

# **ESTRUCTURAS DE CONTROL**

# Metodología de Programación Estructurada

## Programación estructurada:

Metodología que consiste en escribir programas con las siguientes reglas:

- El programa tiene un **diseño modular**
- Los módulos son diseñados de **modo descendente**
- Cada módulo se codifica utilizando las **tres estructuras de control básicas:**

**Secuencia,**

**Selección y**

**Repetición**

# Metodología de Programación Estructurada

## Teorema fundamental de la PE (Böhm y Jacopini, 1966):

“**Todo programa** propio puede ser escrito usando sólo las estructuras de control básicas: secuencia, selección y repetición”

### **Propio:**

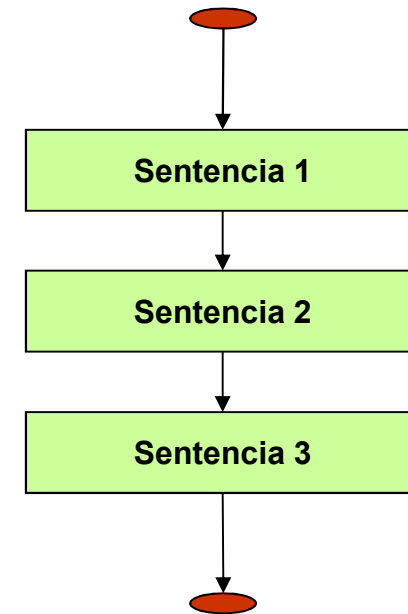
- Tiene un único punto de entrada y un único punto de salida;
- Existen caminos desde la E a la S que se pueden seguir y que pasan por todas las partes del programa;
- Todas las instrucciones son ejecutables y no hay bucles infinitos.

# ESTRUCTURAS DE CONTROL

- **Controlan el flujo de ejecución** de las acciones de un programa
- **Permiten combinar instrucciones o sentencias individuales** en una simple unidad lógica con un único punto de entrada y un único punto de salida
- Se clasifican en **tres tipos**:
  - ✓ Secuencia (Composición secuencial)
  - ✓ Selección (Composición alternativa), y
  - ✓ Repetición (Composición iterativa).

# SECUENCIA

- ❑ Conjunto de sentencias que se ejecutan una luego de la otra (flujo secuencial).



- ❑ En C++: conjunto de sentencias que finalizan con ';' y están encerradas entre llaves.

```
{  
    sentencia-1;  
    sentencia-2;  
    .....  
    sentencia-n;  
}
```

# SECUENCIA (Ejemplos)

**Ejemplo 1a:** Se ingresan las notas de 3 parciales y se muestra la nota promedio.

```
int main() {  
    int n1, n2, n3;  
    cout << "Ingrese las notas de los 3 parciales: " << endl;  
    cin >> n1 >> n2 >> n3;  
    cout << endl << "La nota promedio es: " << (n1+n2+n3)/3;  
    return 0;  
}
```

```
Ingrese las notas de los 3 parciales:  
10  
7  
4  
  
La nota promedio es: 7
```

**Ejemplo 2a:** Se ingresan las coordenadas de un punto del plano y se debe informar la distancia al origen.

```
int main() {  
    float x,y;  
    cout << "Ingrese las coordenadas: " << endl;  
    cin >> x >> y;  
    cout << "La distancia de " << x << " y " << y << " al origen es: ";  
    cout << sqrt(x*x+y*y);  
    return 0;  
}
```

Usar  
<cmath>

```
Ingrese las coordenadas:  
2.5  
4.7  
La distancia de 2.5 y 4.7 al origen es: 5.32353
```

# BLOQUE

- Un bloque es una unidad de ejecución mayor que la sentencia, y permite agrupar una secuencia de sentencias como una unidad.
- Para ello, formaremos un bloque delimitando entre dos llaves la secuencia de sentencias que agrupa.
- Es posible anidar bloques, es decir, se pueden definir bloques dentro de otros bloques.

```
int main()
{
    <sentencia_1> ;
    <sentencia_2> ;
    {
        <sentencia_3> ;
        <sentencia_4> ;
        . . .
    }
    <sentencia_n> ;
}
```

# DECLARACIONES GLOBALES Y LOCALES

- Se pueden declarar identificadores en diferentes bloques, pudiendo entrar en conflicto unos con otros, por lo que debemos conocer las reglas utilizadas en el lenguaje C++ para decidir a qué identificador nos referimos en cada caso.
- Hay dos clases de declaraciones: globales y locales.
- **Entidades globales** son aquellas que han sido definidas **fuera** de cualquier bloque. Su ámbito de visibilidad comprende desde el punto en el que se definen hasta el final del fichero. Se crean al principio del programa y se destruyen al finalizar éste.
- **Entidades locales** son aquellas que se definen **dentro** de un bloque. Su ámbito de visibilidad comprende desde el punto en el que se definen hasta el final de dicho bloque. Se crean en el punto donde se realiza la definición, y se destruyen **al finalizar el bloque**. Normalmente serán variables locales, aunque también es posible declarar constantes localmente.



# DECLARACIONES GLOBALES Y LOCALES

```
#include <iostream>
using namespace std;
const double EUR_PTS = 166.386; //Declaración de constante GLOBAL

int main() {
    cout << "Introduce la cantidad (en euros): ";
    double euros; //Declaración de variable LOCAL
    cin >> euros;
    double pesetas = euros * EUR_PTS; //Declaración de variable LOCAL
    cout << euros << " Euros equivalen a " << pesetas << " Pts" << endl;
    return 0;
}
```

```
Introduce la cantidad (en euros): 147
147 Euros equivalen a 24458.7 Pts
```

```
<< El programa ha finalizado: codigo de salida: 0 >>
<< Presione enter para cerrar esta ventana >>
```

