encuentra la instrucción `termino = termino + 2`, con la cual se incrementa la variable `termino` en 2 unidades, ya que los términos de la serie tienen este comportamiento (inician en 1 y se van incrementando de 2 en 2: $1, 3, 5, 7, 11, \ldots$).

Como última instrucción del cuerpo del ciclo se tiene la instrucción `contadorNumeros = contadorNumeros + 1`, que tiene como propósito ir contando la cantidad de números que se imprimen y a la vez es la variable que sirve para controlar el número de iteraciones del ciclo. De acuerdo a la forma general presentada anteriormente para el ciclo `Mientras`, esta es la instrucción modificadora de condición.

Después de incrementar el contador de los números se encuentra la instrucción `FinMientras`, la cual devuelve el control al inicio del ciclo, es decir, a la instrucción `Mientras`, donde una vez más se evalúa la

condición (línea 19). Mientras el valor de la condición sea `Verdadero` continuará la repetición del cuerpo del ciclo. En el momento que la variable `contadorNumeros` alcance el valor de la variable `cantidadTerminos - 1` el valor de la condición será `Falso` y el control lo asume la instrucción que está por debajo del `FinMientras`, la cual imprime el último termino, para luego pasar al final de la ejecución del algoritmo (`FinAlgoritmo`).

Buena práctica:



Es aconsejable que los contadores inicien en el valor de 0, salvo en aquellas soluciones que exigen un valor diferente.

.:Ejemplo 4.2. *Un profesor de Fundamentos de Programación, desea que le diseñen un algoritmo con el cuál se pueda determinar cuántos de sus estudiantes, de uno de sus grupos, aprobaron o reprobaron la materia, así mismo desea conocer el promedio general del grupo. Se considera que la materia es aprobada con una nota mínima de* 3.0.

Para esta tarea el profesor posee el código de cada estudiante y la nota definitiva que obtuvo en la materia.

Análisis del problema:

- **Resultados esperados:** de acuerdo al enunciado, se espera que el algoritmo informe los siguientes datos:
  - Cantidad de estudiantes que aprobaron la materia.
  - Cantidad de estudiantes que reprobaron la materia.
  - Promedio general del grupo.
- **Datos disponibles:** en este problema se conocen tres datos que deben ser proporcionados por el usuario del algoritmo para su correcta ejecución. El primero es el código de cada estudiante, el segundo es la nota definitiva de la materia cada uno de ellos; aunque en el enunciado no está explícito, se infiere que se tiene claridad sobre la cantidad de estudiantes del grupo, este sería el tercer dato de entrada que se le debe proporcionar al algoritmo para solucionar el problema.

- **Proceso:** para poder determinar la cantidad de estudiantes que aprobaron y reprobaron la materia se requiere el uso de dos contadores, uno para cada caso. Recuerde que la fórmula general de un contador es la siguiente:

```
contador = contador + valorIncrementar
```

Para el caso específico de este ejemplo, el valor a incrementar tendrá un valor de 1, el cual corresponde a cada uno de los estudiantes que aprobaron o reprobaron la materia.

Adicionalmente se debe calcular el promedio general del grupo. Todo promedio involucra tres valores:

```
promedio = sumatoria / cantidad
```

En este ejemplo la `sumatoria` es la suma de todas las notas definitivas de los estudiantes, y la `cantidad` está representada por el número de estudiantes del curso.

De lo anterior se deduce que es necesario una fórmula adicional, con la cual se pueda calcular la `sumatoria`. Para este caso, se usará la forma general del cálculo de un acumulador, concepto analizado en el numeral 4.1.2:

```
acumulador = acumulador + valorIncrementar
```

El `acumulador` está representado por la sumatoria y el valor a incrementar por la nota definitiva de cada estudiante:

```
sumatoria = sumatoria + notaDefinitiva
```

En esta parte del análisis de la solución del problema, se puede determinar que ya se tienen establecidos los cálculos necesarios para su solución. Pero falta determinar cuáles de todas estas instrucciones van antes, dentro y después del ciclo.

Una posible solución es la siguiente: después de la declaración de las variables, se debe solicitar el número de estudiantes a los cuales se les va a procesar la nota, este dato determinará el número de veces que se debe ejecutar el ciclo. Luego se inicializan los contadores y los acumuladores necesarios. Seguidamente se debe establecer el ciclo y su respectiva condición.

Dentro de las instrucciones del cuerpo del ciclo se debe solicitar el código del estudiante y la nota definitiva en la materia.

Una vez se tenga el valor de la nota definitiva se puede determinar si aprobó o reprobó la materia; para ello se hará uso del árbol de decisión de la Figura 4.4.

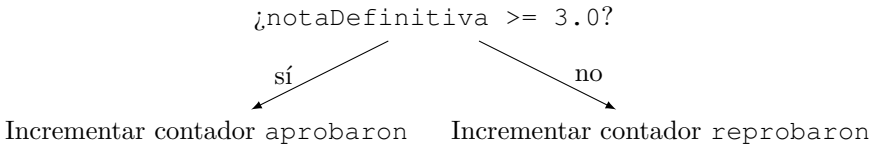


Figura 4.4: Árbol de decisión del Ejemplo 4.2

Otra instrucción que debe estar dentro del ciclo, es la que acumula la sumatoria de las notas definitivas de cada estudiante.

Cuando se termine con la ejecución del ciclo, se calculará el promedio general del grupo; finalmente se mostrarán los resultados que solicita el enunciado.

- **Variables requeridas:**
  - `cantidadEstudiantes`: determinará cuántas veces debe ejecutarse el ciclo.
  - `codigoEstudiante`: identifica el estudiante a procesar.
  - `notaDefinitiva`: nota obtenida por el estudiante que se está procesando en el momento.
  - `contadorEstudiantes`: llevará el control del número de estudiantes que se van procesando, a la vez se utilizará en la condición del ciclo.
  - `aprobaron`: es un contador que se incrementará cuando la nota definitiva obtenida en la materia sea mayor o igual a 3.0.
  - `reprobaron`: es un contador que se incrementará cuando la nota definitiva obtenida en la materia sea menor a 3.0.
  - `sumaDefinitivas`: acumulador que llevará la sumatoria de las notas definitivas.
  - `promedioGrupo`: una vez finalizado el ciclo se usará para calcular el promedio general del grupo.

Con el anterior análisis, se establece la solución (Ver Algoritmo 4.5).

Algoritmo 4.5: Estudiantes

```
1  Algoritmo Estudiantes
2     /* De una cantidad de estudiantes se determina cuántos
3        aprobaron y reprobaron la materia.
4        Adicionalmente se calcula el promedio general.
5     */
6
7     // Declaración de variables
8     Real   notaDefinitiva, sumaDefinitivas, promedioGrupo
9     Entero cantidadEstudiantes, contadorEstudiantes
10    Entero aprobaron, reprobaron
11    Cadena codigoEstudiante
12
13    // Entrada de datos
14    imprimir( "Ingrese la cantidad de estudiantes: " )
15    leer( cantidadEstudiantes )
16
17    // Inicialización de variables
18    contadorEstudiantes = 0
19    aprobaron = 0
20    reprobaron = 0
21    sumaDefinitivas = 0
22
23    // Inicia el proceso repetitivo
24    Mientras( contadorEstudiantes < cantidadEstudiantes )
25      imprimir( "Ingrese el código del estudiante: " )
26      leer( codigoEstudiante )
27      imprimir( "Ingrese la nota definitiva: " )
28      leer( notaDefinitiva )
29
30      Si( notaDefinitiva >= 3.0 ) Entonces
31         aprobaron = aprobaron + 1
32      SiNo
33         reprobaron = reprobaron + 1
34      FinSi
35
36      sumaDefinitivas = sumaDefinitivas + notaDefinitiva
37      contadorEstudiantes = contadorEstudiantes + 1
38    FinMientras
39
40    // Calculo necesario para el promedio
41    promedioGrupo = sumaDefinitivas / cantidadEstudiantes
42
43    // Resultados esperados
44    imprimir( "La cantidad que aprobaron es: ", aprobaron )
45    imprimir( "La cantidad que reprobaron es: ", reprobaron )
46    imprimir( "El promedio es: ", promedioGrupo )
47 FinAlgoritmo
```

Explicación del algoritmo:

En este algoritmo fue necesario trabajar conjuntamente con una estructura condicional `Si-FinSi` (líneas 30 a la 34) y una estructura repetitiva `Mientras-FinMientras` (líneas 24 a la 38). Como se observa la instrucción `Si-FinSi` está dentro del ciclo `Mientras-FinMientras`, es decir forma parte de su cuerpo.

La instrucción `Mientras` se utiliza para lograr que el conjunto de instrucciones que conforman su cuerpo se ejecuten un determinado número de veces. La instrucción `Si-FinSi` será la responsable de determinar, en cada iteración, si la materia fue o no aprobada.

La cantidad de veces que se repetirá el ciclo, estará determinada por el valor que digite el usuario (líneas 14 y 15):

```
14 imprimir( "Ingrese la cantidad de estudiantes: " )
15 leer( cantidadEstudiantes )
```

De lo anterior, se deduce que se está hablando de un ciclo cuyas iteraciones están controladas por un contador, como se evidencia en la condición planteada en el `Mientras` (línea 24):

```
24 Mientras( contadorEstudiantes < cantidadEstudiantes )
```

Donde, `contadorEstudiantes` es una variable de control que llevará la cuenta del número de estudiantes a los que se les vaya procesando la nota (línea 37). Para ello se utiliza la siguiente expresión:

```
37 contadorEstudiantes = contadorEstudiantes + 1
```

Dentro de la forma general que se explicó para el ciclo `Mientras`, esta operación representa la instrucción modificadora de condición; en cada iteración que se haga, su contenido se incrementará en 1. Cuando el contenido de la variable `contadorEstudiantes` sea igual al de la variable `cantidadEstudiantes`, el ciclo terminará.

Después de la instrucción `FinMientras` (línea 38) se efectúa una división para calcular el promedio general del grupo (línea 41). Esta operación se debe realizar por fuera del ciclo, una vez se terminen sus iteraciones:

```
41 promedioGrupo = sumaDefinitivas / cantidadEstudiantes
```

Como observación adicional a la solución algorítmica presentada para este ejemplo, se debe tener en cuenta que la variable

cantidadEstudiantes, que actúa como divisor en la anterior expresión, recibirá su valor por parte del usuario del algoritmo; en el caso de que le asignen como valor el 0, esta operación generará un error, puesto que la división entre 0 no está definida. Es responsabilidad de los diseñadores de algoritmos o de programas de computador evitar que se generen errores en la ejecución de los mismos.

Dado lo anterior, se puede mejorar el funcionamiento de este algoritmo incluyendo las siguientes líneas, en reemplazo de la instrucción de la línea 41. De esta forma, al calcular el promedio general del grupo se evita un posible error de ejecución:

```
// Se asume 0 en el promedio cuando la cantidad de
// estudiantes es 0.
Si( cantidadEstudiantes >= 1 ) Entonces
  promedioGrupo = sumaDefinitivas / cantidadEstudiantes
SiNo
  promedioGrupo = 0
FinSi
```

Esta nueva decisión, verificaría que la cantidad de estudiantes sea mínimo de 1 y así poder determinar el promedioGrupo; en caso contrario, es decir que el valor sea 0 o un valor negativo, el algoritmo asignaría el valor de 0 a la variable promedioGrupo y este será el resultado que informaría.

En las Figuras 4.5 y 4.6 se muestra la solución del Ejemplo 4.2 mediante un diagrama de flujo.

Buena práctica:



Mediante las instrucciones necesarias se debe controlar que los algoritmos no generen errores en su ejecución; como por ejemplo, cuando se trata de realizar la división entre 0.

Algunos lenguaje de programación poseen instrucciones adecuadas para el manejo de errores y excepciones que facilitan este tipo de control.

También es posible definir condiciones que se deben garantizar para que el algoritmo funcione correctamente. A estas condiciones se les denomina **precondiciones**.

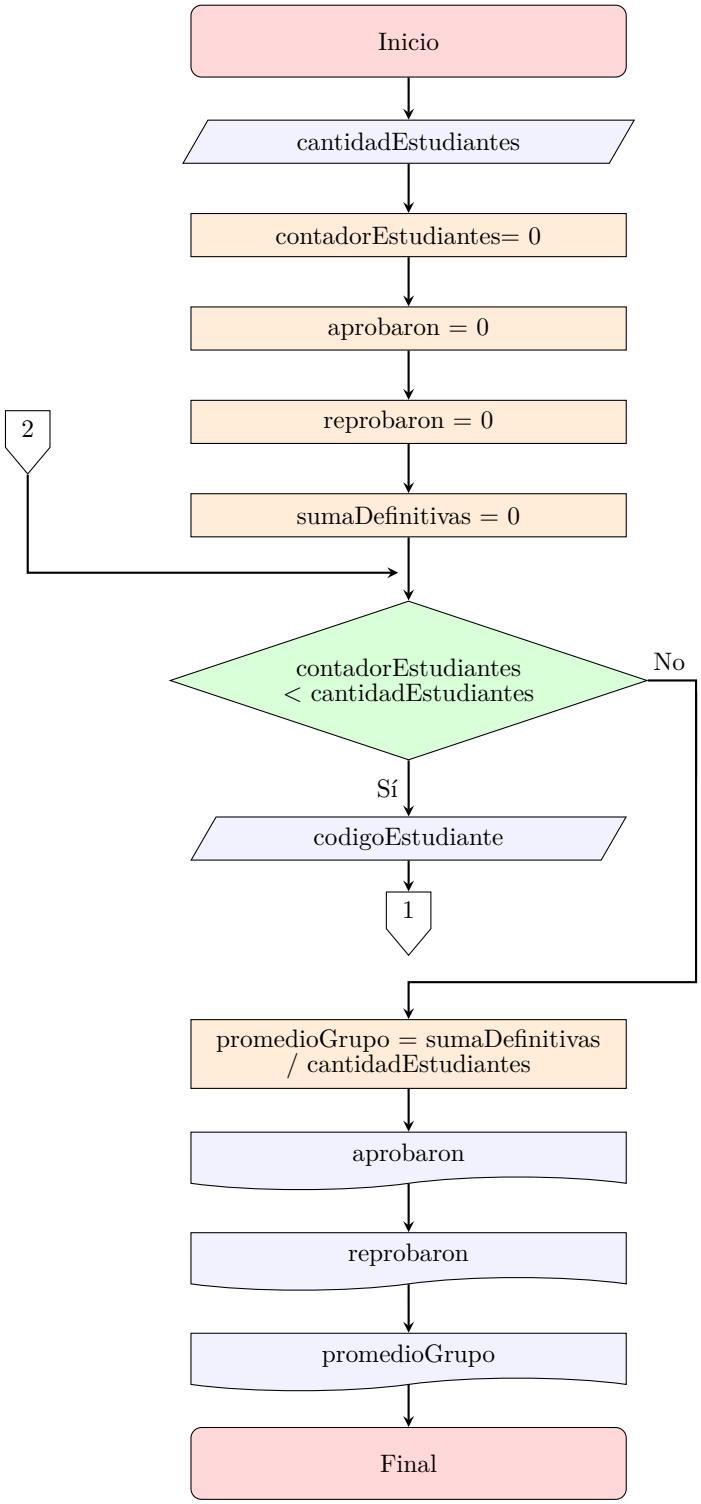


Figura 4.5: Algoritmo Estudiantes - Parte 1

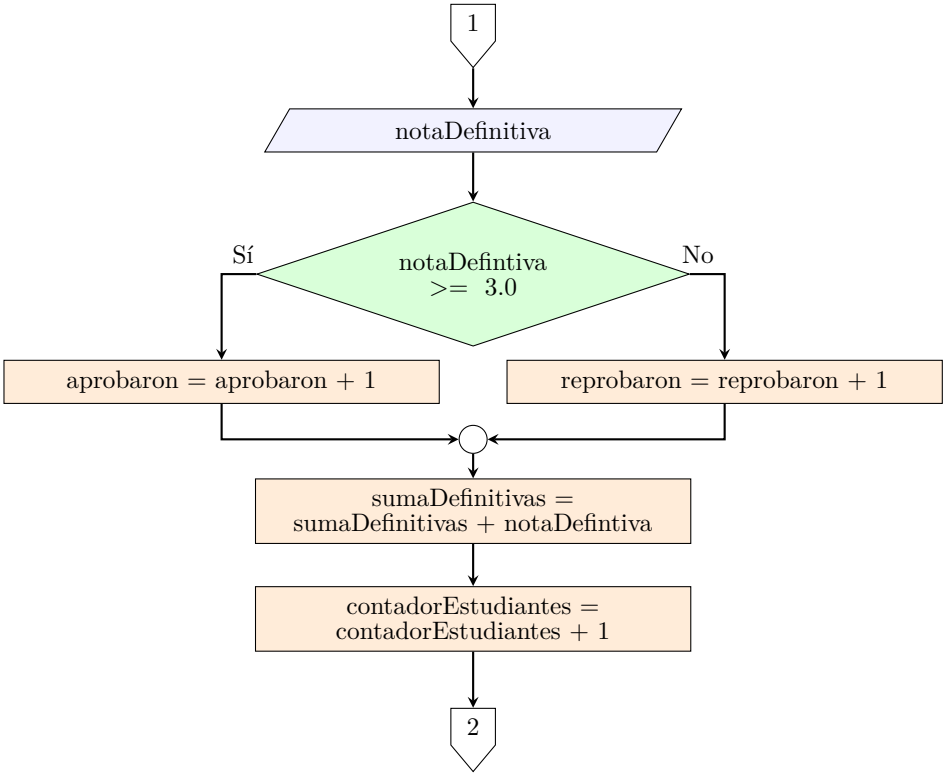


Figura 4.6: Algoritmo Estudiantes - Parte 2

.:Ejemplo 4.3. *Diseñe un algoritmo que reciba como dato de entrada un número entero perteneciente al sistema decimal*[1]*. Si cumple la condición de ser positivo, informe el número de cifras que posee, adicionalmente calcule la sumatoria de ellas; en caso contrario, imprima un mensaje que diga que el número no es positivo.*

Análisis del problema:

- **Resultados esperados:** la cantidad de cifras que tiene el número leído y la sumatoria de ellas. Adicionalmente un mensaje en el caso que el número no sea positivo.
- **Datos disponibles:** un número entero.
- **Proceso:** leer un número entero. Tomar una decisión para determinar si es positivo, en caso de serlo se deben separar cada una de sus cifras y simultáneamente se van sumando; en caso contrario informar que no es positivo.

[1]Sistema decimal: que tiene como base el número 10.

Se puede determinar que un número es positivo con la siguiente decisión (Ver Figura 4.7).

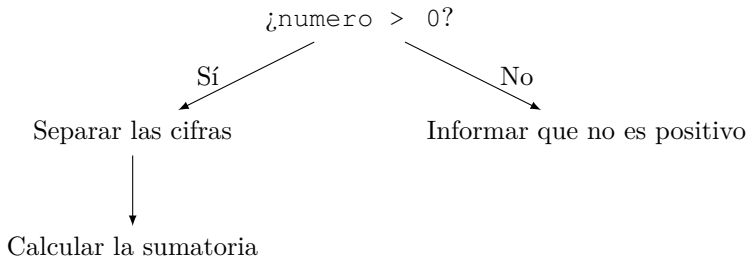


Figura 4.7: Árbol de decisión del Ejemplo 4.3

La descomposición de un número entero del sistema decimal en sus respectivas cifras, se puede realizar con divisiones enteras sucesivas entre 10. Suponga que desea separar las cifras que tiene el número 8345, para ello se hacen las siguientes operaciones:

Iteración	**Dividendo**	**Divisor**	**Cociente**	**Resto**
1	8345	10	834	5
2	834	10	83	4
3	83	10	8	3
4	8	10	0	8

Tabla 4.2: Descomposición del número 8345 en cifras

Para mayor claridad en la siguiente explicación, tenga presente el nombre de los términos que intervienen en una división: dividendo, divisor, residuo (resto) y cociente. En una división entera, al residuo se le denomina resto.

$$\begin{array}{r|l} \text{dividendo} & \text{divisor} \\ \hline \text{resto} & \text{cociente} \end{array}$$

Como se anotó anteriormente, la descomposición de un número entero en sus cifras, se logra haciendo divisiones sucesivas. El número de veces que se repite esta operación, es igual al número de cifras del número a descomponer (para este ejemplo es 4). El proceso termina cuando el cociente tome un valor de 0.

En la primera iteración que se debe realizar se hace la siguiente operación:

```
8 3 4 5 | 1 0
  3 4   | 8 3 4
    4 5 |
      5 |
```

Observe que el resto (residuo) es 5, que corresponde a la última cifra del número, y el cociente es 834, que equivale al número original sin la última cifra. Para obtener estos valores se deben hacer dos divisiones enteras[2] entre 10.

La primera división que se efectuará será el módulo o resto de la división, para ello se usa el operador %: con el cual se consigue el resto (residuo):

```
8345 % 10 = 5
```

Con la segunda división se obtiene el cociente, el cual va tomando el valor del número original pero reducido en una cifra; ese cociente pasa a ser el dividendo en la siguiente iteración, para obtenerlo se hace una división entera entre 10 usando el operador /:

```
8345 / 10 = 834
```

En la segunda iteración se hacen las mismas operaciones, teniendo en cuenta que el cociente obtenido en la división anterior, pasa a ser el nuevo dividendo:

```
8 3 4 | 1 0
  3 4 | 8 3
    4 |
```

Para obtener el resto de la división se hace la siguiente operación:

```
834 % 10 = 4
```

Para calcular el cociente, se realiza la división entera:

```
834 / 10 = 83
```

[2]Se obtienen resultados sin decimales (fracciones).

Una vez más se deben realizar las divisiones, teniendo en cuenta que el nuevo dividendo es el cociente anterior:

$$\begin{array}{r|l} 8\ 3 & 1\ 0 \\ \cline{2-2} 3 & 8 \end{array}$$

Para obtener el resto de la división y el cociente se hacen las siguientes divisiones:

```
83 % 10 = 3
```

```
83 / 10 = 8
```

Como el cociente aún no es 0, se debe realizar una última iteración:

$$\begin{array}{r|l} 8 & 1\ 0 \\ \cline{2-2} 8 & 0 \end{array}$$

Para obtener el resto de la división y el cociente se hacen las siguientes divisiones:

```
8 % 10 = 8
```

```
8 / 10 = 0
```

De lo anterior se deduce que, para separar las cifras se deben realizar una o varias divisiones enteras sucesivas, en donde se obtenga el cociente de tipo entero y el residuo o resto de la división; para el caso de este texto, esos resultados se logran usando los operadores `/` y `%`, respectivamente.

- **Variables requeridas:**
  - `numero`: se usará para almacenar el número a leer.
  - `copiaNumero`: almacenará una copia del valor original del número, con el propósito de ir eliminando la última cifra. Esta variable representa el cociente dentro de los términos de una división, pero tenga presente que ese cociente pasa a ser el dividendo en la siguiente iteración.
  - `contadorCifras`: cuenta las cifras del número.
  - `sumaCifras`: acumula la sumatoria del valor de cada una de las cifras del número.

- `cifra`: variable que almacena una a una las cifras en las que se va descomponiendo el número. Dentro de los términos de una división representa al residuo o al resto de la división entera.

De acuerdo al análisis planteado, se propone el Algoritmo 4.6.

Algoritmo 4.6: CifrasNumero

```
1  Algoritmo CifrasNumero
2    /* Lee un número decimal (base 10). Si es positivo calcula
3       la sumatoria de sus cifras, en caso contrario informa
4       que no es un número positivo.
5    */
6
7    // Declaración de variables
8    Entero numero, cifra, contadorCifras, sumaCifras,
9           copiaNumero
10
11   // Se lee el dato conocido
12   imprimir( "Escriba un número entero: " )
13   leer( numero )
14
15   // Determina si el número es positivo
16   Si( numero > 0 ) Entonces
17     copiaNumero = numero
18     contadorCifras = 0
19     sumaCifras = 0
20
21     // Se inicia el proceso de separación de las cifras
22     Mientras( copiaNumero > 0 )
23       cifra = copiaNumero % 10
24       copiaNumero = copiaNumero / 10
25       sumaCifras = sumaCifras + cifra
26       contadorCifras = contadorCifras + 1
27     FinMientras
28
29     // Resultados esperados en caso de ser positivo
30     imprimir( "La cantidad de cifras de: ", numero )
31     imprimir( "son: ", contadorCifras )
32     imprimir( "La sumatoria es: ", sumaCifras )
33   SiNo
34     // Resultado esperado si no es positivo
35     imprimir( "No es un número positivo" )
36   FinSi
37 FinAlgoritmo
```

Al ejecutar el algoritmo:

Primera ejecución:

```
Escriba un número entero: 459

La cantidad de cifras de: 459 son: 3
La sumatoria es: 18
```

Segunda ejecución:

```
Escriba un número entero: -459

No es un número positivo
```

Explicación del algoritmo:

Al contenido de la variable `numero` se le hace una copia en la variable `copiaNumero`, esto con el propósito de conservar el valor original para poderlo imprimir con los resultados. De esta manera la descomposición de cifras se hace sobre la copia y no sobre la variable `numero`.

Al igual que en el Ejemplo 4.2 la solución del problema requiere el uso en conjunto de la estructura condicional `Si-FinSi` y de la estructura repetitiva `Mientras-FinMientras`, pero en esta ocasión el ciclo `Mientras-FinMientras` hace parte del grupo de instrucciones que se ejecutan cuando la condición del `Si-FinSi` es verdadera.

El ciclo `Mientras-FinMientras` en cada iteración hace lo siguiente:

- Evalúa la condición (línea 22)
- Separa una cifra del número (línea 23)
- Elimina la última cifra al número (línea 24)
- Acumula la sumatoria de las cifras (línea 25)
- Cuenta las cifras (línea 26)
- Retorna el control al Mientras (línea 27)

Todo este proceso lo realiza mientras el valor de la variable `copiaNumero` sea mayor a 0.

Si al evaluar la condición `Si (numero > 0)` arroja un resultado falso, el ciclo `Mientras-FinMientras` no se ejecuta, en su lugar se imprime el mensaje: “No es un número positivo”.

Dos aspectos a resaltar en este ejemplo:

- Las iteraciones del ciclo no están condicionadas mediante un contador, en este caso se hacen mediante un acumulador:

```
22 Mientras( copiaNumero > 0 )
```

- La instrucción modificadora de condición está a cargo de una división, la cual se ejecuta de manera sucesiva mientras la condición sea verdadera, reduciendo en cada iteración el valor de la variable `copiaNumero`:

```
24 copiaNumero = copiaNumero / 10
```

En la Figura 4.8 se muestra la solución del Ejemplo 4.3 mediante un diagrama de flujo.

Aclaración:



Una estructura de decisión puede hacer parte del cuerpo o conjunto de instrucciones de una estructura de repetición, o viceversa.

Buena práctica:



Cuando los datos de entrada van a ser modificados dentro del proceso que ejecuta el algoritmo, se debe utilizar otra variable para copiarlos y así conservar su valor original.

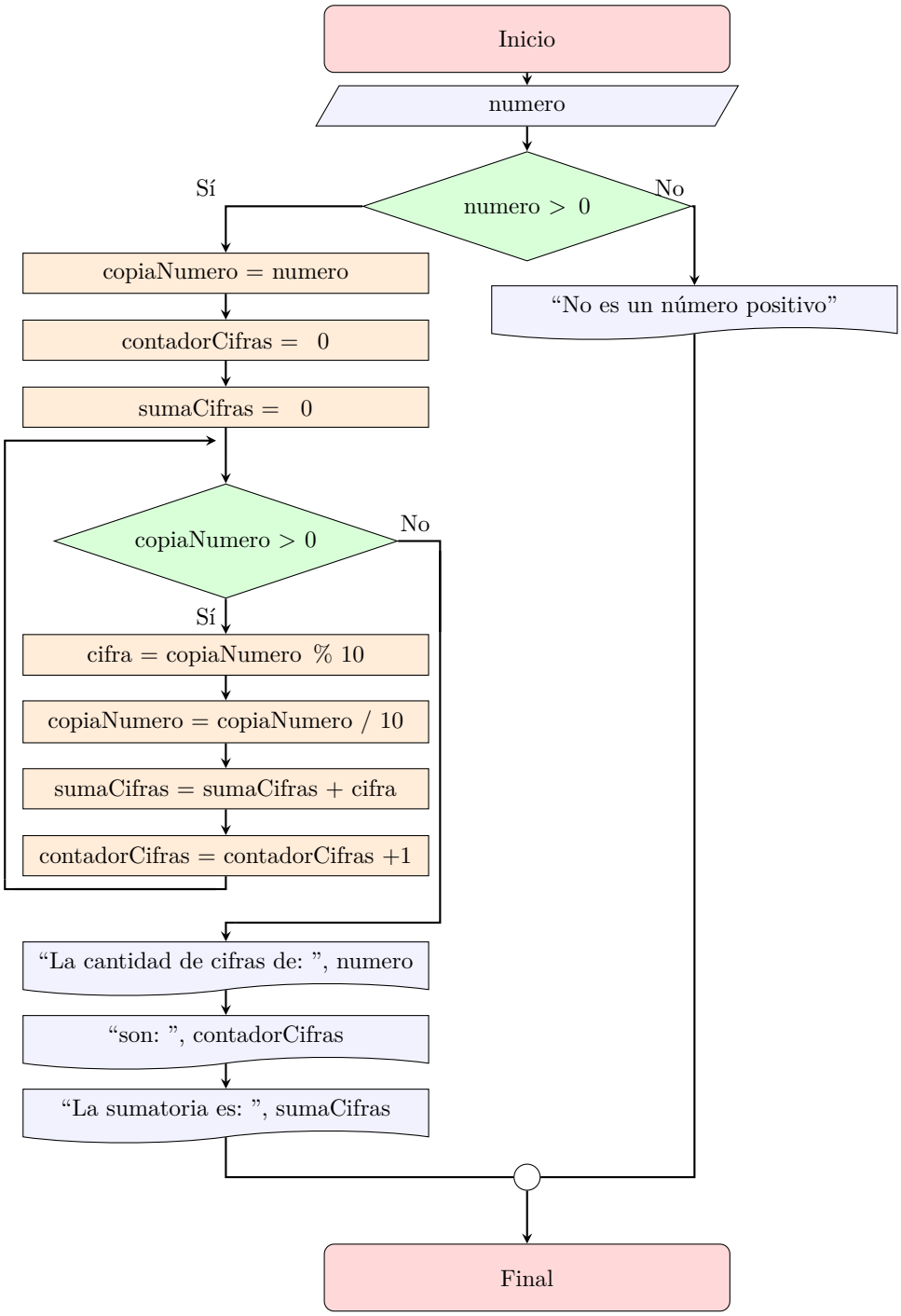


Figura 4.8: Diagrama de flujo del Algoritmo `CifrasNumero`

.:Ejemplo 4.4. *Dentro del contexto de las matemáticas recreativas*[3] *se encuentra el concepto de número de Armstrong, también conocido como número narcisista; definido como aquel en que la suma de cada una de sus cifras elevadas a la potencia n es igual a él mismo, donde n está dada por la cantidad de cifras o dígitos del número.*

Por ejemplo, el número 407*, que posee* 3 *cifras, es un Armstrong dado que:*

$$4^3 + 0^3 + 7^3 = 64 + 0 + 343 = 407$$

Son también números de Armstrong los siguientes: 1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407, 1634, 8208, 9474, 54748, 92727, 93084, 548834, 1741725, entre otros

De acuerdo al anterior contexto, construya un algoritmo que reciba un número entero positivo en base 10 y determine si es o no un número de Armstrong.

Análisis del problema:

- **Resultados esperados:** mensaje que informe si el número ingresado es o no un Armstrong.
- **Datos disponibles:** un número entero positivo en base 10.
- **Proceso:** la solución de este problema se puede realizar con los siguientes pasos:
  - Leer un número entero positivo.
  - Determinar el número de cifras o dígitos de ese número. Para ello se usa el procedimiento explicado en el Ejemplo 4.3[4].
  - Calcular la sumatoria de sus dígitos elevados a la potencia n, donde n estará determinada por el número de cifras resultantes del proceso anterior.
  - Tomar una decisión que determine si el número leído es igual a la sumatoria mencionada en el paso anterior (Ver Figura 4.9).

[3]Estudio y solución por puro pasatiempo de problemas y acertijos relacionados con las matemáticas [Bishop et al., 2004].

[4]Se recomienda leer el análisis de dicho ejemplo.

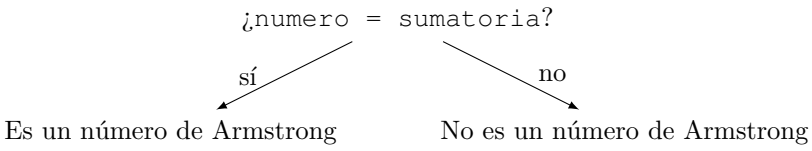


Figura 4.9: Árbol de decisión del Ejemplo 4.4

- **Variables requeridas:**
  - `numero`: almacenará el valor proporcionado por el usuario del algoritmo.
  - `copiaNumero`: en esta variable se hará copia del valor original del número y con ella se efectuarán los cálculos necesarios para la separación de las cifras; así se podrá conservar el contenido original en la variable `numero`.
  - `contadorCifras`: contará el número de dígitos que tiene el número, a la vez será utilizada como el valor de la potencia (`n`) a la que se eleva cada uno de los dígitos.
  - `cifra`: se utilizará para ir almacenando las cifras o dígitos que se separen del número.
  - `sumaCifras`: en ella se calculará la sumatoria de las cifras elevadas a la potencia n (`contadorCifras`).

En las Figuras 4.10 y 4.11 se muestra la solución del Ejemplo 4.4 mediante un diagrama de flujo.

Debido a que la solución es un poco extensa, fue necesario dividir el diagrama en dos páginas, observe que para poder seguir el flujo se utilizó un conector a otra página.

Buena práctica:



Cuando los diagramas son muy extensos, se deben dividir en varias partes, las cuales se enlazan con conectores a la misma página o con conectores a otra página.

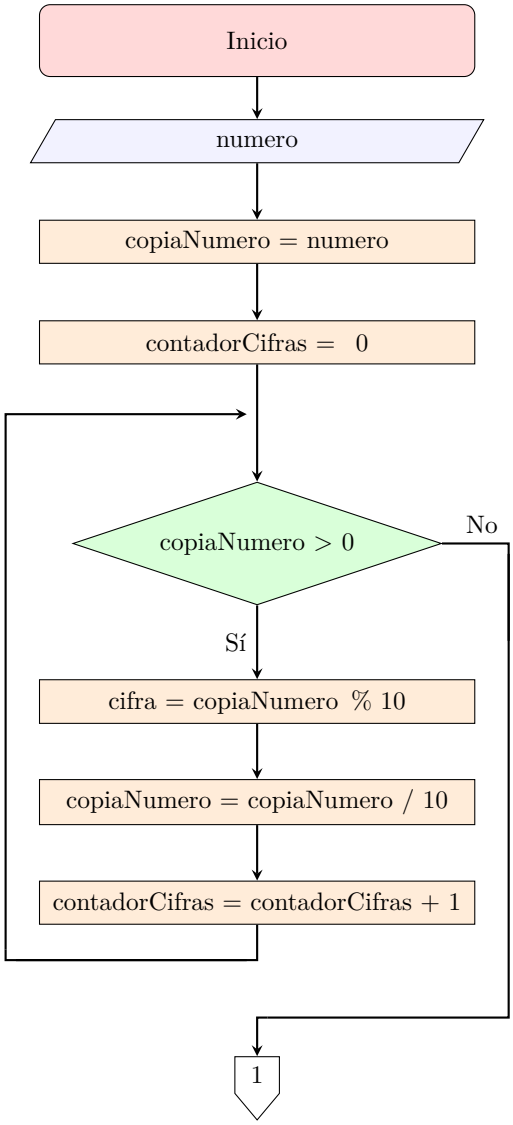


Figura 4.10: Diagrama del flujo del Algoritmo `Armstrong` - Parte 1

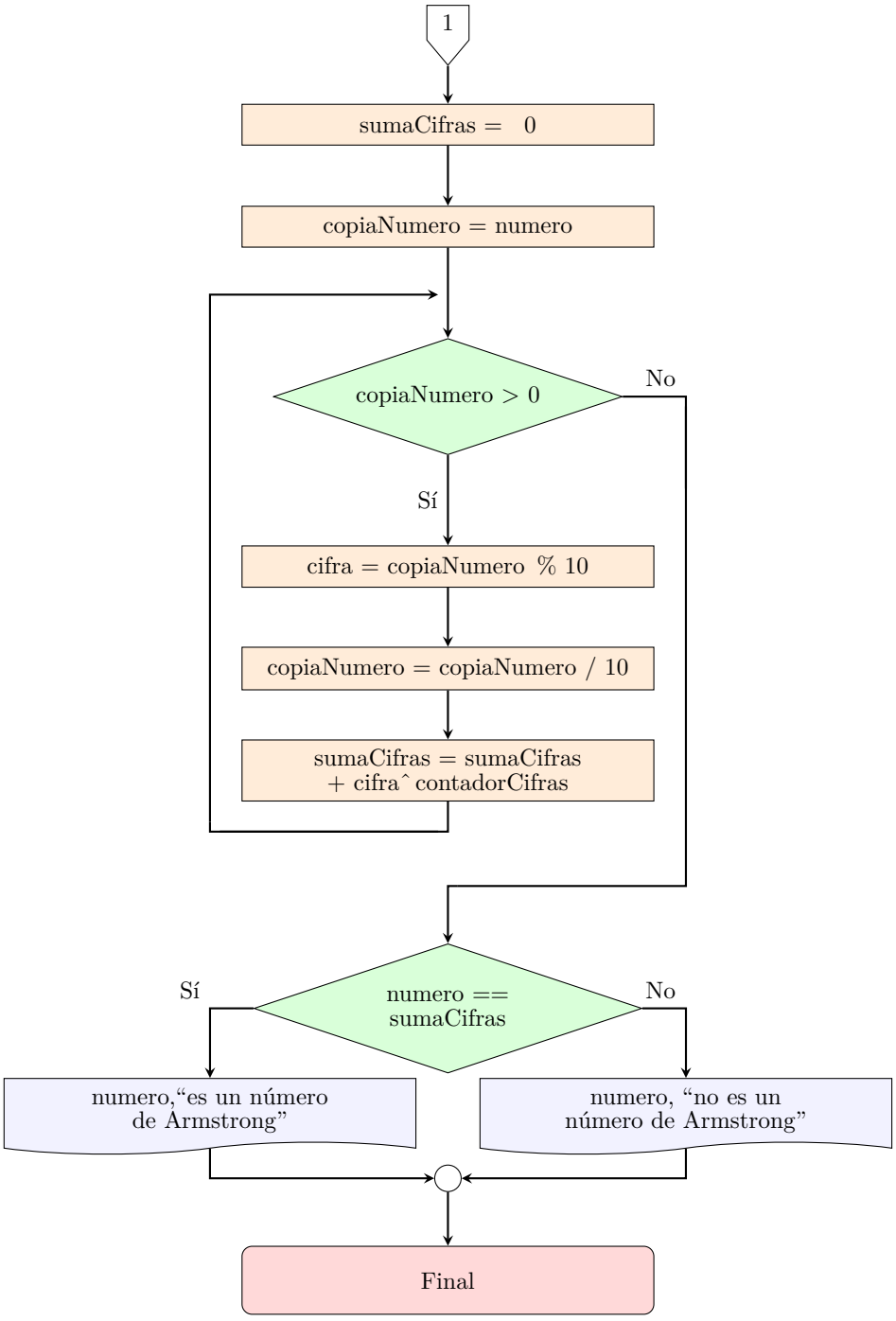


Figura 4.11: Diagrama del flujo del Algoritmo `Armstrong` - Parte 2

A continuación se muestra la solución a través de pseudocódigo (Algoritmo 4.7).

Algoritmo 4.7: Armstrong

```
1  Algoritmo Armstrong
2    /* Recibe un número entero positivo y determina si es
3       o no un número de Armstrong.
4     */
5    // Declaración de variables
6    Entero numero, copiaNumero, contadorCifras, cifra,
7           sumaCifras
8
9    // Lectura del dato disponible
10   imprimir( "Escriba el número a analizar: " )
11   leer( numero )
12
13   // Inicialización de variables
14   // Se cuenta el número de cifras del número
15   contadorCifras = 0
16   copiaNumero = numero
17   Mientras( copiaNumero > 0 )
18     cifra = copiaNumero % 10
19     copiaNumero = copiaNumero / 10
20     contadorCifras = contadorCifras + 1
21   FinMientras
22
23   // Se calcula la sumatoria de sus dígitos
24   // elevados a la potencia n (contadorCifras)
25   sumaCifras = 0
26   copiaNumero = numero
27   Mientras( copiaNumero > 0 )
28     cifra = copiaNumero % 10
29     copiaNumero = copiaNumero / 10
30     sumaCifras = sumaCifras + cifra ^ contadorCifras
31   FinMientras
32
33   // Resultado esperado
34   Si( numero == sumaCifras ) Entonces
35     imprimir( numero, "es un número de Armstrong." )
36   SiNo
37     imprimir( numero, " no es un número de Armstrong." )
38   FinSi
39 FinAlgoritmo
```

Al ejecutar el algoritmo:

Primera ejecución:

```
Escriba el número a analizar: 1634

1634 es un número de Armstrong.
```

Segunda ejecución:

```
Escriba el número a analizar: 14321

14321 no es un número de Armstrong.
```

Explicación del algoritmo:

En esta solución se utilizan dos estructuras repetitivas `Mientras-FinMientras` independientes. Cuando se habla de estructuras independientes, se quiere decir que primero se ejecuta una completamente y luego se procede de igual manera con la siguiente.

La primera estructura repetitiva en esta solución, tiene la función de contar el número de cifras o dígitos que conforman al número que proporcione el usuario, de esta manera se determina el valor de la potencia (`contadorCifras`) a la cual se deben elevar cada uno de los dígitos. Para ello se usó el mismo procedimiento explicado en el Ejemplo 4.3.

La segunda instrucción `Mientras-FinMientras` del algoritmo, calcula la sumatoria de los dígitos elevados a la potencia `n` (`contadorCifras`).

Luego de que se ejecuten los dos ciclos, se encuentra una estructura de decisión `Si-FinSi` independiente a ellos, cuya función es determinar si el número suministrado es o no un Armstrong.

Una característica a destacar en esta solución, es que las dos estructuras repetitivas `Mientras-FinMientras` tienen instrucciones similares, en ambas se hace la separación de las cifras.

.:Ejemplo 4.5. *Diseñe un algoritmo que calcule el Máximo Común Divisor (MCD[5]) y el Mínimo Común Múltiplo (mcm[6]) de dos números enteros positivos, mediante el algoritmo de Euclides. El algoritmo debe permitir hacer varios cálculos hasta que el usuario decida que no desea continuar.*

Euclides, matemático griego del año 350 a.C, formuló el algoritmo que lleva su nombre y que permite encontrar el MCD y el mcm de dos números enteros, `a` *y* `b` *(*`a` $>$ `b`*), mediante los siguientes pasos:*

1. *Se divide* `a` *entre* `b` *y se obtiene el cociente* c_1 *y el resto* r_1*. Los resultados del cociente no se tendrán en cuenta en la solución del problema.*
2. *Si* r_1 *es diferente de 0, se divide* `b` *entre* r_1*, obteniéndose el cociente* c_2 *y el resto* r_2*.*
3. *Si* r_2 *es diferente de 0, se divide* r_1 *entre* r_2*, obteniéndose un nuevo cociente y un nuevo residuo.*
4. *Estos pasos se repiten mientras que* r_n *sea diferente de 0.*
5. *El resto anterior (*r_{n-1}*), es decir, el último divisor es el MCD de* `a` *y* `b`*.*

De igual manera se establece que:

$$mcm = \frac{a * b}{MCD}$$

Análisis del problema:

- **Resultados esperados:** de acuerdo al enunciado, el algoritmo debe informar el MCD y el mcm de dos números enteros.
- **Datos disponibles:** el usuario del algoritmo proporcionará dos valores enteros (`a` y `b`).
- **Proceso:** una vez se ingresen los dos números (`a` y `b`), se debe garantizar que se cumpla la condición que `a > b`, en caso contrario, se debe hacer el cambio de valores entre las dos variables (Ver Figura 4.12).

[5]El MCD es el mayor número que divide exactamente a dos o más números.

[6]El mcm es el número más pequeño, que no sea 0, que es múltiplo de dos o más números.

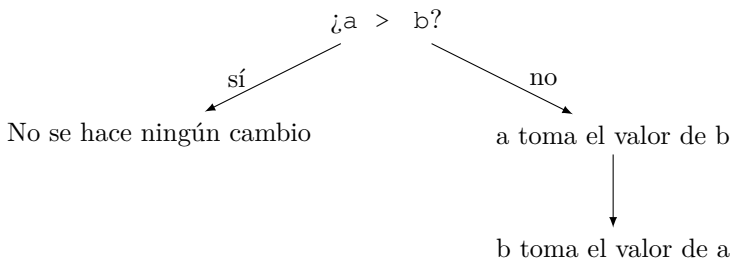


Figura 4.12: Árbol de decisión del Ejemplo 4.5

Mediante el siguiente ejemplo se analizará el procedimiento explicado en los pasos anteriores. Suponga que se desea encontrar el MCD y el mcm de 532 y 112.

1. Sea `a = 532` y `b = 112`. La primera operación a realizar es dividir `a` entre `b`, obteniendo un resto de 84 (r_1=`84`).

$$\frac{532}{112}$$

2. El primer resto (r_1) es diferente de 0, entonces se divide `b` entre `r1`, obteniendo un resto de 28 (r_2=`28`).

$$\frac{112}{84}$$

3. El segundo resto (r_2) es diferente de 0, se procede a una nueva división, r_1 entre r_2, obteniendo un resto de 0 (r_3=`0`).

$$\frac{84}{28}$$

4. El nuevo resto (r_3) ya no es diferente de 0, por lo tanto, no se hacen nuevas divisiones.
5. El resto anterior (r_{n-1}), es decir r_2 o el divisor de la última división realizada, cuyo valor es `28`, es el MCD.

Analizando las operaciones realizadas, se observa que, en cada repetición de la división, el divisor pasa a ser el nuevo dividendo y cada nuevo resto pasa a ser el nuevo divisor.

Una vez obtenido el MCD de ambos números, se puede calcular el mcm, aplicando la fórmula dada en el enunciado.

Se debe tener presente que el enunciado solicita que se puedan realizar varios cálculos, por lo tanto, el proceso iterativo del cálculo del MCD debe estar dentro de otra estructura repetitiva que permita realizar los cálculos hasta que el usuario determine que no desea continuar.

- **Variables requeridas:** para la solución de este problema se nombrarán algunas de las variables de acuerdo al concepto matemático y las otras siguiendo la recomendación de establecer nombres nemotécnicos.

  - `a` y `b`: números a los cuales se les hallará el MCD y el mcm.
  - `dividendo` y `divisor`: variables que se usarán para copiar el valor de a y b y realizar las operaciones correspondientes, de esta manera no se perderán los valores originales. La variable `divisor` almacenará el MCD en la última ejecución del ciclo.
  - `resto`: será utilizada para calcular el resto de la división entera. Este resto se convertirá en el divisor de la siguiente división que se efectúe. Adicionalmente controlará la repetición del ciclo donde se calculará el MCD.
  - `mcm`: en esta variable se calculará el mínimo común múltiplo de los dos números ingresados al algoritmo.
  - `seguir`: es una variable bandera o centinela que recibirá una respuesta del usuario, con relación a si quiere o no realizar un nuevo cálculo. Controlará la iteración del ciclo externo.

De acuerdo al análisis planteado, se propone el Algoritmo 4.8.

Algoritmo 4.8: Euclides

```
1  Algoritmo Euclides
2    /* Este algoritmo calcula el MCD de dos números, usando
3       el Algoritmo de Euclides. Adicionalmente informa el mcm
4     */
5
6    // Declaración de variables
7    Entero   a, b, dividendo, divisor, resto, mcm
8    Caracter seguir
9
10   // Se inicializa la variable centinela
11   seguir = 'S'
12
13   // Se condiciona el ciclo externo que controlará
14   // la cantidad de cálculos que se quieran hacer
```

```
15   Mientras( seguir == 'S' O seguir == 's' )  // Ciclo
        externo
16     imprimir( "Ingrese el primer número: " )
17     leer( a )
18     imprimir ( "Ingrese el segundo número: " )
19     leer( b )
20
21     // Se debe garantizar que el valor de a sea mayor
22     // que el valor de b
23     Si( a > b ) Entonces
24        dividendo = a
25        divisor   = b
26     SiNo
27        dividendo = b
28        divisor   = a
29     FinSi
30
31     resto = dividendo % divisor
32
33     // Se realizan las diferentes divisiones
34     Mientras( resto != 0 ) // Ciclo interno
35       dividendo = divisor
36       divisor   = resto
37       resto     = dividendo % divisor
38     FinMientras
39
40     mcm = (a * b) / divisor
41
42     // Resultados esperados
43     imprimir( "El máximo común divisor de: ", a, " y ", b )
44     imprimir( " es: ", divisor )
45     imprimir( "El mínimo común múltiplo es: ", mcm )
46
47     // Se controla la ejecución de nuevos cálculos
48     imprimir( "Desea realizar nuevos cálculos [S] o [N]?:" )
49     leer( seguir )
50   FinMientras
51 FinAlgoritmo
```

Al ejecutar el algoritmo:

```
Ingrese el primer número: 532
Ingrese el segundo número: 112

El máximo común divisor de:  532 y 112 es: 28
El mínimo común múltiplo es: 2128

Desea realizar nuevos cálculos [S] o [N]?: N
```

Explicación del algoritmo:

Aclaración:



Antes de entrar en detalle sobre las particularidades de este algoritmo, se explicará el funcionamiento de los ciclos anidados.

Los **ciclos anidados**, son estructuras repetitivas que se encuentran unas dentro de otras. El ciclo que está contenido dentro de otro, generalmente se le denomina ciclo interno y al ciclo contenedor se le da el nombre del ciclo externo. Se puede anidar cualquier cantidad de ciclos (Ver Figura 4.13).

La forma de operar es muy sencilla. En la Figura 4.13 se aprecia la Condición 1, que marca el inicio del ciclo externo. Mientras esta condición sea verdadera, se ejecutará su cuerpo de ciclo que consta de otras instrucciones, incluyendo al ciclo interno; que está controlado por la Condición 2. Mientras esta Condición 2 sea verdadera, se ejecutará completamente el cuerpo del ciclo interno. Cuando esta condición se haga falsa, el control lo vuelve a tomar el ciclo externo y continúa con el resto de sus instrucciones; al encontrar el fin de su cuerpo de ciclo se regresa a Condición 1 para evaluar si aún es verdadera y volver a iterar completamente, eso incluye una nueva ejecución del ciclo interno, siempre y cuando la Condición 2 sea verdadera.

Estas iteraciones se ejecutan mientras la condición del ciclo externo sea verdadera.

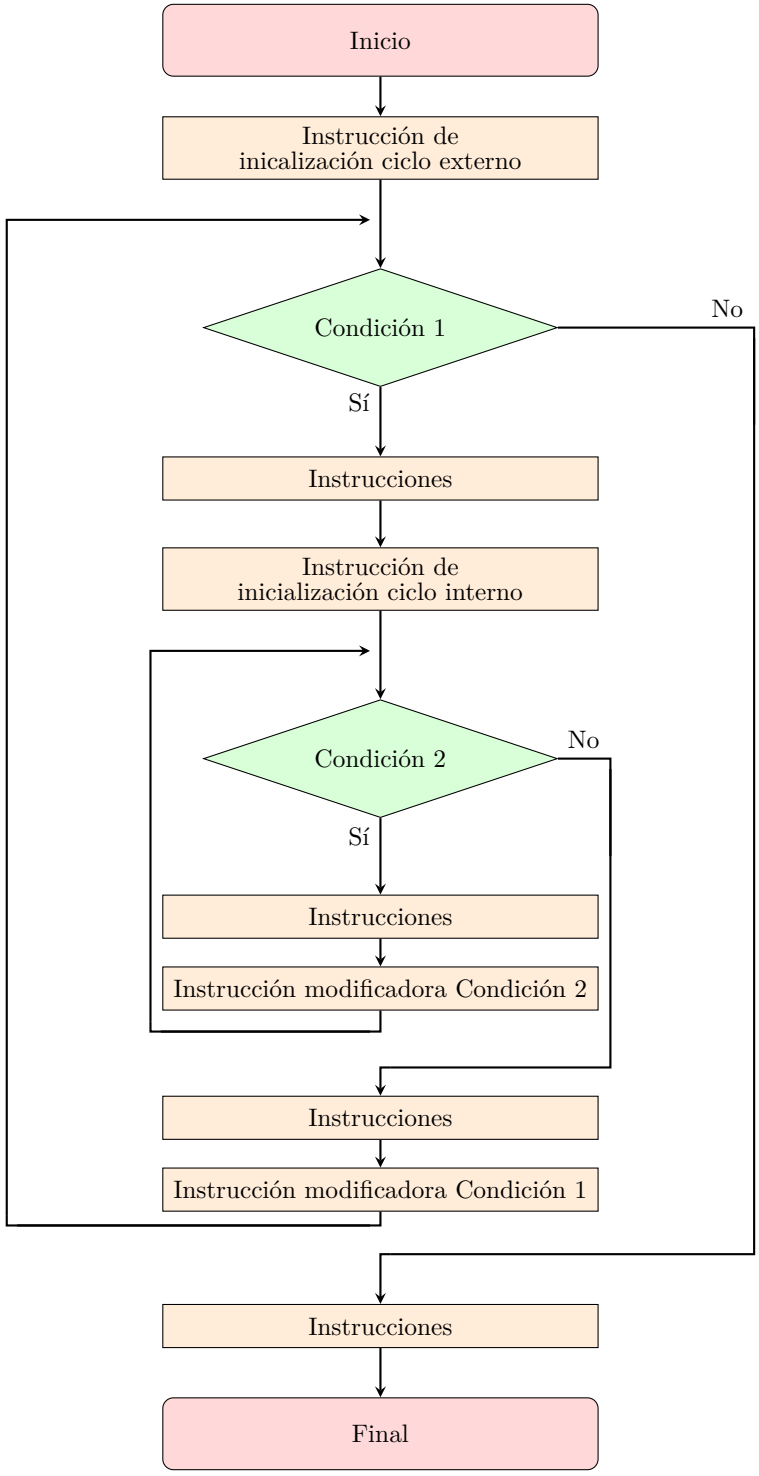


Figura 4.13: Ciclos Anidados

Una vez hecha la aclaración del funcionamiento de los ciclos anidados, se continuará con la explicación del Ejemplo 4.5.

En la solución planteada en el Algoritmo 4.8, en pseudocódigo, se poseen dos estructuras repetitivas `Mientras` - `FinMientras` anidadas.

El cuerpo del ciclo externo, está comprendido entre las líneas 15 y 50, el interno comprende desde la línea 34 hasta la 38.

Primero se ejecuta parcialmente el ciclo externo (desde la línea 16 a la línea 33), luego pasa el control al ciclo interno que se ejecuta mientras su condición sea verdadera (línea 34).

Cuando la condición del ciclo interno se haga falsa, el control lo vuelva a asumir el ciclo externo (líneas 39 a la 49). Al encontrar la instrucción `FinMientras` (línea 50) se regresa a evaluar su condición (línea 15), si el resultado es verdadero se ingresa nuevamente y se ejecuta una vez más todos los pasos, incluyendo el ciclo interno. Este proceso se repite mientras las dos condiciones de las estructuras `Mientras` sean verdaderas.

En este algoritmo la estructura externa controla si se repite o no todo el proceso, para ello se utiliza la variable `seguir` que toma un valor inicial de `'S'`, con el propósito de que al evaluar por primera vez la condición `Mientras` (línea 15) se obtenga un resultado verdadero y se pueda ejecutar el cuerpo de este ciclo.

Finalizando el cuerpo del ciclo externo, se encuentra la instrucción modificadora de condición, compuesta por las líneas 48 y 49.

Teniendo en cuenta que la respuesta puede darse en mayúscula o minúscula, la condición del `Mientras` contempla estas alternativas (línea 15).

Mientras la respuesta sea afirmativa se solicitarán dos nuevos valores para a y b, repitiéndose todo el proceso del cálculo del MCD y el mcm. La ejecución del algoritmo terminará cuando la respuesta a la anterior pregunta sea negativa.

La segunda estructura `Mientras` - `FinMientras` (ciclo interno), tiene como finalidad realizar las diferentes divisiones mientras la variable resto tenga un valor diferente de 0. En las Figuras 4.14 y 4.15 se muestra la solución del Ejemplo 4.5 mediante un diagrama de flujo.

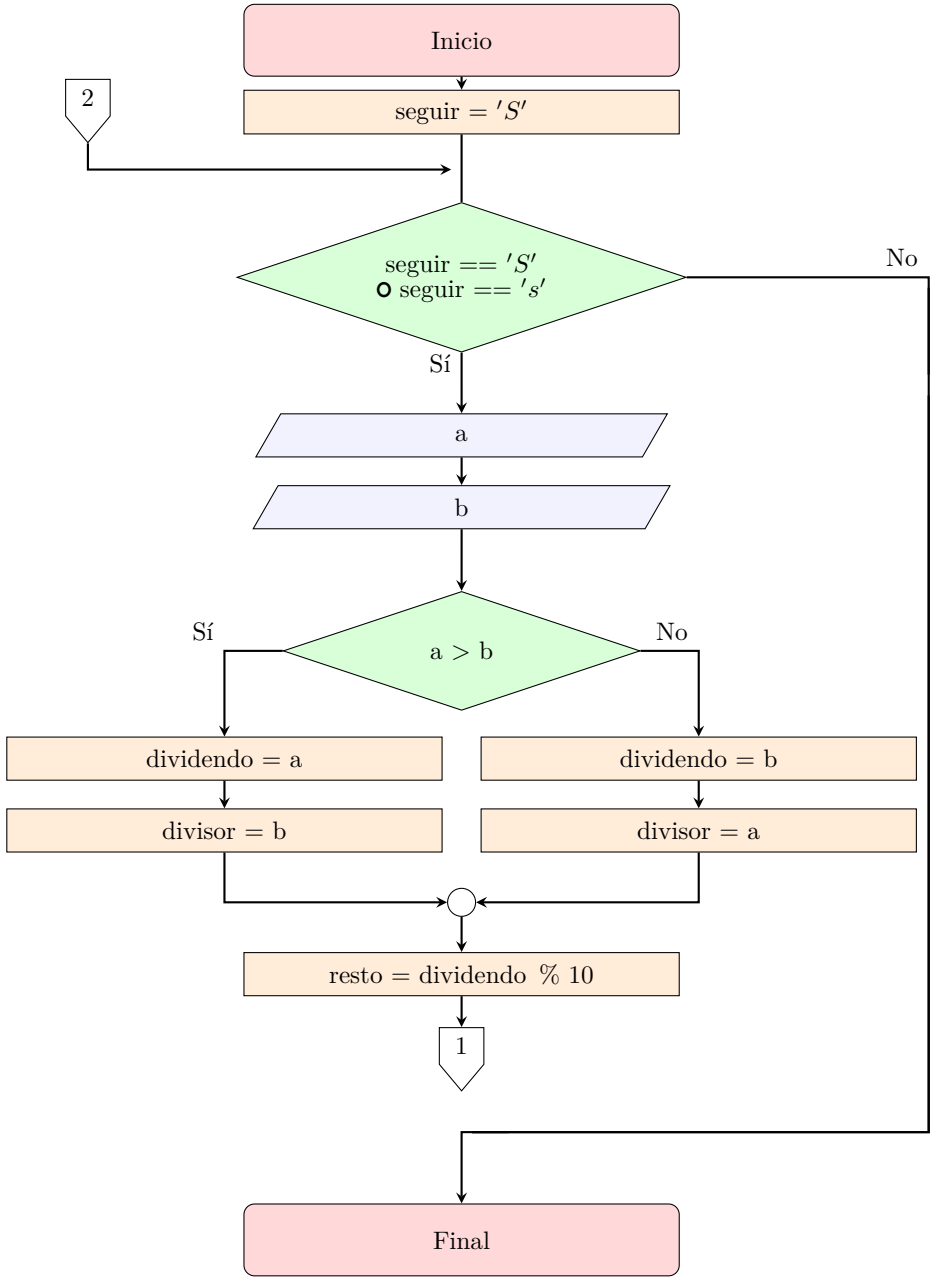


Figura 4.14: Diagrama del flujo del Algoritmo `Euclides` - Parte 1

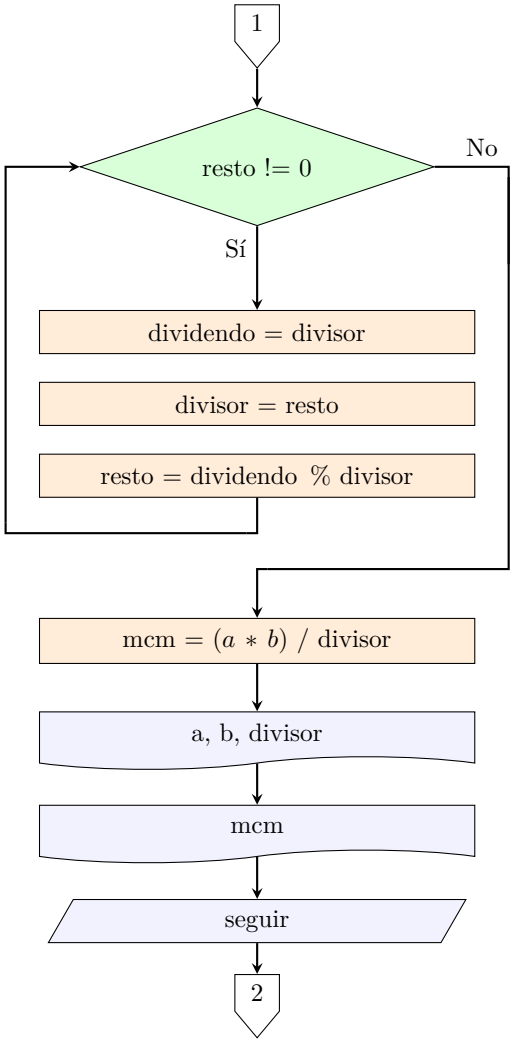


Figura 4.15: Diagrama del flujo del Algoritmo `Euclides` - Parte 2

Aclaración:



En las estructuras repetitivas anidadas, hasta que la ejecución de la estructura interna no se termine, el control no pasará a la estructura externa.

4.2.1 Prueba de escritorio

Tal como ese explicó en el Capítulo 2, una prueba de escritorio o tabla de verificación es un seguimiento que se hace de forma manual a un algoritmo para comprobar su funcionamiento.

Los siguientes 2 ejemplos son enunciados clásicos de programación; en este libro fueron diseñados utilizando el ciclo `Mientras` y serán utilizados para mostrar la aplicación de pruebas de escritorio.

