# Programación en C

## *Curso introductorio*

En este curso se verá las nociones básicas del lenguaje de programación en C. Abarcará los temas principales del lenguaje: estructura, librerías, principales funciones, trabajos integrativos entre las distintas funciones, entre otras cosas más. Cada unidad cuenta con explicación teórica sobre el tema acompañado de ejercicios prácticos para un mejor desempeño intelectual.

Arduino Edu

## Índice

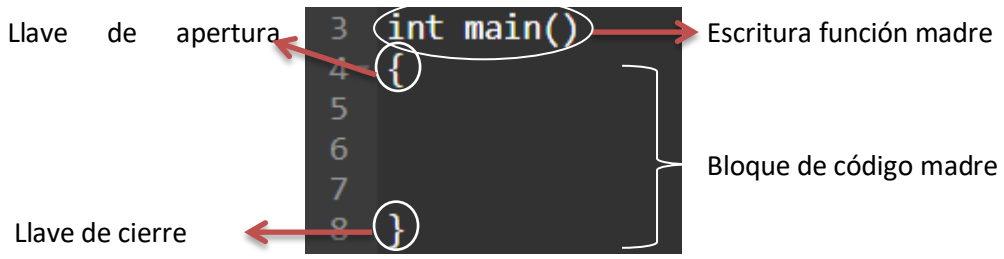
# ➢ Unidad 1: ❖*Estructura del lenguaje*

Antes de ver como programar y conocer las funciones básicas, es necesario entender cómo es que funciona y como está conformada la estructura principal del lenguaje.

1)- En primer lugar tenemos la librería, esta misma alberga las funciones básicas del lenguaje; es decir que la librería es quien contiene todas las funciones que utilizaremos a lo largo de este curso. Existen múltiples librerías y cada una de ellas contiene su pack de funciones específicas para realizar distintos tipos de tareas; para añadir una librería se hace de la siguiente manera: #include <nombrelibreria>
La librería que utilizaremos esta predeterminada en el compilador online (lo que no nos hará falta añadirla) y es la siguiente:

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

2)- En siguiente instancia, tenemos el cuerpo general de nuestro programa, se lo denomina "Función madre", aquí es donde se programará las líneas de nuestro código.
El nombre que recibe esta función es: "main".
Su estructura, partes y escritura son las siguientes:




- ❖ Llaves de apertura y cierre: Es lo que nos delimita en donde empieza y termina nuestro bloque de código; es fundamental tener en cuenta que tienen que estar presentes si o si, ya que en caso contrario nos saldrá error.
  Nota: si se abre una llave, se tiene que cerrar, no pueden haber llaves abiertas si no son cerradas.
- ❖ Bloque de código madre: Acá es donde se escriben las líneas de código del programa, todas las funciones, parámetros, operaciones, etc, tienen que estar escritas dentro del bloque de código.
  Nota: hay funciones que tienen bloques de código propios, es necesario saber diferenciar entre el bloque de código específico de cada función y el bloque de código madre.
- Tener en cuenta: Es indispensable que esto se respete para el correcto funcionamiento del programa, ya que, sin esta estructura inicial, el código no se ejecutara de forma satisfactoria.
- Otra cosa a tener en cuenta, es que no se pueden utilizar tildes en las funciones.

## ❖Tipos de variable

Para realizar cualquier tipo de código, es necesario el uso de variables.
Las variables son un espacio en memoria que almacena datos como números o letras, que después son utilizados para diferentes propósitos.
Existen distintos tipos de variables, veremos las más importantes: int, float, char.
◆La variable tipo "int" sirve para guardar números enteros, tiene un límite de hasta 10 dígitos.
◆La variable tipo "float" se usa para guardar números con decimales. Para separar enteros de decimales, se utiliza "."(Punto). Por ejemplo 10.53.
◆La variable tipo "char" es utilizada para guardar un único carácter de a-z.

1 ) - Forma de declarar una variable "declaración": primero tendremos que declarar un tipo de dato y un nombre. Para realizar dicha acción hay que seguir los siguientes pasos:
a ) - En primer lugar, hay que indicarle a nuestro compilador el tipo de dato que va a ser nuestra variable, es decir, si es tipo int, float o char. Quedando de la siguiente manera:

```
1  #include <stdio.h>
2
3  int main()
4  {
5
6      int
7
8  }
```

b ) - Luego hay que nombrar a nuestra variable, podemos elegir un nombre cualquiera que cumpla con estas características: dígitos a-z / números 0-9 / _ (guión bajo).
Quedando de la siguiente manera:

```
1  #include <stdio.h>
2
3  int main()
4  {
5
6      int a;
7
8  }
```

❖**IMPORTANTE:**
Al finalizar una instrucción para nuestro compilador (en este caso la instrucción de crear una variable), **se debe terminar el renglón con un ";"** (punto y coma), ya que de caso contrario, nos saldrá un error de compilación.

En este caso la variable de ejemplo se llama "a" y es tipo int, lo que quiere decir que almacenará un número entero.

2)- Asignación: Una vez declarada nuestra variable, vamos a asignarle un valor en concreto.
Para realizar esto, haremos lo siguiente:
a)- Debajo de nuestra variable creada, pondremos el nombre que hemos elegido para la misma, seguido de un signo igual (=), y a continuación pondremos un valor.

```
#include <stdio.h>

int main()
{

    int a;
    a=10;

}
```

De esta forma declararemos y asignaremos todas nuestras variables.

## ❖*Mostrar en pantalla*

Una vez que nosotros hayamos realizado todo nuestro código, hay que mostrar, de alguna manera, el resultado de lo que programamos. Para ello utilizaremos la función "**printf**".
La estructura de esta función es la siguiente: **printf(" ")**
En donde la parte que usaremos para mostrar el resultado es dentro de las 2 comillas que están dentro de los paréntesis, quedando de la siguiente forma:

```
#include <stdio.h>

int main()
{

    printf("Hola Mundo");

}
```

Como vemos la parte en rojo es el nombre de la función y la verde es texto que se mostrará exactamente de la misma manera en la cual está escrita.

Ahora, si queremos mostrar el valor de alguna variable que hemos creado, tendremos que usar lo que se denomina como "máscaras".

Una máscara es simplemente una función auxiliar del lenguaje que lo que hace es adoptar el valor de una variable.

Existen diferentes tipos de máscaras dependiendo del tipo de dato al cual nos estemos refiriendo:

Tipo de dato: int → %d
float → %f
char → %c

Para usar esta función lo que tenemos que hacer es fijarnos en el tipo de dato de la variable a la cual vamos a mostrar su valor, buscar su máscara correspondiente (tabla previamente mencionada) y ponerla dentro de las comillas como se muestra en la imagen.

```c
#include <stdio.h>

int main()
{
    int a;
    a=10;

    printf("el valor de a es: %d");
}
```

En este caso utilizamos una variable tipo int (entero) por lo cual su máscara correspondiente es %d.
Como vemos, la máscara se nos puso de otro color.

Luego lo que debemos de hacer es, justo después de las segundas comillas y antes del segundo paréntesis, agregar una "," (coma) y escribir el nombre de la variable que queremos enseñar.

```c
#include <stdio.h>

int main()
{
    int a;
    a=10;

   printf("el valor de a es: %d",a);
}
```

Si mandamos a ejecutar estas líneas de código, lo que nos mostrara es el texto que escribimos dentro del printf y luego el valor de nuestra variable.

```
el valor de a es: 10

...Program finished with exit code 0
Press ENTER to exit console.
```

Como vemos, nos salió lo que programamos.
El texto en color verde al final, nos indica que nuestro código se ha ejecutado correctamente.

## ❖Operaciones básicas

Una vez que ya sabemos cómo declarar variables y como mostrar su valor, lo que haremos ahora, son las 5 operaciones básicas (suma(+), resta(-), multiplicación(*), división(/) y módulo(%)).
Los pasos a seguir son:

1)- Declaramos 3 variables distintas.

```
#include <stdio.h>

int main()
{
    int a;
    int b;
    int c;
}
```

2)- Asignamos a 2 de ellas valores enteros distintos.

```
#include <stdio.h>

int main()
{
    int a;
    int b;
    int c;
    a=10;
    b=5;
}
```

3)- Utilizaremos la variable “c” para realizar la operación y guardar el resultado de la misma, por lo que igualaremos nuestra tercera variable (c) a la suma de las variables con valores previamente asignados.

```
#include <stdio.h>

int main()
{
    int a;
    int b;
    int c;
    a=10;
    b=5;
    c=a+b;
}
```

Como vemos, la operación tiene que ir después de ya haber asignado los valores a operar, ya que si no, no se ejecutara correctamente nuestro programa.