# DECLARACIONES GLOBALES Y LOCALES

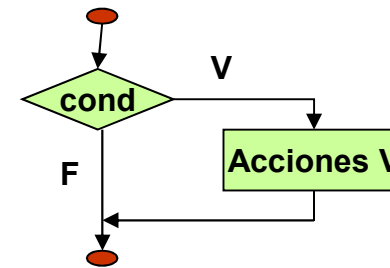
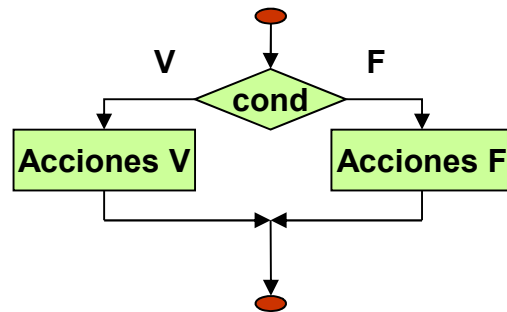
- En caso de tener declaraciones de **diferentes entidades con el mismo identificador en diferentes niveles de anidamiento**, la entidad visible será aquella que se encuentre declarada en el bloque de nivel de anidamiento más interno.
- La **declaración** más interna **oculta** a aquella más externa.

```
int main() {  
    int x = 3;  
    int z = x * 2;    // x es vble de tipo int con valor 3  
    {  
        double x = 5.0;  
        double n = x * 2;    // x es vble de tipo double con valor 5.0  
    }  
    int y = x + 4;    // x es vble de tipo int con valor 3  
    return 0;  
}
```

# **Estructuras de Control de Selección**

# SELECCIÓN: SENTENCIA IF

- ❑ Se evalúa una expresión booleana y según sea verdadera o falsa, se ejecuta una acción u otra; por cualquier caso la ejecución continúa con la sentencia siguiente al if. (**Selección completa**)
- ❑ Si sólo se desea realizar una acción en caso que la expresión booleana sea verdadera y no hay acción por el falso, se trata de una **Selección incompleta**



- En C++:

**if (expresión) acción1;**  
**else acción2;**

**if (expresión) acción1;**

Las acciones pueden ser una sentencia que termina en ';' o un grupo de sentencias encerradas entre llaves.

# SELECCIÓN INCOMPLETA: Ejemplos

**Ejemplo 1b:** Se ingresan las notas de 3 parciales y se debe indicar la nota final, siendo ésta el promedio ajustado (si está entre 6,3 y 6,5, se considera 6,5).

```
int main(){
    int n1, n2, n3;
    float prom;
    cout << "Ingrese las notas de los 3 parciales: " << endl;
    cin >> n1 >> n2 >> n3;
    prom = (float)(n1+n2+n3)/3;
    if (prom >= 6.3 && prom <= 6.5)
        prom = 6.5;
    cout << "La nota promedio es: " << prom;
    return 0;
}
```

```
Ingrese las notas de los 3 parciales:
10
7
4
La nota promedio es:7
```

```
Ingrese las notas de los 3 parciales:
10
5
4
La nota promedio es: 6.5
```

# SELECCIÓN INCOMPLETA: Ejemplos

**Ejemplo 2b:** Se ingresan las coordenadas de un punto del plano y se debe informar si se encuentra en uno de los ejes coordenados.

```
int main() {  
    float x,y;  
    cout << "Ingrese las coordenadas: " << endl;  
    cin >> x >> y;  
    if ((x == 0) || (y==0))  
        cout << "El punto (" << x << "," << y << ") esta sobre un eje del sistema"  
        << endl;  
    return 0;  
}
```

```
Ingrese las coordenadas:  
2.7  
0  
El punto (2.7,0) esta sobre un eje del sistema
```

```
Ingrese las coordenadas:  
1.3  
6.8
```

# SELECCIÓN COMPLETA: Ejemplos

**Ejemplo 1c:** Se ingresan las notas de 3 parciales y se debe indicar si ha aprobado (se aprueba con promedio mayor o igual a 6,5) o debe realizar un recuperatorio, y en ambos casos se muestra la nota.

```
int main() {  
    int n1, n2, n3;  
    float prom;  
    cout << "Ingrese las notas de los 3 parciales: " << endl;  
    cin >> n1 >> n2 >> n3;  
    prom = (float) (n1+n2+n3)/3;  
    if (prom >= 6.3 && prom <= 6.5)  
        prom = 6.5;  
    if (prom >= 6.5)  
        cout << "Ha aprobado el curso" << endl;  
    else  
        cout << "Debe realizar un examen recuperatorio" << endl;  
    cout << "Su nota promedio es: " << prom;  
    return 0;  
}
```

```
Ingrese las notas de los 3 parciales:  
10  
10  
9  
Ha aprobado el curso  
Su nota promedio es: 9.66667
```

```
Ingrese las notas de los 3 parciales:  
5  
6  
6  
Debe realizar un examen recuperatorio  
Su nota promedio es: 5.66667
```

# SELECCIÓN COMPLETA: Ejemplos

**Ejemplo 2c:** Se ingresan las coordenadas de un punto del plano.

Validar que no sea el origen y mostrar la distancia al origen.

```
int main() {  
    float x,y;  
    cout << "Ingrese las coordenadas: " << endl;  
    cin >> x >> y;  
    if (x == 0 && y==0)  
        cout << endl << "Incorrecto, ha ingresado el origen";  
    else  
        cout << endl << "La distancia de " << x << "," << y << " al origen es: "  
        << sqrt(x*x + y*y);  
    return 0;  
}
```

Ingrese las coordenadas:

1  
0

La distancia de 1,0 al origen es: 1

Ingrese las coordenadas:

0  
0

Incorrecto, ha ingresado el origen



# SELECCIÓN ANIDADA

Una sentencia if es anidada cuando la acción de la rama verdadera y/o la rama falsa, **son también una sentencia if**.

- En C++:

```
if (exp1)
    acción 1;
else
    if (exp2)
        acción2;
    else
        acción3;
```

*Se abre la rama falsa*

```
if (exp1)
    if (exp2)
        acción 1;
    else
        acción 2;
else
    acción 3;
```

*Se abre la rama verdadera*

- El if exterior o los interiores pueden ser incompletos. Ejemplos:

```
if (exp1)
{
    if (exp2)
        acción2;
    else
        acción3;
}
```

*El if exterior es incompleto*

```
if (exp1)
{
    if (exp2)
        acción 1;
}
else
    acción 3;
```

*El interior es incompleto*

- Las llaves no son requeridas, pero se deben agregar para indicar al compilador C++ la asociación de los if-else.

