PROCESO GESTIÓN DEL TALENTO HUMANO

FORMATO INFORME MENSUAL EJECUCIÓN CONTRACTUAL

Barranquilla, Agosto de 2020

Señor (a)
MANUEL GREGORIO HORMECHEA LANCE
SUPERVISOR(A) CONTRATO No. **1700803**
Coordinador de Automatización
Automatización
Barranquilla

Asunto: Informe mensual de ejecución contractual Mes Agosto del año 2020

Referencia: No 1700803 del año 2020

RICARDO ENRIQUE CANTILLO CARRILLO, identificado con la cédula de ciudadanía No. 72.201.581 de Barranquilla, en mi calidad de Contratista del SENA, en Automatización, en cumplimiento del Contrato de Prestación de Servicios de la referencia, a continuación presento el Informe de actividades realizadas en el mes objeto de cobro.

Valor y forma de Pago: b). un pago correspondiente al mes de Agosto de 2020 por valor de (3.278.181). Los honorarios serán pagados por el SENA al contratista de acuerdo al cronograma definido por la Dirección Administrativa y Financiera de la Dirección General, en la cuenta de Ahorros No. 0098000200063852 BBVA, cuyo titular es el Contratista.

Plazo: Será hasta el 20 de Noviembre de 2020.

OBJETO:
Prestación de servicios profesionales como instructor para impartir formación profesional titulada y complementaria en el área de teleinformática - análisis y desarrollo de sistemas de información de acuerdo a la programación académica del SENA Centro Nacional Colombo Alemán

CLÁUSULA (No. de cláusula donde se encuentren las obligaciones del contratista)

No	Obligaciones	Acciones realizadas	Evidencias
1	Programar las actividades de enseñanza, aprendizaje y evaluación de conformidad con los de formación y el calendario institucional y el Manual de procedimientos para la ejecución.	Programar las actividades de enseñanza, aprendizaje y evaluación de conformidad con los de formación y el calendario institucional y el Manual de procedimientos para la ejecución.	Tutorias y Laboratorios de Programación en los grupos ADSI 54 y ADSI 55. y participación en el Bootcamp de Blockchain.

A continuación relaciono los desplazamientos que realicé previo a la presentación de este informe. Una vez finalizado cada desplazamiento presenté al ordenador del gasto el informe en el Formato Informe Legalización Desplazamiento Contratista GTH-F-087, en el que se describieron las actividades desarrolladas y los resultados de cada desplazamiento. Cada informe cuenta con el visto bueno del Supervisor.
Se lista a continuación el soporte de la legalización de los desplazamientos realizados, los cuales forman parte integral del presente informe de ejecución contractual.

ITEM	No DE LA ORDEN DE VIAJE	LUGAR DE DESPLAZAMIENTO	FECHAS DE DESPLAZAMIENTOS	
			INICIO	FINALIZACIÓN

Nota 1: *Por cada desplazamiento que haya realizado el contratista, adjuntará el respectivo informe que la soporte. En caso de haber realizado el desplazamiento en fecha posterior a la presentación del informe de ejecución contractual, deberá reportarlo en el siguiente informe de ejecución contractual.*

Para el trámite de la cuenta me permito adjuntar: Documentos enunciados como evidencias del cumplimiento de las obligaciones contractuales y los desplazamientos realizados y el No. de la planilla, operador y periodo. (Decreto Ley 2106 de 2019 – " *Decreto Ley Antitrámites"*)

Evidencias en ( ) folios

Cordialmente,

RICARDO ENRIQUE CANTILLO CARRILLO
Contratista
C.C. No. 72.201.581

Recibí a satisfacción:

MANUEL GREGORIO HORMECHEA LANCE
Supervisor(a) Contrato 1700803 de 2020
Coordinador de Automatización

FUNDAMENTOS DE PROGRAMACIÓN ESTRUCTURADA



FAVA - Formación en Ambientes Virtuales de Aprendizaje

SENA - Servicio Nacional de Aprendizaje.

ESTRUCTURA DE CONTENIDOS

Pág.

Introducción 3
Mapa de contenido 4
1. El primer programa. 5
1.1. La codificación. 5
1.2. La compilación 5
1.3. La depuración. 6
1.4. La ejecución. 7
2. Tipos de datos. 7
3. Operadores y expresiones aritméticas. 9
3.1. Los operadores aritméticos. 9
3.2. Reglas de prioridad en los operadores aritméticos. 9
3.3. Ejemplos de expresiones aritméticas. 10
4. Operadores relacionales y lógicos. 12
4.1. Los operadores relacionales. 12
4.2 Los operadores lógicos. 13
4.3. Expresiones con operadores relacionales y lógicos. 13
5. Estructuras básicas de programación. 14
5.1. Estructuras secuenciales 14
5.2. Estructuras condicionales 15
6. Estructuras cíclicas indeterminadas. 20
6.1. La estructura cíclica “para”. 21
6.3. La estructura cíclica “repita”. 24
7. Armando el rompecabezas con estructuras de programación. 26
8. Ejemplo de codificación y ejecución de un programa con diferentes estructuras de programación 26
Glosario 29
Bibliografía 30
Control del documento 31

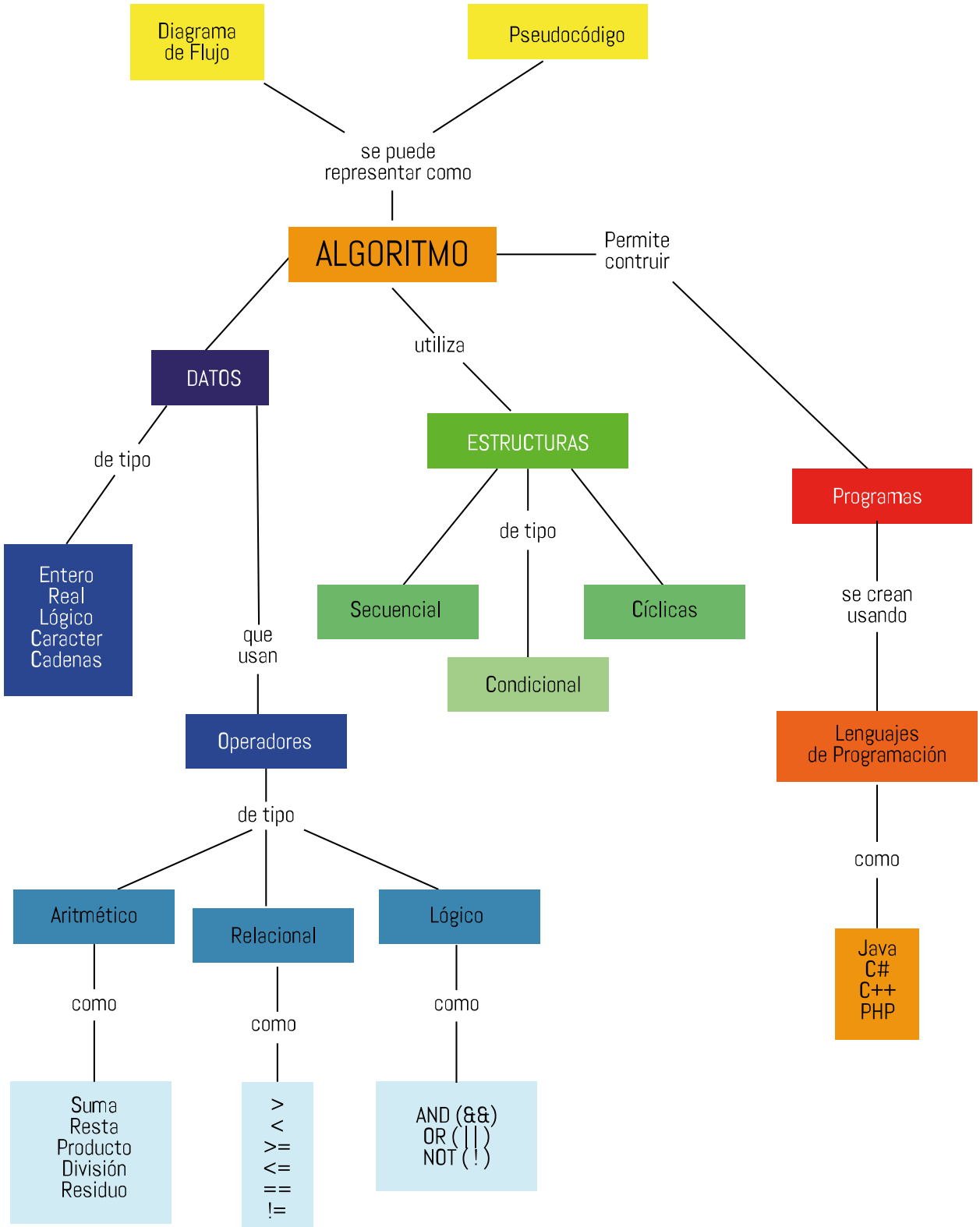
FUNDAMENTOS DE PROGRAMACIÓN ESTRUCTURADA

INTRODUCCIÓN

Para un Analista y Desarrollador de Sistemas de Información es primordial adquirir una gran destreza en el desarrollo de soluciones algorítmicas, ya que estas se convertirán posteriormente en programas de computador capaces de automatizar las tareas cotidianas de una organización, empresa o individuo. Los fundamentos de programación pueden ser comparados con los fundamentos para conducir un vehículo, pues estos fundamentos son aplicables a cualquier tipo de vehículo sin importar su marca o cilindraje. De la misma manera, los fundamentos de programación son aplicados en cualquier lenguaje de programación.

La herramienta LPP (Lenguaje de Programación para Principiantes), permite adquirir los fundamentos de programación necesarios para construir soluciones de software, familiarizando al programador con tareas rutinarias como la declaración de variables, el uso de estructuras de control de flujo, arreglos, subrutinas y muchas otras actividades que forman parte del día a día de un desarrollador de sistemas de información.

