CONCEPTOS BÁSICOS DEL LENGUAJE C

ING. GIANKARIS G. MORENO R., M.SC.

OBJETIVOS

- Conocer las características del lenguaje de programación C.

- Identificar las reglas de escrituras sobre las cuales se desarrollan programas en el lenguaje C.

CONCEPTOS GENERALES DEL LENGUAJE C

1. Origen
2. Reglas generales del C

1.

Origen

ORIGEN DE C

- Creado a principios de los años 70's por Dennis Ritchie en los Laboratorios Bell de AT&T.

- Basado en el lenguaje B y a su vez en BCPL (Basic Combined Programming Language)

ORIGEN DE C

- Fue creado principalmente para el desarrollo de Sistemas Operativos (Unix).

- Uno de los principales lenguajes de programación para el desarrollo de aplicaciones y de otros lenguajes de programación.

CARACTERÍSTICAS DE C

- Es de propósito general, pero mayormente orientado al área de sistemas operativos.
- Requiere de un compilador para ejecutar sus programas.
- Permite programar tanto en alto como en bajo nivel.

CARACTERÍSTICAS DE C

- Estructurado se maneja a través de funciones.
- Portable puede emigrar de una plataforma a otra sin grandes costos.
- Es sensible en su sintaxis

2.

Reglas Generales del C

REGLAS GENERALES DEL C

- Todo programa debe contener la función main ( )

```
void main () {
    /* Declaraciones de variables y constantes*/
    :
    /* Declaraciones de instrucciones ejecutables*/
    instrucción1;
    instrucción2;
    :
    sentencia n;
}
```

REGLAS GENERALES DEL C

- Toda entidad del programa, entendiéndose por entidad (variables, constantes, arreglos, etc), debe ser declarado antes de utilizarse.

Ejemplos:

```
int x, y;
float result;

scanf("%i %i", &x, &y);
result= x + y;
```

3.1. Mayúsculas vs minúsculas

- C es sensible a la sintaxis por lo que hace diferencia entre la mayúscula y minúscula en los nombres de identificadores creados por el usuario.

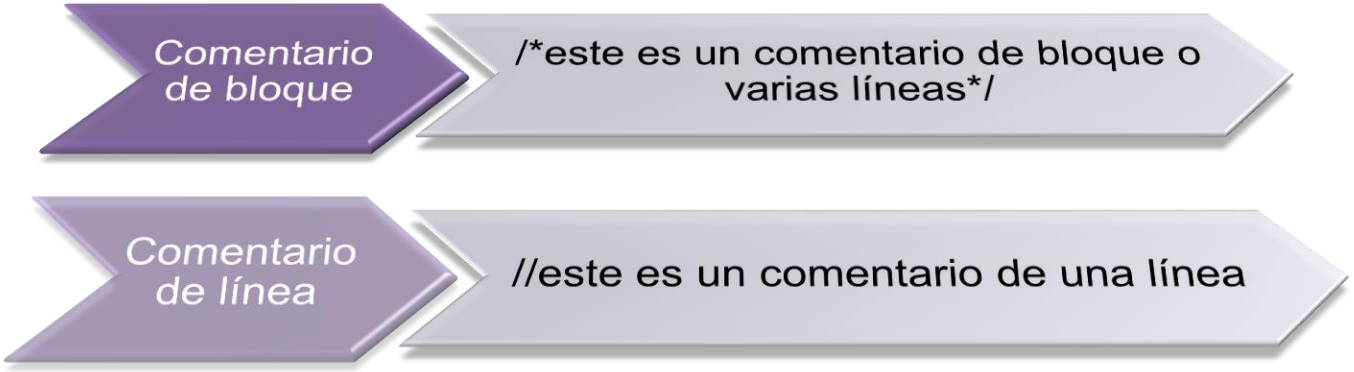
- Regla válida para cualquier instrucción en el lenguaje.

Ejemplos:

Edad edad EdAd EDad EDAd eDad
Salario salario SalariO SaLaRiO

3.3. COMENTARIOS

Cuando deseamos documentar nuestros programas en C, existen dos tipos de comentarios.

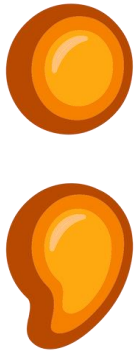


3.3. PUNTO Y COMA

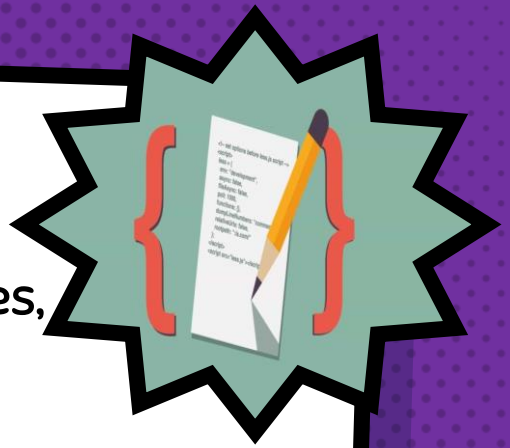
📌 Toda sentencia o instrucción debe finalizar con punto y coma (;)

Ejemplos:

```
int edad;
cont= cont + 1;
distancia = vel * tiempo;
```



3.4. LLAVES



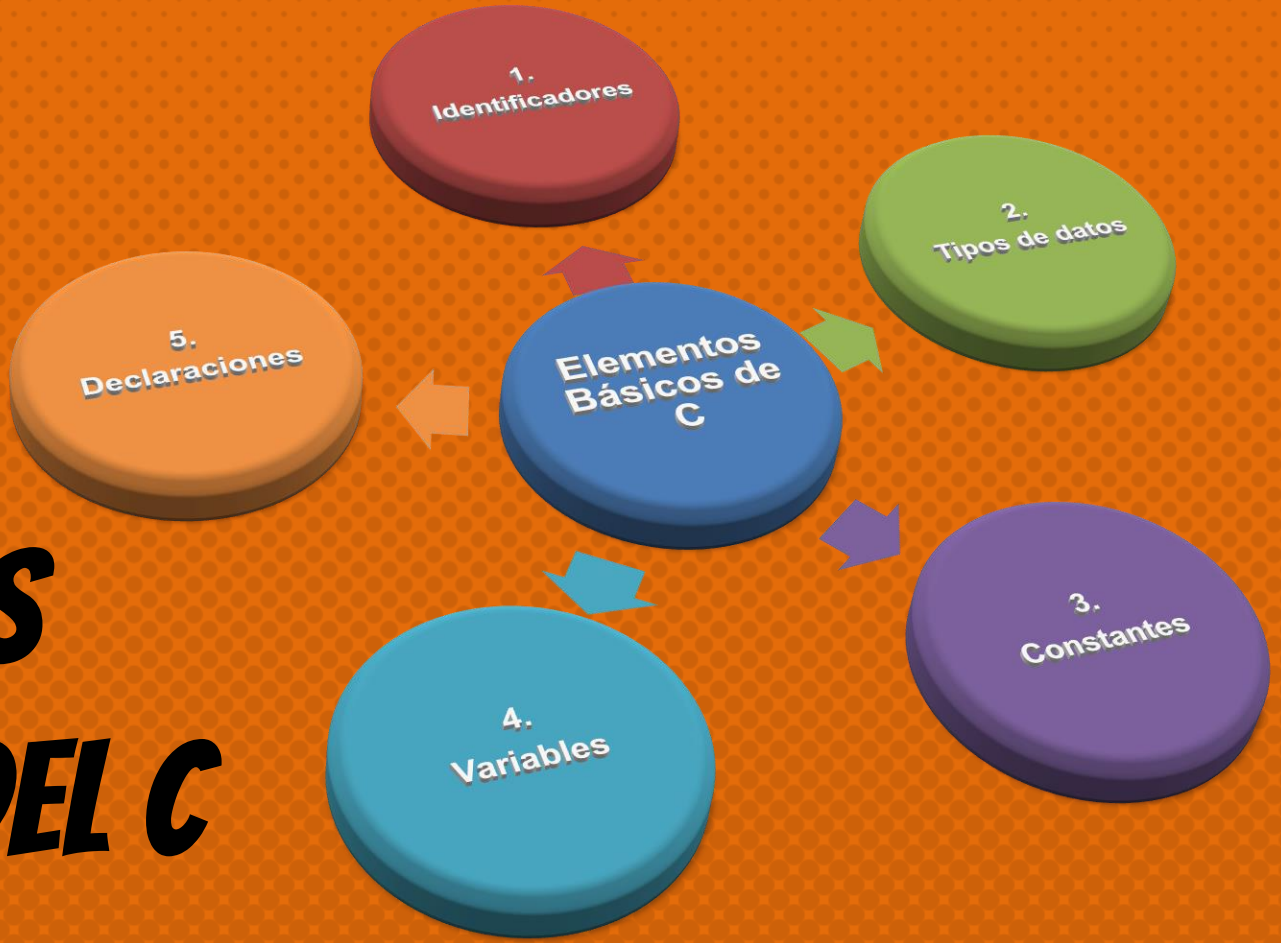
- Se utilizan para delimitar bloques de instrucciones, como las estructuras alternativas, estructuras Repetitivas, las funciones, etc.

