

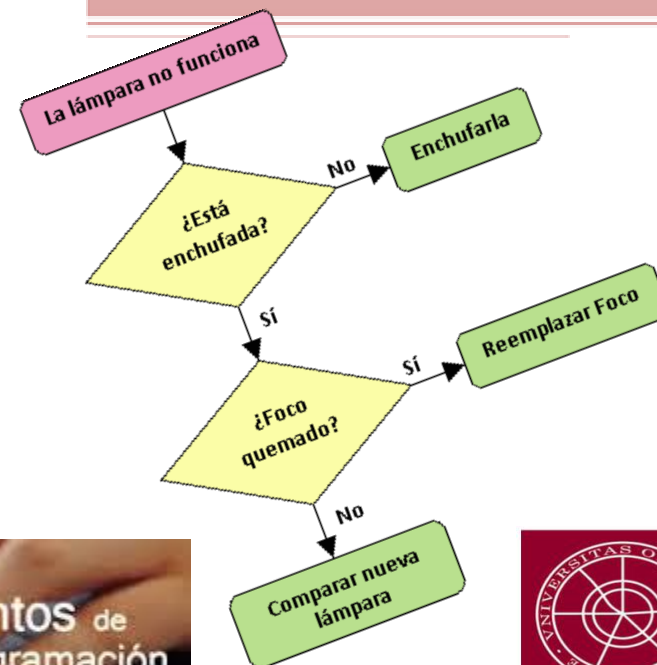
# Tema 3.

## Estructuras de Control

### Fundamentos de Programación Grado en Ingeniería Informática

Profesores:

José Manuel Martín Ramos  
Francisco Roche Beltrán.



DEPARTAMENTO DE  
TECNOLOGÍAS DE  
LA INFORMACIÓN

Universidad de Huelva

# ÍNDICE

1. Sentencias y Bloques de Sentencias.
2. Sentencias Secuenciales.
3. Sentencias Condicionales.
4. Sentencias Iterativas.
5. Macros de Sentencias.

# 1. Sentencias y Bloques de Sentencias.

- Una **Sentencia** en c/c++ es una **instrucción** o una **expresión** del lenguaje de alto nivel que puede escribirse en una o más líneas de código pero que siempre termina con el símbolo **;** para indicar el final de la sentencia actual y el principio de la siguiente.
- Una **instrucción** del lenguaje puede ser una **declaración de variables o contantes, palabras reservadas** (teniendo en cuenta su sintaxis) como por ejemplo **return**, **cout**, **cin** y más ejemplos que se verán a lo largo del curso.

```
#include <iostream>
using namespace std;

#define pi 3.1415926

int main()
{
    float longitud;
    float radio;

    cout << "Introduce el radio: ";
    cin >> radio;

    longitud = 2 * pi * radio;

    cout << "La longitud es: "
         << longitud << endl;
    return 0;
}
```

- Una **expresión** es aquella que está formada por operandos, operadores y/o funciones (del estilo al main), como por ejemplo:

```
longitud = 2 * pi* radio;
```

- Un **bloque de sentencias** es un conjunto de sentencias agrupadas entre llaves { }.
- Como se verá más adelante, los bloques de sentencias se utilizan para declarar:
  - a) Sentencias agrupadas bajo un nombre.
  - b) Sentencias que se ejecutarán un número de veces o que lo harán en función de una condición.

```
#include <iostream>
using namespace std;

#define pi 3.1415926

int main()
{
    float longitud;
    float radio;

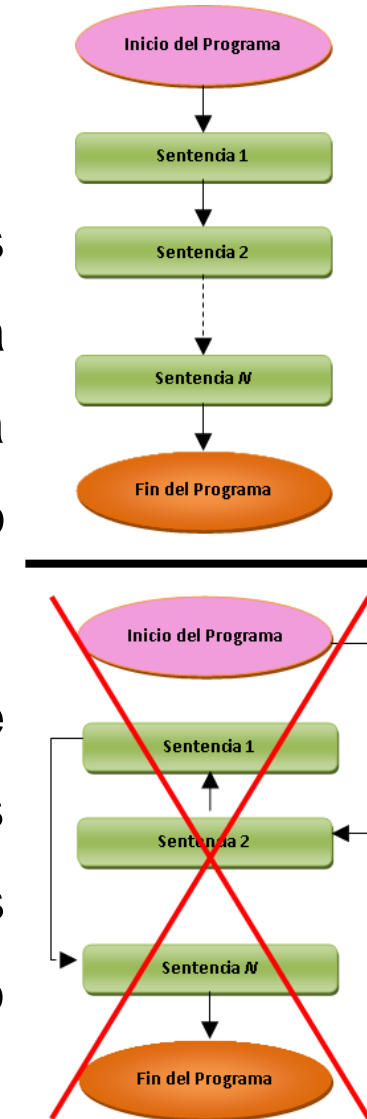
    cout << "Introduce el radio: ";
    cin >> radio;

    longitud = 2 * pi* radio;

    cout << "La longitud es: "
         << longitud << endl;
    return 0;
}
```

