



Universidad Tecnológica Nacional
Facultad Regional Resistencia
Técnico Universitario en Programación

Lenguaje De Programación C

*Entrada
y
Salida
de
Datos*



Funciones de Entrada y Salida

- **getchar**
- **putchar**
- **scanf**
- **printf**
- **gets**
- **puts.**
- Estas seis funciones permiten la transferencia de información entre la computadora y los dispositivos de entrada/ salida estándar (por ejemplo, un teclado y un monitor).
- **getchar** y **putchar**, permiten la transferencia de caracteres individuales hacia dentro y hacia fuera de la computadora;
- **gets** y **puts** permiten la entrada y salida de cadenas de caracteres.
- **scanf** y **printf** son más complicadas, pero permiten la transferencia de caracteres individuales, valores numéricos y cadenas de caracteres;

- **getchar** y **putchar**, permiten la transferencia de caracteres individuales hacia dentro y hacia fuera de la computadora;

```
#include <stdio.h> /* necesario para utilizar printf()*/
#include <conio.h> /* necesario para utilizar getch()*/
```

```
int main()
{
    char c;
    c=getchar();
    putchar (c);

    getch();
    return 0;
}
```

```
#include <stdio.h> /* necesario para utilizar printf()*/
#include <conio.h> /* necesario para utilizar getch()*/

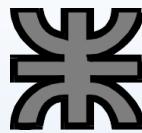
int main()
{
    char c;
    printf("Ingrese un caracter por teclado");
    printf("\n");
    c=getchar();
    printf("El caracter ingresado es:");
    putchar (c);
    return 0;
}
```

- **gets** y **puts** permiten la entrada y salida de cadenas de caracteres.

```
#include <stdio.h> /* necesario para utilizar printf()*/
#include <conio.h> /* necesario para utilizar getch()*/

int main()
{
    char c[10];
    printf("Ingrese su nombre por teclado");
    printf("\n");
    gets(c);
    printf("El nombre ingresado es:");
    puts(c);

    return 0;
}
```



Función printf()

Es la función de salida de datos con formato .

La sintaxis es :

printf (texto,cadena de control, lista de argumentos);

donde texto y cadena de control son opcionales, dependiendo de lo que se desee mostrar. Cadena de control es una cadena de caracteres “%tipo” que indica el tipo de dato a desplegar (lo requiere la función *printf()*). Por otro lado, argumento o argumentos es el valor o los valores que se pretende mostrar, y pueden ser variables, constantes, expresiones aritméticas, resultados de funciones o simplemente texto que el programa debe mostrar al usuario. A continuación se muestra un ejemplo del uso y la explicación.

```
int a;  
a= 2;  
...  
printf ("el numero es :%d", a);
```



Función printf() – Cadena de Control

Cadena de tipo	Descripción
%d	<i>El dato es un entero decimal (int).</i>
%i	<i>El dato es un entero.</i>
%o	<i>El dato es un entero octal.</i>
%x	<i>El dato es un entero hexadecimal.</i>
%u	<i>El dato es un entero sin signo en decimal (unsigned int).</i>
%c	<i>El dato es un carácter (char).</i>
%e	<i>El dato es un real expresado en base y exponente (float).</i>
%f	<i>El dato es un real escrito con punto decimal con signo (float).</i>
%g	<i>El dato es un real (float).</i>
%s	<i>El dato es una cadena de caracteres que finaliza con el carácter nulo \0.</i>
%lf	<i>El dato es real de tipo long double.</i>



Función printf()-Ejemplos

```
int a=7;  
float b=8.2;  
char c='s';
```



```
printf("%d", a);
```

Se visualiza un 7, que es el contenido de la variable a.

```
printf("%d", a+b);
```

Se visualiza un 15, ya que es la suma de a + b mostrada como valor entero.

```
printf("%f", a+b);
```

Se visualiza un 15.2, ya que es la suma de a + b mostrada como valor real.

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

Se visualiza la letra 's' que es el contenido de la variable c.

```
printf("%c %d %f", c, a,b);
```

Se visualiza s 7 8.2 que son los valores de las variables c, a y b respectivamente.

```
printf("HOLA");
```

Se visualiza la palabra HOLA.

```
#include <stdio.h> /* necesario para utilizar printf()*/
#include <conio.h> /* necesario para utilizar getch()*/

int main()
{
    char car;
    printf("Ingrese un caracter por teclado");
    printf("\n");
    car=getchar();
    printf("El caracter ingresado es:");
    putchar (car);
    printf("\n");
    printf("El caracter ingresado es %c:",car);
    return 0;
}
```

```
#include <stdio.h> /* necesario para utilizar printf()*/
#include <conio.h> /* necesario para utilizar getch()*/

int main()
{
    char car[10];
    printf("Ingrese su nombre por teclado");
    printf("\n");
    gets(car);
    printf("El nombre ingresado es:");
    puts(car);
    printf("\n");
    printf("El nombre ingresado es %s:",car);
    return 0;
}
```



Secuencia de Escape

Las secuencias de escape son cadenas de caracteres que tienen distintos significados dependiendo de la cadena que se utilice.

La forma más sencilla de escribir una secuencia de escape es con el carácter *barra invertida* (\), seguido de un carácter especial. Por tanto, cuando en la cadena de control de la función printf se escriba una secuencia de escape, o bien se mostrará un carácter gráfico por pantalla, o bien se realizará una acción.



Secuencia de Escape

<i>Secuencia de escape</i>	<i>Descripción</i>
\a	<i>Alarma</i>
\b	<i>Retroceso</i>
\f	<i>Avance de página</i>
\n	<i>Retorno de carro y avance de línea</i>
\r	<i>Retorno de carro</i>
\t	<i>Tabulación</i>
\v	<i>Tabulación vertical</i>
\	<i>Diagonal invertida</i>
\?	<i>Signo de interrogación</i>
\"	<i>Comillas dobles</i>
\000	<i>Octal</i>
\xhh	<i>Hexadecimal</i>
\0	<i>Carácter nulo</i>



- Permite escribir datos en el dispositivo de salida estándar utilizando la función de biblioteca printf.
- Permite mostrar una cadena con formato y muestra la misma por la pantalla.
- La función printf del ejemplo muestra el mensaje "Mi primer programa" aparece el símbolo '\n'; este hace que después de mostrar el mensaje se pase a la línea siguiente, corresponde a un carácter Ascii no imprimible.
- La cadena con formato provee una descripción de la salida con el uso de un atributo marcador de posición específico que describe el valor esperado de un campo de entrada usando caracteres de escape "%" para especificar la posición relativa y el tipo de salida que la función debe producir.

```
printf("Color %s, numero1 %d, numero2 %05d, hex %x, real %5.2f.\n", "rojo", 12345, 89, 255, 3.14);
```

imprimirá la siguiente línea (incluyendo el carácter de nueva línea \n):

```
Color rojo, numero1 12345, numero2 00089, hex ff, real 3.14.
```



Secuencia de Escape- Ejemplo

```
int a=7;  
float b=8.2;  
char c='s';
```

```
printf("%d \n \t %f \n \t\t %c", a,b,c);
```

7

8.2

s

Formateadores: Permite dar formato específico a la salida.



Formateador	Salida
%d ó %i	entero en base 10 con signo (int)
%u	entero en base 10 sin signo (int)
%o	entero en base 8 sin signo (int)
%x	entero en base 16, letras en minúscula (int)
%X	entero en base 16, letras en mayúscula (int)
%f	Coma flotante decimal de precisión simple (float)
%lf	Coma flotante decimal de precisión doble (double)
%e	La notación científica (mantisa / exponente), minúsculas (decimal precisión simple ó doble)
%E	La notación científica (mantisa / exponente), mayúsculas (decimal precisión simple ó doble)
%c	caracter (char)
%s	cadena de caracteres (string)

Ejemplo del uso de Formateadores:

%.7i	largo mínimo de 7 dígitos, justificado a la derecha, rellena con ceros
%8.2f	tamaño total de 8 dígitos, con dos decimales



Ejemplo de Funciones de Entrada y Salida

```
#include <stdio.h> int main ()  
{  
    char c; float x, y; int i;  
  
    printf("Ingrese un carácter:");  
    c = getchar(); /* entrada de un carácter */  
    printf("Ingrese un valor flotante:");  
    scanf("%f\n",&x);/* entrada de número en coma flotante */  
    printf("Ingrese un valor entero:");  
    scanf("%d\n",&i);/* entrada de enteros */  
    printf("Mostrar los valores ingresados:");  
    putchar(c);/* salida de un carácter */  
    printf("%3d %7.4f", i, x); /* salida de números*/  
}
```



Función scanf()

Es la función de entrada de datos con formato .