.:Ejemplo 4.6. *Dado el siguiente algoritmo, que multiplica dos números enteros positivos mediante sumas sucesivas, realice una prueba de escritorio o tabla de verificación.*

Algoritmo 4.9: Multiplicacion

```
1  Algoritmo Multiplicacion
2    /* Multiplica dos números enteros positivos mediante sumas
3       sucesivas
4     */
5    // Declaración de variables
6    Entero multiplicando, multiplicador, producto, contador
7
8    // Datos disponibles
9    imprimir( "Ingrese el multiplicando: " )
10   leer( multiplicando)
11   imprimir( "Ingrese el multiplicador: " )
12   leer( multiplicador )
13
14   // Inicialización de acumulador y contador
15   producto = 0
16   contador = 0
17
18   // Multiplicación a través de sumas sucesivas
19   Mientras( contador < multiplicador )
20      producto = producto + multiplicando
21      contador = contador + 1
22   FinMientras
23
```

```
24   // Resultado esperado
25   imprimir( "El producto es: ", producto )
26 FinAlgoritmo
```

Al ejecutar el algoritmo:

```
Ingrese el multiplicando: 75
Ingrese el multiplicador: 6
El producto es: 450
```

Aplicación de la prueba de escritorio:

El algoritmo solicita como datos de entrada el multiplicando y el multiplicador. Para realizar la prueba, se van a tomar dos valores de manera aleatoria. Generalmente se debe trabajar con datos que sean fáciles de procesar. En el presente ejemplo, los valores para las dos variables se presentan en la Tabla 4.3.

multiplicando = 75
multiplicador = 6

producto	contador	contador< multiplicador
0	0	$0 < 6$ (V)
75	1	$1 < 6$ (V)
150	2	$2 < 6$ (V)
225	3	$3 < 6$ (V)
300	4	$4 < 6$ (V)
375	5	$5 < 6$ (V)
450	6	$6 < 6$ (F)

producto = 450

Tabla 4.3: Prueba de escritorio - Algoritmo 4.6

Explicación de la prueba de escritorio:

Para el desarrollo de una prueba de escritorio de un algoritmo que utiliza estructuras de repetición, es indispensable la creación de una tabla, llamada, "Tabla de verificación", donde se muestre la forma en que las diferentemente variables que se encuentran dentro del ciclo cambian en cada una de las iteraciones. Las demás variables son mostradas antes de la tabla (valores iniciales), o después de ella, cuando se trate por ejemplo de valores de salida. La tabla de verificación suele tener columnas asociadas a las expresiones lógicas / relacionales que ayuden al seguimiento del funcionamiento del algoritmo.

En este ejemplo, las variables `multiplicando`, `multiplicador`, `producto` y `contador`, inician con los siguientes contenidos 75, 6, 0 y 0 respectivamente.

Se evalúa la condición del `Mientras` (línea 19), en este caso, $0 < 6$, obteniendo un resultado verdadero, lo cual indica que se debe ejecutar el cuerpo del ciclo.

Al ejecutar el cuerpo del ciclo se procede a realizar la operación `producto = producto + multiplicando`, de esta forma se van efectuando las sumas sucesivas del multiplicando. La variable `producto` aumenta su valor en 75 en cada iteración.

La siguiente instrucción incrementa la variable `contador`:

```
21 contador = contador + 1
```

Seguidamente se ejecuta el `FinMientras`, retornando la secuencia de operación del algoritmo al código `Mientras`, para que una vez más se evalúe la condición.

Este proceso se repite mientras `contador` tenga un valor menor al de `multiplicador`. Cuando la expresión de la línea 19 sea $6 < 6$, el resultado será falso y se procederá a ejecutar la instrucción que hay después del `FinMientras`, es decir, imprime el resultado de la multiplicación (`producto`).

Buena práctica:



Cada uno de los resultados obtenidos de las operaciones y de evaluaciones realizadas a las condiciones, deben quedar registrados en la tabla de verificación.

Para realizar una prueba de escritorio, deben asignarse a las variables valores fáciles de procesar.

.:Ejemplo 4.7. *El presente ejemplo muestra un diagrama de flujo (Figura 4.16) que calcula una potencia entera mediante sumas sucesivas.*

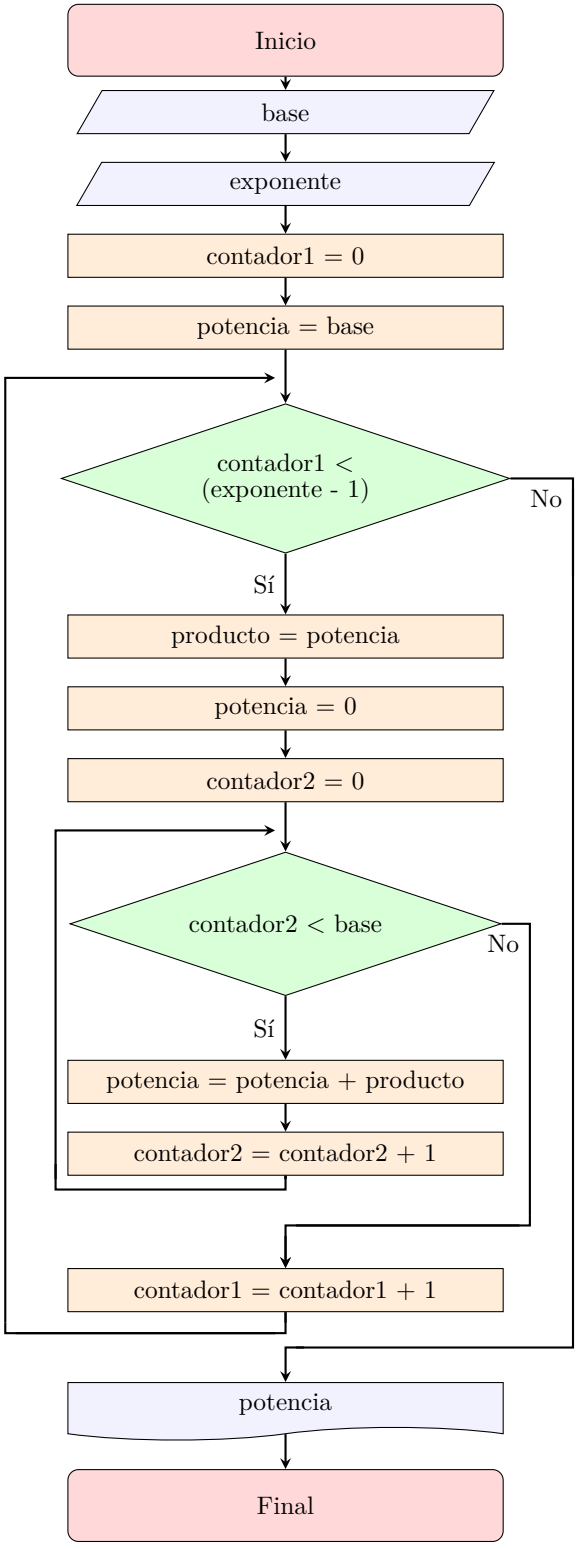


Figura 4.16: Diagrama de flujo del Algoritmo `PotenciaEntera`

Para realizar la prueba de escritorio se calculará la siguiente potencia `5^3` *que es equivalente a* 5^3.

Tenga presente el nombre de los términos en una operación de potencia:

$$potencia = base^{exponente}$$

`base = 5`
`exponente = 3`

contador1	**potencia**	**contador1 <= exponente-1**	**producto**	**contador2**	**contador2 <= base**
0	5	0 < 2 (V)	5		
	0			0	0 < 5 (V)
	5			1	1 < 5 (V)
	10			2	2 < 5 (V)
	15			3	3 < 5 (V)
	20			4	4 < 5 (V)
	25			5	5 < 5 (F)
1		1 < 2 (V)	25		
	0			0	0 < 5 (V)
	25			1	1 < 5 (V)
	50			2	2 < 5 (V)
	75			3	3 < 5 (V)
	100			4	4 < 5 (V)
	125			5	5 < 5 (F)
2		2 < 2 (F)			

`potencia = 125`

Tabla 4.4: Prueba de escritorio - Algoritmo 4.7

Explicación de la prueba de escritorio:

Las variables `base` y `exponente` se inicializan en 5 y 3, respectivamente (5^3). La variable `contador1` inicia en 0 y `potencia` asume el valor de base.

Este diagrama de flujo trabaja con ciclos anidados. El ciclo externo está controlado por la condición `contador1<(exponente - 1)` el ciclo interno presenta la condición `contador2<base`. En la tabla se aprecian ambas condiciones.

Se evalúa la condición del ciclo externo:

```
contador1 < ( exponente - 1 )
```

En la primera evaluación `contador1` vale 0 y `exponente` 3, por lo tanto, la condición es verdadera (`0 <2`). Se ejecuta la primera parte del cuerpo de este ciclo:

A la variable `producto` se le asigna el valor de potencia (5), `potencia` y `contador2` almacenan un 0 como valor. Estos datos deben quedar registrados en la tabla de verificación.

Seguidamente se encuentra el ciclo interno (Ver Figura 4.17).

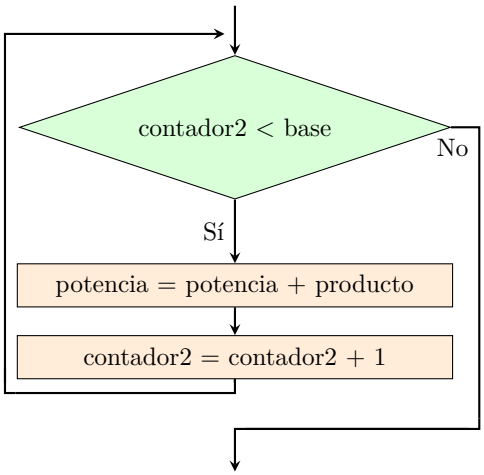


Figura 4.17: Ciclo interno para el Ejemplo 4.7

Se evalúa su condición `contador2 < base`, obteniéndose un resultado verdadero en su primera evaluación, (`0 < 5`). Este resultado se consigna en la tabla.

A continuación, como resultado de la evaluación anterior, se ejecutan las dos instrucciones de este ciclo; la primera aumenta el valor de `potencia` en el valor que tenga almacenado `producto`, la segunda incrementa `contador2` en una unidad. Luego de ejecutar estas dos operaciones el control regresa a la evaluación de la condición, tal como se indica en la anterior figura y en la tabla de verificación.

Para este ejemplo, estas instrucciones iteran 5 veces (valor de la `base`). En la columna que corresponde a la variable `potencia`, se registran los valores que va almacenando a medida que se incrementa, en la primera iteración toma un valor de 5, para su quinta iteración tendrá un resultado de 25. La variable `contador2` va aumentando su valor, en la primera vuelta toma el valor de 1, finalizando en 5.

Cuando contador2 almacena el número 5, la condición del ciclo interno se hace falsa; por consiguiente, se termina su ejecución y el control pasa de nuevo al ciclo externo. Una vez allí se aumenta a 1 el valor de la variable contador1.

Después del incremento, se valora por segunda vez la condición del ciclo externo:

```
contador1< ( exponente - 1 )
```

De acuerdo a los valores almacenados en este momento, el resultado es verdadero (1 < 2), tal como se evidencia en la tabla de verificación.

El cuerpo del ciclo externo vuelve a ejecutarse; la variable producto toma el valor de potencia (25), luego potencia y contador2 vuelven a tomar el valor de 0. Por segunda vez el ciclo interno vuelve a iterar otras 5 veces. La variable potencia en la primera vuelta del ciclo, toma un valor de 25 y en su quinta iteración valdrá 125; contador2 toma el valor de 1 y terminará en 5 al final de las iteraciones. Observe la tabla de verificación para ver el registro de estos valores.

Cuando la variable contador2 alcance el valor de 5, la condición del ciclo interno pasará a ser falsa, terminando su ejecución.

De nuevo en el ciclo externo contador1 incrementa su valor a 2. Al evaluar por tercera vez la condición contador1<(exponente - 1), se obtiene un resultado falso, dando así por terminadas las iteraciones del ciclo externo y por consiguiente las del interno. Ahora se procede a imprimir el contenido de potencia (125), para luego finalizar la ejecución del diagrama de flujo.

4.3. Estructura Haga - MientrasQue

Esta estructura de repetición, permite que una instrucción o un conjunto de ellas se ejecuten una o más veces.

El Haga-MientrasQue es un ciclo condicionado al final, lo cual garantiza que sus instrucciones se ejecuten por lo menos una vez.

La forma general de esta estructura de repetición es presentada en el segmento de Algoritmo 4.10.

Algoritmo 4.10: Forma general - Haga-MientrasQue

```
1 Instrucción de inicialización
2 Haga
3   Instrucción-1
4   Instrucción-2
5     ...                    /* Cuerpo del ciclo */
6   Instrucción-n
7   Instrucción modificadora de condición
8 MientrasQue( condición )
9 Instrucción externa
```

Esta forma general se interpreta así:

La Instrucción de inicialización se usa para dar un valor inicial a las variables que harán el papel de contadores o acumuladores dentro del ciclo. En el caso de no tener este tipo de variables, no es necesario su uso. Las variables que intervienen en la condición del `MientrasQue` no necesariamente deben ser inicializadas antes de entrar al ciclo, ya que su valor puede ser asignado dentro del cuerpo del mismo.

Al encontrar la instrucción `Haga`, se ejecutan las instrucciones que conforman el cuerpo del ciclo hasta encontrar la instrucción `MientrasQue (condición)`; si al evaluar la condición, esta es verdadera se regresa hasta el código `Haga` y una vez más se repite la ejecución del cuerpo del ciclo. La condición estará representada por una expresión relacional o lógica.

Las iteraciones terminarán en el momento que la evaluación de la condición produzca un resultado falso, en cuyo caso el control del algoritmo lo asume la Instrucción externa, es decir, la que esté escrita debajo del `MientrasQue`, la cual no hace parte del ciclo.

Al igual que en el `Mientras-FinMientras`, en esta forma general también está presente la Instrucción modificadora de condición (línea 7), cuyo propósito es cambiar el estado de la condición. De omitir la instrucción modificadora, se obtendrá lo que se conoce como un ciclo infinito, debido a que su ejecución "nunca termina". Aunque en la forma general está representada de última en la secuencia de instrucciones que componen el cuerpo del ciclo, no necesariamente debe ocupar ese lugar.

En la Figura 4.18 se muestra la estructura de repetición `Haga - MientrasQue` mediante un diagrama de flujo.

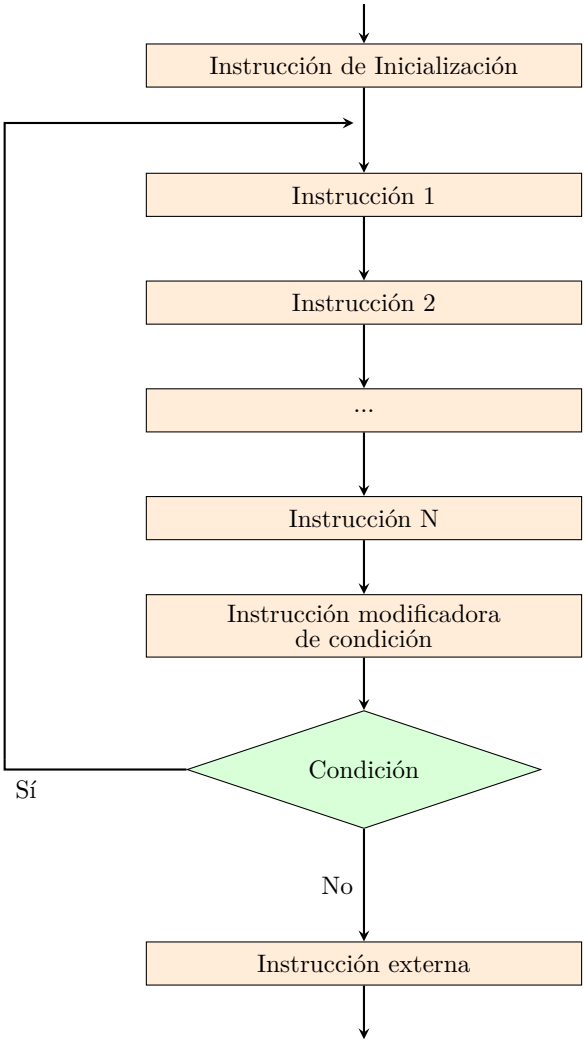


Figura 4.18: Estructura de repetición `Haga - MientrasQue`

Aclaración:



Como lo expresa [Trejos, 2017] la forma general del ciclo `Haga-MientrasQue` podría decirse que es una inversión de la estructura del ciclo `Mientras-FinMientras`.

Para explicar el funcionamiento del ciclo `Haga - MientrasQue` se hará uso del mismo ejemplo empleado con la estructura `Mientras - FinMientras`; de esta forma usted podrá observar su diferencia.

.:Ejemplo 4.8. *Crear un algoritmo que imprima los números del 1 al 10, utilizando el ciclo* `Haga-MientrasQue`.

Algoritmo 4.11: Algoritmo que imprime los números del 1 al 10 - Versión 1

```
1 numero = 1
2 Haga
3   imprimir( numero )
4   numero = numero + 1
5 MientrasQue( numero <= 10 )
```

Para una explicación del segmento del Algoritmo 4.11, se identificarán sus partes de acuerdo a la forma general expresada en párrafos anteriores.

Línea	Explicación
1	Instrucción de inicialización
2	Inicio del ciclo
3	Cuerpo del ciclo
4	Cuerpo del ciclo e instrucción modificadora de condición
5	Fin del ciclo y condición. Regresa el control a la línea 2.

Tabla 4.5: Explicación Algoritmos 4.8

En la primera línea, la variable `numero` se inicializa en 1. Pasando a la línea 2 se encuentra la instrucción `Haga`, que indica el inicio de un ciclo condicionado al final. El cuerpo del ciclo está conformado por dos instrucciones (líneas 3 y 4), que se ejecutan por lo menos la primera vez. Al encontrar la condición `MientrasQue (numero <= 10)` se hace el testeo, si el resultado es verdadero el control regresa a la instrucción `Haga` (línea 2) y se vuelve a ejecutar el cuerpo del ciclo. Las iteraciones del ciclo terminan cuando la condición resulte falsa.

La instrucción `imprimir` se ejecuta 10 veces, imprimiendo uno a uno los números del 1 al 10.

La operación `numero = numero + 1` además de incrementar, en cada iteración, en 1 el valor de la variable `numero`, también hace el papel de Instrucción modificadora de condición. Cuando la variable tome el valor de 11, la condición `(numero <= 10)` entregará un resultado falso, con lo cual se dará terminada la ejecución del `Haga-MientrasQue`.

Tenga en cuenta que cada problema puede tener múltiples soluciones, por ejemplo, el segmento del Algoritmo 4.12 también imprime los números del 1 al 10 usando el ciclo Haga-MientrasQue:

Algoritmo 4.12: Algoritmo que imprime los números del 1 al 10 - Versión 2

```
1 numero = 0
2 Haga
3    numero = numero + 1
4    imprimir( numero )
5 MientrasQue( numero < 10 )
```

Comparando la versión 2 (Algoritmo 4.12) con la versión 1 (Algoritmo 4.11) se puede observar que:

Línea 1: la inicialización es diferente, se hizo en 0.

Línea 2: en ambas versiones es igual, se indica el inicio del ciclo condicionado al final.

Líneas 3 y 4: se codificaron las expresiones `numero = numero + 1` e `imprimir(numero)`, que son iguales a la planteadas en las líneas 4 y 3 del Algoritmo versión 1, respectivamente. En esta solución fue necesario cambiar el orden de ejecución de las instrucciones del cuerpo del ciclo, ya que si se conservaban en el orden como habían sido escritas en la versión 1 del algoritmo, el resultado sería la impresión de los números del 0 al 9 y no del 1 al 10 como se esperaría.

Línea 5: en la nueva versión el ciclo se ejecuta mientras el valor de `numero` sea menor a 10, en lugar de menor o igual a 10, como se planteó en la versión anterior. Esto obedece a que la inicialización fue en 0 en lugar de 1. Recuerde que en la versión 1 del algoritmo la variable `numero` alcanza a tomar el valor de 11, en esta nueva versión el valor llega hasta 10.

Aclaración:



El bloque de instrucciones o cuerpo del ciclo Haga-MientrasQue, se ejecutará por lo menos una vez.

Estos dos segmentos de algoritmos se pueden representar gráficamente con los diagramas de flujo de la Figura 4.19.

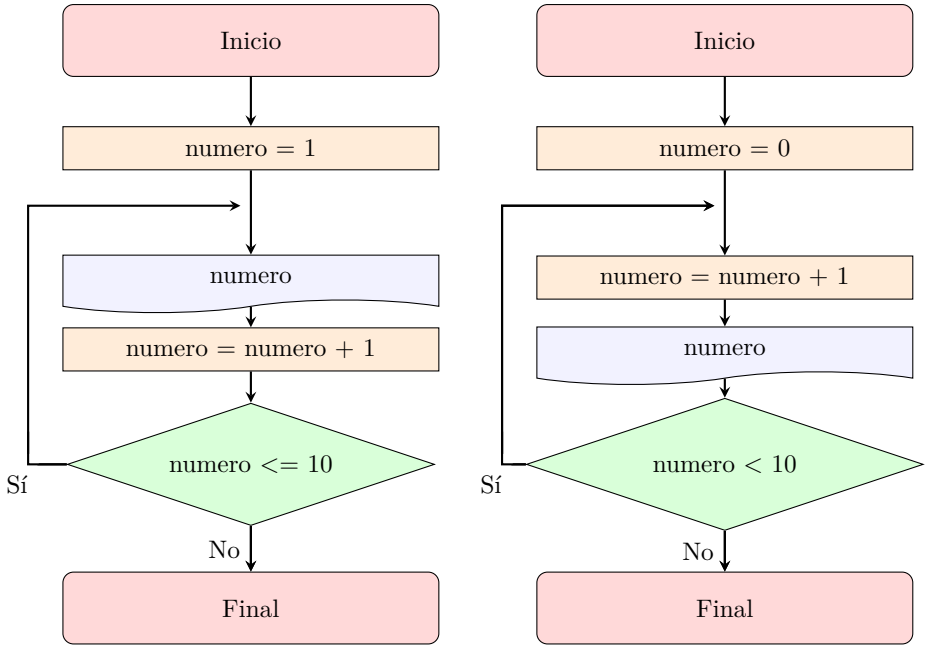


Figura 4.19: Imprimen los números del 1 al 10 (Haga-MientrasQue)

A continuación, se darán algunos enunciados como ejemplo para ser resueltos mediante la estructura repetitiva Haga - MientrasQue.

.:Ejemplo 4.9. *Diseñe un algoritmo que genere e imprima la siguiente serie:* $1, 3, 5, 7, 9, 11, \ldots, n$

Análisis del problema:

El análisis completo a este problema lo encuentra en el Ejemplo 4.1, allí fue solucionado usando la estructura Mientras-FinMientras. En este momento se procederá a mostrar la solución usando el ciclo Haga-MientrasQue.

Algoritmo 4.13: Serie

```
1 Algoritmo Serie
2   // Declaración de variables
3   Entero contadorNumeros, cantidadTerminos, termino
4
5   // Dato disponible
6   imprimir( "Ingrese la cantidad de términos a generar: " )
7   leer( cantidadTerminos )
```

```
8
9   // Inicialización de variables
10  contadorNumeros = 0
11  termino = 1
12
13  // Generación de la serie
14  Haga
15    imprimir( termino, ", " )
16    termino = termino + 2
17    contadorNumeros = contadorNumeros + 1
18  MientrasQue( contadorNumeros < cantidadTerminos - 1 )
19
20  imprimir( termino )
21 FinAlgoritmo
```

Al ejecutar el algoritmo:

```
Ingrese la cantidad de términos a generar: 15
1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29
```

Explicación del algoritmo:

La única diferencia entre esta solución y la planteada en el Algoritmo 4.4 radica en el ciclo utilizado. En el Algoritmo 4.4 se usó el ciclo `Mientras-FinMientras` cuya condición se planteó al inicio del ciclo, en este algoritmo el ciclo utilizado es el `Haga-MientrasQue`, con la condición al final; todas las demás instrucciones son exactamente iguales y producen el mismo resultado.

Aclaración:



Tanto el ciclo `Mientras-FinMientras` y el `Haga-MientrasQue` se ejecutan mientras la evaluación de la condición arroje un resultado verdadero.

El `Mientras-FinMientras` evalúa la condición al comienzo, el `Haga-MientrasQue` la evalúa al final.

En la Figura 4.20 se muestra la solución del Ejemplo 4.8 mediante un diagrama de flujo.

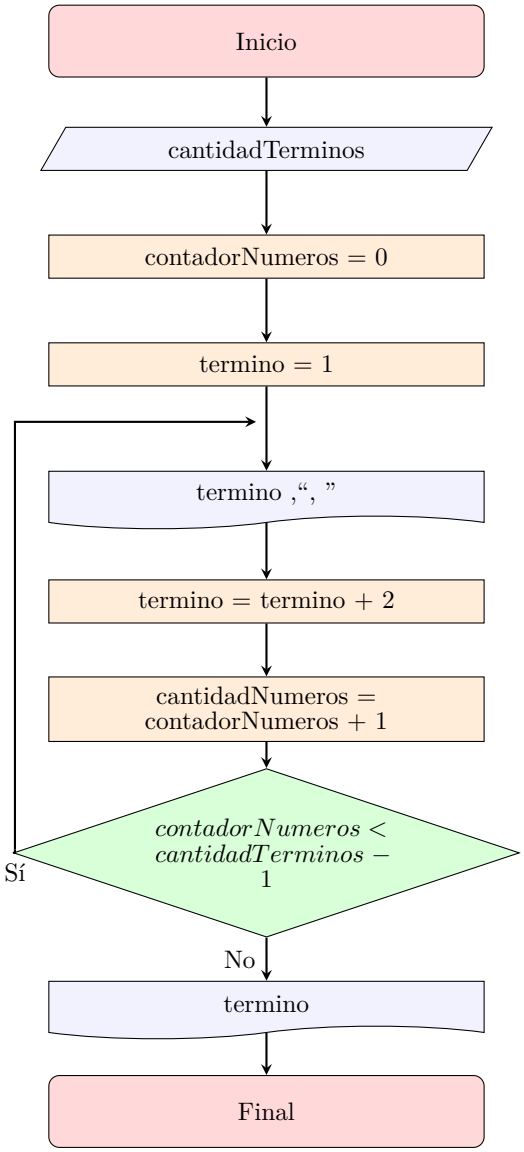


Figura 4.20: Diagrama de flujo del Algoritmo `Serie`

.:Ejemplo 4.10. *El profesor de la materia de Programación para dispositivos móviles, desea hacer una encuesta con sus estudiantes para determinar sobre cuál de dos posibles plataformas desarrollará las temáticas de este espacio académico. Las plataformas disponibles son Android e iOS, en caso de elegir una diferente se debe informar la situación y no será tenida en cuenta en los resultados. El profesor elegirá la plataforma de mayor votación; si se presenta un empate en la cantidad de votos, se usará otro mecanismo de elección.*

Cada estudiante deberá digitar su código y su elección por una de las dos plataformas.

Análisis del problema:

- **Resultados esperados:** informe sobre cuál de las dos plataformas obtuvo mayor votación.

  En el caso que el estudiante elija una opción diferente a estas plataformas se debe informar mediante un mensaje.

  Aunque en el enunciado no se pide, se informarán los votos obtenidos por cada una de las plataformas. Es una buena práctica informar algunos resultados que sean relevantes en el problema.

- **Datos disponibles:** de cada estudiante se conoce el código y la elección de la plataforma.

  En vista a que el algoritmo debe ejecutarse un número indeterminado de veces, se formulará una pregunta para saber si desea continuar o terminar con la votación. Consecuente a esto, un dato adicional que estará disponible, es la respuesta a esta pregunta.

- **Proceso:** dentro de un proceso repetitivo se debe solicitar el código y la elección de plataforma del estudiante. Luego se debe incrementar el contador de votos de acuerdo a la plataforma elegida (Ver Figura 4.21).

  Para repetir o terminar el proceso cíclico se planteará una pregunta, que debe ser contestada por el usuario.

  Una vez se termine la votación, se debe proceder a informar la cantidad de votos por cada una de las plataformas y tomar la decisión de cuál fue la de mayor votación; hay que tener en cuenta que existe la posibilidad de un empate en el número de votos (Ver Figura 4.22).

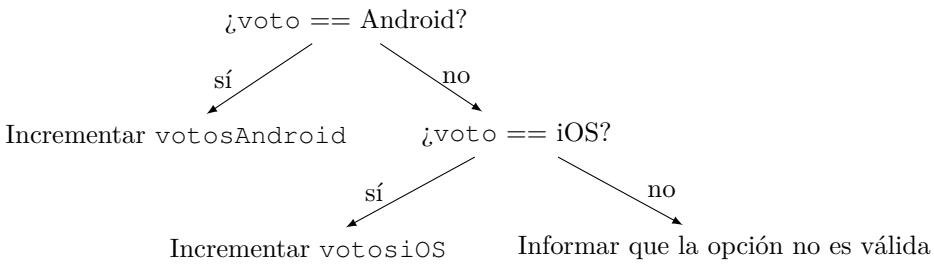


Figura 4.21: Árbol de decisión del Ejemplo 4.10 - Parte 1

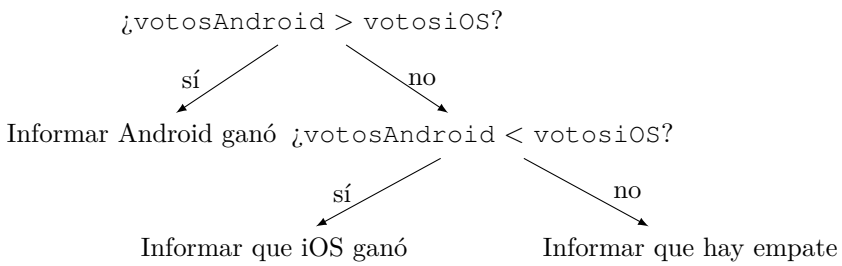


Figura 4.22: Árbol de decisión del Ejemplo 4.10 - Parte 2

- **Variables requeridas:**
  - `codigo`: almacenará el código del estudiante.
  - `voto`: es la opción que el estudiante elija entre las dos plataformas.
  - `votosAndroid`: contador de los votos para esta plataforma.
  - `votosiOS`: contador de votos para la plataforma iOS.
  - `seguir`: variable para controlar si se desea ingresar un nuevo voto.

En las Figuras 4.23 y 4.24 se muestra la solución del Ejemplo 4.4 mediante un diagrama de flujo.

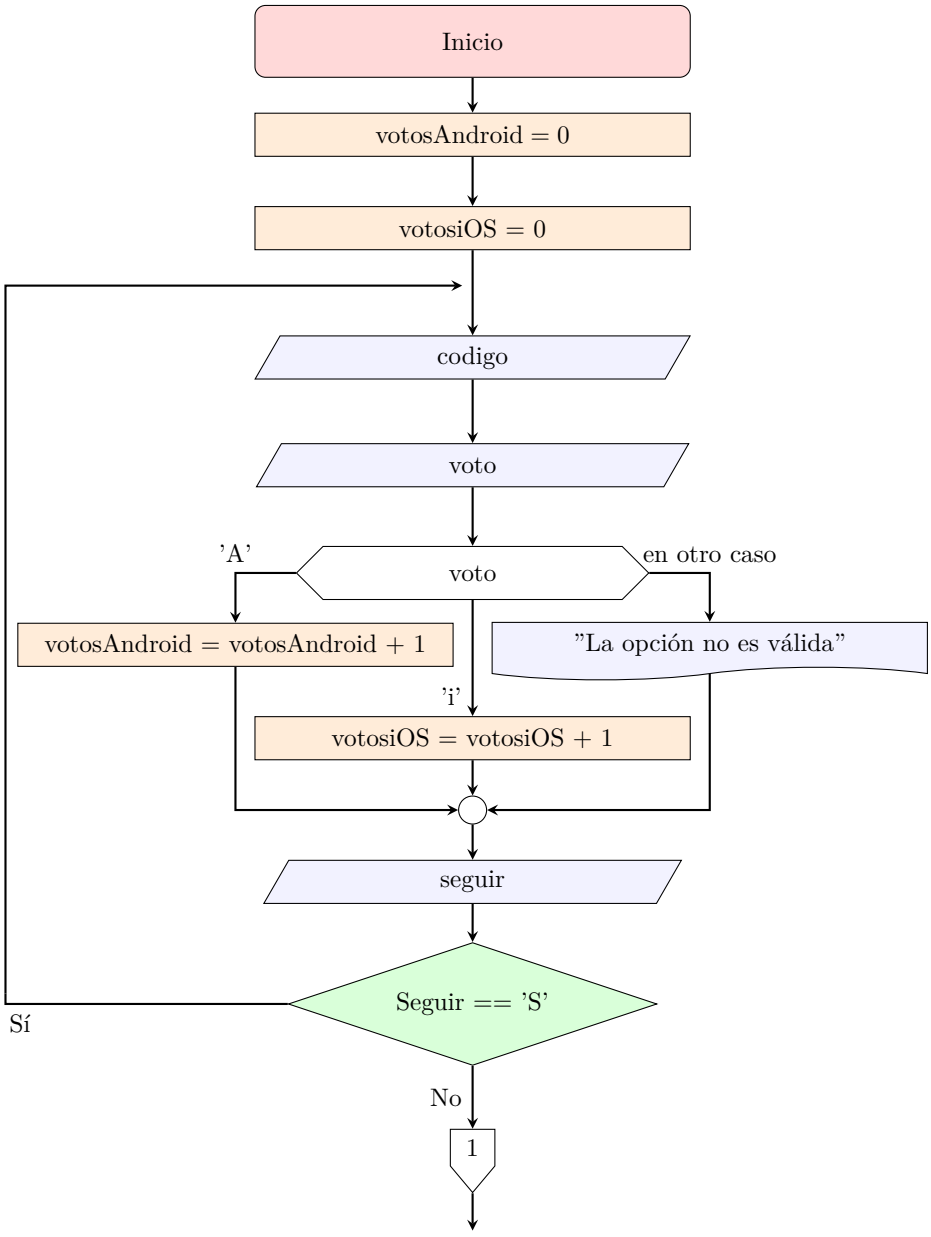


Figura 4.23: Diagrama de flujo del Algoritmo `Plataformas` - Parte 1

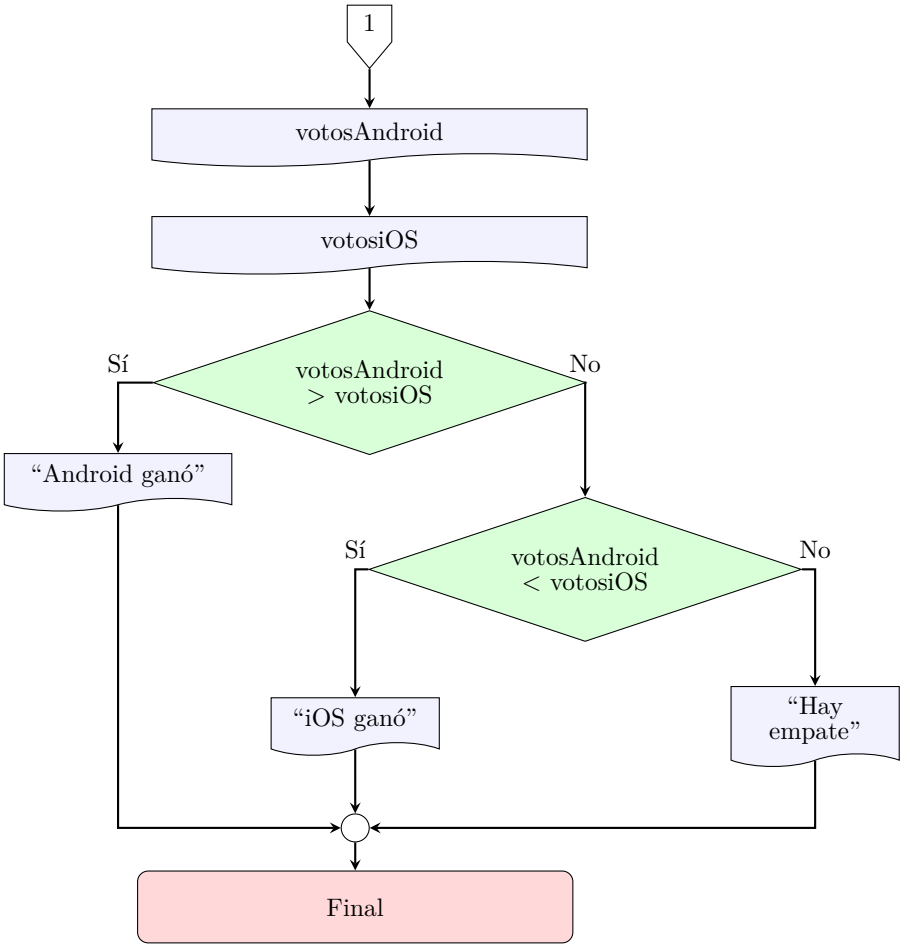


Figura 4.24: Diagrama de flujo del Algoritmo `Plataformas` - Parte 2

La solución en pseudocódigo se presenta en el Algoritmo 4.14.

Algoritmo 4.14: Plataformas

```
1  Algoritmo Plataformas
2    // Declaración de variables
3    Caracter seguir, voto
4    Cadena   codigo
5    Entero   votosAndroid, votosiOS
6
7    // Se inicializan los contadores en 0
8    votosAndroid = 0
9    votosiOS     = 0
10
```

```
11    // Proceso de solicitud y conteo de votos
12    Haga
13       imprimir( "Ingrese el código del estudiante: " )
14       leer( codigo )
15
16       imprimir( "PLATAFORMAS DISPONIBLES" )
17       imprimir( "[A]Android" )
18       imprimir( "[i]OS" )
19       imprimir( "Elija su opción: " )
20       leer( voto )
21
22       Segun( voto )
23         Caso 'A':
24         Caso 'a': votosAndroid = votosAndroid + 1
25                   FinCaso
26         Caso 'I':
27         Caso 'i': votosiOS = votosiOS + 1
28                   FinCaso
29         EnOtroCaso: imprimir( "La opción no es válida" )
30       FinSegun
31
32       imprimir( "Desea realizar un nuevo voto [S] [N]?: " )
33       leer( seguir )
34    MientrasQue ( seguir == 'S' O seguir == 's' )
35
36    // Se informa la cantidad de votos para cada plataforma
37    imprimir( "Votos por Android:", votosAndroid )
38    imprimir( "Votos por iOS: ", votosiOS )
39
40    // Se toma la decisión de cuál fue la más votada
41    Si( votosAndroid > votosiOS ) Entonces
42      imprima( "Android ganó" )
43    SiNo
44      Si(votosAndroid < votosiOS ) Entonces
45        imprimir( "iOS ganó" )
46      SiNo
47        imprimir( "Hay empate" )
48      FinSi
49    FinSi
50 FinAlgoritmo
```

Explicación del algoritmo:

Después de la declaración de las variables se inicializan en 0 los contadores `votosAndroid` y `votosiOS`, estos contadores podrán ser modificados dentro del ciclo.

La instrucción `Haga` indica el inicio del proceso repetitivo, de acuerdo a la operatividad de este ciclo se ejecuta su cuerpo por lo menos una vez.

Dicho cuerpo contiene las instrucciones para leer el codigo del estudiante, así como su elección de la plataforma.

El estudiante vota por Android ingresando una letra 'A', o vota por iOS ingresando una letra 'i'. Una vez hayan ingresado estos datos, se procede a contabilizar el respectivo voto, para ello se empleó una estructura de decisión múltiple con tres opciones:

```
22 Segun( voto )
23   Caso 'A':
24   Caso 'a': votosAndroid = votosAndroid + 1
25             FinCaso
26   Caso 'I':
27   Caso 'i': votosiOS = votosiOS + 1
28             FinCaso
29   EnOtroCaso: imprimir( "La opción no es válida" )
30 FinSegun
```

Se tuvo la precaución de contemplar la elección del estudiante tanto en mayúsculas como en minúsculas (líneas 23-24 y 26-27).

En el caso de que el voto no sea por ninguna de las dos plataformas, se mostrará el mensaje "La opción no es válida" y el voto no será contabilizado.

Luego de la decisión múltiple, se presenta la instrucción modificadora de condición:

```
30 imprimir( "Desea realizar un nuevo voto [S] [N]?: " )
31 leer( seguir )
```

A diferencia de los algoritmos que se realizaron, en el apartado anterior, con el ciclo `Mientras-FinMientras`, en esta solución no fue necesaria la inicialización de la variable `seguir` antes de entrar al ciclo. El valor que toma esta variable para poder evaluar la condición del `MientrasQue`, es ingresado por el usuario (`leer( seguir )`).

Una vez se obtenga la respuesta a la anterior pregunta, se evalúa la condición:

```
32 MientrasQue( seguir == 'S' O seguir == 's' )
```

Si la condición es verdadera, se regresa el control a la instrucción `Haga` y una vez más se repite el proceso. Cuando la condición sea falsa, se termina el ciclo y se continúa con la instrucción que está escrita debajo del `MientrasQue`.

A continuación, el algoritmo muestra la cantidad de votos por cada una de las plataformas (líneas 37 y 38) e informa cuál es la elegida; esta tarea se realizó usando uma estructura de decisión anidada (líneas 41 a 49). Finalmente, con la instrucción `FinAlgoritmo` se da por terminada la ejecución del algoritmo.

.:Ejemplo 4.11. *El factorial de un número, es el producto obtenido al multiplicar un número dado por todos los enteros positivos sucesivos inferiores. Además, se tiene establecido que los números negativos no poseen factorial y que el factorial de 0 es 1. El factorial se representa con un signo de exclamación precedido de un número (n!).*

***Por ejemplo**:*

$7! = 7 * 6 * 5 * 4 * 3 * 2 * 1 = 5040$

o se puede expresar como:

$7! = 1 * 2 * 3 * 4 * 5 * 6 * 7 = 5040.$

A $7!$ *se le denomina* 7 *factorial, también es llamado el factorial de* 7*.*

Teniendo en cuenta el anterior contexto, diseñe un algoritmo que calcule el factorial de un número entero.

Análisis del problema:

- **Resultados esperados:** factorial calculado o un mensaje que informe que los números negativos no poseen factorial.
- **Datos disponibles:** el número al cuál se le va a calcular el factorial.
- **Proceso:** se debe leer el número de entrada.

  Seguidamente se debe tomar una decisión para determinar si se hace el cálculo del factorial o se informa que es un número negativo y no posee factorial.

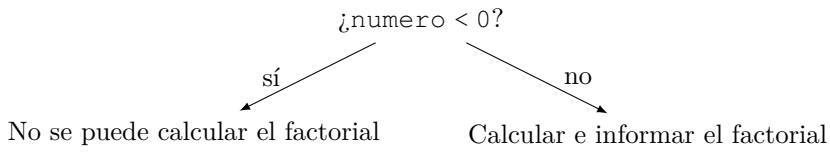


Figura 4.25: Árbol de decisión del Ejemplo 4.11

Para el cálculo del factorial se debe hacer un proceso repetitivo. Con el propósito de entender este procedimiento se analizará el cálculo

del factorial de 7 ($7! = 1 * 2 * 3 * 4 * 5 * 6 * 7 = 5040$), tal como se muestra en la Tabla 4.6.

Inferiores	**Cálculo**	**n!**
1	1	1
2	1! x 2	2
3	2! x 3	6
4	3! x 4	24
5	4! x 5	120
6	5! x 6	720
7	6! x 7	5040

Tabla 4.6: Cálculo de 7!

La columna `Inferiores` muestra los valores de cada uno de los números menores sucesivos hasta 7. En la columna `Cálculo` se observa el proceso acumulativo que se hace para el cálculo del factorial (`n!`), el factorial anterior es multiplicado por el número que está en la misma fila de la primera columna. La tercera columna acumula los resultados en cada iteración o cálculo realizado; al final se obtiene el resultado: $7! = 5040$.

Para diseñar el proceso descrito en la Tabla 4.6 y en el párrafo anterior, se requiere el uso de una estructura de repetición que se ejecute mientras que un contador que inicie en 1 llegue hasta el valor del número al cual se le calculará el factorial.

- **Variables requeridas:**
  - `numero`: número al cual se le calculará el factorial (n).
  - `factorial`: resultado del cálculo del factorial ($n!$).
  - `inferiores`: variable que almacenará los valores desde 1 hasta el número al cual se le calculará el factorial. Hará parte de la condición que controlará el ciclo que realizará el cálculo.

En la Figura 4.26 se muestra la solución del Ejemplo 4.11 mediante un diagrama de flujo.

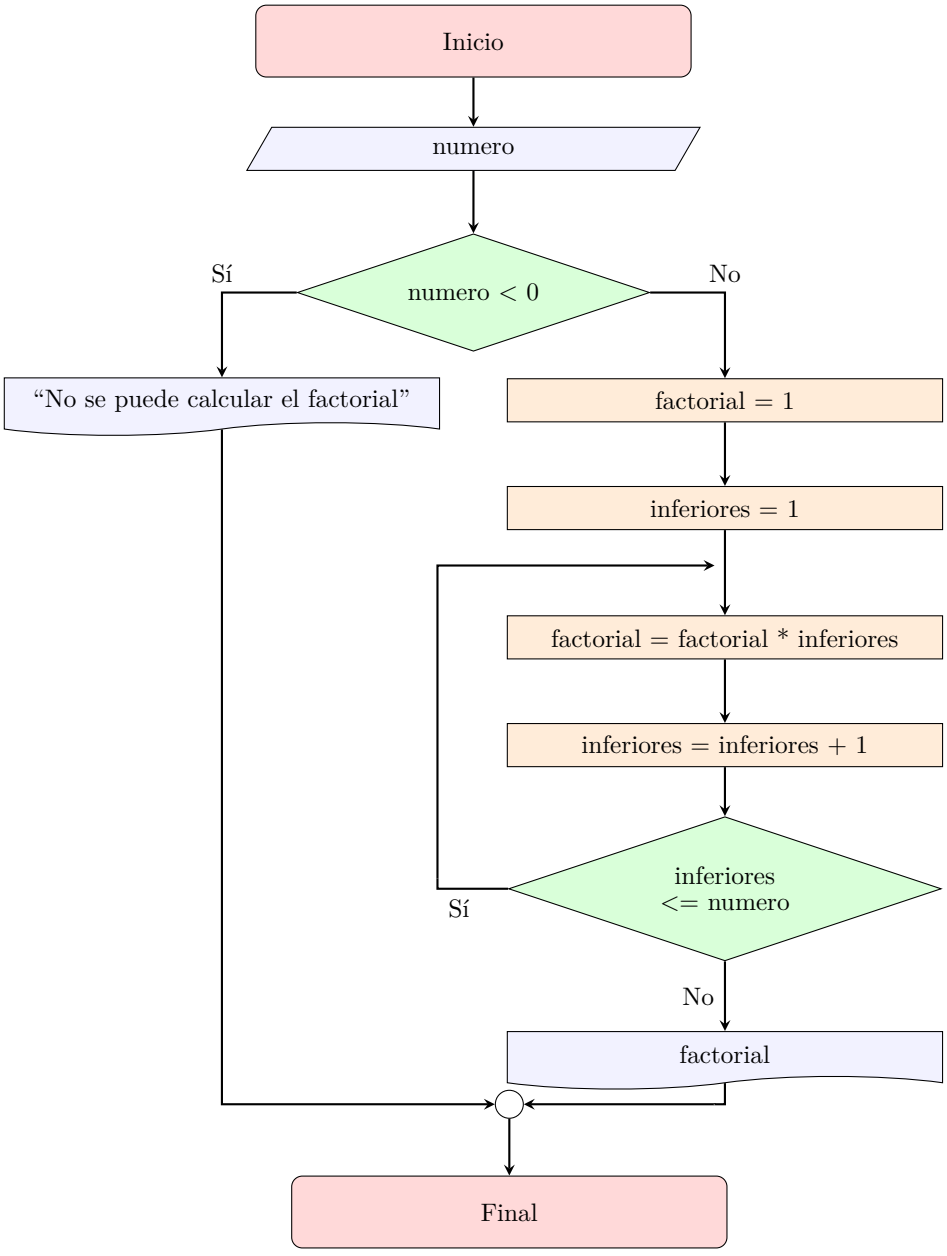


Figura 4.26: Diagrama de flujo del Algoritmo `FactorialNumero`

Para la solución en pseudocódigo se propone el Algoritmo 4.15.

Algoritmo 4.15: FactorialNumero

```
1  Algoritmo FactorialNumero
2    // Declaración de variables
3    Entero  numero, factorial, inferiores
4
5    // Dato disponible
6    imprimir( "Ingrese el número para el factorial: " )
7    leer( numero )
8
9    // Proceso y resultados esperados
10   Si( numero < 0 ) Entonces
11     imprimir( "No se puede calcular el factorial" )
12   SiNo
13     factorial  = 1
14     inferiores = 1
15     Haga
16       factorial  = factorial * inferiores
17       inferiores = inferiores + 1
18     MientrasQue( inferiores <= numero )
19
20     // Resultado esperado
21     imprimir( "Factorial de ", numero, " es: ", factorial )
22   FinSi
23 FinAlgoritmo
```

Al ejecutar el algoritmo:

Primera ejecución:

```
Ingrese el número para el factorial: 9
Factorial de 9 es: 362880
```

Segunda ejecución:

```
Ingrese el número para el factorial: 0
Factorial de 0 es: 1
```

Tercera ejecución:

```
Ingrese el número para el factorial: -12
No se puede calcular el factorial
```

Explicación del algoritmo:

Después de ingresar el número para el cálculo, se toma la decisión si este es un valor negativo (línea 10), en cuyo caso se informa que para los números negativos el factorial no se puede calcular. En caso de que el

número sea 0 o positivo, se procede a calcular el factorial haciendo uso de un ciclo `Haga-MientrasQue`.

Dentro del cuerpo del ciclo, el acumulador `factorial`, se incrementa mediante multiplicaciones sucesivas, razón por la cual se inicializó en 1 (línea 13). Hay que tener especial cuidado en este valor inicial, si se establece en 0, todos los cálculos van a dar 0.

En cada iteración del `Haga`, se va calculando el factorial mediante la expresión `factorial = factorial * inferiores` (línea 16). En la variable `inferiores` se generan todos los números inferiores que son multiplicados para obtener el factorial. Las iteraciones se ejecutan mientras el contenido de inferiores sea menor o igual a `numero` (`MientrasQue(inferiores <= numero)` (línea 18)).

.:Ejemplo 4.12. *Se requiere una solución algorítmica que resuelva la siguiente expresión matemática:*

$$expresion = 1 + \frac{x^2}{2!} - \frac{x^3}{4!} + \frac{x^4}{6!} - \frac{x^5}{8!} + \cdots \pm \frac{x^n}{(2(n-1))!}$$

Donde n es la cantidad de términos a calcular.

Análisis del problema:

- **Resultados esperados:** resultado del cálculo de la expresión.
- **Datos disponibles:** el valor para la constante x y la cantidad de términos (n).
- **Proceso:** inicialmente se debe obtener el valor para x y el valor para n.

  Para el cálculo se requiere un proceso repetitivo que vaya generando y acumulando el valor de los términos. El primer término es el 1, los siguientes están conformados por una división donde el numerador es la constante x elevada a un exponente; dicho exponente inicia en 2 y va incrementado su valor de 1 en 1 hasta llegar a n. El denominador tiene la característica de ser el factorial de los valores pares en forma creciente y consecutiva; inicia con un valor de 2 en el segundo término de la expresión y va hasta $2(n-1)$. Para el cálculo del factorial se requiere de otro proceso repetitivo, igual al que se explicó en el

Ejemplo 4.11. De acuerdo a lo anterior, en la solución planteada para resolver la expresión se deben trabajar dos ciclos anidados.

Otro aspecto importante que requiere análisis dentro de la expresión, es el hecho de que está conformada por sumas y restas sucesivas de manera alterna; a partir del segundo término, donde el exponente de x sea un número par se debe sumar y donde sea impar se debe restar.

- **Variables requeridas:**
  - `x`: representa la constante de la expresión. Cuando se le da la denominación de constante, no se refiere a la clasificación de los datos usados en los algoritmos, sino a que dentro de la expresión de este problema va a tener un valor constante. Para la solución algorítmica que se va a plantear, es una variable que toma un valor diferente cada que se ejecute.
  - `n`: almacena la cantidad de términos que tendrá la expresión.
  - `contadorTerminos`: esta variable tendrá tres papeles fundamentales en la solución. Primero servirá para contar la cantidad de términos que se van generando dentro de la expresión, segundo hará parte de la condición que contralará la cantidad de veces que se debe repetir el ciclo externo y tercero se usará como el exponente al cual se elevará la constante x.
  - `inferiores`: variable que almacenará los valores desde 1 hasta el valor del denominador en cada uno de los términos. Se usará en la condición que controlará el ciclo con el cual se calculará el factorial del denominador.
  - `denominador`: en esta variable se almacenará el denominador que se va generando para cada término de la expresión, tomará valores pares consecutivos iniciando desde 2. Su propósito es hacer parte de la condición del ciclo que calculará su factorial.
  - `factorial`: acumula el factorial que se le calcule al denominador de la expresión.
  - `expresion`: en este acumulador se guardará el valor del cálculo de la expresión. Tiene la característica que sufre incrementos y decrementos de acuerdo al valor del exponente de la constante x.

De acuerdo al análisis planteado, se propone el Algoritmo 4.16.

Algoritmo 4.16: ExpresionMatematica

```
1  Algoritmo ExpresionMatematica
2    // Declaración de variables
3    Entero  x, n, contadorTerminos, inferiores,
4            denominador,factorial
5    Real    expresion
6
7    // Datos disponibles
8    imprimir( "Ingrese el valor para x: " )
9    leer( x )
10   imprimir( "Ingrese la cantidad de términos (n): " )
11   leer( n )
12
13   // Proceso
14   expresion = 1
15   contadorTerminos = 2
16   denominador = 2
17
18   // Ciclo externo
19   Haga
20     factorial  = 1
21     inferiores = 1
22
23     // Ciclo interno
24     Haga
25       factorial  = factorial  * inferiores
26       inferiores = inferiores + 1
27
28     MientrasQue( inferiores <= denominador )
29
30     // Final del ciclo interno
31
32     denominador = denominador + 2
33
34     Si( (contadorTerminos % 2) == 0 ) Entonces
35       expresion = expresion^(x+contadorTerminos)/factorial
36     SiNo
37       expresion = expresion^(x-contadorTerminos)/factorial
38     FinSi
39
40     contadorTerminos = contadorTerminos + 1
41
42   MientrasQue( contadorTerminos <= n )
43   // Final del ciclo externo
44
45   imprimir( "El valor de la expresión es: ", expresion )
46 FinAlgoritmo
```

Al ejecutar el algoritmo:

Primera ejecución:

```
Ingrese el valor para x: 3
Ingrese la cantidad de términos (n): 6
El valor de la expresión es: 4.4815
```

Segunda ejecución:

```
Ingrese el valor para x: 5
Ingrese la cantidad de términos (n): 7
El valor de la expresión es: 9.0863
```

Explicación del algoritmo:

Para el cálculo de la expresión fue necesario trabajar con dos estructuras de repetición `Haga-MientrasQue` anidadas. El ciclo externo controla que se vayan generando y calculando cada uno de los términos de la expresión, desde el segundo hasta el término n. El ciclo interno es el responsable de calcular el factorial de los denominadores que se van generando en cada uno de los términos.

Los acumuladores y contadores tienen una inicialización bien particular con relación a los ejemplos trabajados hasta el momento:

`expresion = 1` (línea 14): a pesar que es un acumulador que va a ser modificado por sumas y restas sucesivas, no se inicializó en 0; en este caso tomó el valor de 1 que corresponde al primer término de la expresión.

Dado lo anterior, la generación de los términos y los cálculos inician a partir del segundo término, lo cual justifica las siguientes inicializaciones que se encuentran en el algoritmo: `contadorTerminos = 2` y `denominador = 2` (líneas 15 y 16 respectivamente). Recuerde que `contadorTerminos`, también se usó como el exponente de x.

Después de calcular el factorial dentro del ciclo interno[7], se hace el incremento o decremento del acumulador de la expresión; para ello se planteó la decisión de la línea 34, en donde se determina si el exponente de x (`contadorTerminos`) es par[8]. Para los valores pares se hace un incremento (línea 35), para los impares se aplica el decremento (línea 37).

En las Figuras 4.27 y 4.28 se muestra la solución del Ejemplo 4.12 mediante un diagrama de flujo.

[7]Este proceso se explicó en el Ejemplo 4.11.

[8]Todo número par da 0 como resultado del resto, en la divisón entera entre 2.

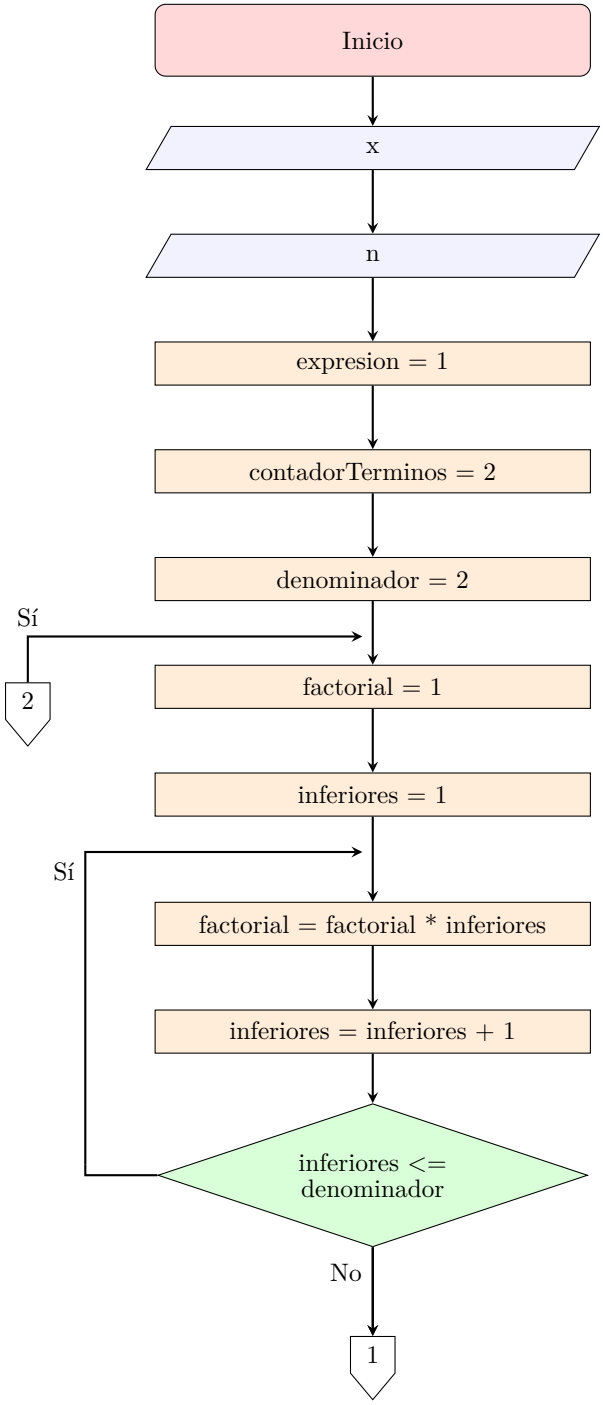


Figura 4.27: Diagrama de flujo `ExpresionMatematica` - Parte 1

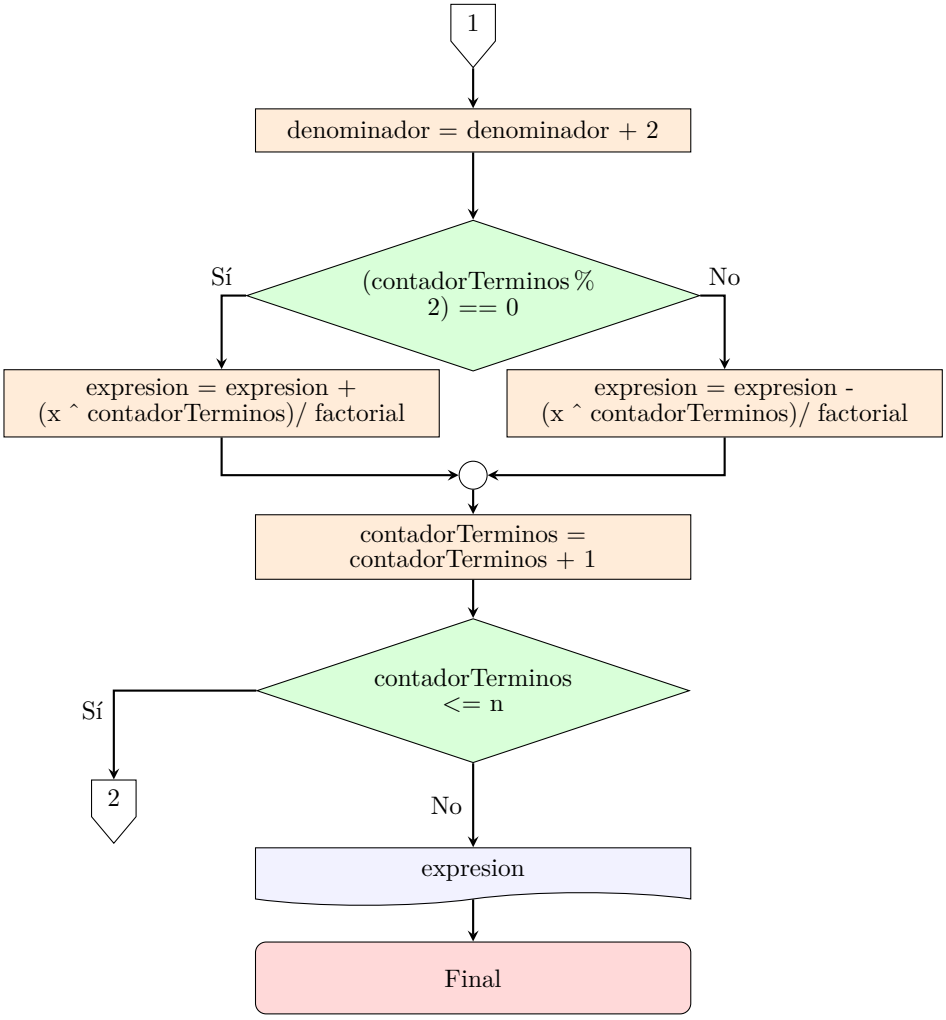


Figura 4.28: Diagrama de flujo `ExpresionMatematica` - Parte 2

Aclaración:



El ciclo `Haga-MientrasQue` se suele usar en procesos de validación de datos de entrada, en donde se condiciona a que el usuario ingrese datos de acuerdo a los valores establecidos en los requisitos del problema.

Validación de datos de entrada

Hasta el momento se han trabajado algoritmos donde se asume que la entrada de los datos es correcta o en su defecto se han colocado mensajes que informen la situación. Tal es el caso del Ejemplo 4.10 donde el usuario debe elegir entre Android o iOS digitando una 'A' o una 'i'; allí se mostró el mensaje "La opción no es válida" para el caso que el usuario del algoritmo digite un valor diferente a los establecidos. Lo ideal, aparte de informar la situación, es que el algoritmo esté diseñado de tal forma que no se puedan hacer entradas con datos errados.

De acuerdo al anterior contexto, se usará el ciclo `Haga-MientrasQue` para validar que las entradas sean correctas. Esta estructura es la ideal para hacer este proceso, debido a que primero ejecuta el cuerpo de ciclo y luego revisa la condición.

Se plantea entonces que, para estas validaciones se sigan estos pasos:

1. Pedir el dato
2. Escribir la instrucción `Haga`
3. Leer el dato
4. Escribir la instrucción `MientrasQue` y su condición.

Con base a lo enunciado, para la validación de un dato numérico se procede tal y como se muestra en el segmento de Algoritmo 4.17, el cual permite ingresar un número que sea mayor o igual a un valor mínimo y al mismo tiempo sea menor o igual a un valor máximo, en otro caso[9] el ciclo se hace verdadero y se solicita nuevamente el valor.

Algoritmo 4.17: Validación de un valor numérico en un rango

```
1 imprimir( "Ingrese un valor numérico: " )
2 Haga
3   leer( valor )
4 MientrasQue( valor < mínimo O valor > máximo )
```

Para ilustrar este caso se muestra el segmento de Algoritmo 4.18 en el cual se solicita una edad entre 18 y 90 años.

[9] Que le número sea estrictamente menor que un valor mínimo **O** mayor a un valor máximo

Algoritmo 4.18: Ejemplo de validación de un valor numérico en un rango

```
1 imprimir( "Ingrese la edad (entre 18 y 90 años): " )
2 Haga
3   leer( edad )
4 MientrasQue( edad < 18 O edad > 90 )
```

En este ejemplo (segmento de Algoritmo 4.18) se solicita ingresar una edad entre 18 y 90 años, luego se abre el ciclo con la instrucción `Haga`, seguidamente se lee la variable `edad` que almacenará el valor digitado.

