

# TEMA 4.

## ESTRUCTURAS DE CONTROL

# CONTENIDOS

## 4.1. INTRODUCCIÓN

## 4.2. ESTRUCTURAS DE CONTROL ALTERNATIVAS

### 4.2.1. IF-ELSE

### 4.2.2. SWITCH

## 4.3. ESTRUCTURAS DE CONTROL REPETITIVAS

### 4.3.1. WHILE

### 4.3.2. DO-WHILE

### 4.3.3. FOR

## 4.4. ANIDAMIENTO DE ESTRUCTURAS DE CONTROL

- 4.1. INTRODUCCIÓN
- 4.2. ESTRUCTURAS DE CONTROL ALTERNATIVAS
  - 4.2.1. IF-ELSE
  - 4.2.2. SWITCH
- 4.3. ESTRUCTURAS DE CONTROL REPETITIVAS (BUCLES)
  - 4.3.1. WHILE
  - 4.3.2. DO-WHILE
  - 4.3.3. FOR
- 4.4. ANIDAMIENTO DE ESTRUCTURAS DE CONTROL

## 4.1. INTRODUCCIÓN

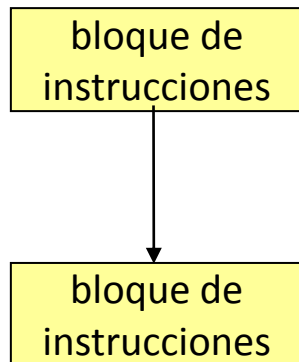
# Las estructuras de control

- Permiten alterar la secuencia de ejecución de las instrucciones de un programa.
- Estructuras de control
  - Ejecución **secuencial**
    - Las instrucciones se ejecutan una detrás de otra
  - Estructuras de control **alternativas**
    - La ejecución secuencial se rompe dependiendo del resultado de una expresión lógica (condición)
  - Estructuras de control **repetitivas**
    - La ejecución secuencial se rompe al repetir un conjunto de instrucciones varias veces

# Programación estructurada

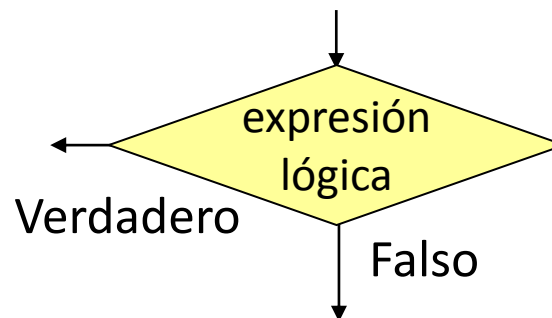
- Técnica de programación
  - Garantiza crear buenos programas (fáciles de mantener)
  - Sólo se permiten tres tipos de estructuras de control

## Secuencial



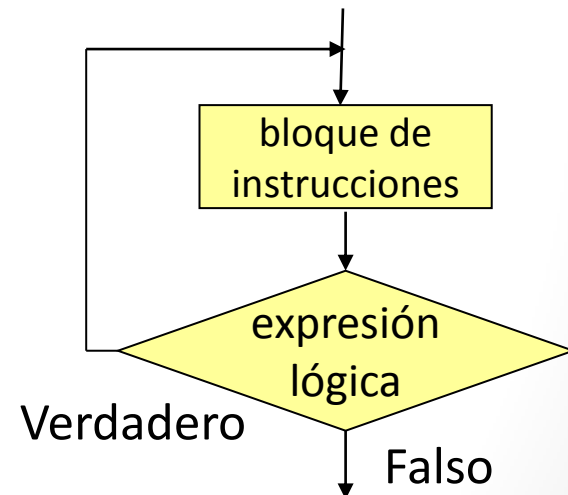
## Alternativa

`if-else, switch`



## Repetitiva

`for, while, do-while`



Es posible escribir cualquier algoritmo usando sólo estos tres tipos de estructuras, sin necesitar otros, como “saltos”.

- 4.1. INTRODUCCIÓN
- 4.2. ESTRUCTURAS DE CONTROL ALTERNATIVAS
  - 4.2.1. IF-ELSE
  - 4.2.2. SWITCH
- 4.3. ESTRUCTURAS DE CONTROL REPETITIVAS (BUCLES)
  - 4.3.1. WHILE
  - 4.3.2. DO-WHILE
  - 4.3.3. FOR
- 4.4. ANIDAMIENTO DE ESTRUCTURAS DE CONTROL

## 4.2. ESTRUCTURAS DE CONTROL ALTERNATIVAS

# Estructuras de control alternativas

- Alteran la secuencia de ejecución según el resultado de evaluar una expresión.
- También llamadas de selección o condicionales.

```
if (expresión) {  
    bloque de instrucciones 1;  
}else{  
    bloque de instrucciones 2;  
}
```

```
switch (expresión){  
    case <etiqueta1> : bloque de instrucciones 1;  
    case <etiqueta2> : bloque de instrucciones 2; break;  
    default : bloque de instrucciones 3;  
}
```

