

# CONTENIDO DE LA ASIGNATURA ALGORITMOS Y LOGICA DE PROGRAMACIÓN

## CARRERA DE SOFTWARE

En el documento señores estudiantes encontrarán los temas que serán tratados durante el periodo académico mayo- septiembre 2020, separados por semana, de acuerdo a la planificación establecida en la plataforma EVEA

### INDICE DE CONTENIDOS

INTRODUCCION.....	1
CAPITULO I. CONCEPTOS BÁSICOS Y METODOLOGÍA PARA LA SOLUCIÓN DE PROBLEMAS POR MEDIO DE COMPUTADORAS.....	2
1.1 Introducción.....	3
- De los problemas a los programas.....	3
- Breves prácticas de programación.....	3
1.2 Definición de lenguaje.....	5
1.3 Definición de algoritmo.....	6
1.4 Algoritmos cotidianos.....	6
1.5 Definición de lenguajes algorítmicos.....	6
1.6 Metodología para la solución de problemas por medio de computadora.....	6
1.7 Definición del problema.....	6
1.8 Análisis del problema.....	6
1.9 Diseño del algoritmo.....	7
1.10 Codificación.....	7
1.11 Prueba y depuración.....	7
1.12 Documentación.....	7
1.13 Mantenimiento.....	8
CAPITULO II. ENTIDADES PRIMITIVAS PARA EL DESARROLLO DE ALGORITMOS.....	9

2.1 Tipos de datos.....	10
2.2 Expresiones.....	10
2.3 Operadores y operandos.....	11
2.4 Identificadores como localidades de memoria.....	15
 CAPITULO III. TÉCNICAS DE DISEÑO.....	 17
3.1 Top down.....	18
3.2 Bottom up.....	18
 CAPITULO IV. TÉCNICAS PARA LA FORMULACIÓN DE ALGORITMOS.....	 19
4.1 Diagrama de flujo.....	20
4.2 Pseudocódigo.....	21
4.3 Diagrama estructurado (nassi-schneiderman).....	22
 CAPITULO V. ESTRUCTURAS ALGORITMICAS.....	 23
5.1 Secuenciales.....	24
- Asignación.....	24
- Entrada.....	24
- Salida.....	24
5.2 Condicionales.....	25
- Simples.....	25
- Múltiples.....	25
5.3 Repetición fila condicional.....	39
 CAPITULO VI. ARREGLOS.....	 51
6.1 Vectores.....	52
6.1.1 Operaciones con Vectores.....	52
6.2 Matrices.....	54
6.3 Manejo de cadenas de caracteres.....	54
 CAPITULO VII. MANEJO DE MÓDULOS.....	 57
7.1 Definición.....	58
7.2 Función.....	58
7.3 Manipulación.....	59
 CAPITULO VIII. USO DE LA SENTENCIA STRUCT .....	 57
8.1 Definición.....	76

8.2 Estructuras Anidadas .....	79
--------------------------------	----

## INTRODUCCION

El desarrollo de algoritmos es un tema fundamental en el diseño de programas por lo cual el alumno debe tener buenas bases que le sirvan para poder desarrollar de manera fácil y rápida sus programas.

Estos apuntes servirán de apoyo al catedrático de la Universidad Estatal de Bolívar, en su labor cotidiana de enseñanza y al estudiante le facilitará desarrollar su capacidad analítica y creadora, para de esta manera mejorar su destreza en la elaboración de algoritmos que sirven como base para la codificación de los diferentes programas que tendrá que desarrollar a lo largo de su carrera.

## CONCEPTOS BÁSICOS Y METODOLOGÍA PARA LA SOLUCIÓN DE PROBLEMAS POR MEDIO DE COMPUTADORAS.

### 1.1 Introducción

- De los problemas a los programas
- Breves prácticas de programación

### 1.2 Definición de lenguaje

### 1.3 Definición de algoritmo

### 1.4 Algoritmos cotidianos

### 1.5 Definición de lenguajes algorítmicos

### 1.6 Metodología para la solución de problemas por medio de computadora

### 1.7 Definición del problema

### 1.8 Análisis del problema

### 1.9 Diseño del algoritmo

### 1.10 Codificación

### 1.11 Prueba y depuración

### 1.12 Documentación

### 1.13 Mantenimiento

## OBJETIVO EDUCACIONAL:

El alumno:

- Conocerá la terminología relacionada con los algoritmos; así como la importancia de aplicar técnicas adecuadas de programación.
- Conocerá la metodología en cada una de sus etapas.



## **SEMANA 1: DEL 1 AL 5 DE JUNIO /2020**

### **1.1 Introducción**

La computadora no solamente es una máquina que puede realizar procesos para darnos resultados, sin que tengamos la noción exacta de las operaciones que realiza para llegar a esos resultados. Con la computadora además de lo anterior también podemos diseñar soluciones a la medida, de problemas específicos que se nos presenten. Más aún, si estos involucran operaciones matemáticas complejas y/o repetitivas, o requieren del manejo de un volumen muy grande de datos.

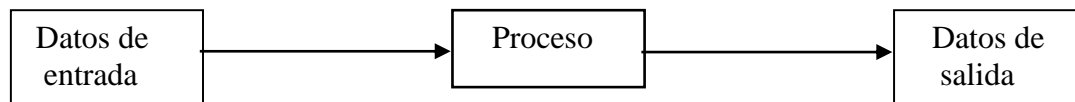
El diseño de soluciones a la medida de nuestros problemas, requiere como en otras disciplinas una metodología que nos enseñe de manera gradual, la forma de llegar a estas soluciones.

A las soluciones creadas por computadora se les conoce como **programas** y no son más que una serie de operaciones que realiza la computadora para llegar a un resultado, con un grupo de datos específicos. Lo anterior nos lleva al razonamiento de que un **programa** nos sirve para solucionar un problema específico.

Para poder realizar **programas**, además de conocer la metodología mencionada, también debemos de conocer, de manera específica las funciones que pueden realizar la computadora y las formas en que se pueden manejar los elementos que hay en la misma.

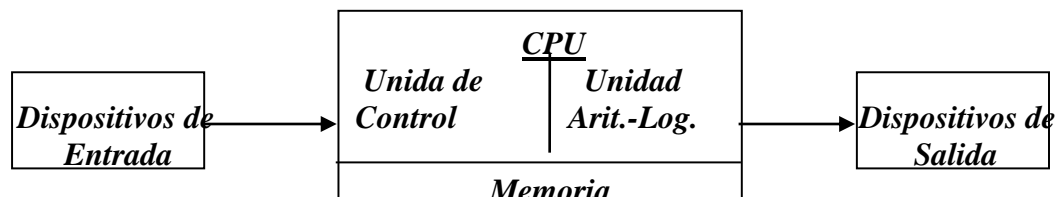
**Computadora:** Es un dispositivo electrónico utilizado para procesar información y obtener resultados. Los datos y la información se pueden introducir en la computadora como entrada (input) y a continuación se procesan para producir una salida (output).

#### ***Proceso de información en la computadora***



**Programa:** Es el conjunto de instrucciones escritas de algún lenguaje de programación y que ejecutadas secuencialmente resuelven un problema específico.

#### ***Organización física de una computadora***



**Dispositivos de Entrada:** Como su nombre lo indica, sirven para introducir datos (información) en la computadora para su proceso. Los datos se leen de los dispositivos de entrada y se almacenan en la memoria central o interna. Ejemplos: teclado, scanners (digitalizadores de rastreo), mouse (ratón), trackball (bola de ratón estacionario), joystick (palancas de juego), lápiz óptico.

**Dispositivos de Salida:** Regresan los datos procesados que sirven de información al usuario. Ejemplo: monitor, impresora.

**La Unidad Central de Procesamiento (C.P.U)** se divide en dos:

- Unidad de control
- Unidad Aritmético - Lógica

**Unidad de Control:** Coordina las actividades de la computadora y determina que operaciones se deben realizar y en qué orden; así mismo controla todo el proceso de la computadora.

**Unidad Aritmética - Lógica:** Realiza operaciones aritméticas y lógicas, tales como suma, resta, multiplicación, división y comparaciones.

**La Memoria** de la computadora se divide en dos:

- Memoria Central o Interna
- Memoria Auxiliar o Externa

**Memoria Central (interna):** La CPU utiliza la memoria de la computadora para guardar información mientras trabaja con ella; mientras esta información permanezca en memoria, la computadora puede tener acceso a ella en forma directa. Esta memoria construida internamente se llama memoria de acceso aleatorio (RAM).

La **memoria interna** consta de dos áreas de memoria:

La memoria **RAM (Random Access Memory)**: Recibe el nombre de memoria principal o memoria del usuario, en ella se almacena información solo mientras la computadora esta encendida. Cuando se apaga o arranca nuevamente la computadora, la información se pierde, por lo que se dice que la memoria RAM es una memoria volátil.

La memoria **ROM (Read Only Memory)**: Es una memoria estática que no puede cambiar, la computadora puede leer los datos almacenados en la memoria ROM, pero no se pueden introducir datos en ella, o cambiar los datos que ahí se encuentran; por lo que se dice que esta memoria es de solo lectura. Los datos de la memoria ROM están grabados en forma permanente y son introducidos por el fabricante de la computadora.



**Memoria Auxiliar (Externa):** Es donde se almacenan todos los programas o datos que el usuario desee. Los dispositivos de almacenamiento o memorias auxiliares (externas o secundarias) mas comúnmente utilizados son: cintas magnéticas y discos magnéticos.

## 1.2 Definición de Lenguaje

**Lenguaje:** Es una serie de símbolos que sirven para transmitir uno o mas mensajes (ideas) entre dos entidades diferentes. A la transmisión de mensajes se le conoce comúnmente como **comunicación**.

La **comunicación** es un proceso complejo que requiere una serie de reglas simples, pero indispensables para poderse llevar a cabo. Las dos principales son las siguientes:

\*0 Los mensajes deben correr en un sentido a la vez.

\*1 Debe forzosamente existir 4 elementos: Emisor, Receptor, Medio de Comunicación y Mensaje.

### **Lenguajes de Programación**

Es un conjunto de símbolos, caracteres y reglas (programas) que le permiten a las personas comunicarse con la computadora.

Los lenguajes de programación tienen un conjunto de instrucciones que nos permiten realizar operaciones de entrada/salida, calculo, manipulación de textos, lógica/comparación y almacenamiento/recuperación.

Los lenguajes de programación se clasifican en:

➤ **Lenguaje Maquina:** Son aquellos cuyas instrucciones son directamente entendibles por la computadora y no necesitan traducción posterior para que la CPU pueda comprender y ejecutar el programa. Las instrucciones en lenguaje maquina se expresan en términos de la unidad de memoria mas pequeña el bit (dígito binario 0 o 1).

➤ **Lenguaje de Bajo Nivel (Ensamblador):** En este lenguaje las instrucciones se escriben en códigos alfabéticos conocidos como mnemotécnicos para las operaciones y direcciones simbólicas.

➤ **Lenguaje de Alto Nivel:** Los lenguajes de programación de alto nivel (BASIC, pascal, cobol, fortran, etc.) son aquellos en los que las instrucciones o sentencias a la computadora son escritas con palabras similares a los lenguajes humanos (en general en ingles), lo que facilita la escritura y comprensión del programa.

SEMANA 2:

### ***1.3 Definición de Algoritmo***

La palabra algoritmo se deriva de la traducción al latín de la palabra árabe alkhwarizmi, nombre de un matemático y astrónomo árabe que escribió un tratado sobre manipulación de números y ecuaciones en el siglo IX.

Un algoritmo es una serie de pasos organizados que describe el proceso que se debe seguir, para dar solución a un problema específico.

### ***1.4 Tipos de Algoritmos***

- ***Cualitativos:*** Son aquellos en los que se describen los pasos utilizando palabras.
- ***Cuantitativos:*** Son aquellos en los que se utilizan cálculos numéricos para definir los pasos del proceso.

### ***1.5 Lenguajes Algorítmicos***

Es una serie de símbolos y reglas que se utilizan para describir de manera explícita un proceso.

#### ***Tipos de Lenguajes Algorítmicos***

- ***Gráficos:*** Es la representación gráfica de las operaciones que realiza un algoritmo (diagrama de flujo).
- ***No Gráficos:*** Representa en forma descriptiva las operaciones que debe realizar un algoritmo (pseudocódigo).

### ***1.6 Metodología para la solución de problemas por medio de computadora***

#### ***1.7 Definición del Problema***

Esta fase está dada por el enunciado del problema, el cual requiere una definición clara y precisa. Es importante que se conozca lo que se desea que realice la computadora; mientras esto no se conozca del todo no tiene mucho caso continuar con la siguiente etapa.

#### ***1.8 Análisis del Problema***

Una vez que se ha comprendido lo que se desea de la computadora, es necesario definir:

Los datos de entrada.

Cual es la información que se desea producir (salida)

Los métodos y fórmulas que se necesitan para procesar los datos.

Una recomendación muy práctica es el que nos pongamos en el lugar de la computadora y analicemos que es lo que necesitamos que nos ordenen y en que secuencia para producir los resultados esperados.

### **1.9 Diseño del Algoritmo**

Las características de un buen algoritmo son:

- Debe tener un punto particular de inicio.

- Debe ser definido, no debe permitir dobles interpretaciones.

- Debe ser general, es decir, soportar la mayoría de las variantes que se puedan presentar en la definición del problema.

- Debe ser finito en tamaño y tiempo de ejecución.

### **1.10 Codificación**

La codificación es la operación de escribir la solución del problema (de acuerdo a la lógica del diagrama de flujo o pseudocódigo), en una serie de instrucciones detalladas, en un código reconocible por la computadora, la serie de instrucciones detalladas se le conoce como código fuente, el cual se escribe en un lenguaje de programación o lenguaje de alto nivel.

### **1.11 Prueba y Depuración**

Los errores humanos dentro de la programación de computadoras son muchos y aumentan considerablemente con la complejidad del problema. El proceso de identificar y eliminar errores, para dar paso a una solución sin errores se le llama **depuración**.

La **depuración o prueba** resulta una tarea tan creativa como el mismo desarrollo de la solución, por ello se debe considerar con el mismo interés y entusiasmo.

Resulta conveniente observar los siguientes principios al realizar una depuración, ya que de este trabajo depende el éxito de nuestra solución.

### **1.12 Documentación**

Es la guía o comunicación escrita en sus variadas formas, ya sea en enunciados, procedimientos, dibujos o diagramas.

A menudo un programa escrito por una persona, es usado por otra. Por ello la documentación sirve para ayudar a comprender o usar un programa o para facilitar futuras modificaciones (mantenimiento).

La **documentación** se divide en tres partes:

Documentación Interna  
Documentación Externa  
Manual del Usuario

- Documentación Interna: Son los comentarios o mensaje que se añaden al código fuente para hacer más claro el entendimiento de un proceso.
- Documentación Externa: Se define en un documento escrito los siguientes puntos:
  - Descripción del Problema
  - Nombre del Autor
  - Algoritmo (diagrama de flujo o pseudocódigo)
  - Diccionario de Datos
  - Código Fuente (programa)
- Manual del Usuario: Describe paso a paso la manera como funciona el programa, con el fin de que el usuario obtenga el resultado deseado.

### **1.13 Mantenimiento**

Se lleva acabo después de terminado el programa, cuando se detecta que es necesario hacer algún cambio, ajuste o complementación al programa para que siga trabajando de manera correcta. Para poder realizar este trabajo se requiere que el programa este correctamente documentado.