Finalmente se encuentra la condición `MientrasQue (edad < 18 O edad > 90)`, utilizada para validar que la entrada del valor esté entre 18 y 90. Las validaciones para este tipo de dato se hacen teniendo en cuenta que solo se van a ingresar números pertenecientes a un rango de valores.

Suponga que como dato de entrada proporcionan una edad de 15 años, entonces la variable `edad` almacena este valor, al evaluar la condición se tiene que:

```
edad < 18 O edad > 90
  15 < 18 O 15 > 90
Verdadero O Falso
      Verdadero
```

Como el resultado final de la condición es verdadero, el ciclo se repite, solicitando nuevamente el valor de la edad, y así hasta que el usuario ingrese un valor válido (entre 18 y 90).

En los pasos para validar mostrados en el segmento del Algoritmo 4.17, `mínimo` y `máximo` se refieren al rango de valores que aceptará el algoritmo como entrada, pero puede suceder que solamente se requiera solo uno de los valores, por ejemplo, si el problema establece que se va a trabajar como dato de entrada un valor positivo (no incluido el cero), sin importar el tope superior, la validación puede expresarse como se muestra en el segmento de Algoritmo 4.19.

Algoritmo 4.19: Validación de un valor inferior

```
1 imprimir( "Ingrese un valor numérico: " )
2 Haga
3   leer( valor )
4 MientrasQue( valor < 1 )
```

En cuanto a la validación de los datos de tipo `Caracter`, se deben tener en cuenta, por separado, cada uno de los valores (caracteres) que puede recibir. Para ello se siguen estos pasos:

Algoritmo 4.20: Validación de un Caracter

```
imprimir ( "Ingrese una letra: " )
Haga
  leer( letra )
MientrasQue( letra != 'letra1' Y letra != 'letra2' )
```

Por ejemplo, suponga que se desea validar que el usuario ingrese la letra 'S' o 'N' tanto mayúsculas como minúsculas, entonces se puede proceder como se muestra el segmento de Algoritmo 4.21.

Algoritmo 4.21: Ejemplo de validación de un Caracter

```
imprimir ( "Desea continuar [S] o [N]: " )
Haga
  leer( seguir )
MientrasQue( seguir != 'S' Y seguir != 's' Y
             seguir != 'N' Y seguir != 'n' )
```

La condición especificada tiene en cuenta que solo pueda ingresar la letra 'S' o la letra 'N' en mayúscula o en minúscula. La variable `seguir` debe estar declarada de tipo `Caracter`.

Sin embargo, en la mayoría de los lenguajes existe la posibilidad de convertir las letras que digiten a mayúscula, de esta manera no se requieren tener en cuenta las letras minúsculas.

Otro de los datos que se deben validar, son los de tipo `Cadena`, es decir, aquellos que son un conjunto de caracteres. Generalmente la validación se enfoca a que no se reciban valores vacíos (Ver Algoritmo 4.22)

Algoritmo 4.22: Validación de un Cadena

```
imprimir ( "Ingrese una cadena: " )
Haga
  leer( cadena )
MientrasQue( longitud ( cadena ) == 0 )
```

Por ejemplo, para validar que en la solicitud de un nombre no se omita el dato, se debe proceder de acuerdo al segmento de Algoritmo 4.23

Algoritmo 4.23: Validación de un nombre

```
imprimir ( "Ingrese su nombre: " )
Haga
  leer( nombre )
MientrasQue( longitud ( nombre ) == 0 )
```

Para evitar que las variables de este tipo de dato queden en blanco, una de las formas es trabajar con la función `longitud`, cuyo propósito es medir la cantidad de caracteres almacenados dentro de una variable de tipo `Cadena`. Cuando en este tipo de variables no se ha almacenado ningún dato, se dice que su longitud es de 0, por el contrario, si tuviera almacenado por ejemplo, el nombre `"Liliana"`, su longitud sería de 7.

La expresión relacional `MientrasQue(Longitud(nombre )==0)`, evalúa si la longitud del dato que almacene la variable nombre es igual a 0. Si el resultado es verdadero, significa que se está frente a un dato vacío, en consecuencia, el ciclo debe repetirse para leer nuevamente la variable nombre. Esta condición tomará el valor de falso en el momento que ingresen un nombre, mínimo de 1 carácter.

También es válido condicionar a que mínimo sea un determinado número de caracteres, por ejemplo, si se quisiera que el nombre tuviera mínimo 3 caracteres, la condición se plantearía así:

```
MientrasQue( longitud( nombre ) < 3 )
```

De forma similar, se puede establecer la cantidad mínima y máxima de caracteres en el nombre. La siguiente condición valida que no se vaya a dejar el nombre como un dato en blanco y que máximo tenga 15 caracteres:

```
MientrasQue( longitud( nombre ) == 0 O
             longitud( nombre ) > 15 )
```

También es posible validar si el contenido de la cadena corresponde o no a ciertos valores[10] (Ver Algoritmo 4.24).

Algoritmo 4.24: Ejemplo de validación del contenido de una `Caracter`

```
1 imprimir( "Ingrese una cadena: " )
2 Haga
3   leer( cadena )
4 MientrasQue( cadena != "texto1" Y cadena != "texto2" )
```

Ahora que ya conoce como validar la entrada de datos a un algoritmo, se procederá a analizar algunos ejemplos haciendo uso de estas validaciones.

.:Ejemplo 4.13. *El método de multiplicación de los campesinos rusos, consiste en tomar los dos factores de la operación (multiplicando y multiplicador) y disponerlos cada uno en una columna. El primer factor*

[10]Existen ciertos lenguajes de programación que requieren instrucciones especiales para comparar cadenas, en este libro no serán necesarias tales instrucciones especiales

se va multiplicando sucesivamente por 2, simultáneamente en la segunda columna al segundo factor se le van aplicando divisiones enteras entre 2. Estas operaciones se realizan hasta que el segundo factor llegue a 1. El siguiente paso es sumar todos los números de la primera columna que estén al frente de un número impar de la segunda columna. El resultado que se obtenga es el producto de los dos números.

A continuación, se ilustra el método descrito, con la multiplicación de 3 por 19 donde el resultado esperado es 57.

Primer factor	**Segundo factor**	**Impar**	**Producto**
3	19	Sí	3
6	9	Sí	6
12	4	No	
24	2	No	
48	1	Sí	48
		Total suma:	57

Tabla 4.7: Ejemplo ilustrativo - Ejemplo 4.13

Basados en el método descrito, se va a desarrollar un algoritmo que halle el producto de dos números enteros entre 0 y 10000.

Análisis del problema:

- **Resultados esperados:** producto de dos números enteros.
- **Datos disponibles:** multiplicando y multiplicador.
- **Proceso:** se hace la lectura del multiplicando y del multiplicador, teniendo en cuenta que sus valores deben estar entre 0 y 10000[11].

  Una vez se tengan los valores de entrada, se implementará un proceso repetitivo en donde en cada iteración se multiplique por 2 el primer factor o multiplicando; de manera simultánea se realizan divisiones enteras, entre 2, del segundo factor o multiplicador. En cada vuelta del ciclo se debe analizar si cada uno de los valores que va tomando el segundo factor es impar y así proceder a hacer la acumulación del primer factor. Este proceso iterativo terminará en el momento que las divisiones del segundo factor lo lleven a un valor de 1.

[11] Tenga en cuenta que el tope para el valor del dato de entrada no afecta el proceso que se va a realizar, puede establecer topes diferentes. Acá se estableció este valor a manera de ejemplo.

- **Variables requeridas:**
  - `multiplicando` y `multiplicador`: datos de entrada para calcular el producto.
  - `factor1` y `factor2`: almacenarán una copia del valor original del multiplicando y del multiplicador.
  - `producto`: resultado esperado que se obtendrá de la acumulación sucesiva de los valores del primer factor.

De acuerdo al análisis planteado, se propone el Algoritmo 4.25.

Algoritmo 4.25: MultiplicacionRusa

```
Algoritmo MultiplicacionRusa
  // Declaración de variables
  Entero multiplicando, multiplicador, producto,
         factor1, factor2

  // Datos disponibles
  imprimir( "Ingrese el multiplicando: " )
  Haga
    leer( multiplicando )
  MientrasQue( multiplicando < 0 || multiplicando > 10000 )

  imprimir( "Ingrese el multiplicador: " )
  Haga
    leer( multiplicador )
  MientrasQue( multiplicador < 0 || multiplicador > 10000 )

  // Proceso
  factor1 = multiplicando  // Se hace copia del valor
  factor2 = multiplicador  // Se hace copia del valor
  producto = 0

  Haga
    Si( factor2 % 2 != 0 ) Entonces
      producto = producto + factor1
    FinSi

    factor1 = factor1 * 2
    factor2 = factor2 / 2
  MientrasQue( factor2 >= 1 )

  // Resultados esperados
  imprimir( "Producto de ", multiplicando,
            " x ", multiplicador, " = ", producto )
FinAlgoritmo
```

Al ejecutar el algoritmo:

```
Ingrese el multiplicando: 5
Ingrese el multiplicador: 17

Producto de 5 x 17 = 85
```

Explicación del algoritmo:

Se pide ingresar el multiplicando y el multiplicador. La instrucción `leer` se encuentra como el cuerpo de un ciclo `Haga-MientrasQue`; esto se hizo con el propósito de condicionar que el dato de entrada para cada una de estas variables debe ser un valor entre 0 y 10000.

El acumulador, denominado `producto` se inicializó en 0; se usa para almacenar el cálculo de la multiplicación. Dependiendo del resultado de una decisión, se hace su incremento mediante sumas sucesivas dentro del ciclo:

```
23 Si( factor2 % 2 != 0 ) Entonces
24   producto = producto + factor1
25 FinSi
```

La anterior estructura de decisión simple, se usó para determinar si el valor de la variable `factor2` es impar. Esta evaluación se hace en cada vuelta del ciclo. Observe que solamente se codificó una instrucción en el caso que sea verdadera, en caso contrario no se hace ninguna acción que dependa de la decisión.

Seguido a la decisión se duplica `factor1`, luego `factor2` se divide entre 2.

Continuando con las instrucciones, se testea la expresión relacional `MientrasQue( factor2 >= 1 )`, puede suceder una de dos situaciones. Si la expresión es falsa se termina el ciclo, se informa el resultado y termina el algoritmo; si es verdadera, se regresa a la instrucción `Haga` para que el ciclo vuelva a iterar. Todo se repite mientras `factor2` no haya tomado el valor de 0 para cambiarle el estado a la condición y hacerla falsa.

Aclaración:



La condición `MientrasQue (factor2 >= 1)`, es equivalente a `MientrasQue (factor2 > 0)`.

.:Ejemplo 4.14. *Diseñe un algoritmo, que simule una calculadora con las 4 operaciones básicas (suma, resta, división y multiplicación). Las operaciones deben realizarse a medida que se van ingresando los datos, de igual forma se debe ir mostrando el resultado parcial. Se terminará de hacer operaciones en el momento que se presione el signo igual (=).*

Análisis del problema:

- **Resultados esperados:** cálculo de todas las operaciones realizadas. Aunque el enunciado no lo dice, se debe contemplar la posibilidad de que el usuario trate de dividir entre 0, si se detecta esta situación, el algoritmo deberá informarla y terminar su ejecución.
- **Datos disponibles:** números a calcular, operadores básicos ('+', '-' ,'/' y '*') y el signo '=' para terminar la ejecución.
- **Proceso:** se solicita un número inicial, luego uno de los operadores básicos o el signo '='. Si el valor digitado no es el signo '=', se solicitará un segundo número; dependiendo del operador se ejecuta el respectivo cálculo, informando el resultado parcial. Todo esto se hace en un proceso repetitivo mientras no se presione el signo '=' o se trate de dividir entre 0.
- **Variables requeridas:**
  - `numero`: esta variable almacena el número o los números que intervienen en las operaciones.
  - `calculo`: es el acumulador de los resultados de todas las operaciones.
  - `operador`: guarda el operador o el signo '=' que ingrese el usuario.
  - `bandera`: en el caso que se trate de dividir entre 0, tomará el valor de `Falso` y se mostrará un mensaje de error sin el resultado de los cálculos; si la terminación del algoritmo se hace de forma normal, su valor será `Verdadero` y se imprimirá el resultado de las operaciones.

De acuerdo al análisis planteado, se propone el Algoritmo 4.26.

Algoritmo 4.26: Calculadora

```
Algoritmo Calculadora
  // Este algoritmo simula una calculadora con las
  // 4 operaciones básicas

  // Declaración de variables
  Entero   numero, calculo
  Caracter operador
  Logico   bandera

  imprimir( "Digite un número: " )
  Haga
    leer( numero )
  MientrasQue( numero < -200000 O numero > 200000 )

  // Inicialización de variables
  calculo = numero
  bandera = Verdadero

  // Proceso repetitivo
  Haga

    imprimir( "Digite un operador: " )
    Haga
       leer( operador )
    MientrasQue( operador != '+' Y operador != '-' Y
                 operador != '/' Y operador != '*' Y
                 operador != '=')

    Si( operador != '=' ) Entonces
      imprimir( "Digite otro número: " )
      Haga
       imprimir( numero )
      MientrasQue( numero < -200000 O numero > 200000 )

      Segun( operador )
        Caso '+': calculo = calculo + numero
                  FinCaso
        Caso '-': calculo = calculo - numero
                  FinCaso
        Caso '*': calculo = calculo * numero
                  FinCaso
        Caso '/': Si( numero == 0 ) Entonces
                    imprimir( "Error. División entre cero" )
                    bandera = Falso // Cambio de estado
                  SiNo
                    calculo =  calculo / numero
                  FinSi
                  FinCaso
```

```
49          FinSegun
50      FinSi
51
52      Si( bandera ) Entonces  //Si (bandera == Verdadero )
53          imprimir( calculo )
54      FinSi
55
56      MientrasQue( operador != '=' )
57  FinAlgoritmo
```

Explicación del algoritmo:

Este algoritmo hace uso de varias estructuras Haga-MientrasQue independientes y anidadas. También se trabajaron estructuras de decisión simple y múltiple anidadas.

Parte de las estructuras Haga-MientrasQue se destinaron a la validación de la entrada de los datos. Los de tipo numérico están validados para que acepten valores entre -200000 y 200000, este tope se eligió más por mostrar el ejemplo de cómo validar un dato numérico que, por algún requerimiento en el enunciado del problema; el diseñador del algoritmo puede tomar los valores que estime conveniente. El ciclo dedicado a la validación de la lectura del operador matemático condiciona a que solo pueda ingresar uno de los siguientes caracteres: '+','-' ,'*' y '/' o el signo '='.

La variable `calculo` que almacena el resultado de todas las operaciones que se realicen, se inicializó con el valor de la variable `numero`, el cual corresponde al primer número que se obtendrá para iniciar el proceso de la calculadora. De igual forma se hace la inicialización de la variable `bandera` con un valor `Verdadero`, el cual cambiará en el caso que se trate de hacer una división entre 0.

Seguidamente se encuentra el ciclo `Haga`, donde las primeras instrucciones que ejecuta son solicitar y leer un operador con el fin de determinar qué proceso se hace con el valor almacenado en la variable `calculo`. Si ingresan un signo '=' se termina el proceso y se informa el resultado. Si por el contrario ingresan uno de los operadores básicos, se procede a solicitar el segundo número; si el operador corresponde a uno de los siguientes '+', '-' o '*' se realiza la respectiva operación, en el caso que el operador sea '/' se toma una decisión.

```
42 Caso '/': Si( numero == 0 ) Entonces
43              imprimir( "Error. División entre cero" )
44              bandera = Falso  // Cambio de estado
45           SiNo
46              calculo =  calculo / numero
47           FinSi
48           FinCaso
```

Esta decisión es parte de una estructura `Segun-FinSegun` y corresponde al caso '/'. Cuando se ingresa el operador de división se verifica el valor del número con el propósito de informar la situación de división entre 0.

Observe que cuando el número es 0, se imprime un mensaje informando la situación (línea 43). A `bandera` se le cambia el estado de `Verdadero` a `Falso`, con el fin de omitir la instrucción que imprime el resultado (línea 53). Finalmente a la variable `operador` se le asigna el signo '='; de esta manera cuando se evalúe la condición `MientrasQue (operador != '=')` el resultado será `Falso` y se dará por terminada la ejecución del ciclo y en consecuencia la del algoritmo.

Contrario a todo lo expuesto en el párrafo anterior, si el número es diferente de 0, se realiza la división, se informa el contenido de la variable `calculo` (línea 53) y se testea la condición del `MientrasQue`; al obtener un resultado `Verdadero` se repite el ciclo `Haga-MientrasQue`.

La variable `operador`, tiene la particularidad, que además de determinar qué tipo de operación se realiza con los números ingresados, está haciendo el papel del centinela que controla el ciclo `Haga-MientrasQue`.

.:Ejemplo 4.15. *En matemáticas un número es perfecto si es igual a la suma de sus divisores propios positivos. Un ejemplo de número perfecto es el* 28 *dado que:* $1 + 2 + 4 + 7 + 14 = 28$.

Construya un algoritmo que acepte como dato de entrada un número entero positivo e informe se es o no un número perfecto. El algoritmo debe ejecutarse hasta que el usuario determine lo contrario.

Análisis del problema:

- **Resultados esperados:** un mensaje que informe si el número ingresado al algoritmo es o no perfecto.
- **Datos disponibles:** un número entero positivo.

- **Proceso:** teniendo en cuenta que el enunciado condiciona a que el número sea positivo, la lectura del dato debe hacerse de tal forma que solo acepte valores mayores o iguales a 1.

  Una vez se posea el número, debe hacerse un proceso repetitivo que determine si cada uno de los números menores a él es su divisor. Recuerde que se puede determinar si un número es divisor de otro, si al dividir el mayor entre el menor se obtiene una división exacta, es decir cuando el residuo es 0. En el caso de los algoritmos, un residuo o resto de la división se logra usando el operador módulo: '%'. En tal sentido, la siguiente decisión determina si un número es divisor de otro:

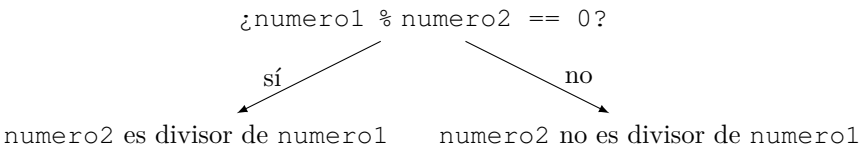


Figura 4.29: Árbol de decisión del Ejemplo 4.15

  Para el problema, objeto de este análisis, si la decisión es verdadera se procederá a incrementar un acumulador donde se sumarán todos los números menores que sean divisores del número ingresado. Una vez terminada la ejecución del proceso cíclico, se decidirá si la sumatoria es igual al número, en cuyo caso se informará que si es un número perfecto, o que no lo es en caso contrario. De acuerdo a la definición de número perfecto, el 1 no lo es, ya que no tiene divisores propios (menores a él).

  Todo este proceso debe estar anidado dentro de otro ciclo, que permita ejecutarse hasta que el usuario no desee ingresar más números.

- **Variables requeridas:**

  - `numero`: almacena el dato que ingresará al algoritmo.
  - `numeroMenor`: en esta variable se generarán todos los números menores al número ingresado, con el fin de comprobar si son sus divisores.
  - `sumaDivisores`: acumulador de la suma de los divisores del número ingresado.
  - `seguir`: almacenará la respuesta si desea o no continuar con la ejecución del algoritmo.

De acuerdo al análisis planteado, se propone el Algoritmo 4.27.

Algoritmo 4.27: Perfecto

```
Algoritmo Perfecto
  // Declaración de variables
  Entero numero, numeroMenor, sumaDivisores
  Caracter seguir

  // Inicialización centinela ciclo externo
  seguir = 'S'

  // Inicia ciclo externo
  Mientras( seguir == 'S' O seguir == 's' )
    // Dato disponible
    imprimir ("Ingrese un número entero positivo: ")
    Haga
       leer( numero )
    MientrasQue( numero < 1 )  // Valida - solo positivos

    // Inicialización variables ciclo interno
    sumaDivisores = 0
    numeroMenor   = 1   //  divisor de una operación

    // Inicia ciclo interno
    Haga
      Si( numero % numeroMenor == 0 ) Entonces
        sumaDivisores = sumaDivisores + numeroMenor
      FinSi
        numeroMenor = numeroMenor + 1
    MientrasQue( numeroMenor <  numero )

    // Resultados esperados
    Si( sumaDivisores == numero Y numero != 1 ) Entonces
      imprimir( numero, "es un número perfecto." )
    SiNo
      imprimir( numero, " no es un número perfecto." )
    FinSi

    imprimir( "Desea continuar [S] [N]?: " )
    Haga
      leer( seguir )
    MientrasQue( seguir != 'S' Y seguir != 'N' Y
                 seguir != 's' Y seguir != 'n' )
  FinMientras
FinAlgoritmo
```

Al ejecutar el algoritmo:

Primera ejecución:

```
Ingrese un número entero positivo: 496
496 es un número perfecto.
Desea continuar [S] [N]?: S
```

Segunda ejecución:

```
Ingrese un número entero positivo: 57
57 no es un número perfecto.
Desea continuar [S] [N]?: N
```

Explicación del algoritmo:

El algoritmo tiene ciclos anidados. Para el ciclo externo se usó un `Mientras-FinMientras`, dentro de él se anidaron 3 ciclos `Haga-MientrasQue`, adicionalmente se plantearon dos estructuras de decisión. El primer `Haga-MientrasQue` (líneas 13 a 15) controla que el número ingresado sea positivo. El segundo (líneas 22 a 27) tiene la función de generar todos los números menores al número dado y de acumular la suma de sus divisores propios.

Observe que la decisión planteada dentro de este segundo ciclo, no posee parte falsa:

```
23 Si( numero % numeroMenor == 0 ) Entonces
24   sumaDivisores = sumaDivisores + numeroMenor
25 FinSi
```

Además, su expresión relacional está indicada mediante una operación matemática y una comparación de igualdad. La operación matemática `(numero % numeroMenor)` calcula el resto de la división entre el número y cada uno de los números menores a él; este resultado se compara con el valor 0. Si el resultado de dicha comparación es verdadero, indica que es una división exacta, lo cual concluye que el número menor es un divisor del número analizado, por lo tanto, se procede a acumular su valor en la variable `sumaDivisores`.

Los siguientes pasos son: ejecutar el incremento de la variable `numeroMenor` (línea 26); evaluar la condición `MientrasQue (numeroMenor < numero)`, para determinar si se ejecuta otra vez o se termina este ciclo `Haga-MientrasQue`.

Una vez este ciclo termine de iterar, se evalúa la siguiente decisión para determinar si el número es perfecto o no:

```
30 Si( sumaDivisores == numero Y numero != 1 ) Entonces
```

La expresión relacional del tercer ciclo Haga-MientrasQue (líneas 37 a 39), valida que la variable seguir, reciba como dato de entrada uno de los siguientes caracteres: 'S', 'N', 's' o 'n'.

Después de esta condición, se encuentra el FinMientras que regresa el control del algoritmo al Mientras (línea 10), donde se evalúa su condición. Si es verdadera, se vuelve a repetir todo su cuerpo, de lo contrario se termina el ciclo; a continuación, se pone fin al algoritmo para que termine su ejecución.

4.3.1 Prueba de escritorio

Para terminar con el tema de la instrucción repetitiva Haga-MientrasQue, se hará la prueba de escritorio o tabla de verificación para los siguientes dos algoritmos:

.:Ejemplo 4.16. *Para el Algoritmo 4.28 que halla la división entera (cociente y resto), de 38 entre 6, mediante restas sucesivas, realice la respectiva prueba de escritorio.*

Algoritmo 4.28: Division

```
1  Algoritmo Division
2    // Declaración de variables
3    Entero dividendo, divisor, cociente, resto
4
5    // Inicialización de variables
6    dividendo = 38
7    divisor   = 6
8    cociente  = 0
9
10   // Proceso de la división mediante restas sucesivas
11   Haga
12     dividendo = dividendo - divisor
13     cociente  = cociente + 1
14   MientrasQue( dividendo >= divisor )
15
16   resto = dividendo
17
18   // Informe de resultados
19   imprimir( "El cociente es: ", cociente )
20   imprimir( "El resto es: ", resto )
21 FinAlgoritmo
```

La tabla de verificación o prueba de escritorio para el Ejemplo 4.16 se puede observar en la Tabla 4.8.

divisor = 6

dividendo	cociente	dividendo>= divisor
38	0	
32	1	32 >= 6 (Verdadero)
26	2	26 >= 6 (Verdadero)
20	3	20 >= 6 (Verdadero)
14	4	14 >= 6 (Verdadero)
8	5	8 >= 6 (Verdadero)
2	6	2 >= 6 (Falso)

cociente = 6

resto = 2

Tabla 4.8: Prueba de escritorio - Algoritmo 4.16

Las dos primeras columnas de la tabla se destinaron para almacenar los valores de las variables `dividendo` y `cociente`, los cuales se van actualizando en cada iteración hasta llegar al valor esperado; en la tercera columna se evalúa la condición del `Haga-MientrasQue`.

Se inicializan las variables `dividendo`, `divisor` y `cociente`, en 38, 6 y 0 respectivamente. Estos valores se registran en la tabla de verificación.

Al continuar con la ejecución del algoritmo se encuentra la instrucción `Haga`, que indica el inicio de una estructura repetitiva. La primera instrucción de este ciclo reduce la variable `dividendo` en 6, que es el valor del `divisor` en este ejemplo `(dividendo = dividendo - divisor)`; la segunda instrucción aumenta en 1 el valor del `cociente`, su propósito es contar el número de restas que se hagan, las cuales equivalen al cociente de la división de estos dos números. Los resultados de ambas operaciones son anotados en la tabla de verificación.

Seguidamente, se encuentra la condición del ciclo, al testearla el resultado es verdadero, tal como se evidencia en la tabla (`32 >= 6 (Verdadero)`). El control del algoritmo lo vuelve a asumir la instrucción `Haga` y una vez más se repite el procedimiento anotado en el párrafo anterior.

Los pasos descritos en los dos párrafos anteriores, se repiten mientras el valor almacenado en el `dividendo` sea mayor o igual al del `divisor`. Cuando el `dividendo` toma el valor de 2 y se vuelve a evaluar la condición del `Haga-MientrasQue`, se tiene que `2 >= 6` arroja un resultado falso.

Como consecuencia de esta evaluación el ciclo deja de ejecutarse y se realiza la siguiente operación: `resto = dividendo`, asignándose el valor de 2 a `resto`. Observe la tabla en detalle y analice los valores registrados.

Por último, se hace la impresión de los valores almacenados en `cociente` y en `resto`, para luego finalizar con el algoritmo.

.:Ejemplo 4.17. *La Tabla 4.9 corresponde a la prueba de escritorio del diagrama de flujo de la Figura 4.30. Imprime la tabla de multiplicar del 7 y del 9, cada una desde la fila 1 hasta la fila 5. Analice la explicación que se da sobre su funcionamiento.*

`primera = 7`
`ultima = 9`

primera	**fila**	**producto**	**fila <= 5**	**primera <= ultima**	imprimir		
					primera	**fila**	**producto**
7	1	7			7	1	7
	2		2 <= 5 (V)				
		14			7	2	14
	3		3 <= 5 (V)				
		21			7	3	21
	4		4 <= 5 (V)				
		28			7	4	28
	5		5 <= 5 (V)				
		35			7	5	35
	6		6 <= 5 (F)				
9				9 <= 9 (V)			
	1	9			9	1	9
	2		2 <= 5 (V)				
		18			9	2	18
	3		3 <= 5 (V)				
		27			9	3	27
	4		4 <= 5 (V)				
		36			9	4	36
	5		5 <= 5 (V)				
		45			9	5	45
	6		6 <= 5 (F)				
11				11 <= 9(F)			

Tabla 4.9: Prueba de escritorio - Algoritmo 4.17

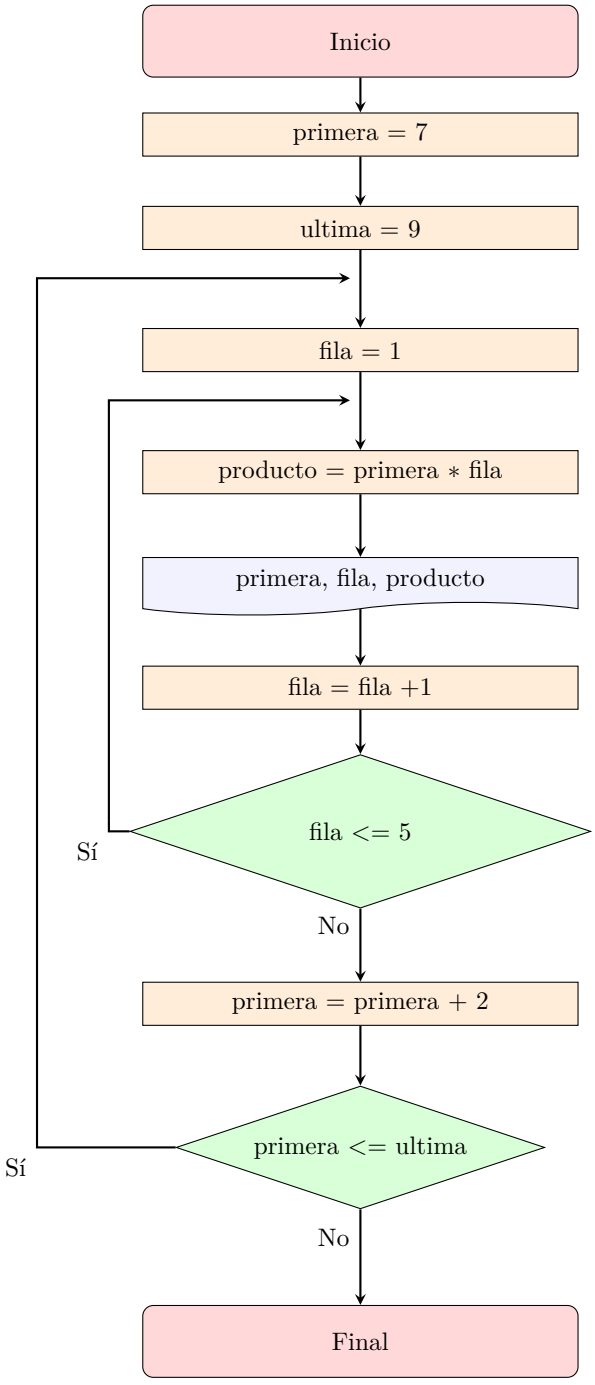


Figura 4.30: Tabla de multiplicar del 7 y del 9

Explicación de la prueba de escritorio:

En esta prueba de escritorio (Ver Tabla 4.9) se trabaja con dos ciclos `Haga-MientrasQue` anidados. El ciclo que tiene la condición `(fila <= 5)` es el ciclo interno y es el que más veces se ejecuta, en la tabla se registraron 10 evaluaciones a la condición, mientras que a la del ciclo externo `(primera <= ultima)` solamente se le hicieron dos evaluaciones. Esto significa, que el ciclo externo itera 2 veces, mientras que el ciclo interno itera 5 veces por cada iteración del externo.

En la primera iteración del ciclo externo, la variable `primera` inicia con un valor de 7; al entrar al cuerpo del ciclo interno, se calcula la tabla de este número, desde la fila 1 hasta la fila 5 y se van mostrando los resultados, observe las 3 últimas columnas de la tabla anterior. Una vez la variable `fila` llega al valor de 6, la condición del ciclo interno se hace falsa y se procede a incrementar, en dos unidades, el valor de la variable `primera`, tomando el valor de 9. El proceso se repite y esta vez se hace el cálculo para la tabla del 9, también desde la fila 1 hasta la fila 5.

Todo este proceso finaliza en el momento que la variable `fila` toma el valor de 6 y la variable `primera` alcanza el valor de 11, con lo cual, al evaluar las dos condiciones de los ciclos, se obtiene un resultado falso. En las dos últimas filas de la tabla de verificación se evidencia esa situación.

4.4. Estructura de repetición `Para-FinPara`

Igual que las dos estructuras anteriores, `Mientras-FinMientras` y `Haga-MientrasQue`, esta estructura también se usa para lograr que una instrucción o un conjunto de ellas se ejecuten repetidas veces.

De manera similar al `Mientras-FinMientras`, el `Para-FinPara` es un ciclo condicionado al inicio, lo cual implica que su ejecución está determinada por la evaluación de su condición. Puede suceder que el cuerpo del ciclo no llegue a ejecutarse cuando al evaluar por primera vez su condición resulte falsa.

Este ciclo es muy útil cuando se conoce de antemano, el número de iteraciones que se deben hacer. La cantidad de ejecuciones que realice, no dependerá de ninguna de las instrucciones del cuerpo del ciclo, es decir, no habrá una instrucción modificadora de condición dentro de ellas.

La forma general de esta estructura de repetición es presentada en los siguientes segmentos de los Algoritmos 4.29 y 4.30.

Algoritmo 4.29: Forma general (incremento) - Para-FinPara

```
1 Instrucción de inicialización
2 Para var = valor1 Hasta valor2 Incremento valor3
3   Instrucción-1
4   Instrucción-2
5   ...                    /* Cuerpo del ciclo */
6   Instrucción-n
7 FinPara
8 Instrucción externa
```

Algoritmo 4.30: Forma general (decremento)- Para-FinPara

```
1 Instrucción de inicialización
2 Para var = valor1 Hasta valor2 Decremento valor3
3   Instrucción-1
4   Instrucción-2
5   ...                    /* Cuerpo del ciclo */
6   Instrucción-n
7 FinPara
8 Instrucción externa
```

Las anteriores formas generales se interpretan así:

La Instrucción de inicialización se usa para dar un valor inicial a los contadores o acumuladores que se modificaran dentro del ciclo. En el caso de no tener este tipo de variables, no es necesario su uso.

Las instrucciones:

```
Para var = valor1 Hasta valor2 Incremento valor3
```

```
Para var = valor1 Hasta valor2 Decremento valor3
```

se interpretan de la siguiente manera:

- `var`: es una variable declarada de tipo `Entero` o `Caracter`. Esta es la variable de control del ciclo.
- `valor1`: puede ser una constante o una variable, cuyo valor se le asigna a `var`.
- `Hasta`: es un código que indica que `var` debe ir hasta un valor determinado. Se interpreta como si se hubiese establecido una de las siguientes condiciones:

  (`var <= valor2`), cuando se hacen incrementos.
  (`var >= valor2`), cuando se hacen decrementos.

- `valor2`: puede ser una constante o una variable, indica el valor hasta el cual llegará `var`.
- `Incremento` o `Decremento`: estos códigos señalan que `var` debe sufrir una modificación, aumentando o disminuyendo su valor, respectivamente.
- `valor3`: es una variable o constante en cuyo valor se modifica `var`.

Es importante resaltar que `valor1`, `valor2` y `valor3`, pueden ser variables o constantes. Cuando uno o más son variables, estas se deben inicializar antes del código `Para`, bien sea, con una expresión de asignación o mediante una instrucción `leer`. Si se trabajan como constantes, no requieren de inicialización.

La forma de operar del ciclo `Para`, es la siguiente: `var` se inicializa en `valor1`, seguidamente se hace la evaluación de una de las posibles condiciones (`var <= valor2` o `var >= valor2`), que están implícitas con el código `Hasta` y `valor2`. Si el resultado de esta evaluación es verdadero, se ejecutan las instrucciones del cuerpo del ciclo hasta encontrar la instrucción `FinPara` que devuelve el control al inicio del ciclo. Al regresar a la instrucción `Para`, se produce el incremento o decremento de `var`, de acuerdo al valor almacenado en `valor3`; una vez más se evalúa la condición, repitiéndose el proceso si el resultado es verdadero o ejecutándose la Instrucción externa, en caso contrario. Instrucción externa no hace parte del ciclo.

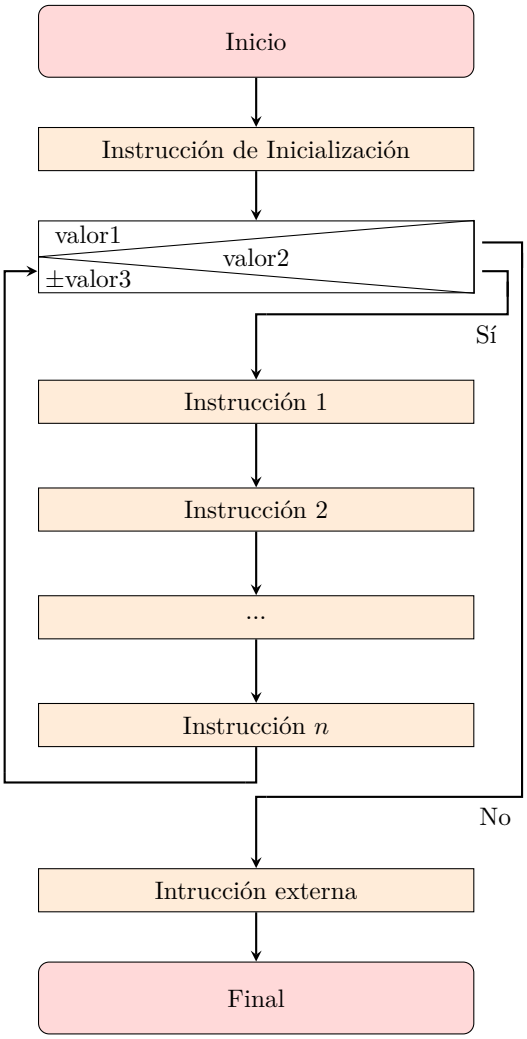
En estas formas generales no están presentes, de forma explícita, la Instrucción modificadora de condición como en los dos ciclos anteriores, su papel lo desempeña el código `Incremento` o `Decremento` acompañado de `valor3`.

Aclaración:



Al encontrarse por primera vez un código `Para`, se ejecutan la inicialización de la variable de control y la evaluación de la condición. El incremento o decremento no se realizan.

Para representar la estructura `Para-FinPara` con un diagrama de flujo, se usa la notación presentada en la Figura 4.31.



Se emplea:
$+valor3$ para indicar un `Incremento` y $-valor3$ un `Decremento`

Figura 4.31: Forma general del ciclo `Para-FinPara`

Aclaración:



Algunos de los problemas que se solucionan con una estructura `Mientras-FinMientras` o `Haga-MientrasQue`, no pueden ser solucionados con una estructura `Para-FinPara`.

Para explicar el funcionamiento del ciclo `Para-FinPara` se hará uso del mismo ejemplo empleado con las dos estructuras anteriores (Algoritmo 4.2 y 4.11), de esta forma usted podrá observar su diferencia. Ver segmento del Algoritmo 4.31.

Algoritmo 4.31: Imprimir los números del 1 al 10

```
1 Para numero = 1 Hasta 10 Incremento 1
2   imprimir( numero )
3 FinPara
```

Para una explicación más clara, se identificarán sus partes de acuerdo a la forma general expresada en párrafos anteriores.

- Línea 1: Inicio del ciclo
- Línea 2: Cuerpo del ciclo
- Línea 3: Fin del ciclo. Retorna el control a la línea 1.

En la primera línea se encuentra el inicio del ciclo, en el cual se ejecutan las siguientes acciones:

1. Se inicializa la variable `numero` en 1.
2. Se evalúa la condición del `Hasta`, que para este caso se interpreta como $numero <= 10$.
3. Luego de que se ejecute el cuerpo del ciclo, se incrementa la variable `numero` en 1 unidad.

La segunda línea, es el cuerpo del ciclo, el cual imprime el valor de `numero`. La última línea indica el final del ciclo y retorno a la línea 1, donde se produce el incremento de la variable `numero` y la evaluación de la condición implícita.

Las iteraciones terminan en el momento que `numero` tome el valor de 11, donde la evaluación de la condición resulta falsa.

En la Figura 4.32 se muestra el diagrama de flujo del Algoritmo 4.31.

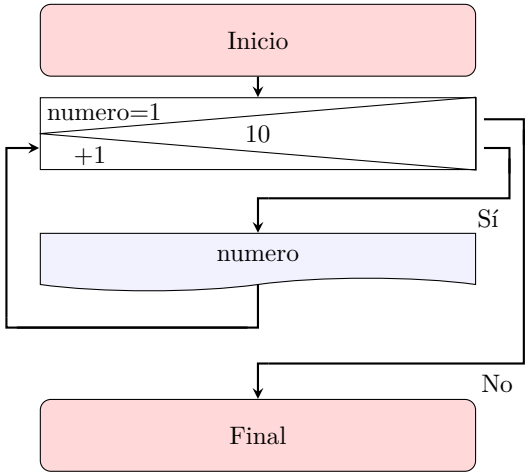


Figura 4.32: Diagrama de flujo para el Ejemplo 4.31

Aclaración:



El bloque de instrucciones o cuerpo del ciclo `Para-FinPara`, se ejecutará solamente si la evaluación de la condición implícita es verdadera.

El valor inicial (`valor1`) y el valor final (`valor2`) de la variable (`var`) que controla el ciclo, pueden establecerse mediante una variable o una constante.

.:Ejemplo 4.18. *Diseñe un algoritmo que genere e imprima la siguiente serie:*

$$1, 3, 5, 7, 9, 11, \ldots, n$$

Análisis del problema:

Este problema ya fue solucionado anteriormente, usando las dos estructuras de ciclo previamente estudiadas. El análisis completo a este problema lo encuentra en el Ejemplo 4.1. El Algoritmo 4.4 muestra la solución usando el ciclo `Mientras` - `FinMientras` y el Ejemplo 4.9 usa el ciclo `Haga-MientrasQue`, ver Algoritmo 4.13.

De acuerdo al análisis planteado, se propone el Algoritmo 4.32, mediante el ciclo `Para-FinPara`.

Algoritmo 4.32: Serie

```
1  Algoritmo Serie
2     /* Este algoritmo imprime la siguiente serie, de acuerdo
3        al número de términos que se le especifique:
4        1, 3, 5, 7, 9, 11, ..., n
5     */
6
7     // Declaración de variables
8     Entero cantidadTerminos, contadorNumeros, termino
9
10    // Dato disponible
11    imprimir( "Ingrese la cantidad de términos a generar: " )
12    leer( cantidadTerminos )
13
14    // Inicialización de variable
15    termino = 1
16
17    // Generación de la serie
18    Para contadorNumeros = 1 Hasta cantidadTerminos - 1
19     Incremento 1
20        imprimir( termino, ", " )
21        termino = termino + 2
22    FinPara
23
24    imprimir( termino )
25 FinAlgoritmo
```

Explicación del algoritmo:

La parte inicial del algoritmo es la misma que la de los Ejemplos 4.1 y 4.9. Los cambios se aprecian desde la inicialización de las variables, acá solo es necesario inicializar la variable `termino`.

La instrucción:

```
18    Para contadorNumeros = 1 Hasta cantidadTerminos - 1
19     Incremento 1
```

En la línea 18 empieza un proceso repetitivo condicionado al comienzo. Se inicializa en 1 a `contadorNumeros`. El código `Hasta` señala que las iteraciones del ciclo deben hacerse mientras se cumpla la condición implícita `contadorNumeros <= cantidadTerminos - 1`. En cada ejecución del ciclo, el código `Incremento` adiciona 1 unidad a `contadorNumeros`.

Si cada que se evalúe la condición, da un resultado verdadero, se imprime el acumulado de `termino` y luego se incrementa en dos unidades. Al arrojar un resultado falso, se termina el ciclo y seguidamente se ejecuta la instrucción de la línea 24 la cual imprime el último término se la serie sin imprimir la respectiva coma de separación, posteriormente termina la ejecución del algoritmo.

En la Figura 4.33 se muestra la solución del Ejemplo 4.18 mediante un diagrama de flujo.

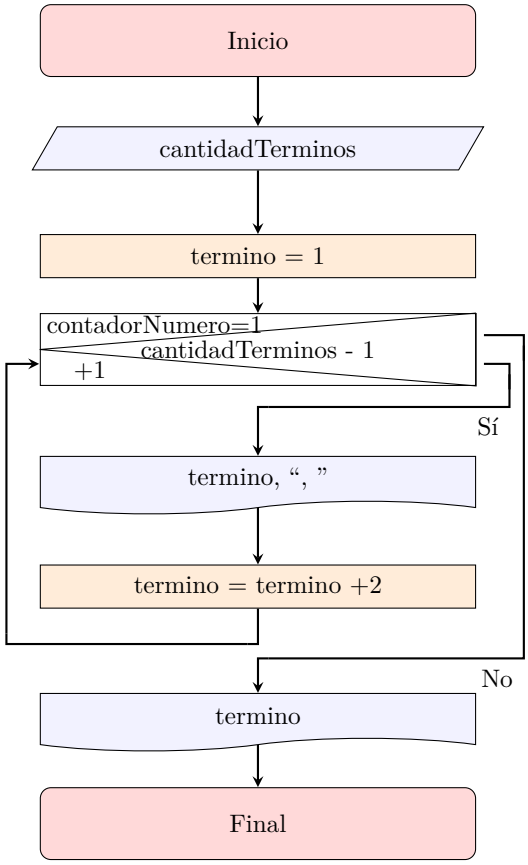


Figura 4.33: Diagrama de flujo del Algoritmo `Serie`

.:Ejemplo 4.19. *Construya un algoritmo que imprima tres columnas de números, conforme a la Tabla 4.11, en donde el valor de n será proporcionado por el usuario.*

1	1	2
2	4	6
3	9	12
4	16	20
5	25	30
...	...	...
n	...	...

Tabla 4.10: Tabla de datos del Ejemplo 4.19

Análisis del problema:

- **Resultados esperados:** se espera que el algoritmo genere la tabla mostrada en el enunciado.
- **Datos disponibles:** con la cantidad de números o cantidad de filas de la tabla, se podrá hacer el proceso, dato que será suministrado por el usuario del algoritmo.
- **Proceso:** luego de la lectura de la cantidad de números, se procede a realizar la generación de ellos y los cálculos correspondientes.

  La primera columna de la tabla está conformada por los números desde el 1 hasta n, en forma consecutiva. En la segunda columna se presentan los cuadrados de cada uno de los números de la primera; en la última se muestra la suma de cada número de la primera columna, con su respectivo cuadrado.

  Para la generación de esta tabla se requiere de un proceso iterativo que inicia un contador en 1 y termina en n; en cada una de sus vueltas, se debe generar el cuadrado de dicho contador y la suma correspondiente, así mismo se hace la impresión de los resultados.
- **Variables requeridas:**

  - `cantidadNumeros`: indicará el total de filas o números que tendrá la tabla (n).
  - `numero`: esta variable tendrá la función de ir almacenando los números desde 1 hasta la cantidad que se especifique; con ella se calculará el cuadrado y la suma que se requiere para generar los números de la tabla del enunciado.

De acuerdo al análisis planteado, se propone el Algoritmo 4.33.

Algoritmo 4.33: Tabla

```
1  Algoritmo Tabla
2    // Declaración de variables
3    Entero cantidadNumeros, numero
4
5    // Dato disponible
6    imprimir( "Ingrese la cantidad de números: " )
7    Haga
8      leer( cantidadNumeros )
9    MientrasQue( cantidadNumeros < 0 )
10   // Se valida para que no reciba valores negativos
11
12   // Generación de los valores
13   Para numero = 1 Hasta cantidadNumeros Incremento 1
14       imprimir( numero, " ",
15                 numero * numero, " ",
16                 numero + numero * numero )
17   FinPara
18 FinAlgoritmo
```

Explicación del algoritmo:

Lo primero que se hizo fue declarar las variables necesarias y obtener la cantidad de números que tendrá la tabla, validando que no se ingresen números negativos.

```
3    Entero cantidadNumeros, numero
4
5    // Dato disponible
6    imprimir( "Ingrese la cantidad de números: " )
7    Haga
8      leer( cantidadNumeros )
9    MientrasQue( cantidadNumeros < 0 )
```

Posteriormente se procede a crearla; para ello se usa un ciclo `Para`, el cual inicializa la variable `numero` con el valor de 1 e itera mientras que el contenido de esta variable sea menor o igual al valor de la variable `cantidadTerminos`.

```
13   Para numero = 1 Hasta cantidadNumeros Incremento 1
14       imprimir( numero, " ",
15                 numero * numero, " ",
16                 numero + numero * numero )
17   FinPara
```

En cada iteración del ciclo `Para`, se imprime el valor de la variable

numero, su cuadrado y la suma de estos dos valores. Luego de la impresión se encuentra el FinPara, se regresa al Para y la instrucción Incremento 1 incrementa a la variable numero en 1 unidad. Una vez más se evalúa la condición, mientras el valor de numero sea menor o igual al de cantidadNumeros, se ejecuta el cuerpo del ciclo.

El cuerpo del ciclo está conformado solamente por la instrucción imprimir. Observe que los cálculos se hacen directamente dentro de ella:

```
14        imprimir( numero, " ",
15                  numero * numero, " ",
16                  numero + numero * numero )
```

Es una forma correcta de hacerlo, aunque también lo es incluir nuevas variables para almacenar los cálculos, tal como se muestra a continuación:

```
    Para numero = 1 Hasta cantidadNumeros Incremento 1
       cuadrado = numero * numero
       suma     = numero + cuadrado
       imprimir( numero, " ", cuadrado, " ", suma )
    FinPara
```

A nivel del usuario, ambas soluciones son equivalentes, sin embargo, a nivel interno existen diferencias, por ejemplo: en la primera solución es necesario hacer dos veces la misma operación numero * numero, mientras que en la segunda solución solo es necesario realizar una vez esta operación. Para un algoritmo como el presente, esto no tiene un mayor impacto, pero en soluciones que realicen un gran número de iteraciones la implicaciones en tiempo pueden ser notorias.

.:Ejemplo 4.20. *Diseñe un algoritmo que permita calcular la siguiente función:*

$$f(x) = x^3 + x^2 - 5$$

Para x *con valores desde* 0 *hasta* n*, con incrementos de a* 2*.*

Análisis del problema:

- **Resultados esperados:** cada uno de los valores que va tomando x, con el respectivo resultado del cálculo de la función.
- **Datos disponibles:** n (el valor máximo que tomará x).
- **Proceso:** en primer lugar, se solicita el valor n (máximo que tendrá x). Seguidamente se debe hacer el cálculo de la función y la impresión

de los resultados esperados, todo esto en un proceso repetitivo donde x debe iniciar en 0 y llegar hasta el valor máximo (n) que indique el usuario del algoritmo; en cada iteración x debe incrementarse en 2 unidades.

- **Variables requeridas:**
  - valorX: almacena el valor máximo que tomará x, este dato será suministrado por el usuario del algoritmo. Esta variable representa la n.
  - funcion: variable que almacenará el cálculo de la función.
  - x: actuará como la variable de control del ciclo, tomará valores entre 0 y n.

De acuerdo al análisis planteado, se propone el Algoritmo 4.34.

Algoritmo 4.34: FuncionX

```
1  Algoritmo FuncionX
2    // Declaración de variables
3    Entero valorX, x, funcion
4
5    // Dato disponible
6    imprimir( "Ingrese el máximo valor para x: " )
7    Haga
8      leer( valorX )
9    MientrasQue( valorX < 0 )
10   // Se valida para que no reciba valores negativos
11
12   // Cálculo de la función
13   Para x = 0 Hasta valorX Incremento 2
14     funcion = x^3 + x^2 - 5
15     imprimir( "Para x = ", x, " f(x) = ", funcion )
16   FinPara
17 FinAlgoritmo
```

Al ejecutar el algoritmo:

```
Ingrese el máximo valor para x: 10

Para x = 0  fx(x) = -5
Para x = 2  fx(x) = 7
Para x = 4  fx(x) = 75
Para x = 6  fx(x) = 247
Para x = 8  fx(x) = 571
Para x = 10 fx(x) = 1095
```

Explicación del algoritmo:

Después de declarar las respectivas variables, el algoritmo solicita el ingreso del máximo valor que tendrá x, para el caso es llamado *valorX* y se hace la validación de tal forma que no acepte números negativos.

```
3   Entero valorX, x, funcion
4
5   // Dato disponible
6   imprimir( "Ingrese el máximo valor para x: " )
7   Haga
8     leer( valorX )
9   MientrasQue( valorX < 0 )
```

La instrucción `Para x = 0 Hasta valorX Incremento 2`, inicializa a la variable `x` en 0 y la incrementa mientras sea menor o igual a `valorX`; en esta solución algorítmica los incrementos se hacen en 2 unidades, a diferencia de los ejemplos anteriores que se hacían en 1 unidad.

En cada iteración del ciclo, se calcula la función y se imprime el resultado.

```
13  Para x = 0 Hasta valorX Incremento 2
14    funcion = x^3 + x^2 - 5
15    imprimir( "Para x = ", x, " f(x) = ", funcion )
16  FinPara
```

.:Ejemplo 4.21. *El Código Americano Estándar para el Intercambio de Información o mejor conocido como código ASCII, por su nombre en inglés, American Standard Code for Information Interchange, permite unificar la representación de los caracteres alfanuméricos y códigos de control en los computadores. Antes del surgimiento de este código, cada familia de computadores utilizaba una regla diferente para la representación.*

Son 256 códigos ASCII[12] *los que integran este conjunto; van desde el 0 al 255, en su representación numérica decimal. En la Tabla 4.11 se muestran 25 de ellos.*

Los primeros 32 códigos (del 0 al 31) se conocen como códigos de control, están reservados para controlar algunos dispositivos, por ejemplo, entre ellos está el salto de línea, la tecla escape, entre otros. Estos códigos no son imprimibles.

[12]En esta dirección de internet, puede consultar todos los códigos ASCII: http://www.asciitable.com/. De igual forma, como dato curioso, a través de esta otra http://www.asciiarte.com/, encontrará dibujos realizados con estos caracteres.

ASCII	Carácter	ASCII	Carácter	ASCII	Carácter	ASCII	Carácter
48	'0'	49	'1'	50	'2'	51	'3'
52	'4'	53	'5'	54	'6'	55	'7'
56	'8'	57	'9'	58	':'	59	';'
60	'<'	61	'='	62	'>'	63	'?'
64	'@'	...		...		...	
...		...		...		...	
91	'['	92	'\'	93	']'	94	'^'
...		...		...		...	
123	'{'	124	'|'	125	'}'	126	'~'

Tabla 4.11: Tabla de datos del Ejemplo 4.19

De acuerdo al anterior contexto diseñe un algoritmo que genere e imprima los códigos ASCII desde el 32 al 255, en su representación numérica decimal y su carácter equivalente. Deben imprimirse en orden inverso.

Análisis del problema:

- **Resultados esperados:** listado de los códigos ASCII, desde el 255 hasta el 32.
- **Datos disponibles:** para este algoritmo no se requiere ningún dato proveniente del usuario.
- **Proceso:** se requiere de un ciclo, que sea controlado a través de una variable inicializada en 255; debe disminuir de 1 en 1 y mientras llega a 32, debe generar e imprimir los códigos ASCII.
- **Variables requeridas:**
  - `decimal`: tendrá la función de ser la variable de control del ciclo. Tomará el valor inicial de 255 e irá decrementando en 1 unidad. Estos valores son la representación numérica en decimal de cada uno de los códigos.
  - `ascii`: almacenará el carácter correspondiente a cada uno de los códigos numéricos decimales.

De acuerdo al análisis planteado, se propone el Algoritmo 4.35.

Algoritmo 4.35: Codigos

```
1  Algoritmo Codigos
2    // Declaración de variables
3    Entero decimal
4    Caracter ascii
5
6    // Generación e impresión de los códigos ASCII
7    Para decimal = 255 Hasta 32 Decremento 1
8      ascii = decimal
9      imprimir( decimal, " = ", ascii )
10   FinPara
11 FinAlgoritmo
```

Explicación del algoritmo:

Este es un ejemplo no hay entrada de datos, por lo tanto, este algoritmo tiene una función específica y única, siempre que se ejecute mostrará el mismo resultado.

La instrucción `Para decimal = 255 Hasta 32 Decremento 1`, inicializa la variable `decimal` en `255`, con decremento de 1 unidad en cada iteración; esto ocurre mientras `decimal` tenga un valor mayor o igual a 32.

En el cuerpo del ciclo se encuentra esta asignación: `ascii = decimal`. La variable `ascii` es de tipo `Caracter` y la variable `decimal` es de tipo `Entero`; algunos lenguajes soportan este tipo de operación de asignación solamente utilizando el signo igual ('='), en cambio otros requieren el uso de alguna función especial para tal fin. Al hacer esta asignación, internamente la computadora realiza la conversión entre los tipos de datos; para este caso convierte el valor numérico decimal, en su correspondiente dato de tipo `Caracter`.

La siguiente línea en el cuerpo del ciclo, imprime el valor decimal y su correspondiente carácter.

```
9      imprimir( decimal, " = ", ascii )
```

Luego de la impresión, el código `FinPara` indica el final del ciclo, el control regresa al código `Para` (línea 7) y la variable `decimal` decrementa su valor en 1 unidad y una vez más se vuelve a evaluar la condición implícita, si el valor de `decimal` es mayor o igual a 32, el ciclo vuelve a iterar. Cuando esta condición no se cumpla, finaliza el ciclo y se ejecuta la instrucción que hay debajo del `FinPara`, por lo tanto, finaliza el algoritmo.

.:Ejemplo 4.22. *Elabore un algoritmo, que genere e imprima las letras del abecedario de la siguiente forma:*

```
Z
Z Y
Z Y X
Z Y X W
Z Y X W V
Z Y X W V U
Z Y X W V U T
Z Y X W V U T S
Z Y X W V U T S R
Z Y X W V U T S R Q
Z Y X W V U T S R Q P
Z Y X W V U T S R Q P O
Z Y X W V U T S R Q P O N
Z Y X W V U T S R Q P O N M
Z Y X W V U T S R Q P O N M L
Z Y X W V U T S R Q P O N M L K
Z Y X W V U T S R Q P O N M L K J
Z Y X W V U T S R Q P O N M L K J I
Z Y X W V U T S R Q P O N M L K J I H
Z Y X W V U T S R Q P O N M L K J I H G
Z Y X W V U T S R Q P O N M L K J I H G F
Z Y X W V U T S R Q P O N M L K J I H G F E
Z Y X W V U T S R Q P O N M L K J I H G F E D
Z Y X W V U T S R Q P O N M L K J I H G F E D C
Z Y X W V U T S R Q P O N M L K J I H G F E D C B
Z Y X W V U T S R Q P O N M L K J I H G F E D C B A
```

Análisis del problema:

- **Resultados esperados:** imprimir las letras del alfabeto, conforme a lo solicitado en el enunciado.
- **Datos disponibles:** no se requiere de ninguna entrada de datos.
- **Proceso:** se requiere imprimir el alfabeto en orden descendente, de la `Z` a la `A`. La primera fila está conformada únicamente por la letra `Z`, la segunda por las letras `Z` y `Y`, en la tercera se encuentran las letras `Z Y` y `X`; de acuerdo al patrón que se observa, en cada nueva fila se imprime una letra más. La impresión termina cuando se complete todo el alfabeto.

  Este proceso requiere de dos ciclos anidados. El interno imprimirá las letras de cada fila, mientras el externo permitirá el avance de las filas.

De acuerdo al proceso explicado, ambos ciclos deberán estar diseñados para trabajar con decrementos, iniciando en la letra `'Z'` y descendiendo hasta la letra `'A'`.

- **Variables requeridas:**
  - `letraFila`: controla el ciclo externo. Con esta variable se podrá tener control para el avance de las filas.
  - `letra`: variable de control del ciclo interno. Almacenará las letras que se van imprimiendo en cada fila.

De acuerdo al análisis planteado, se propone el Algoritmo 4.36.

Algoritmo 4.36: Alfabeto

```
1 Algoritmo Alfabeto
2   // Declaración de variables
3   Caracter letraFila, letra
4
5   // Generación del alfabeto
6   Para  letraFila = 'Z' Hasta 'A' Decremento 1
7     Para letra = 'Z' Hasta letraFila Decremento 1
8       imprimir( " ", letra )
9     FinPara
10    // Salto de línea
11  FinPara
12 FinAlgoritmo
```

Explicación del algoritmo:

Al igual que el algoritmo anterior, en este, tampoco hay datos de entrada.

Con base al análisis realizado para la solución del problema, se escribieron dos ciclos `Para` anidados.

El primer ciclo presenta la siguiente instrucción:

```
6   Para  letraFila = 'Z' Hasta 'A' Decremento 1
```

Antes de entrar en detalle de su modo de operar, es importante dar claridad y aprender a distinguir entre las siguientes expresiones, que son comunes dentro de los algoritmos:

```
1.  letraFila = Z
2.  letraFila = 'Z'
```

En la expresión número 1, se está trabajando con dos variables que deben estar declaradas. En la variable `letraFila` se almacena el valor de la variable `Z`.

En la expresión número 2, se trata de una variable y un dato, en este caso solamente la variable `letraFila` debe estar declarada. El carácter o letra 'Z' se almacena en la variable `letraFila`.

En el caso de este ejemplo, la expresión 2 es la que se codificó en los dos ciclos `Para`.

```
6   Para  letraFila = 'Z' Hasta 'A' Decremento 1
7     Para letra = 'Z' Hasta letraFila Decremento 1
8       imprimir( " ", letra )
9     FinPara
10    // Salto de línea
11  FinPara
```

El ciclo externo inicializa la variable `letraFila` con la letra 'Z' y en cada iteración va decrementando en una letra hasta llegar a la 'A'. Cada que entra al cuerpo del ciclo ejecuta el `Para` interno, este inicializa su variable de control en 'Z' (`letra = 'Z'`) y en cada iteración la decrementa hasta alcanzar el valor de `letraFila`. Cada que se ejecuta el ciclo interno, se imprime el valor de la variable `letra`; en la primera iteración imprime la letra 'Z', luego de que este ciclo termina su ejecución debe producirse un salto de línea, es decir, se pasa al siguiente renglón para volver a iniciar la impresión.

En este ejemplo el salto de línea se hizo a manera de comentario:

```
10    // Salto de línea
```

Aclaración:



Cada uno de los lenguajes que Usted aprenderá tendrán su propia instrucción para ejecutar el salto de línea, como por ejemplo, el carácter especial[a] '\n' suele emplearse para este fin.

[a]Se denomina especial por ser una secuencia de escape y se identifica debido al uso de la barra invertida (\) antes de una letra. Diversas letras tienen usos diferentes.

Cuando se regresa a la instrucción `Para` del ciclo externo, `letraFila` se decrementa y su valor ya no es 'Z', sino 'Y', su condición sigue siendo verdadera, aún no se ha alcanzado el valor de 'A'. Se vuelve a ejecutar el ciclo interno y esta vez se imprime la 'Z' y luego la letra 'Y' en el mismo renglón, esta vez el ciclo se ejecuta dos veces. Luego se produce el salto de línea.

Para la tercera evaluación de la condición del ciclo externo, la variable `letraFila` ha decrementado su valor a 'X', dando aún un resultado verdadero. Nuevamente se ejecuta el ciclo interno, esta vez debe hacer 3 iteraciones, en la primera imprime la letra 'Z', en la segunda la letra 'Y' y en la tercera la letra 'X', todas sobre un mismo renglón. Seguidamente, una vez más se produce el cambio de línea.

En este momento se debe estar visualizando la siguiente impresión:

```
Z
Z Y
Z Y X
```

Este proceso continúa mientras la variable `letraFila`, del ciclo externo, no alcance el valor de la letra A y no se haya impreso todo el alfabeto por parte del ciclo interno.

En la Figura 4.34 se muestra la solución del Ejemplo 4.22 mediante un diagrama de flujo.

