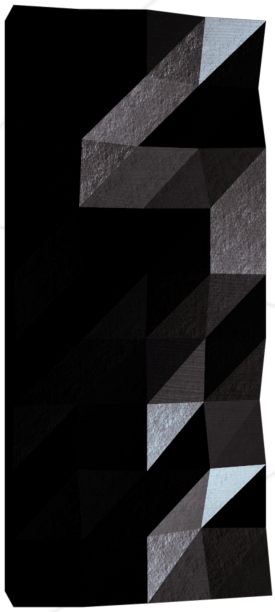

C++ - Módulo 01

Reserva de memoria, referencias, punteros a miembros, switch

Resumen: Este documento contiene el enunciado del módulo 01 de la piscina de C++ de 42.

Índice general

Capítulo I

Instrucciones generales

Para los módulos de C++ utilizarás y aprenderás exclusivamente C++98. Tu objetivo es aprender las nociones de la programación orientada a objetos. Sabemos que las últimas versiones de C++ son muy diferentes en muchos aspectos, si quieres volverte un experto en C++ deberás aprender C++ moderno más adelante. Este es el principio de tu largo viaje por C++, es cosa tuya ir más allá después del common core.

- Cualquier función implementada en un header (excepto en el caso de templates), y cualquier header desprotegido, significa un 0 en el ejercicio.
- Todos los output deben ir al standard output, y deben terminar con un salto de línea, salvo que se indique lo contrario.
- Se deben seguir los nombres de archivos impuestos al pie de la letra, así como las clases, funciones y métodos.
- Recuerda: estás programando en `C++`, no en `C`. Por lo tanto:
    - Las siguientes funciones están PROHIBIDAS, y su uso será sancionado con un `0`, sin preguntas: `*alloc`, `*printf` y `free`.
    - Tienes permitido utilizar básicamente todo de la librería estándar. SIN EMBARGO, sería inteligente utilizar la versión de C++ de las funciones a las que estás acostumbrado en C, en lugar de simplemente seguir utilizando lo que sabes... En realidad, estás aprendiendo un lenguaje nuevo. Y NO, no tienes permitido utilizar STL hasta que debas hacerlo (es decir, el módulo 08). Esto significa que nada de vectors/ lists/maps/etc. O nada que requiera un "include <algorithm>" hasta entonces.
- De hecho, el uso de cualquier función o mecánica explícitamente prohibida será recompensada con un `0`, sin preguntas.
- Ten en cuenta que, salvo especificado de otro modo, las palabras de `C++ üsing namespace"` y `"friend"` están terminantemente prohibidas. Su uso será sancionado con `-42`, sin preguntas.
- Los archivos asociados con una clase se llamarán siempre `ClassName.hpp` y `ClassName.cpp`, salvo especificado de otro modo.
- Entrega en directorios denominados `ex00/`, `ex01/` ... `exn/`.

- Debes leer los ejemplos con cuidado. Pueden contener requisitos no tan obvios en la descripción del ejercicio.
- Dado que tienes permitido utilizar herramientas de `C++` que llevas aprendiendo desde el principio, no tienes permitido utilizar librerías externas. Y antes de que te lo preguntes, esto significa que ningún derivado de `C++11`, ni `Boost` o similares se permite.
- Puede que se requiera entregar un importante número de clases. Esto puede parecer tedioso, salvo que sepas instalar scripts en tu editor de texto favorito.
- Lee cuidadosamente los ejercicios POR COMPLETO antes de empezarlos. En serio, hazlo.
- El compilador a usar es `clang++`.
- Tu código debe compilar con las flags: `-Wall -Werror -Wextra`.
- Cada uno de tus `include` debe poder incluirse independientemente del resto. Los `include` deben contener obviamente los `include` de los que dependan.
- Por si te lo preguntas, no se requiere ningún estilo de código durante `C++`. Puedes utilizar una guía de estilos que te guste, sin limitaciones. Recuerda que si tu evaluador no es capaz de leer tu código, tampoco lo será de evaluarte.
- Algo importante: NO te evaluará un programa, salvo que el subject lo indique explícitamente. Por lo tanto, tienes cierta libertad en cómo hagas los ejercicios. Sin embargo, sé inteligente con los principios de cada ejercicio, y NO seas perezoso, te perderás MUCHO de lo que estos proyectos te pueden ofrecer.
- No es un problema real si tienes archivos adicionales a los que se te solicita, puedes elegir separar el código en más archivos de los que se te piden. Siéntete libre, siempre y cuando el resultado no lo evalúe un programa.
- Aunque el subject de un ejercicio sea corto, merece la pena gastar algo de tiempo para estar absolutamente seguro de que entiendes lo que se espera que entiendas, y que lo has hecho de la mejor forma posible.
- Por Odin, por Thor. Utiliza tu cerebro.