La sintaxis es :

scanf (cadena de control, &variable);

El operador & es necesario en scanf() para simular las *llamadas por referencia*, y hace que la función trabaje internamente con la dirección de la variable.

```
int edad;
```

```
printf("Teclea tu edad");
```

```
scanf("%d", &edad);
```



Ejemplo de Funciones de Entrada y Salida

```
#include <stdio.h>

int main ()
{
    float x; int k;

    printf("Ingrese un valor flotante:");
    scanf("%f",&x);/* entrada de número en coma flotante */

    printf("Ingrese un valor entero:");
    scanf("%d\n",&k);/* entrada de enteros */
    printf("Mostrar los valores ingresados:");
    printf("Valor flotante: %f" y valor entero:%d", x,k);
}
```

```
#include <stdio.h> /* necesario para utilizar printf()*/
#include <conio.h> /* necesario para utilizar getch()*/

int main()
{
    char p;
    printf("Ingresar un caracter por teclado");
    p=getchar();
    printf ("El valor ingresado por teclado es:"), 
    putchar (p);
    printf("Ingresar un caracter por teclado\n");
    scanf (" %c",&p);
    printf ("El valor ingresado por teclado es:"), 
    putchar (p);
    return 0;
}
```

```
#include <stdio.h> /* necesario para utilizar printf()*/
#include <conio.h> /* necesario para utilizar getch()*/\n\nint main()\n{\nchar p;\nprintf("Ingresar un caracter por teclado");\np=getchar();\nprintf ("El valor ingresado por teclado es:"),\nputchar (p);\nprintf("Ingresar un caracter por teclado\n");\nscanf ("%c",&p);\nprintf ("El valor ingresado por teclado es %c",p);\n\nreturn 0;\n}
```

EJERCICIOS

Ejercicio: Encontrar los errores y corregir

```
#include <stdio.h> /* necesario para utilizar printf()*/
```

```
int main()
{
char p;
p=gets();
printf ("El valor ingresado por teclado es:");
putchar (p);

return 0;
}
```

CORRECCIÓN

```
#include <stdio.h> /* necesario para utilizar printf()*/
#include <conio.h> /* necesario para utilizar getch()*/
```

```
int main()
{
char p;
p=getchar();
printf ("El valor ingresado por teclado es:")
putchar (p);

return 0;
}
```

Ejercicio:

Completar los siguientes trozos de programa e indicar si se obtiene el mismo resultado.

Expresion 1	Expresion 1
<pre>int a,b,c; a= 3; b=4; c=a-b; printf("El resultado de la resta es %d",c);</pre>	<pre>int a,b,c; a= 3; b=4; printf("El resultado de la resta es %d",a-b);</pre>

Ejercicio: Se pide:

- Corregir los errores del siguiente programa y
- documentarlo usando comentarios
- cada mensaje que se muestra por pantalla debe aparecer uno debajo del otro.

```
#include stdio.h
```

```
int main()
{
int a,b,c;
printf("Ingrese los valores para realizar la operación resta");
printf ("Ingrese el primer valor");
scanf("%f",&a);
printf ("Ingrese el segundo valor");
scanf("%f",&b);
c=a-b;
printf("El resultado de la resta es %d",c);
return 0;
}
```



Universidad Tecnológica Nacional
Facultad Regional Resistencia
Técnico Universitario en Programación

Programación I

Sentencia IF

Programación Estructurada

Esta forma de programar (paradigma) se basa en un famoso teorema, desarrollado por Edsger Dijkstra, que demuestra que todo programa puede escribirse utilizando únicamente las tres estructuras básicas de control:

- Secuencia: el bloque secuencial de instrucciones, ejecutadas sucesivamente, una detrás de otra.
- Selección: la instrucción condicional con doble alternativa, de la forma “*if condición then instrucción-1 else instrucción 2*”.
- Iteración: el bucle condicional “*while condición do instrucción*”, que ejecuta la instrucción repetidamente mientras la condición se cumpla.

Sentencias de Control

Las sentencias de control permiten controlar el flujo del programa, tomando decisiones a partir de comparaciones.

- Se usan instrucciones condicionales y de ciclos.
- Un **condicional** es un conjunto de sentencias que pueden o no ejecutarse, dependiendo del resultado de una condición.
- Un **ciclo** es un conjunto de sentencias que son ejecutadas varias veces, hasta que una condición de término es satisfecha.
- Tanto los condicionales como los ciclos contienen a otras sentencias. Para indicar esta relación ,las sentencias contenidas no se escriben en la misma columna que la sentencia de control, sino un poco más a la derecha

Sentencias de Control

➤ Las instrucciones **condicionales** son:

- IF
- SWITCH

➤ Las instrucciones de **ciclo** son:

- WHILE
- FOR
- REPEAT

Operadores Relacionales

Se usan para expresar condiciones y describir una relación entre dos valores

```
if (a == b) printf ("Son iguales");
```

	OPERADOR	DESCRIPCIÓN
BINARIOS	>	Mayor que
	>=	Mayor o igual que
	<	Menor que
	<=	Menor o igual que
	==	Igual que
	!=	Diferente que

Sentencia IF



Restricciones: Uso de las estructuras de selección if-then-else.

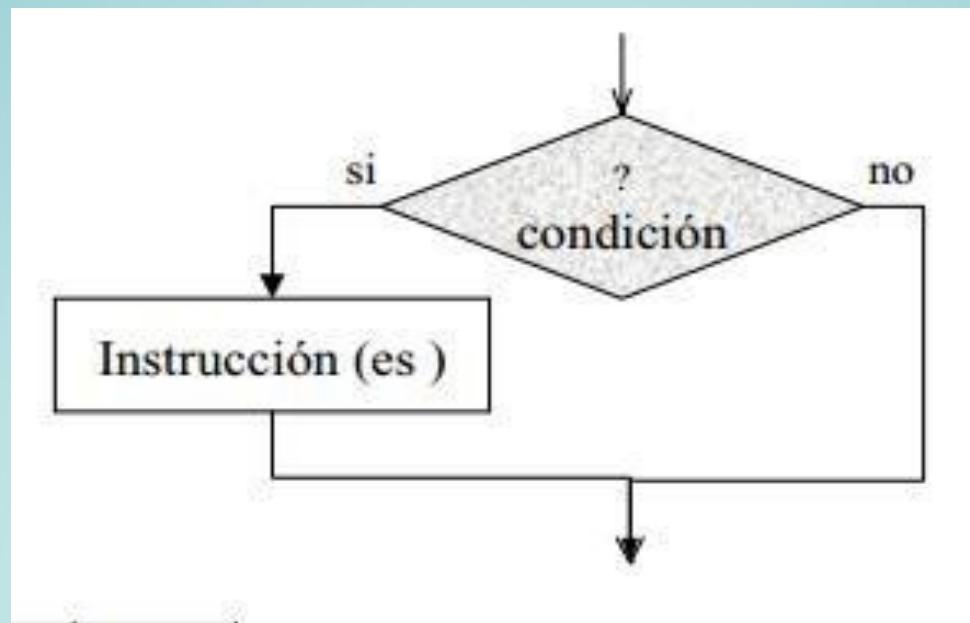
Algoritmo	Diagrama de Flujo o Pseudocódigo	Código en C
<p>Datos de Entrada: Número de tipo real</p> <p>Datos de Salida: Mensaje de que está dentro de los límites, mensaje de que está fuera de rango o no alcanza</p> <p>Algoritmo:</p> <p>Inicio</p> <p> Limite_Inferior = 100 Limite_Superior = 200 Solicitar número al usuario. Almaceno en mi variable Número Si Número es mayor o igual que Limite_Inferior entonces Si Número es menor o igual que Limite_superior entonces Imprimo en pantalla que está dentro de los límites Sino Imprimo en pantalla que supera al límite máximo Sino Imprimo en pantalla que no alcanza el límite mínimo</p> <p>fin</p>	<pre>graph TD; Start(()) --> Init[Limite_Inferior = 100; Limite_Superior = 200; Numero = 0]; Init --> Input[Proporcione un dato entero]; Input --> Read[/Lea Número/]; Read --> Cond1{Número >= Limite_Inferior}; Cond1 -- NO --> Out1[No alcanza el limite inferior]; Out1 --> Read; Cond1 -- SI --> Cond2{Número <= Limite_Superior}; Cond2 -- NO --> Out2[Supera el limite superior]; Out2 --> Read; Cond2 -- SI --> Out3[Está dentro de los limites]; Out3 --> Read;</pre>	#include <stdio.h> #include <conio.h> #define Limite_Inferior 100 #define Limite_Superior 200 int main() { float Numero=0; //Definimos nuestra variable printf("---Problema 1----\n"); printf("Introduzca un número: "); scanf("%f", &Numero); if (Numero >= Limite_Inferior) { if (Numero <= Limite_Superior) { printf("Está dentro del intervalo"); } else printf("Supera el límite máximo "); } else printf("No alcanza el límite mínimo"); return 0; }

Sentencia IF simple en C

La estructura if adopta una de las dos formas siguientes:

if (condición) sentencia;

en donde condición es una sentencia que se evalúa como verdadera



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

```
int main()
{
    int num;
```

```
    printf ("Ingrese un número entero por teclado: ");
```

```
    scanf ("%d", &num);
```

```
// ejemplo de IF
```

```
    if (num > 0) printf (" %d es POSITIVO",num);
```

```
    getch();
```

