Autores:
Mejía Viteri, José T.
Gonzáles Valero, María I.
España León, Ángel R.
Pauta Ríos, Roberto C.
Ruiz Parrales, Iván R.

Fundamentos de Programación ¿Qué se necesita saber para aprender a programar?

ISBN 978-9942-8679-2-6

Fundamentos de Programación ¿Qué se necesita saber para aprender a programar?

José Teodoro Mejía Viteri

María Isabel Gonzáles Valero

Ángel Rafael España León

Roberto Carlos Pauta Ríos

Iván Rubén Ruiz Parrales

Dedicatoria

Este libro está dedicado a los esfuerzos realizados por nuestros padres, por invertir en nosotros, en lo más importante que es una eduación para garantizar nuestro futuro y el de nuestras familias.

Introducción

El libro titulado :Fundamentos de Programación ¿Qué se necesita saber para aprender a programar?, ha sido desarrollado por los autores con la finalidad de expresar los conceptos básicos necesario para comprender la programación, a tal efecto el texto y/o libro tiene como finalidad exponerle al lector la historia de la programación, la evolución épocal, las estructuras y diferentes dispositivos tanto de entrada como de salida que subyace en esta área del conocimiento.

Los primeros aspectos tratados sobre el desarrollo de soluciones, se establecen como base para lograr comprender un problema y posteriormente automatizarlo a través del computador, este paradigma de la programación se modela a través del desarrollo de algoritmos, para luego desarrollar el código fuente en un lenguaje. Por ejemplo, la matriculación de un estudiante se representan como una serie de funciones que manipulan datos.

Por tanto este texto proporciona a los alumnos una serie de problemas que abarcán desde los conocimientos de las operaciones básicas hasta lograr enrolarlo en soluciones de las estructuras de la programación(secuencias, desiciones y ciclos), de tal forma que logre la experticia para saber cuando aplicar una estructura determinada para solucionar un problema.

Este libro no pretende establecer un estandar para resolver problemas, solo proporcionar conocimiento a todas las personas para desarrollar la lógica apropiada para solucionar problemas en cualquier lenguaje de programación es por aquello que se enfoca una parte del texto al desarrollo de los problemas en lenguaje de programación C, porque permite una programación estructurada y no esta enfocado a ningún sistema operativo especifico.

Índice

CAPÍTULO I

CONTENIDO

- Que es una Computadora
- Estructura de las Computadoras
- Redes, web e Internet
- Lenguajes de Programación

1. QUE ES UNA COMPUTADORA

1.1 INTRODUCCIÓN

Según (Long, 2009) la computadora sirve a la sociedad como una herramienta para realizar y simplificar muchas de sus actividades y procesos. En sí, es un dispositivo electrónico capaz de interpretar y ejecutar los comandos programados para realizar en forma general las funciones de:

- ✓ Operaciones de entrada
- ✓ Operaciones de cálculo, lógica y almacenamiento.
- ✓ Clasificar u ordenar, seleccionar, corregir y automatizar.
- ✓ Operaciones de salida

La computadora, es una máquina que realiza operaciones lógicas y matemáticas utilizando rutinas o programas informáticos; fue **John Atanassoff** quién inventó el prototipo de la computadora moderna.

Al inicio, sólo existían máquinas análogas, las cuales eran consideradas por el físico, como lentas y muy imprecisas. Por tanto Atanassoff comenzó rápidamente a idear su máquina digital (en **1933**). Para la creación de la computadora, el físico ideo cuatro conceptos básicos para su desarrollo que son, electricidad y componentes electrónicos, un sistema binario, condensadores para almacenar datos o información y un sistema lógico para el cómputo.

Con la ayuda de Clifford Edward Berry, comenzaron a trabajar en la computadora, en uno de los sótanos de la universidad del estado de Iowa. La máquina que desarrollaron, tuvo un costo final, de más de mil cuatrocientos dólares. La máquina en sí, estaba constituida, por un tambor rotatorio para manejar la información en la memoria, un sistema lógico, capacitadores y tubos al vacío.

1.1.1 Breve historia

Blaise Pascal invento en 1642 la primera máquina de calcular mecánica, este dispositivo utilizaba una serie de ruedas de diez dientes en las que cada uno de los dientes representaba un dígito del 0 al 9. Las ruedas estaban conectadas de manera que podían sumarse números haciéndolas avanzar al número de dientes correctos. En 1670 el filósofo Gottfried Wilhelm Leibniz, perfeccionó esta máquina e inventó una que también podía multiplicar.

El inventor francés Joseph Marie Jacquard, al diseñar un telar automático en el año de 1801, utilizó delgadas placas de madera perforadas para controlar el tejido utilizado en los diseños complejos. Durante la década de 1880 el estadístico estadounidense Herman Hollerith concibió la idea de utilizar tarjetas perforadas, similares a las placas de Jacquard, para procesar datos.

Es difícil atribuir a una sola persona, la invención de la computadora. Para los expertos, son varias las personas que aportaron conocimientos y creaciones, como contribución para el desarrollo de este invento.

1.1.2. Actualidad

Una computadora es una máquina electrónica usada para procesar todo tipo de información. Podemos hacer trabajos de oficina con ella, guardar datos, imágenes, escribir cartas, leer libros, comunicarnos con familiares o amigos a través de correos electrónicos o video llamadas, ver videos, dibujar, hacer informes, crear software de computadoras que llevan a cabo diversas funciones e incluso nos permite hacer presentaciones que pueden ver otros usuarios de computadoras alrededor del mundo.

Es una herramienta esencial, en todos los campos de conocimiento; es útil, ayuda a la mejora y excelencia del trabajo, hace más fácil y práctico el desarrollo los procesos.

En poco tiempo, las computadoras se han integrado a nuestra vida cotidiana, y han transformado los procesos laborales complejos en actividades simples.

Los periféricos de las computadoras son esenciales, por que sin ellos no sería útil a los usuarios, a interactuar con los aplicativos que se encuentran instalados en el ordenador.

Figura #1: periféricos de entrada y salida
Fuente: desarrollada por los autores

1.2 ESTRUCTURA DE UNA COMPUTADORA

Cuando se trata acerca de la estructura de una computadora es hablar de organización, distribución y combinación de elementos que llegan a formar parte de un sistema, que representan un sistema que integra diferentes componentes organizados en niveles independientes, uno de otros en cuanto a su estructura, conservando la interdependencia desde el nivel más alto hasta el nivel más bajo en su funcionamiento.

Un computador tiene dos partes que funcionan de manera conjunta, ninguna puede existir sin la otra y estas son: el hardware y el software.

La estructura general de las computadoras considera las siguientes funciones:

El procesamiento de datos. - Se encarga de transformar y dar tratamiento a los datos aplicando funciones básicas como las aritméticas y las lógicas.

El almacenamiento de los datos.- Los datos son almacenados temporalmente en localidades de memoria y su contenido cambia continuamente debido a la gran cantidad de cálculos que realiza el procesador.

La entrada y salida de datos.- Cuando los datos son recibidos o enviados desde algún dispositivo conectado a la computadora se conoce como proceso de Entrada / Salida de datos; y el dispositivo de referencia es conocido como periférico; cuando los datos son movidos a grandes distancias, el proceso es conocido como comunicación de datos. Se da internamente en la computadora y de manera dinámica con el medio ambiente operativo constituido por dispositivos que sirven como fuente o destino de los datos.

La función de control.- Administra los recursos de la computadora, así como partes funcionales en respuesta a los programas residentes en la computadora. La estructura interna de una computadora está compuesta por cuatro componentes principales:

Unidad de Central de Proceso (CPU). - Controla, coordina y temporiza las funciones de operación de la computadora y ejecuta funciones de procesamiento de datos, se le conoce generalmente como procesador.

Unidad de Control. - Controla las operaciones del CPU.

Unidad Aritmética y lógica (ALU). - Ejecuta las funciones de procesamiento de datos.

Memoria Principal. - Almacena toda la información que el computador está usando.

Registros. - Provee almacenamiento en el CPU.

Entrada / Salida (E/S). - Movimiento de datos entre el pc y su medio ambiente externo.

Sistema de Interconexión.- Son mecanismos que permiten la comunicación con la unidad de control, la unidad aritmética y lógica y los registros.

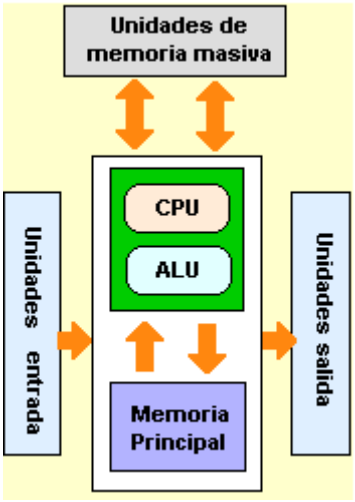


Figura #2: estructura de una computadora
Fuente: desarrollada por los autores

1.2.1. Unidades de Entrada y Salida

1.2.1.1. Unidades o dispositivos de entrada.

Mediante estos dispositivos, se introducen instrucciones y/o programas en la CPU, permiten introducir datos externos a la computadora para su posterior procesamiento y pueden provenir de distintas fuentes, algunos dispositivos de entrada más utilizados son:

- ✓ Teclado
- ✓ Mouse
- ✓ Escáner
- ✓ Lector de código de barras
- ✓ Joystick (controles para juegos)
- ✓ Micrófono
- ✓ WebCam
- ✓ Tableta digitalizadora
- ✓ Pantalla sencibles al tacto

- ✓ La Unidad Central de Proceso.

La Unidad Central de Proceso o *Central Processing Unit*(CPU), Interpreta las instrucciones y procesa los datos contenidos en los programas de la computadora, es el cerebro de ella, donde se realizan la mayoría de los cálculos. La unidad lógica/aritmética (ALU), realiza operaciones aritméticas y lógicas. La unidad de control (CU), extrae instrucciones de la memoria, las descifra y ejecuta llamando a la ALU cuando es necesario.

1.2.1.2. Unidades o dispositivos de salida.

Permiten ver los resultados de los cálculos o de las manipulaciones de datos de la computadora, reciben información que es procesada por la CPU y la reproducen para que sea entendible su función es eminentemente receptora, algunos dispositivos de salida de uso común son:

- ✓ Monitores
- ✓ Impresoras
- ✓ Bocinas
- ✓ Auriculares
- ✓ Graficadores

1.3 REDES, WEB E INTERNET

Una **red** de ordenadores se trata de un conjunto de computadores conectados entre sí, esta conexión entre los equipos constituye la infraestructura imprescindible que permite compartir datos y recursos entre ellos, para que esta comunicación entre máquinas pueda producirse es necesario disponer de un conjunto de normas, al que se denomina protocolo, que organiza los diferentes aspectos que intervienen en el proceso.

Internet es la red de redes de ordenadores conectada a nivel mundial, denominada "Internet", que tiene la particularidad que cada una de las redes es independiente y autónoma.

Las redes que forman parte de Internet son de diversos tipos, propósitos y tamaños. Existen redes públicas y privadas; locales, regionales e internacionales; institucionales, educativas, universitarias, dedicadas a la

investigación, al entretenimiento entre otras.

Web es un documento electrónico que contiene información, cuyo formato se adapta para estar insertado en la World Wide Web, donde los usuarios a nivel mundial puedan ingresar por medio del uso de un navegador y realizar diferentes actividades (García, 2010).

1.4 LÓGICA

La lógica es una capacidad innata del ser humano, es el proceso de tener claro en nuestra mente cuál es el camino más viable para poder solucionar un problema, es decir que la lógica nos permite analizar los medios que se tiene y cómo lograr un objetivo.

La lógica la utilizamos durante todo el proceso de nuestra vida, en las rutinas cotidianas del hogar, en la oficina, en el deporte, en la toma de decisiones incluso en el amor, es pensar cómo hacer, como hacerlo y cómo lograrlo.

1.5 LENGUAJES DE PROGRAMACIÓN

1.5.1 Lógica de Programación

La lógica de programación es la capacidad que debe tener de un programador de computadoras, se la debe de aprender y ejercitar en el proceso de formación académica, esta nos enseña a diseñar soluciones a problemas de la vida real para su posterior complementación en forma de aplicaciones, que se ejecutan en una gran variedad de dispositivos. (Gottfried, 2011).

La lógica de Programación no exige de ningún conocimiento previo de computación y ningún tipo de tecnología en general, tampoco exige el conocimiento previo de algún tipo de lenguaje de programación, aunque no puede negarse el aprender en paralelo de alguno de ellos, su complementación se da después de que se manejen bien los conceptos de lógica de programación, para implementar y ver convertida en realidad las soluciones lógicas a sus objetivos.

1.5.2 Que es un Programa de Computador.

Según (Tucker, 2015) un programa es un conjunto de instrucciones que permiten lograr un objetivo al ser ejecutado. Cuando hablamos específicamente de un programa, se refiere a un software, que trata de aplicaciones y recursos que permiten desarrollar diferentes tareas en una gran gama de equipos tecnológicos.

1.5.3 Que es Programación

Es una serie de pasos a través de instrucciones para solucionar un problema. En este proceso se diseña, codifica, depura y se tiene como resultado un código fuente. Este código fuente es escrito basado en un lenguaje de programación. El propósito es desarrollar aplicativos que presenten un comportamiento deseado. El proceso de escribir código requiere frecuentemente un amplio conocimiento en varias áreas, además del dominio del lenguaje de programación (Muñoz & Palacios, 2006).

El lenguaje de programación se utiliza para describir el conjunto de acciones consecutivas que un equipo debe ejecutar. Se puede decir que es un modo práctico para que los seres humanos puedan dar instrucciones a un equipo. También se lo conoce como código de máquinas o lenguaje de máquinas.

El lenguaje de máquina no es comprensible para los seres humanos, por lo cual se han desarrollado lenguajes intermediarios comprensibles para el ser humano ya que son herramientas que permiten crear software. Entre ellos: Delphi, Visual. Net, Php, Java, c# y otros.

1.5.4 Historia de los Lenguajes de Programación

Charles Babbage inicia la historia de los lenguajes de programación, inventó su computadora en el año 1822, porque vio la necesidad de que exista un lenguaje que permita comunicarse con esta máquina. Estos primeros lenguajes de programación estaban muy unidos a la computadora y fueron creados para cada una de ellas. Era muy rudimentario y consistía en la programación de los diferentes cambios de engranajes que ejecutaban los cálculos.

En el año 1942 se construyó la ENIAC (Computador e Integrador Numérico Electrónico), computadora que se programaba con interruptores y era preciso reescribir el sistema entero para cada nuevo programa, en el año 1945 se desarrolló una nueva técnica que establecía que las instrucciones complejas se deben utilizar para controlar el hardware simple.

En el año 1957 aparece el primero de los lenguajes de programación más importantes, Fortran de alto nivel, luego en 1958 se creó el lenguaje de programación Lisp o proceso de lista que fue diseñado para la investigación de la inteligencia artificial.

En 1968 apareció Pascal creado por el profesor suizo Niklaus Wirth entre los años 1968 y 1969 este lenguaje de programación se usó como uno de los mejores para enseñar programación a los estudiantes.

El lenguaje de programación C fue creado en 1972 por Dennis Ritchie para desarrollar los sistemas operativos Linux. Posteriormente se han desarrollado otros lenguajes de programación entre los más importantes se pueden mencionar: C++, Java, BASIC, Visual Basic.Net, Cobol, SQL, C#, Php entre otros.

1.5.5. Evolución y Paradigmas

Un paradigma de programación es una propuesta tecnológica cuyo núcleo central es incuestionable en cuanto a que unívocamente trata de resolver uno o varios problemas claramente delimitados en el tiempo en cuanto a aceptación y uso porque nuevos paradigmas aportan soluciones que la sustituyen parcial o totalmente.

Los paradigmas de programación, más conocidos son los siguientes:

- **Paradigma de la programación estructurada o imperativa:** Es el paradigma de programación que fue utilizado en el pasado, lo soportan lenguajes como *'C', 'BASIC'* y *'Pascal'*.

- **Paradigma de la programación funcional:** Paradigma basado en la utilización de funciones aritméticas que no manejan datos mutables o estados. Este paradigma lo soportan lenguajes como *'Haskell'* y *'Lisp'*.

- **Paradigma de la programación Lógica:** Es un paradigma orientado a las matemáticas se basa en el concepto de función. Este paradigma lo soportan lenguajes como *'Prolog'*.

- **Paradigma de la Programación Orientada a Objetos (POO):** Es quizás el paradigma de programación más utilizado que se acerca al mundo real y permite aumentar la comprensibilidad de los problemas. Este paradigma de programación lo soportan lenguajes como *'C++'*, *'Java'* y *'C#'*.

- **Paradigma de la programación orientada a Aspectos:** Incluye como característica el concepto de "Aspecto" que pretende dar una determinada funcionalidad al sistema. Este nuevo paradigma de programación lo soporta el lenguaje de programación *'AspectJ'*.

CAPÍTULO II

CONTENIDO

- **Datos y Tipos de Datos**
- **Operadores**
- **Resolución de Problemas**
- **Pseudocódigo**

Introducción

En este capítulo se menciona, los tipos de datos y operadores que se utilizan en los lenguajes de programación, además de especificar como se debe realizar el análisis de un problema para posteriormente trasladarlo a un lenguaje de programación, los pasos básicos son los siguientes:

1.- Definición o análisis del problema

2- Diseño del Algoritmo

3.- Pseudocódigo (transformación del algoritmo)

2.1 DATOS Y TIPOS DE DATOS

Los Datos son una representación simbólica que son de diferentes tipos (números, letras, alfanumérica, otros.) estos son cualitativo o cuantitativo.

Los principales tipos de datos que van hacer utilizados en este libro para resolver problemas a través de los algoritmos y pseudocódigos se detallan a continuación:

- Numéricos
- Lógicos
- Carácter (una letra, o cadenas de letras o caracteres).

2.1.1 Datos Numéricos.

Este tipo de dato representa números que son utilizados para las diferentes operaciones matemáticas los cuales son:

- Enteros
- Reales

2.1.1.1 Enteros.

Los números enteros están conformados por los números naturales positivos, negativos y el cero, por ejemplo:

2	-2
0	243
5	-25

Los números enteros pueden ser enteros cortos y enteros largos esto depende mucho de los lenguajes de programación, los rangos son los siguientes:

Enteros cortos	-32768	a	32767
Enteros largos	-2147483648	a	2147483647

2.1.1.2 Reales.

Los números reales están compuestos por una parte entera y la otra está separada por un punto que es la parte fraccionaria o decimal, por ejemplo:

4.5	1234.567
3.0	23.45
23.5	789.98

Existen números de notación denominada científica, que son utilizados para números muy grandes o muy pequeños, por ejemplo.

4.45678×10^9

2.1.2 Datos Lógicos

Estos datos solo pueden tomar dos valores verdadero o falso, este tipo de datos es utilizado en las estructuras condicionales en pseudocódigo.

2.1.3 Datos Carácter y Cadena de caracteres.

Carácter es la forma de nombrar a la combinación de letras, números y símbolos que pueden ser representados por el computador, por ejemplo:

Caracteres (a, A, b, X, y, s, g).

Cadena de caracteres (José, Carlos, pato, ropa, 10 de agosto).

2.1.4. Constantes y Variables.

Las constantes son los datos que no cambian durante la ejecución de algoritmo o programa, mientras que una variable puede cambiar el valor durante la ejecución del mismo.

