Objetivos de la práctica:
- Trabajar las estructuras de control repetitivas en la realización de programas.

Sentencias de Control repetitivas (o bucles)

Al igual que las estructuras selectivas, el bucle rompe la secuencialidad en la ejecución de las instrucciones, pero a diferencia de aquéllas, rompe además el avance hacia adelante, volviendo a ejecutar un conjunto de sentencias una y otra vez.
Con este tipo de estructuras podemos ejecutar una tarea repetidas veces, e incluso hacer que cada nueva vuelta no sea exactamente igual a la anterior, cambiando algunos valores.
Cuando finaliza el bucle, continúa la ejecución del resto del programa.
Dentro del C/C++ existen tres estructuras de control repetitivas: el bucle **`for`**, el bucle **`while`** y el bucle **`do .. while`**. Aunque estas tres estructuras en el fondo son equivalentes, se recomienda el uso de la estructura **`for`** cuando el bucle tiene que ejecutarse un número de veces conocido, y el de las otras dos cuando lo que se conoce es la condición a cumplirse para que se siga repitiendo el bucle.

- SENTENCIA **`while`**

Este bucle se suele utilizar cuando no se conoce de antemano el número de veces que se va a repetir el bucle, sino que, lo que se conoce es la condición que ha de cumplirse para que se repita.
El bucle **`while`** se corresponde con la estructura "mientras" que existe en otros lenguajes. Esta estructura repite el cuerpo del bucle mientras sea verdadera una expresión condicional, por tanto, ejecuta un conjunto de sentencias cero o más veces dependiendo del valor de la expresión.

- Sintaxis:

```
while ( expresión condicional )
{
   Bloque de sentencias;
}
```

Las llaves no son necesarias cuando el bloque de sentencias está formado por una única sentencia.
La ejecución sucede de la siguiente manera:

1. Se evalúa la expresión. Si el resultado es falso, acaba el bucle. Si es verdadero continua en el punto 2.
2. Se ejecutan el bloque de sentencias.
3. Vuelve al punto 1.

- *Ejemplo:*

```
while ( xt<39 )
{
   df *= xt;
   xt += 2;
}
```

- SENTENCIA **`do .. while`**

En C/C++ no existe la estructura "repetir - hasta", en cambio proporciona una estructura muy similar, la estructura "hacer - mientras", en donde se repite el cuerpo del bucle mientras sea verdadera una expresión condicional, por tanto, el bucle **`do.. while`** ejecuta un conjunto de sentencias una o más veces, dependiendo del valor de una expresión condicional.
Igual que la anterior sentencia (**`while`**), se recomienda su uso cuando no se conoce el número de veces que debe repetirse el bucle. Lo que lo diferencia del bucle anterior, es que la condición se comprueba después de ejecutarse el bloque de sentencias

- Sintaxis:

```
do
{
   Bloque de sentencias;
}
while ( expresión condicional );
```

La ejecución sucede de la siguiente manera:

1. Se ejecuta el bloque de sentencias.
2. Se evalúa la expresión condicional. Si el resultado es falso, se sale del bucle. Si es verdadero, se vuelve al punto 1.

- *Ejemplo:*

```
#include <iostream>
using namespace std;
int main ( void )
{
   char opcion;

   do
   {
      cout << "Pulsa una tecla numérica\n" ;
      cin >> opcion;
   }
   while ( ( opcion < '0' ) || ( opcion > '9' ) );

   return 0;
}
```

Cuidado: la sentencia **do .. while** termina siempre con un **punto y coma (;)**. Esto está también puesto en la parte de "comentarios de interés"

- SENTENCIA **`for`**

Se utiliza cuando se necesita ejecutar repetidamente una sentencia o bloque de sentencias, un número de veces conocido.