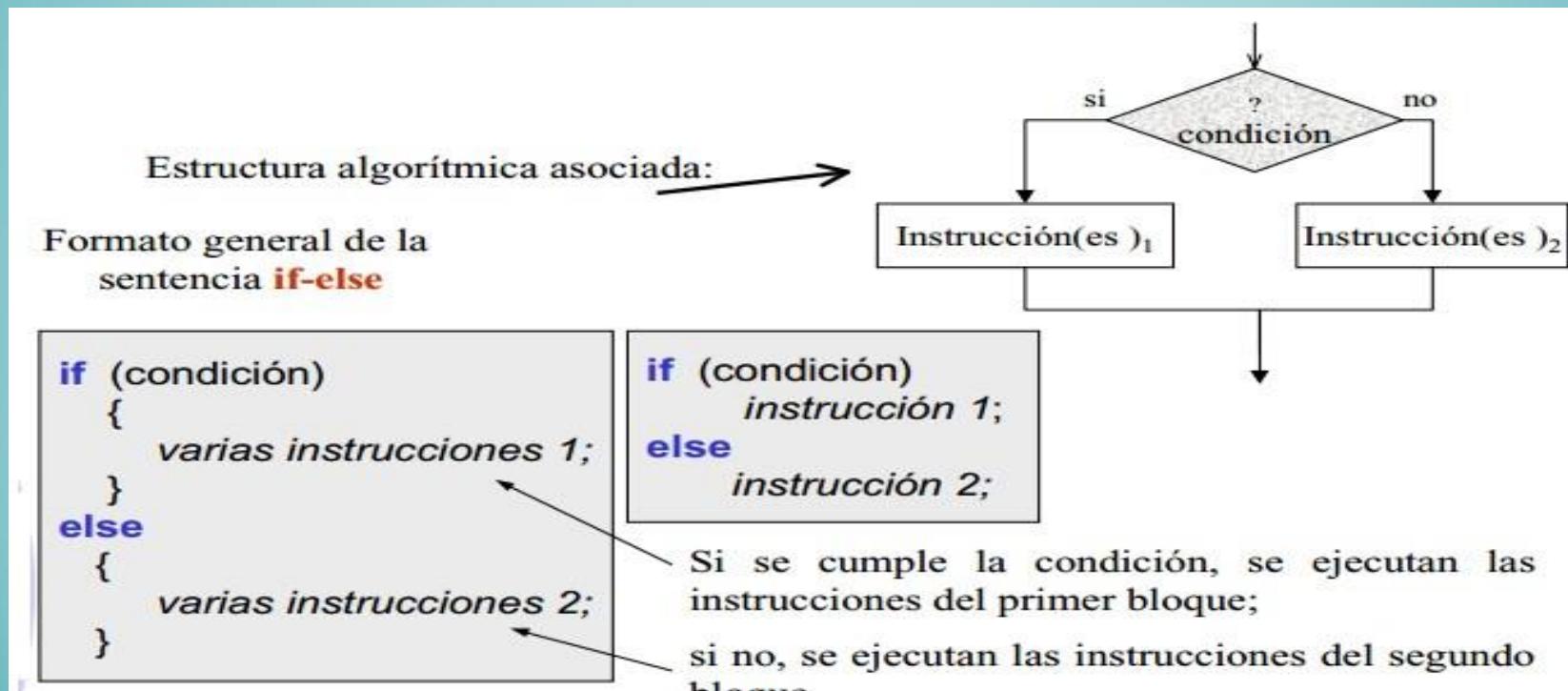
```
    return 0;
```

EJEMPLO IF SIMPLE

Sentencia IF doble en C

```
if (condición) sentencia1;  
else sentencia2;
```

en donde *expresión* es una sentencia que se evalúa como verdadera (devuelve un valor no nulo) o falsa (devuelve cero). La palabra *sentencia* puede ser una sentencia simple terminada con un punto y coma, o un grupo de sentencias encerradas entre llaves {}.



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

int main()
{
    int num;

    printf ("Ingrese un número entero por teclado: ");

    scanf ("%d", &num);

    // ejemplo de IF

    if (num > 0) printf (" %d es POSITIVO",num);
        else printf ("%d es negativo o igual a cero\n",num);

    getch();

    return 0;
}
```

EJEMPLO IF CON ELSE



(Zinjal is not just another IDE)

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

int main ()
{
    int n;

    printf ("Teclee un número entero: ");

    scanf ("%d", &n);
    // ejemplo de IF anidado simple

    if (n % 3== 0) printf (" %d es divisible por 3 \n",n);
    else printf ("%d No es múltiplo de 3 \n",n);

    getch();

    return 0;
}
```

EJEMPLO IF CON ELSE



(Zinjal is not just another IDE)

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

int main ()
{
    char car;

    printf ("Teclee un caractero:\n ");

    scanf ("%c", &car);
    // ejemplo de IF anidado

    if (car == 'a') printf (" Se ingreso la letra a \n");
    else printf ("Se ingreso el carácter %c \n",car);

    getch();
    return 0;
}
```

EJEMPLO IF CON ELSE

Comilla simple: alt+39

Ejercicio:

Verificar si el siguiente código cumple con el enunciado. Si no cumple con el enunciado realizar las modificaciones que considere necesarias.

Enunciado: El programa lee un numero entero y lo transforma en el impar inmediatamente mayor, si es que no era ya impar.

```
#include <stdio.h>

int main ()
{
    int a;
    scanf("%d", &a);

    if (a % 2 == 0) /* Comprobar si a es par. */
        a = a + 1;
    printf( "Ahora es impar: %d n", a );

    return 0;
}
```

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

int main ()
{
    int num,m ;
    m=0;
    printf ("Teclee un numero: ");
    scanf ("%d", &num);

    if (num <= 10)
    {
        m= 2*num;
        printf (" El duplo de %d es %d", num,m);
    }
    else
    {
        m=3*num;
        printf ("El triplo de %d es %d \n", num,m);
    }
    getch();
    return 0;
}
```

EJEMPLO IF CON MÁS DE UNA ACCIÓN POR CONDICIÓN

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

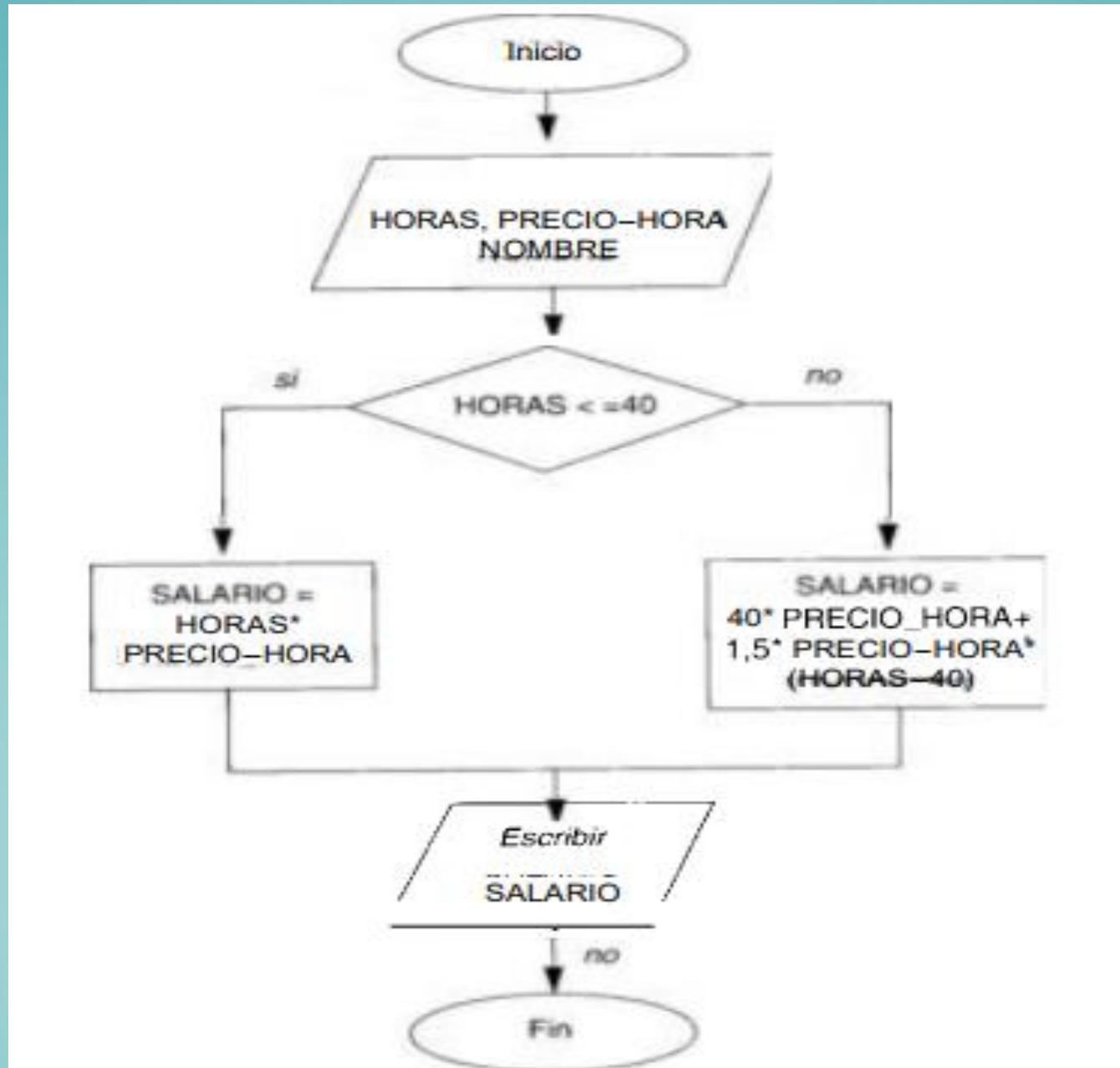
int main ()
{
    int num,m ;
    m=0;
    printf ("Teclee un numero: ");
    scanf ("%d", &num);

    if ( num >= 0 && num <= 10)
    {
        m= 2*num;
        printf (" El duplo de %d es %d", num,m);
    }
    else
        printf ("No realizar ninguna operacion \n");

