SISTEMAS OPERATIVOS EN RED
TEMA 2

2º SMR
Profesora: Noelia Huguet Chacón
Centro de Estudios Tobalcaide
Curso 2.020 – 2.021

BUCLES

- Hemos visto cómo comprobar condiciones , pero no cómo hacer que una cierta parte de un programa se repita un cierto número de veces o **mientras** se cumpla una condición (lo que llamaremos un "**bucle**").
- En C y en C++ tenemos varias formas de conseguirlo:
  - **WHILE**
  - **DO…WHILE**
  - **FOR**

WHILE

- Si queremos hacer que una sección de nuestro programa se repita mientras se cumpla una cierta condición, usaremos la orden "while".
- Esta orden tiene dos formatos distintos, según comprobemos la condición al principio o al final.
- En el primer caso, su sintaxis es:

```
while  (condición)
       sentencia;
```

- Es decir, la sentencia se repetirá **mientras** la condición sea cierta.
- Si la condición es falsa ya desde un principio, la sentencia no se ejecuta nunca.
- Si queremos que se repita más de una sentencia, basta agruparlas entre { y }.

WHILE

- Ejemplo:

```
#include <iostream>
using namespace std;
int main()
{
    int x;
    cout << "Teclea un numero (0 para salir): ";
    cin >> x;
    while (x!=0)
    {
        if (x > 0)
            cout << "Es positivo" << endl;
        else
            cout << "Es negativo" << endl;
        cout << "Teclea otro numero (0 para salir): ";
        cin >> x;
    }
    return 0;
}
```

DO…WHILE

- Este es el otro formato que puede tener la orden "while": la condición se comprueba **al final**.
- El punto en que comienza a repetirse se indica con la orden "do", así:
- Al igual que en el caso anterior, si queremos que se repitan varias órdenes (es lo habitual), deberemos encerrarlas entre llaves. Nuevamente, puede ser recomendable incluir siempre las llaves, como costumbre.

```
do sentencia;
while (condición);
```

-

DO…WHILE

- Ejemplo:

```
#include <iostream>
using namespace std;
int main()
{
    int valida = 711;
    int clave;

    do
    {
        cout << "Introduzca su clave numérica: ";
        cin >> clave;
    }
    while (clave != valida);

    cout << "Aceptada." << endl;
    return 0;
}
```

FOR

- Ésta es la orden que usaremos habitualmente para crear partes del programa que se repitan un cierto **número de veces**.
- El formato de ***for*** es:

```
for   (valorInicial; CondiciónRepetición; Incremento)
      sentencia;
```

FOR

- Ejemplo:

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

int main()
{
    int contador;

    for (contador=1; contador<=10; contador++)
        cout << contador << " ";

    return 0;
}
```

- Saldría por pantalla los número del 1 al 10 sin escribir nada.

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

BUCLE

- Un bucle for puede convertirse en while y viceversa.

```
for (i=0; i<=10; i++) {
        cout << i << " ";
    }
```

```
int i=0;
while (i<10)    {
        cout << i << " ";
       i++; }
```

```
int i=0;
do{
        cout << i << " ";
       i++; }
While ( i<10);
```

FOR

- Los bucles "for" se pueden **anidar** (incluir uno dentro de otro), de modo que podríamos escribir las tablas de multiplicar del 1 al 5 con:

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

int main()
{
    int tabla, numero;

    for (tabla=1; tabla<=5; tabla++)
        for (numero=1; numero<=10; numero++)
            cout << tabla <<  " por " <<  numero
                << " es " << tabla*numero << endl;

    return 0;
}
```

SENTENCIAS DE SALTO

- Una sentencia de salto es aquella que interrumpe de algún modo la ejecución de una sentencia de control.
  - **BREAK**: interrumpe un bucle indicándose que la ejecución del programa debe continuar en la siguiente instrucción después del mismo.
  - **CONTINUE**: Cuando es ejecutada no se ejecutan el resto de instrucciones siguientes incluidas en el bucle, se transfiere el control a la condición en caso de que se encuentre en un bucle while o do…while o se produce el incremento de la variable de control en un bucle for.

