



C++ - Module 01

Reserva de memoria, punteros a miembros,
referencias, switch

Resumen:

Este documento contiene los ejercicios del módulo 01 de C++.

Versión: 9.1

Índice general

I.	Introducción	2
II.	Reglas generales	3
III.	Ejercicio 00: BraiiiiiiinnnzzzZ	5
IV.	Ejercicio 01: Moar brainz!	6
V.	Ejercicio 02: HI THIS IS BRAIN	7
VI.	Ejercicio 03: Violencia innecesaria	8
VII.	Ejercicio 04: Sed es para pringaos	10
VIII.	Ejercicio 05: Harl 2.0	11
IX.	Ejercicio 06: Harl filter	13

Capítulo I

Introducción

C++ Es un lenguaje de programación de propósito general creado por Bjarne Stroustrup como una extensión del lenguaje de programación C, o C con clases"(fuente: [Wikipedia](#)).

El objetivo de estos módulos es presentarte la **Programación orientada a objetos**. Este será el punto de partida de tu viaje en C++. Hay muchas opciones para aprender POO. Decidimos elegir C++ ya que se deriva de tu viejo amigo C y para mantener las cosas simples, debido a la complejidad del lenguaje tu código deberá compilar con el estándar C++98.

Somos consciente, el lenguaje C++ es moderno y diferente en muchos aspectos. Si quieres convertirte en un especialista en este lenguaje, ¡Dependerá de ti ir más allá del 42 Common Core!

Capítulo II

Reglas generales

Compilando

- Compila tu código con `c++` y los flags `-Wall -Wextra -Werror`
- Tu código deberá compilar si además agregas el flag `-std=c++98`

Formato y convención de nomenclatura

- Los directorios de los ejercicios serán nombrados de la siguiente manera: `ex00`, `ex01`, ... , `exn`
- Nombra tus ficheros, clases, funciones, funciones miembros y atributos como lo requieran las reglas.
- Escribe los nombres de las clases con el formato **UpperCamelCase**. Los archivos que contengan código de clase deben ser nombrados de acuerdo con el nombre de la clase. Por ejemplo:
`ClassName.hpp/ClassName.h`, `ClassName.cpp`, o `ClassName.tpp`. Entonces, si tu tienes un archivo header que contiene la definición de la clase "BrickWall" que representa una pared de ladrillos, su nombre será `BrickWall.hpp`.
- Si no se especifica lo contrario, todos los mensajes de salida deben terminar con un salto de línea y mostrarse en la salida estándar.
- ¡Adiós Norminette! No se aplica ningún estilo de codificación en los módulos de C++. Puedes seguir a tu favorito. Pero ten en cuenta que un código que tus evaluadores no puede entender es un código que no pueden calificar. Haz tu mejor esfuerzo para escribir un código limpio y legible.

Permitido/Prohibido

No estas codificando en C. ¡llegó la hora de C++! Por lo tanto:

- Se te permite utilizar casi todo de la biblioteca estándar. De este modo, en lugar de seguir aferrado a lo que ya sabes sería inteligente usar, tanto como sea posible, las versiones de C++ de las funciones de C a las que estás acostumbrado.

- Sin embargo, no puedes usar ninguna otra biblioteca externa. Esto significa que ni C++11 ni ninguno de sus derivados, así como las librerías **Boost** están permitidos. Las siguientes funciones también están prohibidas: `*printf()`, `*alloc()` y `free()`. Si los usas, tu calificación sera un 0 definitivo.
- Ten en cuenta que, a menos que se indique explícitamente lo contrario, el `using namespace <ns_name>` y la palabra reservada `friend` están prohibidas. De lo contrario, tu calificación será -42.
- **Podrás usar la librerías STL únicamente en el modulo 08.** Esto significa: No podrás utilizar **Contenedores** (vector/list/map/etcétera) ni **Algoritmos** (o cualquier cosa que requiera incluir el header `<algorithm>`) hasta ese momento. De lo contrario, tu calificación será un -42.