## **CAPITULO II.**

### **ENTIDADES PRIMITIVAS PARA EL DESARROLLO DE**

## ALGORITMOS

- 2.1 Tipos de datos
- 2.2 Expresiones
- 2.3 Operadores y operandos
- 2.4 Identificadores como localidades de memoria

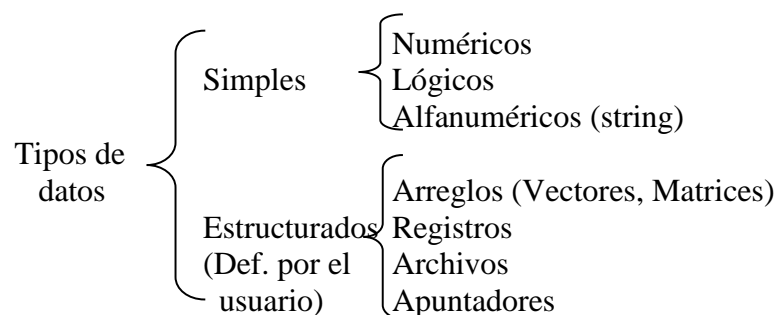
### OBJETIVO EDUCACIONAL:

El alumno:

- Conocerá las reglas para cambiar fórmulas matemáticas a expresiones válidas para la computadora, además de diferenciar constantes e identificadores y tipos de datos simples.

## SEMANA 2: DEL 8 AL 12 DE JUNIO /2020 2.1 Tipos De Datos

Todos los datos tienen un tipo asociado con ellos. Un dato puede ser un simple carácter, tal como 'b', un valor entero tal como 35. El tipo de dato determina la naturaleza del conjunto de valores que puede tomar una variable.



### *Tipos de Datos Simples*

- **Datos Numéricos:** Permiten representar valores escalares de forma numérica, esto incluye a los números enteros y los reales. Este tipo de datos permiten realizar operaciones aritméticas comunes.
- **Datos Lógicos:** Son aquellos que solo pueden tener dos valores (cierto o falso) ya que representan el resultado de una comparación entre otros datos (numéricos o alfanuméricos).
- **Datos Alfanuméricos (String):** Es una secuencia de caracteres alfanuméricos que permiten representar valores identificables de forma descriptiva, esto incluye nombres de personas, direcciones, etc. Es posible representar números como alfanuméricos, pero estos pierden su propiedad matemática, es decir no es posible hacer operaciones con ellos. Este tipo de datos se representan encerrados entre comillas.

Ejemplo:

“Instituto Tecnológico de Tuxtepec”  
“1997”

## 2.2 Expresiones

Las expresiones son combinaciones de constantes, variables, símbolos de operación, paréntesis y nombres de funciones especiales. Por ejemplo:

$$a+(b + 3)/c$$

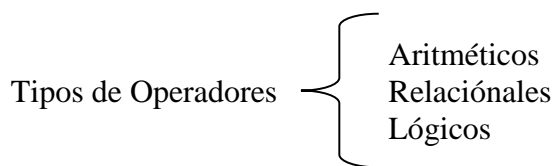
Cada expresión toma un valor que se determina tomando los valores de las variables y constantes implicadas y la ejecución de las operaciones indicadas.

Una expresión consta de operadores y operandos. Según sea el tipo de datos que manipulan, se clasifican las expresiones en:

- Aritméticas
- Relacionales
- Lógicas

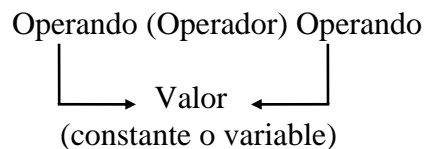
## 2.3 Operadores y Operandos

➤ **Operadores:** Son elementos que relacionan de forma diferente, los valores de una o mas variables y/o constantes. Es decir, los operadores nos permiten manipular valores.



➤ **Operadores Aritméticos:** Los operadores aritméticos permiten la realización de operaciones matemáticas con los valores (variables y constantes).

Los operadores aritméticos pueden ser utilizados con tipos de datos enteros o reales. Si ambos son enteros, el resultado es entero; si alguno de ellos es real, el resultado es real.



### Operadores Aritméticos

- |     |  |
|-----|--|
| +   | Suma                                   |
| -   | Resta                                  |
| *   | Multipliación                          |
| /   | División                               |
| Mod | Modulo (residuo de la división entera) |

Ejemplos:

Expresión	Resultado
7 / 2	=3.5

$12 \bmod 7 = 5$   
 $4 + 2 * 5 = 14$   
 $20 - 7/5 = 18.6$   
 $18/4 = 4.5$   
 $57 \bmod 3 = 0$

### ***Prioridad de los Operadores Aritméticos***

η Todas las expresiones entre paréntesis se evalúan primero. Las expresiones con paréntesis anidados se evalúan de dentro a fuera, el paréntesis mas interno se evalúa primero.

η Dentro de una misma expresión los operadores se evalúan en el siguiente orden.

- 1.- ^ Exponenciación
- 2.- \*, /, mod Multiplicación, división, modulo.
- 3.- +, - Suma y resta.

η Los operadores en una misma expresión con igual nivel de prioridad se evalúan de izquierda a derecha.

Ejemplos:

$4 + 2 * 5 = 14$	
$23 * 2 / 5 = 9.2$	$46 / 5 = 9.2$
$3 + 5 * (10 - (2 + 4)) = 23$	$3 + 5 * (10 - 6) = 3 + 5 * 4 = 3 + 20 = 23$
$3.5 + 5.09 - 14.0 / 40 = 5.09$	$3.5 + 5.09 - 3.5 = 8.59 - 3.5 = 5.09$
$2.1 * (1.5 + 3.0 * 4.1) = 28.98$	$2.1 * (1.5 + 12.3) = 2.1 * 13.8 = 28.98$

### **➤ Operadores Relacionales:**

η Se utilizan para establecer una relación entre dos valores.

η Compara estos valores entre si y esta comparación produce un resultado de certeza o falsedad (verdadero o falso).

η Los operadores relacionales comparan valores del mismo tipo (numéricos o cadenas)

η Tienen el mismo nivel de prioridad en su evaluación.

η Los operadores relacionales tiene menor prioridad que los aritméticos.

### ***Operadores Relacionales***

>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
<>	Diferente
=	Igual



Ejemplos:

Si  $a = 10$        $b = 20$        $c = 30$

$a + b > c$	Falso
$a - b < c$	Verdadero
$a - b = c$	Falso
$a * b < > c$	Verdadero

Ejemplos no lógicos:

$a < b < c$

$10 < 20 < 30$

$T < 30$  (no es lógico porque tiene diferentes operandos)

### ➤ **Operadores Lógicos:**

η Estos operadores se utilizan para establecer relaciones entre valores lógicos.

η Estos valores pueden ser resultado de una expresión relacional.

#### **Operadores Lógicos**

And	Y
Or	O
Not	Negación

#### **Operador And**

Operando1	Operador	Operando2	Resultado
T	AND	T	T
T		F	F
F		T	F
F		F	F

#### **Operador Or**

Operando1	Operador	Operando2	Resultado
T	OR	T	T
T		F	T
F		T	T
F		F	F

#### **Operador Not**

Operando	Resultado
T	F
F	T

Ejemplos:

$(a < b) \text{ and } (b < c)$   
 $(10 < 20) \text{ and } (20 < 30)$   
 T    and    T  
   └───┬───┘  
       T

### ***Prioridad de los Operadores Lógicos***

Not  
 And  
 Or

### ***Prioridad de los Operadores en General***

- 1.- ( )
- 2.- ^
- 3.- \*, /, Mod, Not
- 4.- +, -, And
- 5.- >, <, >=, <=, <>, =, Or

***Ejemplos:***

a = 10   b = 12   c = 13   d = 10

- 1)  $((a > b) \text{ or } (a < c)) \text{ and } ((a = c) \text{ or } (a >= b))$

      F            T            F            F  
       └─┬─┘        └─┬─┘  
           T            F

- 2)  $((a >= b) \text{ or } (a < d)) \text{ and } ((a >= d) \text{ and } (c > d))$

      F            F            T            T  
       └─┬─┘        └─┬─┘  
           F            T

- 3)  $\text{not } (a = c) \text{ and } (c > b)$

      └─┬─┘  
       F            T  
       └─┬─┘  
       T            T

Deber 3 ejercicios basicos.

## 2.4 Identificadores

Los *identificadores* representan los datos de un programa (constantes, variables, tipos de datos). Un identificador es una secuencia de caracteres que sirve para identificar una posición en la memoria de la computadora, que nos permite acceder a su contenido.

Ejemplo:      Nombre  
                  Num\_hrs  
                  Calif2

### *Reglas para formar un Identificador*

- η Debe comenzar con una letra (A a Z, mayúsculas o minúsculas) y no deben contener espacios en blanco.
- η Letras, dígitos y caracteres como la subraya ( \_ ) están permitidos después del primer carácter.
- η La longitud de identificadores puede ser de hasta 8 caracteres.

### *Constantes y Variables*

- **Constante:** Una constante es un dato numérico o alfanumérico que no cambia durante la ejecución del programa.

Ejemplo:

pi = 3.1416

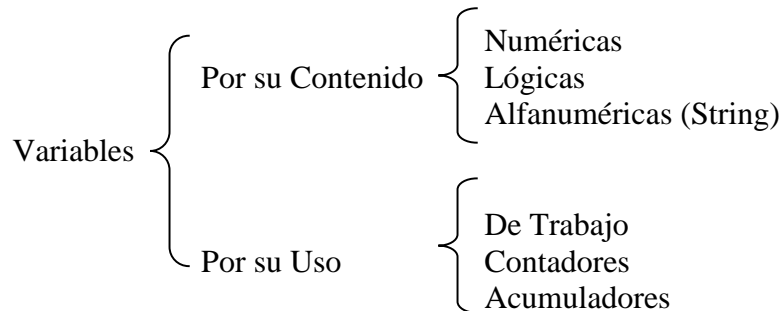
- **Variable:** Es un espacio en la memoria de la computadora que permite almacenar temporalmente un dato durante la ejecución de un proceso, su contenido puede cambiar durante la ejecución del programa. Para poder reconocer una variable en la memoria de la computadora, es necesario darle un nombre con el cual podamos identificarla dentro de un algoritmo.

Ejemplo:

área = pi \* radio ^ 2

Las variables son : el radio, el área y la constante es pi

### *Clasificación de las Variables*



#### *Por su Contenido*

➤ **Variable Numéricas:** Son aquellas en las cuales se almacenan valores numéricos, positivos o negativos, es decir almacenan números del 0 al 9, signos (+ y -) y el punto decimal. Ejemplo:

iva=0.15      pi=3.1416      costo=2500

➤ **Variables Lógicas:** Son aquellas que solo pueden tener dos valores (cierto o falso) estos representan el resultado de una comparación entre otros datos.

➤ **Variables Alfanuméricas:** Está formada por caracteres alfanuméricos (letras, números y caracteres especiales). Ejemplo:

letra='a'      apellido='lopez'      direccion='Av. Libertad #190'

#### *Por su Uso*

➤ **Variables de Trabajo:** Variables que reciben el resultado de una operación matemática completa y que se usan normalmente dentro de un programa. Ejemplo:

suma=a+b/c

➤ **Contadores:** Se utilizan para llevar el control del número de ocasiones en que se realiza una operación o se cumple una condición. Con los incrementos generalmente de uno en uno.

➤ **Acumuladores:** Forma que toma una variable y que sirve para llevar la suma acumulativa de una serie de valores que se van leyendo o calculando progresivamente.

### CAPITULO III. TÉCNICAS DE DISEÑO

3.1 Top down

3.2 Bottom up

#### OBJETIVO EDUCACIONAL:

El alumno:

- Conocerá las características de las técnicas de diseño más empleadas, así como su aplicación a cada tipo de problemas

### **3.1 Top Down**

También conocida como de arriba-abajo y consiste en establecer una serie de niveles de mayor a menor complejidad (arriba-abajo) que den solución al problema. Consiste en efectuar una relación entre las etapas de la estructuración de forma que una etapa jerárquica y su inmediato inferior se relacionen mediante entradas y salidas de información.

Este diseño consiste en una serie de descomposiciones sucesivas del problema inicial, que recibe el refinamiento progresivo del repertorio de instrucciones que van a formar parte del programa.

La utilización de la técnica de diseño **Top-Down** tiene los siguientes objetivos básicos:

- Simplificación del problema y de los subprogramas de cada descomposición.
- Las diferentes partes del problema pueden ser programadas de modo independiente e incluso por diferentes personas.
- El programa final queda estructurado en forma de bloque o módulos lo que hace más sencilla su lectura y mantenimiento.

### **3.2 Bottom Up**

El diseño ascendente se refiere a la identificación de aquellos procesos que necesitan computarizarse con forme vayan apareciendo, su análisis como sistema y su codificación, o bien, la adquisición de paquetes de software para satisfacer el problema inmediato.

Cuando la programación se realiza internamente y haciendo un enfoque ascendente, es difícil llegar a integrar los subsistemas al grado tal de que el desempeño global, sea fluido. Los problemas de integración entre los subsistemas son sumamente costosos y muchos de ellos no se solucionan hasta que la programación alcanza la fecha límite para la integración total del sistema. En esta fecha, ya se cuenta con muy poco tiempo, presupuesto o paciencia de los usuarios, como para corregir aquellas delicadas interfaces, que, en un principio, se ignoran.

Aunque cada subsistema parece ofrecer lo que se requiere, cuando se contempla al sistema como una entidad global, adolece de ciertas limitaciones por haber tomado un enfoque ascendente. Uno de ellos es la duplicación de esfuerzos para acceder el software y más aún al introducir los datos. Otro es, que se introducen al sistema muchos datos carentes de valor. Un tercero y tal vez el más serio inconveniente del enfoque ascendente, es que los objetivos globales de la organización no fueron considerados y en consecuencia no se satisfacen.

## TÉCNICAS PARA LA FORMULACIÓN DE ALGORITMOS

4.1 Diagrama de flujo

4.2 Pseudocódigo

### OBJETIVO EDUCACIONAL:

El alumno:

- Será capaz de diferenciar los métodos de representación y formulación de algoritmos, así como de conocer las características más importantes de cada técnica.

Las dos herramientas utilizadas comúnmente para diseñar algoritmos son:

Diagrama de Flujo


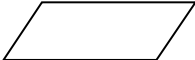

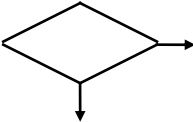


Pseudocódigo

#### ***4.1 Diagrama de Flujo***

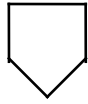
Un diagrama de flujo es la representación gráfica de un algoritmo. También se puede decir que es la representación detallada en forma gráfica de como deben realizarse los pasos en la computadora para producir resultados.

Esta representación gráfica se da cuando varios símbolos (que indican diferentes procesos en la computadora), se relacionan entre si mediante líneas que indican el orden en que se deben ejecutar los procesos.

Los símbolos utilizados han sido normalizados por el instituto norteamericano de normalización (ANSI).

<b><u>SÍMBOLO</u></b>	<b><u>DESCRIPCIÓN</u></b>
	Indica el inicio y el final de nuestro diagrama de flujo.
	Indica la entrada y salida de datos.
	Símbolo de proceso y nos indica la asignación de un valor en la memoria y/o la ejecución de una operación aritmética.
	Símbolo de decisión indica la realización de una comparación de valores.
	Se utiliza para representar los subprogramas.
	Conector dentro de página. Representa la continuidad del diagrama dentro de la misma pagina.

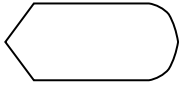




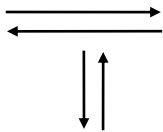
Conector fuera de página. Representa la continuidad del diagrama en otra página.



Indica la salida de información por impresora.



Indica la salida de información en la pantalla o monitor.



Líneas de flujo o dirección. Indican la secuencia en que se realizan las operaciones.

### ***Recomendaciones para el diseño de Diagramas de Flujo***

- Se deben usar solamente líneas de flujo horizontales y/o verticales.
- Se debe evitar el cruce de líneas utilizando los conectores.
- Se deben usar conectores solo cuando sea necesario.
- No deben quedar líneas de flujo sin conectar.
- Se deben trazar los símbolos de manera que se puedan leer de arriba hacia abajo y de izquierda a derecha.
- Todo texto escrito dentro de un símbolo deberá ser escrito claramente, evitando el uso de muchas palabras.

### ***4.2 Pseudocódigo***

Mezcla de lenguaje de programación y español (o inglés o cualquier otro idioma) que se emplea, dentro de la programación estructurada, para realizar el diseño de un programa. En esencial, el pseudocódigo se puede definir como un lenguaje de especificaciones de algoritmos.