Ejemplos:

```
if (a > b) {
        a= a * 5;
}
```

```
void main() {
    printf("Hola mundo");
}
```

ELEMENTOS BÁSICOS DEL C



1.

Identificadores y Palabras Clave

IDENTIFICADORES

- Son nombres creados por el usuario para identificar las entidades dentro del programa.
- Están compuesto de letras, números, y por el caracter subrayado (_).

REGLAS PARA NOMBRES DE IDENTIFICADORES

- El primer carácter debe ser siempre una letra
- No hay limite en la longitud, manejaremos un largo máximo de 10 caracteres.
- Son sensibles a la mayúscula y minúscula.
- No pueden ser palabras reservadas.

PALABRAS CLAVES (RESERVADAS)

Son nombres de identificadores que tiene una función específica para el compilador. A continuación presentamos un listado con algunas de ellas:

char int float double if else do while for

switch short long extern static default

continue break register sizeof typedef

No pueden ser utilizadas como identificadores creados por el usuario. Sería un error

2.

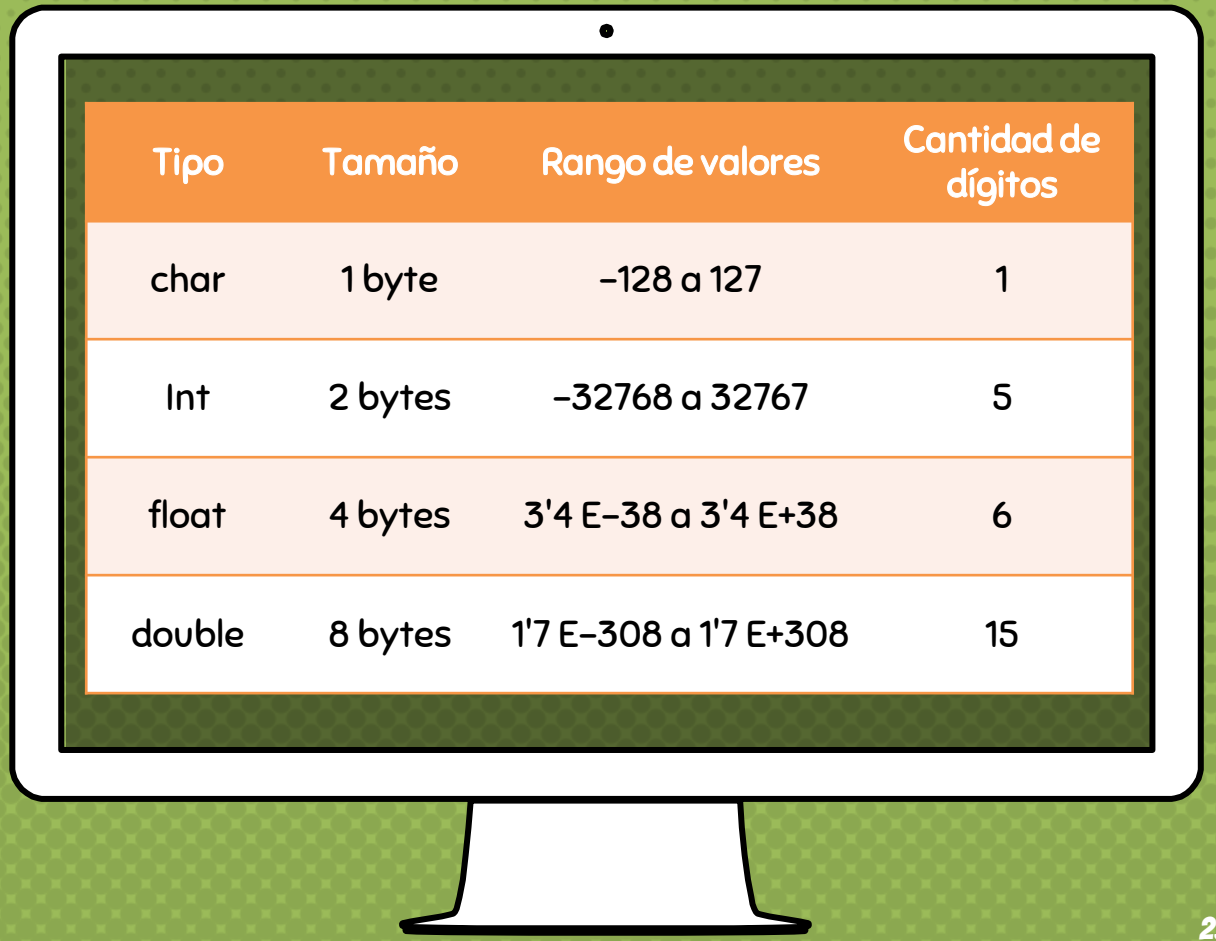
Tipos de Datos Básicos

TIPOS DE DATOS BÁSICOS

- Representan los valores que pueden tomar un objeto definido por el usuario.

- C manipula cuatro tipos de datos estándar:
  - char
  - int
  - float
  - double

TIPOS DE DATOS BÁSICOS



Tipo	Tamaño	Rango de valores	Cantidad de dígitos
char	1 byte	-128 a 127	1
Int	2 bytes	-32768 a 32767	5
float	4 bytes	3'4 E-38 a 3'4 E+38	6
double	8 bytes	1'7 E-308 a 1'7 E+308	15

3.

Constantes

CONSTANTES

- Son aquellas que mantienen su valor durante la ejecución de un programa.
- Existen dos maneras de declarar constantes, a través de:
  - La palabra reservada **const**
  - Una directiva **#define**

4.

Variables

VARIABLES

- Son objetos/elementos que pueden cambiar su valor durante la ejecución del programa.

- Todas las variables deben ser declaradas antes de ser utilizadas.

VARIABLES

Globales

Se declaran antes de la función main(). Pueden ser utilizadas en cualquier parte del programa y se destruyen al finalizar éste.

Locales

Se declaran dentro de la función en la que vayan a ser utilizadas. Sólo existen dentro de esta función y se destruyen al finalizar ésta.

5.

Declaraciones

DECLARACIÓN DE CONSTANTES

const

Formato:

```
const  tipo_dato  NOMBRE_IDENTIFICADOR = valor;
```

Ejemplos:

```
const  float PI = 3.1416;
const  int LIMITE = 85;
```

DECLARACIÓN DE CONSTANTES

#define

Formato:

```
#define  NOMBRE_IDENTIFICADOR  valor
```

Ejemplos:

```
#define PI 3.1416
#define MAX 110
```

DELCARACIÓN DE VARIABLES

Formato:

```
tipo_dato   nombre_identificador ;
```

Ejemplos:

```
float promedio;
int nota1, nota2, nota3;
char genero;
```

ENTRADA Y SALIDA DE DATOS

1. Con formato
    a. scanf ( )
    b. printf ( )
2. Sin formato
    a. gets ( ), getch ( ), getchar ( )
    b. puts ( ), putchar ( )

1.

Con Formato

CON FORMATO

- En C existen sentencias que realizan tareas directamente sobre los datos, algunas de ellas requieren de dispositivos de entrada o de salida para manipular a los datos.

- Para trabajar las sentencias de entrada, salida, el lenguaje C requiere el manejo de conjuntos de caracteres especiales que identifiquen a cada tipo de dato.

CON FORMATO

Para entrada y salida de datos en C se debe emplear el uso de caracteres de control que indican el tipo de dato que se desea representar, para ello se utiliza el carácter especial **%**.

% carácter que define el tipo de dato a usar

Ejemplos:

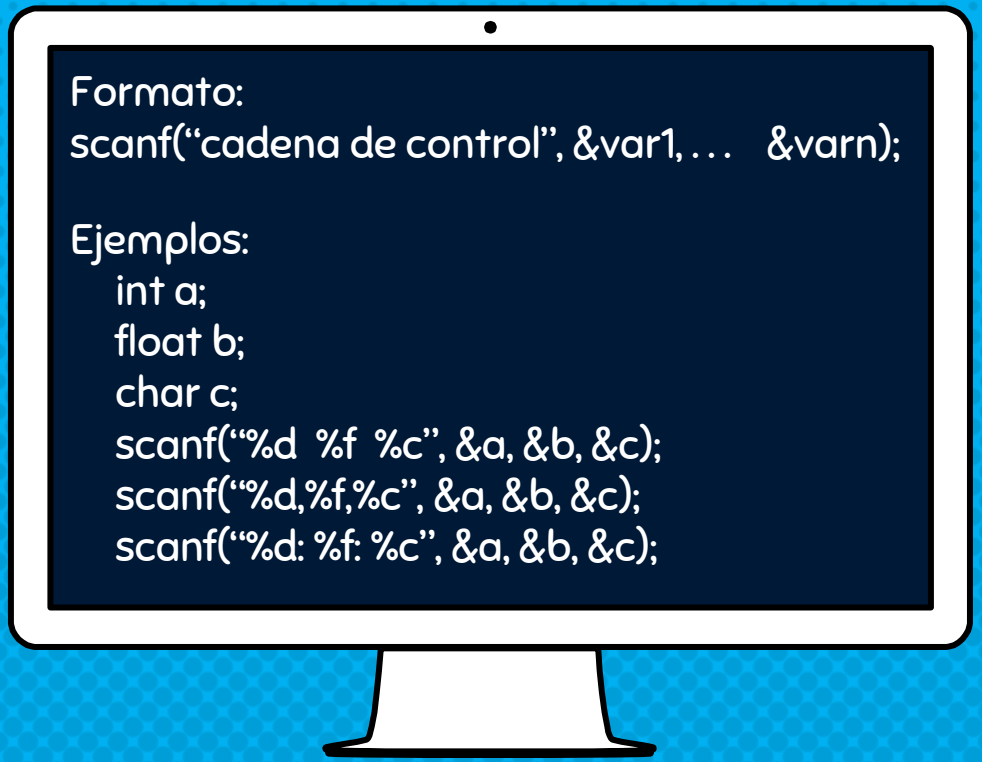
%i entero	%s cadena
%f flotante	%d entero