MAPA DE CONTENIDO

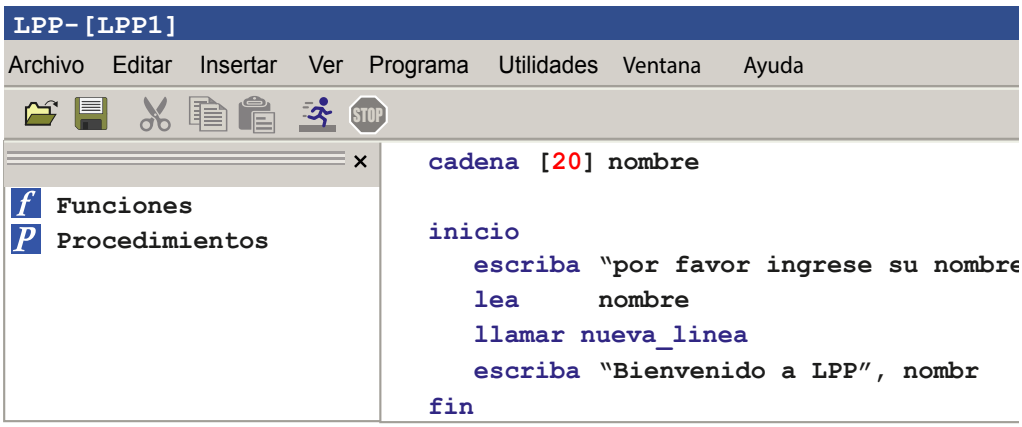


DESARROLLO DE CONTENIDOS

1. El primer programa.

1.1. La codificación.

La Codificación consiste en pasar el algoritmo al lenguaje de programación seleccionado, en este caso el lenguaje es LPP. El primer programa consiste en leer el nombre de una persona y presentar un mensaje personalizado.

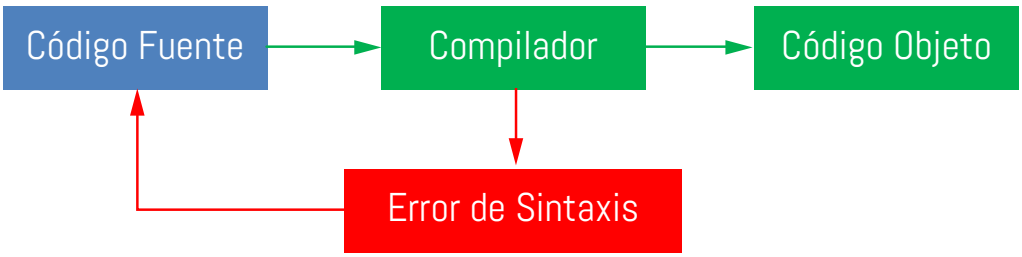


LINEA	SIGNIFICADO
Cadena [20] nombre	Antes del inicio se deben declarar todas las variables que se van a emplear en el programa.
Inicio	Marcar el inicio del programa.
escriba "Por favor ingrese su nombre"	Presenta un mensaje en la pantalla.
lea nombre	Captura información por parte del usuario.
llamar nueva_línea.	Permite pasar a la siguiente línea de la pantalla del usuario.
escriba "Bienvenido a LPP", nombre	Presenta mensaje combinando parte textual con parte variable.
Fin	Marca el final del programa.

1.2. La compilación.

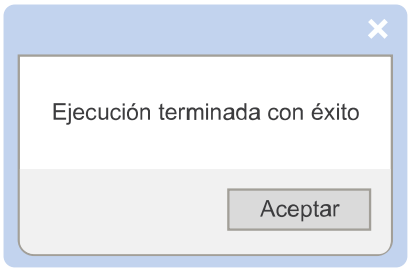
El proceso de Compilación permite traducir el código fuente de un lenguaje de programación a lenguaje de máquina (código objeto) para que pueda ser ejecutado por la computadora. Las computadoras sólo entienden el lenguaje de máquina.

Dentro de la ejecución del proceso de Compilación, se permite detectar los errores sintácticos (sintaxis: conjunto de normas que regulan la codificación de un programa), también conocidos como errores de compilación.



Para compilar un programa en LPP, seleccionamos del menú Programa la opción Compilar.

```
cadena [20] nombre
inicio
   escriba "Por favor ingrese su nombre:"
   lea nombre
   llamar nueva_linea
   escriba "Bienvenido a LPP" , nombr
```



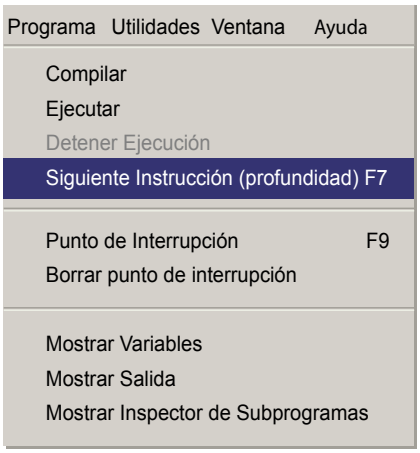
1.3. La depuración.

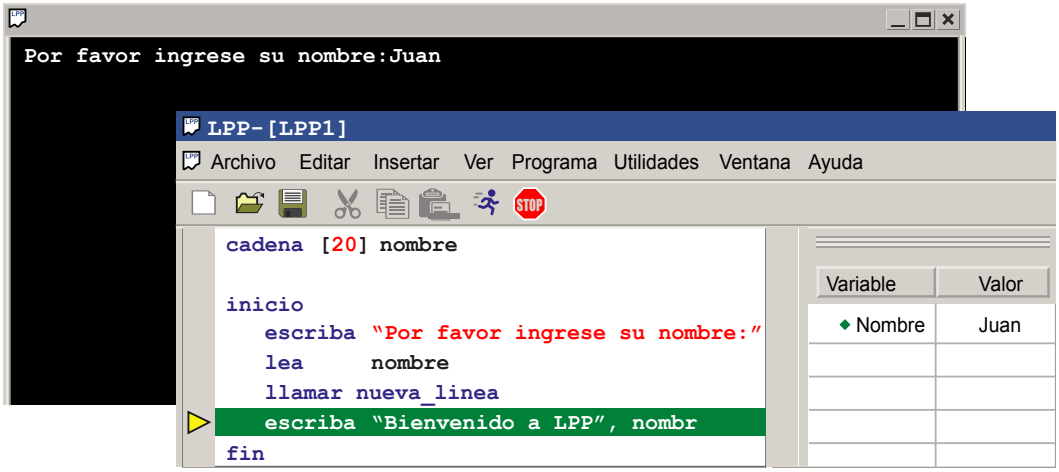
La Depuración permite hacer el seguimiento paso a paso de un programa.

Con la depuración es posible pasar de instrucción en instrucción e ir observando el comportamiento que va teniendo el programa y los valores que van tomando las variables. De esta manera el programador puede encontrar tanto errores de sintaxis como errores de lógica.

Igualmente, se pueden establecer puntos de interrupción de manera que el programa se ejecuta hasta esa línea de instrucciones y el programador puede ver el contenido de las variables, así como el comportamiento del programa hasta ese punto.

Para depurar un programa en LPP, se cuenta con una serie de opciones desde el menú Programa.





1.4. La ejecución.

La Ejecución del programa permite observar su comportamiento de la manera como lo percibirá el usuario final. Para ejecutar un programa en LPP, se selecciona del menú Programa la opción Ejecutar, o se hace click en el botón correr de la barra de herramientas.

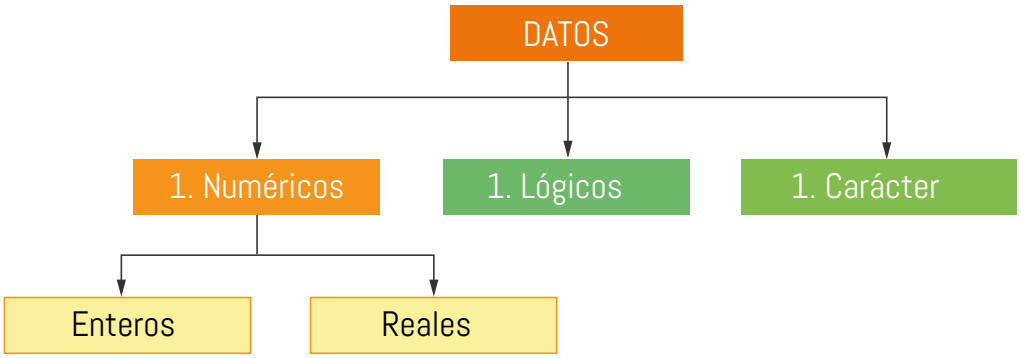
2. Tipos de datos.

FAl desarrollar un programa de computador, el programador debe trabajar con diferentes datos, por ejemplo, si el programa necesita mostrar la información de un estudiante, seguramente requerirá de datos como su nombre, edad, género, nota, está Matriculado, entre otros. Cada uno de estos datos son de una naturaleza diferente.

TIPO DE DATO	NOMBRE EN LPP	COMENTARIOS
Entero	Entero	Números sin decimales.
Real	Real	Números que pueden tener decimales.
Lógico	Booleano	Solo recibe valores de falso y verdadero.
Carácter	Carácter	Sólo recibe un único carácter, que puede ser una letra, un número o un signo.
	Cadena	Puede recibir un conjunto de carácteres.



