TEMA 4: Programación estructurada

4.1.-Introducción. Teorema de la programación estructurada

El principio fundamental de la programación estructurada es que en todo momento el programador pueda mantener el programa "dentro" de la cabeza. Esto se consigue con:

a) un diseño descendente del programa,

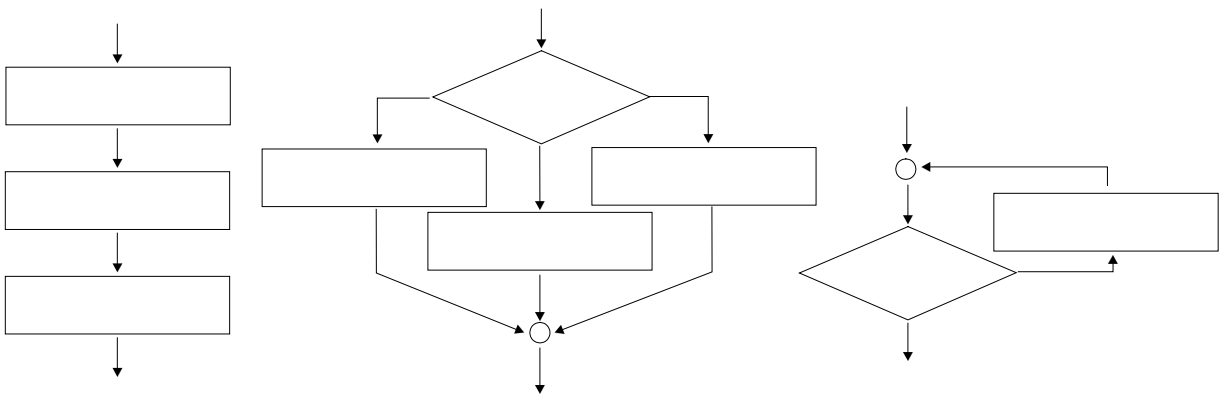
b) unas estructuras de control limitadas y

c) un ámbito limitado de las estructuras de datos del programa.

Hace más fácil la escritura y verificación de programas. Se adapta perfectamente al diseño descendente.

Para realizar un programa estructurado existen tres tipos básicos de estructuras de control:

- *Secuencial*: Ejecuta una sentencia detrás de otra.
- *Condicional*: Se evalúa una expresión y, dependiendo del resultado, se decide la siguiente sentencia a ejecutar.
- *Iterativa*: Repetimos un bloque de sentencias hasta que sea verdadera una determinada condición.



Existe un teorema debido a [C.Böhm, G.Jacopini, Comm. ACM vol.9, nº5, 366-371, 1966] (Teorema Fundamental de la programación estructurada) que establece lo siguiente:

"Todo programa propio se puede escribir utilizando únicamente las estructuras de control secuencial, condicional e iterativa"

Un programa propio es aquel que:

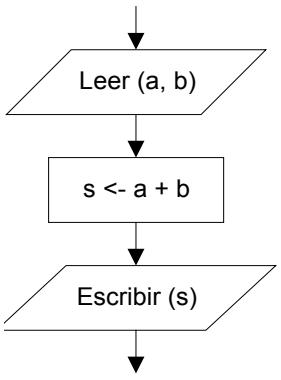
- Tiene un único punto de entrada y un único punto de salida.
- Existen caminos desde la entrada hasta la salida que pasan por todas las partes del programa.
- Todas las instrucciones son ejecutables y no existen bucles sin fin.

Este teorema implica que la utilización de la sentencia GOTO es totalmente innecesaria, lo que permite eliminar esta sentencia. Un programa escrito con GOTO es más difícil de entender que un programa escrito con las estructuras mencionadas.

4.2 Estructura secuencial

Ejecución de sentencias una detrás de la otra. En C++, toda una estructura secuencial se agrupa mediante los símbolos **{** y **}**.

Ejemplo:

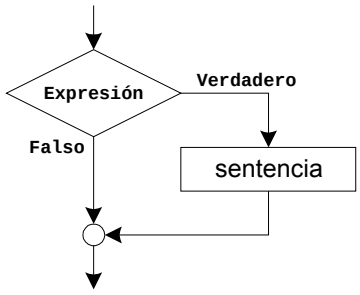



```
{
    cin >> a;
    cin >> b;
    s = a + b;
    cout << s;
}
```

Todo el bloque se considera una sola sentencia. Después de las llaves no se pone punto y coma.