    getch();
    return 0;
}
```

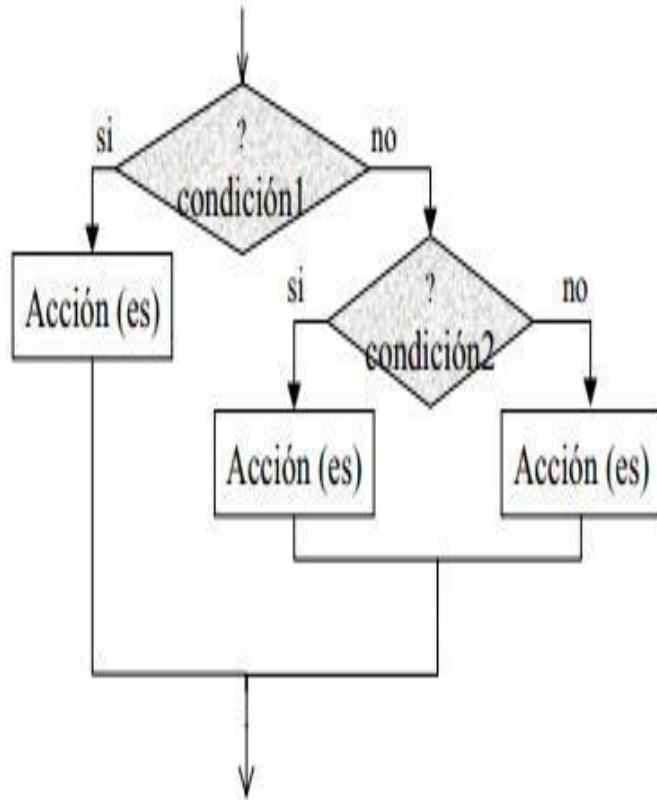
EJEMPLO IF CON MÁS DE UNA CONDICIÓN COMBINADA

Ejercicio: Escribir el programa asociado al siguiente diagrama de flujo



Sentencia IF anidado en C

- Es posible utilizar las instrucciones IF-ELSE anidadas, es decir, que alguna de las ramas sea a su vez otra instrucción IF-ELSE.
- Permite implementar decisiones que implican más de dos alternativas.



La sintaxis de
instrucciones IF-ELSE anidadas

```
if (condición1)  
    instrucción 1;  
else  
    if (condición2)  
        instrucción 2;  
    else  
        if (condición3)  
            instrucción 3;  
        else  
            instrucción 4;
```

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

int main()
{
    int num;

    printf ("Ingrese un número entero por teclado: ");

    scanf ("%d", &num);

    // ejemplo de IF

    if (num > 0) printf (" %d es POSITIVO",num);
    else
        if (num == 0) printf ("Es igual a cero\n");
        else printf ("%d es negativo \n",num);
    getch();

    return 0;
}
```

EJEMPLO IF ANIDADO

¿Qué dificultad presenta el programa con respecto al uso de IF?

```
#include <stdio.h>
#define TARIFA1 1.2
#define TARIFA2 1.0
#define TARIFA3 0.9

int main()
{
    float gasto, tasa;
    printf("\n Gasto de corriente: ");
    scanf("%f",&gasto);
    if (gasto< 1000.0)
        tasa = TARIFA1;
    if (gasto>=1000.0 && gasto <=1850.0)
        tasa = TARIFA2;
    if (gasto>1850.0)
        tasa = TARIFA3;

    printf("\nTasa que le corresponde a %.1f Kwxh es de %f\n",
           gasto,tasa);
    return 0;
}
```

Operadores Lógicos

Actúan sobre expresiones booleanas, es decir, sobre valores *verdadero* o *falso* generados por expresiones como las explicadas en el caso anterior.

	<i>OPERADOR</i>	<i>DESCRIPCIÓN</i>
UNARIOS	!	<i>not</i>
BINARIOS	&&	<i>and</i>
		<i>or</i>

Tabla de verdad del operador lógico NOT (!).

Operando (a)	NOT a
Verdadero (1)	Falso (0)
Falso (0)	Verdadero (1)

Tabla de verdad del operador lógico AND.

Operandos

a	b	a && b
Verdadero (1)	Verdadero (1)	Verdadero (1)
Verdadero (1)	Falso (0)	Falso (0)
Falso (0)	Verdadero (1)	Falso (0)
Falso (0)	Falso (0)	Falso (0)

Tabla de verdad del operador lógico OR (||).

Operandos

a	b	a b
Verdadero (1)	Verdadero (1)	Verdadero (1)
Verdadero (1)	Falso (0)	Verdadero (1)
Falso (0)	Verdadero (1)	Verdadero (1)
Falso (0)	Falso (0)	Falso (0)

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

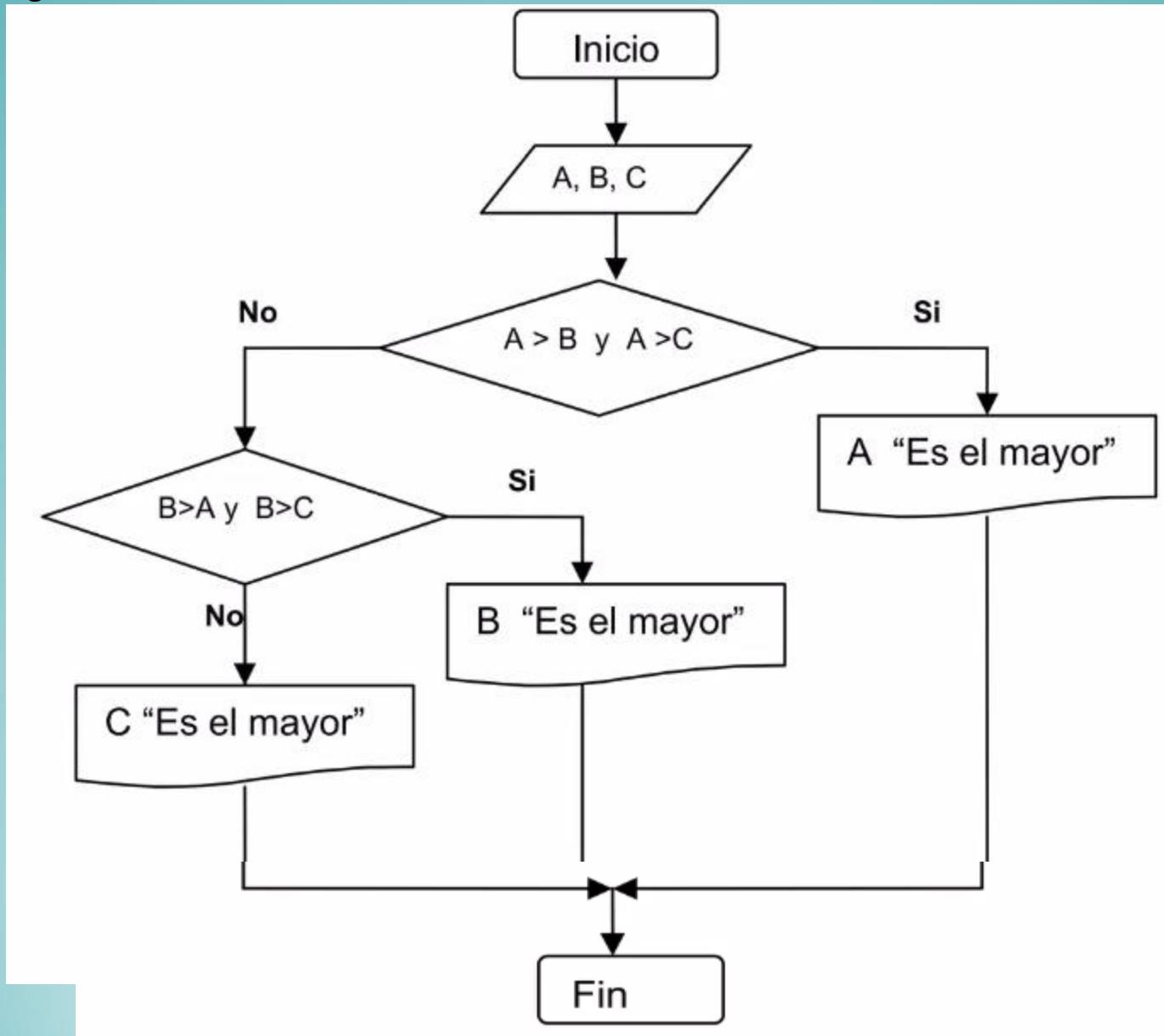
int main()
{
    int hora;
    scanf( "%d", &hora );

    if ((hora >= 0) && (hora < 12)) printf( "Buenos días" );
    else
        if ((hora >= 12) && (hora < 18)) printf( "Buenas tardes" );
        else
            if ((hora >= 18) && (hora < 24)) printf( "Buenas noches" );
            else printf( "Hora no válida" );
    getch();

    return 0;
}
```

Ejercicio:

Escribir un programa que lea tres valores enteros y muestre por pantalla el máximo de ellos a partir del siguiente algoritmo,



¿Es posible usar en este programa los operadores lógicos ?

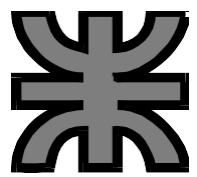
```
#include <stdio.h>
int main()
{
    int a, b, c, mayor;
    printf("\nIntroduzca tres enteros:");
    scanf("%d %d %d", &a, &b, &c);
    if (a > b)
        if (a > c) mayor = a;
        else mayor = c;
    else
        if (b > c) mayor = b;
        else mayor = c;
    printf("El mayor es %d \n", mayor);
    return 0;
}
```

Ejercicio: Calcular las raíces de una ecuación cuadrática.

Universidad Tecnológica Nacional
Facultad Regional Resistencia
Técnico Superior en Programación

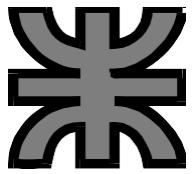
PROGRAMACIÓN I

Sentencia
While



Ejemplo: Ingresar tres números y mostrarlos por pantalla