## 4.1. INTRODUCCIÓN

## 4.2. ESTRUCTURAS DE CONTROL ALTERNATIVAS

### 4.2.1. IF-ELSE

### 4.2.2. SWITCH

## 4.3. ESTRUCTURAS DE CONTROL REPETITIVAS

### 4.3.1. WHILE

### 4.3.2. DO-WHILE

### 4.3.3. FOR

## 4.4. ANIDAMIENTO DE ESTRUCTURAS DE CONTROL



# Estructura de Control *If*

- Es la estructura de control más simple.
- Se evalúa la expresión lógica contenida entre paréntesis
  - Si es verdadera se ejecutan las sentencias
  - Si es falsa se continua con la siguiente instrucción

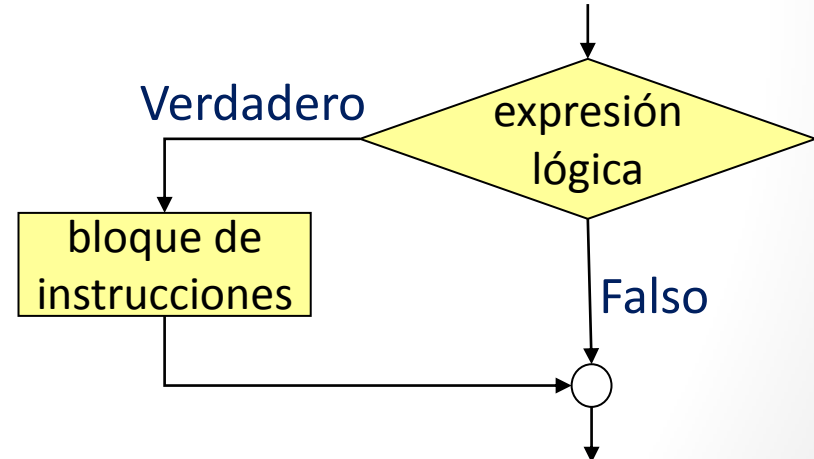
## Sintaxis:

```
if (expresión_lógica) {  
    bloque_de_instrucciones;  
}
```

## Ejemplo

```
if (edad > 18){  
    printf("ADULTO");  
    precioEntrada = 20;  
}
```

## Diagrama de flujo



# Estructura de Control *If-Else*

- Si la expresión es verdadera se ejecuta el bloque de código asociado a `if`.
- Si la expresión es falsa se ejecuta el bloque de código asociado a `else`.

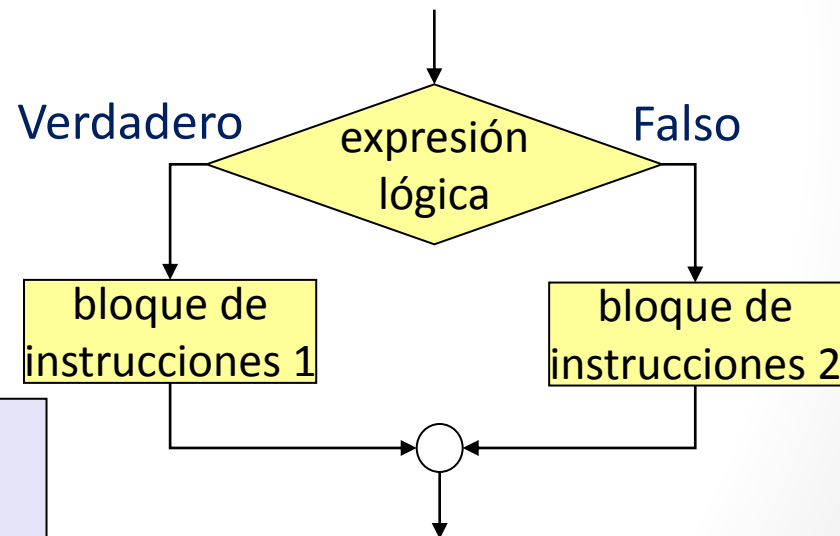
## Sintaxis:

```
if (expresión_lógica){  
    bloque_instrucciones_1;  
} else {  
    bloque_instrucciones_2;  
}
```