4.3 Estructura condicional

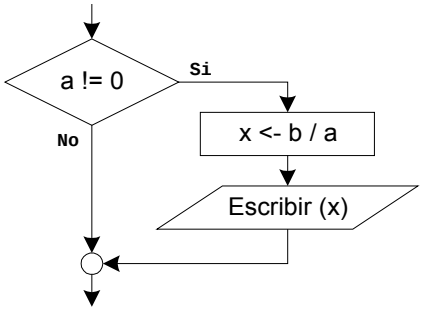
4.3.1.-Alternativa simple:



Pseudocódigo:

si <Expresión Lógica> **entonces**
    <sentencia>
fin si

Ejemplo:



si (a != 0) **entonces**
    x <- b / a
    escribir x
fin_si

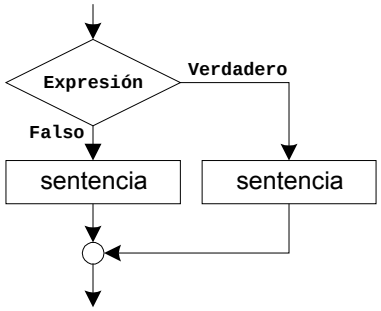
En C++ la sintaxis de la alternativa simple sería:

```
if (expresión lógica)
    sentencia
```

Ejemplo:

```
if (a != 0)
{
    x = b / a;
    cout << x;
}
```

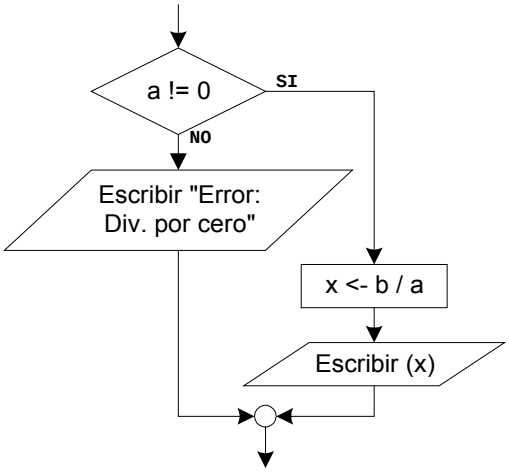
4.3.2.-Alternativa doble:



Pseudocódigo:

si <Expresión Lógica> **entonces**
    <sentencia>
sino
    <sentencia>
fin_si

Ejemplo:




```
si (a != 0) entonces
  x <- b/a
  escribir x
sino
  escribir "Error: División por cero"
fin_si
```

En C++ la sintaxis sería:

```
if (expresión lógica)
    sentencia1
else
    sentencia2
```

Ejemplo:

```
if (a != 0)
{
    x = b/a;
    cout << x;
}
else
    cout << "Error: División por cero";
```

4.3.3.-Sentencias if anidadas:

Cuando la sentencia dentro del **`if`** es otra sentencia **`if`**.

Ejemplo:

```
si (a > b)
   si (a > c)
      max = a;
   sino
      max = c;
   fin_si
sino
   si (b > c)
      max = b;
   sino
      max = c;
   fin_si
fin_si
```

En C++:

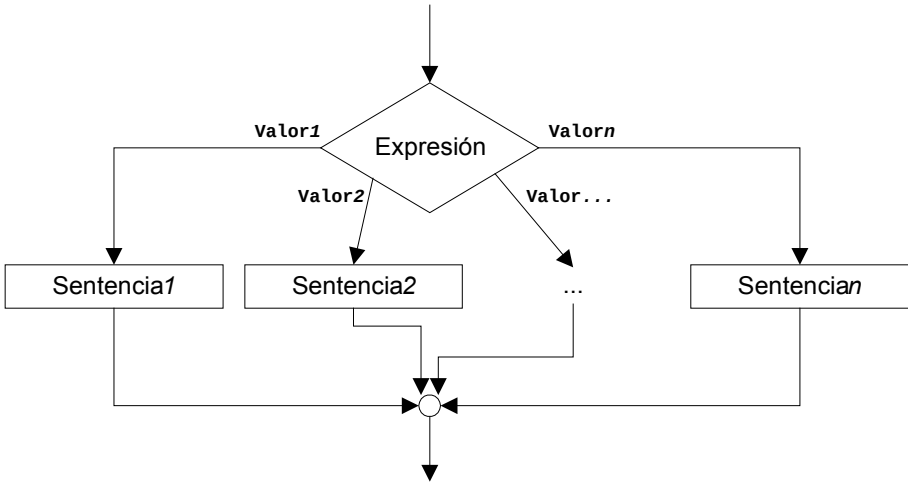
```
if (a > b)
   if (a > c)
      max = a;
   else
      max = c;
else
   if (b > c)
      max = b;
   else
      max = c;
```

