Práctica 3: Estructuras de control repetitivas

Objetivos

- Comprender el uso de las estructuras de control iterativas.

- Conocer la implementación en C++ de las estructuras de control iterativas

- Aplicar la concatenación y anidamiento de estructuras de control selectivas y repetitivas en sencillos programas.

Estructuras de control iterativas

Las estructuras de control repetitivas rompen la secuencialidad en la ejecución de los programas, generando repeticiones de sentencias o bucles.

Estructura de control	**While**

Sintaxis	**while** (condición continuidad)
	{
	  Bloque de sentencias;
	}

Ejecución	1º Se evalúa la condición: - Si es F, sale del bucle. - Si es V, continúa en 2º. 2º Se ejecuta el bloque de sentencias. 3º Vuelve a 1º.
Cuándo utilizarlo	• Cuando no se conoce el número de veces que se ha de repetir la secuencia. • Puede que el bloque de sentencias no se ejecute.
Otras características	• Suele utilizarse cuando se conoce la condición lógica, "mientras que..." • Para que pueda terminar, debe haber código dentro del bloque de sentencias que en algún momento haga falsa la condición.

Las llaves { … } sólo son necesarias en el caso de que el bloque esté formado por más de una sentencia.

Estructura de control	**do-while**

Sintaxis	**do**
	{
	  Bloque de sentencias;
	}
	while (condición continuidad);

Ejecución	1º Se ejecuta el bloque de sentencias. 2º Se evalúa la condición: - Si es F, sale del bucle. - Si es V, vuelve a 1º

Cuándo utilizarlo	• Cuando no se conoce el número de veces que se debe repetir la secuencia. • Suele utilizarse cuando se conoce la condición lógica para que se ejecute un bloque de sentencias, pero asegurando que el bloque de sentencias se ejecute al menos una vez.
Otras características	• El bloque de sentencias se ejecuta al menos 1 vez. • Para que pueda terminar, debe haber código dentro del bloque que en algún momento haga falsa la condición.

Estructura de control	**For**

Sintaxis	**for** (inicio_contador; condición; actualizacion_contador)
	{
	Bloque de sentencias;
	}

Ejecución	1º Se inicia el contador. 2º Se evalúa la condición: - Si es F, sale del bucle. - Si es V, continúa en 3º. 3º Se ejecuta el bloque de sentencias. 4º Se actualiza el contador y vuelve a 2º.
Cuándo utilizarlo	• Cuando se conoce el número de veces que el bloque de sentencias debe repetirse. Este número se controla con el incremento del contador.
Otras características	• Puede que nunca llegue a ejecutarse el cuerpo del bucle.

BLOQUE DE EJERCICIOS

Ejercicio 1. Depurando un programa. Abre el fichero ejercicio1.cpp. ¡Cuidado contiene errores! Compílalo y ejecútalo. Comprueba que el resultado no es correcto. ¿Uhmmm? Vamos a ver qué ocurre paso a paso usando el depurador.

Establece un punto de ruptura en la línea 35 y añade un `watch` para cada una de las variables `digito`, `c` y `num`. Rellena las siguientes tablas con el contenido de las variables **al final** de cada iteración. Usa el valor que aparece en los `watch`.

Para la entrada 6147:

Iteracion	1	2	3
digito			
c			
num			

Para la entrada 42:

Iteracion	1
digito	
c	
num	

¿Cuál es el error que se ha cometido en el algoritmo? Corrígelo.

Ejercicio 2 (ejercicio2.cpp). Implementa un programa en C++ que pida al usuario introducir desde el teclado un número entero e indique si es divisible por 2, 3 ó 5. El programa acaba cuando el usuario introduce el valor 0 o un número negativo; de lo contrario pide al usuario un nuevo número y realiza el mismo proceso.

Indica el diagrama de flujo. Documenta el programa usando los comandos de Doxygen y genera la documentación.

Ejercicio 3 (ejercicio3.cpp). Escribe un programa en C++ que calcule la siguiente función *f* sobre los números enteros positivos:

$$f(x) = a \cdot x + b,$$

en el intervalo [x_1, x_2]. Los valores de los coeficientes *a*, *b* y de los *límites* del intervalo se introducen por teclado. Por pantalla se muestran los valores de la función en el intervalo. El proceso debe poder repetirse hasta que el usuario decida terminar introduciendo el carácter 'N'.

Documenta el programa usando los comandos de Doxygen.

Ejemplo de ejecución:

```
Funcion: f(x) = 2 * x + 3

Intervalo: [1,3]

Valor para x = 1, f(x) = 5
Valor para x = 2, f(x) = 7
Valor para x = 3, f(x) = 9

Desea continuar? Pulsa S para continuar y N para terminar:
N

Ha decidido terminar. Bye-bye!
```

Ejercicio 4. Número Creciente (ejercicio4.cpp).

Parte 1. Un número entero positivo que cumple que cada uno de sus dígitos no es excedido por la cifra que aparece a su izquierda se dice que es creciente. El número 34468 es creciente, y el número 34826 no lo es. Implementa un programa en C++ que determine si un número es creciente o no.

Parte 2. Añade el código necesario para que se calcule la cantidad de números crecientes que existen en el intervalo [1, 10000].

Documenta el programa usando los comandos de Doxygen.

IMPORTANTE:

- Todos los programas deben seguir la Guía de Estilo de C++ e incluir los comandos de Doxygen (pdf en Aula Virtual) para generar la documentación.
- Subir a Aula Virtual todos los archivos .cpp (de uno en uno) sin comprimir (enunciado y adicionales). Sólo los sube un miembro de la pareja.
- Fecha de entrega: durante la sesión de vuestro grupo de lab.
- Realizar una foto del pseudocódigo y/o el diagrama de flujo. Subir el archivo junto con los ficheros cpp.