```
#include <stdio.h>
int main()
{
    int num;
    printf ("Ingrese una variable");
    scanf ("%d",&a);
    printf (" Numero ingresado: %d ",a);
    printf ("Ingrese una variable");
    scanf ("%d",&b);
    printf (" Numero ingresado: %d ",b);
    printf ("Ingrese una variable");
    scanf ("%d",&c);
    printf (" Numero ingresado: %d ",c);
    return 0;
}
```



Para el ejercicio planteado, se puede observar que por cada vez que se ingresa una variable se repiten las instrucciones `printf` y `scanf`. Así por ejemplo si se quisiera realizar la misma operación para 10 números, siguiendo este método de programación se debería escribir 30 líneas de código lo cual es ineficiente.

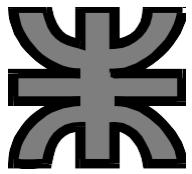
Para solución este problema se utilizan las **ESTRUCTURAS CÍCLICAS O REPETITIVAS**.

Las tres construcciones para realizar bucles son:

WHILE

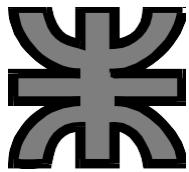
DO – WHILE

FOR



Estructuradas Repetitivas o cíclicas

La estructura repetitiva se utiliza cuando se quiere que un conjunto de instrucciones se ejecuten un cierto número finito de veces. Se le llama bucle o ciclo a todo proceso que se repite un cierto número de veces dentro de un programa.



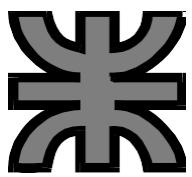
Estructuradas Repetitivas o cíclicas

Existen dos tipos de estructuras repetitivas;

➤ la primera es aquella en donde se tiene perfectamente establecido el número de veces que un grupo de acciones se van a ejecutar (20, 5, 2 veces).

En este caso se utiliza un contador. En este caso el contador se puede incrementar como decrementar.

➤ y la segunda en la que el número de repeticiones es desconocido y se hará hasta que se cumpla o no cierta condición. Por ejemplo que un número sea mayor que cero.



Acciones Simples

- Contador: Es una variable que se incrementa, cuando se ejecuta, en una unidad o en una cantidad constante.

Ejemplo: **contador := contador + 1**
multiplo := multiplo + 3

- Acumulador: Es una variable que se incrementa en una cantidad variable.

Ejemplo: **suma := suma + numero**

Sentencia WHILE - (Mientras)

While primero evalúa la condición y si se cumple entra en el ciclo hasta que la condición no se cumpla.

Sintaxis

1 **while** (*condición-bucle*)
 Sentencia; → *cuerpo*

2 **while** (*condición-bucle*)

{

sentencia-1;
 sentencia-2;

 ⋮

sentencia-n;

}

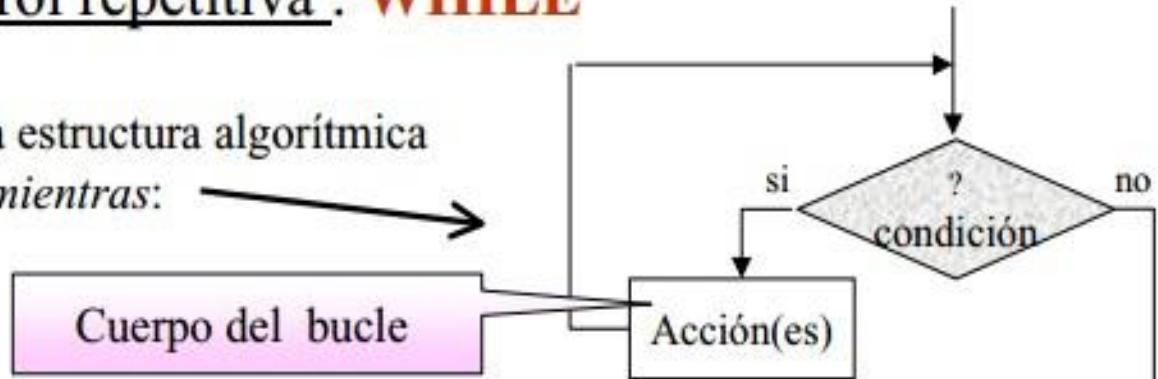
}

cuerpo

Estructuradas Repetitivas o cíclicas

Instrucción de control repetitiva : WHILE

Se corresponde con la estructura algorítmica
hacer_mientras:



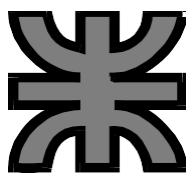
Formato general de la sentencia **while**

```
while ( condición )
{
    instrucción 1;
    ...
    instrucción n;
}
```

Controla la ejecución de un bloque de **instrucciones** de tal forma que éstas **se ejecutan mientras se cumpla la condición**, que será evaluada siempre antes de cada repetición.

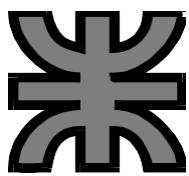
```
while ( condición )
    instrucción;
```

Cada repetición del cuerpo del bucle se denomina **iteración**



Funcionamiento Sentencia WHILE - (Mientras)

1. Evalúa la condición
2. Si la condición es verdadera ejecuta el bloque de sentencias y vuelve a paso 1.
3. En caso contrario pasa a ejecutar la sentencia que se encuentra después del cuerpo.



Funcionamiento Sentencia WHILE - (Mientras)

Instrucción de control repetitiva : WHILE

Inicialización

Se realiza antes de la instrucción **while**

```
...  
contador = 0;  
while (contador < 100)  
{  
    cout << "Hola";  
    contador++;  
}  
...
```

cuerpo

Actualización

Se realiza dentro del cuerpo del bucle
durante cada iteración

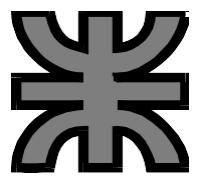
La variable que representa la condición del bucle se denomina **variable de control del bucle**.

Dicha variable debe ser:

- inicializada
- comprobada
- actualizada

Comprobación

Se comprueba el valor de la variable
antes de comenzar la repetición



Ejemplo: Ingresar tres números y mostrarlos por pantalla usando WHILE

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

```
int main()
{
    int cont,acum,a;
    cont=0; /* inicializa contador*/
    acum=0; /* inicializa acumulador*/
    while (cont <3)
    {
        printf ("Ingrese un valor");
        scanf ("%d",&a);
        printf (" Numero ingresado:%d\n ",a);
        cont=cont+1; /* incrementa contador*/
        acum=acum+a;
    }
    return 0;
    printf(" El valor acumulado es :%d\n",acum);
}
```

En este problema se tiene perfectamente establecido el número de veces que un grupo de acciones se van a ejecutar. En este caso tres veces.

La condición será evaluada antes de cada iteración

El cuerpo del bucle while se ejecuta mientras la condición sea verdadera

El contador y el acumulador se actualiza en cada iteración

¿Qué hace el siguiente código?

```
int main()
{
    int a,b,c;
    printf("\n Introduce un número: ");
    scanf("%d",&a);
    b=1;
    while (b <= 10)
    {
        c=a*b;
        printf (" %d x %d es = %d \n",a,b,c);
        b=b+1;
    }
    return 0;
}
```

Comprobación o Prueba de Escritorio

a	b	c
4	1	4
4	2	8
4	3	12
4	4	16
4	5	20
4	6	24
..
4	10	40

Ejercicio



Calcular el perímetro de un cuadrado solamente cuando el valor ingresado sea correcto. La verificación se realizará solo tres veces. En caso de que se supere las tres posibilidades se mostrará por pantalla un mensaje explicando el error.

Usar sentencia While.

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

el perimetro de un cuadrado. Verificar que el valor ingresado sea correcto.

```
int main()
{
    int p,a,c;
    p=0;
    a=0;
    c=0;

    while (a <= 0 && c < 3 )
    {
        printf ("Ingrese un valor positivo\n");
        scanf ("%d",&a);
        c++;
    }
    if ( c==3) printf("Ingreso tres veces un valor incorrecto para calcular el
perímetro.Debe ingresar un valor positivo");
    else
    {
        p= a*4;
        printf ("Perimetro= %d\n",p);
    }
    return 0;
}
```



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

int main()
{
    int p,a;
    p=0;
    a=0;
    while (a <= 0)
    {
        printf ("Ingrese un valor positivo\n");
        scanf ("%d",&a);
    }
    p= a*4;
    printf ("Perimetro= %d\n",p);
    getch();
    return 0;
}
```

¿Qué dificultad
tiene este
programa?

Ejemplo: Calcular el perímetro de un cuadrado. Verificar que el lado ingresado sea correcto.



Calcular el perímetro de un cuadrado solamente cuando el valor ingresado sea correcto. Ingresar la cantidad de veces que se verificará que el valor ingresado sea correcto.



```
# include <stdio.h>

int main()
{
    int p,a,b,i;
    p=0;
    a=0;
    i=0;

    printf ("Ingrese la cantidad de veces que verificará el valor del lado\n");
    scanf ("%d",&b);

    while (a <= 0 && i<b )
    {
        printf ("Ingrese un valor positivo\n");
        scanf ("%d",&a);
        i++;
    }

    if (i==b) printf ("Supero la cantidad de intentos permitidos");
    else
    {
        p= a*4;
        printf ("Perimetro= %d\n",p);
    }
    return 0;
}
```

Ejemplo: Calcular el perímetro de un cuadrado. Verificar que el lado ingresado sea correcto.



Modificar el programa anterior para que el usuario pueda realizar otro cálculo de perímetro si lo desea.

Ejemplo: Calcular el perímetro de un cuadrado. Verificar que el lado ingresado sea correcto.

