



C
O
D
I
G
O

L
P
P

ALGORITMOS Y PROGRAMACIÓN BÁSICA EN LPP

Autores

Pedro Harvey Álvarez Sánchez
Alberto Bravo Buchelly
Freddy Alonso Vidal Alegría
Gabriel Elías Chanchí Golondrino

C
L
O
D
E
P
P

ALGORITMOS Y PROGRAMACIÓN BÁSICA EN LPP

Autores:

Pedro Harvey Álvarez Sánchez

Alberto Bravo Buchelly

Fredy Alonso Vidal Alegría

Institución Universitaria Colegio Mayor del Cauca

Gabriel Elías Chanchí Golondrino

Universidad de Cartagena

INTRODUCCIÓN

Este libro tiene como propósito servir como una guía para explorar los conceptos esenciales de algoritmos y programación básica. Se hará uso del pseudocódigo como herramienta principal, diseñado y ejecutado para el entorno de desarrollo LPP, con el fin de facilitar la comprensión y aplicación de los fundamentos de la programación. A lo largo de sus capítulos se estudiará cómo crear algoritmos para diferentes tipos de ejercicios tanto sencillos como complejos. Cada capítulo contará con ejemplos prácticos y ejercicios que te ayudarán a poner en práctica lo aprendido y fortalecer los conocimientos en relación con los temas estudiados.

Por consiguiente, aprender a elaborar algoritmos utilizando pseudocódigo en un lenguaje de programación para personas inexpertos LPP, es un paso esencial para desarrollar habilidades sólidas en programación. Desarrollar programas no se limita simplemente a escribir código, implica cultivar una manera de pensar lógica y organizada para dar órdenes una máquina, esencial para la resolución de los ejercicios propuestos. En este sentido, los algoritmos son fundamentales.

En este orden de ideas, el pseudocódigo es una herramienta efectiva que permite expresar algoritmos de manera clara y sencilla, evitando las complicaciones sintácticas de un lenguaje de programación específico. Al trabajar con pseudocódigo, los principiantes pueden centrarse en la lógica del problema y en cómo descomponerlo en pasos más simples, sin perderse en detalles técnicos. Esta metodología no solo facilita el proceso de aprendizaje, sino que también contribuye al desarrollo de habilidades importantes como el pensamiento lógico y la capacidad de resolución de problemas.

TABLA DE CONTENIDO

1 Capítulo - Conceptos Básicos.....	15
1.1 Contenido conceptos básicos	15
1.1.1 Temas del Capítulo.....	15
1.2 Capítulo 1. Conceptos Básicos	15
1.3 Definición de algoritmo	16
1.3.1 Problema	16
1.3.2 Solución:	16
1.3.3 ¿Preguntas?	17
1.3.4 Características del algoritmo.....	17
1.4 Programa.....	18
1.5 Lenguaje de programación	18
1.5.1 Lenguaje Máquina.....	19
1.5.2 Lenguajes de Bajo Nivel.....	19
1.5.3 Lenguajes de Alto Nivel.....	19
1.5.4 Lenguajes de propósito específico	20
1.5.5 Importancia de los lenguajes de programación.....	20
1.6 Traductores de Lenguaje	20
1.6.1 Compilador.....	21
1.6.2 Intérprete	21
1.6.3 Ensamblador.....	21
1.7 Fases para desarrollar un programa.....	22
1.7.1 Fase de análisis	22
1.7.2 Fase de diseño	22
1.7.3 Fase de codificación.....	22
1.7.4 Fase de pruebas.....	23
1.7.5 Despliegue	23
1.7.6 Fase de documentación y mantenimiento:	23
1.8 Tipos de algoritmos	24
1.9 Pseudocódigo:	25
1.9.1 Tablas de decisión	25
1.9.2 Diagramas de flujo u ordinogramas.....	25
1.9.3 Entornos para diseños de diagramas de flujo.....	25
1.9.4 REPRESENTACIÓN DE LOS DIAGRAMAS DE FLUJO.....	27
1.9.5 Diagramas de flujo básico de inicio y fin	28
1.9.6 Diagrama con comentario.....	29
1.9.7 Diagramas declaración de variables.....	29

1.9.8	Diagramas entrada y salida.....	30
1.9.9	Diagramas condicionales por asignación.....	30
1.9.10	Diagramas condicionales con petición	31
1.9.11	Diagramas con funciones	33
1.10	Pseudocódigo	33
1.11	Definición de LPP	34
1.12	Prueba de Escritorio	35
1.13	Ejercicios de la unidad	37
2	CAPÍTULO. INSTALACIÓN DE LPP	38
2.1	Contenido Instalación de LPP	38
2.1.1	Temas del capítulo	38
2.2	Capítulo 2. Instalación en LPP	38
2.3	Pasos para la instalación de LPP	39
2.4	Generación de un programa inicial LPP	42
2.5	Ejecución de un programa inicial en LPP	45
2.6	Opciones adicionales en LPP	48
2.7	Ejercicios de la unidad	53
2.8	Bibliografía	53
3	CAPÍTULO – CONCEPTOS DE BÁSICOS DE PROGRAMACION.....	54
3.1	Temas del capítulo	54
3.1.1	Contenido	54
3.2	Estructura de un programa en LPP	56
3.3	Palabras reservadas	58
3.4	Comentarios en LPP	59
3.4.1	Comentarios para documentar la funcionalidad.....	59
3.4.2	Comentarios para inhabilitar el código.....	59
3.5	Nomenclaturas de programación	60
3.5.1	<i>Camel Case</i>	60
3.5.2	<i>Snake Case</i>	61
3.5.3	<i>Pascal Case</i>	61
3.5.4	Nomenclatura de identificadores en programación.....	61
3.6	Identificador	63
3.6.1	Recomendaciones.....	64
3.6.2	Restricciones	67
3.6.3	Recomendaciones para las funciones.....	68
3.7	Entrada proceso y salida de datos	70

3.7.1	Entrada (lea).....	71
3.7.2	Proceso.....	72
3.7.3	Salida.....	74
3.7.4	Proceso interno de las variables de la memoria.....	77
3.8	Datos, tipos de datos	80
3.8.1	Datos.....	80
3.9	Tipos de datos	80
3.9.1	Tipos de datos básicos	80
3.9.2	Tipos de datos compuestos.....	87
3.10	Datos numéricos	81
3.11	Tipo entero	81
3.12	Tipo real	82
3.13	Datos lógicos	82
3.14	Datos tipo texto	83
3.14.1	Tipo carácter	83
3.15	Representación interna del código ASCII	84
3.15.1	ASCII (American Standard Code for Information Interchange):.....	84
3.15.2	Tipo de dato cadena.....	86
3.15.3	Asignación de datos	88
3.16	Operadores Matemáticos	90
3.16.1	Operadores aritméticos.....	91
3.16.2	Operador mod	92
3.17	Operador div	94
3.17.1	Operador ^	95
3.17.2	Operador de agrupar expresiones ():.....	96
3.18	Operadores relacionales	97
3.19	Operadores lógicos	98
3.20	Jerarquía de operadores	98
3.20.1	Paréntesis internos prioridad 1	100
3.20.2	Paréntesis prioridad 2	101
3.20.3	Exponenciación prioridad 3	101
3.20.4	Mod y Div prioridad 4	102
3.20.5	Multiplicación (*) y división (/) prioridad 5	103
3.20.6	Suma (+) y resta (-) prioridad 6.....	104
3.20.7	Operaciones con radicales en LPP	104
4	CAPÍTULO ESTRUCTURAS DE CONTROL.....	108
4.1	Temas del capítulo	108
4.2	Concepto estructuras de control de selección	109

4.2.1	Flujo de control.....	109
4.2.2	Operadores relacionales	111
4.3	Estructuras de selección simple	112
4.4	Estructuras de selección dobles	115
4.5	Expresiones lógicas	118
4.6	Lógica Proposicional	119
4.6.1	La lógica Proposicional	119
4.6.2	Proposición:.....	120
4.6.3	Conjunción (Y).....	120
4.6.4	Disyunción (O).....	120
4.6.5	Negación (NO)	121
4.7	Estructuras de selección anidadas	126
4.8	Estructuras de selección múltiples	129
4.9	Ejercicios de la unidad	132
4.10	Bibliografía.....	Error! Bookmark not defined.

5 CAPÍTULO - ESTRUCTURAS DE CONTROL PARA REPETICIÓN *Error!* *Bookmark not defined.*

5.1	Temas del capítulo	Error! Bookmark not defined.
5.2	Ciclos o bucles	Error! Bookmark not defined.
5.2.1	Partes del ciclo	Error! Bookmark not defined.
5.2.2	En qué momento debo usar un ciclo.....	Error! Bookmark not defined.
5.2.3	Tipos de bucles en LPP	Error! Bookmark not defined.
5.3	Bucle Mientras	Error! Bookmark not defined.
5.4	Ciclo Para	Error! Bookmark not defined.
5.5	Pseudocódigo Bucle Para – Hasta.....	Error! Bookmark not defined.
5.6	Acumulador y Contador	Error! Bookmark not defined.
5.6.1	Contador	Error! Bookmark not defined.
5.6.2	Uso de contadores	Error! Bookmark not defined.
5.7	Acumulador.....	Error! Bookmark not defined.
5.7.1	Consideraciones	Error! Bookmark not defined.
5.7.2	Ejemplo de uso del acumulador	Error! Bookmark not defined.
5.7.3	Ejemplo acumulador edades con ciclo mientras:.....	Error! Bookmark not defined.
5.7.4	Ejemplo acumulador, contador y promedio	Error! Bookmark not defined.
5.7.5	Ejemplo con prueba de escritorio	Error! Bookmark not defined.
5.7.6	Prueba de escritorio.....	Error! Bookmark not defined.
5.8	Ciclos o bucles anidados	Error! Bookmark not defined.
5.8.1	PRUEBA DE ESCRITORIO	Error! Bookmark not defined.

5.9 Ejercicios de la unidad _____ **Error! Bookmark not defined.**

6 CAPITULO - VECTORES UNIDIMENSIONALES EN LPP.....*Error! Bookmark not defined.*

6.1 Temas del capítulo _____ **Error! Bookmark not defined.**

6.2 Conceptos vectores unidimensionales **Error! Bookmark not defined.**

6.3 Vector unidimensional _____ **Error! Bookmark not defined.**

6.4 Declaración vector unidimensional ____ **Error! Bookmark not defined.**

6.4.1 Asignación de valores en un vector unidimensional **Error! Bookmark not defined.**

6.4.2 Asignación de valores en un vector unidimensional **Error! Bookmark not defined.**

6.4.3 Asignación de valores con ciclo **Error! Bookmark not defined.**

6.4.4 Imprimir un elemento **Error! Bookmark not defined.**

6.4.5 Recorrer un vector unidimensional..... **Error! Bookmark not defined.**

6.4.6 Operaciones con vector unidimensionales..... **Error! Bookmark not defined.**

6.5 Ejercicios de la unidad _____ **Error! Bookmark not defined.**

7 VECTORES BIDIMENSIONALES*Error! Bookmark not defined.*

7.1 Temas del capítulo 7 _____ **Error! Bookmark not defined.**

7.2 Capítulo 7. Matrices _____ **Error! Bookmark not defined.**

7.3 Concepto Matrices _____ **Error! Bookmark not defined.**

7.4 Declaración de una Matriz _____ **Error! Bookmark not defined.**

7.5 Llenar una Matriz _____ **Error! Bookmark not defined.**

7.6 Imprimir elementos de la matriz ____ **Error! Bookmark not defined.**

7.7 Recorrer e imprimir la matriz _____ **Error! Bookmark not defined.**

7.8 Operaciones con Matrices _____ **Error! Bookmark not defined.**

7.9 Ejercicios de la unidad _____ **Error! Bookmark not defined.**

8 CAPITULO - PROCEDIMIENTOS Y FUNCIONES...*Error! Bookmark not defined.*

8.1 Temas del capítulo _____ **Error! Bookmark not defined.**

8.2 Procedimientos y Funciones _____ **Error! Bookmark not defined.**

8.3 Concepto de función _____ **Error! Bookmark not defined.**

8.3.1 Ventajas al usar funciones..... **Error! Bookmark not defined.**

8.3.2 Estructura de la función en LPP..... **Error! Bookmark not defined.**

8.3.3 Como funciona **Error! Bookmark not defined.**

8.3.4 Parámetros, Argumento, Valor **Error! Bookmark not defined.**

8.3.5 Función sin parámetros..... **Error! Bookmark not defined.**

8.3.6 Función con retorno **Error! Bookmark not defined.**

8.3.7 Función sin retorno **Error! Bookmark not defined.**

8.4	Concepto de procedimiento	Error! Bookmark not defined.
8.5	Ejercicios de la unidad	Error! Bookmark not defined.
8.6	Bibliografía	135

Tabla de Figuras

Figura 1-1 Programa	18
Figura 1-2 Fases para desarrollar un programa	24
Figura 1-3 Tipos de algoritmos	24
Figura 1-4 Inicio – fin	28
Figura 1-5 Comentario	29
Figura 1-6 Declaración de variables	29
Figura 1-7 Entrada y salida	30
Figura 1-8 Condicionales por petición	30
Figura 1-9 Condicionales con petición	31
Figura 1-10 Diagramas con funciones	33
Figura 2-1 Instalador	39
Figura 2-2 Ventana de instalación	39
Figura 2-3 Selección de idioma	40
Figura 2-4 Progreso de instalación	40
Figura 2-5 Instalación finalizada	41
Figura 2-6 Lanzando LPP	41
Figura 2-7 Entorno de trabajo LPP	42
Figura 2-8 Editor de LPP	43
Figura 2-9 Ejemplo básico LPP	43

Figura 2-10 Opción guardar abreviada	44
Figura 2-11 Opción guardar del menú.....	44
Figura 2-12 Opción abrir abreviada.....	44
Figura 2-13 Opción abrir del menú archivo	45
Figura 2-14 Opción compilar	46
Figura 2-15 Opción ejecutar abreviada	46
Figura 2-16 Opción ejecutar del menú programa	47
Figura 2-17 Resultado de la ejecución del programa	47
Figura 2-18 Generar código en C++	48
Figura 2-19 Código generado en C++	49
Figura 2-20 Generar código en Ada95.....	49
Figura 3-1 Área de código	56
Figura 3-2 Estructura en LPP	57
Figura 3-3 Comentarios de información	59
Figura 3-4 Comentarios en LPP.....	60
Figura 3-5 Entrada proceso y salida	71
Figura 3-6 Entrada de datos	72
Figura 3-7 Proceso de datos	73
Figura 3-8 Salida de datos	75
Figura 3-9 Salidas con variables	75
Figura 3-10 Entrada, proceso y salida	76
Figura 3-11 Aproximación de variables	77
Figura 3-12 Tipo entero	81
Figura 3-13 Tipo real	82
Figura 3-14 Tipo de datos lógicos.....	83
Figura 3-15 Tipo de datos carácter.....	84
Figura 3-16 Tipo cadena	87
Figura 3-17 Asignaciones.....	88
Figura 3-18 Constantes.....	90
Figura 3-19 Código con operadores.....	92
Figura 3-20 División con mod	93
Figura 3-21 División con mod en LPP.....	94
Figura 3-22 División con div.....	94
Figura 3-23 División entera	95
Figura 3-24 Agrupación con paréntesis	97
Figura 3-25 Paréntesis internos.....	100
Figura 3-26 Prioridad paréntesis	101
Figura 3-27 Prioridad exponentiación	102
Figura 3-28 Prioridad mod div.....	103
Figura 3-29 Prioridad multiplicación división	103
Figura 3-30 Prioridad suma y resta	104
Figura 3-31 Operación con la raíz enésima	105
Figura 4-1 Control de selección	109
Figura 4-2 Estructuras de control	111
Figura 4-3 Condición simple	112
Figura 4-4 Selección simple	113
Figura 4-5 Código selección simple.....	113
Figura 4-6 Ejemplo booleano y carácter.....	114
Figura 4-7 Selección doble	115

Figura 4-8 Diagrama de flujo Selección con doble	115
Figura 4-9 Diagrama de selección doble	117
Figura 4-10 Código de selección doble	117
Figura 4-11 Lógica de proposicional.....	119
Figura 4-12 Proposicional simple.....	121
Figura 4-13 Conector Y	122
Figura 4-14 Conector O.....	123
Figura 4-15 Conector negación.....	124
Figura 4-16 Selección con conectores.....	125
Figura 4-17 Selección anidada.....	127
Figura 4-18 Condicionales en cascada.....	127
Figura 4-19 Ejemplo de selección anidada	128
Figura 4-20 Selección múltiple.....	129
Figura 4-21 Selección múltiple.....	130
Figura 4-22 Ejemplo selección múltiple	131
Figura 5-1 Ciclos o bucles	Error! Bookmark not defined.
Figura 5-2 Inicio del ciclo	Error! Bookmark not defined.
Figura 5-3 Fin del ciclo	Error! Bookmark not defined.
Figura 5-4 Condición verdadera	Error! Bookmark not defined.
Figura 5-5 Incremento del contador	Error! Bookmark not defined.
Figura 5-6 Diagrama ciclo mientras	Error! Bookmark not defined.
Figura 5-7 Bucle Mientras	Error! Bookmark not defined.
Figura 5-8 Ciclo cuenta números	Error! Bookmark not defined.
Figura 5-9 Ciclo pares	Error! Bookmark not defined.
Figura 5-10 Ciclo del 50 al 100	Error! Bookmark not defined.
Figura 5-11 Disminución del contador.....	Error! Bookmark not defined.
Figura 5-12 Ciclo sin contador	Error! Bookmark not defined.
Figura 5-13 Diagrama para	Error! Bookmark not defined.
Figura 5-14 Ciclo para – hasta.....	Error! Bookmark not defined.
Figura 5-15 Descripción ciclo para.....	Error! Bookmark not defined.
Figura 5-16 Ciclo cuenta géneros	Error! Bookmark not defined.
Figura 5-17 Ciclo cuenta votos	Error! Bookmark not defined.
Figura 5-18 Diagrama acumulador	Error! Bookmark not defined.
Figura 5-19 Contadores y acumuladores.....	Error! Bookmark not defined.
Figura 5-20 Ciclo mientras suma edades.....	Error! Bookmark not defined.
Figura 5-21 Ciclo profesión.....	Error! Bookmark not defined.
Figura 5-22 Ciclo tabla del 7.....	Error! Bookmark not defined.
Figura 5-23 Promedio usando ciclo mientras y para	Error! Bookmark not defined.
Figura 5-24 Salida ciclo mientras y para	Error! Bookmark not defined.
Figura 5-25 Ciclos anidados.....	Error! Bookmark not defined.
Figura 5-26 Sintaxis ciclos anidados	Error! Bookmark not defined.
Figura 5-27 Ciclos anidados.....	Error! Bookmark not defined.
Figura 5-28 Ciclo anidado tablas de multiplicar	Error! Bookmark not defined.
Figura 6-1 Vector unidimensional.....	Error! Bookmark not defined.
Figura 6-2 Representación del vector unidimensional.....	Error! Bookmark not defined.
Figura 6-3 Asignación lista de números.....	Error! Bookmark not defined.
Figura 6-4 Representación lista de números	Error! Bookmark not defined.
Figura 6-5 Código asignación lista de números.....	Error! Bookmark not defined.
Figura 6-6 Representación lista de números	Error! Bookmark not defined.

Figura 6-7 Impresión lista de números	Error! Bookmark not defined.
Figura 6-8 Impresión lista de números	Error! Bookmark not defined.
Figura 6-9 Impresión lista de números con ciclo	Error! Bookmark not defined.
Figura 6-10 Llenar lista de números	Error! Bookmark not defined.
Figura 7-1 Vector bidimensional.....	Error! Bookmark not defined.
Figura 7-2 Representación de matriz.....	Error! Bookmark not defined.
Figura 7-3 Declaración matriz	Error! Bookmark not defined.
Figura 7-4 Llenar matriz	Error! Bookmark not defined.
Figura 7-5 Simulación de almacenamiento de matriz	Error! Bookmark not defined.
Figura 7-6 Almacenamiento con ciclos.....	Error! Bookmark not defined.
Figura 7-7 Primer iteración de las notas.....	Error! Bookmark not defined.
Figura 7-8 Representación iteraciones de la matriz	Error! Bookmark not defined.
Figura 7-9 Impresión matriz por consola.....	Error! Bookmark not defined.
Figura 7-10 Código imprimir una posición.....	Error! Bookmark not defined.
Figura 7-11 Representación matriz llena.....	Error! Bookmark not defined.
Figura 7-12 Código llenar matriz nombres	Error! Bookmark not defined.
Figura 7-13 Impresión por ciclos.....	Error! Bookmark not defined.
Figura 7-14 Operación en matrices	Error! Bookmark not defined.
Figura 8-1Función saluda	Error! Bookmark not defined.
Figura 8-2 Sintaxis de la función.....	Error! Bookmark not defined.
Figura 8-3 Función con parámetros.....	Error! Bookmark not defined.
Figura 8-4 Función sin parámetros.....	Error! Bookmark not defined.
Figura 8-5 Función con retorno	Error! Bookmark not defined.
Figura 8-6 Función sin retorno.....	Error! Bookmark not defined.
Figura 8-7 Función sumar.....	Error! Bookmark not defined.
Figura 8-8 Función que retorna resultado	Error! Bookmark not defined.
Figura 8-9 Función que retorna resultado	Error! Bookmark not defined.
Figura 8-10 Procedimiento listarPares	Error! Bookmark not defined.
Figura 8-11 Sintaxis del procedimiento	Error! Bookmark not defined.
Figura 8-12 Procedimiento sin parámetros	Error! Bookmark not defined.
Figura 8-13 Procedimiento con parámetros.....	Error! Bookmark not defined.
Figura 8-14 Variable de alcance local.....	Error! Bookmark not defined.
Figura 8-16 Variable alcance global	Error! Bookmark not defined.

Lista de tablas

Lista de tablas

Tabla 1-1 Aplicaciones para diagramas de flujo.....	26
Tabla 1-2 Diagramas de flujo	28
Tabla 1-3 Prueba de escritorio.....	35
Tabla 3-1 Palabras reservadas	58
Tabla 3-2 Estilos de nomenclatura	63
Tabla 3-3 Lenguajes y nomenclatura.....	63
Tabla 3-4 Restricciones de palabras reservadas.....	65
Tabla 3-5 Longitud de variables	65
Tabla 3-6 Recomendación estilo de identificador.....	66
Tabla 3-7 Restricciones de palabras compuestas.....	67
Tabla 3-8 Restricciones de letra.....	68
Tabla 3-9 Restricciones de funciones.....	68
Tabla 3-10 Identificador de funciones	69
Tabla 3-11 Restricciones de constantes.....	69
Tabla 3-12 Indicadores válidos	70
Tabla 3-13 Indicadores no válidos	70
Tabla 3-142 Representación Dirección de memoria ficticia	79
Tabla 3-15 Tipos enteros	81
Tabla 3-16 Tipo de datos lógicos	82
Tabla 3-17 ASCII Caracteres especiales y números.....	85
Tabla 3-18 Código ASCII, Caracteres alfabéticos.....	86
Tabla 3-19 Operadores matemáticos	91
Tabla 3-20 Operadores matemáticos especiales.....	92
Tabla 3-21 Operadores relacionales.....	97
Tabla 3-22 Operadores lógicos	98
Tabla 3-23 Jerarquía de operadores.....	99
Tabla 4-1 Operadores relacionales.....	111
Tabla 4-2 Operadores lógicos.....	119
Tabla 4-3 Tabla de verdad conjunción.....	122
Tabla 4-4 Tabla de verdad conjunción.....	123
Tabla 4-5 Tabla de verdad negación	124
Tabla 5-1 Prueba de escritorio.....	Error! Bookmark not defined.
Tabla 5-2 Prueba ciclo anidado	Error! Bookmark not defined.

1 Capítulo - Conceptos Básicos

1.1 Contenido conceptos básicos

1.1.1 Temas del capítulo

- **Concepto de algoritmo**
- **Concepto de programa**
- **Concepto de lenguaje de programación**
- **Lenguajes de programación**
- **Traductores de lenguaje**
- **Concepto de pseudocódigo**
- **Fases para desarrollar un programa**
- **Tipos de algoritmos**
- **Concepto de LPP**
- **Prueba de escritorio**
- **Ejercicios de la unidad**

1.2 Capítulo 1. Conceptos básicos

Este capítulo presenta los fundamentos básicos de la programación y el diseño de algoritmos. Se inicia con una definición de algoritmo, destacando su relevancia como base para resolver ejercicios computacionales. Luego, se introduce el concepto de programa, diferenciándolo del algoritmo y explicando su implementación en un lenguaje de programación.

Se hace énfasis en el uso del pseudocódigo como herramienta para representar algoritmos de manera clara antes de su conversión a un lenguaje formal.

Asimismo, se detallan las etapas para desarrollar un programa, abarcando desde la identificación del problema hasta su implementación y pruebas. Se analizan los distintos tipos de algoritmos y el uso de diagramas de flujo para visualizar su funcionamiento.

Finalmente, se aborda la técnica de prueba de escritorio, que se utiliza para comprobar la correcta ejecución de un algoritmo antes de su codificación. Al concluir el capítulo, se incluirán ejercicios que ayudarán a consolidar estos conceptos fundamentales.

1.3 Definición de algoritmo

ALGORITMO

En informática un algoritmo es un conjunto de pasos ordenados o secuencia de instrucciones u operaciones específicas con el fin de cumplir una tarea o que permiten controlar determinados procesos, hasta llegar a su final, según [15].

Para clarificar el concepto de algoritmo, es útil presentar un ejemplo que ilustre este término de manera más concreta. A continuación, se explicará un ejemplo de la vida real que facilitará una mejor comprensión de su significado.

1.3.1 Problema

Mario y Lucia se dirigen a la ciudad a visitar a su abuela en el automóvil de sus padres, viajando a una velocidad de 80 km/h, en el transcurso del viaje se dan cuenta, que una llanta está desinflada. A partir de ese momento describa la secuencia de pasos que debe seguir el conductor para encontrar la llanta averiada.

El ejercicio consiste en enumerar los pasos necesarios para realizar la búsqueda de la llanta pinchada.

1.3.2 Solución:

A continuación, se describen los pasos que debe seguir el conductor para detener el automóvil y encontrar la llanta pinchada.

Inicio

- PASO 1. Disminuir velocidad.
- PASO 2. Colocar luces de estacionamiento.
- PASO 3. Mirar retrovisor.
- PASO 4. Buscar lugar para estacionarse.
- PASO 5. Estacionarse.
- PASO 6. Colocar en neutro el carro.
- PASO 7. Colocar en freno de mano.
- PASO 8. Apagar el carro.
- PASO 9. Quitar cinturón de seguridad.
- PASO 10. Quitar las llaves.
- PASO 11. Abrir la puerta.
- PASO 12. Salir del carro.
- PASO 13. Caminar y buscar llanta pinchada.

1.3.3 ¿Preguntas?

Piensa y contesta.

- ¿Podrá el conductor realizar el paso 12 "salirse del carro", sin antes al realizar el paso número 9 sin quitar el cinturón de seguridad?
- "¿El conductor del puede salir del automotor a una velocidad de 80 km por hora?"
- ¿Podrá el conductor cambiar el paso 7 "Colocar el freno de mano" al 1 sin disminuir la velocidad?
- ¿El conductor puede hacer el paso 12 salir del carro sin antes abrir la puerta?

Para un programa de computador los pasos a seguir deben ser ordenados, con un estricto orden, además estas instrucciones deben estar cronológicamente establecidas, en caso contrario la computadora interpretará una acción inadecuada.

1.3.4 Características del algoritmo

- Un algoritmo es una técnica para resolver un problema y sus características fundamentales son:
- Un algoritmo debe ser preciso indicando la secuencia en la cual se debe realizar cada paso.
- Un algoritmo debe estar definido. Si se sigue un algoritmo dos veces, se debe obtener el mismo resultado cada vez.
- Un algoritmo debe ser finito. Si se sigue un algoritmo, se debe terminar en algún momento; o sea, debe tener un número finito de pasos.

Conclusión:

Un algoritmo es un conjunto de pasos ordenados, para cumplir una tarea o función específica, hasta llegar a su final.

"Un procedimiento computacional bien definido que toma un valor o un conjunto de valores como entrada y produce un valor o un conjunto de valores como salida. Es una secuencia de pasos lógicos que transforman la entrada en la salida de manera eficiente." [17].

1.4 Programa



Figura 1-1 Programa
Fuente Propia

Programa:

Un programa es una serie de instrucciones lógicas y secuenciales que el computador ejecuta para realizar una función específica o resolver un problema determinado [18].

Ejemplo:

Al realizar un programa como: Calcular el promedio de un conjunto de datos, el descuento sobre un producto de la canasta familiar o determinar la sumatoria de una serie de números. Se debe tener en cuenta que un computador trabaja con lenguaje binario, es necesario que estos programas o secuencia de instrucciones sean escritos en lenguajes más comprensibles para un desarrollador, es decir un lenguaje de programación. A partir de lo anterior es de resaltar que un programa es entonces la transcripción de un algoritmo en un lenguaje de programación, ver Figura 1.1.

1.5 Lenguaje de programación

Concepto

Es un lenguaje que ha sido diseñado con el propósito de describir un conjunto de acciones o instrucciones (algoritmo) que un equipo de cómputo debe realizar.

También se puede definir como: "Un lenguaje artificial diseñado para expresar computaciones que pueden ser realizadas por una máquina, particularmente una computadora. Los lenguajes de programación pueden usarse para crear programas que controlen el comportamiento de una máquina, expresen algoritmos con precisión, o sirvan como medio de comunicación humana"[1].