## Ejemplo:

```
if (a > b){  
    printf("A mayor que B");  
} else {  
    printf("A menor o igual a B");  
}
```

## Diagrama de flujo



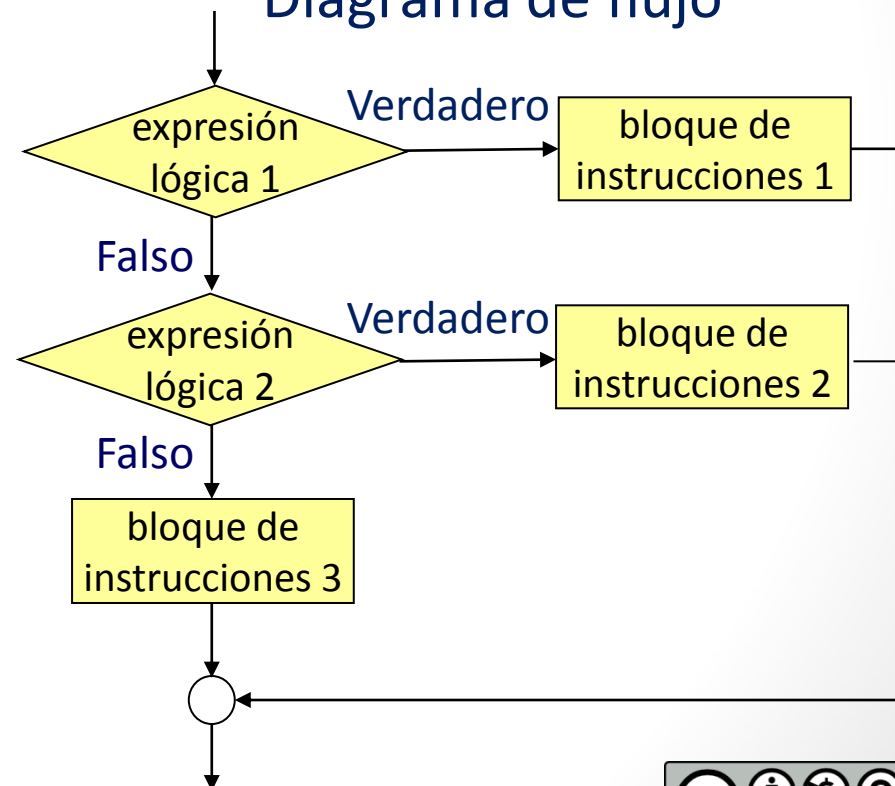
# Estructura de Control *If Anidadas*

- Representan diferentes ejecuciones alternativas y mutuamente exclusivas.
  - En caso de que todas las expresiones lógicas sean falsas se ejecutará el último bloque.

## Sintaxis:

```
if (expresión_lógica_1){  
    bloque_instrucciones_1;  
} else if (expresión_lógica_2){  
    bloque_instrucciones_2;  
} else {  
    bloque_instrucciones_3;  
}
```

## Diagrama de flujo



# Ejemplo 1

```
#include <stdio.h>

int main(void)
{
    int nota; //Variable para almacenar la nota

    printf("Introduzca una nota numerica para el alumno: (0-10) \n");
    scanf("%i", &nota);

    if ( (nota >= 0) && (nota < 5) ){
        printf("El alumno ha suspendido \n");
    } else if ( nota <= 10 ){
        printf("El alumno ha aprobado \n");
    } else{
        printf("La nota introducida es incorrecta. \n");
        printf("Rango valido 0 - 10 \n");
    }

    return 0;
}
```

} alternativa 1

} alternativa 2

} alternativa por defecto

# Ejemplo 2

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

int main(int argc, char *argv[]){
    int num1, num2;
    printf ("Introduzca el valor de los numeros: num1 y num2 \n");
    scanf ("%d", &num1);
    scanf ("%d", &num2);

    if (num1 > num2) {
        printf ("Bloque 1: num1 es mayor que num2 \n");
        num2=num2+10;
        printf ("Bloque 1: Ahora num1 vale %d y num2 vale %d \n",num1, num2);
    }else if (num2>num1){
        printf ("Bloque 2: num2 es mayor que num1 \n");
        num1=num1+10;
        printf ("Bloque 2: Ahora num1 vale %d y num2 vale %d \n",num1, num2);
    }
    if (num1 > num2) {
        printf ("Bloque 3: num1 es mayor que num2 \n");
        num2=num2+10;
        printf ("Bloque 3:Ahora num1 vale %d y num2 vale %d \n",num1, num2);
    }
    if (num2>num1){
        printf ("Bloque 4: num2 es mayor que num1 \n");
        num1=num1+10;
        printf ("Bloque 4: Ahora num1 vale %d y num2 vale %d \n",num1, num2);
    }
    system("PAUSE");
    return 0;
}
```

## Ejemplo 2 (cont)

**Si num1 es MENOR que num2:**

- If-else if (BLOQUE 1 y 2): Entramos en BLOQUE 2 (excluyente con BLOQUE 1)
- If (BLOQUE 3 y 4): Entramos en ambos BLOQUES (NO excluyentes)

```
C:\Dev-Cpp\IF_else.exe

Introduzca el valor de los numeros: num1 y num2
2 3
Bloque 2: num2 es mayor que num1
Bloque 2: Ahora num1 vale 12 y num2 vale 3
Bloque 3: num1 es mayor que num2
Bloque 3: Ahora num1 vale 12 y num2 vale 13
Bloque 4: num2 es mayor que num1
Bloque 4: Ahora num1 vale 22 y num2 vale 13
Presione una tecla para continuar . . .
```

## Ejemplo 2 (cont)

**Si num1 es MAYOR que num2:**

- If-else if (BLOQUE 1 y 2): Entramos en BLOQUE 1 (excluyente con BLOQUE 2)
- If (Bloque 3 y 4): Entramos en BLOQUE 4 ya que NO se cumple la condición para entrar en BLOQUE 3