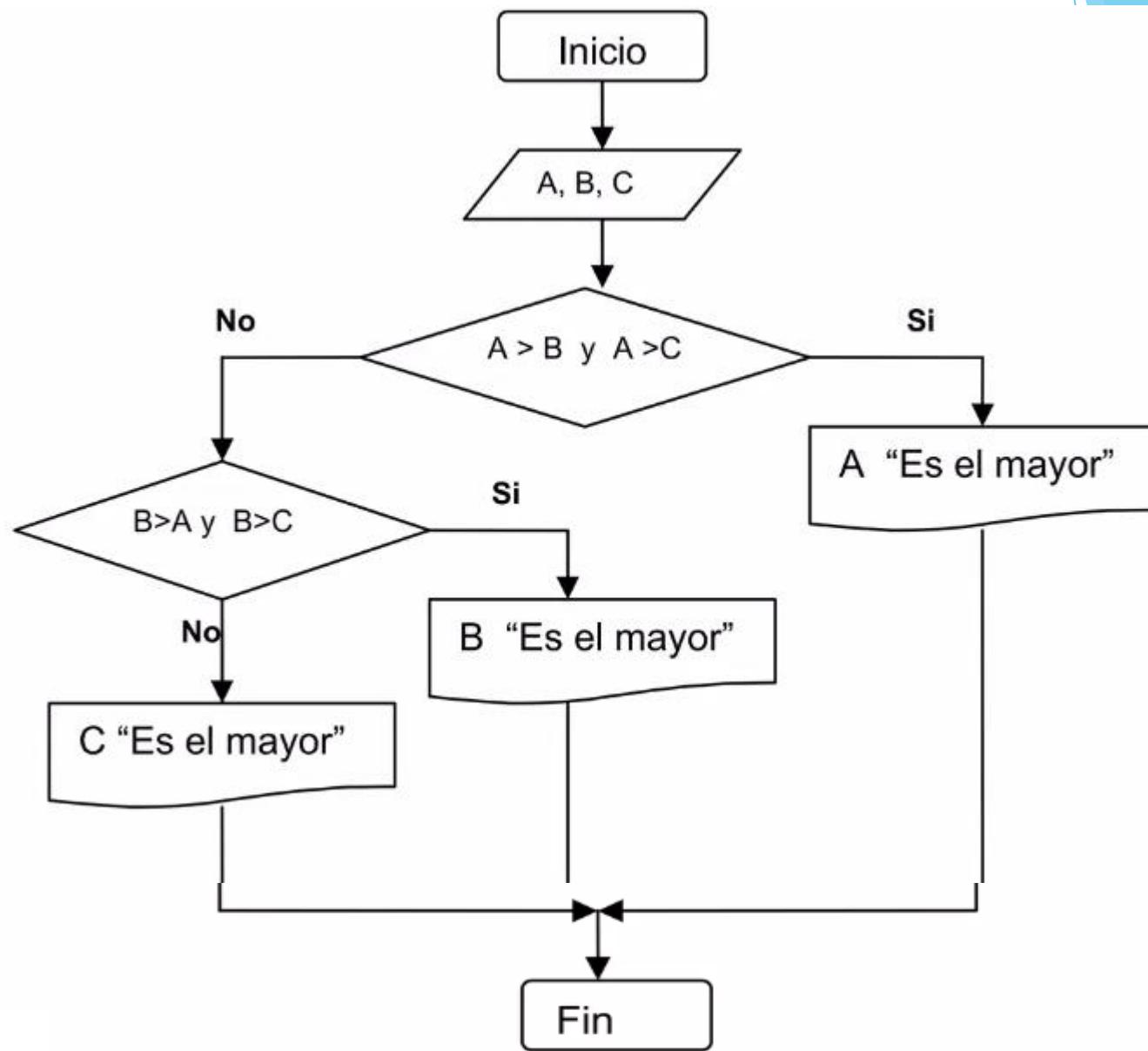


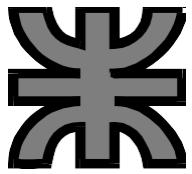
Realizar un programa que permita contar la cantidad de números pares ingresados por teclado.



```
# include <stdio.h> # include  
<conio.h>  
  
int main()  
{  
    /* Escribir y contar los números pares de 10 números ingresados */  
  
    int num, c, d, nu; c=1;  
    d=0;  
    printf("Ingrese cantidad de numeros a ingresar\n");  
    scanf ("%d",&nu);  
  
    while(c<=nu)  
    {  
        printf("Ingrese un numero\n");  
        scanf ("%d",&num);  
        if (num % 2 ==0)  
        {  
            printf ("Numero:%d\n",num);  
            d=d+1;  
        }  
        c++;  
    }  
    if (d==0) printf ("No se ingresaron números pares\n");  
    else printf("Se ingresaron %d números pares \n",d);  
    getch();  
    return 0;  
}
```

Dado el programa realizado anteriormente, modificarlo para que el usuario pueda repetirlo si lo desea.





Responder las siguientes preguntas

1. La sentencia break ¿Se puede utilizar en una estructura repetitiva?
2. La sentencia continue ¿Se puede utilizar en una estructura repetitiva?
3. ¿Existe relación entre la sentencia break y la sentencia continue?
4. ¿Qué es un ciclo controlado por Bandera o cantinela? Ejemplificar, crear un enunciado y realizar el correspondiente programa.



Universidad Tecnológica Nacional
Facultad Regional Resistencia
Técnico Superior en Programación

Programación I

Sentencia SWITCH

Sentencia SWITCH

Sentencia Switch

Se puede programar con un grupo de sentencias if then else anidadas, aunque ello puede ser de complicada lectura. Para evitarlo se puede usar la sentencia switch.

Su utilización es:

```
switch (valor)
{
    case valor1: <sentencias>
    case valor2: <sentencias>
    ...
    default: <sentencias>
}
```

La sentencia switch() en Lenguaje C es una sentencia de selección. Esta sentencia permite seleccionar las acciones a realizar de acuerdo al valor que tome una variable.

En la sentencia switch se comprueba el valor que ingrese el usuario que es el que tomará la variable expresión y lo compara con algunas de las etiquetas que acompañen a cada case. En caso de no encontrar ninguna coincidencia, el valor lo tomará como default.

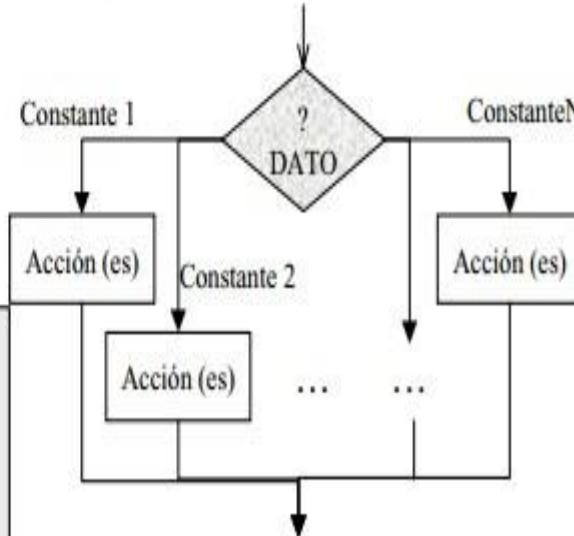
Sentencia Switch

Instrucción condicional múltiple : SWITCH

Estructura algorítmica asociada

Formato general de la
sentencia **witch**

```
switch (selector)
{
    case constante1:
        instrucción1 ó bloque de instrucciones
        break;
    case constante2:
        instrucción2 ó bloque de instrucciones
        break;
    default:
        instrucción2 ó bloque de instrucciones
}
```



Permiten comparar una ‘variable’ con distintos valores posibles, ejecutando para cada caso una serie de instrucciones específicas.

Sentencia Switch

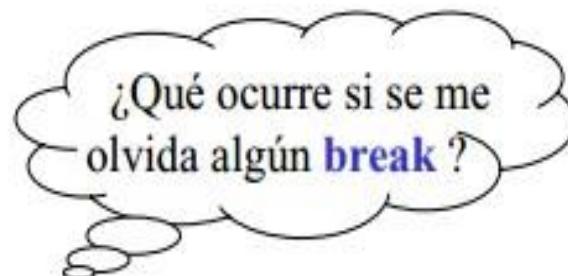
Instrucción condicional múltiple : SWITCH

Formato general de la
sentencia **witch**

```
switch (selector)
{
    case constante1:
        instrucción1 ó bloque de instrucciones
        break;
    case constante2:
        instrucción2 ó bloque de instrucciones
        break;
    default:
        instrucción3 ó bloque de instrucciones
}
```

El valor de *selector* debe ser un número entero. Puede ser una variable, una expresión ó una llamada a una función.

Cada caso comienza con un **case** y termina con un **break**



IMPORTANCIA DE BREAK

Cuando el control del programa llega a una sentencia switch y se evalúa la expresión de control de la misma, el programa comienza a ejecutar las sentencias presentes en el bloque correspondiente a alguna cláusula case, o quizá a la cláusula default.

¿Qué sucede después de ejecutar la última sentencia del bloque? Si el bloque es el último, el programa ejecuta la sentencia inmediatamente siguiente al switch. Pero si no es el último, el programa ejecuta la primera sentencia inmediatamente siguiente al bloque. Y luego la siguiente, y la siguiente, etc. La sentencia break, sirve para hacer que el programa abandone la sentencia switch. Si no se encuentra un break, se ejecuta el siguiente bloque de código, y así hasta hallar un break o salir de la sentencia switch.

Dicho de otro modo, las cláusulas case son puntos de entrada al conjunto de líneas de código que hay en el interior del switch. Las sentencias break son puntos de salida, y debemos organizar cuidadosamente ambas cosas para evitar errores que no podrá detectar el compilador.