Los lenguajes de programación usan un juego de palabras que son comunes al desarrollador, con el fin de que el compilador pueda traducirlos a lenguaje de máquina. Este lenguaje es usado por el procesador de un computador y consiste en el uso de datos binarios (0 y 1). El lenguaje de máquina no es comprensible para los seres humanos, razón por la cual los lenguajes de programación son lenguajes intermedios que evitan la tediosa tarea de escribir las instrucciones en lenguaje de máquina.

Dentro de este marco hay varios lenguajes de programación como Kotlin, Phyton, Java, C# etc. Cada uno con sus particularidades y aplicaciones específicas. A continuación, se presenta una clasificación general con ejemplos:

1.5.1 Lenguaje Máquina

Es el más primitivo; utiliza numeración binaria (ceros y unos) es el único lenguaje que una computadora puede entender directamente, donde cada comando corresponde directamente a una operación que el hardware puede ejecutar.

"El conjunto de instrucciones que el procesador de una computadora entiende y ejecuta directamente, representado en código binario"[2].

1.5.2 Lenguajes de Bajo Nivel

Están diseñados para un hardware específico, se relacionan con la arquitectura del hardware de la computadora.

Ejemplo

- Assembly.

1.5.3 Lenguajes de Alto Nivel

Los lenguajes de alto nivel son diseñados para ser comprensibles y fáciles de usar para los programadores, utilizando una sintaxis muy fácil de usar, cercana al lenguaje humano y son independientes del hardware.

Ejemplos

- C

- C++
- Java
- Python
- JavaScript47.

1.5.4 Lenguajes de propósito específico

Son aquellos diseñados para resolver problemas dentro de un dominio o área particular.

Ejemplo:

SQL Lenguaje de estructurado de consultas para bases de datos.

R para análisis estadístico, para analizar datos y crear gráficos.

1.5.5 Importancia de los lenguajes de programación

Comunicación: Permiten a las personas dar instrucciones precisas a las maquinas computadoras.

Expresión de algoritmos: Facilitan la implementación de soluciones a problemas complejos por medio un lenguaje de programación.

Aplicaciones: Son la base para realizar desarrollos de software de diferentes tipos.

1.6 Traductores de Lenguaje

Concepto

"Un programa que convierte código escrito en un lenguaje de programación a otro lenguaje o formato, permitiendo su ejecución o interpretación por parte de una máquina"[1]

Los traductores de lenguaje son programas que transforman el código escrito en un lenguaje de programación, como Java o C, en un formato que pueda ser ejecutado directamente por la computadora. Este proceso implica convertir el código fuente a lenguaje de máquina, que es el único formato que puede ser interpretado y procesado por el hardware.

Por ejemplo, cuando se escribe un programa en Java el compilador de java lo traduce a código binario, permitiendo su ejecución eficiente por parte del procesador. De esta manera, los ordenadores pueden entender y realizar las instrucciones contenidas en el programa.

Estos traductores del lenguaje incluyen compiladores, intérpretes **y ensambladores**, y son fundamentales en el proceso de desarrollo de software, a continuación se presenta una breve descripción.

1.6.1 Compilador

Convierte el código fuente en código máquina o en un lenguaje intermedio.

Ejemplo

C:/java/javac **Calculadora.java** resulta el bytecode **Calculadora.class**

El archivo **Calculadora.java** es el código fuente escrito en el lenguaje de programación Java. Cuando se compila utilizando el comando javac.exe, se genera un archivo llamado **Calculadora.class**. Este archivo contiene bytecode, que es una forma intermedia entre el código fuente y la máquina. El bytecode no es directamente ejecutable por la computadora. En su lugar, es interpretado por la Máquina Virtual de Java (JVM) al momento de su ejecución, lo que permite que los programas escritos en Java sean independientes del sistema operativo y hardware específico.

Desventajas

Al encontrar un error, se debe corregir antes de ejecutar el programa, lo que puede hacer el proceso de depuración más complejo.

1.6.2 Intérprete

Ejecuta el código fuente directamente, línea por línea, sin generar un archivo ejecutable.

Ventaja

- Facilita la detección de errores, ya que detiene la ejecución en caso de error.
- Permite una ejecución más inmediata porque no necesita una compilación previa.

Desventajas:

- Es más lento en la ejecución, ya que traduce el código en tiempo real.

Ejemplo

Intérpretes de Python o JavaScript.

1.6.3 Ensamblador

Convierte código ensamblador (assembly) en código máquina.

Ejemplo

Ensambladores para arquitecturas x86 o ARM.

1.7 Fases para desarrollar un programa.

1.7.1 Fase de análisis

Decidir y planear qué debe hacer el programa y qué problemas resolver.

Actividades

- Realizar el análisis de requisitos que consiste en identificar, documentar y validar las necesidades y expectativas de los usuarios o clientes para el desarrollo del programa.
- Reunirse con los stakeholders que son los usuarios o clientes, para conocer las necesidades que actualmente tienen.
- Definir los requisitos funcionales o sea lo que debe hacer el programa y los requisitos no funcionales como por ejemplo rendimiento, la seguridad, etc.
- **Entrega:** documento de los requisitos en un documento de especificación.

1.7.2 Fase de diseño

(Es el desarrollo de la solución), Define y planificar cómo se construirá el algoritmo.

Actividades:

- Realiza los diagramas de arquitectura del sistema diagramas de bloques o UML.
- Realizar el diseño de la estructura de datos el diagrama de entidad relación. y los algoritmos que se utilizarán.
- Definir la interfaz de usuario colores, logos.
- Elegir las tecnologías y herramientas de desarrollo, ejemplo Java, Phyton, C#, si es para móviles o web.
- **Entrega:** documento de diseño del sistema.

1.7.3 Fase de codificación

Es la implementación del algoritmo en el lenguaje de programación más adecuado.

Actividades:

- Codificar en el lenguaje de programación elegido (Java. C#, Kotlin, C++, etc).

- Seguir buenas prácticas de codificación, código limpio, uso de comentarios, nombres descriptivos de variables, etc.
- Integrar bibliotecas o frameworks necesarios especificando versiones.
- **Entrega:** Se debe entregar el código fuente del programa.

1.7.4 Fase de pruebas

No basta que el programa esté terminado, Hay que comprobar que el programa no falle y funcione perfectamente en todos los casos posibles que se puedan presentar.

Actividades:

- Realizar pruebas unitarias significa probar funciones individuales.
- Realizar pruebas de integración ósea probar cómo interactúan los módulos.
- Realizar pruebas de sistema significa probar el programa completo.
- Realizar pruebas de aceptación se realiza con cliente o usuario final.
- **Entrega:** Informe de pruebas y corrección de errores, lo que es los bugs defectos del código implementado.

1.7.5 Despliegue

Es la Implementación, es poner el programa en funcionamiento en el entorno real.

Actividades:

- Instalar el programa en los servidores o dispositivos necesarios, con la tecnología adecuada.
- Configurar el entorno de producción, con librerías y fuera de fallos.
- Realizar pruebas finales de la aplicación en el entorno real.
- **Entrega:** Dejar la aplicación o el programa en funcionamiento para los usuarios finales.

1.7.6 Fase de documentación y mantenimiento:

Se elabora la documentación del programa, y se realizan las actualizaciones oportunas que se vayan necesitando y se adapte a nuevas necesidades.

Actividades:

- Corregir errores que surjan después de la instalación.
- Realizar actualizaciones al programa para mejorar su funcionalidad o rendimiento y adaptar a nuevos requisitos o cambios en el entorno.
- **Entrega:** Versiones actualizadas y mejoradas del programa en funcionamiento.

Las etapas para crear un programa facilitan el diseño, la implementación y el mantenimiento del software de manera efectiva y ordenada. Estas etapas son esenciales en el proceso de desarrollo de software, ya que aseguran que el programa satisfaga los requisitos esperados, Ver figura 1.2.



Figura 1-2 Fases para desarrollar un programa
Fuente propia

1.8 Tipos de algoritmos

En el diseño de algoritmos y la resolución de problemas informáticos, es fundamental utilizar métodos y técnicas que permitan estructurar el algoritmo de manera eficiente. Existen diversas estrategias que facilitan su desarrollo y comprensión, entre las cuales se destacan: ver Figura 1.3.



Figura 1-3 Tipos de algoritmos
Fuente Propia

1.9 Diagramas de Flujo

"Pseudocódigo es una descripción informal de alto nivel de un algoritmo que combina elementos de lenguaje natural con estructuras de programación para representar la lógica de un programa de manera clara y concisa" [1].

1.9.1 Tablas de decisión

Concepto

"Una representación tabular de las combinaciones de condiciones y las acciones correspondientes que deben tomarse en un sistema, facilitando la identificación de reglas de negocio y la detección de inconsistencias" [2].

1.9.2 Diagramas de flujo u ordinogramas

Diagrama de flujo:

El diagrama de flujo es la representación gráfica para la resolución de un problema. A través de un diagrama de flujo es posible visualizar de manera más clara la forma en la que se soluciona un problema, por lo cual éste cuenta con un inicio, fin, entradas, salidas, decisiones, bucles, etc. De manera específica un diagrama de flujo corresponde a una representación de un algoritmo.

El diagrama de flujo presenta inconvenientes:

- Si hay error en el algoritmo hay que reestructurar el diagrama de nuevo.
- Presenta técnica lineal ya no utilizada.
- Al seguir el diagrama hasta el final puede resultar complejo.

1.9.3 Entornos para diseños de diagramas de flujo

Existen programas para realizar diagramas de flujo y ofrecen una variedad de características, desde opciones gratuitas hasta soluciones más avanzadas. En la tabla 1.1 presenta una lista de programas para crear Diagramas de Flujo y Pseudocódigo.

[1]

Aplicación

PSelInt	Características
Entorno diseñado específicamente para representar algoritmos en forma de pseudocódigo y diagramas de flujo.	Permite la conversión entre pseudocódigo y diagramas de flujo.
FreeDFD	Características
Herramienta gratuita para el diseño de diagramas de flujo.	Interfaz sencilla y fácil de usar, permite crear diagramas de flujo básicos.
Dia	Características
Programa gratuito que permite crear diagramas de flujo y otros tipos de diagramas	Interfaz gráfica amigable, Soporte para múltiples tipos de diagramas.
Lucidchart	Características
Herramienta en línea para crear diagramas de flujo con una amplia gama de plantillas.	Colaboración en tiempo real, integración con otras herramientas (Google Docs, Microsoft Office).
SmartDraw	Características
Software versátil que permite crear más de 70 tipos de diagramas, incluyendo diagramas de flujo.	Plantillas modernas e inteligentes. Posibilidad de crear diagramas a partir de datos. Soporte para dibujo CAD.
Draw.io (ahora diagrams.net)	Características
Herramienta gratuita basada en la web para crear diagramas, incluyendo diagramas de flujo.	Interfaz intuitiva con funciones de arrastrar y soltar. Integración con Google Drive y otras plataformas.
ConceptDraw	Características
Software profesional que ofrece herramientas avanzadas para crear diagramas complejos.	Amplia biblioteca de símbolos y plantillas. Exportación a múltiples formatos

Tabla 1.1 Aplicaciones para diagramas de flujo

Fuente propia

FLOWGORITHM: "Es un lenguaje de programación gratuito para principiantes que se basa en diagramas de flujo gráficos" [2], continuación se presentan los diagramas en Flowgorithm.

1.9.4 Representación de los diagramas de flujo

La representación de los diagramas de flujo en la aplicación Flowgorithm, son una excelente manera de visualizar y planificar algoritmos antes de implementarlos en código. Para realizar un problema dependiendo de la dificultad, puedes elegir entre diferentes tipos de diagramas para representar de manera clara y eficiente la lógica del programa, a continuación, en la tabla 12, se presenta la lista de los diferentes recursos que pueden ser usados con esta herramienta.

FUNCION	SÍMBOLO	DESCRIPCION
Inicio	Principal	Representación gráfica del inicio: Es una instrucción para iniciar el algoritmo.
Fin	Fin	Representación gráfica del fin: Es una instrucción para finalizar el algoritmo.
Declaración	Declaración	Representación gráfica de la declaración de variables: Es una instrucción para declarar variables con el tipo de dato adecuado en el algoritmo.
Asignación	Asignación	Representación gráfica Preparar. Implica un proceso para asignar valores de variables.
Decisión		Representación gráfica de la elección. Representa una pregunta e indica el destino del flujo de información con base en respuestas alternativas de sí y no.
Salida	Salida	Representación gráfica y Salida Operación que implica salida de información por pantalla o impresora.

		Representación gráfica Entrada
Entrada		Operación que implica entrada de información.
Ciclo mientras		Representación gráfica Ciclo Mientras: Implica realizar acciones repetitivas.
Ciclo para		Representación gráfica Ciclo Para: Implica realizar acciones repetitivas.
Ciclo Haga		Representación gráfica Ciclo Haga: Implica realizar acciones repetitivas.
Llamada a subprograma		Representación gráfica para el Llamada: Implica llamada al subprograma
Comentario		Representación gráfica comentario: Permite realizar alguna nota o comentario.
Flechas		Representación gráfica Flechas: Representan flujo de información. Indican dirección que sigue el flujo en el sistema.

Tabla 1.2 Diagramas de flujo

Fuente propia

A continuación, se presentarán los diagramas de flujo de más representativos en la aplicación Flowgorithm, con lo cual se ilustra la manera de ver un algoritmo gráficamente.

1.9.5 Diagramas de flujo básico de inicio y fin

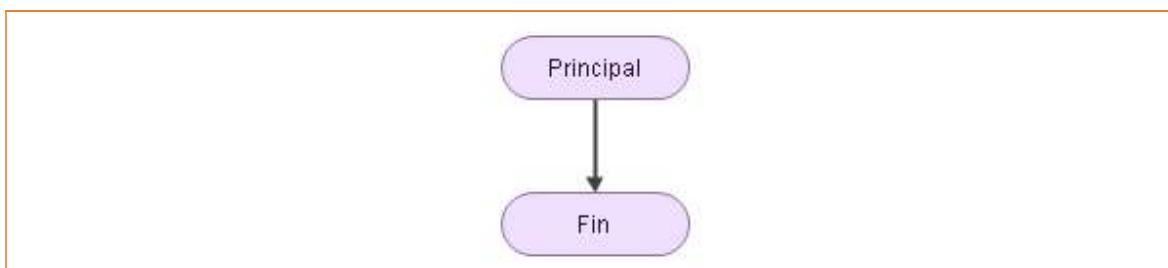


Figura 1-4 Inicio – fin
Fuente propia

Es una instrucción gráfica básica indicada en la figura 1.4 y utilizada para indicar el inicio y el fin de un algoritmo por medio del diagrama de flujo.

1.9.6 Diagrama con comentario

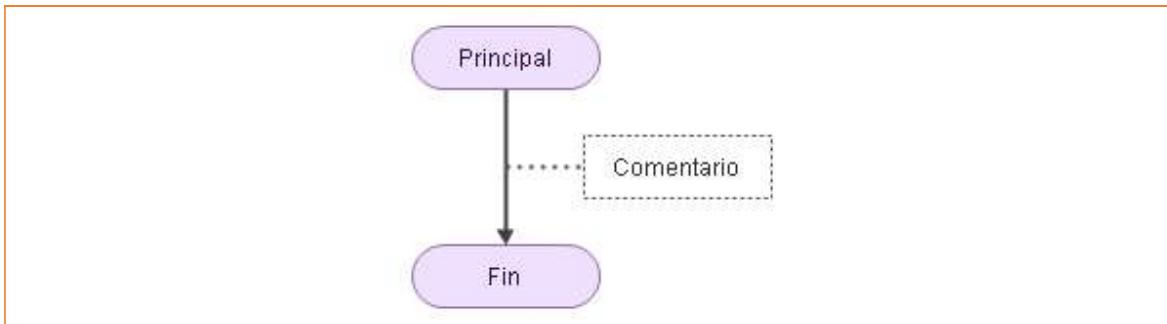


Figura 1-5 Comentario

Fuente propia

Es la representación gráfica del comentario que permite realizar alguna nota o comentario textual pero no tiene ningún efecto en el algoritmo, figura 1.5.

1.9.7 Diagramas declaración de variables

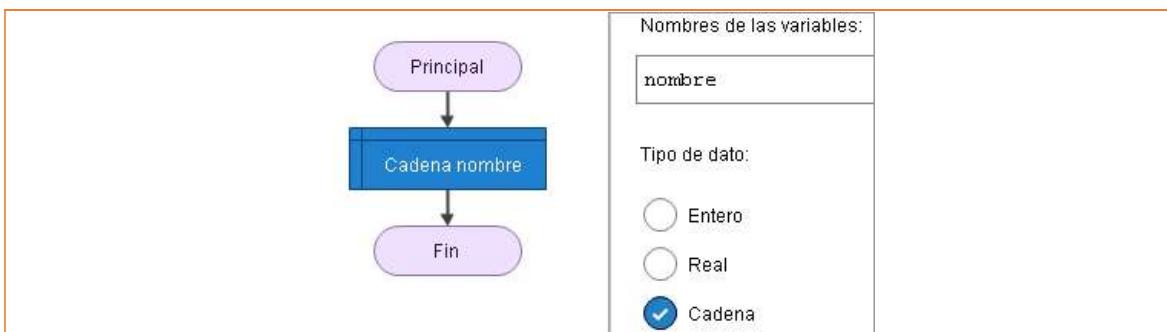


Figura 1-6 Declaración de variables

Fuente propia

Es la ilustración gráfica para la declaración de la variable. La figura 1.6 es una instrucción para declarar la variable **nombre** que hace uso de un tipo de dato cadena que posteriormente almacenará un valor del nombre de la persona.

1.9.8 Diagramas entrada y salida

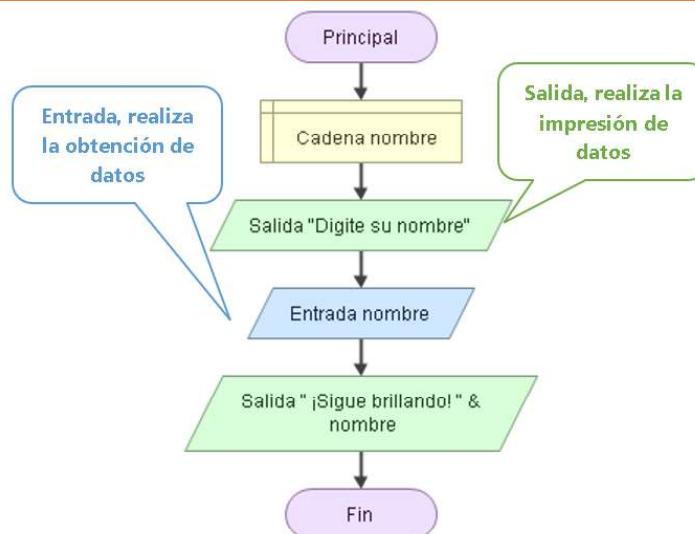


Figura 1-7 Entrada y salida

Fuente propia

En la figura 1.7 se ilustra la entrada y salida de datos, por medio del diagrama de flujo de entradas y salidas, la **figura salida** implica la presentación de la información por consola o pantalla además de la impresora, la **figura entrada** implica la captura de datos al sistema.

1.9.9 Diagramas condicionales por asignación

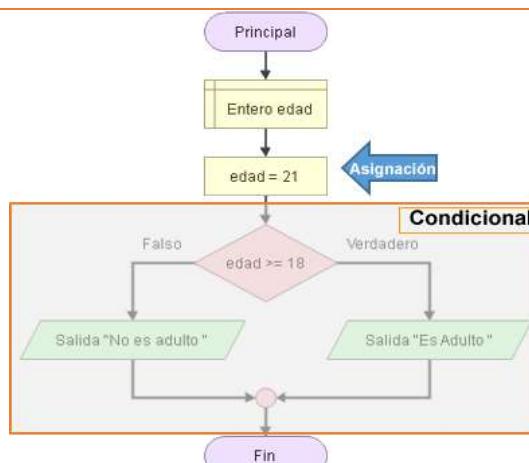


Figura 1-8 Condicionales por petición

Fuente propia

En la figura 1.8 condicionales por petición es la representación gráfica de la elección donde una pregunta o condición e indica el destino del flujo de información con base en respuestas alternativas de sí y no.

1.9.10 Diagramas condicionales con petición

En la figura 1.9 presenta la entrada y salida de datos, por medio del diagrama de flujo, en la cual se presenta al usuario un mensaje "Digite su edad" a través de la figura salida, luego la figura entrada pide que se digite la edad del usuario y la cual se almacena en la variable **edad**, para posteriormente ser evaluado en el diagrama de condicionales para saber si es mayor o menor de edad.

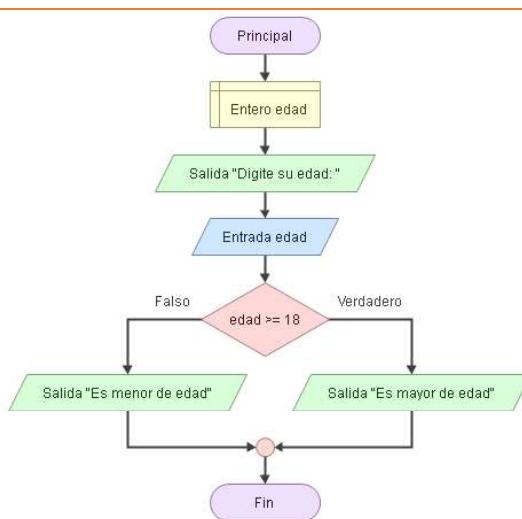


Figura 1-9 Condicionales con petición
Fuente propia

1.9.11 Diagrama de flujo ciclo mientras con salida de datos

En la figura siguiente presenta un diagrama de flujo del ciclo mientras con salida de datos, este diagrama encerrado en el recuadro azul es útil para representar procesos donde se requieren repeticiones hasta un numero determinado y controladas por una condición, ademas se debe mostrar un resultado calculado durante las iteraciones.

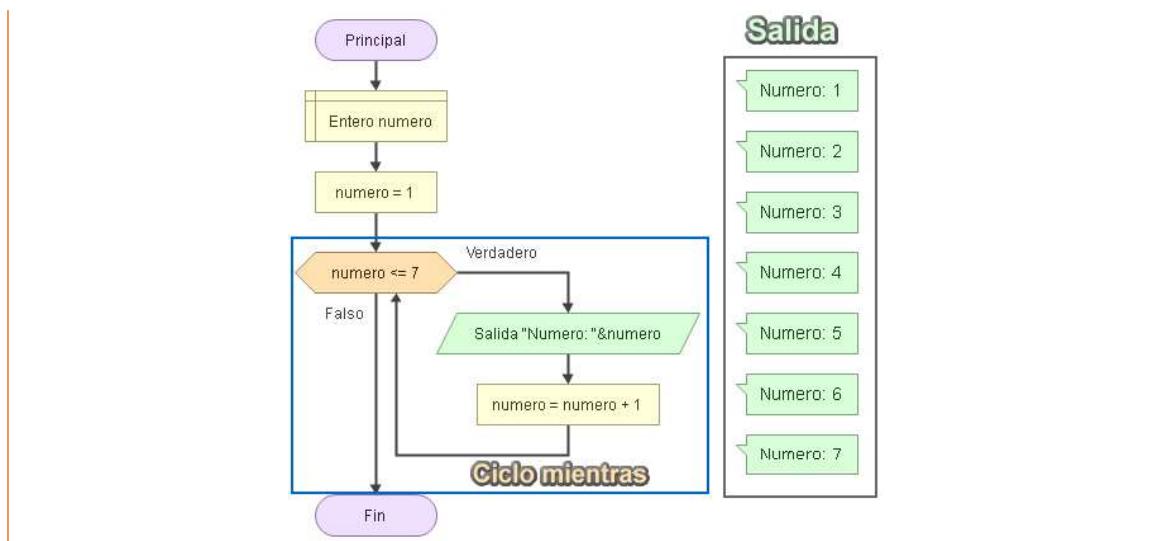


Figura 1-11 Condicionales por petición
Fuente propia

1.9.12 Diagrama de flujo ciclo para

En la figura 1.12 se ilustra el ciclo para, por medio de la **representación gráfica ciclo** implica un inicio, condición, cuerpo del ciclo, incremento de variable iteradora o disminución, salida de datos y fin del ciclo y se usa para realizar acciones repetitivas.

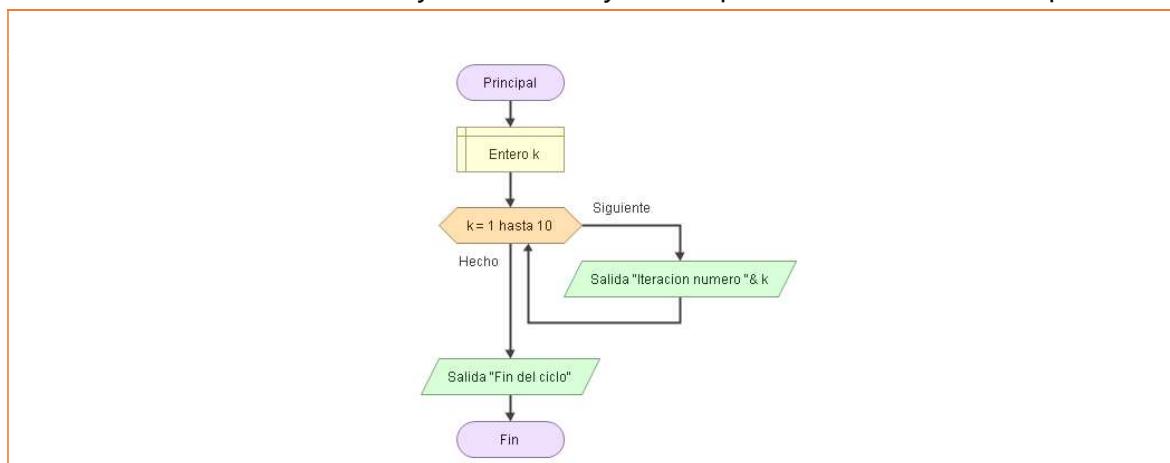


Figura 1-12 Condicionales por petición
Fuente propia

1.9.13 Diagramas con funciones

Este diagrama con funciones de la Figura 1.10 Ilustra de manera sencilla cómo se estructura y se ejecuta una función dentro de un algoritmo. El diagrama de la izquierda es el flujo principal del algoritmo que llama a la función saluda de la derecha para la ejecución de la tarea y para que imprima el mensaje "Hola suertudos".

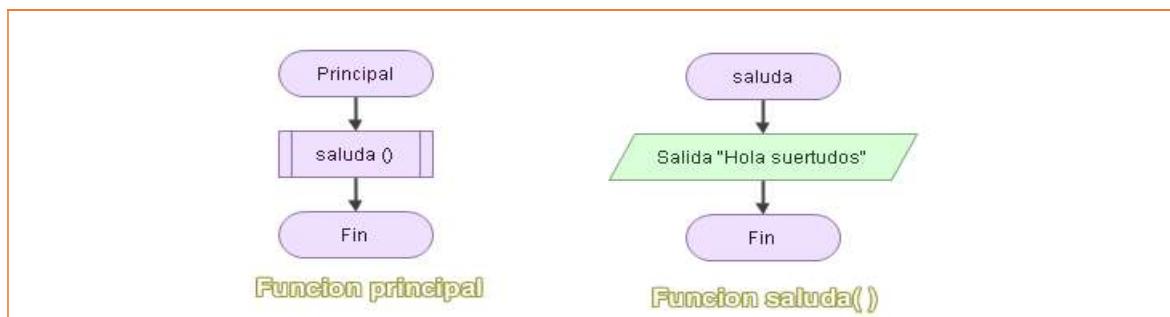


Figura 1-10 Diagramas con funciones

Fuente propia

1.10 Pseudocódigo

Concepto

El pseudocódigo: Es una mezcla entre un lenguaje de programación y el idioma español (o cualquier otro idioma), el cual se emplea dentro de la programación estructurada, para realizar el diseño de un programa. El pseudocódigo es la representación narrativa de los pasos que debe seguir un algoritmo para dar solución a un problema determinado.

Concepto

Describe un algoritmo utilizando una mezcla de frases en lenguaje común, instrucciones de lenguaje de programación y palabras clave que definen las estructuras básicas.

El pseudocódigo o falso lenguaje es un lenguaje informal cuyo propósito es ayudar a los programadores a desarrollar algoritmos sin tener que preocuparse por detalles estrictos de la sintaxis del lenguaje de programación. Este lenguaje describe un

algoritmo utilizando una mezcla de frases en lenguaje común el español, remplazando instrucciones típicas de lenguajes de programación y palabras clave que definen la estructura básica de un programa. Resultando fácil y sencillo codificar algoritmos en pseudocódigo que luego se convertirán en programas. Aunque no es un lenguaje de programación, el pseudocódigo incluye entradas, salidas o cálculos.

A modo de ejemplo consideremos el siguiente bloque de código escrito usando pseudocódigo cuyo propósito es hallar el área de un cuadrado:

Ejemplo

Entero lado, area_cuadrado

Inicio

```
Escribir "Ingrese el lado del cuadrado"  
Lea lado  
area_cuadrado <- lado*lado  
Escribir "El área del cuadrado es", area_cuadrado
```

Fin

1.11 Definición de LPP

Concepto

LPP es un lenguaje de programación para pseudocódigo creado a partir del proyecto de grado del Ingeniero Iván Deras, con el propósito de facilitar el proceso de enseñanza de la lógica de programación en principiantes, de allí su acrónimo Lenguaje de Programación para Principiantes. LPP es un lenguaje de programación muy simple y funcional que sirve para el estudio de algoritmos, en especial para verificar y visualizar los programas escritos en pseudocódigo. Lo anterior teniendo en cuenta que este lenguaje usa en su sintaxis instrucciones en español, que son comunes a la mayoría de los lenguajes de programación.