CARÁCTER DE CONTROL	TIPO DE DATO	EXPLICACIÓN
%d	int	entero con signo (6 dígitos)
%i *	int	entero con signo
%u	Int	entero sin signo
%f *	float	flotante con signo de la forma $^{+}_{-}$ ddd.dd
%e	double	flotante con signo de la forma $^{+}_{-}$ d.ddde $^{+}_{-}$ ddd
%c *	char	un solo caracter
%s *	char	cadena de caracteres sin espacios en blanco
%[^\n]	char	Cadena de caracteres con espacios en blanco
%o	int	octal sin signo (0 - 7)
%x	Int	hexadecimal sin signo(0 -9 , A - F)
%ld	Long	entero grande
%lf	double	double

1.1.
SCANF( )

SCANF( )

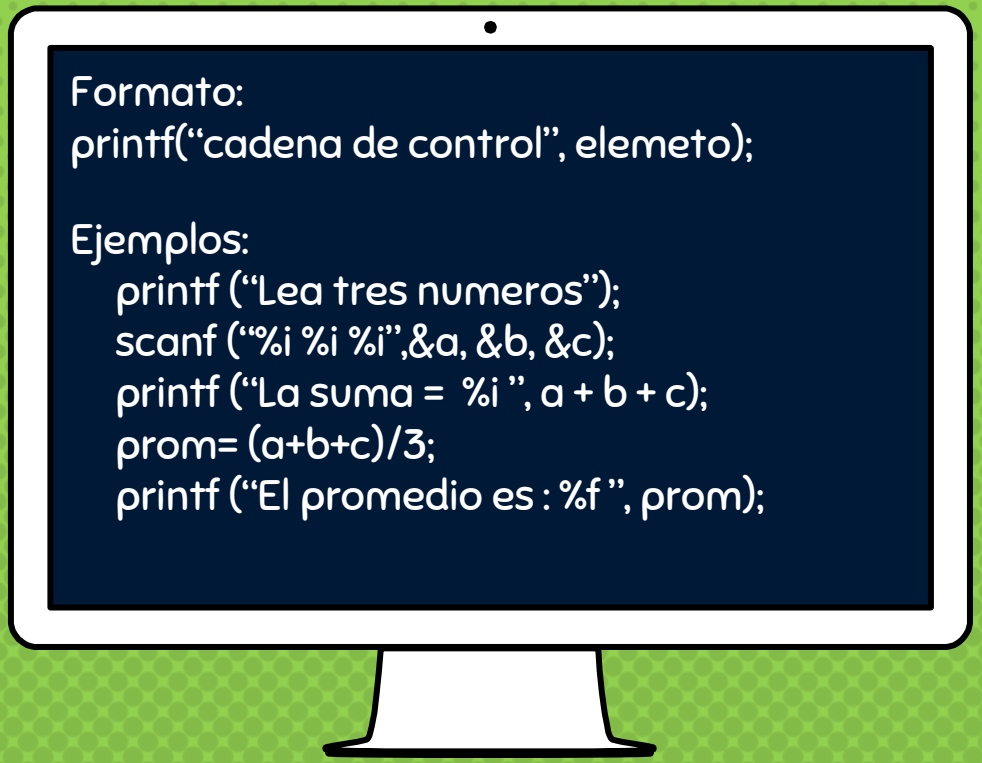
Toma un dato del dispositivo estándar de entrada (teclado) y lo almacena en una variable. Si se leen más de una variable se pueden colocar en una misma instrucción separándolo por comas.



1.2.
PRINTF( )

PRINTF()

Permite mostrar en pantalla todos los resultados del programa al igual que los mensajes que se deseen



2.

Sin Formato

SIN FORMATO

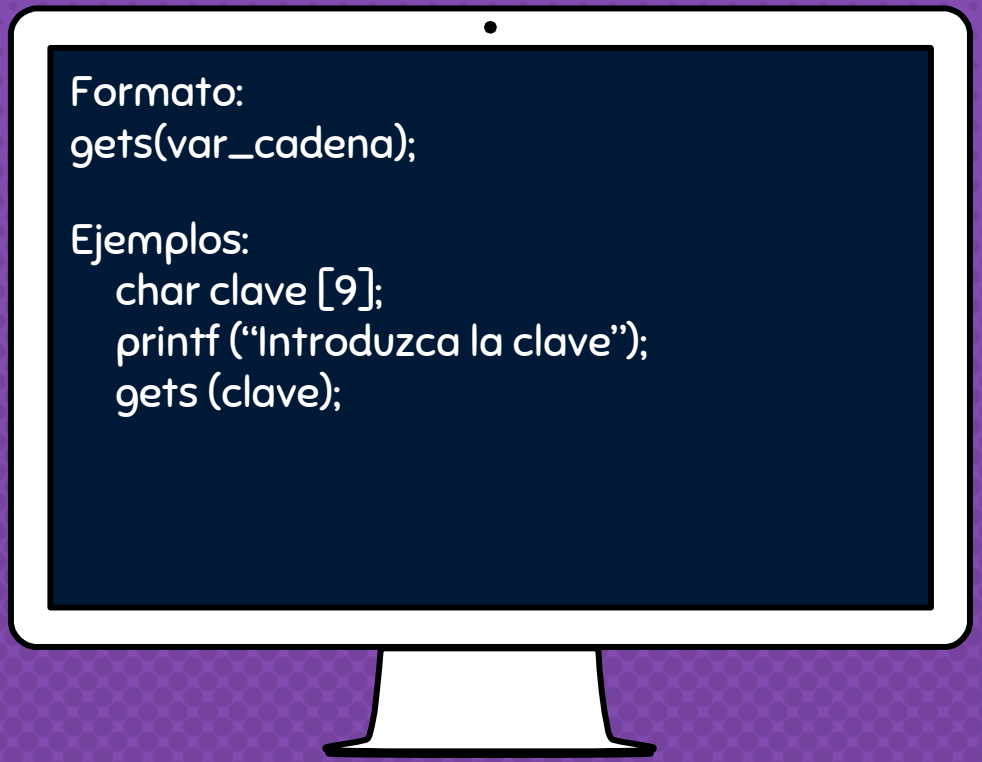
- En C también existen sentencias o funciones que no requieren especificar el formato de los valores que se desean manipular en las entradas y salidas de los datos.

2.1.

GETS(), GETCH(), GETCHAR()

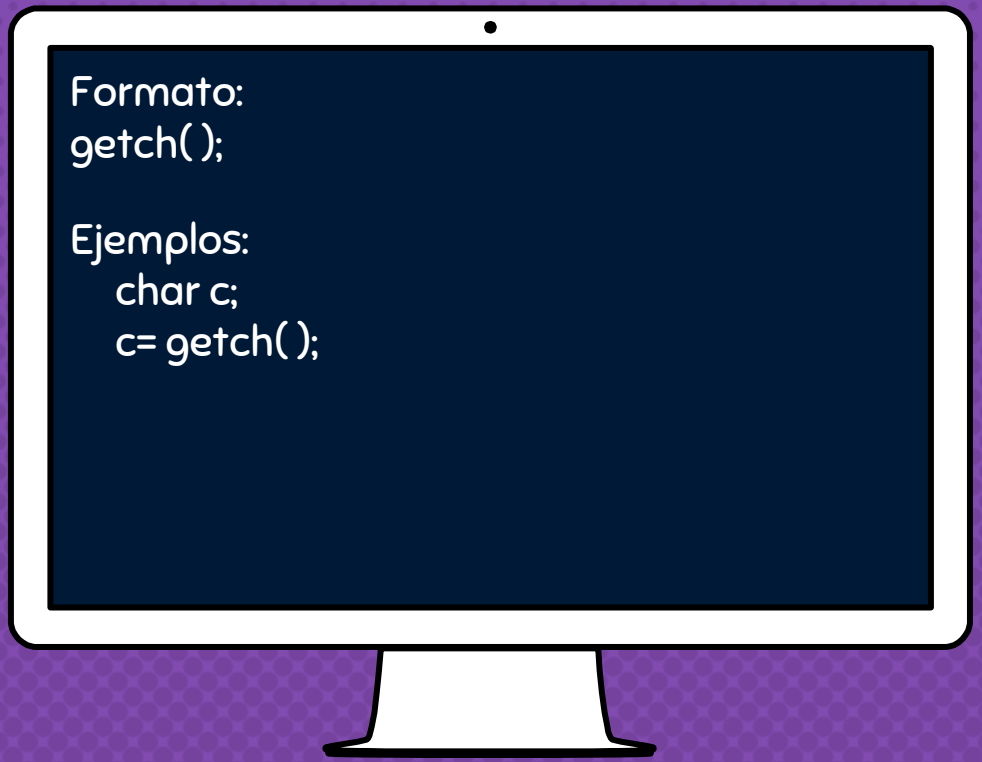
GETS()

Esta función solo maneja un argumento, el cual es una variable tipo cadena, lee la cadena y la devuelve en la variable dada como argumento.