4)- Escribir la función “printf” para enseñar por pantalla el resultado obtenido de la operación realizada y ejecutar programa.

```
#include <stdio.h>

int main()
{
    int a;
    int b;
    int c;
    a=10;
    b=5;
    c=a+b;

    printf("el resultado de sumar a + b es: %d",c);
}
```

**❖Nota❖**

Como vemos en la imagen, la variable que nosotros debemos mostrar es la “c”, ya que es la variable que tiene almacenado el resultado de sumar las variables “a” y “b”.

## ➢Variable “char”

Este tipo de variable se utiliza para guardar un único carácter del alfabeto (a-z).
Esto generalmente se utiliza para elegir opciones (opción a, b, c, etc.).
Para generar una variable tipo char haremos lo siguiente:

1)- Escribimos el tipo de dato seguido del nombre de nuestra variable.

```
#include <stdio.h>

int main()
{
    char a;
}
```

2)- Para asignarle una letra alfabética, lo que haremos es lo siguiente, colocaremos el nombre que le hemos asignado a nuestra variable, seguido del signo “=” (igual) y entre comillas simples pondremos la letra correspondiente.

```
#include <stdio.h>

int main()
{
    char a;
    a='b';
}
```

En este caso nuestra variable con nombre “a”, tiene almacenada la letra “b”.

3)- Para mostrar esta variable utilizaremos la máscara correspondiente al tipo de dato char y lo pondremos en nuestra función printf de la misma manera que en los ejemplos anteriores.

```
#include <stdio.h>

int main()
{
    char a;
    a='b';
    printf("la variable a contiene la letra: %c",a);
}
```

4)- Mandamos a ejecutar nuestro programa.

```
la variable a contiene la letra: b

...Program finished with exit code 0
Press ENTER to exit console.
```

## ❖*Ejercicios:*

1)- Realizar una multiplicación entre 3 números y mostrar resultado.
2)- Realizar una resta entre 3 números decimales.
3)- Crear 3 números distintos, en donde los primeros 2 números sean sumados, y al resultado de dicha suma, hacer una resta con el tercer número.
4)- Indicar que la opción elegida es la opción “c”.

## ❖*Interacción con usuario*

Es importante saber que algo fundamental de la programación es que tenga interacción con el usuario, es decir, que el usuario por ejemplo sea quien ingrese los datos para realizar una operación simple como una suma o resta.
Esta cualidad es posible gracias a la función que veremos a continuación.

El nombre que recibe esta función es “scanf” y nos permitirá dicha interacción con el usuario. La estructura de esta función es: scanf(“ ”,& ) en donde dentro de las comillas va la máscara del tipo de dato deseado y al lado del símbolo ampersand, el nombre de la variable a la cual le estemos asignando el valor.

Básicamente esta función lo que hace es almacenar el dato que el usuario ingresa por teclado en la variable previamente creada.

Cabe destacar que el tipo de dato de la variable creada tiene que ser el mismo que el tipo de dato que se esté pidiendo con dicha función, es decir, que si tenemos creada una variable tipo int, nuestra función scanf tiene que almacenar un dato tipo int.

Haremos un ejercicio con esta función.
Los pasos a seguir para utilizar esta función son los siguientes:
Haremos una suma.

1 )- Declararemos 3 variables tipo float.

```c
#include <stdio.h>

int main()
{
    float a,b,c;
}
```

**❖Nota❖**

Se pueden declarar 2 o más variables del mismo tipo en el mismo renglón, siempre y cuando se separen entre ellas con una "," (coma), al hacerlo ahorraremos líneas de código por lo que será más corto y más practico a la hora de descubrir posibles errores.

2 )- Escribiremos un mensaje por pantalla avisándole al usuario que tiene que ingresar 2 valores decimales.

```c
#include <stdio.h>

int main()
{
    float a,b,c;

    printf("ingrese 2 valores decimales:");
}
```

3 )- Ahora pondremos la función scanf para almacenar esos datos que el usuario va a ingresar. Tener en cuenta que como estamos trabajando con variables tipo float, la máscara que utilizaremos es %f.

```c
#include <stdio.h>

int main()
{
    float a,b,c;

    printf("ingrese 2 valores decimales:");
    scanf("%f",&a);
    scanf("%f",&b);
}
```

Como vemos, hay que utilizar una función scanf para cada una de las variables que le queremos asignar el valor, en este caso, hay que asignarle valor a 2 variables por lo que tenemos que utilizar 2 funciones de scanf distintas.

4)- Una vez ingresados los datos hay que realizar la operación, por lo que utilizaremos la tercera variable que hemos creado para ello.

```
#include <stdio.h>

int main()
{
    float a,b,c;

    printf("ingrese 2 valores decimales:");
    scanf("%f",&a);
    scanf("%f",&b);
    c=a+b;
}
```

5)- Por ultimo lo que haremos es mostrar el resultado obtenido.

```
#include <stdio.h>

int main()
{
    float a,b,c;

    printf("ingrese 2 valores decimales:");
    scanf("%f",&a);
    scanf("%f",&b);
    c=a+b;
    printf("el resultado obtenido es: %f",c);
}
```

6)- Ejecutamos el programa.

```
ingrese 2 valores decimales:4.2
3.4
el resultado obtenido es: 7.600000

...Program finished with exit code 0
Press ENTER to exit console.
```

Como vemos funciona correctamente el código, pero nos muestra 6 decimales que en su mayoría son 0.

Para modificar la cantidad de decimales que nos muestra a la hora de ejecutar el programa, nos tenemos que dirigir hacia la función printf que nos muestra el resultado y en la máscara tenemos que hacer lo siguiente, después del "%" (porcentaje) y antes de la "f", justo en el medio, pondremos ".2" para que nos muestre solo 2 decimales.

Quedando de la siguiente manera:

```c
#include <stdio.h>

int main()
{
    float a,b,c;

    printf("ingrese 2 valores decimales:");
    scanf("%f",&a);
    scanf("%f",&b);
    c=a+b;
    printf("el resultado obtenido es: %.2f",c);
}
```

Por lo que si ahora lo ejecutamos se vería de la siguiente forma.

```
ingrese 2 valores decimales:4.2
3.4
el resultado obtenido es: 7.60

...Program finished with exit code 0
Press ENTER to exit console.
```

Como observamos, con el ".2" que ponemos en la máscara, le estamos indicando a nuestro compilador que queremos ver solamente 2 decimales en nuestro resultado final.

Podemos decidir cuantos decimales queremos que nos muestre usando el mismo procedimiento, es decir, que si queremos observar 1 único decimal, lo que pondremos es ".1" en el mismo lugar, si queremos ver 4 decimales, pondremos ".4" en ese lugar.

Básicamente la cantidad de decimales que vemos en el resultado final es el mismo que el número que ponemos después del "." (punto).

## ❖*Ejercicios con scanf:*

1)- Realizar una división decimal y que el resultado se muestre con 3 decimales.

2)- Ingresar 3 números decimales, calcular y mostrar el promedio de dichos números.

3)- Dada la fórmula de volumen de un cilindro: $\pi * r^2 * h$. En donde "r" implica radio y "h" altura, pedir al usuario que ingrese radio y altura y calcular su volumen. Expresar resultado con 2 decimales.

# ➢ Unidad 2: ❖*Prototipos*

Hay que tener en cuenta que nuestra función main (función madre) tiene que quedar con la menor cantidad de líneas de código posible.
Esto es así para llevar un orden y poder segmentar cada conjunto de funciones por la tarea que cumplen.
Es por esto que existe lo que se denomina como “prototipos”.
Un prototipo es una función que nos facilitará el propósito de alivianar la cantidad de líneas de código en nuestro main.
Básicamente lo que hace es ejecutar líneas de código fuera de nuestro bloque de código madre.

Vamos a dar un ejemplo con lo que ya hemos visto, las operaciones básicas.

Los pasos a seguir para crear un prototipo con todas sus partes son los siguientes:

1)- Ponemos el tipo de dato que vayamos a utilizar, seguido del nombre del prototipo y ponemos apertura y cierre de paréntesis. Esto se debe de realizar arriba de nuestro main.

```
1  #include <stdio.h>
2  int suma();
3  int main()
4  {
5
6
7  }
```

En este caso nuestro prototipo se llama “suma” ya que lo utilizaremos para sumar variables enteras.