La parte **`else`** siempre se asocia al **`if`** más cercano posible.

Ejemplo:

```
if (a == 1)
    if(b == 1)
      cout << "1 1";
else       // Con que if esta asociado este else ??
    cout << " ? ";
```

4.3.4.-Alternativa múltiple:



Pseudocódigo:

según <Expresión> **hacer**
    <valor*1*>: <sentencia*1*>
    <valor*2*>: <sentencia*2*>
    ...
    <valor*n*>: <sentencia*n*>
fin_segun

La expresión ya no es lógica, sino de tipo ordinal.

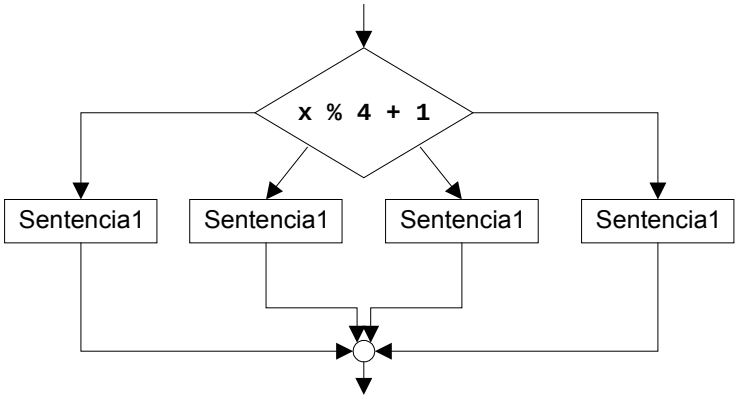
Una forma de implementar esta estructura es mediante sentencias **si** anidadas.

si (Expresion = Valor*1*) **entonces**
    Sentencia*1*
sino
    si (Expresion = Valor2) **entonces**
        Sentencia*2*
    sino
        si (Expresion = Valor...) **entonces**
            ...
        sino
            ...
            si (Expresion = Valor*n*) **entonces**
                Sentencia*n*
            fin_si
        ...
        fin_si
    fin_si
fin_si

Ejemplo:

```
si (x % 4 + 1 == 1) entonces
    sentencia1
sino
    si (x % 4 + 1 == 2) entonces
      sentencia2
    sino
      si (x % 4 + 1 == 3) entonces
           sentencia3
      sino
           sentencia4
      fin_si
    fin_si
fin_si
```

Mediante la estructura **según**:




```
según (x % 4 + 1) hacer
   1: sentencia1
   2: sentencia2
   3: sentencia3
   4: sentencia4
fin según
```

La sintaxis en C++ será:

```
switch (Expresión ordinal)
{
    case valor1:
        sentencias
        break;
    case valor2:
        sentencias
        break;
...
    default:
        sentencias     // Opcional
}
```

Ejemplo:

```
switch (x)
{
    case 1:
        cout << "Uno";
        break;
    case 2:
        cout << "Dos";
        break;
    case 3:
        cout << "Tres";
        break;
    default:
        cout << "> 3";
}
```

Uno de los ejemplos más claros de uso del switch es en menús:

```
cin >> opcion;
switch (opcion)
{
    case 1:
        IntroducirNombre();
        break;
    case 2:
        Listar();
        break;
    case 3:
        Salir();
        break;
    default:
        cout << "Opcion no valida";
}
```

La instrucción **switch** también admite poner varias opciones juntas.

```
switch (Expresión ordinal)
{
    case valor1:
    case valor2:
        sentencias
        break;
     ...
    case valorn:
        sentencias
        break;
...
    default:
        sentencias     // Opcional
}
```