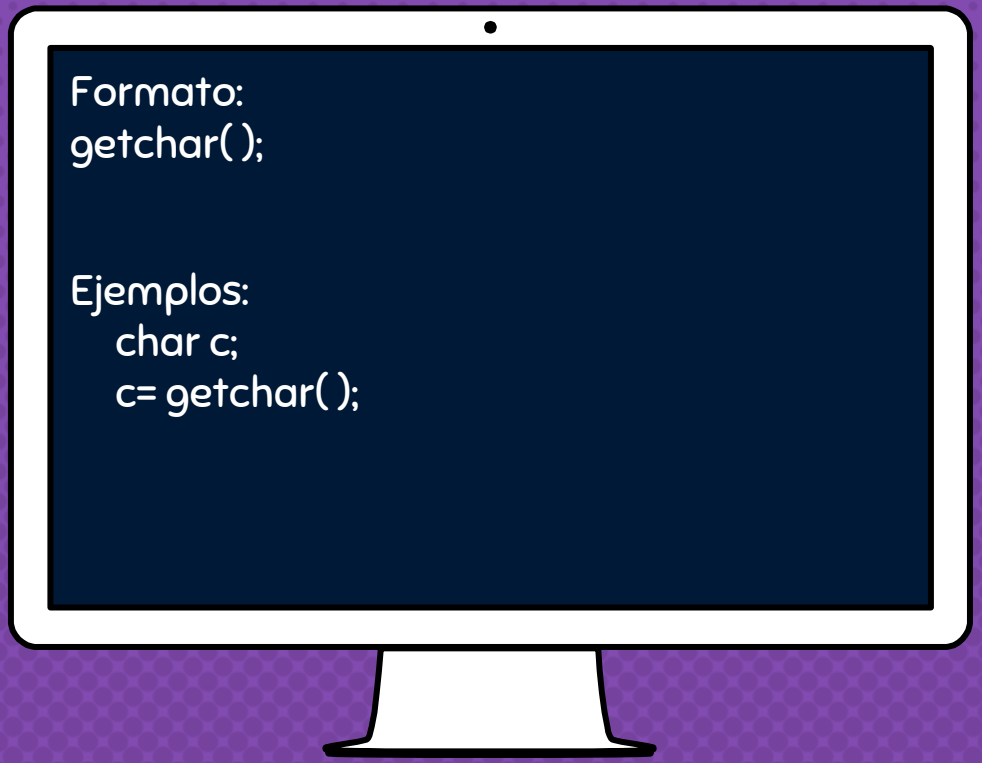
GETCH( )

Esta función permite ingresar un carácter, el mismo no se ve en la pantalla al teclearse, pero lo acepta sin la necesidad de presionar enter



GETCHAR( )

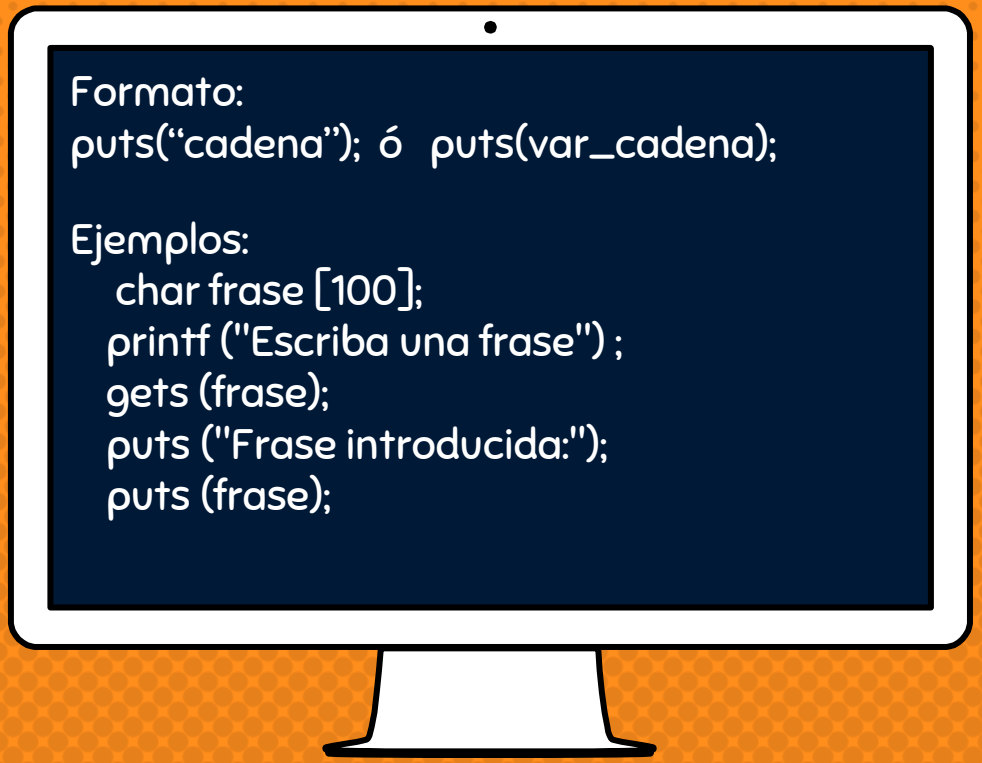
Esta función permite ingresar un carácter, el mismo se ve en la pantalla al teclearse



2.2.
PUTS( ), PUTCHAR( )

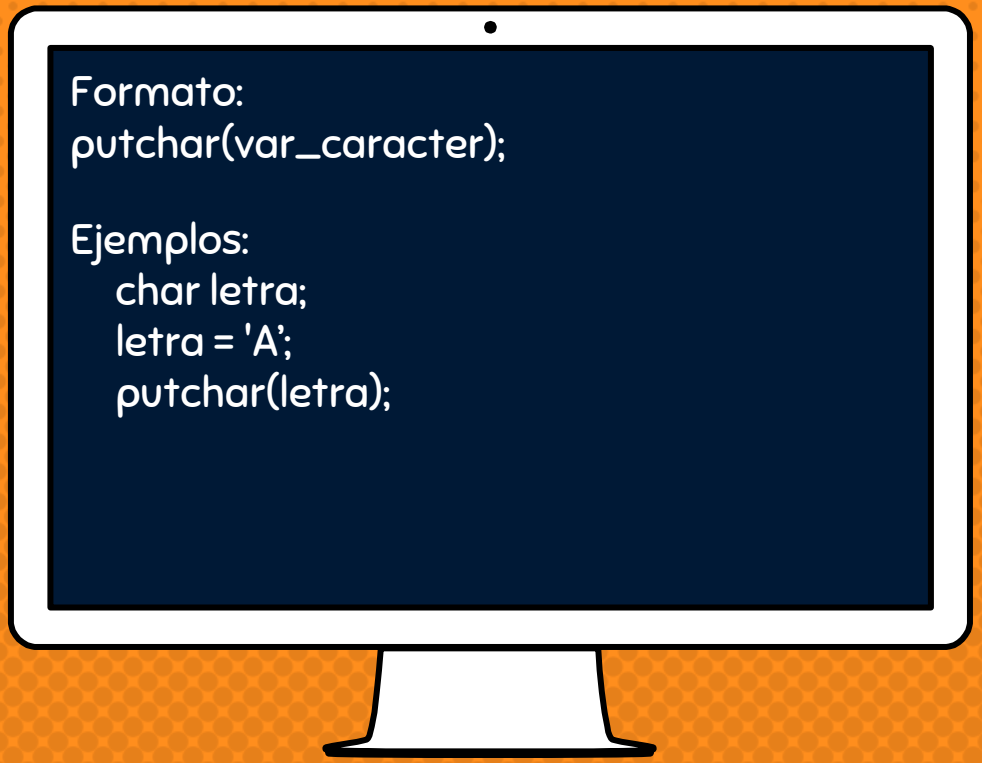
PUTS()

Permite mostrar en pantalla datos de tipo cadena.



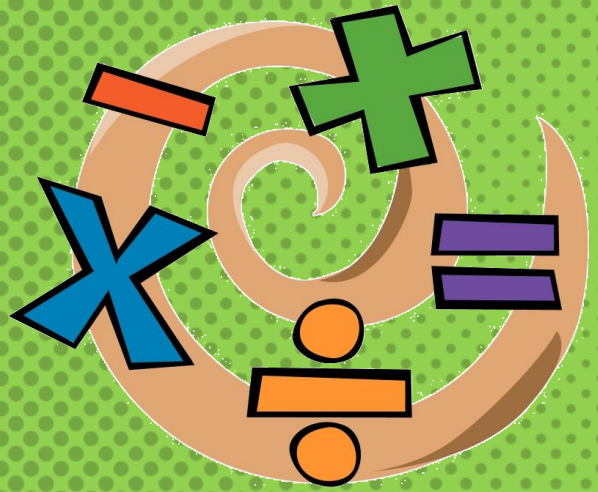
PUTCHAR()

Permite mostrar en pantalla un dato de tipo caracter.