Lenguaje de Programación C

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int num;
    printf( "Introduce un número " );
    scanf( "%i", &num );
    if ( num==1 ) printf ( "Es un 1\n" );
    else if ( num==2 ) printf ( "Es un 2\n" );
    else if ( num==3 ) printf ( "Es un 3\n" );
    else printf ( "No era ni 1, ni 2, ni 3\n" );
    getch();
    return 0;
}
```

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int num;

    printf( "Introduce un número " );
    scanf( "%d", &num );
    switch( num )
    {
        case 1: printf( "Es un 1\n" ); break;
        case 2: printf( "Es un 2\n" ); break;
        case 3: printf( "Es un 3\n" ); break;
        default: printf( "No es ni 1, ni 2, ni 3\n" ) break;
    }
    getch();
    return 0;
}
```

La sentencia **break** provoca la salida de **switch**.

En caso contrario continua la siguiente secuencia **case** o **default** aunque no se cumpla la condición.

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

int main() /* Escribe el día de la semana */
{
    int dia;

    printf("Introduce el día: ");

    scanf("%d",&dia);

    switch(dia)
    {
        case 1: printf ("Lunes"); break;
        case 2: printf ("Martes"); break;
        case 3: printf ("Miércoles"); break;
        case 4: printf ("Jueves"); break;
        case 5: printf ("Viernes"); break;
        case 6: printf ("Sábado"); break;
        case 7: printf ("Domingo"); break;
        default: printf ("No corresponde a ningun dia de la semana\n");break;
    }

    return 0;
    getch();
}
```



Resumen:

- La estructura *switch* es especialmente útil cuando la selección se basa en el valor de una variable simple o de una expresión simple denominada expresión de control o selector.
- Los valores de cada *case* del switch han de ser constantes.
- El valor de esta expresión puede ser de tipo *int* o *char*, pero no puede ser del tipo *float* ni *double*.
- La etiqueta *default* marca el bloque de código que se ejecuta por defecto (cuando al evaluar la expresión se obtiene un valor no especificado por los casos anteriores del *switch*).
- La sentencia **SWITCH** compara solamente igualdad.
- Por cada **CASE** puede haber 1 o más instrucciones.

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

int main()
{
    char car;
    printf("Ingrese un caracter");
    scanf ("%c",&car);

    switch(car)
    {
        case 'a': printf (" Vocal a");break;
        case 'e': printf (" Vocal e");break;
        case 'i': printf (" Vocal i");break;
        case 'o': printf (" Vocal o");break;
        case 'u': printf (" Vocal u");break;
        default: printf("No es vocal");break;
    }
    return 0;
    getch();
}
```



(Zinjal is not just another IDE)

Ejercicios:

El Estadio “Amigos en el futbol” tiene diversos sectores. El costo de la entrada a los eventos futbolísticos del estadio se asignan en virtud de los sectores del estadio mediante la siguiente tabla:

<i>Sector</i>	<i>Costo de la entrada</i>
Sol general	\$3
Sol preferente	\$5
Sombra	\$8
Tribuna	\$15
Platea	\$20

Se pide construir un programa que permita seleccionar un sector del estadio, ingresar la cantidad de entradas solicitadas y calcular el total a pagar por las entradas.

El programa de contar con un menú de opciones correspondiente con la siguiente imagen:

```
*.*****BIENVENIDO AL ESTADIO CUSCATLAN*****  
* Sectores del estadio *  
* A- Sol general *  
* B- Sol preferente *  
* C- Sombra *  
* D- Tribuna *  
* E- Platea *  
*****  
Selecciona la letra del sector del estadio :
```

IMPORTANTE: - Observe que las lecturas de datos están “validadas”, esto significa que en caso que el usuario ingrese valores incorrectos se genera un mensaje que indica al usuario cuál es el error cometido. –

Pruebe lo anterior ingresando un número de sector inexistente por ejemplo 10) o una cantidad de entradas incorrecta (por ejemplo -5).

PROGRAMACIÓN I

Unidad 2

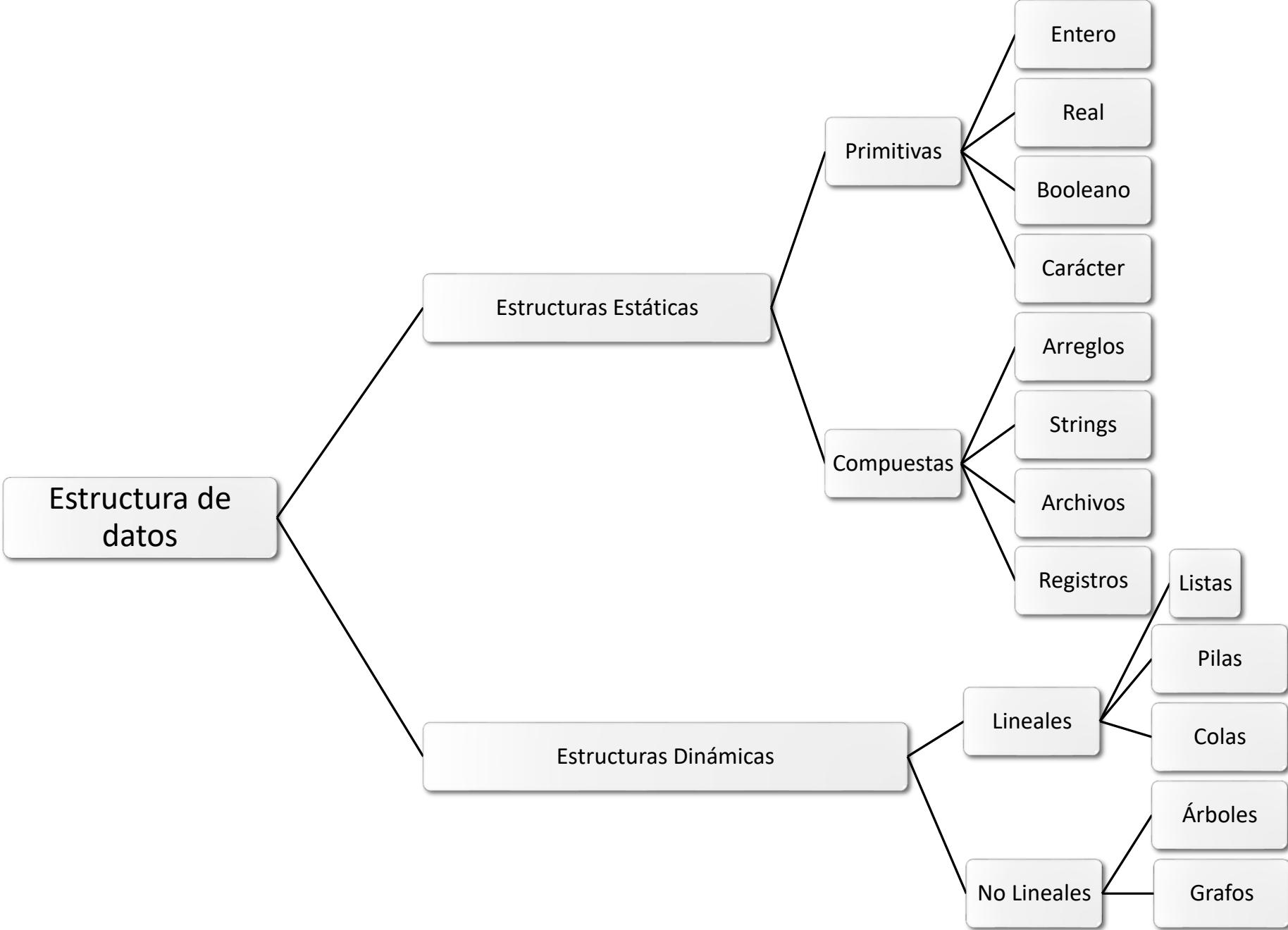
Tipos de Datos. Tipos Primitivos de Datos. Dominio de cada Tipo. Operaciones válidas. Ocupación de Memoria. Datos Compuestos. Estructuras de Datos. Estructuras estáticas y dinámicas. Tipos de Datos: Constantes, variables. Operadores matemáticos, relacionales y lógicos. Precedencia entre operadores. Operadores de asignación. Funciones básicas de entrada/salida.

Estructuras de Datos

Una estructura de datos es una forma de organizar y almacenar datos de manera que puedan ser utilizados eficientemente

Esta "disposición" permite que una estructura de datos sea eficiente en algunas operaciones e ineficiente en otras.

Las **estructuras de datos** son aquellas que permiten organizar la información de manera eficiente, y en definitiva diseñar la solución correcta para un determinado problema.



Las **ESTRUCTURAS DE DATOS ESTÁTICAS** son aquellas en las que el tamaño ocupado en memoria se define antes de que el programa se ejecute y no puede modificarse dicho tamaño durante la ejecución del programa, mientras que una **ESTRUCTURA DE DATOS DINÁMICA** es aquella en la que el tamaño ocupado en memoria puede modificarse durante la ejecución del programa.

ARREGLOS

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



STRINGS

Una cadena de caracteres es una secuencia ordenada de longitud finita de elementos que pertenecen a un cierto lenguaje formal o alfabeto análogas a una fórmula o a una oración.

En general, una cadena de caracteres es una sucesión de caracteres (letras, números u otros signos o símbolos).

Si no se ponen restricciones al alfabeto, una cadena podrá estar formada por cualquier combinación finita de los caracteres disponibles (las letras de la 'a' a la 'z' y de la 'A' a la 'Z', los números del '0' al '9', el espacio en blanco ' ', símbolos diversos '!', '@', '%', etcétera).