Capítulo II

Ejercicio 00: BraiiiiiiinnnzzzZ

	Ejercicio: 00
BraiiiiiiinnnzzzZ	
Directorio de entrega: *ex*00/	
Archivos de entrega: `Makefile, main.cpp, Zombie.cpp, Zombie.hpp, newZombie.cpp, randomChump.cpp`	
Funciones prohibidas: `Ninguna`	

Primero, haz una clase `Zombie`. Los zombies tienen un nombre privado y son capaces de anunciarse (`announce`) a ellos mismos.

`<name> BraiiiiiiinnnzzzZ...`

Sí, `announce(void)` es una función miembro. Añade también un mensaje de debug en el destructor que incluya el nombre del `Zombie`.

Tras esto, escribe una función que cree un `Zombie`, lo nombre, y lo devuelva para utilizarse en otro punto de tu código. El prototipo de la función es el siguiente:

```
Zombie* newZombie( std::string name );
```

Tendrás que programar otra función que cree un `Zombie`, y lo haga anunciarse a sí mismo. El prototipo de la función deberá ser:

```
void randomChump( std::string name );
```

Ahora vamos al objetivo real del ejercicio: tus `Zombies` deben destruirse cuando toque (es decir, cuando no se necesiten). Deben estar localizados en el stack o en el heap dependiendo de su uso: a veces lo más apropiado es tenerlos en el stack, y otras veces el heap parece la mejor elección.

Capítulo III

Ejercicio 01: ¡Másss cerebross!

	Ejercicio: 01
¡Másss cerebross!	
Directorio de entrega: *ex*01/	
Archivos de entrega: `Makefile, main.cpp, Zombie.cpp, Zombie.hpp, ZombieHorde.cpp`	
Funciones prohibidas: `Ninguna`	

Vamos a reutilizar la clase `Zombie`, pero ahora crearemos una horda de zombies.

Escribe una función que acepte un entero N. Cuando se llame, reservará `N` objetos `Zombie`. Debe reservar cada uno de los `N` objetos `Zombie` de golpe. Después debes iniciar cada `Zombie` dándole un nombre. Por último, debe devolver un puntero al primer `Zombie`. La función debe prototiparse así:

```
Zombie* zombieHorde( int N, std::string name );
```

Entrega un main que demuestre que tu función `zombieHorde` funciona como debe. Quizá quieres hacerlo llamando `announce()` en cada uno de los `Zombies`. No te olvides de eliminar **todos** los `Zombies` cuando ya no los necesites.

Capítulo IV

Ejercicio 02: HI THIS IS BRAIN

	Ejercicio: 02
HI THIS IS BRAIN	
Directorio de entrega: *ex*02/	
Archivos de entrega: `Makefile, main.cpp`	
Funciones prohibidas: `Ninguna`	

Haz un programa en el que crees una string que contenga `"HI THIS IS BRAIN"`. Crea un `stringPTR` que apunte a la string; y una `stringREF` que referencie la string.

Ahora, muestra la dirección en memoria de la string. Después, muestra la dirección de la string utilizando `stringPTR` y `stringREF`.

Por último, muestra la string utilizando su puntero, y finalmente utilizando su referencia.

Eso es todo, sin trucos. El objetivo de este ejercicio es que desmitifiques las referencias. No es algo completamente nuevo, es otro tipo de sintaxis para algo que ya sabías: direcciones. Aún así, hay algunos detalles minúsculos...

Capítulo V

Ejercicio 03: Violencia innecesaria

	Ejercicio: 03
Violencia innecesaria	
Directorio de entrega: *ex*03/	
Archivos de entrega: `Makefile, main.cpp, Weapon.cpp, Weapon.hpp, HumanA.cpp, HumanA.hpp, HumanB.cpp, HumanB.hpp`	
Funciones prohibidas: `Ninguna`	

Crea una clase `Weapon`, que tenga una string `type`, y un método `getType` que devuelva una referencia constante a esa string. Por supuesto, crea también un `setType`.

Ahora, crea dos clases: `HumanA` y `HumanB`. Ambas tendrán un `Weapon`, un nombre, y una función `attack()` que mostrará:

```
NAME attacks with his WEAPON_TYPE
```

`HumanA` y `HumanB` son prácticamente lo mismo, salvo por un detalle casi irrelevante:

- Mientras que `HumanA` acepta `Weapon` como parte de su constructor, `HumanB` no lo hace.
- `HumanB` no tiene por qué tener siempre un `Weapon`, pero `HumanA` **siempre** estará armado.