OPERADORES Y EXPRESIONES

1. Operadores aritméticos
2. Operadores monarios
3. Operadores relacionales y lógicos
4. Operadores de asignación
5. Operador condicional
6. Operador sizeof (tamaño de)



1. Operadores Aritméticos

OPERADORES ARITMÉTICOS

Son aquellos que permiten realizar cálculos con valores numéricos para obtener un resultado.

SÍMBOLO	OPERACIÓN	TIPO DE OPERANDOS	TIPO DE RESULTADOS
*	Multiplicación	entero o real	entero o real
/	División	real	real
+	Suma	entero o real	entero o real
-	Resta	entero o real	entero o real
%	Residuo	entero	entero

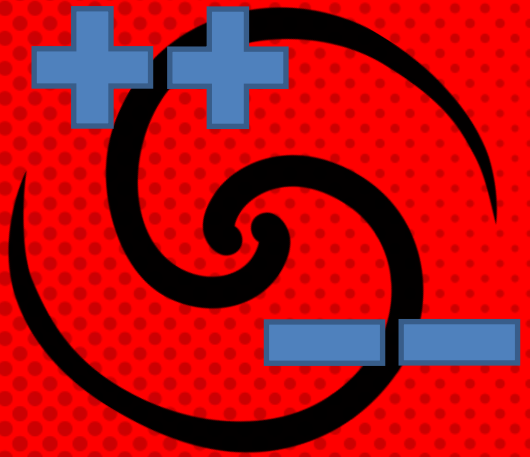
OPERADORES ARITMÉTICOS

- No existe operador de exponenciación en C, sin embargo la función pow(b,e), realiza la exponenciación a cualquier potencia.

- El operador % requiere que los dos operandos sean enteros y el segundo operando no puede ser cero.

2.

Operadores Monarios

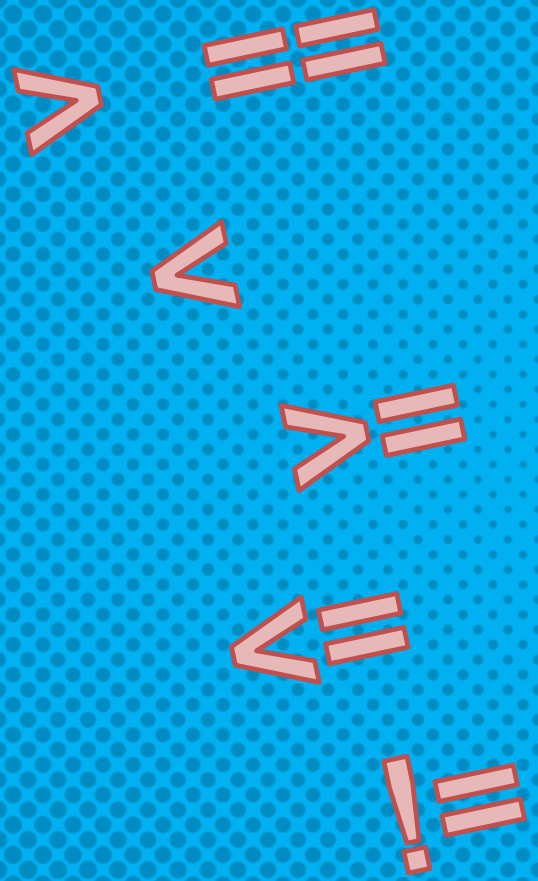


OPERADORES MONARIOS

- Son aquellos que actúan sobre un solo operando, para producir un nuevo valor.

- Los operadores monarios suelen preceder a su único operando, aunque algunos operadores monarios se escriben después de su operando.

SÍMBOLO	OPERACIÓN	TIPO DE OPERANDOS	TIPO DE RESULTADOS
++	Incremento	entero	entero
--	Decremento	entero	entero



3. Operadores Relacionales y Lógicos

OPERADORES RELACIONALES

📌 Son aquellos que se utilizan para expresar condiciones. Se emplean para realizar comparaciones entre valores.

OPERADOR	DESCRIPCIÓN
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
==	Igual a
!=	Distinto a

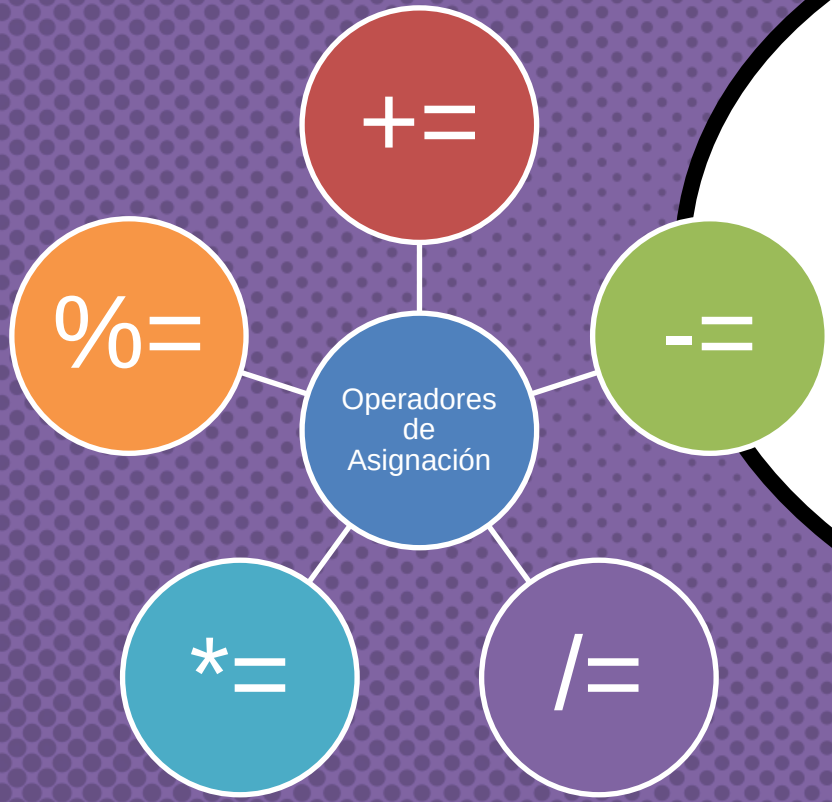
OPERADORES LÓGICOS

📌 Son aquellos que permiten relaciones lógicas y se emplean para representar condiciones compuestas.

SÍMBOLO	EXPRESIÓN LÓGICA	SIGNIFICADO
!	!variable	Negación
&&	(num > 5 && num < 10)	Afirmación
||	(num < 5 || num > 10)	Disyunción

4.

Operadores de Asignación



OPERADORES DE ASIGNACIÓN

Son aquellos que permiten evaluar la expresión y asigna el valor a la variable a la izquierda del símbolo de asignación.

OPERADOR	DESCRIPCIÓN
=	asignación
+=	suma más asignación
-=	resta más asignación
*=	multiplicación más asignación
/=	división más asignación
%=	residuo más asignación

5. Operador Condicional

OPERADOR CONDICIONAL

- Permite comparer valores y de ser cierta la condición asigna un valor, y en caso contrario asigna otro.

Formato:

```
var= operando1 ? operando2 : operando3;
```

- La expresión operando1 debe ser de tipo int o float, si es cierto, la variable toma el valor del operando2, de ser falso toma el valor del operando3.

OPERADOR CONDICIONAL

Ejemplo:

```
int tiempo,num, mayor;
mayor = 100;
printf("Introduzca un numero: ");
scanf("%i",&num);
tiempo= (num > mayor) ? num : mayor;
printf("El valor de tiempo es: %i", tiempo);
```

num→ 67 tiempo= 100

num→104 tiempo=104

6.

Operador sizeof

OPERADOR SIZEOF (TAMAÑO DE)

- Este operaror da como resultado el tamaño en bytes de un tipo de dato especificado, sea simple o compuesto.

Formato:

sizeof (expresión/variable);

- Este operador devuelve un entero sin signo.

