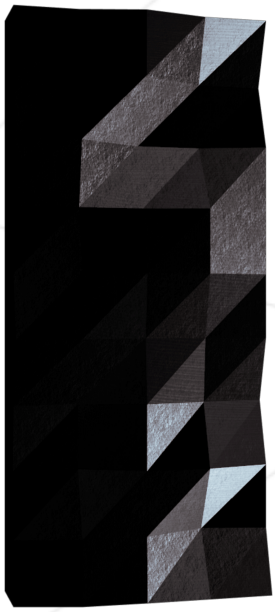

C++ - Módulo 08

Plantilla de contenedor, iteradores, algoritmos

Resumen: Este documento contiene la evaluación del módulo 08 de los módulos C++ de 42.

Índice general

Capítulo I

Reglas Generales

- La declaración de una función en un header (excepto para los templates) o la inclusión de un header no protegido conllevará un 0 en el ejercicio.
- Salvo que se indique lo contrario, cualquier salida se mostrará en stdout y terminará con un newline.
- Los nombres de ficheros impuestos deben seguirse escrupulosamente, así como los nombres de clase, de función y de método.
- Recordatorio : ahora está codificando en `C++`, no en `C`. Por eso :
  - Las funciones siguientes están **PROHIBIDAS**, y su uso conllevará un 0: `*alloc`, `*printf` et `free`
  - Puede utilizar prácticamente toda la librería estándar. NO OBSTANTE, sería más inteligente intentar usar la versión para C++ que a lo que ya está acostumbrado en C, para no basarse en lo que ya ha asimilado. Y no está autorizado a utilizar la STL hasta que le llegue el momento de trabajar con ella (módulo 08). Eso significa que hasta entonces no se puede utilizar Vector/List/Map/etc... ni nada similar que requiera un include <algorithm>.
- El uso de una función o de una mecánica explícitamente prohibida será sancionado con un `0`
- Tenga también en cuenta que, a menos que se autorice de manera expresa, las palabras clave `using namespace` y `friend` están prohibidas. Su uso será castigado con un `0`.
- Los ficheros asociados a una clase se llamarán siempre `ClassName.cpp` y `ClassName.hpp`, a menos que se indique otra cosa.
- Tiene que leer los ejemplos en detalle. Pueden contener prerrequisitos no indicados en las instrucciones.
- No está permitido el uso de librerías externas, de las que forman parte `C++11`, `Boost`, ni ninguna de las herramientas que ese amigo suyo que es un figura le ha recomendado.
- Probablemente tenga que entregar muchos ficheros de clase, lo que le va a parecer repetitivo hasta que aprenda a hacer un script con su editor de código favorito.

- Lea cada ejercicio en su totalidad antes de empezar a resolverlo.
- El compilador es `clang++`
- Se compilará su código con los flags `-Wall -Wextra -Werror`
- Se debe poder incluir cada include con independencia de los demás include. Por lo tanto, un include debe incluir todas sus dependencias.
- No está obligado a respetar ninguna norma en `C++`. Puede utilizar el estilo que prefiera. Ahora bien, un código ilegible es un código que no se puede calificar.
- Importante: no va a ser calificado por un programa (a menos que el enunciado especifique lo contrario). Eso quiere decir que dispone cierto grado de libertad en el método que elija para resolver sus ejercicios.
- Tenga cuidado con las obligaciones, y no sea zángano; podría dejar escapar mucho de lo que los ejercicios le ofrecen.
- Si tiene ficheros adicionales, no es un problema. Puede decidir separar el código de lo que se le pide en varios ficheros, siempre que no haya moulinette.
- Aun cuando un enunciado sea corto, merece la pena dedicarle algo de tiempo, para asegurarse de que comprende bien lo que se espera de usted, y de que lo ha hecho de la mejor manera posible.

Capítulo II

Day-specific rules

- Se dará cuenta de que, en este enunciado en particular, muchos de los problemas que le pedimos que resuelva se pueden resolver sin los algoritmos o sin los contenedores de la STL. No obstante, su objetivo es utilizarlos. Si no hace el esfuerzo de utilizar los contenedores y los algoritmos, tendrá una mala nota. No sea vago.