## 2. Sentencias Secuenciales.

- Los programas tienen un **inicio** y un **final** establecido.
- Desde el inicio de un programa hasta su final, sus sentencias son ejecutadas desde la primera hasta la última de manera secuencial, es decir, una sentencia no empezará a ser ejecutada si la anterior no ha sido terminada.
- Este orden secuencial no puede ser alterado aunque localmente podrá haber condiciones y/o repeticiones de sentencias o bloques de sentencias que nos permitan tener distintas líneas de ejecución pero siempre de manera determinista.



### 3. Sentencias Condicionales.

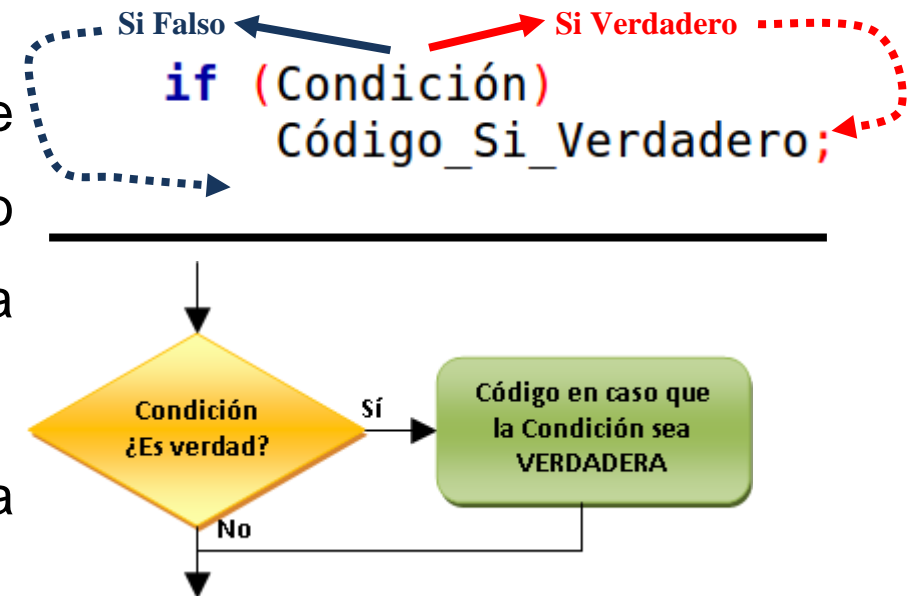
- Las sentencias condicionales se utilizan para ejecutar determinadas Sentencias o bloques de Sentencias en función de una condición.
- Las sentencias condicionales que nos ofrece el lenguaje son:

**if**, **if-else**, el operador **?**, **switch**.

#### I. Sentencia **if**

- Solo si la **Condición** es verdadera se ejecutará el código asociado y en caso de que sea falsa pasará a ejecutar la siguiente instrucción.