En C, el bucle **`for`** coincide más o menos con el bucle "desde - hasta" que se utilizan en otros lenguajes como el Pascal, Fortran, Basic, etc. Sin embargo, en C es algo diferente y bastante más flexible. Como en otros lenguajes, permite ejecutar un bloque de sentencias un número de veces conocido, utilizando un contador que toma valores desde un valor inicial y que incrementa (o decrementa) su valor mientras (aquí comienzan las diferencias) sea verdadera una condición.

- Sintaxis:

```
for ( iniciación del contador; condición; progresión del contador )
{
   Bloque de sentencias del bucle;
}
```

El bucle comienza con la palabra reservada **`for`** seguida de paréntesis. El contenido de los paréntesis se divide en tres campos, separados cada uno de ellos por un punto y coma:

- En la primera parte se fija el valor inicial del contador.
- En la segunda parte se fija la expresión condicional bajo la cual el bucle se repite.
- En la tercera parte se fijan los cambios que tienen lugar sobre el contador en cada vuelta del bucle.

A continuación de los paréntesis y encerrado entre llaves viene el bloque de sentencias que se ejecuta en cada vuelta del bucle (las llaves sólo son necesarias en el caso de que el bloque esté formado por más de una sentencia).

La ejecución del bucle se realiza del siguiente modo:

1. Se inicia el contador.
2. Se evalúa la expresión condicional. Si es falsa acaba el bucle. Si es verdadera, continua con el paso 3.
3. Se ejecutan el bloque de sentencias.
4. Se actualiza el contador.
5. Vuelve al paso 2.

- *Ejemplo:*

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

   for ( i = 1; i <= 10; i++ )
      cout << i ;

   return 0;
}
```

El bucle *for* **es mucho más versátil**

El bucle **for** es muy flexible en cuanto al contenido de cada uno de los campos que encierra los paréntesis. Así, puede tener varias expresiones de inicialización y varias expresiones de progresión. En este caso, estas expresiones se separan por comas.

- Sintaxis:

```
for ( [v1 = e1, [v2 = e2]..]; [condición]; [progresión de las variables] )
{
   Bloque de sentencias;
}
```

donde:

vi = ei: Representan los valores iniciales que adquieren las variables de control.

condicion: Si es cierta, se ejecuta el bloque de sentencias.

progresion-condicion: Como evolucionan las variables de control.

Ejemplos:

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

int main ( void )
{
   short i, j;

   cout << "los diez primeros numeros pares son \n" ;

   for ( i = 0, j = 2; i < 10; i++, j += 2 )
      cout << " el " << i << " es " << j << endl ;

   cout << "los 10 primeros nums. en la numeración arábica:\n" ;

   for ( i = 0, j = 0; i < 10; i++, j++ )
      cout << " el " << i << " es " << j << endl ;

   return 0;
}
```

```
#include <iostream>
using namespace std;
int main ( void )
{
   long k;
   float m;

   for ( k = 1, m = 9.0; k * m <= 15.0; k++, m -= 0.5 )
   {
      cout << k * m << endl ;
   }

   return 0;
}
```

Nota Importante: pueden haber varias expresiones de inicialización o varias expresiones de progresión pero no puede haber más de una expresión de condición.

Por otra parte, el bucle **`for`** admite que alguno de los campos esté vacío, es decir, que no exista ninguna expresión de inicialización y/o ninguna expresión de condición y/o ninguna expresión de progresión. Incluso, es posible un bucle **`for`** con todos sus campos vacíos:

```
for ( ; ; )
{
   Bloque de sentencias;
}
```

Siempre que falta la expresión condicional, estamos ante el caso de un bucle infinito que se ejecuta indefinidamente, a no ser que dentro del bloque de sentencias exista una sentencia de salto (son sentencias que interrumpen el flujo normal del programa, desviándolo hacia un punto diferente). Como las sentencias de salto no son imprescindibles y provocan códigos de programa poco estructurados, se recomienda restringir su uso y por tanto evitaremos el uso de bucles infinitos.

Bucles anidados

Como parte del bloque de sentencias que pertenece al cuerpo de un bucle, se pueden incluir otros bucles, es lo que se llama anidamiento de bucles. Esto proporciona una mayor flexibilidad a la hora de resolver ciertos problemas.