# SELECCIÓN ANIDADA: Ejemplos

**Ejemplo 3:** Se ingresa un número, se debe incrementar un contador general de números tratados (ContGral) y también se debe incrementar el contador correspondiente: ContPos, ContNul, ContNeg

```
{  
    cin >> n;  
    if (n > 0)  
        ContPos ++;  
    else  
        if (n < 0)  
            ContNeg ++;  
        else  
            ContNul ++;  
    ContGral ++;  
}
```

**Que diferencia hay con esta otra forma?**

```
cin >> n;  
if (n > 0)  
    ContPos ++;  
if (n < 0)  
    ContNeg ++;  
if (n==0)  
    ContNul ++;  
ContGral ++;
```

Las sangrías no son requeridas,  
pero aportan claridad.

# SELECCIÓN ANIDADA: Ejemplos

**Ejemplo 1d:** Se ingresan las notas de 2 parciales y se debe indicar si ha aprobado o debe realizar un recuperatorio (se aprueba con promedio mayor o igual a 6,5). En ambos casos se muestra la nota. Si debe recuperar, y en uno de los parciales sacó 7 o más, solo recupera la Parte no aprobada (A o B).

```
int main(){
    int n1, n2;
    float prom;
    cout << "Ingrese las notas de los 2 parciales: " << endl;
    cin >> n1 >> n2;
    prom = (float)(n1+n2)/2;
    if (prom >= 6.5)
        cout << "Ha aprobado el curso" << endl;
    else
        if (n1 >= 7)
            cout << endl << "Debe realizar un recuperatorio de la parte B";
        else
            if (n2 >= 7)
                cout << endl << "Debe realizar un recuperatorio de la parte A";
            else cout << endl << "Debe realizar un recuperatorio global";
    cout << endl << "Su nota promedio es: " << prom;
    return 0;
}
```

Ingrese las notas de los 2 parciales:

7  
2

Debe realizar un recuperatorio de la parte B  
Su nota promedio es: 4.5

# SELECCIÓN ANIDADA: Ejemplos

- Ejemplo 1e: Idem 1d, pero si aprobó con promedio mayor a 7, se le informa que está becado para el siguiente curso.

```
int main(){
    int n1, n2;
    float prom;
    cout << "Ingrese las notas de los 2 parciales: " << endl;
    cin >> n1 >> n2;
    prom = (float)(n1+n2)/2;
    if (prom >= 6.5){
        cout << "Ha aprobado el curso" << endl;
        if (prom > 7)
            cout << "Ademas esta becado para el curso siguiente" << endl;
    }
    else
        if (n1 >= 7)
            cout << endl << "Debe realizar un recuperatorio de la parte B";
        else
            if (n2 >= 7)
                cout << endl << "Debe realizar un recuperatorio de la parte A";
            else cout << endl << "Debe realizar un recuperatorio global";
    cout << endl << "Su nota promedio es: " << prom;
    return 0;
}
```

Ver que ocurre si  
no se colocaran  
las llaves !!!

```
Ingrese las notas de los 2 parciales:
9
8
Ha aprobado el curso
Ademas esta becado para el curso siguiente
Su nota promedio es: 8.5
```

# SELECCIÓN ANIDADA: Ejemplos

- **Ejemplo 2d:** Dado un punto del plano informar sobre que eje o en qué cuadrante está.

```
int main() {
    float x,y;
    cout << "Ingrese las coordenadas: " << endl;
    cin >> x >> y;
    if (x == 0)
        cout << endl << "El punto " << x << "," << y << " esta sobre el eje y";
    else if (y == 0)
        cout << endl << "El punto " << x << "," << y << " esta sobre el eje x";
    else if (x > 0 && y > 0)
        cout << endl << "El punto " << x << "," << y << " esta en el 1er cuadrante";
    else if (x > 0 && y < 0)
        cout << endl << "El punto " << x << "," << y << " esta en el 4to cuadrante";
    else if (x < 0 && y > 0)
        cout << endl << "El punto " << x << "," << y << " esta en el 2do cuadrante";
    else
        cout << endl << "El punto " << x << "," << y << " esta en el 3er cuadrante";
    return 0;
}
```

# Comparación de if anidados y secuencias de if

- El ejemplo 2d usa una única sentencia if anidada. Otra alternativa es usar **una secuencia de if**:

```
int main() {  
    float x,y;  
    cout << "Ingrese las coordenadas: " << endl;  
    cin >> x >> y;  
    if (x == 0)  
        cout << endl << "El punto " << x << "," << y << " esta sobre el eje y";  
    if (y == 0)  
        cout << endl << "El punto " << x << "," << y << " esta sobre el eje x";  
    if (x > 0 && y > 0)  
        cout << endl << "El punto " << x << "," << y << " esta en el 1er cuadrante";  
    if (x > 0 && y < 0)  
        cout << endl << "El punto " << x << "," << y << " esta en el 4to cuadrante";  
    if (x < 0 && y > 0)  
        cout << endl << "El punto " << x << "," << y << " esta en el 2do cuadrante";  
    if (x < 0 && y < 0)  
        cout << endl << "El punto " << x << "," << y << " esta en el 3er cuadrante";  
    return 0;  
}
```

- Ambas son lógicamente equivalentes, pero ésta es menos EFICIENTE, ya que si alguna de las condiciones es verdadera, las siguientes condiciones se evalúan igualmente (para cada punto se realizan entre 1 y 6 evaluaciones).

# SELECCIÓN MÚLTIPLE: Sentencia SWITCH

- Se emplea para los casos en que se debe **seleccionar una de entre múltiples alternativas**, en forma exclusiva, basada en el valor de una variable o expresión denominada **selector o expresión de control**, la cual sólo puede ser **de tipo int o char**.
- El mecanismo se completa con la sentencia “**break**” que hace que siga la ejecución en la siguiente sentencia del programa en la que se está, en este caso la sentencia siguiente al switch.
- El switch es muy útil para transformaciones de **datos para salida** y para el **manejo de menú** en las interfases de los programas.