2)- Dentro de los paréntesis pondremos la cantidad de variables que vamos a usar anteponiendo el tipo de dato.

```
1  #include <stdio.h>
2  int suma(int a, int b);
3  int main()
4  {
5
6
7  }
```

Se declaró que utilizaremos 2 variables enteras.

3)- Ahora hay que desarrollar las líneas de código de este prototipo para su funcionamiento. Debajo de nuestro bloque de código madre, pondremos el nombre de nuestro prototipo seguido de las variables creadas y su correspondiente bloque de código (recordar que un bloque de código está formado por una llave de apertura y una de cierre).

```
1  #include <stdio.h>
2  int suma(int a, int b);
3  int main()
4  {
5
6
7  }
8  int suma(int a, int b)
9  {
10
11 }
```

Como podemos observar, cuando creamos el prototipo en la parte de arriba, vemos como tenemos que terminar la oración con el ";" (punto y coma) ya que estamos ejerciendo una instrucción.
Luego observamos que en la parte de abajo de la imagen, que no lleva, ya que si llevara, el bloque de código que se encuentra abajo dejará de pertenecerle al prototipo por lo que tendremos problema para el correcto funcionamiento del código.

Hasta ahora sabemos que nuestro prototipo "suma" va a recibir 2 parámetros (variable a y b).

4)- Creamos la tercera variable dentro de nuestro prototipo para almacenar nuestro resultado y realizamos la operación correspondiente.

```
1  #include <stdio.h>
2  int suma(int a, int b);
3  int main()
4  {
5
6
7  }
8  int suma(int a, int b)
9  {
10     int c;
11     c=a+b;
12 }
```

5)- Una vez realizada la operación, hay que retornar el valor resultante a nuestro main, es decir, "c" es nuestra variable que almacena el resultado, por lo tanto, tendremos que retornar el valor de la variable c. Para ello usaremos la función "return" seguido del nombre de la variable a retornar.

```
#include <stdio.h>
int suma(int a, int b);
int main()
{


}
int suma(int a, int b)
{
    int c;
    c=a+b;
    return c;
}
```

Una vez hayamos hecho esto, ya tenemos completo nuestro prototipo con todas sus partes.
Si bien el prototipo está completo, el código aún no, porque como vemos, nuestro main, sigue vacío.

6)- Dentro del bloque de código main, crearemos 2 variables enteras que usaremos para realizar la operación.

```
#include <stdio.h>
int suma(int a, int b);
int main()
{
   int a,b;

}
int suma(int a, int b)
{
    int c;
    c=a+b;
    return c;
}
```

7)- Le daremos un valor a estas variables por teclado (scanf).

```
#include <stdio.h>
int suma(int a, int b);
int main()
{
   int a,b;
   printf("ingrese 2 valores:");
   scanf("%d",&a);
   scanf("%d",&b);
}
int suma(int a, int b)
{
    int c;
    c=a+b;
    return c;
}
```

8)- Mostramos el resultado por pantalla con su máscara correspondiente.

En la parte de nombrar la variable que queremos mostrar, llamaremos al prototipo que hemos creado de la siguiente manera, pondremos el nombre del mismo, y dentro de los paréntesis tenemos que pasarle los parámetros (nombre de las variables) con los que tiene que operar separados con una "," (coma).

```
1  #include <stdio.h>
2  int suma(int a, int b);
3  int main()
4  {
5     int a,b;
6     printf("ingrese 2 valores:");
7     scanf("%d",&a);
8     scanf("%d",&b);
9     printf("el resultado es: %d",suma(a,b));
10 }
11 int suma(int a, int b)
12 {
13     int c;
14     c=a+b;
15     return c;
16 }
```

Llamamos a nuestro prototipo

Con esto lo que estamos haciendo es indicarle a nuestro compilador que tiene que hacer uso de nuestro prototipo para realizar la operación correspondiente.
Como nuestra función lo que hace es retornar el valor de "c", ese es el valor que adoptará nuestra máscara.

9)- Ejecutamos.

```
ingrese 2 valores:4
8
el resultado es: 12

...Program finished with exit code 0
Press ENTER to exit console.
```

Nuestro código se ejecutó correctamente, el usuario ingreso los valores "8" y "4", el programa llamó a nuestro prototipo, se realizó la operación y nos devolvió el resultado.

## ❖*Ejercicios con prototipos:*

1)- Crear un prototipo que realice una resta.
2)-Crear un prototipo que calcule el volumen de un cilindro dada la fórmula: $\pi * r^2 * h$. En donde "r" implica radio y "h" altura.

# ➢ Unidad 3: ❖*Estructuras de control*

Una de las funciones más importantes de los lenguajes de programación, son la toma de decisiones, ya que con estas nos permiten tanto al usuario como al programador, elegir entre múltiples opciones y ejecutar las líneas de código correspondientes a la situación.
La función que nos permitirá llevar a cabo esta acción se denomina: "**if**".
Esta función permitirá al programador crear diferentes opciones o preferencias para que el usuario sea quien elija una de ellas según su objetivo.
Esta función no viene sola, está acompañada de otras sub-funciones que complementan y ayudan para completar la función.
Ellas son: "**else**" y "**else if**".

En primer lugar tenemos la función "if", cuya estructura de escritura es "if() { }" en donde dentro de los paréntesis pondremos la condición a cumplir para que sea verídico y las llaves implican el bloque de código de la función.
Ahora bien, los tipos de condiciones que disponemos son las siguientes:

- a<b símbolo "<" (menor): a menor que b
- a>b símbolo ">" (mayor): a mayor que b
- a<=b símbolo "<=" (menor igual): a menor o igual que b
- a>=b símbolo ">=" (mayor igual): a mayor o igual que b
- a==b símbolo "==" (comparativo): a igual que b
- a!=b símbolo "!=" (distinto): a distinto que b

(Siendo a y b números o variables electas).

Ahora bien, también tenemos lo que se denomina "operadores lógicos" cuya función es complementar los tipos de condiciones.
Estos operadores son "**AND (&&)**" y "**OR (||)**".
Dichos operadores se usan para unir 2 tipos de condiciones a un mismo criterio, es decir, se pueden combinar las condiciones en base a las necesidades.

AND: Este operador se utiliza para combinar 2 condiciones. Para que la condición sea verdadera, se tienen que **cumplir las 2 condiciones usadas obligatoriamente**. Por ejemplo: if (a<10 && a>3) { }. En este ejemplo podemos ver que hay 2 condiciones unidas por el operador. Para que esta condición sea verdadera, el valor que debe de tener "a" tiene que ser menor a 10 **y** mayor a 3.

OR: Este operador se utiliza para combinar 2 condiciones. Para que la condición sea verdadera, se tienen que **cumplir una de las 2 condiciones usadas**. Por ejemplo:
If (a>=10 || a==4) { }. En este ejemplo, para que la condición sea verdadera, "a" tiene que ser mayor o igual a 10 **o** igual a 4.

Estas condiciones tienen que ir dentro de los paréntesis de nuestra función if.
Si la condición se cumple, es decir, es verdadera, se ejecutará el bloque de código correspondiente, en caso contrario, no se ejecutara dicho bloque.

Dentro de las sub-funciones (se denominan así porque si no hay un if previamente, estas funciones no funcionan), tenemos el "else", básicamente esta función lo que hace es ejecutarse cada vez que nuestra condición del if anterior es falsa, por lo que, si la condición del if anterior es verdadera, el "else" no se ejecutará.
Veamos esta sub-función en un ejemplo.

```c
#include <stdio.h>

int main()
{
    int a;
    printf("ingrese su edad:");
    scanf("%d",&a);
    if(a>=18)
    {
        printf("usted es mayor de edad");
    }
    else
    {
        printf("usted es menor de edad");
    }
}
```

Como vemos, este es un ejemplo simple aplicando la función "if" con el complemento "else". Este código le pide al usuario que ingrese su edad, en base a la edad del usuario, el programa toma una decisión y nos devuelve si el usuario es mayor o menor de edad. **Si la condición del if es verdadero (a>=18) se ejecutará el bloque de código del "if" y no se ejecutara el "else".** Caso contrario, si la condición no se cumple, no se ejecutará el "if" y pasa a ejecutarse el "else".