Es la representación narrativa de los pasos que debe seguir un algoritmo para dar solución a un problema determinado. El pseudocódigo utiliza palabras que indican el proceso a realizar.

### ***Ventajas de utilizar un Pseudocódigo a un Diagrama de Flujo***

- Ocupa menos espacio en una hoja de papel
- Permite representar en forma fácil operaciones repetitivas complejas
- Es muy fácil pasar de pseudocódigo a un programa en algún lenguaje de programación.
- Si se siguen las reglas se puede observar claramente los niveles que tiene cada operación.

## **CAPITULO V.**

### **ESTRUCTURAS ALGORITMICAS**

#### **5.1 Secuenciales**

- Asignación
- Entrada
- Salida

#### **5.2 Condicionales**

- Simples
- Múltiples

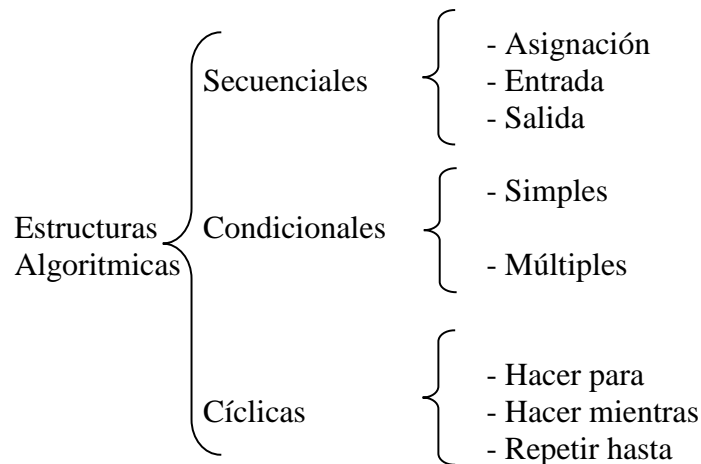
#### **5.3 Repetición fila condicional**

### **OBJETIVO EDUCACIONAL:**

El alumno:

- Conocerá las diferentes estructuras algorítmicas como componentes básicos de los programas y aplicará la combinación de ellas para el desarrollo de algoritmos más complejos.

Las estructuras de operación de programas son un grupo de formas de trabajo, que permiten, mediante la manipulación de variables, realizar ciertos procesos específicos que nos lleven a la solución de problemas. Estas estructuras se clasifican de acuerdo con su complejidad en:



### 5.1. Estructuras Secuenciales

La estructura secuencial es aquella en la que una acción (instrucción) sigue a otra en secuencia. Las tareas se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el fin del proceso. Una estructura secuencial se representa de la siguiente forma:

```
Inicio
Accion1
Accion2
.
.
AccionN
Fin
```

- **Asignación:** La asignación consiste, en el paso de valores o resultados a una zona de la memoria. Dicha zona será reconocida con el nombre de la variable que recibe el valor. La asignación se puede clasificar de la siguiente forma:

- **Simple:** Consiste en pasar un valor constante a una variable ( $a=15$ )
- **Contador:** Consiste en usarla como un verificador del número de veces que se realiza un proceso ( $a=a+1$ )
- **Acumulador:** Consiste en usarla como un sumador en un proceso ( $a=a+b$ )

- **De trabajo:** Donde puede recibir el resultado de una operación matemática que involucre muchas variables ( $a=c+b*2/4$ ).

- **Lectura:** La lectura consiste en recibir desde un dispositivo de entrada (p.ej. el teclado) un valor. Esta operación se representa en un pseudocódigo como sigue:

Leer a, b

Donde “a” y “b” son las variables que recibirán los valores

**Escritura:** Consiste en mandar por un dispositivo de salida (p.ej. monitor o impresora) un resultado o mensaje. Este proceso se representa en un pseudocódigo como sigue:

Escribe “El resultado es:”, R

Donde “El resultado es:” es un mensaje que se desea aparezca y R es una variable que contiene un valor.

### ***Problemas Secuenciales***

1) Suponga que un individuo desea invertir su capital en un banco y desea saber cuánto dinero ganará después de un mes si el banco paga a razón de 2% mensual.

Inicio

Leer cap\_inv

gan = cap\_inv \* 0.02

Imprimir gan

Fin

2) Un vendedor recibe un sueldo base más un 10% extra por comisión de sus ventas, el vendedor desea saber cuánto dinero obtendrá por concepto de comisiones por las tres ventas que realiza en el mes y el total que recibirá en el mes tomando en cuenta su sueldo base y comisiones.

Inicio

Leer sb, v1, v2, v3

tot\_vta = v1 + v2 + v3

com = tot\_vta \* 0.10

tpag = sb + com

Imprimir tpag, com

Fin

3) Una tienda ofrece un descuento del 15% sobre el total de la compra y un cliente desea saber cuánto deberá pagar finalmente por su compra.

Inicio

Leer tc

d = tc \* 0.15

tp = tc - d

Imprimir tp

Fin

4) Un alumno desea saber cuál será su calificación final en la materia de Algoritmos. Dicha calificación se compone de los siguientes porcentajes:

55% del promedio de sus tres calificaciones parciales.

30% de la calificación del examen final.

15% de la calificación de un trabajo final.

Inicio

Leer c1, c2, c3, ef, tf

$\text{prom} = (c1 + c2 + c3)/3$

$\text{ppar} = \text{prom} * 0.55$

$\text{pef} = \text{ef} * 0.30$

$\text{ptf} = \text{tf} * 0.15$

$\text{cf} = \text{ppar} + \text{pef} + \text{ptf}$

Imprimir cf

Fin

5) Un maestro desea saber qué porcentaje de hombres y que porcentaje de mujeres hay en un grupo de estudiantes.

Inicio

Leer nh, nm

$\text{ta} = \text{nh} + \text{nm}$

$\text{ph} = \text{nh} * 100 / \text{ta}$

$\text{pm} = \text{nm} * 100 / \text{ta}$

Imprimir ph, pm

Fin

6) Realizar un algoritmo que calcule la edad de una persona.

Inicio

Leer fnac, fact

$\text{edad} = \text{fact} - \text{fnac}$

Imprimir edad

Fin.

## **SEMANA 4: DEL 22 AL 26 DE JUNIO/2020**

### **5.2 Estructuras de Condicionales**

Las estructuras condicionales comparan una variable contra otro(s) valor(es), para que, en base al resultado de esta comparación, se siga un curso de acción dentro del programa. Cabe mencionar que la comparación se puede hacer contra otra variable o contra una constante, según se necesite. Existen dos tipos básicos, las simples y las múltiples.

- **Simples:** Las estructuras condicionales simples se les conoce como “Tomas de decisión”. Estas tomas de decisión tienen la siguiente forma:

Si <condición> entonces  
 Acción(es)  
 Fin-si

- **Dobles:** Las estructuras condicionales dobles permiten elegir entre dos opciones o alternativas posibles en función del cumplimiento o no de una determinada condición. Se representa de la siguiente forma:

Si <condición> entonces  
 Acción(es)  
 si no  
 Acción(es)  
 Fin-si

Donde:

Si .....	Indica el comando de comparación n
Condición.....	Indica la condición a evaluar
entonces.....	Precede a las acciones a realizar cuando se cumple la condición
acción(es).....	Son las acciones a realizar cuando se cumple o no la condición
si no.....	Precede a las acciones a realizar cuando no se cumple la condición

Dependiendo de si la comparación es cierta o falsa, se pueden realizar una o más acciones.

- **Múltiples:** Las estructuras de comparación múltiples, son tomas de decisión especializadas que permiten comparar una variable contra distintos posibles resultados, ejecutando para cada caso una serie de instrucciones específicas. La forma común es la siguiente:

Si <condición> entonces  
 Acción(es)  
 si no  
 Si <condición> entonces  
 Acción(es)  
 si no  
 .  
 .  
 .  
 > Varias condiciones

- **Forma General**

Casos Variable  
 Op1: Acción(es)  
 Op2: Acción(es)  
 .  
 .  
 OpN: acción  
 Fin-casos

### ***Problemas Condicionales***

#### ***a) Problemas Selectivos Simples***

1) Un hombre desea saber cuánto dinero se genera por concepto de intereses sobre la cantidad que tiene en inversión en el banco. El decidirá reinvertir los intereses siempre y cuando estos excedan a \$7000, y en ese caso desea saber cuánto dinero tendrá finalmente en su cuenta.

```

Inicio
  Leer p_int, cap
  int = cap * p_int
  si int > 7000 entonces
    capf = cap + int
  fin-si
  Imprimir capf
fin
  
```

2) Determinar si un alumno aprueba a reprueba un curso, sabiendo que aprobará si su promedio de tres calificaciones es mayor o igual a 70; reprueba en caso contrario.

```

Inicio
  Leer calif1, calif2, calif3
  prom = (calif1 + calif2 + calif3)/3
  Si prom >= 70 entonces
    Imprimir "alumno aprobado"
  si no
    Imprimir "alumno reprobado"
  Fin-si
Fin
  
```

3) En un almacén se hace un 20% de descuento a los clientes cuya compra supere los \$1000  
 ¿Cuál será la cantidad que pagara una persona por su compra?

```

Inicio
  Leer compra
  Si compra > 1000 entonces
    desc = compra * 0.20
  si no
    desc = 0
  
```

```

    fin-si
    tot_pag = compra - desc
    imprimir tot_pag
fin.

```

- 4) Un obrero necesita calcular su salario semanal, el cual se obtiene de la sig. manera:  
 Si trabaja 40 horas o menos se le paga \$16 por hora  
 Si trabaja más de 40 horas se le paga \$16 por cada una de las primeras 40 horas y \$20 por cada hora extra.

```

Inicio
Leer ht
Si ht > 40 entonces
    he = ht - 40
    ss = he * 20 + 40 * 16
si no
    ss = ht * 16
Fin-si
Imprimir ss
Fin

```

- 5) Un hombre desea saber cuánto dinero se genera por concepto de intereses sobre la cantidad que tiene en inversión en el banco. El decidirá reinvertir los intereses siempre y cuando estos excedan a \$7000, y en ese caso desea saber cuánto dinero tendrá finalmente en su cuenta.

```

Inicio
Leer p_int, cap
int = cap * p_int
si int > 7000 entonces
    capf = cap + int
fin-si
Imprimir capf
fin

```

- 6) Que lea dos números y los imprima en forma ascendente

```

Inicio
Leer num1, num2
Si num1 < num2 entonces
    Imprimir num1, num2
si no
    Imprimir num2, num1
fin-si
fin

```

- 7) Una persona enferma, que pesa 70 kg, se encuentra en reposo y desea saber cuántas calorías consume su cuerpo durante todo el tiempo que realice una misma actividad. Las actividades que tiene permitido realizar son únicamente dormir o estar sentado en reposo.



Los datos que tiene son que estando dormido consume 1.08 calorías por minuto y estando sentado en reposo consume 1.66 calorías por minuto.

```
Inicio
  Leer act$, tiemp
  Si act$ = "dormido" entonces
     $cg = 1.08 * tiemp$ 
  si no
     $cg = 1.66 * tiemp$ 
  fin-si
  Imprimir cg
Fin
```

8) Hacer un algoritmo que imprima el nombre de un artículo, clave, precio original y su precio con descuento. El descuento lo hace en base a la clave, si la clave es 01 el descuento es del 10% y si la clave es 02 el descuento en del 20% (solo existen dos claves).

```
Inicio
  Leer nomb, cve, prec_orig
  Si cve = 01 entonces
     $prec\_desc = prec\_orig - prec\_orig * 0.10$ 

    si no
       $prec\_desc = prec\_orig - prec\_orig * 0.20$ 
  fin-si
  Imprimir nomb, cve, prec_orig, prec_desc
fin
```

9) Hacer un algoritmo que calcule el total a pagar por la compra de camisas. Si se compran tres camisas o más se aplica un descuento del 20% sobre el total de la compra y si son menos de tres camisas un descuento del 10%

```
Inicio
  Leer num_camisas, prec
   $tot\_comp = num\_camisas * prec$ 
  Si num_camisas  $\geq 3$  entonces
     $tot\_pag = tot\_comp - tot\_comp * 0.20$ 
  si no
     $tot\_pag = tot\_comp - tot\_comp * 0.10$ 
  fin-si
  Imprimir tot_pag
fin
```

10) Una empresa quiere hacer una compra de varias piezas de la misma clase a una fábrica de refacciones. La empresa, dependiendo del monto total de la compra, decidirá qué hacer para pagar al fabricante.

Si el monto total de la compra excede de \$500 000 la empresa tendrá la capacidad de invertir de su propio dinero un 55% del monto de la compra, pedir prestado al banco un 30% y el resto lo pagará solicitando un crédito al fabricante.

Si el monto total de la compra no excede de \$500 000 la empresa tendrá capacidad de invertir de su propio dinero un 70% y el restante 30% lo pagará solicitando crédito al fabricante.

El fabricante cobra por concepto de intereses un 20% sobre la cantidad que se le pague a crédito.

```
Inicio
  Leer costopza, numpza
  totcomp = costopza * numpza
  Si totcomp > 500 000 entonces
    cantinv = totcomp * 0.55
    préstamo = totcomp * 0.30
    crédito = totcomp * 0.15
  si no
    cantinv = totcomp * 0.70
    crédito = totcomp * 0.30
    préstamo = 0
  fin-si
  int = crédito * 0.20
  Imprimir cantinv, préstamo, crédito, int
Fin
```

## LA ESTRUCTURA CONDICIONAL IF...ELSE

es la que nos permite tomar ese tipo de decisiones. Traducida literalmente del inglés, se la podría llamar la estructura "si...si no", es decir, "si se cumple la condición, haz esto, y sino, haz esto otro"

```
if (edad < 18)
    printf("No puedes acceder.\n");
else
    printf("Bienvenido.\n");
```

### *Operadores de comparación*

El símbolo > visto en el último ejemplo es un **operador**, que en este caso compara dos números enteros y devuelve verdadero si el primero es mayor, falso en caso contrario.

A continuación un listado de los posibles operadores de comparación en C y su significado.

