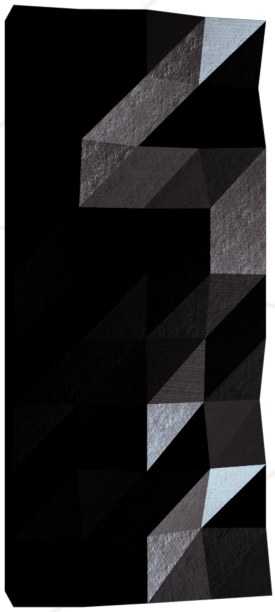

C++ - Módulo 06

C++ Casts

Resumen: Ejercicios para el Módulo 06 de C++.

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

Más reglas

- Para cada ejercicio, todas las situaciones de casting se resuelve mediante un cast específico. Durante la evaluación se validará que tu elección se corresponda con lo que esperamos.

Capítulo III

Ejercicio 00: Conversión escalar

	Ejercicio: 00
Ejercicio 00: Conversión escalar	
Directorio de entrega: *ex*00/	
Archivos a entregar: `Los que necesites y un Makefile`	
Funciones autorizadas: `Funciones que permitan convertir de string a int, float o double. Ayudará, pero no hará todo el trabajo.`	

Escribe un programa que acepte la representación de una string de `C++`. Será un valor literal de `C++` (en su forma más común) y se pasará como parámetro. Este literal será uno de los siguientes tipos escalares: `char`, `int`, `float` o `double`. Solo se utilizará la notación decimal.

Algunos ejemplos de literales de tipo `char`: `'c'`, `'a'`, etc. Para hacerlo más fácil, ten en cuenta: todos los caracteres no imprimibles no se pueden pasar como parámetro a tu programa, y si una conversión a `char` no es representable, muestra una notificación en su lugar.

Ejemplos de literales de tipo `int`: `0`, `-42`, `42`...

Ejemplos de `float`: `0.0f`, `-4.2f`, `4.2f`, etc. Deberás aceptar estos pseudoliterales también, ya sabes, porque es divertido: `-inff`, `+inff` y `nanf`.

Ejemplos de `double`: `0.0f`, `-4.2f`, `4.2f`, etc. Deberás aceptar estos pseudoliterales también, ya sabes, porque es divertido: `-inff`, `+inff` y `nanf`.

Tu programa debe detectar el tipo de literal, obtener este literal en el formato adecuado (por lo que dejará de ser una string), y luego convertirlo **explícitamente** a cada uno de los otros tres tipos para mostrar el resultado utilizando el formato dado debajo. Si una conversión no tiene sentido o hace overflow, muestra que la converrsión no es posible. Puedes incluir los headers que necesites para verificar límites numéricos y valores especiales.

Ejemplos:

```
./convert 0
char: Non displayable
int: 0
float: 0.0f
double: 0.0
./convert nan
char: impossible
int: impossible
float: nanf
double: nan
./convert 42.0f
char: '*'
int: 42
float: 42.0f
double: 42.0
```

Capítulo IV

Ejercicio 01: Serialización

	Ejercicio: 01
Ejercicio 01: Serialización	
Directorio de entrega: *ex*01/	
Archivos a entregar: `Los que necesites y un Makefile`	
Funciones autorizadas: `Ninguna`	

Escribe una función `üintptr_t serialize(Data *ptr);"`. Esta función devolverá el parámetro como un entero.

Escribe una función `"Data *deserialize(uintptr_t raw);"`. Esta función devolverá los datos iniciales que pasaste utilizando `"serialize"` a una estructura `Data`.

Escribe un programa basado en estas dos funciones que demuestre que todo funciona como se pide.

Debes crear una estructura de Data válida.
Utiliza serialize sobre una dirección de Data.
Lo que devuelva, pásaselo a deserialize.
Verifica que el valor devuelto es el mismo que el del primer puntero.
No olvides incluir la estructura `Data` que utilices.

Capítulo V

Ejercicio 02: Identifica el tipo real

	Ejercicio: 02
Ejercicio 02: Identifica el tipo real	
Directorio de entrega: $ex02/$	
Archivos a entregar: `Los que necesites y un Makefile`	
Funciones autorizadas: `Ninguna`	

Crea una clase `Base` que solo tenga un public virtual destructor. Crea tres clases vacías `A`, `B` y `C` que hereden públicamente de `Base`.

Escribe una función `"Base *generate(boid);"` que aleatoriamente instancie `A`, `B` o `C` y devuelva la instancia como un puntero de tipo `Base`. Siéntete libre de utilizar lo que quieras para la aleatoriedad.

Escribe una función `"void identify(Base *p);"` que muestre `."A"`, `"B"` o `"C"` de acuerdo al tipo real de `p`.

Escribe un programa en torno a estas funciones que demuestre el correcto funcionamiento de este ejercicio. `<typeinfo>` está prohibido.