OPERADOR SIZEOF

📌 Ejemplo:

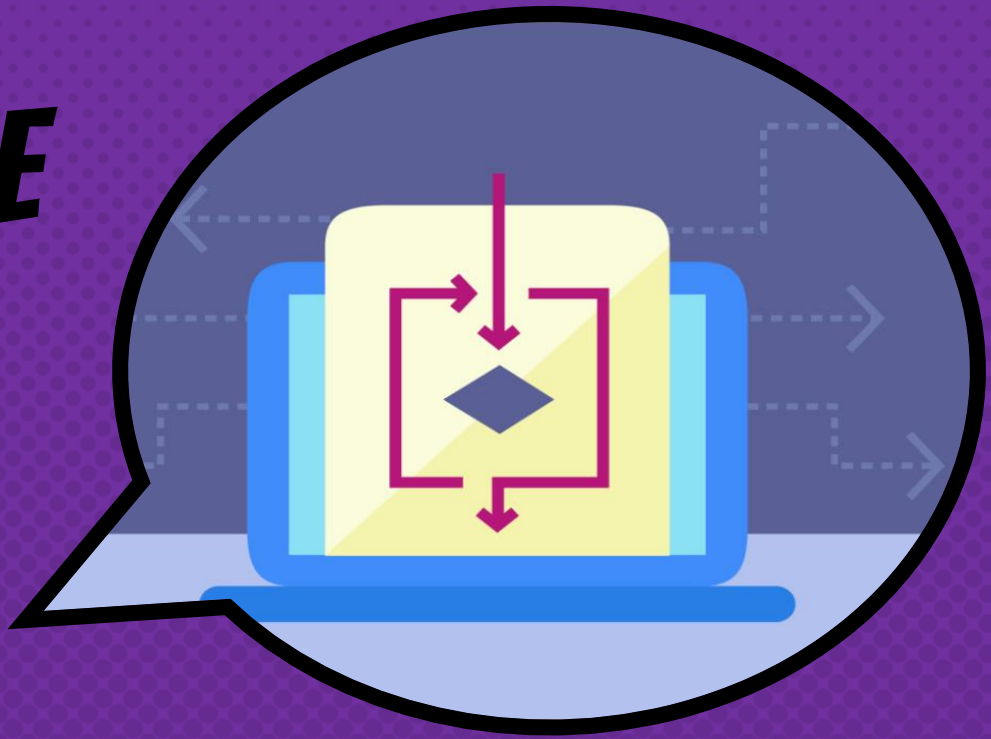
```
float tamano;
int t;

t= sizeof (tamano);
```

t almacenará 4, ya que los float ocupan 4 bytes en memoria

SENTENCIAS DE CONTROL

1. Alternativas
    1. Simple
    2. Doble
    3. Múltiples
2. Repetitivas
    1. while
    2. for



SENTENCIAS DE CONTROL

- Se definen como las instrucciones que permiten tomar decisiones basándose en el cumplimiento o no de condición o expresión lógica.

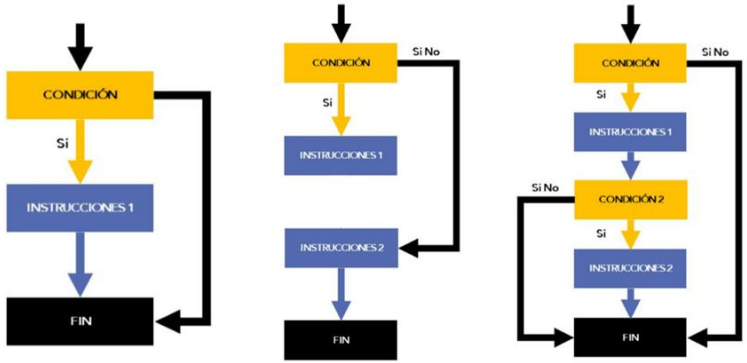
- Las sentencias de control se dividen en:

ALTERNATIVAS Y REPETITIVAS

1. Alternativas

SENTENCIAS ALTERNATIVAS

- Son instrucciones que se emplean en la toma de decisiones, ya que permiten controlar la ejecución o la no ejecución de una o más instrucciones en función de que se cumpla o no una o varias condiciones establecidas.



1. ALTERNATIVA SIMPLE (IF)

Es aquella que solo evalúa la parte cierta de una expresión lógica.

FORMATOS:

```
if (condición)
    sentencia;
```

```
if (condición) {
   sentencia1;
   sentencia2;
       ...
   sentenciaN;
}
```

* Una condición es una expresión lógica que puede dar como respuesta cierto o falso.

1. ALTERNATIVA SIMPLE (IF)

📌 EJEMPLOS:

```
if (error <= 0.009)
    cuenta ++;
```

```
if (velo > 95) {
    cuenta ++;
    printf("velo: %i",velo);
    multa = velo –limite * 30.00;
}
```

2. ALTERNATIVA DOBLE (IF-ELSE)

Es aquella que evalúa tanto la parte cierta de una expresión lógica, como la falsa.

FORMATOS:

```
if (condición)
   sentencia;
else
   sentencia;
```

```
if (condición) {
    sentencia1;
    ...
    sentenciaN;
}
else {
    sentencia1;
    ...
    sentenciaN;
}
```

2. ALTERNATIVA DOBLE (IF-ELSE)

EJEMPLOS:

```
if (error <= 0.009)
        cuenta ++;
else
        suma = suma +  pow(i,j);
```

```
if (velo >95) {
    cuenta ++;
    printf ("velo: %i",velo);
    multa =  velo –limite * 30.00;
}
else {
    normal++;
    printf ("normal: %i",normal);
}
```

3. ALTERNATIVA MÚLTIPLE (IF-ELSE IF)

Es aquella que evalúa más de una condición, pero solo ejecuta una de ellas.

FORMATOS:

```
if (condición)
    sentencia1;
else if (condición2)
    sentencia1;
else if (condición3)
    sentencia1;
else
    sentencia1;
```

```
if (condición) {
   sentencia1;
   ...
}
else if (condición2) {
   sentencia1;
   ...
}
else {
    sentencia1;
}
```

3. ALTERNATIVA MÚLTIPLE (IF-ELSE IF)

📌 EJEMPLOS:

```
if (codigo >= 10 && codigo<=100)
      printf (“Codigo Defectuoso 1”);
else if (codigo >=800 && codigo <=1000)
     printf (“ Codigo Defectuoso 2”);
else if (codigo >1010 && codigo <=1030)
     printf (“Codigo Defectuoso 3”);
else
     printf (“CODIGOS CORRECTOS”);
```

* Supongamos que el código fuese 1019, cuál sería el mensaje en pantalla?

SENTENCIA SWITCH-CASE

- La instrucción switch case es una sentencia alternativa que ejecuta diferentes bloques de instrucciones dependiendo del valor de una variable.

- Esta sentencia evalúa una expresión que puede tomar N valores distintos. Dicha expresión sólo acepta valores enteros o caracteres para entrar a la opción correspondiente.

SENTENCIA SWITCH-CASE

FORMATO:

```
switch (expresión) {
    case opc1:
        // instrucciones si la expresión= opc1
        break;
   case opcN:
        // instrucciones si expresión= opcN
        break;
   default:
        // instrucciones par defecto
        break;
}
```

SENTENCIA SWITCH-CASE

Ejemplo:

```
switch (mes) {
   case 1:
        printf("Enero");
        break;
   case 2:
        printf("Febrero");
        break;
   case ...
   default:
        printf("Error");
        break;
 }
```

2.

Repetitivas

SENTENCIAS REPETITIVAS

- Son el conjunto de instrucciones que modifican el flujo secuencial de un programa, permitiendo la ejecución iterativa o cíclica de una o varias sentencias, bajo el control de una condición.

- Existen tres tipos de estructuras repetitivas en C:

WHILE **DO WHILE** **FOR**

1. CICLO WHILE

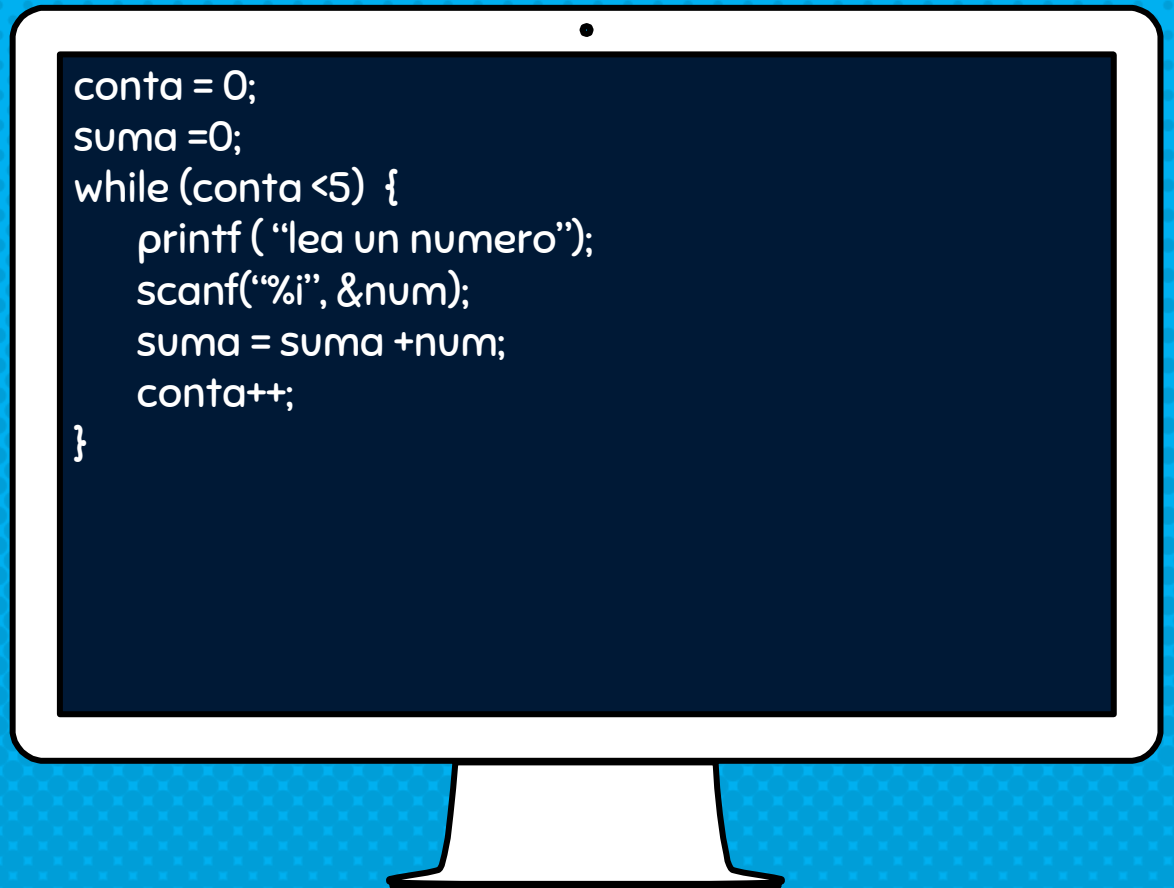
📌 Esta estructura permite ejecutar un bloque de instrucciones mientras la condición que se está evaluando sea verdadera.