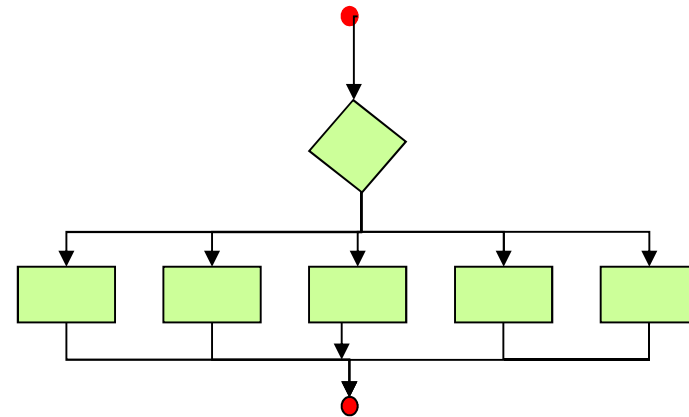


# Sentencia SWITCH

- En C++:

```
switch (selector) {  
    case etiqueta_1:  
        sentencias;  
        break;  
  
    case etiqueta_2:  
        sentencias;  
        break;  
  
    .....  
  
    default : /* opcional */  
        sentencias;  
        break;  
}
```

No se requieren { }



- Cada etiqueta es un **valor único, constante** (int o char) y debe tener un valor **diferente de los otros**.
- Se van evaluando las etiquetas y cuando alguna coincide con el valor del selector se ejecutan todas las sentencias asociadas.
- Si el valor del selector no es igual a ninguna etiqueta, no se ejecuta ninguna opción, a menos que se use el default.



# Sentencia SWITCH

- Ejemplo 3: Mostrar la carrera de UTN teniendo los valores K, M, C, E, I en la variable carrera de tipo char.

```
int main() {
    char carrera;
    cout << "Ingrese carrera (K, M, C, E o I): ";
    cin >> carrera;
    switch (carrera) {
        case 'K' :
            cout << " Ing. en Sistemas de Informacion ";
            break;
        case 'M' :
            cout << " Ing. Mecanica ";
            break;
        case 'C' :
            cout << " Ing. Civil ";
            break;
        case 'E' :
            cout << " Ing. Electrica ";
            break;
        case 'I' :
            cout << " Ing. Industrial ";
            break;
        default:
            cout << "No existe esa carrera";
    }
    return 0;
}
```

Ingrese carrera (K, M, C, E o I): M  
Ing. Mecanica

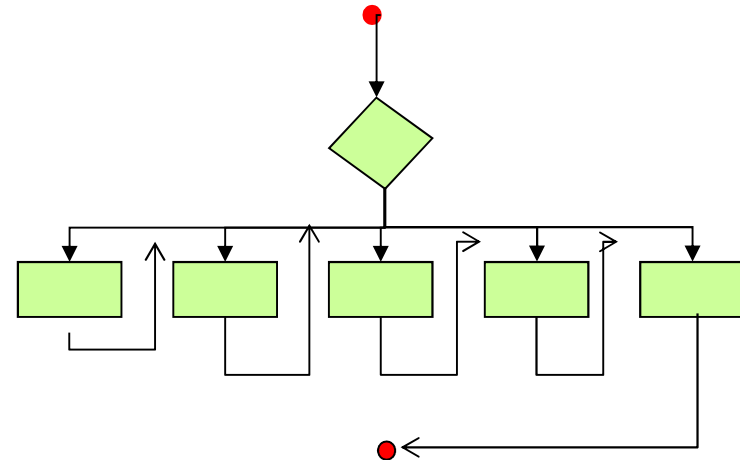
Ingrese carrera (K, M, C, E o I): L  
No existe esa carrera

# Sentencia SWITCH (sin break)

- Tiene un **comportamiento en cascada**.

- En C++:

```
switch (selector) {  
    case etiqueta_1:  
        sentencias;  
    case etiqueta_2:  
        sentencias;  
    .....  
    default : /* opcional */  
              sentencias;  
}
```


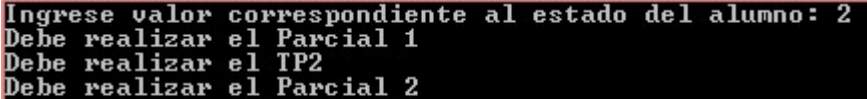


- Se van evaluando las etiquetas y cuando el valor del selector es igual a una de las etiquetas, se ejecutan todas las sentencias a partir de la que corresponde a ese selector.

## Sentencia SWITCH (sin break)

- Ejemplo 4: En una asignatura hay que realizar 4 evaluaciones que son correlativas. El estado de un alumno se maneja con un natural de 0 a 4 según las actividades que ya ha realizado. Se deben mostrar los mensajes correspondientes para un alumno conociendo su estado.

```
int main(){
    int estado;
    cout << "Ingrese valor correspondiente al estado del alumno: ";
    cin >> estado;
    switch (estado){
        case 0:
            cout << "Debe realizar la Evaluación inicial" << endl;
        case 1:
            cout << "Debe realizar el TP1" << endl;
        case 2:
            cout << "Debe realizar el Parcial 1" << endl;
        case 3:
            cout << "Debe realizar el TP2" << endl;
        case 4:
            cout << "Debe realizar el Parcial 2" << endl;
        }
    return 0;
}
```

- Si la variable *estado* tiene el valor 2  

# Sentencia SWITCH (con varios case)

- Está permitido **tener varias expresiones case** en una alternativa dada dentro de la sentencia switch, si se necesita.
- Ejemplo 5: Dado un valor dígito, indicar si es mayor, igual o menor que 3.

```
int main(){
    int digito;
    cout << "Ingrese valor digito: ";
    cin >> digito;
    switch (digito){
        case 0:
        case 1:
        case 2:
            cout << "Es menor que 3";
            break;
        case 3:
            cout << "Es igual a 3";
            break;
        default:
            cout << "Es mayor que 3" << endl;
    }
    return 0;
}
```

```
int main(){
    int digito;
    cout << "Ingrese valor digito: ";
    cin >> digito;
    switch (digito){
        case 0: case 1: case 2:
            cout << "Es menor que 3";
            break;
        case 3:
            cout << "Es igual a 3";
            break;
        default:
            cout << "Es mayor que 3" << endl;
    }
    return 0;
}
```

```
Ingrese valor digito: 1
Es menor que 3
```

```
Ingrese valor digito: 3
Es igual a 3
```

```
Ingrese valor digito: 8
Es mayor que 3
```

# Errores frecuentes de programación

- Usar `=` en lugar de `==` en las condiciones.
- Confundir la asociación de `else` con el `if` correspondiente (solución: **encerrar entre llaves**).
- Que no haya concordancia entre el tipo del selector y el tipo de las etiquetas, en el `switch`.
- Omitir el `break` cuando sea requerido.