La constante tiene las siguientes sintaxis:

conts <Tipo> <nombre>= <valor que permanecerá durante la ejecución del programa>

Ejemplo:

const cadena apellido=' Mejia'

const entero valor=125

La variable tiene la siguiente sintaxis:

<tipo de dato>: <nombre de la variable>

Ejemplo:

entero: numero

real: cifra

2.2 OPERADORES UTILIZADOS EN EL DESARROLLO DE ALGORITMOS Y PSEUDOCÓDIGO.

Existen diversos tipos de operadores que pueden ser utilizados en el desarrollo de algoritmos y pseudocódigos, que a continuación se detallan:

2.2.1 Operadores Aritméticos.

Estos operadores tienen las mismas funciones de las operaciones matemáticas y son:

+	suma
-	resta
*	multiplicación
div	división (cociente de la división)
mod	módulo o resto
^	Potenciación

Las operaciones que se realizan con estos operadores aritméticos en los algoritmos y pseudocódigos, se detallan a continuación:

Aritmética		**Algoritmo o Pseudocódigo**
3+4	se representa por	3+4
2-1	se representa por	2-1

3X2	se representa por	3*2
2^7	se representa por	2^7
3 ÷ 2	se representa por	3 div 2

Operador Mod (módulo o residuo de la división)

Este operador se utiliza en caso de realizar operaciones donde se requiere trabajar con el residuo de la división, por ejemplo, cuando necesita saber si un número es par o impar, aritméticamente sabemos que un número es par si el residuo de su división para 2 es igual a 0, para representarlo en operación aritmética en un algoritmo, ejemplo:

8 mod 2 resultado de la operación es 0

18 mod 4 resultado de la operación es 2

2.2.2 Resolución de Expresiones Aritméticas.

Las expresiones aritméticas que están formadas por más de un operador tienen sus reglas para la ejecución y obtención de resultados, que a continuación se detallan.

- Las operaciones que se encuentran entre paréntesis se evalúan primero. Si existen paréntesis anidados (interiores unos a otros), las expresiones más internas se evalúan primero.
- Las operaciones aritméticas dentro de una expresión suelen seguir el siguiente orden de prioridad:
  1. operador ( )
  2. operadores ++, – – + y – unitarios,
  3. operadores *, /, % (producto, división, módulo)
  4. operadores +, – (suma y resta).

En los lenguajes de programación que manejan exponenciación, este operador supera todas las prioridades.

Ejercicios sobre solución de expresiones aritméticas.

1) 5+4*5

Solución

Procedemos a realizar primero la multiplicación

5+ 4* 5

Y luego se realizan las sumas

5+20

25

2) 9+4 * 5 + 3*5

Procedemos a realizar las multiplicaciones

9+4 * 5 + 3*5

Y Luego la sumas

9+20+15

3) -4 * 4 + 2 ^ 3 / 2-5

- 4 * 4 + 2 ^ 3 / 2-5

- 4*4 + 8 /2-5

-16+4-5

-16-1

-17

Los ejercicios que tienen paréntesis son prioridad de las operaciones

X*(Y+2) el orden de la operación sería primero la constante 2 se sumaría al valor de Y, después este resultado se multiplica por el valor de X.

2.2.3 Operadores de Relación.

Los operadores relaciónales se utilizan para comparaciones entre valores numéricos, carácter o cadena de caracteres. Son muy utilizados para representar condiciones en los algoritmos y estos pueden ser:

Operador	Significado
<	menor que
>	mayor que
=	igual
<=	menor igual
>=	mayor igual
<>	distinto de

En términos generales la sintaxis de los operadores relaciones es:

Expresión 1 operador Expresión 2

Ejercicios con operadores relacionales.

Expresión	Resultado
5>6	falso
8>2	verdadero
5=5	verdadero
4<=4	verdadero.
3<>3	falso.

2.2.4 Operadores Lógicos.

Los operadores lógicos utilizados en los algoritmos son and, or, not su utilización se basa en las siguientes especificaciones:

Operador not

a	Not a
Verdadero	Falso
Falso	Verdadero

Operador and

a	a	a and b
Verdadero	Verdadero	Verdadero

Verdadero	**Falso**	**Falso**
Falso	**Verdadero**	**Falso**
Falso	**Falso**	**Falso**

Operador or

a	**b**	**a or b**
Verdadero	**Verdadero**	**Verdadero**
Verdadero	**Falso**	**Verdadero**
Falso	**Verdadero**	**Verdadero**
Falso	**Falso**	**Falso**

Ejercicios de expresiones relacionales y lógicas.

1) (2<5) **y** (3<4)

Primero evaluamos las expresiones con los operadores relacionales.

(2<5) y (3<4)

verdadero y verdadero

Luego se evalúa la expresión lógica que nos resulta de acuerdo a lo establecido con la tabla de valores del operador y da como respuesta final verdadero.

2) **No** (5<>5)

Primero evaluamos la expresión con el operador relacional.

No (5<>5)

Falso

Luego aplicamos el operador lógico de la negación y el resultado final es verdadero.

En estos ejercicios se observa la prioridad que tiene los paréntesis para resolver las expresiones.

2.2.5.Funciones

Existen funciones internas que de acuerdo al lenguaje de programación difiere su sintaxis y estas van hacer expuestas de una manera estándar utilizadas para el entendimiento de estas en los algoritmos y pseudocódigo, a continuación, se detallan:

Función	**Descripción**
raiz2	calcula la raíz cuadrada de un número.
redondeo	realiza el redondeo de un número.
exp	Exponencial de un número.
sen	Calcula el seno de un valor
cos	Calcula el coseno de un valor
abs	calcula el valor absoluto de un valor

Ejercicio de Funciones

Expresión	**Resultado**
Raiz2(25)	5
Redondeo(6.5)	7
Redondeo(2.2)	2
abs(4)	9
abs(-4)	9

2.2.6 Asignación de valores.

La operación de asignación incorpora un valor a una variable, esta difiere de los lenguajes de programación, esta puede representarse por el signo igual (=) o dos puntos igual (: =), por razones de uso del pseudocódigo el cual no tiene sintaxis definida, utiliza la flecha apuntando hacia la izquierda (←) su sintaxis es:

<nombre de la variable> ← <expresión>

Ejemplos de operación de asignación.

1) a ← 5

El ejercicio1 indica que a la variable **a** se le asignó el valor de 5

2) a ← b+1

El ejercicio 2 indica que a la variable a se le asigna el valor de b+1

Asignación lógica.

En esta asignación el valor que se asigna es un valor de verdadero o falso.

1) a ← 2<3

El valor asignado a la variable **a** es verdadero.

Asignación de cadenas de caracteres.

El valor de asignación a la variable va hacer una cadena de caracteres.

1) x← 'hola'

El resultado es la asignación de la cadena de caracteres 'hola' a la variable x.

En las operaciones de asignación es importante el tipo de variable porque una variable no puede recibir valores de diferente tipo al que pertenece.

2.3. RESOLUCIÓN DE PROBLEMAS.

Para solucionar un problema con el computador es necesario realizar varias etapas que contemplan una serie de procesos que los programadores deben seguir para obtener una solución que cumpla con las especificaciones del usuario, y se detallan a continuación:

- Análisis del problema.
- Diseño de un algoritmo.
- Codificación.

- Compilación y ejecución.
- Verificación.
- Depuración.
- Mantenimiento.

Análisis del problema. En esta etapa existe un participante importante, el usuario final del software porque él se encargará de usarlo.

Diseño del Algoritmo. Una vez que se realiza el análisis del problema este se debe plasmar en un algoritmo que es la solución al problema.

Codificación. Este es el proceso de llevar el algoritmo a un lenguaje de programación.

Compilación y Ejecución. En esta etapa el programador verifica posibles errores en los procesos planteados tanto en el algoritmo o en el lenguaje de programación.

Mantenimiento. A toda solución a medida que el usuario solicita nuevos requerimientos o cambios en una normativa que contempla un proceso en el software.

Una etapa no planteada, pero debe ser considerada como una exigencia al momento de la entrega de la versión del software final es la documentación, donde se describe todos los procesos automatizados.

2.3.1 Análisis del problema.

Esta es la fase inicial para poder solucionar un problema con la ayuda del computador por tanto los procedimientos a seguir para realizar esta fase se detallan a continuación.

- **Primero** se deben identificar los datos de entrada que se requieren para resolver el problema.
- **Segundo** se deben reconocer los procesos que tienen que llevarse a cabo para obtener la solución deseada.

- **Tercero. -**Verificar si la salida de datos es la deseada y restricciones a la solución si existieran.

Diseño del Algoritmo.

Un algoritmo es un número finito de pasos que se utilizan para resolver un problema, estos algoritmos deben ser precisos, indicar el orden de las instrucciones, hay que evitar toda ambigüedad.

Ejemplos de Algoritmo:

1) Realizar un algoritmo que describa el cambio de una llanta de un vehículo.

1.-Levantar carro con el gato hidráulico

2.-Quitar tuercas de la llanta

3.-Quitar la llanta dañada.

4.-Poner llanta en buen estado.

5.-Apretar las tuercas.

6.-Bajar el carro del gato hidráulico.

En este ejercicio se describe el cambio de una llanta en mal estado, en pasos ordenados y secuenciales.

2)Realizar un algoritmo del sueldo de un empleado, este se calcula tomando en cuenta el número de horas trabajadas y se multiplica por el valor de la hora.

1.-Leer número de horas.

2.- Leer valor por hora.

3.-Calcular sueldo número de horas * valor por hora

4.-Visualizar sueldo

En este ejemplo de cálculo de sueldo de un empleado se tiene dos consideraciones importantes, la primera, leer número de horas, porque describe la instrucción que el computador solicita información al usuario y es ingresada por el teclado, la segunda es la última instrucción donde se indica la visualización del sueldo, esto se realiza porque una vez ingresado los valores

solicitados por el problema y procesados a través del cálculo del sueldo lo último es que el computador visualice el resultado a través de algún dispositivo de salida por ejemplo la pantalla.

3)Algoritmo para realizar la suma de dos números.

1.- Leer primer número

2.- Leer segundo número.

3.-Sumar primer número + segundo número

4.-Visualizar suma

2.4 PSEUDOCÓDIGO.

Es llamado falso lenguaje porque no se puede implementar en un computador por tanto es usado para la lectura humana, este omite muchos detalles de los lenguajes de programación no tiene una codificación definida pero este texto con fines didácticos lo establece en este capítulo que se ha estudiado, la mayoría de los componentes de este pseudocódigo son: tipos de datos, operadores aritméticos, lógicos y relacionales.

El pseudocódigo está formado por:

Cabecera. - En esta sección se asigna el nombre del pseudocódigo, que por lo general se asigna un nombre descriptivo del problema y precedido por la palabra ***algoritmo*** ejemplo: algoritmo suma.

Declaración de variables. - En esta sección se sitúan todas las variables utilizadas, donde se almacenan los datos de entrada y salida del pseudocódigo, para identificar esta sección se colocará la palabra ***var,*** por ejemplo:

var

entero: valor

real: resultado

real: salario

En esta misma sección se puede realizar la declaración de contantes estas irán precedidas por la palabra ***const,*** por ejemplo:

const

real costo por hora←10

entera iva←14

cadena mensaje ← "hola"

Cuerpo. En esta sección comienza con la palabra ***inicio,*** esta referencia indica que se inicia con los procesos para solucionar el problema, una vez terminada esta parte se debe indicar que es el ***fin*** de nuestro pseudocódigo, y este quedaría de la siguiente forma:

algoritmo súma_numeros_enteros

var

entero: n1, n2, resultado

inicio

leer n1

leer n2

resultado← n1+n2

escribir ('La suma es:')

escribir(resultado)

fin

2.4.1 Ejercicios de Pseudocódigo.

Para este ejercicio tomaremos las partes de análisis del problema y su respectivo algoritmo para finalizar con la creación del pseudocódigo.

Ejercicio 1:

Enunciado

Realizar un pseudocódigo del sueldo de un empleado, este se calcula tomando en cuenta el número de horas trabajadas y se multiplica por el valor por hora.

Análisis del problema

1. **Identificar datos de entrada. -** Para este problema es necesario tener dos variables una para las horas trabajadas y otra para el valor por hora.
2. **Procesos. -** El proceso que se desarrolla es la multiplicación de las horas trabajadas por el

costo por hora.

3. **Salida. -** La salida que se requiere en el problema es el salario de un empleado

Algoritmo

1.-leer número de horas.

2.-leer valor por horas.

3.-calcular sueldo número de horas * valor por hora

4.-visualizar sueldo

Pseudocódigo. Desarrollo	**Posible Salida en Pantalla**
algoritmo cálculo_sueldo_empleado **var** entero: horas real: hora_valor, salario **inicio** leer horas; leer hora_valor; salario← hora_valor*horas escribir ('El salario es:') escribir(salario) **fin**	Salario del empleado

Ejercicio 2:

Enunciado

Realizar un pseudocódigo que permita calcular el área de un triángulo y su fórmula es: base por altura divido para 2.

Análisis del problema

1. **Identificar datos de entrada. -** Para este problema es necesario tener dos variables de tipo entero que permita almacenar la base y la altura del triángulo.
2. **Procesos. -** El proceso que se desarrolla es el de multiplicar la base del triángulo por su altura para luego dividirla entre 2.
3. **Salida. -** La salida que se requiere en el problema es el área del triángulo.

Algoritmo

1.-leer la base del triángulo.

2.- leer la altura del triángulo.

3.-calcular el área (base del triángulo * altura de triangulo) divido entre 2

4.-visualizar área

Pseudocódigo. Desarrollo	**Posible Salida en Pantalla**
algoritmo área_triangulo **var** real: base, altura, área **inicio** leer base; leer altura; area← (base*altura) div 2 escribir ('El área del triángulo es:')	Área del triangulo

escribir(área)

fin

Ejercicio 3:

Enunciado

Realizar un pseudocódigo que permita calcular el área de un círculo su fórmula de cálculo es: pi*radio^2, siendo pi un valor constante de 3.1416 y el radio del circulo elevado al cuadrado.

Análisis del problema

1. **Identificar datos de entrada. -** Para este problema es necesario tener una variable para el radio de la circunferencia y declarar una constante llamada pi que tendrá el valor de 3.14.
2. **Procesos. -** El proceso que se desarrolla es el de multiplicar la constante pi con el valor del radio del círculo.
3. **Salida. -** La salida que se requiere en el problema es el área del círculo.

Algoritmo

1.-leer el radio del círculo.

2.- constante_pi es igual a 3.14

3.-calcular el área (constante_pi *radio*radio)

4.-visualizar área

Pseudocódigo. Desarrollo	**Posible Salida en Pantalla**
algoritmo área_circulo **var** real: radio, área real pi=3.14 **inicio** leer radio; area← (pi*radio^2) escribir (‘El área del circulo es:’) escribir(área) **fin**	Radio del circulo

CAPÍTULO III

CONTENIDO

- **Toma de Decisiones o Condicionales**
- **Switch o Interruptor**
- **Sumadores o Acumuladores**

Introducción

Este capítulo enfoca los conceptos básicos de los controladores lógicos como herramienta fundamental y se estudiará la aplicación de toma de decisiones o condicionales el switch o interruptor y sus relaciones en las estructuras básicas para la programación estructural.

3.1. TOMA DE DESICIÓN O CONDICIÓN.

Es utilizada cuando los algoritmos que se requieren desarrollar tienen una descripción más complicada que una lista simple de instrucciones, es decir cuando existe un número de posibles alternativas resultantes de la evaluación de una condición. En esta toma de decisión o condición se evalúa y con el resultado obtenido (verdadero o falso) se realiza una o varias instrucciones**.**

La representación de una estructura condicional se realiza con las palabras en pseudocódigo (si- entonces, sino) y pueden ser simples, dobles y múltiples.

Alternativa simple.

Esta estructura simple si-entonces ejecuta una o varias acciones, cuando se cumple la condición y cuando es falsa no se realiza ninguna acción.

Su sintaxis es la siguiente:

Si (expresión lógica)

Inicio

<instrucciones>

Fin sin

Alternativa Doble.

La alternativa doble es la que tiene más de una opción.

Su sintaxis es la siguiente:

Si<condición>entonces

<Instrucciones>

Si_no

<Instrucciones>

Fin_si

En la sintaxis de esta alternativa se observa las instrucciones luego del entones se realizan si la condición tiene una repuesta afirmativa, en caso contrario se pueden realizar otro bloque de instrucciones. Sin embargo, se pueden realizar en la sección de si_no otras alternativas con más instrucciones que se realicen, en los ejercicios que a continuación se detallan estarán incluidos este tipo de problemas.

3.1.1 Ejercicios de Toma de Decisión o Condición

Ejercicio 1

Enunciado

Realizar un Pseudocódigo que permita verificar el mayor de 2 números enteros

Análisis del problema

1. **Identificar datos de entrada. -** Para este problema es necesario tener dos variables una para cada número entero
2. **Procesos. -** El proceso que se desarrolla es una condición que permite verificar cuál es mayor de los dos números ingresados.
3. **Salida. -** Número mayor

Algoritmo

1.-leer primer número.

2.- leer segundo número.

3.-Si primer número es mayor que segundo número

visualizar primer número

Si_no

visualizar segundo número

Pseudocódigo.

Desarrollo **Posible Salida en Pantalla**

Algoritmo mayor _de_dos _números **var** entero: num1, num2 **inicio** leer num1; leer num2; Si (num1>num2) entonces escribir ('El número mayor es:',num1) Sino escribir ('El número mayor es:',num2) **fin**	El mayor de dos números

Ejercicio 2

Enunciado

Realizar un pseudocódigo que permita verificar si un número es positivo o negativo.

Análisis del problema

1. **Identificar datos de entrada. -** Para este problema es necesario tener una variable de tipo entero.
2. **Procesos. -** El proceso que se desarrolla es una condición que permita verificar si un número es mayor o menor que cero y con esto lograremos identificar si es un número entero positivo o negativo
3. **Salida. -** tipo de numero positivo o negativo

Algoritmo

1.-leer número

2.-Si número es mayo que 0

visualizar el número es positivo

Si_no

visualizar el número es negativo.

Pseudocódigo. Desarrollo	**Posible Salida en Pantalla**
algoritmo número_positivo_o_negativo **var** entero: número; **inicio** leer número; Si (número>0) entonces escribir ('El número es positivo'); Si_no escribir ('El número es negativo'); **fin**	El número es positivo o negativo

Ejercicio 3

Enunciado

Realizar un Pseudocódigo que permita verificar si un número entero ingresado es par o impar.

Análisis del problema

1. **Identificar datos de entrada. -** Para este problema es necesario tener una variable de tipo entero.
2. **Procesos. -** El proceso que se desarrolla es una condición que permita verificar si un número es par o impar, para esto se debe calcular el residuo del número entre dos, y utilizar la condición para preguntar si el residuo de la división entre dos es igual a 0 entonces es

par y si es igual a uno entonces es impar.

3. **Salida. -** tipo de número par o impar.

Algoritmo

1.-leer número
2.-Si número módulo 2 es igual a 0
visualizar el número es par
Si_no
visualizar el número es impar.

Pseudocódigo. Desarrollo	**Posible Salida en Pantalla**
algoritmo número_par_o_impar **var** entero: número; **inicio** leer número; Si (número%2=0) entonces escribir ('El número es par'); Si_no escribir ('El número es impar'); **fin**	El número es par o impar

Hasta este momento se ha combinado en la condición con operadores relacionales, operadores de asignación, comparación y también se puede utilizar operadores lógicos que permiten evaluar expresiones que se necesitan en ejercicios con condición.

Ejercicio 1

Enunciado

Realizar un pseudocódigo para determinar cuánto se debe pagar por una cantidad de lápices considerando que si son más de 500 el costo por lápiz es de 0.85 en caso contrario el costo es de 1 dólar.