Nombre= Juan Pérez
Edad= 19
Género= M
Nota= 4.5
estaMatriculado= Verdadero



Programa de Ejemplo:

A

```
Cadena [30] nombre
Entero edad
Carácter genero
Real nota
Booleano esta Matriculado
Inicio
```

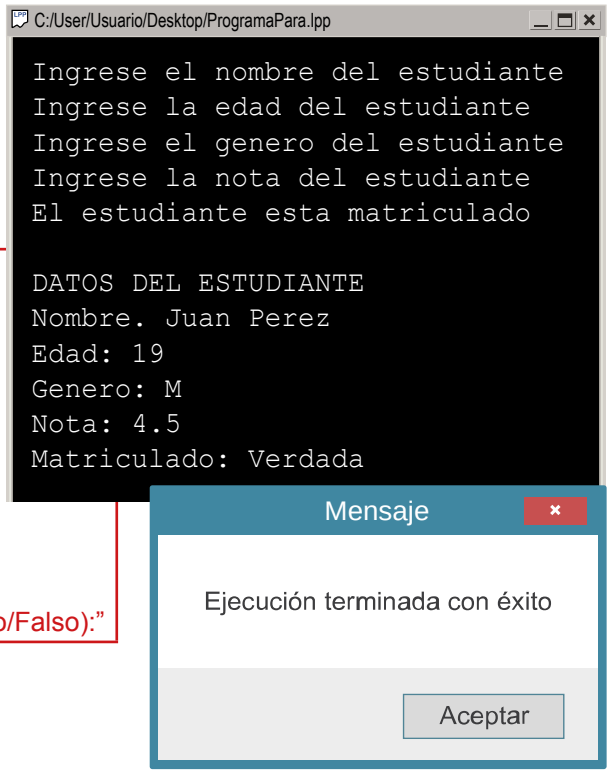
B

```
escriba "Ingrese el nombre del estudiante:"
lea nombre
escriba "Ingrese la edad del estudiante:"
lea edad
escriba "Ingrese el genero del estudiante (M/F):"
lea genero

escriba "Ingrese la nota del El estudiante"
lea nota
escriba "Ingrese la nota del estudiante:"
lea nota
escriba "El estudiante está matriculado? (Verdadero/Falso):"
lea está Matriculado
llamar nueva_linea
escriba "DATOS DEL ESTUDIANTE"
llamar nueva_linea
```

C

```
escriba "Nombre: ",nombre
llamar nueva_linea
escriba "Edad: ",edad
llamar nueva_linea
escriba "Género: ",genero
llamar nueva_linea
escriba "Nota: ",nota
llamar nueva_linea
escriba "Matriculado: ", esta Matriculado
Fin
```



SECCIÓN	COMENTARIO
A	Declaración de cada una de las variables empleadas en el programa con su respectivo tipo de dato.
B	Lectura o asignación de datos a cada una de las variables empleadas en el programa.
C	Escritura o impresión de los datos contenidos en cada una de las variables empleadas en el programa.

3. Operadores y expresiones aritméticas.

3.1. Los operadores aritméticos.

La mayoría de los programas de computador requieren realizar cálculos u operaciones que involucran operadores aritméticos, por esta razón, como programadores es necesario conocer cada uno de ellos y la manera como el computador los interpreta para calcular los resultados de una determinada expresión o fórmula.

¿Cuál es el resultado de la siguiente expresión?

3 + 5 x 2 = _____?

Explicación

5 x 2 = 10

3 + 10 = 13

la respuesta correcta es 13, pues bien, además de conocer los diferentes operadores aritméticos, también es importantísimo conocer los niveles de prioridad de cada uno de ellos.

En el caso de la expresión 3 + 5 x 2, primero se realiza la multiplicación 5 x 2 cuyo resultado es 10 y posteriormente se realiza la operación 3 + 10, dando como resultado final 13.

3.2. Reglas de prioridad en los operadores aritméticos.

Cuando dos operadores tienen el mismo nivel de prioridad, dentro de una expresión se evalúan de izquierda a derecha.

En LPP el signo igual (=), se representa mediante una flecha dirigida hacia la variable que recibe el valor, esta flecha está conformada por los carácteres menor que (<) y menos (-) así: <-

Prioridad	Operador	Significado	Ejemplo	
1	^	Exponenciación	4 ^ 2 = 16	3 ^ 3 = 27
2	*	Multiplicación	2 * 4 = 8	7 * 5 = 35
	/	División	5 / 2 = 2.5	6 / 3 = 2
3	DIV	División entera	5 DIV 2 = 2	7 DIV 4 = 1
	MOD	Residuo de la División	5 MOD 2 = 1	8 MOD 4 = 0
4	+	Suma	3 + 4 = 7	2 + 9 = 11
	-	Resta	8 - 5 = 3	7 – 6 = 1

ejemplo, para representar la siguiente expresión: X = 3 + 5
En LPP sería: X <- 3 + 5

Programa de Ejemplo:

Ahora, después de conocer los operadores aritméticos y sus reglas de prioridad, puedes encontrar el resultado de la siguiente expresión:

2 + 2 ^ 2 * 2 + 2 MOD 2 =____?
2 + 2 ^ 2 * 2 + 2 MOD 2 =____?
2 + 2 ^ 2 + 2 =____?
2 + 8 +2 MOD 2 =____?
2 + 8 + 0 = 10

3.3. Ejemplos de expresiones aritméticas.

Cuando el programador desea determinar un orden específico de ejecución en una expresión aritmética, puede emplear los paréntesis para agrupar, de esta manera, las operaciones que se encuentren dentro del paréntesis serán las primeras en ejecutarse. Retomando el ejemplo de la expresión:

3 + 5 x 2 = 13 pero (3 + 5) x 2 = 16
5 x 2 = 10 (3 + 5) = 8
3 + 10 = 13 8 x 2 = 16

3.4. Ejemplo, manejando expresiones en un programa.

Programa No. 3 Manejando expresiones: a un programador le solicitan realizar una aplicación que calcule la nota promedio de un alumno a partir de las 2 notas que tiene en una asignatura.

Durante el análisis, el programador toma un caso de prueba para descubrir cuál es el procedimiento que debe llevar a cabo. En el caso de prueba toma como la primera nota el valor de 4 y como segunda nota el valor de 3.

Nota 1	Nota 2	NotaPromedio
4	3	3.5

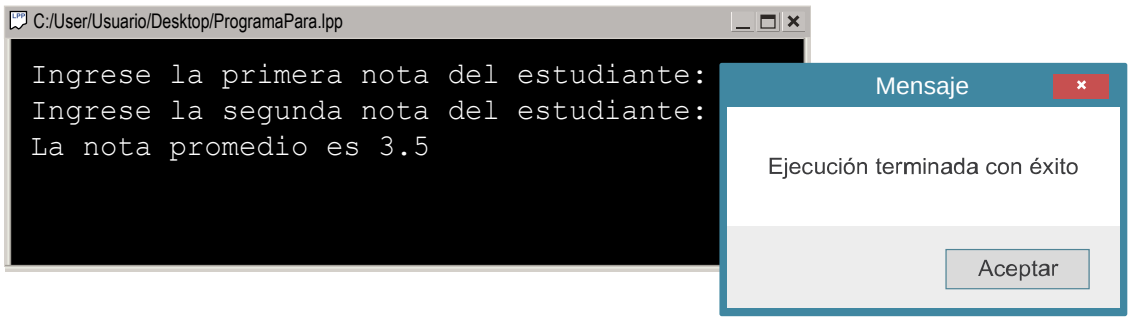
Durante el análisis, el programador identifica que debe sumar las dos notas y el resultado lo debe dividir entre dos: 4 + 3 = 7 luego 7 / 2 = 3.5

A partir de este análisis, el programador desarrolla la siguiente aplicación en LPP:

```
Real nota1, nota2, notaPromedio

Inicio
  escriba "Ingrese la primera nota del estudiante:"
  lea nota1
  escriba "Ingrese la primera nota del estudiante:"
  lea nota2
  notaPromedio <-nota1 + nota2 / 2
  escriba "la nota promerio es;" notaPromedio
Fin
```

Al ejecutar la aplicación ingresando los datos de prueba, el programador obtiene el siguiente resultado:



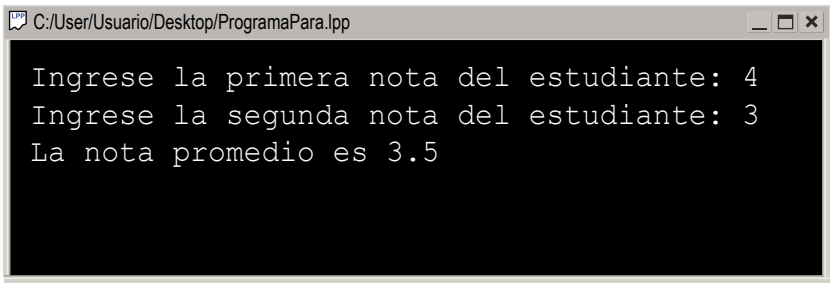
Después de buscar el error, se da cuenta que este se encuentra en la siguiente línea:

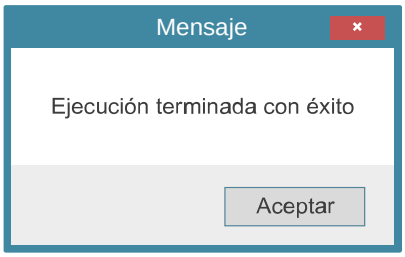
```
Real nota1, nota2, notaPromedio

Inicio
  escriba "Ingrese la primera nota del estudiante:"
  lea nota1
  escriba "Ingrese la primera nota del estudiante:"
  lea nota2
  notaPromedio <- (nota1 + nota2) / 2
  escriba "la nota promerio es;" notaPromedio
Fin
```

Se debe recordar las reglas de prioridad y concluir que la primera operación que se está ejecutando es la división, por lo tanto, el programador realiza un ajuste al programa para definir el orden deseado de ejecución de los operadores aritméticos mediante el uso de paréntesis.
3 / 2 = 1.5 y
4 + 1.5 = 5.5

Al ejecutar nuevamente la aplicación, obtiene el resultado esperado:





4. Operadores relacionales y lógicos.

4.1. Los operadores relacionales.

Los operadores relacionales y lógicos son empleados para definir condiciones dentro de un programa. El resultado de una expresión que contiene estos operadores es un resultado de tipo lógico, es decir, solo puede ser FALSO o VERDADERO.