# **Estructuras de Control Repetitivas**

# Estructuras Repetitivas

- En C++ hay **tres alternativas**:
  - ✓ **Bucle while** (test prebucle – bucle 0-n)
  - ✓ **Bucle for** (bucle exacto – N veces)
  - ✓ **Bucle do-while** (test postbucle – bucle 1-n)

# REPETICIÓN - CICLO - BUCLE

- Un **bucle** es una construcción que repite una sentencia, o una secuencia de sentencias, un número de veces. Se definen:
  - **Cuerpo del bucle**: la sentencia o grupo de sentencias que se repiten.
  - **Condición del bucle**: expresión lógica que controla la secuencia de repetición. El fin del bucle DEBE alcanzarse.
  - **Iteración**: Cada repetición del cuerpo del bucle.



# Ciclo WHILE

- **Bucle while - bucle pretest - bucle 0-n:**
  - ✓ La condición se evalúa **antes** de que se ejecute el cuerpo.
  - ✓ La ejecución del cuerpo se repite **mientras la condición del bucle sea verdadera** y termina cuando se hace falsa
  - ✓ Si la condición inicialmente es falsa **el cuerpo no se ejecutará** (el bucle se ejecuta cero o más veces)

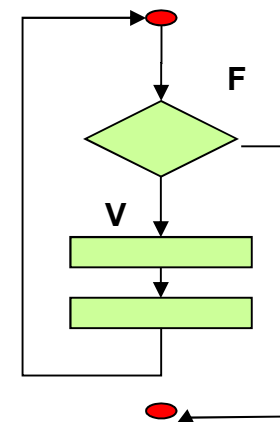
- **Formato en C++:**

```
while (condición_bucle)
{
    sentencia 1;
    .....
    sentencia n;
}
```

Los ( ) son obligatorios.

Cuerpo

Las { } pueden suprimirse si el bloque contiene una sola sentencia.



# Ciclo WHILE

- La condición del bucle evalúa una variable (variable de control del bucle).
- Los pasos son:
  - **Inicializar** la variable de control antes de entrar al ciclo while
  - **Comprobar** el valor de la variable de control antes de que comience la repetición del ciclo (denominada pasada o iteración)
  - **Actualizar** la variable de control durante cada iteración.

# Ciclo WHILE: Funcionamiento

1. Se **evalúa la condición\_bucle**
2. Si la **condición\_bucle es verdadera** (distinta de 0):
  - a. Se ejecuta la sentencia especificada en el cuerpo del bucle.
  - b. Se devuelve el control al paso 1.
3. Si la **condición\_bucle es falsa** (igual a 0):

El control se transfiere a la sentencia siguiente al bucle while.

Las sentencias del cuerpo del bucle se repiten mientras que la condición\_bucle sea verdadera.

Cuando la condición\_bucle es falsa se sale del bucle y se ejecuta la sentencia que viene después de la sentencia while, en el programa.

# Ciclo WHILE

- Ejemplo 1: Sumar los 10 primeros números naturales.

```
int main(){
    int numero = 1, suma = 0;
    while (numero <= 10){
        suma += numero;
        numero++;
    }

    cout << "La suma de los 10 primeros numeros naturales da: " << suma;
    return 0;
}
```

```
La suma de los 10 primeros numeros naturales da: 55
```

# Ciclo WHILE

- Ejemplo 2: Leer la cantidad de facturas emitidas en una empresa y a continuación los montos (valores decimales) de c/u e informar el total facturado y el mayor valor de una factura emitida.

```
int main() {
    int cant, contador;
    float sum=0, may, montofact;
    cout << "Ingrese cantidad de facturas a procesar: ";
    cin >> cant;
    sum = 0;
    may = 0;
    contador = 1;
    while (contador <= cant) {
        cout << "Ingrese monto de la factura " << contador << " : ";
        cin >> montofact;
        sum += montofact;
        if ( montofact > may) may = montofact;
        contador++;
    }
    cout << "El total facturado es: " << sum << endl;
    cout << "El mayor valor de factura es: " << may;

    return 0;
}
```

```
Ingrese cantidad de facturas a procesar: 3
Ingrese monto de la factura 1 : 150.74
Ingrese monto de la factura 2 : 359.14
Ingrese monto de la factura 3 : 83.46
El total facturado es: 593.34
El mayor valor de factura es: 359.14

<< El programa ha finalizado: codigo de salida: 0 >>
<< Presione enter para cerrar esta ventana >>
```

# Ciclo WHILE

## Cuidado con usos no adecuados:

- **Ciclo infinito:** se produce cuando la condición del bucle permanece verdadera, y no se hace falsa en ninguna iteración.

```
p=0;  
  while (p >=0) {  
    sum +=p;  
    p++;  
  }
```

```
p=0;  
  while (p < 12)  
    sum +=p;  
    p++;
```

El “;” al final de sum +=p; hace que el bucle termine ahí, aunque el indentado pueda dar la sensación de que el cuerpo del while contiene dos sentencias.

- **Cero iteraciones**

```
p=0;  
  while (p > 0) {  
    sum +=p;  
    p++;  
  }
```

# Ciclo WHILE con centinela

A veces no se conoce con exactitud cuantos datos se procesarán antes de comenzar la ejecución. Una forma de manejarlo es introducir como **último dato** un valor denominado *valor centinela*.

La condición del bucle comprueba cada dato y termina cuando se lee el valor centinela.

- Ejemplo 3: Ingresar las notas de parciales de los alumnos de un curso, finalizando con -1 y mostrar el promedio.

```
int main(){
    int nota, sum, contador;
    cout << "Ingrese la primera nota: ";
    cin >> nota;
    sum = 0;
    contador = 0;
    while (nota != -1)
    {
        sum += nota;
        contador++;
        cout << endl << "Ingrese la siguiente nota: ";
        cin >> nota;
    }
    if (contador > 0) cout << endl << "Promedio de notas: " << sum/contador;
    else cout << "No hay datos para hallar promedio";
    return 0;
}
```

Valor centinela: permite  
terminar el proceso del bucle

```
Ingrese la primera nota: 8
Ingrese la siguiente nota: 10
Ingrese la siguiente nota: 7
Ingrese la siguiente nota: -1
Promedio de notas: 8
```

## Ciclo WHILE con bandera

Se establece el indicador de control (**bandera**) a falso o verdadero, según corresponda, para que el ciclo se ejecute siempre por primera vez.

Mientras la condición de control sea verdadera se deben realizar las sentencias del ciclo y cuando se introduzca la condición de salida **se cambia el valor de la bandera** para que la próxima evaluación de la condición de control del ciclo sea falsa.