Análisis del problema

1. **Identificar datos de entrada. -** Para este problema es necesario tener una variable de tipo entero para la cantidad de lápices.
2. **Procesos. -** El proceso que se desarrolla es una condición que permita verificar si la cantidad es mayor que 500 lápices se debe calcular el costo total que será igual al número de lápices por 0.85, en caso contrario será igual al número de lápices por 1dolar.
3. **Salida. -** costo total de los lápices

Algoritmo

1.-leer cantidad lápices

2.-Si cantidad lápices mayor que 500

Costo total es igual a cantidad lápices por 0.85

visualizar costo total

Sino

Costo total es igual a cantidad lápices por 1.00

visualizar costo total

Pseudocódigo.

Desarrollo **Posible Salida en Pantalla**

```
algoritmo costo_total_lapices
var
entero: cantidad_lapices;
real: costo_total;
inicio
leer cantidad_lapices;
Si (cantidad_lapices >500) entonces
 costo_total=cantidad_lapices*0.85;
escribir ('el costo total es:', costo_total);
Si_no
costo_total=cantidad_lapices*1.00;
escribir ("el costo total es:',costo_total);
fin_si
fin
```

El costo total de los lapices

Ejercicio 2

Enunciado

Realizar un Pseudocódigo para determinar el costo y el descuento de un producto. Para este proceso se debe considerar que si el precio es mayor o igual a 300 se aplica un descuento de 20%; si su precio es mayor o igual a 100 y menor que 300, el descuento es del 15%, y si es menor a 100 el descuento es de 10%.

Análisis del problema

1. **Identificar datos de entrada. -** Para este problema es necesario tener una variable de tipo real para el valor del producto.
2. **Procesos. -** El proceso que se desarrolla es una condición que permita verificar el costo del producto si es mayor o igual que 300 se le restara al producto el 20%, si el producto está entre mayor igual que 100 y menor que 300 al costo del producto se le resta 15% y si el costo del producto es menor a 100 entonces se

deberá restar el 10 % del costo total.

3. **Salida.-** costo total del producto

Algoritmo

1.-leer costo producto

2.-Si costo producto mayor igual que 300

Costo total es igual a costo producto menos ((costo producto por 20) dividido por 100)

visualizar costo total

Si no

Si (costo producto es menor que 300) y (costo producto mayor igual que 100) entonces

Costo total igual a costo producto menos ((costo producto por 15) dividido por 100)

visualizar costo total

Si no

Costo total es igual a costo producto menos ((costo producto por 10) dividido por 100)

visualizar costo total

Pseudocódigo.
Desarrollo

Posible Salida en Pantalla

```
algoritmo producto_descuento
var
real:costo_producto;
inicio
Si (costo_producto >=300) entonces
   costo_total=costo_producto -
(costo_producto*20)/100);
   escribir costo_total;
    Si_no

Si(costo_producto<300)and(costo_producto
>=100) entonces
    costo_total=costo_producto -
((costo_producto*15)/100);
    escribir costo_total;
  Si_no
   costo_total=costo producto –
((costo_producto*10/100);
    escribir costo_total;
    fin_si
fin_si
fin
```

El costo total del producto con el descuento

Ejercicio 3

Enunciado

Realizar un pseudocódigo para determinar el salario de una persona semanal teniendo las horas trabajadas y el pago por hora, considerando que después de las cuarenta horas se paga el doble.

Análisis del problema

1. **Identificar datos de entrada. -** Para este problema es

necesario tener dos variables de tipo real para las horas trabajadas y el pago por hora

2. **Procesos.-** El proceso que se desarrollará es una alternativa que permita verificar el número de horas trabajadas si es mayor a cuarenta se deberán pagar al doble las excedentes es decir si son 48 solo las 8 se deberán pagar el doble.
3. **Salida.-** salario del empleado

Algoritmo

1.-leer horas_trabajadas

2.-leer valor_hora

3.-Si horas_trabajadas mayor que 40

valor_horas_doble es igual (horas_trabajadas por valor_hora)

salario_total es igual (valor_hora por 40)más valor_horas_doble

escribir('El salario total es:',salario_total)

Si_no

salario_total es igual horas_trabajadas por valor_hora

escribir('El salario total es:', salario_total)

Pseudocódigo. Desarrollo	**Posible Salida en Pantalla**
algoritmo salario_empleado **var** real: horas_trabajadas,valor_hora; **inicio** leer horas_trabajadas; leer valor_hora; Si (horas_trabjadas > 40) Entonces	El salario total del empleado

```
   valor_horas_doble=(horas_trabajada -
40)*valor_hora;

salario_total=(valor_hora*40)+valor_horas_
doble;
     escribir('El salario total es:',salario_total);
    Si_no

salario_total=horas_trabajadas*valor_hora;
       escribir('El salario total es:',
salario_total);
fin
```

3.2. INTERRUPTOR O SWITCH

El interruptor o switch es una estructura que permite seleccionar entre varias posibilidades. Este tipo de estructura es una variación de la **condicional de selección anidada.** La estructura interruptora evalúa una expresión que puede tomar diferentes valores depende del valor que coincida, se ejecutan las instrucciones que se encuentran dentro del cuerpo de la opción. Al terminar las instrucciones de la opción se debe finalizar mediante la palabra reservada salir (*break*), lo que permite que salte al final de la estructura.

Si ninguna opción cumple con la expresión que se evalúa, es necesario definir una opción por omisión, que también puede tener instrucciones, esta opción hace referencia al ***caso contrario de las estructuras condicionales dobles y anidadas***, que para la estructura interruptor se la define al caso contrario como (*default)* que es opcional en este tipo de estructuras.

La expresión sólo permite valores enteros o caracteres para validar con la opción y su operador de relación predefinido es el igual.

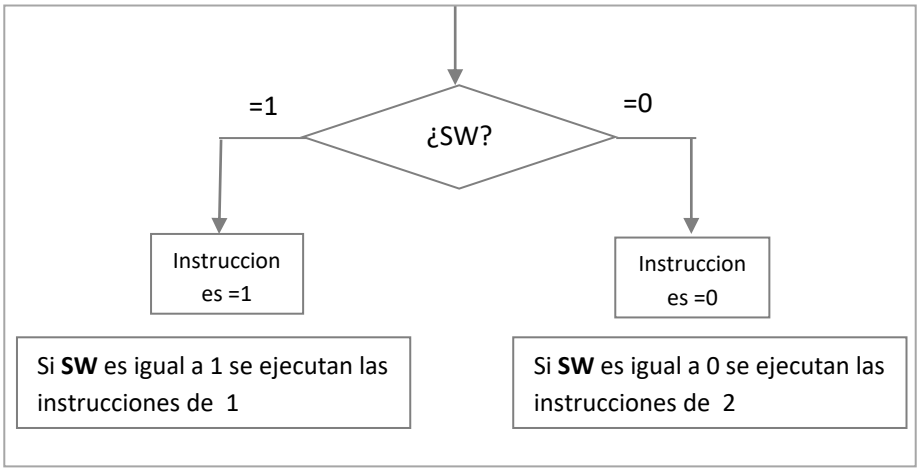


Figura 3.- Diagrama de una Estructura Condicional Interruptor
Fuente: desarrollada por los autores

Se puede mencionar algunas características o diferencias entre la el Interruptor y la estructura condicional Si.

- El interruptor sólo evalúa una expresión por igualdad, mientras que la condicional si puede evaluar una expresión por cualquier método relacional.
- El interruptor no puede tener dos constantes con idénticos valores en las opciones.

Sintaxis:

```
Según [Condición] Hacer
    Caso , Evaluar Valor1:
        Instrucción 1
            .
            .
        Instrucción n
    Salir
```

```
Caso , Evaluar Valor2:
        Instrucción 1
                .
                .
        Instrucción n
    Salir
                .
                .
Caso , Evaluar Valor N:
        Instrucción 1
                .
                .
        Instrucción n
    Salir
Caso, Si No
            Instrucción 1
                .
                .
        Instrucción n
    Salir
Fin Según
```

Un caso específico donde se implementa la estructura interruptor es en los menús de opciones.

Puede darse el caso que dentro de la estructura de un interruptor exista otro interruptor a esto se lo conoce como interruptores anidados.

3.2.1 Ejercicios de Pseudocódigo

Ejercicio 1:

Enunciado

Se requiere que una persona digite un número, evalúe si la entrada corresponde a un día de la semana; en caso de que así sea, muestre el nombre del día que le corresponde.

Análisis del problema

1. **Identificar datos de entrada. -** Para este problema es necesario una variable de tipo entero que almacene el número del día a comparar.
2. **Proceso. –** Se compara el dato ingresado si pertenece al número correspondiente de un día de la semana, caso contrario rechazara los datos ingresados.
3. **Salida. –** Muestra el día de acuerdo al número ingresado.

Algoritmo

1. Inicio
2. Leer ndia
3. Según ndia Hacer
4. Caso, igual a 1
5. Mostrar "Lunes"
6. Salir
7. Caso, igual a 2
8. Mostrar "Martes"
9. Salir
10. Caso, igual a 3
11. Mostrar "Miércoles"
12. Salir
13. Caso, igual a 4
14. Mostrar "Jueves"
15. Salir
16. Caso, igual a 5
17. Mostrar "Viernes"
18. Salir
19. Caso, igual a 6
20. Mostrar "Sábado"
21. Salir
22. Caso, igual a 7
23. Mostrar "Domingo"
24. Salir
25. Caso, SiNo
26. Mostrar "El número proporcionado no representa ningún día de la semana"
27. Salir
28. Fin Según
29. Fin

Pseudocódigo.
Desarrollo

```
algoritmo día_de_la_semana
 var
 entero: ndia
inicio
 escribir(" Introduzca el número del día ")
 leer ndia
 Según ndia Hacer
   Caso  1    escribir(" Lunes ")

   Caso   2    escribir(" Martes ")

   Caso   3    escribir(" Miércoles ")

   Caso  4    escribir(" Jueves ")

   Caso  5    escribir(" Viernes ")

   Caso  6    escribir(" Sábado ")

    Caso  7    escribir(" Domingo ")

    Caso  SiNo

      escribir( " El número proporcionado no representa ningún día de la semana ")
finSegun
fin
```

Posible Salida en Pantalla

El posible resultado será el nombre del día de la semana que corresponda al número digitado por el usuario

Ejercicio 2:

Enunciado

Se requiere realizar un menú básico para un restaurante con los platos que se servirán en el desayuno. Los

comensales podrán seleccionar un plato y se les mostrará el precio que tiene ese plato.

Análisis del problema

1. **Identificar datos de entrada. -** Para este problema es necesario que el cliente seleccione un ítem del menú y presenta el precio que tiene.
2. **Proceso. –** Se comprobara si número seleccionado corresponde a un Ítem del menú de ser verdadero mostrará su precio, caso contrario rechazará los datos ingresados.
3. **Salida. –** Muestra el precio del plato seleccionado por el cliente.

Algoritmo

1. Inicio
2. escribir "- Menú - "
3. escribir "- 1 Bolón Mixto - "
4. escribir "- 2 Bolón con Queso - "
5. escribir "- 3 Bolón con Chicharrón - "
6. escribir "- 4 Taza de Café - "
7. escribir "- 5 Salir - "
8. escribir " Seleccione una Opción"
9. Leer opc
10. Según opc Hacer
11. Caso, igual a 1
12. escribir "Precio del Bolón Mixto es: $ 1,75 "
13. salir
14. Caso, igual a 2
15. escribir "Precio del Bolón con Queso es: $ 1,25 "
16. Salir
17. Caso, igual a 3
18. escribir "Precio del Bolón con Chicharrón es: $ 1,25 "
19. salir
20. Caso, igual a 4
21. Imprimir "Precio de la Taza de café es: $ 0,50 "
22. salir
23. Caso, igual a 5
24. escribir "Usted ha salido del Menú "
25. salir

26. Caso, SiNo
27. escribir "La opción no existe en el menú"
28. salir
29. Fin Según
30. Fin

Pseudocódigo.
Desarrollo

```
algoritmo Menú_Restaurat
 var
 entero: opc
inicio
     escribir ("- Menú  - ")
     escribir ("- 1 Bolón Mixto  - ")
     escribir ("- 2 Bolón con Queso  - ")
     escribir ("- 3 Bolón con Chicharrón
     - ")
     escribir ("- 4 Taza de Café  - ")
     escribir ("- 5 Salir  - " )
     escribir (" Seleccione una Opción")
 leer opc
 Según opc Hacer

Caso 1
  escribir(" Precio del Bolón Mixto es: $
1,75")

 Caso 2
  escribir(" Precio del Bolón con Queso
es: $ 1,25 ")

 Caso 3
  escribir(" Precio del Bolón con
Chicharrón es: $ 1,25 ")

 Caso 4
  escribir(" Precio de la Taza de café es:
$ 0,50")

Caso 5
escribir(" Usted ha salido del Menú ")
```

Posible Salida en Pantalla

El posible resultado será el precio del Plato seleccionado por el cliente.

```
Caso SiNo
escribir( " La opción no existe en el
menú ")

finSegun
fin
```

Ejercicio 3:

Enunciado

Se necesita verificar si un usuario a digitado una vocal por medio del teclado.

Análisis del problema

1. **Identificar datos de entrada. -** Para este problema es necesario una variable de tipo carácter para almacenar el valor digitado por el usuario.
2. **Proceso. –** Se comprueba si el carácter digitado por el usuario corresponde a una vocal, caso contrario envía un mensaje indicando que el carácter ingresado no es una vocal.
3. **Salida. –** Muestra la vocal digitada por el usuario.

Algoritmo

```
Inicio
1. Leer carácter
2. Según carácter  Hacer
3. Caso, igual a "a"
4. escribir "Es una vocal "
5. salir
6. Caso, igual a "e"
7. escribir  "Es una vocal "
8. Salir
9. Caso, igual a "i"
10. escribir  "Es una vocal "
11. salir
12. Caso, igual a "o"
13. escribir  "Es una vocal "
14. salir
15. Caso, igual a "u"
16. escribir  "Es una vocal "
17. salir
```

18. Caso, SiNo
19. escribir "La tecla digitada no es una Vocal"
20. salir
21. Fin Según
22. Fin

Pseudocódigo.
Desarrollo

```
algoritmo Vocal
 var
 entero: Tecla
inicio
 escribir(“  Digite un caracter ”)
 leer Tecla
 Según Tecla Hacer
   Caso  “a”    escribir(“  Es una vocal ”)

   Caso “e”    escribir(“  Es una vocal ”)

   Caso  “i”    escribir(“  Es una vocal ”)

   Caso “o”   escribir(“  Es una vocal ”)

   Caso “u”    escribir(“  Es una vocal ”)

    Caso SiNo

      escribir( “ La tecla digitada no es
una Vocal ”)
finSegun
fin
```

Posible Salida en Pantalla

El posible resultado será un mensaje indicando que se presionó una vocal.

CAPÍTULO IV

CONTENIDO

- **Bucles**
- **Sumadores o acumuladores**
- **Arreglos**
- **Algoritmos de búsqueda**

Introducción

Este capítulo introduce al lector, en los tipos de estructuras Cíclicas (Bucles) que pueden ser utilizados en los lenguajes de programación, además de identificar la estructura que le permita resolver un problema en el menor tiempo posible para posteriormente trasladarlo a un lenguaje de programación.

En lo que respecta a los arreglos se detalla su estructura y funcionamiento y como permiten optimizar recursos, a través de su implementación.

4.1. BUCLES

Los bucles son estructuras de repetición, que permiten que una o varias instrucciones se ejecuten un determinado número de veces o en ocasiones indefinidamente. Como su nombre lo indica (Ciclo, Bucles) estos segmentos de algoritmos o programas en cada iteración, comprueban si la condición es verdadera, de cumplirse, genera otro ciclo, este proceso se romperá cuando la condición devuelva falsa, de no ser así el bucle se hará infinito. Los ordenadores están diseñados para ejecutar aplicaciones en las que se puede ejecutar una operación o conjunto de operaciones más de una vez.

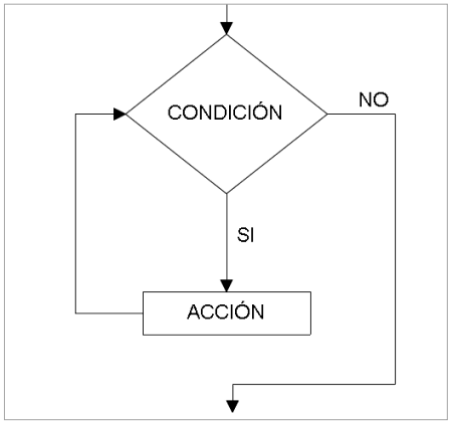


Figura 4.- Diagrama de una Estructura Cíclica
Fuente: desarrollada por los autores

Estas estructuras permiten la simplificación de los algoritmos, ahorrando tiempo para intentar resolver problemas mediante un ordenador. Los Bucles según su característica pueden variar en tres estructuras:

- Mientras - Hacer
- Hacer - Mientras
- Desde

Ejemplo: Como se mencionó, un Bucle permite ejecutar una o varias acciones, un determinado número de veces.
Se plantea lo siguiente: "Juan camina en el parque la Carolina Hasta que su cronometro marque las 8:00 am.".
Si analizamos el enunciado anterior tenemos una estructura cíclica que ejecuta la acción de caminar.
Esta acción se ejecutará mientras la condición Cronometro no marque las 8:00 am.

Las estructuras cíclicas en ocasiones requieren saber con anterioridad el número de veces que se va a repetir. Una forma de hacerlo es utilizar contadores, acumuladores, centinelas y banderas.

4.1.1 Contadores, acumuladores, centinelas y banderas

Contadores y acumuladores

Es muy común en escuchar estas dos palabras en los desarrolladores de software, debido a que se utilizan con mucha frecuencia en el desarrollo de algoritmos, estos términos hacen referencia a variables que se incrementa su valor luego de realizar un proceso dentro de un algoritmo o programa. Por lo general son de tipo numérico.

Contador	Acumulador
cont ← cont + 1	**sum ← sum + b**
i ← i + 2	**total ← total + x**
p ← p – 1	**result ← result - y**

Tabla 1.- Contadores y Acumuladores
Fuente: desarrollada por los autores

Entre las características que podemos mencionar de los contadores y acumuladores son:

- Deben ser inicializados antes de ser usados dentro de un bucle.
- Dentro del bucle deben aparecer tanto a la izquierda como a la derecha de la asignación (izq. ← der).

Antes de realizar cualquier operación con estas variables es necesario saber que para realizar sumas o restas, estas deben ser inicializadas en cero, debido que el cero es el elemento neutro de las operaciones básicas. En cuanto a la multiplicación su inicialización se debe dar en 1 ya que si se asigna cero el resultado de las operaciones con esta variable siempre dará cero.

Contador

El contador es una variable que se utiliza como incremento o decremento, dependiendo del caso en el que se esté aplicando. Es utilizado para controlar los bucles, esta variable (contador) determina las veces que se realiza una o varias acciones en el bucle.

Sintaxis:

nombre_contador ← nombre_contador **+** variable_constante

La sintaxis hace referencia a una variable denominada contador a la cual se le asigna el valor de la misma variable contador más un valor constante, esta acción permite realizar un incremento controlado.

Ejemplo

entero cont ← 0	// En esta línea de código se crea la variable cont y se inicializa con el valor neutro cero.
cont ←cont + 2	// En esta línea de código se incrementa en dos el valor de la variable cont.

Si se desea realizar un decremento de la variable **cont** se remplaza el signo más (+) por el signo menos (-).

Acumulador

El acumulador también conocido como totalizador realiza la misma acción que el contador con la diferencia que el valor de incremento o decremento es variable en lugar de ser constante. También se puede decir que es una variable que acumula el valor de una expresión matemática.

Sintaxis:

nombre_acumulador ← nombre_ acumulador + valor_variable

La sintaxis hace referencia a una variable denominada acumulador la cual se le asigna el valor de la misma variable acumulador más un valor que puede variar.