Ahora bien, nos queda la segunda sub-función que es "else if". Esta sub-función lo que hace es agregarle una condición a la sub-función "else", al agregarle una condición, va a implicar que no siempre se ejecute el "else" como en el caso anterior. Veamos un ejemplo.

```c
#include <stdio.h>

int main()
{
    int a;
    printf("ingrese su edad:");
    scanf("%d",&a);
    if(a>=18)
    {
        printf("su voto es obligatorio");
    }
    else if(a==16 || a==17)
    {
        printf("su voto es opcional");
    }
   else
    {
        printf("usted no puede votar");
    }
}
```

Este programa contiene todo lo que se vio en esta unidad, vamos a explicarlo por partes.

En primer lugar lo que realiza el código es decirnos a nosotros en base a nuestra edad, si debemos de votar o no.

Como vemos el primer "if" sirve para los que son mayores de edad, ya que la condición que tiene es mayores o iguales a 18, si esa condición se cumple, quiere decir que la persona es mayor de edad y que debe de votar.

En segundo lugar, tenemos el "else if", que tiene la condición para las personas que tienen 16 o 17 años, ya que su voto es opcional. Podemos observar que se implementó el operador lógico OR, quiere decir, que el número ingresado puede ser 16 o 17.

Por ultimo si ninguna de estas condiciones se cumplen se ejecutará el "else", que en este caso significa que la persona tiene de 15 años para abajo.

## ❖*Ejercicios con if:*

1)- Pedir al usuario ingresar un número, determinar si el numero ingresado es mayor o menor a 50.

2)- Pedir al usuario ingresar su edad y determinar si es mayor o menor de edad.

3)- Calcular el promedio de 3 números ingresados por teclado y determinar si está por debajo o por encima de la media, teniendo en cuenta que la media es de 30.

## ❖Casos comparativos

Existen otras formas de tomar decisiones, una de esas opciones son los casos comparativos, que si bien la forma de hacerlo es distinta, también tienen otro objetivo.
La función que usaremos para este tema se denomina “switch”. La estructura de escritura de esta función es: “switch() { }” en donde dentro del paréntesis va la variable a la cual queremos comparar con los casos comparativos y dentro del bloque de código van los casos comparativos propiamente dichos.
Esta función trabaja comparando el valor de una variable (entero o caracter) a distintos casos comparativos que nosotros declararemos dentro de la función.
Para llevar a cabo esta función, lo haremos con un ejemplo para su mejor comprensión. Supongamos que estamos realizando un menú interactivo con 4 opciones, las cuales son a, b, c o d, y tenemos que identificar cual es la opción que el usuario ingresó para luego realizar las operaciones correspondientes.
Para ello haremos lo siguiente:

1)- Crearemos una variable de tipo caracter.

```
1  #include <stdio.h>
2
3  int main()
4  {
5      char a;
6  }
```

2)- Pediremos al usuario ingresar algunas de las opciones disponibles y guardaremos la opción elegida.

```
1  #include <stdio.h>
2
3  int main()
4  {
5      char a;
6      printf("ingrese opción: a)- b)- c)- d)- :");
7      scanf("%c",&a);
8  }
```

3)- Ahora usaremos la función switch, usando la variable creada que contiene el carácter que necesitamos.

```c
#include <stdio.h>

int main()
{
    char a;
    printf("ingrese opción: a)- b)- c)- d)- :");
    scanf("%c",&a);
    switch(a)
    {

    }
}
```

4)- El siguiente paso, es crear los casos comparativos, podemos tener tantos casos comparativos como necesitemos. En este caso crearemos 4 casos ya que hay 4 opciones posibles. Para crear un caso debemos de poner la palabra “case” seguido del valor a comparar, en este ejercicio estamos comparando letras que son las opciones posibles asique entre comillas simples pondremos la primera opción que es la letra “a” y cerramos el renglón con “:” (dos puntos).

```c
#include <stdio.h>

int main()
{
    char a;
    printf("ingrese opción: a)- b)- c)- d)- :");
    scanf("%c",&a);
    switch(a)
    {
        case 'a':
    }
}
```

Como vemos, estamos comparando la letra “a” con el valor de nuestra variable “a”, que son cosas distintas.

5)- En el renglón de abajo pondremos las líneas de código que queremos que se ejecute cuando esta condición sea verdadera, en este caso queremos que nos muestre solamente que eligió esa opción, por lo que usaremos un printf con un texto simple.

```c
#include <stdio.h>

int main()
{
    char a;
    printf("ingrese opción: a)- b)- c)- d)- :");
    scanf("%c",&a);
    switch(a)
    {
        case 'a':
        printf("usted eligió la opcion a.");
    }
}
```

6)- Para cerrar el primer caso comparativo, en el renglón de abajo, usaremos la función “break”. Esta función nos sirve a nosotros para indicarle a nuestro compilador que justo en ese punto queremos que deje de ejecutarse la función switch ya que encontró satisfactoriamente el resultado que buscamos.

```c
#include <stdio.h>

int main()
{
    char a;
    printf("ingrese opción: a)- b)- c)- d)- :");
    scanf("%c",&a);
    switch(a)
    {
        case 'a':
        printf("usted eligió la opcion a.");
        break;
    }
}
```

7)- Repetiremos este proceso para el resto de las opciones disponibles.

```c
#include <stdio.h>

int main()
{
    char a;
    printf("ingrese opción: a)- b)- c)- d)- :");
    scanf("%c",&a);
    switch(a)
    {
        case 'a':
        printf("usted eligió la opcion a.");
        break;
        case 'b':
        printf("usted eligió la opcion b.");
        break;
        case 'c':
        printf("usted eligió la opcion c.");
        break;
        case 'd':
        printf("usted eligió la opcion d.");
        break;
    }
}
```

8)- Por ultimo pondremos un mensaje de error al final de los casos comparativos. Usaremos la palabra “default” con los correspondientes “:” (dos puntos). Luego en el renglón de abajo pondremos el mensaje de error y terminaremos cerrando con el “break”.

```c
#include <stdio.h>

int main()
{
    char a;
    printf("ingrese opción: a)- b)- c)- d)- :");
    scanf("%c",&a);
    switch(a)
    {
        case 'a':
        printf("usted eligió la opcion a.");
        break;
        case 'b':
        printf("usted eligió la opcion b.");
        break;
        case 'c':
        printf("usted eligió la opcion c.");
        break;
        case 'd':
        printf("usted eligió la opcion d.");
        break;
        default:
        printf("error");
        break;
    }
}
```

Este mensaje de error nos saldrá únicamente cuando nuestra variable “a” que es la que estamos comparando, tenga un valor distinto a los casos comparativos anteriores, es decir, que cuando ningún caso comparativo sea igual a la variable que estamos comparando, ahí se ejecutara el default con nuestro mensaje de error.

## ❖*Ejercicios con switch:*

1)- Descubrir que opción eligió el usuario. Las opciones son de 1 a 5.

2)- Realizar una operación según la opción que elija el usuario: opción 1: suma, opción 2: resta, opción 3: multiplicación y opción 4: división. Es importante tener en cuenta que la función switch trabaja con números enteros (int) o caracteres (char).

# ➢ Unidad 4: ❖*Sentencias repetitivas*

Como ya es hábito, sabemos que mientras más compacto y funcional sea nuestro código, mejor y más práctico es. Es por eso que se crearon las secuencias repetitivas para solucionar problemas de repeticiones de líneas de código, es decir, que si queremos que una línea de código se ejecute más de una vez, usaremos estas sentencias (sentencias, secuencias o ciclos son todas palabras sinónimas).
Disponemos de 2 funciones que cumplen con estas características.