- Ejemplo 4: Solicitar al usuario que ingrese una letra minúscula.

```
int main(){
    int es_minus;
    char letra;
    es_minus = 0;
    while (! es_minus) {
        cout << endl << "Ingrese una letra minuscula: ";
        cin >> letra;
        es_minus = (('a' <= letra) && (letra <= 'z'));
    }
    return 0;
}
```

Valor bandera

```
Ingrese una letra minuscula: A
Ingrese una letra minuscula: Z
Ingrese una letra minuscula: b
```



# Ciclo FOR

- El **bucle for**: es un método para ejecutar un bloque de sentencias un **número fijo de veces**.

Las operaciones de control del bucle se sitúan en un solo sitio: la cabecera de la sentencia for.

- En C++ es aún más potente que en otros lenguajes. Su formato es:

```
for (Inicialización; Condición-Iteración; Incremento)
{
    sentencias....
}
```

Las { } pueden suprimirse si el bloque contiene una sola sentencia.

**Inicialización**: se inicializan las variables de control (simples o múltiples).

**Condición-Iteración**: expresión lógica que hace que el bucle realice las iteraciones mientras que la expresión sea verdadera.

**Incremento**: incremento o decremento de las variables de control.

# Ciclo FOR - Ejemplos

Se utiliza una **variable de control** que se inicializa a un **valor inicial** y en cada pasada, **antes de cada iteración**, se la compara con el **límite**.

1) **Mostrar los cuadrados de los primeros 10 números naturales.**

```
int main() {  
    int n;  
    for (n=1; n<=10; n++)  
        cout << n << " " << n*n << endl;  
    return 0;  
}
```

```
1 1  
2 4  
3 9  
4 16  
5 25  
6 36  
7 49  
8 64  
9 81  
10 100
```

# Ciclo FOR - Ejemplos

## 2) Obtener el factorial de n.

```
int main(){
    int n, factorial, x;
    cout << "Ingrese n: ";
    cin >> n;
    factorial = 1;
    for (x=2; x<=n; x++)
    {
        factorial = factorial * x;
    }
    cout << n << "! = " << factorial;

    return 0;
}
```

```
Ingrese n: 3
3! = 6
```

## 3) Leer y sumar 10 notas.

```
int main(){
    int nota, sum, x;
    sum = 0;
    cout << "Ingrese 10 notas: " << endl;
    for (x=1; x<=10; x++){
        cin >> nota;
        sum +=nota;
    }
    cout << "La suma de las notas es: " << sum;
    return 0;
}
```

```
Ingrese 10 notas:
9
7
6
8
10
5
9
7
10
10
La suma de las notas es: 81
```

# Ciclo FOR - Ejemplos

- Los **incrementos / decrementos** pueden ser enteros o reales, positivos o negativos.

## 4) Mostrar los números múltiplos de 10, menores que 100, ordenados de mayor a menor:

a)

```
int main() {  
    int n;  
    for (n=90; n>0; n-=10)  
        cout << n << endl;  
    return 0;  
}
```

n = n - 10

```
90  
80  
70  
60  
50  
40  
30  
20  
10
```

b)

```
int main() {  
    int n;  
    for (n=9; n>0; n--)  
        cout << n*10 << endl;  
    return 0;  
}
```

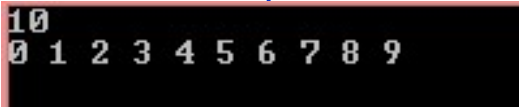
# Ciclo FOR

- Es posible y adecuado **declarar e inicializar la variable de control del bucle** en el **lugar de la inicialización**.
- En este caso especial, el ámbito de visibilidad de la variable de control del bucle es solamente hasta el final del bloque de la estructura for.
- **5) Ingresar un número natural y mostrar todos los naturales comprendidos entre 0 y el número ingresado:**

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

int main() {
    int n;
    cin >> n;
    for (int i=0 ; i < n ; ++i){
        cout << i << " ";
    }
    // i NO es visible aqui
    cout << endl;
    return 0;
}
```

Declaración e inicialización de la variable de control



```
10
0 1 2 3 4 5 6 7 8 9
```

## Ciclo FOR: Ejemplos con múltiples variables de control

6) **Mostrar todos los pares de números naturales que suman un valor MAX**

```
#define MAX 15

int main(){
    int i,j;
    for (i=0, j=MAX; i < j ; i++, j--)
        cout << "(" << i << "," << j << ")" << endl;
    return 0;
}
```

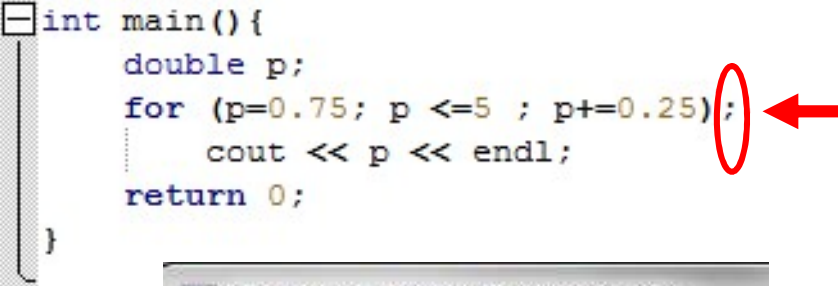
```
(0,15)
(1,14)
(2,13)
(3,12)
(4,11)
(5,10)
(6,9)
(7,8)
```

# Ciclo FOR

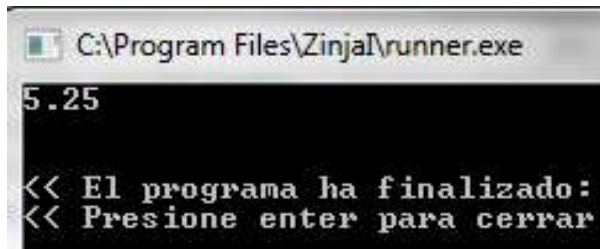
## Precauciones:

- **QUE TERMINE**: La inicialización, comprobación e incremento deben asegurar que la condición en algún momento sea falsa. Si dentro del cuerpo del for la **variable de control se modifica**, entonces esto no podrá asegurarse, por lo que esto se considera **mala práctica de programación**.

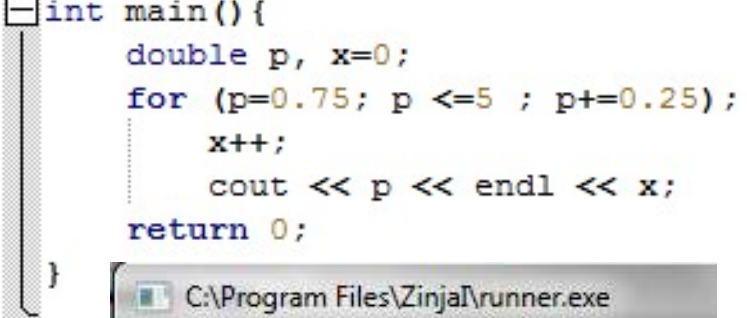
## ➤ CUIDADO CON LOS BUCLES VACÍOS:

1) 

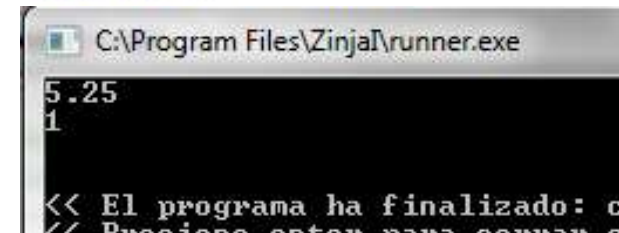
```
int main() {  
    double p;  
    for (p=0.75; p <=5 ; p+=0.25);  
        cout << p << endl;  
    return 0;  
}
```



