



# C++ - Module 06

## C++ casts

*Summary:*

*This document contains the exercises of Module 06 from C++ modules.*

*Version: 6.2*







- Note that unless explicitly stated otherwise, the `using namespace <ns_name>` and `friend` keywords are forbidden. Otherwise, your grade will be -42.
- **You are allowed to use the STL in the Module 08 and 09 only.** That means: no **Containers** (vector/list/map/and so forth) and no **Algorithms** (anything that requires to include the `<algorithm>` header) until then. Otherwise, your grade will be -42.

### A few design requirements

- Memory leakage occurs in C++ too. When you allocate memory (by using the `new` keyword), you must avoid **memory leaks**.
- From Module 02 to Module 09, your classes must be designed in the **Orthodox Canonical Form**, except when explicitly stated otherwise.
- Any function implementation put in a header file (except for function templates) means 0 to the exercise.
- You should be able to use each of your headers independently from others. Thus, they must include all the dependencies they need. However, you must avoid the problem of double inclusion by adding **include guards**. Otherwise, your grade will be 0.

### Read me

- You can add some additional files if you need to (i.e., to split your code). As these assignments are not verified by a program, feel free to do so as long as you turn in the mandatory files.
- Sometimes, the guidelines of an exercise look short but the examples can show requirements that are not explicitly written in the instructions.
- Read each module completely before starting! Really, do it.
- By Odin, by Thor! Use your brain!!!



Regarding the Makefile for C++ projects, the same rules as in C apply (see the Norm chapter about the Makefile).



You will have to implement a lot of classes. This can seem tedious, unless you're able to script your favorite text editor.



You are given a certain amount of freedom to complete the exercises. However, follow the mandatory rules and don't be lazy. You would miss a lot of useful information! Do not hesitate to read about theoretical concepts.









Write a program to test that your class works as expected.

You have to first detect the type of the literal passed as parameter, convert it from string to its actual type, then convert it **explicitly** to the three other data types. Lastly, display the results as shown below.


If a conversion does not make any sense or overflows, display a message to inform the user that the type conversion is impossible. Include any header you need in order to handle numeric limits and special values.

```
./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 VI

## Exercise 02: Identify real type

	Exercise : 02
Identify real type	
Turn-in directory : <i>ex02/</i>	
Files to turn in : Makefile, *.cpp, *.{h, hpp}	
Forbidden functions : <code>std::typeinfo</code>	

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



These four classes don't have to be designed in the Orthodox Canonical Form.

Implement the following functions:

```
Base * generate(void);
```

It randomly instantiates A, B or C and returns the instance as a Base pointer. Feel free to use anything you like for the random choice implementation.

```
void identify(Base* p);
```

It prints the actual type of the object pointed to by p: "A", "B" or "C".

```
void identify(Base& p);
```

It prints the actual type of the object pointed to by p: "A", "B" or "C". Using a pointer inside this function is forbidden.

Including the `typeinfo` header is forbidden.

Write a program to test that everything works as expected.