```
C:\Dev-Cpp\IF_else.exe

Introduzca el valor de los numeros: num1 y num2
3 2
Bloque 1: num1 es mayor que num2
Bloque 1: Ahora num1 vale 3 y num2 vale 12
Bloque 4: num2 es mayor que num1
Bloque 4: Ahora num1 vale 13 y num2 vale 12
Presione una tecla para continuar . . .
```

## 4.1. INTRODUCCIÓN

## 4.2. ESTRUCTURAS DE CONTROL ALTERNATIVAS

### 4.2.1. IF-ELSE

### 4.2.2. SWITCH

## 4.3. ESTRUCTURAS DE CONTROL REPETITIVAS

### 4.3.1. WHILE

### 4.3.2. DO-WHILE

### 4.3.3. FOR

## 4.4. ANIDAMIENTO DE ESTRUCTURAS DE CONTROL

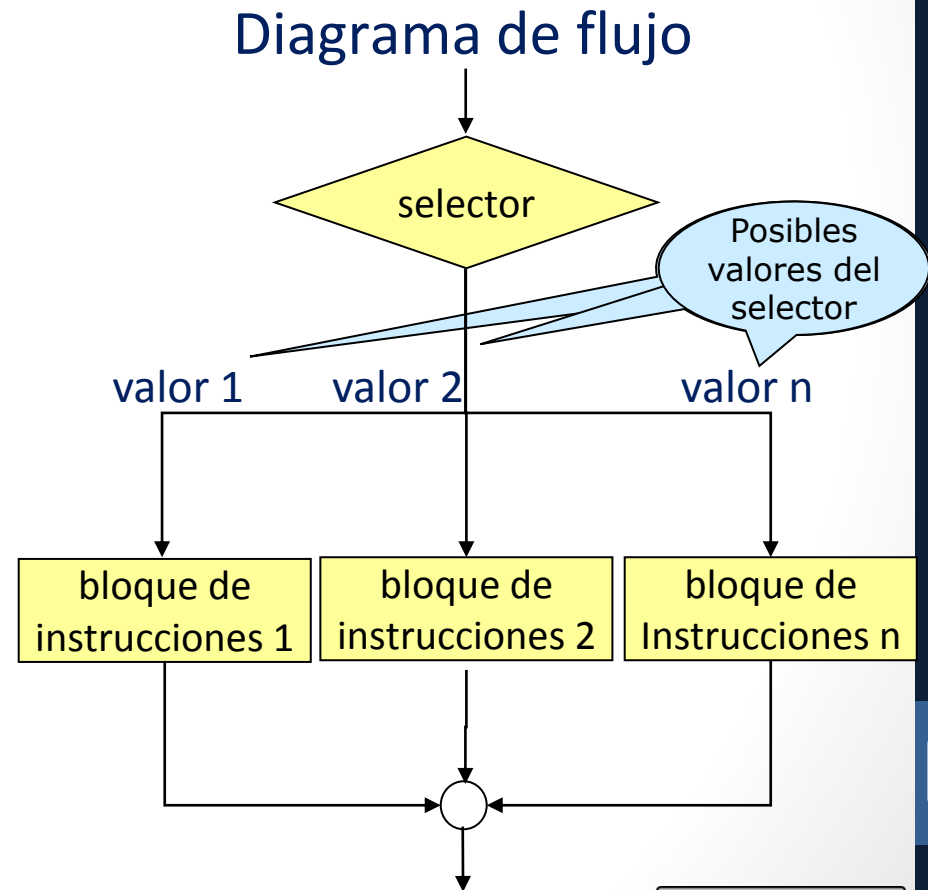


# Estructura de Control *Switch*

- Permite implementar estructuras de selección múltiple de forma sencilla, a partir de una variable selectora.

## Sintaxis:

```
switch(selector){  
  case valor 1:  
    bloque_instrucciones_1;  
    break;  
  case valor 2:  
    bloque_instrucciones_2;  
    break;  
  case valor n:  
    bloque_instrucciones_n;  
    break;  
  .....  
}
```



## Ejemplo. Mostrar nombre de un polígono en función del número de lados

```
#include <stdio.h>

int main(void)
{
    int numeroLados; //Variable para almacenar el valor
    scanf("%i", &numeroLados); //Se lee el número de lados

    switch (numeroLados){ //La variable es el selector
        case 0: case 1: case 2: //Varios posibles valores agrupados
            printf("no es un poligono");
            break;
        case 3:
            printf("triangulo");
            break;
        case 4:
            printf("rectangulo");
            break;
        case 5:
            printf("pentagono");
        }

    return 0;
}
```

# Estructura de Control *Switch*

- El selector:
  - Debe ser una variable o expresión de tipo entera, lógica o carácter.
    - No puede ser una expresión real.
- Al finalizar cada bloque se debe incluir la instrucción `break`
  - El efecto de la instrucción es dar por terminada la ejecución de la instrucción `switch`.
  - Si se omite la instrucción `break`, se ejecutarán todas las instrucciones del `switch` a partir de ese punto, hasta encontrar una nueva instrucción `break`.
- Bloque `default`:
  - Si el valor de la variable selectora no coincide con el valor de algún bloque, se ejecuta (si existe) el bloque por defecto o `default`.

# Ejemplo-Mostrar si una letra introducida por teclado es una vocal (utilizando switch)

