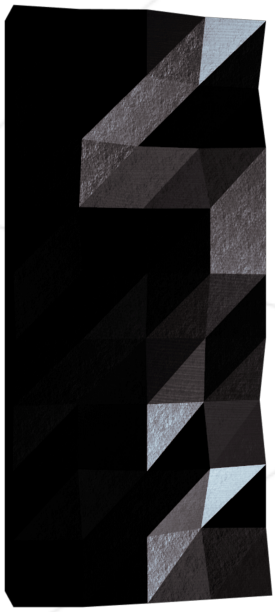

C++ - Módulo 01

Reserva de memoria, referencia, punteros miembro, file streams

Resumen: Este documento contiene la evaluación del módulo 01 de la piscina 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

Ejercicio 00: Variedades de cuadrípedos

	Ejercicio : 00
Variedades de cuadrípedos	
Directorio de entrega : $ex00/$	
Ficheros a entregar : `Pony.cpp Pony.hpp main.cpp`	
Funciones prohibidas : `Ninguna`	

Un ejercicio fácil, para empezar.

Cree una clase `Pony`, que contenga lo necesario para describir a un poni. (Incluya lo que quiera).
Después, cree dos funciones `ponyOnTheStack` y `ponyOnTheHeap` con las que reservará memoria para un `Pony`. Procure también que haga algunas cosas.

Por supuesto, tendrá que reservar la memoria del primer `Pony` en el stack y la del segundo en el heap.

Debe entregar un `main` con pruebas que demuestren que ha entregado algo que funciona. En ambos casos, cuando salga de la función el objeto `Pony` tendrá que haber desaparecido. Su main tendrá que demostrarlo.

Capítulo III

Ejercicio 01: Problema de cañerías

	Ejercicio : 01
Problema de cañerías	
Directorio de entrega : $ex01/$	
Ficheros a entregar : `ex01.cpp`	
Funciones prohibidas : `Ninguna`	

Otro ejercicio fácil.

Cuando haya corregido la fuga de memoria, entregue las funciones que se le han facilitado en el enunciado.

Por supuesto, esperamos que trabaje con la reserva/liberación de memoria. Si solo quita una variable u otra cosa sin resolver el problema, se considerará una respuesta incorrecta.

```
void        memoryLeak()
{
        std::string*        panthere = new std::string("String panthere");

        std::cout << *panthere << std::endl;
}
```

Capítulo IV

Ejercicio 02: Machacando cerebros

	Ejercicio : 02
Machacando cerebros	
Directorio de entrega : $ex02/$	
Ficheros a entregar : `Zombie.cpp Zombie.hpp ZombieEvent.cpp ZombieEvent.hpp main.cpp`	
Funciones prohibidas : `Ninguna`	

Empiece creando la clase `Zombie`. Añádale un tipo y un nombre (como mínimo) y también una función miembro `advert()` que muestre algo parecido a:

```
<nom (type)> Braiiiiiiinnnssss ...
```

Lo que quiera, de verdad, siempre y cuando indique el nombre y el tipo de `Zombie`.

Después, cree la clase `ZombieEvent`. Tendrá un `setZombieType` que almacenará un tipo en el objeto y una función `Zombie * newZombie (std::string name)` que creará un `Zombie` con el tipo elegido. Dele un nombre y devuélvalo.

Escriba también una función `randomChump` que cree un `Zombie` con un nombre aleatorio. Después haga que se presente con `announce()`. Puede elaborar el método .ªleatorio"que más le guste. Los nombres pueden crearse de forma aleatoria o ser elegidos al azar en un grupo de nombres.

Tiene que entregar un programa completo que incluya el `main` y elementos suficientes para demostrar que todo ha funcionado como quería. Por ejemplo, haga que su zombi se presente.

Y ahora, el verdadero objetivo del ejercicio: sus `Zombies` tienen que ser destruidos en el momento adecuado (es decir, cuando ya no sean necesarios). También se les tiene que reservar la memoria de forma adecuada: a veces, lo más apropiado es tenerlos en el stack; en otros casos, es preferible elegir el heap. Para obtener una nota positiva, tendrá que justificar lo que ha realizado.