Operador	Significado	Ejemplo
>	Mayor que	3 > 4 FALSO
		8 > 5 VERDADERO
<	Menor que	4 < 6 VERDADERO
		7 < 4 FALSO
>=	Mayor o igual que	3 >= 3 VERDADERO
		4 >= 4 VERDADERO
<=	Menor o igual que	2 <= 2 VERDADERO
		3 <= 2 FALSO
=	Igual que	4 = 4 VERDADERO
		3 = 4 FALSO
<>	Diferente que	6 <> 7 VERDADERO
		7 <> 7 FALSO

4.2 Los operadores lógicos.

Los operadores lógicos son empleados para concatenar dos o más expresiones con operadores relacionales.

Por ejemplo, la expresión:

3 > 2 Y 4 < 5 es **VERDADERO**, porque ambas expresiones son verdaderas

3 > 2 Y 4 < 3 es **FALSO**, porque hay una expresión falsa

3 > 2 O 4 < 3 es **VERDADERO**, Porque hay una expresión verdadera

6 < 4 O 7 > 8 es **FALSO**, Porque ambas expresiones son falsas

- El operador lógico "Y" solo da como resultado Verdadero si ambas expresiones son verdaderas.

Operador "Y"			
Expresión 1	Operador	Expresión 2	Resultado
FALSO	Y	FALSO	FALSO
FALSO	Y	VERDADERO	FALSO
VERDADERO	Y	FALSO	FALSO
VERDADERO	Y	VERDADERO	VERDADERO

- El operador "O" da como resultado Verdadero cuando al menos una de las expresiones sea verdadera.

Operador "O"			
Expresión 1	Operador	Expresión 2	Resultado
FALSO	O	FALSO	FALSO
FALSO	O	VERDADERO	VERDADERO
VERDADERO	O	FALSO	VERDADERO
VERDADERO	O	VERDADERO	VERDADERO

4.3. Expresiones con operadores relacionales y lógicos.

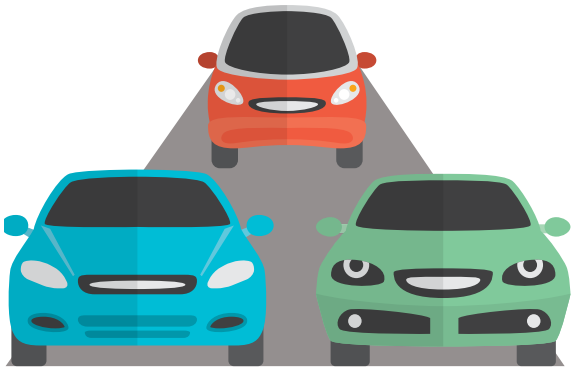
Una tarea habitual y muy importante a la hora de desarrollar programas de computador consiste en definir condiciones dentro del programa. Para que estas condiciones respondan exactamente a lo que el cliente de la aplicación necesita, se deben crear correctamente las expresiones que usen los operadores relacionales y lógicos.

Para determinar los estudiantes menores de edad cuya nota es 4 o superior y que pertenecen a un estrato inferior a 3, la expresión

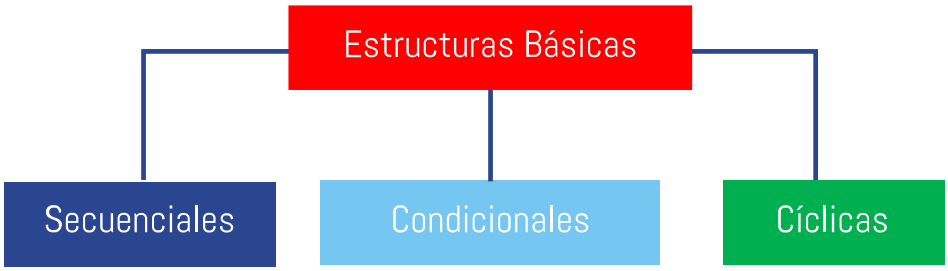


(edad < 18)
(nota >= 4)
(estrato < 3)

Por ejemplo, para conocer los vehículos cuyo modelo sea inferior al año 2010, la expresión sería: (modelo < 2010)



5. Estructuras básicas de programación.



5.1. Estructuras secuenciales.

La estructura secuencial está conformada por instrucciones que se ejecutan consecutivamente una después de la otra. En una aplicación, después de ejecutar una instrucción secuencial, el programa siempre continúa con la siguiente instrucción

Instrucción	Significado	Ejemplo
Inicio	Determina el comienzo del programa	Inicio
Fin	Determina el final del programa	Fin
Escriba	Muestra un mensaje en pantalla	Escriba “Bienvenido”
Lea	Almacena un dato suministrado por el usuario en una variable del programa	Lea nombre
Asignación	Permite asignarle a una variable un valor o el resultado de una expresión	X <- 2 area<- (b*h)/2

Programa Secuencial:

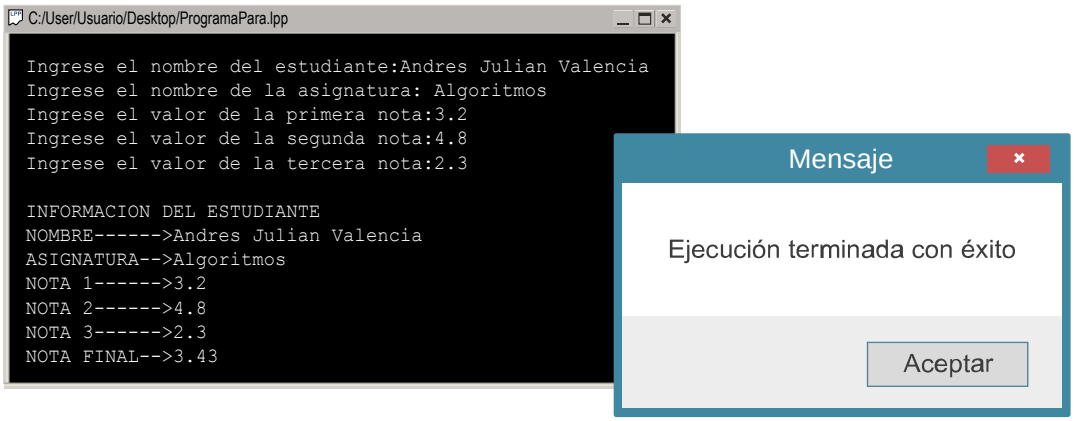
Se requiere una aplicación que lea el nombre de un estudiante, el nombre de la asignatura y sus 3 notas parciales y presente un mensaje con sus datos y nota final.

```
//Declaración de Variables
Cadena [25] nombre
Cadena [20] asignatura
Real nota1, nota2, nota3, notaFinal

Inicio
    //Lectura de los datos de entrada
    escriba "Ingrese el nombre del estudiante:"
    lea nombre
    escriba "Ingrese el nombre de la asignatura:"
    lea asignatura
    escriba "Ingrese el valor de la primera nota:"
    lea nota1
    escriba "Ingrese el valor de la segunda nota:"
    lea nota2
    escriba "Ingrese el valor de la tercera nota:"
    lea nota3

    //Cálculo de la nota final
    notaFinal <- (nota1 + nota2 + nota3) / 3

    //Escritura de la salida
    llamar nueva_linea
    escriba "INFORMACION DEL ESTUDIANTE"
    llamar nueva_linea
    escriba "NOMBRE------>",nombre
    llamar nueva_linea
    escriba "ASIGNATURA-->",asignatura
    llamar nueva_linea
    escriba "NOTA 1------>",nota1
    llamar nueva_linea
    escriba "NOTA 2------>",nota2
    llamar nueva_linea
    escriba "NOTA 3------>",nota3
    llamar nueva_linea
    escriba "NOTA FINAL-->",notaFinal
Fin
```



5.2. Estructuras Condicionales.

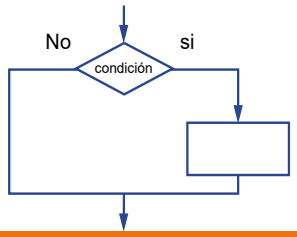
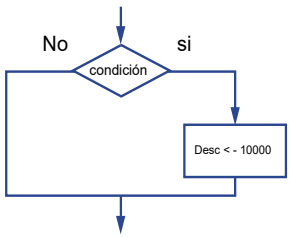
5.2.1 Condicional Simple.

La estructura condicional simple verifica una condición y si esta es verdadera ejecuta las instrucciones que se encuentren dentro de la estructura. Por esta razón, a diferencia de los programas secuenciales donde siempre se ejecutan las mismas instrucciones,

en los programas condicionales, existen instrucciones que su ejecución depende del cumplimiento de determinadas condiciones.

```
Inicia Robot
    SI hay obstáculo Quitar
    SI no hay obstáculo Avanzar
    SI hay obstáculo Quitar
    SI no hay obstáculo Avanzar
    SI hay obstáculo Quitar
    SI no hay obstáculo Avanzar
    SI hay obstáculo Quitar
    SI no hay obstáculo Avanzar
    GirarIzquierda
    SI hay obstáculoQuitar
    SI no hay obstáculo Avanzar
    SI hay obstáculoQuitar
    SI no hay obstáculo Avanzar
    GirarDerecha
    SI hay obstáculoQuitar
    SI no hay obstáculo Avanzar
    SI hay obstáculoQuitar
    SI no hay obstáculo Avanzar
    SI hay obstáculoQuitar
    SI no hay obstáculo Avanzar
    Detener
Termina Robot
```

Sintaxis de una estructura condicional simple

Diagrama de Flujo	LPP
	Si condición Entonces xxxxxxx Fin Si
Ejemplo Diagrama de Flujo	**Ejemplo LPP**
	Si edad < 18 Entonces Desc < - 10000 Fin Si