FORMATO:

```
while (condición) {
    sentencia1;
    sentencia2;
        ...
    sentenciaN;
}
```

* El bloque de instrucciones puede que no se ejecute ni una sola vez.

EJEMPLO:


conta = 0;
suma =0;
while (conta <5) {
printf (“lea un numero”);
scanf(“%i”, &num);
suma = suma +num;
conta++;
}

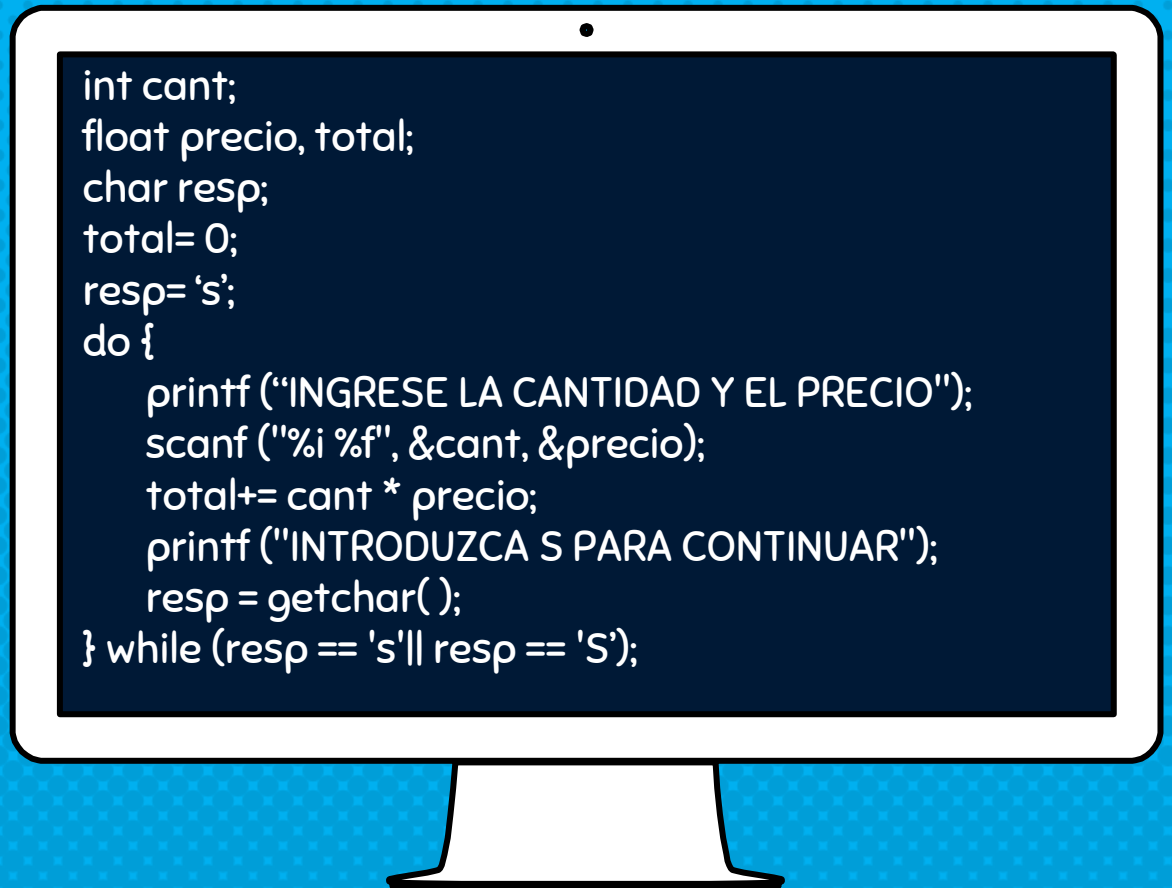
2. CICLO DO-WHILE

📌 Estructura que permite ejecutar un bloque de instrucciones mientras la condición que se está evaluando sea verdadera. En esta estructura la condición se evalúa al final por lo que el bloque de instrucciones se ejecuta al menos una vez.

FORMATO:

```
do {
    sentencia1;
    sentencia2;
        …
    sentenciaN;
 } while (condición);
```

EJEMPLO:



```
int cant;
float precio, total;
char resp;
total= 0;
resp= ‘s’;
do {
      printf (“INGRESE LA CANTIDAD Y EL PRECIO");
      scanf ("%i %f", &cant, &precio);
      total+= cant * precio;
      printf ("INTRODUZCA S PARA CONTINUAR");
      resp = getchar( );
} while (resp == 's'|| resp == 'S’);
```

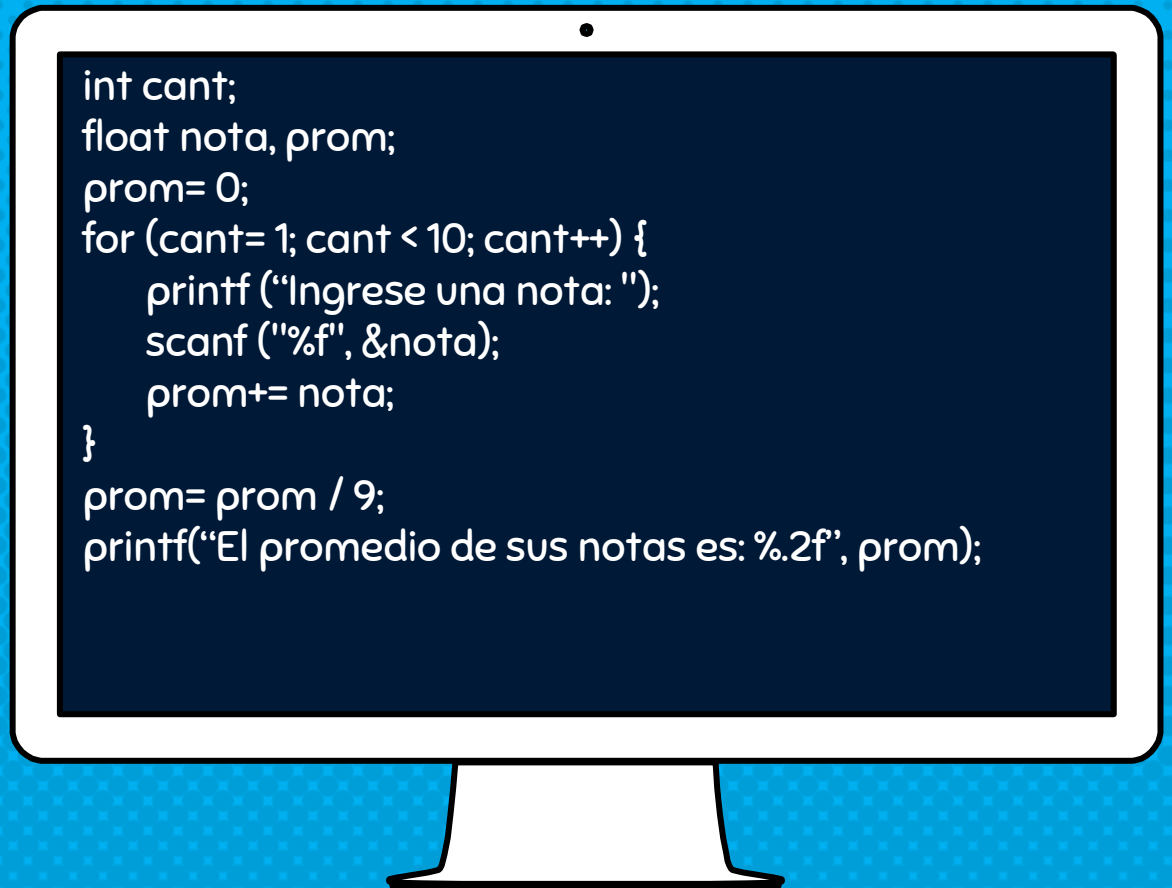
3. CICLO FOR

📌 Estructura cíclica que trabaja con iteraciones definidas, es decir, que se conoce la cantidad de repeticiones que deben realizar cierto bloque de instrucciones.