```
int main(void)
{
    char c; //Se define la variable
    scanf("%c",&c); // Se lee la variable
    switch (c){
        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: //Bloque que se ejecuta si no coincide ningún valor
            printf ("consonante");
    }
    system("PAUSE");
    return 0;
}
```

- 4.1. INTRODUCCIÓN
- 4.2. ESTRUCTURAS DE CONTROL ALTERNATIVAS
  - 4.2.1. IF-ELSE
  - 4.2.2. SWITCH
- 4.3. ESTRUCTURAS DE CONTROL REPETITIVAS (BUCLES)
  - 4.3.1. WHILE
  - 4.3.2. DO-WHILE
  - 4.3.3. FOR
- 4.4. ANIDAMIENTO DE ESTRUCTURAS DE CONTROL

## 4.3. ESTRUCTURAS DE CONTROL REPETITIVAS

# Estructuras de control repetitivas

- Llamadas también iterativas o bucles
- Tres tipos
  - `while` y `do-while`
    - Repiten un bloque instrucciones mientras una condición sea cierta
    - Se emplea si no se sabe a priori el número de repeticiones
    - Ejemplo uso:
      - Leer números mientras no se lea el valor 7.
      - Leer números mientras su suma no llegue a 100.
  - `for`
    - Repite las instrucciones para un número determinado de veces. Se emplea cuando se sabe el número de repeticiones necesarias.
      - Ejemplo uso:
        - Leer 100 números.

## 4.1. INTRODUCCIÓN

## 4.2. ESTRUCTURAS DE CONTROL ALTERNATIVAS

### 4.2.1. IF-ELSE

### 4.2.2. SWITCH

## 4.3. ESTRUCTURAS DE CONTROL REPETITIVAS

### 4.3.1. WHILE

### 4.3.2. DO-WHILE

### 4.3.3. FOR

## 4.4. ANIDAMIENTO DE ESTRUCTURAS DE CONTROL

# Estructura de control `while`

- Ejecuta un bloque de instrucciones **mientras** una condición o expresión lógica sea cierta
- La expresión lógica se evalúa antes de iniciar la ejecución del bloque de instrucciones.
  - El número de repeticiones mínimo es 0
  - Después de ejecutar el bloque de instrucciones la condición se vuelve a evaluar.
  - Si la condición continua siendo cierta comienza una nueva ejecución del bloque.
  - Si la condición ha pasado a ser falsa se termina el bucle.
- Debemos asegurarnos que la condición llegue a ser falsa alguna vez
  - O el bucle se ejecuta eternamente (bucle infinito)

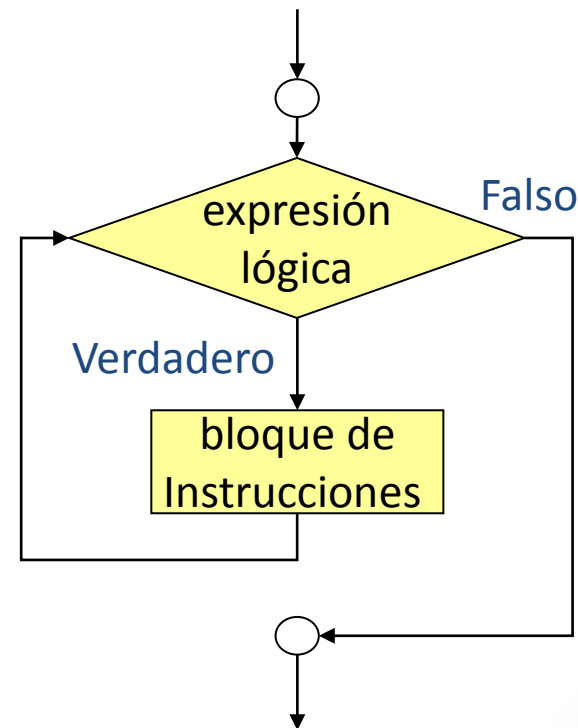


# Estructura de control `while`

## Sintaxis:

```
while (expresión_lógica) {  
    bloque_de_instrucciones;  
}
```

## Diagrama de flujo



## Ejemplo. Programa que solicita números hasta que se introduce el 0 (utilizando while).

```
int main(void)
{
    int clave;
    clave= 1; // inicialmente, el valor de la variable es 1

    // el valor de la variable debe ser distinto a 0
    // para poder ejecutar las instrucciones del bucle
    while (clave != 0) { //mientras sea diferente a 0
        printf("Introduzca la clave: ");
        scanf("%d",&clave);
    }
    // el bucle dejará de ejecutarse únicamente
    // cuando el valor de la variable sea 0
    printf("Ha introducido la clave correcta");