Algunos requisitos de diseños

- Aunque no lo creas se producen fugas de memoria en C++. Cuando reservas memoria (utilizando la palabra reservada `new`), debes evitar **memory leaks**.
- Desde el Modulo 02 al Modulo 08, Tus clases deben estar diseñadas en la **forma canónica ortodoxa**, excepto cuando expresamente se indique lo contrario.
- Cualquier implementación de una función en un archivo header (excepto para funciones de tipo template) significaría un 0 para el ejercicio.
- Deberías poder usar cada uno de tus headers independientemente de los demás. Por lo tanto, deben incluir todas las dependencias que necesitan. No obstante, debes evitar el problema de la doble inclusión agregando **include guards**. De lo contrario, tu calificación sera un 0.

Léeme

- Puedes agregar algunos archivos adicionales si necesitas (p.ej. para organizar tu código). Como estos ejercicios no serán verificados por un programa, siéntete libre de hacerlo siempre y cuando entregues los archivos obligatorios.
- a veces, los requerimientos de un ejercicio parecen breves, pero los ejemplos pueden mostrar requisitos que no están escritos explícitamente en las instrucciones.
- ¡Lee cada modulo completamente antes de empezarlos! De verdad, hazlo.
- ¡Por Odin!, ¡Por Thor! ¡¡¡Usa tu cerebro!!!




Tendrás que implementar muchas clases. Esto puede ser tedioso, a no ser que domines los scripts en tu editor de texto favorito.



Tienes cierta libertad para completar los ejercicios. Sin embargo, Sigues las reglas obligatorias y no seas un sangana/o. ¡Te perderías mucha infomación útil! No dudes en leer sobre conceptos teóricos.

Capítulo III

Ejercicio 00: BraiiiiiiinnnzzzzZ

	Ejercicio: 00
BraiiiiiiinnnzzzzZ	
Directorio de entrega: <i>ex00/</i>	
Archivos de entrega: <i>Makefile</i> , <i>main.cpp</i> , <i>Zombie.{h, hpp}</i> , <i>Zombie.cpp</i> , <i>newZombie.cpp</i> , <i>randomChump.cpp</i>	
Funciones prohibidas: Ninguna	

Primero, implementa una clase llamada **Zombie**. Esta contiene un atributo privado de tipo string **name**

Agrega una función miembro `void announce(void);` a la clase **Zombie**. Los zombis se anuncian de la siguiente manera:

```
<name>: BraiiiiiiinnnzzzzZ...
```

No imprimas los corchetes angulares (`<y >`). Para un zombi llamado **Foo**, el mensaje sería:

```
Foo: BraiiiiiiinnnzzzzZ...
```

Después, implementa las siguientes funciones:


- `Zombie* newZombie(std::string name);`
Crea un zombi, dale un nombre, y devuélvelo para que puedas usarlo fuera del alcance de la función.
- `void randomChump(std::string name);`
Crea un zombi, dale un nombre, y haz que el zombi se anuncie a si mismo.

Ahora, ¿Cuál es el objetivo real del ejercicio? vas a determinar en que caso es mejor el uso de la memoria para los zombis, si en el stack o heap.

Los zombis deben ser destruidos cuando ya no los necesites. El destructor debe imprimir en la salida estándar un mensaje con el nombre del zombis para fines de depuración.

Capítulo IV

Ejercicio 01: Moar brainz!

	Ejercicio: 01
Moar brainz!	
Directorio de entrega: <i>ex01/</i>	
Archivos de entrega: <code>Makefile</code> , <code>main.cpp</code> , <code>Zombie.{h, hpp}</code> , <code>Zombie.cpp</code> , <code>zombieHorde.cpp</code>	
Funciones prohibidas: Ninguna	

¡Es tiempo de crear una **horda de Zombis!**

Implementa la siguiente función en el archivo correspondiente:

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


Debes reservar `N` numero de objetos de tipo `Zombie` en una única reserva. además, tendrás que inicializar los zombis, dando a cada uno de ellos el nombre pasado como parámetro. La función retornara un puntero al primer zombi.