```
C:\Program Files\Zinja\runner.exe  
5.25  
  
<< El programa ha finalizado:  
<< Presione enter para cerrar
```

2) 

```
int main() {  
    double p, x=0;  
    for (p=0.75; p <=5 ; p+=0.25);  
        x++;  
        cout << p << endl << x;  
    return 0;  
}
```



```
C:\Program Files\Zinja\runner.exe  
5.25  
1  
  
<< El programa ha finalizado: c  
<< Presione enter para cerrar
```

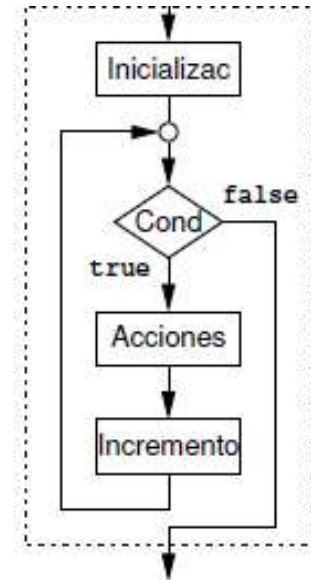
# Ciclo FOR

- La sentencia for puede ser vista como una **construcción especializada de la sentencia while**, pero con una sintaxis diferente para hacer más explícitos los casos en los que la iteración está controlada por los valores que toma una determinada variable de control, de tal forma que existe una clara inicialización y una clara modificación (incremento) de la variable de control, hasta llegar al caso final.

```
for ( <inicialización> ; <expresión_lógica> ; <incremento> ) {  
    <secuencia_de_sentencias> ;  
}
```

y es equivalente a:

```
<inicialización> ;  
while ( <expresión_lógica> ) {  
    <secuencia_de_sentencias> ;  
    <incremento> ;  
}
```





# Ciclo DO-WHILE

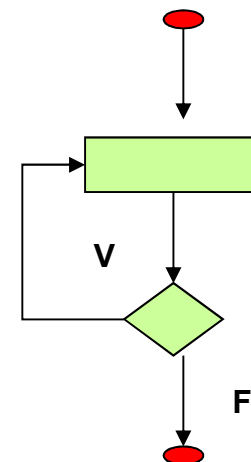
- **El bucle do-while:** se utiliza para especificar un bucle condicional que se ejecuta **al menos una vez** (ciclo 1-n veces).

Primero se ejecutan las sentencias, luego se evalúa la expresión, si es verdadera se repite la ejecución del ciclo y el proceso continúa hasta que sea falsa.

- **En C++:**

Las { } pueden suprimirse si el bloque contiene una sola sentencia.

```
do {  
    sentencias;  
} while (expresión);
```



# Ciclo DO-WHILE

**Ejemplo 7: Imprimir todas las letras mayúsculas.**

```
int main() {  
    char car = 'A';  
    do {  
        cout << car << endl;  
        car++;  
    } while (car <= 'Z') ;  
  
    return 0;  
}
```



```
C:\WINDOWS\system32\cmd.exe  
A  
B  
C  
D  
E  
F  
G  
H  
I  
J  
K  
L  
M  
N  
O  
P  
Q  
R  
S  
T  
U  
V  
W  
X  
Y  
Z  
Presione una tecla
```

# Comparación de bucles while, for y do-while

- C++ brinda 3 sentencias para el control de bucles: **while**, **for** y **do-while**.

While	For	Do-while
Comienza su ejecución y se repite <i>mientras</i> su condición de repetición del bucle es verdadera.	Se utiliza cuando el número de repeticiones se conoce en forma exacta por anticipado.	Se ejecuta una vez y luego se repite <i>mientras</i> la condición de repetición del bucle sea verdadera.
El número de repeticiones depende de una condición (verdadera o falsa) → se realiza de 0 a n veces.	El número de repeticiones está establecido de antemano → se realiza N veces	El número de repeticiones depende de una condición (verdadera o falsa) → se realiza de 1 a n veces.
La condición está en la cabecera del bucle.	La condición está en la cabecera del bucle.	La condición está al final del bucle.
Puede ejecutarse 0 veces.	Puede ejecutarse 0 veces	Se ejecuta 1 vez como mínimo.
Es la más general.	Es una estructura muy flexible en C (no así en otros lenguajes).	Se puede simular con un while o un for.

# Ciclos Anidados

- En C++ es posible **anidar** bucles.
- Los bucles anidados constan de un bucle externo con uno o más bucles internos.
- Cada vez que se repite el bucle externo, los bucles internos se repiten, se vuelven a evaluar los componentes de control y se ejecutan todas las iteraciones requeridas.

The diagram illustrates nested loops in C++ with the following components:

- Code Snippet:**

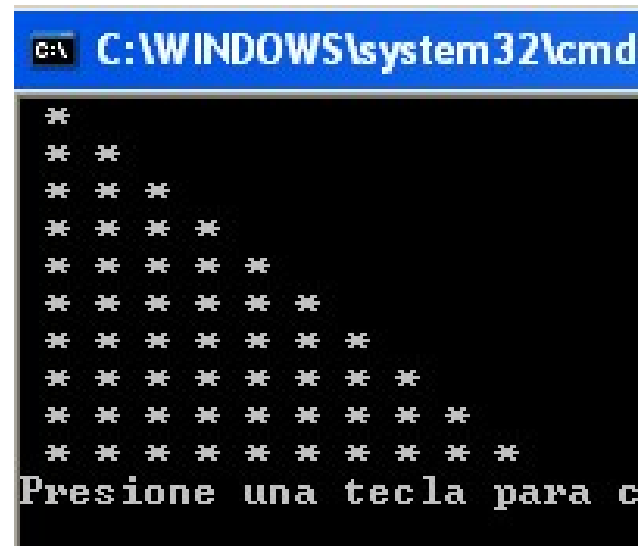
```
int main() {  
    for (int x=1; x<=10; x++)  
    {  
        for (int y=1; y<=10; y++)  
        {  
            int producto;  
            producto = x * y;  
            cout << x << "*" << y << "=" << producto;  
            cout << endl;  
        }  
    }  
    return 0;  
}
```
- Annotations:**
  - A green box labeled "Variable de control externa" points to the variable `x` in the outer loop's header.
  - A yellow box labeled "Variable de control interna" points to the variable `y` in the inner loop's header.
- Labels:**
  - "Bucle Externo" is placed next to the outer `for` loop.
  - "Bucle Interno" is placed next to the inner `for` loop.
- Output:** A vertical list of multiplication results on the right side of the diagram:

```
1*1=1  
1*2=2  
1*3=3  
1*4=4  
1*5=5  
1*6=6  
1*7=7  
1*8=8  
1*9=9  
1*10=10  
2*1=2  
2*2=4  
2*3=6  
2*4=8  
2*5=10  
2*6=12  
2*7=14  
2*8=16  
2*9=18  
2*10=20  
3*1=3  
3*2=6  
3*3=9  
3*4=12  
3*5=15
```

## CICLOS ANIDADOS – Ejemplo 9

- Imprimir un triángulo rectángulo izquierdo que tenga en la primera fila un \*, en la segunda 2 y así por un total de 10 filas.

```
int main() {  
    for(int n=1; n<=10; n++) {  
        for(int k=1; k<=n; k++)  
            cout << " *";  
        cout << endl;  
    }  
  
    return 0;  
}
```



```
C:\WINDOWS\system32\cmd  
*  
* *  
* * *  
* * * *  
* * * * *  
* * * * * *  
* * * * * * *  
* * * * * * * *  
* * * * * * * * *  
* * * * * * * * * *  
Presione una tecla para c
```

## CICLOS ANIDADOS – Ejemplo 10-a)

- Mostrar todos los divisores de un conjunto de números naturales leídos, que termina con cero.

```
int main(){
    int nro;
    cout << "Ingrese el primer numero: ";
    cin >> nro;
    while (nro != 0)
    {
        cout << "Los divisores de " << nro << " son: ";
        for (int k=2; k<=nro/2; k++)
            if (nro%k==0) cout << k << " - ";
        cout << endl << endl << "Ingrese el siguiente numero: ";
        cin >> nro;
    }

    return 0;
}
```

```
Ingrese el primer numero: 20
Los divisores de 20 son: 2 - 4 - 5 - 10 -

Ingrese el siguiente numero: 15
Los divisores de 15 son: 3 - 5 -

Ingrese el siguiente numero: 17
Los divisores de 17 son:

Ingrese el siguiente numero: 0
```

```
// El programa ha finalizado, presione cualquier tecla para continuar
```

## CICLOS ANIDADOS - Ejemplo 10-b)

- El último ejemplo puede mejorarse, considerando que un impar no puede tener divisores pares.

```
int main(){
    int nro;
    cout << "Ingrese el primer numero: ";
    cin >> nro;
    while (nro != 0) {
        cout << "Los divisores de " << nro << " son: ";
        if (nro%2 != 0) {
            for(int k=3; k<=nro/2; k=k+2)
                if (nro%k==0) cout << k << " - ";
        }
        else
            for(int k=2; k<=nro/2; k++)
                if (nro%k==0) cout << k << " - ";
        cout << endl << endl << "Ingrese el siguiente numero: ";
        cin >> nro;
    }

    return 0;
}
```

```
Ingrese el primer numero: 56
Los divisores de 56 son: 2 - 4 - 7 - 8 - 14 - 28 -
Ingrese el siguiente numero: 57
Los divisores de 57 son: 3 - 19 -
Ingrese el siguiente numero: 47
Los divisores de 47 son:
Ingrese el siguiente numero: 0
```

# CICLOS ANIDADOS – Ejemplo 11

- Leer una secuencia ordenada de números naturales que termina con 1000. Ir informando cuantos números de la secuencia ingresada hay en cada centena (0-99,100-199,.....).

```
int main() {
    int nro, kcent;
    int cent = 99;
    cout << "Ingrese el primer numero: ";
    cin >> nro;
    while (nro != 1000) {
        kcent=0;
        while (nro <= cent) {
            kcent++;
            cout << endl << "Ingrese el siguiente numero: ";
            cin >> nro;
        }
        cout << endl << "En la centena " << cent-99 << "-" << cent << " hay: "
            << kcent << " valores";
        cent = cent +100;
    }
    return 0;
}
```

Ver resultado para los valores:

67,350,375,403,456,  
477,702, 1000



```
Ingrese el primer numero: 67
Ingrese el siguiente numero: 350
En la centena 0-99 hay: 1 valores
En la centena 100-199 hay: 0 valores
En la centena 200-299 hay: 0 valores
Ingrese el siguiente numero: 375
Ingrese el siguiente numero: 403
En la centena 300-399 hay: 2 valores
Ingrese el siguiente numero: 456
Ingrese el siguiente numero: 477
Ingrese el siguiente numero: 702
En la centena 400-499 hay: 3 valores
En la centena 500-599 hay: 0 valores
En la centena 600-699 hay: 0 valores
Ingrese el siguiente numero: 1000
En la centena 700-799 hay: 1 valores
<< El programa ha finalizado: codigo de e
```

**Leer Capítulo 4**

**Libro:**

**Benjumea-Roldan  
Univ-de-Málaga**