    system("PAUSE");
    return 0;
}
```

## 4.1. INTRODUCCIÓN

## 4.2. ESTRUCTURAS DE CONTROL ALTERNATIVAS

### 4.2.1. IF-ELSE

### 4.2.2. SWITCH

## 4.3. ESTRUCTURAS DE CONTROL REPETITIVAS

### 4.3.1. WHILE

### 4.3.2. DO-WHILE

### 4.3.3. FOR

## 4.4. ANIDAMIENTO DE ESTRUCTURAS DE CONTROL

## Estructura de control `do-while`

- Al igual que *while*, ejecuta un bloque de instrucciones **mientras** una condición o expresión lógica sea cierta.
- La expresión lógica se evalúa **después** de ejecutar el bloque de instrucciones.
  - El número de repeticiones mínimo es uno.
- Después de ejecutar el bloque de instrucciones la condición se vuelve a evaluar.
  - Si la condición continua siendo cierta comienza una nueva ejecución del bloque.
  - Si la condición ha pasado a ser falsa se termina el bucle
- Al igual que con *while*, se puede crear un bucle infinito

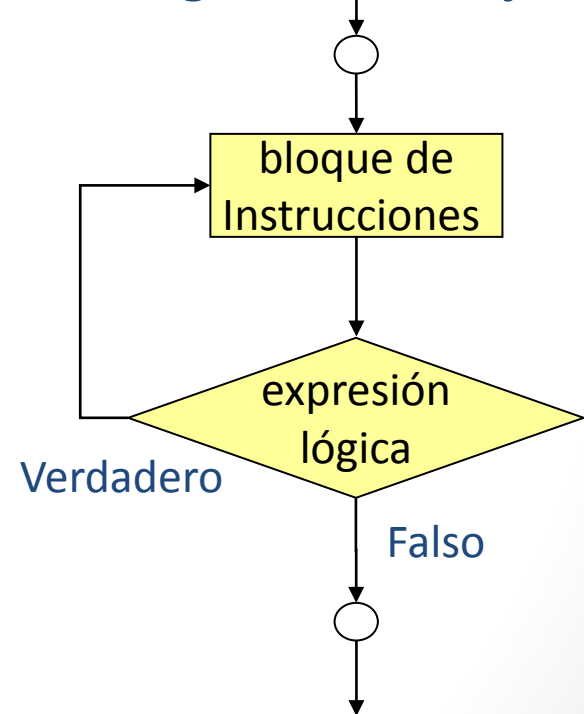
# Estructura de control do-while

- El bloque de instrucciones se ejecutará siempre al menos una vez

## Sintaxis:

```
do {  
    bloque_de_instrucciones;  
} while (expresión_lógica);
```

## Diagrama de flujo



## Ejemplo. Programa que solicita números hasta que se introduce el 0 (utilizando do-while).

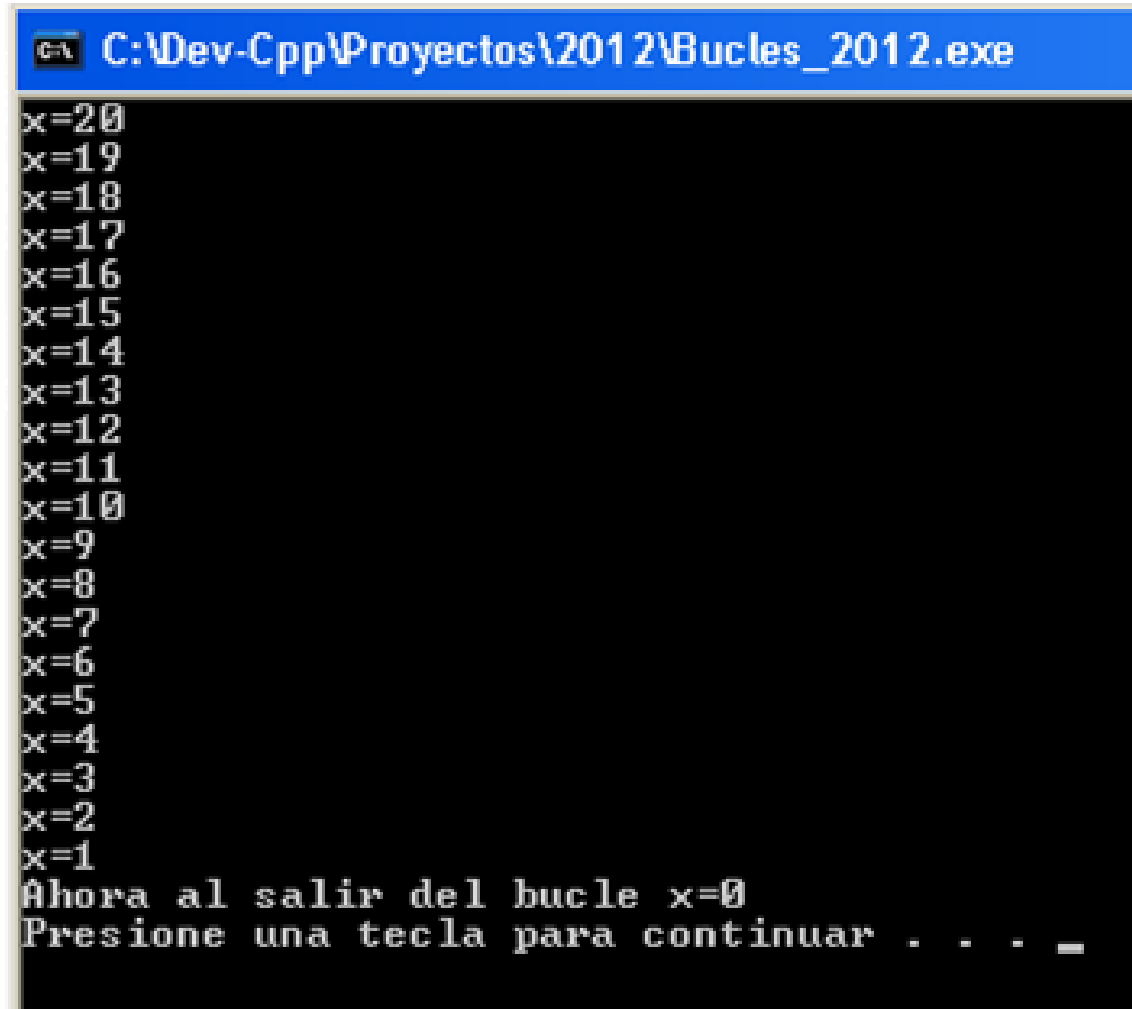
```
int main(void)
{
    int clave;
    // no es necesario dar un valor inicial a la variable
    // porque el bucle se ejecutara al menos 1 vez

    do{
        printf("Introduzca la clave: ");
        scanf("%d",&clave);
    } while (clave != 0);

    // el bucle dejara de ejecutarse unicamente
    // cuando el valor de la variable sea 0
    printf("Ha introducido la clave correcta");

    system("PAUSE");
    return 0;
}
```

# Ejemplo.Salida...