Capítulo III

Ejercicio 00: Easy find

	Ejercicio : 00
Easy find	
Directorio de entrega : $ex00/$	
Ficheros a entregar : `easyfind.hpp main.cpp`	
Funciones prohibidas : `Ninguna`	

Un ejercicio fácil para empezar con buen pie...
Cree una plantilla de función que se llame `easyfind`, basada en el tipo T, que reciba un T y un int.
T es un contenedor de int. Encuentre la primera aparición del segundo parámetro dentro del primer parámetro. Si no la encuentra, gestione el error utilizando una excepción o un valor de retorno. Inspírese en los contenedores que ya existen.
Entregue un `main` que pruebe de forma exhaustiva su código.

Capítulo IV

Ejercicio 01: Span

	Ejercicio : 01
Span	
Directorio de entrega : $ex01/$	
Ficheros a entregar : `span.cpp span.hpp main.cpp`	
Funciones prohibidas : `Ninguna`	

Cree una clase que pueda almacenar N `int`s. N será un unsigned int y será pasado al constructor como su único parámetro.

Esta clase tendrá una función para almacenar un número único llamada (`addNumber`), que se utilizará para llenarla. Si ya hay N números almacenados y se intenta añadir uno más se considerará un error y se lanzará una excepción.

Ahora, cree dos funciones: `shortestSpan` y `longestSpan`. Estas funciones encontrarán la diferencia más pequeña y más grande, respectivamente, entre los números que contiene el objeto y las devolverán. Si no hay un ningún número almacenado o solo uno, no se podrá encontrar ninguna diferencia y tendrá que lanzar una excepción.

Abajo encontrará un ejemplo (demasiado corto) de `main` y el output esperado. Entregue su propio `main` para la evaluación que, por supuesto, tendrá que ser mucho más completo. Debe probar con un mínimo de 10000 números. Estaría bien que probara con aún más números. También estaría bien probar números utilizando un rango de iteradores, lo que evitaría realizar miles de llamadas a `addNumber`...

```cpp
int main()
{
Span sp = Span(5);

sp.addNumber(5);
sp.addNumber(3);
sp.addNumber(17);
sp.addNumber(9);
sp.addNumber(11);

std::cout << sp.shortestSpan() << std::endl;
std::cout << sp.longestSpan() << std::endl;
}
```

```
$> ./ex01
2
14
$>
```

Capítulo V

Ejercicio 02: Abominación mutante

	Ejercicio : 02
Abominación mutante	
Directorio de entrega : $ex02/$	
Ficheros a entregar : `mutantstack.cpp mutantstack.hpp main.cpp`	
Funciones prohibidas : `Ninguna`	

Hemos acabado con los aperitivos, ahora pasemos a las cosas serias.

El contenedor `std::stack` MOLA UN MONTÓN, pero es uno de los pocos contenedores de la STL que no es iterable. Qué lástima. ¿Porque deberíamos conformarnos con ello si podemos destrozarlo todo y añadir lo que nos plazca?
Tiene que añadirle esta facultad al contenedor `std::stack`, para corregir esa gran injusticia.

Cree la clase `MutantStack` que se parecerá mucho al contenedor `std::stack`, pero además ofrecerá un iterador.

He aquí un ejemplo de código cuyo output debería ser el mismo que si hubiésemos utilizado una `std::list`.
Por supuesto, tendrá que entregar un `main` con más contenido que el que le estamos proponiendo.

```
int main()
{
MutantStack<int>    mstack;

mstack.push(5);
mstack.push(17);

std::cout << mstack.top() << std::endl;

mstack.pop();

std::cout << mstack.size() << std::endl;

mstack.push(3);
mstack.push(5);
mstack.push(737);
//[...]
mstack.push(0);

MutantStack<int>::iterator it = mstack.begin();
MutantStack<int>::iterator ite = mstack.end();

++it;
--it;
while (it != ite)
{
    std::cout << *it << std::endl;
    ++it;
}
std::stack<int> s(mstack);
return 0;
}
```

Capítulo VI

Ejercicio 03: Abra su mente, pero por favor, no la reviente

	Ejercicio : 03
Abra su mente, pero por favor no la joda	
Directorio de entrega : $ex03/$	
Ficheros a entregar : `main.cpp + Lo que quiera`	
Funciones prohibidas : `Ninguna`	

`Este ejercicio no puntúa, pero puede resultar interesante para su piscina. No está obligado a hacerlo.`

Brainfuck es un lenguaje de programación que mola mucho. Al contrario de lo que todo el mundo piensa, no significa .ªbra su mente". Significaría más bien "tener relaciones íntimas con su cerebelo".