## ❖*Ciclo flor*

En primer lugar, y más conocido, tenemos al ciclo for. Este ciclo nos permite a los programadores, desarrollar líneas de código con carácter repetitivo.
Cuenta con su propia estructura y reglas que harán funcionar a este programa.
La estructura de esta función es la siguiente:

```
for( ; ; )
{

}
```

Como vemos, dentro de los paréntesis se divide en 3 partes que ahora las completaremos.
La primera parte de este ciclo se denomina “inicialización”. Esto quiere decir inicializar la función, se hace de la siguiente manera. Tenemos que utilizar una variable como índice, esta variable auxiliar es indispensable para esta función y nos ayudara a llevar un conteo de cuantas veces queremos que se repita este ciclo repetitivo, por ende, en la primera sección de nuestra función, crearemos una variable llamada “i” de tipo entero que funcionará como nuestro índice y la igualamos a 0.

```
for(int i=0; ; )
{

}
```

La segunda parte de este ciclo, se denomina “criterio o condición”.
En esta sección pondremos, al igual que la función “if”, la condición que queremos que se cumpla. En otras palabras, el ciclo se va a repetir hasta que la condición de esta sección sea falsa, es decir, que acá es donde elegiremos la cantidad de veces que queremos que se ejecute este ciclo. Para este ejemplo, lo repetiremos 10 veces.

```
for(int i=0;i<10; )
{

}
```

Por último, en la tercera sección, tenemos la “incrementación”.
En esta sección utilizaremos una operación matemática propia del lenguaje que es agregarle “++” del lado derecho de una variable numérica para aumentar en una unidad su valor. Por ejemplo si tenemos a=0 y luego en otro renglón ponemos a++, esto quiere decir que a que valía 0 en un comienzo, ahora tiene el valor de 1.
Usamos esto para la tercera sección del for.

```
for(int i=0;i<10;i++)
{

}
```

Con esto lo que estamos haciendo es incrementar nuestro indice de 1 en 1 **por cada vez que se repite el ciclo,** es decir, nuestro indice (i) empieza con el valor 0, la condición es verdadera porque 0 si es menor que 10 entonces se ejecuta el bloque de código, una vez se terminó de ejecutar, aumenta nuestro índice en una unidad, y ahora en vez de valer 0, vale 1. Y asi sucesivamente hasta que el valor que tenga “i” sea 10 o más porque es cuando nuestra condición pasaría a ser falsa y el ciclo se terminaría.

Veamos con un ejemplo, vamos a mostrar por pantalla el valor de nuestro índice para corroborar que nuestro ciclo se este ejecutando con éxito.

```
1  #include <stdio.h>
2
3  int main()
4  {
5      for(int i=0;i<10;i++)
6      {
7          printf("%d",i);
8      }
9  }
```

```
0123456789

...Program finished with exit code 0
Press ENTER to exit console.
```

Como podemos observar nuestro índice “i” empieza con el valor 0 y fue incrementandose hasta llegar al valor 10, porque una vez que “i” vale 10, la condición del ciclo es falsa y se da por finalizado.

## ❖Ciclo while

Este ciclo es igual al for en términos de funcionalidad pero son distintos estructuralmente. Esto quiere decir, que como son ciclos iguales funcionalmente podremos utilizar el que más cómodos nos parezca.
La estructura de este ciclo es la siguiente “while() { }”. Las mismas 3 secciones que el ciclo anterior pero ubicados en otros lugares distintos del código.
En primer lugar tenemos la inicialización de nuestra variable “i” que utilizaremos como índice.

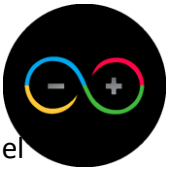
```
1  #include <stdio.h>
2
3  int main()
4  {
5     int i=0;
6  }
```

En segundo lugar escribiremos nuestro ciclo con su estructura correspondiente.

```
1  #include <stdio.h>
2
3  int main()
4  {
5     int i=0;
6     while()
7     {
8
9     }
10 }
```

Dentro de los paréntesis escribiremos nuestra condición, en este caso haremos el mismo ejemplo anterior.

```
1  #include <stdio.h>
2
3  int main()
4  {
5     int i=0;
6     while(i<10)
7     {
8
9     }
10 }
```



Solo nos queda la última sección que es la de incrementación, esta parte irá dentro del bloque de código del while al final.

```c
#include <stdio.h>

int main()
{
   int i=0;
   while(i<10)
   {


      i++;
   }
}
```

Ahora haremos el mismo ejemplo anterior, mostraremos por pantalla el valor incrementándose de nuestro índice.

```c
#include <stdio.h>

int main()
{
   int i=0;
   while(i<10)
   {
      printf("%d",i);
      i++;
   }
}
```

```
0123456789

...Program finished with exit code 0
Press ENTER to exit console.
```

Como podemos observar, el resultado es exactamente el mismo, la diferencia está en la función que utilizamos.

**❖Nota❖**

Para mostrar un resultado debajo de otro, tenemos que utilizar el comando "\n" en el momento que queramos bajar un renglón.

De esta manera nos mostraría todos los valores uno debajo del otro.

Una vez explicados estos ciclos resolveremos un problema paso a paso para ver su desenvolvimiento.
Supongamos que quiero saber todos los múltiplos de 2, desde el 0 al 100 utilizando alguna secuencia repetitiva.

1) - Primero definiremos que secuencia vamos a utilizar y la armamos.

```
#include <stdio.h>

int main()
{
    for(int i=0;i<=100;i++)
    {

    }
}
```

En este caso como nos interesa llegar al número 100, nuestro criterio debe llegar hasta ese número.

Una vez definida la secuencia, hay que pensar que nuestro índice va a pasar a valer todos los valores, de 0 a 100, por lo que usaremos esta misma para la operación.
Como sabemos este dato, lo que hay que hacer ahora es tomar una decisión de que si el número que vale "i" es o no múltiplo de 2, por ende, utilizaremos un "if", pero, ¿con que condición?
En los capítulos anteriores hemos visto cuales son los principales operadores matemáticos que se utilizan en el lenguaje, uno de ellos es el módulo que se representa con "%" en la operación. ¿Qué hace específicamente este operador?
Lo que hace es devolvernos de una división el resto, es decir, que si hacemos 10%2 nos dará como resultado 0 porque es una división exacta, en cambio, si hacemos 11%2, nos dará como resultado 1, porque no es una división exacta. Al saber esto, nos planteamos que si el resto de la división que nos devuelve es 0, quiere decir, que el número que vale "i" en ese momento, si es divisible por 2 por lo tanto si es múltiplo.

2) - Escribimos la función "if" con su correspondiente criterio.

```
#include <stdio.h>

int main()
{
    for(int i=0;i<=100;i++)
    {
        if(i%2==0)
        {

        }
    }
}
```

I%2 nos dará 2 posibles resultados: 0 o 1, es decir, múltiplo o no múltiplo, por ende, lo comparamos a 0 que si son múltiplos.

Como sabemos, esa condición va a ser verdadera únicamente cuando el valor que tenga "i" sea un múltiplo de 2, por lo tanto necesitamos mostrarlo para cumplir con la consigna.

3)- Ponemos el printf correspondiente.

```c
#include <stdio.h>

int main()
{
    for(int i=0;i<=100;i++)
    {
        if(i%2==0)
        {
            printf("%d.",i);
        }
    }
}
```

En este caso pusimos un "." Para hacer una separación entre números.

4)- Ejecutamos y vemos resultado.

```
2.4.6.8.10.12.14.16.18.20.22.24.26.28.30.32.34.36.38.40.42.44.46.48.50.52.54.56.58.60.62.64.66.68.70.72.74.76.78.8
82.84.86.88.90.92.94.96.98.100.
```

Como podemos observar. Están todos los múltiplos de 2 de 0 a 100.

## ❖*Ejercicios con sentencias repetitivas:*

1)- Ingresar 10 números por teclado y determinar su promedio. (El promedio se realiza sumando todos los números y dividiéndolos por la cantidad de números sumados).
2)- Una profesora de secundaria quiere que le programen un código, en donde ella quiere cargar la nota de sus alumnos (la cantidad de alumnos varía, por lo que ella requiere ingresar la cantidad de alumnos) y quiere saber cuántos alumnos aprobaron y cuántos alumnos reprobaron, teniendo en cuenta que se aprueba con 6.

## ❖Sentencia repetitiva con interacción

A diferencia de las sentencias repetitivas anteriores, la interacción con el usuario es limitada en algunos aspectos, es por eso que ciertas cosas no se podían hacer. Debido a eso, se originó esta alternativa.
La función que veremos a continuación es la siguiente.

```
do{

}while();
```

Esta función nos permitirá realizar un menú.
Funciona de la siguiente manera, en el bloque de código delimitado por las llaves, escribiremos todo nuestro código y lo que queramos mostrar, una vez ejecutado dicho código, se tendrá en cuenta la condición del while. Mientras esta condición sea verdadera, la función se seguirá ejecutando una y otra vez sin finalizar el programa, en el momento en el que la condición sea falsa, es cuando dará por finalizada la función.
Esto nos permite, junto con otras funciones, crear por ejemplo un menú interactivo con opciones.
Vamos a realizar un menú simple con las operaciones básicas.