Capítulo V

Ejercicio 03: ¡Otra tanda de cerebros!

	Ejercicio : 03
¡Otra tanda de cerebros!	
Directorio de entrega : $ex03/$	
Ficheros a entregar : `Zombie.cpp Zombie.hpp ZombieHorde.cpp ZombieHorde.hpp main.cpp`	
Funciones prohibidas : `Ninguna`	

Cree una clase `ZombieHorde` reutilizando la clase `Zombie` del ejercicio anterior.

Esta clase tendrá un constructor que recibirá un entero `n`. En el momento de su creación, tendrá que reservar memoria para `n` objetos zombis, con un nombre aleatorio (lo mismo que en el ejercicio anterior) y almacenarlos.
Implemente también un método `announce()` que llame a la función `announce()` de cada uno de los zombis almacenados.

Tendrá que reservar la memoria de los objetos `Zombie` mediante una reserva única y destruirlos cuando se destruya `ZombieHorde`.

Entregue un `main` con las pruebas que justifiquen sus decisiones.

Capítulo VI

Ejercicio 04: HOLA, LES HABLA EL CEREBRO

	Ejercicio : 04
HOLA, LES HABLA EL CEREBRO	
Directorio de entrega : $ex04/$	
Ficheros a entregar : `ex04.cpp`	
Funciones prohibidas : `Ninguna`	

Escriba un programa que cree una cadena que incluya `"HI THIS IS BRAIN"`, un puntero a esa cadena y una referencia.
Tendrá que mostrarla utilizando el puntero y después utilizando la referencia.
Nada más. No hay trampas.

Capítulo VII

Ejercicio 05: HI BRAIN THIS IS HUMAN

	Ejercicio : 05
HI BRAIN THIS IS HUMAN	
Directorio de entrega : $ex05/$	
Ficheros a entregar : `Brain.cpp Brain.hpp Human.cpp Human.hpp main.cpp`	
Funciones prohibidas : `Ninguna`	

Cree una clase `Brain` con todo lo que considere digno de un cerebro. Tendrá que tener una función `identifier()` que devuelva una cadena con la dirección del cerebro en la memoria, en formato hexadecimal y con el prefijo 0x (Por ejemplo, `"0x194F87EA"`).

Después, cree una clase `Human`, que tenga un atributo constante `Brain`, con la misma vida. Tendrá una función `identifier()` que simplemente llama a la función `identifier()` de su `cerebro` y devuelve su resultado.

Ahora, asegúrese de que este código compila y muestra dos direcciones idénticas:

```
int main()
{
        Human        bob;

        std::cout << bob.identify() << std::endl;
        std::cout << bob.getBrain().identify() << std::endl;
}
```

Tendrá que entregar este código como `main` y justificar todo lo que ha añadido a las clases `Human` o `Brain` para conseguir que funcione (Con argumentos que no sean "Pues, he estado cacharreando hasta que ha funcionado").

Capítulo VIII

Ejercicio 06: Unnecessary violence

	Ejercicio : 06
Unnecessary violence	
Directorio de entrega : $ex06/$	
Ficheros a entregar : `Weapon.cpp Weapon.hpp HumanA.cpp HumanA.hpp HumanB.cpp HumanB.hpp main.cpp`	
Funciones prohibidas : `Ninguna`	

Cree una clase `Weapon` que contenga una cadena `type` y un `getType` que reenvíe una referencia const a dicha cadena. Añada también un `setType`.