Ejemplo:

entero acum ← 0	// En esta línea de código se crea la variable acum y se inicializa con el valor neutro cero.
acum ←acum + x	// En esta línea de código se incrementa el valor de la variable acum dependiendo del valor que tenga la variable x.

Centinela

Un centinela también conocido como interruptor, es una variable que permite al bucle tener la opción de iniciar y terminar. Esta variable inicia con un valor que es evaluado por el bucle si retorna verdadero, le permite realizar las instrucciones que se encuentran dentro del mismo. En el bucle esta variable puede cambiar su valor, dando como resultado falso en la evaluación del bucle y terminarlo.

Entre sus características se destaca al centinela dentro de una estructura cíclica que no se conoce el número de repeticiones que tendrá.

Ejemplo:

Se le solicita a un docente que digite "S" o "N" para indicar si se registra un estudiante en su asignatura. Este proceso terminará cuando el docente digite "N".

Bandera

La bandera es una variable que puede almacenar solo dos valores, verdadero y falso o 1 y 0.
Es utilizada en los bucles para la validación de la condición. Su valor cambia dentro del ciclo, lo que permite que el bucle pueda terminar.

Ejemplo:

entero band ← 0	// En esta línea de código se crea la variable band y se inicializa con el valor neutro cero (falso).
band ←1	// En esta línea de código cambia el valor de la variable band por 1 (verdadero)

4.1.2 Estructura repetitiva Mientras - Hacer

Este tipo de bucle permite evaluar la condición del ciclo al inicio, si el resultado que arroja la expresión es verdadero el bucle sigue ejecutando las instrucciones que se encuentran después la expresión es evaluada nuevamente y se procede de la misma manera. Cuando el resultado de la expresión da falso en la siguiente evaluación, se da por finalizado el bucle.

Si por contrario la evaluación de la expresión da falso al inicio del bucle, este no realiza ninguna acción y se da por terminado el bucle; si la expresión nunca devuelve falso, el bucle no terminará, lo que podrá generar un error dependiendo de las instrucciones que se encuentren dentro de él, ocasionando la finalización del programa o que se quede iterando indefinidamente hasta que se lo detenga de forma manual.

En este tipo de estructura no se conoce con anterioridad el número de veces que se ejecutarán las instrucciones dentro del bucle, ya que esto va a depender de la expresión evaluada.

Sintaxis:

Mientras [***Condición***] **Hacer**
Instrucción 1
Instrucción 2
.
.
.
Instrucción n
Fin_Mientras(Repetir)

4.1.2.1 Ejercicios de Pseudocódigo

Ejercicio 1:

Enunciado
La empresa "XY", requiere presentar al operador de máquina los10 primeros números enteros como guías de los embarques que debe realizar.

Análisis del problema

1. **Identificar datos de entrada.-** Para este problema es necesario tener una variable contador la que se inicializa con el valor de 1; debido a que permitirá iniciar la secuencia de los números enteros.
2. **Procesos.-** El proceso que se desarrollará es incrementar la variable contador mientras sea

menor a 11.

3. **Salida.-** La salida que se requiere en el problema son los diez primeros números enteros.

Algoritmo

1. Inicio
2. Leer n
3. Asignar el valor de 1 a la variable n
4. **Mientras** la variable n sea menor a 11 H**acer**(*Si n es igual o mayor a 11 ir al paso 8*)
5. escribir el valor de n
6. Incrementar el valor de n en 1
7. Regresar al paso 4
8. Finalizar

Pseudocódigo.

Desarrollo	**Posible Salida en Pantalla**
algoritmo listar_numeros **var** entero : n **inicio** n ← 1 **mientras** n < 11 **hacer** escribir(n) n ← n + 1 **fin_mientras** **fin**	1 2 3 4 5 6 7 8 9 10

Ejercicio 2:

Enunciado

En la escuela "San José", la profesora de matemática le solicita a sus estudiantes listen los números del 40 al 50 de manera descendente.

Análisis del problema

1. **Identificar datos de entrada.-** Para este problema es necesario tener una variable contador la que se inicializa con el valor de 50; debido a que permitirá iniciar la secuencia de los números enteros de manera descendente.
2. **Procesos**.- El proceso que se desarrollará es el decremento de la variable contador mientras sea mayor a 39.
3. **Salida.-** La salida que se requiere en el problema son los diez números enteros en forma descendente en el rango de 50 al 40.

Algoritmo

1. Inicio
2. Leer n
3. n es igual a 50
4. **Mientras** la variable n sea mayor a 39 **Hacer**(*Si n es igual o menor a 39 ir al paso 8*)
5. escribir n
6. decrementar el valor de n en 1
7. Regresar al paso 4
8. Finalizar

Pseudocódigo.

Desarrollo **Posible Salida en Pantalla**

```
algoritmo listar_números2
 var
    entero : n
 inicio
   n ← 50
   mientras  n  > 39 hacer
       escribir(n)
       n ← n - 1
   fin_mientras
fin
```

```
50
49
48
47
46
45
44
43
42
41
40
```

Ejercicio 3:

Enunciado

Ayuda a Pedro a resolver un problema que le enviaron en el colegio. Dice lo siguiente: Debe realizar un algoritmo que le permita ingresar números enteros positivos e irlos sumando a medida que son ingresados. Esta operación termina si Pedro ingresa un número negativo.

Análisis del problema

1. **Identificar datos de entrada.-** Para este problema es necesario tener dos variables de tipo entero; la primera permitirá guardar cada valor ingresado por el usuario y la segunda permitirá almacenar la suma de los números ingresados.
2. **Procesos**.- El proceso que se desarrollará es solicitar al usuario que ingrese números enteros positivos hasta que ingrese un número negativo, a su vez deberá ir sumando los números ingresados uno a la vez.
3. **Salida.-** La salida que se requiere en el problema es la suma de los números enteros positivos ingresados.

Algoritmo

1. Inicio
2. Leer n

3. suma es igual a 0
4. **Mientras** la variable n sea un número entero positivo **Hacer**(*Si n es un número entero positivo ir al paso 9*)
5. Leer n
6. suma es igual a suma más n
7. Regresar al paso 4
8. Escribir suma.
9. Finalizar

Pseudocódigo. **Desarrollo**	**Posible Salida en Pantalla**
algoritmo sumar_números **var** entero : n, suma **inicio** suma ← 0 leer(n) **mientras** n > -1 **hacer** leer(n) suma ← suma +n **fin_mientras** escribir ('*La suma de los números ingresados es:* 'suma) **fin**	Para este caso el resultado dependerá de los valores ingresados por el usuario. Estos valores se irán sumando acorde se vayan ingresado y el resultado se acumula en la variable ***suma***

4.1.3 Estructura repetitiva Hacer – Mientras

Esta estructura primero realiza una o varias instrucciones y luego evalúa la condición al final de su estructura, si el resultado de la evaluación retorna

verdadero el bucle sigue ejecutando las instrucciones que se encuentran en él. Cuando el resultado de la expresión da falso en la siguiente evaluación, se da por finalizado el bucle.

Si por contrario la evaluación de la expresión da falso, por lo menos se ha ejecutado una vez las instrucciones, si la expresión nunca devuelve falso, el bucle no terminara. En este tipo de estructura no se conoce con anterioridad el número de veces que se ejecutarán las instrucciones dentro del bucle, ya que esto va a depender de la expresión evaluada.

Las variables que se evalúan en la condición pueden inicializarse antes de ingresar al bucle o durante ésta, ya que la ventaja de esta estructura es que evalúa la condición al final.

Es muy semejante al bucle mientras-hacer con la diferencia que la evaluación se realiza al final del cuerpo del ciclo. El bucle mientras-hacer y hacer-mientras son utilizados cuando no se conoce con anterioridad el número de veces que se desea realizar una o varias instrucciones.

Sintaxis:

```
Hacer
        Instrucción 1
        Instrucción 2
            .
            .
            .
        Instrucción n
Mientras [Condición]
```

El ejercicio que se plantea a continuación realiza la misma operación que el ejercicio 1 de la estructura Mientras-Hacer, pero utilizando la estructura Hacer-Mientras, de esta manera se podrá observar las diferencias que existen en ambas estructuras cíclicas.

4.1.3.1 Ejercicios de Pseudocódigo

Ejercicio 1:

Enunciado

Se solicita que se impriman los diez primeros números enteros positivos

Análisis del problema

1. **Identificar datos de entrada.-** Para este problema es necesario tener una variable (contador) de tipo entero que se inicializa en 1;
2. **Procesos**.- El proceso que se requiere es ir imprimiendo los valor que tiene la variable (contador) e ir incrementado su valor en 1 hasta que llegue a 10
3. **Salida.-** La salida que se requiere en el problema es la lista de los diez primeros números enteros positivos.

Algoritmo

1. Inicio
2. Leer n
3. n es igual a 1
4. **Hacer**
5. escribir n
6. Incrementar en 1 el valor de la variable n
7. **Mientras** n sea menor a 11, ir al paso 4
8. Finalizar

Pseudocódigo.

Desarrollo **Posible Salida en Pantalla**

```
algoritmo listar_números                1
 var                                    2
    entero : n                          3
inicio                                  4
   n ← 1                                5
   hacer                                6
        escribir(n)                     7
        n ← n + 1                       8
    mientras  n  <  11                  9
fin                                     10
```

Si comparamos este ejercicio con el expuesto en la estructura mientras hacer podemos encontrar algunas diferencias. La primera es que primero imprime el valor de la variable y luego lo incrementa; esto se debe a que debe presentarse el número 1 que tiene almacenado la variable n, porque si primero se incrementa y luego se imprime como en el ejercicio anterior presentaría primero el 2 y no el 1.

Otra diferencia es que la evaluación se realiza al final y no al inicio de la estructura.

Ejercicio 2:

Enunciado

Se requiere imprimir los números múltiplos de 5 que se encuentran desde el 1 al 50 y sumarlos.

Análisis del problema

1. **Identificar datos de entrada.-** Para este problema es necesario tener una variable (contador) de tipo entero que se inicializa en 0 y una variable sum inicializada en 0.
2. **Procesos**.- El proceso que se requiere es el de ir imprimiendo el valor que tiene la variable (contador), e ir incrementando su valor en 5, también se sumará el valor de la variable(contador) hasta que llegue a 50
3. **Salida.-** La salida que se requiere en el problema

es la impresión de los números múltiplos de 5 y el total de la suma.

Algoritmo

1. Inicio
2. Leer n
3. sum es igual a 0
4. **Hacer**
5. n es igual n más 5
6. sum es igual a sum más 5
7. escribir n
8. **Mientras** n sea menor igual a 50, ir al paso 4
9. Escribir sum
10. Finalizar

Pseudocódigo.

Desarrollo	**Posible Salida en Pantalla**
algoritmo sumar_múltiplos **var** entero : n,sum **inicio** n ← 0 sum ← 0 **hacer** n ← n + 5 sum ← sum + n escribir (n) **mientras** n < = 50 escribir (‘*La suma es:*’sum) **fin**	5 10 15 20 25 30 35 40 45 50 La suma es: 275

Ejercicio 3:

Enunciado

Desarrolle un algoritmo que permita calcular el factorial de un número entero positivo.

Nota: *El factorial de un número se obtiene multiplicando una serie*

de números que descienden por ejemplo el factorial de 4.
4! = 4 × 3 × 2 × 1 = 24

Análisis del problema

1. **Identificar datos de entrada.-** Para este problema es necesario tener una variable (acumulador) de tipo entero que se inicializa en 1 y una variable que reciba el valor ingresado por el usuario.
2. **Procesos**.- Se requiere ir multiplicando la variable acumulador por el número ingresado. El valor del número ingresado se decrementa en 1.
3. **Salida.-** La salida que se requiere en el problema es la impresión del factorial del número ingresado.

Algoritmo

1. Inicio
2. Leer n
3. fact es igual a uno
   Hacer
4. fact es igual fact por n
5. Decrementar en 1 el valor de la variable n
6. **Mientras** n sea mayor a cero, ir al paso 4
7. escribir fact
8. Finalizar

Pseudocódigo.
Desarrollo.

```
algoritmo factorial
 var
    entero :n,fact
inicio
   leer(n)
   fact ← 1
   hacer
        fact ← fact * n
        n  ←  n - 1

    mientras  n  > 0
 escribir("El factorial
```

Posible Salida en Pantalla

Se imprime por pantalla el valor de la variable **fact**, cuyo valor es el resultado de multiplicar todos los números desde el 1 y el número ingresado **n**.

```
es:", fact)
fin
```

4.1.4 Estructura repetitiva Desde

En este tipo de estructura repetitiva a diferencia de otras se conoce con anterioridad el número de veces que se ejecutará el bucle. Esta estructura se basa en la evaluación de una variable tipo contador, la que se incrementa por cada iteración ejecutada.

Sintaxis:

```
Desde( [Valor Inicial] , [Condición] , [Incremento])
                    Instrucción 1
                    Instrucción 2
                        .
                        .
                        .
                    Instrucción n
Fin
```

Los tres componentes del bucle **Desde** se detallan a continuación.

Valor Inicial: Este componente permite iniciar el ciclo, esta variable recibe el valor inicial.

Condición: Al igual que los anteriores bucles es necesario realizar una evaluación de la condición, la misma que permite ejecutar el ciclo o parar su ejecución dependiendo del valor que retorne (verdadero o falso).

Incremento: Este componente permite realizar el incremento al valor inicial del bucle, este valor se incrementará mientras la condición retorne verdadero.

Nota: *También se puede realizar decremento.*

4.1.4.1 Ejercicios de Pseudocódigo

Ejercicio 1:

Enunciado

Desarrollar un algoritmo que permita imprimir la secuencia de los números enteros positivos del 1 al 10

Análisis del problema

1. **Identificar datos de entrada.-** Para este problema es necesario tener una variable (contador) de tipo entero que se inicializa en 1.
2. **Procesos**.- Se requiere ir imprimiendo el valor de la variable contador, esta variable se incrementa en uno hasta que llegue a 10
3. **Salida.-** La salida que se requiere en el problema es la impresión los número desde el 1 hasta el 10

Algoritmo

1. Inicio
2. i es igual a 1
3. Si **i** es menor que 11 **hacer;** si no, ir al paso 6
4. Imprime el valor de **i**
5. Incrementa la variable **i** en 1, ir al paso 3
6. Finalizar

Pseudocódigo.

Desarrollo	**Posible Salida en Pantalla**
	1
algoritmo	2
números_enteros	3
var	4
entero : i	5
inicio	6
desde (i ← 1 , i < 11, i ← i + 1)	7
	8
escribir (i)	9
fin_desde	10
fin	

Ejercicio 2:

Enunciado

Desarrollar un Algoritmo que permita ingresar 10 números e imprima sólo los números pares

Análisis del problema

a. **Identificar datos de entrada.-** Para este problema es necesario tener una variable (contador) y una variable que almacene el número ingresado por el usuario.
b. **Procesos**.- Se requiere solicitar al usuario que ingrese un número. Si este número es divisible para dos entonces lo imprime. Esta acción se realiza mientras la variable (contador) sea menor que 10.
c. **Salida.-** La salida que se obtendrá será la impresión solo de los números pares ingresados por el usuario.

Algoritmo

1. Inicio
2. i es igual 0
3. Si **i** es menor que 10 **hacer;** si no, ir al paso 8
4. Leer **n**
5. **Si** el residuo de la división de **n** por dos es igual a cero hacer; si no, ir al paso 4
6. Escribe n
7. Incrementa la variable **i** en 1; ir al paso 3
8. Finalizar

Pseudocódigo.

Desarrollo

```
algoritmo
contar_números_pares
 var
    entero : i,n
inicio
   desde (i ← 0 , i < 10, i
← i + 1)
                leer ( n )
      si   n  mod  2 = 0
entonces
```

Posible Salida en Pantalla

El resultado de este ejercicio dependerá de los valores ingresados por el usuario.

Ejemplo:

3
8
22
.
.
.

escribir(n) **fin_si** **fin_desde** **fin**	. 4

Ejercicio 3:

Enunciado

Desarrollar un algoritmo de tal manera que permita visualizar los números impares comprendidos entre 1 y 10

Análisis del problema

1. **Identificar datos de entrada.-** Para este problema es necesario tener una variable (contador)
2. **Procesos**.- Se requiere imprimir los números impares comprendidos entre 1 y 10. Esta acción se realiza mientras la variable (contador) sea menor que 10. Y su incremento se realiza de dos en dos.
3. **Salida.-** La salida que se obtendrá será la impresión de los números impares comprendidos entre 1 y 10.

Algoritmo

1. Inicio
2. i es igual a 1
3. Si **i** es menor que 10 **hacer;** si no, ir al paso 6
4. Escribir el valor de **i**
5. Incrementa la variable **i** en 2; ir al paso 3
6. Finalizar

Pseudocódigo.

Desarrollo	**Posible Salida en Pantalla**

```
algoritmo números_impares
 var
    entero : i
inicio
desde (i ← 1 , i < 10, i ← i +
2)

          escribir(i)

   fin_desde
fin
```

```
1
3
5
7
9
```

4.2. ARREGLOS

También llamados vectores, son estructuras de datos que permiten almacenar diferentes valores, del mismo tipo, es decir, la información almacenada en ellos pertenece al mismo tipo de dato (numérico, alfabético, etc.).

Los arreglos son elementos ordenados por índices, cada elemento posee una ubicación determinada. Cuando se declara un arreglo, el ordenador reserva memoria para cada uno de sus elementos de manera continua donde la dirección de memoria más baja corresponde al primer elemento del arreglo.

Los datos que almacena el arreglo se los denomina elementos del arreglo. Y su posición se enumera consecutivamente 0, 1, 2, 3, 4, 5,6……n. Los arreglos en casi todos los lenguajes de programación en su primera posición se inicializa en cero.

Arreglo b tamaño 4

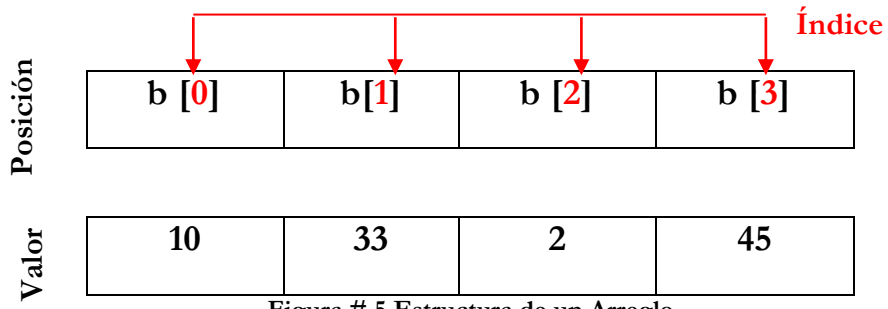


Figura # 5 Estructura de un Arreglo

Fuente: desarrollada por los autores

Como vemos en la figura 2, un programador puede definir un arreglo con un tamaño específico, para este caso tenemos un arreglo denominado B con un tamaño 4, esto permite al ordenador reservar espacio en memoria para cuatro valores de tipo numérico. Se debe tomar en cuenta que si se declara un arreglo con tamaño 50 y solo se utiliza 10 espacios del mismo, el ordenador desperdicia 40 espacios de memoria para valores numéricos.

Arreglo
entero a [50]
entero b[5], c[10]

Los arreglos se clasifican en:

Unidimensionales (Listas o Vectores)

Bidimensionales (Matrices o Tablas)

Multidimensionales (Más de dos dimensiones)

Los más utilizados son los unidimensionales y bidimensionales.