1.12 Prueba de Escritorio

Concepto

Una prueba de escritorio es una simulación o comprobación lógica de un algoritmo, cuyo propósito es evaluar la validez de este. Para realizar una prueba de escritorio generalmente se crea una tabla, con tantas columnas como variables existan y tantas filas como instrucciones haya en el algoritmo. Sobre la cual se toman los valores que puede ingresar el usuario en la ejecución real del programa. De esta manera, las pruebas de escritorio permiten detectar errores y omisiones en el algoritmo.

Si por ejemplo se considera el ejercicio de la tabla 1.3, y se asigna el valor de 7 a la variable lado, la prueba de escritorio resultante sería:

Línea	Instrucción	Variables	
		lado	area_cuadrado
1	Inicio		
2	Escriba "Ingrese el lado del cuadrado"		
3	Lea lado	7	
4	area_cuadrado = lado*lado		49
5	Fin		

Tabla 1.3 Prueba de escritorio

Fuente propia

Esta prueba de escritorio de la tabla 0.3 evalúa el funcionamiento del algoritmo paso a paso, utilizando valores concretos para verificar que los cálculos sean correctos.

Explicación de la prueba de escritorio

Línea 1

Inicio del algoritmo.

Línea 2

Se solicita al usuario ingresar el valor del lado del cuadrado.

Se muestra en pantalla: "Ingrese el lado del cuadrado".

Línea 3

El usuario ingresa el valor 7 y se almacena en la variable lado.

Línea 4

Se calcula el área del cuadrado aplicando la fórmula:

area_cuadrado = lado * lado

area_cuadrado = 7 * 7 = 49

Línea 5

El algoritmo finaliza con la variable area_cuadrado almacenando el valor de 49.

1.13 Ejercicios de la unidad

Al realizar esta serie de actividades prácticas, es más difícil para el estudiante entender la necesidad de precisión, claridad y eficiencia en los algoritmos. El desarrollo de estas también fomenta, el pensamiento lógico y ayudan a comprender el orden prioritario de cada instrucción.

Realiza los siguientes ejercicios.

- Describa paso a paso cómo llegar de su casa a la universidad, Analice la precisión: ¿Qué exprese que pasa si omite un giro o un paso intermedio?
- Escribe un algoritmo para contar desde 10 hasta 1, piense en cómo estructurarlo para que pueda repetirse fácilmente si necesita contar de nuevo.
- Diseña un algoritmo para repartir equitativamente 10 manzanas entre 3 personas, reflexione sobre la claridad y precisión necesarias para que el algoritmo funcione correctamente.

2 CAPITULO. INSTALACIÓN DE LPP

2.1 Contenido Instalación de LPP

2.1.1 Temas del capítulo

- **Pasos para la instalación de LPP**
- **Generación de un programa inicial LPP**
- **Ejecución de un programa inicial en LPP**
- **Opciones adicionales en LPP**
- **Ejercicios de la unidad**

2.2 Capítulo 2. Instalación en LPP

Una vez abordados los conceptos fundamentales de programación en la sección anterior, es momento de pensar en el uso de la herramienta y que esta permita la escritura y compilación de algoritmos en el lenguaje LPP. En este capítulo, se trabajará la configuración y puesta a punto de la aplicación LPP, detallando los pasos necesarios para su correcta instalación y preparación.

Además, se explicará de manera sencilla cómo crear, editar y ejecutar un programa básico en LPP, planteando ejemplos sencillos que faciliten la comprensión de su funcionamiento. Por último, se presentarán algunas funcionalidades avanzadas de la herramienta, como la generación de código fuente en otros lenguajes de programación y la creación de ejecutables directamente en LPP, ampliando así sus posibilidades de uso.

2.3 Pasos para la instalación de LPP

Paso 0

En primer lugar, procedemos a descargar el instalador de LPP (LPP Setup.exe), el cual se encuentra disponible en el siguiente enlace:
<http://mediatecnica.weebly.com/lpp.html>

Paso 1

Una vez descargado el instalador, ejecutamos el archivo Setup.exe descargado en el paso anterior, ver Figura 2.1.



Figura 2-1 Instalador
Fuente propia

Paso 2

A continuación, se presenta la interfaz gráfica de instalación, en la cual se puede elegir la unidad de destino del programa, ver Figura 2.2. Dejamos la opción por defecto, puesto que LPP ocupa muy poco espacio en disco.



Figura 2-2 Ventana de instalación

Fuente propia

Paso 3

En la interfaz de instalación, se escoge también el idioma en el que va a quedar instalado LPP, ver Figura 2.3. Dado que no existe la opción “Español”, dejamos por defecto el idioma Inglés y presionamos clic en el botón “Install”.



Figura 2-3 Selección de idioma

Fuente propia

Paso 4

Después de escoger el idioma, se muestra una interfaz con la información referente al progreso de instalación del programa en el disco seleccionado, ver Figura 2.4.

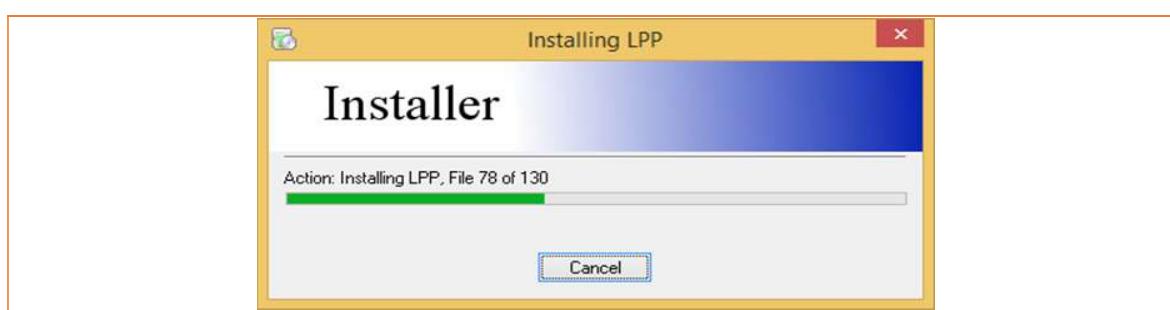


Figura 2-4 Progreso de instalación

Fuente Propia

Posteriormente presenta la pantalla Figura 2.5, con la notificación de que el programa ha sido instalado de forma satisfactoria.

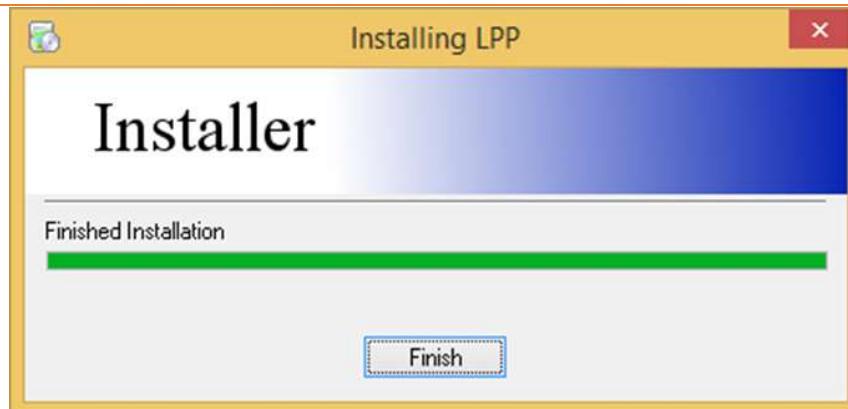


Figura 2-5 Instalación finalizada

Fuente Propia

Paso 5

Una vez finalizada la instalación del programa LPP, se procede a ejecutarlo. Para ello, basta con escribir 'LPP' en la barra de inicio del sistema operativo Windows Figura 2.6.

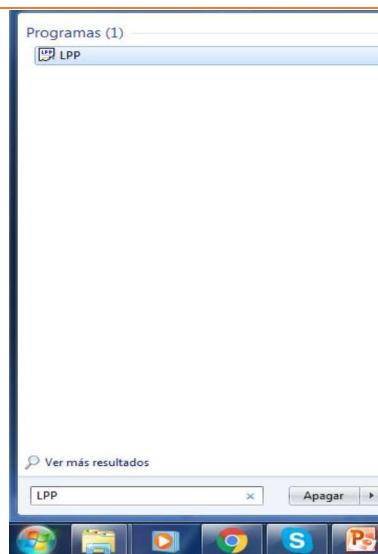


Figura 2-6 Lanzando LPP

Fuente Propia

Se despliega la aplicación de LPP 2.7, lenguaje de programación para principiantes

La aplicación es un lenguaje de programación para principiantes para la enseñanza, la herramienta es un entorno diseñado para facilitar el aprendizaje de la programación básica y algoritmos utilizando pseudocódigo. Entre sus principales funciones se destacan:

- Entorno intuitivo
- Interpretación de pseudocódigo
- Corrección de errores
- Soporte para estructuras de control básicas
- Manejo de variables y operaciones
- Visualización de la ejecución paso a paso
- Exportación de código

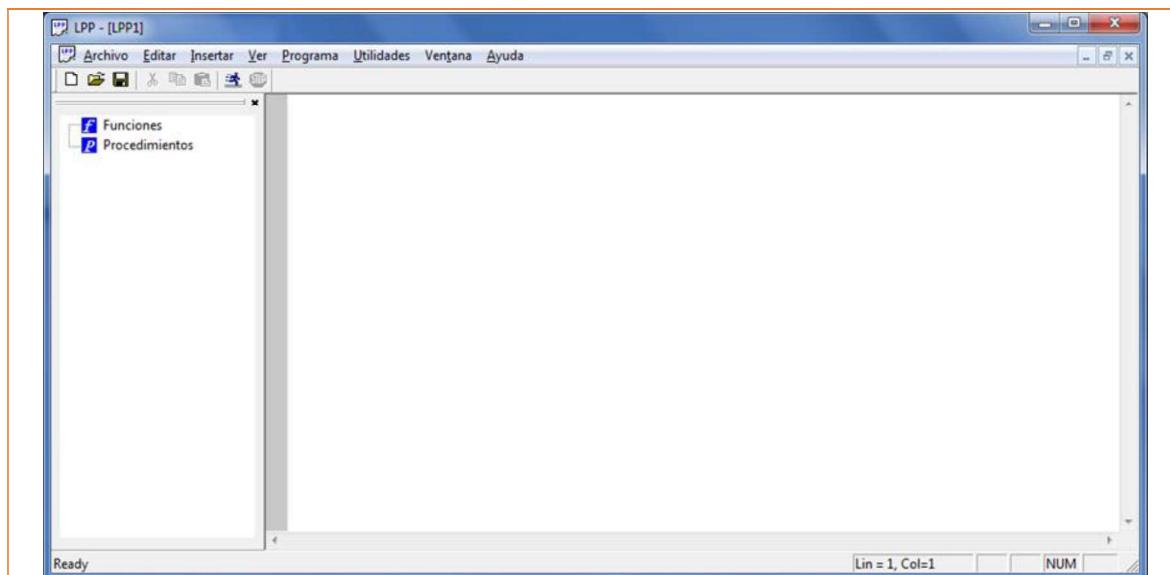


Figura 2-7 Entorno de trabajo LPP
Fuente Propia

2.4 Generación de un programa inicial LPP

Una vez abierto el entorno de trabajo del programa LPP (ver Figura 2.7), se pueden identificar 5 secciones bien definidas: el menú superior de opciones (Archivo, Editar, Insertar, Ver, Programa, Utilidades, Ventana y Ayuda), el menú superior de íconos de acceso rápido, el panel de funciones de la izquierda, la sección de codificación de la derecha y el panel inferior de conteo de líneas y columnas, ver Figura 2.8.

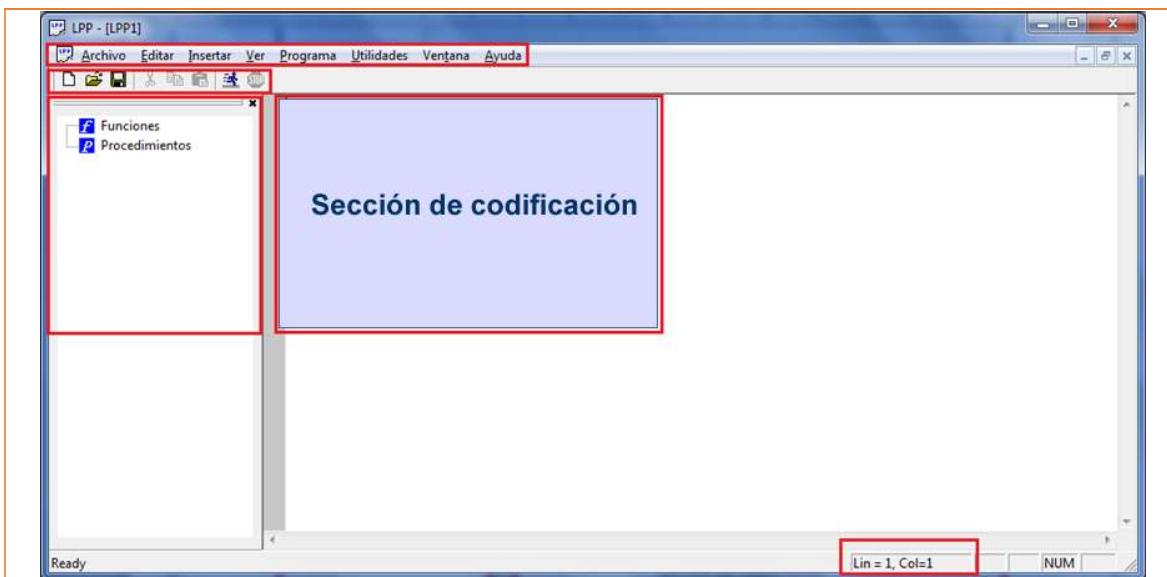


Figura 2-8 Editor de LPP
Fuente Propia

Para empezar a escribir un programa en LPP, se hace uso de la sección de codificación de la parte derecha, ver Figuras 2.8. El programa de la Figura 1.9 en la sección de codificación se declara una variable de tipo entero llamada x, luego se le asigna el valor de 3 y finalmente muestra el valor de la variable en pantalla.

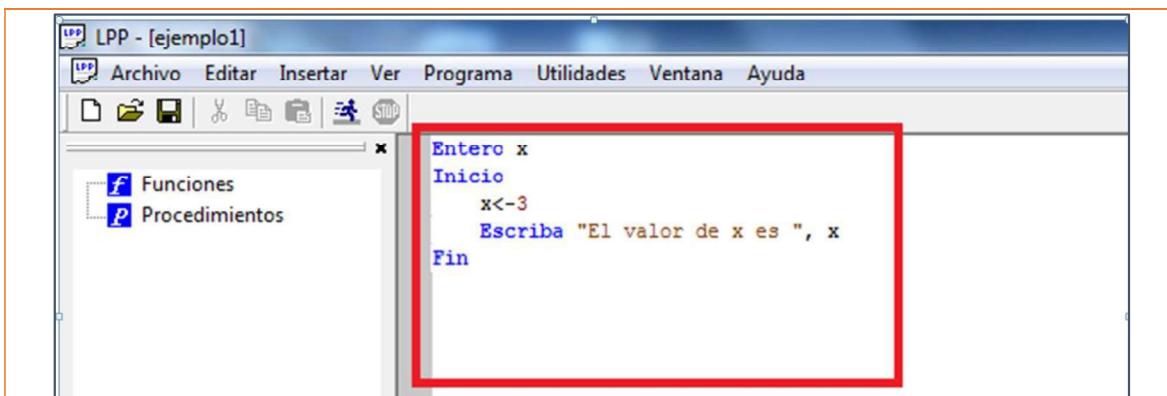


Figura 2-9 Ejemplo básico LPP
Fuente propia

Una vez escrito el programa, se procede a guardarlo, usando para ello el ícono con el diskette ver Figura 2.10 del menú de iconos de acceso rápido o usando la opción Salvar como del menú superior: Archivo ver Figura 2.11.

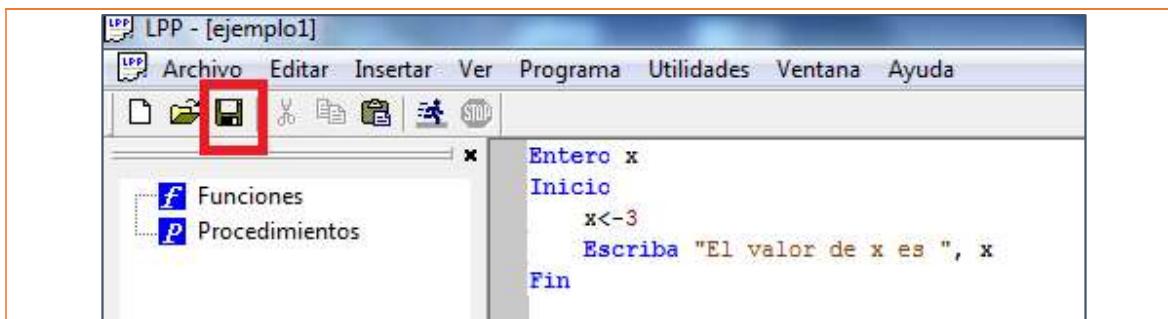


Figura 2-10 Opción guardar abreviada

Fuente propia

La otra opción para guardar el código, se procede usando el menú superior Archivo en la opción Salvar como ver Figura 2.11.

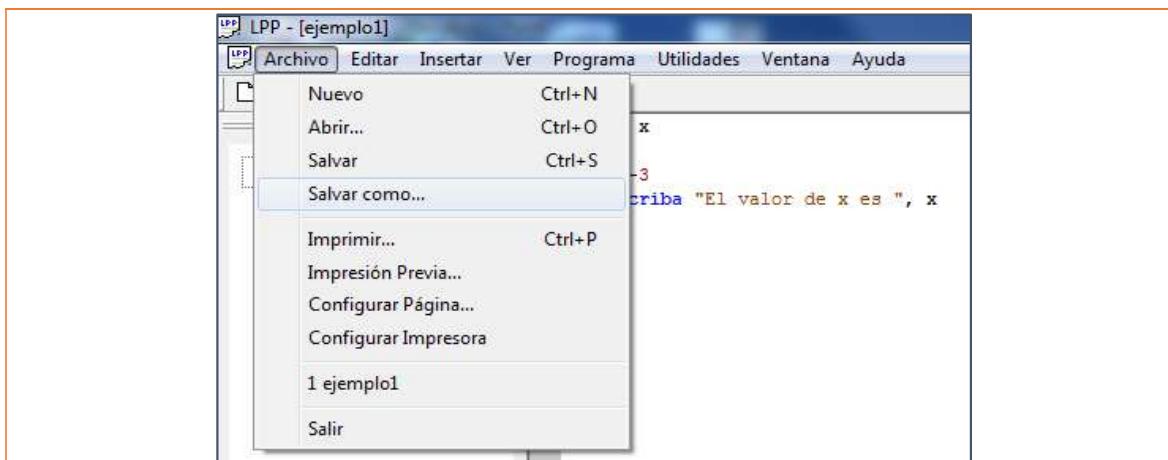


Figura 2-11 Opción guardar del menú

Fuente propia

De igual forma si se desea abrir un programa guardado en el disco duro del computador, se escoge el ícono abreviado de abrir, ver Figura 2.12.

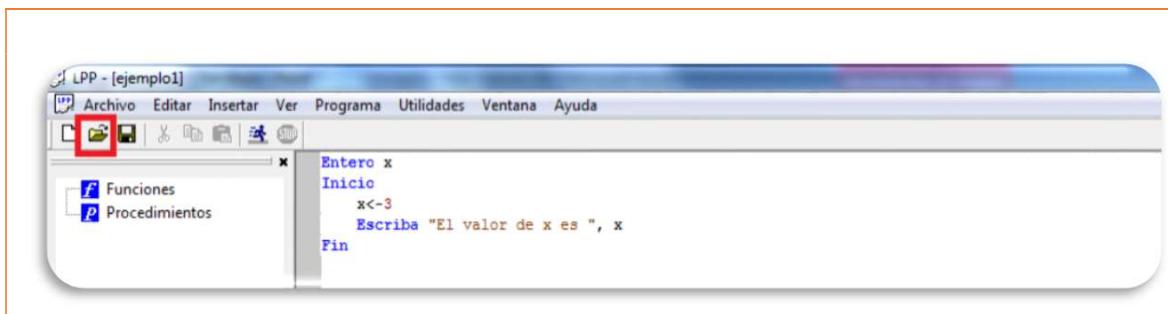


Figura 2-12 Opción abrir abreviada

Fuente propia

La otra opción de abrir un archivo de código en LPP, es usar el ícono de carpeta del menú superior Archivo, en el ítem Abrir, ver Figura 2.13. Una vez guardado o abierto el programa se procede a la compilación y ejecución de este.

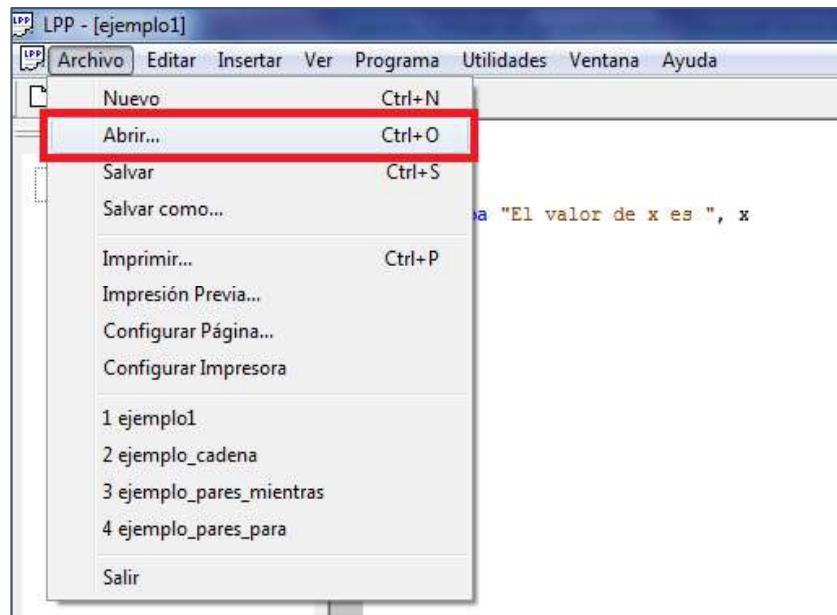


Figura 2-13 Opción abrir del menú archivo
Fuente propia

2.5 Ejecución de un programa inicial en LPP

Después de abrir el programa Figura 2.9, se procede a compilarlo y ejecutarlo. El proceso de compilación arrojará posibles errores en la sintaxis del lenguaje LPP y será requisito fundamental para el proceso de ejecución. Para compilar el programa, se escoge la opción “Compilar” del menú superior “Programa”, ver Figura 2.14 y se verifica que no existan errores en la sintaxis del programa escrito.

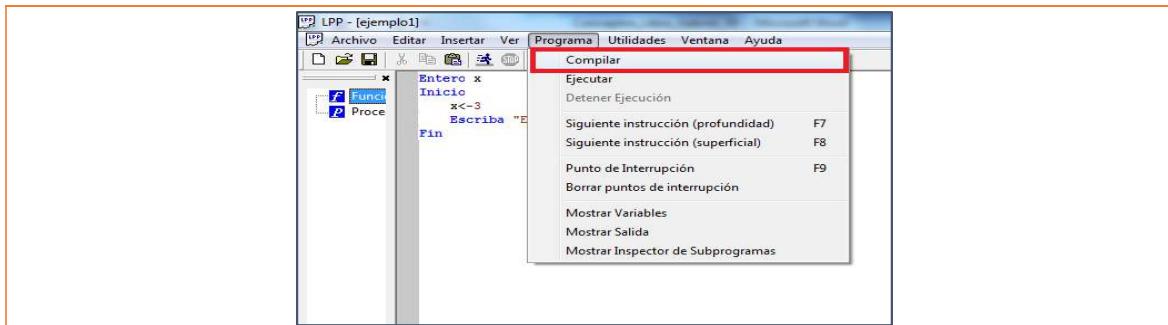


Figura 2-14 Opción compilar

Fuente propia

Posteriormente, se procede a ejecutar el programa, para lo cual existen 2 alternativas, la primera opción es presionar el ícono azul de la barra de herramientas ver Figura 2.15.

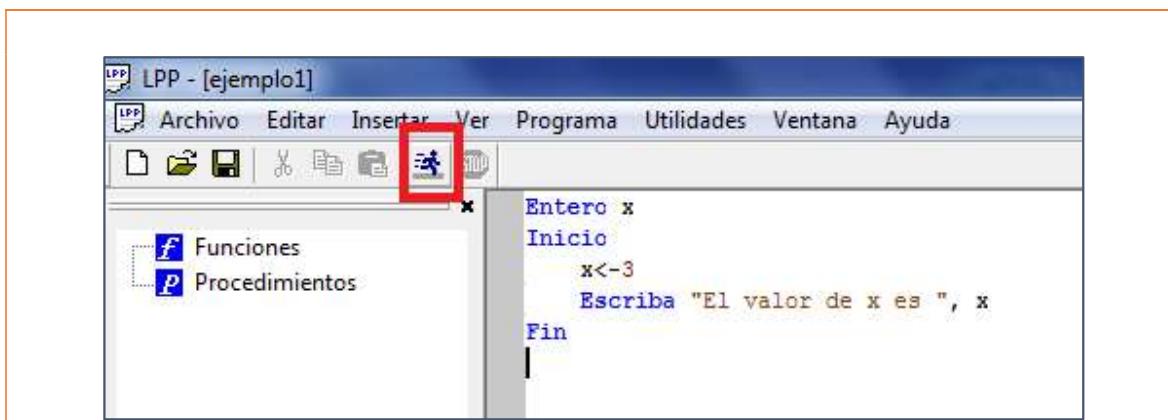


Figura 2-15 Opción ejecutar abreviada

Fuente propia

La segunda alternativa para ejecutar el código corregido o sin errores, es escoger la opción "Ejecutar" del menú superior Programa ver Figura 2.16.

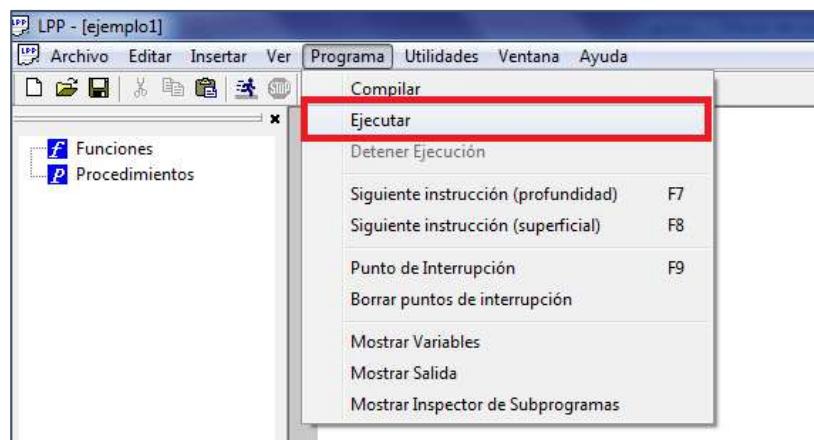


Figura 2-16 Opción ejecutar del menú programa

Fuente propia

Como resultado del procedimiento anterior, aparecerá una consola con el resultado de ejecución del programa, ver Figura 2.17.

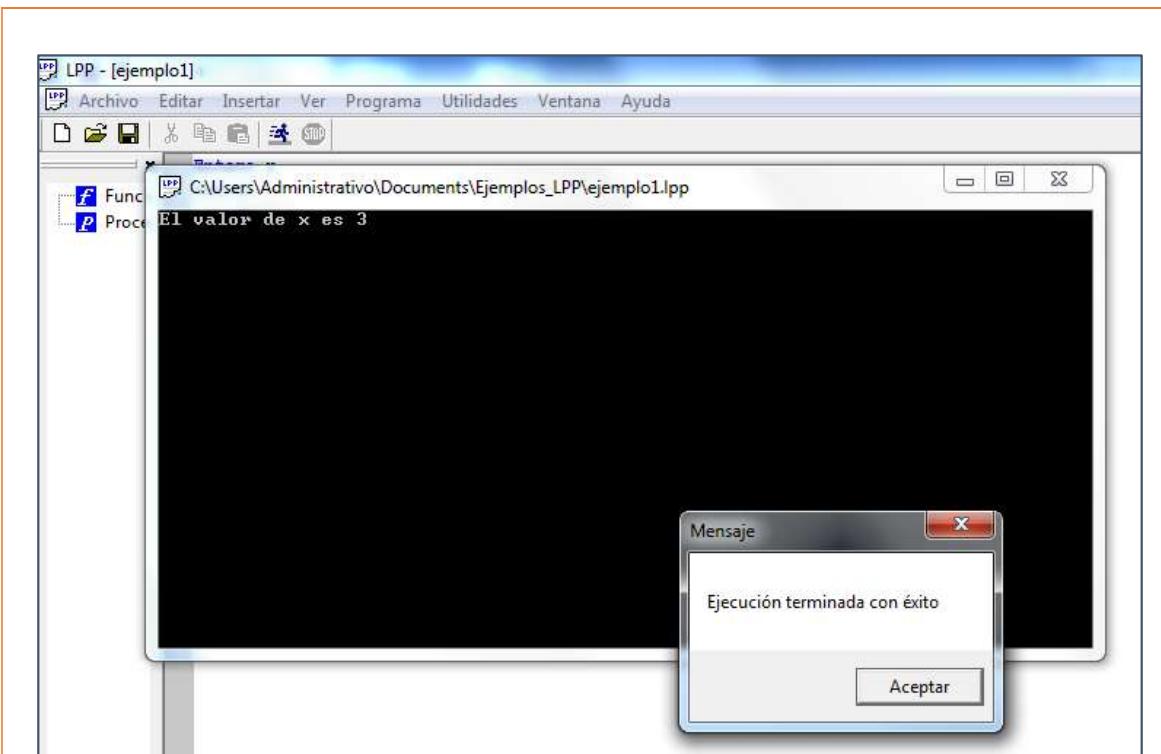


Figura 2-17 Resultado de la ejecución del programa

Fuente propia

2.6 Opciones adicionales en LPP

La herramienta LPP permite traducir el código escrito en otros lenguajes de programación como son: C++ y Ada 95. Adicionalmente la herramienta tiene la opción de generar un ejecutable del pseudocódigo escrito. Si se desea generar un equivalente en el lenguaje C++ del programa escrito en la Figura 2.10, se escoge la opción “Generar código C++” del menú superior Utilidades, ver Figura 2.18.