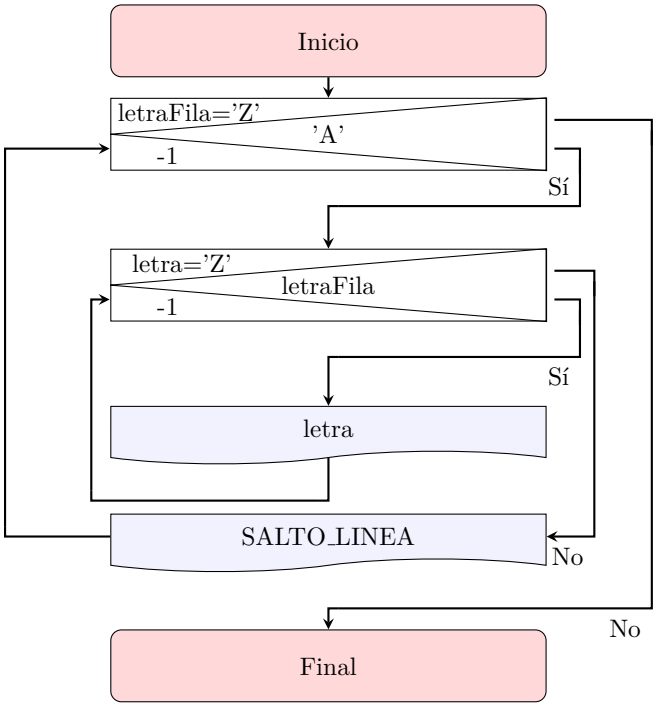


Figura 4.34: Diagrama de flujo del Algoritmo `Alfabeto`

.:Ejemplo 4.23. *Construya una solución algorítmica que simule el comportamiento de un reloj digital para un día; con horas, minutos y segundos. Debe trabajarse con un formato de 24 horas.*

Análisis del problema:

- **Resultados esperados:** mostrar la simulación de un reloj digital en un formato de 24 horas: HH:MM:SS.
- **Datos disponibles:** para este ejemplo no se requiere que sea ingresado ningún valor. Sin embargo, es de conocimiento general que un minuto tiene 60 segundos y que una hora tiene 60 minutos.
- **Proceso:** se deben diseñar 3 ciclos anidados. El externo controlará las horas, el del medio los minutos y el más interno los segundos. La hora debe iniciar desde 0 e ir hasta 23; los minutos y segundos irán desde 0 hasta 59, cada que uno de ellos llegue a 59 se cambiará a la hora o minuto siguiente, respectivamente.
- **Variables requeridas:**
  - `hora`: será el índice para el ciclo externo, tomará valores entre 0 y 23.
  - `minuto`: almacena valores entre 0 y 59, esta variable controlará el ciclo del medio.
  - `segundo`: con valores entre 0 y 59, se usará para controlar el ciclo interno.

De acuerdo al análisis planteado, se propone el Algoritmo 4.37.

Algoritmo 4.37: Reloj

```
1  Algoritmo Reloj
2    // Declaración de variables
3    Entero hora, minuto, segundo
4
5    // Simulación del reloj
6    Para hora = 0 Hasta 23 Incremento 1
7      Para minuto = 0 Hasta 59 Incremento 1
8        Para segundo = 0 Hasta 59 Incremento 1
9          imprimir( hora, ":", minuto, ":", segundo )
10       FinPara
11     FinPara
12   FinPara
13 FinAlgoritmo
```

Explicación del algoritmo:

Con base al análisis realizado, se codificaron 3 ciclos anidados.

La forma de operar de estos 3 ciclos es muy sencilla. El ciclo externo inicializa la variable `hora` en 0, luego el control lo asume el ciclo intermedio que inicializa la variable `minuto` en 0 y le entrega el control al ciclo interno que inicializa la variable `segundo` en 0; este último ciclo ejecuta la instrucción `imprimir( hora, ":", minuto, ":", segundo )`. Se hacen 60 iteraciones de esta instrucción, de acuerdo a lo especificado en el algoritmo:

```
8       Para segundos = 0 Hasta 59 Incremento 1
9         imprimir( hora, ":", minuto, ":", segundo )
10      FinPara
```

Cuando la variable `segundo` tome el valor de 60, la condición `(segundo <= 59)` termina la ejecución de este ciclo interno. De acuerdo a lo descrito, en este momento se observa la siguiente salida del algoritmo:

```
00:00:00
00:00:01
00:00:02
...
00:00:59
```

En el momento que el ciclo interno deja de ejecutarse, el control vuelve a asumirlo el ciclo intermedio, incrementando el valor de `minuto` en 1 unidad. La condición `(minuto <= 59)` vuelve a ser verdadera y el ciclo interno vuelve a tomar protagonismo, repitiéndose todo el proceso explicado en los párrafos anteriores.

Teniendo en cuenta que la variable `minuto` tiene almacenado el valor de 1, en este momento, los resultados de la instrucción `imprimir` son los siguientes:

```
00:01:00
00:01:01
00:01:02
...
00:01:59
```

Cuando la variable `minuto` se incremente 60 veces, se pasará el control al ciclo externo, donde se incrementa la variable `hora` en 1. Seguidamente se pasa, una vez más, el control al ciclo intermedio, inicializando la variable `minuto` en 0 y ejecutándose su cuerpo de ciclo. El control lo toma el ciclo

interno, se inicializa la variable `segundo` en 0 y se inicia el proceso de mostrar los segundos y minutos para una nueva hora.

Todo este proceso termina cuando las condiciones de los tres ciclos den un resultado falso, en forma simultánea. La última salida que imprime el algoritmo es:

```
23:59:59
```

En la Figura 4.35 se muestra la solución del Ejemplo 4.23 mediante un diagrama de flujo.

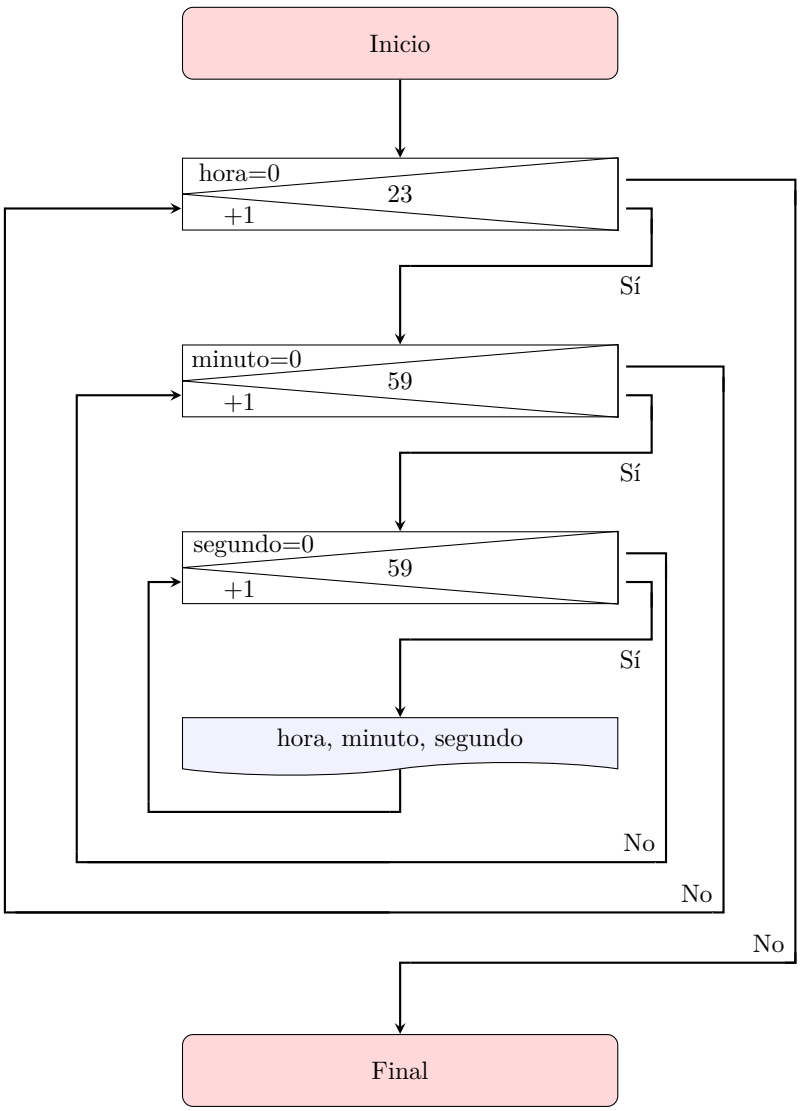


Figura 4.35: Diagrama de flujo del Algoritmo `Reloj`

.:Ejemplo 4.24. *Se requiere de un algoritmo que le permita a un niño repasar las tablas de multiplicar, del 1 al 20. El niño podrá indicar que tabla desea repasar y el algoritmo empezará a preguntar los resultados, desde la fila uno hasta la fila 10; si la respuesta es correcta se mostrará un mensaje de felicitaciones, en caso contrario mostrará el resultado, acompañado de un mensaje que informe la situación.*

Por cada tabla que el niño repase se debe dar una calificación, de acuerdo al número de respuestas correctas (por cada respuesta correcta se asigna un punto), basada en la escala de la Tabla 4.12.

Aciertos	**Valoración**
De 0 a 5	Insuficiente
6 o 7	Aceptable
8 o 9	Sobresaliente
10	Excelente

Tabla 4.12: Tabla de valoración - Ejercicio 4.24

El niño podrá repasar la cantidad de veces que así lo quiera.

Análisis del problema:

- **Resultados esperados:** un mensaje de acuerdo a la respuesta del niño y la calificación obtenida en cada tabla.
- **Datos disponibles:** el número de la tabla que desea repasar.
- **Proceso:** se requiere de dos procesos repetitivos anidados. El interno debe calcular la tabla que el niño solicite, desde la fila 1 hasta la fila 10. Igualmente, mostrará un mensaje de acuerdo a la respuesta del niño (Ver Figura 4.36).

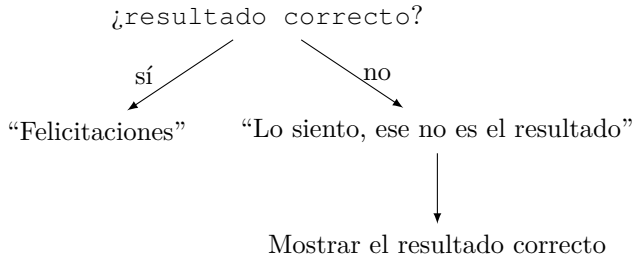


Figura 4.36: Árbol de decisión del Ejemplo 4.23 - Respuesta

Esta misma decisión, servirá para contar los aciertos o desaciertos que el niño tenga al responder.

Por último, se espera que se informe la valoración obtenida en cada tabla de multiplicar, de acuerdo al número de aciertos, para ello se deben contemplar los rangos establecidos en la Tabla 4.12. Las decisiones que se deben establecer se observan en la Figura 4.37.

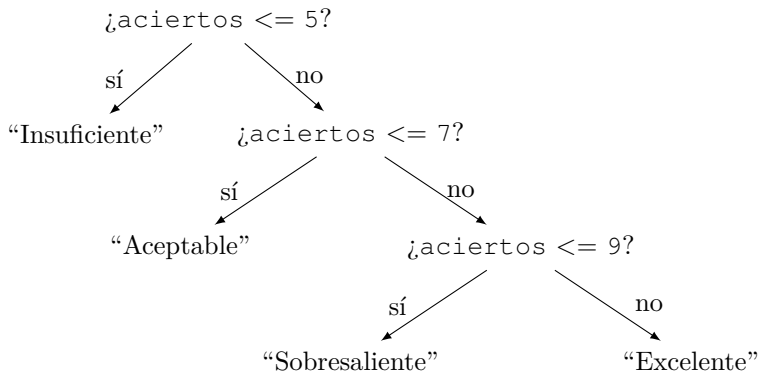


Figura 4.37: Árbol de decisión del Ejemplo 4.23 - Valoración

Todo este proceso debe anidarse dentro de otro ciclo. En este ciclo externo, adicionalmente se debe solicitar la tabla que el niño repasará, se inicializan los contadores de `aciertos` y `desaciertos`; por último se preguntará si se desea calcular una nueva tabla.

- **Variables requeridas:**
  - `tabla`: almacena el número de la tabla que el niño va a repasar.
  - `contadorFilas`: esta variable contará las filas de cada tabla, tomará valores entre 1 y 10.
  - `producto`: en esta variable se calcula el resultado de cada fila de la tabla.
  - `respuesta`: almacena la respuesta que ingrese el niño, como resultado de la multiplicación.
  - `aciertos`: almacena la cuenta de las respuestas correctas.
  - `desaciertos`: contador para las respuestas incorrectas.
  - `seguir`: centinela para controlar el ciclo externo, recibirá la respuesta si se desea continuar o terminar con la ejecución del algoritmo.

En las Figuras 4.38, 4.39, 4.40 y 4.41 se muestra la solución del Ejemplo 4.24 mediante un diagrama de flujo.

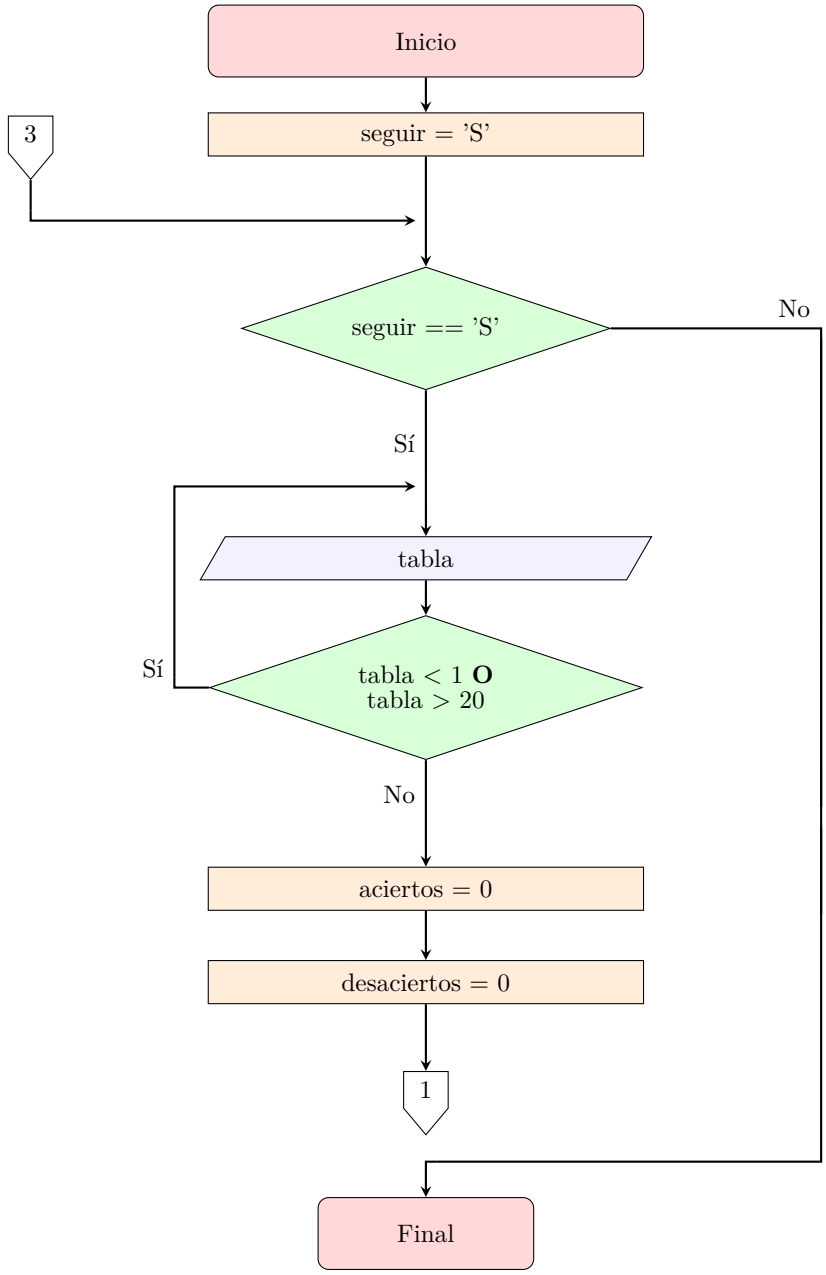


Figura 4.38: Diagrama de flujo del Algoritmo `JuegoTablas` - Parte 1

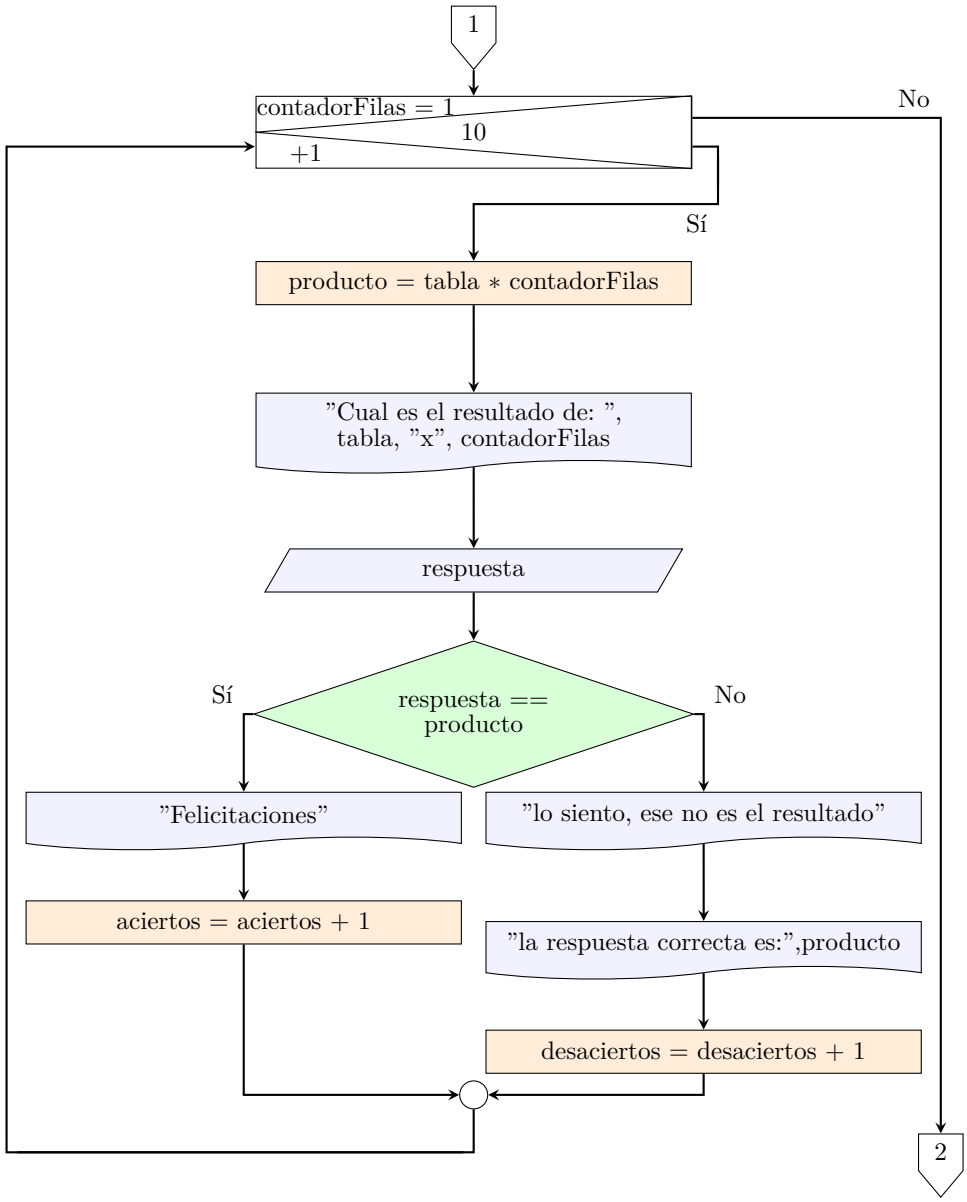


Figura 4.39: Diagrama de flujo del Algoritmo `JuegoTablas` - Parte 2

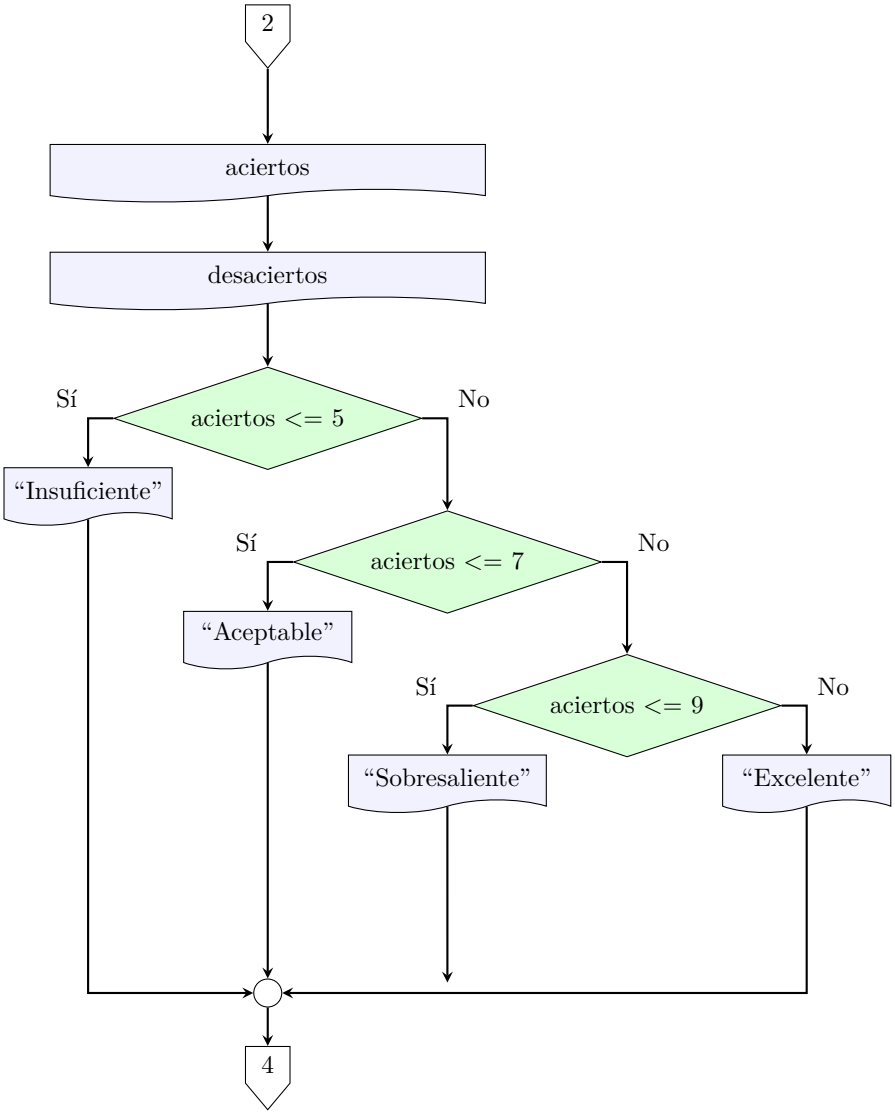


Figura 4.40: Diagrama de flujo del Algoritmo `JuegoTablas` - Parte 3

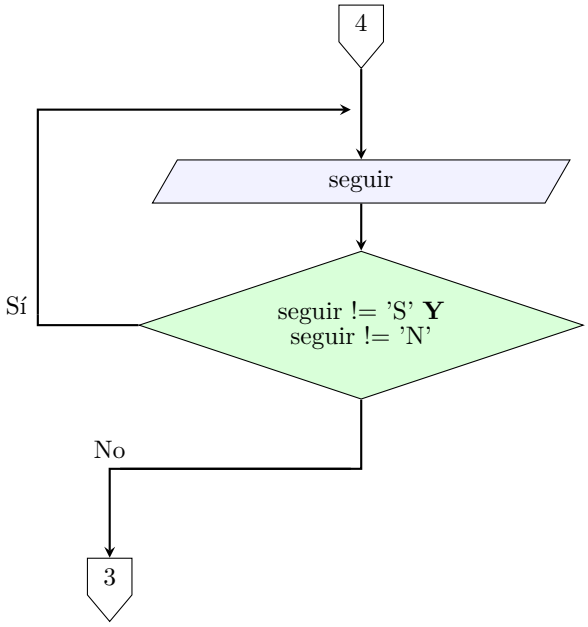


Figura 4.41: Diagrama de flujo del Algoritmo `JuegoTablas` - Parte 4

El Algoritmo 4.38 es la propuesta en pseudocódigo para la solución del juego de las tablas de multiplicar.

Algoritmo 4.38: JuegoTablas

```
1  Algoritmo JuegoTablas
2    // Declaración de variables
3    Entero   tabla, contadorFilas, producto,
4             respuesta, aciertos, desaciertos
5    Caracter seguir
6
7    seguir = 'S'
8    Mientras( seguir == 'S' O seguir == 's' )
9
10     imprimir( "Con cuál tabla desea jugar?: " )
11     Haga
12       leer( tabla )
13     MientrasQue( tabla < 1 O tabla > 20 )
14
15     aciertos = 0
16     desaciertos = 0
17     Para contadorFilas = 1 Hasta 10 Incremento 1
18       producto = tabla * contadorFilas
```

```
19
20        imprimir( "Escriba el resultado de ", tabla,
21                  " x ", contadorFilas )
22        leer( respuesta )
23
24        Si( respuesta == producto ) Entonces
25          imprimir( "Felicitaciones" )
26          aciertos = aciertos + 1
27        SiNo
28          imprimir( "Lo siento, ese no es el resultado" )
29          imprimir( "La respuesta correcta es: ", producto )
30          desaciertos = desaciertos + 1
31        FinSi
32      FinPara
33
34      imprimir( "Aciertos: ", aciertos )
35      imprimir( "Desaciertos: ", desaciertos )
36
37      Si( aciertos <= 5 )
38        imprimir( "Insuficiente" )
39      SiNo
40        Si( aciertos <= 7 )
41          imprimir ("Aceptable")
42        SiNo
43          Si( aciertos <= 9 )
44            imprimir( "Sobresaliente" )
45          SiNo
46            imprimir( "Excelente" )
47          FinSi
48        FinSi
49      FinSi
50
51      imprimir( "¿Desea volver a jugar [S] o [N]?: " )
52      Haga
53        leer( seguir )
54      MientrasQue( seguir != 'S' Y seguir != 'N' Y
55                   seguir != 's' Y seguir != 'n' )
56
57    FinMientras
58  FinAlgoritmo
```

Explicación del algoritmo:

En esta solución algorítmica se usaron las 3 estructuras de ciclo estudiadas en este capítulo. A cada una se le dio una función acorde para lo que están concebidas.

Se usó un ciclo `Mientras-FinMientras`, con el propósito de ejecutar repetidamente todo el proceso. Su ejecución se realizará mientras el niño

responda con una letra 'S' a la pregunta: "¿Desea volver a jugar [S] o [N]?:". Esta estructura repetitiva, es el ciclo externo en este algoritmo.

Los ciclos Haga-MientrasQue, fueron útiles en la validación de la entrada de los datos, por ejemplo, en la lectura del número de la tabla con que el niño va a jugar, se controla que se ingrese una tabla entre 1 y 20, requisito que fue dado en el enunciado del problema.

```
10      imprimir( "Con cuál tabla desea jugar?: " )
11      Haga
12        leer( tabla )
13      MientrasQue( tabla < 1 O tabla > 20 )
```

De igual forma, el ciclo se utilizó en la validación de la lectura de la respuesta a la pregunta final, con la cual se controla que la variable seguir acepte únicamente la letra 'S' o 'N', bien sea en mayúscula o minúscula.

```
51      imprimir( "¿Desea volver a jugar [S] o [N]?: " )
52      Haga
53        leer( seguir )
54      MientrasQue( seguir != 'S' Y seguir != 'N' Y
55                   seguir != 's' Y seguir != 'n' )
```

Se aprovechó la forma de operar del ciclo Para, con el fin de calcular la tabla que el niño seleccione. Este ciclo se encuentra anidado dentro del Mientras-FinMientras externo. En el cuerpo del ciclo Para se calcula la tabla, se le hace la pregunta al niño sobre el resultado de la multiplicación y se toma la decisión si la respuesta fue acertada o no; en caso afirmativo se imprime un mensaje de "Felicitaciones" y se cuenta un acierto. En caso contrario se informa que esa no es la respuesta acertada, se da el resultado correcto y se incrementa el contador de desaciertos.

```
15      aciertos = 0
16      desaciertos = 0
17      Para contadorFilas = 1 Hasta 10 Incremento 1
18        producto = tabla * contadorFilas
19
20        imprimir( "Escriba el resultado de ", tabla,
21                  " x ", contadorFilas )
22        leer( respuesta )
23
24        Si( respuesta == producto ) Entonces
25          imprimir( "Felicitaciones" )
26          aciertos = aciertos + 1
27        SiNo
28          imprimir( "Lo siento, ese no es el resultado" )
29          imprimir( "La respuesta correcta es: ", producto )
```

```
30          desaciertos = desaciertos + 1
31        FinSi
32      FinPara
```

El proceso descrito en el párrafo anterior se ejecuta 10 veces, que equivalen al número de filas de la tabla. Una vez el ciclo `Para` termine de iterar, se continua con la ejecución del algoritmo en el ciclo externo, es decir, en el ciclo `Mientras-FinMientras`.

Seguidamente se informan los `aciertos` y `desaciertos`.

```
34      imprimir( "Aciertos: ", aciertos )
35      imprimir( "Desaciertos: ", desaciertos )
```

Se toma la decisión para otorgar la calificación, para ello se empleó un conjunto de decisiones anidadas, que son parte del cuerpo del ciclo externo del algoritmo.

```
37      Si( aciertos <= 5 )
38        imprimir( "Insuficiente" )
39      SiNo
40        Si( aciertos <= 7 )
41          imprimir ("Aceptable")
42        SiNo
43          Si( aciertos <= 9 )
44            imprimir( "Sobresaliente" )
45          SiNo
46            imprimir( "Excelente" )
47          FinSi
48        FinSi
49      FinSi
```

Antes de finalizar las instrucciones del ciclo `Mientras-FinMientras` se le pregunta al niño si desea volver a jugar o no, la respuesta es almacenada en la variable `seguir`; luego se encuentra el `FinMientras` y el control regresa al `Mientras` donde se evalúa la condición:

```
8   Mientras( seguir == 'S' O seguir == 's' )
```

Si el resultado es verdadero, se vuelve a ejecutar todo el proceso. El juego termina en el momento que el niño responda con una letra `'N'`, al cuestionamiento si desea volver a jugar.

.:Ejemplo 4.25. *Una Universidad está interesada en tener una solución algorítmica que le permita conocer de sus estudiantes, de primer semestre que acaban de concluir el periodo académico, la siguiente información:*

- *El mayor y el menor promedio general.*
- *Cantidad total de estudiantes que aprobaron y reprobaron, de igual forma, el total de los que quedaron en situación condicional. Porcentaje de estudiantes que quedaron excluidos por bajo rendimiento y también el porcentaje que aprobaron el periodo, con relación al total de ellos.*
- *Por cada grupo se requiere la cantidad de estudiantes que aprobaron y reprobaron el periodo. Los grupos son identificados como Grupo A, Grupo B, Grupo C y así sucesivamente.*
- *Finalmente, por cada estudiante se debe informar su nota definitiva acompañada de un mensaje que especifique su situación académica.*

Dentro del reglamento estudiantil se tienen contemplados los siguientes aspectos:

1. *Todo estudiante de primer semestre debe cursar 6 espacios académicos o materias.*
2. *La nota mínima es de 0.0 y la nota máxima es de 5.0.*
3. *Se considera que un espacio académico se aprueba si su nota definitiva es mayor o igual a 3.0.*
4. *Un periodo académico se considera aprobado, si al promediar las notas definitivas de las 6 materias, se obtiene un resultado igual o superior a 3.0.*

   Nota: *si un estudiante es excluido por bajo rendimiento, se le considerará perdido el periodo, sin importar el promedio obtenido.*
5. *Al finalizar el semestre, el estudiante será clasificado en una de las siguientes situaciones académicas:*

   a) *Excluido por bajo rendimiento.*

   b) *Condicional.*

   c) *Continúa de manera normal.*

6. *Un estudiante quedará excluido por bajo rendimiento, si cumple una o ambas de las siguientes condiciones:*

   a) *Si el promedio del periodo académico, es inferior a 2.0.*

   b) *Si pierde más del 50 % de las materias cursadas.*

7. *Se considera que un estudiante queda en situación condicional, si no fue expulsado por bajo rendimiento y si obtuvo un promedio del periodo académico entre 2.0 y 2.9.*

8. *Si el estudiante no presenta ninguna de las dos anteriores situaciones, se considera que continúa de manera normal.*

Análisis del problema:

- **Resultados esperados:** al analizar el enunciado, se evidencia que se requiere información parcial y general. La información parcial, se puede dividir entre la información que se espera por cada estudiante y la información de los grupos. La información general, se refiere a lo concerniente a todos los estudiantes de primer semestre.

  - Por cada estudiante se requiere:
    - Nota definitiva acompañada de un mensaje que especifique su situación académica.
  - Por cada grupo de primer semestre, la Universidad necesita conocer:
    - Cantidad de estudiantes que aprobaron el periodo.
    - Cantidad de estudiantes que reprobaron el periodo.
  - A nivel general, es decir, con relación a todos los estudiantes procesados, la información que se solicita es:
    - Mayor promedio.
    - Menor promedio.
    - Cantidad de estudiantes que aprobaron el periodo.
    - Cantidad de estudiantes que reprobaron el periodo.
    - Porcentaje de estudiantes que aprobaron el periodo, con relación al total de ellos.
    - Porcentaje de estudiantes excluidos con relación al total.
    - Cantidad de estudiantes que quedaron en situación condicional.

- **Datos disponibles:** de cada estudiante se conoce el código, nombre y la nota definitiva (entre 0.0 y 5.0) de cada una de las 6 materias.

  De los grupos se sabe que están identificados con letras: Grupo A, Grupo B..., de igual forma el enunciado deja claro que los estudiantes están distribuidos en cada uno de los grupos.

  Adicionalmente, a la información que se debe ingresar al algoritmo, el reglamento estudiantil da precisión de varios aspectos que deben ser tenidos en cuenta.

- **Proceso:** con base a los resultados esperados, se evidencia que la información se debe presentar agrupada de la siguiente manera:

  1. Por estudiante.
  2. Por grupo.
  3. General, es decir, del total de los estudiantes.

  De cada estudiante se solicita informar el promedio del periodo y su situación académica. Para calcular el promedio, se puede hacer de una de las siguientes dos formas:

  1. Se pueden definir 6 variables para almacenar la nota definitiva de cada materia, se calcula su sumatoria y luego su promedio.
  2. Se define una sola variable para leer la nota definitiva de cada materia y mediante un ciclo que itere 6 veces se hace la lectura y esta se va acumulando; una vez finalizado el ciclo se calcula el promedio.

  Para la solución a implementar, se optará por la segunda opción. El ciclo al que se hace mención en esta solución, deberá estar dentro de otro ciclo intermedio, que manejará la información correspondiente a los estudiantes y este a su vez estará anidado dentro de un ciclo externo que trabajará con la información de los grupos.

  En la Figura 4.42 se puede apreciar de forma global la estructura de la solución, la cual se interpreta de la siguiente manera:

  La zona denominada Inicialización general, será utilizada para inicializar las variables que vayan a intervenir en la condición del ciclo externo, en el caso de ser necesarias para su ejecución. Así mismo, se inicializarán los contadores de los datos generales que solicitan, por ejemplo, el total de estudiantes, estudiantes en situación condicional, los que aprobaron o reprobaron el promedio y los excluidos.

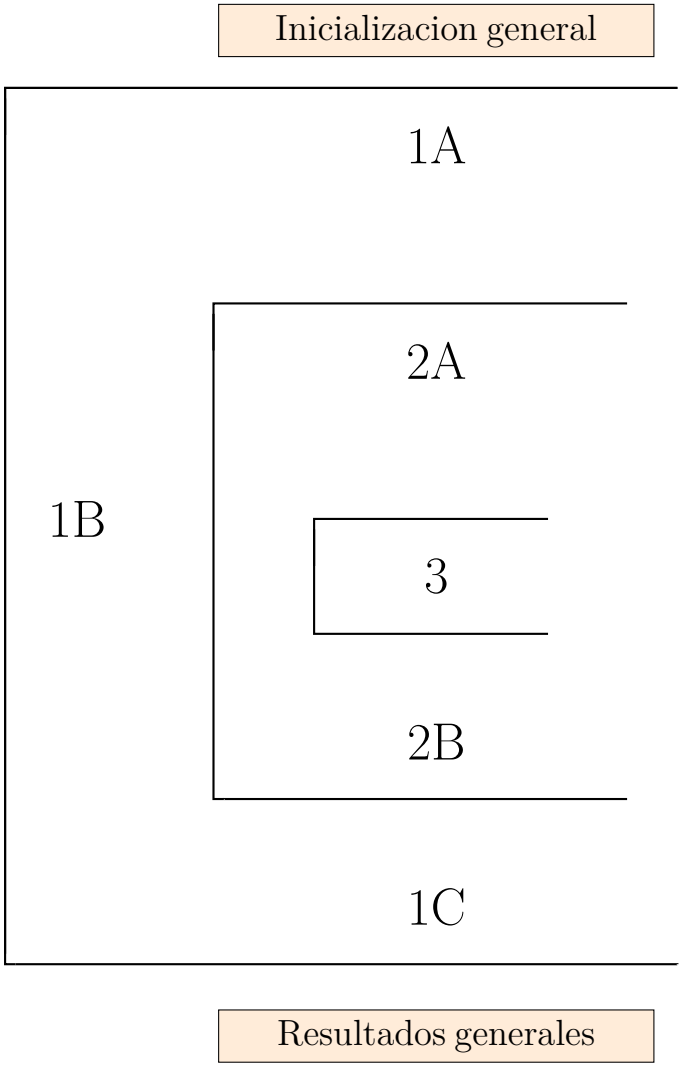


Figura 4.42: Forma general para el Ejemplo 4.25

Dentro del ciclo externo se identifican las siguientes zonas: `1A` y `1B` y `1C`.

En la zona `1A`, se deben inicializar los contadores que se incrementarán dentro del ciclo intermedio, ubicado en la Zona 1B. En estos contadores se almacenará la cantidad de estudiantes que aprobaron o reprobaron en cada uno de los grupos. Adicionalmente se especificará el grupo al cuál se le van a procesar los datos de sus respectivos estudiantes.

La zona `1B`, conformada por las zonas `2A`, `2B` y `3`, estará ocupada por el ciclo intermedio que procesará los datos de los estudiantes; dentro de él, estará el ciclo más interno (zona 3) que se encargará de la lectura de las 6 notas definitivas.

Para terminar con el ciclo externo, se identifica la zona `1C`, en ella, se da la información que corresponde a los grupos, se incrementan algunos contadores generales y se incrementa la identificación del grupo (Grupo A, Grupo B...).

Pasando al ciclo intermedio se encuentran las zonas `2A`, `2B` y `3`. En este ciclo se van a procesar los datos pertenecientes a cada estudiante. En la zona 2A, se solicitarán los datos que se tienen del estudiante: el código, el nombre y las notas. Recuerde que las notas serán leídas dentro de un ciclo, que estará anidado dentro del ciclo intermedio. En la zona 2A también se deben inicializar el contador de materias reprobadas y el acumulador de la sumatoria de las notas definitivas, que servirá para calcular el promedio del periodo.

Entre la zona `2A` y `2B` estará el ciclo encargado de procesar las notas definitivas de cada materia (zona `3`).

La zona `2B` se destina a los siguientes objetivos:

- Calcular del promedio del periodo.
- Determinar el mayor y menor promedio.
- Informar la situación académica de cada estudiante.
- Contar el número de estudiantes excluidos por bajo rendimiento, los que quedan en situación condicional y los que aprobaron el periodo.

El tercer ciclo que compone esta solución algorítmica, se encuentra en la zona `3`. Este ciclo debe iterar 6 veces, en cada ejecución debe cumplir con 3 funciones: leer la nota definitiva de cada 1 de las 6 materias que cursó el estudiante, incrementar la sumatoria de notas

definitivas y determinar si el estudiante perdió alguna materia, es caso de ser así, debe incrementar un contador que servirá luego, en la zona 2B, para decidir si el estudiante es excluido por perder más del 50 % de las materias.

Por último, se encuentra la zona denominada Resultados generales, destinada a los siguientes cálculos e informes globales:

- Cálculo de los porcentajes generales.
- Se informarán los datos del mayor y menor promedio, cantidad de estudiantes que aprobaron y reprobaron el promedio; porcentaje de estudiantes que aprobaron y el de los que quedaron excluidos por bajo rendimiento y finalmente informará la cantidad de estudiantes que quedaron en situación condicional.

De acuerdo a las características de este problema, se puede determinar que el ciclo externo y el intermedio pueden ser estructuras `Mientras-FinMientras` o `Haga-MientrasQue`, tenga presente que ambas operan de manera similar, la diferencia radica en que la primera se condiciona al inicio y la segunda al final. Para la solución que se va a plantear, se elegirá el `Mientras-FinMientras` para el ciclo externo y el `Haga-MientrasQue` para el intermedio; ambos ciclos se controlarán con una pregunta de continuar o no con la ejecución. El ciclo interno, el que está en la zona 3, se diseñará con una estructura `Para-FinPara` ya que es la más indicada puesto que se conoce que debe iterar un número definido de veces (6).

- **Variables requeridas:**
  - Para los estudiantes:
    - `codigo`: identificación de cada estudiante.
    - `nombre`: nombre del estudiante.
    - `definitiva`: nota definitiva obtenida en cada materia.
    - `sumaDefinitivas`: sumatoria de las 6 notas definitivas.
    - `promedioEstudiante`: promedio de las 6 notas definitivas.
    - `reprobadas`: cantidad de materias que reprobó el estudiante. Se usará para determinar si el estudiante es expulsado por reprobar más del 50 % de las materias.

- Para cada grupo:
  - `grupo`: identificará a cada uno de los grupos. Inicia en la letra A y se va incrementando por cada uno (Grupo A, Grupo B, ...).
  - `aprobaronGrupo`: cantidad de estudiantes que aprobaron el periodo por grupo.
  - `reprobaronGrupo`: cantidad de estudiantes que reprobaron el periodo por grupo.
- A nivel general:
  - `condicionalGeneral`: cantidad de estudiantes que en situación condicional.
  - `estudiantesGeneral`: contará el número de estudiantes que se procesen.
  - `aprobaronGeneral`: cantidad total de estudiantes que aprobaron el periodo académico.
  - `reprobaronGeneral`: cantidad total de estudiantes que reprobaron el periodo académico.
  - `excluidosGeneral`: número de estudiantes excluidos por bajo rendimiento.
  - `mayorPromedioGral`: mejor promedio entre todos los estudiantes.
  - `menorPromedioGral`: menor promedio entre todos los estudiantes.
  - `porcentajeAprobaronGral`: porcentaje de estudiantes que aprobaron, con relación a la población total.
  - `porcentajeExcluidosGral`: porcentaje de estudiantes que fueron excluidos por bajo rendimiento, con relación a la población total.
- Para controlar los ciclos:
  - `seguir`: variable centinela para controlar los ciclos externo e intermedio.
  - `materia`: controla el ciclo que leerá las 6 notas definitivas de cada estudiante.

De acuerdo al análisis planteado, se propone el Algoritmo 4.39.

Algoritmo 4.39: Universidad

```
1  Algoritmo Universidad
2    Entero   condicionalGeneral, estudiantesGeneral,
3             aprobaronGeneral, reprobaronGeneral,
4             excluidosGeneral, aprobaronGrupo,
5             reprobaronGrupo, reprobadas,
6             materia
7    Real     mayorPromedioGral, menorPromedioGral,
8             definitiva, sumaDefinitivas,
9             promedioEstudiante, porcentajeAprobaronGral,
10            porcentajeExcluidosGral
11   Caracter seguir, grupo
12   Cadena   codigo, nombre
13
14   // Zona Inicialización general
15   seguir = 'S'
16   grupo  = 'A'
17   condicionalGeneral = 0
18   estudiantesGeneral = 0
19   aprobaronGeneral   = 0
20   reprobaronGeneral  = 0
21   excluidosGeneral   = 0
22
23   // Ciclo externo que controla los grupos
24   Mientras( seguir == 'S' O seguir == 's' )
25     // Zona 1A
26     imprimir( "Grupo: ", grupo )
27
28     aprobaronGrupo  = 0
29     reprobaronGrupo = 0
30
31     // Ciclo intermedio que procesa los datos del estudiante
32     // Inicio zona 1B
33     Haga
34       // Zona 2A
35       imprimir( "Digite los datos del estudiante" )
36
37       imprimir( "Código: " )
38       Haga
39         leer( codigo )
40       MientrasQue( longitud( codigo ) == 0 )
41
42       imprimir( "Nombre: " )
43       Haga
44         leer( nombre )
45       MientrasQue( longitud( nombre ) == 0 )
46
47       sumaDefinitivas = 0
48       reprobadas      = 0
```

```
49
50        // Ciclo interno, procesa las notas de cada estudiante
51        Para materia = 1 Hasta 6 Incremento 1
52          // Zona 3
53          imprimir( "Nota definitiva de Materia:  ", materia )
54          Haga
55            leer( definitiva )
56          MientrasQue( definitiva < 0.0 O definitiva > 5.0 )
57
58          sumaDefinitivas = sumaDefinitivas + definitiva
59
60          // Para saber si lo excluyen
61          Si( definitiva < 3.0 ) Entonces
62            reprobadas = reprobadas + 1
63          FinSi
64        FinPara
65
66        // Zona 2B
67        promedioEstudiante = sumaDefinitivas / 6
68        imprimir( "Su promedio es: ", promedioEstudiante )
69
70        Si( estudiantesGeneral == 0 ) Entonces
71          mayorPromedioGral = promedioEstudiante
72          menorPromedioGral = promedioEstudiante
73        SiNo
74          Si(promedioEstudiante >  mayorPromedioGral) Entonces
75             mayorPromedioGral = promedioEstudiante
76          FinSi
77
78          Si(promedioEstudiante < menorPromedioGral) Entonces
79             menorPromedioGral = promedioEstudiante
80          FinSi
81        FinSi
82
83        // Determina situación académica del estudiante
84        Si(promedioEstudiante < 2.0 O reprobadas > 3) Entonces
85          imprimir( nombre, " excluido por bajo rendimiento" )
86          excluidosGeneral = excluidosGeneral + 1
87
88          // como no hay parcial, se incrementa el general.
89          reprobaronGrupo = reprobaronGrupo + 1
90        SiNo
91          Si( promedioEstudiante < 3.0 ) Entonces
92            imprimir( nombre," en situación condicional" )
93            condicionalGeneral = condicionalGeneral + 1
94
95            // como no hay parcial, se incrementa el general.
96            reprobaronGrupo = reprobaronGrupo + 1;
97          SiNo
```

```
98            imprimir( nombre, " continúa normalmente" )
99            aprobaronGrupo = aprobaronGrupo + 1
100         FinSi
101       FinSi
102
103       estudiantesGeneral = estudiantesGeneral + 1
104
105       imprimir( "Hay más estudiantes [S] o [N]?: " )
106       Haga
107         leer( seguir )
108       MientrasQue( seguir != 'S' Y seguir != 'N' Y
109                    seguir != 's' Y seguir != 'n' )
110
111     MientrasQue( seguir == 'S' O seguir == 's' )
112
113     // Final zona 1B
114     // Zona 1C
115     // Calculos e información por grupo...
116     imprimir( "Cantidad que aprobaron : ", aprobaronGrupo  )
117     imprimir( "Cantidad que reprobaron: ", reprobaronGrupo )
118
119     aprobaronGeneral  = aprobaronGeneral  + aprobaronGrupo
120     reprobaronGeneral = reprobaronGeneral + reprobaronGrupo
121
122     grupo = grupo + 1
123
124     imprimir( "¿Hay más grupos [S] o [N]?: " )
125
126     Haga
127       leer( seguir )
128     MientrasQue ( seguir != 'S' Y seguir != 'N' Y
129                   seguir != 's' Y seguir != 'n' )
130
131   FinMientras
132
133   // Zona Resultados generales
134   porcentajeAprobaronGral = aprobaronGeneral * 100 /
          estudiantesGeneral
135   porcentajeExcluidosGral = excluidosGeneral * 100 /
          estudiantesGeneral
136
137   imprimir( "RESUSLTADOS GENERALES" )
138   imprimir( "Mayor promedio: ", mayorPromedioGral )
139   imprimir( "Menor promedio: ", menorPromedioGral )
140
141   imprimir( "Cantidad que aprobaron: ", aprobaronGeneral )
142   imprimir( "Cantidad que reprobaron: ", reprobaronGeneral )
143
144   imprimir( "% aprobación:", porcentajeAprobaronGral, "%" );
```

```
145
146   imprimir( "De los ", estudiantesGeneral, " estudiantes" )
147   imprimir( " fueron excluidos ", excluidosGeneral,
148             " equivale al ", porcentajeExcluidosGral, "%" )
149   imprimir( " y en situación condicional: ",
150             condicionalGeneral, " estudiantes" )
151 FinAlgoritmo
```

Explicación del algoritmo:

Aunque el funcionamiento de este algoritmo quedó detallado en el análisis, se complementarán algunos aspectos relevantes.

La característica principal de este algoritmo, es que se trabajaron varias estructuras de repetición de manera anidada. Dentro de estas estructuras, a su vez, se codificaron estructuras de decisión. Algunas simples como las presentadas en las líneas 61, 74 y 78; otras anidadas como las de las líneas 70 y 84. Una de estas decisiones contempla dentro de su parte falsa dos nuevas decisiones simples, las cuales se ejecutan de manera independiente en el caso de que la condición de la línea 70 sea falsa.

Hay otros aspectos interesantes para resaltar en esta solución:

Se usa una sola variable centinela (`seguir`) para controlar la ejecución de dos ciclos. En la línea 15, la instrucción `seguir = 'S'` inicializa la variable de control del ciclo externo `Mientras-FinMientras` (línea 24), para que, al evaluar la condición al inicio, se dé un resultado `Verdadero` y se ejecute el cuerpo. Desde la línea 124 a la línea 128, se da la lectura de esta variable para determinar si se continúa o no con la repetición del proceso. De forma similar se hace desde las líneas 105 a la 109, pero en este caso la lectura se hace para verificar la repetición o no del ciclo intermedio `Haga-MientrasQue`; teniendo en cuenta que este es un ciclo condicionado al final, no fue necesaria la inicialización de esta variable para que se realizara su primera iteración.

En la línea 16, se inicializa la variable grupo con el valor de `'A'`, ya que los grupos están identificados como Grupo A, Grupo B, Grupo C y así sucesivamente. Esta variable en la línea 122, se incrementa en 1; al ser de tipo carácter toma el siguiente valor, para el caso de la primera iteración toma el valor de `'B'`, en la siguiente iteración toma el valor de `'C'` y seguirá avanzando con las letras del alfabeto mientras se estén ingresando nuevos grupos. Todas las lecturas de entrada de datos, están condicionadas con ciclos `Haga-MientrasQue` y funcionan tal cual, se ha explicado en algoritmos anteriores.

Vale la pena resaltar que, aunque fueron utilizadas 3 estructuras repetitivas diferentes en esta solución, es igualmente válido, usar cualquier tipo de combinación de ellas. El asunto radica, en que dependiendo del tipo de ciclo que se use, se deben adicionar o eliminar algunas instrucciones.

4.4.1 Prueba de escritorio

A continuación se estudiarán las pruebas de escritorio utilizando el ciclo `Para-FinPara`.

.:Ejemplo 4.26. *Realice la prueba de escritorio o tabla de verificación para el Algoritmo 4.40.*

Algoritmo 4.40: Tabla1

```
1 Algoritmo Tabla1
2   Entero indice
3
4   Para indice = 0 Hasta 30 Incremento 5)
5     Si( indice % 10 == 0 ) Entonces
6       imprimir( indice )
7     FinSi
8   FinPara
9 FinAlgoritmo
```

La Tabla 4.13 presenta la tabla de verificación para el Algoritmo 4.40.

indice	**indice<= 30**	**indice % 10 == 0**	imprimir **indice**
0	0 <= 30 (V)	0 == 0 (V)	0
5	5 <= 30 (V)	5 == 0 (F)	
10	10 <= 30 (V)	0 == 0 (V)	10
15	15 <= 30 (V)	5 == 0 (F)	
20	20 <= 30 (V)	0 == 0 (V)	20
25	25 <= 30 (V)	5 == 0 (F)	
30	30 <= 30 (V)	0 == 0 (V)	30
35	35 <= 30 (F)		

Tabla 4.13: Prueba de escritorio - Algoritmo 4.40

Explicación de la prueba de escritorio:

Se presenta un ciclo `Para`, donde la variable `indice` se inicializa en 0; mientras la condición del ciclo, que se interpreta como `indice <= 30`, sea verdadera, el ciclo itera.

En cada iteración, mediante la condición de la estructura `Si (indice % 10 == 0)`, se pregunta que si al dividir el valor de `indice` entre `10`, deja como resto el valor de `0`. En pocas palabras, está preguntando que si el valor de `indice` es múltiplo de `10`. Si la condición es verdadera, entonces se procede a realizar la impresión del valor de `indice`.

.:Ejemplo 4.27. *En el Algoritmo 4.41, se emplearán de forma anidada, los tres ciclos que se estudiaron en este capítulo. Se tendrá la versión en pseudocódigo y luego en diagrama de flujo.*

Se quiere saber cuál es el resultado que imprime, para ello se realizará una prueba de escritorio o tabla de verificación.

Aclaración:



Con las tres estructuras repetitivas estudiadas en este capítulo, se pueden hacer casi las mismas cosas. Cada uno de estos ciclos es más funcional en determinadas ocasiones: El `Mientras-FinMientras` es útil cuando se requiera que un proceso se ejecute o no, dependiendo de una condición. El `Haga-MientrasQue`, se puede usar en aquellos procesos que deben ejecutarse por lo menos una vez. El `Para-FinPara`, es funcional cuando se tiene claro el número de iteraciones que debe hacer el ciclo.

A continuación se muestra el algoritmo en su representación en pseudocódigo y posteriomente su diagrama de flujo (Figura 4.43).

Algoritmo 4.41: Tabla2

```
1  Algoritmo Tabla2
2    Entero a, b, c
3    a = 1
4    b = 3
5    Mientras( a < 5 )
6      Haga
7        Para c = 1 Hasta b Incremento 1
8          imprimir( a, b, c )
9        FinPara
10       b = b - 2
11     MientrasQue( b >= 1 )
12     b = 2
13     a = a + b
14   FinMientras
15 FinAlgoritmo
```

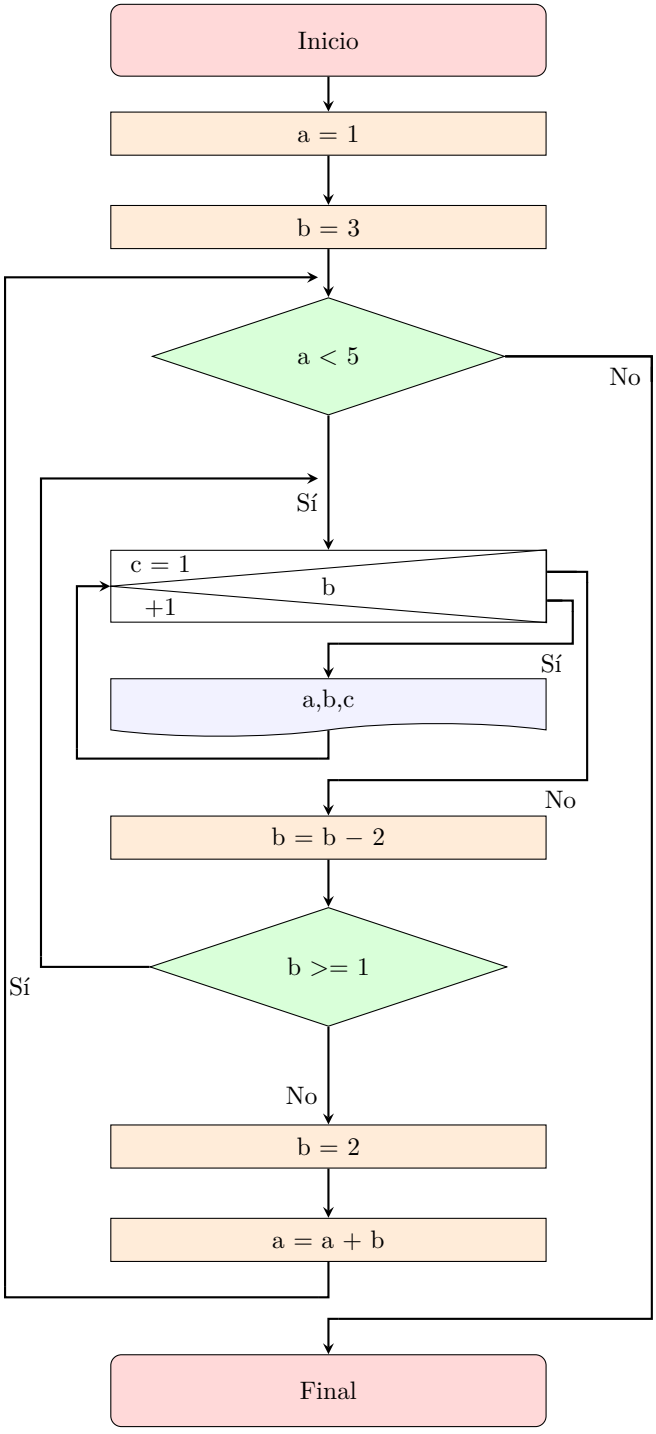


Figura 4.43: Diagrama de flujo del Algoritmo `Tabla2`

La prueba de escritorio para el anterior algoritmo es la siguiente:

a	**b**	**c**	**c<= b**	**a< 5**	**b>= 1**	imprimir **a, b, c**
1	3		1 < 5 (V)			
		1		1 <= 3 (V)		1, 3, 1
		2		2 <= 3 (V)		1, 3, 2
		3		3 <= 3 (V)		1, 3, 3
		4		4 <= 3 (F)		
	1				1 >= 1 (V)	
		1		1 <= 1 (V)		1, 1, 1
		2		2 <= 1 (F)		
	-1				−1 >= 1 (F)	
	2					
3			3 < 5 (V)			
		1		1 <= 2 (V)		3, 2, 1
		2		2 <= 2 (V)		3, 2, 2
		3		3 <= 2 (F)		
	0				0 >= 1 (F)	
	2					
5			5 < 5 (F)			

Tabla 4.14: Prueba de escritorio - Algoritmo 4.41

Para cada una de las variables se reservó una columna en la Tabla 4.14. De igual forma se procedió con las condiciones de cada uno de los ciclos.

Al evaluar por primera vez la condición del `Mientras-FinMientras`, el resultado que se obtiene es verdadero, por lo tanto, se procede a ejecutar su cuerpo que está conformado por un ciclo `Haga-MientrasQue`; este al ser un ciclo condicionado al comienzo no requiere evaluación inicial para poder iterar.

Dentro del cuerpo del `Haga-MientrasQue`, está una estructura `Para`, que inicializa su variable de control `c` en 1; al encontrar su condición verdadera procede a ejecutarse `(1 <= 3)`.

Cuando el control es asumido por el ciclo `Para`, la ejecución del algoritmo itera en este sitio, hasta que la variable `c` tome un valor de 4. Cuando esto suceda, el control es entregado al ciclo `Haga-MientrasQue` donde se le restan dos unidades a la variable `b`. Al encontrar el `MientrasQue`, se evalúa la condición, cumpliéndose que `1 >= 1`. El `Haga-MientrasQue` itera una vez más; por segunda ocasión la variable `c` del ciclo `Para`, toma el valor de 1. Este inicia de nuevo su iteración, cuando la variable `c` tome el valor de 2, la condición será falsa (`c <= b`, o sea, `2 <= 1 (F)`). De nuevo el control vuelve a ser asumido por el ciclo `Haga-MientrasQue`, donde la variable `b` se decrementa en 2

unidades y toma el valor de -1; la condición del MientrasQue arroja un resultado falso (-1 >= 1 (F)), el control vuelve a ser asumido por el ciclo Mientras-FinMientras. Ahora la variable b toma el valor de 2 y la variable a el valor de 3. Al encontrar el FinMientras se regresa al Mientras y se testea la condición (a <5, es decir, 3 <5 (V)), logrando un resultado verdadero. Nuevamente se entra al Haga-MientrasQue y por consiguiente al Para, el índice c inicia en 1 y este ciclo se ejecuta dos veces. Una vez más el Haga-MientrasQue tiene el control, al realizar la operación b = b - 2, la variable b toma el valor de 0 con lo cual la condición del MientrasQue se vuelve falsa. Por tercera vez el control lo asume el ciclo Mientras-FinMientras, la variable b toma el valor de 2 y la variable a incrementa en el valor de b, almacenando un 5; el FinMientras retorna al Mientras para evaluar la condición (a <5), obteniendo un resultado falso, con lo cual se termina la ejecución de los ciclos y del algoritmo.

Observe en la tabla, que las tres condiciones de los ciclos terminan con un valor de falso: 3 <= 2 (F), 0 >= 1 (F) y 5 < 5 (F). Ellos solamente se ejecutan mientras la condición sea verdadera.

En cada iteración del Para se imprimen los valores de a, b y c.

Cuando el ciclo intermedio Haga-MientrasQue no se ejecute, tampoco se ejecuta el Para.

Cuando el ciclo externo Mientras-FinMientras termine de iterar, los otros dos ciclos tampoco podrán volver a hacerlo.

4.5. Ejercicios propuestos

1. Responda las siguientes preguntas:

   a) ¿Cuáles son las estructuras repetitivas condicionadas al comienzo?

   b) ¿De las estructuras repetitivas estudiadas en este capítulo, cuál o cuáles de ellas pueden llegar a no ejecutarse y por qué?

   c) ¿De las estructuras repetitivas estudiadas en este capítulo, cuál o cuáles de ellas se ejecutan por lo menos una vez y por qué?

   d) ¿A qué se le llama iteración?

   e) Escriba dos situaciones en las que usaría una variable bandera.

   f) ¿Cuál es la diferencia entre un acumulador y un contador?

g) ¿Qué sucedería si Usted no escribe la instrucción modificadora de condición dentro de un ciclo `Mientras-FinMientras`?

h) ¿Por qué dentro del cuerpo del ciclo `Para`, no está explícita la instrucción modificadora de condición?

i) ¿Para validar la entrada de un dato (lectura) en un algoritmo, cuál es el ciclo ideal? Justifique su respuesta.

j) Dentro de la cultura popular siempre se ha dicho que, si alguien no puede dormir, debe contar ovejas hasta que logre conciliar el sueño. Suponga que le piden a Usted, que mediante un diagrama de flujo represente esta situación.

De las 3 estructuras repetitivas estudiadas en este capítulo, ¿cuál es la menos indicada para hacerlo? Justifique su respuesta.

2. Tome cada uno de los ejemplos de este capítulo y reescríbalos usando una estructura de ciclo diferente. En caso de no ser posible el cambio, justifique su respuesta.

3. Usando la instrucción `Para`, represente los siguientes enunciados:

   a) La variable x con un valor inicial de 4, un valor final de 40 e incrementos de a 1.

   b) La variable x que va desde 100 a 20, disminuyendo de 1 en 1.

   c) La variable x que inicia en 10, incrementando de 5 en 5, hasta llegar a 200.

4. Dado los extremos de un intervalo [M, N], halle la sumatoria de los números pares y de los impares que pertenezcan a él.

5. Dada una población, máximo de 500 habitantes, determinar cuántos son mayores y cuántos menores de edad. De cada uno de ellos se conoce la edad en años.

6. Genere e imprima los múltiplos de 3 que se encuentren entre 6 y n, donde n tiene que ser superior a 6.

7. En 1937, el matemático alemán Lothar Collatz, enunció la conjetura de Collatz, también conocida como el problema de Ulam, conjetura 3n + 1, entre otros.

   Collatz enunció que, a partir de cualquier número natural, siempre se obtiene la unidad. Para ello se hace el siguiente procedimiento:

Tome un número n y ejecute las siguientes operaciones:

Si n es par, halle la división entera entre 2. Si n es impar, multiplíquelo por 3 y súmele 1.

