Hoja de ejercicios del Tema 1

En los siguientes ejercicios, los parámetros *N* y *M* se pueden considerar constantes, establecidas a valores concretos cualesquiera al principio del programa. Hay formas más convenientes de definir estas funciones usando plantillas o estructuras de tamaño variable como `vector`, pero eso lo veremos más adelante.

Constantes en C++

Las constantes en C++ se pueden definir anteponiendo las palabras clave `const` o `constexpr` a una definición de variable. `constexpr` es más exigente que `const` y su definición debe ser una expresión que se pueda evaluar completamente en el momento de la compilación.

1. Define un tipo `Secuencia` que permita representar secuencias de *N* enteros e implementa, además de funciones para leerlas y mostrarlas:

(a). una función que, dada una secuencia, mueva sus componentes un lugar a la derecha. El último componente se moverá al 1er lugar.

(b). una función que, dada una secuencia, calcule y devuelva la suma de los elementos que se encuentran en las posiciones pares de la secuencia.

(c). una función que, dada una secuencia, encuentre y devuelva la componente de mayor valor.

(d). una función que, dadas dos secuencias, devuelva un valor que indique si son iguales.

(e). una función que obtenga si alguno de los valores almacenados en la secuencia es igual a la suma del resto de los valores de la misma.

(f). una función que obtenga si alguno de los valores almacenados en la secuencia está repetido.

2. Dado un cadena de caracteres `v1` en el que no hay elementos repetidos y otra cadena `v2` de mayor longitud, se quiere comprobar si todos los caracteres de la cadena `v1` están también en la cadena `v2`. Por ejemplo, si:

`v1` = ‘a’ ‘h’ ‘i’ ‘m’
`v2` = ‘h’ ‘a’ ‘x’ ‘x’ ‘m’ ‘i’

El resultado sería cierto, ya que todos los elementos de `v1` están en `v2`.

3. Implementa un programa que permita realizar operaciones sobre matrices de $N \times N$. El programa debe permitir al usuario la selección de alguna de las siguientes operaciones:

(a). Sumar 2 matrices (y guardar el resultado en una tercera matriz recibida como argumento).

(b). Restar 2 matrices (y devolver el resultado como argumento de salida).

(c). Multiplicar 2 matrices.

(d). Trasponer una matriz (sobre sí misma).

(e). Mostrar una matriz señalando cuáles son los *puntos de silla* (aquellas posiciones que cumplen ser el mínimo de su fila y el máximo de su columna).

Utiliza el tipo `array` de la STL para representar las matrices. Habrá también dos funciones para leer del teclado o mostrar en la pantalla una matriz.

4. Escribe un programa que lea dos cadenas del teclado y determine si una es un anagrama de la otra, es decir, si una cadena es una permutación de los caracteres de la otra. Por ejemplo, “acre” es un anagrama de “cera” y de “arce”. Ten en cuenta que puede haber letras repetidas en la cadena (“carro”, “llave”).

5. Escribe un programa que lea del teclado una frase y a continuación visualice las palabras de la frase en columna, seguida cada una del número de letras que la componen. Implementa tres versiones diferentes:

(a). Sin utilizar ningún `string`, simplemente procesando la entrada carácter a carácter hasta el final (con `cin.get()`).

(b). Leyendo la entrada palabra a palabra con el extractor de cadenas.

(c). Leyendo la frase de una vez con `getline(cin, frase)` y luego procesando el `string`.

6. Para que los cálculos resultasen más fáciles, los romanos solían utilizar una notación aditiva algo distinta de la habitual. Un número romano en esta notación es una cadena de dígitos romanos, `I` = 1, `V` = 5, `X` = 10, `L` = 50, `C` = 100, `D` = 500 y `M` = 1000, ordenados de mayor a menor valor. El valor del número completo es la suma de los valores de sus dígitos. Por ejemplo, 19 se representa como `XVIIII` (10 + 5 + 4 · 1) en lugar de `XIX`.

(a). Escribe una función `romanoAEntero` que reciba una cadena de caracteres introducida por el teclado y devuelva el número arábigo equivalente o un valor que indique que la cadena no se ha podido interpretar como un número romano válido en notación aditiva pura. Así, por ejemplo, si se introduce por teclado `MDCCCCLXXXVIIII` la función devolverá el entero 1989. En cambio, si se introduce por teclado `aX` o `CIX`, devolverá un valor que indique que no es un número romano válido en notación aditiva.

(b). Escribe una función `enteroARomano` que reciba un entero positivo y devuelva su representación como número romano en notación aditiva.

(c). Escribe una función `sumaRomanos` que reciba dos números romanos y devuelva su suma como número romano. El resultado se debe construir en forma simplificada, sin que las repeticiones de un dígito sumen más que el valor del siguiente dígito. Por ejemplo, en lugar de `VV` habría que devolver `X`.