Consigue hacer que el siguiente código produzca ataques con una çrude spiked club", **y después** "some other type of club", en ambas pruebas:

```
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 más apropiado guardar `Weapon` como un puntero? ¿Y como referencia? ¿Por qué?
Esas son las preguntas que tienes que hacerte antes de entregar este ejercicio.

Capítulo VI

Ejercicio 04: Sed es para perdedores

	Ejercicio: 04
Sed es para perdedores	
Directorio de entrega: *ex*04/	
Archivos de entrega: `Makefile, main.cpp, and whatever else you need`	
Funciones prohibidas: `Ninguna`	

Crea un programa llamado `replace` que acepte el nombre de un archivo y dos strings, vamos a llamarlas s1 y s2. **No están vacías**.

Abrirá el archivo, y escribirá su contenido en `nombre_de_archivo.replace`, tras reemplazar todas las ocurrencias de s1 y s2 que encuentre.

Todas las funciones miembro de la clase std::string se permiten, excepto `replace`. Utilízalas con inteligencia.

Por supuesto, deberás gestionar errores lo mejor que puedas. No utilices las funciones de manipulación de archivos de C, porque sería hacer trampas y eso está mal.

Deberás entregar un main con tus pruebas para validar el funcionamiento de tu programa.

Capítulo VII

Ejercicio 05: Karen 2.0

	Ejercicio: 05
Karen 2.0	
Directorio de entrega: *ex*05/	
Archivos de entrega: `Makefile, main.cpp, Karen.hpp, and Karen.cpp`	
Funciones prohibidas: `Ninguna`	

¿Conoces a Karen? Todos lo hacemos, ¿no? En caso de que no la conozcas, aquí tienes algunos comentarios que Karen hace:

- Nivel "DEBUG": los mensajes de este nivel contienen información con un contexto extenso. Se utilizan sobre todo para diagnosticar problemas. Por ejemplo: `I love to get extra bacon for my 7XL-double-cheese-triple-pickle-special-ketchup burger. I just love it!`. *Me gustaría algo de bacon extra para mi hamburguesa 7XL-doble-queso-triple-pepinillo-ketchup-especial. Simplemente, me encanta.*

- Nivel ÏNFO": estos mensajes tienen algo de información contextual para ayudarte a rastrear el flujo del programa en un entorno de producción. Por ejemplo: `I cannot believe adding extra bacon cost more money. You don't put enough! If you did I would not have to ask for it!`. *No me puedo creer que añadir bacon extra cueste más dinero. No ponéis suficiente. Si lo hicierais no tendría que pedirlo..*

- Nivel "WARNING": un mensaje de aviso indica un potencial problema en el sistema. El sistema es capaz de gestionar el problema por sí mismo, o de continuar a pesar de él. Por ejemplo: `I think I deserve to have some extra bacon for free. I've been coming here for years and you just started working here last month.`. *Creo que me merezco algo más de bacon gratis. Llevo viniendo años y tú empezaste a trabajar aquí el mes pasado.*

- Nivel .ERROR": un mensaje de error indica un problema serio en el sistema. El problema es habitualmente un estado irrecuperable que requiere intervención manual. Ejemplo: `This is unacceptable, I want to speak to the manager now.`. *Esto es inaceptable. Quiero hablar con el responsable ahora mismo..*

Vamos a automatizar a Karen, de todas formas siempre dice lo mismo. Tendrás que crear una clase llamada `Karen` que contendrá las siguientes funciones miembro privadas:

- `void debug( void );`
- `void info( void );`
- `void warning( void );`
- `void error( void );`

Karen tiene que tener también una función pública que llame a las funciones privadas en base al nivel pasado como parámetro. El prototipo de la función es:

```
void complain( std::string level );
```

El objetivo de este ejercicio es utilizar punteros a funciones miembro. No es una sugerencia, Karen tiene que quejarse sin utilizar un puñado de if/elseif/else. Ella ni duda ni se lo piensa dos veces.

Envía un main que demuestre que Karen se queja un montón. Tanto si utilizas las quejas dadas de ejemplo como si eliges escribir las tuyas propias, está bien.

Capítulo VIII

Ejercicio 06: Vamos a filtrar a Karen

	Ejercicio: 06
Vamos a filtrar a Karen	
Directorio de entrega: *ex*06/	
Archivos de entrega: `Makefile, main.cpp, Karen.hpp, and Karen.cpp`	
Funciones prohibidas: `Ninguna`	

Vamos a implementar un sistema para filtrar si lo que Karen dice es importante o no, a veces no necesitamos prestar atención a todo lo que Karen dice.

Tendrás que escribir el programa `karenFilter` que recibirá como parámetro el nivel de log que quiere escuchar y toda la información que esté por encima. Como ejemplo:

```
$> ./karenFilter "WARNING"
[ WARNING ]
I think I deserve to have some extra bacon for free.
I've been coming here for years an you just started working here last month.

[ ERROR ]
This is unacceptable, I want to speak to the manager now.

$> ./karenFilter "I am not sure how tired I am today..."
[ Probably complaining about insignificant problems ]
```

Hay muchas formas de filtrar a Karen, pero probablemente la mejor es pulsar su *SWITCH* para apagarla ;)