Operadores de Comparación

Operador	Significado
<	estrictamente menor que
>	estrictamente mayor que
<=	menor o igual que
>=	mayor o igual que
==	igual a
!=	distinto de

Por ejemplo: determinar si un año es bisiesto o no. Los años son bisiestos si son divisibles por 4, pero no si son divisibles por 100, a menos que también sean divisibles por 400.

```
if ( !(a % 4) && (a % 100) || !(a % 400) ) {  
    printf("es un año bisiesto.\n");  
} else {  
    printf("no es un año bisiesto.\n");  
}
```

### ***Problemas Selectivos Compuestos***

- 1) Leer 2 números; si son iguales que los multiplique, si el primero es mayor que el segundo que los reste y si no que los sume.

```

Inicio
  Leer num1, num2
  si num1 = num2 entonces
    resul = num1 * num2
  si no
    si num1 > num2 entonces
      resul = num1 - num2
    si no
      resul = num1 + num2
    fin-si
  fin-si
fin

```

2) Leer tres números diferentes e imprimir el número mayor de los tres.

```

Inicio
  Leer num1, num2, num3
  Si (num1 > num2) and (num1 > num3) entonces
    mayor = num1
  si no
    Si (num2 > num1) and (num2 > num3) entonces
      mayor = num2
    si no
      mayor = num3
    fin-si
  fin-si
  Imprimir mayor
fin

```

3) Determinar la cantidad de dinero que recibirá un trabajador por concepto de las horas extras trabajadas en una empresa, sabiendo que cuando las horas de trabajo exceden de 40, el resto se consideran horas extras y que estas se pagan al doble de una hora normal cuando no exceden de 8; si las horas extras exceden de 8 se pagan las primeras 8 al doble de lo que se pagan las horas normales y el resto al triple.

```

Inicio
  Leer ht, pph
  Si ht <= 40 entonces
    tp = ht * pph
  si no
    he = ht - 40
    Si he <= 8 entonces
      pe = he * pph * 2
    si no
      pd = 8 * pph * 2
      pt = (he - 8) * pph * 3
      pe = pd + pt

```

```

    fin-si
    tp = 40 * pph + pe
  fin-si
  Imprimir tp
fin

```

4) Calcular la utilidad que un trabajador recibe en el reparto anual de utilidades si este se le asigna como un porcentaje de su salario mensual que depende de su antigüedad en la empresa de acuerdo con la sig. tabla:

Tiempo	Utilidad
Menos de 1 año	5 % del salario
1 año o mas y menos de 2 años	7% del salario
2 años o mas y menos de 5 años	10% del salario
5 años o mas y menos de 10 años	15% del salario
10 años o mas	20% del salario

```

Inicio
  Leer sm, antig
  Si antig < 1 entonces
    util = sm * 0.05
  si no
    Si (antig >= 1) and (antig < 2) entonces
      util = sm * 0.07
    si no
      Si (antig >= 2) and (antig < 5) entonces
        util = sm * 0.10
      si no
        Si (antig >= 5) and (antig < 10) entonces
          util = sm * 0.15
        si no
          util = sm * 0.20
        fin-si
      fin-si
    fin-si
  fin-si
  Imprimir util
fin

```

5) En una tienda de descuento se efectúa una promoción en la cual se hace un descuento sobre el valor de la compra total según el color de la bolita que el cliente saque al pagar en caja. Si la bolita es de color blanco no se le hará descuento alguno, si es verde se le hará un 10% de descuento, si es amarilla un 25%, si es azul un 50% y si es roja un 100%. Determinar la cantidad final que el cliente deberá pagar por su compra. se sabe que solo hay bolitas de los colores mencionados.

```

Inicio
  leer tc, b$
  si b$ = 'blanca' entonces

```

```

    d=0
  si no
  si b$ = 'verde' entonces
    d=tc*0.10
  si no
  si b$ = 'amarilla' entonces
    d=tc*0.25
  si no
  si b$ = 'azul' entonces
    d=tc*0.50
  si no
    d=tc
  fin-si
fin-si
fin-si
fin-si
fin

```

6) El IMSS requiere clasificar a las personas que se jubilaran en el año de 1997. Existen tres tipos de jubilaciones: por edad, por antigüedad joven y por antigüedad adulta. Las personas adscritas a la jubilación por edad deben tener 60 años o más y una antigüedad en su empleo de menos de 25 años. Las personas adscritas a la jubilación por antigüedad joven deben tener menos de 60 años y una antigüedad en su empleo de 25 años o más.

Las personas adscritas a la jubilación por antigüedad adulta deben tener 60 años o más y una antigüedad en su empleo de 25 años o más.

Determinar en que tipo de jubilación, quedara adscrita una persona.

Inicio

```

  leer edad,ant
  si edad >= 60 and ant < 25 entonces
    imprimir "la jubilación es por edad"
  si no
    si edad >= 60 and ant > 25 entonces
      imprimir "la jubilación es por edad adulta"
    si no
      si edad < 60 and ant > 25 entonces
        imprimir "la jubilación es por antigüedad joven"
      si no
        imprimir "no tiene por que jubilarse"
      fin-si
    fin-si
  fin-si
fin

```

## ***La estructura condicional abierta y cerrada switch ... case***

se utiliza cuando queremos evitarnos las llamadas escaleras de decisiones. La estructura `if` nos puede proporcionar, únicamente, dos resultados, uno para verdadero y otro para falso. Una estructura `switch ... case`, por su parte, nos permite elegir entre muchas opciones. Ejemplo:

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    int dia;

    printf("¿Qué número de día de la semana es?");
    scanf("%i",&dia);

    switch(dia) {
        case 1 :
            printf("Lun, Lunes");
            break;
        case 2 :
            printf("Mar, Martes");
            break;
        case 3 :
            printf("Mier, Miercoles");
            break;
        case 4 :
            printf("Jue, Jueves");
            break;
        case 5 :
            printf("Vie, Viernes");
            break;
        case 6 :
            printf("Sab, Sabado");
            break;
        case 7 :
            printf("Dom, Domingo");
            break;
        default :
            printf("No existe");
    }
    return 0;
}
```

la sentencia `switch` proporciona una ganancia en velocidad del código, pues permite al compilador trabajar en base a que se trata de una decisión múltiple para una única variable, cosa que con sentencias `if` el compilador no tiene por qué detectar.

Como vemos, para cada valor de la variable se ejecuta un bloque de sentencias distinto, en el que no necesitamos llaves. Hay un caso especial, `default`, que se ejecuta si ningún otro corresponde, y que no es necesario poner

**Estructuras Cíclicas o repetitivas**

Se llaman problemas repetitivos o cíclicos a aquellos en cuya solución es necesario utilizar un mismo conjunto de acciones que se puedan ejecutar una cantidad específica de veces. Esta cantidad puede ser fija (previamente determinada por el programador) o puede ser variable (estar en función de algún dato dentro del programa). Los ciclos se clasifican en:

- **Ciclos con un Número Determinado de Iteraciones (Hacer-Para)**

Son aquellos en que el número de iteraciones se conoce antes de ejecutarse el ciclo. La forma de esta estructura es la siguiente:

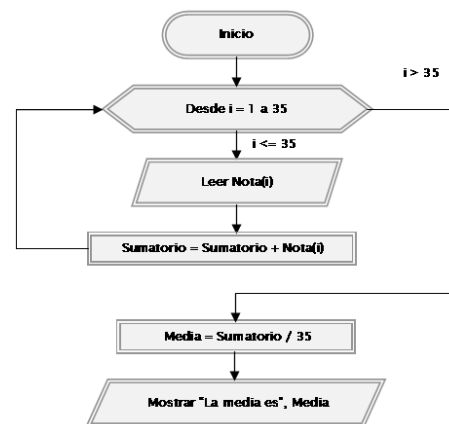
Hacer para V.C = L.I a L.S

Accion1

.

.

AccionN



Donde:

V.C    Variable de control del ciclo  
L.I    Limite inferior  
L.S    Límite superior

En este ciclo la variable de control toma el valor inicial del ciclo y el ciclo se repite hasta que la variable de control llegue al límite superior.

**Problemas ( Hacer para )**

1) Calcular el promedio de un alumno que tiene 7 calificaciones en la materia de Diseño Estructurado de Algoritmos

Inicio  
Sum=0  
Leer Nom  
Hacer para c = 1 a 7  
    Leer calif  
    Sum = sum + calif  
Fin-para  
prom = sum /7  
Imprimir prom  
Fin.



2) Leer 10 números y obtener su cubo y su cuarta.

```
Inicio
  Hacer para n = 1 a 10
    Leer num
    cubo = num * num * num
    cuarta = cubo * num
    Imprimir cubo, cuarta
  Fin-para
Fin.
```

3) Leer 10 números e imprimir solamente los números positivos

```
Inicio
  Hacer para n = 1 a 10
    Leer num
    Si num > 0 entonces
      Imprimir num
    fin-si
  Fin-para
Fin.
```

4) Leer 20 números e imprimir cuantos son positivos, cuantos negativos y cuantos neutros.

```
Inicio
  cn = 0
  cp = 0
  cneg = 0
  Hacer para x = 1 a 20
    Leer num
    Sin num = 0 entonces
      cn = cn + 1
    si no
      Si num > 0 entonces
        cp = cp + 1
      si no
        cneg = cneg + 1
    Fin-si
  Fin-si
  Fin-para
  Imprimir cn, cp, cneg
Fin.
```

5) Leer 15 números negativos y convertirlos a positivos e imprimir dichos números.

```
Inicio
  Hacer para x = 1 a 15
```

```

        Leer num
        pos = num * -1
        Imprimir num, pos
    Fin-para
Fin.

```

6) Suponga que se tiene un conjunto de calificaciones de un grupo de 40 alumnos. Realizar un algoritmo para calcular la calificación media y la calificación mas baja de todo el grupo.

```

Inicio
    sum = 0
    baja = 9999
    Hacer para a = 1 a 40
        Leer calif
        sum = sum + calif
        Si calif < baja entonces
            baja = calif
        fin-si
    Fin-para
    media = sum / 2
    Imprimir media, baja
fin

```

7) Calcular e imprimir la tabla de multiplicar de un numero cualquiera. Imprimir el multiplicando, el multiplicador y el producto.

```

Inicio
    Leer num
    Hacer para X = 1 a 10
        resul = num * x
        Imprimir num, " * ", X, " = ", resul
    Fin-para
fin.

```

8) Simular el comportamiento de un reloj digital, imprimiendo la hora, minutos y segundos de un día desde las 0:00:00 horas hasta las 23:59:59 horas

```

Inicio
    Hacer para h = 1 a 23
        Hacer para m = 1 a 59
            Hacer para s = 1 a 59
                Imprimir h, m, s
            Fin-para
        Fin-para
    Fin-para
fin.

```

El bucle `for` es un bucle muy flexible y a la vez muy potente ya que tiene varias formas interesantes de implementarlo, su forma más tradicional es la siguiente:

```
for (/* inicialización */; /* condición */; /* incremento */) {  
    /* código a ejecutar */  
}
```

**Inicialización:** en esta parte se inicia la variable que controla el bucle y es la primera sentencia que ejecuta el bucle. Sólo se ejecuta una vez ya que solo se necesita al principio del bucle.

**Expresión condicional:** al igual que en el bucle `while`, esta expresión determina si el bucle continuará ejecutándose o no.

**Incremento:** es una sentencia que ejecuta al final de cada iteración del bucle. Por lo general, se utiliza para incrementar la variable con que se inicio el ciclo. Luego de ejecutar el incremento, el bucle revisa nuevamente la condición, si es verdadera tiene lugar una ejecución más del cuerpo del ciclo, si es falsa se termina el ciclo y así.

Aquí se muestra el mismo ejemplo visto para el bucle `while`, pero implementado con un bucle `for`:

```
int i;  
for (i=0; i < 100; i = i + 1) {  
    printf("%d\n", i);  
}
```

## SEMANA 7: DEL 13 AL 17 DE JULIO/2020

### ***Ciclos con un Número Indeterminado de Iteraciones ( Hacer-Mientras, Repetir-Hasta)***

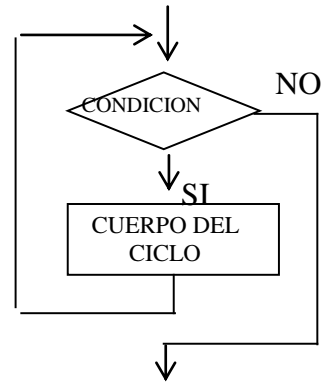
Son aquellos en que el número de iteraciones no se conoce con exactitud, ya que esta dado en función de un dato dentro del programa.

- ***Hacer-Mientras:*** Esta es una estructura que repetirá un proceso durante “N” veces, donde “N” puede ser fijo o variable. Para esto, la instrucción se vale de una condición que es la que debe cumplirse para que se siga ejecutando. Cuando la condición ya no se cumple, entonces ya no se ejecuta el proceso. La forma de esta estructura es la siguiente:

```

Hacer mientras <condición>
    Accion1
    Accion2
    .
    .
    AccionN
Fin-mientras

```



### ***Problema (Hacer Mientras)***

1) Una compañía de seguros tiene contratados a  $n$  vendedores. Cada uno hace tres ventas a la semana. Su política de pagos es que un vendedor recibe un sueldo base, y un 10% extra por comisiones de sus ventas. El gerente de su compañía desea saber cuánto dinero obtendrá en la semana cada vendedor por concepto de comisiones por las tres ventas realizadas, y cuanto tomando en cuenta su sueldo base y sus comisiones.

Inicio

```

Leer sueldo base
Leer n vendedores
C= contador
Hacer mientras c <= n
    Leer venta1, venta2, venta3
    Comision1=venta1*10%
    Comisión 2= venta2*10%
    Comision3=venta3*10%
    Escribir comisión total=comision1+comision2+comision3
    Escribir sueldo base +comisión total
Fin-mientras
fin.

```

### **BUCLE WHILE**

El bucle while sirve para ejecutar código reiteradas veces.

```

while (/*condicion*/) {
    /* Código */
}

```

La condición debe de ser una expresión lógica, similar a la de la sentencia `if`. Primero se evalúa la condición. Si el resultado es verdadero, se ejecuta el bloque de código. Luego se

vuelve a evaluar la condición, y en caso de dar verdadero se vuelve a ejecutar el bloque. El bucle se corta cuando la condición da falso.

Ejemplo: imprimir los números de 0 a 99:

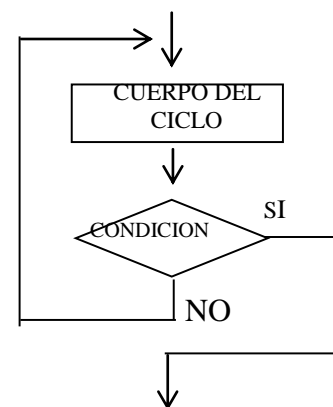
```
int i = 0;
while (i < 100) {
    printf("%d\n", i);
    i = i + 1;
}
```

Inicialmente se declara que la variable *i* tiene un valor de 0. Al iniciar el bucle, se cumple la condición *i < 100*, por lo que se procede a la instrucción de imprimir dicho número (cero, en el caso inicial). Posteriormente *i* cambiará su valor de uno en uno por la instrucción *i = i + 1* y seguidamente dicho valor nuevo, será evaluado en la condición *while* hasta que *i* llegue al valor 100, donde debido a la condicional, éste será un valor falso, dando fin al código.

## SEMANA 8: 20 AL 24 DE JULIO/2020

- ***Repetir-Hasta:*** Esta es una estructura similar en algunas características, a la anterior. Repite un proceso una cantidad de veces, pero a diferencia del Hacer-Mientras, el Repetir-Hasta lo hace hasta que la condición se cumple y no mientras, como en el Hacer-Mientras. Por otra parte, esta estructura permite realizar el proceso cuando menos una vez, ya que la condición se evalúa al final del proceso, mientras que en el Hacer-Mientras puede ser que nunca llegue a entrar si la condición no se cumple desde un principio. La forma de esta estructura es la siguiente:

Repetir  
    Accion1  
    Accion2  
    .  
    .  
    AccionN  
Hasta <condición>



### ***Problemas Repetir - Hasta***

- 1) En una tienda se desea conocer el salario de n trabajadores, considerando que por sus ventas reciben el 10% como comisión por ventas, el programa terminara cuando el propietario ya no quiera calcular más salarios.

Inicio

Repetir

Leer salario

Sal\_fin =salario\*10%

Escribir “El salario con aumento es “, sal\_fin

Escribir “hay otro empleado”

Leer empleado

Hasta que empleado= “n”

Fin

EN EL LENGUAJE C+ : El bucle do...while es un bucle que, por lo menos, se ejecuta una vez. Do significa literalmente "hacer", y while significa "mientras".

Su forma es esta:

```
do {  
    /* CODIGO */  
} while (/* Condición de ejecución del bucle */)
```

### **EJEMPLO**