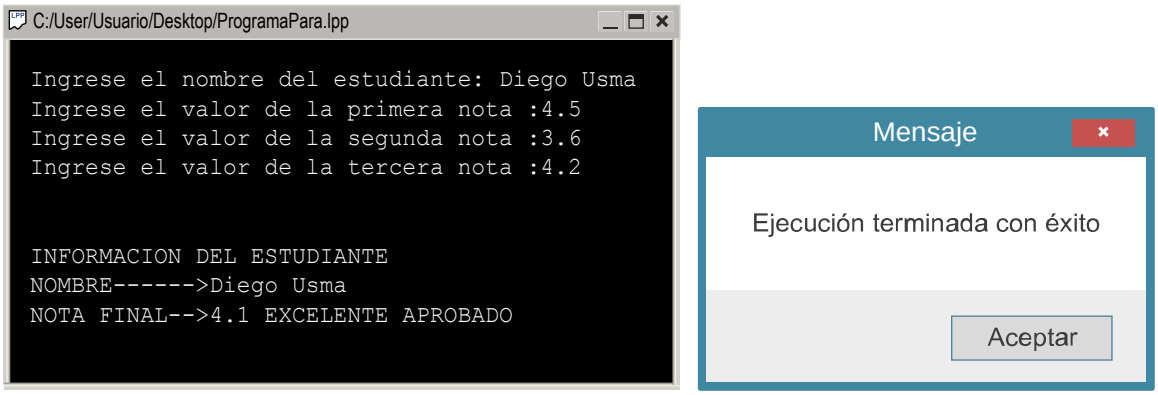
Programa condicional Simple:

Se requiere una aplicación que lea el nombre de un estudiante y sus 3 notas parciales y presente un mensaje con su nombre y nota final. Si la nota final es inferior a 3, presentar el mensaje “REPROBADO”, si su nota final es superior o igual a 3 y menor a 4, presentar el mensaje “APROBADO”, y si su nota final es 4 o superior, presentar el mensaje “EXCELENTEMENTE APROBADO”.

```
//Declaración de Variables
Cadena [25] nombre
Real nota1, nota2, nota3, notaFinal

Inicio
    //Lectura de los datos de entrada
    escriba “Ingrese el nombre del estudiante:”
    lea nombre
    escriba “Ingrese el valor de la primera nota:”
    lea nota1
    escriba “Ingrese el valor de la segunda nota:”
    lea nota2
    escriba “Ingrese el valor de la tercera nota:”
    lea nota3
    //Cálculo de la nota final
    notaFinal <- (nota1 + nota2 + nota3) / 3
```

```
    //Escritura de la salida
    llamar nueva linea
    llamar nueva_linea
    escriba “INFORMACION DEL ESTUDIANTE”
    llamar nueva_linea
    escriba “NOMBRE------>”,nombre
    llamar nueva_linea
    escriba “NOTA FINAL-->”,notaFinal
    Si notaFinal < 3 Entonces escriba “
    REPROBADO”
Fin Si
Si (notaFinal >= 3) Y (notaFinal < 4) Entonces
    escriba “APROBADO”
Fin Si
Si notaFinal >= 4 Entonces
    escriba “EXCELENTEMENTE APROBADO”
Fin Si
```



5.2.2. Condicional compuesto.

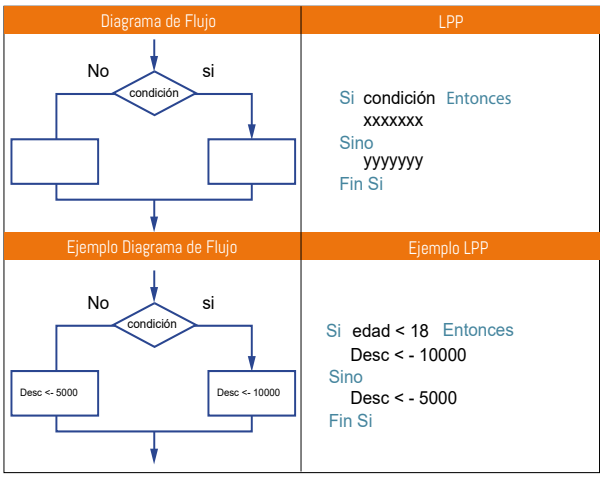
La estructura condicional compuesta verifica una condición y si esta es verdadera ejecuta unas instrucciones y si la condición es falsa, ejecuta otras instrucciones.

Su estructura es muy similar a la condicional simple, pero ahora se debe indicar las instrucciones que se deben ejecutar cuando la condición no se cumpla.

```
Inicia Robot
SI hay obstáculo
    Quitar
SINO
    Avanzar
FinSI

SI hay obstáculo Quitar SINO Avanzar
SI hay obstáculo Quitar SINO Avanzar
SI hay obstáculo Quitar SINO Avanzar
GirarIzquierda
SI hay obstáculo Quitar SINO Avanzar
SI hay obstáculo Quitar SINO Avanzar
GirarDerecha
SI hay obstáculo Quitar SINO Avanzar
SI hay obstáculo Quitar SINO Avanzar
SI hay obstáculo Quitar SINO Avanzar
Detener
Termina Robot
```

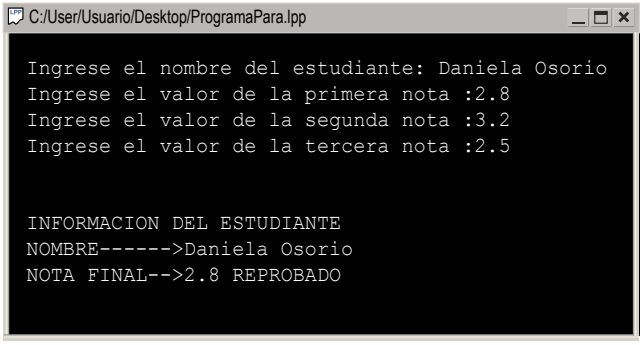
Sintaxis de una estructura condicional compuesta

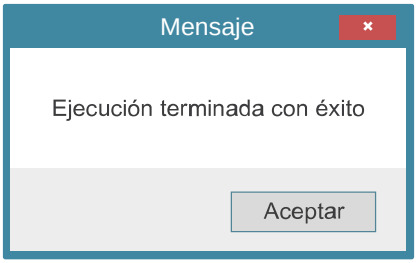


Se requiere una aplicación que lea el nombre de un estudiante y sus 3 notas parciales y presente un mensaje con su nombre y nota final. Si la nota final es inferior a 3, presentar el mensaje “REPROBADO”, en caso contrario, presentar el mensaje “APROBADO”.

```
//Declaración de Variables
Cadena [25] nombre
Real nota1, nota2, nota3, notaFinal

Inicio
    //Lectura de los datos de entrada
    escriba "Ingrese el nombre del estudiante:"
    lea nombre
    escriba "Ingrese el valor de la primera nota:"
    lea nota1
    escriba "Ingrese el valor de la segunda nota:"
    lea nota2
    escriba "Ingrese el valor de la tercera nota:"
    lea nota3
    //Cálculo de la nota final
    notaFinal <- (nota1 + nota2 + nota3) / 3
    //Escritura de la salida
    llamar nueva linea
    llamar nueva_linea
    escriba "INFORMACION DEL ESTUDIANTE"
    llamar nueva_linea
    escriba "NOMBRE------>",nombre
    llamar nueva_linea
    escriba "NOTA FINAL-->",notaFinal
    Si notaFinal < 3 Entonces
    escriba " REPROBADO"
  Sino
    escriba "APROBADO"
  Fin Si
Fin
```





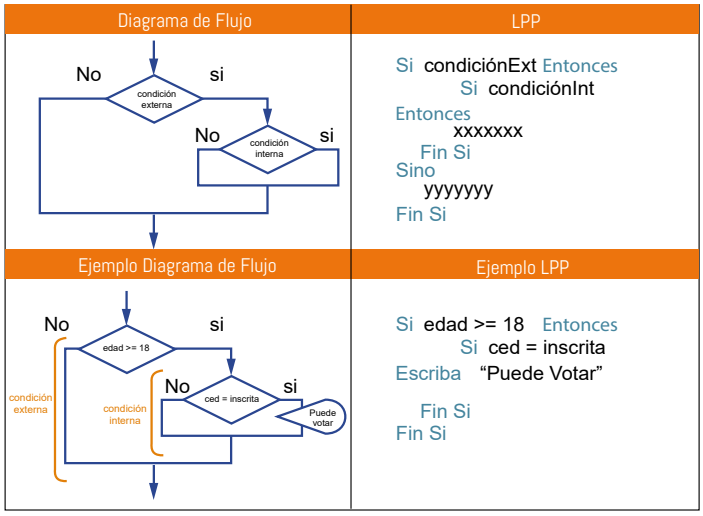
5.2.3. Condiciones anidadas.

Las Condiciones Anidadas son simplemente la definición de una condición al interior de otra, no se trata de una estructura diferente o nueva, pero se debe tener un especial cuidado con su implementación al interior de un programa debido a que cada una de las estructuras debe estar correctamente definida.