Ahora, cree dos clases, `HumanA` et `HumanB`, que tengan las dos un `Weapon`, un nombre y una función `attack()` que muestre algo parecido a:

```
NAME attacks with his WEAPON_TYPE
```

Haga que el siguiente código genere ataques con çrude spiked club"Y DESPUÉS "some other type of club", en los dos tipos de prueba:

```
int main()
{
	{
		Weapon		club = Weapon("crude spiked club");

		HumanA bob("Bob", club);
		bob.attack();
		club.setType("some other type of club");
		bob.attack();
	}
	{
		Weapon		club = Weapon("crude spiked club");

		HumanB jim("Jim");
		jim.setWeapon(club);
		jim.attack();
		club.setType("some other type of club");
		jim.attack();
	}
}
```

¿En qué caso es preferible almacenar `Weapon` como puntero? ¿y como referencia? ¿Por qué? ¿Cuál sería la mejor opción teniendo en cuenta lo que se le ha pedido? Estas son las preguntas que se tendría que hacer antes de empezar el ejercicio.

Capítulo IX

Ejercicio 07: Sed es para los perdedores

	Ejercicio : 07
Sed es para los perdedores	
Directorio de entrega : $ex07/$	
Ficheros a entregar : `Makefile y todo lo que necesite`	
Funciones prohibidas : `Ninguna`	

Cree un programa que se llame `replace` y que reciba un nombre de archivo y dos cadenas, llamémoslas s1 y s2, que NO estén vacías.

El programa abrirá el archivo y escribirá su contenido en FILENAME.replace, remplazando cada aparición de s1 por s2.

Por supuesto, gestionará los errores lo mejor que pueda y no utilizará las funciones de manipulación de archivos del C, porque sería hacer trampas y eso está muy feo, ¿vale?

Entregará los archivos de prueba para demostrar que su programa funciona.

Capítulo X

Ejercicio 08: ¡Los switchs no pasarán!

	Ejercicio : 08
¡Los switchs no pasarán!	
Directorio de entrega : $ex08/$	
Ficheros a entregar : `Human.hpp Human.cpp main.cpp`	
Funciones prohibidas : `Ninguna`	

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

Utilice la siguiente clase `Human`:

```
class Human
{
        private:
        void        meleeAttack(std::string const & target);
        void        rangedAttack(std::string const & target);
        void        intimidatingShout(std::string const & target);

        public:
        void        action(std::string const & action_name, std::string const & target);
};
```

Desarrolle todas estas funciones. Las tres primeras solo mostrarán algo en la salida estándar para que pueda comprobar que han sido llamadas. La última tendrá que llamar a la acción adecuada sobre el objetivo adecuado. Tendrá que utilizar una tabla de punteros a miembros para seleccionar la función que hay que llamar: está prohibido usar las instrucciones `if` o `switch`.

Capítulo XI

Ejercicio 09: El log muere de éxito

	Ejercicio : 09
El log muere de éxito	
Directorio de entrega : $ex09/$	
Ficheros a entregar : `Logger.cpp Logger.hpp main.cpp`	
Funciones prohibidas : `Ninguna`	

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

Cree una clase `Logger` que, obviamente, escribirá logs.

Tendrá dos funciones privadas, `logToConsole` y `logToFile`, que recibirán ambas una cadena. La primera función escribirá la cadena en la salida estándar y la segunda la añadirá a un archivo, cuyo nombre se almacenará en el `Logger` en el momento de la creación.

Cree también una función privada que se llame `makeLogEntry`, que reciba un mensaje sencillo en forma de cadena de caracteres, y que devuelva una cadena nueva con el mensaje en un formato que se parezca a un log verdadero. Como mínimo añada la fecha del día anterior al mensaje para que podamos ver cuándo fue guardado.

Por último, cree un `log(std::string const &dest, std::string const &message)` que creará un log con el mensaje y lo transmitirá a `logToFile` o a `logToConsole`, en función del parámetro `dest`. Igual que en el ejercicio anterior, tendrá que utilizar punteros a miembros para seleccionar la función que hay que llamar.

Capítulo XII

Ejercicio 10: Cat o' nine tails

	Ejercicio : 10
Cat o' nine tails	
Directorio de entrega : $ex10/$	
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.`

Escriba un programa que se llame `cato9tails` que haga lo mismo que el sistema `cat`, sin las opciones. Podrá realizar lecturas de archivos y/o o desde la entrada estándar. Cuidado con las pruebas. No es tan fácil como parece.