FORMATO:

```
for (varControl= inicialización; condición; incremento/decremento) {
      sentencia1;
      sentencia2;
          ...
      sentenciaN;
 }
```

EJEMPLO:



```
int cant;
float nota, prom;
prom= 0;
for (cant= 1; cant < 10; cant++) {
      printf (“Ingrese una nota: ");
      scanf ("%f", &nota);
      prom+= nota;
}
prom= prom / 9;
printf(“El promedio de sus notas es: %.2f”, prom);
```

ANIDAMIENTO

- Es el concepto empleado cuando una estructura puede estar dentro de otra estructura.
- También se le conoce como estructuras anidadas.
- Al igual que las instrucciones alternativas, las instrucciones repetitivas también se pueden anidar, es decir que se puede tener una sentencia repetitiva dentro del bloque de instrucciones de otra sentencia repetitiva.

OPERADOR COMA

- La coma puede ser actuar como operador separado de expresiones "de coma".

- En estas expresiones cada operando es evaluado como una expresión, pero los resultados obtenidos anteriormente se tienen en cuenta en las subsiguientes evaluaciones.

EJEMPLO:

```
z= (x= 5, y= x + 3,  x * y + 15);
```

*La expresión de coma se evalúa de izquierda a derecha.

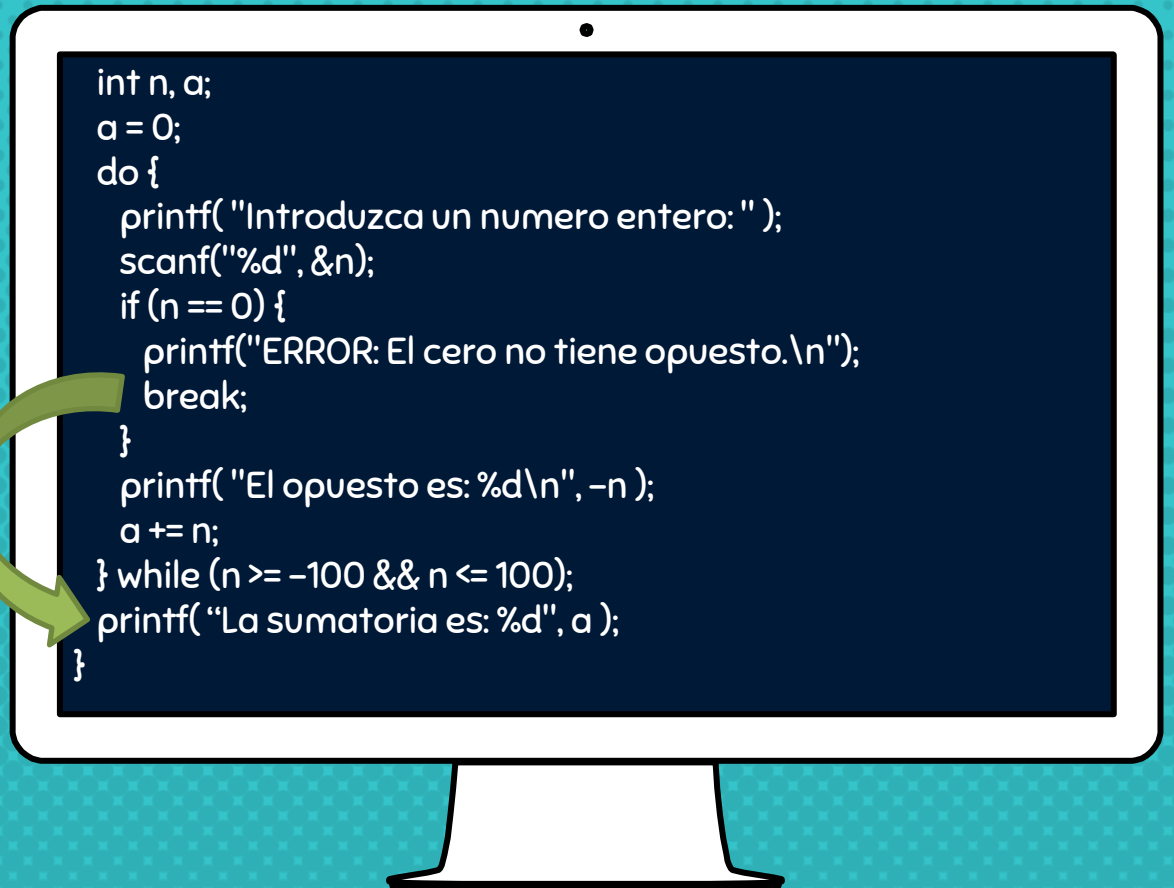
BREAK

📌La instrucción de salto **break** se usa para interrumpir (romper) la ejecución normal de un bucle, es decir, la instrucción break finaliza (termina) la ejecución de un bucle y, por tanto, el control del programa se transfiere (salta) a la primera instrucción después del bucle.

FORMATO:

```
break;
```

EJEMPLO:



```
int n, a;
a = 0;
do {
  printf( "Introduzca un numero entero: " );
  scanf("%d", &n);
  if (n == 0) {
    printf("ERROR: El cero no tiene opuesto.\n");
    break;
  }
  printf( "El opuesto es: %d\n", -n );
  a += n;
} while (n >= -100 && n <= 100);
printf( "La sumatoria es: %d", a );
}
```

CONTINUE

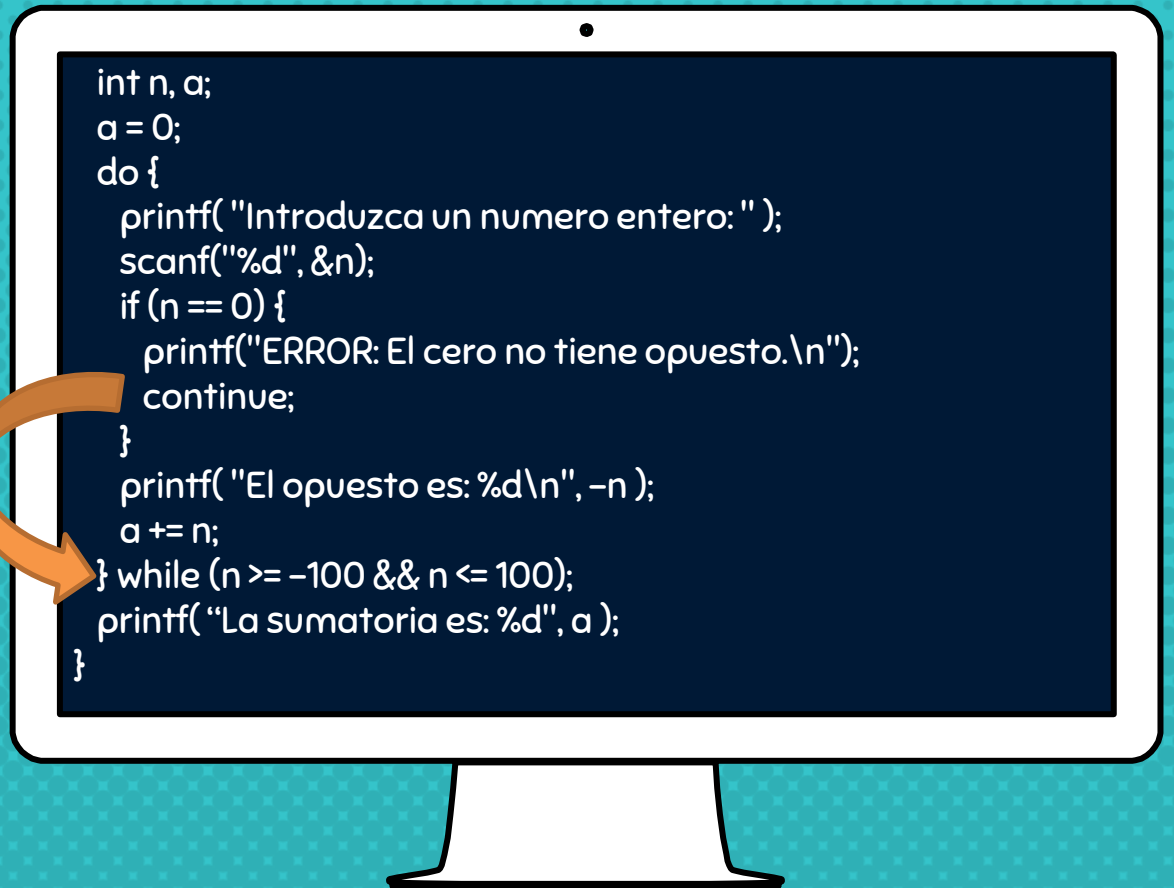
📌 La instrucción de salto **continue** siempre se usa para interrumpir la ejecución normal de un bucle.

La instrucción continue finaliza (termina) la ejecución de una iteración de un bucle, pero, no la ejecución del bucle en sí.

FORMATO:

```
continue;
```

EJEMPLO:



```
int n, a;
a = 0;
do {
  printf( "Introduzca un numero entero: " );
  scanf("%d", &n);
  if (n == 0) {
    printf("ERROR: El cero no tiene opuesto.\n");
    continue;
  }
  printf( "El opuesto es: %d\n", -n );
  a += n;
} while (n >= -100 && n <= 100);
printf( "La sumatoria es: %d", a );
}
```

Preguntas?