La condición interna puede estar en el flujo verdadero de la condición externa, en el flujo falso o en ambos. La condición interna, además, puede contener otras condiciones en su interior.

```
Inicia Robot
SI hay obstáculo Quitar
SI COLOR = AZUL Acomodar Izquierda
SINO Acomodar Derecha
SINO Avanzar
SI hay obstáculo Quitar
SI COLOR = AZUL Acomodar Izquierda
SINO Acomodar Derecha
SINO Avanzar
SI hay obstáculo Quitar
SI COLOR = AZUL Acomodar Izquierda
SINO Acomodar Derecha
SINO Avanzar
SI hay obstáculo Quitar
SI COLOR = AZUL Acomodar Izquierda
SINO Acomodar Derecha
SINO Avanzar
GirarIzquierda
SI hay obstáculo Quitar
SI COLOR = AZUL Acomodar Izquierda
SINO Acomodar Derecha
SINO Avanzar
```

Sintaxis de una estructura condicional anidada

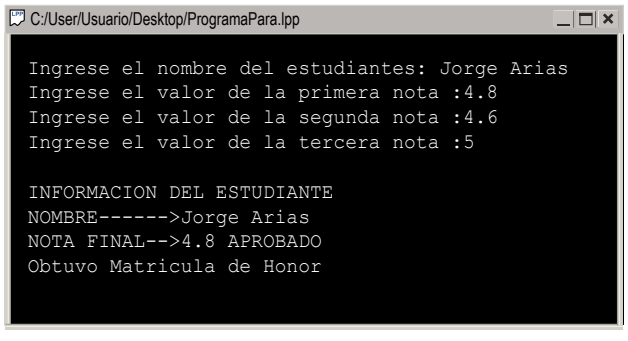


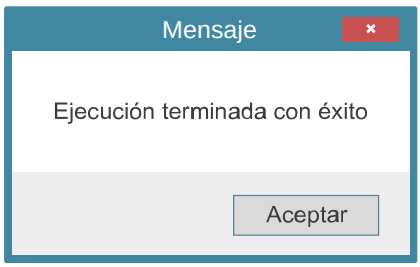

```
SI hay obstáculo Quitar
SI COLOR = AZUL Acomodar Izquierda
SINO Acomodar Derecha
SINO Avanzar
GirarDerecha
SI hay obstáculo Quitar
SI COLOR = AZUL Acomodar Izquierda
SINO Acomodar Derecha
SINO Avanzar
SI hay obstáculo Quitar
SI COLOR = AZUL Acomodar Izquierda
SINO Acomodar Derecha
SINO Avanzar
SI hay obstáculo Quitar
SI COLOR = AZUL Acomodar Izquierda
SINO Acomodar Derecha
SINO Avanzar
Detener
Termina Robot
```

Se requiere una aplicación que lea el nombre de un estudiante y sus 3 notas parciales y presente un mensaje con su nombre y nota final. Si la nota final es inferior a 3, presentar el mensaje "REPROBADO", en caso contrario, presentar el mensaje "APROBADO". A los estudiantes Aprobados cuya nota final esté por encima de 4.7 Indicarles que obtienen mención de honor.

```
//Declaración de Variables
Cadena [25] nombre
Real nota1, nota2, nota3, notaFinal

Inicio
    //Lectura de los datos de entrada
    escriba "Ingrese el nombre del estudiante:"
    lea nombre
    escriba "Ingrese el valor de la primera nota:"
    lea nota1
    escriba "Ingrese el valor de la segunda nota:"
    lea nota2
    escriba "Ingrese el valor de la tercera nota:"
    lea nota3
    //Cálculo de la nota final
    notaFinal <- (nota1 + nota2 + nota3) / 3Z
    //Escritura de la salida
    llamar nueva linea
    llamar nueva_linea
    escriba "INFORMACION DEL ESTUDIANTE"
    llamar nueva_linea
    escriba "NOMBRE------>",nombre
    llamar nueva_linea
    escriba "NOTA FINAL-->",notaFinal
    //Estructurar Condicional Doble
    Si notaFinal < 3 Entonces
    escriba " REPROBADO"
  Sino
    escriba "APROBADO"
    Si notaFinal > 4.7 Entonces
    escriba " Obtuvo Matrícula de Honor"
  Fin Si
Fin
```





6. Estructuras cíclicas indeterminadas.

A continuación, se definen los tipos de estructuras de programación cíclicas, las características de cada estructura y ejemplos de las mismas.



Estructuras Cíclicas Determinadas: son aquellas estructuras donde el número de ejecuciones (iteraciones) se conoce antes de ejecutar el ciclo.

Este tipo de estructura tiene la siguiente forma:

```
PARA <variable> ← <expresion1> hasta <expresion2> paso <expresion3> haga
        <acciones a repetir>
FIN PARA
```

Y tiene el siguiente significado:

Dado un valor inicial expresion1 que es asignado a la variable, esta se irá aumentando o disminuyendo de acuerdo a la expresion3 hasta llegar a la expresion2. Tener en cuenta que, si se omite el paso, significa que la variable aumentará de uno en uno (valor por defecto).

Estructuras Cíclicas Indeterminadas: son aquellas estructuras donde el número de ejecuciones (iteraciones) no se conoce con exactitud, y depende la cantidad de las ejecuciones de una condición o variable dentro del ciclo.

Este tipo de estructura tiene la siguiente forma:

```
MIENTRAS QUE <condición> haga
        <acciones a repetir>
FIN MIENTRAS
```

Y tiene el siguiente significado:

Dada una condición, que debe de cumplirse para que se siga ejecutando las acciones dentro del ciclo. Una vez la condición deja de cumplirse, el ciclo ya no se ejecuta.

Los ciclos se deben de programar para que se ejecuten un número finito de veces, de lo contrario nos encontraremos con un ciclo infinito y el algoritmo no funcionará.

Para conseguir que el ciclo sólo se repita un número finito de veces, debe de existir una condición de salida del mismo, es decir, una instrucción o situación en la que ya no sea necesario seguir repitiendo las instrucciones.

Con la herramienta LPP se pueden crear aplicaciones que hagan uso de las estructuras cíclicas PARA, MIENTRAS y REPITA, permitiendo adquirir los fundamentos necesarios para el manejo de estas estructuras de programación.

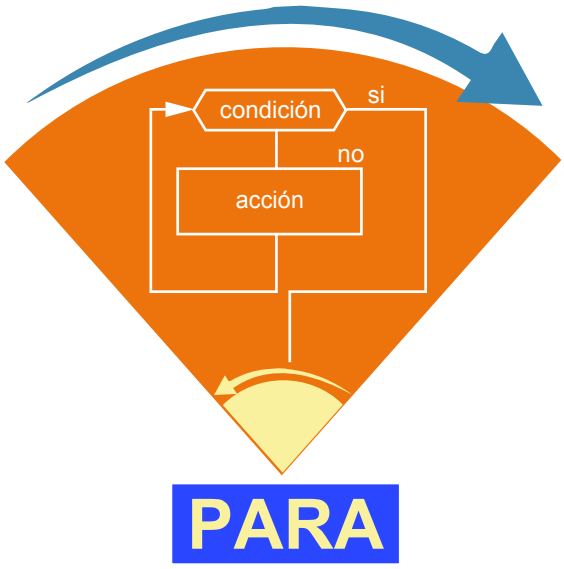
6.1. La estructura cíclica "PARA".

La estructura cíclica PARA, permite ejecutar una serie de instrucciones un número determinado de veces. Es habitual en programación que existan instrucciones que se deben ejecutar cíclicamente cierta cantidad de veces. Gracias a la estructura PARA, estas instrucciones se escriben una sola vez dentro de la estructura cíclica y la configuración de esta estructura es la que determina cuántas veces se deben ejecutar.

Sintaxis de una estructura Cíclica "PARA".

Sintaxis LPP
Para variable <- valor Inicial Hasta valor Final Haga //código que se desea repetir Fin Para
Ejemplo
Para x<--- 1 Hasta 100 Haga Escriba "Hola" Fin Para

El ejemplo anterior presenta 100 veces la palabra Hola en pantalla.



EJEMPLO: programa para el uso de ciclo "PARA".

Se requiere una aplicación que lea el nombre de 3 estudiantes de un salón de clase, las 2 notas parciales de cada uno y presente un mensaje con sus nombres y notas finales. Si la nota final es inferior a 3, presentar el mensaje "REPROBADO", en caso contrario presentar el mensaje "APROBADO" a cada estudiante.

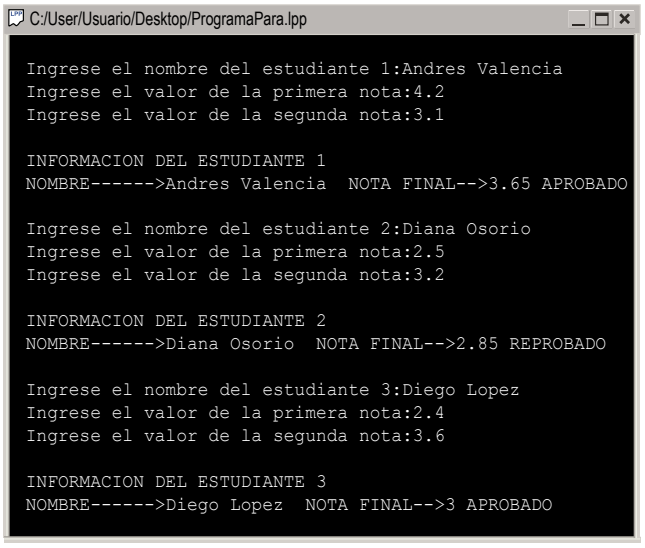
```
//Declaración de Variables
Cadena [25] nombre
Real nota1, nota2, nota3, notaFinal

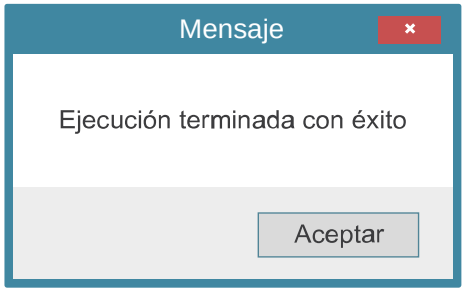
Inicio
    //Configuración del Ciclo PARA
    //La variable estudiante es la que controla el
    ciclo

    Para estudiante <----  1 Hasta 3  Haga
    //Lectura de los datos de entrada
    escriba  "Ingrese el nombre del estudiante:"
    ",estudiante,":"
    lea nombre
    escriba "Ingrese el valor de la primera nota:"
    lea nota1
    escriba "Ingrese el valor de la segunda nota:"
    lea nota2

    //Cálculo de la nota final
    notaFinal <- (nota1 + nota2 ) / 2
    //Escritura de la salida
    llamar nueva_linea
    escriba  "INFORMACION DEL ESTUDIANTE"
    ,estudiante
    llamar nueva_linea
    escriba "NOMBRE------>",nombre
    llamar nueva_linea
    escriba "NOTA FINAL-->",notaFinal
    //Estructurar Condicional Doble
    Si notaFinal < 3 Entonces
    escriba "REPROBADO"
  Sino
    escriba "APROBADO"
  Fin Si
    llamar nueva_linea
    llamar nueva_linea
    //Fin del ciclo PARA
Fin
```

El resultado del programa para el uso de ciclo "PARA" es el siguiente:





Se puede observar que el programa se ejecutó las tres veces indicadas en el ciclo para. Para este ejemplo, cada vez que se ejecute el programa, se repetirán tres veces las instrucciones indicadas en el ciclo.