Con el resultado que obtenga, repita las operaciones anteriores, hasta obtener 1 como respuesta. Ejemplos:

n = 13, se obtienen los siguientes resultados:

40, 20, 10, 5, 16, 8, 4, 2, 1.

n = 6, se obtienen los siguientes resultados:

3, 10, 5, 16, 8, 4, 2, 1.

Otra de las curiosidades de esta conjetura, es que cuando se llegue a 1 y se apliquen nuevamente las fórmulas, obtendrá la secuencia 4, 2, 1 de forma infinita.

El algoritmo que Usted diseñe, debe solicitar un número y aplicar el anterior concepto, imprimiendo los resultados que se obtienen hasta llegar a la unidad.

8. Lea un número entero positivo, descompóngalo en cada una de sus cifras y con ellas genere el número invertido. Por ejemplo, si el número a leer es el 5432, el resultado será 2345.

9. Dado un número menor o igual a 50, calcule su factorial mediante sumas sucesivas.

10. El cajero de un restaurante desea controlar el flujo de caja en un día de trabajo cualquiera. Antes de abrir al público, el gerente del establecimiento le entrega la base para el día, la cual consiste en una suma de dinero que debe registrar en la caja y con la cual se espera pueda desempeñarse sin contratiempos. Durante su jornada tendrá ingresos por concepto de las ventas que se realicen, también habrán salidas de caja para la compra de insumos o gastos eventuales que deban realizarse.

    Se espera un algoritmo, que reciba el registro de cada una de las operaciones a medida que vayan sucediendo. El cajero también requiere un informe del saldo en caja después de cada registro. El algoritmo deberá dar un mensaje de alerta en el caso que el saldo sea inferior o igual al 15 % de la base asignada. Al cierre del restaurante, se requiere los saldos finales (saldo en caja, ingresos y egresos) y la cantidad de cada una de las operaciones realizadas.

11. En el Ejemplo 4.15 se hizo un algoritmo que informaba si un número es o no perfecto. Basado en esa solución, diseñe un nuevo algoritmo que lea un número n e imprima los números perfectos entre 1 y n.

12. Simule el funcionamiento de un temporizador, que reciba como entrada una cantidad de minutos y segundos. En el momento que falten 5 minutos para cumplir el tiempo, deberá dar un mensaje de alerta, cuando finalice mostrará el siguiente mensaje "Tiempo fuera". Debe funcionar máximo para 1 hora.

13. Usando estructuras repetitivas, elabore un algoritmo, que genere e imprima las letras del abecedario de la siguiente forma:

```
Z Y X W V U T S R Q P O N M L K J I H G F E D C B A
Y X W V U T S R Q P O N M L K J I H G F E D C B A
X W V U T S R Q P O N M L K J I H G F E D C B A
W V U T S R Q P O N M L K J I H G F E D C B A
V U T S R Q P O N M L K J I H G F E D C B A
U T S R Q P O N M L K J I H G F E D C B A
T S R Q P O N M L K J I H G F E D C B A
S R Q P O N M L K J I H G F E D C B A
R Q P O N M L K J I H G F E D C B A
Q P O N M L K J I H G F E D C B A
P O N M L K J I H G F E D C B A
O N M L K J I H G F E D C B A
N M L K J I H G F E D C B A
M L K J I H G F E D C B A
L K J I H G F E D C B A
K J I H G F E D C B A
J I H G F E D C B A
I H G F E D C B A
H G F E D C B A
G F E D C B A
F E D C B A
E D C B A
D C B A
C B A
B A
A
```

14. A un amigo que vive en el norte de la ciudad, su terapeuta le recomendó caminar mínimo 3 días a la semana desde su apartamento hasta el centro, lo cual él hace sin falta alguna; el recorrido tiene aproximadamente 22 cuadras y debe hacerlo con ropa cómoda. Este ejercicio lo debe realizar durante 4 meses.

    Al momento de volver a consulta, el amigo debe informarle a su terapeuta lo siguiente:

Promedio de tiempo por semana, por mes y por los 4 meses. Adicionalmente, cuál fue el menor y el mayor tiempo empleado en el recorrido.

Para esta tarea, el amigo lleva un registro del tiempo que invierte en cada caminata.

15. Imprima los 10 múltiplos sucesivos de 3 en orden descendente, a partir de un número n que será ingresado por el usuario y que representará el menor valor. Si el n, no es múltiplo de 3, debe llevarse al múltiplo más cercano superior.

    Por ejemplo, si el número ingresado es el 28, al no ser múltiplo de 3 debe llevarse al siguiente múltiplo superior, o sea, 30. Entonces la salida del algoritmo se visualizaría así:

    57, 54, 51, 48, 45, 42, 39, 36, 33, 30.

16. En una compañía que tiene varias sucursales a nivel nacional, una o varias en cada departamento y solo una por ciudad, se requiere de un censo que permita conocer la siguiente información de sus empleados:

    Porcentaje total de personas que tienen estudios de primaria, secundaria, profesional, maestría o doctorado. Se tiene en cuenta solamente, el nivel más alto de estudio.

    Porcentaje total de mujeres con posgrado, con relación a todas las mujeres de la compañía.

    Cantidad de hombres con estudios de solo primaria en cada departamento.

    Cantidad de hombres y mujeres en cada sucursal que hayan recibido su título profesional antes de cumplir 25 años de edad.

    De cada empleado se conoce el número de identificación y el nombre, adicional a los datos que se requieren en la solución del problema.

17. Para los siguientes algoritmos, realice la respectiva prueba de escritorio o tabla de verificación:

a) Algoritmo Prueba1

```
Algoritmo Prueba1
  Entero numero, divisor

  numero = 6 * 4
  Para divisor = 1 Hasta numero / 2 Incremento 1
    Si ( numero / divisor * divisor = numero )
       Entonces
         imprimir ( divisor )
    FinSi
  FinPara
FinAlgoritmo
```

b) Algoritmo Prueba2

```
Algoritmo Prueba2
  Entero t, i, r, a

  t = 0
  Haga
    i = 2
    Haga
      Para r = 4 Hasta 1 Decremento 2
        a = i + r / 2.0
        imprimir( a )
      FinPara
      i = i - 1
    MientrasQue (i >= 1)
    t = t + 1
  MientrasQue (t <= 1)
FinAlgoritmo
```

c) Algoritmo Prueba3

```
Algoritmo Prueba3
  Entero r, p, g

  r = 4
  Mientras (r >= 2)
    Para p = 1 Hasta 2 Incremento 1
      g = 2
      Haga
        imprimir( p, g )
        g = g + p
      MientrasQue (g <= 4)
    FinPara
    r = r - 2
  FinMientras
FinAlgoritmo
```

Capítulo 5

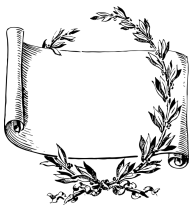
Procedimientos y funciones

Cada cosa tiene su belleza, pero no todos pueden verla.

Confucio

Objetivos del capítulo:

- Construir algoritmos para la solución de problemas mediante el uso de funciones y procedimientos.
- Identificar los parámetros de cada procedimiento / función adecuadamente.
- Diferenciar entre el paso de parámetros por valor y por referencia.



5.1. Procedimiento

Un procedimiento es un conjunto finito de instrucciones con **un único propósito** bien definido (responsabilidad) y que puede ser invocado (utilizado) por medio de un nombre que lo identifica de manera única. Lo ideal es que el nombre indique la acción o propósito del procedimiento por medio de un verbo y algún complemento que da sentido al verbo, por ejemplo, `imprimirMensaje`, `calcularDefinitiva`, `validarNota`; y dicho nombre es utilizado para su invocación en cualquier parte del algoritmo, cuantas veces sea necesario. Lo que esta invocación implica es que cuando el algoritmo llegue a la línea de invocación, se entiende que se ejecutan todas las instrucciones definidas dentro del procedimiento. La invocación de un procedimiento puede ser realizada incluso dentro de él mismo.

Una de las razones para usar procedimientos es poder descomponer un algoritmo en partes de menor tamaño que sean, en principio, más fáciles de comprender que el todo; por tanto, para resolver un problema se emplean uno o varios procedimientos que en conjunto realizan la tarea deseada. Otra razón importante para usar procedimientos es la reutilización de código que implica el poder utilizar las instrucciones de un procedimiento tantas veces como sea necesario dentro del algoritmo. En otras palabras, un procedimiento puede ser invocado tantas veces como sea necesario.

La forma general de esta estructura es presentada en el segmento del Algoritmo 5.1.

Algoritmo 5.1: Forma general del `Procedimiento`

```
1 Procedimiento verboComplemento ( [lista de parámetros] )
2   Instrucción1
3   Instrucción2
4   ...
5   Instrucciónk
6 FinProcedimiento
```

La mayoría de procedimientos, en un algoritmo, requiere de información para poder realizar su propósito, en la estructura general (Algoritmo 5.1) se puede observar `[lista de parámetros]` que representa el lugar en donde se debe enumerar los tipos y las variables que reciben la información enviada al procedimiento. Estas variables se conocen como parámetros y solo existen dentro del procedimiento, se dice entonces que el alcance o ámbito de los parámetros es local al procedimiento. Se emplean los corchetes (`[ ]`) para indicar que la lista puede estar vacía.

Los procedimientos pueden tener cualquier cantidad de parámetros dependiendo de la funcionalidad a realizar. El orden en que son definidos los parámetros solo influye en el orden en que los argumentos (valores enviados al procedimiento) se escriben, sin alterar la funcionalidad.

Buenas prácticas:



- Tener una cantidad razonable de parámetros según la funcionalidad
- La longitud de un procedimiento en principio no debería ser mayor a una página para facilitar su seguimiento, de ser mayor, seguramente podrá ser subdividido en más procedimientos.

Aclaración:



En este libro todos los parámetros de los procedimientos, almacenan siempre una copia del valor de los argumentos, esto implica que si dentro de un procedimiento se modifica el valor de uno o varios de los parámetros, este cambio no se ve reflejado en la variable usada como argumento. El único procedimiento en este libro que tiene un comportamiento diferente es el procedimiento `leer` el cual asigna un valor a la variable que se usa como argumento. En este último caso, se dice que hay **paso por referencia**, y en el primer caso, se dice que hay **paso por valor**.

Al igual que los parámetros, toda las variables declaradas dentro de un procedimiento, tienen alcance local, es decir, una vez termina el procedimiento, la variable deja de existir y ya no es posible recuperar su valor.

Como todos los parámetros son variables locales, sus nombres son totalmente independientes de los nombres de las variables utilizadas para enviar los valores a los argumentos al momento de invocar el procedimiento y no existirá conflicto entre ellas si los nombre coinciden.

.:Ejemplo 5.1. *Diseñe un algoritmos con procedimientos que permita mostrar el mensaje de saludo "Hola mundo".*

Análisis del problema:

- **Resultados esperados:** imprimir el mensaje "Hola mundo".
- **Datos disponibles:** Ninguno
- **Proceso:** imprimir el mensaje solicitado
- **Variables requeridas:** Ninguna

De acuerdo al análisis planteado, se propone el Algoritmo 5.2.

Algoritmo 5.2: Mensaje

```
1 Algoritmo Mensaje
2   imprimirSaludo( )
3 FinAlgoritmo
4
5 Procedimiento imprimirSaludo( )
6   imprimir( "Hola mundo" )
7 FinProcedimiento
```

Al ejecutar el algoritmo:

```
Hola mundo
```

Explicación del algoritmo:

La única línea del algoritmo es precisamente al invocación al método `imprimirSaludo( )` el cual indica que se deben ejecutar todas la líneas definidas con este nombre, para el caso, la línea 6.

```
6   imprimir( "Hola mundo" )
```

En la Figura 5.1 se muestra la solución del Ejemplo 5.1 mediante un diagrama de flujo.

Observe el símbolo empleado para indicar la invocación de un procedimiento (un rectángulo con dos pequeños rectángulos a los extremos). En el momento de la invocación siempre se deben usar los paréntesis para dejar explicito la información que se desea enviar al procedimiento (los argumentos), que para el caso está vacío ya que el procedimiento no los solicita; y que serán recibidos en las variables locales del procedimiento llamados parámetros.

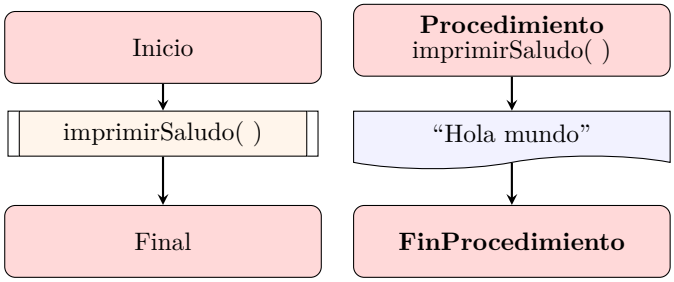


Figura 5.1: Diagrama de flujo del Algoritmo `Mensaje`

.:Ejemplo 5.2. *Diseñe un algoritmos con procedimientos que permita ingresar el nombre de dos personas y mostrar el mensaje de saludo "Hola" seguido del nombre de cada una de ellas de forma independiente.*

Análisis del problema:

- **Resultados esperados:** dos mensajes de saludo, uno por cada nombre ingresado.
- **Datos disponibles:** los dos nombres de las personas.
- **Proceso:** solicitar al usuario los dos nombre de las personas, luego invocar dos veces un procedimiento que imprima el saludo del nombre que se enviará como argumento.
- **Variables requeridas:**
  - `nombre1`: nombre de la primera persona
  - `nombre2`: nombre de la segunda persona

  . Internos al procedimiento se requiere el parámetro:
  - `n`: nombre de la persona a saludar.

De acuerdo al análisis planteado, se propone el Algoritmo 5.3.

Algoritmo 5.3: Mensaje2

```
1 Algoritmo Mensaje2
2   // Declaración de las variables
3   Cadena nombre1, nombre2
4
5   // Datos disponibles
6   imprimir( "Ingrese el primer nombre: " )
7   leer( nombre1 )
```

```
8
9   imprimir( "Ingrese el segundo nombre: " )
10  leer( nombre2 )
11
12  // Resultados esperados
13  imprimirSaludo( nombre1 )
14  imprimirSaludo( nombre2 )
15 FinAlgoritmo
16
17 Procedimiento imprimirSaludo( Cadena n )
18  imprimir( "Hola ", n )
19 FinProcedimiento
```

Al ejecutar el algoritmo:

```
Ingrese el primer nombre: Diana
Ingrese el segundo nombre: Ana
Hola Diana
Hola Ana
```

Explicación del algoritmo:

Lo primero es declarar y solicitar los datos disponibles (líneas de la 3 a la 10), posteriormente se invoca dos veces el método `imprimirSaludo`.

```
13  imprimirSaludo( nombre1 )
14  imprimirSaludo( nombre2 )
```

Es importante comprender el significado de estas invocaciones. Lo primero, es que mediante el nombre `imprimirSaludo` se identifica el procedimiento a invocar, luego en medio de los parámetros se envía al procedimiento la información que él requiera, para el caso, el valor de la variable `nombre1` y `nombre2` en sus respectivas líneas. El valor de la variable se almacena en el parámetro `n` del procedimiento, el cual es impreso posteriormente.

Observe que el tipo del parámetro `n` (`Cadena`) coincide plenamente con el tipo del argumento, en este caso el valor de la variable `nombre1` y `nombre2`, las cuales son de tipo (`Cadena`).

En la Figura 5.2 se muestra la solución del Ejemplo 5.2 mediante un diagrama de flujo.

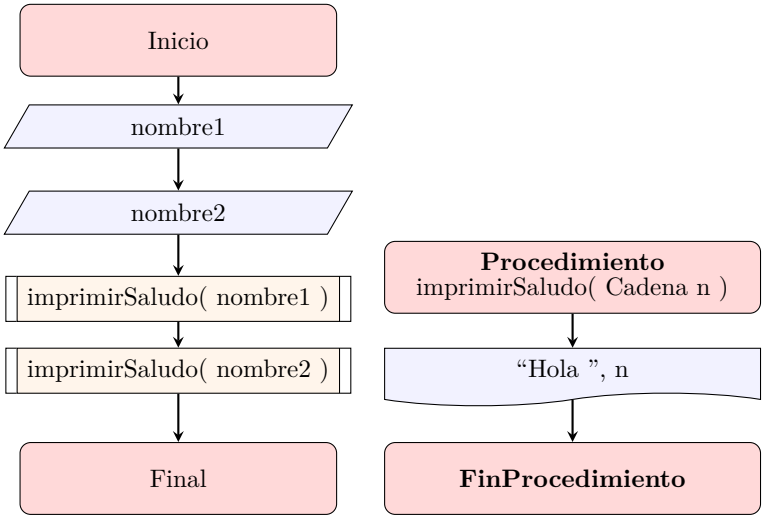


Figura 5.2: Diagrama de flujo del Algoritmo `Mensaje2`

Aclaración:



Sin duda alguna, los diagramas de flujo son una herramienta importante para la representación gráfica de un algoritmo, pero como se ha podido evidenciar, usar ambas representaciones es redundante, por tal motivo, solo se usará una u otra representación según sea la necesidad en el momento.

.:Ejemplo 5.3. *Diseñe un algoritmos con procedimientos que permita imprimir dos secuencias de números: “*3 6 9 12 ... $3 * n$*” y “*5 10 15 20 ... $5 * m$*”, en donde n y m representan la cantidad de términos de la serie.*

Análisis del problema:

- **Resultados esperados:** la impresión de las dos secuencias de números.
- **Datos disponibles:** el valor de n y el de m para poder generar las dos secuencias de números.
- **Proceso:** primero se le solicita al usuario los valores de n y m, luego se invoca dos veces un procedimiento que imprima una secuencia dado un valor base (3 ó 5) y el valor final (n ó m).

En términos generales el procedimiento requiere dos parámetros, un valor base (*base*) y la cantidad (*cantidad*) de términos. Con esta información se imprime la serie:

$$1 * base \;\; 2 * base \;\; 3 * base \;\; \ldots \;\; cantidad * base$$

Ahora si:

$base = 3$ y $cantidad = \mathbf{4}$	$base = 5$ y $cantidad = \mathbf{6}$
$1*3 \;\; 2*3 \;\; 3*3 \;\; \mathbf{4}*3$	$1*5 \;\; 2*5 \;\; 3*5 \;\; 4*5 \;\; 5*5 \;\; \mathbf{6}*5$
$3 \;\; 6 \;\; 9 \;\; 12$	$5 \;\; 10 \;\; 15 \;\; 20 \;\; 25 \;\; 30$

- **Variables requeridas:**

  - `n`: cantidad de términos de la primera serie
  - `m`: cantidad de términos de la segunda serie

  Internos al procedimiento se necesitan:

  - `base`: valor de la base para generar la serie (parámetro 1).
  - `cantidad`: número de términos de la serie (parámetro 2).
  - `i`: variable que controla el ciclo que va desde 1 hasta `cantidad` se requiere.
  - `termino`: variable que almacena el término actual de la serie `termino = i * base`.

De acuerdo al análisis planteado, se propone el Algoritmo 5.4.

Algoritmo 5.4: Mensaje2

```
1  Algoritmo Mensaje2
2    // Declaración de las variables
3    Entero n, m
4
5    // Datos disponibles
6    imprimir( "Cantidad de términos de la primera serie: " )
7    leer( n )
8
9    imprimir( "Cantidad de términos de la segunda serie: " )
10   leer( m )
11
12   // Resultados esperados
13   imprimirSerie( 3, n )
14   imprimirSerie( 5, m )
15 FinAlgoritmo
16
```

```
17 Procedimiento imprimirSerie( Entero base, Entero cantidad )
18   Entero termino
19
20   Para i = 1 Hasta cantidad Incremento 1
21     termino = i * base
22     imprimir( termino, " " )
23   FinPara
24 FinProcedimiento
```

Al ejecutar el algoritmo:

```
Ingrese la cantidad de términos de la primera serie: 4
Ingrese la cantidad de términos de la segunda serie: 6
3 6 9 12
5 10 15 20 25 30
```

Explicación del algoritmo:

En las primeras líneas del algoritmo (líneas de la 3 a la 10) se declaran las variables del algoritmo y se solicitan datos disponibles.

Luego se invoca dos veces el método `imprimirSerie` para generar las respectivas series. La diferencia entre ellas es el valor de la base (3 y 5) y la cantidad de términos (`n` y `m`).

```
13   imprimirSerie( 3, n )
14   imprimirSerie( 5, m )
```

Lo interesante es que no es necesario dos procedimientos para imprimir las dos series. El procedimiento generaliza la impresión de cualquier serie en donde se conoce un valor de la `base` y una `cantidad` de términos.

```
17 Procedimiento imprimirSerie( Entero base, Entero cantidad )
```

Para la primera invocación:

- el parámetro `base` toma el valor de 3
- el parámetro `cantidad` toma el valor de la variable `n`.

Para la segunda invocación:

- el parámetro `base` toma el valor de 5
- el parámetro `cantidad` toma el valor de la variable `m`.

Una vez los parámetros tiene sus valores, se ejecutan las instrucciones para imprimir la respectiva serie.

```
18  Entero termino
19
20  Para i = 1 Hasta cantidad Incremento 1
21    termino = i * base
22    imprimir( termino, " " )
23  FinPara
```

En este caso la variable `termino` podría ser eliminada haciendo lo siguiente:

```
    Para i = 1 Hasta cantidad Incremento 1
      imprimir( (i * base), " " )
    FinPara
```

Aclaración:



El uso apropiado de procedimientos facilita no solo la reutilización de código, sino que se puede ganar legibilidad al simplificar la cantidad de líneas de la parte central del algoritmo o de otros procedimientos. Usar procedimientos permite "ocultar" o encapsular funcionalidades y por medio de las invocaciones se tiene acceso a ellas. Cada procedimiento se puede estudiar y comprender aisladamente y luego se crea un algoritmo y otros procedimientos que haciendo las respectivas invocaciones resuelven un problema mayor. Para el caso, el procedimiento imprime una serie, pero su doble invocación resolvió el problema inicial.

.:Ejemplo 5.4. *Se desea crear un procedimiento genérico que permita imprimir la coordenadas (x, y) de un punto cualquiera en un plano cartesiano, con su respectivo nombre del punto. (Ver Figura 5.3).*

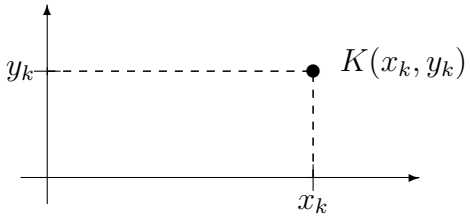


Figura 5.3: Coordenada de un punto K

Análisis del problema:

- **Resultados esperados:** la impresión del nombre del punto ("K") con sus respectivas coordenadas `x`, `y`.
- **Datos disponibles:** el valor de las coordenadas `x`, `y`.
- **Proceso:** solicitar al usuario las coordenadas del punto "K" y luego se invoca el procedimiento para imprimirlo.
- **Variables requeridas:**

  - `xk`: valor de la coordenada `x` del punto "K".
  - `yk`: valor de la coordenada `y` del punto "K".

  . Internos al procedimiento se necesitan:

  - `nombre`: nombre del punto arbitrario.
  - `x`: valor de la coordenada `x` del punto arbitrario.
  - `y`: valor de la coordenada `y` del punto arbitrario.

De acuerdo al análisis planteado, se propone el Algoritmo 5.5.

Algoritmo 5.5: Punto

```
1  Algoritmo Punto
2    // Declaración de las variables
3    Real xk, yk
4
5    // Datos disponibles
6    imprimir( "Ingrese el valor de x del punto K: " )
7    leer( xk )
8
9    imprimir( "Ingrese el valor de y del punto K: " )
10   leer( yk )
11
12   // Resultados esperados
13   imprimirPunto ( "K", xk, yk )
14 FinAlgoritmo
15
16 Procedimiento imprimirPunto( Cadena nombre, Real x, Real y )
17   imprimir( "El punto ", nombre )
18   imprimir( "tiene coordenadas (", x, ", ", y ")" )
19 FinProcedimiento
```

Al ejecutar el algoritmo:

```
Ingrese el valor de x del punto K: 5
Ingrese el valor de y del punto K: 73
El punto K tiene coordenadas ( 5, 73 )
```

```
Ingrese el valor de x del punto K: 31
Ingrese el valor de y del punto K: 5
El punto K tiene coordenadas ( 31, 5 )
```

Explicación del algoritmo:

En las primeras líneas de la tercera a la décima, se declaran las variables del algoritmo y posteriormente se leer los datos disponibles. Luego le invoca el método para imprimir el punto (`imprimirPunto`), enviando la información que él requiere (nombre del punto, las coordenadas `x`, `y`).

Observe que los parámetros (`nombre`, `x`, `y`) toman los valores "K", el valor de la coordenada x del punto k (`xk`) y el valor de la coordenada y del punto k (`yx`).

.:Ejemplo 5.5. *Usando el procedimiento del punto anterior, diseñe un algoritmo que permita imprimir los datos de dos puntos "T" y "S" (Ver Figura 5.4).*

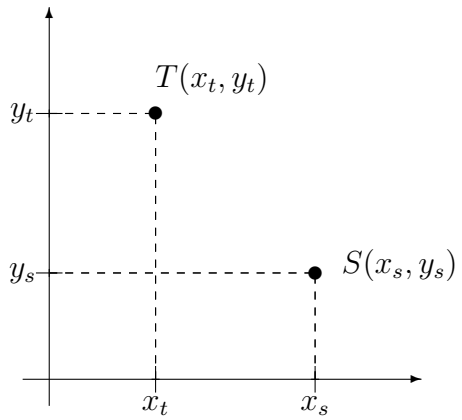


Figura 5.4: Puntos S, T en el plano cartesiano

Análisis del problema:

- **Resultados esperados:** la impresión del nombre del punto ("T" y "S") con sus respectivas coordenadas `x`, `y`.

- **Datos disponibles:** el valor de las coordenadas `x`, `y` de ambos puntos.
- **Proceso:** solicitar al usuario las coordenadas de los puntos "T" y "S" y luego se invoca dos veces el procedimiento para imprimir la información.
- **Variables requeridas:**

  - `xt`: valor de la coordenada `x` del punto "T".
  - `yt`: valor de la coordenada `y` del punto "T".
  - `xs`: valor de la coordenada `x` del punto "S".
  - `ys`: valor de la coordenada `y` del punto "S".

  . Internos al procedimiento se necesitan:

  - `nombre`: nombre del punto arbitrario.
  - `x`: valor de la coordenada `x` del punto arbitrario.
  - `y`: valor de la coordenada `y` del punto arbitrario.

  .

De acuerdo al análisis planteado, se propone el Algoritmo 5.6.

Algoritmo 5.6: Puntos

```
1  Algoritmo Puntos
2    // Declaración de las variables
3    Real xt, yt, xs, ys
4
5    // Datos disponibles
6    imprimir( "Ingrese el valor de x del punto T: " )
7    leer( xt )
8
9    imprimir( "Ingrese el valor de y del punto T: " )
10   leer( yt )
11
12   imprimir( "Ingrese el valor de x del punto S: " )
13   leer( xs )
14
15   imprimir( "Ingrese el valor de y del punto S: " )
16   leer( ys )
17
18   // Resultados esperados
19   imprimirPunto ( "T", xt, yt )
20   imprimirPunto ( "S", xs, ys )
21 FinAlgoritmo
```

```
22
23 Procedimiento imprimirPunto( Cadena nombre, Real x, Real y )
24   imprimir( "El punto ", nombre )
25   imprimir( "tiene co
```

Al ejecutar el algoritmo:

```
Ingrese el valor de x del punto T: 31
Ingrese el valor de y del punto T: 5
Ingrese el valor de x del punto S: 19
Ingrese el valor de y del punto S: 73
El punto T tiene coordenadas ( 31, 5 )
El punto S tiene coordenadas ( 19, 73 )
```

```
Ingrese el valor de x del punto T: 10
Ingrese el valor de y del punto T: 27
Ingrese el valor de x del punto S: 1
Ingrese el valor de y del punto S: 31
El punto T tiene coordenadas ( 10, 27 )
El punto S tiene coordenadas ( 1, 31 )
```

Explicación del algoritmo:

Observe que adicional a la declaración y lectura de los datos disponibles, se hace la invocación al mismo procedimiento pero enviando la información correspondiente en cada caso. Note que el procedimiento `imprimirPunto` no sufrió modificación alguna, esta es una de las fortalezas del uso de procedimientos en los algoritmos, el poder reutilizar código.

En la Figura 5.5 se muestra la solución del Ejemplo 5.5 mediante un diagrama de flujo.

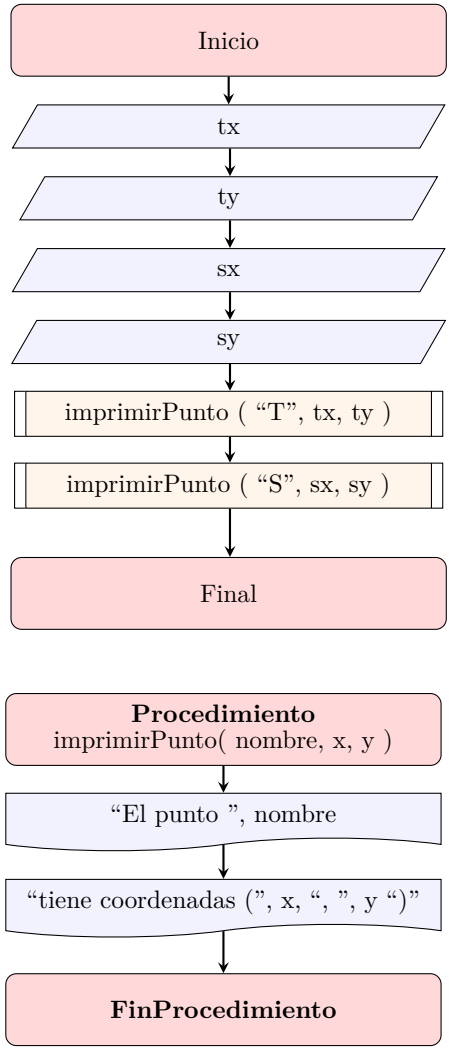


Figura 5.5: Diagrama de flujo del Algoritmo `Puntos`

5.2. Funciones

Un tipo especial de procedimiento es aquel que tiene la capacidad de entregar un valor como respuesta. A este tipo de procedimiento se le conoce con el nombre de **Función**. Algunos autores definen los procedimientos como funciones que no retornan ningún tipo de valor, aunque esta aproximación en principio es correcta, es en realidad imprecisa desde el punto de vista del concepto matemático de función el cual se desea emplear en programación.

Así como se definió la palabra reservada `Procedimiento`, ahora se define `Funcion` (sin acento), para declarar una función en un algoritmo. Adicional a esta palabra, se declaran `FinFuncion` y `Retornar`; la primera se utiliza para indicar donde termina la nueva función y la última es necesaria para indicar cual es el valor que la función entrega como resultado. Este valor debe coincidir con el tipo de dato que se indicó en la función, justo antes de su nombre.

La forma general de esta estructura es presentada en el segmento del Algoritmo 5.7.

Algoritmo 5.7: Forma general de una `Funcion`

```
1  Funcion tipoDato verboComplemento ( [lista de parámetros]
       )
2    Instrucción1
3    Instrucción2
4    ...
5    Instrucciónn
6
7    Retornar valor
8  FinFuncion
```

Aclaración:



En los lenguajes que no tienen paso de parámetros por referencia, como es el caso del lenguaje algorítmico utilizado en este libro, las funciones solo retornan un único valor.

Cuando el lenguaje dispone de paso de parámetros por referencia, es posible usar más de un parámetro para "Retornar" valores, sin embargo, al final, esta no es una buena práctica. Lo ideal en cualquier caso, es declarar perfectamente la funcionalidad y retornar un único valor.

En este libro, así como en los lenguajes de programación, existe un conjunto de procedimientos y funciones disponibles que pueden ser utilizados directa o indirectamente para resolver problemas. Ejemplos para el libro: `leer`, `imprimir`, `longitud`, `sin`, `cos`, `tan`, `abs`, `raizCuadrada`, entre otros.

.:Ejemplo 5.6. *Diseñe un algoritmo que permita determinar la distancia entre dos puntos "T" y "S" (Ver Figura 5.6).*

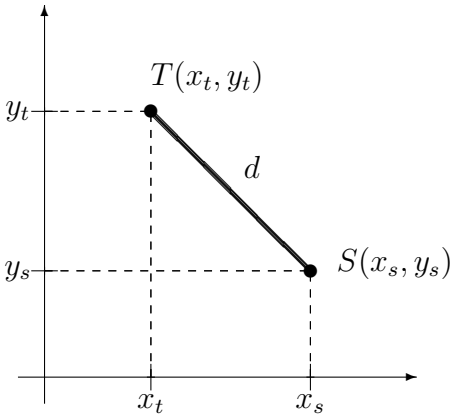


Figura 5.6: Distancia entre dos puntos T y S

Análisis del problema:

- **Resultados esperados:** la distancia entre los puntos "T" y "S".
- **Datos disponibles:** las coordenadas `x`, `y` de ambos puntos.
- **Proceso:** solicitar al usuario que ingrese las coordenadas `x`, `y` de ambos puntos. Posteriormente se procede a calcular la distancia entre los puntos usando la ecuación: $d = \sqrt{(x_s - x_t)^2 + (y_s - y_t)^2}$
- **Variables requeridas:**
  - `xt`: valor de la coordenada `x` del punto "T".
  - `yt`: valor de la coordenada `y` del punto "T".
  - `xs`: valor de la coordenada `x` del punto "S".
  - `ys`: valor de la coordenada `y` del punto "S".
  - `distancia`: valor de la distancia entre los puntos "T" y "S".

  . Internos al procedimiento `calcularDistancia` se necesitan:

  - Parámetros:
    - `x1`: valor de la coordenada `x` del primer punto.
    - `y1`: valor de la coordenada `y` del primer punto. arbitrario.
    - `x2`: valor de la coordenada `x` del segundo punto.

- y2: valor de la coordenada y del segundo punto.

.

- d: distancia entre dos puntos.

De àcuerdo al análisis planteado, se propone el Algoritmo 5.8.

Algoritmo 5.8: DistanciaPuntos

```
1  Algoritmo DistanciaPuntos
2    // Declaración de las variables
3    Real xt, yt, xs, ys, distancia
4
5    // Datos disponibles
6    imprimir( "Ingrese el valor de x del punto T: " )
7    leer( xt )
8
9    imprimir( "Ingrese el valor de y del punto T: " )
10   leer( yt )
11
12   imprimir( "Ingrese el valor de x del punto S: " )
13   leer( xs )
14
15   imprimir( "Ingrese el valor de y del punto S: " )
16   leer( ys )
17
18   // Calculo de los datos esperados
19   distancia = calcularDistancia ( xt, yt, xs, ys )
20
21   // Resultados esperados
22   imprimir ( "La distancia entre T y S es de ", distancia )
23 FinAlgoritmo
24
25 Funcion Real calcularDistancia( Real x1, Real y1,
                                   Real x2, Real y2 )
26   Real d
27   d = raizCuadrada( (x2 - x1)^2 + (y2 - y1)^2 )
28   Retornar d
29 FinFuncion
```

Al ejecutar el algoritmo:

Primera ejecución

```
Ingrese el valor de x del punto T: 10
Ingrese el valor de y del punto T: 4
Ingrese el valor de x del punto S: 6
Ingrese el valor de y del punto S: 7
La distancia entre T y S es de 5.0
```

Segunda ejecución

```
Ingrese el valor de x del punto T: 10
Ingrese el valor de y del punto T: 27
Ingrese el valor de x del punto S: 1
Ingrese el valor de y del punto S: 31
La distancia entre T y S es de 9,848857802
```

Explicación del algoritmo:

En primeras líneas (de la 6 a la 16) se declaran las variables y se leer los datos disponibles, luego se invoca el procedimiento `calcularDistancia` enviando la información disponibles, es decir, se invoca con las coordenadas del punto "T" y el punto "S". El procedimiento recibe esta información en sus respectivos parámetros `x1`, `y1`, `x2`, `y2` paa luego aplicar la formula de distancia y retornar el valor calculado mediante la variable local `d`. El procedimiento puede ser usado para calcular la distancia entre cualquier par de puntos (ver el siguiente ejemplo).

.:Ejemplo 5.7. *Diseñe un algoritmo que calcule e imprima el perímetro*[1] *de un triangulo dadas las coordenadas de cada uno de sus vértices. (Ver Figura 5.7).*

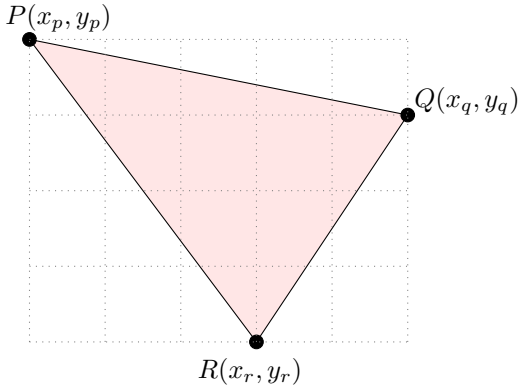


Figura 5.7: Triángulo de vértices P, Q y R

Análisis del problema:

- **Resultados esperados:** el perímetro de un triángulo dados los tres vértices del mismo, denominados arbitrariamente "P", "Q", y "R".

[1]El perímetro de una figura, se calcula como la suma de las longitudes de los lados de la figura. La longitud se puede determinar calculando la distancia entre los puntos extremos de cada lado (vertices).

- **Datos disponibles:** las coordenadas `x`, `y` de cada uno de los tres vértices.
- **Proceso:** se le solicita al usuario que ingrese las coordenadas `x`, `y` de cada uno de los tres vértices, luego se determina la distancia entre ellos:

```
d1 = calcularDistancia( x1, y1, x2, y2 )
d2 = calcularDistancia( x1, y1, x3, y3 )
d3 = calcularDistancia( x2, y2, x3, y3 )
```

  Posteriormente se realiza la suma de las distancias para encontrar el valor del perímetro:

```
perimetro   = d1 + d2 + d3
```

- **Variables requeridas:**

  - `xt`: valor de la coordenada `x` del punto "P".
  - `yt`: valor de la coordenada `y` del punto "P".
  - `xs`: valor de la coordenada `x` del punto "Q".
  - `ys`: valor de la coordenada `y` del punto "Q".
  - `xs`: valor de la coordenada `x` del punto "R".
  - `ys`: valor de la coordenada `y` del punto "R".
  - `perimetro`: valor del triangulo con vértices "P", "Q" y "R".

  . Internos al procedimiento `calcularPerimetro` se necesitan:

  - Parámetros:
    - `x1`: valor de la coordenada `x` del primer punto.
    - `y1`: valor de la coordenada `y` del primer punto.
    - `x2`: valor de la coordenada `x` del segundo punto.
    - `y2`: valor de la coordenada `y` del segundo punto.
    - `x3`: valor de la coordenada `x` del tercer punto.
    - `y3`: valor de la coordenada `y` del tercer punto.
  - `d1`: valor de distancia entre el primer y segundo punto.
  - `d2`: valor de distancia entre el primero y tercer punto.
  - `d3`: valor de distancia entre el segundo y tercer punto.
  - `perimetro`: valor del perímetro calculado como la suma de las distancias entre los puntos.

  . Internos al procedimiento `calcularDistancia` se necesitan:

  - Parámetros:

  - x1: valor de la coordenada x del primer punto.
  - y1: valor de la coordenada y del primer punto. arbitrario.
  - x2: valor de la coordenada x del segundo punto.
  - y2: valor de la coordenada y del segundo punto.

  .

- d: distancia entre dos puntos.

De acuerdo al análisis planteado, se propone el Algoritmo 5.9.

Algoritmo 5.9: DistanciaPuntos

```
1  Algoritmo DistanciaPuntos
2    // Declaración de las variables
3    Real xp, yp, xq, yq, xr, yr, perimetro
4
5    // Datos disponibles
6    imprimir( "Ingrese el valor de x del punto P: " )
7    leer( xp )
8
9    imprimir( "Ingrese el valor de y del punto P: " )
10   leer( yp )
11
12   imprimir( "Ingrese el valor de x del punto Q: " )
13   leer( xq )
14
15   imprimir( "Ingrese el valor de y del punto Q: " )
16   leer( yq )
17
18   imprimir( "Ingrese el valor de x del punto R: " )
19   leer( xr )
20
21   imprimir( "Ingrese el valor de y del punto R: " )
22   leer( yr )
23
24   // Calculo de los datos esperados
25   perimetro = calcularPerimetro ( xp, yp, xq, yq, xr, yr )
26
27   // Resultados esperados
28   imprimir ( "El perímetro del triangulo es ", perimetro )
29 FinAlgoritmo
30
31 Funcion Real calcularPerimetro( Real x1, Real y1,
32                                 Real x2, Real y2,
33                                 Real x3, Real y3 )
34   Real perimetro, d1, d2, d3
35
36   d1 = calcularDistancia ( x1, y1, x2, y2 )
37   d2 = calcularDistancia ( x1, y1, x3, y3 )
```

```
38   d3 = calcularDistancia ( x2, y2, x3, y3 )
39
40   perimetro = d1 + d2 + d3
41
42   Retornar perimetro
43 FinFuncion
44
45 Funcion Real calcularDistancia( Real x1, Real Y1,
46                                 Real X2, Real Y2 )
47   Real d
48   d = raizCuadrada( (x2 - x1)^2 + (y2 - y1)^2 )
49   Retornar d
50 FinFuncion
```

Al ejecutar el algoritmo:

Primera ejecución

```
Ingrese el valor de x del punto P: 0
Ingrese el valor de y del punto P: 4
Ingrese el valor de x del punto Q: 5
Ingrese el valor de y del punto Q: 3
Ingrese el valor de x del punto R: 3
Ingrese el valor de y del punto R: 0
El perímetro del triangulo es 13,70457079
```

Segunda ejecución

```
Ingrese el valor de x del punto P: 5
Ingrese el valor de y del punto P: 3
Ingrese el valor de x del punto Q: 5
Ingrese el valor de y del punto Q: 7
Ingrese el valor de x del punto R: 8
Ingrese el valor de y del punto R: 7
El perímetro del triangulo es 12
```

Explicación del algoritmo:

Lo primero es observar que en este ejemplo se emplearon dos procedimientos más la parte central del algoritmo. En la parte central se declaran y se solicitan todos los datos disponibles (líneas de la 3 a la 22), luego se invoca la función que calcula el perímetro del triángulo y finalmente se imprime el resultado obtenido.

Hasta este punto no es necesario conocer el cómo se calcula el perímetro, debido a que todo está “oculto” dentro el procedimiento (encapsulado). Por tanto, la parte central se limita a solicitar la información, calcular los resultados esperados e imprimirlos.

Por otro lado, la función `calcularPerimetro` recibe la información de las coordenadas de los tres vértices y procede a calcular el perímetro mediante la suma de las distancias entre los vértices. Pero esta distan es calculada mediante la función `calcularDistancia`.

```
31 Funcion Real calcularPerimetro( Real x1, Real y1,
32                                 Real x2, Real y2,
33                                 Real x3, Real y3 )
34   Real perimetro, d1, d2, d3
35
36   d1 = calcularDistancia ( x1, y1, x2, y2 )
37   d2 = calcularDistancia ( x1, y1, x3, y3 )
38   d3 = calcularDistancia ( x2, y2, x3, y3 )
39
40   perimetro = d1 + d2 + d3
41
42   Retornar perimetro
43 FinFuncion
```

La función `calcularDistancia` recibe la información de los dos vertices arbitrarios y calcula la distancia entre ellos, así, al invocar tres veces la función con la información apropiada (líneas de la 36 a la 38) es posible calcular todas las distancias necesarias.

```
45 Funcion Real calcularDistancia( Real x1, Real Y1,
46                                 Real X2, Real Y2 )
47   Real d
48   d = raizCuadrada( (x2 - x1)^2 + (y2 - y1)^2 )
49   Retornar d
50 FinFuncion
```

Aclaración:



Aunque en apariencia los algoritmos se vuelven más largos, el hacer uso de procedimientos / funciones permite la solución de problemas cada vez más complejos, al permitir atacar la complejidad del problema al dividiéndolos en componentes más pequeños y fáciles de manejar.

Buenas prácticas:



Dejar lo más claro posible cada elemento del algoritmo:

- Documente cuando sea conveniente y sin abusar de este recurso.
- Definida cual es la responsabilidad de cada función / procedimiento. Evite que ellas tengan más de una responsabilidad.
- Use nombre para las funciones / procedimiento acordes con la responsabilidad asociada.
- Evite exagerar en la cantidad de parámetros de un función / procedimiento necesita.
- Controle la longitud de todas y cada una de las funciones / procedimiento para evitar que superen una página, de ser el caso, delegue parte de la responsabilidad en otras funciones / procedimientos.

A continuación se presentan una serie de ejercicios para ser resueltos mediante algoritmos que hagan uso de funciones / procedimiento. Cada ejercicio debe estar acompañado de su respectivo análisis y ejemplos ilustrativos de su funcionamiento. En algunos casos, el lector deberá investigar las formulas necesarias para llegar a la solución.

5.3. Ejercicios propuestos

1. Diseñe un algoritmos con funciones y procedimientos que permita determinar el área de un hexágono.

2. Diseñe un algoritmos con funciones y procedimientos que permita que un arquitecto pueda determinar fácilmente el área total de un parque infantil que esta diseñando. El arquitecto puede ajustar cada una de las longitudes de las zonas del parque para ver como el cambio afecta el área total. El parque está formado por: 3 zonas en forma de hexágono en donde los niños juegan con arena, 2 zonas circulares para reuniones y 4 zonas rectangulares para los juegos mecánicos.

3. Diseñe un algoritmos con funciones y procedimientos que permita determinar las coordenadas de un triangulo si se conocen las longitudes de cada uno de los tres lados.

4. Diseñe un algoritmos con funciones y procedimientos que indique el valor del descuento de un artículo el cual es del 5% solo si el artículo tiene un costo superior al $150.000.

5. Diseñe un algoritmos con funciones y procedimientos que indique si la llave de un tanque de agua debe ser abierta o cerrada. El tanque debe estar siempre entre 250 y 450 litros.

6. Diseñe un algoritmos con funciones y procedimientos que dado un número entero entre 0 y 20 diga si es o no un número primo.

   Recuerde que los números primeros menores o iguales a 20 son: $2, 3, 5, 7, 11, 13, 17, 19$.

7. Diseñe un algoritmos con funciones y procedimientos que indique si un estudiante ganó o perdió un curso después de presentar los cinco trabajos asociados al curso (Notas entre 0.0 y 5.0). Los trabajos tiene igual peso sobre la nota final y se gana el curso si la nota definitiva es superior a 3.5

8. Diseñe un algoritmos con funciones y procedimientos que permita saber si una ecuación cuadrática tiene o no solución.

   Recuerde que una ecuación cuadrática se define como:

   $$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

   Y se dice que tiene solución si el valor a calcular la raíz cuadrada es mayor o igual a cero y el valor de a es diferente de cero.

9. Diseñe un algoritmos con funciones y procedimientos que indique si un número entero x se encuentra por dentro o por fuera el intervalo cerrado-cerrado $[minimoValor, maximoValor)]$

   Por ejemplo: Si los valores mínimos y máximo son 3 y 7 respectivamente, el valor 5 está dentro, mientras que el valor de 8 está por fuera del intervalo.

10. Diseñe un algoritmos con funciones y procedimientos que indique si un número entero x se encuentra por dentro o por fuera de tres intervalos abierto-abierto cuyo rangos no se interceptan entre sí y sus límites son ingresados por el usuario.

Capítulo 6

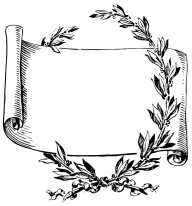
Vectores y matrices

La mejor forma de predecir el futuro es implementarlo

David Heinemeier Hansson

Objetivos del capítulo:

- Aprender a declarar e inicializar vectores y matrices.
- Conocer la forma de leer e imprimir los elementos almacenados en vectores y matrices.
- Construir algoritmos para buscar elementos en vectores y matrices.
- Realizar operaciones básicas con varios vectores y matrices.



En los capítulos anteriores se ha trabajado con variables que almacenan solamente un dato a la vez. Esto quiere decir que, si se asigna un valor a la variable y, posteriormente se asigna otro valor, el primer valor se pierde. Este tipo de variables reciben el nombre de **Variables escalares**.

Sin embargo, en muchos problemas de lógica de programación existe la necesidad de almacenar muchos datos del mismo tipo como, por ejemplo, las estaturas de 50 personas, las edades de los 30 estudiantes de un grupo o el peso de 40 pacientes de una clínica. Una solución para estos problemas sería declarar 50 variables para almacenar las estaturas, 30 variables para las edades y 40 para los pesos, sin embargo, esto es poco práctico. Para este tipo de problemas, existen unas variables denominadas **Variables suscritas** que almacenan el lugar (referencia) en donde se crearon múltiples espacios o celdas de memoria que pueden ser usados para guardar y recuperar datos (llamado **Arreglo**).

Un arreglo es entonces, un conjunto de espacios o celdas de memoria donde se pueden almacenar temporalmente muchos datos siempre y cuando sean del mismo tipo. De esta forma y, regresando al caso de tener la necesidad de guardar en un algoritmo 50 estaturas del mismo número de personas, se podrían declarar 50 variables distintas o, sencillamente, una variable suscrita que referencia a un arreglo de 50 posiciones.

Existen varios tipos de arreglos, a saber: arreglos unidimensionales también conocidos como **Vectores**, los arreglos bidimensionales o **Matrices** y los arreglos multidimensionales. Todo arreglo requiere de una variable suscrita para poder acceder a sus datos y todos ellos son del mismo tipo, el cual debe coincidir con el tipo de dato de la variable suscrita. El vector funciona como una fila en la cual se ubican los datos desde la primera celda hasta la última. Las matrices son similares, pero con la diferencia de que trabajan como una tabla de celdas con varias filas y columnas, es decir, que ocupa dos dimensiones; los arreglos multidimensionales, por su parte, funcionan con celdas que ocupan espacios en más de dos dimensiones.

6.1. Vectores

Los vectores son conjuntos finitos de celdas que permiten almacenar datos del mismo tipo, de acuerdo a como se haya declarado. Para facilitar su comprensión, gráficamente un arreglo unidimensional se observa como una fila donde cada celda corresponde a un espacio de memoria en el que se puede ubicar un dato. Cada celda o espacio de memoria es identificado

con un número llamado índice. En la Figura 6.1 se puede ver un vector de 10 posiciones:

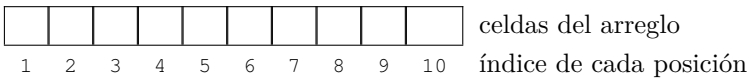


Figura 6.1: Diagrama de un Vector

Los índices empiezan en uno[1], aunque existen lenguajes de programación en que empiezan en cero, y son números enteros. El último número índice representa no solo a la última celda, sino también el tamaño del arreglo[2], es decir, el número de celdas que posee. Aunque normalmente los índices son números positivos, existen lenguajes como Python que soportan números negativos, los cuales se interpretan como moverse en sentido contrario, es decir, el índice -1 representa la última posición. Este tipo de índices no será utilizado en este libro.

6.1.1 Declaración de un vector

Con el fin de poder utilizar un vector (o arreglo en general) de cualquier tipo dentro de un algoritmo, se requiere el uso de una variable suscrita que lo referencie. Una variable suscrita se diferencia de las demás por tener en su declaración los símbolos corchetes [ ].

```
Entero   edadEstudiante [ ]
Real     estaturaPersona[ ]
Cadena   nombreEmpleado [ ]
Caracter letrasDocumento[ ]
```

Es importante aclarar que, luego de declarar la variable suscrita, se debe especificar el tamaño del vector; esto permitirá determinar la cantidad de celdas, casillas o espacios de memoria que tendrá el vector. Lo anterior se puede hacer de dos maneras: usando la función llamada `dimensionar()` (esta función es propia del libro, cada lenguaje tiene su forma particular de crear los arreglos), o en el momento de declarar la variable suscrita.

[1]En algunos lenguajes de programación como por ejemplo C y Java, los índices comienzan en cero.

[2]En lenguajes como C y Java es el tamaño del arreglo menos uno, debido a que la primera posición (índice 0) también es contada.

Ejemplos de creación de arreglos con la función `dimensionar()` :

1. Se declara una variable suscrita llamada `edadEstudiante` y se le da un tamaño de 6 posiciones para datos de tipo `Entero`. (Ver Figura 6.2).

```
Entero edadEstudiante[ ]

edadEstudiante = dimensionar( 6 )
```

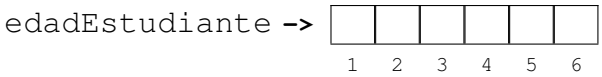


Figura 6.2: Vector `edadEstudiante`

2. Se declara una variable suscrita llamada `estaturaPersona` y se le da un tamaño de 4 posiciones para datos de tipo `Real`. (Ver Figura 6.3).

```
Real estaturaPersona[ ]

estaturaPersona = dimensionar( 4 )
```

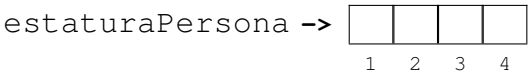


Figura 6.3: Vector `estaturaPersona`

3. Se declara una variable suscrita llamada `nombreEmpleado` y se le da un tamaño de 3 posiciones para datos de tipo `Cadena`.(Ver Figura 6.4).

```
Cadena nombreEmpleado[ ]

nombreEmpleado = dimensionar( 3 )
```



Figura 6.4: Vector `nombreEmpleado`

4. Se declara una variable suscrita llamada `letrasDocumento` y se le da un tamaño de 50 posiciones para datos de tipo `Caracter`. (Ver Figura 6.5).

```
Caracter letrasDocumento [ ]

letrasDocumento = dimensionar( 50 )
```

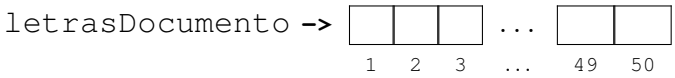


Figura 6.5: Vector `letrasDocumento`

La función `dimensionar()` no requiere que se envíe una constante numérica como argumento, también se puede usar el contenido de una variable o constante de tipo `Entero`. Por ejemplo:

```
Entero edadEstudiante[ ]
Entero cantidadPersonas

cantidadPersonas = 20
edadEstudiante   = dimensionar( cantidadPersonas )
```

Aclaración:



Aunque se puede declarar una variable en cualquier parte del algoritmo, sin ser un error de lógica, lo ideal sería realizar siempre las declaraciones al inicio del algoritmo o de los procedimientos o funciones.

La función `dimensionar()`, internamente está construida creando un vector en el momento de declarar la variable suscrita del tamaño indicado en el argumento cuando se invoca la función, para posteriormente retornar la referencia a este nuevo vector:

```
Funcion tipoDato[ ] dimensionar( Entero n )
  tipoDato elemento[ n ]

  Retornar elemento
FinFuncion
```

Note que esta función no indica un tipo de dato específico, ya que funciona igual para cualquier tipo de dato; se describe de esta forma para lograr explicar el funcionamiento en términos generales. Se asume que esta

función es interna al pseudocódigo del libro y no requiere ser escrita en los algoritmos que la utilicen.

La estructura interna de la función es: crear un vector de n posiciones en el momento de la declaración de la variable suscrita, y luego simplemente se retorna el valor de la variable, es decir, la referencia al nuevo vector.

Ejemplos de creación de vectores en el momento de la declaración :

1. Se declara una variable suscrita llamada `semestreEstudiante` con un tamaño de 35 posiciones para datos de tipo `Entero`. (Ver Figura 6.6).

```
Entero semestreEstudiante [ 35 ]
```

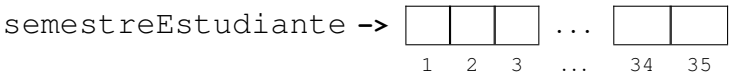


Figura 6.6: Vector `semestreEstudiante`

2. Se declara una variable suscrita llamada `correoElectronicoEmpleado` con un tamaño de 30 posiciones para datos de tipo `Cadena`. (Ver Figura 6.7).

```
Cadena correoElectronicoEmpleado [ 30 ]
```

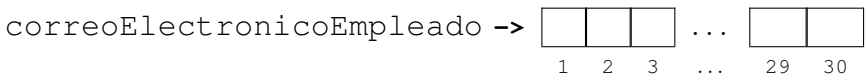


Figura 6.7: Vector `correoElectronicoEmpleado`

3. Se declara una variable suscrita llamada `sueldoPersona` con un tamaño de 100 posiciones para datos de tipo `Real`. (Ver Figura 6.8).

```
Real sueldoPersona [ 100 ]
```

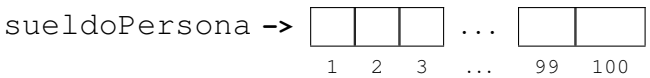


Figura 6.8: Vector `sueldoPersona`

En los anteriores casos, se declaró la variable suscrita y también se especificó su tamaño. Realizar este tipo de declaración y especificación de tamaño es útil cuando se tiene certeza del tamaño requerido del vector. Es de anotar que, en la mayoría de los ejercicios que se analizan más adelante, se utiliza la función `dimensionar()` ya que, en ellos, será el usuario quien determinará el tamaño que más le convenga en el momento de la ejecución del algoritmo.

Buena práctica:



Antes de intentar leer el contenido de un vector (o arreglo en general), debe haber certeza de que ya se ha almacenado previamente información en él.

Los vectores, una vez creados tienen disponibles un cierto número de casillas y cada una de ellas tiene un potencial dato almacenado. En algunos lenguajes de programación, por ejemplo Java, el contenido inicial siempre se conoce por definición; sin embargo, otros lenguajes como C y en especial en este libro, no se da ninguna certeza del valor inicial, por tanto es una buena práctica estar seguros del contenido de las casillas del vector antes de intentar usar su contenido. La información almacenada en este último caso se denomina "Información basura".

Para terminar, es importante decir que no es posible eliminar o insertar celdas en un arreglo, tal y como sucede en lenguajes como C y Java. Existen otras estructuras de almacenamiento que están por fuera del alcance del libro que sí permiten este tipo de operaciones, como por ejemplo las listas.

6.1.2 Almacenamiento de datos en un vector

Al almacenar datos en un vector, se debe tener en cuenta que el almacenamiento se debe hacer celda a celda de forma paulatina. Esto quiere decir que cada dato debe irse ubicando en una celda específica que se identifica con su número índice. De esta manera y suponiendo que se tiene ya un arreglo declarado de tipo `Entero`, denominado `numero` en el que se van a almacenar 5 números, el almacenamiento se haría de la siguiente forma:

```
Entero numero[ 5 ]
```

```
numero[ 1 ] = 4
numero[ 2 ] = 2
numero[ 3 ] = 15
numero[ 4 ] = 8
numero[ 5 ] = 3
```

Estas asignaciones hacen que en la primera posición del vector `numero` se almacene un 4, en la segunda posición un 2, en la tercera un 15, en la cuarta un 8 y en la quinta un 3. Gráficamente, el arreglo se vería como en la Figura 6.9.

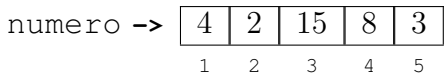


Figura 6.9: Ejemplo almacenamiento en un vector

También es posible almacenar en una posición específica de un vector un dato que ya esté almacenado en una variable, siempre y cuando la variable y el vector sean del mismo tipo de dato. Por ejemplo, suponga que se tiene una variable entera denominada `edad` que tiene almacenado un 20. El contenido de esta variable puede almacenarse en una posición cualquiera del vector de la siguiente forma:

```
Entero numero[ 5 ], edad

edad = 20
numero[ 4 ] = edad
```

Esta instrucción hará que el valor almacenado en la variable `edad` quede también almacenado en la posición 4 del vector.

También es posible, en el momento de la declaración, almacenar varios datos en un vector con una sola instrucción, de la siguiente manera:

```
Entero numero[ ] = {7, 21, 93, 48, 5}
```

De esta forma, el número 7 se almacenará en la posición 1 de `numero`, 21 en la posición 2, 93 en la posición 3, 48 en la posición 4 y 5 en la posición 5 de este arreglo. (Ver Figura 6.10).

Ahora, es importante analizar cómo puede el usuario realizar una lectura de datos y almacenarlos en un vector. Aunque es posible hacer una lectura manual para cada posición, lo ideal sería hacer uso de un ciclo. Suponga que se desean leer, por ejemplo, 20 números enteros:

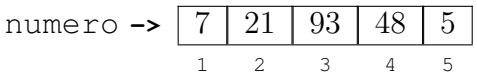


Figura 6.10: Ejemplo de asignación directa

```
Entero numero[ 20 ], i

Para i = 1 Hasta 20 Incremento 1
  imprimir( "Ingrese el número " , i, ": " )
  leer( numero[ i ] )
FinPara
```

En esta porción de código, la variable `i` representa el índice con el cual se irá accediendo a cada posición del arreglo. El ciclo se ejecutará 20 veces desde 1 hasta 20, lo que coincide con todas las posiciones del arreglo. La función `imprimir()` solicitará al usuario el ingreso de un número cualquiera. La variable `i` que va al final de la función `imprimir()` dentro del paréntesis servirá para que el usuario reconozca la secuencia de los números que va ingresando. La función `leer()` llevará el número ingresado por el usuario a la respectiva posición del vector `numero` representada por el índice.

6.1.3 Recuperación de datos almacenados en un vector

Al recuperar un dato que se encuentre en una posición específica de un vector, es necesario saber cuál es esa posición y luego acceder a ella a través de alguna instrucción. Por ejemplo, supóngase que se quiere conocer cuál es el dato almacenado en la posición 3 del vector `numero`. (Ver Figura 6.11).

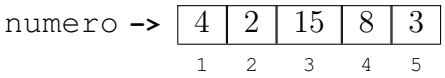


Figura 6.11: Ejemplo recuperación de datos

Para ello, será necesario escribir la siguiente instrucción:

```
imprimir( "El tercer elemento es " , numero[ 3 ] )
```

Luego de utilizar esta instrucción y, suponiendo que los datos almacenados son los de la imagen anterior, el dato mostrado con la función

imprimir() es 15.

Ahora bien, si este número se requiere para llevar a cabo algún cálculo, este puede llevarse a otra variable, siempre y cuando la variable que lo va a almacenar temporalmente sea del mismo tipo de dato que el arreglo de donde proviene el dato. Como ejemplo, suponga que el dato de la posición 5 del arreglo se requiere para un cálculo particular, con la siguiente instrucción, el dato se copia en la variable n que se usará para ese fin:

```
Entero n, numero[ ] = {7, 21, 93, 48, 5}

n = numero[ 4 ]
```

Luego de ejecutar esta instrucción, una copia del valor 48 quedará almacenado en la variable n. Tenga en cuenta que el 48 no se borrará de la posición 4 de numero, solo se habrá copiado.

También es posible que en algunos problemas de lógica de programación se presente la necesidad de recuperar todos los datos almacenados en las diferentes posiciones del vector. En estos casos, deberá recorrerse con un ciclo el vector, desde la primera hasta la última posición. Como ejemplo de esta situación, imagine que se desean sumar los valores almacenados en numero e informarle al usuario el total de la suma. Para ello, se tiene la siguiente porción de código:

```
Entero numero[  ] = {7, 21, 93, 48, 5}
Entero i, suma

suma = 0
Para i = 1 Hasta 5 Incremento 1
  suma = suma + numero[ i ]
FinPara

imprimir( "La suma de los elementos es: ", suma )
```

En la anterior porción de código, se llevan a cabo varias tareas que se explican a continuación:

- Se declara una variable suma de tipo Entero que almacenará la suma de los datos almacenados en numero.
- La variable suma se inicializa en 0, pues antes de empezar a sumar no se tiene nada acumulado.
- Se utiliza el ciclo Para con la variable i como contador e índice, con el fin de recorrer el vector desde la posición 1 hasta el tamaño del mismo (5).