```
C:\Dev-Cpp\Proyectos\2012\Bucles_2012.exe
x=20
x=19
x=18
x=17
x=16
x=15
x=14
x=13
x=12
x=11
x=10
x=9
x=8
x=7
x=6
x=5
x=4
x=3
x=2
x=1
Ahora al salir del bucle x=0
Presione una tecla para continuar . . . _
```

## 4.1. INTRODUCCIÓN

## 4.2. ESTRUCTURAS DE CONTROL ALTERNATIVAS

### 4.2.1. IF-ELSE

### 4.2.2. SWITCH

## 4.3. ESTRUCTURAS DE CONTROL REPETITIVAS

### 4.3.1. WHILE

### 4.3.2. DO-WHILE

### 4.3.3. FOR

## 4.4. ANIDAMIENTO DE ESTRUCTURAS DE CONTROL



## Estructura de control `for`

- Ejecuta un bloque de instrucciones repetidamente mientras una condición o expresión lógica sea cierta.
- A diferencia de `while` y `do-while`, contiene una instrucción de **inicialización** y otra de **actualización**.
  - En la primera iteración de una estructura de control `for`, la instrucción de inicialización es ejecutada.
  - Mientras la expresión de control sea cierta, el bloque de instrucciones se ejecuta.
  - Tras finalizar el bloque de instrucciones, la instrucción de actualización es ejecutada.

# Estructura de control for

## Sintaxis:

```
for (inicialización; expresión_lógica; actualización) {  
    bloque_de_instrucciones;  
}
```

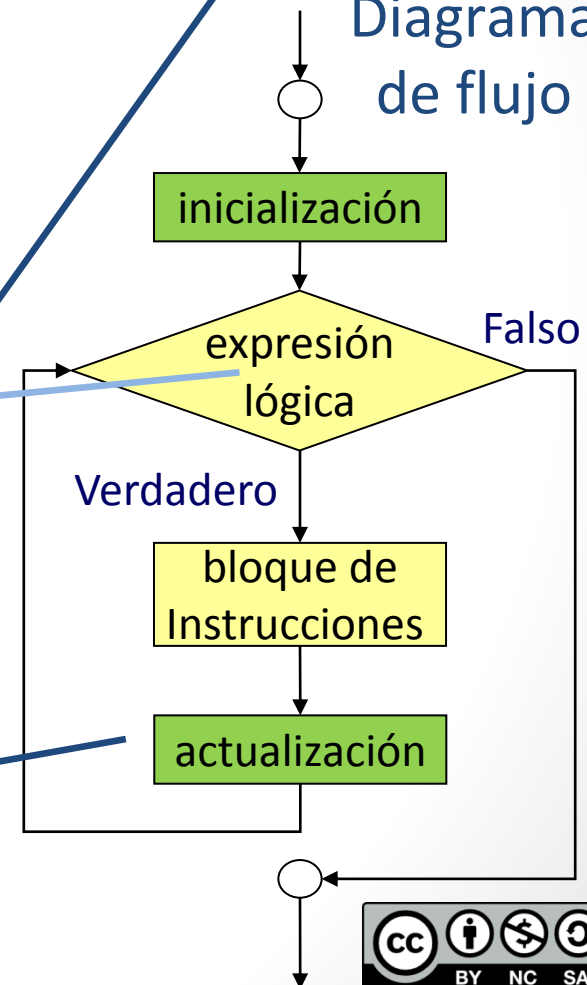
### expresión lógica:

Expresión que se comprueba en cada repetición del bucle, y determina si continúa o no

### actualización

Actualización que se realiza en cada repetición del bucle

## Diagrama de flujo



## Ejemplo. Estructura de control `for`

- Ejemplo: Programa que suma todos los enteros entre 1 y 10

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

int main(void)
{
    int i;
    int suma = 0;

    for (i=1; i<=10; i++){
        suma = suma + 1;
        printf(" El valor de i es %d \n", i);
        printf(" El valor de suma es %d \n", suma);
    }
    printf(" El valor final de i es %d \n ", i);
    printf(" El valor de la suma final es %d ", suma);
    system("PAUSE");
    return 0;
}
```

inicialización

expresión de control

actualización

## Ejemplo: Ejecución

```
El valor de i es 1
El valor de suma es 1
El valor de i es 2
El valor de suma es 2
El valor de i es 3
El valor de suma es 3
El valor de i es 4
El valor de suma es 4
El valor de i es 5
El valor de suma es 5
El valor de i es 6
El valor de suma es 6
El valor de i es 7
El valor de suma es 7
El valor de i es 8
El valor de suma es 8
El valor de i es 9
El valor de suma es 9
El valor de i es 10
El valor de suma es 10
El valor final de i es 11
    El valor de la suma final es 10
```

