**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 introducirte en la Programación Orientada a Objetos.

Este será el punto de partida de tu viaje en C++. Se recomiendan muchos lenguajes

para aprender POO. Decidimos elegir C++ ya que deriva de tu viejo amigo C.

Debido a que este es un lenguaje complejo, y con el fin de mantener las cosas simples, su código se

cumplirá con el estándar C++98.

Somos conscientes de que el C++ moderno es muy diferente en muchos aspectos. Así que si quieres

convertirte en un desarrollador C++ competente, depende de ti ir más allá después de los 42 Common

¡Core!

Descubrirás nuevos conceptos paso a paso. Los ejercicios aumentarán progresivamente

en complejidad.

**Capítulo II**

Reglas generales

Compilación

• Compila tu código con C++ y los indicadores -Wall -Wextra -Werror

• Tu código debería compilar incluso si añades el indicador -std=c++98

Formato y convenciones de nomenclatura

• Los directorios de ejercicios se nombrarán de la siguiente manera: ex00, ex01, ..., exn

• Nombra tus archivos, clases, funciones, funciones miembro y atributos según lo requerido en las directrices.

• Escribe los nombres de clase en formato Mayúsculas y minúsculas. Los archivos que contienen código de clase siempre se nombrarán según el nombre de la clase. Por ejemplo: ClassName.hpp/ClassName.h, ClassName.cpp o ClassName.tpp. Si tienes un archivo de encabezado que contiene la definición de la clase "BrickWall" que representa un muro de ladrillos, su nombre será BrickWall.hpp.

• A menos que se especifique lo contrario, todos los mensajes de salida deben terminar con un carácter de nueva línea y mostrarse en la salida estándar.

• ¡Adiós Norminette! No se impone ningún estilo de codificación en los módulos de C++. Puedes seguir tu estilo favorito. Pero recuerda que el código que tus compañeros evaluadores no pueden entender es código que no pueden calificar. Haz lo posible por escribir un código limpio y legible.

Permitido/Prohibido

Ya no estás programando en C. ¡Hora de C++! Por lo tanto:

• Puedes usar casi todo de la biblioteca estándar. Por lo tanto, en lugar de limitarte a lo que ya sabes, sería inteligente usar, en la medida de lo posible, las versiones similares a C++ de las funciones de C que usas.

• Sin embargo, no puedes usar ninguna otra biblioteca externa. Esto significa que C++11 (y sus derivados) y las bibliotecas Boost están prohibidas. Las siguientes funciones también están prohibidas: \*printf(), \*alloc() y free(). Si las usas, tu calificación será 0 y listo.

Tenga en cuenta que, a menos que se indique explícitamente lo contrario, está prohibido usar las palabras clave <ns\_name> y friend en el espacio de nombres. De lo contrario, su calificación será de -42.

Solo se permite usar la STL en los Módulos 08 y 09. Esto significa:

No se permiten contenedores (vector, lista, mapa, etc.) ni algoritmos (cualquier cosa que requiera incluir la cabecera <algorithm>) hasta entonces. De lo contrario, su calificación será de -42.

Algunos requisitos de diseño

Las fugas de memoria también ocurren en C++. Al asignar memoria (usando la palabra clave new), debe evitarlas.

Del Módulo 02 al Módulo 09, sus clases deben diseñarse en la forma canónica ortodoxa, a menos que se indique explícitamente lo contrario.

Cualquier implementación de función incluida en un archivo de cabecera (excepto las plantillas de función)

significa un 0 en el ejercicio.

Debería poder usar cada una de sus cabeceras independientemente de las demás. Por lo tanto, deben incluir todas las dependencias necesarias. Sin embargo, debes evitar el problema de la doble inclusión añadiendo protecciones de inclusión. De lo contrario, tu calificación será 0.

Léeme

• Puedes añadir archivos adicionales si lo necesitas (por ejemplo, para dividir tu código). Como estas tareas no son verificadas por un programa, puedes hacerlo siempre que entregues los archivos obligatorios.

• A veces, las pautas de un ejercicio parecen cortas, pero los ejemplos pueden mostrar requisitos que no están explícitamente escritos en las instrucciones.

• ¡Lee cada módulo completo antes de empezar! ¡En serio, hazlo!

• ¡Por Odín, por Thor! ¡Usa tu cerebro!

En cuanto al Makefile para proyectos de C++, se aplican las mismas reglas que en C (consulta el capítulo de Normas sobre el Makefile).

Tendrás que implementar muchas clases. Esto puede parecer tedioso, a menos que puedas programar tu editor de texto favorito.

Tienes cierta libertad para completar los ejercicios. Sin embargo, sigue las reglas obligatorias y no seas perezoso. ¡Te perderás mucha información útil! No dudes en leer sobre conceptos teóricos.