Ejemplo

Arreglo	Tipo	Espacio en Memoria
entero a [4]	Vector o Lista	4
entero b [3][3]	Matriz o Tabla	9
entero c [10]	Vector o Lista	10
entero d [2][3]	Matriz o Tabla	6

4.2.1 Arreglos Unidimensionales Listas o Vectores

Los arreglos unidimensionales o también conocidos como vectores tiene la característica de poseer una dimensión. Para poder acceder a cada uno de los elementos se requiere de un índice de tipo entero, el que representa la posición en que se encuentra.

Sintaxis	Pseudocódigo
	tipo_de_dato identificador _arreglo [tamaño] entero b [5]

Dónde:

- **tipo_de_dato** hace referencia al tipo de datos que el arreglo va almacenar, puede ser entero, carácter, real, etc.
- **identificador_arreglo** indica el nombre que se le asigna al arreglo.
- **tamaño** indica el espacio en memoria reservado para los elementos que tendrá el arreglo; la cantidad de elementos.

Arreglo a

Valor(Elemento)	**2**	**5**	**8**	**4**
Posición(Índice)	**0**	**1**	**2**	**3**

Si se desea acceder al valor de un elemento del arreglo mediante pseudocódigo:

imprimir(a [2])

La línea anterior imprime el valor **8**, esto se debe a que el arreglo **a** en la posición **2** tiene almacenado el número **8.**

Inicialización de Arreglos Unidimensionales

Cuando se define un arreglo, se pueden asignar valores, esto permite que el arreglo este lleno antes de realizar cualquier acción u operación.

Sintaxis

Pseudocódigo
tipo_de_dato identificador _arreglo [tamaño]←{valores} entero b [5]←{2,4,1,7,10}

La inicialización es realizada al crear el arreglo mediante el operador ← o =, los valores que se encuentran en las llaves {}, cada valor se ocupa una posición en el arreglo, no olvidar que su índice inicia en cero. Estos valores son separados por medio de la coma (,). Si el número de valores ingresados en las llaves es menor al tamaño del arreglo los espacios restantes se llenan con cero.

4.2.1.1 Ejercicios de Pseudocódigo

Ejercicio 1:

Enunciado

Desarrollar un algoritmo que permita llenar un arreglo con 10 números entero y luego visualizar la suma de todos los números ingresados.

Análisis del problema

a. **Identificar datos de entrada.-** Para este problema es necesario tener una variable (índice) , además definir un arreglo de tipo entero de tamaño 10 y una variable que almacene la suma de todos los elementos del arreglo.
b. **Procesos**.- El proceso que se requiere es de solicitar que se ingrese un número y almacenarlo en el arreglo, al mismo tiempo se lo suma con los demás números ya ingresados. Para lograr esto se requiere de una iteración (Bucles) que permita repetir este proceso diez veces.
c. **Salida.-** La salida que se obtendrá será la impresión de la suma de todos los números ingresados en el arreglo

Algoritmo

1. Inicio
2. Declarar el arreglo **a** de tipo entro tamaño **10**
3. i es igual a 0
4. suma es igual a 0

5. Si **i** es menor que 10 **hacer;** si no, ir al paso **12**
6. escribir "Ingrese un Número"
7. Leer un valor
8. Almacena el valor en el Arreglo **a** en la posición **i**
9. Suma es igual a suma más el valor del Arreglo **a** en la posición **i**
10. Incrementa la variable **i** en 1; ir al paso **6**
11. escribir el valor de **suma**
12. Finalizar

Pseudocódigo.

Desarrollo

```
algoritmo
suma_números_arreglos
 var
    entero : i, suma, a[10]
inicio
   desde (i ← 0 , i < 10, i
← i + 1)

          escribir("Ingrese un
valor")
          leer a[i]
          suma←suma+a[i]

   fin_desde
    escribir("La suma es ",
suma)
fin
```

Posible Salida en Pantalla

El posible resultado dependerá de los valores ingresados por el usuario.

Ejercicio 2:

Enunciado

Desarrollar un algoritmo que permita llenar un arreglo con 10 números enteros y luego visualizar sólo los números pares ingresados.

Análisis del problema

a. **Identificar datos de entrada.-** Para este problema es

necesario tener una variable (índice), además definir un arreglo de tipo entero de tamaño 10.

b. **Procesos**.- El proceso que se requiere es de solicitar que se ingrese un número y almacenarlo en el arreglo. Luego de haber llenado todo el arreglo con números enteros se requiere otra iteración que permitirá recorrer el arreglo, desde el primer elemento hasta el último durante este recorrido se requiere verificar cada elemento del arreglo, si es un número par si resulta verdadera esta comparación lo imprime por pantalla.

c. **Salida.-** La salida que se obtendrá será la impresión de todos los números pares ingresados en el arreglo; de no existir números pares el algoritmo no presenta nada.

Algoritmo

1. Inicio
2. i igual a 0
3. Si **i** es menor que 10 **hacer;** si no, ir al paso **10**
4. Imprime "Ingrese un Número"
5. Lee un valor del arreglo
6. Almacena el valor ingresado en el Arreglo **a** en la posición **i**
7. Incrementa la variable **i** en 1; ir al paso **3**
8. j igual a 0
9. Si **j** es menor que 10 **hacer;** si no, ir al paso **13**
10. Si **a**[**j**] divido para 2 es igual a 0 Entonces; si no ir al paso **12**
11. Imprimir **a**[**j**]
12. Incrementa la variable **j** en 1; ir al paso **9**
13. Finalizar

Pseudocódigo. **Desarrollo**	**Posible Salida en Pantalla**
algoritmo	El posible resultado serán

```
imprimir_números_pares_ar
reglo
 var
    entero : i, j, a[10]
inicio
   desde (i ← 0 , i < 10, i
← i + 1)

          escribir("Ingrese un
valor")
         leer a[i]

   fin_desde

   desde (j ← 0 , j < 10, j
← j + 1)

      si   a[j] mod  2 = 0
entonces
          escribir(a[j])
     fin_si

   fin_desde
fin
```

los números pares ingresados por el usuario.

Ejercicio 3:

Enunciado

Desarrollar un algoritmo que permita llenar un arreglo con 5 números enteros los cuales corresponden a las edades de una familia y luego visualizar cuantos miembros de esa familia son mayores de edad; sabiendo que para ser mayor de edad debe tener 18 o más años.

Análisis del problema

a. **Identificar datos de entrada.-** Para este problema es necesario tener una variable (índice) , además definir un arreglo de tipo entero de tamaño 5, que permitirá almacenar las diferentes edades de los miembros de una familia, también es necesario una

variable(contador) que se incrementa cada vez que encuentre que la edad sea mayor o igual a 18 años

b. **Procesos**.- El proceso que se requiere es de solicitar que se ingrese la edad de una persona y almacenarlo en el arreglo. Luego de haber llenado todo el arreglo con las diferentes edades se requiere contar cuantas edades son mayores o iguales a 18 este proceso se debe llevar a cabo dentro de una iteración.
c. **Salida.-** La salida que se obtendrá será la impresión del número de personas mayores de edad que integran una familia de cinco.

Algoritmo

1. Inicio
2. Declarar el arreglo **a** de tipo entro tamaño **5**
3. c igual a 0
4. i igual a **0**
5. Si **i** es menor que 5 **hacer;** si no, ir al paso **13**
6. Imprime "Ingrese la edad de un integrante de la familia"
7. Lee la edad
8. Almacena la edad en el Arreglo **a** en la posición **i**
9. Si **a**[**i**] es mayor o igual a **18** Entonces; si no ir al paso **11**
10. Incrementa la variable **c** en 1
11. Incrementa la variable **i** en 1; ir al paso **5**
12. Imprimir **c**
13. Finalizar

Pseudocódigo.	
Desarrollo	**Posible Salida en Pantalla**
algoritmo	El posible resultado serán la

contar_mayores_arreglo **var** entero : i, c, a[5] **inicio** c ← 0 **desde** (i ← 0 , i < 5, i ← i + 1) escribir(“ Ingrese la edad de un integrante de la familia”) leer a[i] **si** a[i] >= 18 **entonces** c ← c + 1 **fin_si** escribir (c) **fin_desde** **fin**	cantidad de personas mayores de edad que existen en una familia de 5 integrantes.

4.3. ALGORITMOS DE BÚSQUEDA

Los algoritmos de búsqueda se caracterizan por localizar un elemento específico dentro de una estructura. En los arreglos unidimensionales el algoritmo de búsqueda debe recorrer todo el arreglo posición por posición con el fin de encontrar el elemento solicitado.

Esto permite determinar la existencia o no de un elemento en un arreglo. Para este apartado se estudiaran dos tipos de algoritmos de búsqueda, los cuales se detallan a continuación:

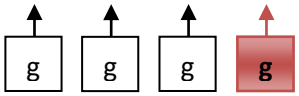
4.3.1 Búsqueda Secuencial

El algoritmo de Búsqueda Secuencial o también llamado lineal compara secuencialmente cada elemento del arreglo con el elemento que se desea buscar hasta que lo encuentre o llegue al final del arreglo, se garantiza la existencia del elemento cuando este es localizado, pero la no existencia sólo garantiza que se recorra la totalidad del arreglo, el resultado que se obtiene de la búsqueda es la posición donde se encuentra el elemento encontrado.

Búsqueda de la letra **g** en el arreglo **a**

Arreglo **a**

Índice	0	1	2	3	4	5	6	7	8	9	10	11
Elemento	P	r	o	G	r	a	m	a	c	i	ó	n



La grafica muestra como el algoritmo empieza a comparar desde el inicio del arreglo cada elemento con el valor que se desea buscar; dando como resultado que el elemento se encuentra en la posición 3 del arreglo.

4.3.1.1 Ejercicios de Pseudocódigo

Ejercicio 1:

Enunciado

Se desea buscar el número 4 en un arreglo definido {24, 2, 31, 5, 4, 73}.Desarrollar un algoritmo de búsqueda para determinar su posición en el arreglo.

Análisis del problema

1. **Identificar datos de entrada.-** Para este problema es necesario tener una variable (bus) que permita almacenar el número que se desea buscar; para este

caso es el número 4, además definir un arreglo de tipo entero de tamaño 6 con los siguientes valores 24, 2, 31, 5, 4, 73. Así como dos variables que servirán como índices para recorrer el arreglo y una variable que almacene la posición del elemento encontrado.

2. **Procesos**.- El primer proceso que se requiere es de ingresar los números requeridos en el enunciado. Este proceso se debe llevar a cabo dentro de una iteración. El segundo proceso debe comparar cada elemento del arreglo con el valor a buscar. Para lograr este proceso se requiere de un ciclo repetitivo que permita iniciar desde la primera posición del arreglo e ir incrementado las posiciones hasta el final del arreglo; este proceso termina cuando se encuentre el elemento o se haya llegado al final del arreglo.
3. **Salida.-** La salida que se obtendrá será la impresión de la posición en que se encuentra el elemento de no encontrarse se retornará un cero (falso).

Algoritmo

1. Inicio
2. Declarar el arreglo **a** de tipo entero tamaño **6**
3. enc igual a0
4. i igual a 0
5. Si **i** es menor que 6 **hacer;** si no, ir al paso **10**
6. escribe "Ingrese un número "
7. Lee el número
8. Almacena el número en el Arreglo **a** en la posición **i**
9. Incrementa la variable **i** en 1; ir al paso **5**
10. escribe "Ingrese un número que desea buscar "
11. Lee **bus**
12. j es igual a 0
13. **Hacer**
14. Si **a**[**j**] es igual a **bus** Entonces; si no ir al paso **18**
15. **c** es igual a j+1
16. Incrementar en 1 el valor de la variable **j**
17. **Mientras enc** sea igual a **cero y j** sea menor a 6, ir

al paso **13**

18. Imprimir **c**

19. Finalizar

Pseudocódigo.
Desarrollo

```
algoritmo búsqueda_secuencial
 var
    entero : i, enc, j, a[6]
inicio
enc ← 0

   desde (i ← 0 , i < 6, i ← i +
1)

     escribir(" Ingrese un
número")

          leer a[i]

  fin_desde


escribir(" Ingrese el número a
buscar")
leer bus
 j ← 0

hacer

        si   a[j] = bus  entonces
             enc ← j + 1
        fin_si

     j ← j + 1
    mientras  enc  = 0  y  j < 6

    escribir (enc)
```

Posible Salida en Pantalla

El posible resultado serán la posición en que se encuentra el número a buscar dentro del arreglo de no ser encontrado el resultado será cero.

fin

Ejercicio 2:

Enunciado

Se desea buscar e imprimir el mayor elemento en un arreglo con números enteros con una dimensión de 10.

Análisis del problema

1. **Identificar datos de entrada.-** Para este problema es necesario tener una variable (mayor) que permita almacenar el mayor número que se encuentra en el arreglo, también es necesario dos variable enteras que servirán de índices para el recorrido del arreglo y definir un arreglo de tipo entero de tamaño 10.
2. **Procesos**.- El primer proceso que se requiere es de ingresar los números enteros en el arreglo; este proceso se debe llevar a cabo dentro de una iteración. El segundo proceso es el de almacenar en la variable mayor el primer elemento del arreglo que servirá de guía para las comparaciones. En tercer lugar se debe generar el proceso de comparar cada elemento del arreglo con el valor de la variable mayor para determinar cuál de las dos es mayor; si se da el caso que el elemento del arreglo es mayor a la variable mayor, se debe asignar a la variable mayor el valor del elemento del arreglo . Para lograr este proceso se requiere de un ciclo repetitivo que permita iniciar desde la

segunda posición del arreglo e ir incrementado las posiciones hasta el final del arreglo.

3. **Salida.-** La salida que se obtendrá será la impresión del número mayor que se encuentra almacenado en el arreglo.

Algoritmo

1. Inicio
2. Declarar el arreglo **a** de tipo entero tamaño **10**
3. i es igual a 0
4. Si **i** es menor que 10 **hacer;** si no, ir al paso **8**
5. escribe "Ingrese un número "
6. Lee el número
7. Almacena el número en el Arreglo **a** en la posición **i**
8. Incrementa la variable **i** en 1; ir al paso **4**
9. **mayor** es igual **a[0]** *-----El primer elemento----*
10. **j es igual a 0**
11. **Hacer**
12. Si **a**[**j**]es mayor a la variable **mayor**Entonces; si no ir al paso **16**
13. **mayor es igual** al valor **a[j]**
14. Incrementar en 1 el valor de la variable **j**
15. **Mientras j** sea menor a 10, ir al paso **12**
16. Imprimir **mayor**
17. Finalizar

Pseudocódigo. **Desarrollo**	**Posible Salida en Pantalla**
algoritmo búsqueda _mayor **var**	El posible

```
    entero : i, mayor, j, A[10]
inicio

   desde (i ← 0 , i < 10, i ← i + 1)

     escribir(" Ingrese un número")

          leer a[i]

  fin_desde

mayor ← a[0]

hacer

        si   a[j] > mayor  entonces
             mayor ← a[j]
        fin_si

        j ← j + 1

   mientras  j < 10

    escribir (mayor)

fin
```

resultado será el mayor número que se encuentra en el arreglo.

4.3.2 Búsqueda Binaria

El algoritmo de Búsqueda Secuencial no es efectivo debido que en el peor de los casos de deberá realizar n comparaciones. Una búsqueda más, es mediante el algoritmo de búsqueda binaria que aplica la frase "divide y vencerás", para lograr esto el arreglo debe estar ordenado. Primero se divide el arreglo a la mitad, esto permite comparar el elemento a buscar con el número que se encuentra en el centro del arreglo si es igual el elemento fue encontrado, pero si es menor se realiza la búsqueda en los elementos que se encuentran a la izquierda del número central, o si es mayor se busca en los elementos a la derecha del número central.

En cualquier caso se vuelve a dividir la parte donde sea posible que se encuentre el número que se desea encontrar, esta acción se deberá repetir tantas veces sea necesaria hasta encontrar el número deseado. Estas divisiones optimizan los tiempos de búsqueda en arreglos grandes.

Búsqueda del número 17 en el arreglo **A**

(0 + 11) / 2 = **5** → ***Es la mitad del arreglo***

Arreglo A

Índice	0	1	2	3	4	**5**	6	7	8	9	10	11
Elemento	2	4	7	10	12	13	15	16	17	20	21	23

El primer paso es verificar si el número 17 es igual o menor 17<=13, para este ejemplo no cumple la condición por lo tanto se deberá realizar la búsqueda en el lado derecho del arreglo a partir de la posición 6, por lo que es necesario una subdivisión.

(6 + 11) / 2= **8** → ***Es la mitad del segundo bloque del arreglo***

Arreglo **A**

Índice	0	1	2	3	4	5	**6**	7	**8**	9	10	11
Elemento	2	4	7	10	12	13	15	16	17	20	21	23

En este caso al volver a comparar da como verdadero el número **17** se encuentra en la posición **8.**

4.3.2.1 Ejercicios de Pseudocódigo

Ejercicio 1: **Enunciado**

Se desea buscar un número ingresado por el usuario en un arreglo de números enteros con una dimensión de 10.

Análisis del problema

1. **Identificar datos de entrada.-** Para este problema es necesario tener una variable (bus) que permita almacenar el número que se desea buscar en el arreglo, también es necesario tres variables enteras que servirán de índices para el recorrido del arreglo. Sin olvidar definir un arreglo de tipo entero de tamaño 10.
2. **Procesos**.- El primer proceso que se requiere es de ingresar los números enteros de manera ascendente para asegurar que el arreglo se encuentre ordenado al momento de ser llenado arreglo; este proceso se debe llevar a cabo dentro de una iteración. El segundo proceso es el dividir el arreglo para 2 para obtener el número de la posición central del arreglo y comenzar a comparar si el elemento a buscar es igual, menor o mayor al número del centro, para poder determinar si se subdivide la parte izquierda o derecha del arreglo. Este proceso se lleva a cabo con un ciclo repetitivo.
3. **Salida.-** La salida que se obtendrá será la impresión de la posición donde se encuentra el número a buscar o cero de no ser encontrado.

Algoritmo

1. Inicio
2. Declarar cuatro variables de tipo entero **i, j, k, bus**
3. Declarar el arreglo **A** de tipo entro tamaño **10**
4. Asigna 0 a la variable **i**
5. Si **i** es menor que 10 **hacer;** si no, ir al paso **9**
6. Imprime "Ingrese un número "

7. Lee el número
8. Almacena el número en el Arreglo **A** en la posición **i**
9. Incrementa la variable **i** en 1; ir al paso **5**
10. Lee **bus**
11. Asigna 0 a la variable **i**
12. Asigna 9 a la variable **j**
13. **Hacer**
14. Asigna **i** más **j** dividido para **2** a la variable **k** ----- *mitad del arreglo*
15. Si **A**[**k**] es menor o igual a la variable **bus** Entonces; si no ir al paso **17**
16. Asigna a la variable **i** el valor **k** más 1
17. Si **A**[**k**] es mayor o igual a la variable **bus** Entonces; si no ir al paso **19**
18. Asigna a la variable **j** el valor **k** menos 1
19. **Mientras i** sea menor **j**
20. Imprimir **k**
21. Finalizar

Pseudocódigo. **Desarrollo**	**Posible Salida en Pantalla**
algoritmo búsqueda_binaria **var** entero : i, k, j, bus, A[10] **inicio** **desde** (i ← 0 , i < 10, i ← i + 1) escribir(" Ingrese un número") leer A[i] **fin_desde** escribir(" Ingrese un elemento a desea buscar") leer **bus**	El posible resultado será el número de la posición donde se encuentre el elemento que se está buscando en el arreglo.

```
i ← 0
j ← 9


hacer
k←(i + j) / 2

        si   A[k] <= bus  entonces
             i ← k+1
        fin_si

        si   A[k] >= bus  entonces
             j ← k-1
        fin_si

   mientras  i < =j

     escribir (k)

fin
```

CAPITULO V

CONTENIDO

- **Lenguaje C**
- **Variables Locales y Globales.**
- **Lenguaje de Programación Estructurado C**
- **Directivas**
- **Tipos de Datos en C**
- **Operadores, Expresiones y Estructura**
- **Expresiones.**
- **Estructuras**

Introducción

En este capítulo el lector comienza a conocer el lenguaje de programación en C, su estructura como se realiza las declaraciones de variables, como se define y utilizan los diferentes operadores lógicos, matemáticos y relacionarlos con los utilizados en los algoritmos para poder realizar una implementación exitosa de de los ejercicios de los capitulos analizados anteriormente.