Representar la ausencia de valor

En el caso de `romanoAEntero` se puede devolver un entero negativo o cero para indicar que el argumento no es válido. Una alternativa más general es el tipo `optional<T>` de la biblioteca estándar que puede tomar como valor un elemento de tipo `T` o la constante `nullopt`.

7. Queremos programar el juego de las 3 en raya:

X	O	
	X	X
	O	O

(a). Define una estructura de datos para representar el tablero de juego.

(b). Implementa una función para inicializar el tablero con blancos.

(c). Implementa una función para dibujar el tablero con el contenido de cada posición (`'X'`, `'O'`, `''`).

(d). Implementa una función que devuelva `true` o `false`, dependiendo de si todo el tablero está ocupado o existen casillas libres.

(e). Implementa la función booleana `tresEnRaya(tableroJuego, jugador)` que devuelva `true` en caso de que un jugador tenga la jugada de tres en raya y `false` en caso contrario.

8. Se quiere desarrollar una función que dadas dos matrices cuadradas de enteros, *M* de 10 × 10 y *P* de 3 × 3, compruebe si entre todas las submatrices de 3 × 3 que se pueden formar en la matriz *M*, desplazándose por filas o columnas, existe al menos una que coincida con la matriz *P*. En ese caso, se indicarán la fila y la columna de la matriz *M* en la que empieza la submatriz que coincide. Si hay varias, bastará con indicar la primera.

9. El juego de las 4 en línea consta de un tablero formado por siete columnas y seis filas. En una partida participan dos jugadores, uno con fichas blancas y otro rojas. Inicialmente todas las posiciones del tablero están libres. Cada jugador coloca alternativamente una ficha en una columna. La ficha colocada cae por su propio peso hasta el fondo de la columna correspondiente (primera fila de la columna libre); por ejemplo, en la figura si el jugador Rojo coloca una ficha en la columna 2, la ficha se coloca en la fila 3. La partida la gana el jugador que coloque en primer lugar cuatro de sus fichas en línea horizontal, vertical o en diagonal. La partida queda en tablas si ninguno de los jugadores es capaz de alinear cuatro fichas después de llenar el tablero.

6	L	L	L	L	L	L	L
5	L	L	L	L	L	L	L
4	L	L	L	■	L	L	L
3	L	L	■	□	■	L	L
2	L	■	□	□	□	■	L
1	L	■	■	□	□	□	■
	1	2	3	4	5	6	7

(a). Codifica las estructuras de datos necesarias para jugar una partida.

(b). Escribe una función `inicializarJuego()` que quite todas las fichas del tablero y prepare el tablero de juego.

(c). Escribe una función `colocarFicha()` que dado un jugador (blanco o rojo) y una columna (1 a 7), coloque la ficha en la posición correspondiente.

(d). Escribe una función `presentarTablero()` que visualice en la pantalla el estado del tablero (como en la figura, excepto la rejilla).

(e). Escribe una función `comprobarGanador()` que, dado un jugador (blanco o rojo) y una determinada casilla (1 a 6, 1 a 7), determine si hay cuatro fichas del mismo jugador alineadas en horizontal.

10. En un archivo de texto `inventario.txt` se guarda información acerca de los productos existentes en un almacén. Para cada producto se guardan los siguientes datos:

- Código del producto (4 dígitos).
- Nombre del producto.
- Número de unidades (entero).
- Precio por unidad (real).

El archivo está ordenado por orden creciente del código del producto.

Se dispone además de otro archivo, `modificaciones.txt`, en el que se guardan las modificaciones en el número de artículos que se han producido a lo largo de todo un día. Cada componente del archivo `modificaciones.txt` consta de los siguientes campos:

- Código del producto (4 dígitos).
- Código de operación: venta (V), compra (C), o devolución (D)
- Número de unidades (entero).

El archivo `modificaciones.txt` está organizado secuencialmente y no se encuentra ordenado de ninguna manera especial. Puede haber varias modificaciones relativas al mismo producto. Es decir, el producto de código 2331 puede tener asociadas, por ejemplo, tres componentes del archivo `modificaciones.txt`: la primera puede tratarse de la devolución de 100 unidades, la segunda de la devolución de 50 unidades y la tercera de la venta de 550 unidades. Se pide:

(a). Efectuar las declaraciones necesarias para almacenar los datos necesarios.

(b). Implementar una función que almacene la información del fichero `inventario.txt` en las estructuras de datos correspondientes.

(c). Implementar una función que guarde en fichero la información contenida en la lista del inventario.

(d). Implementar una función que dado un código de producto devuelva la posición del mismo en la lista del inventario usando la búsqueda binaria.

(e). Implementar una función que guarde en un nuevo archivo `inventarioActualizado.txt` la actualización del archivo `inventario.txt` inicial usando el archivo `modificaciones.txt` (las operaciones de venta disminuyen el número de unidades y las de compra y devolución lo aumentan).