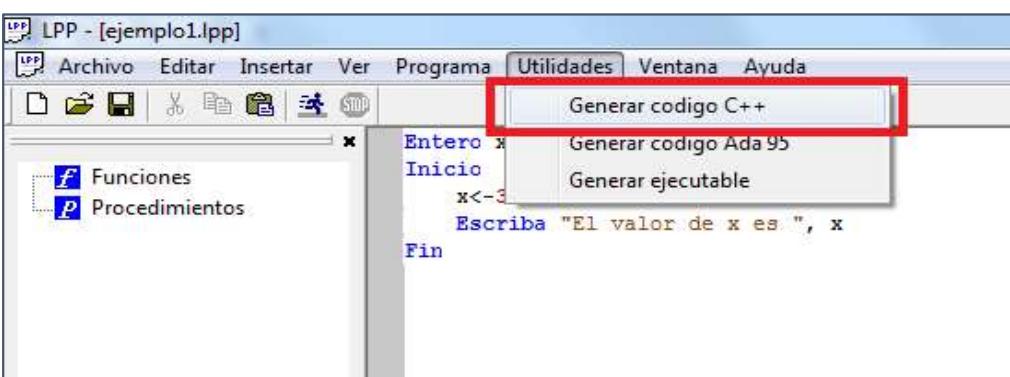


Figura 2-18 Generar código en C++
Fuente propia

Como resultado al generar al código en C++, la herramienta LPP genera en la misma carpeta del archivo fuente LPP, un archivo con la extensión .cpp ver Figura 2.19, la cual es propia del lenguaje C++. Así, el programa mostrado en el lenguaje C++ es equivalente funcionalmente al programa de LPP de la Figura 2.10.

```
ejemplo1.cpp
1 #include <stdio.h>
2 #include <conio.h>
3 #include <stdlib.h>
4 #include <math.h>
5 #include <io.h>
6 #include <fcntl.h>
7 #include <time.h>
8 #include <sys/stat.h>
9 #include <sys/types.h>
10 #include <string>
11 #include <iostream>
12 using namespace std;
13
14 int x;
15 void main() {
16     x = 3;
17     cprintf("El valor de x es %d", x);
18 }
```

Figura 2-19 Código generado en C++
Fuente propia

Si se desea generar un equivalente en el lenguaje Ada del programa escrito en la Figura 2.10, se escoge la opción Generar código Ada95 del menú superior Utilidades, ver Figura 2.20.

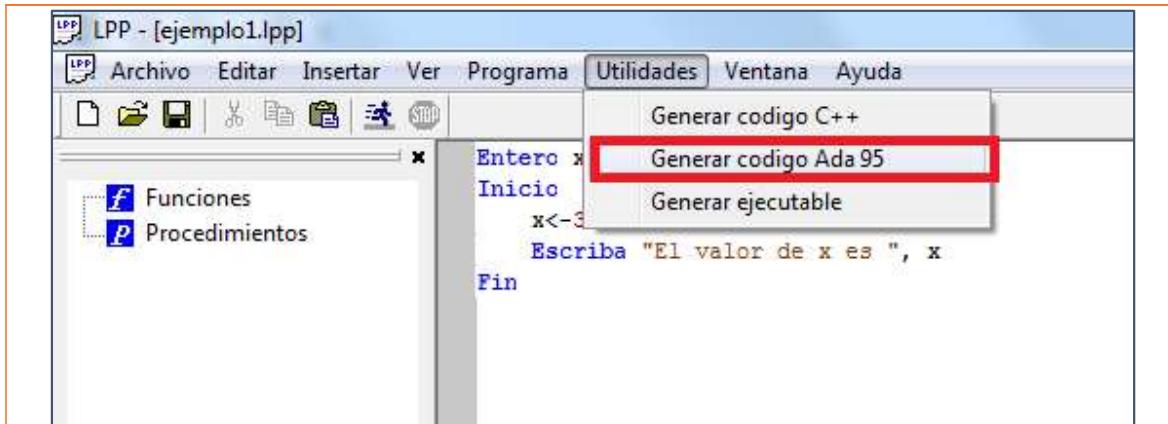


Figura 2-20 Generar código en Ada95
Fuente propia

Como resultado de lo anterior, la herramienta LPP genera en la misma carpeta del archivo fuente LPP, un archivo con la extensión .adb ver Figura 2.21, la cual es propia del lenguaje Ada95. Así, el programa mostrado en esta figura corresponde a un código generado en Ada95 es equivalente funcionalmente al programa de la Figura 2.10 que está hecho con código en LPP.

```

1  with Ada.Text_IO; use Ada.Text_IO;
2  with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
3
4  procedure ejemplo1 is
5    x : Integer;
6
7  begin --principal
8    x := 3;
9    Put("El valor de x es ");
10   Put(x, 0);
11 end ejemplo1;
12
13
14
  
```

Figura 1.21 – Código generado en Ada95

Fuente propia

Finalmente, si se desea generar un ejecutable del programa mostrado en la Figura 1.10, se selecciona la opción “Generar Ejecutable” del menú superior “Utilidades”, ver Figura 1.22.

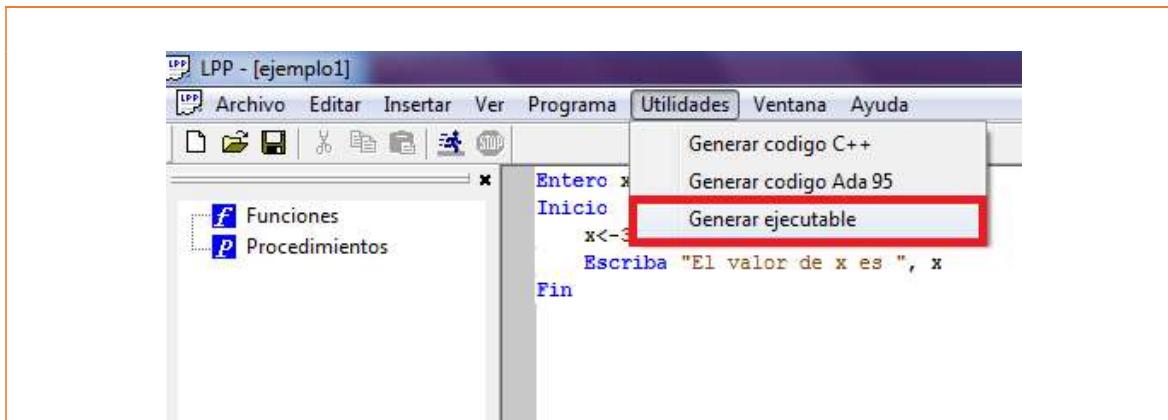


Figura 1.22 – Opción generar ejecutable

Fuente propia

Dado que la herramienta LPP trabaja por debajo con el compilador del lenguaje C, se hace uso de este para generar el ejecutable, ver Figura 1.23.

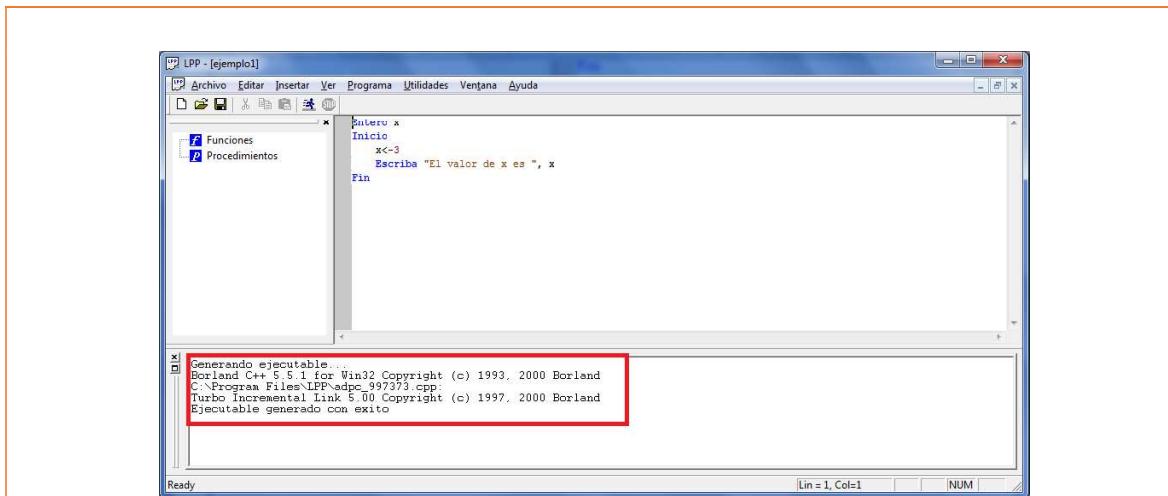


Figura 1.23 Generación del ejecutable en LPP

Fuente propia

Como resultado del procedimiento anterior, en la misma carpeta donde se encuentra el archivo fuente con la extensión LPP (ejemplo1.lpp), se genera un nuevo archivo ejecutable con extensión .exe (ejemplo1.exe), ver Figura 1.24. Un archivo ejecutable es aquel que puede ser lanzado de forma independiente, sin necesidad de usar una aplicación externa.

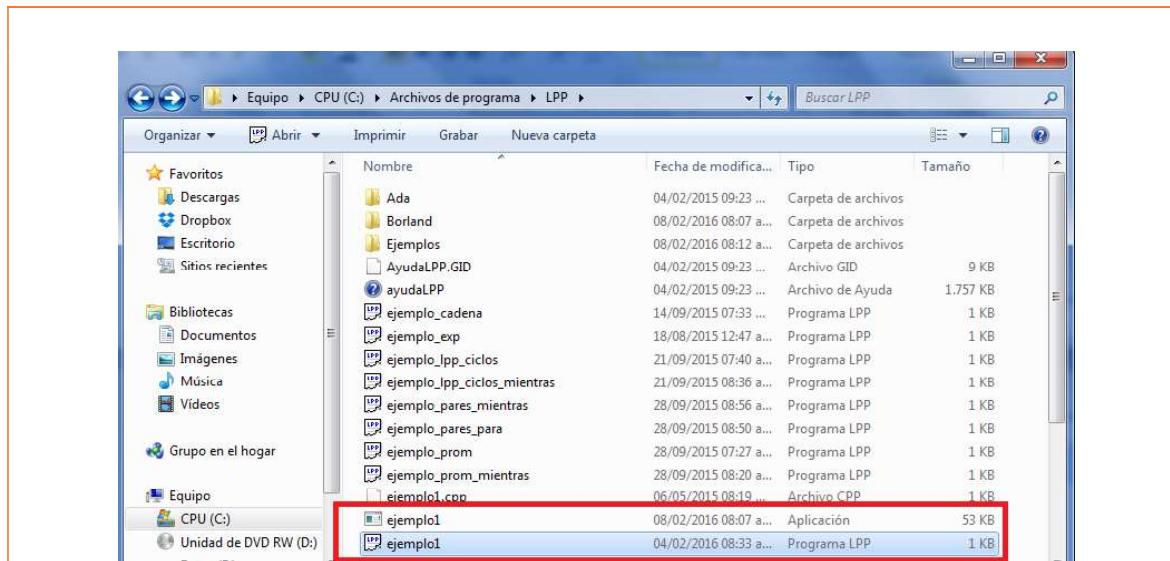


Figura 1.24 Archivo ejecutable generado
Fuente propia

Al lanzar el archivo ejecutable de la Figura 1.24, se observa como una ventana de color negro aparece y desaparece casi de manera instantánea, lo cual se debe a que el programa ejecutado no requiere de ningún valor de entrada por parte del usuario, por lo cual se ejecuta y se cierra. Para que el programa no se cierre, puede agregársele al programa de la Figura 1.10 una variable “valor” y una instrucción de lectura (Lea valor), ver Figura 1.25.

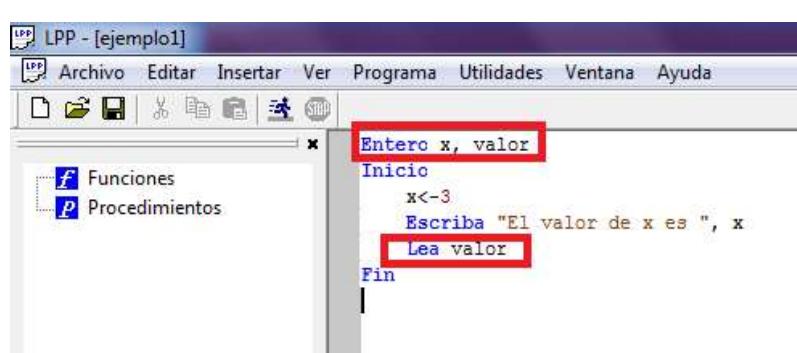


Figura 1.25 Captura de valor de entrada
Fuente propia

Las instrucciones agregadas en la Figura 1.25, permiten al usuario ingresar usuario deba ingresar un valor de entrada antes de que el programa se cierre. Este valor de entrada será almacenado en la variable "valor". De esta manera al generar el ejecutable del programa de la Figura 1.25, se puede apreciar que la consola de Windows se mantiene activa esperando que el usuario ingrese un valor determinado, ver Figura 1.26.

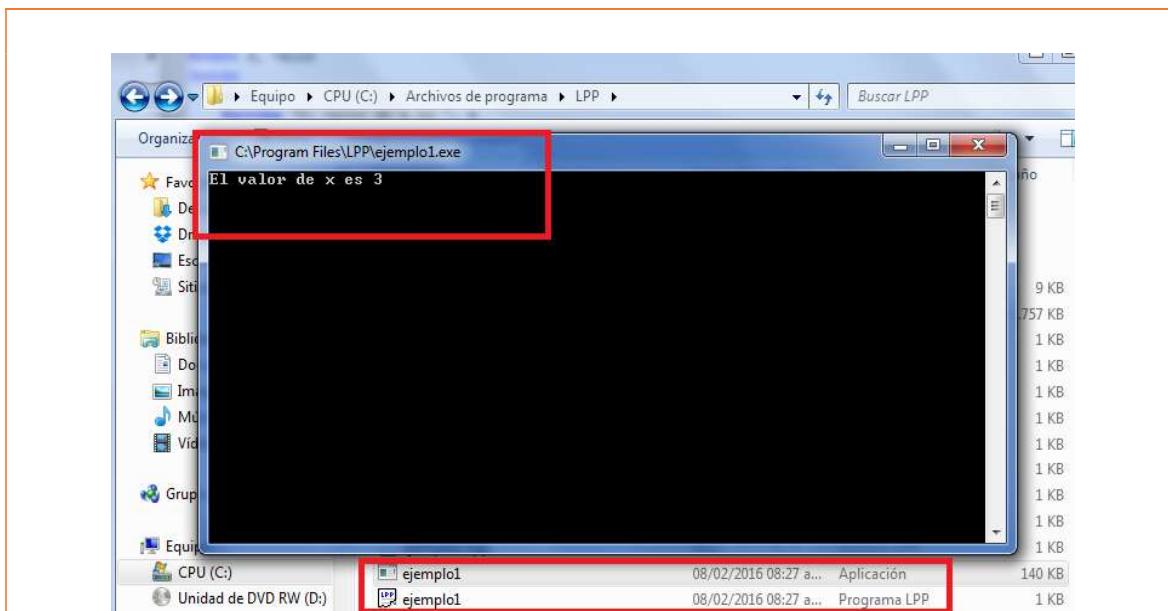


Figura 1.26 Lanzamiento ejecutable

Fuente propia

2.7 Ejercicios de la unidad

1. Sobre el entorno de la herramienta LPP, identificar la funcionalidad de cada una de las opciones del menú “Insertar”.
2. Haciendo uso del compilador en línea multilenguaje: repl.it, pruebe el funcionamiento del programa de la Figura 1.19. El compilador de c++ puede ser encontrado en el siguiente subdominio: <https://repl.it/languages/cpp>
3. Haciendo uso del compilador en línea del lenguaje Ada95 (disponible en: http://www.tutorialspoint.com/compile_ada_online.php), pruebe el funcionamiento del programa de la Figura 1.21.

2.8 Bibliografía

3 CAPÍTULO – CONCEPTOS DE BASICOS DE PROGRAMACION

3.1 Temas del capítulo

3.1.1 Contenido

- **Estructura de un programa en LPP**
- **Palabras reservadas**
- **Comentarios**
- **Salida de datos y entrada de datos**
- **Identificadores**
- **VARIABLES**
- **Tipos de datos primitivos**
- **Constantes**
- **Asignaciones**
- **Operadores matemáticos**
- **Operadores lógicos**
- **Jerarquía de operadores**
- **Operaciones matemáticas avanzadas (raíces y potencias)**
- **Ejercicios de la unidad**

En este capítulo, se explorará los elementos esenciales que constituyen la base para escribir y comprender programas en pseudocódigo en (LPP) lenguaje de programación para principiantes.

Comenzaremos con la estructura básica de un programa, entendiendo cómo se organizan sus componentes y cómo interactúan entre sí.

A continuación, se aprenderá las palabras reservadas, términos específicos del lenguaje en pseudocódigo que no pueden usarse como identificadores, y los comentarios, herramientas esenciales para documentar y mejorar la legibilidad del código. Además, profundizaremos en las operaciones básicas de interacción con el usuario, como la entrada y salida de datos, que facilitan la comunicación entre el programa y el usuario.

Dentro de este orden de ideas, estudiarás los conceptos como identificadores, variables, y tipos de datos primitivos y los errores comunes al definirlos, como también elementos fundamentales para almacenar y manipular información en el pseudocódigo. Sumado a esto veremos cómo usar constantes para mantener valores fijos, así como asignaciones para actualizar los valores de las variables.

Por otro lado, en el ámbito de las operaciones, analizaremos los operadores matemáticos, operadores lógicos, y la jerarquía de operadores, herramientas indispensables para construir expresiones complejas y controlar el flujo de ejecución como lo es la jerarquía de operaciones. También abordaremos operaciones matemáticas avanzadas como cálculos de raíces y potencias, ampliando las capacidades matemáticas de los programas.

Finalmente, el capítulo termina con una sección práctica de ejercicios de la unidad, diseñados para reforzar la comprensión de los conceptos abordados y fomentar la habilidad de aplicarlos en la resolución de problemas.

3.2 Estructura de un programa en LPP

La estructura de la aplicación LPP, lenguaje de programación para principiantes se refiere a cómo está diseñado el entorno en el que los usuarios interactúan, ejecutan y depuran programas en LPP.

Esta plataforma LPP es sencilla, intuitiva y estructurada, ofrece herramientas básicas que faciliten el aprendizaje, en la figura 3.1 presenta el área para codificar.

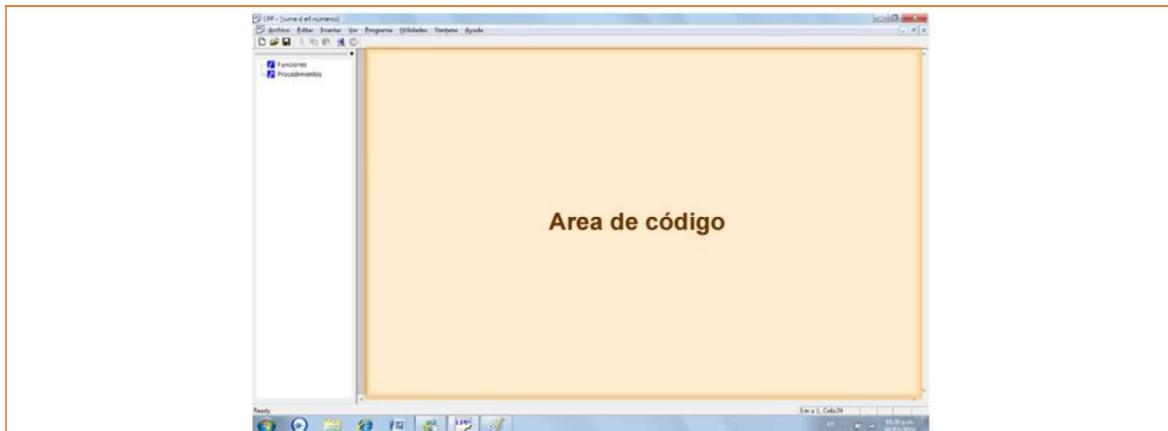


Figura 3-1 Área de código
Fuente propia

El área de código

El área de código del programa en LPP, es un editor donde el usuario puede escribir y editar programas, se puede dividir en 2 bloques principales.

Incluye características como:

- Coloreado de sintaxis.
- Numeración de líneas.
- Resaltado de errores sintácticos.
- Estructura del área del código:

Primer bloque: Está destinado a la declaración de variables y constantes que se usarán en el desarrollo de un programa.

Ejemplo

entero área, lado

Segundo bloque: Está enfocado a la escritura del flujo principal del programa.

Ejemplo

Inicio

Escriba “Digite lado del cuadrado”

Lea lado

area = lado * lado

Escriba “El área del cuadrado es ”,area

Fin

Consola o pantalla: Es un entorno basado en texto usado para:

- No tiene elementos gráficos
- Para interactuar con el sistema o un programa.
- Mostrar mensajes del programa.
- Permitir que el usuario ingrese datos para ser procesados por el programa

En la Figura 3.2 se muestra la estructura de un programa en LPP, en el primer bloque se declaran las variables, mientras que en el segundo bloque está delimitado por las palabras claves **inicio**, **fin**, y la consola de color negro que permite digitar y ver la información.

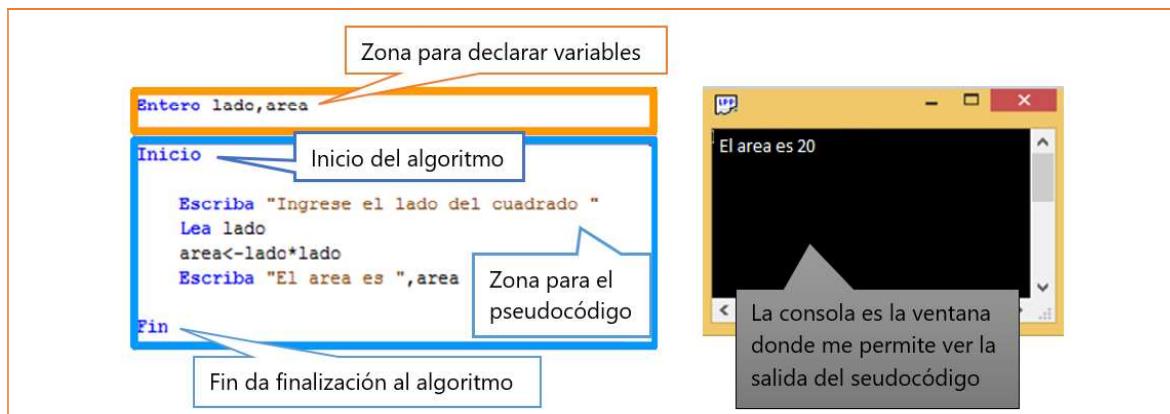


Figura 3-2 Estructura en LPP

Fuente propia

3.3 Palabras reservadas

Palabras reservadas en LPP

"Son identificadores de un lenguaje de programación que tienen un significado especial para el compilador"[3]

En LPP existen un conjunto de palabras reservadas por el programa, estos son términos que tienen un significado propio dentro del lenguaje y no pueden ser utilizados como identificadores, como nombres de variables o funciones.

Tabla 2.1 Palabras Reservadas

A continuación, se presenta una tabla con las principales palabras reservadas del lenguaje LPP, clasificadas de acuerdo con su funcionalidad. Dado que estas son palabras reservadas del lenguaje LPP, no pueden ser usadas para la definición de las variables.

Clasificación	Palabras reservadas
Estructura del programa	inicio, fin
Tipos de datos	entero, real, cadena, caracter, booleano, registro
Operadores matemáticos	mod, div
Entrada y salida de datos	escriba, lea, nueva_linea, llamar, Limpiar_pantalla, escribir
Sentencias condicionales	si, entonces, sino
Conectores lógicos	y, o
Sentencias repetitivas	mientras, para, repita, hasta, haga, repita
Procedimientos y funciones	procedimiento, funcion, llamar, Limpiar_pantalla, nueva_linea
Vectores y matrices	arreglo
Registros	Registro, de
Archivos	Tipo, es archivo secuencial, abrir, leer, como, escritura, lectura, escribir, cerrar, no, fda

Tabla 3.1 Palabras reservadas

Fuente propia

3.4 Comentarios en LPP

Definición: Los comentarios en pseudocódigo en LPP y programación son utilizados para inhabilitar bloques de código dentro de los programas, o para documentar la funcionalidad de una o varias líneas de código.

3.4.1 Comentarios para documentar la funcionalidad

Son anotaciones u observaciones del programador dentro del código fuente que aclaran el propósito, el funcionamiento de alguna parte del código. El compilador o intérprete no los ejecuta y son fundamentales para que otros desarrolladores puedan comprender fácilmente lo que realiza el código, ver figura 3.3.

```
/*
Comentario de bloque
usado para comentar varias lineas
Algoritmo
para hallar el area del cuadrado
*/
Inicio
    //comentario de una sola linea
    Escriba "Digite lado del cuadrado"
    Lea lado
    area = lado * lado
    Escriba "El área del cuadrado es ",area
Fin //finaliza el algoritmo
```

Figura 3-3 Comentarios de información

Fuente Propia

En LPP existen comentarios de línea y comentarios de bloque, en la figura anterior presenta al inicio del código en color verde el comentario en bloque usado para comentar varias líneas de información usando los caracteres **/* comentarios */**, también se visualiza los comentario de una sola línea **//comentario de una línea** para dar información de una funcionalidad del código.

3.4.2 Comentarios para inhabilitar el código

Se usa para deshabilitar temporalmente una o más líneas de código sin tener que borrarlas del archivo fuente como se ilustra en la figura 3.4.

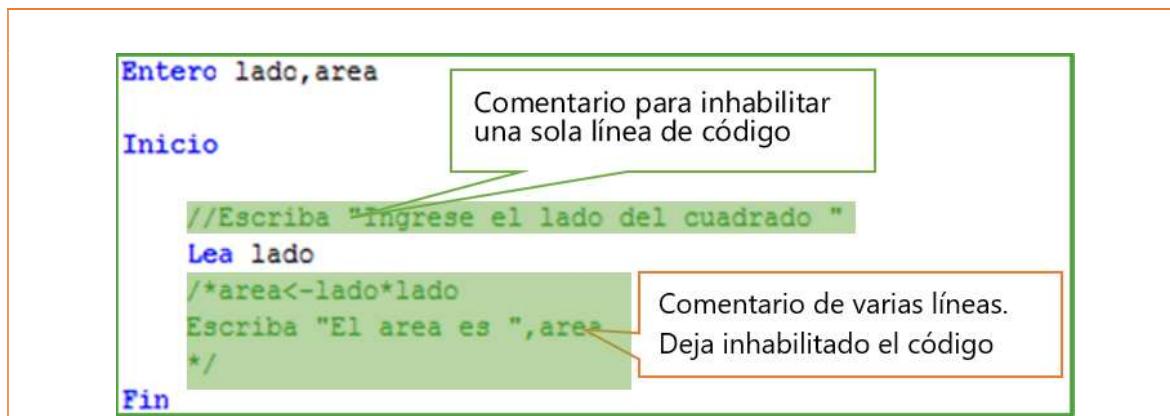


Figura 3-4 Comentarios en LPP

Fuente propia

3.5 Nomenclaturas de programación

Las nomenclaturas hacen referencia a las normas o convenciones que se aplican para otorgar nombres a variables, atributos métodos, procedimientos, clases, nombres de archivo y funciones. Estas convenciones tienen como objetivo potenciar la legibilidad, preservar la coherencia y simplificar la conservación del código.

Hay diferentes tipos de nomenclaturas o estilos de declarar el nombre de los identificadores de las variables a continuación se presentan las siguientes

3.5.1 Camel Case

Es un estilo de escritura usualmente utilizado para nombrar variables en el que las palabras compuestas se unen sin espacios, y cada palabra después de la primera comienza con una letra mayúscula [4].

Lower Camel Case, camelCase: La unión de dos identificadores para dar nombre a una variable, La primera palabra comienza con minúscula y las siguientes con mayúscula.

Ejemplo nombreDeEstudiante, edadVigilante.

Upper Camel Case o PascalCase: La primera letra de cada palabra comienza con mayúscula.

Ejemplo NombreUsuario, GeneroDelDocente.

Utilizado normalmente con los lenguajes de programación como Java, JavaScript y C# para nombrar variables, funciones y clases.

3.5.2 **Snake Case**

Upper Snake Case Utilizado para asignar nombres de variables constantes [4], es un estilo de escritura en el que las palabras compuestas, se separan con guiones bajos (_) y todas las letras se escriben en mayúsculas con la opción.

Ejemplo NOMBRE_DE_USUARIO, EDAD_DEL_CLIENTE.

Lower Snake Case: Todas las letras están en minúsculas separadas con (_) guion bajo.

Ejemplo nombre_de_estudiante, edad_de_usuario.

3.5.3 **Pascal Case**

Es una nomenclatura en la que cada palabra comienza con una letra mayúscula, sin espacios ni separadores, usadas en nombramiento de clases e interfaces espacios de nombres[4], utilizada frecuentemente para nombrar clases en muchos lenguajes, como C#, Java y TypeScript.

Ejemplo: AgendaDeContactos, CalculadoraBasica, TallerAlgoritmos.

Camel Case: nombreProfesor, edadDelVendedor.

Snake Case: suma_del_total, edad_de_usuario.