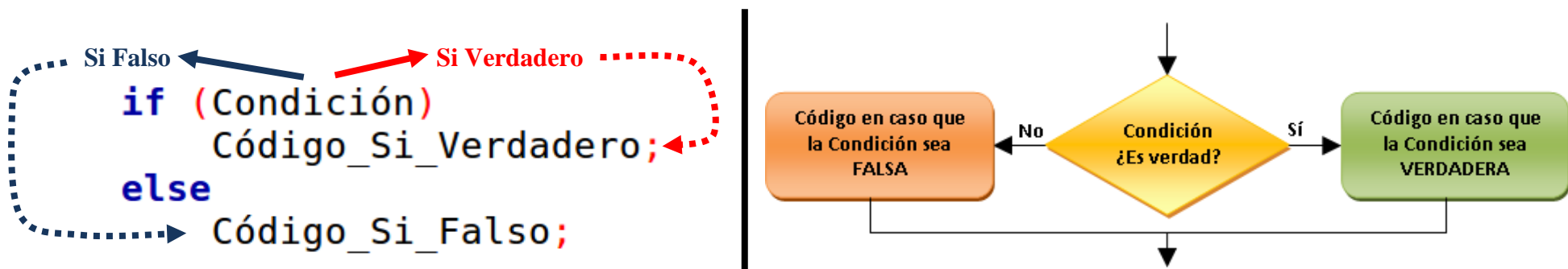
**Código\_Si\_Verdadero** puede ser una sentencia o un bloque de sentencias.



### 3. Sentencias Condicionales.

#### II. Sentencia if-else

- Si la **Condición** es verdadera se ejecutará el código correspondiente de la parte de verdad situado entre las palabras clave **if** y **else**. Si es falsa se ejecutará el código correspondiente a la parte de falsedad que va situado después de la palabra clave **else**.



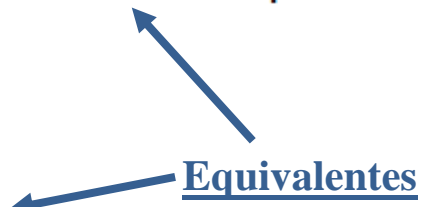
**Código\_Si\_Verdadero** y **Código\_Si\_Falso** pueden ser una sentencia y/o un bloque de sentencias.

### 3. Sentencias Condicionales.

#### III. Operador ?

- Determinadas instrucciones del tipo **if-else** pueden ser abreviadas con el operador interrogación (?) de la siguiente forma:

```
if (Condición)
    Variable = Expresión1;
else
    Variable = Expresión2;
```

Variable = Condición ? Expresión1 : Expresión2; 

Ejemplos de **if**, **if-else** y operador **?**:

```
float x,y;
cout << "Introduce valor:"
cin >> x;
if (x < 100)
    y = x + 50;
else
    y = x - 60;
```

```
int v;
cout << "Introduce valor:"
cin >> v;
if (v < 0)
    cout << v << " es negativo";
```

```
float x,y;
cout << "Introduce valor:"
cin >> x;
y = x < 100 ? x + 50 : x - 60;
```