1)- Escribimos la función.

```
#include <stdio.h>

int main()
{
    do{

    }while();
}
```

2)- Creamos una variable, mostramos por pantalla las opciones posibles y guardamos el valor ingresado.

```
#include <stdio.h>

int main()
{
    int a;
   do{
       printf("opción 1: suma\nopción 2: resta\nopción 3: salir\n");
       printf("digite opción:");
       scanf("%d",&a);
   }while();
}
```

Como vemos hay 3 posibles opciones del menú y la opción 3 es la que nosotros decidimos mostrarle al usuario para que finalice el programa cuando lo requiera, por lo tanto sabemos que cuando el usuario ingresa “3” es porque quiere que el programa finalice y es lo que usaremos para la condición de nuestro while.
Recordemos que bajamos un reglón con \n entre opciones para mejor entendimiento.

3)- Escribimos nuestra condición de while.

```c
#include <stdio.h>

int main()
{
    int a;
   do{

       printf("opción 1: suma\nopción 2: resta\nopción 3: salir\n");
       printf("digite opción:");
       scanf("%d",&a);
   }while(a!=3);
}
```

4)- Ahora debemos de realizar las operaciones según la opción que ingrese el usuario, para ello, primero deberemos de identificar que opción ingresó.

```c
#include <stdio.h>

int main()
{
    int a;
   do{

       printf("opción 1: suma\nopción 2: resta\nopción 3: salir\n");
       printf("digite opción:");
       scanf("%d",&a);
        if(a==1)
        {

        }
        if(a==2)
        {

        }
   }while(a!=3);
}
```

5 )- Una vez identificada la opción, haremos las operaciones correspondientes a cada opción.

```c
#include <stdio.h>

int main()
{
    int a;
   do{

       printf("opción 1: suma\nopción 2: resta\nopción 3: salir\n");
       printf("digite opción:");
       scanf("%d",&a);
        if(a==1)
        {
            int b,c,d;
            printf("ingrese numeros:");
            scanf("%d",&b);
            scanf("%d",&c);
            d=b+c;
            printf("resultado:%d\n",d);
        }
        if(a==2)
        {
            int b,c,d;
            printf("ingrese numeros:");
            scanf("%d",&b);
            scanf("%d",&c);
            d=b-c;
           printf("resultado:%d\n",d);
        }
   }while(a!=3);
}
```

Una vez hecho todo esto, nuestro código ya estará funcionando a la perfección.

## ❖*Ejercicio de sentencia repetitiva con interacción:*

1)- Realizar un menú interactivo con 5 opciones, la cuales son: suma, resta, multiplicación, división y salir. Cada una de las operaciones debe ser realizada aparte en un prototipo previamente diseñado.

# ➢ Unidad 5: ❖*Arreglos (Vectores)*

Siempre encontramos funciones destinadas a un objetivo en concreto, pero aun así estas funciones tienen sus limitaciones, como por ejemplo en las variables siempre almacenamos un único dato. Es por esto que surgen los arreglos o vectores en la programación.
Los arreglos son un conjunto de esos datos que se almacenan dentro de una misma variable. Por lo tanto no es una función propiamente dicho sino un complemento a las variables.
Lo principal de los arreglos es su longitud, es decir, la cantidad de datos que vamos a poder guardar en nuestra variable.
La forma de crear un arreglo es la siguiente:

```
1  #include <stdio.h>
2
3  int main()
4  {
5     int a [];
6  }
```

Como podemos observar lo único que se hizo fue agregarle "[]" al lado de una variable entera. Si bien es sencillo, no está terminado, dentro de los corchetes pondremos la longitud de nuestro arreglo. Hagamos un ejemplo.

1)- Creamos una variable entera con una determinada longitud.

```
1  #include <stdio.h>
2
3  int main()
4  {
5    int a[4]
6  }
```

❖Importante❖

Cuando nosotros creamos un vector de x cantidad, el primer valor se guardará en la posición 0, por lo tanto, si creamos un vector de 4 elementos, las posiciones de los valores quedaría de la siguiente manera: 0, 1, 2, 3.

2)- Ahora le daremos los 4 valores de nuestro vector. Pondremos el signo "=", abrimos y cerramos llaves, y adentro, separados por una ",", pondremos todos los valores.

```
1  #include <stdio.h>
2
3  int main()
4  {
5    int a[4] = {1,2,3,4};
6  }
```

3)- Una vez tengamos cargado nuestro vector, lo mostraremos. Para ello utilizaremos un ciclo repetitivo porque tenemos que recorrer todo nuestro vector, posición por posición (de la pos 0 a la pos 3), e ir imprimiendo esos valores.
Creamos una sentencia repetitiva que se repita la misma cantidad de veces que la longitud del vector.

```
1  #include <stdio.h>
2
3  int main()
4  {
5    int a[4] = {1,2,3,4};
6
7    for(int i=0;i<4;i++)
8      {
9
10     }
11 }
```

Una vez tenemos el ciclo, sabemos que se va a repetir 4 veces que son las 4 posiciones de nuestro vector. Imprimiremos posición por posición. Para ello sabemos que nuestro índice va a recorrer los valores de 0 a 3, entonces usaremos esa herramienta para recorrer las posiciones del vector.
Mostraremos por pantalla el contenido del vector, dentro del printf, pondremos una máscara de tipo entero, pondremos el nombre de nuestro vector y entre corchetes la variable índice (i) que va a ser la encargada de recorrer el vector.

```
1  #include <stdio.h>
2
3  int main()
4  {
5    int a[4] = {1,2,3,4};
6
7    for(int i=0;i<4;i++)
8      {
9        printf("%d.",a[i]);
10     }
11 }
```

Como vemos la máscara tomara el valor del vector “a” posición “i”, y como “i” va a ir aumentando su valor, irá imprimiendo todas las posiciones.
En este caso utilizamos un “.” Para separar valores.

De esta manera creamos un vector y mostramos su contenido gracias a un ciclo repetitivo.

Ahora bien, lo importante de esto es que el usuario sea quien decida el tamaño del vector y los valores que contendrá.
Para ello haremos lo siguiente.

1)- Creamos una variable y le ingresamos el valor por teclado. Esta variable nos servirá para determinar la longitud del vector.

```c
#include <stdio.h>

int main()
{
    int a;
    printf("ingrese longitud del vector:");
    scanf("%d",&a);
}
```

2)- Creamos nuestro vector con la longitud del valor de la variable que el usuario ingresó.

```c
#include <stdio.h>

int main()
{
    int a;
    printf("ingrese longitud del vector:");
    scanf("%d",&a);
    int vector[a];
}
```

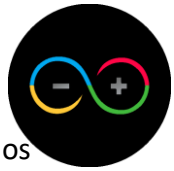
Una vez declarado el vector, tenemos que asignarle los valores, pero el usuario es quien tiene que hacerlo. Para ello pediremos que ingrese un valor por cada espacio del vector que se creó, es decir, que si ingresó que la longitud es de 4 elementos, debemos de pedir que ingrese 4 números, que se traduce en repetir 4 “scanf”.

3)- Creamos una sentencia repetitiva para ingresar los valores del vector.

```c
#include <stdio.h>

int main()
{
    int a;
    printf("ingrese longitud del vector:");
    scanf("%d",&a);
    int vector[a];
    for(int i=0;i<a;i++)
    {

    }
}
```



4)- Dentro del ciclo, tenemos que informarle al usuario que tiene que ingresar los valores para de esa manera ir guardándolos.

```c
#include <stdio.h>

int main()
{
    int a;
    printf("ingrese longitud del vector:");
    scanf("%d",&a);
    int vector[a];
    for(int i=0;i<a;i++)
    {
        printf("ingrese valor posición %d:",i+1);
        scanf("%d",&vector[i]);
    }
}
```

Como vemos en el printf le decimos que ingrese el valor y mostramos la variable “i” para que sepa en qué posición se está guardando el valor que va a ingresar, ponemos el “+1” para evitar confusiones ya que como “i” vale 0 al principio, el usuario se puede confundir.

En el “scanf” ponemos el nombre de nuestro vector sub[i], esto significa que el valor que ingrese el usuario se guardará en el vector en la posición i, por lo tanto, cuando “i” valga 0, se guardará en el primer lugar, cuando valga 1, en el segundo y así sucesivamente.