Pascal Case: JTextField, Scanner, CalculadoraCientifica.

Además de Camel Case y Snake Case, y Pascal Case hay diversos estilos de nomenclaturas empleadas en la programación, cada una con sus propias normativas y situaciones de aplicación. A continuación, se presenta algunas de las más utilizadas según [5] .

3.5.4 **Nomenclatura de identificadores en programación**

Nomenclatura de identificadores en programación		
Kebab Case - Dash Case	Ejemplo	Usado en

Las palabras se separan con guiones medios (-) y todas las letras suelen estar en minúsculas.	nombre-de-usuario, edad-del-cliente.	URLs, nombres de archivos en proyectos web, CSS, clases, IDs.
Upper Case - Screaming Snake Case		
Todas las letras están en mayúsculas y las palabras se separan con guiones bajos (_).	NOMBRE_DE_USUARIO, EDAD_DEL_CLIENTE.	Para definir constantes en muchos lenguajes, como Python, JavaScript o Java.
Flat Case		
Todas las palabras se escriben en minúsculas y sin separadores.	edaddelusuario, pesodelcliente.	Nombres de archivos o identificadores simples.
Hungarian Notation		
Se utiliza un prefijo para indicar el tipo de dato o el propósito de la variable.	strNombre txtApellido, intEdad boolActivo (booleano).	C y C++, ya no es utilizado.
Train Case		
Similar al Kebab Case, pero cada palabra comienza con mayúscula.	Apellido-De-Usuario, Edad-Del-Cliente.	Nombres de archivos o títulos.
Dot Notation:		
Las palabras se separan con puntos (.)	nombre.de.usuario, cliente.sueldo estudiante.nombre	Frameworks para acceder a propiedades o métodos objeto.propiedad.
Spinal Case:		
Las palabras se separan con guiones medios (-).	nombre-de-usuario, sueldo-cliente.	Nombres de archivos o rutas.
Lisp Case		
Las palabras se separan con guiones medios (-) y todas las letras están en minúsculas.	nombre-de-usuario, edad-del-cliente.	Lisp o Scheme.
Macro Case		
Todas las letras están en mayúsculas y las palabras se unen sin guiones bajos.	NOMBREDEUSUARIO, EDADDELCLIENTE.	Nombres de macros o constantes.
Cobol Case		

Las palabras se separan con guiones medios (-) y todas las letras están en mayúsculas.	NOMBRE-DE-USUARIO, EDAD-DEL-CLIENTE.	Uso común: Se utiliza en el lenguaje COBOL.
Reverse Domain Notation		
Utilizado en nombres de paquetes o espacios de nombres, donde se invierte el dominio y se separa con puntos (.).	com.ejemplo.proyecto, org.openoffice.util.	Lenguajes como Java en los paquetes o en sistemas de gestión de dependencias, Android estudio.

Tabla 3.2 Estilos de nomenclatura

Fuente propia

En la tabla 3.3 presenta los lenguajes de programación y su estilo de nomenclatura en variables, funciones, constantes y funciones.

Resumen por lenguaje:			
Lenguaje	Variables/Funciones	Clases	Constantes
JavaScript	camelCase	PascalCase	UPPER_CASE
Python	snake_case	PascalCase	UPPER_CASE
Java	camelCase	PascalCase	UPPER_CASE
C#	camelCase	PascalCase	UPPER_CASE
C++	camelCase	PascalCase	UPPER_CASE
Ruby	snake_case	PascalCase	UPPER_CASE
PHP	snake_case	PascalCase	UPPER_CASE

Tabla 3.3 Lenguajes y nomenclatura

Fuente propia

3.6 Identificador

Concepto

Es el nombre único en un ambiente preciso que se le da a las variables, constantes, clases, métodos o funciones dentro de un programa[6]. El identificador se compone de una serie de caracteres formados por letras, dígitos y el carácter subrayado (_), pero estos identificadores tienen una serie de restricciones. A continuación, se presentan algunas restricciones y recomendaciones para el uso de los identificadores.

En la sintaxis siguiente se usan los identificadores nombre, apellido, edad, peso

Sintaxis

Cabecera:

```
entero edad  
cadena[15] nombre, apellido  
real peso
```

inicio

Fin

3.6.1 Recomendaciones

Nombres significativos

Al utilizar identificadores es recomendable utilizar nombres significativos que reflejen el uso funcional del elemento al que se refieren, pero no excesivamente largo, es una buena práctica para saber para qué se usó el identificador según el nombre.

Identificadores en minúsculas

Los identificadores de las variables se escriben con minúsculas, si son palabras compuestas, se les coloca un guion bajo (_) para separar cada palabra.

Seguir un estilo de escritura consistente

Usa un estilo de escritura o nomenclatura coherente para los nombres de los identificadores de las variables, constantes funciones o métodos. Los estilos más comunes son:

camelCase: nombreCompleto, sumaTotal.

snake_case: edad_estudiante, sumar_pares.

PascalCase: EdadUsuario, SumarImpares (común en nombres de clases o tipos).

La recomendación elige un estilo *de código coherente, aunque en algunos casos puede ser necesario combinar diferentes estilos*. Mas delante en relación con este tema se explicarán varios estilos o nomenclatura de la escritura de los identificadores.

Caracteres permitidos

Los identificadores se pueden escribir letras, dígitos (0-9) y guiones bajos (_). Sin embargo, no se permiten caracteres especiales como espacios, signos de puntuación o símbolos, ni tildes.

Evitar palabras reservadas

No se debe usar palabras reservadas del lenguaje de programación o pseudocódigo como nombres de identificadores.

Ejemplo

Identificador	Evaluación
entonces	X
inicio	X

Tabla 3.4 Restricciones de palabras reservadas

Fuente propia

El identificador no puede ser una palabra reservada del lenguaje de programación usado, en la tabla 3.5 se usaron los identificadores **entonces**, **inicio** que son palabras reservadas y el lenguaje genera un conflicto presentando un error al compilar.

Mantener una longitud adecuada

Los nombres deben ser lo suficientemente largos para ser descriptivos, pero no tan largos que dificulten la lectura.

Ejemplo

Identificador	Evaluación
contador	✓
suma_total	✓
registrarEstudiantesAprobados	✓
contador_de_elementos_en_la_lista_de_entrada	X

Tabla 3.5 Longitud de variables

Fuente propia

Usar nombres en inglés (opcional pero recomendado)

En la programación experta, es común usar nombres en inglés para mantener un estándar global, para ser entendido por diferentes programadores.

Ejemplo

En lugar de edad, usa age. En lugar de suma_numeros, usa add_numbers.

Evitar Indefiniciones

Asegúrate que al crear los identificadores los nombres no sean confusos.

Ejemplo

- En lugar de datos, usa dato_del_usuario.
- En lugar de seguro, usa seguro_accidentes.
- En lugar de suma, usa suma_pares.

Evitar Indefiniciones

Asegúrate que al crear los identificadores los nombres no que no sean confusos.

Ejemplo

- En lugar de datos, usa dato_del_usuario.
- En lugar de seguro, usa seguro_accidentes.
- En lugar de suma, usa suma_pares.

Comentarios

Debes documentar cuando el código es complejo y sea necesario. Si un identificador no es completamente auto explicativo, agrega comentarios para aclarar su propósito.

Diferenciar entre variables, constantes y funciones

Al asignar un identificador debe distinguir las variables, constantes y funciones con nomenclatura diferente, en la figura 3.7 puedes usar la siguiente recomendación.

Tipo	Recomendación	Ejemplo
Variable	snake_case	edad_usuario, suma_total
Constantes	UPPER snake_case	MAX_INTENTOS, PI, IVA_COLOMBIA
Funciones	camelCase	calcularPromedio, verificarUsuario.
Clase, Archivo	PascalCase	CalculadoraBasica, EstudiantesElegidos

Tabla 3.6 Recomendación estilo de identificador

Fuente propia

3.6.2 Restricciones

A continuación, se presentan las restricciones para el uso de identificadores.

Restricción de nombres genéricos

Evita asignar nombres o identificadores genéricos como x, y, temp, a, etc., a menos que su uso sea obvio en un contexto muy específico.

Restricción de caracteres especiales y espacios

Los identificadores no deben contener espacios en blanco ni caracteres especiales como: @, #, \$, etc, \, +, ?. Solo se permiten letras, números y guiones bajos (_).

Ejemplo

Identificador	Evaluación
<i>numero_auxiliar</i>	✓
<i>numero par</i>	✗
<i>numero*total</i>	✗

Tabla 3.7 Restricciones de palabras compuestas

Fuente propia

En la tabla 3.8 8 presenta 2 errores por usar espacio en blanco en medio de dos palabras y el segundo error se colocó el símbolo asterisco.

Restricción longitud del Identificador

El identificador no debe pasar de la longitud máxima y varía según el lenguaje de programación.

No comenzar con números

Los identificadores no deben empezar con un número, aunque pueden contener números en otras posiciones.

Ejemplo

Identificador	Evaluación
<i>nombre</i>	✓

numero1	✓
2numero	✗

Tabla 3.8 Restricciones de letra
Fuente propia

La tabla 3.9 presenta un error al iniciar con un número, la recomendación es iniciar el identificador con letras.

3.6.3 Recomendaciones para las funciones

Se aplican las mismas reglas generales que para cualquier otra variable como:

- Los caracteres permitidos letras (A-Z, a-z), números (0-9)
- Usar guion bajo (_) para unir palabras.
- No puede comenzar con un número,
- No usar palabras reservadas,
- No puede contener espacios ni caracteres especiales,

Pero hay algunas consideraciones adicionales para mantener un código limpio, legible y funcional.

- El nombre debe ser descriptivo, pero no excesivamente largo, El nombre de la función debe estar relacionado con la tarea que realiza la función
Ejemplo `sumaTotal`
- Al declarar una variable local, su identificador o nombre no deben ser el mismo que la variable global o nombre de la función.
- Evitar nombres genéricos:

Sigue las convenciones del lenguaje que estés utilizando. Por ejemplo:

- **Camel Case:** `calcularTotal` (usado en JavaScript, Java, C#).
- **Snake Case:** `suma_pares`(común en Python, Ruby).

Ejemplo

Identificador	Evaluación
listarApellidos()	✓
promedioNotas()	✓

Tabla 3.9 Restricciones de funciones

Fuente propia

La tabla 3.10 describe dos funciones claras y útiles: una para listar apellidos y otra para calcular promedios.

Métodos o funciones en java

En el lenguaje Java, cuando un identificador de un método o función está compuesto por dos o más palabras, se utiliza la notación camelCase conocida como notación de camello. Esto significa que la primera palabra se escribe completamente en minúsculas, y si hay palabras adicionales, cada una de ellas comienza con una letra mayúscula. Esta práctica es estándar en Java y ayuda a mejorar la legibilidad del código.

Ejemplo

Identificador	Evaluación
<code>operacionBasica()</code>	✓
<code>calculoTotalNomina ()</code>	✓
<code>calcularAreaTriangulo</code>	✓

Tabla 3.10 Identificador de funciones

Fuente propia

Constantes

Los identificadores de las constantes se escriben en mayúsculas, si son dos o más palabras, se les coloca un guion bajo (_) para separar cada palabra, usa la sintaxis **UPPER SNAKE_CASE**.

Ejemplo

Identificador	Evaluación
<code>PI</code>	✓
<code>TOTAL_ALUMNOS</code>	✓

Tabla 3.11 Restricciones de constantes

Fuente propia

Ejemplo Identificadores válidos:

<i>Identificador</i>	<i>Evaluación</i>
<i>numero</i>	✓
<i>numeroPar</i>	✓
<i>numero_impar</i>	✓
<i>numeroimpar</i>	✓
<i>nombre</i>	✓
<i>apellido1</i>	✓
<i>apellido2</i>	✓

Tabla 3.12 Indicadores válidos

Fuente Propia

Ejemplos identificadores no válidos

<i>Identificador</i>	<i>Evaluación</i>
<i>1nombre</i>	✗
<i>\$valor</i>	✗
<i>_color</i>	✗
<i>saldo total</i>	✗
<i>para</i>	✗
<i>A</i>	✗

Tabla 3.13 Indicadores no válidos

Fuente Propia

En las tablas anteriores se describen y se evalúan ejemplos de los de los Indicadores válidos y no válidos.

3.7 Entrada proceso y salida de datos

En todo algoritmo, se deben tener en cuenta las entradas los procesos y por último la salida. La entrada y salida de datos consiste en asignar y obtener datos de la

memoria principal en la computadora, estos datos son introducidos por el usuario mediante un dispositivo de entrada como: teclado, o mouse, entre otros.

En la manipulación o transformación sobre los datos de entrada se genera un resultado. Estas operaciones incluyen cálculos, comparaciones, almacenamiento temporal, operaciones lógicas, etc. A continuación, se indica que los resultados pueden ser presentados en otro dispositivo como la pantalla o impresora, etc. La entrada y salidas de datos se le conoce como: lectura y escritura de datos ver figura 3.5.



Figura 3-5 Entrada proceso y salida

Fuente propia

3.7.1 Entrada (lea)

Definición:

La entrada en donde se proporcionan valores a una variable [6]. La instrucción **lea**, es la sentencia utilizada en LPP para capturar la entrada de datos que el usuario introduce en el sistema. Esta palabra clave se implementa de la siguiente manera:

Lea número

Sintaxis

Cabecera:

inicio

lea variable
Fin

Ejemplo

En la Figura 3.6 presenta el pseudocódigo de LPP con la instrucción Lea, la cual indica a la computadora que capture por consola un dato digitado por el teclado y a continuación este se le asigna a la variable número, para ser llevada a la memoria principal RAM donde se guarda temporalmente.



Figura 3-6 Entrada de datos
Fuente propia

3.7.2 Proceso

Definición:

El proceso es la etapa donde se indica la acción, operación o instrucción a ejecutar [6]. Se refiere al conjunto de pasos o acciones definidas que se ejecutan para transformar una entrada (datos iniciales) en una salida (resultado). Es la secuencia lógica y ordenada de operaciones que resuelven un problema o realizan una tarea específica.

En la figura 3.7 presenta un teclado como entrada de datos luego el proceso para ser presentado en la pantalla.



Figura 3-7 Proceso de datos
Fuente propia

Sintaxis

Cabecera:

entero numero1, numero2, suma

inicio

numero1 <- 5

numero2 <- 3

suma <- numero1 + numero2

escriba "La suma es ", suma

Fin

Al tener los datos de entrada, asignados a numero1 y numero2 almacenados en memoria, se realiza el proceso o cálculo de esas dos variables y en seguida toma el resultado de la adición a la variable suma y se presenta en un dispositivo de salida como: pantalla, impresora.

En las tareas del proceso de datos podemos considerar lo siguiente:

- Realizar cálculos matemáticos.
- Buscar, clasificar, ordenar los datos, encontrar mayor y menor etc.
- Tomar decisiones basadas en los condicionales
- Transformar los datos de un formato a otro.
- Gestionar información en la base de datos.

3.7.3 Salida

Salida de datos (escriba)

Es la etapa en donde se obtienen los resultados de las operaciones y el resultado o la solución del problema[6]. La instrucción **escriba** se utiliza para imprimir información en la salida estándar de la consola, ya sea texto o resultados de cálculos

En LPP, para la salida de datos se utiliza la sentencia **escriba**, y luego entre comillas (" ") se encierra el texto a mostrar. Si se necesita imprimir el valor de una variable, se coloca una coma después del texto y se escribe el nombre de la variable.

Sintaxis

Cabecera

inicio

escriba "Digita tu nombre: "

lea nombre

escriba "Tu nombre es: ", nombre

Fin

Ejemplo:

En la Figura 3.8 se muestra un código en LPP que utiliza las instrucciones inicio y fin. Dentro del flujo del proceso, se encuentra la sentencia **escriba "Mensaje de información"**, la cual permite imprimir o mostrar información en la ventana o consola (por pantalla).

Explicación:

inicio y fin: Delimitan el bloque de código en LPP.

escriba: Es la instrucción que imprime texto o variables en la pantalla.

"Mensaje de información": Es el texto que se muestra en la consola.

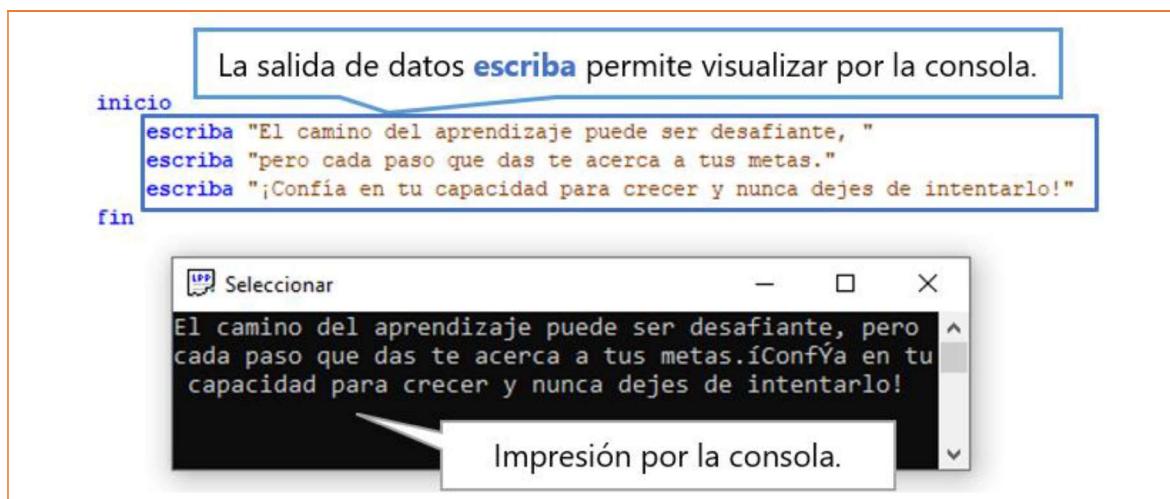


Figura 3-8 Salida de datos
Fuente propia

Ejemplo

A continuación, se presenta el pseudocódigo figura 3.9 donde se pide el nombre y apellido, el cual tiene dos variables de entrada nombre, apellido, y posteriormente se imprime los datos con la sentencia escriba.

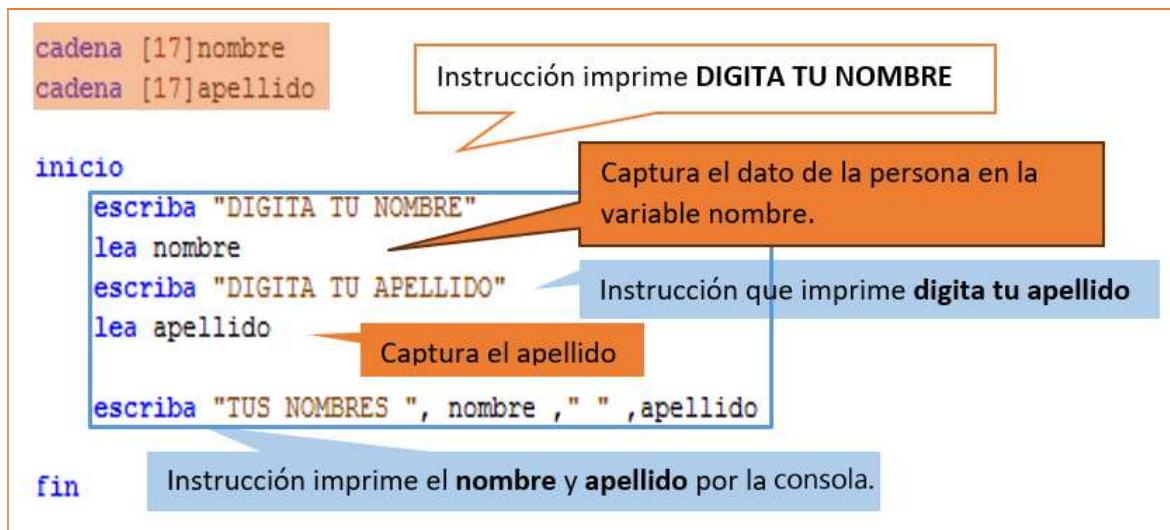


Figura 3-9 Salidas con variables
Fuente propia

Ejemplo

A continuación, se presenta el pseudocódigo de entrada, proceso y salida de datos representado en la figura 3.10 Algoritmo que permite calcular el área de un triángulo utilizando la siguiente fórmula $\text{area} = (\text{base} \times \text{altura})/2$.

```

entero base, altura, area
inicio

    //Entrada de datos
    escriba "Digite la base "
    lea base
    escriba "De la altura "
    lea altura

    //Proceso
    area <- (base * altura) / 2

    //SALIDA
    escriba "El area es: ", area
fin

```

Figura 3-10 Entrada, proceso y salida
Fuente propia

3.8 Variables

Variable

Una variable permite reservar un espacio en memoria para almacenar o guardar datos de forma temporal y solo guarda un dato a la vez [7]. La variable no es más que un símbolo o identificador, que representa un valor en la memoria y además tiene la capacidad de cambiar de contenido, mientras se desarrolla la ejecución del programa. Al declarar una variable, se reserva un área de la memoria RAM memoria de acceso aleatorio. Una variable se compone de un identificador, un tipo de dato.

Sintaxis

Área para las variables

```

entero numero1, numero2, suma
cadena[50] nombres
carácter estado_civil
booleano estudiante_activo

inicio
    suma <- numero1 + numero2
    escriba "La suma es ", suma
Fin

```

En la sintaxis se presenta como se declaran las variables en el área de variables los diferentes tipos de datos primitivos.

3.8.1 Proceso interno de las variables de la memoria

En programación las variables se almacenan en diferentes áreas de la memoria RAM memoria de acceso directo dependiendo el tipo de dato (entero, carácter, decimal etc.) y clase de almacenamiento[7] . Ejemplos aproximados

MEMORIA	
Dirección	Contenido
0	
1	
2	
3	
3000	
3001	120
3002	7
3003	127
3004	

X	3001
Y	3002
Rta	3003

Ejemplo
X =120
Y = 7
Rta=X+Y
X apunta a la dirección 3001
Y apunta a la dirección 3002
Rta apunta a la dirección 3003

Figura 3-11 Aproximación de variables
Fuente propia

En la figura 3.11 presenta una aproximación de uso de las variables en la memoria RAM, las variables **x**, **y** de tipo primitivo entero, con los valores de 120 y 7 se almacenan directamente en el stack. Una vez que el método termina, la memoria asignada se libera automáticamente.

Al declarar una variable, la computadora conoce cómo codificar la información que se va a almacenar en la posición correspondiente de la memoria. Así mismo, se reserva el espacio de memoria necesario para almacenar un valor del tipo de la variable. Este identificador tiene acceso al dato almacenado en la memoria y puede ser modificado.

Ejemplo

En el ejemplo siguiente se realiza una aproximación del proceso interno de almacenar y acceder a las variables **a**, **b**, en la memoria RAM, pero en realidad, la gestión de memoria es más compleja y tiene demasiados factores como la arquitectura del sistema y el modelo de memoria utilizado.

entero a, b

inicio

```
a <- 1  
b <- 2
```

Fin

En la memoria la memoria de acceso aleatorio RAM del computador imagina una serie de casillas o celdas donde se almacenan los datos (binarios). Cada casilla tiene una dirección única y puede contener un valor específico, representado en la tabla 3.14.

Paso 1. Declaración de variables a y b

Al declarar las variables **a** y **b** en LPP, el sistema operativo reserva espacio en la memoria RAM para almacenar sus valores. Simulando que la variable **a** se almacena en la dirección de memoria 0x1000 y variable **b** en la dirección 0x1004.

Por lo tanto, el código, al declarar las variables, el compilador o intérprete reserva espacio en la memoria RAM para almacenar sus valores. Esto lo hace automáticamente la máquina.

Paso 2. Asignación de valores

Después de dar inicio al algoritmo se asignan los valores **a**-1 y **b**<-2, estos valores se escriben en las respectivas direcciones de memoria en la tabla siguiente.

Dirección de memoria	Valor almacenado (en bytes)	Variable
0x1000	01 00 00 00	a = 1
0x1004	02 00 00 00	b = 2

Tabla 3.14 Representación Dirección de memoria ficticia
Fuente propia

En el ejemplo de la 3.14 se simula las variables (a, b) usadas en LPP, es necesario mencionar que cada posición de memoria o byte se identifica mediante una dirección o número de posición con el nombre de dirección de memoria [8], y en esta ilustración cada variable ocupa un espacio de memoria de 4 bytes, suponiendo que se trata de enteros de 32 bits. La dirección de memoria aumenta en 4 bytes para cada variable, ya que cada variable ocupa ese espacio de memoria.

Para usar las variables el sistema accede a la memoria RAM en las direcciones asignadas y de esta manera con la dirección se puede obtener o modificar esos valores.

Datos adicionales

Tamaño de las variables: Cada variable depende del tipo de dato.

Ejemplo.

- Un entero de 32 bits ocupa 4 bytes.
- Un entero de 64 bits ocupa 8 bytes.

Estructura de la Memoria

La memoria RAM almacena datos en forma temporal hasta que se apaga o reinicia el ordenador y estos datos se pueden acceder en cualquier momento[8] , también organiza en bloques o páginas, y el acceso a ella se gestiona mediante direcciones de memoria únicas.

3.9 Datos y tipos de datos

3.9.1 Datos

El primer objetivo de toda computadora es el tratamiento de la información o de los datos. Estos datos son necesarios para efectuar el manejo de nombres de personas, productos, precios, ventas en un supermercado o las notas del estudiante.

Un dato es la expresión o nombre que describe los objetos con los cuales opera una computadora. Las computadoras operan con varios tipos para dar un resultado.

Concepto:

Un dato es una unidad mínima de información [9], corresponde a una representación simbólica (numérica, alfabética, algorítmica, espacial, etc.) de un atributo o variable.

En programación existen dos tipos de datos: tipos de datos básicos primitivos o simples y tipos de datos estructurados. A continuación, se describe cada uno de ellos.

3.10 Tipos de datos básicos

"Los datos simples son datos que contienen un solo valor que puede ser un número, letras o datos lógicos" [10]. Estos también son conocidos como tipos de datos simples, primitivos o sin estructura, es decir son tipos de datos que almacenan valores numéricos y alfanuméricos.

Los tipos de datos básicos del lenguaje LPP, son los básicos, a nivel de máquina son un conjunto o secuencia de bits (dígitos 0 o 1). Los tipos de datos simples en LPP son los siguientes:

- Numéricos (enteros, reales)
- Lógicos (booleano)
- Carácter (caracter, cadena)

3.10.1 Datos numéricos

El tipo de dato numérico es el conjunto de los valores numéricos y se representan en dos formas distintas:

- Tipo numérico entero.
- Tipo numérico real.

3.10.2 Tipo entero

Representan los números enteros, también son tipos simples o primitivos [11]. El tipo entero es un subconjunto finito del conjunto infinito de los números enteros. Los enteros son números completos, es decir no tienen componentes fraccionarios o decimales y pueden ser negativos o positivos. Algunos ejemplos de números enteros se presentan en la tabla 3.15 tipos de enteros.

7

-5

20

-31

Tabla 3.15 Tipos enteros

Fuente propia

En la figura 3.12 se presenta un ejemplo en LPP, que declara una variable de tipo entero (edad), le asigna el valor de 37 y la muestra en pantalla.

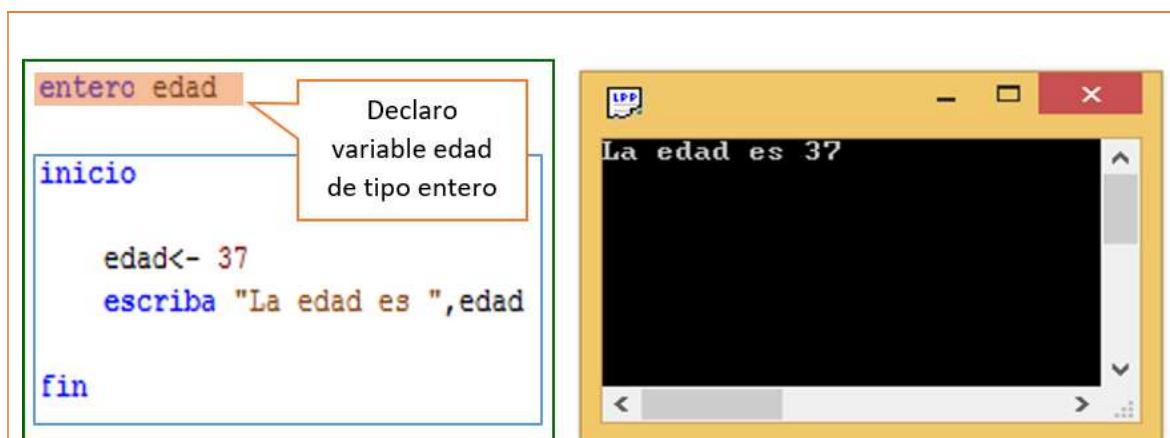


Figura 3-12 Tipo entero

Fuente propia

3.11 Tipo real

Representan números reales[11], el tipo real consiste en un subconjunto de los números reales. Los números reales son los que tienen decimales y pueden ser positivos o negativos. Un número real consta de un entero y una parte decimal.

Ejemplo

0.08	3739.41	3.7452	-52.321
------	---------	--------	---------

```

real pi
Declaro variable Real PI

inicio
    pi<-3.14
    escriba " El valor de Pi es ", pi
fin

```

The output window shows the text: "El valor de Pi es 3.14".

Figura 3-13 Tipo real
Fuente propia

3.12 Datos lógicos

Representa un valor booleano que puede ser verdadero (true) o falso (false)[11], El dato lógico es un tipo de información que se representa mediante valores booleanos.

El dato lógico sólo puede tomar un valor a la vez, es verdadero o es falso, El tipo lógico también denominado booleano, se utiliza ampliamente en estructuras de control, para tomar decisiones basadas en condiciones específicas como en la tabla siguiente.