### 3. Sentencias Condicionales.

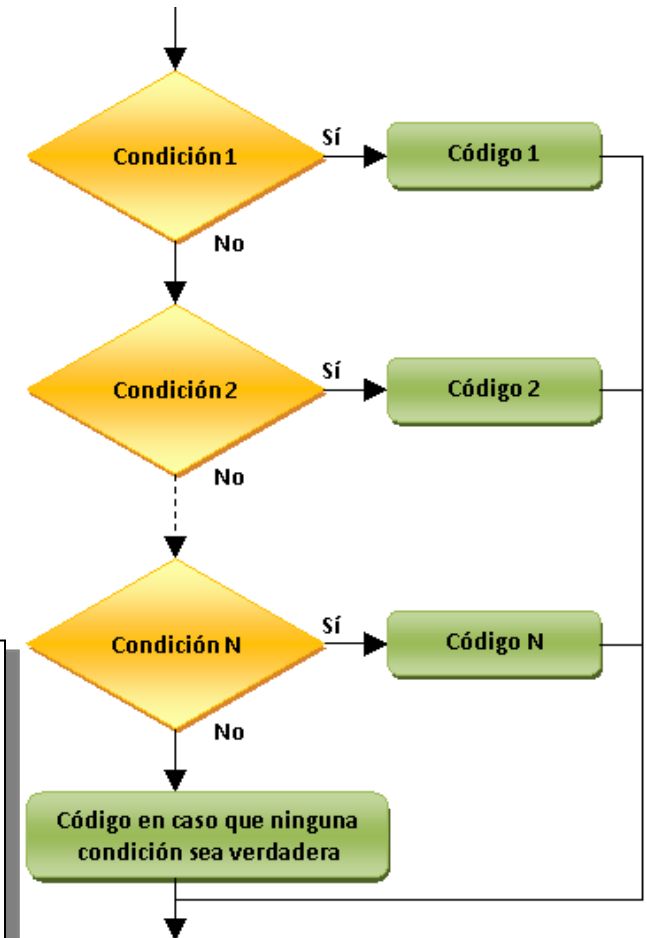
#### IV. Sentencias anidadas if-else

- El código asociado a la parte de verdad y falsedad en una sentencia **if-else** puede a su vez estar compuesto por más sentencias **if-else**.

#### Equivalentes

```
int x;  
cout << "Introduce valor:"  
cin >> x;  
if (x > 150)  
    cout <<x<<" es mayor de 150";  
else  
    if (x >= 100)  
        cout <<x<<" está entre 100 y 150";  
    else  
        if (x > 50)  
            cout <<x<<" está entre 51 y 99";  
        else  
            cout <<x<<" es menor de 50";
```

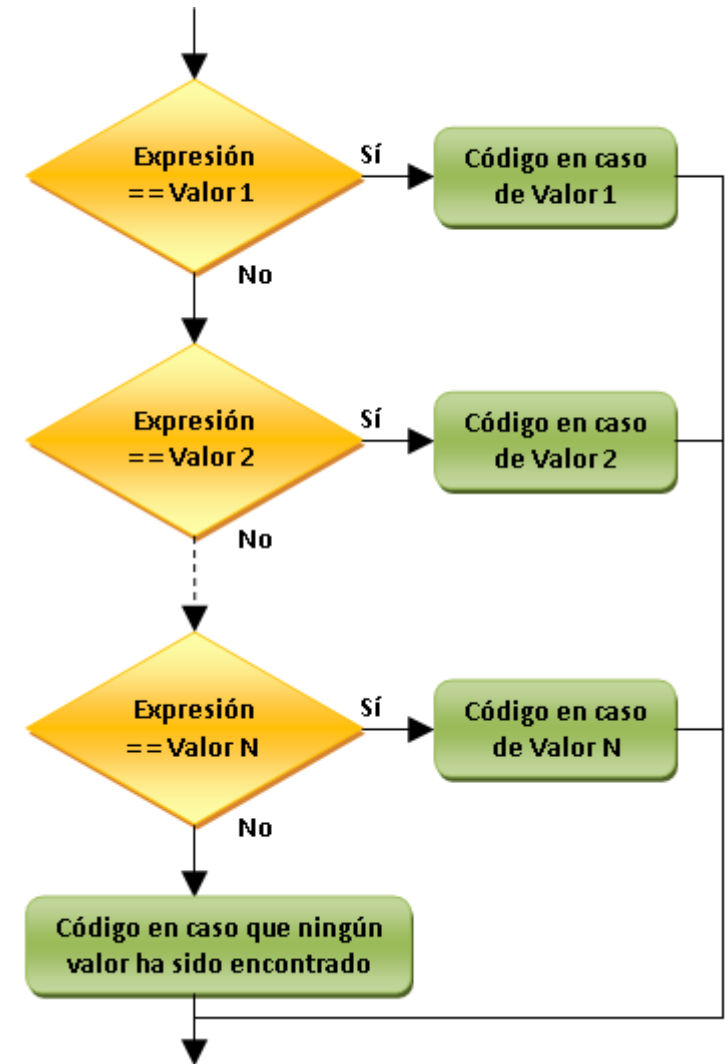
```
int x;  
cout << "Introduce valor:"  
cin >> x;  
if (x >= 100)  
    if (x > 150)  
        cout <<x<<" es mayor de 150";  
    else  
        cout <<x<<" está entre 100 y 150";  
else  
    if (x > 50)  
        cout <<x<<" está entre 51 y 99";  
    else  
        cout <<x<<" es menor de 50";
```



### 3. Sentencias Condicionales.

#### V. Sentencia `switch`

- Se utiliza en los casos en los que el resultado de una expresión deba compararse con diferentes valores.
- Permite varios formatos lo que le permite ser muy flexible.
- En su forma más sencilla, es similar a un conjunto de sentencias anidadas `if-else`.
- La diferencia con respecto a las sentencias `if-else` es que la expresión sólo se compara con valores enteros.