- 4.1. INTRODUCCIÓN
- 4.2. ESTRUCTURAS DE CONTROL ALTERNATIVAS
  - 4.2.1. IF-ELSE
  - 4.2.2. SWITCH
- 4.3. ESTRUCTURAS DE CONTROL REPETITIVAS (BUCLES)
  - 4.3.1. WHILE
  - 4.3.2. DO-WHILE
  - 4.3.3. FOR
- 4.4. ANIDAMIENTO DE ESTRUCTURAS DE CONTROL

## 4.4. ANIDAMIENTO DE ESTRUCTURAS DE CONTROL

# Anidamiento de estructuras de control

- En el bloque de instrucciones de cualquier estructura de control se puede incluir otra estructura de control
- Permite desarrollar programas más complejos y completos
- Se deben seguir las siguientes reglas:
  - La estructura de control interna debe estar totalmente incluida dentro de la externa y no pueden existir solapamientos.
  - Las expresiones de control deben estar definidas de tal modo, que por cada iteración del bucle externo se ejecute totalmente la estructura del bucle interno.

**Ejemplo:** Programa que pide un número al usuario y escribe todos los naturales entre 1 y ese número. Esta operación se repetirá hasta que el usuario indique que quiere parar, introduciendo un 0.

Bucle *for*  
anidado  
dentro de  
un bucle  
*while*

```
int main(void) {
    int num, i;
    int salir = 1;
    //Se repite hasta que el usuario inserte 0
    while (salir!=0){
        printf("Introduzca un numero");
        scanf("%d", &num);
        printf("Los números del 1 al %d son: ", num);
        for (i=1; i<=num; i++){
            printf("%d , ", i);
        } //Fin for
        printf("¿Desea salir? (0-si, 1-no) ");
        scanf("%d", &salir);
    } //Fin while

    return 0;
}
```

- **Ejemplo:** Programa que muestre una matriz de números enteros de 2 columnas y 3 filas.

Bucle *for*  
anidado  
dentro de  
otro bucle  
*for*

```
int main(void) {  
  
    int i, j;  
    int columnas = 2;  
    int filas = 3;  
  
    for (i=1; i<=filas; i++){  
        for (j=1; j<=columnas; j++){  
            printf("(%d, %d)", i, j);  
        } //fin for j  
        printf("\n");  
    } //fin for i  
  
    system("PAUSE");  
    return 0;  
}
```

- Salida del programa:

```
(1,1) (1,2)  
(2,1) (2,2)  
(3,1) (3,2)
```



- **Ejemplo:** Programa que pida un número representando un día de la semana y escriba el nombre correspondiente. Preguntará al usuario si desea seguir o no. Repetirá la operación hasta que el usuario lo indique.

Bucle  
*switch*  
anidado  
dentro de un  
bucle *while*.

```
int main(void) {
    char seguir;
    int n;

    do {
        printf("\n Introduzca un entero [0..7]: ");
        scanf("%d", &n);

        switch (n) {
            case 1: printf("\n Lunes"); break;
            case 2: printf("\n Martes"); break;
            case 3: printf("\n Miercoles"); break;
            case 4: printf("\n Jueves"); break;
            case 5: printf("\n Viernes"); break;
            case 6: printf("\n Sabado"); break;
            case 7: printf("\n Domingo"); break;
            default: printf("\n Numero incorrecto");
        } //fin switch

        printf("\n Desea seguir? s/n: ");
        scanf("%s", &seguir);
    } while (seguir == 's')

    system("PAUSE");
    return 0;
}
```

# Resumen de programación en C hasta ahora...

- Estructura de un programa

```
#include "stdio.h"

int main(void)
{
    Instrucciones de declaración...
    Instrucciones ejecutables ...
}
```

- Operador de asignación

=

- Instrucciones de declaración

```
tipo nombre_variable;
const tipo nombre_constante = valor;
```

- Instrucciones básicas de entrada y salida (con enteros)

```
scanf("%i", &variable);
printf("%i", variable);
```

# Resumen de programación en C hasta ahora...

- Estructuras de control alternativas (condicionales)

```
if (expresión_lógica_1){  
    bloque_instrucciones_1;  
} else if(expresión_lógica_2){  
    bloque_instrucciones_2;  
} else {  
    bloque_instrucciones_3;  
}
```

```
switch(selector){  
    case valor 1:  
        bloque_instrucciones_1;  
        break;  
    case valor 2:  
        bloque_instrucciones_2;  
        break;  
    case valor n:  
        bloque_instrucciones_n;  
        break;  
    default:  
        bloque_instrucciones;  
}
```

# Resumen de programación en C hasta ahora...

- Estructuras de control repetitivas (bucles)

```
for (inicialización; expresión_logica; actualización) {  
  
    bloque_de_instrucciones;  
  
}
```

```
while (expresión_logica) {  
  
    bloque_de_instrucciones;  
  
}
```

**Se ejecuta cero  
o más veces.**

```
do {  
  
    bloque_de_instrucciones;  
  
} while (expresión_logica);
```

**¡Se ejecuta al  
menos 1 vez!**