5.1 LENGUAJE C

El lenguaje C, es un lenguaje de programación que fue diseñado por Dennies Ritchie en 1970, en los laboratorios Bell de Estados Unidos.

Este lenguaje tiene las siguientes caracteristicas:

- Lenguaje de programación de propósitos generales
- Permite la Programación Estructurada
- Abundancia de Operadores y Tipos de Datos
- No está asociado a ningún sistema operativo ni a ninguna máquina
- Permite el desarrollo de Sistemas Operativos y programas de aplicación
- Existen las librerías en las bibliotecas
- Tiene sólo 32 palabras reservadas.

Bibliotecas: es un archivo que contiene código objeto de una colección de rutinas o funciones las mismas que realizan tareas determinadas y se pueden ser utilizadas en los programas.

Enlazador: Programa que convierte las funciones de la biblioteca estándar de C, con el código que ha traducido el compilador.

5.1.2 Estructura de un programa en C

Para realizar la programación en C es necesario, primero mencionar algunos de los errores típicos al programar, esto sirve para identificarlos y así poderlos corregir.

Error de Sintaxis: Estos errores son producidos cuando no se toma en consideración o se hace mal uso de las reglas del lenguaje de programación, llegando a violar las normas de sintaxis de ese lenguaje (en nuestro caso C); estos errores son identificados por el compilador y muestra la línea donde se encuentran.

Errores de Ejecución: Estos errores se producen cuando se le indica a la computadora realizar una determinada acción, pero no puede ejecutarla. Por ejemplo indicarle a la computadora una división entre cero, sumar dos variables a las cuales no se les ha signado un valor, etc.

Errores de Lógica: Muchas veces cuando se esta programando el compilador no indica errores de sintaxis, ni de ejecución pero el resultado de nuestro programa, esta fuera del rango esperado, esto es producto de un error de lógica en el código fuente de nuestro programa.

La estructura de un programa en C, consta de algunas partes esenciales: las cuales son uno o más módulos llamadas funciones, siendo main() la primera función que es llamada cuando empieza la ejecución del programa .

Cada función debe tener:

Directivas de pre-procesador o instrucciones que se le dan al compilador

#include

#define

ejemplo: #include <stdio.h>

Lo que se le está indicando, es que incluya las librerías de lenguaje C, en nuestro programa la librería stdio.h, que contiene las funciones de entrada y

salida estándar de datos. Si se necesita las funciones matemáticas, debemos especificarlo con la declaratoria:

#include **<**math.h**>**

Si necesitamos las funciones de cadenas:

#inlcude **<**stlib.h**>**

Esto se realiza al inicio del programa, y las declaratorias deben llevar el símbolo de numeral (#) seguido de la palabra reservada "include", y entre signos de mayor y menor que (<>) el nombre de la librería.

5.2 VARIABLES LOCALES Y GLOBALES.

Las funciones permiten modular un código, es decir las variables declaradas en las definiciones de función son variables locales.
Las variables globales pueden ser utilizadas en todo el código. Estas variables existen permanentemente durante la ejecución del programa y son utilizadas para comunicar información entre funciones.

5.3 LENGUAJE DE PROGRAMACIÓN ESTRUCTURADO C

¿Por qué programación estructurada?

En los párrafos anteriores se habla de las características del lenguaje C, en una de ellas se menciona que el Lenguaje de Programación C, permite la programación estructurada, esto significa que se hace uso de una técnica llamada **Lógica Estructurada,** donde el objetivo principal es diseñar soluciones "correctas" y confiables a los problemas, tomando en consideración la eficiencia en la minimización del uso de memoria y el tiempo de respuesta.

5.3.1 Sintaxis y Algunos Elementos de Un Programa en C

Identificadores. Como su nombre lo indica, estos son los nombres, con los que identificamos las variables, constantes, funciones, vectores, etc., de un programa. Para ello se debe tener presente algunas reglas:

- Se puede tener de 1 hasta un máximo de 31 caracteres

- Debe de iniciar con una letra o subrayado

Ejemplo:

(Correctos)

c2

_c2

(Incorrectos)

2c

2 c

- No es lo mismo una minúscula que una mayúscula, ya que c distingue de entre ellas. Ejemplo:
BETA ¹ Beta ¹ beta ¹ BeTa

- No son válidos los identificadores de palabras reservadas. En un inicio hablamos que c posee 32 palabras reservadas, entre ellas están:float, char , while, int , else, return, include, define ,etc.
Estas palabras no pueden ser utilizadas para identificar variables, constantes, funciones etc.

5.3.2 Comentarios en lenguaje C.

Todo programa que se diseña en C es necesario agregar comentarios en el código, que se usa en futuras modificaciones y para cuando se realice el mantenimiento podamos recordar cosas importantes con los comentarios.

El formato de los comentarios en C, es el siguiente:

/*comentario en una línea*/

/*Podemos colocar mucha información importante

de nuestro Programa en varias líneas */

//este comentario en una línea

5.4 DIRECTIVAS.

5.4.1 La Directiva #include

Permite que, el pre-procesador, incluya funciones proporcionadas por el fabricante, a nuestro programa. Ejemplo:

#include <stdio.h> /* le decimos al compilador que incluya la librería stdio.h */.

Variables: Las variables pueden cambiar de valor, durante la ejecución del programa.

Constantes: Las constantes como su nombre lo indica, son valores que permanecen constantemente durante toda la ejecución del programa, es recomendable que las constantes se escriban con mayúsculas para realizar una diferencia con las variables en el desarrollo del código fuente, un ejemplo de ello es el valor de (PI) que equivale a 3.14159....

5.4.2 La directiva #define

Esta directiva permite declarar una constante (#define), y va después de los #include. Se escribe la directiva. Ejemplo: #define PI 3.14159

Se deja un espacio y se escribe el identificador de la constante, otro espacio y su valor.

5.4.3 Signos de Puntuación y de Separación

Al momento de programar en C, esta es una regla que se debe de tomar en consideración, y la causa por la cual el programa puede dar muchos errores de sintaxis, cuando se omite al final de cada sentencia un punto y coma (;).

NOTA: el lector no debe confundirse, las directivas: #include, #define. Main(), no llevan punto y coma, por que no son sentencias.

#include <stdio.h>

```
#include <conio.h>

main()

{

float radio, area;

printf("Radio=\n");

scanf("%f", &radio);

area=3.14159*radio*radio;

printf("El Area es %f\n\n", area);

getch();

return 0;

}
```

Todas las Instrucciones o sentencias del programa terminan con un punto y coma, se debe considerar cuando se vea las instrucciones anidadas en condiciones de ciclos, etc.

Ejemplo:

```
{

...

printf("Hola\n\b");

...

}
```

Todo bloque de instrucciones debe ir entre llaves.

En una línea se pueden escribir más de una instrucción separada por un punto y coma.

Esto es posible, por que con el punto y coma, indica al compilador el fin de una sentencia o instrucción.

Ejemplo:

b = c + d; d = 2 * k;

5.5 TIPOS DE DATOS EN C

Un tipo de dato, se define como un conjunto de valores que puede tener una variable, junto con otras operaciones que se pueden realizar con ellas.

5.5.1 Tipos de Datos predefinidos.

TABLA CON LOS TIPOS DE DATOS PREDEFINIDOS EN C			
ENTEROS: números completos y sus negativos			
Palabra reservada:	**Ejemplo**	**Tamaño (byte)**	**Rango de valores**
Int	-850	2	-32767 a 32767
VARIANTES DE ENTEROS			
short int	-10	1	-128 a 127
unsigned int	45689	2	0 a 65535
long int	588458	4	-2147483648 a 2147483647
unsigned long	20000	4	0 a 4294967295

REALES: números con decimales o punto flotante			
Palabra reservada:	**Ejemplo**	**Tamaño (byte)**	**Rango de valores**
Float	85	4	3.4x10-38 a 3.4x1038
VARIANTES DE LOS REALES			
Doublé	0.0058	8	1.7x10-308 a 1.7x10308
long double	1.00E-07	10	3.4x10-4932 a 1.1x104932
>CARÁCTER: letras, digitos, símbolos, signos de puntuación.			
Palabra reservada:	**Ejemplo**	**Tamaño (byte)**	**Rango de valores**
Char	'O'	1	0 255

Tabla #2: Tipos de datos predefinidos en c
Fuente: desarrollada por los autores

5.5.2 Declaración de Variables

Una Variable, como su nombre lo indica, es capaz de almacenar diferentes valores durante la ejecución del programa. La declaración de variables en C, se hace en minúsculas.

Sintaxis:

Tipo_de_dato nombre_de_la_variable;

5.5.3 Declaración de Constantes

Las constantes, como su nombre lo indica, son valores que se mantiene invariables durante la ejecución del programa.

Su formato es el siguiente:

const tipo_de_dato nombre= valor;

const, es una palabra reservada, para indicarle al compilador que se esta declarando una constante**.**

Ejemplo:

const int dia=7;

const float pi=3.14159;

const char caracter= 'm';

const char fecha[]="25 de diciembre";

Caso Especial, Constantes Simbólicas

Las constantes simbólicas, se declaran mediante la directiva #define, cuando C, encuentra el símbolo que representa a la constante, lo sustituye por su respectivo valor.

Ejemplo:

#define N 150

#define PI 3.1416

#define P 50

NOTA: se debe tomar en consideración algunas diferencias fundamentales entre la declaratoria const y #define; la primera, se encuentra dentro del programa, es decir, dentro de la función main() o alguna función definida por

el usuario, mientras que #define se encuentra en el encabezado, después de los #include, las mismas que no llevan al final el punto y coma (;).

5.5.4 Entrada y Salida de Datos.

Son operaciones que se realizan a través del teclado y la pantalla del computador. En C no hay palabras claves para realizar estas acciones de Entrada/Salida, estas se hacen mediante el uso de las funciones de la biblioteca estándar (stdio.h).

Para utilizar las funciones de E/S debemos incluir en el programa el archivo de cabecera stdio.h, mediante la declaratoria:

#include <stdio.h>

Las Funciones de E/S más simples son getchar() que se encarga de leer un carácter del teclado, espera un enter y el carácter aparece. Es decir, la tecla presionada.

putchar(): Imprime un carácter en la pantalla, en la posición actual del cursor.

Algunas variaciones:

getche(): Aparece el carácter

getch(): No aparece el carácter

estas instrucciones se encuentran en la biblioteca conio.h

5.5.5 Entrada / Salida Por Consola con Formato

Las funciones gets, puts, getch, etc; son utilizadas, en una forma un poco rudimentaria, sin embargo; C posee otra serie de funciones, que son más completas, las cuales nos permiten leer e imprimir (en pantalla), datos con un formato determinado, el cual ha sido definido por el programador.

Salida Hacia Pantalla [printf()]

Se utiliza para imprimir en pantalla cadenas de texto, o mandar a pantalla el valor de alguna variable, o constante, o una combinación de las anteriores. Su formato es el siguiente:

Printf("cadena de control", nombre_de_variables**);**

En donde, cadena de control contiene códigos de formato que se asocian con los tipos de datos contenidos en las variables.

Código	**Formato**
%d	Un entero
%i	Un entero
%c	Un carácter
%s	Una cadena
%f	Un real
%ld	Entero largo
%u	Decimal sin signo
%lf	Doble posición
%h	Entero corto
%o	Octal
%x	Hexadecimal
%e	Notación Científica
%p	Puntero

%%	Imprime Porcentaje

Tabla #3: cadena de control en c
Fuente: desarrollada por los autores

Ejemplo:

Int suma=10;

Printf("La suma es %d", suma);

5.5.6 Secuencias de Escapes

Indica que debe ejecutar algo extraordinario.

Carácter de Escape	**Explicación**
\n	Simula un Enter. Se utiliza para dejar una línea de por medio
\t	Tabulador horizontal. Mueve el cursor al próximo tabulador
\v	Tabulador vertical.
\a	Hace sonar la alarma del sistema
\\	Imprime un carácter de diagonal invertida
\?	Imprime el carácter del signo de interrogación
\"	Imprime una doble comilla

Tabla #4: secuencias de escape en c
Fuente: desarrollada por los autores

5.5.7 Entrada de datos (Scanf).

Se utiliza para permitir asignar o capturar un valor a una variable

ejemplo: Scanf ("Cadena de control",& variable);

5.6 OPERADORES, EXPRESIONES Y ESTRUCTURAS.

Hasta el momento se realiza la revisión de los procesos básico para un programa en C; y algunas funciones muy importantes, como son las funciones de lectura e impresión (scanf y printf, respectivamente). Otros aspectos fundamentales, son los operadores, que pueden ser: lógicos, matemáticos, relacionales, etc. Las expresiones, y las estructuras: de secuenciación, de selección y de iteración.

5.6.1 Operadores Aritméticos

Un operador, es un símbolo que indica al compilador que se lleve a cabo ciertas acciones matemáticas o lógicas.

Operador	Propósito
+	Suma
-	Resta
*	Multiplicación
/	División
%	Resto de la división entera

Tabla #5: Operadores aritméticos en c
Fuente: desarrollada por los autores

Estos operadores se pueden aplicar a constantes, variables y expresiones. El resultado es el que se obtiene de aplicar la operación correspondiente.

Las variables sobre los que actúan los operadores aritméticos deben ser valores **Numéricos,** es decir datos enteros, punto flotante (int, float).

Una aclaración especial e necesaria para el operador "%", que indica el resto de la división entera. Veámoslo con un ejemplo:

Si dividimos 30/3, su cociente es 10, y su residuo es 0. Si dividimos 25/3, su cociente es 8, y tiene un residuo de 1, lo que se encarga, este operador, es devolver el valor del residuo de una división. Cabe aclarar que los datos deben de ser tipo entero.

NOTA: Este Operador, NO puede aplicarse a los datos de tipo float.

5.6.2 Operadores de Relaciónales, Lógicos y Unarios

Estos Operadores, los podemos dividir, en varios tipos, entre los cuales están:

Operadores Unarios : El lenguaje C incluye una clase de operadores que actúan sobre un solo operador para producir un nuevo valor. Por eso el nombre de unarios, por que para poder funcionar solo necesitan de un operador a continuación se detallan:

Operador	Propósito
-	Menos Unario: Es el signo menos que va delante de una variable, constante o expresión.
++	Operador Incremento: Hace que la variable, constante o expresión se aumente en uno.
--	Operador Decremento: Hace que su variable, constante o expresión disminuya en uno.

Tabla #6: Operadores unarios en c
Fuente: desarrollada por los autores

Ejemplo:

Int i=1, x=5;

Printf("%d", ++i);

Printf("%d", - -i);

Los operadores de incremento y decremento, pueden utilizarse de dos maneras, eso depende del orden de aparición de los mismos:

- Si el operador precede al operando el valor del operando se modifica antes de ser utilizado.
- Si el operador aparece después del operando, este se modifica después de ser utilizado.

5.6.3 Operadores de Relación.

Estos operadores son utilizados en las estructuras condicionales a continuación se muestra las representación de estos operadores.

Operador	Significado
<	Menor que
<=	Menor o igual que
>	Mayor que
>=	Mayor o igual que
==	Igual que (Para las comparaciones)
! =	No igual a

Tabla #7: Operadores relacionales o de comparacion en c
Fuente: desarrollada por los autores

La asociatividad de estos operadores es de izquierda a derecha. La diferencia entre los operadores = y ==, el primero (=), se utiliza para asignaciones de valores, mientras que el otro (==), se usa para comparaciones. Ejemplo: Si x=5, entonces x==6.

5.6.4 Operadores Lógicos.

Estos son los que nos permiten unir varias comparaciones: 10>5 y 6==6. Los operadores lógicos son: AND (&&), OR (||), NOT(!) en paraentesis se encuentra como se representan en C.

Operador && (AND, en castellano Y): Devuelve verdadero si se cumplen dos condiciones.

Ejemplo: 5>4 && 3>0 **La respuesta es verdadero**

Operador || (OR, en castellano O): Devuelve verdadero si se cumple una de las dos condiciones.

Ejemplo: 5>4 || 0>2 **La respuesta es verdadero**

Operador ! (NOT, negación): Si la condición se cumple NOT hace que no se cumpla y viceversa.

Ejemplo: !(5>4) **La respuesta es falso**

5.6.5 Operadores de Asignación

Los operadores de asignación se encargan de asignar a una variable el resultado de una expresión o el valor de otra.

Se utilizan en forma de expresiones de asignación en los que se asigna en el valor de una expresión a un identificador. El operador de asignación más utilizado es "=" y su formato es:

identificador = expresión;

Donde el identificador representa una variable, una constante o una expresión más compleja.

Si los dos operandos de la expresión de asignación son de tipo de datos diferentes el valor de la expresión de la derecha se convertirá automáticamente al tipo de identificador de la izquierda de esta forma la expresión de asignación será del mismo tipo de datos..

Lenguaje C posee además los siguientes operadores de asignación:

Operador	**Explicación**	
+=	Expresión1+=expresión2. Equivale a: expresión1=expresión1 + expresión2	
-=	i-=1. equivale a: i=i-1	
*=	J*=2. Equivale a: j=j*2	
/=	K/=m, equivale a: k=k/m	
%=	P%n. Equivale a: p=p%n	

Tabla #8: Operadores de asignación en c
Fuente: desarrollada por los autores

5.6.6 Jerarquía de Operadores

Categoría del Operador	**Operador**
Operadores Unarios	-, ++, --, !
Operadores Aritméticos Multiplicación, división y Resto entero	*, /, %

Suma y Resta	+,-
3. Operadores Relacionales	<, <=, >, >=
4. Operadores de Igualdad	==, ! =
5. Operadores Lógicos	&& (Y Lógico), || (NO Lógico)
6. Operadores de Asignación	=, +=, -=, *=, /?, %=,

Tabla #9: jerarquía de operadores en c
Fuente: desarrollada por los autores

5.6.7 Reglas de Jerarquía.

- Se ejecuta primero el operador de más alta jerarquía
- Operadores que tienen igual jerarquía se evalúan de izquierda a derecha
- Si existen expresiones encerradas entre paréntesis, estas se evalúan primero.
- Si existen paréntesis anidados se evalúan primero los paréntesis más internos.

5.6.8 Expresiones

Es una combinación de variables, constantes, y operadores. El resultado de la expresión es el que se expresa aplicar sus operadores a sus operandos. Por ejemplo 1 + 5 es una expresión formada por dos operandos (1 y 5) y el operador (el +); esta expresión es equivalente al valor 6, en el momento de la ejecución es evaluada y sustituida por su resultado.

Una expresión puede estar formada por otras expresiones más sencillas, y puede contener paréntesis de varios niveles agrupando distintos términos. En C, existen diferentes tipos de expresiones, que depende del tipo de operadores que se estén utilizando. Por ejemplo: Expresiones lógicas, aritméticas.

Existen algunas expresiones encerradas entre paréntesis, estas se evalúan primero. Ejemplo: 9*(8+5)

Primero sumamos 8+5, cuyo resultado es 13, y este lo multiplicamos por nueve, con lo que la expresión anterior, da cómo resultado: 117.

Si existen expresiones en paréntesis anidadas, es decir, que uno se encuentra dentro de otros paréntesis, se evalúan los más internos.