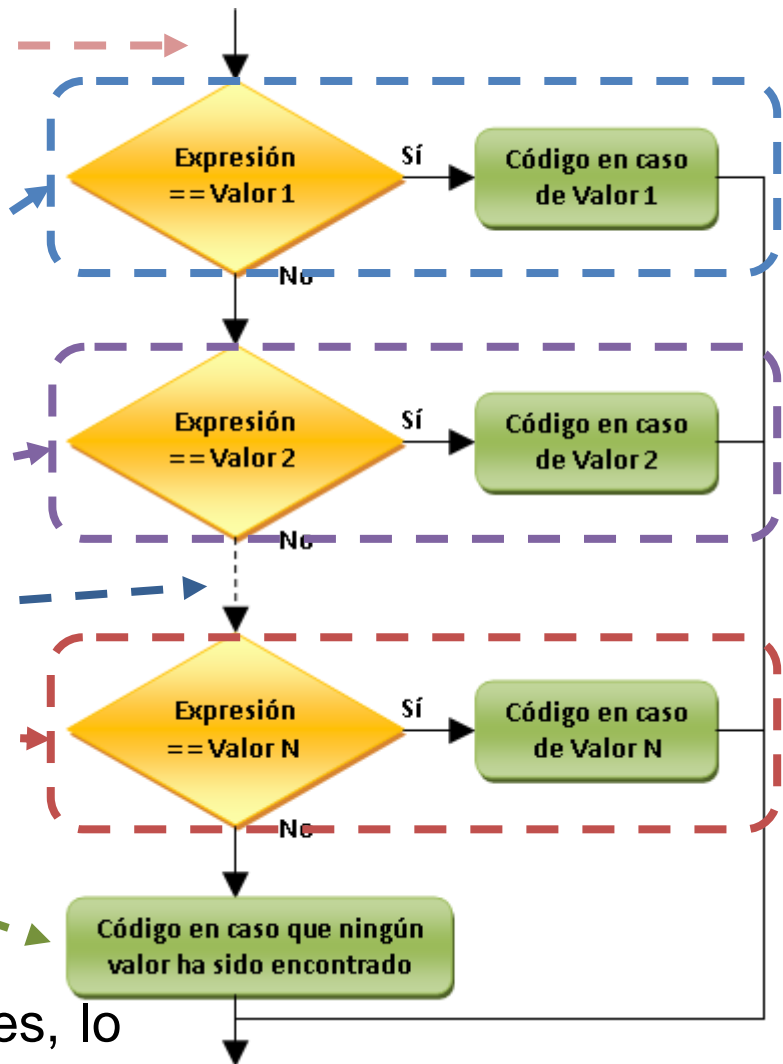
### 3. Sentencias Condicionales.

#### V. Sentencia switch

El formato básico es:

```
switch (Expresión)
{
    case valor1: Operaciones1;
                break;
    case valor2: Operaciones2;
                break;
    case valor1: Operaciones1;
                break;
    case valorN: OperacionesN;
                break;
    default: Otras_Operaciones;
}
```

Tanto **break** como **default** son opcionales, lo que permite tener distintos comportamientos.



### 3. Sentencias Condicionales.

#### V. Sentencia `switch`

- En caso de omitir un **`break`**, hace que una vez ejecutado el código asociado a un valor, continuará ejecutando el código asociado al resto de valores hasta encontrar un **`break`**.
- Si no existe la opción **`default`**, en caso de no encontrar un valor igual a la expresión, no se ejecutará nada y pasará a la siguiente sentencia del programa a continuación de la sentencia **`switch`**.

```
int vble, res=0;
cin >> vble; //Si es 1

switch (vble)
{
    case 1: res = res + 1;
    case 2: res = res * 2;
            res = res + 4;
            break;
    default: res = 3;
}
cout << "Salida = " << res;
```