Ejemplo

vendido=verdadero	vendido=falso
--------------------------	----------------------

Tabla 3.16 Tipo de datos lógicos
Fuente propia

En la figura 3.14 de los tipos de datos lógicos, se declaró la variable comprada de tipo booleano. Después de iniciar el algoritmo se captura por teclado a la variable comprado la pulsación digitada por el usuario,

Nota Al digitar el valor se usa 1 para verdadero y 0 para falso.

El tipo de dato lógico es usado para condiciones (si entonces, sino), si es verdadero o si no lo es.

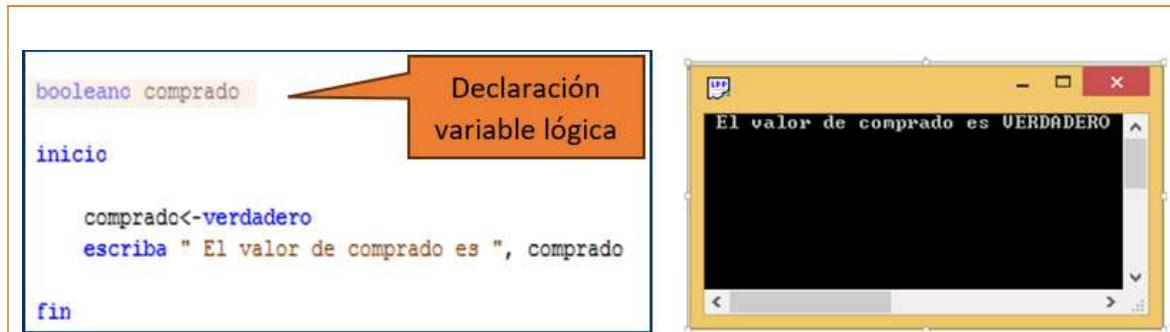


Figura 3-14 Tipo de datos lógicos
Fuente propia

En LPP también se emplean los datos tipo texto estos se dividen en carácter y cadena.

3.12.1 Tipo carácter

El tipo de carácter es el conjunto de caracteres, son los caracteres individuales [10]: letras, cifras, signos de puntuación etc. Este tipo dato solo tiene un carácter y pueden ser alfabéticos, numéricos y especiales, Se especifica con comillas simples 'X'.

- **Caracteres alfabéticos (A, B, C, . . . , Z) (a, b, c, . . . , z).**
- **Caracteres numéricos (0,1, 2, . . . , 9).**
- **Caracteres especiales (+, -, *, /, ^, . . . ;, <, >, \$, . . .).**

Ejemplo

En el ejemplo de la figura 3.15 se presenta un algoritmo que declara una variable de tipo carácter, a la cual se le puede asignar una letra que represente el estado civil, como, por ejemplo: C para casado, S para soltero, V para viudo y U para unión libre. Luego, el valor asignado se imprime en pantalla.

```

    /* Opciones carácter de estado civil
    C = casado
    S = soltero
    V = viudo
    U = union libre */

    caracter estado_civil

    inicio
        estado_civil <- 'C'
        escriba "mi estado civil es ", estado_civil
    fin

```

Declaración de variables tipo carácter

Asignación con comilla simple

Figura 3-15 Tipo de datos carácter
Fuente propia

3.12.2 Representación interna del código ASCII

El estándar ASCII facilitó la estandarización en la representación de texto en dispositivos electrónicos, permitiendo la interoperabilidad entre diferentes sistemas informáticos, de esta manera es la base para muchos otros sistemas de codificación, como UTF-8 y ISO-8859.

ASCII (American Standard Code for Information Interchange):

Es un sistema de codificación que asigna valores numéricos únicos a caracteres utilizados en el inglés, incluyendo letras, números, signos de puntuación y ciertos caracteres de control. Fue desarrollado en 1963 y se utiliza ampliamente en sistemas informáticos para representar texto. [1]

Un carácter es un número binario de 7 bits (entre 0 y 127). Ejemplo interno de los caracteres, a continuación, se muestra la representación interna del código ASCII.

Caracteres especiales		
32	Espacio	10 0000
33	!	10 0001
34	"	10 0010
35	#	10 0011
36	\$	10 0100
37	%	10 0101

38	&	10 0110
39	'	10 0111
40	(10 1000
41)	10 1001
42	*	10 1010
43	+	10 1011
44	,	10 1100
45	-	10 1101
46	.	10 1110
47	/	10 1111

Caracteres numéricos		
48	0	11 0000
49	1	11 0001
50	2	11 0010
51	3	11 0011
52	4	11 0100
53	5	11 0101
54	6	11 0110
55	7	11 0111
56	8	11 1000
57	9	11 1001

Tabla 3.17 **ASCCI** Caracteres especiales y números.
Fuente propia

Caracteres Alfabéticos					
65	A	100 0001	97	a	110 0001
66	B	100 0010	98	b	110 0010
67	C	100 0011	99	c	110 0011
68	D	100 0100	100	d	110 0100
69	E	100 0101	101	e	110 0101
70	F	100 0110	102	f	110 0110
71	G	100 0111	103	g	110 0111
72	H	100 1000	104	h	110 1000
73	I	100 1001	105	i	110 1001
74	J	100 1010	106	j	110 1010
75	K	100 1011	107	k	110 1011
76	L	100 1100	108	l	110 1100

77	M	100 1101	109	m	110 1101
78	N	100 1110	110	n	110 1110
79	O	100 1111	111	o	110 1111
80	P	101 0000	112	p	111 0000
81	Q	101 0001	113	q	111 0001
82	R	101 0010	114	r	111 0010
83	S	101 0011	115	s	111 0011
84	T	101 0100	116	t	111 0100
85	U	101 0101	117	u	111 0101
86	V	101 0110	118	v	111 0110
87	W	101 0111	119	w	111 0111
88	X	101 1000	120	x	111 1000
89	Y	101 1001	121	y	111 1001
90	Z	101 1010	122	z	111 1010

Tabla 3.18 Código ASCII, Caracteres alfabéticos

Fuente propia

3.12.3 Tipo de dato cadena

El tipo de dato cadena de caracteres se forman agrupando varias caracteres [10], es una secuencia o seguidilla de caracteres que se encuentran delimitados por doble "comilla", pueden tener una longitud de caracteres según el tipo de lenguaje de programación.

Longitud de cadena

Al declarar la variable se debe especificar la longitud en medio de los símbolos [X], el numero de la longitud se especifica según el número de caracteres.

Ejemplo

nombre<- "Soyla Rosa Espinoza"	Estudios <- "Psicología Forestal"
--------------------------------	-----------------------------------

Ambas variables con longitud 19

Sintaxis

En la figura siguiente presenta el pseudocódigo que declara una variable nombre de tipo cadena de longitud de 15 caracteres y pide al usuario que ingrese su nombre. Luego, lee el texto ingresado y lo almacena en la variable nombre. Finalmente, muestra un mensaje para imprimir el texto inmediatamente la variable **nombre**.

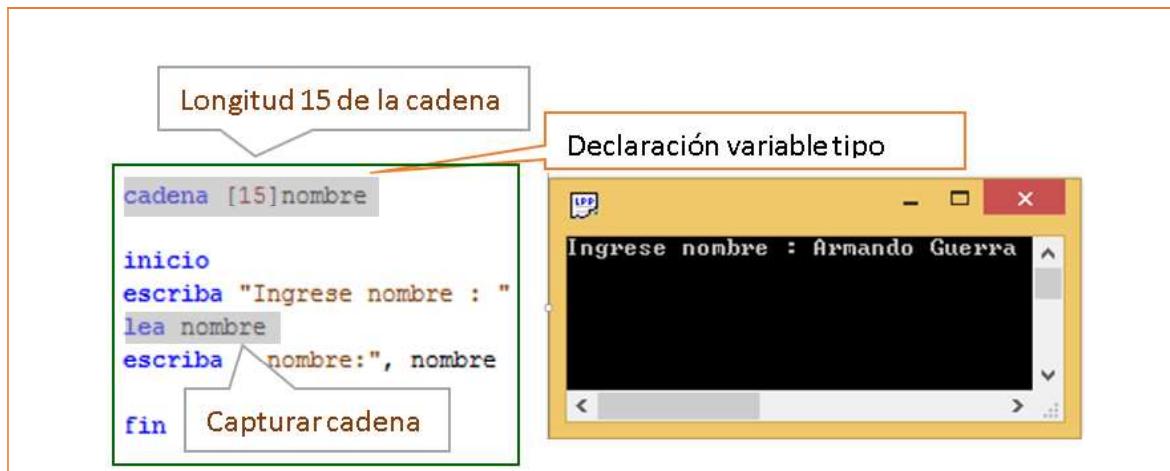


Figura 3-16 Tipo cadena

Fuente Propia

3.13 Tipos de datos compuestos

Los datos estructurados son compuestos de datos simples creados por el programador [10].

Ejemplo

registro Estudiante

cadena[50] nombre

edad entero

caracter sexo

fin registro

3.13.1 Asignación de datos**Asignación:**

En LPP la asignación de datos en una variable, se refiere al proceso de asociar un valor o dato a una variable y se hace posible por el operador (<-), que sirve para almacenar un valor en una variable dada.

Ejemplo:

Nombre <- "Elena Nito del Bosque "	edad <- 25
--	----------------------

Tabla 2.17 Asignación

Es necesario saber que una variable solo puede tener un único contenido en el momento dado, al que llamamos valor actual. Pero este contenido es de un solo tipo de dato y este valor puede cambiar en el tiempo, por lo tanto, los anteriores valores no quedan registrados en el sistema.

La asignación de contenido se hará escribiendo el nombre del identificador de la variable seguido de un signo <- y del tipo de contenido a asignar.

Ejemplo:

```

cadena [30] nombre
entero edad, cantidad_horas, precioHora
real salario

inicio
    nombre <- "SOYLA ROSA ESPINOZA"
    edad <- 27
    cantidad_horas <- 80
    precioHora <- 12000

    salario <- cantidad_horas * precioHora

    escriba "LA DAMA : ", nombre
    escriba " TIENE UN SALARIO DE: ", salario
fin

```

Figura 3-17 Asignaciones

Fuente propia

En el ejercicio de la Figura 3.17, tiene una declaración de cinco variables. El identificador nombre de tipo cadena de [30] caracteres, edad, cantidad_horas y precio_hora de tipo entero y por último se declaró la variable salario de tipo real.

Después del inicio de pseudocódigo, se le asignó a la variable nombre, el dato de la persona, (nombre <- " SOYLA ROSA ESPINOZA"), después a edad se le asignó los 27 años, a la cantidad de horas se le asignó 80 horas, al precio de hora se le asignó 12000 y por último a la variable salario se le asignó la cantidad de horas * el precio de la hora, y todas estas asignaciones se realizaron por medio del operador (<-), dando como resultado el sueldo del empleado.

Asignaciones del ejemplo

edad	<- 27
cantidad_horas	<- 80
precio_hora	<- 12000
salario	<- cantidad_horas * precio_hora

Constantes

Constantes en LPP:

La constante es un valor que se define al inicio del programa y ese dato no cambia durante la ejecución de un programa, es decir que reciben un valor en el momento de la compilación y este permanece invariable durante todo el programa.

Recomendación usar la nomenclatura **UPPER SNAKE_CASE** para constantes ejemplo MAX_INTENTOS, PI, IVA_COLOMBIA

En LPP las constantes son un caso particular de las variables, con la diferencia de que su nombre suele escribirse o declararse todo en mayúscula. Una vez declaradas las constantes pueden ser usadas en cualquier parte del programa. A modo de ejemplo podemos considerar el programa de la Figura 3.18.

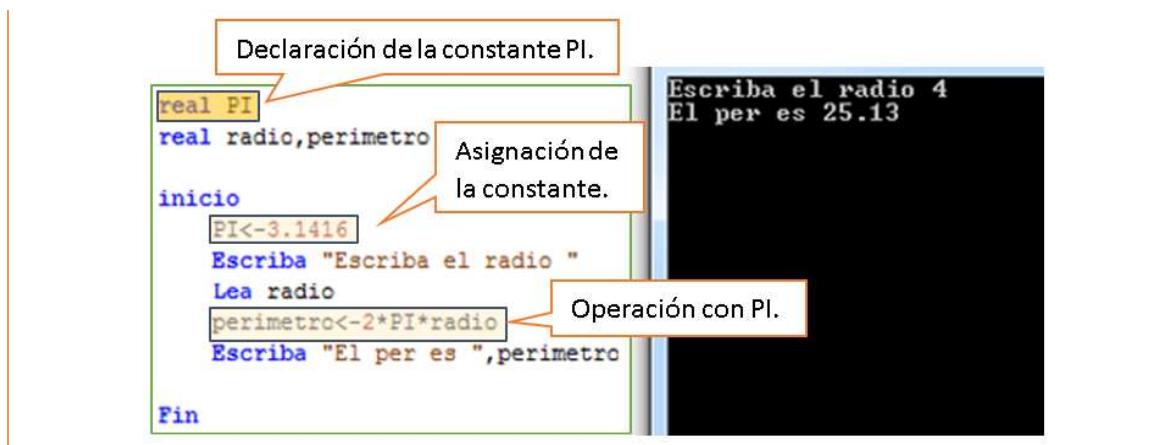


Figura 3-18 Constantes.

Fuente propia

En la Figura 3.18 se presenta un programa que permite el cálculo del perímetro de una circunferencia. En este programa se declara una constante de tipo real llamada PI, cuyo valor es asignado en el cuerpo del programa. Esta constante no cambia su valor durante el programa y es utilizada para calcular el perímetro de una circunferencia.

3.14 Operadores Matemáticos

Operador:

Un operador es un símbolo que representa una acción determinada para una variable o valor y así como en español la coma (,) indica una pausa corta y el punto (.), indica una pausa un poco más larga, en aritmética existen operadores que indican la ejecución de un determinado proceso [1].

Ejemplo:

$$3 + 4 = 7$$

En el ejemplo, hay dos operadores es el signo más (+) y el signo igual (=) que indica que se debe hacer una suma y obtener el resultado.

En la programación estos símbolos se dividen en operadores aritméticos que a su vez se fraccionan en binarios y unarios, los operadores relacionales y los operadores Lógicos.

Nota: (en LPP no soporta los operadores unarios ejemplo: n++, p--).

3.14.1 Operadores aritméticos

Binarios:

Los operadores aritméticos son binarios, es decir que se necesitan de un operador matemático binario y dos operandos (numero1, numero2), ejemplo: numero1 + numero2 [12], Estos operadores representan operaciones y son los habituales para una calculadora. Como lo son: suma (+), resta (-), multiplicación (*) y división (/).

Estos son los operadores binarios

Operadores aritmético	
Operador	concepto
+	Operador de suma
-	Operador de Resta
*	Operador de multiplicación
/	Operador de división

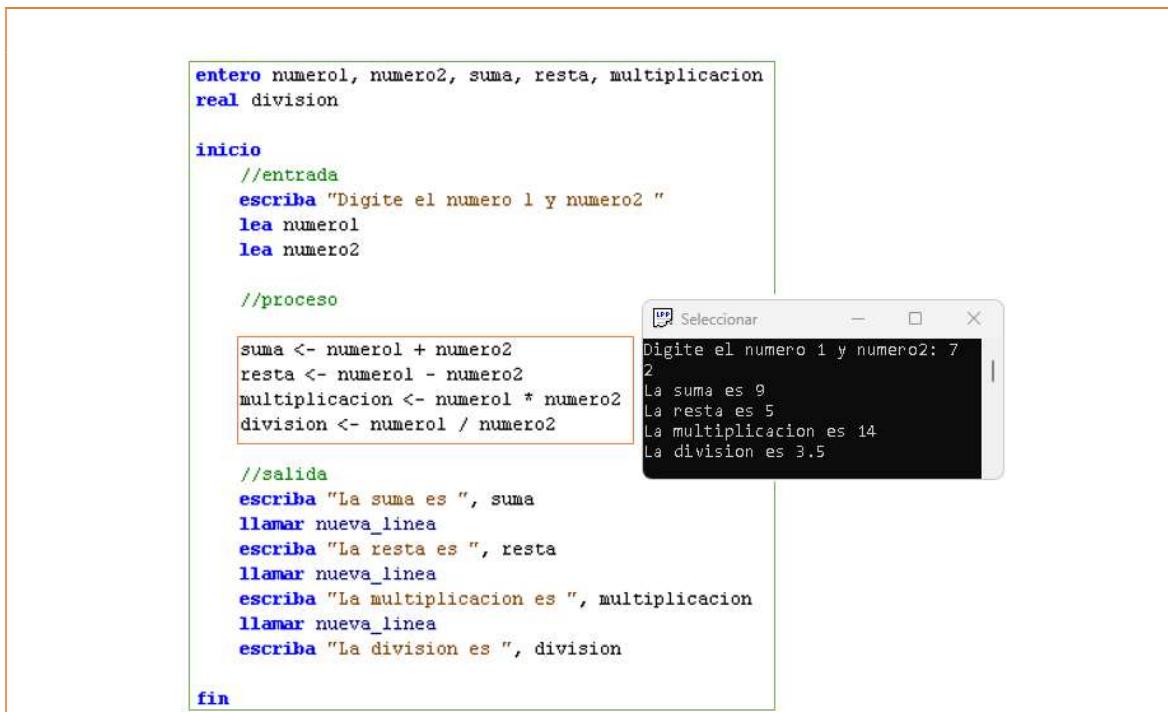
Tabla 3.19 Operadores matemáticos
Fuente propia

Ejemplo

$$\begin{aligned}3 + 7 &= 10 \\4 - 1 &= 3 \\2 * 5 &= 10 \\7 / 2 &= 3\end{aligned}$$

En la figura 3.20 presenta un algoritmo que utiliza los operadores aritméticos de suma (+), resta (-), multiplicación (*) y división (/) para realizar operaciones con dos números enteros ingresados por el usuario. Declara variables para almacenar los

resultados de cada operación, utilizando tipos de datos enteros para suma, resta y multiplicación, y real para la división.



```

entero numero1, numero2, suma, resta, multiplicacion
real division

inicio
    //entrada
    escriba "Digite el numero 1 y numero2 "
    lea numero1
    lea numero2

    //proceso
    suma <- numero1 + numero2
    resta <- numero1 - numero2
    multiplicacion <- numero1 * numero2
    division <- numero1 / numero2

    //salida
    escriba "La suma es ", suma
    llamar nueva_linea
    escriba "La resta es ", resta
    llamar nueva_linea
    escriba "La multiplicacion es ", multiplicacion
    llamar nueva_linea
    escriba "La division es ", division

fin

```

The screenshot shows a window titled "Seleccionar" with the following text:
 Dígite el numero 1 y numero2: 7
 2
 La suma es 9
 La resta es 5
 La multiplicacion es 14
 La division es 3.5

Figura 3-19 Código con operadores
Fuente propia

Operadores matemáticos especiales	
Operador	concepto
mod	Operador de cálculo de residuo
div	Operador de división entera
()	Agrupar expresiones
^	Operador para exponentiación

Tabla 3.20 Operadores matemáticos especiales
Fuente propia

3.14.2 Operador mod

MOD

El operador de cálculo de residuo "mod", devuelve el residuo de una división entre 2 números. El operador mod se aplica solamente a variables de tipo enteros [12], mod es la abreviatura de "módulo".

En la figura 3.20 División con mod presenta la división 5 entre 2, pero utilizando la operación módulo (mod), **5 mod 2** el cual entrega el residuo de la división. Al dividir los dos números $5 \div 2$ da como resultado 1 que es el residuo de color naranja.

Ejemplo:



Figura 3-20 División con mod
Fuente propia

Ejemplo

5 mod 2 = 1
4 mod 1 = 0
7 mod 5 = 2
7 mod 2 = 1

Ejemplo en LPP

En la figura 3.21 representa el un algoritmo que realiza una división entre dos números enteros numero1 y numero2 y muestra tanto el resultado de la división como el residuo. Primero, se declaran las variables numero1 y numero2, asignándoles los valores 5 y 2, respectivamente. Luego, se calcula el residuo de la división utilizando el operador mod que devuelve el resto o módulo de la división y se almacena en la variable residuo. Finalmente, el algoritmo imprime en pantalla los

resultados de "Al dividir 5 / 2", el resultado de la división cociente con decimales y el residuo.

```

entero numerol, numero2
entero residuo
inicio
    //entrada
    numerol <- 5
    numero2 <- 2
    //proceso

    residuo <- numerol mod numero2

    //salida
    escriba "Al dividir ", numerol, " / ", numero2
    llamar nueva_linea
    escriba "La division es: ", (numerol / numero2)
    llamar nueva_linea
    escriba "El residuo es: ", residuo
fin

```

Figura 3-21 División con mod en LPP

Fuente propia

3.15 Operador div

El operador de división entera div, se utiliza para obtener la división entera entre dos números enteros [13].

En la figura 3.22 **División con div** presenta la división 5 entre 2, utilizando la operación **5 div 2**, el cual entrega la parte entera del cociente de la división. Al dividir los dos números $5 \div 2$ da como resultado la parte entera: 2 que es el cociente de color rojo.



Figura 3-22 División con div

Fuente propia

Ejemplo

```
5 div 2 = 2
3 div 2 = 1
7 div 5 = 1
7 div 2 = 3
```

Ejemplo en LPP

En la figura 3.23 el código presenta una división entera entre dos números, 5 y 2, y almacena el resultado en la variable resultado. La operación se realiza utilizando el operador div, que devuelve el cociente de la división entera, ignorando el resto o residuo. Por lo tanto, al dividir 5 entre 2, el resultado es 2, ya que la parte decimal se descarta. Finalmente, el programa muestra en pantalla el resultado de la división entera, que es 2.

The screenshot shows a programming interface with a code editor and a terminal window. The code editor contains the following pseudocode:

```

entero resultado
inicio
    resultado <- 5 div 2
    escriba "La respuesta es: ", resultado
fin

```

A callout bubble points to the line `resultado <- 5 div 2` with the text "La parte entera de división 5 / 2 es 2". The terminal window displays the output: "La respuesta es: 2".

Figura 3-23 División entera

Fuente propia

3.15.1 Operador ^

El operador para exponenciación: Se define como la operación matemática en la que un número base se eleva a una potencia [12], esta operación devuelve el resultado de una exponenciación.

La exponenciación representa mediante un operador `^` o una función que indica cuántas veces se debe multiplicar la base por sí misma según el exponente

Ejemplo:

```
4 ^ 1 = 4
2 ^ 3 = 8
3 ^ 3 = 27
```

3.15.2 Operador de agrupar expresiones ():

Este operador permite agrupar expresiones para una mejor comprensión y desarrollo correcto de la operación.

Los paréntesis () de agrupación de expresiones matemáticas es una herramienta utilizada en programación, matemáticas y lógica para establecer prioridades y organizar cálculos. Su función principal es controlar el orden en el que se evalúan las operaciones en una expresión.

Ejemplo:

```
(3 + 2) = 5  
(7 - 2) div 2 = 2  
(7 div ( 2 + 3 )) = 1
```

Ejemplo en LPP

En la figura 3.24 el código presenta una división entera entre dos números, 5 y 2, y almacena el resultado en la variable resultado. La operación se realiza utilizando el operador div, que devuelve el cociente de la división entera, ignorando el resto o residuo. Por lo tanto, al dividir 5 entre 2, el resultado es 2, ya que la parte decimal se descarta. Finalmente, el programa muestra en pantalla el resultado de la división entera, que es 2.

```

entero resultado
inicio

    resultado <- 10 div ( 2 ^ 3 )

    escriba "El resultado de la operacion es: ", resultado
fin

```

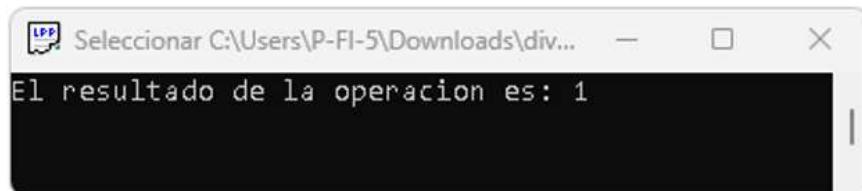


Figura 3-24 Agrupación con paréntesis
Fuente propia

3.16 Operadores relacionales

Operadores relacionales

Los operadores relacionales permiten determinar si se cumple o no una determinada condición o comparación. En caso de cumplirse la condición producen como resultado un valor “Verdadero”, de lo contrario el resultado será un valor “Falso” [2].

En LPP existen 6 tipos de operadores y se presentan a continuación:

Operadores relacionales			
Operador	Descripción	Ejemplo	Resultado
=	Igual a	5 = 5	Verdadero
<>	Distinto de	3 <> 3	Falso
<	Menor que	3 < 5	Verdadero
>	Mayor que	1 > 3	Falso
<=	Menor o igual que	3 <= 3	Verdadero
>=	Mayor o igual que	5 >= 5	Verdadero

Tabla 3.21 Operadores relacionales
Fuente propia

3.17 Operadores lógicos

Operadores lógicos:

Los operadores lógicos son símbolos o palabras que se utilizan en programación y matemáticas para trabajar con valores booleanos verdaderos (True) o falsos [14]. En LPP existen dos tipos de operadores lógicos: "y", "o". Estos valores arrojan como resultado de la operación un valor de verdad verdadero o falso.

Operadores lógicos	
Operador	Operador
O	y

Tabla 3.22 Operadores lógicos

Fuente propia

Ejemplo:

- $(3 > 2)$ y $(2 > 2)$ => falso
 $(3 > 2)$ es verdadero y $(2 > 2)$ falso, por lo tanto $(V \text{ y } F)$ es falso
- $(7 > 1)$ o $(2 = 5)$ => $(V \text{ o } F)$ verdadero
- $(7 = 7)$ o $(5 <> 5)$ => $(V \text{ o } F)$ verdadero

3.18 Jerarquía de operadores

Jerarquía

La jerarquía de operadores es un conjunto de reglas que determina el orden en que se interpretan los operadores en una expresión matemática en la computadora las operaciones a realizar, determinando cuales tienen mayor prioridad entre ellas.

Al evaluar expresiones con operadores aritméticos debemos respetar la jerarquía en el orden de aplicación, si hay varios operadores en una expresión, se debe realizar el operador de mayor jerarquía[15].

Cuando varios operadores tienen la misma prioridad, se evalúan de izquierda a derecha (asociatividad).

Estas reglas tienen una jerarquía y aseguran que las operaciones se realicen de manera consistente cuando se combinan varios operadores en una misma expresión.

En la tabla 3.23 se presenta la jerarquía de los operadores en LPP. En esta tabulación se puede apreciar que el operador que presenta la más alta jerarquía es el de paréntesis, seguido del operador de exponenciación. En la jerarquía los operadores con mayor prioridad se ejecutan antes que aquellos con menor prioridad.

JERARQUÍA DE OPERADORES	
OPERADOR	CONCEPTO
()	Paréntesis internos
()	Paréntesis
^	Operador para exponenciación
mod , div	Operador de cálculo de residuo
* , /	Operador de multiplicación tiene la misma jerarquía que el operador de división
+ , -	El operador de suma tiene la misma jerarquía que el operador de resta

Tabla 3.23 Jerarquía de operadores

Fuente propia

Ejemplo

$$3 + 1 + ((4 \text{ mod } 2) * 700) + 1$$

$$3 + 1 + (0 * 700) + 1$$

$$3 + 1 + 0 + 1$$

$$4 + 1$$

$$4 + 1$$

$$5$$

- 8 div 4 mod 1 rta 0
- n<- 8 div 4 mod 1 rta 0

- **8 div 2 mod 3** rta 1
- **3 + 2 ^ 3** rta 11
- **3 * 2 mod 2** rta 0
- **3 - 2 + 3** rta 4

```
x <- 3 + (4 - 1 + 7 mod 2 + (5 + 1) - 4 MOD 2) + (3 ^ 2) / 3 ) / 2
```

La respuesta es: **x <- 9.5**

3.18.1 Paréntesis internos prioridad 1

La primera prioridad jerárquica corresponde a las operaciones dentro de paréntesis. Esto significa que, al evaluar una expresión como en la figura 25, siempre se resuelven primero las operaciones del paréntesis más interno.

En esta figura 3.25 presenta un algoritmo con la prioridad de los paréntesis internos y luego las operaciones básicas. Comienza resolviendo la expresión dentro del paréntesis más interno, donde calcula $4 \text{ div } 2$ (división entera), que resulta en 2, de la siguiente manera.

```
Paso 1 calculo <- 5 + ( 3 + ( 4 div 2 ) )
Paso 2 calculo <- 5 + ( 3 +2 )
Paso 3 calculo <- 5 + 5
Paso 4 calculo <- 10
```

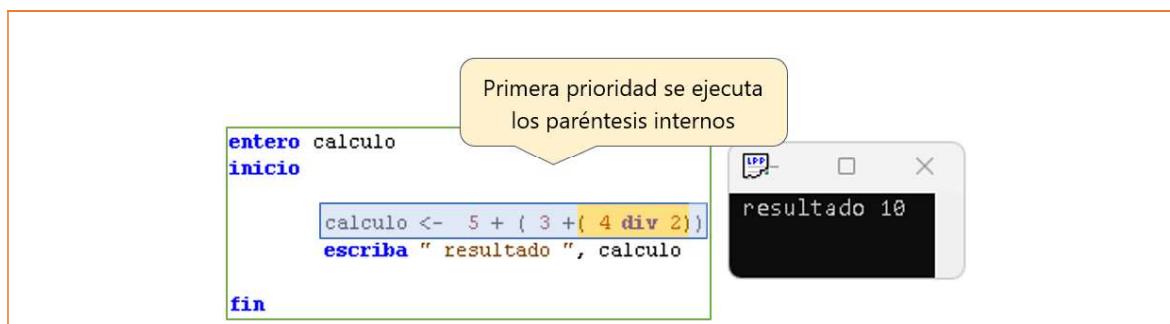


Figura 3-25 Paréntesis internos
Fuente propia

3.18.2 Paréntesis prioridad 2

En algoritmos y expresiones matemáticas la mayor prioridad de jerarquía de evaluación la tiene los paréntesis. Esto significa que cualquier operación que esté dentro de un paréntesis se primero que las operaciones fuera de ellos.