**Capítulo III**

Ejercicio 00: ¡Ceree ... Esta función crea un zombi, le asigna un nombre y lo devuelve para que puedas usarlo fuera del ámbito de la función.

• void randomChump( std::string name );

Esta función crea un zombi, le asigna un nombre y hace que se anuncie a sí mismo.

Ahora bien, ¿cuál es el objetivo real del ejercicio? Debes determinar en qué caso es mejor asignar zombis en la pila o en el montón.

Los zombis deben destruirse cuando ya no se necesitan. El destructor debe imprimir un mensaje con el nombre del zombi para fines de depuración.

Vas a automatizar Harl. No será difícil, ya que siempre dice lo mismo. Debes crear una clase Harl con las siguientes funciones miembro privadas:

• void debug( void );

• void info( void );

• void Warning( void );

• void Error( void );

Harl también tiene una función miembro pública que llama a las cuatro funciones miembro anteriores, según el nivel pasado como parámetro:

void complaint( std::string level );

El objetivo de este ejercicio es usar punteros a funciones miembro. Esto no es una sugerencia. Harl debe quejarse sin usar un montón de if/else. ¡No lo piensa dos veces!

Crea y entrega pruebas para demostrar que Harl se queja mucho. Puedes usar los ejemplos de comentarios mencionados anteriormente en el asunto o usar tus propios comentarios.

**Capítulo IV**

Ejercicio 01: ¡Más cerebros!

Ejercicio: 01

¡Más cerebros!

Directorio de entrega: ex01/

Archivos a entregar: Makefile, main.cpp, Zombie.{h, hpp}, Zombie.cpp,

zombieHorde.cpp

Funciones prohibidas: Ninguna

¡Es hora de crear una horda de zombis!

Implementa la siguiente función en el archivo correspondiente:

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

Debe asignar N objetos zombi en una sola asignación. Luego, debe inicializar los zombis, asignando a cada uno el nombre pasado como parámetro. La función devuelve un puntero al primer zombi.

Implementa tus propias pruebas para asegurarte de que la función zombieHorde() funcione correctamente. Intenta llamar a Announce() para cada uno de los zombis.

No olvides usar la función "delete" para desasignar todos los zombis y comprobar si hay fugas de memoria.

**Capítulo V**

Ejercicio 02: HOLA, ESTO ES CEREBRO

Ejercicio: 02

HOLA, ESTO ES CEREBRO

Directorio de entrega: ex02/

Archivos a entregar: Makefile, main.cpp

Funciones prohibidas: Ninguna

Escribe un programa que contenga:

• Una variable de cadena inicializada como "HOLA, ESTO ES CEREBRO".

• stringPTR: un puntero a la cadena.

• stringREF: una referencia a la cadena.

Tu programa debe imprimir:

• La dirección de memoria de la variable de cadena.

• La dirección de memoria de stringPTR.

• La dirección de memoria de stringREF.

Y luego:

• El valor de la variable de cadena.

• El valor al que apunta stringPTR.

• El valor al que apunta stringREF.

Eso es todo, sin trucos. El objetivo de este ejercicio es desmitificar las referencias, que pueden parecer completamente nuevas. Aunque hay algunas pequeñas diferencias, esto es simplemente otra sintaxis para algo que ya haces: manipulación de direcciones.

**Capítulo VI**

Ejercicio 03: Violencia innecesaria

Ejercicio: 03

Violencia innecesaria

Directorio de entrega: ex03/

Archivos a entregar: Makefile, main.cpp, Weapon.{h, hpp}, Weapon.cpp, HumanA.{h,

hpp}, HumanA.cpp, HumanB.{h, hpp}, HumanB.cpp

Funciones prohibidas: Ninguna

Implemente una clase Weapon que tenga:

• Un tipo de atributo privado, que es una cadena.

• Una función miembro getType() que devuelve una referencia constante al tipo.

• Una función miembro setType() que establece el tipo utilizando el nuevo valor pasado como parámetro.

Ahora, cree dos clases: HumanA y HumanB. Ambas tienen un Weapon y un nombre. También tienen una función miembro `attack()` que muestra (sin los corchetes angulares):

<nombre> ataca con su <tipo de arma>.

HumanA y HumanB son casi idénticos, excepto por estos dos pequeños detalles:

• Mientras que HumanA usa `Arma` en su constructor, HumanB no.

• HumanB puede no siempre tener un arma, mientras que HumanA siempre estará armado.

Si su implementación es correcta, al ejecutar el siguiente código se imprimirá un ataque con "garrote con púas tosco" seguido de un segundo ataque con "otro tipo de garrote" para ambos casos 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();

}

return 0;

}

No olvide comprobar si hay fugas de memoria. ¿En qué caso crees que sería mejor usar un puntero a Arma? ¿Y una referencia a Arma? ¿Por qué? Piénsalo antes de empezar este ejercicio.