Ejecuta

res = 6

```
int vble, res=0;
cin >> vble; //Si es 5

switch (vble)
{
    case 1: res = res + 1;
    case 2: res = res * 2;
            res = res + 4;
            break;
}
cout << "Salida = " << res;
```

No se Ejecuta Nada

res = 0

### 3. Sentencias Condicionales.

#### V. Sentencia `switch`

```
int vble, res=0;
cin >> vble; //Si es 2

switch (vble)
{
    case 1: res = res + 1;
    case 2: res = res * 2;
            res = res + 4;
            break;
}
cout << "Salida = " << res;
```

Ejecuta

res = 4

```
int vble, res=0;
cin >> vble; //Si es 5

switch (vble)
{
    case 1: res = res + 1;
    case 2: res = res * 2;
            res = res + 4;
            break;
    default: res = 3;
}
cout << "Salida = " << res;
```

Ejecuta

res = 3

#### Ejemplo

```
float Result, V1, V2;
int Operacion;

cout << "Introduce valor1 y Valor2:";
cin >> V1 >> V2;
cout << "Introduce 1.- Sumar\n2.- Restar\n"
      << "3.- Multiplicar\n4.- Dividir:";
cin >> Operacion;
switch (Operacion)
{
    case 1: Result = V1 + V2;
            break;
    case 2: Result = V1 - V2;
            break;
    case 3: Result = V1 * V2;
            break;
    case 4: Result = V1 / V2;
            break;
    default: cout << "Operación Imposible";
             Result = 0;
}
cout << "Resultado = " << Result;
```

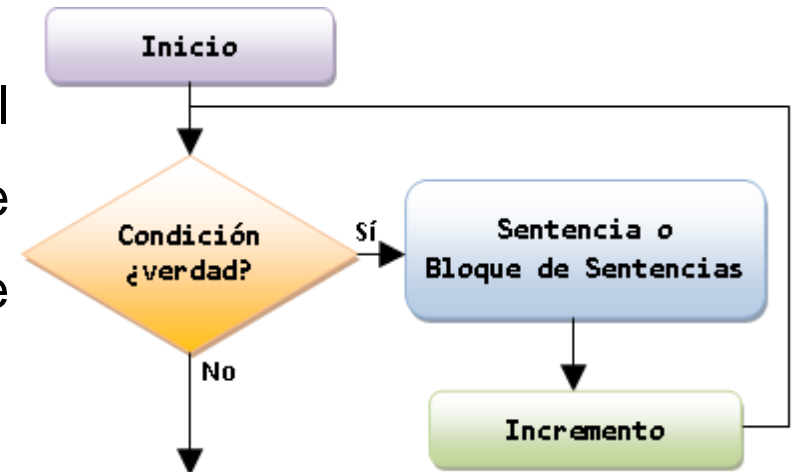
## 4. Sentencias Iterativas.

- Las sentencias iterativas (también conocidas como bucles) se utilizan cuando se necesita ejecutar más de una vez una sentencia o bloque de sentencias.
- Existen tres tipos de sentencias iterativas **for**, **while** y **do-while**.

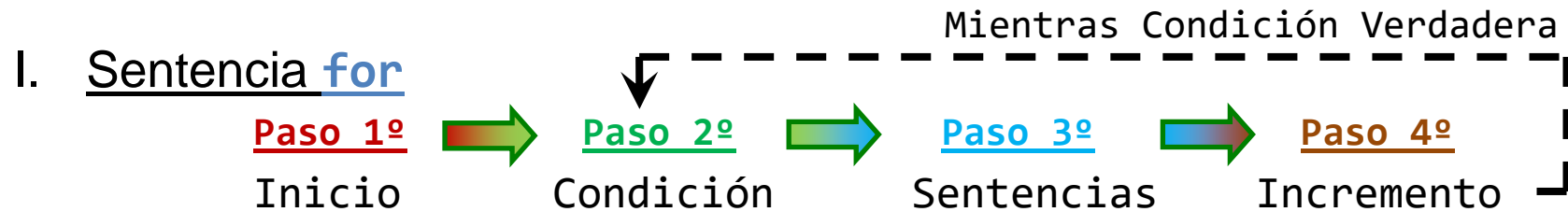
### I. Sentencia **for**

- Se utiliza en caso de conocer el número de veces que se ha de ejecutar una sentencia o bloque de sentencias.