Ejemplo

En la figura 3.26 presenta un código con una operación aritmética con paréntesis. Primero, evalúa la expresión dentro del paréntesis ($4 \text{ div } 2$), donde div representa la división entera, resultando en 2. Y luego realiza las operaciones externas, de la siguiente manera.

Paso 1. Primero se resuelve el paréntesis: `calculo <- 3 + (4 div 2)`.

Paso 2. Luego se resuelve la operación externa: `calculo <- 3 + 2`.

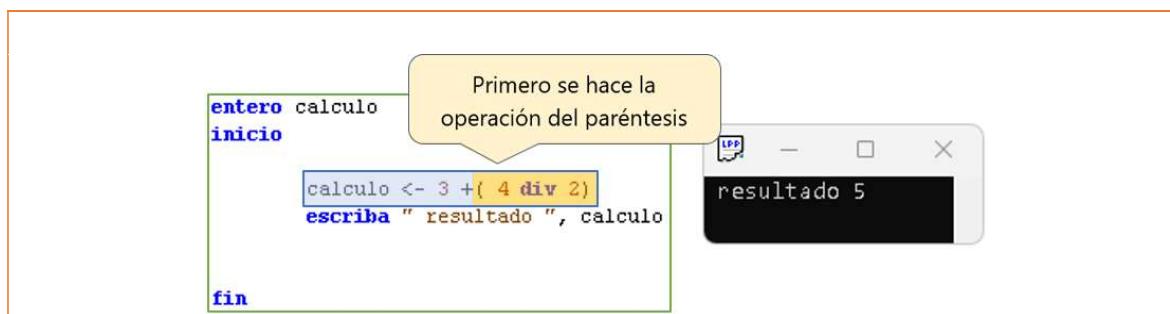


Figura 3-26 Prioridad paréntesis

Fuente propia

3.18.3 Exponenciación prioridad 3

La exponenciación tiene una prioridad mayor que residuo mod, la division entera div, la multiplicación, división, suma y resta. La exponenciación se resuelve antes que las demás, a menos que esté dentro de paréntesis, ya que los paréntesis tienen la máxima prioridad.

Ejemplo

En la figura 3.27 presenta la jerarquía de operadores, donde la exponenciación ($^$) tiene la mayor prioridad y se evalúa primero. Por lo tanto, se calcula $3 ^ 2$, que dando de la siguiente forma.

Paso1. calculo <- $3^2 - 5$

Paso2. calculo <- $9 - 5$

Paso3. calculo <- 4

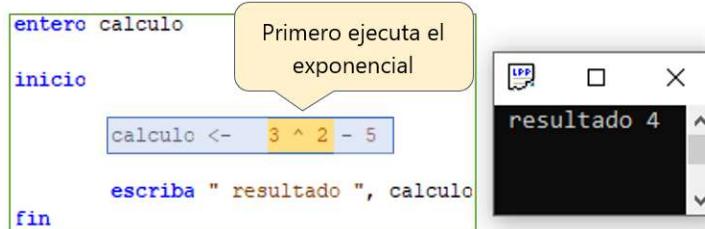


Figura 3-27 Prioridad exponenciación

Fuente propia

3.18.4 Mod y Div prioridad 4

Mod el residuo y div división entera tienen la misma prioridad en la mayoría de los lenguajes de programación. Esto significa que cuando ambas operaciones aparecen en una expresión sin paréntesis, se evalúan de izquierda a derecha.

Ambas tienen una prioridad mayor que la suma y la resta, pero menor que la exponenciación y los paréntesis.

Ejemplo

En la figura 3.28 presenta la jerarquía de operadores, donde la el **mod y div** tiene la mayor prioridad y se evalúan primero, pero de izquierda a derecha por que tienen la misma prioridad. Por lo tanto se calcula de la siguiente forma.

Paso 1. calculo <- 2 + 3 mod 2 + 3 div 2

Paso 2. calculo <- 2 + 1 + 3 div 2

Paso 3. calculo <- 2 + 1 + 1

Paso 4. calculo <- 2 + 1 + 1

Paso 5. calculo <- 3 + 1

Paso 6. calculo <- 4

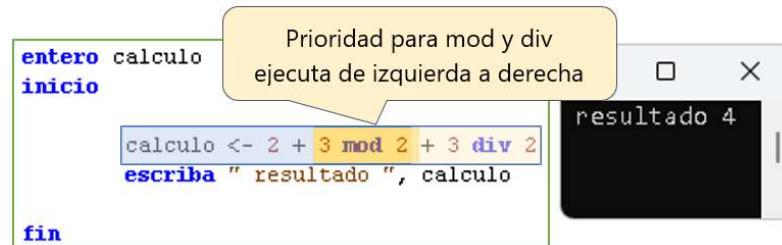


Figura 3-28 Prioridad mod div
Fuente propia

3.18.5 Multiplicación (*) y división (/) prioridad 5

La multiplicación (*) y la división (/) tienen la misma prioridad. Ambas operaciones tienen una prioridad mayor que la suma y la resta, pero menor las demás. Si hay varias multiplicaciones y divisiones en la misma expresión, se resuelven de izquierda a derecha.

Ejemplo

En la figura 3.29 presenta la jerarquía de operadores, donde la multiplicación (*) tiene la mayor prioridad que la suma y resta. Por lo tanto se calcula de la siguiente forma.

- Paso 1.** calculo <- 5 - 3 + 2 * 3
- Paso 2.** calculo <- 5 - 3 + 6
- Paso 3.** calculo <- 5 - 3 + 6
- Paso 4.** calculo <- 2 + 6
- Paso 5.** calculo <- 8

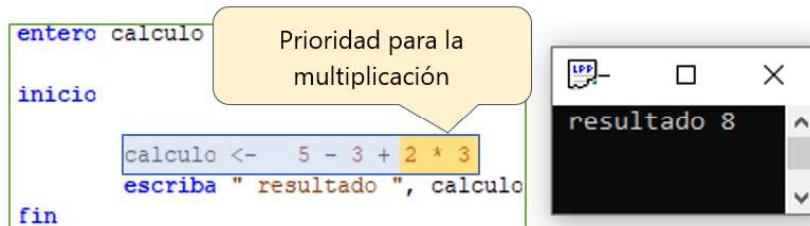


Figura 3-29 Prioridad multiplicación división
Fuente propia

3.18.6 Suma (+) y resta (-) prioridad 6

La suma (+) y la resta (-) tienen la misma prioridad, pero esta es la de menor prioridad. Si hay varias sumas y restas en la misma expresión, se resuelven de izquierda a derecha.

Ejemplo

En la figura 3.30 presenta la jerarquía de operadores, donde la la suma (+) y la resta tienen la menor prioridad y como hay sumas y restas se evalua de izquierda a derecha. Por lo tanto se calcula de la siguiente forma.

Paso 1. calculo <- **2 + 3 - 5**

Paso 2. calculo <- **5 - 5**

Paso 3. calculo <- **0**

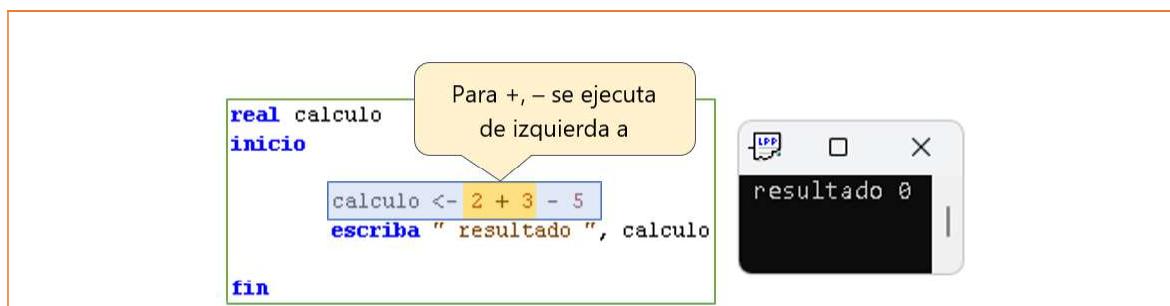


Figura 3-30 Prioridad suma y resta
Fuente propia

3.18.7 Operaciones con radicales en LPP

Concepto

En LPP no existe directamente la opción de realizar operaciones con raíces, sin embargo, es posible aplicar la propiedad matemática que dice que toda raíz puede ser expresada en términos de una potencia. De este modo la raíz enésima de un número es equivalente a elevar dicho número a la $1/n$.

Ejemplo

$\sqrt[2]{3}$ es equivalente a $3^{(1/2)}$

$\sqrt[3]{x}$ es equivalente a $x^{(1/3)}$

$\sqrt[5]{m}$ es equivalente a $m^{(1/5)}$

$\sqrt[5]{x+1}$ es equivalente a $(x+1)^{(1/5)}$

Lo anterior, permite concluir que para realizar cualquier operación relacionada con raíces de cualquier orden en LPP, dicha operación puede ser expresada en términos una potencia, como se evidencia en el ejemplo siguiente Figura 3.31.

Ejemplo

En la figura 3.31 presenta la operación con la raíz enésima de un número, El código presenta como se realiza la raíz cuadrada de 4 y raíz cubica de 49. De esta manera se calcula de la siguiente forma.

Paso 1. $\sqrt[2]{4}$ es equivalente a $= 4^{(1/2)}$

Paso 2. $\sqrt[3]{49}$ es equivalente a $= 49^{(1/3)}$

```
real raiz
inicio
    raiz <- 4 ^{1/2}
    escriba "La raiz es ", raiz
    //Hallar la raiz cubica de 49
    llamar nueva_linea
    raiz <- 49 ^{1/3}
    escriba "La raiz es ", raiz
fin
```

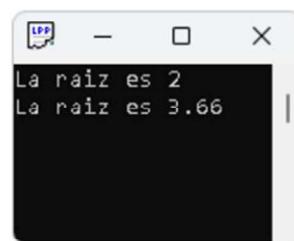


Figura 3-31 Operación con la raíz enésima

Fuente propia

Ejercicios de la unidad

Tipos de datos en LPP

- Escribe un pseudocódigo que lea el nombre y el tipo de usuario y luego imprima de esta forma.

Ejemplo de entrada: Rita, cliente → Salida: "Bienvenido cliente Rita"

- Dados los 3 lados de un triángulo determinar en LPP su área usando el teorema de Herón.

Formula del semiperímetro:

$$s = \frac{a + b + c}{2}$$

Formula del Area:

$$A = \sqrt{s(s - a)(s - b)(s - c)}$$

- Realizar un programa en LPP que permita calcular el volumen y el área de una esfera maciza.

$$A=4\pi r^2$$

$$V=(4/3)\pi r^3$$

- Realizar un seudocódigo en LPP que permita convertir grados Centígrados a Fahrenheit.

Fórmula:

$$\text{Grados Fahrenheit} = (\text{grados centígrados} \times 9/5) + 32.$$

- Realiza el algoritmo que calcule el promedio de tres números ingresados por el usuario.

- Escribe un seudocódigo que capture el precio de un producto y cantidad luego calcule el precio total incluyendo el IVA.

- Realiza un pseudocódigo que lea una cantidad de horas y las convierta en minutos.

$$\text{Fórmula: Minutos} = \text{Horas} \times 60$$

- Realiza el algoritmo que lea el año de nacimiento de una persona y calcule su edad actual donde el año actual es una constante.
- Realiza un pseudocódigo que lea una cantidad en dólares y la convierta a pesos colombianos.
- Crea un pseudocódigo que convierta una cantidad en kilogramos a libras.
Fórmula: Libras = Kilogramos × 2.20462
- Escribe un programa que lea las horas trabajadas en una semana y calcule el salario por semana y el mes.

Bibliografía

4 CAPÍTULO ESTRUCTURAS DE CONTROL

Contenido estructuras de control de selección

4.1 Temas del capítulo

- Concepto estructuras de control de selección
- Estructuras de selección simple
- Operadores relacionales
- Conectores Lógicos
- Lógica Proposicional
- Estructuras de selección compuestas
- Estructuras de selección dobles
- Estructuras de selección anidadas
- Estructuras de selección múltiples
- Ejercicios de la unidad

En los lenguajes de programación existen diferentes estructuras de control que especifican los procesos necesarios y la capacidad de tomar decisiones basadas en condiciones guiando correctamente la funcionalidad del programa. Las estructuras de control de selección permiten a los desarrolladores dirigir el flujo de ejecución de un programa dependiendo de una o varias condiciones específicas y adaptarse a diferentes escenarios que plantean los requisitos de un problema.

En el capítulo 4, abordamos los conceptos relacionados con las estructuras de control de selección, fundamentales para permitir que los programas respondan a distintas condiciones y ejecuten bloques de código específicos según corresponda, en consecuencia, se presentaran las estructuras de selección simple, que son las condiciones más básicas que permiten ejecutar un conjunto de instrucciones, usando los conectores lógicos, para combinar múltiples condiciones, ampliando así la complejidad y flexibilidad de nuestras decisiones programáticas. Esto llevará a las estructuras de selección compuestas, dobles y múltiples, donde aprenderemos a manejar situaciones más complejas mediante la inclusión de múltiples condiciones, lo que te permitirá desarrollar aplicaciones más robustas y adaptativas a cualquier problema real.

4.2 Concepto estructuras de control de selección



Figura 4-1 Control de selección
Fuente propia

4.2.1 Flujo de control

Concepto

En el seudocódigo el flujo de control define el orden de ejecución en el que se ejecutan las distintas instrucciones de un algoritmo. Como también es el mecanismo que determina cómo se ejecutan las sentencias, permitiendo alterar el camino tradicional de ejecución lineal.

En el seudocódigo el flujo de control define el orden de ejecución en el que se ejecutan las distintas instrucciones de un algoritmo. Como también es el mecanismo que determina cómo se ejecutan las sentencias, permitiendo alterar el camino tradicional de ejecución lineal.

Para que se usa:

Al desarrollar un algoritmo, es común enfrentarse a diversas condiciones necesarias para resolver un problema. Las estructuras de control condicionales son esenciales para incorporar lógica y adaptabilidad en el flujo del programa, permitiendo las siguientes alternativas

- **Controlar el flujo:** Define el orden y decide qué parte del código se ejecuta en función de condiciones específicas.

- **Permitir alternativas:** Proveer diferentes caminos para manejar situaciones diversas mediante opciones como "si", "si no" o "según".
- **Adaptabilidad:** Hacer que el programa responda dinámicamente a las entradas del usuario, datos procesados o eventos externos.
- Toma de Decisiones Simples
- Selección Múltiple
- Evaluar múltiples condiciones de manera secuencial
- Decisiones Anidadas
- Combinar condiciones complejas
- Evaluación de Rangos
- Validar intervalos de valores
- Clasificar información según criterios específicos
- Controlar situaciones imprevistas

Estructura de control secuencial en un algoritmo

Es aquella que permite decidir la ejecución de una acción o varias instrucciones del programa. En la cual evalúa la condición con los datos necesarios y si este resultado es verdadero realiza una determinada secuencia de instrucciones para dar solución a la elección positiva según [3].

Sintaxis

Si <condición> Entonces

```
< Instrucción si 1>
< Instrucción si 2>
< Instrucción si 3>
< Instrucción si ... >
```

Fin si

Sintaxis desde LPP

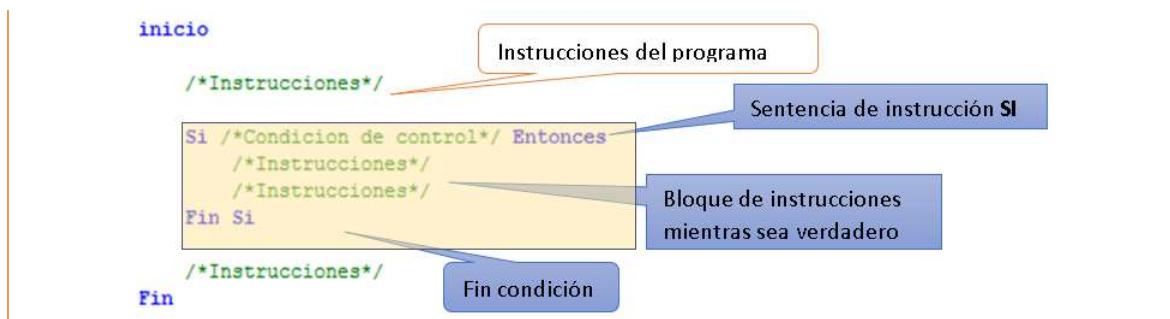


Figura 4-2 Estructuras de control

Fuente propia

4.2.2 Operadores relacionales

El operador relacional es un símbolo que se usa para establecer una relación. Para comparar entre dos o más expresiones (combinación de operadores y variables) donde el resultado es verdadero o falso según [4].

Entre los diferentes operadores lógicos tenemos:

Operadores relacionales
= igual a
<> diferente de
> mayor que
>= mayor igual que
<= menor igual que
< menor que

Tabla 4.1 Operadores relacionales

Fuente propia

4.3 Estructuras de selección simple

Se aplican para escoger una decisión lógica. Al evaluar la condición toma uno de dos valores verdadero o falso, si es verdadera la condición lógica ejecuta una serie de instrucciones y continua con el programa. Si es falsa continúa la ejecución normal como en la figura 3 realizada en un diagrama de flujo que ejecuta dos instrucciones si la condición es verdadera.

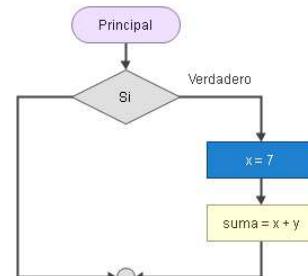


Figura 4-3 Condición simple
Fuente propia

Concepto:

Las estructuras de selección simple permiten que un programa ejecute un bloque de código únicamente si se cumple una condición determinada.

Sintaxis

Si VALOR 1 operador relacional VALOR 2 Entonces

```
< Instrucción 1>
< Instrucción 2>
< Instrucción 3>
< Instrucción X >
```

Fin si

Ejemplo:

Ejemplo figura 4.4 de un algoritmo en LPP que permite recibir un número desde el teclado y determina si el número es positivo.

```

real numero
/* declaro la variable y el tipo de dato */
inicio /*inicio del programa*/
    /*pide un numero desde teclado*/
    escriba "ingrese el numero :"
    lea numero /*captura el numero el programa*/
    /* pregunta si el numero es mayor que cero*/
    si numero > 0 entonces
        escriba " EL NUMERO DIGITADO ES POSITIVO "
    fin si
    /* Finaliza la condición */
fin /*fin del programa*/

```

Figura 4-4 Selección simple

Fuente propia

Descripción del ejercicio

El ejemplo anterior solicita un número ingresado por teclado para determinar si es positivo. A continuación, en la figura 4.5, se detalla el código explicado por medio de mensajes.

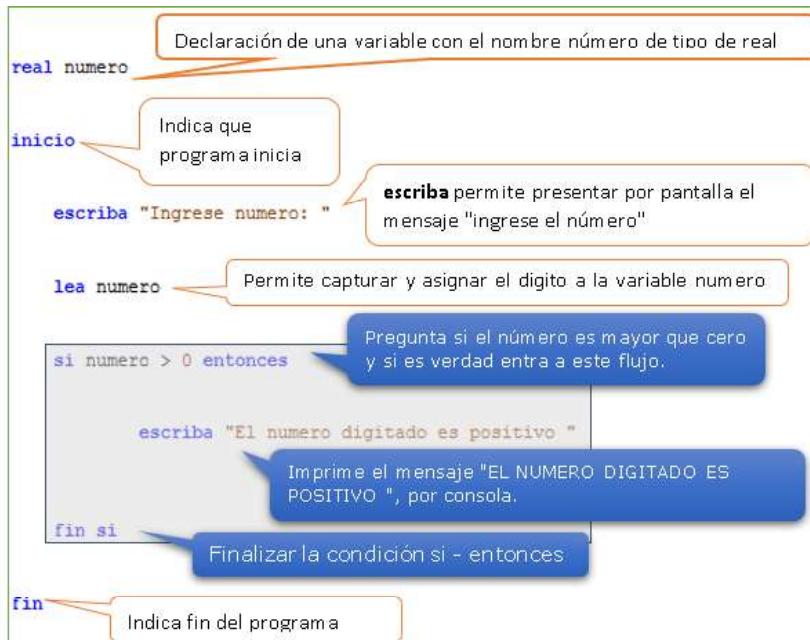


Figura 4-5 Código selección simple

Fuente propia

Ejemplo con booleanos, carácter y el conector (Y)

En la Figura 4.6 se presenta un código para evaluar únicamente si es estudiante con RH+. En seudocódigo se declaran dos variables: una de tipo booleano y otra de tipo carácter. La variable booleana solicita al usuario indicar si es estudiante o no, utilizando (0 para representar falso y 1 para representar verdadero). La variable de tipo carácter solicita al usuario ingresar su tipo de Rh, indicando si es positivo (+) o negativo (-).

Posteriormente, el programa evalúa si el usuario es estudiante y tiene Rh positivo. Si ambas condiciones se cumplen, muestra un mensaje indicando: "Usted es estudiante con Rh positivo".

```
// Declaración de variables
booleano esEstudiante
caracter rh
inicio
    // Entradas
    escriba "Digite las opciones "
    llamar nueva_linea
    escriba " 1 es estudiante "
    llamar nueva_linea
    escriba " 2 no es estudiante "
    lea esEstudiante
    escriba " RH (+) Positivo "
    llamar nueva_linea
    escriba " RH (-) negativo "
    lea rh
    //Condicion
    si esEstudiante y (rh ='+') entonces
        escriba "Es estudiante con RH+"
    fin si
fin
```

Figura 4-6 Ejemplo booleano y carácter
Fuente propia

Consideración

Cabe destacar que el lenguaje LPP no permite el uso directo de valores booleanos como verdadero o falso. En su lugar, restringe el uso a 0 (falso) y 1 (verdadero) para representar estos estados.

4.4 Estructuras de selección dobles



Figura 4-7 Selección doble
Fuente propia

Concepto

Estructuras de selección dobles permiten controlar casos, donde se evalúa dos alternativas posibles, por lo tanto, la estructura de selección doble debe ejecutar una de las dos opciones, pero solo una elección al tiempo.

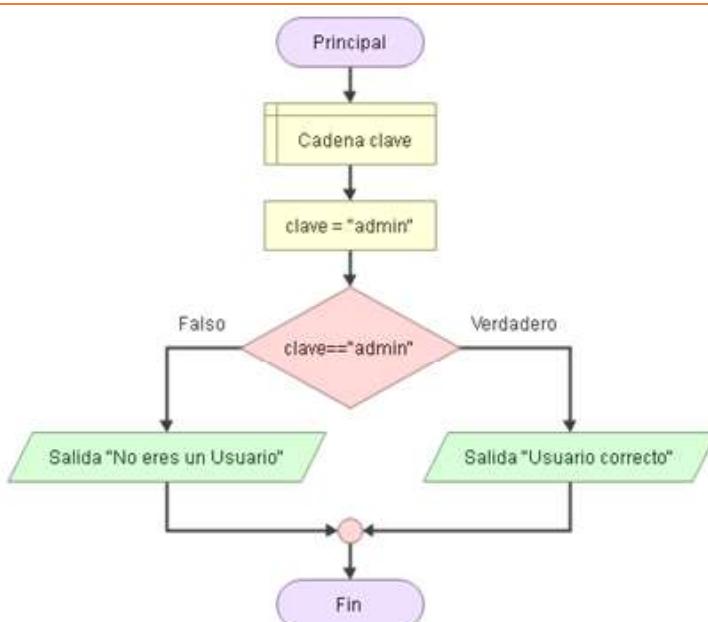


Figura 4-8 Diagrama de flujo Selección con doble
Fuente propia

En la figura 4.8 la condición doble permite ejecutar diferentes bloques de código dependiendo de si la condición es verdadera o falsa.

Sintaxis en LPP

Si (valor1 operador relacional valor2) entonces

Instrucción a1
Instrucción a2
Instrucción a3

Instrucción aN

Bloque de instrucciones
cuando es **verdad**

sino

Instrucción b1
Instrucción b2
Instrucción b3

Instrucción bN

Bloque de instrucciones
cuando es **falso**

fin si

La sintaxis evalúa una condición lógica entre el valor1 y valor2, si es verdadera ejecuta una o varias instrucciones (conjunto de instrucciones aN), si no ejecuta el otro conjunto de instrucciones bN.

Ejemplo

La figura 3.7 ilustra gráficamente un ejercicio que recibe la edad por teclado, la cual evalúa e imprime si es mayor de edad o no.

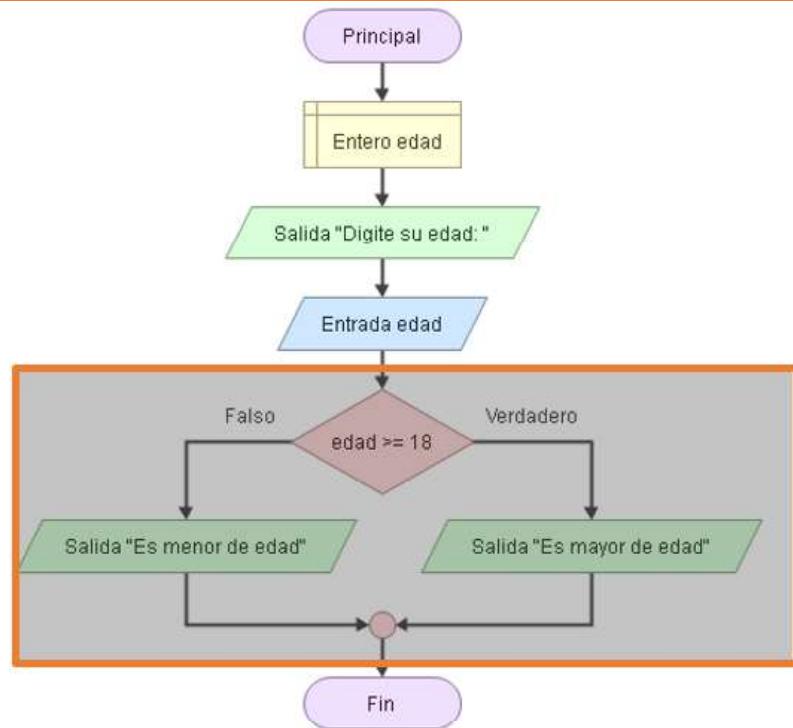


Figura 4-9 Diagrama de selección doble

Fuente propia

La Figura 4.10 Código de selección doble muestra el diagrama del ejemplo anterior implementado en LPP. En este código, se solicita al usuario que ingrese su edad por teclado. Luego, el programa evalúa si la edad ingresada corresponde a una persona mayor de edad y, en función de esto, imprime un mensaje indicando si es mayor de edad o no.

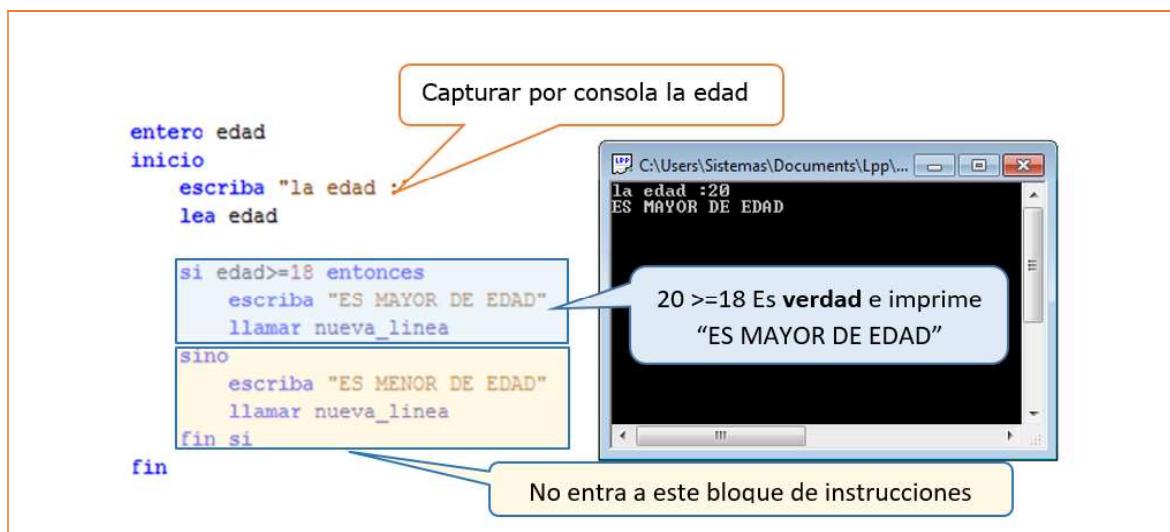


Figura 4-10 Código de selección doble

Fuente propia

Explicación

La expresión condicional se describe a continuación:

si (edad >= 18) entonces

Si la condición es verdadera (20 mayor o igual 18):

Se ejecuta la instrucción dentro del bloque de la opción verdadera, que en este caso es:

escriba "ES MAYOR DE EDAD"

Esto significa que, si la persona tiene 18 años o más, se mostrará el mensaje indicando que es mayor de edad.

Sino

Cuando es falsa ejemplo edad = 7

Si la condición es falsa (**7 es menor de 18**):

Se ejecuta el bloque de instrucciones dentro de sentencia sino, esto significa que la edad de la persona tiene menos de 18 años y se mostrará el mensaje indicando que es menor de edad.

escriba "ES MENOR DE EDAD"

fin si

Termina el condicional

4.5 Expresiones lógicas

Para la utilización de las estructuras de selección en diversos escenarios es necesario saber que se pueden usar diferentes expresiones lógicas, que facilitan la ejecución de tareas condicionales.