```
int aleatorio;  
do {  
    aleatorio = rand();  
} while (aleatorio != 25);
```

## **CAPITULO VI.**

### **ARREGLOS**

6.1 Vectores

6.2 Matrices

6.3 Manejo de cadenas de caracteres

## OBJETIVO EDUCACIONAL:

El alumno:

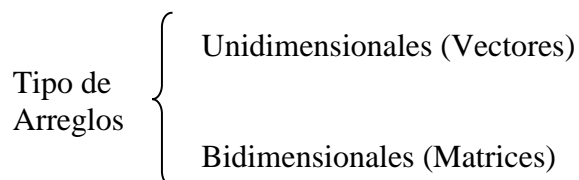
- Será capaz de utilizar los datos de tipo arreglo para plantear la solución de problemas que requieran de esta estructura.

## SEMANA 9: DEL 27 AL 31 DE JULIO/2020

**Arreglo:** Un *Arreglo* es una estructura de datos que almacena bajo el mismo nombre (variable) a una colección de datos del mismo tipo.

Los arreglos se caracterizan por:

- Almacenan los elementos en posiciones contiguas de memoria
- Tienen un mismo nombre de variable que representa a todos los elementos. Para hacer referencia a esos elementos es necesario utilizar un índice que especifica el lugar que ocupa cada elemento dentro del archivo.



### 6.1. Vectores

Es un arreglo de “N” elementos organizados en una dimensión donde “N” recibe el nombre de longitud o tamaño del vector. Para hacer referencia a un elemento del vector se usa el nombre del mismo, seguido del índice (entre corchetes), el cual indica una posición en particular del vector. Por ejemplo:

Vec[x]

Donde:

Vec..... Nombre del arreglo

x..... Numero de datos que constituyen el arreglo

#### *Representación gráfica de un vector*

Vec[1]	7
Vec[2]	8
Vec[3]	9
Vec[4]	1
	0

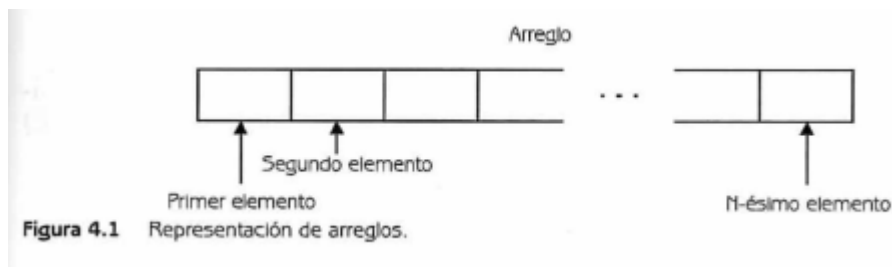
Arreglos unidimensionales



Un arreglo se define como una colección finita, homogénea y ordenada de elementos.

Finita:	todo arreglo tiene un límite, es decirse debe determinar cuál será el número máximo de elementos que podrán formar parte del arreglo.
Homogénea:	todos los elementos de un arreglo son del mismo tipo (todos enteros, todos reales, etc., pero nunca una combinación de distintos tipos).
Ordenada:	se puede determinar cuál es el primer elemento, el segundo, el tercero,... y el n-ésimo elemento.

Un arreglo puede representarse gráficamente como se muestra en la figura 4.1.



Un arreglo tiene la característica de que puede almacenar a N elementos del mismo tipo y además permite el acceso a cada uno de estos elementos. Así, se distinguen dos partes en los arreglos:

Con tipo se declara el tipo de datos para todos los elementos del arreglo. El tipo de los elementos no tiene que ser necesariamente el mismo que el de los índices.

Observaciones:

- El tipo del índice puede ser cualquier tipo ordinal (carácter, entero, etc.).
- El tipo de los componentes puede ser cualquier tipo (entero, real, cadena de caracteres, registro, arreglo, etc.).
- Se utilizan los corchetes “[ ]” para indicar el índice de un arreglo. Entre los [ ] se debe escribir un valor ordinal (puede ser una variable, una constante o una expresión tan compleja como se quiera, pero que dé como resultado un valor ordinal).

Se verán a continuación algunos ejemplos de arreglos:

#### Ejemplo 4.3

Sea ARRE un arreglo de 70 elementos enteros con índices enteros. Su representación queda como se muestra en la figura 4.3.



Figura 4.3

- $NTE = (70 - 1 + 1) = 70$
- Cada elemento del arreglo ARRE será un número entero y podrá accesearse por medio de un índice que será un valor comprendido entre 1 y 70.

Así por ejemplo:

- ARRE[1] hace referencia al elemento de la posición 1.
- ARRE[2] hace referencia al elemento de la posición 2.
- ...
- ...
- ARRE[70] hace referencia al elemento de la posición 70.

### ***Llenado de un Vector***

- Hacer para I = 1 a 10  
Leer vec[I]  
Fin-para
- Hacer mientras I <= 10  
Leer vec[I]  
Fin-mientras
- I=1  
Repetir  
Leer vec[I]  
I = I + 1  
Hasta-que I > 10

## **SEMANA 10: 3 AL 7 DE AGOSTO/2020**

### **Operaciones con arreglos**

A continuación, se presentan las operaciones más comunes en arreglos:

- Lectura/Escritura.

- Asignación.
- Actualización: Inserción.  
Eliminación.  
Modificación.
- Ordenación.
- Búsqueda.

Como los arreglos son datos estructurados, muchas de estas operaciones no pueden llevarse a cabo de manera global, sino que se debe trabajar sobre cada elemento.

A continuación, se analizará cada una de estas operaciones. Ordenación y búsqueda, serán analizadas en los problemas que presentaremos posteriormente.

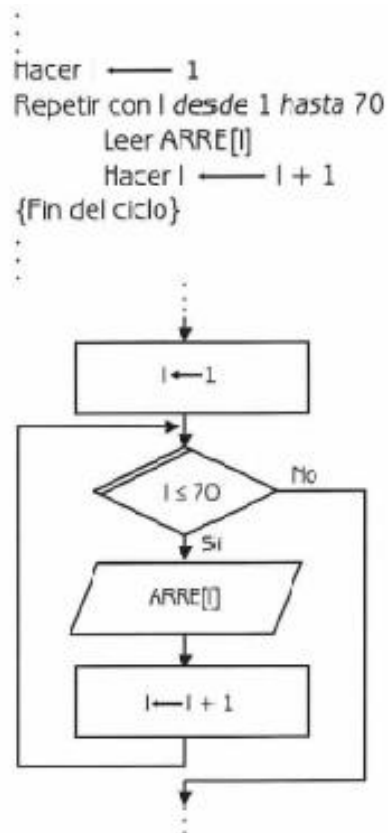
### Lectura

El proceso de lectura de un arreglo consiste en leer y asignar un valor a cada uno de sus elementos. Supóngase que se desea leer todos los elementos del arreglo ARRE presentado en el ejemplo 4.3, en forma consecutiva. Podría hacerse de la siguiente manera:

Leer ARRE[1],  
Leer ARRE [2],  
.....  
Leer ARRE[70]

ARRE[1], ARRE[2], . . . , ARRE[70]

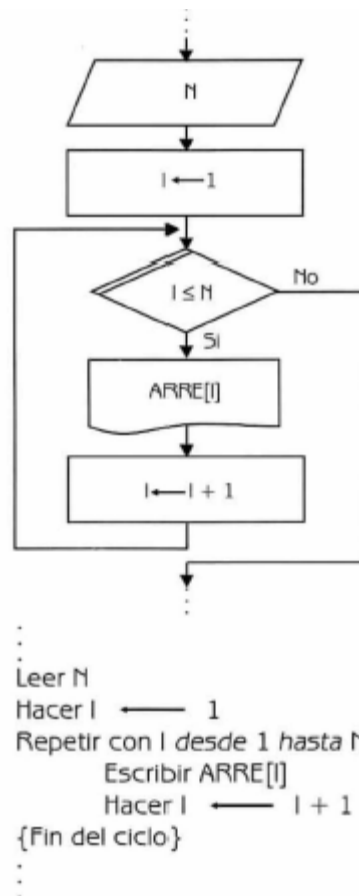
De esta forma no resulta práctico, por lo tanto, se usará un ciclo para leer todos los elementos del arreglo.



Al variar el valor de I, cada elemento leído se asigna al correspondiente componente del arreglo según la posición indicada por I.

### Escritura

El caso de escritura es similar al de lectura. Se debe escribir el valor de cada uno de los componentes. Supóngase que se desea escribir los primeros N componentes del arreglo ARRE (ejemplo 4.3) en forma consecutiva. Los pasos a seguir son los siguientes:



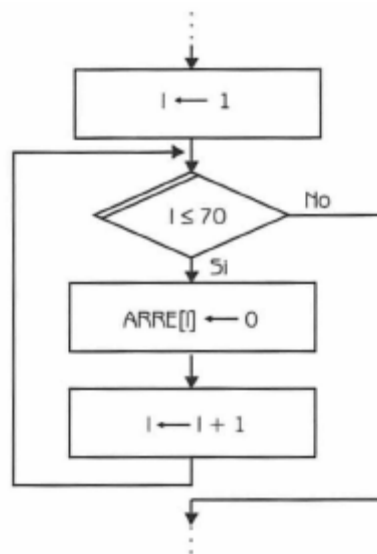
### Asignación

En general no es posible asignar directamente un valor a todo el arreglo, sino que se debe asignar el valor deseado a cada componente. En seguida se analizan algunos ejemplos de asignación.

En los dos primeros casos se asigna un valor a una determinada casilla del arreglo (en el primero a la señalada por el índice 1 y en el segundo a la indicada por el índice 3).

$\text{ARRE}[1] \leftarrow 120$   
 $\text{ARRE}[3] \leftarrow \text{ARRE}[1]/4$

En el tercer caso se asigna el 0 a todas las casillas del arreglo, con lo que éste queda como se muestra en la figura 4.7.



### Actualización

En un arreglo se pueden insertar, eliminar y/o modificar elementos. Para llevar a cabo estas operaciones eficientemente se debe tener en cuenta si el arreglo está ordenado o desordenado. Es decir, si sus componentes respetan algún orden entre sí. Las operaciones de inserción, eliminación y modificación serán tratadas separadamente para arreglos desordenados y ordenados.

#### a) ARREGLOS DESORDENADOS

Considere un arreglo A de 100 elementos como el presentado en la figura 4.8.

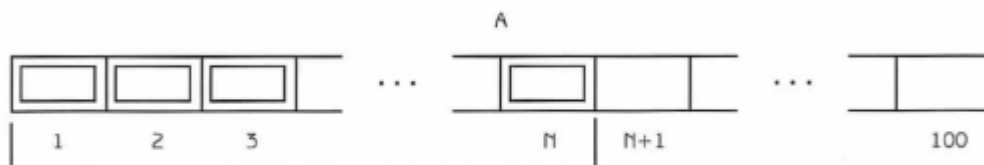


Figura 4.8 Actualización de arreglos desordenados.

La figura indica que los primeros N elementos tienen asignado un valor.

**Inserción:** Para insertar un elemento Y en un arreglo A desordenado debe verificarse que exista espacio. Si se cumple esta condición, entonces se asignará a la posición N + 1 el nuevo elemento.

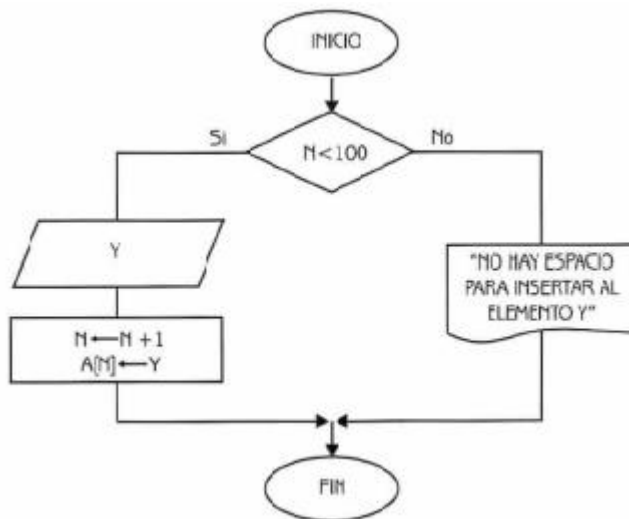
A continuación, presentamos el diagrama de flujo (4.3) correspondiente.

Explicación de las variables

N: Variable de tipo entero. Almacena el número actual de elementos del arreglo.

Y: Variable de tipo entero. **Representa al valor que se va a insertar.**

**A: Arreglo unidimensional** de tipo entero. Su capacidad máxima es de 100 elementos.



A continuación, presentamos el programa correspondiente.

### Programa 4.3

INSERTA\_DESORDENADO

{El programa inserta un elemento en un arreglo desordenado}.

{N y Y son variables de tipo entero. A es un arreglo unidimensional de tipo entero}

1. Si  $N < 100$

entonces

Leer Y

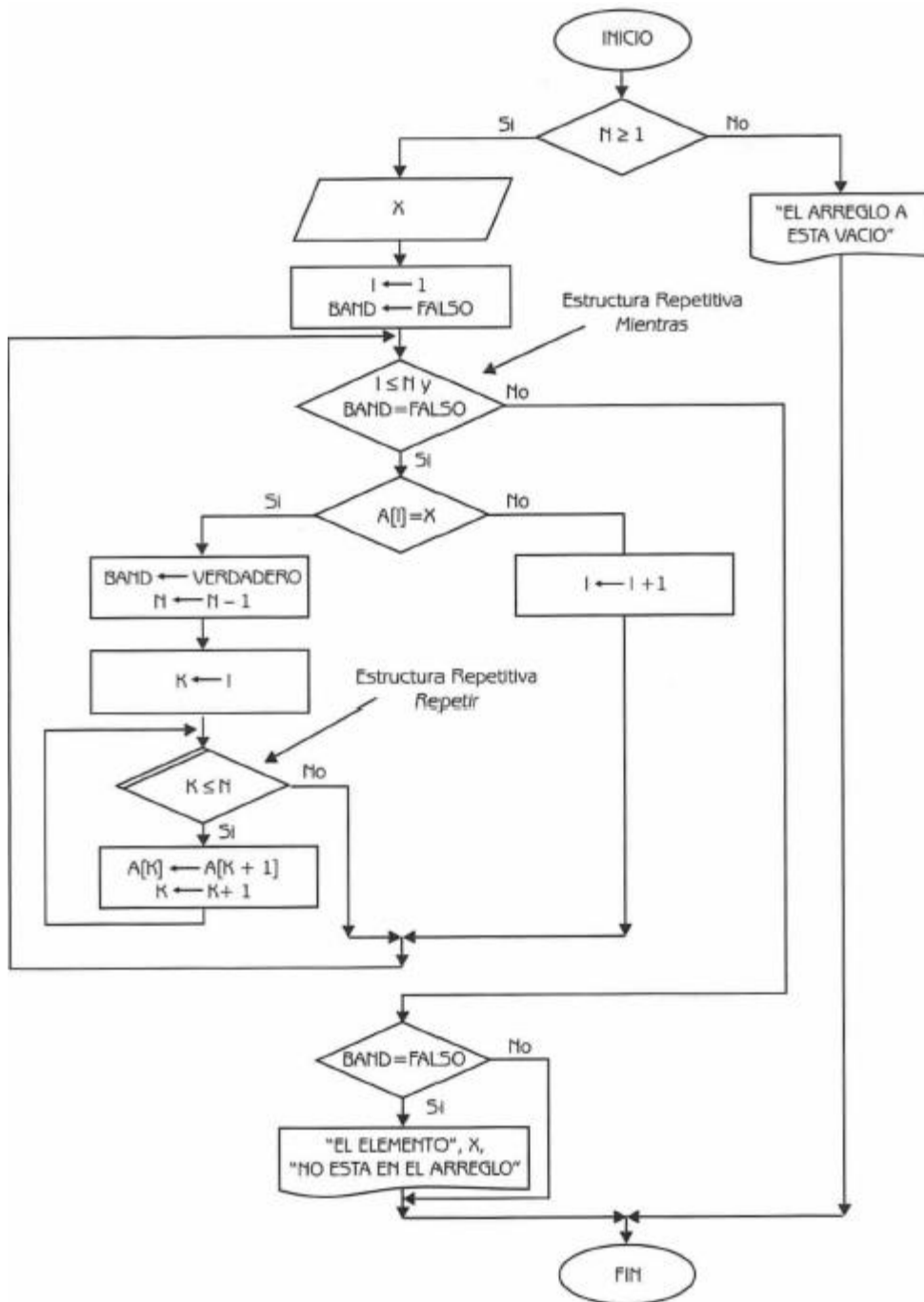
Hacer  $N \leftarrow N + 1$  y  $A[N] \leftarrow Y$

sino

Escribir "No hay espacio para insertar al elemento Y"

2. {Fin del condicional del paso 1}

**Eliminación:** Para eliminar un elemento X de un arreglo A desordenado debe verificarse que el arreglo no esté vacío y que X se encuentre en el arreglo. Si se cumplen estas condiciones entonces se procederá a recorrer todos los elementos que están a su derecha una posición a la izquierda, decrementando finalmente el número de componentes del arreglo. A continuación presentamos el diagrama de flujo correspondiente.



Explicación de las variables:

N: Variable de tipo entero. Almacena el número actual de elementos del arreglo.

X: Variable de tipo entero. Representa al valor que se va a eliminar.

I: Variable de tipo entero. Se utiliza como variable de control del ciclo externo y como índice del arreglo V.



**BAND:** Variable de tipo booleano. Se inicializa en falso. Cambia su valor a verdadero si se encuentra el elemento a eliminar en el arreglo, en cuyo caso se interrumpe el ciclo.

**K:** Variable de tipo entero. Se utiliza como variable de control del ciclo interno y como índice del arreglo A.

**A:** Arreglo unidimensional de tipo entero. Su capacidad máxima es de 100 elementos.

**Modificación:** Para modificar un elemento X por un elemento Y, de un arreglo A que se encuentra desordenado debe verificarse que el arreglo no esté vacío y que X se encuentre en el arreglo. Si

se cumplen estas condiciones entonces se procederá a su actualización. Puede observarse que existen tareas comunes con la operación de eliminación:

- Determinar que el arreglo no esté vacío.
- Encontrar el elemento a modificar (eliminar).

A continuación, presentamos el diagrama de flujo correspondiente.

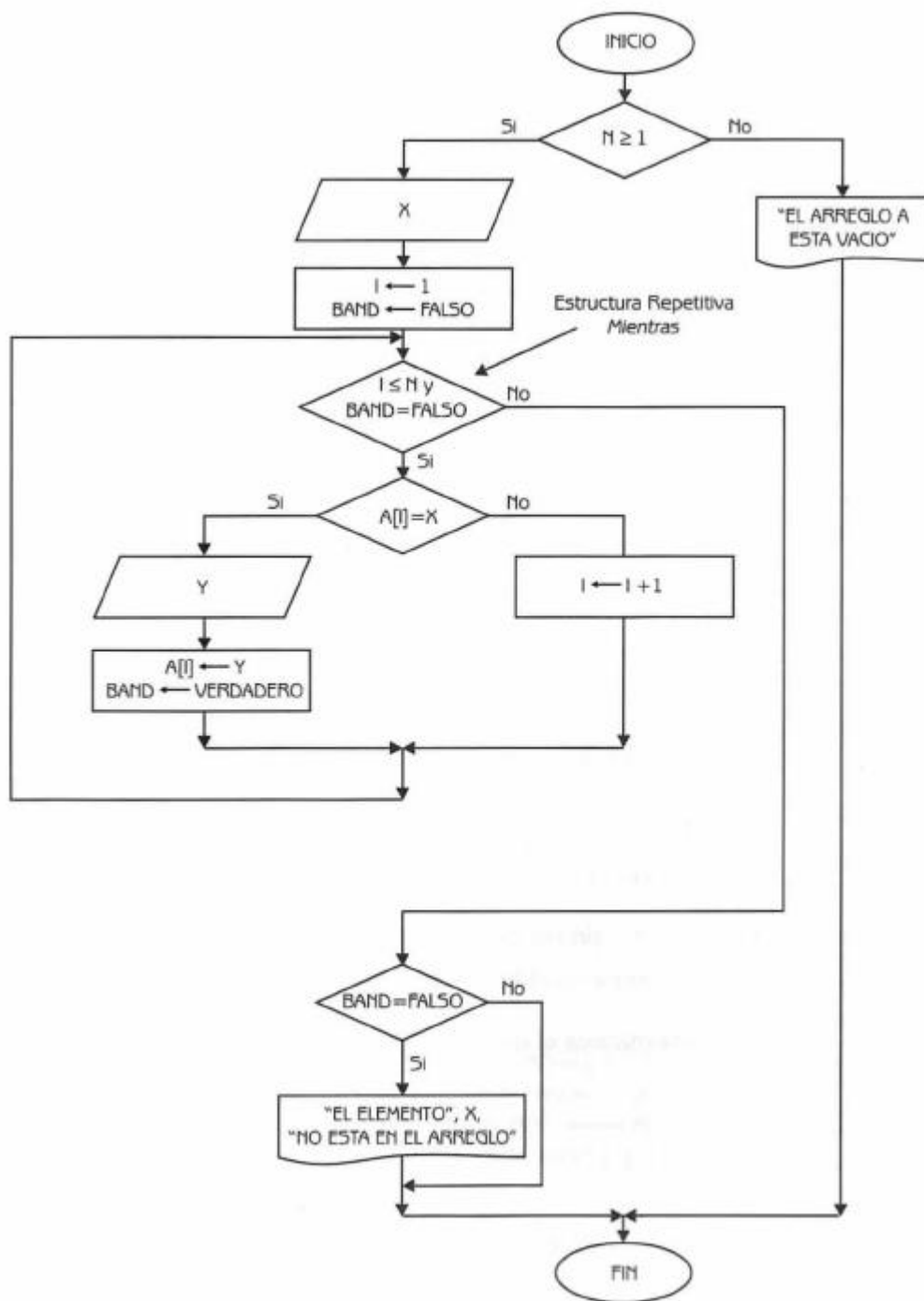


Diagrama de Flujo 4.5

Explicación de las variables

N: Variable de tipo entero. Almacena el número actual de elementos del arreglo

X: Variable de tipo entero. Representa el elemento que se va a modificar.

I: Variable de tipo entero. Se utiliza como variable de control del ciclo y como índice del arreglo.

BAND: Variable de tipo booleano. Se inicializa en FALSO. Cambia su valor a VERDADERO si se encuentra el elemento a modificar en el arreglo, en cuyo caso se interrumpe el ciclo.

A: Arreglo unidimensional de tipo entero. Su capacidad máxima es de 100 elementos.

Y: Variable de tipo entero. Representa al elemento que se introduce y modifica al elemento X.

## ARREGLOS ORDENADOS

Considere el arreglo ordenado A de 100 elementos de la figura 4.12. Los primeros N componentes del mismo tienen asignado un valor. En este caso se trabajará con un arreglo ordenado de manera creciente, es decir:

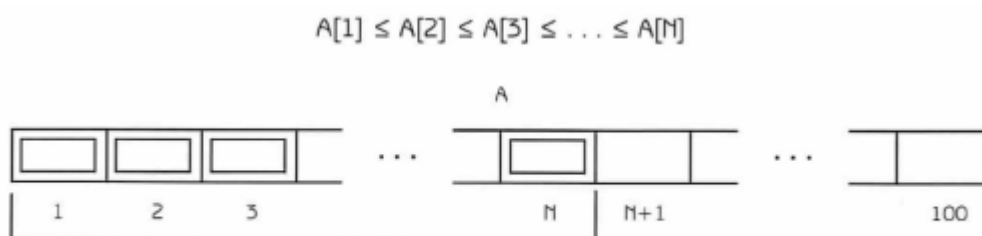


Figura 4.12 Actualización de arreglos ordenados.

Cuando se trabaja con arreglos ordenados no se debe alterar el orden al insertar nuevos elementos o al modificar los existentes.

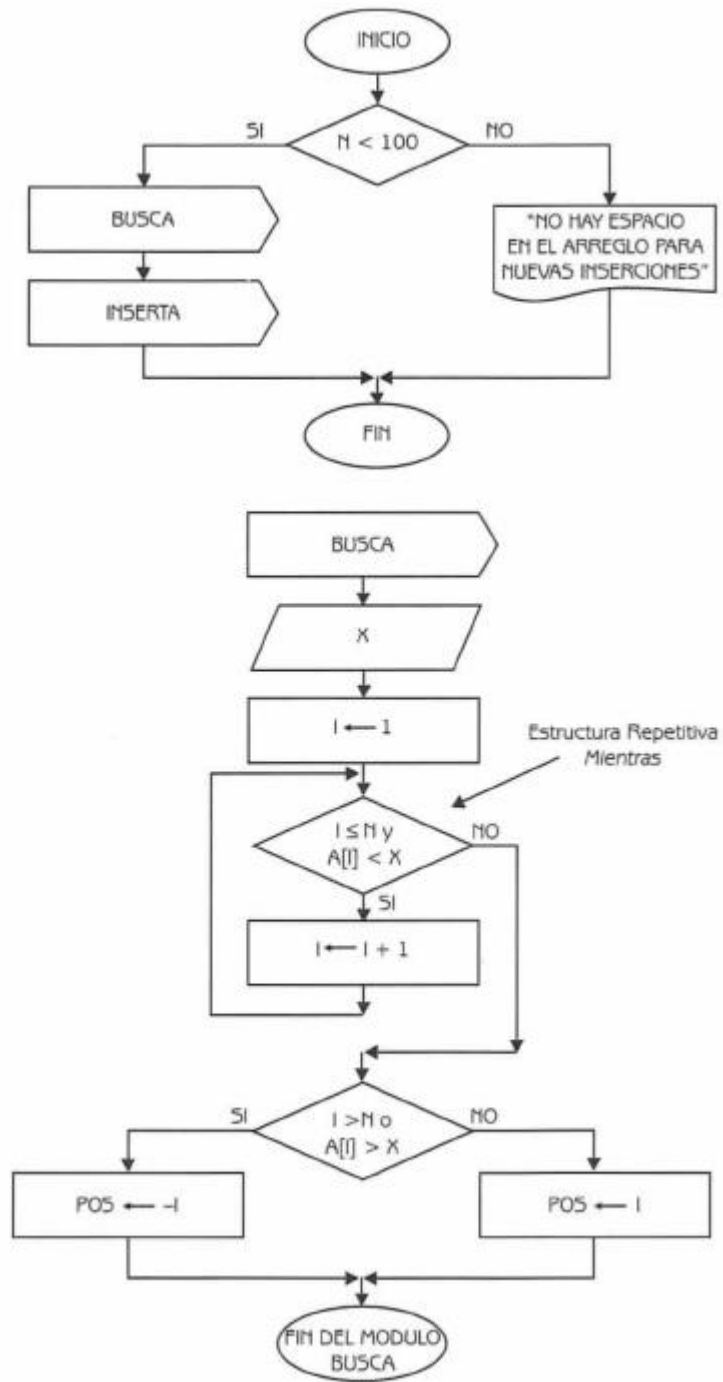
1) **Inserción:** Para insertar un elemento X en un arreglo A que se encuentra ordenado, debe verificarse que exista espacio. Luego tendrá que encontrarse la posición en la que debería estar el nuevo valor para no alterar el orden del arreglo. Una vez detectada la posición, se procederá a recorrer todos los elementos desde la misma hasta la N-ésima posición, un lugar a la derecha. Finalmente se asignará el valor de X en la posición encontrada (Los pasos del re corrimiento no se llevan a cabo cuando el valor a insertar es mayor que el último elemento del arreglo).

Generalmente cuando se quiere hacer una inserción debe verificarse que el elemento no se encuentre en el arreglo. En la mayoría de los casos prácticos no interesa tener información duplicada, por lo tanto si el valor a insertar ya estuviera en el arreglo, la operación no se llevaría a cabo.

Debemos señalar que tanto en los procesos de inserción y eliminación en arreglos ordenados es recomendable tener un procedimiento que busque el elemento X en el arreglo ordenado.

Este procedimiento dará como resultado la posición en la que se encuentre el elemento X (en cuyo caso el elemento ya pertenece al arreglo y por lo tanto no debemos insertarlo) o el negativo de la posición en la que debería estar.

A continuación, en el diagrama de flujo 4.6, presentamos la solución al problema.



I: Variable de tipo entero. Se utiliza como variable de control de los ciclos. En el módulo BUSCA, detecta además la posición donde se encuentra o debería estar el elemento X. Es utilizado también como índice del arreglo.

X: Variable de tipo entero. Representa al elemento que se va a insertar (si no se encuentra en el arreglo).

A: Arreglo unidimensional de tipo entero. Su capacidad máxima es de 100 elementos.

POS: Variable de tipo entero. Almacena la posición donde se encuentra o debería estar el elemento X.

SEMANA 11: DEL 10 AL 14 DE AGOSTO /2020

## 6.2 Arreglos Bidimensionales o Matriz

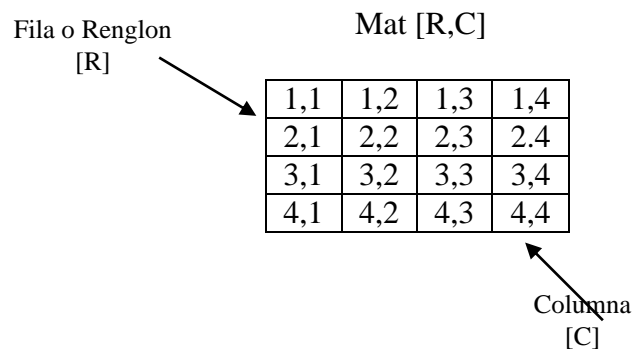
Es un arreglo de  $M * N$  elementos organizados en dos dimensiones donde “M” es el número de filas o renglones y “N” el número de columnas.

Para representar una matriz se necesita un nombre de matriz se necesita un nombre de matriz acompañado de dos índices.

Mat [R,C]

Donde R indica el renglón y C indica la columna, donde se encuentra almacenado el dato.

### Representación gráfica de una matriz



### Llenado de una matriz

- **Por renglones**  
Hacer para R = 1 a 5  
    Hacer para C = 1 a 5  
        Leer Mat [R,C]  
    Fin-para  
Fin-para
- **Por columnas**