**for** (Paso 1º; Paso 2º; Paso 4º)  
Sentencia\_o\_Bloque\_de\_Sentencias; Paso 3º



## 4. Sentencias Iterativas.



- El **Inicio** se utiliza para declarar la variable que controla el bucle.
- La **Condición** controla el bucle mediante la variable del bucle. Si es verdadera continua ejecutando las sentencias del bucle.
- El **Incremento** se utiliza para modificar la variable del bucle.

```
int Suma=0;
for (int i=1; i<=10; i++)
    Suma = Suma + i*i;
cout << "Suma Total: " << Suma;
```

Soluciones Idénticas

```
int Suma=0, i=1;
for (; i<=10;)
{
    Suma = Suma + i*i;
    i = i + 1;
}
cout << "Suma Total: " << Suma;
```

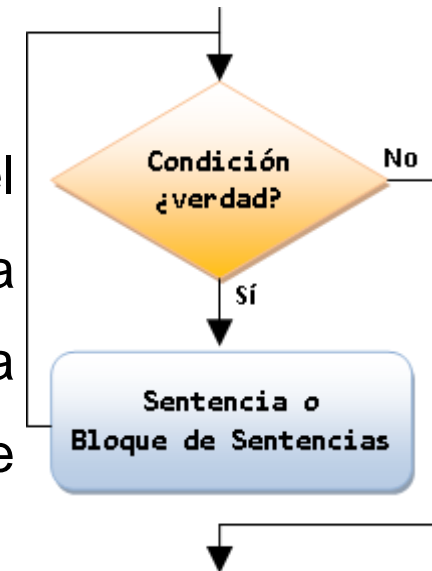
```
for (int i=1; i<=10; i++)
    for (int j=1; j<=10; j++)
        cout << i << "x" << j << "=" << i * j;
```

Bucles Anidados

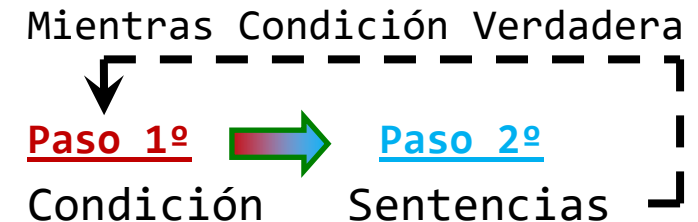
## 4. Sentencias Iterativas.

### II. Sentencia while

- La sentencia **while** se utiliza en el caso en el que el número de veces que se ha de ejecutar una sentencia o bloque de sentencias depende de una condición y por lo tanto se desconoce exactamente el número de veces que el bucle se va a repetir.
- El bucle se ejecutará mientras que la condición sea verdadera y solo parará cuando ésta sea falsa. En el caso que la condición sea inicialmente falsa no se ejecutará ninguna sentencia.



```
while (Paso 1º Condición) Paso 2º  
    Sentencia_o_Bloque_de_Sentencias;
```





## 4. Sentencias Iterativas.


### II. Sentencia while

```
int Suma=0, i=1, nveces;
cout << "Número de Veces: ";
cin >> nveces;
while (i < nveces)
{
    Suma = Suma + i;
    i = i + 1;
}
cout << "Suma total: " << Suma;
```

- Dentro del bucle debe existir una sentencia que en un determinado momento cambie el valor de verdad de la condición, ya que en caso contrario se producirá un bucle infinito bloqueando el programa.

```
int Suma=0, Vini, Vfin;
cout << "Introduce Fin e Inicio: ";
cin >> Vfin >> Vini;
while (Vfin >= Vini)
{
    cout << "Sumando " << Vfin << endl;
    Suma = Suma + Vfin;
    Vfin = Vfin - 1;
}
cout << "Suma total: " << Suma;
```

```
int Suma=0, Vini, Vfin;
cout << "Introduce Fin e Inicio: ";
cin >> Vfin >> Vini;
while (Vfin >= Vini)
{
    cout << "Sumando " << Vfin << endl;
    Suma += Vfin;
}
cout << "Suma total: " << Suma;
```