Para cambiar el número de ejecuciones deberá cambiar la condición del ciclo PARA.

6.2. La estructura cíclica "MIENTRAS".

La estructura cíclica MIENTRAS, permite ejecutar una serie de instrucciones un número indeterminado de veces. La cantidad de veces que se repite el ciclo MIENTRAS depende del cumplimiento de una condición, por esta razón es frecuente que el programador no conozca de antemano cuántas veces el ciclo será ejecutado y esta es la principal diferencia con el Ciclo PARA.

¿Qué pasaría si no quisiéramos que nuestro Robot Cíclico realice su recorrido de manera automática un número determinado de veces, sino que lo hiciera hasta que una condición suceda, por ejemplo, una orden del usuario?

La estructura cíclica MIENTRAS soluciona esta necesidad, donde cada vez que se ejecuta el ciclo, se evalúa si la condición todavía es verdadera para saber si se debe realizar un nuevo ciclo o no.

Sintaxis de una estructura cíclica "MIENTRAS".

Sintaxis LPP

Mientras condición Haga
//código que se repite mientras la condición sea
//verdadera
Fin Mientras

Ejemplo
Mientras ahorro<10000 Haga Lea Dinero Ahorro<-ahorro+dinero Fin Mientras

El ejemplo anterior se ejecuta hasta que el ahorro es de $100.000 o mas.



EJEMPLO: programa para el uso de ciclo "MIENTRAS".

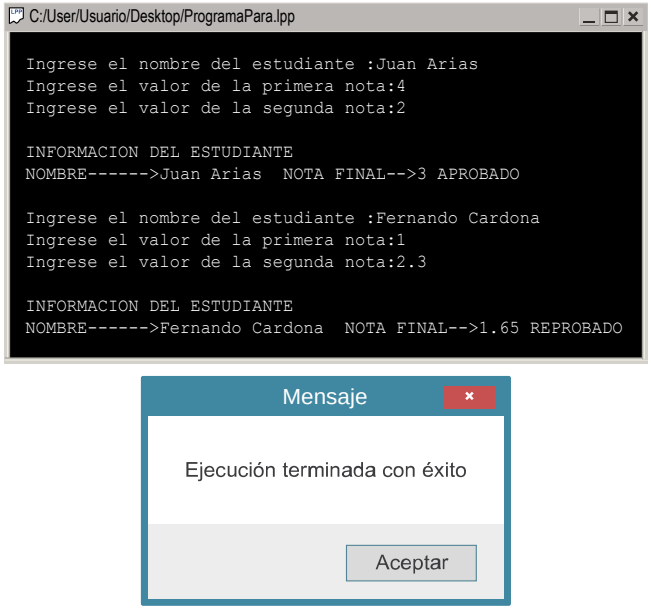
Se requiere una aplicación que lea constantemente el nombre de los estudiantes de un salón de clase, las 2 notas parciales de cada uno y presente un mensaje con sus nombres y notas finales. Si la nota final es inferior a 3, presentar el mensaje "REPROBADO", en caso contrario presentar el mensaje "APROBADO" a cada estudiante. Repetir este proceso hasta que alguno de ellos tenga una nota final por debajo de 2

```
//Declaración de Variables
Cadena [25] nombre
Real nota1, nota2, nota3, notaFinal

Inicio
    notaFinal <- 99 //se da un valor cualquiera
    mayor a 2 para que entre al ciclo
    //Configuración del Ciclo MIENTRAS
    Mientras notaFinal >= 2  Haga
    //Lectura de los datos de entrada
    //Lectura de los datos de entrada
```

```
    escriba "Ingrese el nombre del estudiante:"
    lea nombre
    escriba "Ingrese el valor de la primera nota:"
    lea nota1
    escriba "Ingrese el valor de la segunda nota:"
    lea nota2
    //Cálculo de la nota final
    notaFinal <--- (nota1 + nota2 ) / 2
    //Escritura de la salida
    llamar nueva_linea
    escriba "INFORMACION DEL ESTUDIANTE"
    llamar nueva_linea
    escriba "NOMBRE------>",nombre
    llamar nueva_linea
    escriba "NOTA FINAL-->",notaFinal
    //Estructurar Condicional Doble
    Si notaFinal < 3 Entonces
    escriba " REPROBADO"
  Sino
    escriba "APROBADO"
  Fin Si
    llamar nueva_linea
    llamar nueva_linea
    //Fin del ciclo MIENTRAS
Fin
```

El resultado del programa para el uso de ciclo "MIENTRAS" es el siguiente:



Se puede observar que el programa se ejecutó solamente dos veces porque la nota final del segundo estudiante fue menor a 2. Pero si este programa se ejecuta de nuevo, la cantidad de veces de ejecución del ciclo puede ser diferente, dependiendo de los datos ingresados por el usuario

6.3. La estructura cíclica "REPITA".

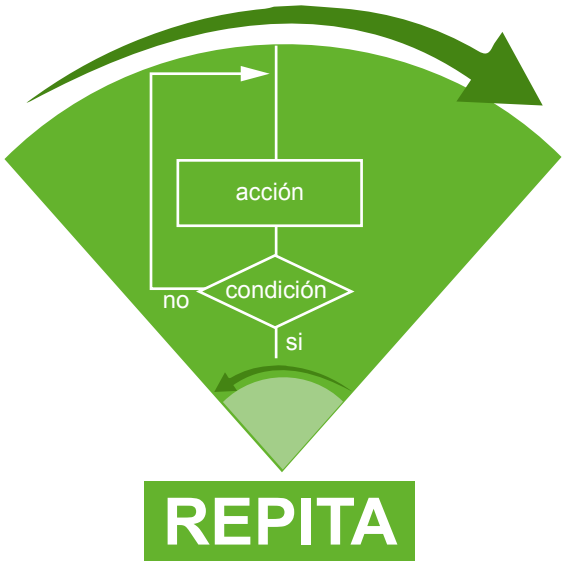
La estructura cíclica REPITA, al igual que la estructura cíclica MIENTRAS, se ejecuta un número indeterminado de veces, estas dos estructuras tienen un comportamiento similar, presentando su principal diferencia en el lugar de la estructura donde se evalúa la condición, dado que la estructura MIENTRAS evalúa la condición del ciclo al inicio del mismo y la estructura REPITA lo hace al final del mismo, de este modo, en la estructura cíclica REPITA, el programador garantiza que el ciclo se ejecuta al menos una vez.

Sintaxis de una estructura Cíclica "REPITA".

Sintaxis LPP	Ejemplo

Repita
//código que se repite mientras la condición sea
//verdadera
Fin Repita
	Repita
	Lea Dinero
	Ahorro<-ahorro+dinero
	Hasta ahorro >= 100000

El ejemplo anterior se ejecuta hasta que el ahorro es de $100.000 o más.

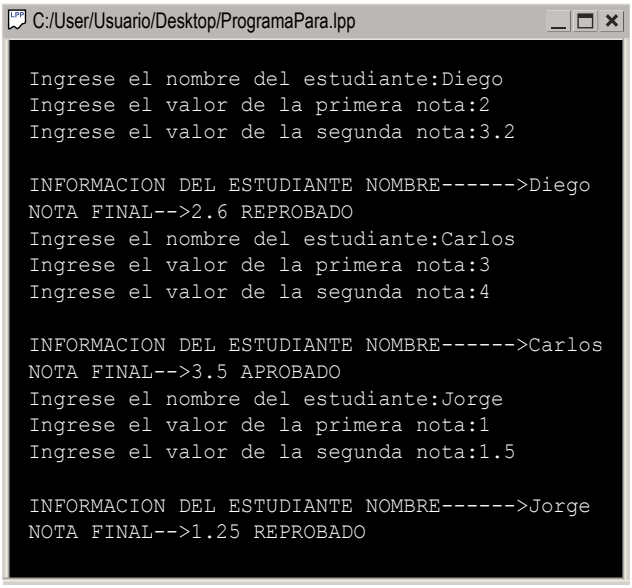


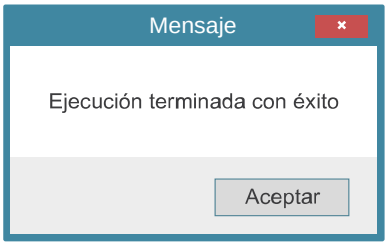
EJEMPLO: Programa para el uso de ciclo “MIENTRAS”.

```
//Declaración de Variables
Cadena [25] nombre
Real nota1, nota2, nota3, notaFinal

Inicio
    //Inicio del ciclo Repita
    Repita
    //Lectura de los datos de entrada
    llamar nueva_linea
    escriba “Ingrese el nombre del estudiante:”
    lea nombre
    escriba “Ingrese el valor de la primera nota:”
    lea nota1
    escriba “Ingrese el valor de la segunda nota:”
    lea nota2
    //Cálculo de la nota final
    notaFinal <- (nota1 + nota2 ) / 2
    //Escritura de la salida
    llamar nueva_linea
    llamar nueva_linea
    escriba “INFORMACION DEL ESTUDIANTE”
    llamar nueva_linea
    escriba “NOMBRE------>”,nombre
    llamar nueva_linea
    escriba “NOTA FINAL-->”,notaFinal
    Si notaFinal < 3 Entonces
    escriba “ REPROBADO”
  Sino
    escriba “APROBADO”
  Fin Si
    Hasta notaFinal < 2
    //Fin del ciclo REPITA
Fin
```

El resultado del programa para el uso de ciclo “MIENTRAS” es el siguiente:



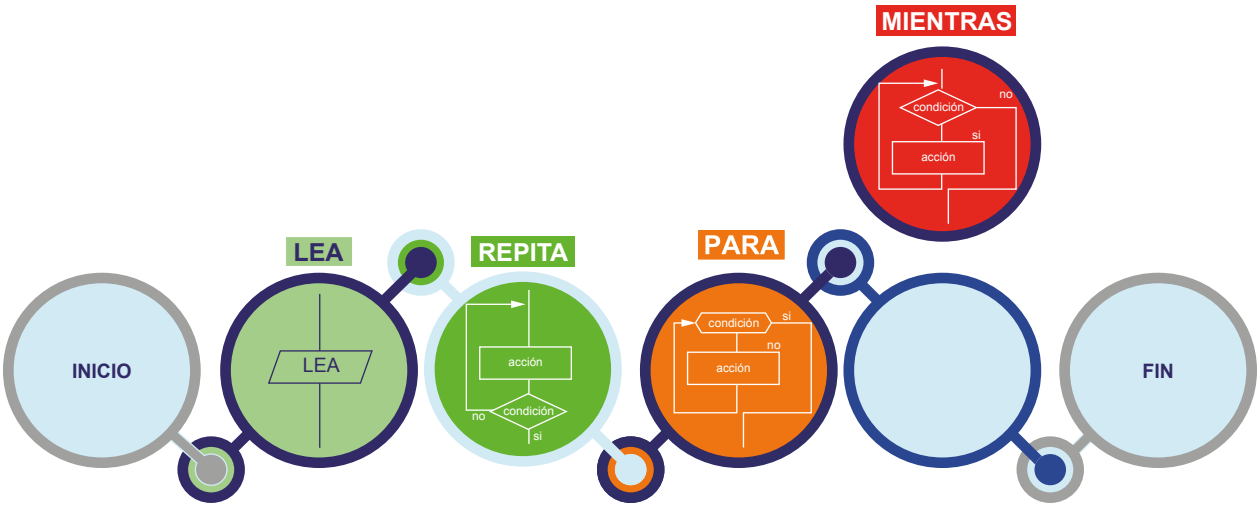


El programa con el ciclo REPITA presenta el mismo comportamiento que el programa con el ciclo MIENTRAS, el cambio está en el código utilizado en cada una de las estructuras; por lo tanto, el uso de un ciclo o el otro es indiferente cuando se requiere una estructura cíclica indeterminada.

7. Armando el rompecabezas con estructuras de programación.

Cómo vimos, las estructuras de programación son de tipo secuenciales, condicionales y cíclicas y con estos 3 tipos de estructuras se pueden desarrollar una gran cantidad de aplicaciones de software, la clave está en la organización lógica que se le den a las instrucciones y estructuras para resolver un problema mediante un programa de computador.

La organización de las estructuras en un programa depende exclusivamente del problema que se desea resolver, por lo tanto, es posible tener un programa con una estructura condicional al interior de una estructura cíclica o un ciclo al interior de otro (ciclos anidados), o una estructura cíclica al interior de una condicional y estructuras secuenciales al interior o por fuera de estructuras cíclicas o condicionales. Por esta razón, el desarrollo de un programa es similar a la construcción de un rompecabezas donde cada elemento debe estar en el lugar adecuado para interactuar con los demás elementos y así construir una solución integral a un problema determinado.



8. Ejemplo de codificación y ejecución de un programa con diferentes estructuras de programación.

Se desea desarrollar una aplicación que lea una cantidad determinada de números y para cada uno indique si se trata de un número primo o no. (Un número primo es aquel que solo es divisible por 1 o por sí mismo).

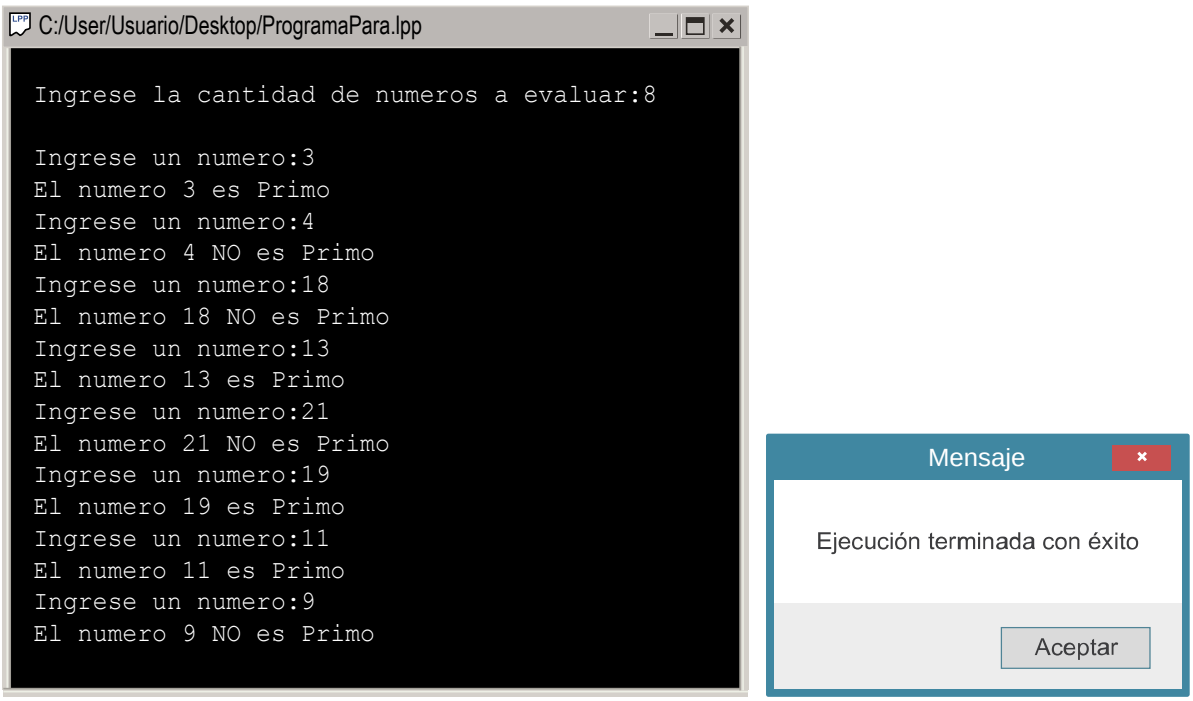
En el siguiente ejemplo se combinan diferentes estructuras de programación:

Ciclo PARA que recorre desde 1 hasta la cantidad de números leída.
Ciclo Repita que se ejecuta hasta que exista más de 2 divisores o el índice sea mayor que el número.
Estructura condicional SI que evalúa la cantidad de divisores.

```
//Declaración de variables
Entero cantidadNumeros, numero, cantidadDivisores, x, indice
Inicio
//Instrucciones Secuenciales
escriba "Ingrese la cantidad de numeros a evaluar:"
lea cantidadNumeros
//Estructura cíclica PARA, permite evaluar la cantidad de numeros deseada
Para x <- 1 Hasta cantidadNumeros Haga
   llamar nueva_linea
   escriba "Ingrese un numero:"
   lea numero
   indice <---- 1
   cantidadDivisores <---- 0
   //Estructura cíclica REPITA, buscar los divisores de cada número leído
   Repita
      //Estructura condicional simple, permite incrementar la cantidad de divisores
      Si numero mod indice = 0 Entonces
         cantidadDivisores <- cantidadDivisores + 1
      Fin Si
      indice <- indice + 1
   Hasta (cantidadDivisores > 2) O (indice > numero)
   //Fin estructura cíclica REPITA
   //Estructura Condicional compuesta, permite saber si el número evaluado es primo
   Si cantidadDivisores = 2 Entonces
      escriba "El numero ", numero, " es Primo"
   Sino
      escriba "El numero ", numero, " NO es Primo"
   Fin Si
Fin Para
//Fin Estructura cíclica PARA
Fin
```

Estructuras secuenciales | Estructuras condicionales | Estructuras cíclicas

El resultado del programa con diferentes estructuras de programación es el siguiente:



De esta manera, podemos observar como las diferentes estructuras de programación se pueden combinar para lograr dar solución a un determinado problema.

GLOSARIO

API: interfaz de programación de aplicaciones, es utilizada en procesos de programación orientada a objetos.

EXT3: (third extended filesystem o “tercer sistema de archivos extendido”), es el sistema de archivo más usado en distribuciones Linux, aunque en la actualidad está siendo remplazado por su sucesor, ext4

DATO: es una representación simbólica numérica, alfabética, algorítmica que puede ser un atributo o carácterística. Este, procesado se convierte en información.

INSTRUCCIÓN: una instrucción es una unidad de creación de procedimientos a partir de la cual se construyen los programas.

LPP: Lenguaje de Programación para Principiantes.

SINTAXIS: conjunto de normas que regulan la codificación de un programa.

VARIABLE: en ellas se pueden almacenar valores y son nombradas con identificadores, es decir nombres para poder identificarlas.

BIBLIOGRAFÍA

Castillo Suazo, R. (2001). *Programación en LPP*. *PSeInt*.

Pressman, R. (2010). *Ingeniería del software, un enfoque práctico*. Bogotá: McGraw-Hill.

Sommerville, I. (2005). *Ingeniería del software.* Madrid: Pearson.

CONTROL DEL DOCUMENTO

CONSTRUCCIÓN OBJETO DE APRENDIZAJE

FUNDAMENTOS DE PROGRAMACIÓN ESTRUCTURADA	
Centro Industrial de Mantenimiento Integral - CIMI Regional Santander	
Líder línea de producción:	Santiago Lozada Garcés
Asesores pedagógicos:	Rosa Elvia Quintero Guasca Claudia Milena Hernández Naranjo
Líder expertos temáticos:	Rita Rubiela Rincón Badillo
Experto temático:	Andres Julian Valencia Osorio. (V1)
Experto temático:	Edgar Eduardo Vega. (V2)
Diseño multimedia:	Tirso Fernán Tabares Carreño
Programador:	Francisco José Lizcano Reyes
Producción de audio:	Víctor Hugo Tabares Carreño

creative commons

BY NC SA

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 la licencia que el trabajo original.