5)- Una vez hecho esto, ya tendremos cargado todo nuestro vector, por lo que ahora hay que mostrar su contenido. Para ello haremos lo mismo que en el ejercicio anterior para mostrar su contenido.

```c
#include <stdio.h>

int main()
{
    int a;
    printf("ingrese longitud del vector:");
    scanf("%d",&a);
    int vector[a];
    for(int i=0;i<a;i++)
    {
        printf("ingrese valor posición %d:",i+1);
        scanf("%d",&vector[i]);
    }
    printf("valores del vector:");
    for(int i=0;i<a;i++)
    {
        printf("%d.",vector[i]);
    }
}
```

Con esto ya habremos finalizado vectores.

## ❖Arreglos 2 dimensiones (matrices)

Al igual que los vectores, las matrices también son un complemento de las variables, la diferencia radica en las dimensiones entre uno y otro. Los vectores poseen una única dimensión, por lo que podemos decir que no poseen dimensiones tales como las conocemos, en cambio las matrices son bidimensionales por lo que significa que hay 2 dimensiones a las cuales se les puede almacenar datos, las cuales se denominan filas y columnas.
El hecho de que exista este complemento, nos permite almacenar muchos datos dentro de una misma variable y de esa forma ordenar de una forma más óptima el código final.

1)- Para crear una variable con este complemento, primero creamos una variable y pondremos 2 corchetes al lado. Estos significan los 2 ejes de la matriz.

```
1  #include <stdio.h>
2
3  int main()
4  {
5    int matriz[][]
6  }
```

2)- Definiremos la magnitud de esta matriz, en los primeros corchetes pondremos la cantidad de filas, mientras que en los siguientes pondremos la cantidad de columnas.

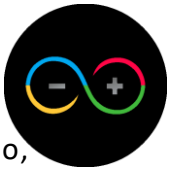
```
1  #include <stdio.h>
2
3  int main()
4  {
5    int matriz[2][3]
6  }
```

Para este ejemplo se creó una matriz con 2 filas y 3 columnas

Al igual que en los vectores, la primera posición en donde se almacenarán los datos es la 0, quedándonos la siguiente tabla representativa

| | col0 | col1 | col 2 |
|---|---|---|---|
| fila 0 | x | x | x |
| fila 1 | x | x | x |

Como vemos, tiene 2 filas con 3 columnas, tal como lo hemos definido.



3)- Ahora le asignaremos valores a los espacios creados en la matriz. Para ello, ponemos "=" y abriremos 2 pares de llaves separadas por una "," que significan las 2 filas previamente creadas.

```
#include <stdio.h>

int main()
{
  int matriz[2][3] = {{1,2,3},{4,5,6}};
}
```

Como vemos, dentro de los 2 pares de llaves, pondremos los valores de la matriz, separados con una ",". Una vez definidos los valores, encerramos esos pares de llaves con otras llaves que engloben todo.
Como son 2 filas, son 2 pares de llaves con 3 valores dentro, su fuese al revés, es decir, 3 filas con 2 columnas, tendremos 3 pares de llaves con 2 valores dentro.
Una vez tengamos cargada la matriz, debemos de mostrarla. Para ello utilizaremos un ciclo for anidado, es decir, un ciclo for dentro de otro.

4)- Creamos un ciclo for para que sea el encargado de recorrer las filas de nuestra matriz.

```
#include <stdio.h>

int main()
{
  int matriz[2][3] = {{1,2,3},{4,5,6}};
   for(int i=0;i<2;i++)
   {

   }
}
```

Como tenemos 2 filas, necesitamos que se repita 2 veces únicamente.

Gracias a este ciclo, recorreremos la fila correspondiente.

5)- Ahora crearemos otro ciclo for, ya que necesitamos que también recorra las columnas de la matriz.

```
#include <stdio.h>

int main()
{
  int matriz[2][3] = {{1,2,3},{4,5,6}};
   for(int i=0;i<2;i++)
   {
       for(int j=0;j<3;j++)
       {

       }
   }
}
```

Como vemos, el límite es 3 ya que son 3 columnas.

Como se muestra en la imagen, utilizamos 2 variables distintas para el ciclo for, ya que la variable “i” la vamos a utilizar para las filas de la matriz, y la variable “j” la usaremos para las columnas.
Una vez hecho esto sabemos que “i” arranca con 0 ya que es la primera fila, y con ese valor ingresa al segundo for que es quien recorre las columnas, por lo que debemos ahora de imprimir los valores de la primera fila recorriendo columna por columna.

6)- Escribiremos un printf en donde mostraremos la matriz en la posición i=0 y j=0.

```
#include <stdio.h>

int main()
{
  int matriz[2][3] = {{1,2,3},{4,5,6}};
   for(int i=0;i<2;i++)
   {
       for(int j=0;j<3;j++)
       {
           printf("%d ",matriz[i][j]);
       }
   }
}
```

7)- Una vez hecho esto nos faltaría separar las filas de las columnas para que se vea en formato de matriz, por lo que una vez impreso toda la columna, haremos un salto de renglón.

```
#include <stdio.h>

int main()
{
  int matriz[2][3] = {{1,2,3},{4,5,6}};
   for(int i=0;i<2;i++)
   {
       for(int j=0;j<3;j++)
       {
           printf("%d ",matriz[i][j]);
       }
       printf("\n");
   }
}
```

8)- Ejecutamos y vemos el resultado.

```
1 2 3
4 5 6


...Program finished with exit code 0
Press ENTER to exit console.
```

Como ya sabemos, el usuario es quien tiene que ser el protagonista, por lo que él es quien tiene que decidir la magnitud de nuestra matriz y los valores.

1)- Creamos 2 variables enteras y le pedimos al usuario que ingrese los valores correspondientes.

```c
#include <stdio.h>

int main()
{
    int a,b;
    printf("ingrese cantidad de filas de la matriz:");
    scanf("%d",&a);
    printf("ingrese cantidad de columnas de la matriz:");
    scanf("%d",&b);
}
```

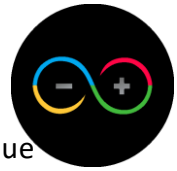
Nuestra variable “a” almacenará la cantidad de filas y nuestra variable “b”, almacenará nuestra cantidad de columnas.

2)- Creamos la matriz con esos valores.

```c
#include <stdio.h>

int main()
{
    int a,b;
    printf("ingrese cantidad de filas de la matriz:");
    scanf("%d",&a);
    printf("ingrese cantidad de columnas de la matriz:");
    scanf("%d",&b);
    int matriz[a][b];
}
```

Como sabemos que los primeros corchetes indican las filas, y la variable “a” es quien las contiene, la nombramos, de igual manera hacemos con las columnas con la variable “b”.



Una vez creada y determinada su magnitud, hay que completarla con los datos que ingrese el usuario.

3 )- Seguiremos el principio que hicimos para mostrar la matriz. Primero creamos un "for" para las filas y otro dentro para las columnas.

```
#include <stdio.h>

int main()
{
    int a,b;
    printf("ingrese cantidad de filas de la matriz:");
    scanf("%d",&a);
    printf("ingrese cantidad de columnas de la matriz:");
    scanf("%d",&b);
    int matriz[a][b];
     printf("ingrese datos de la matriz:\n");
    for(int i=0;i<a;i++)
    {
        for(int j=0;j<b;j++)
        {

        }
    }
}
```

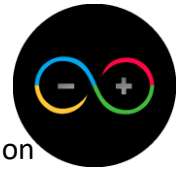
Como sabemos, los ciclos se tienen que repetir en base a la cantidad de filas/columnas que haya, por lo que nuestra condición está determinada por las variables "a" y "b". Con esta estructura de ciclos, recorreremos toda la matriz, pero en vez de mostrar los datos, hay que almacenarlos, por lo que usaremos un "scanf" para ir guardando esos datos.

4)- Para que el usuario entienda en qué lugar de la matriz está guardando el valor, primero, por una cuestión de practicidad, le mostraremos el lugar exacto de la matriz, es decir, decirle en que fila y que columna esta parado, seguido del scanf.

```c
#include <stdio.h>

int main()
{
    int a,b;
    printf("ingrese cantidad de filas de la matriz:");
    scanf("%d",&a);
    printf("ingrese cantidad de columnas de la matriz:");
    scanf("%d",&b);
    int matriz[a][b];
     printf("ingrese datos de la matriz:\n");
    for(int i=0;i<a;i++)
    {
        for(int j=0;j<b;j++)
        {
            printf("fila %d columna %d: ",i+1,j+1);
            scanf("%d",&matriz[i][j]);
        }
    }
}
```

Como vemos en el printf le está dando la información de en qué fila y columna está.
Para evitar confusiones con la posición 0, le ponemos un “+1” para que empiece a contar de 1 en adelante. Esto no afecta a la matriz original.
Luego con el scanf guardamos los valores en la posición de filas “i” y posición de columnas “j”.
Hecho esto ya tenemos cargada toda la matriz.