- Se utiliza la instrucción dentro del ciclo para ir sumando el valor de la posición actual de `numero` con lo que se están acumulando los datos del vector en la variable `suma`.

```
suma = suma + numero[ i ]
```

- Al finalizar el ciclo se muestra el total de la `suma` mediante la función `imprimir()`.

En las siguientes secciones se utilizan varios ejercicios para ejemplificar el uso de los arreglos unidimensionales o vectores. Se describirán inicialmente, los primeros algoritmos escritos de manera secuencial; posteriormente, los algoritmos que aparecerán, se escribirán haciendo uso de funciones y procedimientos.

.:Ejemplo 6.1. *Diseñe un algoritmo que permita almacenar en un vector de 4 posiciones o celdas, números `Enteros` comprendidos entre 10 y 20 y que, a continuación, determine cuántas veces se encuentra almacenado el número 14 en el arreglo.*

Análisis del problema:

- **Resultados esperados:** mostrar la cantidad de veces que aparece el número 14 en las celdas de un arreglo de 4 posiciones.
- **Datos disponibles:** los cuatro números que se almacenan en el arreglo en el intervalo 10 al 20.
- **Proceso:** se solicita al usuario el tamaño del arreglo, posteriormente se hace la lectura de todos lo datos, luego se lleva a cabo una búsqueda del número 14 celda a celda, desde la primera posición hasta la última; cada vez que se encuentre, se va incrementando un contador. Al final del algoritmo, se mostrará la cantidad de veces que se ha encontrado el número 14.
- **Variables requeridas:**
  - `numero`: arreglo que almacena los números ingresados
  - `i`: variable que controla el ciclo
  - `contador14`: variable que almacena la cantidad de veces[3] que aparece el número 14 en el arreglo.

[3] contador

- `repetir`: bandera utilizada para determinar si es necesario solicitar nuevamente un dato debido a que el valor previamente ingresado no cumpla con la condición de estar entre 10 y 20.

De acuerdo al análisis planteado, se propone el Algoritmo 6.1.

Algoritmo 6.1: Numero14

```
1  Algoritmo Numero14
2    Constante Entero MAX = 4
3    Entero numero [ MAX ],  contador14, i
4    Logico repetir
5
6    Para i = 1 Hasta MAX Incremento 1
7      Haga
8        imprimir( "Ingrese el número " , i, ": " )
9        leer( numero[ i ] )
10
11       repetir = Falso
12       Si( numero[ i ] < 10 O
13           numero[ i ] > 20 ) Entonces
14         repetir = Verdadero
15         imprimir( "Este número no es válido" )
16       FinSi
17     MientrasQue( repetir == Verdadero )
18   FinPara
19
20   contador14 = 0
21   Para i = 1 Hasta MAX Incremento 1
22     Si( numero[  i ] == 14 ) Entonces
23       contador14 = contador14 + 1
24     FinSi
25   FinPara
26
27   imprimir( "El número 14 está ", contador14, " veces" )
28 FinAlgoritmo
```

Al ejecutar el algoritmo:

```
Ingrese el número 1: 14
Ingrese el número 2: 2
Este número no es válido
Ingrese el número 2: 18
Ingrese el número 3: 20
Ingrese el número 4: 14
El número 14 está 2 veces
```

Explicación del algoritmo:

Inicialmente, como en todos los algoritmos vistos, se declaran las variables necesarias (líneas 2 a 4), incluyendo el tamaño del arreglo mediante la constante MAX, la cual, para el ejemplo es de 4 elementos. Esta constante permite no solo inicializar el vector, sino que será el tope hasta el que iterará el ciclo recorriendo el vector hasta su última posición.

```
2   Constante Entero MAX = 4
3   Entero numero [ MAX ],  contador14, tamanio, i
4   Logico repetir
```

Con el primer ciclo Para, se recorre el vector desde la posición 1 representada por la variable i hasta la última posición del vector (especificada por la constante MAX) y se solicitan los números que se van a almacenar.

Dentro de este ciclo se encuentra otro ciclo Haga-MientrasQue, el cual se utiliza para validar que los números ingresados y que van a ser almacenados en las diversas posiciones del arreglo si se encuentren en el intervalo 10 al 20.

```
7     Haga
8       imprimir( "Ingrese el número " , i, ": " )
9       leer( numero[ i ] )
10
11      repetir = Falso
12      Si( numero[ i ] < 10 O
13          numero[ i ] > 20 ) Entonces
14        repetir = Verdadero
15        imprimir( "Este número no es válido" )
16      FinSi
17    MientrasQue( repetir == Verdadero )
```

Allí mismo se ubica una instrucción Si, que muestra un mensaje en el caso de que el número ingresado no haga parte del intervalo especificado, y cambia el estado de la bandera repetir para que pase de Falso a Verdadero. La bandera es importante para poder hacer que el ciclo Haga-MientrasQue realice una nueva iteración[4] o no.

Luego de este primer ciclo Para, se inicializa el contador de números 14 en cero, ya que todavía no se ha encontrado ninguno; a continuación, se utiliza un segundo ciclo Para que vuelve a recorrer el arreglo de números y dentro de él, con una instrucción Si, se pregunta si el dato contenido en cada celda es un 14, de ser así, se cuenta.

[4]El dato ingresado es inválido

```
20   contador14 = 0
21   Para i = 1 Hasta MAX Incremento 1
22     Si( numero[ i ] == 14 ) Entonces
23       contador14 = contador14 + 1
24     FinSi
25   FinPara
```

Al final, se muestra con la instrucción `imprimir` la cantidad de 14 encontrados en el arreglo.

```
27   imprimir( "El número 14 está ", contador14, " veces" )
```

.:Ejemplo 6.2. *Diseñe un algoritmo que permita almacenar los nombres de un grupo de* `n` *personas en un arreglo unidimensional y que posteriormente, busque la posición del arreglo en la que quedó almacenado el nombre de una persona que el usuario ingresa. Si ese nombre no aparece en el arreglo, se debe mostrar el respectivo mensaje.*

Aclaración:



Este ejercicio, aunque parecido al anterior, contiene varias diferencias fundamentales: se debe utilizar un vector que guarde los nombres de personas, una vez almacenadas, se debe buscar un nombre en particular y verificar si está almacenado o no en el arreglo; si está almacenado, deberá decirse la posición en la que se encuentra. En este ejercicio es necesario suponer que no se almacena un nombre más de una vez, es decir, no existen dos o más personas con el mismo nombre.

Análisis del problema:

- **Resultados esperados:** la posición del arreglo en la que se encuentra el nombre a buscar ingresado por el usuario.
- **Datos disponibles:** el tamaño del arreglo y el nombre de la persona que se desea buscar.
- **Proceso:** solicitar la cantidad de nombres a ingresar (`n`), luego se solicitan los nombres de las `n` personas y se almacenan en el arreglo, posteriormente se hace una búsqueda secuencial desde la primera hasta la última posición del arreglo de nombres comparando si el nombre que se está buscando es igual al nombre que hay en la posición actual, Si esto ocurre, ya se encontró lo que se buscaba y se puede

terminar la búsqueda allí mostrando la posición en la que el nombre se encuentra; si no se ha encontrado, se pasa a la siguiente posición y se vuelve a comparar hasta llegar al final del arreglo. Si se ha llegado al final del arreglo y no se encontró nada, deberá informarse al usuario.

- **Variables requeridas:**
  - `tamanio`[5]: contiene el tamaño del arreglo.
  - `arregloNombres`: es el arreglo que almacena los nombres.
  - `i`: variable de control del ciclo
  - `nombreBuscar`: almacena el nombre que se desea buscar
  - `encontrado`: variable que indica si se encontró o no el nombre en el arreglo.

Algoritmo 6.2: NombrePersonas

```
1  Algoritmo NombrePersonas
2    Entero tamanio, i
3    Cadena arregloNombres[ ], nombreBuscar
4    Logico encontrado
5
6    imprimir( "Ingrese el tamaño del arreglo: ")
7    leer( tamanio )
8
9    arregloNombres = dimensionar( tamanio )
10
11   Para i = 1 Hasta tamanio Incremento 1
12     imprimir( "Ingrese el nombre persona " , i, ": " )
13     leer( arregloNombres[ i ] )
14   FinPara
15
16   imprimir( "Ingrese el nombre de la persona a buscar: " )
17   leer( nombreBuscar )
18
19   encontrado = Falso
20   Para i = 1 Hasta tamanio Incremento 1
21     Si( nombreBuscar == arregloNombres[ i ] ) Entonces
22       imprimir( "El nombre está en la posición ", i )
23       encontrado = Verdadero
24     FinSi
25   FinPara
26
27   Si( encontrado == Falso ) Entonces
28     imprimir( "El nombre no está en el arreglo" )
```

[5]por norma, en los identificadores se evita el uso de caracteres como tildes y eñes.

```
29   FinSi
30 FinAlgoritmo
```

Al ejecutar el algoritmo:

Primera ejecución:

```
Ingrese el tamaño del arreglo: 3
Ingrese el nombre persona 1: Jacobo
Ingrese el nombre persona 2: Gabriela
Ingrese el nombre persona 3: Adriana
Ingrese el nombre de la persona a buscar: Adriana
El nombre está en la posición 3
```

Segunda ejecución:

```
Ingrese el tamaño del arreglo: 3
Ingrese el nombre persona 1: Jacobo
Ingrese el nombre persona 2: Gabriela
Ingrese el nombre persona 3: Adriana
Ingrese el nombre de la persona a buscar: Nidia
El nombre no está en el arreglo
```

Explicación del algoritmo:

En este algoritmo, como en todos los anteriores, lo primero es declarar las variables necesarias (líneas 2 a 4).

Observe que el arreglo de nombres es de tipo `Cadena`. Después de la declaración de las variables, se solicita el tamaño para, con este dato, dimensionar al arreglo.

```
6   imprimir( "Ingrese el tamaño del arreglo: ")
7   leer( tamanio )
8
9   arregloNombres = dimensionar( tamanio )
```

A continuación, se solicitan los nombres de las personas y se almacenan en cada posición del arreglo; esto se hace mediante el uso del primer ciclo `Para`.

```
11  Para i = 1 Hasta tamanio Incremento 1
12    imprimir( "Ingrese el nombre persona " , i, ": " )
13    leer( arregloNombres[ i ] )
14  FinPara
```

Luego, se solicita el nombre que se va a buscar y se inicializa una variable “centinela” (de tipo lógico) denominada encontrado en `Falso`.

Esta variable centinela permitirá terminar el ciclo inmediatamente después de encontrar el nombre que se está buscando.

```
16  imprimir( "Ingrese el nombre de la persona a buscar: " )
17  leer( nombreBuscar )
```

Con objeto de hacer la búsqueda, se utiliza un segundo ciclo `Para` y, dentro de este segundo ciclo se usa la instrucción `Si`, que compara si el nombre que se está buscando es igual al nombre almacenado en la celda actual del arreglo, si esto ocurre, se muestra un mensaje indicando la posición del arreglo donde está el nombre y se cambia el contenido de la variable encontrado a `Verdadero`, lo que indica que ya se encontró el nombre buscado.

```
19  encontrado = Falso
20  Para i = 1 Hasta tamanio Incremento 1
21    Si( nombreBuscar == arregloNombres[ i ] ) Entonces
22      imprimir( "El nombre está en la posición ", i )
23      encontrado = Verdadero
24    FinSi
25  FinPara
```

Finalmente, y luego del segundo ciclo, el algoritmo posee otra instrucción `Si`, que evalúa si la variable encontrado tiene almacenado un valor de `Falso`, lo cual indicaría que no se encontró el nombre buscado en el vector, por lo que se muestra este mensaje con la instrucción `imprimir`. Luego de esto, el algoritmo termina.

```
27  Si( encontrado == Falso ) Entonces
28    imprimir( "El nombre no está en el arreglo" )
29  FinSi
```

.:Ejemplo 6.3. *Se almacenan en un arreglo unidimensional* `n` *números enteros positivos. Diseñe un algoritmo que genere otro arreglo que almacene de forma invertida los elementos guardados en el arreglo inicial, es decir, el último elemento almacenado en el arreglo inicial que ocupa la posición* `n`*, será el primero en el arreglo generado y, el primer elemento almacenado en el arreglo inicial, ocupará la última posición en el arreglo generado.*

Análisis del problema:

- **Resultados esperados:** un segundo arreglo de números enteros y en este se almacenen los números que hay en el arreglo inicial de forma invertida a como están almacenados en este arreglo inicial.

- **Datos disponibles:** el tamaño del arreglo inicial y los números que se almacenarán en él.
- **Proceso:** luego de solicitar el tamaño del arreglo inicial y los números que se van a almacenar en él, se hace un recorrido a este arreglo desde la primera hasta la última posición y se van pasando los números al segundo arreglo, que debe recorrerse paralelamente desde la última hasta la primera posición.
- **Variables requeridas:**
  - `n`: número de elementos del arreglo
  - `i`: variable de control para moverse en el primer arreglo.
  - `j`: variable de control para moverse en el segundo arreglo.
  - `arregloInicial`: arreglo que contiene los números iniciales.
  - `arregloFinal`: arreglo que contiene los números en orden inverso.

De acuerdo al análisis planteado, se propone el Algoritmo 6.3.

Algoritmo 6.3: InversionArreglo

```
1  Algoritmo InversionArreglo
2    Entero arregloInicial[ ], arregloFinal[ ], n, i, j
3
4    imprimir( "Ingrese el tamaño del Arreglo: " )
5    leer( n )
6
7    arregloInicial = dimensionar( n )
8    arregloFinal   = dimensionar( n )
9
10   Para i = 1 Hasta n Incremento 1
11     Haga
12       imprimir( "Ingrese el número " , i, ": " )
13       leer( arregloInicial[ i ] )
14
15       Si( numero[ i ] < 0 ) Entonces
16         imprimir( "Este número no es válido" )
17       FinSi
18     MientrasQue( numero[ i ] < 0 )
19   FinPara
20
21   j = n
22   Para i = 1 Hasta n Incremento 1
23     arregloFinal[ j ] = arregloInicial[ i ]
24     j = j - 1
25   FinPara
```

```
26
27   Para i = 1 Hasta n Incremento 1
28     imprimir( arregloFinal[ i ], " " )
29   FinPara
30 FinAlgoritmo
```

Al ejecutar el algoritmo:

```
Ingrese el tamaño del Arreglo: 4
Ingrese el número 1: 3
Ingrese el número 2: 9
Ingrese el número 3: -8
Este número no es válido
Ingrese el número 3: 45
Ingrese el número 4: 92
92 45 9 3
```

Explicación del algoritmo:

Al principio de este algoritmo se hace la declaración de todas las variables necesarias (línea 2). Acto seguido, se solicita mediante las instrucciones `imprimir` y `leer`, el tamaño del arreglo, que es almacenado en la variable `n` y, con la cual se le da tamaño al arreglo inicial y final.

```
4   imprimir( "Ingrese el tamaño del Arreglo: " )
5   leer( n )
6
7   arregloInicial = dimensionar( n )
8   arregloFinal   = dimensionar( n )
```

Luego, en las instrucciones que siguen a continuación en el algoritmo, se utiliza un ciclo `Para` con el propósito de almacenar en cada posición del arreglo inicial los números enteros positivos que el usuario ingresa. Nótese que dentro de este primer ciclo `Para`, se usa un ciclo `Haga-MientrasQue` que valida y asegura el ingreso de números positivos, lo cual es un requisito propuesto en el enunciado.

Dentro de este ciclo `Haga-MientrasQue`, se usa una instrucción `Si` que muestra un mensaje solo si el número ingresado es menor que cero, haciéndole entender al usuario que este número no es válido para almacenarse en el `arregloInicial`, pues no corresponde a un número positivo.

```
10   Para i = 1 Hasta n Incremento 1
11     Haga
12       imprimir( "Ingrese el número " , i, ": " )
13       leer( arregloInicial[ i ] )
14
15       Si( numero[ i ] < 0 ) Entonces
16         imprimir( "Este número no es válido" )
17       FinSi
18     MientrasQue( numero[ i ] < 0 )
19   FinPara
```

En la segunda parte del algoritmo, se inicializa la variable `j` con el tamaño de este arreglo, es decir, se busca ubicar la última posición de este arreglo utilizando la variable `j`. Se utiliza un segundo ciclo `Para` que recorre el `arregloInicial` desde la primera hasta la última posición de este, pero que paralelamente va recorriendo el `arregloFinal` desde la última posición marcada con la variable `j` hasta la posición 1, pues esta se va reduciendo dentro del ciclo, de esta forma se va pasando el elemento que hay en cada posición del `arregloInicial` a cada posición del `arregloFinal`.

```
21   j = n
22   Para i = 1 Hasta n Incremento 1
23     arregloFinal[ j ] = arregloInicial[ i ]
24     j = j - 1
25   FinPara
```

Luego, con otro ciclo `Para` se imprimen todos los elementos del `arregloFinal`.

```
27   Para i = 1 Hasta n Incremento 1
28     imprimir( arregloFinal[ i ], " " )
29   FinPara
```

Se asume que cada impresión se realiza en la misma línea. Si cada vez que se ejecuta la instrucción `imprimir` el mensaje se ubicase en una línea nueva, sería necesario usar una nueva variable tipo `Cadena` para almacenar todos los datos y luego invocar una sola vez la instrucción `imprimir` (Ver el siguiente ejemplo).

.:Ejemplo 6.4. *Diseñe un algoritmo que permita almacenar en un vector una cantidad n de números pares y en otro vector una cantidad m de números impares ingresados por el usuario. Construya el algoritmo de tal manera que almacene en un tercer vector todos los números contenidos en los dos vectores iniciales y muestre al final el tercer vector.*

Análisis del problema:

- **Resultados esperados:** un vector con los datos almacenados en los dos arreglos iniciales.
- **Datos disponibles:** el tamaño de los dos vectores iniciales y los números que se van a almacenar en ellos.
- **Proceso:** solicitar al usuario el tamaño de cada uno de los vectores, luego, leer todos los números a almacenar en los dos vectores iniciales, posteriormente se toman todos los datos del primer vector (uno a uno) y se almacenan en el tercer vector, luego se toman los datos del segundo vector y se ubican también en el tercer vector y, finalmente se imprime el tercer vector.
- **Variables requeridas:**
  - `arregloPares`: vector que contiene todos los números pares que el usuario ingresó.
  - `arregloImpares`: vector que contiene todos los números impares que el usuario ingresó.
  - `arregloTotal`: vector que contiene todos los números que el usuario ingresó (pares e impares).
  - `tamanioPares`: cantidad de elementos que contiene el vector de pares.
  - `tamanioImpares`: cantidad de elementos que contiene el vector de impares.
  - `tamanioTotal`:
  - `i`: variable de control para los vectores.
  - `j`: variable de control para los vectores
  - `salida`: variable que almacena los datos del tercer vector para su posterior impresión.

De acuerdo al análisis planteado, se propone el Algoritmo 6.4.

Algoritmo 6.4: ConcatenaArreglos

```
1 Algoritmo ConcatenaArreglos
2   Entero arregloPares[ ], arregloImpares[ ], arregloTotal[ ]
3   Entero tamanioPares, tamanioImpares, tamanioTotal, i, j
4   Cadena salida
5
6   imprimir( "Ingrese la cantidad de pares: " )
7   leer( tamanioPares )
```

```
8
9   imprimir( "Ingrese la cantidad de impares: " )
10  leer( tamanioImpares )
11
12  tamanioTotal = tamanioPares + tamanioImpares
13
14  arregloPares   = dimensionar( tamanioPares   )
15  arregloImpares = dimensionar( tamanioImpares )
16  arregloTotal   = dimensionar( tamanioTotal   )
17
18  Para i = 1 Hasta tamanioPares Incremento 1
19    Haga
20      imprimir( "Ingrese el par " , i, ": " )
21      leer( arregloPares[ i ] )
22
23      Si( arregloPares[ i ] % 2 != 0 ) Entonces
24        imprimir( "Este número no es par" )
25      FinSi
26    MientrasQue( arregloPares[ i ] % 2 != 0 )
27  FinPara
28
29  Para i = 1 Hasta tamanioImpares Incremento 1
30    Haga
31      imprimir( "Ingrese el impar " , i," : " )
32      leer( arregloImpares[ i ] )
33
34      Si( arregloImpares[ i ] % 2 != 1 ) Entonces
35        imprimir( "Este número no es impar" )
36      FinSi
37    MientrasQue( arregloImpares[ i ] % 2 != 1 )
38  FinPara
39
40  j = 1
41  Para i = 1 Hasta tamanioPares Incremento 1
42    arregloTotal[ j ] = arregloPares [ i ]
43    j = j + 1
44  FinPara
45
46  Para i = 1 Hasta tamanioImpares Incremento 1
47    arregloTotal[ j ] = arregloImpares [ i ]
48    j = j + 1
49  FinPara
50
51  salida = ""
52  Para i = 1 Hasta arregloTotal Incremento 1
53    salida = salida + arregloTotal [ i ] + " "
54  FinPara
55  imprimir( "Arreglo concatenado: ", salida )
56 FinAlgoritmo
```

Al ejecutar el algoritmo:

```
Ingrese la cantidad de pares: 3
Ingrese la cantidad de impares: 2
Ingrese el par 1: 8
Ingrese el par 2: 9
Este número no es par
Ingrese el par 2: 34
Ingrese el par 3: 86
Ingrese el impar 1: 73
Ingrese el impar 2: 8
Este número no es impar
Ingrese el impar 2: 1
Arreglo concatenado 8 34 86 73 1
```

Explicación del algoritmo:

En las primeras líneas (2 a la 4) se declaran todas las variables que se requieren. A continuación, se solicita la cantidad de números pares e impares y con esta información se determina la cantidad total de números a ingresar.

```
6   imprimir( "Ingrese la cantidad de pares: " )
7   leer( tamanioPares )
8
9   imprimir( "Ingrese la cantidad de impares: " )
10  leer( tamanioImpares )
11
12  tamanioTotal = tamanioPares + tamanioImpares
```

Una vez hecho esto, se procede a `dimensionar` cada uno de los arreglos del tamaño ya definido.

```
14  arregloPares   = dimensionar( tamanioPares   )
15  arregloImpares = dimensionar( tamanioImpares )
16  arregloTotal   = dimensionar( tamanioTotal   )
```

Lo que sigue en el algoritmo, es realizar la lectura de los números pares:

```
18  Para i = 1 Hasta tamanioPares Incremento 1
19    Haga
20      imprimir( "Ingrese el par " , i, ": " )
21      leer( arregloPares[ i ] )
22
23      Si( arregloPares[ i ] % 2 != 0 ) Entonces
24        imprimir( "Este número no es par" )
25      FinSi
26    MientrasQue( arregloPares[ i ] % 2 != 0 )
27  FinPara
```

Observe que dentro de este ciclo `Para` se realiza la respectiva validación con el propósito de evitar que el usuario ingrese un número impar. Para realizar la validación se hace uso de un `Haga-MientrasQue` y un `Si` que informa al usuario, de ser necesario, el intento de ingresar un dato no válido. Algo similar se realiza en las líneas 29 a la 38 pero, esta vez con el ingreso de los números impares.

Después, se inicializa la variable de control `j` con el valor 1, y se usará para "navegar" dentro del vector `arregloTotal`. Posteriormente se recorre con un ciclo `Para` el vector `arregloPares` para copiar todos sus datos al vector `arregloTotal`. De forma similar se recorre con otro ciclo el vector `arregloImpares` y se copian los elementos en el vector `arregloTotal`. Note que la variable `j` no fue inicializada nuevamente en el momento de copiar los elementos del segundo arreglo, así que continúa en la siguiente posición en donde se ubicó el último número par.

```
40  j = 1
41  Para i = 1 Hasta tamanioPares Incremento 1
42    arregloTotal[ j ] = arregloPares [ i ]
43    j = j + 1
44  FinPara
45
46  Para i = 1 Hasta tamanioImpares Incremento 1
47    arregloTotal[ j ] = arregloImpares [ i ]
48    j = j + 1
49  FinPara
```

Para finalizar, se inicializa una variable tipo `Cadena`, denominada `salida` con una cadena vacía (`""`), luego se recorre todo el `arregloTotal` y en cada iteración se concatena a la variable `salida` el elemento del `arregloTotal` en la posición actual, más un espacio en blanco. Una vez se ha recorrido todo este arreglo, se procede a imprimir la variable `salida`.

```
51  salida = ""
52  Para i = 1 Hasta arregloTotal Incremento 1
53    salida = salida + arregloTotal [ i ] + " "
54  FinPara
55  imprimir( "Arreglo concatenado: ", salida )
```

.:Ejemplo 6.5. *Diseñe un algoritmo que reciba como entrada un número entero entre cero y noventa y nueve (0 y 99) y que, como salida muestre ese mismo número, pero expresado esta vez en palabras.*

Análisis del problema:

- **Resultados esperados:** el número ingresado (0 a 99) pero expresado en palabras.
- **Datos disponibles:** el número que se desea convertir a palabras.
- **Proceso:** se le solicita al usuario el número a convertir a palabras, luego se realiza un proceso para su conversión de este número a su equivalente en palabras realizando la búsqueda de las palabras adecuadas en varios vectores, y finalmente se imprime el mensaje.

  El proceso de conversión, consiste en determinar si el número es menor a 11 para buscar la palabra que lo identifica en un primer vector; sino, se verifica si el número está entre 11 y 19, de ser así, se busca su respectiva palabra en otro vector; y si es mayor a 19, se determinan las unidades y las decenas de este número y se buscan sus palabras equivalentes en los vectores respectivos.

  Para la resolución de este ejercicio se utilizan tres vectores que poseen las palabras, correspondientes a ciertos números claves, ya almacenadas desde su declaración. en este sentido, el algoritmo no le solicita al usuario datos que vayan a ser almacenados en algún vector, ni mucho menos se requiere pedir el tamaño del mismo, pues como se acaba de mencionar, los vectores se llenan con los datos (las palabras) desde el momento en que se declaran.
- **Variables requeridas:**
  - `numero`: que se desea convertir a palabras.
  - `numeroPalabras`: que almacena el número en palabras.
  - `arregloUnidades`: vector que contiene los nombres de los primeros once números (iniciando desde el cero).
  - `arregloEspecial`: vector que contiene los de los siguientes ocho números comenzando en once.
  - `arregloDecenas`: vector que contiene los nombres de los números del veinte al noventa, pero de diez en diez.
  - `decenas`: cantidad de decenas que tiene el número ingresado por usuario.
  - `unidades`: cantidad de unidades que tiene el número ingresado.

De acuerdo al análisis planteado, se propone el Algoritmo 6.5.

Algoritmo 6.5: ConversionNumero

```
1  Algoritmo ConversionNumero
2
3    Entero numero, decenas, unidades
4    Cadena numeroPalabras
5    Cadena arregloUnidades[ ] = { "cero",  "uno",    "dos",
6                                  "tres",  "cuatro", "cinco",
7                                  "seis",  "siete",  "ocho",
8                                  "nueve", "diez" }
9
10   Cadena  arregloEspecial[ ] = {"once",       "doce",
11                                 "trece",      "catorce",
12                                 "quince",     "dieciseis",
13                                 "diecisiete", "dieciocho",
14                                 "diecinueve" }
15
16   Cadena arregloDecenas [ ] = { "veinte",   "treinta",
17                                 "cuarenta", "cincuenta",
18                                 "sesenta",  "setenta",
19                                 "ochenta",  "noventa" }
20
21   Haga
22     imprimir( "Ingrese el número a convertir: " )
23     leer( numero )
24
25     Si ( numero < 0 O numero >  99 ) Entonces
26       imprimir( "Este número no es válido" )
27     FinSi
28   MientrasQue( numero < 0 O numero > 99)
29
30   Si( numero <= 10 ) Entonces
31     numeroPalabras =  arregloUnidades[ numero + 1 ]
32   SiNo
33     Si( numero <= 19 ) Entonces
34       numeroPalabras = arregloEspecial[ numero - 10 ]
35     SiNo
36       unidades = numero  % 10
37       decenas  = numero / 10
38
39       Si( unidades == 0 ) Entonces
40          numeroPalabras = arregloDecenas[ decenas - 1]
41       SiNo
42          numeroPalabras = arregloDecenas[ decenas - 1 ] +
43                           "  y " +
44                           arregloUnidades[ unidades + 1 ]
45       FinSi
46     FinSi
47   FinSi
48
```

```
49   imprimir( "El número es: ", numeroPalabras )
50 FinAlgoritmo
```

Al ejecutar el algoritmo:

```
Ingrese el número a convertir: -4
Este número no es válido
Ingrese el número a convertir: 200
Este número no es válido
Ingrese el número a convertir: 87
El número es: ochenta y siete
```

Explicación del algoritmo:

Luego de declarar todas las variables necesarias, incluso de inicializar los arreglos con las palabras correspondientes a los números (líneas 3 a la 19); se procede a solicitar al usuario un número entero 0 y 99, haciendo la respectiva validación. Observe que esta validación no permitira ingresar un número que esté por debajo de cero o por encima de noveinta y nueve.

```
20   Haga
21     imprimir( "Ingrese el número a convertir: " )
22     leer( numero )
23
24     Si ( numero < 0 O numero >  99 ) Entonces
25       imprimir( "Este número no es válido" )
26     FinSi
27   MientrasQue( numero < 0 O numero > 99)
```

A continuación, se determina si el número es menor o igual a 10 con la primera instrucción `Si`. Si esto ocurre, se ubica la palabra correspondiente al número en el vector de unidades. En el caso de que el número sea menor o igual a 19, el algoritmo hace la búsqueda en el vector de especiales. Si el número es mayor a 19, se determina cuántas decenas y cuántas unidades tiene, para así poder mostrar el respectivo nombre. En el caso de que no tenga unidades (`unidades == 0`), solo se muestra el nombre de las decenas, si no, se muestran tanto decenas como unidades.

```
29   Si( numero <= 10 ) Entonces
30     numeroPalabras =  arregloUnidades[ numero + 1 ]
31   SiNo
32     Si( numero <= 19 ) Entonces
33       numeroPalabras = arregloEspecial[ numero - 10 ]
34     SiNo
35       unidades = numero % 10
36       decenas  = numero / 10
37
```

```
38        Si( unidades == 0 ) Entonces
39           numeroPalabras = arregloDecenas[ decenas - 1]
40        SiNo
41           numeroPalabras = arregloDecenas[ decenas - 1 ] +
42                            "  y " +
43                            arregloUnidades[ unidades + 1 ]
44        FinSi
45     FinSi
46  FinSi
```

Note como es necesario sumar o restar ciertas cantidades a las índices para poder obtener el valor correcto. Por ejemplo, si el número fuera 8, sería necesario sumar un uno para llegar a la novena posición del vector de unidades en donde está la palabra "ocho"; esto suecede, porque la palabra "cero" ocupa la primera posición. Algo similar sucede en los otros dos vectores, por ejemplo, si el número a convertir es 17, el algoritmo ingresa a la parte del `SiNo` del primer `Si`, allí encuentra el segundo `Si` que se evaluaría como `Verdadero`, por lo cual el algoritmo almacenaría en la variable `numeroPalabras` lo que hay en el vector `arregloEspecial` en la posición `numero - 10`, lo que coresponde a 17 - 10 = 7 y, al buscar en la posición 7 de este vector, se encuentra la palabra `diecisiete`.

Ahora, si el número ingresado fuera 60, el algoritmo llegaría a la instrucción `Si(numero<=19)` que se evaluaría como `Falso`, por lo cual, el algoritmo determinaría las `unidades` = 0 y las `decenas` = 6, por tanto determinaría la posición 5 (`decenas - 1`) en el vector de `arregloDecenas`, la cual corresponde a `sesenta`.

Finalmente, se imprime el número pero en palabras, con lo que termina el algoritmo.

```
49    imprimir( "El número es: ", numeroPalabras )
```

Aclaración:



En los ejercicios que aparecen a continuación, se utilizarán funciones y procedimientos, ya que estas estructuras facilitarán escribir de forma más modular y comprensible los procesos a realizar; Además, el uso de funciones y procedimientos facilitará la realización de pruebas y la corrección de errores, elemento importantísimo a tener en cuenta en la corrección de los algoritmos.

.:Ejemplo 6.6. *Diseñe un algoritmo que almacene un grupo de* `n` *números enteros en un arreglo y luego determine mediante funciones y procedimientos cuántos de esos números ingresados son pares y cuántos son impares.*

Análisis del problema:

- **Resultados esperados:** la cantidad de números pares y de impares que están almacenados en el arreglo de números.
- **Datos disponibles:** el tamaño que se le va a dar al arreglo y los números enteros que van a ser almacenados en el mismo.
- **Proceso:** solicitar la cantidad de elementos (`n`), después leer todos los elementos para el arreglo, posteriormente recorrer el arreglo y determinar si el elemento que se encuentra en cada posición es par; y si es así, contarlo como par; lo mismo que si es impar. Al final, el algoritmo mostrará el valor con la cuenta final de los números pares y con los impares.
- **Variables requeridas:**
  - En el algoritmo principal
    - `n`: cantidad de elementos del arrreglo a procesar.
    - `numeros`: vector que contiene los elementos que el usuario ingresa.
    - `cantidadPares`: cantidad de números pares que contiene el vector original.
    - `cantidadImpares`: cantidad de números impares que contiene el vector original.
  - En la función `leerDatos`
    - Parámetros:
      - `n`: cantidad de elementos a leer
    - Variables locales:
      - `i`: variable de control del ciclo
      - `arreglo`: vector que almacena los datos que el usuario ingresa.
  - En la función `obtenerCantidadPares`
    - Parámetros:
      - `arreglo`: vector que almacena contiene los datos que el usuario ingresa.

        - ◇ `n`: tamaño del vector.
    - En la función `obtenerCantidadImpares`
        - Parámetros:
            - ◇ `arreglo`: vector que almacena contiene los datos que el usuario ingresa.
            - ◇ `n`: tamaño del vector.

- En la función `contar`
    - Parámetros:
        - `arreglo`: vector que contiene los datos a ser analizados.
        - `resto`: parámetro que permite especificar si lo que se desea contar son pares (`resto=0`) o impares (`resto=1`). `resto` representa el resultado del resto de la división entera con 2 (`arreglo[ i ] % 2 == resto`).
        - `n`: tamaño del vector.
    - Variables locales:
        - `i`: variable de control del ciclo
        - `cantidad`: variable que permite contar la cantidad de pares o impares (dependiendo del valor `resto`) que hay en el arreglo

- En el procedimiento `mostrarResultados`
    - Parámetros:
        - `pares`: cantidad de números pares que se encontraron en el vector.
        - `impares`: cantidad de números impares que se encontraron en el vector.

De acuerdo al análisis planteado, se propone el Algoritmo 6.6.

Algoritmo 6.6: numero

```
1 Algoritmo numero
2   Entero n, numeros [ ], cantidadPares,
3          cantidadImpares
4
5   imprimir( "Ingrese el tamaño del Arreglo: ")
6   leer( n )
7   numeros = leerDatos( n )
8
```

```
9   cantidadPares   = obtenerCantidadPares( numeros, n )
10  cantidadImpares = obtenerCantidadImpares( numeros, n )
11
12  mostrarResultados( cantidadPares, cantidadImpares )
13 FinAlgoritmo
14
15 Funcion Entero[ ] leerDatos( Entero n )
16  Entero i, arreglo[ n ]
17
18  Para i = 1 Hasta n Incremento 1
19    imprimir( "Ingrese el número " , i, ": " )
20    leer( arreglo[ i ] )
21  FinPara
22  Retornar arreglo
23 FinFuncion
24
25
26 Funcion Entero obtenerCantidadPares( Entero arreglo[ ],
27                                      Entero n )
28  Retornar contar ( arreglo, 0, n )
29 FinFuncion
30
31 Funcion Entero obtenerCantidadImpares( Entero arreglo[ ],
32                                        Entero n )
33  Retornar contar ( arreglo, 1, n )
34 FinFuncion
35
36 Funcion Entero contar( Entero arreglo[ ],
37                        Entero resto,
38                        Entero n )
39  Entero i, cantidad
40
41  cantidad = 0
42  Para i = 1 Hasta n Incremento 1
43    Si( arreglo[ i ] % 2 == resto ) Entonces
44      cantidad = cantidad + 1
45    FinSi
46  FinPara
47
48  Retornar cantidad
49 FinFuncion
50
51 Procedimiento mostrarResultados( Entero pares,
52                                  Entero impares )
53   imprimir( "Cantidad de pares  : ", pares )
54   imprimir( "Cantidad de impares: ", impares )
55 FinProcedimiento
```

Al ejecutar el algoritmo:

```
Ingrese el tamaño del Arreglo: 5
Ingrese el número 1: 4
Ingrese el número 2: 71
Ingrese el número 3: 22
Ingrese el número 4: 15
Ingrese el número 5: 78
Cantidad de pares  : 3
Cantidad de impares: 2
```

Explicación del algoritmo:

Este algoritmo se construyó de la siguiente forma:

Se creó un algoritmo principal en el que se declararon las variables, se solicitó el tamaño del vector y se hizo el llamado a: `leerDatos()`, `obtenerCantidadPares()`, `obtenerCantidadImpares()`, (las dos últimas invocan a la función `contar()`) y un procedimiento denominado `mostrarResultados()`. Luego de solicitar el tamaño del vector, este dato se envía como parámetro a la función `leerDatos(n)`, la cual se usa para crear un nuevo vector, llenarlo con la información que el usuario ingresa y retornarlo, el valor de retorno es almacenado en la variable `numero`. A continuación, se explica una a una cada función utilizada y el procedimiento que muestra los resultados.

```
2   Entero n, numeros [ ], cantidadPares,
3          cantidadImpares
4
5   imprimir( "Ingrese el tamaño del Arreglo: ")
6   leer( n )
7   numeros = leerDatos( n )
8
9   cantidadPares   = obtenerCantidadPares( numeros, n )
10  cantidadImpares = obtenerCantidadImpares( numeros, n )
11
12  mostrarResultados( cantidadPares, cantidadImpares )
```

La función `leerDatos()`, se utiliza para capturar los datos ingresados y almacenarlos en cada una de las posiciones del vector; por esta razón, en ella se usa un ciclo `Para`, que inicia la variable `i` en 1 y va hasta el tamaño del vector (`n`). Dentro del ciclo, con las instrucciones de `imprimir` y `leer` se solicitan los datos y se almacenan en cada celda del vector. Esta función recibe como parámetro el tamaño del vector para inicializar el vector interno, llenarlo y retornarlo al final de la función.

```
15 Funcion Entero[ ] leerDatos( Entero n )
16   Entero i, arreglo[ n ]
17
18   Para i = 1 Hasta n Incremento 1
19     imprimir( "Ingrese el número " , i, ": " )
20     leer( arreglo[ i ] )
21   FinPara
22   Retornar arreglo
23 FinFuncion
```

La función que cuenta los pares `obtenerCantidadPares()` y la función que cuenta los impares `obtenerCantidadImpares()` son en realidad casos especiales de una función más general llamada `contar()`. La única diferencia que existe entre ambas funciones es el valor del resto de la división entera con dos. Para el caso de los pares el resto es cero, mientras que para el caso de los impares el resto es 1.

La función, `obtenerCantidadPares( numeros, n )` tiene como parámetros la referencia al vector de números y el tamaño del mismo. Esta función hace el llamado a la función `contar()` pasándole la referencia al vector de números, el cero y el tamaño del vector. El cero será el parámetro que permite contar cuantos números pares hay en el vector.

Igual sucede con la función `obtenerCantidadImpares(numeros, n)` que tiene los mismos parámetros, y que llama a la función `contar()` pero pasando esta vez un uno en lugar de un cero.

```
26 Funcion Entero obtenerCantidadPares( Entero arreglo[ ],
27                                      Entero n )
28   Retornar contar ( arreglo, 0, n )
29 FinFuncion
```

```
31 Funcion Entero obtenerCantidadImpares( Entero arreglo[ ],
32                                        Entero n )
33   Retornar contar ( arreglo, 1, n )
34 FinFuncion
```

La función `contar()` recibe como parámetro el vector, un valor `resto` el cual se utiliza para determinar si el número almacenado es par(0) o impar(1) y, el tamaño del vector. La función inicializa un contador llamado `cantidad` en cero, luego recorre todo el arreglo verificando si el número es par o impar, de ser `Verdadera` la condición del `Si`, se cuenta uno más. Al final, simplemente se retorna el valor de la variable `cantidad`.

```
36 Funcion Entero contar( Entero arreglo[ ],
37                        Entero resto,
38                        Entero n )
39   Entero i, cantidad
40
41   cantidad = 0
42   Para i = 1 Hasta n Incremento 1
43     Si( arreglo[ i ] % 2 == resto ) Entonces
44       cantidad = cantidad + 1
45     FinSi
46   FinPara
47
48   Retornar cantidad
49 FinFuncion
```

Por último, el procedimiento `mostrarResultados()`, recibe como parámetros la cantidad de pares y de impares encontrados en el arreglo y los muestra mediante las instrucciones `imprimir` que están dentro de la función.

.:Ejemplo 6.7. *Escribir un algoritmo que almacene en un arreglo unidimensional las notas finales obtenidas por un grupo de 20 estudiantes en una materia cualquiera y, a través de funciones, determine el promedio del grupo, la nota más alta y más baja obtenida en la materia.*

Aclaración:



Este problema consiste en hacer varios recorridos por el vector cuando este ya se encuentre lleno con las notas de los estudiantes para determinar el promedio y las notas más alta y baja del grupo. Por esta razón, es recomendable resolver el ejercicio por medio de funciones, ya que cada una de ellas resolverá una tarea en particular. El ejercicio se encuentra planteado para un grupo de 20 estudiantes, pero perfectamente se puede adaptar a un grupo de n estudiantes tal como se ha hecho en los ejercicios anteriores.

Análisis del problema:

- **Resultados esperados:** mostrar la nota promedio del grupo de estudiantes, la mayor y menor nota obtenidas entre las 20 que conforman el grupo.
- **Datos disponibles:** las 20 notas de los estudiantes.

- **Proceso:** lo primero que se hace en este algoritmo es solicitar las 20 notas al usuario y almacenarlas en el vector. Posteriormente, para encontrar la nota promedio del grupo, se recorre el vector, se suman todas las notas y se divide esta suma entre los 20 estudiantes; esto implica utilizar un ciclo que facilite el recorrido por las distintas posiciones del vector y el uso de una variable de tipo acumulador en la que se van sumando las notas. También se debe recorrer el vector con el fin de identificar la nota más alta y la más baja que estén almacenadas en él; para determinar la nota más alta, se compara la nota almacenada en la primera posición del arreglo con la segunda para saber cuál es mayor, luego la segunda con la tercera y así sucesivamente hasta llegar a la última nota, lo que implica el uso de una estructura de decisión. De la misma forma se hace si se desea conocer la nota más baja, en este caso la comparación con el Si entre las dos notas se lleva a cabo para conocer la menor nota. Tenga en cuenta que, como el ejercicio se está desarrollando para un grupo de 20 estudiantes, se usa una constante que contenga este valor durante todo el algoritmo.
- **Variables requeridas:**
  - En el algoritmo principal
    - `MAX`: constante que almacena la cantidad de estudiantes, para el ejemplo son 20.
    - `arrregloNotas`: vector que contiene las notas de los estudiantes.
    - `notaMayor`: almacena la menor nota entre todas las ingresadas.
    - `notaMenor`: almacena la mayor nota de los estudiantes.
    - `notaPromedio`: almacena la nota promedio del grupo de estudiantes.
  - En la función `leerDatos`
    - Parámetros:
      - `tamanio`: cantidad de elementos a leer
    - Variables locales:
      - `i`: variable de control del ciclo
      - `arreglo`: vector que almacena las notas que el usuario ingresa.
  - En la función `calcularPromedio`
    - Parámetros:

        - ◇ `arregloNotas`: vector que almacena las notas de los estudiantes.
        - ◇ `n`: tamaño del vector.
      - Variables locales:
        - ◇ `i`: variable de control del ciclo.
        - ◇ `suma`: variable para almacenar la suma de todas las notas.
        - ◇ `promedio`: variable para almacenar el promedio de las notas.
    - En la función `obtenerMayor`
      - Parámetros:
        - ◇ `arregloNotas`: vector que almacena las notas de los estudiantes.
        - ◇ `n`: tamaño del vector.
      - Variables locales:
        - ◇ `i`: variable de control del ciclo.
        - ◇ `mayor`: variable para almacenar el mayor valor del arreglo de notas.
    - En la función `obtenerMenor`
      - Parámetros:
        - ◇ `arregloNotas`: vector que almacena las notas de los estudiantes.
        - ◇ `n`: tamaño del vector.
      - Variables locales:
        - ◇ `i`: variable de control del ciclo.
        - ◇ `menor`: variable para almacenar el menor valor del vector de notas.

- En el procedimiento `mostrarResultados`
  - Parámetros:
    - `promedio`: promedio de las notas de los estudiantes.
    - `mayor`: mayor nota de los estudiantes.
    - `menor`: menor nota de los estudiantes.

De acuerdo al análisis planteado, se propone el Algoritmo 6.7.

Algoritmo 6.7: NotasMateria

```
1  Algoritmo NotasMateria
2    Constante Entero MAX = 20
3    Real arregloNotas [ ], notaPromedio, notaMayor, notaMenor
4
5    arregloNotas = leerDatos( MAX )
6    notaPromedio = calcularPromedio( arregloNotas, MAX )
7    notaMayor    = obtenerMayor( arregloNotas, MAX )
8    notaMenor    = obtenerMenor( arregloNotas, MAX )
9
10   mostrarResultados( notaPromedio, notaMayor, notaMenor )
11 FinAlgoritmo
12
13 Funcion Real[ ] leerDatos( Entero tamanio )
14   Real   arreglo[ tamanio ]
15   Entero i
16
17   Para i = 1 Hasta tamanio Incremento 1
18     imprimir( "Ingrese el nota del estudiante " , i, ": " )
19     leer( arreglo[ i ] )
20   FinPara
21   Retornar arreglo
22 FinFuncion
23
24 Funcion Real calcularPromedio( Real   arregloNotas[ ],
25                                Entero n )
26   Entero i
27   Real suma = 0
28   Real promedio
29
30   Para i = 1 Hasta n Incremento 1
31     suma = suma + arregloNotas[ i ]
32   FinPara
33
34   promedio = suma / n
35   Retornar promedio
36 FinFuncion
37
38 Funcion Real obtenerMayor( Real arregloNotas[ ], Entero n )
39   Entero i
40   Real mayor = arregloNotas[ 1 ]
41
42   Para i = 2 Hasta n Incremento 1
43     Si arregloNotas[ i ] > mayor Entonces
44       mayor = arregloNotas[ i ]
45     FinSi
46   FinPara
47   Retornar mayor
48 FinFuncion
```

```
49
50 Funcion Real obtenerMenor( Real arregloNotas[ ], Entero n )
51   Entero i
52   Real menor = arregloNotas[ 1 ]
53
54   Para i = 2 Hasta n Incremento 1
55     Si arregloNotas[ i ] < menor Entonces
56       menor = arregloNotas[ i ]
57     FinSi
58   FinPara
59   Retornar menor
60 FinFuncion
61
62 Procedimiento mostrarResultados( Entero promedio,
63                                  Entero mayor,
64                                  Entero menor )
65   imprimir( "Promedio del grupo: ", promedio )
66   imprimir( "Nota más alta obtenida: ", mayor )
67   imprimir( "Nota más baja obtenida: ", menor )
68 FinProcedimiento
```

Explicación del algoritmo:

Este algoritmo fue desarrollado mediante funciones y procedimientos que son llamados desde el algoritmo principal. En este algoritmo principal, primero se declaran las variables y posteriormente se hace el llamado a las distintas funciones en el siguiente orden:

- La primera función, `leerDatos()`, permite obtener las notas del usuario y almacenarlas en el arreglo de notas. Esta función tiene un parámetro que permite asignarle el tamaño al arreglo y funciona de la misma forma en que se explicó en el ejercicio anterior.
- En seguida se hace el llamado a La función `calcularPromedio()`, la cual recibe como parámetros el arreglo de notas y su tamaño. El arreglo se recorre desde la primera hasta la última posición por medio de un ciclo `Para` y, mediante la variable `suma` va acumulando la suma de las notas, es por eso que esta variable se inicializa en cero antes de ingresar al ciclo `Para`. Al terminar el recorrido y debajo del `FinPara`, se calcula el promedio de notas del grupo dividiendo la suma entre el número de estudiantes, o sea, 20, representado por la variable `n`.

- A continuación se llama a la función `obtenerMayor()` que recibe como parámetros el arreglo de notas y su tamaño. Esta función almacena la primera posición del arreglo en la variable `mayor` y luego recorre el arreglo desde la posición 2 hasta la última posición (20). Este recorrido se hace mediante el ciclo `Para`. Dentro del ciclo, se va comparando la posición actual del arreglo con la variable `mayor`. Si lo que posee la posición actual del arreglo es mayor que la variable `mayor`, esta posición se almacena en la variable `mayor` y el algoritmo pasa a la próxima iteración. Si la posición actual del arreglo no contiene una nota mayor que la que está almacenada en la variable `mayor`, el ciclo pasa a la siguiente iteración, donde se ubica en la siguiente posición del vector y vuelve a comparar. Al terminar el recorrido del vector, esta función obtendrá la mayor nota del arreglo y la retornará al algoritmo principal.
- Del mismo modo trabaja la función `obtenerMenor()`, que es la siguiente instrucción en el algoritmo principal, pero dentro de esta función se cambia la comparación que se hace al interior del ciclo `Para`, con el fin de saber si lo que posee la posición actual del vector es menor que lo que tiene almacenado la variable `menor`. Esta función retornará la menor nota almacenada en el vector.
- Por último, el procedimiento `mostrarResultados()`, permite mostrar la nota promedio del grupo, la mayor nota y la menor nota almacenadas en el vector, siendo estos los parámetros que recibió la función y que muestra mediante las instrucciones `imprimir` que están en su interior.

.:Ejemplo 6.8. *Diseñe un algoritmo que almacene en vectores el nombre y el género de un grupo de `n` personas y, que a través de funciones permita buscar la posición del vector en que se encuentra una persona cualquiera, buscada por su nombre y el género que esta persona tiene. Con objeto de almacenar los géneros de manera óptima, utilice una 'M' para Masculino y una 'F' para femenino.*

Análisis del problema:

- **Resultados esperados:** mostrar la posición en la que está en el vector de nombres y su género, una persona consultada por su nombre.

- **Datos disponibles:** el tamaño de los vectores, los nombres y géneros de las `n` personas, así como el nombre de la persona que se va a buscar, una vez ya estén cargados los vectores con los datos.
- **Proceso:** luego de almacenar nombres y géneros, el algoritmo debe buscar la posición que ocupa el nombre de una persona en el vector de nombres, esto se lleva a cabo por medio de una búsqueda por todo el vector de nombres desde la primera hasta la última posición utilizando un ciclo y una estructura de decisión. Si el nombre de esta persona se encuentra almacenado, el algoritmo debe mostrar la posición que ocupa ese nombre en el vector y su género correspondiente, que está almacenado en el vector de géneros. Para este ejercicio, es necesario suponer que cada nombre se almacena una sola vez. Si el nombre buscado está almacenado en varias posiciones del arreglo, el algoritmo reporta la posición del último de ellos.
- **Variables requeridas:**
  - En el algoritmo principal
    - `tamanio`: cantidad de personas de las que se conoce la información.
    - `posicion`: posición que ocupa la persona buscada en el vector de nombres.
    - `nombreBuscar`: nombre de la persona que se desea buscar.
    - `arregloNombres`: vector con los nombres de las personas.
    - `arregloGeneros`: vector con los géneros de las personas.
  - En la función `leerNombres`
    - Parámetros:
      - `tamanio`: cantidad de elementos a leer.
    - Variables locales:
      - `i`: variable de control del ciclo.
      - `arregloNom`: vector que almacena los nombres ingresados.
  - En la función `leerGeneros`
    - Parámetros:
      - `tamanio`: cantidad de elementos a leer.
    - Variables locales:
      - `i`: variable de control del ciclo.
      - `arregloG`: vector que almacena los géneros de las personas ingresadas.

- En la función `obtenerPosicion`
  - Parámetros:
    - ◇ `arregloNom`: arreglo con los nombre de las personas.
    - ◇ `nombre`: nombre de la persona que se desea buscar.
    - ◇ `tamanio`: cantidad de elementos del vector.
  - Variables locales:
    - ◇ `i`: variable de control del ciclo.
    - ◇ `pos`: variable que almacena la posición del vector en donde se encuentra el nombre buscado o, -1 de no estar presente.
- En el procedimiento `mostrarResultados`
  - Parámetros:
    - ◇ `pos`: posición en donde se encuentra el nombre, o en su defecto, -1 de no estar presente.
    - ◇ `arregloN`: vector con los nombres de las personas.
    - ◇ `arregloG`: vector con los genéros de las personas.

De acuerdo al análisis planteado, se propone el Algoritmo 6.8.

Algoritmo 6.8: ArregloPersonas

```
1  Algoritmo ArregloPersonas
2    Entero   tamanio, posicion
3    Cadena   nombreBuscar , arregloNombres [ ]
4    Caracter arregloGeneros [ ]
5
6    imprimir( "Ingrese la cantidad de personas " )
7    leer( tamanio )
8
9    arregloNombres = leerNombres( tamanio )
10   arregloGeneros = leerGeneros( tamanio )
11
12   imprimir( "Ingrese el nombre de la persona a buscar: " )
13   leer( nombreBuscar )
14
15   posicion = obtenerPosicion(arregloNombres, nombreBuscar,
         tamanio )
16
17   mostrarResultados( posicion, arregloNombres,
         arregloGeneros )
18 FinAlgoritmo
19
```

```
20 Funcion Cadena[ ] leerNombres( Entero tamanio )
21   Entero i
22   Cadena arregloNom[ tamanio ]
23
24   Para i = 1 Hasta tamanio Incremento 1
25     imprimir( "Ingrese el nombre de la persona " , i, ": ")
26     leer( arregloNom[ i ] )
27   FinPara
28   Retornar arregloNom
29 FinFuncion
30
31 Funcion Caracter[ ] leerGeneros( Entero tamanio )
32   Entero i
33   Caracter arregloG[ tamanio ]
34
35   Para i = 1 Hasta tamanio Incremento 1
36     Haga
37       imprimir( "Género de la persona " , i," (M/F): " )
38       leer( arregloG[ i ] )
39
40       Si( arregloG[ i ] != 'F' Y
41           arregloG[ i ] != 'M' ) Entonces
42         imprimir( "Ha ingresado un género equivocado" )
43       FinSi
44     MientrasQue(arregloG[ i ] != 'F' Y arregloG[ i ]!= 'M')
45   FinPara
46   Retornar arregloG
47 FinFuncion
48
49 Funcion Entero obtenerPosicion( Cadena arregloNom [ ],
50                                 Cadena nombre,
51                                 Entero tamanio )
52   Entero i, pos
53
54   pos = -1
55   Para i = 1 Hasta tamanio Incremento 1
56     Si arregloNom[ i ] == nombre Entonces
57       pos = i
58     FinSi
59   FinPara
60   Retornar pos
61 FinFuncion
62
63 Procedimiento mostrarResultados( Entero   pos,
64                                  Cadena   arregloN [ ],
65                                  Caracter arregloG [ ] )
66   Si( pos == -1 ) Entonces
67     imprimir( "El nombre no está en la lista ")
68   SiNo
```

```
69      imprimir( arregloN[ pos ] )
70      imprimir( "está en la posición ", pos )
71      imprimir( "y su genéro es ", arregloG[ pos ] )
72    FinSi
73  FinProcedimiento
```

Explicación del algoritmo:

En el algoritmo principal:

- Primero se declaran las variables necesarias para almacenar los datos que se van a ingresar (líneas 2 a la 4).
- Luego, se solicita el tamaño de los vectores, lo que equivale a solicitar el número de personas a procesar (líneas 6 y 7).
- A continuación, se hace el llamado a las funciones, `leerNombres()` y `leerGeneros()` (líneas 9 y 10), que se utilizan para pedir los datos de las personas y almacenarlos en los arreglos respectivos.

  La función `leerGeneros()` implementa una validación haciendo uso del ciclo `Haga-MientrasQue`, que obliga al usuario a ingresar solo una 'F' de género Femenino o una 'M' de género Masculino.
- Posteriormente, se solicita el nombre de la persona que se va a buscar entre los datos ya almacenados en el vector de nombres.
- Mediante la función `obtenerPosicion()` se busca la posición en la que se supone debe estar el nombre de la persona que se está buscando y que se pasa como parámetro. Esta función trabaja de la siguiente forma:

  - La función recorre el vector de nombres a través de un ciclo `Para`. Previo al ciclo se ha inicializado la variable `pos` en -1, lo que indica que no se tiene una posición válida del arreglo para retornar, pues todavía no se ha llevado a cabo la búsqueda y no se ha encontrado ningún nombre. Se utiliza la instrucción `Si` dentro del ciclo `Para` con el fin de determinar si el nombre buscado es igual al que está almacenado en la posición actual del arreglo; si esto llegase a ocurrir, la variable `pos` ubicada dentro de la estructura `Si` almacena la posición donde se encuentra el nombre.
  - En la parte final de la función `obtenerPosicion()` se retorna la posición (variable `pos`) donde está el nombre del estudiante que se buscó en el arreglo. Si la función no encontró

el nombre buscado, la variable pos retornará el -1 con que se inicializó.

- Por último, en el algoritmo principal se hace el llamado al procedimiento `mostrarResultados()`, el cual recibe como parámetros la posición encontrada por la función anterior, el vector de nombres y el vector de géneros. Con estos datos, el procedimiento muestra el nombre, la posición y el género de la persona que coincida con el nombre buscado; si el contenido de la variable `pos` es igual a -1, esto quiere decir que no se encontró el nombre, en cuyo caso se imprime un mensaje informando este hecho.

.:Ejemplo 6.9. *Construya un algoritmo utilizando funciones, procedimientos y vectores para ingresar y almacenar las estaturas y los géneros de un conjunto de* `n` *personas y después determine el promedio de estaturas de las mujeres y el porcentaje de hombres ingresados.*

Análisis del problema:

- **Resultados esperados:** el promedio de las estaturas de las mujeres y el porcentaje de hombres que se han ingresado y que están almacenados en el arreglo.
- **Datos disponibles:** se conocen previamente, la cantidad de personas que se van a procesar, la estatura y el género de las mismas.
- **Proceso:** este algoritmo debe tener en primer lugar, una función que permita ingresar los datos de las `n` personas y almacenarlos en los arreglos respectivos. En segundo lugar, debe implementar una función para obtener el promedio de las estaturas de las mujeres sumando solo las estaturas de las personas con género femenino y dividiendo entre el total de estas. Otra función que debe tener el algoritmo, es la que determina el porcentaje de hombres que se ingresaron; para ello, se debe contar el número de personas con género masculino y dividirlo entre el total de personas ingresadas. Por último, el algoritmo deberá mostrar los resultados que se han obtenido.
- **Variables requeridas:**
  - En el algoritmo principal

  - `numeroPersonas`: cantidad de personas de las que se conoce la información
  - `arregloEstatura`: arreglo que contiene todas las estaturas
  - `arregloGenero`: arreglo con los géneros de las personas.
  - `promEstMujeres`: variable para almacenar el promedio de las estaturas de todas las mujeres.
  - `porcHombres`: variable para almacenar el porcentaje de hombres en el vector.

- En la función `leerEstaturas`
  - Parámetros:
    - `tamanio`: cantidad de elementos a leer.
  - Variables locales:
    - `i`: variable de control del ciclo.
    - `arregloE`: vector que almacena las estaturas.

- En la función `leerGeneros`
  - Parámetros:
    - `tamanio`: cantidad de elementos a leer.
  - Variables locales:
    - `i`: variable de control del ciclo.
    - `arregloG`: vector que almacena los géneros de las personas ingresadas.

- En la función `obtenerPromedio`
  - Parámetros:
    - `arregloE`: vector con las estaturas de las personas.
    - `arregloG`: vector con los géneros de las personas.
    - `tamanio`: cantidad de elementos de los vectores.
  - Variables locales:
    - `i`: variable de control del ciclo.
    - `suma`: variable que almacena la suma de las estaturas de las mujeres.
    - `promedio`: variable que almacena el promedio de las estaturas de las mujeres.
    - `contadorMujeres`: variable para almacenar la cantidad de mujeres en el arreglo.

- En la función `obtenerPorcentajeHombres`
  - Parámetros:

- ◇ `arregloG`: vector con los géneros de las personas.
- ◇ `tamanio`: cantidad de elementos de los vectores.

- Variables locales:
  - ◇ `i`: variable de control del ciclo.
  - ◇ `hombres`: cantidad de hombres en el arreglo.
  - ◇ `porcentaje`: variable que almacena el porcentaje de hombres que hay en el vector.

- En el procedimiento `mostrarResultados`
  - Parámetros:
    - ◇ `promEstatura`: promedio de las estaturas de las mujeres.
    - ◇ `porcHombres`: porcentaje de hombres en el arreglo.

De acuerdo al análisis planteado, se propone el Algoritmo 6.9.

Algoritmo 6.9: ArregloPersonas

```
1  Algoritmo ArregloPersonas
2    Entero   numeroPersonas
3    Real     arregloEstatura [ ]
4    Caracter arregloGenero [ ]
5    Real     promEstMujeres, porcHombres
6
7    imprimir( "Ingrese la cantidad de personas " )
8    leer( numeroPersonas )
9
10   arregloEstatura = leerEstaturas( numeroPersonas )
11   arregloGenero    = leerGeneros( numeroPersonas )
12
13   promEstMujeres = obtenerPromedio ( arregloEstatura,
14                                      arregloGenero,
                                           numeroPersonas  )
15
16   porcHombres = obtenerPorcentajeHombres( arregloGenero,
17                                           numeroPersonas )
18
19   mostrarResultados( promEstMujeres, porcHombres )
20 FinAlgoritmo
21
22 Funcion Real[ ] leerEstaturas( Entero tamanio )
23      Entero i
24      Real arregloE [ tamanio ]
25
26      Para i = 1 Hasta tamanio Incremento 1
27         imprimir( "Ingrese estatura de la persona " , i )
28         leer( arregloE[ i ] )
```

```
29      FinPara
30      Retornar arregloE
31  FinFuncion
32
33  Funcion Caracter[ ] leerGeneros( Entero tamanio )
34      Entero i
35      Caracter arregloG [ tamanio ]
36
37      Para i = 1 Hasta tamanio Incremento 1
38          imprimir( "Ingrese el género de la persona: F
                Femenino, M Masculino " , i )
39          leer( arregloG[ i ] )
40      FinPara
41      Retornar arregloG
42  FinFuncion
43
44  Funcion Real obtenerPromedio( Real     arregloE [ ],
45                                Caracter arregloG [ ],
46                                Entero   tamanio )
47    Real suma
48    Real promedio
49    Entero contadorMujeres, i
50
51    suma = 0
52    contadorMujeres = 0
53    Para i = 1 Hasta tamanio Incremento 1
54      Si( arregloG[ i ] == 'F' ) Entonces
55          suma = suma + arregloE[ i ]
56          contadorMujeres = contadorMujeres + 1
57      FinSi
58    FinPara
59
60    promedio = suma / contadorMujeres
61    Retornar promedio
62  FinFuncion
63
64  Funcion Real obtenerPorcentajeHombres( Caracter arregloG[ ],
65                                         Entero   tamanio )
66    Entero hombres, i
67    Real   porcentaje
68
69    hombres = 0
70    Para i = 1 Hasta tamanio Incremento 1
71      Si( arregloG[ i ] == 'M' ) Entonces
72        hombres = hombres + 1
73      FinSi
74    FinPara
75
76    porcentaje = (hombres / tamanio) * 100
```

```
77   Retornar porcentaje
78 FinFuncion
79
80 Procedimiento mostrarResultados( Real promEstatura,
81                                  Real porcHombres )
82   imprimir( "Estatura promedio (mujeres): ", promEstatura)
83   imprimir( "Porcentaje de hombres: ", porcHombres )
84 FinProcedimiento
```

Explicación del algoritmo:

En el algoritmo principal, se solicita el ingreso del número de personas que se van a procesar y este dato se almacena en la variable `numeroPersonas`, misma con la que se inicializan los vectores en los que se guardarán las estaturas y los géneros de las personas.