Comencemos con el caso más sencillo de anidamiento de bucles, el de dos bucles anidados, es decir, el caso de un bucle dentro de otro. El bucle interno (*b.i.*) es una sentencia más, dentro del bloque de sentencias del bucle externo (*b.e.*), por lo tanto, el ciclo completo del *b.i.* se ejecuta repetidamente tantas veces como veces se repita el *b.e.*. Esto quiere decir que una sentencia dentro del cuerpo del *b.i.* se va a ejecutar, las "n" veces que se repetirían en el caso de que no fuera anidado el *b.i.*, por las "m" veces que se ejecutan cada una de las sentencias del *b.e.*, es decir, "m * n" veces.

Paso a paso, sucedería del siguiente modo.

1. Se evalúa la expresión condicional del bucle más externo. Si es falsa, se sale de los bucles y continúa la ejecución del resto del programa. Si es verdadera continúa con el paso 2.
2. Se ejecuta el bloque de sentencias del bucle más externo. Entre ellas, se encontrará el bucle mas interno.
   2.1. Se evalúa la expresión del bucle mas interno. Si es falsa, sale del bucle interno y continúa con la ejecución del bucle externo, en el paso 3. Si es verdadera continúa en el paso 2.2.
   2.2. Se ejecutan el bloque de sentencias del bucle interno.
   2.3. Vuelve al paso 2.1.
3. Vuelve al paso 1.

Ejemplo:

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

int main ( void )
{
   short i, j;

   cout << "la tabla de multiplicar del 1 al 10 es:\n" ;
   for ( i = 1; i <= 10; i++ )
   {
      for ( j = 1; j <= 10; j++ )
         cout << i << " por " << j << " vale " << i * j << endl ;
   }

   return 0;
}
```

Se pueden anidar muchos niveles y con diferentes tipos de bucles.

Cuestión de estilo
El anidamiento de bucles provoca, en los principiantes, auténticos caos mentales debidos sobre todo por la anarquía con el que se escribe el código. Para hacer más fácil la comprensión y la lectura de un programa con bucles anidados, se recomienda al alumno que el bloque de sentencias de los bucles se situen en columnas más interiores al de las llaves, de manera que quede bien claro donde empieza y donde acaba cada bucle.

Resumen de conceptos básicos: ESTRUCTURAS ITERATIVAS

Sentencia while

```
while (expresión lógica)
{
    sentencia1;
    sentencia2;
}
```

Ejemplo:

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

Sentencia do..while

```
do
{
    sentencia1;
    sentencia2;
}
while (expresión lógica);
```

Ejemplo:

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

Sentencia for

```
for(inic; cond; increm)
{
    sentencia1;
    sentencia2;
}
```

Ejemplo:

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

COMENTARIOS DE INTERÉS

1. Las sentencias **while**() y **for**() no finalizan con '**;**', mientras que la sentencia **do..while**(); sí.

2. En el cuerpo de un bucle **for**, no debemos variar el valor de las variables que forman parte de la condición de fin, puesto que pueden producirse efectos inesperados.

3. ¿Cuándo utilizar un bucle u otro?

- La sentencia **while** suele utilizarse cuando no se conoce el número de iteraciones del bucle, pudiendo ser éste mayor o igual a 0.
- La sentencia **do...while** suele utilizarse cuando no se conoce el número de iteraciones del bucle, pudiendo ser éste mayor o igual a 1.
- La sentencia **for** suele utilizarse cuando se conoce exactamente el número de iteraciones del bucle.

4. Técnicas de control de los bucles:

- **Bucles controlados por indicadores (banderas):**
  Se utiliza una variable "bandera" de tipo bool, de cuyo valor depende la terminación del bucle.

```
bool continuar;

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

- **Bucles controlados por centinela:**
  En un proceso de introducción de datos, el centinela es el valor cuya lectura como dato, indica la finalización del proceso.

```
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;
```