```

Hacer para C = 1 a 5
    Hacer para R = 1 a 5
        Leer Mat [R,C]
    Fin-para
Fin-para

```

Nota: Para hacer el llenado de una matriz se deben de usar dos variables para los índices y se utilizan 2 ciclos uno para los renglones y otro para las columnas; a estos ciclos se les llama ciclos anidados (un ciclo dentro de otro ciclo).

Para comprender mejor la estructura de los arreglos bidimensionales, presentamos a continuación un ejemplo.

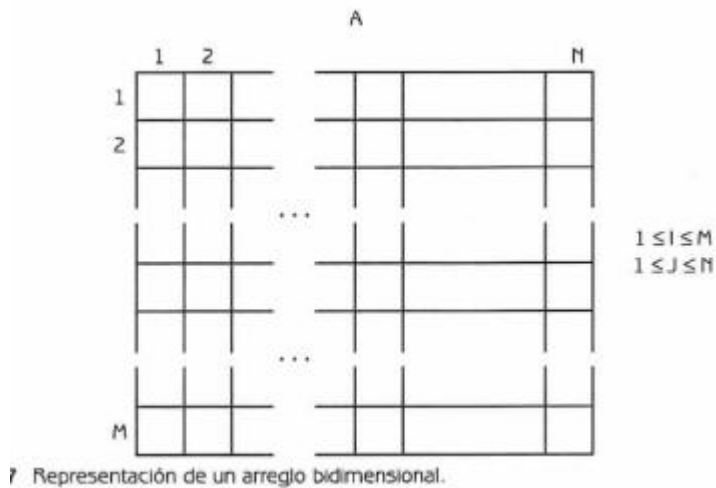
Ejemplo:

La tabla 4.2 contiene las lluvias caídas medidas en milímetros, correspondientes a los 12 meses del año anterior en cuatro estados de la República Mexicana

Tabla 4.2 Lluvias mensuales por estados				
Meses / Lluvias	Morelos	México	Querétaro	Puebla
Enero	50	45	60	58
Febrero	7	3	15	22
Marzo	12	10	8	17
Abril	15	5	20	35
Mayo	22	30	15	22
Junio	50	90	60	100
Julio	85	150	20	88
Agosto	70	75	88	94
Septiembre	65	49	53	105
Octubre	28	37	29	37
Noviembre	35	15	22	4
Diciembre	17	8	14	0

La tabla se interpreta de la siguiente manera: dado un mes, se conocen las lluvias caídas en cada uno de los estados; y dado un estado, se conocen las lluvias caídas en forma mensual. Si se quisiera almacenar esta información con los arreglos que se conocen, se tendrían dos alternativas:

1. Definir 12 arreglos de 4 elementos cada uno. Cada arreglo almacenará la información relativa a un mes.



**Declaración de arreglos bidimensionales.**-Se declararán los arreglos bidimensionales especificando el número de renglones y el número de columnas, junto con el tipo de los componentes

Tipo de dato NOMBRE DE LA MATRIZ [ tamaño en filas, tamaño en columnas]

Lo mismo que en el caso de los arreglos unidimensionales, los índices pueden ser cualquier tipo de dato ordinal (escalar, entero, caracter), mientras que los componentes pueden ser de cualquier tipo (reales, enteros, cadenas de caracteres, etc). Veamos a continuación algunos ejemplos de arreglos bidimensionales.

Sea MATRIZ un arreglo bidimensional de números reales con índices enteros. Su representación queda como se muestra en la figura 4.18.

	1	2	3	4	5
1					
2					
			...		
15					

Figura 4.18

MATRIZ = ARREGLO [1..15,1..5] DE reales

- $NTE = (15 - 1 + 1) * (5 - 1 + 1) = 15 * 5 = 75$
- Cada elemento de MATRIZ será un número real. Para hacer referencia a cada uno de ellos se usarán dos índices y el nombre de la variable tipo arreglo: MATRIZ [i,j].

Donde:  $1 \leq i \leq 15$   
 $1 \leq j \leq 5$

Así por ejemplo:

MATRIZ [2,4] hace referencia al elemento del renglón 2 y la columna 4.

MATRIZ [9,2] hace referencia al elemento del renglón 9 y la columna 2.

...

MATRIZ [15,5] hace referencia al elemento del renglón 15 y la columna 5.

### Operaciones con arreglos bidimensionales

A continuación, se presentan las operaciones que pueden realizarse con arreglos bidimensionales:

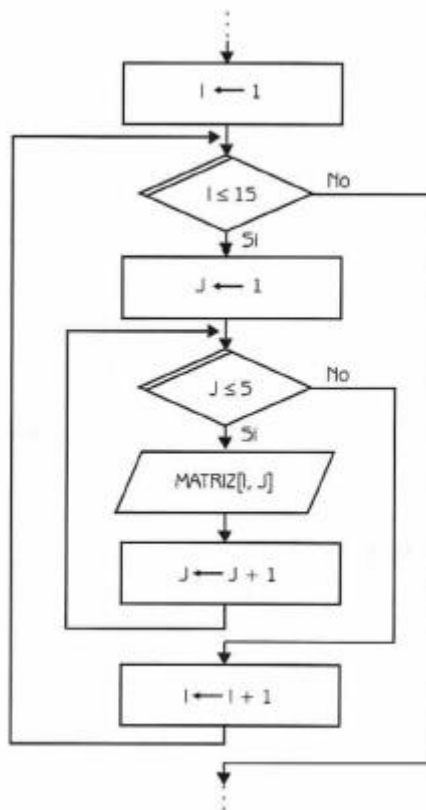
- Lectura / Escritura.
- Asignación.
- Actualización: Inserción.  
Eliminación.  
Modificación.
- Ordenación.
- Búsqueda.

### Lectura

Cuando se introdujo la lectura en arreglos unidimensionales se dijo que se iban asignando valores a cada uno de los componentes. Lo mismo sucede con los arreglos bidimensionales. Sin embargo, como sus elementos deben referenciarse con dos índices, se deben utilizar dos ciclos para lograr la lectura de elementos consecutivos.

Supóngase que se desea leer todos los elementos del arreglo bidimensional MATRIZ. Los pasos a seguir son los siguientes:





```

...
Hacer I ← 1
Repetir con I desde 1 hasta 15
  Hacer J ← 1
  Repetir con J desde 1 hasta 5
    Leer MATRIZ [I, J]
    Hacer J ← J + 1
  {Fin del ciclo interno}
  Hacer I ← I + 1
{Fin del ciclo externo}
...

```

Al variar los índices de I y J se lee un elemento de matriz, según la posición indicada por los índices I y J.

Para I = 1 y J = 1, se lee el elemento del renglón 1 y columna 1.

I = 1 y J = 2, se lee el elemento del renglón 1 y columna 2.

...

I = 15 y J = 5, se lee el elemento del renglón 15 y columna 5.

Los arreglos Bidimensionales utilizan las mismas operaciones de los arreglos unidimensionales con la diferencia que necesitan dos índices para ubicar un elemento de la matriz.

## MANEJO DE MÓDULOS

- 7.1 Definición
- 7.2 Función
- 7.3 Manipulación

### OBJETIVO EDUCACIONAL:

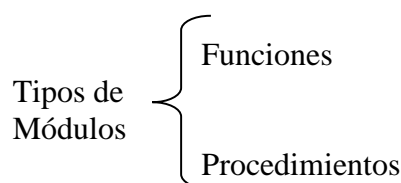
El alumno:

- Identificará y aplicará los datos de tipo cadena de caracteres (string) para la solución de problemas de tipo administrativo.

Un problema complejo se puede dividir en pequeños subproblemas más sencillos. Estos subproblemas se conocen como “*Módulos*” y su complementación en un lenguaje se llama subprograma (procedimientos y funciones).

Un subprograma realiza las mismas acciones que un programa, sin embargo, un subprograma lo utiliza solamente un programa para un propósito específico.

Un subprograma recibe datos de un programa y le devuelve resultados (el programa “llama” o “invoca” al subprograma, este ejecuta una tarea específica y devuelve el “control” al programa que lo llamo).



**Función:** Una función en matemáticas, es una operación que toma un o más valores (argumentos) y devuelve un resultado (valor de la función para los argumentos dados). Por ejemplo:

$$F(X) = X / (1+X^2)$$

Donde:

F ..... Nombre de la función

X ..... Es el argumento (también conocido como parámetro formal)

**Definición de funciones:** Una definición de función se presenta de la siguiente manera:

Función nombre\_funcion (p1, p2, ..., pn)

Inicio

Bloque de instrucciones

Fin

Donde:

Función ..... Es la palabra clave que nos indica una definición de función.

Nombre\_funcion ..... Es el identificador con el cual se reconoce a la función en el cuerpo del

algoritmo principal.

P1,p2,...,pn ..... Es el grupo de parámetros que define a la función.

### ***Llamado a una función***

Cuando definimos una función solo le indicamos al algoritmo que esta función existe, pero una definición de función no implica la realización de las instrucciones que la constituyen. Para hacer uso de una función, el algoritmo principal la debe llamar. Por ejemplo:

```
Función F(X)
  Inicio
    F = X / (1 + X^2)
  Fin
Inicio
  Imprimir "Este es el algoritmo principal"
  Leer N
  R = F(N) ← llamado de la función
  Imprimir "El resultado de la función es:", R
Fin
```

### **Definición de la Función**

La función ha sido declarada, ha sido llamada y por lo tanto deber haber sido definida. Lo cual consta de dos partes, las cuales son:

#### 1. La Primera Línea

Que como su nombre lo indica, es la primera línea de la definición de la función y con ella le indicamos al compilador que está en presencia de una función. Su formato es el siguiente:

Tipo\_de\_dato nombre\_de\_la\_función (tipo y nombre de los argumentos)

#### 2. Cuerpo de la función

Se inicia con una llave "{", y en ella, se pueden realizar asignaciones, cálculos, impresiones, así como la declaración de las variables locales. Puede estar constituida por estructuras secuenciales, selectivas, iterativas, anidamientos, se pueden llamar otras funciones, etc; finaliza con "}". Puede devolver uno o ningún valor.

En C, se conocen como funciones *aquellos trozos de códigos utilizados para dividir un programa con el objetivo que, cada bloque realice una tarea determinada*. En las funciones juegan un papel muy importante las variables, ya que como se ha dicho estas pueden ser locales o globales.

**Variables Globales:** Estas se crean durante toda la ejecución del programa, y son globales, ya que pueden ser llamadas, leídas, modificadas, etc; desde cualquier función. Se definen antes del main().

**Variables Locales:** Estas, pueden ser utilizadas únicamente en la función que hayan sido declaradas. La sintaxis de una función es la siguiente:

**Tipo\_de\_datos nombre\_de\_la\_funcion(tipo y nombre de argumentos)**

{

**acciones**

}

donde:

**Tipo\_de\_datos:** Es el tipo de dato que devolverá esa función, que puede ser real, entera, o tipo void(es decir que no devolverá ningún valor).

**Nombre\_de\_la\_funcion:** Es el identificador que le damos a nuestra función, la cual debe cumplir las reglas que definimos en un principio para los identificadores.

**Tipo y nombre de argumentos:** son los parámetros que recibe la función. Los argumentos de una función no son más que variables locales que reciben un valor. Este valor se lo enviamos al hacer la llamada a la función. Pueden existir funciones que no reciban argumentos.

**Acciones:** Constituye el conjunto de acciones, de sentencias que cumplirá la función, cuando sea ejecutada. Entre ellas están:

1. Asignaciones
2. Lecturas
3. Impresiones
4. Cálculos, etc

Una función, termina con la llave de cerrar, pero antes de esta llave, debemos colocarle la instrucción **return**, con la cual devolverá un valor específico. Es necesario recalcar que si la función no devuelve ningún valor, es decir, es tipo void, no tiene que ir la sentencia return, ya que de lo contrario, nos dará un error.

Un viejo adagio dice: Separa y vencerás, lo cual se acopla perfectamente cuando tenemos un programa que es bastante grande; podemos separarlos en pequeños subprogramas (funciones), y concentrarnos en la solución por separados de cada uno de ellos y así resolver un gran problema, en unos cuantos problemitas más pequeños.

Existen diferentes valores que pueden devolver las funciones

El orden será el siguiente:

1. Funciones que devuelven un valor entero
2. Funciones que devuelven un valor Real

### ¿Cómo es que funcionan los Subprogramas?

A menudo, utilizamos el adjetivo de “Subprogramas”, para referirnos a las funciones, así que, el lector debe familiarizarse también con este término. Los subprogramas se comunican con el programa principal, que es el que contiene a las funciones, mediante parámetros, que estos pueden ser: Parámetros Formales y Parámetros Actuales. Cuando se da la comunicación los parámetros actuales son utilizados en lugar de los parámetros formales

### 5.3. Paso de Parámetros

Existen dos formas de pasar parámetros, las cuales son:

- A) Paso por Valor También conocido como parámetros valor. Los valores se proporcionan en el orden de cálculos de entrada.

Los parámetros se tratan como variables locales y los valores iniciales se proporcionan copiando los valores de correspondientes argumentos.

Los parámetros formales-Locales de una función reciben como iniciales los valores de los parámetros actuales y con ellos se ejecutan las acciones descritas en el subprograma.

Ejemplo:

```
A=5; B=7;
```

```
C=proc1(A, 18, B*3+4);
```

```
Proc1(X, Y, Z)
```

Explicación: Donde, se encuentra c, se está llamando la función, denominada proc1, en la cual se están enviando como parámetros el valor de A, que es cinco; el cual es recibido por la variable X, en la definición de la función proc1; en la misma función, Y tendrá el valor de 18; porque ese es el valor del parámetro formal, mientras que Z, tendrá un valor inicial de 25, ya que ese es el resultado del tercer parámetro que resulta ser una expresión aritmética.

El estándar ANSI de C introdujo una nueva (mejor) forma de hacer lo anterior respecto a las versiones previas de C.

La importancia de usar prototipos de funciones es la siguiente:

Se hace el código más estructurado y por lo tanto, más fácil de leer.

Se permite al compilador de C revisar la sintaxis de las funciones llamadas.

Lo anterior es hecho, dependiendo del alcance de la función. Básicamente si una función ha sido definida antes de que sea usada (o llamada), entonces se puede usar la función sin problemas.

Si no es así, entonces la función se debe declarar. La declaración simplemente maneja el tipo de dato que la función regresa y el tipo de parámetros usados por la función.

Es una práctica usual y conveniente escribir el prototipo de todas las funciones al principio del programa, sin embargo, esto no es estrictamente necesario.

Para declarar un prototipo de una función se indicará el tipo de dato que regresará la función, el nombre de la función y entre paréntesis la lista del tipo de los parámetros de acuerdo al orden que aparecen en la definición de la función. Por ejemplo:

int longcad(int n); Lo anterior declara una función llamada longcad que regresa un valor entero y acepta otro valor entero como parámetro.

### Ejemplo 5.3

Diseñe un programa, que dado un número entero y mayor que cero, muestre su factorial. (El factorial de 5 es 120;  $5 \times 4 \times 3 \times 2 \times 1 = 120$ )

```
#include <stdio.h>
#include <conio.h>
int factorial (int num);
main()
{
    int num, ban=1;
    clrscr();
    while(ban==1)
    {
        printf("Ingrese el valor del número por favor:\n");

        scanf("%d",& num);
        while(num<0)
        {
            printf("ERROR, el valor del número debe ser mayor que cero:\n");
            scanf("%d",& num);
        }

        printf("El valor del factorial es %d\n", factorial (num));

        printf("Desea Realizar otro calculo?Si=1 y No=0\n");
        scanf("%d",& ban);
    }
    getch();
    return 0;
}
int factorial (int num)
{
```

```

int sum=1, i;
for(i=2; i<=num; i++)
{
sum=sum*i;
}
return (sum);
}

```

Explicación:

Quizá, lo único nuevo, e importante de explicar, radica en la llamada y la definición de la función. Cuando una función nos devolverá un valor entero, al identificador de dicha función debe

precederle el tipo de dato. En el lugar, donde llamamos la función, es que aparecerá el valor que nos devuelva, como valor de retorno. En nuestro ejemplo, en una impresión. Y al momento de definirla, no se nos debe olvidar, colocarle la sentencia `return()`; ya que, mediante esta declaratoria, está retornando el valor calculado.

Pero, qué sucede cuando se está trabajando, con valores bastante grandes, al utilizar solamente el `int`, se producirá un error lógico; ya que como valor de retorno podría ser un cero o una cifra negativa. Por tanto, debemos usar el tipo de dato “`long int`”.

### **Funciones que devuelven un valor entero**

Las funciones que devuelven algún valor, se les llama **PROTOTIPOS DE FUNCIONES**: Antes de usar una función C debe tener conocimiento acerca del tipo de dato que regresará y el tipo de los parámetros que la función espera.

El estándar ANSI de C introdujo una nueva (mejor) forma de hacer lo anterior respecto a las versiones previas de C. La importancia de usar prototipos de funciones es la siguiente: Se hace el código más estructurado y por lo tanto, más fácil de leer. Se permite al compilador de C revisar la sintaxis de las funciones llamadas. Lo anterior es hecho, dependiendo del alcance de la función. Básicamente si una función ha sido definida antes de que sea usada (o llamada), entonces se puede usar la función sin problemas. Si no es así, entonces la función se debe declarar. La declaración simplemente maneja el tipo de dato que la función regresa y el tipo de parámetros usados por la función. Es una práctica usual y conveniente escribir el prototipo de todas las funciones al principio del programa, sin embargo, esto no es estrictamente necesario. Para declarar un prototipo de una función se indicará el tipo de dato que regresará la función, el nombre de la función y entre paréntesis la lista del tipo de los parámetros de acuerdo al orden que aparecen en la definición de la función. Por ejemplo: `int longcad(int n);` Lo anterior declara una función llamada `longcad` que regresa un valor entero y acepta otro valor entero como parámetro.



Diseñe un programa, que dado un número entero y mayor que cero, muestre su factorial. (El factorial de 5 es 120;  $5 \times 4 \times 3 \times 2 \times 1 = 120$ )