5 )- Mostramos la matriz obtenida de la misma manera que hicimos previamente con las matrices pre-cargadas.

```c
#include <stdio.h>

int main()
{
    int a,b;
    printf("ingrese cantidad de filas de la matriz:");
    scanf("%d",&a);
    printf("ingrese cantidad de columnas de la matriz:");
    scanf("%d",&b);
    int matriz[a][b];
     printf("ingrese datos de la matriz:\n");
    for(int i=0;i<a;i++)
    {
        for(int j=0;j<b;j++)
        {
            printf("fila %d columna %d: ",i+1,j+1);
            scanf("%d",&matriz[i][j]);
        }
    }
    printf("valores de la matriz:\n");
   for(int i=0;i<a;i++)
   {
       for(int j=0;j<b;j++)
       {
           printf("%d ",matriz[i][j]);
       }
       printf("\n");
   }
}
```

6 )- Ejecutamos y vemos el resultado.

```
ingrese cantidad de filas de la matriz:2
ingrese cantidad de columnas de la matriz:3
ingrese datos de la matriz:
fila 1 columna 1: 1
fila 1 columna 2: 2
fila 1 columna 3: 3
fila 2 columna 1: 4
fila 2 columna 2: 5
fila 2 columna 3: 6
valores de la matriz:
1 2 3
4 5 6


...Program finished with exit code 0
Press ENTER to exit console.
```

## ❖*Ejercicio de vectores y matrices:*

1)- Crear un vector, que el usuario elija magnitud y valores del vector. Mostrar por pantalla primer y último valor del vector.

2)- Crear una matriz, que el usuario elija magnitud y valores de la matriz. Mostrar por pantalla primer y último valor de dicha matriz.

# ➢ Unidad 6: ❖*Estructuras*

Una estructura es una colección de uno o más tipo de elementos denominados miembros, estos miembros pueden ser de distintos tipos de datos.
Al definir la estructura como una colección, podemos deducir que estos miembros estarán todos dentro de la misma estructura sin importar su tipo de dato.
Las estructuras nos permitirán juntar variables de distintos tipos pero que compartan el mismo fin.
Haremos un ejemplo práctico para entender las estructuras.
Supongamos que tenemos una mascota y queremos almacenar su nombre, edad y peso, por lo que crearemos una estructura para ello.

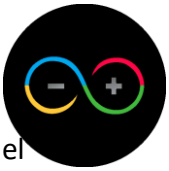
1)- Para crear una estructura usaremos la palabra "struct" seguido del nombre de la estructura (este nombre es solo para nombrar la estructura general), luego abrimos y cerramos llaves. Esto debe ser realizado antes del main, como si fuese un prototipo.

```
1  #include <stdio.h>
2  struct mascota
3  {
4
5  }
6  int main()
7  {
8
9  }
```

2)- Dentro de la estructura guardaremos los datos de nombre, edad y peso. Usaremos un vector para el nombre, un entero para la edad y un decimal para el peso.

```
1  #include <stdio.h>
2  struct mascota
3  {
4    char nombre[30];
5    int edad;
6    float peso;
7  }
8  int main()
9  {
10
11 }
```

Como vemos, únicamente declaramos los tipos de datos y el nombre de las variables que usaremos.



3)- Para asignar los valores a estas variables, luego de la llave de cierre, ponemos el nombre de la variable que va a contener los elementos de la estructura.

```
1  #include <stdio.h>
2  struct mascota
3  {
4    char nombre[30];
5    int edad;
6    float peso;
7  }perro
8  int main()
9  {
10
11 }
```

Como podemos observar, “mascota” es el nombre de la estructura que contiene todos los miembros y “perro” es la variable que usará los miembros de la estructura para otorgarles un valor.

4)- Se otorgan los valores de forma secuencial, esto quiere decir, que primero le daremos el nombre, luego la edad y por último el peso. Para ello pondremos un “=” seguido de llaves de apertura y cierre y dentro de estas llaves, pondremos los datos correspondientes, separados por una “,”.

```
1  #include <stdio.h>
2  struct mascota
3  {
4    char nombre[30];
5    int edad;
6    float peso;
7  }perro={"roco",3,10.3};
8  int main()
9  {
10
11 }
```

En este ejemplo, nuestro perro se llama roco, tiene 3 años y pesa 10.3kg.

5)- Ya terminamos nuestra estructura con sus datos, ahora para mostrar esos datos iremos a nuestro main. Pondremos un “printf” y primero mostraremos el nombre de nuestra mascota, como es una palabra, la máscara que utilizaremos es “%s” para mostrarlo. Fuera de las comillas pondremos el nombre de la variable de la estructura (“perro”) y separado con un punto, el nombre del miembro que estamos haciendo referencia.

```
1  #include <stdio.h>
2  struct mascota
3  {
4    char nombre[30];
5    int edad;
6    float peso;
7  }perro={"roco",3,10.3};
8  int main()
9  {
10   printf("el nombre de mi perro es: %s",perro.nombre);
11 }
```

6)- Ahora mostraremos además del nombre, su edad, por lo que le agregaremos a esa función, otro miembro. Para agregar la edad, como es una variable tipo entero, pondremos la máscara correspondiente, y separado con una "," haremos lo mismo que en el punto anterior, pondremos el nombre de la variable "perro" con el miembro que almacena la edad.

```c
#include <stdio.h>
struct mascota
{
  char nombre[30];
  int edad;
  float peso;
}perro={"roco",3,10.3};
int main()
{
  printf("el nombre de mi perro es: %s, tiene %d años",perro.nombre,perro.edad);
}
```

7)- Por último nos queda mostrar el peso, lo haremos de la misma manera.

```c
#include <stdio.h>
struct mascota
{
  char nombre[30];
  int edad;
  float peso;
}perro={"roco",3,10.3};
int main()
{
  printf("el nombre de mi perro es: %s, tiene %d años y pesa %.2f kg",perro.nombre,perro.edad,perro.peso);
}
```

8)- Si mandamos a ejecutar nos mostrará todos los datos.

```
el nombre de mi perro es: roco, tiene 3 años y pesa 10.30 kg

...Program finished with exit code 0
Press ENTER to exit console.
```

Ahora bien, supongamos que tenemos más de una mascota.
Para agregar a otra mascota en nuestra estructura, cuando cierra la llave de la anterior, pondremos una "," y repetiremos proceso para guardar y mostrar los datos.

```c
#include <stdio.h>
struct mascota
{
  char nombre[30];
  int edad;
  float peso;
}perro={"roco",3,10.3},
gato={"alex",1,5.2};
int main()
{
  printf("el nombre de mi perro es: %s, tiene %d años y pesa %.2f kg\n",perro.nombre,perro.edad,perro.peso);
  printf("el nombre de mi gato es: %s, tiene %d año/s y pesa %.2f kg",gato.nombre,gato.edad,gato.peso);
}
```

Si mandamos a ejecutar estos comandos:

```
el nombre de mi perro es: roco, tiene 3 años y pesa 10.30 kg
el nombre de mi gato es: alex, tiene 1 año/s y pesa 5.20 kg

...Program finished with exit code 0
Press ENTER to exit console.
```

Como vemos obtenemos los distintos resultados para cada una de las variables con la misma estructura.

## ❖*Ejercicio de estructuras:*

1)- Crear una estructura a criterio propio y mostrar 3 tipos distintos de variables con la misma estructura original.

# Proyecto final

Una fábrica de autos decide implementar un sistema de seguridad en la planta industrial para sus 20 empleados, por lo que sus requisitos son:

- Un programa que pida al trabajador un pin de desbloqueo para poder acceder a su puesto de trabajo. En el caso que sea la primera vez del trabajador usando el programa, este mismo mostrará la opción de registrar un nuevo pin de desbloqueo para dicho trabajador, por lo que, una vez registrado el pin de desbloqueo, el trabajador ya podrá iniciar sesión con el pin con el cual se registró previamente.

- El programa nunca debe finalizar, ya que de caso contrario, el programa perderá los datos de los pines registrados de los trabajadores.