Crea tus propios main para asegurarte de que tu función `zombieHorde()` trabaja como se espera. intenta llamar `announce()` para cada uno de los zombis.

No olvides **eliminar** a todos los zombis y buscar **memory leaks**.

Capítulo V

Ejercicio 02: HI THIS IS BRAIN

	Ejercicio: 02
HI THIS IS BRAIN	
Directorio de entrega: <i>ex02/</i>	
Archivos de entrega: Makefile , main.cpp	
Funciones prohibidas: Ninguna	

Escribe un programa que contenga:

- Una variable de tipo string inicializada con "HI THIS IS BRAIN".
- **stringPTR**: Un puntero a la string.
- **stringREF**: Una referencia a la string.

Tu programa imprimirá por la salida estándar:

- La dirección de memoria de la variable de tipo string.
- La dirección de memoria contenida en **stringPTR**.
- La dirección de memoria contenida en **stringREF**.


Y a continuación:

- El valor de la variable string.
- El valor apuntado por **stringPTR**.
- El valor apuntado por **stringREF**.

Esto es todo, no hay trucos. El objetivo de este ejercicio es normalizar las referencias que pueden parecer completamente nuevas. Aunque hay algunas pequeñas diferencias, esta es otra sintaxis para algo que ya haces: manejo de direcciones.

Capítulo VI

Ejercicio 03: Violencia innecesaria

	Ejercicio: 03
Violencia innecesaria	
Directorio de entrega: <i>ex03/</i>	
Archivos de entrega: <i>Makefile</i> , <i>main.cpp</i> , <i>Weapon.{h, hpp}</i> , <i>Weapon.cpp</i> , <i>HumanA.{h, hpp}</i> , <i>HumanA.cpp</i> , <i>HumanB.{h, hpp}</i> , <i>HumanB.cpp</i>	
Funciones prohibidas: Ninguna	

Implemente una clase `Weapon` que contenga:

- Un atributo privado de tipo `string` llamado `type`
- Una función miembro `getType()` que retornará una referencia constante a `type`.
- Una función miembro `setType()` que asigne a la variable `type` usando el nuevo valor pasado por parámetro.

Ahora, crea las siguientes clases: **HumanA** y **HumanB**. Ambas tienen un `Weapon` y un `name`. También tienen una función miembro `attack()` que muestra (por supuesto, sin los corchetes angulares):

```
<name> attacks with their <weapon>
```

`HumanA` y `HumanB` son casi iguales excepto por estos dos pequeños detalles:

- Mientras `HumanA` aceptará `Weapon` como parte del constructor, `HumanB` no lo hace.
- `HumanB` **no siempre** tendrá un `Weapon`, mientras que `HumanA` **siempre** estará armado.

Si tu implementación es correcta, al ejecutar el siguiente código imprimirás un ataque “crude spiked club” y después un segundo ataque “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();
    }