SENTENCIA DE SALTO: BREAK

```
#include <iostream>
using namespace std;
int main()
{
    int i;

    for (i=0; i<=10; i++)
    {
        if (i==5) break;
        cout << i << " ";
    }
     return 0;
}
```

- Se rompe el bucle cuando i es igual a 5. El resultado por pantalla sería:

```
0 1 2 3 4 5
```

SENTENCIA DE SALTO: CONTINUE

```
#include <iostream>
using namespace std;
int main()
{
    int i;

    for (i=0; i<=10; i++)
    {
        if (i==5) continue;
        cout << i << " ";
    }
     return 0;
}
```

- Se salta el bucle cuando i es igual a 5. El resultado por pantalla sería:

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

INSTRUCCIÓN DE SALTO: GOTO

```
int i, j;
    for (i=0; i<=5; i++)
       for (j=0; j<=6; j+=2){
           if ((i==1) && (j>=4))
               goto salida;
           cout << "i vale " << i << " y j vale "
               << j << endl;
       }
   salida:
   cout << "Fin del programa" << endl;
```

- Se sale del programa cuando i=1 y j=4. Por pantalla saldría:

```
i vale 0 y j vale 0
i vale 0 y j vale 2
i vale 0 y j vale 4
i vale 0 y j vale 6
i vale 1 y j vale 0
i vale 1 y j vale 2
Fin del programa
```

VECTORES - ARRAYS

- Un vector es un tipo de datos compuesto que permite almacenar un número x de elementos del mismo tipo.
- El uso de vectores consigue que mediante una sola declaración hagamos referencia a un conjunto de valores, estos valores pueden ser de los tipos de datos simples estudiados o bien algún tipo de datos diseñado por el usuario.
- Cada elemento del vector se referencia mediante su posición en la tabla, empezando a contar en 0.
- El contenido del vector puede ser de cualquier tipo de datos.
- La dimensión del vector indica el grosor modo el número de filas que tendrá. A mayor dimensión mayor número de valores.

Valor	13	17	2	7	9	15	21	27	…	47
Posición	0	1	2	3	4	5	6	7	…	n

VECTORES - ARRAYS

- Creación de vectores:

```
int vector[10];          //crea un vector de 10 numeros enteros
char vector[20];         //crea un vector de 20 letras
float vector[15];        //crea un vector de 15 numeros reales
string vector[5];        //crea un vector de 5 cadenas de texto
```

- Inicialización de vectores:

```
int v[6];
v[6]={10,20,30,40,50,60};

float vec[4] = {1.5,6.3,4.2,5.8};
```

```
int a[3];
a[0]=200;
a[1]=300;
a[2]=600;
```

```
char x[4]={'h','o','l','a'};
string c[2]={"noelia", "huguet"};
```

VECTORES - ARRAYS

- Una manera de sumar los valores de un vector:

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

int main()
{
    int x[3];    // Un array de 3 numeros enteros
    int suma;        // Un entero que guardará la suma

        x[0] = 200;      // Les damos valores
        x[1] = -50;
        x[2] = 100;
        suma = x[0] + x[1] + x[2]; // Hacemos la suma

    cout << "Su suma es " << suma;

    return 0;
}
```

VECTORES - ARRAYS

- Para mostrar por pantalla un vector:
  - Un vector de caracteres se mostraría así:

```
char x[4]={'h','o','l','a'};
cout << x;     //Saldría por pantalla hola
```

  - Para imprimir los número de un vector habrá que hacer un bucle:

```
 int x[5]={23,2,3,4,5};
for(int i=0;i<5;i++)
{
        cout<< x[i];
}
```

TABLA BIDIMENSIONAL

- Podemos declarar tablas de **dos o más dimensiones**.
- Por ejemplo, si queremos guardar datos de dos grupos de alumnos, cada uno de los cuales tiene 20 alumnos, tenemos dos opciones:
  - Podemos usar **int datosAlumnos[40]** y entonces debemos recordar que los 20 primeros datos corresponden realmente a un grupo de alumnos y los 20 siguientes a otro grupo.
  - O bien podemos emplear **int datosAlumnos[2][20**] y entonces sabemos que los datos de la forma **datosAlumnos[0][i]** son los del primer grupo, y los **datosAlumnos[1][i]** son los del segundo.
- En cualquier caso, si queremos indicar valores iniciales, lo haremos entre llaves, igual que si fuera una tabla de una única dimensión.

TABLA BIDIMENSIONAL

- Ejemplo:

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

int main()
{
    int notas[2][10] =
        {
            { 7, 2, 10, 6, 4, 6, 7, 8, 2, 1},
            {1, 4, 6, 7, 8, 2, 10, 8, 1, 5 }
        };

    cout << " La nota del tercer alumno del grupo 1 es: " << notas[0][2];
    cout << "\n La nota del quinto alumno del grupo 2 es: "<< notas[1][4];

    return 0;
}
```