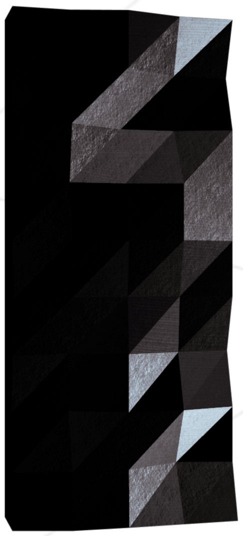

C++ - Module 06

C++ Casts

Summary: This document contains the subject for Module 06 of the C++ modules.

Contents

Chapter I

General rules

- Any function implemented in a header (except in the case of templates), and any unprotected header, means 0 to the exercise.
- Every output goes to the standard output, and will be ended by a newline, unless specified otherwise.
- The imposed filenames must be followed to the letter, as well as class names, function names and method names.
- Remember: You are coding in `C++` now, not in `C` anymore. Therefore:
    - The following functions are FORBIDDEN, and their use will be punished by a `0`, no questions asked: `*alloc`, `*printf` and `free`.
    - You are allowed to use basically everything in the standard library. HOWEVER, it would be smart to try and use the C++-ish versions of the functions you are used to in C, instead of just keeping to what you know, this is a new language after all. And NO, you are not allowed to use the STL until you actually are supposed to (that is, until module 08). That means no vectors/lists/maps/etc... or anything that requires an include <algorithm> until then.
- Actually, the use of any explicitly forbidden function or mechanic will be punished by a `0`, no questions asked.
- Also note that unless otherwise stated, the `C++` keywords `"using namespace"` and `"friend"` are forbidden. Their use will be punished by a `-42`, no questions asked.
- Files associated with a class will always be `ClassName.hpp` and `ClassName.cpp`, unless specified otherwise.
- Turn-in directories are `ex00/`, `ex01/`, . . . , `exn/`.
- You must read the examples thoroughly. They can contain requirements that are not obvious in the exercise's description. If something seems ambiguous, you don't understand `C++` enough.
- Since you are allowed to use the `C++` tools you learned about since the beginning, you are not allowed to use any external library. And before you ask, that also means

no C++11 and derivates, nor Boost or anything your awesomely skilled friend told you C++ can't exist without.

- You may be required to turn in an important number of classes. This can seem tedious, unless you're able to script your favorite text editor.
- Read each exercise FULLY before starting it! Really, do it.
- The compiler to use is `clang++`.
- Your code has to be compiled with the following flags : `-Wall -Wextra -Werror`.
- Each of your includes must be able to be included independently from others. Includes must contains every other includes they are depending on, obviously.
- In case you're wondering, no coding style is enforced during in C++. You can use any style you like, no restrictions. But remember that a code your peer-evaluator can't read is a code she or he can't grade.
- Important stuff now : You will NOT be graded by a program, unless explictly stated in the subject. Therefore, you are afforded a certain amount of freedom in how you choose to do the exercises. However, be mindful of the constraints of each exercise, and DO NOT be lazy, you would miss a LOT of what they have to offer !
- It's not a problem to have some extraneous files in what you turn in, you may choose to separate your code in more files than what's asked of you. Feel free, as long as the result is not graded by a program.
- Even if the subject of an exercise is short, it's worth spending some time on it to be absolutely sure you understand what's expected of you, and that you did it in the best possible way.
- By Odin, by Thor! Use your brain!!!

Chapter II

Bonus rules

- For each exercise, any cast situation is solved by a specific cast. The evaluation will check if your choice corresponds to the expected cast.

Chapter III

Exercise 00: Scalar conversion

	Exercise 00
Exercise 00: Scalar conversion	
Turn-in directory : *ex*00/	
Files to turn in : `Any file you need and a Makefile`	
Allowed functions : `Any function to convert from a string to an int, a float or a double. This will help, but won't do the whole job.`	

Write a program that takes a string representation of a `C++` literal value (in its most common form) as a parameter. This literal must belong to one of the following a scalar types: `char`, `int`, `float` or `double`. Only the decimal notation will be used.

Examples of `char` literal values: `'c'`, `'a'`... To simplify, please note that: non displayable characters can't be passed as a parameter to your program, and if a conversion to `char` is not displayable, output a notification instead.

Examples of `int` literal values: `0`, `-42`, `42`...

Examples of `float` literal values: `0.0f`, `-4.2f`, `4.2f`... You will also accept these pseudo literals as well, you know, for science: `-inff`, `+inff` and `nanf`.

Examples of `double` literal values: `0.0`, `-4.2`, `4.2`... You will also accept these pseudo literals as well, you know, for fun: `-inf`, `+inf` and `nan`.

Your program must detect the literal's type, acquire that literal in the right type (so it's not a string anymore), then convert it **explicitly** to each of the three other types and display the results using the same formatting as below. If a conversion does not make sense or overflows, display that the conversion is impossible. You can include any header you need to handle numeric limits and special values.

Examples:

```
./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
```

Chapter IV

Exercise 01: Serialization

	Exercise 01
Exercise 01: Serialization	
Turn-in directory : *ex*01/	
Files to turn in : `Any file you need and a Makefile`	
Allowed functions : `None`	

Write a function "`void * serialize(void);`". This function will return the address on the heap of a sequence of bytes that represent a piece of serialized data. The serialized data is the concatenation of a pointer to a random string, a random integer and a second pointer to a second random string. Feel free to use anything you like to generate the random values.

Write a function "`Data * deserialize(void * raw);`". This function will deserialize the raw data you created using "`serialize`" to a `Data` structure that contains the same elements that you just serialized and allocated on the heap.

Wrap these two functions in a program that proves that everything works as intended. Do not forget to include the `Data` structure you used.

Chapter V

Exercise 02: Identify real type

	Exercise 02
Exercise 02: Identify real type	
Turn-in directory : *ex*02/	
Files to turn in : `Any file you need and a Makefile`	
Allowed functions : `None`	

Create a class `Base` that only possesses a public virtual destructor. Create three empty classes `A`, `B` and `C` that publicly inherit from `Base`.

Write a function "`Base * generate(void);`" that randomly instanciates `A`, `B` or `C` and returns the instance as a `Base` pointer. Feel free to use anything you like for the randomness.

Write a function "`void identify_from_pointer(Base * p);`" that displays "`A`", "`B`" or "`C`" according to the real type of `p`.

Write a function "`void identify_from_reference( Base & p);`" that displays "`A`", "`B`" or "`C`" according to the real type of `p`.

Wrap these three functions in a program that prooves that everything works as intended. Including `<typeinfo>` is forbidden.