    return 0;
}
```


No olvides revisar **memory leaks**.



¿En qué caso crees que sería mejor usar un puntero a Weapon? ¿Y como una referencia a Weapon? ¿por qué? Reflexiona sobre esto antes de comenzar con este ejercicio.

Capítulo VII

Ejercicio 04: Sed es para pringaos

	Ejercicio: 04
Sed es para pringaos	
Directorio de entrega: <i>ex04/</i>	
Archivos de entrega: Makefile , main.cpp , *.cpp , *.{h, hpp}	
Funciones prohibidas: std::string::replace	

Cree un programa que reciba por parámetros tres argumentos en el siguiente orden: Un nombre de archivo y dos strings, **s1** y **s2**.


Este programa abrirá el archivo **<filename>** y copiará el contenido en un nuevo archivo **<filename>.replace**, reemplazando cada ocurrencia de **s1** con **s2**.

El uso de funciones de manipulación de archivos en C está prohibido y se considerará hacer trampa. Todas las funciones miembro de la clase **std::string** están permitidas, excepto **replace**. ¡Úsalos sabiamente!

Por supuesto, debes controlar entradas inesperadas y errores. Tienes que crear y entregar tu propio main para asegurarte de que tu programa funciona como se espera.

Capítulo VIII

Ejercicio 05: Harl 2.0

	Ejercicio: 05
Harl 2.0	
Directorio de entrega: <i>ex05/</i>	
Archivos de entrega: Makefile , main.cpp , Harl.{h, hpp} , Harl.cpp	
Funciones prohibidas: Ninguna	

¿Conoces a Harl? Todos lo hacemos, ¿No? En caso de que no, aquí tienes algunos comentarios que hace Harl. Estos se clasifican por niveles:

- Nivel **"DEBUG"**: Los mensajes de este nivel contienen información contextual. Se utilizan principalmente para el diagnóstico de problemas
Por ejemplo: *"I love having extra bacon for my 7XL-double-cheese-triple-pickle-special-ketchup burger. I really do!"*
- Nivel **"INFO"**: Los mensajes de este nivel contienen información muy extensa. Son útiles para rastrear la ejecución de un programa en un entorno de producción.
Por ejemplo: *"I cannot believe adding extra bacon costs more money. You didn't put enough bacon in my burger! If you did, I wouldn't be asking for more!"*
- Nivel **"WARNING"**: Los mensajes de advertencia indican un posible problema en el sistema. Sin embargo, pueden ser manejados o ignorados.
Por ejemplo: *"I think I deserve to have some extra bacon for free. I've been coming for years whereas you started working here since last month."*
- Nivel **"ERROR"**: Los mensajes de este nivel indican que se ha producido un error irreparable. Este suele ser un problema crítico que requiere intervención manual.
Por ejemplo: *"This is unacceptable! I want to speak to the manager now."*

Vas a automatizar a Harl. No será difícil, siempre dice las mismas cosas. Tendrás que crear una clase **Harl** con las siguientes funciones miembros privadas:

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

Harl además tiene una función miembro pública que llama a las cuatro funciones miembro anteriores dependiendo del nivel pasado como parámetro:


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

El objetivo de este ejercicio es usar los **punteros a funciones miembros**. Esto no es una sugerencia. Harl tiene que quejarse sin usar un batiburrillo de condicionales if/else if/else. ¡No pienses en esto dos veces!

Crea y entrega tus propios tests para demostrar que Harl se queja mucho. Puedes utilizar los comentarios de ejemplos mencionados anteriormente o si lo prefieres, utilizar tus propios comentarios.

Capítulo IX

Ejercicio 06: Harl filter

	Ejercicio: 06
Harl filter	
Directorio de entrega: <i>ex06/</i>	
Archivos de entrega: <i>Makefile</i> , <i>main.cpp</i> , <i>Harl.{h, hpp}</i> , <i>Harl.cpp</i>	
Funciones prohibidas: Ninguna	

A veces no hay que prestar atención a todo lo que comenta Harl. Así que, implementa un sistema para filtrar lo que dice Harl según los niveles de registro que desees escuchar.

Crea un programa que tome como parámetros uno de los cuatro niveles. Imprimirás por la salida estándar todos los mensajes de este nivel y superiores. Por ejemplo:

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

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

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

Hay muchas formas de lidiar con Harl, Una de las más efectivas es pulsar su SWITCH para apagarlo.

El nombre de tu ejecutable será `harlFilter`.

Debes usar, y tal vez descubrir, la sentencia `switch` en este ejercicio.



Puedes aprobar este módulo sin hacer el ejercicio 06.