Posteriomente, se invocan las funciones `leerEstaturas()` y `leerGeneros()` con las que se hace la lectura de los datos y se almacenan en las diferentes posiciones de los vectores de estaturas y de géneros.

Luego, se hace el llamado a la función `obtenerPromedio()`, la cual calcula el promedio de estaturas de las mujeres recorriendo el vector de estaturas a través de un ciclo `Para`; dentro de este ciclo, se pregunta mediante un `Si` si el género que se encuentra en la posición i del vector de géneros corresponde a un género femenino; de ser así, se acumula la estatura correspondiente que está en el vector de estaturas en la variable `suma` y se cuenta una mujer mas. Al finalizar este ciclo, se calcula el promedio de estaturas dividendo la variable `suma` entre la variable `contadorMujeres` y se retorna este resultado.

Acto seguido, se invoca la función `obtenerPorcentajeHombres()`, que también utiliza un ciclo `Para` con el propósito de recorrer el vector de géneros e ir contando mediante un `Si`, la cantidad de hombres que hay en el arreglo; una vez obtenida esta cantidad, se divide entre el número de personas que hay y se multiplica este valor por 100.00 para obtener el respectivo porcentaje y poderlo retornar.

Al finalizar, con el procedimiento `mostrarResultados()`, se muestran: promedio de estaturas de las mujeres y el porcentaje de hombres.

.:Ejemplo 6.10. *Construya un algoritmo utilizando funciones y procedimientos que almacene en un arreglo 4 números enteros y luego ordene estos números de menor a mayor utilizando el método de ordenamiento denominado "**Método de la burbuja**". El algoritmo debe*

mostrar el arreglo con los datos ingresados inicialmente (en desorden) y el arreglo con los datos ya ordenados.

Análisis del problema:

- **Resultados esperados:** se pretende que el algoritmo muestre un arreglo de números enteros ordenados de menor a mayor, es decir, en orden ascendente.
- **Datos disponibles:** Los 4 números enteros que se van a almacenar en el arreglo.
- **Proceso:** primero se solicitan los 4 números al usuario y se almacenan en el vector, luego se imprime el vector original con los números en desorden, a continuación, se aplica el método de la "burbuja" al vector y finalmente se vuelve a imprimir este vector. Este orden es importante porque el ordenamiento altera la posición de los elementos dentro del vector, impidiendo imprimir el vector original una vez se ordena. Por supuesto, es posible sacar una copia del vector, ordenar la copia e imprimir los dos vectores; también es posible modificar el algoritmo de ordenamiento para retornar un nuevo vector con los índices sin alterar los datos originales[6] que indique en que orden se deben mostrar los elementos del vector.

  Antes de resolver el ejemplo, es necesario comprender tres conceptos:

  Primero, un algoritmo de ordenamiento tiene como finalidad recibir un conjunto de datos y entregarlos ordenados en forma ascendente o descendente según se especifique.

  Segundo, existen diversas formas de realizar esta tarea de ordenamiento, una de ellas es la denominada "Burbuja".

  Tercero, el algoritmo de ordenamiento "Método de la burbuja" tiene a su vez diversas formas de ser escrito. A continuación se presenta una de estas formas. Por medio de un ejemplo se ilustrará su funcionamiento, el cual consiste en ordenar un arreglo con los datos enteros: 7 2 8 1.

  En esta versión del "Método de la burbuja" se requieren dos ciclos[7], el primer ciclo se encarga de indicar la posición del primer elemento

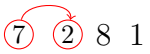
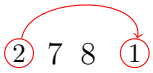
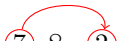
[6] Un vector de indices es interesante porque permite construir diversos vectores que sirvan para recorrer un mismo vector, así es posible recorrer un vector de forma ascendente, descendente, ...

[7] normalmente se emplea el ciclo `Para`

a ser comparado (círculo de donde sale la flecha), el segundo ciclo indica la posición del segundo elemento con el que hay que comparar el primero (círculo donde llega la flecha); en cada comparación se verifica si el primer número es mayor al segundo, de serlo, se intercambian de posición, en otro caso, se continúa con el siguiente número. Una vez se ha comparado el elemento que indica el primer ciclo con todas las posiciones, se continúa con la siguiente en el primer ciclo y así hasta llegar a la penúltima posición[8].

Ejemplo de ordenamiento

`arreglo ->` **7 2 8 1**

$Ciclo_1$	$Ciclo_2$	**Comparación**
1	2	
	3	
	4	
2	3	
	4	
3	4	

`arreglo ->` **1 2 7 8**

- **Variables requeridas:**
  - En el algoritmo principal
    - Constante requerida:
      - `MAX`: constante igual a 4
    - Variable:
      - `numero`: vector con los MAX números a ordenar.

[8]Es la última comparación posible, la penúltima posición con la última.

- En la función `leerArreglo`
  - Parámetros:
    - `tamanio`: cantidad de elementos del vector.
  - Variables locales:
    - `i`: variable de control del ciclo.
    - `arregloE`: vector que almacena los números enteros que el usuario ingresó.
- En la función `imprimirArreglo`
  - Parámetros:
    - `titulo`: texto que se muestra con los datos del vector.
    - `arreglo`: vector que almacena los números enteros que el usuario ingresó.
    - `n`: tamaño del arreglo.
  - Variables locales:
    - `i`: variable de control del primer ciclo.
    - `cadena`: variable que almacena todos los elementos de un arreglo en una `Cadena` mediante concatenación de los elementos.
- En la función `ordenarBurbuja`
  - Parámetros:
    - `arreglo`: arreglo que almacena los números enteros que el usuario ingresó.
    - `n`: tamaño del vector.
  - Variables locales:
    - `i`: variable de control del primer ciclo.
    - `j`: variable de control del segundo ciclo.
    - `auxiliar`: variable necesaria para intercambiar dos posiciones del vector.

De acuerdo al análisis planteado, se propone el Algoritmo 6.10.

Algoritmo 6.10: OrdenamientoBurbuja

```
1 Algoritmo OrdenamientoBurbuja
2   Constante Entero MAX = 4
3   Entero numero [ ]
4
5   numero  = leerArreglo( MAX )
6
7   imprimirArreglo( "Arreglo original: ", numero, MAX )
```

```
8
9   ordenarBurbuja( numero, MAX )
10
11  imprimirArreglo( "Arreglo ordenado: ", numero, MAX )
12 FinAlgoritmo
13
14 Funcion Entero [ ] leerArreglo( Entero tamanio )
15   Entero i
16   Entero arregloE[ tamanio ]
17
18   Para i = 1 Hasta tamanio Incremento 1
19     imprimir( "Ingrese el número " , i, ": " )
20     leer( arregloE[ i ] )
21   FinPara
22   Retornar arregloE
23 FinFuncion
24
25 Procedimiento imprimirArreglo( Cadena titulo,
26                                Entero arreglo [ ],
27                                Entero n )
28   Entero i
29   Cadena  cadena
30
31   Para i = 1 Hasta n Incremento 1
32      cadena = cadena + arreglo[ i ] + " "
33   FinPara
34
35   imprimir( titulo, cadena )
36 FinFuncion
37
38 Procedimiento ordenarBurbuja( Entero arreglo[ ], Entero n)
39      Entero i, j
40      Entero auxiliar
41
42      Para i = 1 Hasta n - 1  Incremento 1
43        Para j = i + 1 Hasta n Incremento 1
44          Si( arreglo[ i ] > arreglo[ j ] Entonces
45            auxiliar     = arreglo[ i ]
46            arreglo[ i ] = arreglo[ j ]
47            arreglo[ j ] = auxiliar
48          FinSi
49        FinPara
50      FinPara
51 FinProcedimiento
```

Explicación del algoritmo:

Inicialmente se declaran las variables necesarias para el algoritmo. Luego, se llama a la función `leerArreglo()`, que se utiliza para capturar los

MAX números enteros (4) y almacenarlos en el vector.

```
2   Constante Entero MAX = 4
3   Entero numero [ ]
4
5   numero  = leerArreglo( MAX )
```

Posteriormente, se procede a imprimir el vector mediante el procedimiento `imprimirArreglo()`, a coontinuación, se invoca la función `ordenarBurbuja()`, que es la encargada de realizar el ordenamiento de los elementos del vector, y finalmente se vuelve a imprimir el vector ya ordenado.

```
7   imprimirArreglo( "Arreglo original: ", numero, MAX )
8
9   ordenarBurbuja( numero, MAX )
10
11  imprimirArreglo( "Arreglo ordenado: ", numero, MAX )
```

Se observa que la forma en que se implementa el algoritmo de ordenamiento coincide con la explicación hecha en el análisis.

```
38 Procedimiento ordenarBurbuja( Entero arreglo[ ], Entero n)
39      Entero i, j
40      Entero auxiliar
41
42      Para i = 1 Hasta n - 1  Incremento 1
43        Para j = i + 1 Hasta n Incremento 1
44          Si( arreglo[ i ] > arreglo[ j ] Entonces
45            auxiliar     = arreglo[ i ]
46            arreglo[ i ] = arreglo[ j ]
47            arreglo[ j ] = auxiliar
48          FinSi
49        FinPara
50      FinPara
51 FinProcedimiento
```

Como se mencionó antes, en la función `ordenarBurbuja()`, se usan dos ciclos; el primero recorre las posiciones del vector, desde la primera hasta la penúltima y el segundo arranca en la segunda posición; lo que permite comparar en un principio la primera con la segunda posición del vector, luego la primera con la tercera y así sucesivamente. al pasar el ciclo externo a la segunda iteración, el algoritmo compara la segunda posición del arreglo con la tercera, cuarta, hasta `n` posición intercambiando los elementos, de allí el nombre de burbuja.

Hasta este punto no se había explicado la forma de intercambiar el contenido de dos posiciones en un arreglo (líneas de la 45 a 47). Para intercambiar el contenido de dos posiciones es necesario el uso de una variable (`auxiliar`) que reciba el contenido de una posición (cualquiera de la dos a intercambiar), en esa posición se almacena el contenido de la segunda posición y en esta posición se almacena en contenido de la variable `auxiliar`.

.:Ejemplo 6.11. *Escriba un algoritmo que almacene un conjunto de* `n` *números enteros en un vector, los ordene ascendentemente mediante el método de la burbuja, que realice la búsqueda de un número cualquiera mediante el método de* ***"Búsqueda binaria"*** *indicando la posición que ocupa en el vector. Utilice funciones para resolver el ejercicio.*

Ejemplo de búsqueda binaria para los siguientes números:

Ejemplo de búsqueda binaria

`numero ->` **1 3 5 9 14 21 45 63 73 87 89 90 97**
`elemento =` **73**

Ciclo	***Comparación***
1	1 3 5 9 14 21 45 **63 73** 87 **89 90 97** *dado que* **45** *es menor que* **73**
2	1 3 5 9 14 21 45 **63** **73** 87 89 90 97 *dado que* **87** *es mayor que* **73**
3	1 3 5 9 14 21 45 63 **73** 87 89 90 97 *Dato encontrado!!!*

La "Búsqueda binaria" permite determinar de una manera muy eficiente la posición de un elemento en un vector ordenado o concluir que el elemento no está presente. Para conocer la posición del elemento, el algoritmo se ubica en la posición del medio y, a través de una comparación, define si debe realizar la busqueda en la mitad superior (mayores a él) o en los elementos de la mitad inferior (menores a él); esto permite excluir de la búsqueda en cada iteración la mitad de los elementos del vector; siguiendo este procedimiento, al final se llega al elemento o se concluye su ausencia.

Análisis del problema:

- **Resultados esperados:** la posición que ocupa en el vector el número buscado.
- **Datos disponibles:** los números que se van a ingresar en dicho arreglo y el número que se va a buscar posteriormente.
- **Proceso:** luego de solicitar los datos que se van a almacenar en el vector, el algoritmo debe ordenar este arreglo, por medio del método de la burbuja como se explicó en el ejercicio anterior. Posteriormente, el algoritmo debe solicitar un número que se va a buscar en el vector y, mediante una función que implemente el método de búsqueda binaria, realice la búsqueda del número e informe en qué posición se encuentra. Este ejercicio se va a resolver con un vector de 13 posiciones.

  - En el algoritmo principal
    - Constante requerida:
      - `MAX`: constante igual a 13.
    - Variables requeridas:
      - `numero`: vector con los MAX números a ordenar.
      - `numeroBuscar`: valor que se desea buscar.
      - `posicion`: posición del vector en donde se encuentra el valor buscado o, -1 de no encontrarlo.
  - En la función `leerArreglo`
    - Parámetros:
      - `tamanio`: tamaño del vector.
    - Variables locales:
      - `i`: variable de control del ciclo.
      - `arregloE`: arreglo que almacena los números enteros que el usuario ingresó.
  - En la función `ordenarBurbuja`
    - Parámetros:
      - `arreglo`: vector que almacena los números enteros que el usuario ingresó.
      - `n`: tamaño del vector.

- Variables locales:
  - ◇ `i`: variable de control del primer ciclo.
  - ◇ `j`: variable de control del segundo ciclo.
  - ◇ `auxiliar`: variable necesaria para intercambiar dos posiciones de un vector.

- En la función `buscarElemento`
  - Parámetros:
    - ◇ `arreglo`: vector que almacena los números enteros que el usuario ingresó.
    - ◇ `n`: tamaño del vector.
    - ◇ `numeroBuscar`: valor que se desea buscar.
  - Variables locales:
    - ◇ `primero`: punto inicial del rango de búsqueda.
    - ◇ `ultimo`: punto final del rango de búsqueda.
    - ◇ `centro`: punto medio del rango de búsqueda.
    - ◇ `posicion`: posición en donde se encuentra el valor buscado o, -1 de no encontrarlo.

De acuerdo al análisis planteado, se propone el Algoritmo 6.11.

Algoritmo 6.11: busquedaBinaria

```
1  Algoritmo busquedaBinaria
2    Constante Entero MAX = 13
3    Entero numero[ ]
4    Entero numeroBuscar, posicion
5
6    numero  = leerArreglo( MAX )
7
8    imprimir( "Número a buscar: " )
9    leer( numeroBuscar )
10
11   ordenarBurbuja ( numero, MAX )
12
13   posicion = buscarElemento( numero, MAX, numeroBuscar )
14
15   Si( posicion > 0 ) Entonces
16     imprimir( "El número buscado se encontró en :", posicion
         )
17   SiNo
18     imprimir( "El número no se encuentra en el arreglo" )
19   FinSi
20 FinAlgoritmo
```

```
21
22 Funcion Entero [ ] leerArreglo( Entero tamanio )
23   Entero i
24   Entero arregloE[ tamanio ]
25
26   Para i = 1 Hasta tamanio Incremento 1
27     imprimir( "Ingrese el número " , i, ": " )
28     leer( arregloE[ i ] )
29   FinPara
30   Retornar arregloE
31 FinFuncion
32
33 Procedimiento ordenarBurbuja( Entero arreglo[ ], Entero n)
34      Entero i, j
35      Entero auxiliar
36
37      Para i = 1 Hasta n - 1  Incremento 1
38        Para j = i + 1 Hasta n Incremento 1
39          Si( arreglo[ i ] > arreglo[ j ] Entonces
40            auxiliar     = arreglo[ i ]
41            arreglo[ i ] = arreglo[ j ]
42            arreglo[ j ] = auxiliar
43          FinSi
44        FinPara
45      FinPara
46 FinProcedimiento
47
48 Funcion Entero buscarElemento( Entero arreglo[ ],
49                                Entero n,
50                                Entero numeroBuscar )
51   Entero primero, ultimo, centro
52   Entero posición = -1
53
54   primero = 1
55   ultimo  = n
56
57   Mientras( posicion == -1 Y primero <= ultimo )
58     centro = (primero + ultimo) / 2
59     Si( numero == arreglo[ centro ] )Entonces
60       posicion = centro
61     SiNo
62       Si numero < arreglo[ centro ] Entonces
63         ultimo = centro - 1
64       SiNo
65         primero = centro + 1
66       FinSi
67     FinSi
68   FinMientras
69
```

```
70   Retornar posicion
71 FinFuncion
```

Explicación del algoritmo:

En el algoritmo principal se leen los 13 elementos (constante `MAX`) por medio de la función `leerArreglo()` (línea 6), que funciona como ya se ha explicado en los anteriores ejercicios; luego, se solicita al usuario el número a buscar con las instrucciones `imprimir y leer` (líneas 8 y 9), posteriormente se ordenan los números usando la función `ordenarBurbuja()` (línea 11) explicada en el ejemplo anterior, luego se hace la búsqueda usando la función `buscarElemento()` (línea 13), dependiendo de la respuesta de la función y usando una decisión, se imprime la posición o se indica que el número solicitado no se encuentra en el arreglo (líneas de la 15 a la 19).

La función `buscarElemento()` implementa la búsqueda binaria explicada al inicio del ejercicio. Esta función trabaja de la siguiente manera: Utiliza una estructura cíclica `Mientras-FinMientas` que tiene como condición que la variable posición sea -1 y que la variable `primero` sea menor o igual a la variable `ultimo`; esto significa que se realizó la búsqueda en el arreglo y no se encontró el elemento.

Dentro del ciclo se encuentra la instrucción que calcula el centro para cada iteración y una instrucción `Si` que permite saber si el elemento se encontró; esto se hace con la instrucción `posicion=centro`; si no se encontró, se reubica la primera o última posición ya sea a la izquierda o derecha del centro(dependiendo si el número buscado es menor o mayor) del vector para, con esto descartar la búsqueda en alguno de los dos lados. Por lo anterior, es que el vector debe estar ordenado.

Merece especial cuidado, dentro de esta función, la siguiente línea:

```
58      centro = (primero + ultimo) / 2
```

Esta instrucción permite reubicar el centro del vector para partirlo y determinar si se realiza la búsqueda del elemento del lado izquierdo o derecho del mismo. En cada iteración se vuelve a ubicar el centro entre aquellos elementos cercanos al número buscado.

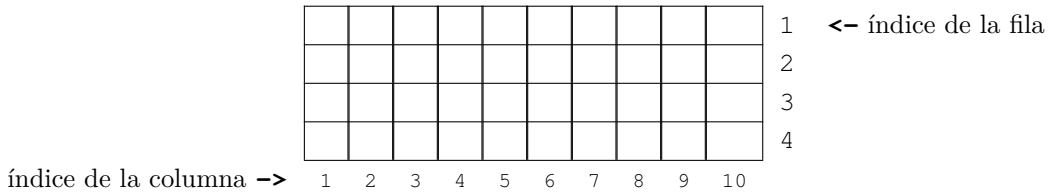
En la línea se observan tres variables de tipo `Entero`, así que la división que allí se presenta es una división entera, es decir, si la variable `primero` igual a 1 y la variable `ultimo` igual a 6 el resultado es 3 y no 3.5.

```
(1 + 6) / 2
7 / 2
3
```

6.2. Matrices

Los arreglos de dos dimensiones se conocen también como matrices. Al igual que los vectores, las matrices son conjuntos de celdas de memoria que almacenan datos del mismo tipo, pero esta vez en forma de tablas, con filas y columnas; de aquí su nombre de arreglos bidimensionales. Las matrices son variables a las que se les debe dar un nombre con un identificador válido y debe especificarse también su tipo de datos al momento de declararlas.

Las matrices son bastante útiles en la solución de aquellos ejercicios en los que se requiera almacenar varios datos de varios objetos, por ejemplo, supóngase que se deben almacenar las 10 notas parciales de cada uno de los 4 estudiantes de un grupo; esto significa que en total se requieren 40 espacios de memoria. Este problema puede resolverse fácilmente utilizando una matriz o arreglo bidimensional.



La forma de interpretarlo es que cada celda o posición requiere de dos coordenadas `(fila, columna)` para ubicar una posición. En términos generales, suponga por ejemplo una matriz de m filas y n columnas, donde el primer indice especifica la fila y el segundo la columna.

$$matriz = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}$$

6.2.1 Declaración de una matriz

Las matrices también requieren declarar una variable suscrita que permita acceder a los datos. Una variable suscrita de una matriz se diferencia de la de un vector por la cantidad de corchetes en su declaración. La cantidad de juegos de corchetes indica la dimensión del arreglo.

```
Entero matrizEdades   [ ][ ]
Real   matrizSalarios [ ][ ]
Cadena matrizNombres  [ ][ ]
```

El doble juego de corchetes al final, se usa para indicar que se debe separar espacio tanto para las filas y las columnas que va a tener la matriz. Por supuesto, luego de declararse la variable suscrita, se deberá especificar el tamaño, es decir, indicar el número de filas y columnas que esta va a tener. Esto se hace de la misma forma que se hizo con los arreglos unidimensionales, es decir, utilizando la función `dimensionar`, de la siguiente forma:

```
Entero matrizEdades   [ ][ ]
Real   matrizSalarios [ ][ ]
Cadena matrizNombres  [ ][ ]

matrizEdades   = dimensionar(  4,  3 )
matrizSalarios = dimensionar( 10, 12 )
matrizNombres  = dimensionar( 10, 10 )
```

O también, se puede dar tamaño cuando se está declarando la matriz de la siguiente forma:

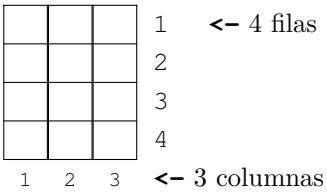
```
Cadena matrizNombres [ 3 ][ 10 ]
Real   matrizSalarios[ 15 ][ 5 ]
```

La forma de dimensionar las matrices, como los vectores, como ya se ha dicho, dependerá del lenguaje de programación y sus instrucciones propias para este propósito.

En todos los casos, el número dentro del primer juego de corchetes especifica la cantidad de filas que tendrá la matriz, mientras que el número en el segundo juego de corchetes indica la cantidad de columnas.

```
Entero matrizEdades [ 4 ][ 3 ]
```

Al declarar la matriz de edades anterior denominada `matrizEdades`, gráficamente se verá de la siguiente manera:

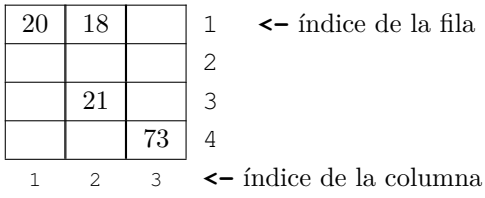


6.2.2 Almacenamiento de datos en una matriz

Cuando se va a almacenar los datos en una matriz, es necesario especificar la celda en la que cada dato va a ir quedando; para ello, se debe indicar, a través de números índice, la fila y la columna en la que se encuentra la celda, de la siguiente manera:

```
matrizEdades [1][1] = 20
matrizEdades [1][2] = 18
matrizEdades [3][2] = 21
matrizEdades [4][3] = 73
```

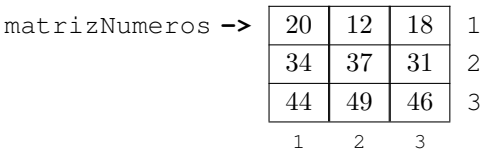
Con estas instrucciones, se almacenaría un 20 en la celda que se encuentra en la fila 1 y columna 1, un 18 en la celda ubicada en la fila 1 y columna 2, un 21 en la celda que está en la fila 3 y columna 2 y un 73 en la celda que está en la fila 3 y columna 4. De tal forma que la matriz de edades quedaría de la siguiente forma:



Si se desea almacenar una serie de datos en una matriz de 3 filas por 3 columnas, también se podrían almacenar directamente de la siguiente forma:

```
Entero matrizNumeros[ ][ ] = { { 20, 12, 18},
                               { 34, 37, 31},
                               { 44, 49, 46} }
```

Donde cada conjunto de llaves interiores especifican cada fila de la matriz.



6.2.3 Recorrido de una matriz

Recorrer una matriz significa pasar por cada una de las celdas que la componen, ya sea para ingresar datos o para leer los que se encuentran almacenados en esta. Una matriz puede recorrerse de diferentes formas, por ejemplo: la manera tradicional consiste en recorrer primero todas las celdas de la primera fila y cuando se llegue a la última celda de esta fila, pasar a la primera celda de la segunda fila y así sucesivamente hasta llegar a la última celda ubicada en la última fila y la última columna.

Una segunda forma de recorrer una matriz sería pasar inicialmente por cada una de las celdas que conforman la primera columna arrancando en la primera fila, luego ubicarse en la primera celda de la segunda columna y recorrer toda la columna y así hasta llegar a la última celda de la matriz en la última fila y última columna. Sin embargo, podrían llevarse a cabo muchas otras formas de recorrer una matriz.

Para efectos prácticos, se llevará a cabo el recorrido tradicional a una matriz, esto es, arrancando en la primera celda ubicada en la primera fila y columna, recorriendo toda la fila y al terminar de recorrer esta, ubicándose en la primera celda de la segunda fila, para de esta forma hacer todo el recorrido. Esta forma de recorrer la matriz se lleva a cabo mediante dos ciclos anidados, el externo permite avanzar de una fila a otra mientras que el interno lo hace a través de las celdas de una columna a la siguiente. A continuación, la porción de código necesaria para recorrer una matriz y llenarla con datos que el usuario ingresa:

```
Constante Entero NUMERO_FILAS    = 4
Constante Entero NUMERO_COLUMNAS = 5

Entero matriz[ NUMERO_FILAS ][ NUMERO_COLUMNAS ]

Para i = 1 Hasta NUMERO_FILAS Incremento 1
  Para j = 1 Hasta NUMERO_COLUMNAS Incremento 1
    imprimir( "Ingrese la posición (", j, ", ", i, ") : " )
    leer( matriz[ i ][ j ] )
  FinPara
FinPara
```

En la anterior porción de código, es necesario tener en cuenta lo siguiente:

- El ciclo externo haría el recorrido de las filas, mientras que el ciclo interno recorrería las columnas.
- Se puede notar que el ciclo interno se ejecuta más rápidamente. Cuando el ciclo interno termine de ejecutarse, el algoritmo pasará al ciclo externo que lo llevará a una fila siguiente.
- La constante `NUMERO_FILAS` almacena el número de filas que tiene la matríz. Así que con la condición del ciclo externo, el recorrido se limita al número de filas que tiene la matriz.
- La constante `NUMERO_COLUMNAS` almacena el número de columnas, con la condición del ciclo interno se recorren las columnas en cada fila.

A continuación, se exponen una serie de ejemplos con el fin de ilustrar los conceptos que se acaban de explicar y el uso practico de las matrices.

Los primeros ejemplos se escribieron en forma secuencial y los siguientes, utilizando funciones y procedimientos.

.:Ejemplo 6.12. *Una matriz transpuesta es el resultado de tomar una matriz inicial e invertir sus filas y columnas, es decir, que la matriz transpuesta contiene como columnas las filas de la matriz original y, las filas corresponden a las columnas de la matriz original. Diseñe un algoritmo que, a partir de una matriz inicial, obtenga su transpuesta y la muestre al usuario.*

El Álgebra lineal estudia las matrices y sus propiedades; uno de los aspectos que se estudia cuando se trabaja con matrices consiste en encontrar la transpuesta de una matriz. En este ejercicio se codifica un algoritmo que permita encontrar la transpuesta de una matriz dada.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}$$

La transpuesta de la matriz A, es simbolizada como A^T.

$$A^T = \begin{bmatrix} a_{11} & a_{21} & a_{31} & \cdots & a_{n1} \\ a_{12} & a_{22} & a_{32} & \cdots & a_{n2} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ a_{1m} & a_{2m} & a_{4m} & \cdots & a_{nm} \end{bmatrix}$$

Análisis del problema:

- **Resultados esperados:** la transpuesta de una matriz ingresada, al igual que la original.
- **Datos disponibles:** Se debe conocer la matriz original, incluyendo su tamaño y los elementos ubicados en sus diferentes posiciones.
- **Proceso:** Luego de tener la matriz original ya cargada con datos, el algoritmo debe recorrer esta matriz fila a fila e ir pasando los elementos a cada columna de la matriz transpuesta. Una vez se haya terminado el proceso, se muestran ambas matrices.
- **Variables requeridas:**
  - `matrizInicial`: matriz con los datos iniciales
  - `matrizTranspuesta`: matriz con los datos iniciales pero con las filas y columnas intercambiadas (transpuesta)
  - `filas`: número de filas de la matriz inicial.
  - `columnas`: número de columnas de la matriz inicial.
  - `i`: variable de control del primer ciclo.
  - `j`: variable de control del segundo ciclo.
  - `cadena1`: cadena que almacenará todos los elementos de la matriz inicial para su posterior impresión.
  - `cadena2`: cadena que almacenará todos los elementos de la matriz transpuesta para su posterior impresión.

De acuerdo al análisis planteado, se propone el Algoritmo 6.12.

Algoritmo 6.12: TranspuestaMatriz

```
1 Algoritmo TranspuestaMatriz
2   Entero matrizInicial [ ][ ], matrizTranspuesta [ ][ ]
3   Entero filas, columnas, i, j
4   Cadena cadena1, cadena2
5
```

```
6   imprimir( "Número de filas para la matriz: " )
7   leer( filas )
8
9   imprimir( "Número de columnas para la matriz: " )
10  leer( columnas )
11
12  matrizInicial     = dimensionar( filas, columnas )
13  matrizTranspuesta = dimensionar( columnas, filas )
14
15  Para i = 1 Hasta filas Incremento 1
16    Para j = 1 Hasta columnas Incremento 1
17      imprimir( "Número para posición " , i, " ", j, ": " )
18      leer( matrizInicial[ i ][ j ] )
19    FinPara
20  FinPara
21
22  Para i = 1 Hasta filas Incremento 1
23    Para j = 1 Hasta columnas Incremento 1
24      matrizTranspuesta[ j ][ i ] = matrizInicial[ i ][ j ]
25    FinPara
26  FinPara
27
28  cadena1 = ""
29  Para i = 1 Hasta filas Incremento 1
30    Para j = 1 Hasta columnas Incremento 1
31      cadena1 = cadena1 + matrizInicial    [ i ][ j ] + " "
32    FinPara
33    cadena1 = cadena1 + SALTO_LINEA
34  FinPara
35
36  cadena2 = ""
37  Para i = 1 Hasta columnas Incremento 1
38    Para j = 1 Hasta filas Incremento 1
39      cadena2 = cadena2 + matrizTranspuesta[ i ][ j ] + " "
40    FinPara
41    cadena2 = cadena2 + SALTO_LINEA
42  FinPara
43
44  imprimir( "Matriz inicial:     ", cadena1 )
45  imprimir( "Matriz transpuesta: ", cadena2 )
46 FinAlgoritmo
```

Explicación del algoritmo:

Inicialmente, se realiza la declaración de todas las variables necesarias (líneas 2 a la 4). A continuación, se solicita la cantidad de `filas` y de `columnas` que tendrá la matriz inicial, que corresponde al mismo número de `columnas` y de `filas` de la matriz transpuesta.

```
6   imprimir( "Número de filas para la matriz: " )
7   leer( filas )
8
9   imprimir( "Número de columnas para la matriz: " )
10  leer( columnas )
11
12  matrizInicial     = dimensionar( filas, columnas )
13  matrizTranspuesta = dimensionar( columnas, filas )
```

Luego, utilizando dos ciclos `Para` anidados, se solicitan los datos para la matriz inicial.

```
15  Para i = 1 Hasta filas Incremento 1
16    Para j = 1 Hasta columnas Incremento 1
17      imprimir( "Número para posición " , i, " ", j, ": " )
18      leer( matrizInicial[ i ][ j ] )
19    FinPara
20  FinPara
```

Después, usando nuevamente dos ciclos `Para` anidados, se pasan los elementos de la matriz inicial a la transpuesta.

```
22  Para i = 1 Hasta filas Incremento 1
23    Para j = 1 Hasta columnas Incremento 1
24      matrizTranspuesta[ j ][ i ] = matrizInicial[ i ][ j ]
25    FinPara
26  FinPara
```

Note que, en esta parte, se usan los índices `i` y `j` para recorrer filas y columnas de la matriz inicial y columnas y filas en la matriz transpuesta. Es por eso que estos índices están cambiados en la instrucción:

```
24      matrizTranspuesta[ j ][ i ] = matrizInicial[ i ][ j ]
```

De tal forma que, a medida que se recorren las filas de la matriz inicial, se recorren las columnas de la transpuesta y se van pasando los elementos respectivos.

```
22  Para i = 1 Hasta filas Incremento 1
23    Para j = 1 Hasta columnas Incremento 1
24      matrizTranspuesta[ j ][ i ] = matrizInicial[ i ][ j ]
25    FinPara
26  FinPara
```

Luego de haber pasado los elementos a la matriz transpuesta, se recorre la matriz inicial y se pasan los elementos a la variable `cadena1` (líneas de la 28 a la 34):

```
28   cadena1 = ""
29   Para i = 1 Hasta filas Incremento 1
30     Para j = 1 Hasta columnas Incremento 1
31       cadena1 = cadena1 + matrizInicial    [ i ][ j ] + " "
32     FinPara
33     cadena1 = cadena1 + SALTO_LINEA
34   FinPara
```

Con el propósito de mostrar todos los datos en una sola invocación de la instrucción `imprimir` (línea 44). Igual se hace al pasar la matriz transpuesta a la variable `cadena2` (líneas de la 36 a 42), lo que facilita su impresión (línea 45).

Al ejecutar el algoritmo, se visualizaría:

```
Matriz Inicial:   20  12  18
                  34  37  31
                  44  49  46

Matriz trasnpuesta:  20  34  44
                     12  37  49
                     18  31  46
```

.:Ejemplo 6.13. *Diseñe un algoritmo que permita almacenar números enteros entre 50 y 100 en una matriz de `n` filas y `m` columnas y luego, se pueda buscar un número cualquiera dentro del rango ingresado y el algoritmo determine en qué posición (`fila` y `columna`) se encuentra, si es que está almacenado, de lo contrario, el algoritmo deberá decir que el número buscado no está en la matriz. Si el número se encuentra almacenado varias veces, el algoritmo retorna la posición de la primera ocurrencia.*

Este segundo ejercicio de matrices considera almacenar números en el rango entre 50 y 100 (lo cual requiere que se haga una validación) en una matriz en la que el número de filas y columnas será ingresado por el usuario. Posteriormente, el algoritmo implica llevar a cabo una búsqueda recorriendo la matriz como se indicó anteriormente. Este ejercicio se resolverá de forma secuencial, sin el uso de funciones.

Análisis del problema:

- **Resultados esperados:** la posición, diciendo en qué fila y columna se encuentra almacenado el número que se está buscando. Si el número no es encontrado, debe mostrarse un mensaje.

- **Datos disponibles:** la cantidad de filas y de columnas que tendrá la matriz y los números que se guardarán en las diversas posiciones de la matriz.
- **Proceso:** luego de almacenar los datos en la matriz, el algoritmo deberá llevar a cabo una búsqueda secuencial, recorriendo las celdas de la primera fila y pasando luego a la siguiente fila para seguir buscando, hasta encontrar el número ingresado por el usuario. Si se ha recorrido toda la matriz y no se ha encontrado el número, se debe informar al usuario. Si el número se encuentra varias veces en la matriz, se muestra la posición de la primer coincidencia.
- **Variables requeridas:**
  - `matriz`: matriz con los datos ingresados por el usuario.
  - `filas`: número de filas de la matriz.
  - `columnas`: número de columnas de la matriz.
  - `numero`: valor que se desea buscar.
  - `encontrado`: bandera que indica si el `numero` si se encuentra o no en la matriz.
  - `i`: variable de control del primer ciclo.
  - `j`: variable de control del segundo ciclo.

De acuerdo al análisis planteado, se propone el Algoritmo 6.13.

Algoritmo 6.13: BusquedaElemento

```
1  Algoritmo BusquedaElemento
2    Entero filas, columnas, i, j, numero
3    Entero matriz [ ]
4    Logico encontrado
5
6    imprimir( "Número de filas para la matriz: " )
7    leer( filas )
8
9    imprimir( "Número de columnas para la matriz: " )
10   leer( columnas )
11
12   matriz = dimensionar( filas, columnas )
13
14   Para i = 1 Hasta filas Incremento 1
15     Para j = 1 Hasta columnas Incremento 1
16       Haga
17         imprimir( "Ingrese numero para posición " , i, j )
18         leer( matriz[ i ][ j ] )
19
```

```
20          Si( matriz[ i ][ j ] < 50 O
21              matriz[ i ][ j ] > 100 ) Entonces
22            imprimir( "Ha ingresado un número equivocado" )
23          FinSi
24        MientrasQue( matriz[ i ][ j ] < 50 O
25                     matriz[ i ][ j ] > 100 )
26      FinPara
27    FinPara
28
29    imprimir( "Ingrese el número a buscar en la matriz ")
30    leer( numero )
31
32    encontrado = Falso
33    Para i = 1 Hasta filas Incremento 1
34      Para j = 1 Hasta columnas Incremento 1
35        Si( matriz[ i ][ j ] == numero Y encontrado == Falso )
              Entonces
36          imprimir( "Encontrado en la posición" , i, " ", j )
37          encontrado = Verdadero
38        FinSi
39      FinPara
40    FinPara
41
42    Si( encontrado == Falso ) Entonces
43      imprimir( "El número no está almacenado en la matriz " )
44    FinSi
45  FinAlgoritmo
```

Explicación del algoritmo:

Después de declarar las variables necesarias, el algoritmo solicita el número de `filas` y `columnas` que va a tener la matriz y captura estos datos. Con la función `dimensionar()`, se le da el tamaño a la matriz.

```
6   imprimir( "Número de filas para la matriz: " )
7   leer( filas )
8
9   imprimir( "Número de columnas para la matriz: " )
10  leer( columnas )
11
12  matriz = dimensionar( filas, columnas )
```

Vienen luego dos ciclos `Para`, uno dentro del otro, con el fín de solicitar los números que se van a almacenar en las celdas de la matríz. Note que dentro del segundo ciclo, está una instrucción `Haga-MientrasQue`, que tiene el propósito de validar los números que está ingresando el usuario y que deben pertenecer al rango entre 50 y 100, según lo requiere el algoritmo.

```
14  Para i = 1 Hasta filas Incremento 1
15    Para j = 1 Hasta columnas Incremento 1
16      Haga
17        imprimir( "Ingrese numero para posición " , i, j )
18        leer( matriz[ i ][ j ] )
19
20        Si( matriz[ i ][ j ] < 50 O
21            matriz[ i ][ j ] > 100 ) Entonces
22          imprimir( "Ha ingresado un número equivocado" )
23        FinSi
24      MientrasQue( matriz[ i ][ j ] < 50 O
25                   matriz[ i ][ j ] > 100 )
26    FinPara
27  FinPara
```

Al terminar los dos ciclos `Para` anidados, se solicita al usuario que ingrese el número que se va a buscar en la matriz.

```
29  imprimir( "Ingrese el número a buscar en la matriz ")
30  leer( numero )
```

Ahora se procede a realizar la búsqueda, para ello se inicializa la variable `encontrado` en `Falso`, pues todavía no se ha encontrado dicho número. Nuevamente se recorre la matriz con dos ciclos `Para` anidados como se explicó al principio del tema de matrices o arreglos bidimensionales. Dentro de estos ciclos se ubica una instrucción `Si` que compara si lo que hay almacenado en la posición `i`, `j` de la matriz, es igual a lo que contiene la variable `numero`; si es así, se encontró el elemento y se muestra al usuario, además se cambia el valor de la variable `encontrado` a `Verdadero`, para evitar imprimir nuevamente el mensaje si el número está en varias posiciones de la matriz.

```
32  encontrado = Falso
33  Para i = 1 Hasta filas Incremento 1
34    Para j = 1 Hasta columnas Incremento 1
35      Si( matriz[ i ][ j ] == numero Y encontrado == Falso )
            Entonces
36        imprimir( "Encontrado en la posición" , i, " ", j )
37        encontrado = Verdadero
38      FinSi
39    FinPara
40  FinPara
```

Después de estos dos ciclos `Para` anidados, se muestra un mensaje dentro de una instrucción `Si` que solo se muestra si la variable `encontrado` es `Falso`, es decir, si no se encontró el número buscado en toda la matriz.

```
42   Si( encontrado == Falso ) Entonces
43     imprimir( "El número no está almacenado en la matriz " )
44   FinSi
```

Los ejercicios de matrices que siguen a continuación, se resolverán haciendo uso de funciones y procedimientos, ya que, como se mencionó antes, los algoritmos se harán más modulares y fáciles de entender.

.:Ejemplo 6.14. *Escriba un algoritmo que almacene en dos matrices de tamaño 3 x 3, es decir, 3 filas y 3 columnas, números enteros y que a través de una función permita calcular y almacenar en una tercera matriz la suma de los números almacenados en las celdas equivalentes de las dos anteriores. Al final, el algoritmo debe mostrar las tres matrices.*

En términos generales:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}$$

$$B = \begin{bmatrix} b_{11} & b_{12} & b_{13} & \cdots & b_{1n} \\ b_{21} & b_{22} & b_{23} & \cdots & b_{2n} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ b_{m1} & b_{m2} & b_{m3} & \cdots & b_{mn} \end{bmatrix}$$

$$C = A + B = \begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & a_{13}+b_{13} & \cdots & b_{1n}+b_{1n} \\ a_{21}+b_{21} & a_{22}+b_{22} & a_{23}+b_{23} & \cdots & a_{2n}+b_{2n} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ a_{m1}+b_{m1} & a_{m2}+b_{m2} & a_{m3}+b_{m3} & \cdots & a_{mn}+b_{mn} \end{bmatrix}$$

Análisis del problema:

- **Resultados esperados:** una matriz donde los números almacenados en cada celda son el resultado de la suma de los números ubicados en las respectivas celdas de las matrices ingresadas.
- **Datos disponibles:** se tienen los números enteros que van a ser almacenados en las dos primeras matrices y que van a ser sumados.

- **Proceso:** este algoritmo tendrá que calcular la suma posición por posición, de los números en las celdas ubicadas en la misma posición de dos matrices con esos números ya ingresados.
- **Variables requeridas:**
  - En el algoritmo principal
    - Constante requerida:
      - ◇ `MAX`: constante que da tamaño a las matrices (para el ejemplo 3).
    - Variables:
      - ◇ `matrizA`: la primera matriz a sumar.
      - ◇ `matrizB`: la segunda matriz a sumar.
      - ◇ `matrizC`: matriz resultante de sumar las dos primeras.
  - En la función `leerDatos`
    - Parámetros:
      - ◇ `tamanio`: para dar la cantidad de filas y columnas a la matriz.
    - Variables locales:
      - ◇ `i`: controla el ciclo externo de las filas.
      - ◇ `j`: controla el ciclo interno de las columnas.
      - ◇ `matriz`: matriz de datos que será retornada.
  - En la función `calcularSuma`
    - Parámetros:
      - ◇ `m1`: primera matriz a sumar.
      - ◇ `m2`: segunda matriz a sumar.
      - ◇ `tamanio`: Tamaño de las matrices (igual número de filas y columnas)
    - Variables locales:
      - ◇ `i`: controla el ciclo externo.
      - ◇ `j`: controla el ciclo interno.
      - ◇ `matriz`: almacena la suma de las matrices.
  - En la función `mostrarResultados`
    - Parámetros:
      - ◇ `m1`: primera matriz a imprimir.
      - ◇ `m2`: segunda matriz a imprimir.
      - ◇ `m3`: tercera matriz a imprimir.

- ◇ `tamanio`: Tamaño de las matrices (igual número de filas y columnas)

- Variables locales:
  - ◇ `i`: variable de control del ciclo externo.
  - ◇ `j`: variable de control del ciclo interno.
  - ◇ `cadenaA`: almacena los números de la matriz 1.
  - ◇ `cadenaB`: almacena los números de la matriz 2.
  - ◇ `cadenaC`: almacena los números de la matriz 3.

De acuerdo al análisis planteado, se propone el Algoritmo 6.14.

Algoritmo 6.14: SumaMatrices

```
1  Algoritmo SumaMatrices
2    Constante Entero MAX = 3
3    Entero matrizA [ ][ ]
4    Entero matrizB [ ][ ]
5    Entero matrizC [ ][ ]
6
7    matrizA = leerDatos( MAX )
8    matrizB = leerDatos( MAX )
9
10   matrizC = calcularSuma( matrizA, matrizB, MAX )
11
12   mostrarResultados( matrizA, matrizB, matrizC, MAX )
13 FinAlgoritmo
14
15 Funcion Entero[ ][ ] leerDatos(Entero tamanio)
16   Entero i, j
17   Entero matriz[ tamanio ] [ tamanio ]
18
19   Para i = 1 Hasta tamanio Incremento 1
20     Para j = 1 Hasta tamanio Incremento 1
21       imprimir( "Número para posición " , i," ", j, ":" )
22       leer( matriz[ i ][ j ] )
23     FinPara
24   FinPara
25
26   Retornar matriz
27 FinFuncion
28
29 Funcion Entero[ ][ ] calcularSuma( Entero m1[ ][ ],
30                                    Entero m2[ ][ ],
31                                    Entero tamanio )
32   Entero i, j
33   Entero matriz[ tamanio ] [ tamanio ]
34
35   Para i = 1 Hasta tamanio Incremento 1
```

```
36      Para j = 1 Hasta tamanio Incremento 1
37        matriz[ i ][ j ] = m1[ i ][ j ] + m2[ i ][ j ]
38    FinPara
39    FinPara
40
41    Retornar matriz
42  FinFuncion
43
44  Procedimiento mostrarResultados( Entero [ ][ ] m1,
45                                   Entero [ ][ ] m2,
46                                   Entero [ ][ ] m3,
47                                   Entero tamanio )
48    Entero i, j
49    Cadena cadenaA, cadenaB, cadenaC
50
51    Para i = 1 Hasta tamanio Incremento 1
52      Para j = 1 Hasta tamanio Incremento 1
53        cadenaA = cadenaA, m1[ i ][ j ] + " "
54        cadenaB = cadenaB, m2[ i ][ j ] + " "
55        cadenaC = cadenaC, m3[ i ][ j ] + " "
56      FinPara
57      cadenaA = cadenaA + SALTO_LINEA
58      cadenaB = cadenaB + SALTO_LINEA
59      cadenaC = cadenaC + SALTO_LINEA
60    FinPara
61
62    imprimir( "Matriz A ", cadenaA )
63    imprimir( "Matriz B ", cadenaB )
64    imprimir( "Matriz Resultado ", cadenaC )
65  FinProcedimiento
```

Explicación del algoritmo:

En este algoritmo, lo primero que se hace es declarar las variables suscritas que necesarias para las tres matrices de tamaño 3 x 3. El tamaño de estas matrices se da a partir de una constante `MAX` de tipo `Entero` que almacena el número 3 y que es el mismo para las filas y las columnas.

```
2   Constante Entero MAX = 3
3   Entero matrizA [ ][ ]
4   Entero matrizB [ ][ ]
5   Entero matrizC [ ][ ]
```

Se utiliza la función `leerDatos()` para solicitar los números enteros que se van a almacenar en las dos primeras matrices, y se envía como argumento el tamaño de las mismas (número de filas igual al número de columnas), para dimensionarlas internamente;

```
7   matrizA = leerDatos( MAX )
```

```
8   matrizB = leerDatos( MAX )
```

Esta función consta de dos ciclos `Para` anidados que permiten recorrer las filas y las columnas e ir ubicándose en cada celda de la matriz y poder almacenar allí cada número que el usuario va ingresando.

Una vez ingresados los datos, se usa la función `calcularSuma()` que recibe como parámetros las dos matrices ya con datos y el tamaño de las mismas, con el que se da tamaño a una matriz que está dentro de esta función y que sirve para almacenar los resultados de la suma; esta matriz será retornada al algoritmo principal con los resultados de la suma de los valores de las celdas ubicadas en la misma posición de las matrices que se van a sumar. Inicialmente se hace el llamado con:

```
10  matrizC = calcularSuma( matrizA, matrizB, MAX )
```

Y posteriormente la función ejecuta su código interno:

```
29 Funcion Entero[ ][ ] calcularSuma( Entero m1[ ][ ],
30                                    Entero m2[ ][ ],
31                                    Entero tamanio )
32   Entero i, j
33   Entero matriz[ tamanio ] [ tamanio ]
34
35   Para i = 1 Hasta tamanio Incremento 1
36     Para j = 1 Hasta tamanio Incremento 1
37       matriz[ i ][ j ] = m1[ i ][ j ] + m2[ i ][ j ]
38   FinPara
39   FinPara
40
41   Retornar matriz
42 FinFuncion
```

Esta función utiliza dos ciclos `Para` con los que se recorren las celdas de las dos matrices con datos y se hace la suma de los valores que están allí, almacenándolos en las celdas de la matriz de resultados.

```
35   Para i = 1 Hasta tamanio Incremento 1
36     Para j = 1 Hasta tamanio Incremento 1
37       matriz[ i ][ j ] = m1[ i ][ j ] + m2[ i ][ j ]
38   FinPara
39   FinPara
```

Por último, las tres matrices y su tamaño se pasan como argumento al procedimiento `mostrarResultados()`

```
12   mostrarResultados( matrizA, matrizB, matrizC, MAX )
```

Esta función recorre mediante dos ciclos `Para` anidados las matrices y va pasando los datos de estas matrices a tres variables de tipo `Cadena` que serán las que se mostrarán al usuario mediante la función `imprimir()`.

```
51  Para i = 1 Hasta tamanio Incremento 1
52    Para j = 1 Hasta tamanio Incremento 1
53      cadenaA = cadenaA, m1[ i ][ j ] + " "
54      cadenaB = cadenaB, m2[ i ][ j ] + " "
55      cadenaC = cadenaC, m3[ i ][ j ] + " "
56    FinPara
57    cadenaA = cadenaA + SALTO_LINEA
58    cadenaB = cadenaB + SALTO_LINEA
59    cadenaC = cadenaC + SALTO_LINEA
60  FinPara
61
62  imprimir( "Matriz A ", cadenaA )
63  imprimir( "Matriz B ", cadenaB )
64  imprimir( "Matriz Resultado ", cadenaC )
```

Para pasar los datos que están en las matrices a las variables de tipo `Cadena`, se recorre con dos ciclos `Para` anidados cada matriz y se usa la variable de tipo `Cadena` para ir almacenando de forma concatenada lo que hay en las posiciones de cada matriz. Al terminar de pasar lo que hay en cada fila, es decir, de recorrer el ciclo `Para` interno, se ingresa un salto de línea, lo que permite que, cuando se muestre esta variable cadena, se vea cada fila de la matriz separada de la anterior fila, dando la sensación al usuario de estar viendo una tabla o matriz.

.:Ejemplo 6.15. *Se desea construir un algoritmo que permita almacenar en una matriz cada uno de los doce sueldos pagados durante un año a un conjunto de 10 empleados. El algoritmo debe determinar el valor total de la nómina pagado durante un mes cualquiera del año y el total pagado durante el año a un empleado cualquiera. Diseñe el algoritmo utilizando funciones y procedimientos.*

$$matrizSueldos = \begin{bmatrix} s_{11} & s_{12} & s_{13} & \cdots & s_{1\,12} \\ s_{21} & s_{22} & s_{23} & \cdots & s_{2\,12} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ s_{10\,1} & s_{10\,2} & s_{10\,3} & \cdots & s_{10\,12} \end{bmatrix}$$

Se tienen 12 columnas, una por cada mes del año y se tienen 10 filas, una por cada empleado. Así, por ejemplo s_{23} representa el sueldo del segundo empleado en el tercer mes (Marzo).

Análisis del problema:

- **Resultados esperados:** el total de la nómina del mes ingresado por el usuario, así como el total pagado durante el año al empleado que se desee analizar.
- **Datos disponibles:** se conocen, por cada empleado los 12 sueldos ganados en los 12 meses del año.
- **Proceso:** el algoritmo debe calcular, a través de una suma, cuánto se pagó en total a todos los empleados durante el mes `m` (`m` representa el mes que el usuario desee); esto implica que se recorra la columna `m` de la matriz para sumar todos sus valores y retornar este total. Igualmente, se deben sumar los valores que hay en las celdas de la fila `e` (Que corresponde al empleado que el usuario desea) y retornar el total de esta suma.
- **Variables requeridas:**
  - En el algoritmo principal
    - Constantes requeridas:
      - `MAX_EMPLEADOS`: constante que posee la cantidad de empleados.
      - `MAX_MESES`: constante que posee el número de meses del año.
    - Variables:
      - `matrizSueldos`: matriz que almacenará los sueldos de los 10 empleados durante los 12 meses.
      - `nominaMesM`: almacenará el cálculo de la nómina del mes `m`.
      - `nominaEmpleadoE`: almacenará el cálculo de la nómina del empleado (`e`).
      - `numeroMes`: recibe el número del mes (`m`) del que se desea calcular la nómina total.
      - `numeroEmpleado`: recibe el número del empleado (`e`) del que se quiere saber su nómina total.
  - En la función `leerDatos`
    - Parámetros:
      - `filas`: la cantidad de filas que representa los empleados.
      - `columnas`: la cantidad de columnas que representa los meses.

    - Variables locales:
        - ◇ `i`: controla el ciclo externo de las filas.
        - ◇ `j`: controla el ciclo interno de las columnas.
        - ◇ `matriz`: matriz de datos que será retornada.
- En la función `calcularNominaMesM`
    - Parámetros:
        - ◇ `matriz`: la matriz con los sueldos de los empleados.
        - ◇ `numeroMes`: el número del mes que se quiere calcular.
        - ◇ `maximoEmpleados`: cantidad de empleados.
    - Variables locales:
        - ◇ `i`: controla el ciclo.
        - ◇ `nominaMesM`: acumula la nómina del mes (`m`).
- En la función `calcularNominaEmpleadoE`
    - Parámetros:
        - ◇ `matriz`: la matriz con los salarios de los empleados.
        - ◇ `numeroEmp`: el número del empleado que se quiere calcular.
        - ◇ `maximoMeses`: cantidad de meses.
    - Variables locales:
        - ◇ `i`: controla el ciclo.
        - ◇ `nominaEmpleadoE`: acumula la nómina del empleado (`e`).
- En la función `mostrarResultados`
    - Parámetros:
        - ◇ `nominaMesM`: contiene el total de la nómina del mes analizado.
        - ◇ `nominaEmpleadoE`: contiene la nómina del empleado analizado.

De acuerdo al análisis planteado, se propone el Algoritmo 6.15.

Algoritmo 6.15: SueldosEmpleados

```
1 Algoritmo SueldosEmpleados
2   Constante Entero MAX_EMPLEADOS = 10
3   Constante Entero MAX_MESES     = 12
4   Real matrizSueldos [ ][ ],  nominaMesM, nominaEmpleadoE
5   Entero numeroMes, numeroEmpleado
```

```
6
7   matrizSueldos = leerDatos( MAX_EMPLEADOS, MAX_MESES )
8
9   imprimir( "Numero del mes que desea totalizar: " )
10  leer( numeroMes )
11
12  imprimir( "Número del empleado que desea totalizar" )
13  leer( numeroEmpleado )
14
15  nominaMesN      = calcularNominaMesM( matrizSueldos,
16                                        numeroMes,
17                                        MAX_EMPLEADOS )
18
19  nominaEmpleadoE = calcularNominaEmpleadoE( matrizSueldos,
20                                             numeroEmpleado,
21                                             MAX_MESES )
22
23  mostrarResultados( nominaMesM, nominaEmpleadoE )
24 FinAlgoritmo
25
26 Funcion Entero[ ][ ] leerDatos( Entero filas, Entero
     columnas)
27  Entero i, j,
28  Real matriz[ filas ] [ columnas ]
29
30  Para i = 1 Hasta filas Incremento 1
31    Para j = 1 Hasta columnas Incremento 1
32      imprimir( "Sueldo del empleado " , i, " mes ",j,":" )
33      leer( matriz[ i ][ j ] )
34    FinPara
35  FinPara
36
37  Retornar matriz
38 FinFuncion
39
40 Funcion Real calcularNominaMesM( Real   matriz[ ][ ],
41                                  Entero numeroMes,
42                                  Entero maximoEmpleados )
43  Entero i
44  Real nominaMesM
45
46  nominaMesM = 0
47  Para i = 1 Hasta maximoEmpleados Incremento 1
48    nominaMesM = nominaMesM + matriz[ i ][ numeroMes ]
49  FinPara
50
51  Retornar nominaMesM
52 FinFuncion
53
```

```
54 Funcion Real calcularNominaEmpleadoE( Real   matriz[ ][ ],
55                                       Entero numeroEmp,
56                                       Entero maximoMeses )
57   Entero i
58   Real nominaEmpleadoE
59
60   nominaEmpleadoE = 0
61   Para i = 1 Hasta maximoMeses Incremento 1
62     nominaEmpleadoE = nominaEmpleadoE + matriz[numeroEmp][i]
63   FinPara
64
65   Retornar nominaEmpleadoE
66 FinFuncion
67
68 Procedimiento mostrarResultados( Real nominaMesM,
69                                  Real nominaEmpleadoE )
70    imprimir( "El total nómina del mes: ", nominaMesM )
71    imprimir( "El total año empleado:   ", nominaEmpleadoE )
72 FinProcedimiento
```

Explicación del algoritmo:

En el algoritmo principal se declaran las variables como la matriz de sueldos, la nomina del mes que se requiere y la nómina del empleado que se requiere. Posteriormente se hace el llamado a la función `leerDatos()`, a la que se le envían como parámetros el número máximo de empleados y el máximo de meses que tiene la matriz de sueldos.

```
7   matrizSueldos = leerDatos( MAX_EMPLEADOS, MAX_MESES )
```

Esta función permite almacenar los 12 sueldos de los 10 empleados mediante dos ciclos `Para` anidados y retorna una matriz con los sueldos ingresados para que sea almacenada en la variable `matrizSueldos`. Recuerde que las filas de la matriz representan los empleados, mientras que las columnas representan los meses del año.

La referencia a la matriz de sueldos se envía como argumento, junto con el número del mes del que se desea conocer el valor total de la nómina y el máximo de empleados, a la función `calcularNominaMesM()` , la cual utiliza un ciclo `Para` con el objetivo de recorrer las filas de la matríz en la columna que representa al mes analizado e ir sumando los sueldos solo de ese mes; una vez hace el recorrido de las celdas que están en la columna del mes analizado, retornando el resultado de esta suma, es decir, lo pagado durante ese mes a los 10 empleados.

```
Funcion Real calcularNominaMesM( Real   matriz[ ][ ],
                                 Entero numeroMes,
                                 Entero maximoEmpleados )
  Entero i
  Real nominaMesM

  nominaMesM = 0
  Para i = 1 Hasta maximoEmpleados Incremento 1
    nominaMesM = nominaMesM + matriz[ i ][ numeroMes ]
  FinPara

  Retornar nominaMesM
FinFuncion
```

Igual sucede con la función `calcularNominaEmpleadoE()`, que recibe la referencia a la matriz de sueldos como argumento, el número de fila que representa al empleado, así como el máximo de meses, y a través de un ciclo `Para` se recorre las columnas de la matriz en la fila que representa al empleado del que se desea conocer su nómina anual, retornando la suma de los 12 sueldos del respectivo empleado.

```
Funcion Real calcularNominaEmpleadoE( Real   matriz[ ][ ],
                                      Entero numeroEmp,
                                      Entero maximoMeses )
  Entero i
  Real nominaEmpleadoE

  nominaEmpleadoE = 0
  Para i = 1 Hasta maximoMeses Incremento 1
    nominaEmpleadoE = nominaEmpleadoE + matriz[numeroEmp][i]
  FinPara

  Retornar nominaEmpleadoE
FinFuncion
```

Al finalizar, el algoritmo muestra, a través del procedimiento `mostrarResultados()` el total de la nómina pagado durante el mes especificado del año y lo pagado durante el año al empleado indicado.

.:Ejemplo 6.16. *Construya un algoritmo que permita almacenar caracteres en una matriz cuadrada de `n` filas y columnas, y que, posteriormente permita intercambiar los caracteres que se encuentran en las celdas de dos filas cualquiera. El algoritmo debe mostrar la matriz inicial y la matriz resultante luego de intercambiar los elementos de las filas. Utilice funciones y procedimientos para realizar los procesos.*

Análisis del problema:

- **Resultados esperados:** la matriz con los caracteres ingresados inicialmente, así como la matriz con los caracteres intercambiados entre dos filas diferentes indicadas por el usuario.
- **Datos disponibles:** los caracteres que se van a almacenar en la matriz original.
- **Proceso:** luego de tener una matriz cuadrada de n filas y columnas conteniendo en sus celdas caracteres, el algoritmo debe intercambiar los caracteres que están en las celdas de dos filas cualquiera y mostrar la matriz original y la resultante después del intercambio de caracteres.
- **Variables requeridas:**
  - En el algoritmo principal
    - `matrizCaracteres`: almacena los caracteres ingresados por el usuario.
    - `matrizIntercambiada`: almacena los caracteres luego de haber intercambiado las filas.
    - `fila1`: número de la primera fila a intercambiar
    - `fila2`: número de la segunda fila a intercambiar.
    - `tamanio`: corresponde al tamaño de la matriz (igual número de filas y columnas).
  - En la función `leerDatos`
    - Parámetros:
      - `tamanio`: corresponde al tamaño de la matriz.
    - Variables locales:
      - `i`: controla el ciclo externo de las filas.
      - `j`: controla el ciclo interno de las columnas.
      - `matriz`: matriz de datos que será retornada.
  - En la función `intercambiarF`
    - Parámetros:
      - `matriz`: matriz con los caracteres.
      - `tamanio`: corresponde al tamaño de la matriz.
      - `f1`: número de la primera fila a intercambiar
      - `f2`: número de la segunda fila a intercambiar.

- Variables locales:
  - ◇ `i`: controla el ciclo.
  - ◇ `aux`: variable auxiliar para almacenar datos temporalmente. (`m`).
  - ◇ `mat`: matriz auxiliar en la que se hacen los cambios. (`m`).

- En la función `mostrarResultados`
  - Parámetros:
    - ◇ `m1`: matriz original con los caracteres ingresados.
    - ◇ `m2`: matriz con las dos filas intercambiadas.
    - ◇ `t`: tamaño de las matrices, representa la cantidad de filas y columnas.

De acuerdo al análisis planteado, se propone el Algoritmo 6.16.

Algoritmo 6.16: MatrizCaracteres

```
1  Algoritmo MatrizCaracteres
2    Caracter matrizCaracteres    [ ][ ]
3    Caracter matrizintercambiada [ ][ ]
4    Entero   fila1, fila2, tamanio
5
6    imprimir( "Ingrese el tamaño de la matriz cuadrada " )
7    leer( tamanio )
8
9    matrizCaracteres = leerDatos( tamanio )
10
11   imprimir( "Ingrese la fila 1 a intercambiar " )
12   leer( fila1 )
13
14   imprimir( "Ingrese la fila 2 a intercambiar " )
15   leer( fila2 )
16
17   matrizIntercambiada = intercambiarF( matrizCaracteres,
18                                        tamanio,
19                                        fila1, fila2 )
20
21   mostrarResultados( matrizCaracteres,
22                      matrizIntercambiada, tamanio )
23 FinAlgoritmo
24
25 Funcion Caracter[ ][ ] leerDatos( Entero tamanio )
26   Entero i, j
27   Caracter matriz[ tamanio ][ tamanio ]
28
29   Para i = 1 Hasta tamanio Incremento 1
```

```
30      Para j = 1 Hasta tamanio Incremento 1
31        imprimir( "Carácter posición ", i, " " j, ":" )
32        leer( matriz[ i ][ j ] )
33      FinPara
34    FinPara
35
36    Retornar matriz
37  FinFuncion
38
39  Funcion Caracter[][] intercambiarF( Caracter  matriz[ ][ ],
40                                      Entero    tamanio,
41                                      Entero    f1, Entero f2)
42    Caracter aux, mat[ tamanio ][ tamanio ]
43    Entero i
44
45    Para i = 1 Hasta tamanio Incremento 1
46      Para j = 1 Hasta tamanio Incremento 1
47        mat [ i ][ j ] = matriz [ i ][ j ]
48      FinPara
49    FinPara
50
51    Para i = 1 Hasta tamanio Incremento 1
52      aux = matriz[ f1 ][ i ]
53      mat [ f1 ][ i ] = mat[ f2 ][ i ]
54      mat [ f2 ][ i ] = aux
55    FinPara
56
57    Retornar mat
58  FinFuncion
59
60  Procedimiento mostrarResultados( Caracter m1[ ][ ],
61                                   Caracter m2[ ][ ],
62                                   Entero   t )
63    Cadena cadenaA, cadenaB
64
65    cadenaA = ""
66    cadenaB = ""
67    Para i = 1 Hasta tamanio Incremento 1
68      Para j = 1 Hasta tamanio Incremento 1
69        cadenaA = cadenaA + m1[ i ][ j ] + " "
70        cadenaB = cadenaB + m2[ i ][ j ] + " "
71      FinPara
72      cadenaA = cadenaA + SALTO_LINEA
73      cadenaB = cadenaB + SALTO_LINEA
74    FinPara
75
76    imprimir( "Matriz Inicial ", cadenaA )
77    imprimir( "Matriz Con filas intercambiadas ", cadenaB )
78  FinProcedimiento
```

Explicación del algoritmo:

El algoritmo empieza declarando las variables y solicitándole al usuario el tamaño de la matriz que será el mismo número de filas y columnas (líneas 2 a 7); con este tamaño se inicializa la matriz dentro de la función `leerDatos()`, es por eso que el tamaño se envía a esta función como argumento (línea 9).