Ejemplo: 2*((20/(12-2))+5)

Se evalúa la operación 12-2, que da como resultado 10, luego se divide 20, entre el resultado anterior, es decir 10. el resultado es 2, y a este número se le suma 5, obteniendo 7. ahora se multiplica por dos, para determinar así que la expresión anterior es igual a 14.

5.7 ESTRUCTURAS

5.7.1 Estructuras Secuenciales

Son estructuras que se encuentran en un programa que después de ejecutar una instrucción o sentencia, continúan con la otra hasta llegar al final del programa.

5.7.2 Estructuras Selectivas

Los programas diseñados hasta el momento, han sido del tipo secuencial, es decir una sentencia se ejecuta después de otra hasta el final del programa.

Pero en la vida diaria muchas veces debemos elegir entre un camino y otro para llegar a nuestro destino. Lo mismo pasa en programación, al realizar alguna actividad, nuestro programa debe ser capaz de elegir uno u otro camino a seguir dependiendo del valor de alguna condición evaluada.

Para ello C, dispone de tres tipos de 3 tipos de estructuras selectivas, la cuales son:

- Estructura Selectiva Simple
- Estructura Selectiva Doble

- Estructura Selectiva Múltiple

5.7.2.1 Estructura Selectiva Simple

En esta estructura se evalúa una condición, de ser cierta efectúa una acción, de lo contrario, continúa con la ejecución normal del programa.

Su sintaxis es la siguiente:

If(condición**)**

Acción;

O también:

If(Condición**)**

Acción;

Donde:

Condición: Es una expresión lógica que es evaluada por el compilador

Acción: es la tarea o grupo de tareas que realizará el programa de resultar cierta la condición

NOTA: En C, no existe la sentencia "End If", como en otros lenguajes de programación para indicar que ha terminado el bloque de selección, sino que este se especifica con el punto y coma al final. Además después de la condición NO se escribe un punto y coma , y si son varias acciones estas deben ir entre de llaves **{}** para indicarle al compilador que son un solo bloque de acciones que deben ejecutarse.

5.7.2.2 Estructura Selectiva Doble

Se caracteriza por el hecho de que ofrece dos caminos a seguir, dependiendo si al evaluar la condición resulta cierta o falsa. Su sintaxis es la siguiente:

if(Condición**)**

Acción 1;

else

Acción 2;

Funciona, de la siguiente manera *si* condición, al evaluarla resulta cierta, realiza la acción 1. de lo contrario es decir; si al evaluar la condición resulta falsa, realiza la acción 2.

Se debe tener en cuenta que la condición puede ser compuesta, es decir haciendo uso de los operadores && y || (Y lógico y O lógico), además que cuando tenemos más de una sentencia por ejecutar ya sea del lado del cierto o de falso, estas van entre de llaves.

5.7.2.3 Estructura de selección Múltiple

Este tipo de estructura permite seleccionar entre varias opciones de acuerdo a la petición que se realiza. En este caso se puede elegir una acción que se debe ejecutar de entre varias posibles a evaluar, llamada selector.

```
Switch (selector)
{
case Etiqueta A1:
Acción A1;
break;

case Etiqueta B1:
Acción B1;
break;

case Etiqueta n:
Acción n;
break;

default:
Excepción;
break;
}
```

5.8 CICLOS.

Es común encontrar en los programas operaciones que se deben ejecutar un número de veces en ciertos periodos. Las instrucciones son las mismas pero los datos sobre los que operan varían. A nuestro alrededor encontramos problemas que presentan esas características, por ejemplo: el cálculo de la nota final de los estudiantes de calculo I, se realizará tantas veces como alumnos hayan inscritos en dicha asignatura, el cálculo del salario de los empleados de una empresa, etc.

En estos casos la solución que se diseñe para un solo grupo de datos se debe repetir tantas veces como sea necesario.

Los cálculos simples o la manipulación de pequeños conjuntos de datos se pueden realizar fácilmente a mano, pero las tareas grandes o repetitivas son realizadas con mayor eficacia por una computadora, ya que estas están especialmente preparadas para ello.

Para repetir varias veces un proceso determinado haremos uso de los ciclos repetitivos, a los cuales se les conoce con el nombre de estructura repetitiva, estructura iterativa, lazo o bucle.

En C, podemos encontrar tres tipos de ciclos:

- Entrada Asegurada (while)
- Ciclo Controlado Por Contador (for)
- Hacer Mientras (do.. while)

Funcionamiento de Un Ciclo

Un ciclo, evalúa una condición de resultar verdadera, realiza una acción o bloque de acciones, luego vuelve a evaluar la condición y si nuevamente resulta verdadera, realiza la (s) acción (es). Cuando la condición de cómo resultado falso sale del ciclo y continúa con la ejecución normal del programa.

Acumulador: Es una variable, que, como su nombre lo indica se encarga de acumular valores. Esto se vuelve muy útil cuando queremos encontrar la

suma de los números del 0 al 9, en el acumulador, vamos guardando los valores de dichas cifras. Puede ser tanto real como entera. Su valor inicial, en la mayoría de los casos es cero.

Contador: Es una variable de tipo entero que se utiliza en el programa para contabilizar el número de ejecuciones de una acción. Un acumulador tiene tres valores distintos:

Valor Inicial: es el valor con el cual iniciamos nuestro contador. Generalmente es cero. Esta asignación puede hacerse cuando se declara la variable.

Valor Final: Después de la ejecución del ciclo el valor del contador será distinto a su valor inicial y puede ser mayor o menor, todo depende si fue una cuenta creciente o decreciente.

Valor de Cambio: Es el valor constante que irá incrementando el contador, este puede ser positivo o negativo; es decir si la cuenta se realiza de manera ascendente o descendente.

NOTA: el lector no debe confundirse entre las variables tipo acumulador y tipo contador, estas se diferencian unas de otras en que: los contadores su valor de cambio es una constante, ya que aumenta y disminuyen en el mismo valor, mientras que los acumuladores su valor de cambio no es constante. Un acumulador necesariamente lo inicializamos con cero (o al menos en la mayoría de los casos). Un contador puede iniciar con cualquier valor.

Bandera: Las variables tipo bandera son aquellas que sólo admiten dos valores: cierto o falso, true o false, hombre o mujer... etc

5.8.1 Ciclo de Entrada Asegurada

La sintaxis es la siguiente:

while(condición)

Acción**;**

Su funcionamiento es el siguiente: primero evalúa la condición, si da como resultado verdadero realiza la acción, luego vuelve a evaluar la condición, si su resultado es falso, se sale del ciclo y continúa con la ejecución del programa.

Hay que tener mucho cuidado, cuando trabajamos con ciclos, ya que podemos caer en un ciclo infinito, es decir que nunca se sale de él. Esto es un error de sintaxis sino de lógica. Por lo cual en las acciones debemos siempre colocar algo que haga que se modifique el resultado de la condición, lo cual puede ser una bandera, un contador o un acumulador.

5.8.2Ciclo For.

En algunas ocasiones, sabemos a ciencia cierta el número de veces que se tiene que repetir una misma acción o bloque de acciones. Y para ello es que nos sirve, esta estructura. Su sintaxis es la siguiente:

for(valor inicial; condición; incremento**)**

acción;

Donde:

Valor inicial: es el valor con el cual inicializamos nuestra variable de control.

Condición: si la cumple, ejecuta la acción o acciones e incrementa o decrementa la variable de control, sino la cumple la condición, se sale del ciclo.

Incremento: que puede ser positivo o negativo (decremento).

5.8.3 Ciclo Do... while

Este ciclo funciona de la siguiente manera, realiza la acción o conjunto de acciones, luego evalúa una condición de resultar cierto vuelve a realizar la/s accion/es. Cuando sea falsa, se sale del ciclo Ejemplo:

Formato :

do {

sentencia;

} **while**(<expL>);

La diferencia fundamental, entre el ciclo while y do...while, es que en este ultimo, las sentencias se realizarán por lo menos una vez, en cambio, con while, solo se cumplirán mientras se cumpla la condición, lo cual puede ser nunca.

CAPÍTULO VI

CONTENIDO

- **Ejercicios en lenguaje C operadores**
- **Ejercicios de toma de desiciones en C**
- **Ejercicios de sumadores y acumuladores en C**

Introducción

Este capítulo el lector podrá observar como un ejercicio desarrollado en algoritmo se transcribe a un lenguaje de programación; para esto se consideró al Lenguaje C, debido a la eficiencia en el código que produce, y la similitud con los algoritmos, se puede mencionar que es el lenguaje base, que todo programador debe conocer para desarrollar habilidades que le permitan la construcción de software.

6.1. TIPOS DE DATOS Y OPERADORES

Enunciados correspondientes a resolver problemas específicamente al manejo de tipos de datos y operadores aritméticos, lógicos y relacionales.

Ejercicio 1:

Enunciado

Realizar un programa en lenguaje C que calcule el sueldo de un empleado; este se calcula tomando en cuenta el número de horas trabajadas y se multiplica por el valor de la hora.

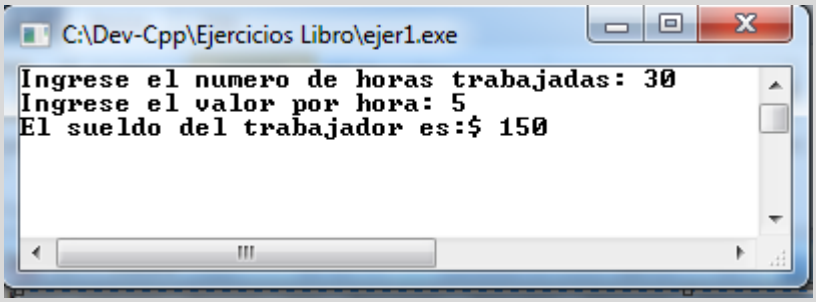
Análisis del problema

1. **Identificar datos de entrada.-** Para este problema es necesario tener una variable que almacene el número de horas trabajadas; el valor por hora.
2. **Procesos.-** El proceso que se desarrollará es multiplicar el número de horas trabajadas por el valor hora.
3. **Salida.-** La salida que requiere en el problema es el sueldo del trabajador.

Código en Lenguaje C

```
#include <stdio.h>
#include <conio.h>
int main(){
int nhoras;
float valorhora,salario;
printf("Ingrese el número de horas trabajadas");
scanf("%i , &nhoras");
printf("Ingrese el valor por hora");
scanf("%f , &valorhora");
salario=nhoras * valorhora;
printf("El sueldo del trabajador es: %i", salario);
getch();
return 0;
}
```

Salida en pantalla de la consola



Ejercicio 2:

Enunciado

Realizar un programa en lenguaje C, que permita la sumar dos números.

Análisis del problema

1. **Identificar datos de entrada.-** Para este problema es necesario tener dos variables que almacenen el primer número y el segundo número que se desean sumar.
2. **Procesos.-** El proceso que se desarrollará es mediante el operador de suma (+) realizar la operación

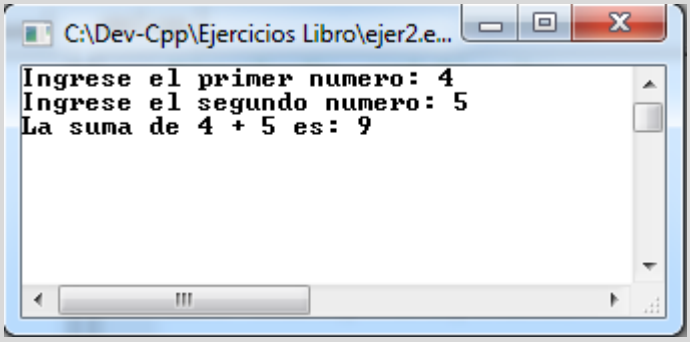
aritmética.

3. **Salida.-** La salida que requiere en el problema es el suma de dos números.

Código en Lenguaje C

```
#include <stdio.h>
#include <conio.h>
int main(){
int num1,num2,suma;
printf("Ingrese el primer número: ");
scanf("%i" , &num1);
printf("Ingrese el segundo número: ");
scanf("%i" , &num2);
suma=num1 + num2;
printf("La suma de %i + %i es: %i",num1,num2,suma);
getch();
return 0;
}
```

Salida en pantalla de la consola



Ejercicio 3: **Enunciado**

Realizar un programa en lenguaje C que permita calcular el área de un triángulo y su fórmula es: base por altura divido para 2.

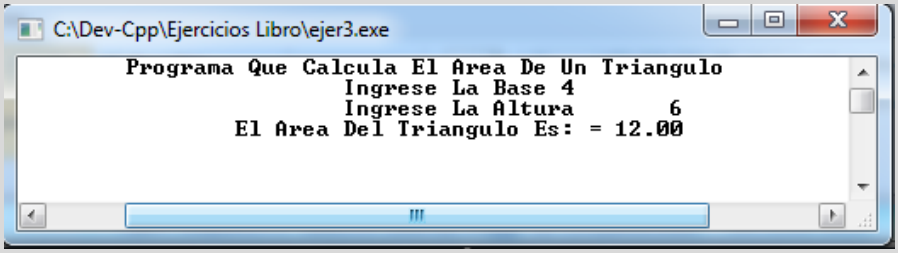
Análisis del problema

1. **Identificar datos de entrada. -** Para este problema es necesario tener dos variables de tipo entero que permita almacenar la base y la altura del triángulo.
2. **Procesos. -** El proceso que se desarrolla es el de multiplicar la base del triángulo por su altura para luego dividirla entre 2.
3. **Salida. -** La salida que se requiere en el problema es el área del triángulo.

Código en Lenguaje C

```
#include <stdio.h>
#include <conio.h>
float A,B,H;
int main(){
    printf("\t\t Programa Que Calcula El Área De Un Triángulo\n");
    printf("\t\t\t\t Ingrese La Base\t");
    scanf("%f",&B);
    printf("\t\t\t\t Ingrese La Altura\t");
    scanf("%f", &H);
    A=B*H/2;
    printf("\t\t\t El Área Del Triángulo Es: = %.2f",A);
    getch();
    return 0;
    }
```

Salida en pantalla de la consola

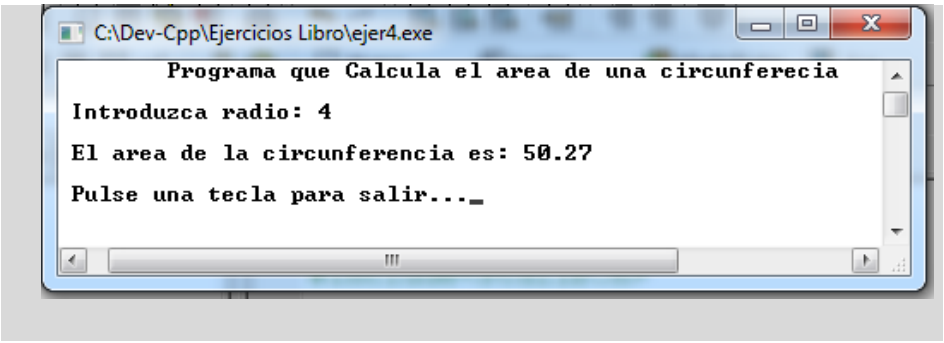


Ejercicio 4:	Enunciado
	Realizar un programa en lenguaje C que permita calcular el área de un círculo su fórmula de cálculo es: $pi*radio^2$, siendo pi un valor constante de 3.1416 y el radio del circulo elevado al cuadrado **Análisis del problema** 1. **Identificar datos de entrada. -** Para este problema es necesario tener una variable para el radio de la circunferencia y declarar una constante llamada pi que tendrá el valor de 3.14. 2. **Procesos. -** El proceso que se desarrolla es el de multiplicar la constante pi con el valor del radio del círculo. 3. **Salida. -** La salida que se requiere en el problema es el área del círculo.

Código en Lenguaje C

```
#include <conio.h>
#include <stdio.h>
#define PI 3.1416
int main(){
   float area, radio;
   printf( "\t  Programa que Calcula el área de una circunferencia " );
   printf( "\n\n   Introduzca radio: " );
   scanf( "%f", &radio );
    area = PI * radio * radio;
    printf( "\n   El área de la circunferencia es: %.2f", area );
    printf( "\n\n   Pulse una tecla para salir..." );
    getch();
    return 0;
}
```

Salida en pantalla de la consola



6.2. TOMA DE DECISIONES, SUMADORES Y ACUMULADORES

Enunciados correspondientes a resolver problemas específicamente al manejo toma de decisiones o también llamados condicionales, así como el uso de sumadores y acumuladores

Ejercicio 1:

Enunciado

Realizar un programa en lenguaje c que permita verificar el mayor de 2 números enteros

Análisis del problema

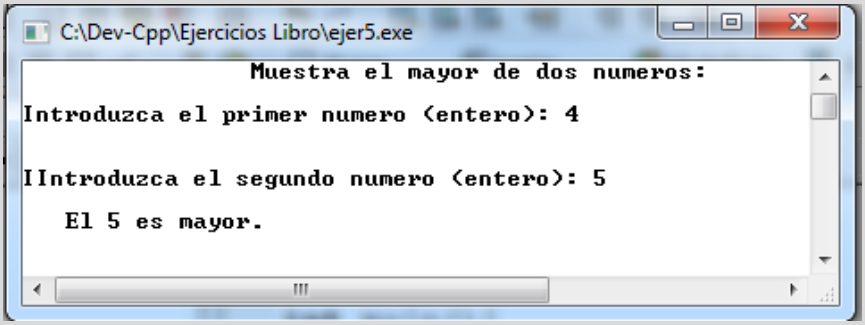
1. **Identificar datos de entrada. -** Para este problema es necesario tener dos variables una para cada número entero
2. **Procesos. -** El proceso que se desarrolla es una condición que permite verificar cuál es mayor de los dos números ingresados.
3. **Salida. -** Número mayor.

Código en Lenguaje C

```
#include <conio.h>
#include <stdio.h>
int main(){
   int n1, n2;
   printf( "\t\t Muestra el mayor de dos números: ");
   printf( "\n\n Introduzca el primer número (entero): " );
   scanf( "%d", &n1 );
```

```
    printf( "\n\n Introduzca el segundo número (entero): ");
    scanf( "%d", &n2 );
    if ( n1 > n2 )
       printf( "\n   El %d es mayor.", n1 );
    else
       if ( n1 < n2 )
          printf( "\n   El %d es mayor.", n2 );
       else
          printf( "\n   SON IGUALES" );
    getch();
    return 0;
}
```

Salida en pantalla de la consola



.

Ejercicio 2:

Enunciado

Realizar un programa en lenguaje C que permita verificar si un número es positivo o negativo.

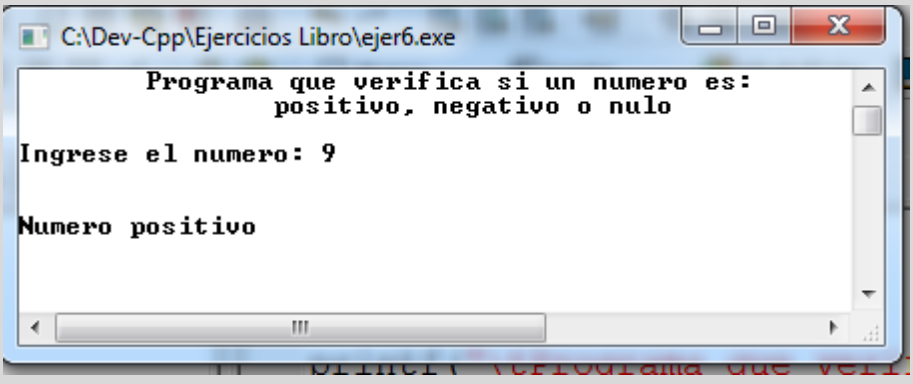
Análisis del problema

1. **Identificar datos de entrada. -** Para este problema es necesario tener una variable de tipo entero.
2. **Procesos. -** El proceso que se desarrolla es una condición que permita verificar si un número es mayor o menor que cero y con esto lograremos identificar si es un número entero positivo o negativo.
3. **Salida. -** tipo de numero positivo o negativo

Código en Lenguaje C

```
#include <stdio.h>
#include <conio.h>
int main()
{
int numero;
printf("\tPrograma que verifica si un número es:\n");
 printf("\t\tpositivo, negativo o nulo\n");
printf("\nIngrese el número: ");
scanf("%i", &numero);
    if(numero>0){
       printf("\n\nNúmero positivo");
     }
    if(numero<0){
       printf("\n\ nNúmero negativo");
    }
    if (numero==0){
       printf("\n\ nNúmero nulo");
    }
getch();
return 0;
}
```

Salida en pantalla de la consola



Ejercicio 3:

Enunciado

Realizar un programa en lenguaje C que permita verificar si un número entero ingresado es par o impar.