**Capítulo VII**

Ejercicio 04: Sed es para perdedores

Ejercicio: 04

Sed es para perdedores

Directorio de entrega: ex04/

Archivos a entregar: Makefile, main.cpp, \*.cpp, \*.{h, hpp}

Funciones prohibidas: std::string::replace

Cree un programa que acepte tres parámetros en el siguiente orden: un nombre de archivo y dos cadenas, s1 y s2.

Debe abrir el archivo <filename> y copiar su contenido en un nuevo archivo <filename>.replace, reemplazando cada ocurrencia de s1 por s2.

El uso de funciones de manipulación de archivos de C está prohibido y se considerará trampa. Todas las funciones miembro de la clase std::string están permitidas, excepto replace. ¡Úselas con prudencia!

Por supuesto, maneje las entradas y errores inesperados. Debe crear y entregar sus propias pruebas para asegurarse de que su programa funcione como se espera.

**Capítulo VIII**

Ejercicio 05: Harl 2.0

Ejercicio: 05

Harl 2.0

Directorio de entrega: ex05/

Archivos a entregar: Makefile, main.cpp, Harl.{h, hpp}, Harl.cpp

Funciones prohibidas: Ninguna

¿Conoces a Harl? Todos lo conocemos, ¿verdad? Si no, a continuación encontrarás los tipos de comentarios que hace Harl. Están clasificados por niveles:

• Nivel "DEBUG": Los mensajes de depuración contienen información contextual. Se utilizan principalmente para el diagnóstico de problemas.

Ejemplo: "¡Me encanta tener beicon extra para mi hamburguesa 7XL con doble queso, triple pepinillo y kétchup especial! ¡De verdad!"

• Nivel "INFO": Estos mensajes contienen información extensa. Son útiles para rastrear la ejecución del programa en un entorno de producción. Ejemplo: "No puedo creer que añadir tocino extra cueste más. ¡No le pusiste suficiente tocino a mi hamburguesa! ¡Si lo hubieras hecho, no pediría más!"

• Nivel "ADVERTENCIA": Los mensajes de advertencia indican un posible problema en el sistema.

Sin embargo, se puede gestionar o ignorar.

Ejemplo: "Creo que merezco un poco de tocino extra gratis. Llevo años viniendo, mientras que tú empezaste a trabajar aquí el mes pasado".

• Nivel "ERROR": Estos mensajes indican que se ha producido un error irrecuperable.

Suele ser un problema crítico que requiere intervención manual.

Ejemplo: "¡Esto es inaceptable! Quiero hablar con el gerente ahora".

Vas a automatizar Harl. No será difícil, ya que siempre dice lo mismo. Debes crear una clase Harl con las siguientes funciones miembro privadas:

• void debug( void );

• void info( void );

• void Warning( void );

• void Error( void );

Harl también tiene una función miembro pública que llama a las cuatro funciones miembro anteriores, según el nivel pasado como parámetro:

void complaint( std::string level );

El objetivo de este ejercicio es usar punteros a funciones miembro. Esto no es una sugerencia. Harl debe quejarse sin usar un montón de if/else if/else. ¡No lo piensa dos veces!

Crea y entrega pruebas para demostrar que Harl se queja mucho. Puedes usar los ejemplos de comentarios mencionados anteriormente en el asunto o usar tus propios comentarios.

**Capítulo IX**

Ejercicio 06: Filtro Harl

Ejercicio: 06

Filtro Harl

Directorio de entrega: ex06/

Archivos a entregar: Makefile, main.cpp, Harl.{h, hpp}, Harl.cpp

Funciones prohibidas: Ninguna

A veces no quieres prestar atención a todo lo que dice Harl. Implementa un sistema para filtrar lo que dice Harl según los niveles de registro que quieras escuchar.

Crea un programa que tome como parámetro uno de los cuatro niveles. Mostrará todos los mensajes de este nivel y superiores. Por ejemplo:

$> ./harlFilter "WARNING"

[ ADVERTENCIA ]

Creo que merezco un poco de tocino extra gratis.

Llevo años viniendo, mientras que tú empezaste a trabajar aquí el mes pasado.

[ ERROR ]

¡Esto es inaceptable! Quiero hablar con el gerente ahora. $> ./harlFilter "No sé qué tan cansado estoy hoy..."

[Probablemente me estoy quejando de problemas insignificantes]

Aunque hay varias maneras de lidiar con Harl, una de las más efectivas es desactivarlo.

Asigna el nombre harlFilter a tu ejecutable.

Debes usar, y quizás descubrir, la sentencia switch en este ejercicio.

Puedes aprobar este módulo sin realizar el ejercicio 06.