Dentro de esta misma función, se ingresan los caracteres a la matriz de caracteres (la inicial); esto se lleva a cabo por medio de los dos ciclos `Para` anidados, como ya se ha explicado en ejercicios anteriores (líneas de la 25 a la 37).

Luego, se solicitan al usuario las filas que se van a intercambiar y, con esta información se llama a la función `intercambiarF()` (líneas de la 11 a la 19) a la que se envían la matriz original, el tamaño y las filas que se van a intercambiar.

Ya dentro de esta función, se utiliza una variable suscrita denominada `mat` que recibe los datos de la matriz y, así no modificar la matriz original, y una variable `aux` que servirá para almacenar temporalmente los caracteres que están en la primera fila que se va a intercambiar y estos no se vayan a perder; a continuación, los caracteres de la segunda fila a intercambiar se pasan a la primera fila y los que estaban en la variable auxiliar (es decir, los que se copiaron inicialmente) se ubican en la segunda fila de la matriz que se está intercambiando. Todo este cambio se realiza con un ciclo `Para` que recorre las columnas de las filas que se van a intercambiar, por eso, la variable `i`, representa la columna en que se va ubicando el control del algoritmo para leer el carácter. De esta forma los caracteres que estaban en las dos filas son intercambiados. Al finalizar esta función, se retorna la variable `mat`.

```
39 Funcion Caracter[][] intercambiarF( Caracter  matriz[ ][ ],
40                                     Entero    tamanio,
41                                     Entero    f1, Entero f2)
42   Caracter aux, mat[ tamanio ][ tamanio ]
43   Entero i
44
45   Para i = 1 Hasta tamanio Incremento 1
46     Para j = 1 Hasta tamanio Incremento 1
47       mat [ i ][ j ] = matriz [ i ][ j ]
48     FinPara
49   FinPara
50
51   Para i = 1 Hasta tamanio Incremento 1
```

```
52      aux = matriz[ f1 ][ i ]
53      mat [ f1 ][ i ] = mat[ f2 ][ i ]
54      mat [ f2 ][ i ] = aux
55    FinPara
56
57    Retornar mat
58  FinFuncion
```

Como en los anteriores algoritmos, se utiliza el procedimiento `mostrarResultados()` para mostrar al usuario las matrices, la que contiene los caracteres iniciales ingresados por el usuario y la que tiene sus filas intercambiadas. En este procedimiento se utilizan variables de tipo cadena que almacenan todos los caracteres y los muestran con la instrucción imprimir, de tal modo que es posible mostrar los datos de las matrices como tablas.

.:Ejemplo 6.17. *Construya un algoritmo que almacene números enteros en una matriz cuadrada de orden* `n` *y que a través de funciones y procedimientos determine si la suma de los elementos de la diagonal principal es igual, mayor o menor a la suma de los elementos de la diagonal secundaria.*

A través de este algoritmo se exploran los conceptos de diagonales en las matrices. La diagonal principal es aquella que se forma empezando en la primera celda de la matriz (fila 1, columna 1) y termina en la última celda (fila n, columna n) de la misma.

$$diagonalPrincipal = \begin{bmatrix} a_{11} & & & \\ & a_{22} & & \\ & & \cdots & \\ & & & a_{nn} \end{bmatrix}$$

Por tanto su suma se define como:

$$sumaDiagonalPrincipal = \sum_{i=1}^{n} a_{ii}$$

Por su parte, la diagonal secundaria es la que inicia en la última celda de la primera fila (fila 1, columna n) y termina en la última fila de la primera columna (fila n, columna 1). Si bien es posible encontrar diagonales en matrices no cuadradas, en este ejercicio se trabaja con una matriz cuadrada, que es aquella donde el número de filas es igual al número de columnas.

$$diagonalSecundaria = \begin{bmatrix} & & & a_{1n} \\ & & a_{2(n-1)} & \\ & \cdots & & \\ a_{n1} & & & \end{bmatrix}$$

La suma de los elementos de la diagonal secundaria es:

$$sumaDiagonalSecundaria = \sum_{i=1}^{n} a_{i(n-i)}$$

Análisis del problema:

- **Resultados esperados:** informar como resultado si la suma de los elementos de la diagonal principal de una matriz es igual, mayor o menor a la suma de los elementos de la diagonal secundaria.
- **Datos disponibles:** estarán disponibles los números enteros que serán almacenados en cada una de las celdas de la matriz y el tamaño que se le va a dar a esta.
- **Proceso:** luego de ingresarse los números enteros a la matriz, este algoritmo deberá recorrer tanto la diagonal principal de la misma como su diagonal secundaria y sumar los elementos que están presentes en estas diagonales, para después comparar las sumas obtenidas y determinar si son iguales o alguna de las dos es mayor.
- **Variables requeridas:**
  - En el algoritmo principal
    - `matriz`: matriz que almacenará los números.
    - `sumaDiagonalPrincipal`: almacena la suma de la diagonal principal de la matriz.
    - `sumaDiagonalSecundaria`: almacena la suma de la diagonal secundaria de la matriz.
    - `n`: corresponde al tamaño de la matriz.
  - En la función `leerDatos`
    - Parámetros:
      - `tamanio`: corresponde al tamaño de la matriz.
    - Variables locales:
      - `i`: controla el ciclo externo de las filas.

        - ◇ `j`: controla el ciclo interno de las columnas.
        - ◇ `matriz`: matriz de datos que será retornada.
    - En la función `sumarDiagPrincipal`
        - Parámetros:
            - ◇ `matriz`: la matriz de números.
            - ◇ `tamanio`: corresponde al tamaño de la matriz.
        - Variables locales:
            - ◇ `i`: controla el ciclo.
            - ◇ `suma`: suma de los elementos de la diagonal principal.
    - En la función `sumarDiagSecundaria`
        - Parámetros:
            - ◇ `matriz`: la matriz de números.
            - ◇ `tamanio`: corresponde al tamaño de la matriz.
        - Variables locales:
            - ◇ `columna`: indica la columna.
            - ◇ `i`: controla el ciclo.
            - ◇ `suma`: suma de los elementos de la diagonal secundaria.
    - En el procedimiento `mostrarResultados`
        - Parámetros:
            - ◇ `sumaDiagonalP`: contiene la suma de los elementos de la diagonal principal.
            - ◇ `sumaDiagonalS`: contiene la suma de los elementos de la diagonal secundaria.

De acuerdo al análisis planteado, se propone el Algoritmo 6.17.

Algoritmo 6.17: DiagonalesMatriz

```
1  Algoritmo DiagonalesMatriz
2    Entero matriz [ ][ ]
3    Entero sumaDiagonalPrincipal, sumaDiagonalSecundaria, n
4
5    imprimir( "Ingrese el tamaño de la matriz " )
6    leer( n )
7
8    matriz = leerDatos( n )
9
10   sumaDiagonalPrincipal  = sumarDiagPrincipal ( matriz, n )
11   sumaDiagonalSecundaria = sumarDiagSecundaria( matriz, n )
12
13   mostrarResultados( sumaDiagonalPrincipal,
```

```
14                        sumaDiagonalSecundaria )
15 FinAlgoritmo
16
17 Funcion Entero[ ][ ] leerDatos( Entero tamanio )
18   Entero i, j, matriz [ tamanio ][ tamanio ]
19
20   Para i = 1 Hasta tamanio Incremento 1
21     Para j = 1 Hasta tamanio Incremento 1
22       imprimir( "Número en posición " , i, " ", j, " :")
23       leer( matriz[ i ][ j ] )
24     FinPara
25   FinPara
26
27   Retornar matriz
28 FinFuncion
29
30 Funcion Entero sumarDiagPrincipal( Entero matriz[ ][ ],
31                                     Entero tamanio )
32   Entero suma, i
33
34   suma = 0
35   Para i = 1 Hasta tamanio Incremento 1
36     suma = suma + matriz[ i ][ i ]
37   FinPara
38
39   Retornar suma
40 FinFuncion
41
42 Funcion Entero sumarDiagSecundaria( Entero matriz[ ][ ],
43                                      Entero tamanio )
44   Entero suma, i, columna
45
46   columna = tamanio
47   suma = 0
48   Para i = 1 Hasta tamanio Incremento 1
49     suma = suma + matriz[ i ][ columna ]
50     columna = columna - 1
51   FinPara
52
53   Retornar suma
54 FinFuncion
55
56 Procedimiento mostrarResultados( Entero sumaDiagonalP,
57                                   Entero sumaDiagonalS )
58
59
60   Si( sumaDiagonalP == sumaDiagonalS ) Entonces
61     imprimir( "Las sumas de las diagonales son iguales " )
62   SiNo
```

```
63      Si( sumaDiagonalP > sumaDiagonalS ) Entonces
64        imprimir( "La suma de la diagonal principal es mayor")
65      SiNo
66        imprimir( "La suma de la diagonal principal es menor")
67      FinSi
68    FinSi
69  FinProcedimiento
```

Explicación del algoritmo:

Luego de declarar las variables en el algoritmo principal, se solicita al usuario ingresar el tamaño de la matriz, que permitirá determinar tanto la cantidad de filas como de columnas ya que el ejercicio requiere de una matriz cuadrada (líneas de la 2 a la 6).

Con la función `leerDatos()`, el algoritmo solicita los datos utilizando dos ciclos Para anidados que recorrerán las filas y las columnas (línea 8 y líneas de la 17 a la 25).

La matriz ya cargada con datos es enviada a la función `sumarDiagPrincipal()` que mediante un ciclo `Para` recorre la diagonal principal pasando por las celdas donde el número de la fila es igual al número de la columna, por eso, en el ciclo la variable `i` especifica tanto fila como columna. Los valores de estas celdas se van acumulando en la variable suma que es retornada al algoritmo principal al final de la función.

```
30 Funcion Entero sumarDiagPrincipal( Entero matriz[ ][ ],
31                                    Entero tamanio )
32   Entero suma, i
33
34   suma = 0
35   Para i = 1 Hasta tamanio Incremento 1
36     suma = suma + matriz[ i ][ i ]
37   FinPara
38
39   Retornar suma
40 FinFuncion
```

Algo similar sucede con la función `sumarDiagSecundaria()`, pero en este caso, el recorrido mediante el ciclo `Para` se hace desde la primera fila con la variable `i` y la última columna con la variable `columna`, que previamente ha tomado el valor del parámetro `tamanio`. Dentro del ciclo, la variable `i` se va incrementando para pasar a las siguientes filas en cada iteración, mientras que la expresión `columna=columna - 1` va decrementando esta variable para ubicarse en una columna anterior.

De esta manera, se van recorriendo las celdas que están ubicadas en la diagonal secundaria; los valores almacenados en estas celdas se acumulan en la variable suma que será lo que retornará la función.

```
42 Funcion Entero sumarDiagSecundaria( Entero matriz[ ][ ],
43                                     Entero tamanio )
44   Entero suma, i, columna
45
46   columna = tamanio
47   suma = 0
48   Para i = 1 Hasta tamanio Incremento 1
49     suma = suma + matriz[ i ][ columna ]
50     columna = columna - 1
51   FinPara
52
53   Retornar suma
54 FinFuncion
```

Los resultados de las dos funciones anteriores se enviarán al procedimiento `mostrarResultados()` con el fin de que este determine y muestre si las sumas son iguales o alguna es mayor (líneas 13 y 14, así como las líneas 56 a la 69).

6.3. Ejercicios propuestos

Para los siguientes ejercicios propuestos, utilice funciones y procedimientos en su solución.

1. Escriba un algoritmo en pseudocódigo que permita almacenar en arreglos diferentes los nombres, los géneros y las edades de un grupo de n personas. El algoritmo debe determinar:

   a) Cuantas personas son de género masculino

   b) Cuántas personas de género femenino superan la mayoría de edad

   c) Cuál es el promedio de edad de las personas de género masculino

   d) Cuál es el nombre de la persona de género femenino mas pequeña.

2. Diseñe un algoritmo que permita ingresar y almacenar los nombres y las estaturas de un grupo de n personas y que muestre como salida los nombres y las estaturas de las personas ordenadas:

   a) Ascendentemente, es decir, de la más baja a la más alta

   b) Descendentemente, es decir, de la más alta a la más baja

3. Construya un algoritmo que almacene n número enteros en un arreglo y que posteriormente determine si un número cualquiera ingresado por el usuario se encuentra o no en el arreglo. Si el número se encuentra en el arreglo, el algoritmo debe decir en qué posición está.

4. Escriba un algoritmo en pseudocódigo que almacene en un arreglo n números enteros entre 50 y 100. El algoritmo deberá encontrar los números pares y los impares que están almacenados en este arreglo y almacenarlos en otros arreglos por separado para, posteriormente mostrarlos al usuario.

5. El profesor de la materia de "Lenguaje de Programación" requiere de un algoritmo que le permita almacenar en un arreglo los nombres de los 30 estudiantes del curso y en una matriz, las 5 notas obtenidas por cada uno de los estudiantes durante el semestre. El algoritmo debe realizar las siguientes operaciones:

   a) Determinar la nota definitiva de cada uno de los estudiantes que se calcula como la media aritmética de las 5 notas obtenidas. Las notas definitivas de los estudiantes deberán ser almacenadas en un arreglo.

   b) Encontrar el estudiante que obtuvo la mayor nota definitiva.

   c) Obtener el nombre de los estudiantes (y almacenarlos en un arreglo) que perdieron la materia y deberán repetirla. Un estudiante pierde la materia si su nota definitiva es inferior a 2.0. (Dos punto cero).

   d) Obtener el nombre de los estudiantes (y almacenarlos en un arreglo) que tendrán que habilitar la materia. Un estudiante puede habilitar la materia si su nota definitiva es inferior a 3.0. pero superior a 2.0, esto es, que su nota definitiva se encuentre entre 2.0 y 2.99.

   e) Determinar cuántos estudiantes ganaron la materia.

6. En una clínica de control al sobrepeso requieren almacenar los nombres y los pesos tomados durante un periodo de tiempo a un grupo de n pacientes. Durante un mes, cada paciente es pesado 3 veces (una pesada inicial, una intermedia y una pesada final) con el fin de determinar su evolución durante ese periodo. Construya

un algoritmo que permita almacenar estos datos y que además encuentre:

a) Cuanto peso ha ganado o perdido cada paciente con respecto al peso inicial.

b) Cuantos pacientes perdieron peso entre la pesada inicial y la pesada intermedia.

c) Suponga que también se tiene otro arreglo en el que se almacena por cada paciente su objetivo: ganar o perder peso durante el periodo. Determine cuantos pacientes alcanzaron el objetivo.

7. Diseñe un algoritmo que almacene números enteros en una matriz de 5 filas por 5 columnas y que luego la recorra (mostrando los números almacenados) desde la última celda (fila 5 columna 5) hasta la primera celda (fila1 columna 1) recorriendo primero las celdas de cada columna.

8. Escriba un algoritmo en pseudocódigo que almacene caracteres en una matriz de 4 filas por 5 columnas y que posteriormente, halle y muestre la matriz transpuesta de la matriz originalmente ingresada. La matriz transpuesta es una matriz que tiene como filas, las columnas de otra matriz.

9. Construya un algoritmo en pseudocódigo que almacene unos (1) en la primera y última fila y en la primera y última columna de una matriz cuadrada de tamaño 5, es decir, en las filas y columnas exteriores de la matriz. El algoritmo debe almacenar ceros (0) en las demás celdas de la matriz, o sea, en las celdas que se encuentran al interior de la matriz.

10. Elabore un algoritmo que permita realizar la multiplicación de dos matrices. Para poder realizar una multiplicación de matrices, en primer lugar, estas deben almacenar números y se debe tener en cuenta que, desde el punto de vista algebraico, solo es posible multiplicar dos matrices si el número de columnas de la primera matriz es igual al número de filas de la segunda matriz y que la matriz resultante tendrá el mismo número de filas que la primera matriz e igual número de columnas de la segunda matriz.

Bibliografía

[Areitio and Areitio, 2009] Areitio, G. and Areitio, A. (2009). *Información, Informática e Internet: del ordenador personal a la Empresa 2.0*. Editorial Visión Libros.

[Bishop et al., 2004] Bishop, A. J., Font, T. C., Marti, L. C., de Pablo, L. F., Rodríguez, J. G., Fernández-Aliseda, A., Azcárate, A. G., Farnés, M. M., SANCHEZ, J. A. M., Martín, J. A. H., et al. (2004). *Matemáticas re-creativas*, volume 29. Grao.

[Boullosa, 2016] Boullosa, P. (2016). Ada Lovelace. https://www.youtube.com/watch?v=QcvuMFveUqk. Tv Azteca [Online; consultado Junio-2017].

[Corona and Ancona, 2011] Corona, M. A. N. and Ancona, M. V. (2011). *Diseño de algoritmos y su codificación en lenguaje C.* McGraw-Hill Interamericana.

[Echeverri and Orrego, 2012] Echeverri, J. A. A. and Orrego, G. A. V. (2012). *Programación, Teoría y aplicaciones*. Universidad de Medellín.

[Jiménez et al., 2016] Jiménez, J. A. M., Jiménez, E. M. H., and Alvarado, L. N. Z. (2016). *Fundamentos de Programación, Diagramas de flujo, Diagramas N-S, Pseudocódigo y Java*. Alfaomega.

[Joyanes A., 1996] Joyanes A., L. (1996). *Problemas de la metodología de la Programación*. McGraw-Hill / Interamericana de España, S.A.

[López, 2015] López, J. A. (2015). *Grandes genios de la historia en 25 historias*. Penguin Random House Grupo Editorial España.

[López, 2009] López, R. (2009). *Metodología de la programación orientada a objetos*. AlfaOmega.

[Mancilla et al., 2016] Mancilla, A. H., Ebratt, R. G., and Capacho, J. P. (2016). *Diseño y construcción de algoritmos*. Universidad del Norte – Editorial.

[Oviedo, 2013] Oviedo (2013). Historia de la informática. https://www.youtube.com/watch?v=6sTPEtzNIsA&t=15s. Universidad Oviedo. [Online; consultado Junio-2017].

[Pimiento, 2009] Pimiento, W. M. C. (2009). *Fundamentos de Lógica para Programación de Computadores*. Universidad Piloto de Colombia.

[Szymanczyk, 2013] Szymanczyk, O. (2013). *Historia de las telecomunicaciones mundiales*. Editorial Dunken.

[Trejos, 2004] Trejos, O. (2004). *La esencia de la lógica de programación*. Ediciones de la U.

[Trejos, 2017] Trejos, O. I. B. (2017). *Lógica de Programación*. Ediciones de la U.

Índice alfabético

Acumulador, 180
Algoritmo, 45
  Clasificación, 46
  Representación, 47
    Diagrama de flujo, 49
    Narrada, 47
    pseudocódigo, 61
Arreglos, 351

Bandera, 181
Bucle, 179
Buenas prácticas, 9, 52, 57, 60, 192, 197, 205, 208, 224, 326, 347, 356

Centinela, 181
Ciclo
  Anidado, 217
Ciclo infinito, 183
Ciclos, 179–321
  Acumulador, 180
  Bandera, 181
  Centinela, 181
  Contador, 179
  Haga-MientrasQue, 228
  Mientras-FinMientras, 182
  Para-FinPara, 270
Circunflejo, 31
Constantes, 30
Contador, 179

Dato, 22–24
  Alfanumérico, 23
    Cadena, 23
    Carácter, 23
  Lógico, 24
  Numérico, 23
    Entero, 23
    Real, 23
Decisión compuesta, 130
Decisión simple, 125
Decisiones, 121–175
  Árbol de decisión, 122
  Anidadas, 142
  Compuesta, 121
  Múltiples, 160
Diagrama de flujo, 49
  Conector, 58, 59
  Decisión, 54, 57
  Entrada, 53
  Proceso, 53
  Símbolos, 49
  Salida, 53
  Terminal, 52

Ejercicios propuestos
  Ciclos, 316
  Decisiones, 174
  Funciones, 347
  Fundamentos, 73
  Matrices, 440
  Secuencial, 115
  Vectores, 440

Esctructura secuencial
Estructura, 81
Expresiones, 30–45
Aritméticas, 34
Conversión, 38
Lógicas, 40
Relacionales, 40

Funciones, 338–347

Haga-MientrasQue, 228–270
Historía, 13–22
Ada Lovelace, 15
Al-Juarismi, 13
Al-Khwārizmī, 13
Alan Mathison Turing, 15
Apple, 19–21
Augusta Ada Byron, 15
Betty Snyder Holberton, 17
Bilas Spence, 17
Bill Gates, 19
Blaise Pascal, 14, 18
Charles Babbage, 14
Colossus, 16
Euclides, 13
George Boole, 15
Google, 22
Gottfried Wilheim Von Leibnitz, 14
Grace Murray Hooper, 17
Herman Hollerith, 15
Howard Aiken, 16
IBM, 15, 16, 19
Jack Kilby, 18
Jean Jennings Bartik, 17
John Napier, 14
John Von Neumann, 17
Joseph Jacquard, 14
Kathleen McNulty Mauchly Antonelli, 17
Konrad Zuse, 16, 17
Lab.BELL, 18
Lab.Bell, 17
Leonardo Da Vinci, 13
Leonardo de Pisa, 13
Leonardo Fibonacci, 13
Linus Torvalds, 20
Marlyn Wescoff M., 17
Microsoft, 20, 21
Paul Allen, 19
Plankalkül, 17
Ray Tomlinson, 19
Ruth Linchterman Teitelbaum, 17
Tommy Flowers, 16
U.Cambridge, 17
U.Stanford, 20
Von Neumann, 17
Wilhelm Schickard, 14
Xerox, 19
Yahoo, 20

Identificadores, 24

Lenguaje
Ada, 19
Algol 58, 18
BASIC, 18
Basic, 20
C, 18, 19
C++, 18, 20
C#, 21
COBOL, 18
Ensamblador, 17
FORTRAN, 18
Google, 21
Java, 20
LISP, 18
Orientado a objetos, 19
Pascal, 18
PHP, 21
PL/1, 18
Prolog, 19
Python, 20

Smalltalk, 19
Visual Basic, 20

Matrices, 408–440
Mientras-FinMientras, 182–228

Notación algoritmica, 38

Operadores, 30–45
Aritméticos, 31
Lógicos, 33
Precedencia, 43
Prioridad, 43
Relacionales, 32

Para-FinPara, 270–316
Precondición, 197
Problema
Estrategia de solución, 66
Procedimientos, 325–338
Programación secuencial, 81–117
Pseudocódigo, 61
pseudocódigo
Comentarios, 64
Forma general, 64
Palabras reservadas, 61
Solución
Datos conocidos, 69
Proceso, 69
Resultado esperado, 69
Variables, 69

Salto de linea, 287

Tabla de verificación, 223

Variables, 25–30
Asignación, 28
Declaración, 26
Vectores, 351–408

El desarrollo del pensamiento computacional y los avances tecnológicos, han fomentado en gran medida el uso de la lógica de programación en la creación de diversas clases de algoritmos para la solución de problemas, reflejando la necesidad de su estudio. Por esta razón los autores decidieron plasmar en este libro sus conocimientos, producto de la experiencia docente en instituciones de educación superior en esta área durante varios años.

Este texto se ha creado con el propósito de facilitar el estudio de la lógica de programación, brindando los conceptos fundamentales en cada uno de los temas tratados. Para hacer esto posible, se ha desarrollado una variedad de ejemplos de algoritmos expresados en pseudocódigo y en algunos casos mediante diagramas de flujo, con lo cual se espera que el lector desarrolle las competencias básicas para la solución de problemas computacionales.

El libro está compuesto por 6 capítulos, en los cuales se van abordando los diferentes temas con un enfoque progresivo, partiendo desde lo más sencillo hasta alcanzar aspectos de mayor complejidad. Se inicia con los fundamentos de la lógica de programación, para luego abordar las estructuras de tipo secuencial, de decisión y de repetición; seguidamente se analizan los procedimientos junto con las funciones y se finaliza con el estudio de vectores y matrices.

9 789588 801650

Elizcom s.a.s

ViewModel APIs cheat sheet

v1.0 by @AndroidDev

This cheat sheet lists the ViewModel APIs in Jetpack. It includes the artifact where they can be found, the scope of the returned ViewModel instance, and an example of how to use them.

You can scope ViewModel instances to a `ViewModelStoreOwner`. The owner can be an Activity, Fragment, Navigation graph, a destination of a Navigation graph, or your own custom owner.

For more use cases and to learn about ViewModel factories: **goo.gle/architecture-viewmodel**

The APIs and artifacts listed in this page follow this template

Library - `artifact`

`ViewModel API` — Scope of the returned ViewModel instance

```
Example code
```

Get an instance in **Jetpack Compose**

Compose - `lifecycle.lifecycle-viewmodel-compose`

`= viewModel()` — closest `ViewModelStoreOwner`

```
@Composable
fun MyScreen(vm: ViewModel = viewModel()) { ... }
```

`= viewModel(vmStoreOwner)` — any `ViewModelStoreOwner`

```
// Example of getting a ParentViewModel instance scoped to the Navigation graph
composable("myScreen") { backStackEntry ->
  val parentEntry = remember(backStackEntry) {
    navController.getBackStackEntry("parentNavigationRoute")
  }
  val parentViewModel = viewModel<ParentViewModel>(parentEntry)
  ...
}
```

Compose + Navigation + Hilt - `hilt.hilt-navigation-compose`

When using Hilt and Navigation together, use this API instead of `viewModel()`

`= hiltViewModel()` — closest `ViewModelStoreOwner`

```
@Composable
fun MyScreen(vm: ViewModel = hiltViewModel()) { ... }
```

Core ViewModel API

ViewModel - `lifecycle:lifecycle-viewmodel-ktx`

`ViewModelProvider` — any `ViewModelStoreOwner`

```
val vm = ViewModelProvider(anyViewModelStoreOwner)[MyViewModel::class.java]
```

Get an instance in an **Activity**

Activity - `activity.activity-ktx`

`by viewModels()` — Activity (`ViewModelStoreOwner`)

```
val vm: MyViewModel by viewModels()
```

Get an instance in a **Fragment**

Fragment - `fragment.fragment-ktx`

`by viewModels()` — Fragment (`ViewModelStoreOwner`)

```
val vm: MyViewModel by viewModels()
```

`by viewModels(ownerProducer)` — any `ViewModelStoreOwner`

```
// Example of getting a SharedViewModel instance scoped to the parent Fragment
val vm: SharedViewModel by viewModels(ownerProducer = { requireParentFragment() })
```

`by activityViewModels()` — Activity (`ViewModelStoreOwner`)

```
val vm: SharedViewModel by activityViewModels()
```

Fragment + Navigation - `navigation.navigation-fragment`

`by navGraphViewModels(graphId)` — NavGraph (`ViewModelStoreOwner`)

```
val vm: SharedViewModel by navGraphViewModels(R.id.nav_graph)
```

Fragment + Navigation + Hilt - `hilt.hilt-navigation-fragment`

When using Hilt and Navigation together, use this API instead of `navGraphViewModels()`. To get an instance not scoped to a NavGraph, you can keep using `viewModels()` or `activityViewModels()`.

`by hiltNavGraphViewModels(graphId)` — NavGraph (`ViewModelStoreOwner`)

```
val vm: SharedViewModel by hiltNavGraphViewModels(R.id.nav_graph)
```

Actividad

Analizar los problemas e identificar las entradas, proceso y salidas en una tabla como se muestran en los siguientes ejemplos

Problema: Preparar una salchipapa		
Entradas	**Proceso**	**Salidas**
• Papas • Hot dog • Aceite • Mayonesa • Mostaza • Sal	Paso 01: Pelar las papas Paso 02: lavar las papas Paso 03: Picar las papas Paso 04: Prender la cocina Paso 05: Poner sarten Paso 06: Poner aceite al sarten Paso 07: Esperar q que caliente el aceite	Salchipapa 

Problema: Mostrar la suma 3 números		
Entradas	**Proceso**	**Salidas**
• Numero1 • Numero2 • Numero3	Numero1 + Numero2 numero3	Suma

Problemas

1. Calcular el área de un rectángulo.

2. Un estacionamiento requiere determinar el cobro que debe aplicar a las personas que lo utilizan. Considere que el cobro es con base en las horas que lo disponen y que las fracciones de hora se toman como completas

3. La compañía de luz y sombras (CLS) requiere determinar el pago que debe realizar una persona por el consumo de energía eléctrica, la cual se mide en kilowatts (KW).

4. Se requiere determinar el costo que tendrá realizar una llamada telefónica con base en el tiempo que dura la llamada y en el costo por minuto.

5. Una tienda ofrece un descuento del 15% sobre el total de la compra y un cliente desea saber cuánto deberá pagar finalmente por su compra.

6. Un alumno desea saber cuál será su calificación final en la materia de Algoritmos. Dicha calificación se compone de los siguientes porcentajes:

    - 55% del promedio de sus tres calificaciones parciales.
    - 30% de la calificación del examen final.
    - 15% de la calificación de un trabajo final

7. Un maestro desea saber qué porcentaje de hombres y que porcentaje de mujeres hay en un grupo de estudiantes

8. Un alumno necesita calcular el promedio de sus cinco notas, cada una tiene una ponderación distinta.

9. Se necesita calcular el sueldo de un asesor, este trabajó tres meses en la empresa.

10. En un terminal de buses dos buses con destino a la ciudad de Valdivia, transportan a 35 pasajeros cada uno, el valor del pasaje fue de $5.500. ¿Cuál es la total obtenido?

GUÍA DE USO

La presente guía de uso pretende orientar al alumno en el cómo resolver cada uno de los problemas que se presentan en este compendio de problemas describiendo las partes que componen cada ejercicio y ejemplificando un programa.

El cuadro que deberá ser completado por los alumnos es el siguiente (ver Figura 1):

<table>
<tr><th>Definición de Variables</th><th>Pseudocódigo</th><th>Diagrama de Flujo de Datos</th></tr>
<tr><td>
<table>
<tr><th></th><th>NOMBRE</th><th>TIPO</th><th>DESCRIPCIÓN</th></tr>
<tr><td>E</td><td></td><td></td><td></td></tr>
<tr><td>P</td><td></td><td>PARTE 1</td><td></td></tr>
<tr><td>S</td><td></td><td></td><td></td></tr>
</table>
Algoritmo

PARTE 2

Prueba de Escritorio

PARTE 5
</td><td>PARTE 3</td><td>PARTE 4</td></tr>
</table>

Figura 1. Formato para solución de problemas.

DESCRIPCIÓN DEL FORMATO

PARTE 1. Definición de Variables:

La definición de variables se compone por las partes:

- Nombre
- Tipo
- Descripción
- Renglones con letras E, P y S

Las cuales se muestran y describen a continuación (ver Figura 2):

	NOMBRE	TIPO	DESCRIPCION
E			
P			
S			

Figura 2. Apartado para Definición de Variables

Donde
:

Nombre: En esta columna se deben diferencias entre variables de entrada proceso y salida. Se debe escribir el nombre que se le dará a la

variable, el cual no deberá contener caracteres especiales (#,$,*,etc), ni espacios.

Tipo: En esta columna se especifica el tipo de datos que puede tomar la variable (entero-E, real-R, carácter-C, lógico-L, cadena-S)

Descripción: Se debe escribir lo que representa la variables, por ejemplo si representa un área, edad, nombre, etc.

E, P y S: Representan las variables de entrada-E, proceso-P y salida-S.

PARTE 2. Algoritmo

El algoritmo es una descripción en español (común) de la solución del problema, marcando los pasos a seguir (ver figura 3).

Algoritmo

Figura 3. Apartado para Algoritmo

PARTE 3.
Pseudocódigo

Es una serie de instrucciones en español estructurado entre las que se encuentran:

Tipo de Instrucciones	**Instrucción a utilizar**
Inicio y fin del programa	Inicio Termina
Instrucción de entrada	Leer
Instrucción de salida	Escribe
Operadores aritméticos	+, -, *, /, mod (cálculo de residuo), div (división entera), sqrt (raíz cuadrada), ** (cálculo de potencias)
Operadores lógicos	Y O No
Operadores relacionales	<, >, <=, >=, =
Asignación	←
Concatenación	+

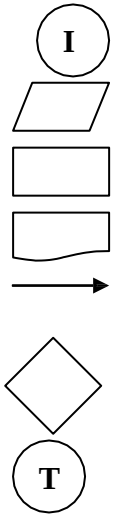
Tabla 1. Instrucciones Elementales

Pseudocódigo
Inicio **Bloque de instrucciones** **Termina**

Figura 4. Apartado para Pseudocódigo

PARTE 4. Diagrama de Flujo de Datos

Se utiliza la siguiente nomenclatura:



Inicia / Comienzo

Lectura de datos

Proceso

Escribir / Imprimir

Dirección del flujo

Decisión

Termina

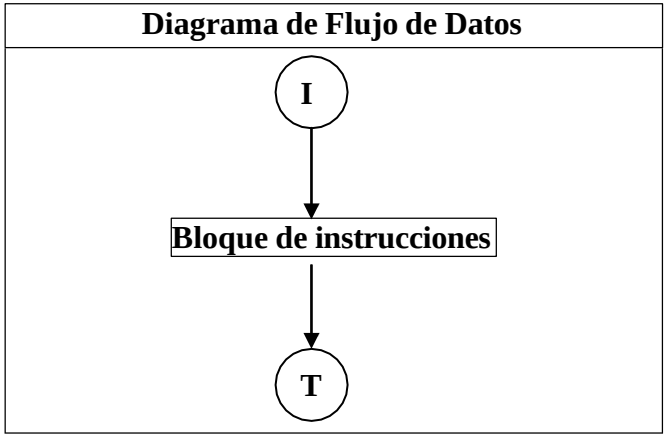


Figura 5. Apartado para Diagrama de Flujo de Datos

PARTE 5. Prueba de Escritorio

En este apartado se deben listar las variables definidas y dar calores para comprobar las instrucciones (ver Figura 6).

Prueba de Escritorio

Figura 6. Apartado para Prueba de Escritorio

Para mayor entendimiento se presenta a continuación un ejercicio resuelto.

Escribir un programa que calcule en área de un triángulo dadas la base y la altura.

Definición de Variables

	NOMBRE	TIPO	DESCRIPCIÓN
E	*B* *A*	*Entero* *Entero*	B, representa la base del triángulo A representa la altura del triángulo
P	PARTE 1		
S	*area*	*Real*	area representa el área del triángulo

Algoritmo

PARTE 2

1. Pedir base y altura del triángulo
2. Obtener la base y altura
3. Calcular el área
4. Mostrar el resultado

Prueba de Escritorio

A 5
B 8
Area 20

PARTE 5

Pseudocódigo

PARTE 3

Inicio

Escribe (“Teclea la base y altura”)

Leer A, B

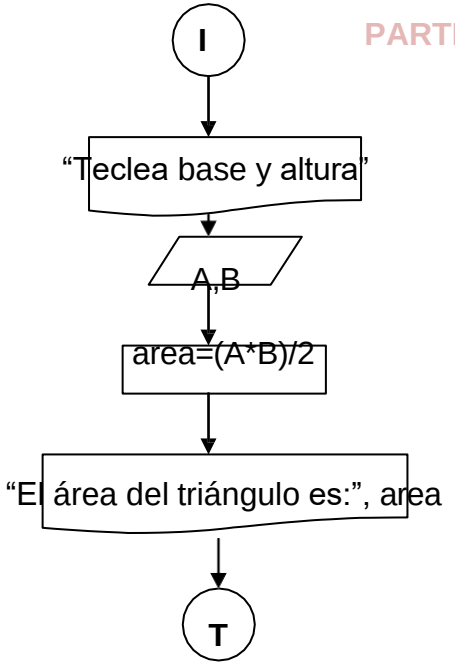
area=(A*B)/2

Escribe (“El área del triángulo es:”, area)

Termina

Diagrama de Flujo de Datos

PARTE 4



Donde por columnas en la Parte 1:

- Los nombres de las variables son:
  B: ubicada en el renglón de E (variables de entrada)
  A: ubicada en el renglón de E (variable de entrada) area: ubicada en el renglón S (variable de salida)
- El tipo de variables corresponde a:
  E la primera variable declarada (B) es de tipo entero. E la segunda variable declarada (A) es de tipo entero. R la tercera variable declarada (area) es de tipo real.

- Descripción:
  En el renglón correspondiente a cada variable, se describe lo que representa en el problema, es decir, B, representa la base del triángulo, A representa la altura del triángulo y area representa el área del triángulo.

PARTE 2:

Se plantearon cuatro pasos utilizando español común.

PARTE 3.

Es el planteamiento de pseudocódigo y se utilizan las instrucciones mencionadas en la Tabla 1.

PARTE 4.

Es el planteamiento de diagrama de flujo de datos y se utilizan las figuras mencionadas antes mencionadas en la página 11

PARTE 5.

Se realiza un listado de variables definidas y se realiza un seguimiento de las instrucciones en el pseudocódigo y/o diagrama de flujo de datos.

EJERCICIOS

Realiza el análisis correspondiente y completa la tabla que se presenta después de cada caso.

1. Escribir un programa para calcular el cuadrado y el cubo de un número entero.

<table>
<tr><th>Definición de Variables</th><th>Pseudocódigo</th><th>Diagrama de Flujo de Datos</th></tr>
<tr><td>
<table>
<tr><th></th><th>NOMBRE</th><th>TIPO</th><th>DESCRIPCIÓN</th></tr>
<tr><td>E</td><td>num</td><td>Entero</td><td>Numero para calcularle el cuadrado y el cubo</td></tr>
<tr><td>P</td><td></td><td></td><td></td></tr>
<tr><td>S</td><td>cuadrado
cubo</td><td>Entero
Entero</td><td>resultado del cuadrado
resultado del cubo</td></tr>
</table>
Algoritmo
1. Pedir el numero
2. Calcular el cuadrado del número dado
3. Calcular el cubo del número dado
4. Mostrar el resultado
</td><td>
Inicio
Entero num, cuadrado, cubo
cuadrado = 0
cubo = 0
Escribe (“Digite un numero entero”)
Leer num
cuadrado = num*num
cubo = num*num*num

Escribe (“El cuadrado del número es:”, cuadrado)
Escribe (“El cubo del número es:”, cubo)
Termina
</td><td></td></tr>
</table>

Prueba de Escritorio		

2. Escribir un programa para calcular el perímetro de una circunferencia y el área del círculo cuyo radio se lee por el teclado.

Definición de Variables

	NOMBRE	TIPO	DESCRIPCIÓN
E			
P			
S			

Algoritmo

Prueba de Escritorio

Pseudocódigo

Diagrama de Flujo de Datos

3. Escribir un programa que lea dos números enteros y calcule el residuo y el cociente entero de uno entre otro.

Definición de Variables	Pseudocódigo	Diagrama de Flujo de Datos

	NOMBRE	TIPO	DESCRIPCIÓN
E			
P			
S			

Algoritmo

Prueba de Escritorio

4. Escribir un programa que calcule el área y perímetro de un pentágono dados el valor de un lado y la apotema.

Definición de Variables

	NOMBRE	TIPO	DESCRIPCIÓN
E			
P			
S			

Algoritmo

Prueba de Escritorio

Pseudocódigo

Diagrama de Flujo de Datos

5. Escribir un programa que calcule la media de cinco valores. La media es la medida estadística más común. La media (y) de una muestra se define como la suma de los datos individuales (yi) dividido por el número de puntos (n).

Definición de Variables

	NOMBRE	TIPO	DESCRIPCIÓN
E			
P			
S			

Algoritmo

Prueba de Escritorio

Pseudocódigo

Diagrama de Flujo de Datos

6. Realizar un programa que sume dos números enteros.

Definición de Variables	Pseudocódigo	Diagrama de Flujo de Datos

	NOMBRE	TIPO	DESCRIPCIÓN
E			
P			
S			

Algoritmo

Prueba de Escritorio

7. Escribir un programa que calcule y escriba el cuadrado de 243.

Definición de Variables	Pseudocódigo	Diagrama de Flujo de Datos

	NOMBRE	TIPO	DESCRIPCIÓN
E			
P			
S			

Algoritmo

Prueba de Escritorio

8. Realizar un programa que de calcule el perímetro y la superficie de un rectángulo dada la base y la altura.

Definición de Variables	Pseudocódigo	Diagrama de Flujo de Datos

	NOMBRE	TIPO	DESCRIPCIÓN
E			
P			
S			

Algoritmo

Prueba de Escritorio

9. Realizar un programa que determine el área y volumen de un cilindro cuyas dimensiones radio y altura se leen desde el teclado.

Definición de Variables	Pseudocódigo	Diagrama de Flujo de Datos

	NOMBRE	TIPO	DESCRIPCIÓN
E			
P			
S			

Algoritmo

Prueba de Escritorio

10. Escribir un programa que lea un nombre de una marca de automóviles seguida del nombre de su modelo e informe el modelo seguido del nombre.

Definición de Variables	Pseudocódigo	Diagrama de Flujo de Datos

	NOMBRE	TIPO	DESCRIPCIÓN
E			
P			
S			

Algoritmo

Prueba de Escritorio

11. Diseñar un programa para determinar la hipotenusa de un triángulo rectángulo conocidas las longitudes de los catetos.

<table>
<tr><th>Definición de Variables</th><th>Pseudocódigo</th><th>Diagrama de Flujo de Datos</th></tr>
<tr><td>
<table>
<tr><th></th><th>NOMBRE</th><th>TIPO</th><th>DESCRIPCIÓN</th></tr>
<tr><td>E</td><td></td><td></td><td></td></tr>
<tr><td>P</td><td></td><td></td><td></td></tr>
<tr><td>S</td><td></td><td></td><td></td></tr>
</table>
Algoritmo

Prueba de Escritorio
</td><td></td><td></td></tr>
</table>

12. Diseñar un programa que calcule el área de un triángulo en función de las longitudes de sus lados: Área = p((p-a)(p-b)(p-c))1/2, donde p=(a+b+c)/2 (semiperímetro).

Definición de Variables

	NOMBRE	TIPO	DESCRIPCIÓN
E			
P			
S			

Algoritmo

Prueba de Escritorio

Pseudocódigo

Diagrama de Flujo de Datos

13. Escribir un programa que lea un número y escriba su cuadrado.

Definición de Variables	Pseudocódigo	Diagrama de Flujo de Datos

	NOMBRE	TIPO	DESCRIPCIÓN
E			
P			
S			

Algoritmo

Prueba de Escritorio

14. Realice la conversión de coordenadas polares a cartesianas

<table>
<tr><th>Definición de Variables</th><th>Pseudocódigo</th><th>Diagrama de Flujo de Datos</th></tr>
<tr><td>
<table>
<tr><th></th><th>NOMBRE</th><th>TIPO</th><th>DESCRIPCIÓN</th></tr>
<tr><td>E</td><td></td><td></td><td></td></tr>
<tr><td>P</td><td></td><td></td><td></td></tr>
<tr><td>S</td><td></td><td></td><td></td></tr>
</table>
</td><td rowspan="3"></td><td rowspan="3"></td></tr>
<tr><td>Algoritmo</td></tr>
<tr><td>Prueba de Escritorio</td></tr>
</table>

15. Realizar un programa que intercambie los valores de dos variables numéricas.

Definición de Variables

	NOMBRE	TIPO	DESCRIPCIÓN
E			
P			
S			

Algoritmo

Prueba de Escritorio

Pseudocódigo

Diagrama de Flujo de Datos

16. Diseñar un programa que permita convertir pulgadas a yardas, y pies a pulgadas.

Definición de Variables	Pseudocódigo	Diagrama de Flujo de Datos

	NOMBRE	TIPO	DESCRIPCIÓN
E			
P			
S			

Algoritmo

Prueba de Escritorio

17. Escribir un programa que imprima la conversión de kilómetros a millas. (1 milla marina = 1852 mts y 1 milla = 1609 mts).

Definición de Variables

	NOMBRE	TIPO	DESCRIPCIÓN
E			
P			
S			

Algoritmo

Prueba de Escritorio

Pseudocódigo

Diagrama de Flujo de Datos

18. Diseñar un programa que permita imprimir la conversión de centímetros a pulgadas (1 pulgada = 24.5 mm).

Definición de Variables	Pseudocódigo	Diagrama de Flujo de Datos

	NOMBRE	TIPO	DESCRIPCIÓN
E			
P			
S			

Algoritmo

Prueba de Escritorio

19. Escribir un programa tal que, dado el costo de un artículo vendido y la cantidad de dinero entregada por el cliente, calcule e imprima el cambio que debe entregársele al mismo.

Definición de Variables	Pseudocódigo	Diagrama de Flujo de Datos

	NOMBRE	TIPO	DESCRIPCIÓN
E			
P			
S			

Algoritmo

Prueba de Escritorio

S

Estructuras Algorítmicas Selectivas



ESTRUCTURAS ALGORÍTMICAS SELECTIVAS



Introducción..... 2
Estructuras algorítmicas selectivas..... 4
Estructura selectiva simple..... 4
Estructura selectiva doble..... 5
Estructura anidada..... 6
Estructura selectiva múltiple..... 8
Glosario..... 10
Bibliografía..... 10

ESTRUCTURAS ALGORÍTMICAS SELECTIVAS

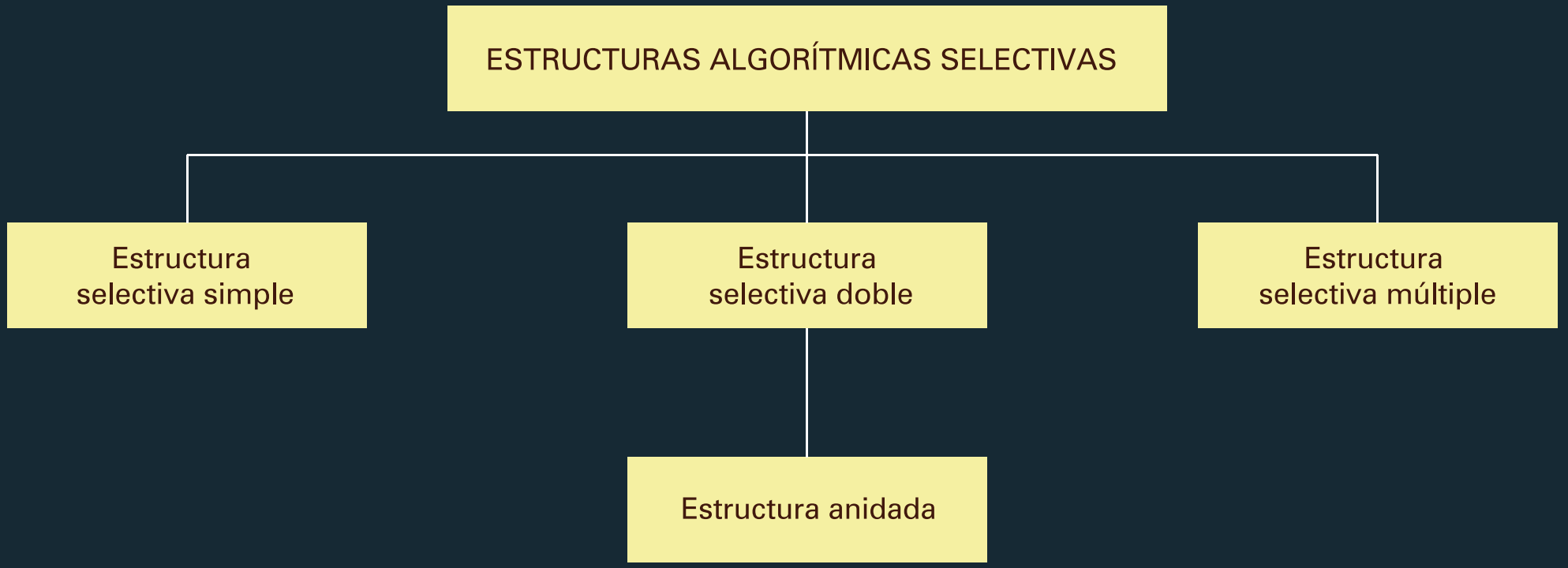


Introducción

Las estructuras de programación son también denominadas estructuras de control, se utilizan para resolver problemas mediante el diseño de algoritmos; de igual forma, estas estructuras de control selectivas se encuentran presentes cuando la solución algorítmica implica la toma de decisiones, la cual consiste en evaluar condiciones que deberán señalar una ruta a seguir.

Las estructuras algorítmicas selectivas que se estudiarán en el presente material de aprendizaje son: las estructuras selectivas simples y las estructuras dobles y las múltiples. Es importante anotar que en la búsqueda de la solución a un problema planteado, estas estructuras podrán combinarse.

Mapa conceptual



Estructura algorítmica selectiva

Las estructuras algorítmicas de selección o selectivas, se utilizan para resolver situaciones que implican la toma de decisiones.

Estas estructuras se caracterizan por contener una decisión lógica llamada condición, si al ser evaluada se cumple dicha condición, se realizarán una serie de instrucciones que darán solución a un caso particular.

Las estructuras algorítmicas selectivas se clasifican en:

- Estructura selectiva simple
- Estructura selectiva doble - Estructura anidada
- Estructura selectiva múltiple

Las estructuras selectivas simples, permiten que el diagrama de flujo siga una ruta específica.

Si la condición a evaluar se cumple, se ejecutarán una serie de instrucciones; en caso de que la condición no se cumpla, se pasa por alto esta operación, una vez evaluada la condición y seleccionada la ruta se continuará con la secuencia normal del diagrama.

Formato de estructura selectiva simple

Si (condición) entonces
Acciones
Fin si



López, en su libro de Programación estructurada, muestra la función de premisas estructura selectiva simple, así:

Donde:

Si	Identifica la estructura de control condicional.
Condición	Es la expresión lógica que denota la situación específica mediante la comparación de dos operadores, para dar un resultado booleano (falso, verdadero).
Acción(es)	Es la instrucción o conjunto de instrucciones que se ejecutarán en el bloque correspondiente.
Fin si	Indica el fin de la estructura de selección.

A continuación se presenta en el ejemplo 1, se presenta una situación que permite dar claridad al contenido estructura selectiva simple, su desarrollo consta de 5 partes:

1. Análisis
2. Declaración de variables
3. Diagrama de flujo
4. Pseudocódigo
5. Prueba de escritorio

Ejemplo 1

Diseñar un diagrama de flujo y el algoritmo que de solución a la siguiente situación.

Lea el promedio de un alumno de la clase de programación y escriba aprobado si la nota del promedio es igual o mayor de 6.

1. Análisis de la solución

El ejercicio solicita que se lea el promedio de notas de un alumno, se debe escribir aprobado si la nota es mayor o igual que 6.

2. Declaración de variables

Solo se necesita una variable para leer el promedio.

3. Diagrama de flujo



Figura 1. Diagrama de flujo estructura selectiva simple. Fuente: SENA

4. Pseudocódigo

Inicio
Real promedio;
Lea promedio;
Si (promedio >= 6) Entonces
Escribir 'Aprobó'
FinSi
Fin

5. Prueba de escritorio

Para este ejemplo se harán 2 pruebas de escritorio con promedios aleatorios, de tal forma que se cumpla primero la condición y después no se cumpla.

PROMEDIO	CONDIC N	SALIDA
7	Se cumple	Aprobó
4	No se cumple	———

Figura 2. Ejemplo prueba de escritorio estructura selectiva simple.
Fuente: SENA

Estructura selectiva doble (si–sino)

La estructura selectiva doble, si-sino, se conoce como alternativa o condicional; esta se encarga de evaluar la condición y si se cumple tomará el camino SI o Verdadero. Si la condición no se cumple, se irá por el camino NO o Falso.

En cualquiera de los casos después de ejecutar las instrucciones que se encuentren en el respectivo camino, saldrá de la estructura, se ejecutan las instrucciones que se encuentran por fuera hasta finalizar el proceso.

Formato estructura selectiva doble (si-sino):

Si (condición) entonces
Acciones
Sino
Acciones
Fin si

López, muestra la función de premisas estructura selectiva doble, así:

Donde:

Si	Identifica la estructura de control condicional.
Condición	Es la expresión lógica que denota la situación específica mediante la comparación de dos operadores para dar un resultado booleano (falso, verdadero) .
Entonces	Indica el curso de acción si se cumple la condición.
Acción+(es)	Es la instrucción o conjunto de instrucciones que se ejecutarán en el bloque correspondiente.
Sino	Indica el curso de acción cuando no se cumple la condición.
Fin si	Indica el fin de la estructura de selección.

Diagrama de flujo

Al utilizar diagramas de flujo, el símbolo que indica selección o pregunta es el rombo y mínimo debe tener 2 salidas.



Figura 3. Diagrama de flujo estructura selectiva doble
Fuente: SENA

Observación: Recuerde la estructura condicional simple puede tener o no, el Sino.

A continuación, se presenta en el ejemplo 2, una situación del libro programación teoría y aplicaciones del autor Echeverry, la cual permite dar claridad al contenido estructura selectiva doble (si – sino); su desarrollo consta de 6 partes:

1. Análisis
2. Declaración de variables
3. Diagrama de flujo
4. Pseudocódigo
5. Prueba de escritorio
6. Ejecución del algoritmo

Ejemplo 2

Diseñar un algoritmo y el correspondiente diagrama de flujo para la siguiente situación, lea la edad de una persona e indique si es mayor de edad o menor de edad.

Nota: Se toma 18 años como edad mínima para ser mayor de edad.

1. Análisis de la solución

El ejercicio solicita que se lea la edad de una persona; luego se evaluará si la edad es > = 18, si la edad resulta ser >= 18, se deberá imprimir el mensaje que indique el enunciado.

2. Declaración de variables

Solo se necesita una variable para leer la edad.

3. Diagrama de flujo



Figura 4. Diagrama de flujo estructura selectiva doble con proceso de edades
Fuente: SENA

4. Pseudocódigo

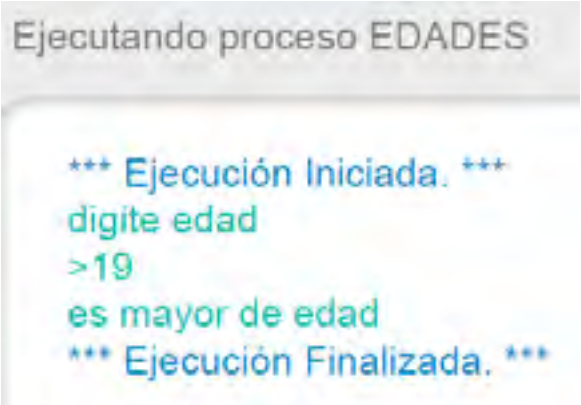
```
Inicio
Int edad;
Escribir "Digite edad"
Lea edad
Si (edad>=18) Entonces
Escribir 'es mayor de edad'
Sino
Escribir 'es menor de edad'
FinSi
Fin
```

5. Prueba de escritorio

Para este ejemplo se realizarán 2 pruebas de escritorio con edades aleatorias, de tal manera que primero se cumpla la condición y luego no se cumpla.

EDAD	CONDIC N	SALIDA
19	Se cumple	Es mayor de edad
15	No se cumple	Es menor de edad

6. Ejecución del algoritmo



Estructura anidada

Una estructura selectiva puede presentar bifurcaciones en los condicionales, es decir, puede tener anidada otra estructura y esta a su vez a otra y así sucesivamente.

Formato de estructura anidada

Si (condición 1) entonces
Si (condición 2) entonces
Acccion1
Sino
Accion2
Fin si
Fin si

Diagrama de flujo



Figura 6. Ejemplo prueba de escritorio estructura anidada. Fuente: SENA

A continuación se presenta una situación en el ejemplo 3, en la cual el autor Echeverry, cita en su libro de Programación estructurada y permite dar claridad al contenido estructura anidada; su desarrollo consta de 6 partes:

1. Análisis
2. Declaración de variables
3. Diagrama de flujo
4. Pseudocódigo
5. Prueba de escritorio
6. Ejecución del algoritmo

Ejemplo 3

Diseñar un algoritmo que le permita realizar operaciones aritméticas básicas (suma, resta, multiplicación), a partir del ingreso de dos números. Realice el respectivo diagrama de flujo.

1. Análisis de la solución

El usuario debe ingresar dos números y seleccionar la operación aritmética a realizar.

2. Declaración de variables

Se deben realizar las siguientes declaraciones:

Dos variables de entrada: numero 1, numero2.

Una variable para realizar la operación aritmética: operación

Una variable que muestre el resultado: resultado.

3. Diagrama de flujo



Figura 7. Ejemplo prueba de escritorio estructura anidada con operaciones aritméticas en diagrama de flujo. Fuente: SENA

4. Pseudocódigo

Inicio
Entero: numero 1, numero2
Real: Resultado
Cadena: operación
Escribir “ingrese primer número”
Lea numero1
Escribir “ingrese segundo número”
Lea numero2
Escribir “ingrese operación”
Lea operación
Si (operación = ”+”) entonces
Resultado=numero1 + numero2
Sino
Si (operación = ” - ”) entonces
Resultado=numero1 - numero2

Sino
Si (operación = " * ") entonces
Resultado=numero1 * numero2
FinSi
FinSi
FinSi

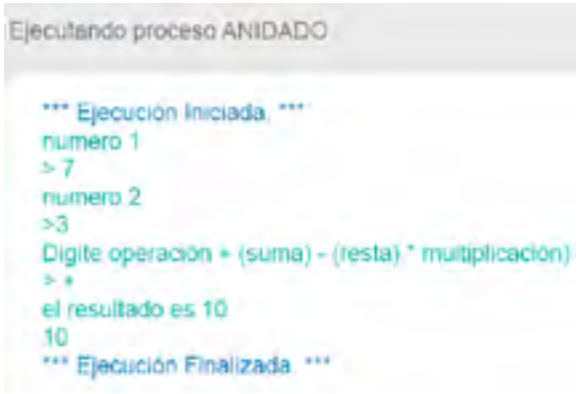
5. Prueba de escritorio

Para este ejemplo se realizará una prueba de escritorio con 2 números y se obtendrán los resultados, de acuerdo a las operaciones seleccionadas.

NÚMERO 1	NÚMERO 2	OPERACIÓN ARITMÉTICA	RESULTADO
7	3	+	10
7	3	-	4
7	3	*	21

Figura 8. Ejemplo prueba de escritorio estructura selectiva anidada. Fuente: SENA

6. Ejecución del algoritmo



Estructura selectiva múltiple

Esta estructura evalúa la expresión dada en función de un valor; se conoce como estructura Caso, permite controlar la ejecución de las acciones cuando se tiene más de dos opciones.

Formato de estructura múltiple

Caso (variable) Haga
Caso 1: acción
Caso 2: acción
:
:
Caso n: acción n
Fincase

López, muestra la función de premisas estructura selectiva múltiple, así:

Donde:

Case	Identifica la estructura de selección múltiple.
Variable	Es la variable que indica la opción a elegir.
1: 2: 3:..	Son las etiquetas que indican cada acción.
Acción(es)	Es la instrucción o conjunto de instrucciones que se ejecutarán en el bloque correspondiente.
Sino	Indica el curso de acción cuando el selector toma un valor distinto a las etiquetas.
Fin case	Indica el fin de la estructura múltiple.

A continuación se presenta una situación en el ejemplo 4, la cual López cita en su libro de Programación estructurada, permite dar claridad al contenido estructura selectiva múltiple; su desarrollo consta de 6 partes:

1. Análisis
2. Declaración de variables
3. Diagrama de flujo
4. Pseudocódigo
5. Prueba de escritorio
6. Ejecución del algoritmo

Ejemplo 4

Diseñar un algoritmo que lea un número entre 1 y 7, muestre en el resultado domingo si digita 1, lunes si digita 2, martes si digita 3, miércoles si digita 4 jueves si digita 5, viernes si digita 6 y sábado si digita 7. Realice el respectivo diagrama de flujo.

1. Análisis de la solución

El usuario debe ingresar un número entre el 1 y 7, dependiendo del valor el resultado que arroje el algoritmo será un día de la semana.

2. Declaración de variables

Solo se utiliza una variable donde se guardará el número correspondiente al día seleccionado: numdia

3. Diagrama de flujo



Figura 9. Ejemplo prueba de escritorio estructura múltiple en diagrama de flujo. Fuente: SENA

4. Pseudocódigo

Inicio
Escribir “Elija un número (1-7): “
Leer numdia
Caso (numdia) Hacer
1: Escribir “domingo”
2: Escribir “lunes”
3: Escribir “martes”
4: Escribir “miércoles”
5: Escribir “jueves”
6: Escribir “viernes”
7: Escribir “sabado”
De otro modo:
Escribir “Opción no válida”
Fin
Fininicio

5.Prueba de escritorio

NÚMERO	RESULTADO
1	Domingo
3	Martes
7	Sábado

Figura 10. Ejemplo prueba de escritorio estructura selectiva múltiple. Fuente: SENA

6.Ejecución del algoritmo

```
Ejecutando proceso

*** Ejecución Iniciada. ***
Elija un número (1-7)
> 1
Domingo
Presione enter para continuar
```



Glosario

Asignar: Almacenar un valor determinado en una variable.

Comentarios: Son anotaciones; recordatorios dentro del programa para uso de los programadores.

Estructura condicional: Permite evaluar y tomar una decisión de acuerdo al resultado de una condición.

Operador: Son todos aquellos signos que indican la realización de operaciones aritméticas y lógicas.

Variable: Estructura de dato, que puede cambiar su contenido a lo largo de la ejecución de un programa.

Bibliografía

Joyanes, L. Echeverri, J. Orrego, G. y Arenas. O. Programación teorías y aplicaciones. (2016). Universidad de Medellín.

López, L. Programación estructurada. Un enfoque algorítmico. (2000). Editorial Computec.

Creative Commons



ATRIBUCIÓN, NO COMERCIAL, COMPARTIR IGUAL

Este material puede ser distribuido, copiado y exhibido por terceros si se muestra en los créditos. No se puede obtener ningún beneficio comercial y las obras derivadas tienen que estar bajo los mismos términos de licencia que el trabajo original.





Créditos

PROGRAMA	Metodología de la programación de sistemas informáticos
NOMBRE DEL OBJETO	ESTRUCTURAS ALGORÍTMICAS SELECTIVAS
EXPERTO TEMÁTICO	Carlos Felipe Domínguez Illera
DISEÑADORES GRÁFICOS	Caren Xiomara Carvajal Pérez
	Luis Guillermo Roberto Báez
	Luis Carlos Reyes Parada
GESTOR DE REPOSITORIO	Nancy Astrid Barón López
	Nilda Inés Camargo Suescún
PROGRAMADORES	Fredy Velandia Figueroa
GUIONISTA Y PRODUCTOR DE MEDIOS AUDIOVISUALES	Jheison Edimer Muñoz Ramírez
GUIONISTA	Adriana Carolina Acosta Caycedo
ASESORAS PEDAGÓGICAS	Kennia Andrea Peña Barrera
	Janet Lucia Villalba Triana.
LÍDER DE LA LÍNEA DE PRODUCCIÓN	Zulma Yurany Vianchá Rodríguez