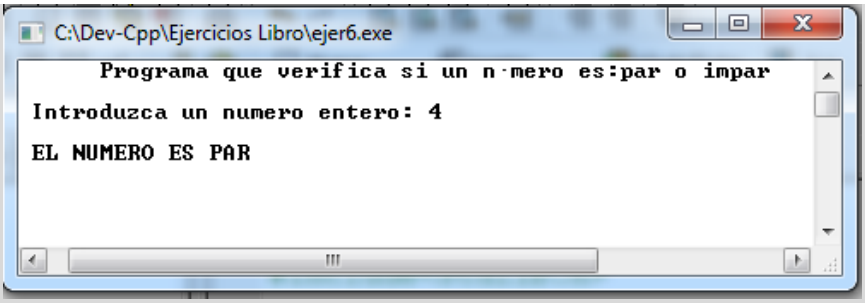
Análisis del problema

1. **Identificar datos de entrada. -** Para este problema es necesario tener una variable de tipo entero.
2. **Procesos. -** El proceso que se desarrolla es una condición que permita verificar si un número es par o impar, para esto se debe calcular el residuo del número entre dos, y utilizar la condición para preguntar si el residuo de la división entre dos es igual a 0 entonces es par y si es igual a uno entonces es impar.
3. **Salida. -** tipo de número par o impar.

Código en Lenguaje C

```
#include <conio.h>
#include <stdio.h>
int main(){
   int numero;
   printf("\tPrograma que verifica si un número es: par o impar \n");
   printf( "\n   Introduzca un número entero: " );
   scanf( "%d", &numero );
   if ( numero % 2 == 0 )
      printf( "\n   EL NÚMERO ES PAR" );
   else
      printf( "\n   EL NÚMERO ES IMPAR" );
   getch();
   return 0;
}
```

Salida en pantalla de la consola



Ejercicio 4:

Enunciado

Realizar un programa en lenguaje c para determinar cuánto se debe pagar por una cantidad de lápices considerando que si son más de 500 el costo por lápiz es de 0.85 en caso contrario el costo es de 1 dólar.

Análisis del problema

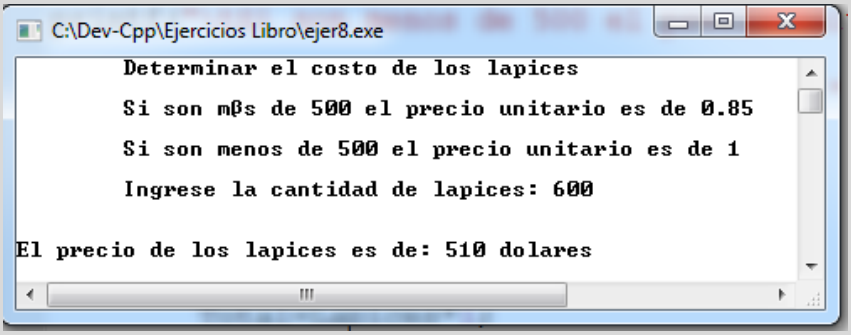
1. **Identificar datos de entrada. -** Para este problema es necesario tener una variable de tipo entero para la cantidad de lápices.
2. **Procesos. -** El proceso que se desarrolla es una condición que permita verificar si la cantidad es mayor que 500 lápices se debe calcular el costo total que será igual al número de lápices por 0.85, en caso contrario será igual al número de lápices por 1dolar.
3. **Salida. -** costo total de los lápices

Código en Lenguaje C

```
#include <stdio.h>
#include <conio.h>
int main(){
   float Lapices, Total;
printf("\tDeterminar el costo de los lapices\n\n");
printf("\tSi son más de 500 el precio unitario es de 0.85\n\n");
printf("\tSi son menos de 500 el precio unitario es de 1\n\n");
```

```
printf("\tIngrese la cantidad de lapices: ");
scanf("%f" , &Lapices);
   if (Lapices>500){
      Total=Lapices*.85;
     }else{
       Total=Lapices*1;
      }
  printf("\t\n\nEl precio de los lapices es de: %.2f dolares",Total);
   getch();
   return 0;
}
```

Salida en pantalla de la consola



Ejercicio 5:

Enunciado

Realizar un programa en lenguaje C para determinar el costo y el descuento de un producto. Para este proceso se debe considerar que si el precio es mayor o igual a 300 se aplica un descuento de 20%; si su precio es mayor o igual a 100 y menor que 300, el descuento es del 15%, y si es menor a 100 el descuento es de 10%.

Análisis del problema

1. **Identificar datos de entrada. -** Para este problema es necesario tener una variable de tipo real para el valor del producto.

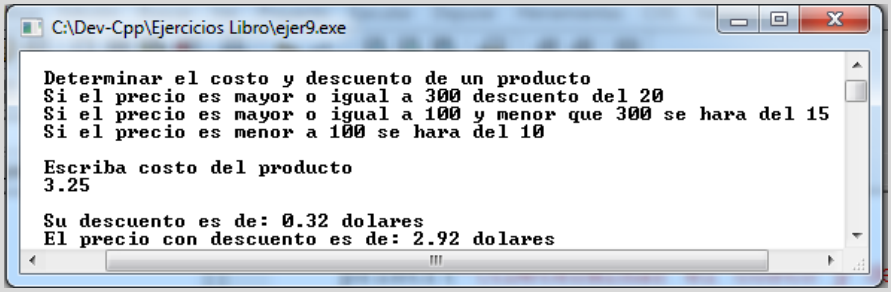
2. **Procesos. -** El proceso que se desarrolla es una condición que permita verificar el costo del producto si es mayor o igual que 300 se le restara al producto el 20%, si el producto está entre mayor igual que 100 y menor que 300 al costo del producto se le resta 15% y si el costo del producto es menor a 100 entonces se deberá restar el 10 % del costo total.
3. **Salida.-** costo total del producto

Código en Lenguaje C

```
#include <stdio.h>
#include <conio.h>
int main(){
float Costo, Total, Descuento;
printf("\tDeterminar el costo y descuento de un producto\n");
printf("\tSi el precio es mayor o igual a 300 descuento del 20%\n");
printf("\tSi el precio es mayor o igual a 100 y menor que 300 se hará del
15%\n");
printf("\tSi el precio es menor a 100 se hara del 10%\n\n");
printf("\tEscriba costo del producto\n\t");
scanf("%f", &Costo);

  if (Costo>=300)
  {
    Descuento=Costo*.20;
    Total=Costo-Descuento;
  }
  else if((Costo>=100)&&(Costo<300)) {
     Descuento=Costo*.15;
     Total=Costo-Descuento;
  }else{
     Descuento=Costo*.10;
     Total=Costo-Descuento;
       }
  printf("\n\tSu descuento es de: %.2f dólares\n",Descuento);
```

```
    printf("\tEl precio con descuento es de: %.2f dólares\n\n\t",Total);
    getch();
    return 0;
}
```

Salida en pantalla de la consola



Ejercicio 6:

Enunciado

Realizar un programa en lenguaje C para determinar el salario de una persona en la semana, teniendo las horas trabajadas y el pago por hora, considerando que después de las cuarenta horas se paga el doble.

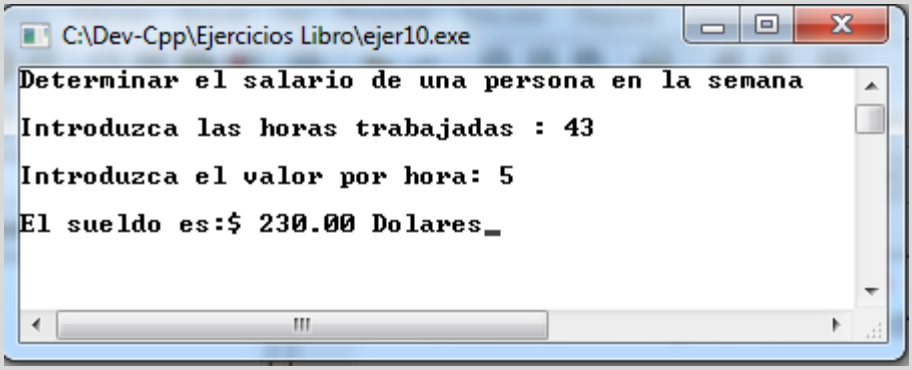
Análisis del problema

1. **Identificar datos de entrada. -** Para este problema es necesario tener dos variables de tipo real para las horas trabajadas y el pago por hora
2. **Procesos.-** El proceso que se desarrollará es una alternativa que permita verificar el número de horas trabajadas si es mayor a cuarenta se deberán pagar al doble las excedentes es decir si son 48 solo las 8 se deberán pagar el doble.
3. **Salida.-** salario del empleado

Código en Lenguaje C

```
#include <stdio.h>
#include <conio.h>
int main(){
   int nhoras,horaextra;
   float valorhora,sueldo,sueldoextra=0,valorhoradoble;

   printf( "Determinar el salario de una persona en la semana\n " );
   printf( "\nIntroduzca las horas trabajadas : " );
   scanf( "%i", &nhoras );
   printf( "\nIntroduzca el valor por hora: " );
   scanf( "%f", &valorhora );

   if ( nhoras > 40){
      horaextra=nhoras-40;
      nhoras=nhoras-horaextra;
      valorhoradoble=valorhora+valorhora;
      sueldoextra=valorhoradoble*horaextra;
      }

      sueldo=nhoras*valorhora;
      sueldo=sueldo+sueldoextra;

        printf( "\nEl sueldo es:$ %.2f Dolares",sueldo );

    getch();
   return 0;
}
```

Salida en pantalla de la consola



Ejercicio 6:

Enunciado

Realizar un programa en lenguaje C que le permita a una persona digitar un número, evalúe si la entrada corresponde a un día de la semana; en caso de que así sea, muestre el nombre del día que le corresponde.

Análisis del problema

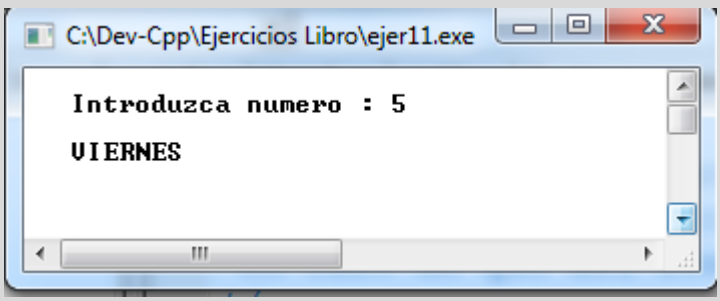
1. **Identificar datos de entrada. -** Para este problema es necesario una variable de tipo entero que almacene el número del día a comparar.
2. **Proceso. –** Se compara el dato ingresado si pertenece al número correspondiente de un día de la semana, caso contrario rechazara los datos ingresados.
3. **Salida. –** Muestra el día de acuerdo al número ingresado.

Código en Lenguaje C

```
#include <stdio.h>
#include <conio.h>
int main(){
   int num;
   printf( "\n   Introduzca número : " );
   scanf( "%d", &num );
```

```
        switch ( num )
        {
           case 1 : printf( "\n   LUNES");
                    break;
           case 2 : printf( "\n   MARTES");
                    break;
           case 3 : printf( "\n   MIÉRCOLES");
                    break;
           case 4 : printf( "\n   JUEVES");
                    break;
           case 5 : printf( "\n   VIERNES");
                    break;
           case 6 : printf( "\n   SÁBADO ");
                    break;
           case 7 : printf( "\n   DOMINGO");
                    break;
           default: printf( "\n  El numero ingresado no corresponde a un
día de la semana");
                    break;
        }
    getch();
    return 0;
}
```

Salida en pantalla de la consola



Ejercicio 7:

Enunciado

Realizar un programa en lenguaje C que le permita realizar un menú básico para un restaurant con los platos que se servirán en el desayuno. Los comensales podrán seleccionar un plato y se les mostrará el precio que tiene ese plato

Análisis del problema

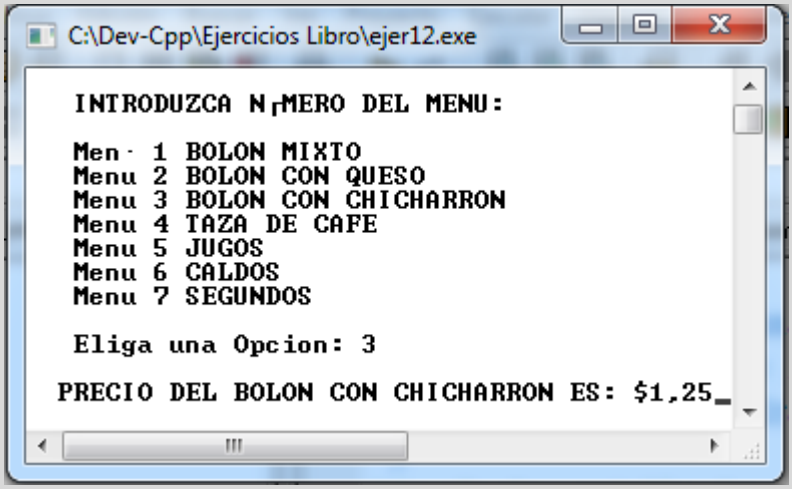
1. **Identificar datos de entrada. -** Para este problema es necesario que el cliente seleccione un ítem del menú y presenta el precio que tiene.
2. **Proceso. –** Se comprobara si número seleccionado corresponde a un Ítem del menú de ser verdadero mostrará su precio, caso contrario rechazará los datos ingresados.
3. **Salida. –** Muestra el precio del plato seleccionado por el cliente.

Código en Lenguaje C

```
#include <stdio.h>
#include <conio.h>
int main(){
int num;
printf( "\n   INTRODUZCA NÚMERO DEL MENU: \n ");
printf( "\n   Menú 1 BOLON MIXTO");
printf( "\n   Menu 2 BOLON CON QUESO");
printf( "\n   Menu 3 BOLON CON CHICHARRON");
printf( "\n   Menu 4 TAZA DE CAFE");
printf( "\n   Menu 5 JUGOS");
printf( "\n   Menu 6 CALDOS" );
printf( "\n   Menu 7 SEGUNDOS");
printf( "\n\n   Eliga una Opcion: ");
scanf( "%d", &num );

switch ( num )      {
 case 1 : printf( "\n  PRECIO DEL BOLON MIXTO ES: $1,75" );
          break;
```

```
 case 2 : printf( "\n  PRECIO DEL BOLON CON QUESO ES: $1,25"
);
            break;
 case 3 : printf( "\n  PRECIO DEL BOLON CON CHICHARRON
ES: $1,25" );
             break;
 case 4 : printf( "\n  PRECIO DE LA TAZA DE CAFE ES: $0.50" );
             break;
 case 5 : printf( "\n  PRECIO DE LOS JUGOS ES: %0.50" );
            break;
 case 6 : printf( "\n  PRECIO DE LOS CALDOS ES: $2.00" );
            break;
 case 7 : printf( "\n  PRECIO DE LOS SEGUNDOS ES; $1,50" );
           break;
default: printf( "\n  El numero ingresado no corresponde a los Platos del
Menu");
           break;
       }
getch();
return 0;
}
```

Salida en pantalla de la consola



Ejercicio 8:

Enunciado

Realizar un programa en lenguaje C que le permita verificar si un usuario a digitado una vocal por medio del teclado

Análisis del problema

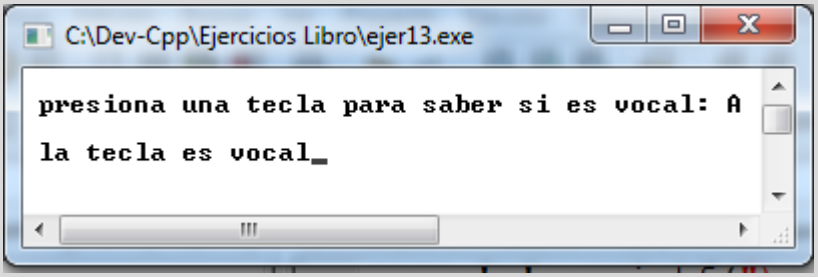
1. **Identificar datos de entrada. -** Para este problema es necesario una variable de tipo carácter para almacenar el valor digitado por el usuario.
2. **Proceso. –** Se comprueba si el carácter digitado por el usuario corresponde a una vocal, caso contrario envía un mensaje indicando que el carácter ingresado no es una vocal.
3. **Salida. –** Muestra la vocal digitada por el usuario

Código en Lenguaje C

```
#include <stdio.h>
#include <conio.h>
int main(){
char op;
printf("\n presiona una tecla para saber si es vocal: ");
scanf("%s", &op);
switch(op)
{
case 'a': printf("\n la tecla es vocal");
break;
case 'e': printf("\n la tecla es vocal");
break;
case 'i': printf("\n la tecla es vocal");
break;
case 'o': printf("\n la tecla es vocal");
break;
case 'u': printf("\n la tecla es vocal");
break;
case 'A': printf("\n la tecla es vocal");
break;
```

```
case 'E': printf("\n la tecla es vocal");
break;
case 'I': printf("\n la tecla es vocal");
break;
case 'O': printf("\n la tecla es vocal");
break;
case 'U': printf("\n la tecla es vocal");
break;
default: printf("\n la tecla no es vocal");
break;
}
getch();
return 0;
}
```

Salida en pantalla de la consola



REFERENCIAS BIBLIOGRAFICAS

Allen B. Tucker , "Lenguajes de programación" 2015, Editorial McGraw Hill

J.D. Muñoz Frías, R. Palacios, "Fundamentos de programación utilizando el lenguaje C", Ed. Universidad Pontificia Comillas. Madrid, España. 2006. ISBN: 84-8468-184-1.

B.S. Gottfried, "Programación en C. Serie Schaum 5ª Edición revisada", Ed. McGrawHill, 2011. ISBN: 84-4819-846-8

J.L. Antonakos, K.C. Mansfield, "Application Programming in Structured C" Ed. Prentice Hall, 2002. ISBN: 01-3356-684-6

G. García Ortiz, "Tecnologías de la información y comunicación", Primera edición, 2010, Colección DGETI

Stair, Ralph M., et al. (2008). Principles of Information Systems, Sixth Edition (en inglés). Thomson Learning, Inc. p. 132. ISBN 0-619-06489-7.

khter, Shameem (2009). Multi-Core Programming (en inglés). Richard Bowles (Intel Press). pp. 11-13. ISBN 0-9764832-4-6.

Groth, David; Skandier, Toby (2005). Guía del estudio de redes, (4ª edición). Sybex, Inc. ISBN 0-7821-4406-3.

LONG, Larry, Et al, "Introducción a las computadoras y a los sistemas de información", Editorial Pretince may, 5ª. Edición, 2009.

http://www.fciencias.unam.mx/revista/temas/contenido.html