#### Bucle Infinito

La variable `Vfin` no se modifica en el bucle

## 4. Sentencias Iterativas.

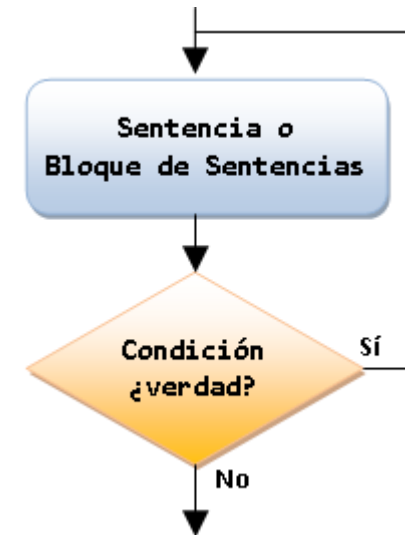
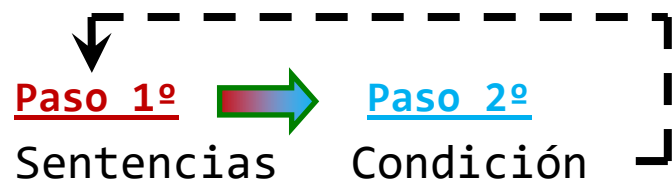
### III. Sentencia do-while

- Esta sentencia es igual a la sentencia **while** salvo que siempre ejecuta **al menos una vez** la sentencia o bloque de sentencias que contiene entre la palabras clave **do** y **while**.

**do**

```
{ Paso 1º
  Sentencia o Bloque de Sentencias;
} while (Condición); Paso 2º
```

Mientras Condición Verdadera



```
float Suma=0, Valor;
do
{
    cout << "Introduce Valor: ";
    cin >> Valor;
    if (Valor >= 0)
        Suma = Suma + Valor;
} while (Valor >= 0);
cout << "Suma Total: " << Suma;
```

## 4. Sentencias Iterativas.

```
int main()
{
    int opc;
    do
    {
        cout << "**** MENÚ PRINCIPAL ****\n";
        cout << "  1.- Leer Datos\n";
        cout << "  2.- Guardar Datos\n";
        cout << "  3.- Salir\n";
        cout << "Elige opción: ";
        cin >> opc;
        switch (opc)
        {
            case 1: //Código que Lee datos
                    break;
            case 2: //Código que Guarda datos
                    break;
            case 3: //Código opcional para salir
                    break;
            default: cout << "Opción incorrecta\n";
        }
    } while (opc != 3);
    return 0;
}
```

### Programa con un Menú

Inicio del bucle

Muestra el Menú Principal

Solicita por teclado una opción.

Sentencia Switch para seleccionar el código de cada opción.

Condición de repetición del bucle

## 5. Macros de Sentencias.

- Las macros son una utilidad que el compilador de c/c++. Éstas empiezan con el símbolo # y no van dirigidas al microprocesador sino el propio compilador para que altere su forma de compilar.
- Existen un conjunto estándar de macros que aparecen en todos los compiladores, aunque algunas compañías software incorporan adicionalmente otras en sus compiladores. Un ejemplo de macros son:
  - a) **#include** elimina la línea donde aparece e incluye el contenido del fichero cuyo nombre aparece entre < y >.
  - b) **#define** sustituye un texto por otro cada vez que se lo encuentra en el código, es como un cortar y pegar pero con más potencia.

## 5. Macros de Sentencias.

- Las macros más comunes son:

<code>#define</code>	<code>#if</code>	<code>#endif</code>	<code>#undef</code>
<code>#error</code>	<code>#else</code>	<code>#ifndef</code>	<code>#line</code>
<code>#include</code>	<code>#elif</code>	<code>#ifnndef</code>	<code>#pragma</code>

- Las macros más utilizadas tiene el siguiente formato:

a) `#include <Nombre_Librería>` donde `Nombre_Librería` es el nombre del fichero del compilador que tiene toda la información para incluir la librería. `#include <iostream>`

b) `#define TCortar TPegar` donde `TCortar` es el texto que buscará el compilador y lo sustituirá por el texto `TPegar`.

```
#define Inf -30
#define Max Inf+60
```