```
#include <stdio.h>
#include <conio.h>
int factorial (int num);
main()
{
int num, ban=1;
clrscr();
while(ban==1)
{
printf("Ingrese el valor del número por favor:\n");
scanf("%d",& num);
while(num<0)
{ printf("ERROR, el valor del número debe ser mayor que cero:\n");
scanf("%d",& num);
}
printf("El valor del factorial es %d\n\n", factorial (num));
printf("Desea Realizar otro calculo?Si=1 y No=0\n");
scanf("%d",& ban);
} getch();
return 0; }
int factorial (int num)
{ int sum=1, i;
for(i=2; i<=num; i++)
{ sum=sum*i;
}
return (sum);
}
```

### **Explicación:**

Quizá, lo único nuevo, e importante de explicar, radica en la llamada y la definición de la función. Cuando una función nos devolverá un valor entero, al identificador de dicha función debe precederle el tipo de dato. En el lugar, donde llamamos la función, es que aparecerá el valor que nos devuelva, como valor de retorno. En nuestro ejemplo, en una impresión. Y al momento de definirla, no se nos debe olvidar, colocarle la sentencia *return()*; ya que, mediante esta declaratoria, está retornando el valor calculado. Pero, qué sucede cuando se está trabajando, con valores bastante grandes, al utilizar solamente el *int*, se producirá un error lógico; ya que como valor de retorno podría ser un cero o una cifra negativa. Por tanto debemos usar el tipo de dato “long int”.

### **Funciones que Devuelven un Valor Real**

Antes que nada, trataremos las funciones predefinidas en C. Ya que C, posee ciertas funciones que nos ayudan hacer nuestros programas más fáciles y utilizar menos código. El lenguaje c, cuenta con una serie de funciones de bibliotecas que realizan operaciones y cálculos de uso frecuente. Para acceder a una función, se

realiza mediante el nombre seguido de los argumentos que le servirán a la función a realizar la tarea específica. **Nombre(arg1, arg2,...argn); Nombre(arg1, arg2,...argn);**

**\*Funciones Matemáticas** Para acceder a ellas, se debe colocar la directiva `#include <math.h>` en el encabezado del programa.

Función (Sintaxis)	Tipo de Dato	Propósito
<code>acos(d)</code>	double	Devuelve el arco coseno de d
<code>asin(d)</code>	double	Devuelve el arco seno de d
<code>atan(d)</code>	double	Devuelve el arco tangente de d
<code>atan(d1, d2)</code>	double	Devuelve el arco tangente de d1/d2
<code>ceil(d)</code>	double	Devuelve el valor redondeado por exceso, al

		siguiente entero mayor
<code>cos(d)</code>	double	Devuelve el coseno de d
<code>cosh(d)</code>	double	Devuelve coseno hiperbólico de d
<code>exp(d)</code>	double	Eleva a la potencia d
<code>fabs(d)</code>	double	Devuelve el valor absoluto de d
<code>floor(d)</code>	double	Devuelve el valor redondeado por defecto al entero menor más cercano
<code>log(d)</code>	double	Devuelve el logaritmo natural de d
<code>log10(d)</code>	double	Devuelve el lo. (base10) de d
<code>pow(d1, d2)</code>	double	Devuelve d1 elevado a la potencia d2
<code>sin(d)</code>	Double	Devuelve el seno de d
<code>sinh(d)</code>	double	Seno hiperbólico de d
<code>sqrt(d)</code>	double	Raíz cuadrada de d
<code>Tan(d)</code>	double	Devuelve la tangente de d
<code>tanh(d)</code>	double	Devuelve la tangente hiperbólica de d

Hay muchas otras funciones, pero para ahondar más, debes saber cuál es la versión de C, instalada en tu máquina y así verificar cuáles funcionan correctamente; pero por lo general, estas funciones son muy estándar para la mayoría de compiladores.

## SEMANA 13: 24 AL 28 DE AGOSTO/2020

### PROCEDIMIENTOS

La definición de procedimientos permite asociar un nombre a un bloque de instrucciones. Luego podemos usar ese nombre para indicar en algún punto de un algoritmo que vamos a utilizar ese bloque de instrucciones, pero sin tener la necesidad de repetirlas, sólo invocando al procedimiento por su nombre.

Los procedimientos pueden ser clasificados en acciones o funciones. Las acciones se caracterizan por no retornar valores al algoritmo que las llama, mientras que las funciones retornan un valor. Sin embargo, aunque las acciones no retornan valores, si pueden informar al algoritmo que las llamó de cambios realizados por sus instrucciones en algunos valores a través de una herramienta que se llama pase de parámetros. Los parámetros permiten utilizar la misma secuencia de instrucciones con diferentes datos de entrada.

Utilizar parámetros es opcional. Cuando entre las instrucciones de un algoritmo vemos el nombre de un procedimiento (acción o función), decimos que estamos llamando o invocando al procedimiento. Los procedimientos facilitan la programación modular, es decir, tener bloques de instrucciones que escribimos una vez pero que podemos llamar y utilizar muchas veces en varios algoritmos. Una vez terminada la ejecución de un procedimiento (acción o función), se retorna el control al punto de algoritmo donde se hizo la llamada, para continuar sus instrucciones.

## ACCIONES

Conjunto de instrucciones con un nombre que pueden ser llamadas a ejecución cuando sea necesario. No retornan valores.

Sintaxis:

```
void <nombre>(<lista de parametros>)  
{  
    <bloque de instrucciones>  
}
```

## Funciones que no devuelven ningún valor.

Cómo se ha dicho las funciones pueden o no devolver algún valor, para mi parecer, este tipo de funciones son las más sencillas, ya que cuando se llama la función, esta realiza lecturas, asignaciones, cálculos o impresiones, finaliza la ejecución de la función y el programa continúa normalmente.

Ejemplo: Diseñe un programa que dados dos números enteros determine la suma y cuál de ellos es mayor, usando dos funciones diferentes.

```
#include <stdio.h>  
#include <conio.h>  
void suma (int a, int b); /*Declaración de la función*/  
void mayor (int a, int b); /*Tipo de dato, nombre de la función y el tipo y nombre de los argumentos*/
```

```

main()
{ int a, b;
printf("Ingrese el valor de a:\n");
scanf("%d", &a);
printf("Ingrese el valor de b:\n");
scanf("%d", &b);
suma(a,b); /*Llamado de la función*/
mayor(a,b); /*Unicamente el nombre de la función y de los par metros*/
getch(); return 0;
}

void suma(int a, int b) /*Definición de la función*/
{ /*Abrimos llaves al inicio de la definición*/
int sum; /*Declaración de las variables locales*/
sum=a+b; printf("El valor de la suma es %d:\n\n", sum);
} /*Fin de la función suma*/

void mayor(int a, int b)
{ if(a==b)
printf("Son iguales\n\n");
else {
if(a>b)
printf("El valor de a es mayor que el de b\n\n");
else
printf("El valor de b es mayor que el de a\n\n");
} }

```

## Ejemplo 2

Sumatoria de 2 valores enteros utilizando procedimientos

```

#include<stdio.h>
Void suma(int a,int b) //declaración del procedimiento
int main()
{
int num1, num2;
printf(" Suma de 2 números enteros ");
printf("Ingrese el Primer valor ");
scanf("%d", num1);
printf("Ingrese el Segundo valor ");
scanf("%d", num2);
suma(num1, num2); // llamada a la función
clrscr();

```

```

Void suma(int a,int b) // Cuerpo de la función
{
    Int res; res=a+b;
    printf(" el resultado es: "+ res);
}

```

## **SEMANA 14: 31 AGOSTO AL 4 DE SEPTIEMBRE/2020**

### **REGISTROS o ESTRUCTURAS**

Las estructuras son colecciones de variables relacionadas bajo un nombre.

Las estructuras pueden contener variables de muchos tipos diferentes de datos

- a diferencia de los arreglos que contienen únicamente elementos de un mismo tipo de datos.

#### Definición de estructuras

Las estructuras son tipos de datos derivados - están construidas utilizando objetos de otros tipos. Considere la siguiente definición de estructura:

```

struct ejemplo f
char c;
int i,g;

```

La palabra reservada struct indica se está definiendo una estructura. El identificador ejemplo es el nombre de la estructura. Las variables declaradas dentro de las llaves de la definición de estructura son los miembros de la estructura. Los miembros de la misma estructura deben tener nombres únicos mientras que dos estructuras diferentes pueden tener miembros con el mismo nombre. Cada definición de estructura debe terminar con un punto y coma.

La definicion de struct ejemplo contiene un miembro de tipo char y otro de tipo int. Los miembros de una estructura pueden ser variables de los tipos de datos básicos (int, char, float, etc) o agregados como ser arreglos y otras estructuras. Una estructura no puede contener una instancia de sí misma.

Declaramos variables del tipo estructura del siguiente modo:

```

Struct ejemplo f1, a[10];

```

Una operación válida entre estructuras es asignar variables de estructura a variables de estructura del mismo tipo. Las estructuras no pueden compararse entre sí.

## Ejemplo

Consideremos la información de una fecha. Una fecha consiste de: el día, el mes, el año y posiblemente el día en el año y el nombre del mes. Declaramos toda esa información en una estructura del siguiente modo:

```
struct fecha {  
    int dia;  
    int mes;  
    int anio;  
    int dia del anio;  
    char nombre mes[9];  
};
```

Un ejemplo de estructura que incluye otras estructuras es la siguiente estructura persona que incluye la estructura fecha:

```
struct persona {  
    char nombre[tamano nombre];  
    char direccion[tamano dir];  
    long codigo postal;  
    long seguridad social;  
    double salario;  
    fecha cumpleaños;  
    fecha contrato;  
};
```

## Como inicializar estructuras

Las estructuras pueden ser inicializadas mediante listas de inicialización como con los arreglos. Para inicializar una estructura escriba en la declaración de la variable a continuación del nombre de la variable un signo igual con los inicializadores entre llaves y separados por coma por ejemplo:

ejemplo `e1 = { 'a', 10 g`;

Si en la lista aparecen menos inicializadores que en la estructura los miembros restantes son automáticamente inicializados a 0. Las variables de estructura también pueden ser inicializadas en enunciados de asignación asignándoles una variable del mismo tipo o asignándole valores a los miembros individuales de la estructura.

Podemos inicializar una variable del tipo fecha como sigue:

```
struct fecha f = {4, 7, 1776, 186, "Julio"};
```

## Como tener acceso a los miembros de estructuras

Para tener acceso a miembros de estructuras utilizamos el operador punto. El operador punto se utiliza colocando el nombre de la variable de tipo estructura seguido de un punto y seguido del nombre del miembro de la estructura. Por ejemplo, para imprimir el miembro `c` de tipo `char` de la estructura `e1` utilizamos el enunciado:

```
printf (" %c", e1.c);
```

Para acceder al miembro `i` de la estructura `e1` escribimos: `e1.i`

En general, un miembro de una estructura particular es referenciada por una construcción de la forma:

nombre de estructura.miembro

por ejemplo para chequear el nombre de mes podemos utilizar:

```
if (strcmp(d.nombre mes,\Agosto")==0) .....
```

Como utilizar estructuras con funciones

Las estructuras pueden ser pasadas a funciones pasando miembros de estructura individuales o pasando toda la estructura.

Cuando se pasan estructuras o miembros individuales de estructura a una función se pasan por llamada por valor. Para pasar una estructura en llamada por referencia tenemos que colocar el '\*' o '&'.

Los arreglos de estructura como todos los demás arreglos son automáticamente pasados en llamadas por referencia.

Si quisiéramos pasar un arreglo en llamada por valor, podemos definir una estructura con único miembro el array.

Una función puede devolver una estructura como valor.

Ejemplo

Consideraremos el ejemplo de un punto dado por dos coordenadas enteras.

```
struct punto {
int x;
int y; };
/* creo punto: crea un punto a partir de sus coordenadas */
punto creo punto(int a, int b)
{
punto temp;
temp.x=a;
temp.y=b;
return temp;
}

/* sumo puntos: suma dos puntos */
punto sumo puntos(punto p1,punto p2)
{
p1.x += p2.x;
p1.y += p2.y;
return p1;
}
/* imprimo punto: imprime las coordenadas de un punto */
void imprimo punto(punto p)
{
printf(\Coordenadas del punto:%d y%d \n", p.x, p.y);
}
```

## Typedef

La palabra reservada typedef proporciona un mecanismo para la creación de sinónimos (o alias) para tipos de datos anteriormente definidos. Por ejemplo:

```
typedef struct ejemplo Ejemplo;  
define Ejemplo como un sinónimo de ejemplo.
```

Una forma alternativa de definir una estructura es:

```
typedef struct {  
    char c;  
    int i;} Ejemplo;
```

Podemos ahora utilizar Ejemplo para declarar variables del tipo struct, por ejemplo

```
Ejemplo a[10];
```

typedef se utiliza a menudo para crear seudónimos para los tipos de datos básicos. Si tenemos por ejemplo un programa que requiere enteros de 4 bytes podría usar el tipo int en un programa y el tipo long en otro. Para garantizar portabilidad podemos utilizar typedef para crear un alias de los enteros de 4 bytes en ambos sistemas.

## **SEMANA 15: 7 AL 11 DE SEPTIEMBRE/2020**

### **ESTRUCTURAS ANIDADAS**

Una estructura puede estar dentro de otra estructura a esto se le conoce como anidamiento o estructuras anidadas. Ya que se trabajan con datos en estructuras si definimos un tipo de dato en una estructura y necesitamos definir ese dato dentro de otra estructura solamente se llama el dato de la estructura anterior.

Definamos una estructura en nuestro programa:

```
struct empleado /* creamos una estructura llamado empleado*/  
{  
    char nombre_empleado[25];  
    char direccion[25];  
    char ciudad[20];  
    char provincia[20];  
    long int codigo_postal;  
    double salario;  
}; /* las estructuras necesitan punto y coma (;) al final */
```

Y luego necesitamos una nueva estructura en nuestro programa:

```
struct cliente /* creamos una estructura llamada cliente */  
{  
    char nombre_cliente[25];  
    char direccion[25];  
    char ciudad[20];  
    char provincia[20];  
    long int codigo_postal;  
    double saldo;
```



```
}; /* las estructuras necesitan punto y coma (;) al final */
```

Podemos ver que tenemos datos muy similares en nuestras estructuras, así que podemos crear una sola estructura llamada **infopersona** con estos datos idénticos:

```
struct infopersona /* creamos la estructura que contiene datos parecidos */
{
    char direccion[25];
    char ciudad[20];
    char provincia[20];
    long int codigo_postal;
}; /* las estructuras necesitan punto y coma (;) al final */
```

Y crear las nuevas estructuras anteriores, anidando la estructura necesaria:

```
/*
Ejemplo de estructuras anidadas
*/

#include <stdio.h>
#include <string.h>

/* creamos nuestra estructura con datos similares */
struct infopersona
{
    char direccion[25];
    char ciudad[20];
    char provincia[20];
    long int codigo_postal;
}; /* las estructuras necesitan punto y coma (;) al final */

/* creamos nuestra estructura empleado */
struct empleado
{
    char nombre_empleado[25];
    /* agregamos la estructura infopersona
    * con nombre direcc_empleado
    */
    struct infopersona direcc_empleado;
    double salario;
}; /* las estructuras necesitan punto y coma (;) al final */

/* creamos nuestra estructura cliente */
struct cliente
{
    char nombre_cliente[25];
    /* agregamos la estructura infopersona
    * con nombre direcc_cliente
    */
    struct infopersona direcc_cliente;
    double saldo;
```

```

}; /* las estructuras necesitan punto y coma (;) al final */

int main(void)
{
    /* creamos un nuevo cliente */
    struct cliente MiCliente;

    /*inicializamos un par de datos de Micliente */
    strcpy(MiCliente.nombre_cliente,"Jose Antonio");
    strcpy(MiCliente.direcc_cliente.direccion, "Altos del Cielo");
    /* notese que se agrega direcc_cliente haciendo referencia
    * a la estructura infopersona por el dato direccion
    */

    /* imprimimos los datos */
    printf("\n Cliente: ");
    printf("\n Nombre: %s", MiCliente.nombre_cliente);
    /* notese la forma de hacer referencia al dato */
    printf("\n Direccion: %s", MiCliente.direcc_cliente.direccion);

    /* creamos un nuevo empleado */
    struct empleado MiEmpleado;

    /*inicializamos un par de datos de MiEmplado */
    strcpy(MiEmpleado.nombre_empleado,"Miguel Angel");
    strcpy(MiEmpleado.direcc_empleado.ciudad,"Madrid");
    /* para hacer referencia a ciudad de la estructura infopersona
    * utilizamos direcc_empleado que es una estructura anidada
    */

    /* imprimimos los datos */
    printf("\n");
    printf("\n Empleado: ");
    printf("\n Nombre: %s", MiEmpleado.nombre_empleado);
    /* notese la forma de hacer referencia al dato */
    printf("\n Ciudad: %s", MiEmpleado.direcc_empleado.ciudad);

    return 0;
}

```

## **SEMANA 16: 14 AL 18 DE SEPTIEMBRE/2020**

Repaso Final

Evaluación del Segundo Parcial

Entrega final de trabajos