Conectores Lógicos

Son símbolos o letras que se utilizan para establecer relaciones entre ideas o proposiciones permitiendo construir expresiones lógicas.

Estos conectores son utilizados para conectar varias expresiones lógicas y se conocen con el nombre de conjunción, disyunción y negación.

Conectores lógicos

O

Y

NO

Tabla 4.2 Operadores lógicos

Fuente propia

4.6 Lógica Proposicional

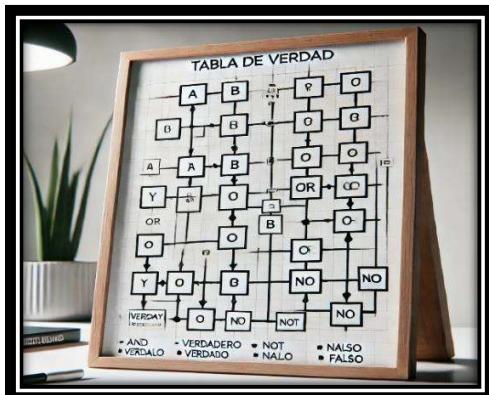


Figura 4-11 Lógica de proposicional
Fuente imagen [1]

4.6.1 La lógica Proposicional

Es una rama de la lógica matemática que se centra en el estudio de las proposiciones y sus conectivas lógicas, permitiendo analizar y evaluar la validez de argumentos basados en ellas, estas afirmaciones pueden ser verdaderas o falsas, pero no ambas [5].

La lógica proposicional es fundamental en programación, ya que se utiliza para construir relaciones condicionales mediante conectores lógicos y proposiciones.

Proposiciones Llamaremos de esta forma a cualquier afirmación que sea verdadera o falsa, pero solo una a la vez.

Proposiciones en Pseudocódigo

Las proposiciones lógicas son fundamentales en la construcción de pseudocódigo, debido a que permiten establecer condiciones para la toma de decisiones y el control del flujo de un programa.

4.6.2 Proposición:

La proposición es una afirmación que toma dos valores verdadera o falsa. En pseudocódigo, estas afirmaciones se evalúan por medio los condiciones si, sino y mientras.

Uso de Conectores Lógicos:

Las proposiciones pueden combinarse utilizando conectores lógicos como Y (AND), O (OR) y NO (NOT). Estos conectores permiten combinados pueden crear expresiones más complejas.

Conexión entre Proposiciones

4.6.3 Conjunción (Y)

"Es una unión de valores de verdad, su símbolo es Y [6]".

Ejemplo (A y B)

4.6.4 Disyunción (O)

"Es una separación o desunión de valores de verdad, su símbolo es O [6]"

Ejemplo (A o B)

4.6.5 Negación (NO)

Invierte (niega) el valor booleano del operando [6], al negar una proposición verdadera, el valor de verdad cambia a falso.

Ejemplo (NO A)

Conectores Lógicos en LPP

Las proposiciones pueden combinarse utilizando conectores lógicos como (Y), (O) y (NO). Estos conectores permiten crear expresiones más complejas.

A continuación, se explica cómo se representan las proposiciones en pseudocódigo, en los siguientes ejemplos.

Ejemplos

Proposición Simple



Figura 4-12 Proposicional simple
Fuente propia

En la Figura 4.12, la proposición A es simplemente una variable booleana que puede ser verdadera o falsa, evaluando solamente si es verdadera.

Ejemplo

Proposición con conector lógico (Y)

```

booleano A,B
Inicio
    A <- verdadero
    B <- falso

    // Evaluando el conector Y
    Si (A Y B) Entonces
        Escriba "Ambas proposiciones son verdaderas."
    Sino
        Escriba "Al menos una proposicion es falsa."
    Fin Si

Fin

```

The output window shows the message: "Al menos una proposicion es falsa." An annotation box points to this message with the text: "(Verdadero y Falso) entonces el resultado es falso."

Figura 4-13 Conector Y
Fuente propia

En la Figura 4.13, la proposición con conector Y, presenta dos variables booleanas A = verdadero y B= falsa, y su evaluación es falsa según la tabla de verdad 3.3 de la Conjunción Y.

A	B	A Y B
V	V	V
V	F	F
F	V	F
F	F	F

Tabla 4.3 Tabla de verdad conjunción
Fuente propia

- A Y B será verdadero únicamente cuando A Y B sean ambos verdaderos.
- En cualquier otro caso, el resultado es falso.

Proposición con conector lógico (O)

```

booleano A,B
Inicio
    A <- verdadero
    B <- falso

    // Evaluando el conector O
    Si (A o B) Entonces
        Escriba "Al menos una proposicion es verdadera."
    Sino
        Escriba "Ambas proposiciones son falsas."
    Fin Si

Fin

```

The screenshot shows a window titled 'LPP' with the path 'C:\Users\P-Fl-5\Pictures\Lib...'. The window displays the pseudocode above. A callout bubble points from the 'Si (A o B) Entonces' block to a note: '(Verdadero O Falso) entonces el resultado es verdadero.' The window also contains the text 'Al menos una proposicion es verdadera.'

Figura 4-14 Conector O
Fuente propia

En la Figura 4.14, la proposición con conector O, presenta dos variables booleanas A = verdadero y B= falsa, y su evaluación es verdadera según la tabla de verdad 3.4 de la conjunción O.

A	B	A O B
V	V	V
V	F	V
F	V	V
F	F	F

Tabla 4.4 Tabla de verdad conjunción
Fuente propia

- A v B será verdadero si al menos una de las proposiciones (A o B) es verdadera.
- Solo es falso cuando ambas proposiciones son falsas.

Ejemplo

Proposición con conector lógico (NO)

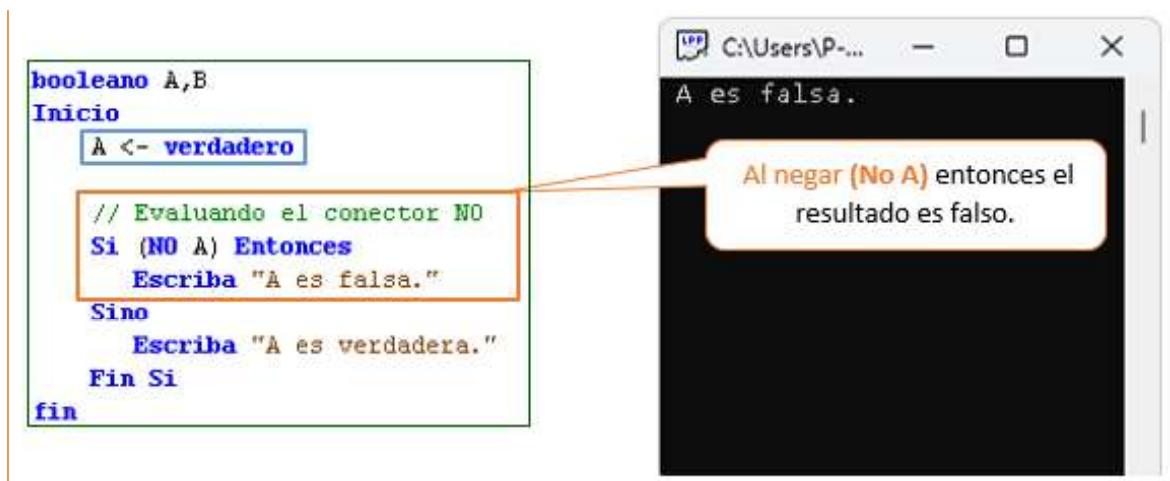


Figura 4.15 Conector negación
Fuente propia

En la figura 4.15, la proposición de negación, presenta la variable booleana $A =$ verdadero, y su evaluación es falsa según la tabla de verdad 3.5 de la negación no.

A	No A
V	F
F	V

Tabla 4.5 Tabla de verdad negación
Fuente propia

- Si A es verdadero (V), entonces $\neg A$ es falso (F)
- Si A es falso (F), entonces $\neg A$ es verdadero (V)

La negación: Se utiliza para invertir condiciones o proposiciones en lógica y programación.

Las proposiciones se representan mediante variables y condiciones evaluadas en estructuras de control. Utilizando los conectores lógicos (y), (o), (\neg), se pueden crear expresiones más complejas, que permiten tomar decisiones basadas en operadores relacionales y condiciones múltiples.

Ejemplo

La Figura 4-16 muestra un algoritmo que solicita al usuario ingresar un número por teclado para determinar si este pertenece al grupo de números menores o iguales a siete, o si es mayor que 50.

En la estructura condicional, se utiliza el conector lógico '**o**', que permite evaluar dos expresiones y ejecutar la acción si al menos una de ellas es verdadera. Si ninguna de las condiciones se cumple, el algoritmo saldrá de la estructura condicional.

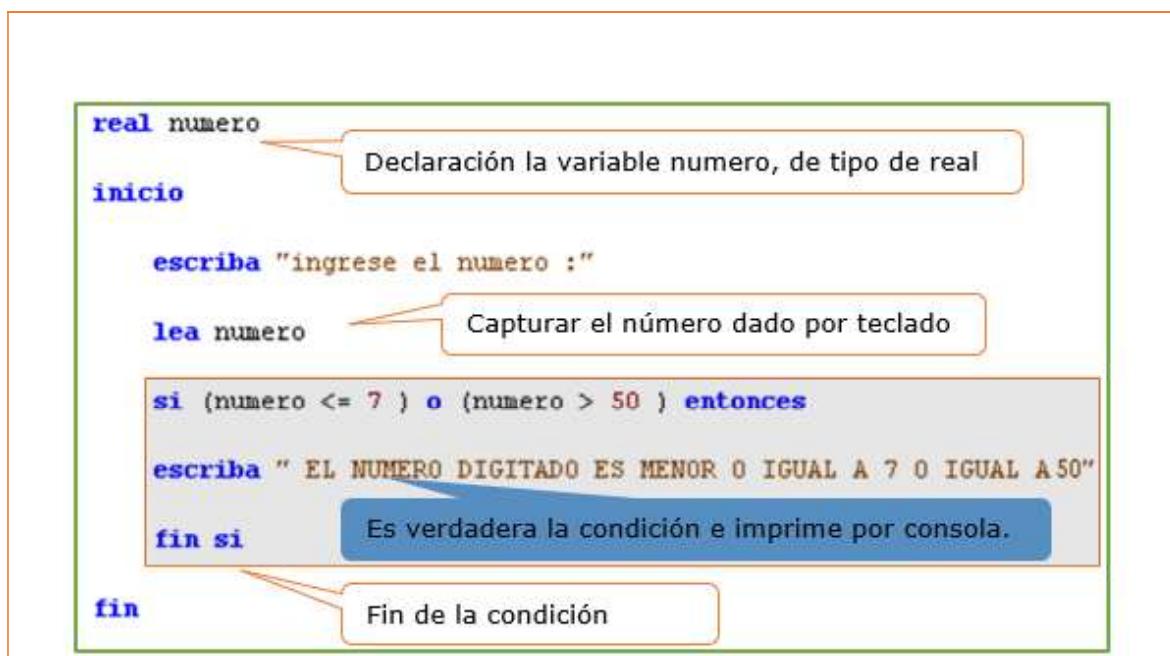


Figura 4-16 Selección con conectores

Fuente propia

Explicación

si (numero <= 7) o (numero > 50) entonces - fin si

En la figura 4.16 presenta una condición en la cual se utilizan operadores relacionales y lógicos para evaluar el valor de la variable numero, si esta condición es verdadera se imprime por consola.

Vamos a analizarla paso a paso:

Paso 1: Contiene dos expresiones (numero <= 7) (numero > 50).

Evalúa (**numero <= 7**) si es **verdadero**

Si el numero es menor o igual que 7.

Devuelve **verdadero**, si la variable numero es cualquier valor desde $-\infty$ hasta 7.

Evalúa (**numero > 50**) si es **verdadero**

Si numero es mayor que 50.

Devuelve **verdadero** si la variable numero es cualquier valor mayor a 50.

Paso 2 Operador lógico (O)

Combina las dos condiciones anteriores con sus operadores relacionales, al usar el operador (**O**), será verdadera si al menos una de las condiciones es verdadera, si las dos son falsas es falso de lo contrario verdadero.

Si el ejemplo la variable **numero** fuera el valor de 57

numero <- 57

si (**57 <= 7**) o (**57 > 50**) entonces

quedaría:

si (**falso**) o (**verdadero**) entonces

Para este ejemplo la condición (**si (falso o verdadero) entonces**) el resultado de la implicación es **verdadero**

4.7 Estructuras de selección anidadas

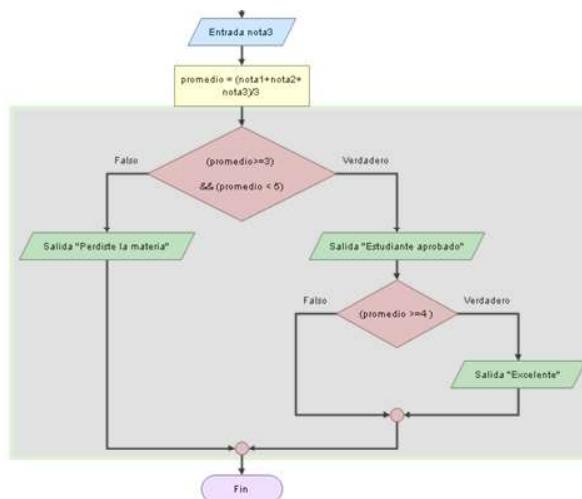


Figura 4-17 Selección anidada

Fuente propia

Concepto

Estructuras de selección anidadas: Son aquellas estructuras de decisión que se pueden asignar una dentro de otra, debido al número de alternativas. Además, las estructuras pueden contener más de dos opciones que sirven para una mejor toma de decisiones.

Las sentencias estructuras de control de selección pueden contener a otras estructuras de control **SI () entonces**, y estas a su vez puede contener a otras.

Este tipo de estructuras de control se les denominan anidadas ver Figura 4.18.

Sintaxis

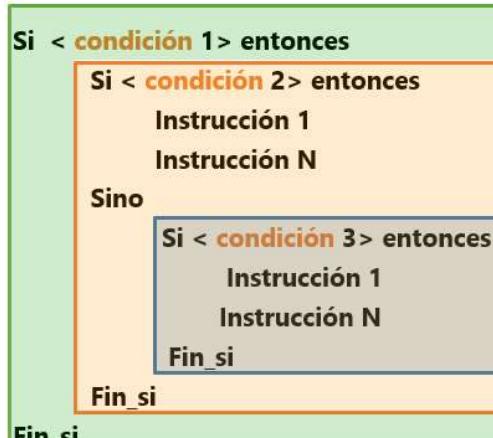


Figura 4-18 Condicionales en cascada

Fuente propia

Ejemplo

A continuación, se presenta un pseudocódigo Figura 4-19, que pide por teclado tres notas de un alumno con las siguientes opciones:

- Calcule el promedio.
- Si el promedio es ≥ 3 y ≤ 5 PRESENTAR "Aprobado".
- Si el promedio es ≥ 4 PRESENTAR "EXCELENTE".
- Si el promedio es < 3 PRESENTAR "PERDISTE".

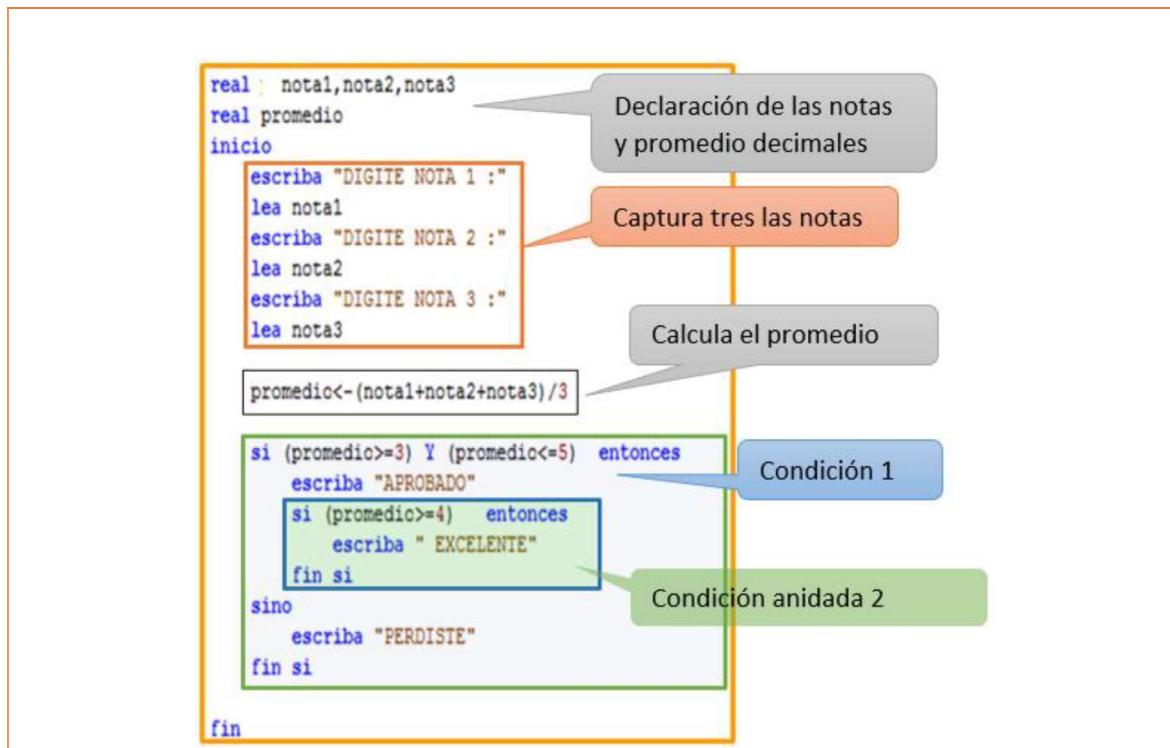


Figura 4-19 Ejemplo de selección anidada

Fuente propia

Explicación Figura 4.19**Entrada de datos**

Se solicitan tres notas al usuario en las variables: nota1, nota2, nota3.

Proceso

Cálculo del promedio: Se calcula el promedio de las tres notas.

Condiciones anidadas

Para realizar el algoritmo se deben realizar las tres condicionales pero anidados.

Si el promedio está entre 3 y 5, se muestra "Aprobado".

Si el promedio es mayor o igual a 4, se muestra "EXCELENTE".

Si el promedio es menor a 3, se muestra "PERDISTE".

4.8 Estructuras de selección múltiples



Figura 4-20 Selección múltiple
Fuente propia

Concepto:

Estructuras de selección múltiples: Permiten controlar decisiones, donde se desea evaluar varias opciones posibles, pero solo una alternativa a la vez.

En ciertos casos de un problema, se presentan diversas decisiones a elegir, para ello podemos utilizar los casos que son estructuras que acceden a una o varias opciones, dependiendo de un valor de un dato de la variable en cuestión.

Sintaxis

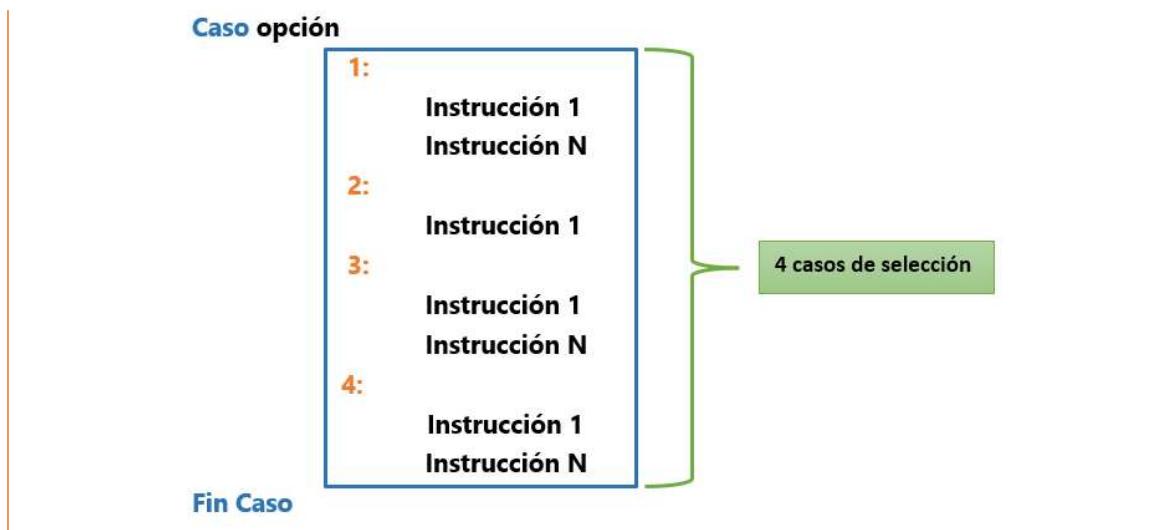


Figura 4-21 Selección múltiple

Fuente propia

Las estructuras de selección múltiples permiten controlar varias opciones, las sentencias de caso evalúan las opciones posibles, y elige la opción correcta, de esta manera permite ejecutar las de instrucciones necesarias.

Ejemplo:

Seudocódigo que pide ingresar al usuario, un numero comprendido entre el 1 y 7, luego el algoritmo debe imprimir el día de la semana.

```

entero dia
Inicio
    // Pedir al usuario que ingrese un número entre 1 y 7
    Escriba "Ingresa un número entre 1 y 7: "
    Lea dia
        7 opciones, solo una escoge
    // Evaluar el número de dia ingresado
    caso dia
        1: Escriba "Lunes"
        2: Escriba "Martes"
        3: Escriba "Miércoles"
        4: Escriba "Jueves"
        5: Escriba "Viernes"
        6: Escriba "Sábado"
        7: Escriba "Domingo"
    Fin caso
Fin

```

7 opciones, solo una escoge

Evaluó un día de 7 opciones

Indica que los casos terminaron

Figura 4-22 Ejemplo selección múltiple
Fuente propia

Explicación

Entrada de datos

El programa solicita al usuario que ingrese un número entre 1 y 7, capturada en la variable **dia**.

Proceso

Selección múltiple: Utiliza una estructura de **caso** para evaluar el número de día ingresado.

Casos multiples

Dependiendo del valor del número de día, imprime el día de la semana correspondiente.

El número está en dia=7, muestra un mensaje “**Domingo**”.

4.9 Ejercicios de la unidad

Estructuras de selección simple

- Realice un pseudocódigo que lea un número entero y muestre si el número es múltiplo de 10.
- Realice un pseudocódigo que lea dos números por teclado compruebe si son iguales.
- En el granero del vecino, se hace un 20% de descuento a los clientes cuya compra supere el precio de \$300.000 ¿Cuál será la cantidad que pagara una persona por su compra?
- Recibir 5 estaturas desde el teclado y determinar en LPP el porcentaje de estaturas por debajo de 1.50

Estructuras de selección compuestas

- Introducir la edad de una persona e imprimir si esa persona es bebe, niño, joven, adulto, anciano o está que se muere ≥ 100 .
- Introducir 3 números por teclado y deducir quien es el mayor medio y menor.
- Dadas las variables de tipo entero con valores $A = 5$, $B = 3$ indicar si la evaluación de estas expresiones daría como resultado verdadero o falso:
 - $A > 3$
 - $A \neq B$
- Dadas las variables de tipo entero con valores $A = 5$, $B = 3$, $C = -12$ indicar si la evaluación de estas expresiones daría como resultado verdadero o falso:

A > 3	A > C	A < C
B < C	$B \neq C$	$A == 3$
A * B == 15	$A * B == -30$	$C / B < A$
C / B == -10	$C / B == -4$	$A + B + C == 5$
(A+B == 8) && (A-B == 2)	$(A+B == 8) (A-B == 6)$	$A > 3 \&\& B > 3 \&\& C < 3$
A > 3 && B >= 3 && C < -3		

Estructuras de selección dobles

- Realice un pseudocódigo que lea un número entero y calcule si es par o impar.

Estructuras de selección anidadas

- Construir un programa que calcule el índice de masa corporal de una persona ($IMC = \text{peso} [\text{kg}] / \text{altura}^2 [\text{m}]$) e indique el estado en el que se encuentra esa persona en función del valor de IMC:

Valor de IMC	Diagnóstico
< 16	Criterio de ingreso en hospital
de 16 a 17	infrapeso
de 17 a 18	bajo peso
de 18 a 25	peso normal (saludable)
de 25 a 30	sobrepeso (obesidad de grado I)
de 30 a 35	sobrepeso crónico (obesidad de grado II)
de 35 a 40	obesidad premórbida (obesidad de grado III)
>40	obesidad mórbida (obesidad de grado IV)

- Crea un pseudocódigo para calificaciones que solicite a un estudiante ingresar una nota y determine la letra correspondiente: A, B, C, D o F. Usa estructuras anidadas para evaluar las calificaciones.
- Con el ejercicio anterior con la nota (de 0 a 100) y clasifique el resultado en "Insuficiente", "Suficiente", "Bien", "Notable" o "Sobresaliente".
- Realiza un pseudocódigo que solicite al usuario el total de la compra en un almacén y determine el porcentaje según el monto. Los descuentos son: 5% para compras menores a \$100, 10% para compras entre \$100 y \$500, y 15% para compras mayores a \$500.

Estructuras de selección múltiples sentencia caso

- Realice un pseudocódigo que indica la correspondencia de un número introducido por el usuario, el numero deberá estar comprendido del 1 al 12, luego debe imprimir el nombre mes.
- Desarrolla un pseudocódigo que solicite un idioma a una persona con las opciones (inglés, español, francés, alemán, etc.) e imprima por consola un saludo en ese idioma.
- Escribe un algoritmo que pida la edad de una persona y determine la etapa de su vida con la siguiente tabla:
 - 0-12 años: Niño.
 - 13-17 años: Adolescente.
 - 18-64 años: Adulto.
 - 65 años o más: Adulto mayor.
- Construir un pseudocódigo que simule el funcionamiento de una calculadora que realice las cuatro operaciones aritméticas básicas (suma, resta, producto y división) con valores Numéricos enteros, división real, el algoritmo deberá imprimir el resultado. El usuario debe especificar la operación con el primer carácter del primer parámetro de la línea de comandos: S o s para la suma, R o r para la resta, P, p, M o m para el producto y D o d para la división.
- Desarrolla un pseudocódigo que presente un menú con las siguientes opciones:
 - Ver saldo.
 - Realizar depósito.
 - Realizar retiro.
 - Salir.

4.10 Bibliografía

Fernandez, Carmen. "Java lo básico que debe saber". Editorial: Ediciones de la U. 2011.

Joyanes, Luis. "Fundamentos de Programación – Algoritmos, estructuras de datos y objetos". Editorial: Mc Graw Hill. 2008.

- [1] G. Brassard y P. Bratley, Fundamentals of Algorithmics, Englewood Cliffs: Prentice-Hall, 1996.
- [2] OpenAI, «ChatGPT [Large language model],» 2024. [En línea]. Available: <https://chatgpt.com>.
- [3] R. Atzori, «Flowgorithm,» 12 junio 2024. [En línea]. Available: <http://www.flowgorithm.org/>.
- [4] R. Berlanga Llavori, Introducción a la programación con Pascal., Universitat Jaume I., 2000.
- [5] M. Hernández Bejarano y L. E. Baquero Rey, Programación orientada a objetos en Java. Buenas prácticas, Ediciones de la U., 2023.
- [6] Digital Content, «IAN,» 2025. [En línea]. Available: <https://ebooks7-24-com.unimayor.lookproxy.co>.
- [7] O. E. Avalos, Programación con GO, vol. 1, RedUsers, 2020.
- [8] O. I. Trejos Buriticá, Lógica de programación ..., Ediciones de la U, 2017.
- [9] L. Joyanes Aguilar, Z. Sánchez García, I. Zahonero Martinez y A. Castillo Sanz, Algoritmos, programación y estructuras de datos, McGraw-Hill, 2005.
- [10] M. Villalobos, Fundamentos de programación JAVA, Macro, 2014.
- [11] M. Hernández Bejarano y L. E. Baquero Rey, Estructuras de datos, Ra-Ma S.A. Editorial y Publicaciones, 2002.

- [12] M. A. Corona Nakamura y M. A. Ancona Valdez, Diseño de algoritmos y su codificación en lenguaje C, McGraw Hill, 2011.
- [13] L. Joyanes Aguilar, Fundamentos de programación: algoritmos, estructura de datos y objetos., McGraw-Hill., 2020.
- [14] OpenAI, 20 1 2024. [En línea]. Available: <https://chat.openai.com/>.
- [15] I. Y. Polanco Guzman y . F. Betan, 115 ejercicios resueltos de programación C++, Rama, 2022.
- [16] L. Joyanes Aguilar, Fundamentos de programación: algoritmos, estructura de datos y objetos, McGraw-Hill., 2020.
- [17] Raffino, «Algoritmo en informática.,» 25 11 2024. [En línea]. Available: <https://concepto.de/algoritmo-en-informatica/..> [Último acceso: 2024 12 6].
- [18] «Introducción a la programación,» 2002. [En línea]. Available: <https://mediatecnica.weebly.com/lpp.html>.
- [19] T. H. Cormen, C. E. Leiserson, R. L. Rivest y C. Stein, Introduction to Algorithms, MA: 3rd ed. Cambridge MIT Press, 2009.
- [20] R. S. Pressman, Software Engineering: A Practitioner's Approach, 8th ed. , NY; New York: McGraw-Hill, 2014.
- [21] E. Yourdon, Modern Structured Analysis., Englewood Cliffs: Yourdon Press, 1989.

ALGORITMOS Y PROGRAMACIÓN BÁSICA EN LPP

C
L
O
P
D
E
P

Autores

Pedro Harvey Álvarez Sánchez
Alberto Bravo Buchelly
Freddy Alonso Vidal Alegría
Gabriel Elías Chanchí Golondrino