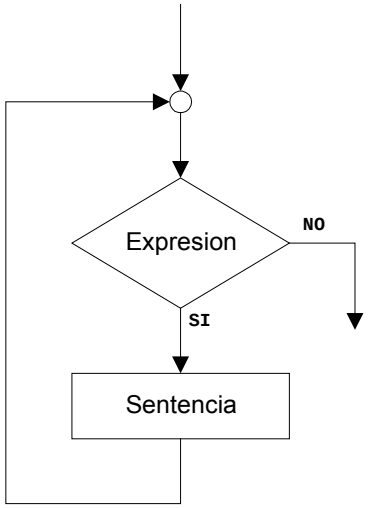
Importante reslaltar la utilización de la palabra reservada 'break' como señal de final de las sentencias que van con cada opción o conjunto de opciones.

4.4. Estructura iterativa (o bucle)

Consiste en repetir una sentencia. También se denominan bucles.

Siempre ha de existir una condición de parada, es decir, hay que garantizar que para cualquier caso, el bucle parará.

4.4.1.-Mientras

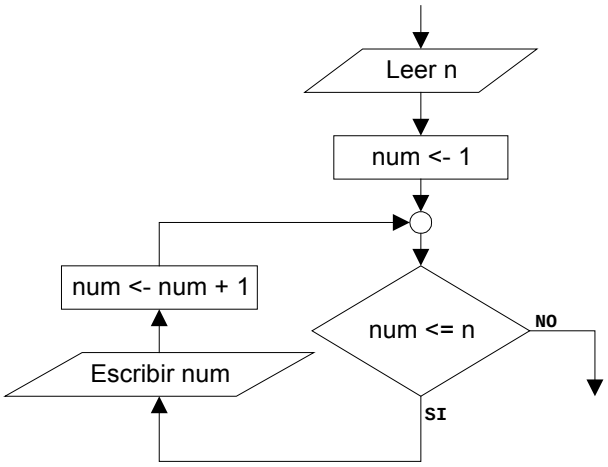


Pseudocódigo

mientras <expresión lógica> **hacer**
    <sentencia>
fin_mientras

Ejemplo:

Escribir los números enteros de 1 a N



Leer n
num ← 1
mientras num <= n hacer
    Escribir num
    num ← num + 1
fin_mientras

La sentencia puede no ejecutarse nunca si la condición no se cumple.

Si la condición está mal hecha, el bucle puede no acabar nunca. Es lo que se denomina un bucle infinito

En **C++** la sintaxis sería:

```
while (expresión lógica)
    sentencia
```

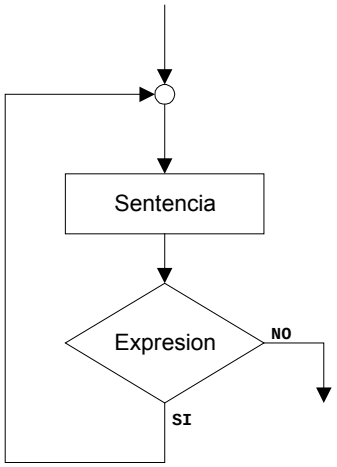
Ejemplo:

```
cin >> n;
```

```
num = 1;
while (num <= n)
{
    cout << num << endl;
    num++;
}
```

Con el bucle mientras se puede construir cualquier estructura iterativa. De todas maneras, por comodidad existen otras estructuras iterativas.

4.4.2.-Repetir (o hacer)



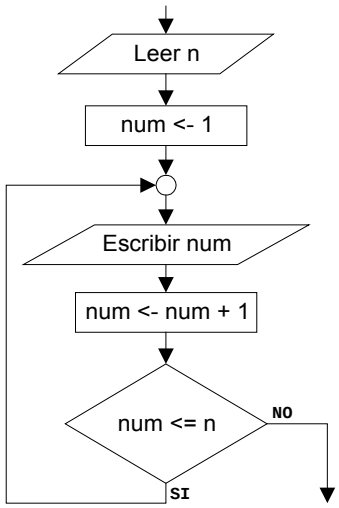
Pseudocódigo

repetir
    <sentencia>
mientras <expresión lógica>

Repite la sentencia mientras la expresión sea cierta.