Nos gustaría que hiciese un intérprete para Brainfuck, pero eso implicaría que tendría que escribir muchas veces "fuckz, como no nos gusta decir palabrotas, preferiríamos que no escribiese tantas veces "fuck". Porque, como ya sabe, "fuck.^{es} un término un poco vulgar y no sería muy profesional que apareciese escrito "fuck.^{en} su código. Además, si escribiésemos "fuck.^{en} este enunciado, estaríamos cayendo en el humor más rastrero para intentar que le interesase un poco más lo que está haciendo aquí. Así que no. No lo vamos a llamar Brainfuck en esta evaluación. Eso implicaría escribir "fuck", aquí mismo, en esta misma frase. Sería una lástima. A ver, imagínese escribir varias veces "fuck.^{en} un proyecto solo para provocar. ¿Quién haría eso?

Por lo tanto, en lugar de programar un intérprete Brainfuck, va a programar un intérprete Mindopen. Me preguntará, ¿pero qué es Mindopen? Es la forma de escribir Brainfuck sin tener que escribir "fuck".

En primer lugar, leerá cosas sobre Brainfuck (mire en Google). Después, definirá su lenguaje Mindopen, para lo que tomará las mismas instrucciones que Brainfuck, a las que asignará otros símbolos nuevos.

A continuación, escribirá un programa que realice las siguientes tareas:

- Abrir un archivo que contenga código Mindopen
- Leer el archivo y, para cada instrucción descifrada, crear un objeto derivado de la instrucción que represente la instrucción que se tiene que ejecutar, y ponerlo en una encólelo en una... bueno, en una cola de instrucciones en memoria.
- Cerrar el archivo
- Ejecutar cada una de las instrucciones de la fila de espera

Si no le parece obvio, quiere decir que también tiene que crear un conjunto de clases de Instrucción, una para cada instrucción del lenguaje, y todas tendrán un método como `execute` o algo que ejecute la instrucción. Probablemente, también necesitará una interfaz para manipular todas esas instrucciones y almacenarlas en el mismo contenedor...

Se espera que entregue un `main` completo y detallado al igual que archivos de pruebas que sean verdaderos programas Mindopen que se puedan ejecutar.

Capítulo VII

Ejercicio 04: In Poland, expression evaluates you

	Ejercicio : 04
In Poland, expression evaluates you	
Directorio de entrega : $ex04/$	
Ficheros a entregar : `main.cpp + Lo que necesite`	
Funciones prohibidas : `Ninguna`	

`Este ejercicio no puntúa, pero puede resultar interesante para su piscina. No está obligado a hacerlo.`

En este último ejercicio, tendrá que realizar un programa que reciba una expresión matemática como argumento. En esa expresión solo encontrará paréntesis, valores int (que quepan en un `int`) y los operadores $+ - /*$.

Lo primero que tendrá que hacer es tokenizar esa expresión, es decir convertirla en un conjunto de objetos derivados de `Token` y convertir éstos en postfix (o sea, en notación polaca inversa)

Una vez terminado, tendrá que evaluar la expresión, mostrando todas las etapas en la salida estándar. Cuando hablamos de ."etapa", nos referimos al input recibido, a la operación resultante y al resultado en sí.

Por supuesto, tendrá que gestionar los errores de forma apropiada.
Tiene que detallar el `main` y debe ser fácil de leer.

He aquí un ejemplo de output:

```
$> ./ex04 "3 + ((1 + 4) * 2) - 1"
Tokens: Num(3) Op(+) ParOpen ParOpen Num(1) Op(+) Num(4) ParClose Op(*) Num(2) ParClose Op(-) Num(1)
Postfix : Num(3) Num(1) Num(4) Op(+) Num(2) Op(*) Op(+) Num(1) Op(-)
I Num(3) | OP Push    | ST 3]
I Num(1) | OP Push    | ST 1 3]
I Num(4) | OP Push    | ST 4 1 3]
I Op(+) | OP Add      | ST 5 3]
I Num(2) | OP Push    | ST 2 5 3]
I Op(*) | OP Multiply | ST 10 3]
I Op(+) | OP Add      | ST 13]
I Num(1) | OP Push    | ST 1 13]
I Op(-) | OP Substract | ST 12]
Result : 12
```