Un fichero o archivo es una colección ordenada de datos que tienen entre sí una relación y que se almacenan de forma permanente en un dispositivo de memoria no volátil.

En este contexto, **permanente** quiere decir que, salvo fallos catastróficos o hasta que sean borrados a propósito, estos datos permanecen en el medio en que se almacenan y continúan existiendo después de que el programa que los creó deja de ejecutarse, incluso después de apagar el ordenador.

Esto marca la diferencia con los datos que son provisionalmente almacenados en la memoria RAM, la **memoria volátil** del ordenador, que no *sobreviven* al programa que los crea y mucho menos a la desconexión del computador.

ESTRUCTURA DE DATOS LINEALES

Las **estructuras de datos lineales** son aquellas en las que los elementos ocupan lugares sucesivos en la estructura y cada uno de ellos tiene un único sucesor y un único predecesor, es decir, sus elementos están ubicados uno al lado del otro relacionados en forma lineal.

Hay tres tipos de **estructuras de datos lineales**:

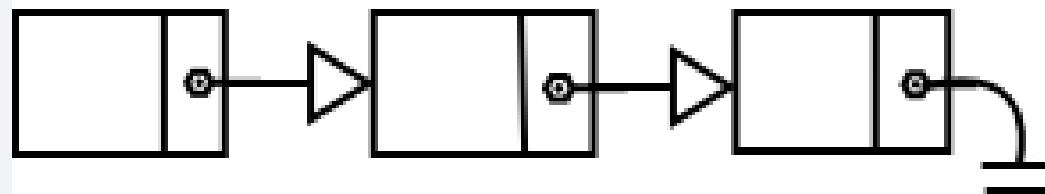
- Listas enlazadas
- Pilas
- Colas

LISTAS ENLAZADAS

Las listas enlazadas se construyen con elementos que están ubicados en una secuencia. Es decir que cada elemento se conecta con el siguiente a través de un enlace que contiene la posición del siguiente elemento. De este modo, teniendo la referencia del principio de la lista se puede acceder a todos los elementos de la misma.



Estructura de un nodo



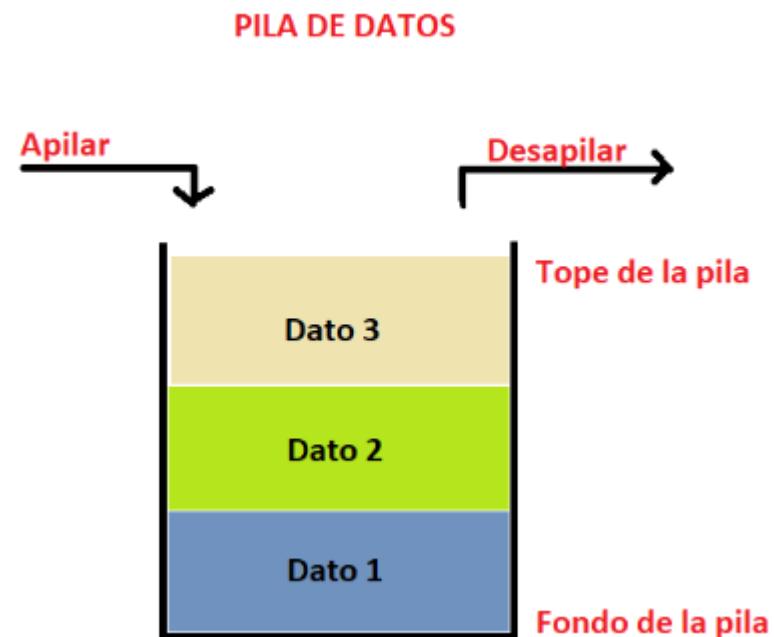
Lista enlazada

PILA

La **PILA** es un tipo especial de **lista lineal** dentro de las **estructuras de datos dinámicas** que permite almacenar y recuperar datos, siendo el modo de acceso a sus elementos de tipo LIFO (del inglés *Last In, First Out*, es decir, *último en entrar, primero en salir*).

¿Cómo funciona?

A través de dos operaciones básicas: apilar (push), que coloca un objeto en la pila, y su operación inversa, desapilar (pop), que retira el último elemento apilado.

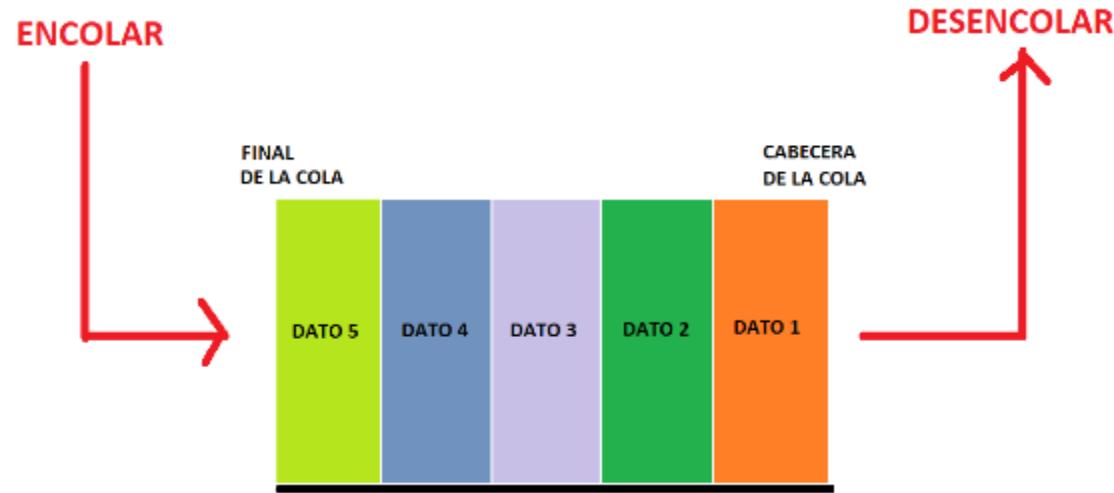


COLA

La COLA es un tipo especial de **lista lineal** dentro de las **estructuras de datos dinámicas** que permite almacenar y recuperar datos, siendo el modo de acceso a sus elementos de tipo FIFO (del inglés *First In, First Out*, es decir, *primero en entrar, primero en salir*).

¿Cómo funciona?

A través de dos operaciones básicas: encolar que coloca un objeto en la cola y su operación inversa, desencolar que retira el primer elemento encolado.



ESTRUCTURA DE DATOS NO LINEALES

Las **estructuras de datos no lineales**, también llamadas multienlazadas, son aquellas en las que cada elemento puede estar enlazado a cualquier otro componente. Es decir, cada elemento puede tener varios sucesores o varios predecesores.

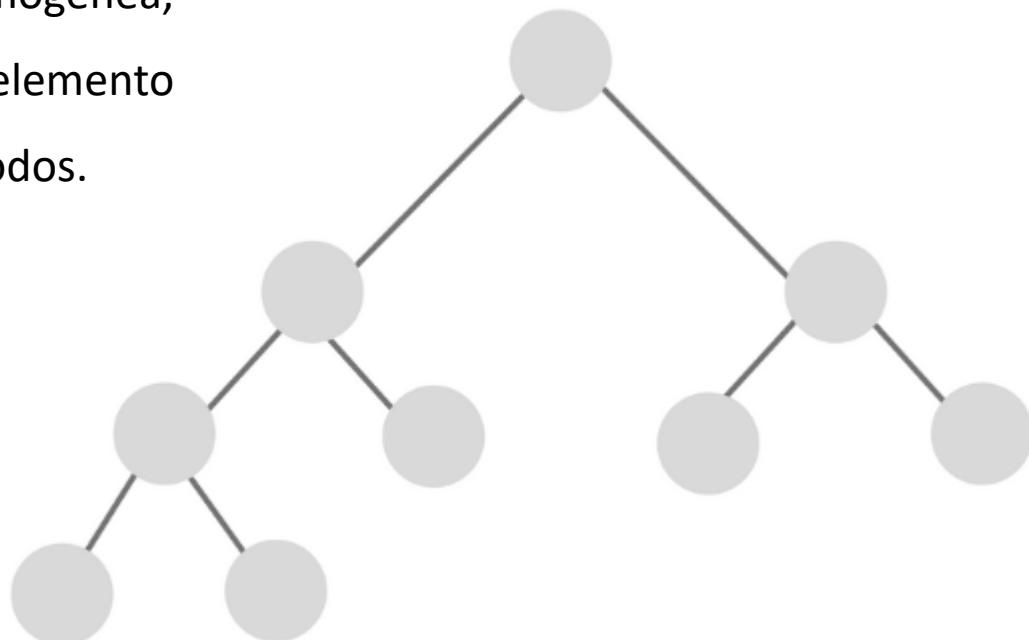
Existen dos tipos:

- Árboles
- Grafos

ARBOLES

En **estructura de datos**, los árboles consisten en una **estructura no lineal** que se utiliza para representar datos con una relación jerárquica en la que cada elemento tiene un único antecesor y puede tener varios sucesores.

Los mismos se encuentran clasificados en: **árbol general**, un árbol donde cada elemento puede tener un número ilimitado de sub árboles y **árboles binarios**, que son una estructura de datos homogénea, dinámica y no lineal en donde a cada elemento le pueden seguir como máximo dos nodos.



GRAFOS

Otro tipo de **no lineal** de **estructura de datos en programación**, son los **grafos**. Se trata de una estructura matemática formada por un conjunto de puntos —una estructura de datos— y un conjunto de líneas, cada una de las cuales une un punto a otro.

Los puntos se llaman **nodos** o **vértices** del grafo y las líneas se llaman **aristas** o **arcos**.