La sentencia siempre se ejecuta como mínimo una vez.

Ejemplo:



Leer n
num <- 1
repetir
    Escribir num
    num <- num + 1
mientras (num <= n)

La sintaxis en **C++**:

```
do
    sentencia
```

```
while (expresión lógica);
```

Ejemplo:

```
cin >> n;
num = 1;
do
{
    cout << num << endl;
    num++;
}
while (num <= n)
```

Se utiliza sobre todo en preguntas o menús, puesto que siempre se han de ejecutar al menos una vez.

```
do
{
    cout << "Introduce un entero";
    cin >> i;
    cout << "¿Es correcto (s/n)?";
    cin >> c;
}
while (c != 's');
```

4.4.3.-Para (o desde)

Es como una estructura mientras, pero especialmente preparada para incorporar un contador.

Pseudocódigo

para <variable> <- <valor inicial> **hasta** <valor final> **hacer**
  <sentencia>
fin_para

La variable del bucle se denomina *variable de control*.

Ejemplo:

```
leer n
para num ← 1 hasta n hacer
        escribir n
fin_para
```

Se utiliza cuando se conocen el número de veces que se van a repetir las sentencias.

La variable de control NO se puede modificar dentro del bucle.

Cuando el bucle acaba, el valor de la variable de control es indeterminado.

La sintaxis en **C**++ es:

```
for(inicialización; condición; incremento)
    sentencia
```

En C++, la instrucción **`for`** es como una instrucción **`while`**, con la salvedad de que tiene huecos especiales para poner la inicialización de la variable de control, la condición de repetición y el incremento de la variable de control.

El incremento se realiza siempre después de ejecutar la sentencia.

Al igual que en la instrucción **`while`**, la sentencia puede no ejecutarse nunca si la condición no se cumple.

Ejemplo:

```
cin >> n;
for(num = 1; num <= n; num ++)
    cout << num << endl;
```

Se utiliza también para realizar sumatorios o productorios con la ayuda de una variable *acumuladora*:

Ejemplo:

```
// inicialización del acumulador
suma = 0;
for(i = 1; i <= n; i++)
    suma = suma + i;        // o suma += i;
cout << suma;
```

```
factorial = 1;
for(i = 1; i <= n; i++)
    factorial = factorial * i;
cout << factorial;
```

4.4.4. Tipos de control de bucles

Existen tres formas típicas de controlar cuando se ejecuta un bucle:

a) Bucles con contador (ya vistos).

b) Bucles controlados por indicadores (banderas o *flags*).

Ejemplo:

```
bool continuar;

continuar = true;   // Inicializacion del indicador
while (continuar)
{
    ...
    if (condición para acabar)
        continuar = false;
    ...
}
```

Decir si un numero introducido por teclado contiene sólo cifras menores que cinco;

```
bool menor;
int num;

cin >> num;
menor = true;
while (menor && (num > 0) )
{
    if (num % 10 >= 5)
        menor = false;
    num = num / 10;
}

if (menor)
    cout << "Todas las cifras son menores que 5";
else
    cout << "Hay alguna cifra mayor o igual que 5";
```

Decir si un número introducido por teclado es capicua o no.

```
bool capicua;
int num, ultima, primera, cifras;

cin >> num;
capicua = true;     // Inicializacion del indicador
while (capicua && (num > 10) )
{
    ultima = num % 10;
    cifras = int(log (num) / log (10) );
    primera = num / pow (10, cifras);
    if (ultima != primera)
        capicua = false;

    num = (num – primera * pow (10, cifras) ) / 10;
}
if (capicua)
    cout << num << " es capicua";
else
    cout << num << " NO es capicua";
```

c) Bucles controlados por centinela.

Ejemplo:

```
suma = 0;
cout << "Introduce números a sumar, 0 para acabar";
cin >> num;
while (num != 0)
{
    suma = suma + num;
    cout << "Introduce números a sumar, 0 para acabar";
    cin >> num;
}
cout << suma;
```

4.4.5. Bucles anidados

Los bucles, al igual que las sentencias condicionales, también se pueden anidar. Esto es especialmente útil para el manejo de matrices, como veremos en el Tema 6.

Ejemplo: *Tabla de multiplicar*

```
int i, j;

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