Piscine C++ - d14a

The Martian

Jeremie Taboada koala@epitech.eu
Victor Hippolyte Boudon koala@epitech.eu

Abstract: This document is the subject of d14a

KOALA

Contents

Chapter I

Basic rules

- If you do half the exercises because you have comprehension problems, it's okay, it happens. But it you do half the exercises because you're lazy, and leave at 2PM, you WILL have problems. Do NOT tempt the devil.
- Every function implemented in a header, or 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, functions and method names.
- Remember : You're coding in C++ now, and not in C. Therefore, the following functions are FORBIDDEN, and their use will be punished by a -42, no questions asked :
  - *alloc
  - *printf
  - free
- Generally, files associated to a class will always be `CLASS_NAME.hpp` and `CLASS_NAME.cpp` (class name can be in lower case if applicable).
- Turn-in directories are ex00, ex01, ..., exN
- The turn-in dirs are `(cpp_d14a/exN)` , N being the exercise number
- Any use of `"friend"` will result in the grade -42, `no questions asked` .

- You must read the examples thoroughly. They can contain requirements that are not obvious in the exercise's description.
- As much as you are allowed to the C++ tools you are using since the beginning of the swimming pool, you are not authorized to use any external library.
- You are asked to turn in an important number of classes. However, these classes are pretty short. Slackers not accepted!
- Read each exercise FULLY before starting it!
- USE YOUR BRAIN, Please!

EXERCISES COMPILATION:

- The Koalinette compiles your code with the following flags : `-W -Wall -Werror -std=c++03` .
- Each of your includes must be able to be included independently from others. Includes must contains every other includes they are depending on.
- Each include file will be included in the main of the correction.
- Please note that none of your files must contain the `main` function except if it is explicitly asked. We will use our own `main` function to compile and test your code.
- The exercise description can be modified until 4h before the final turn in time!
- The compilator to use is g++!
- You must use file streams to solve some exercises. The video lecture on streams details how IOStream is working. There are numerous ways to implement it, within which file streams. This particular type of stream substitutes the file handling using the `C` mode based on `*open` and `close` .

Chapter II

Error handling

II.1 Exercise 0

	Exercise : 00	points : 4
Exercise 0		
Turn-in directory: `(cpp_d14a)/ex00`		
Compiler: `g++`	Compilation flags: `-W -Wall -Werror -std=c++03`	
Makefile: `No`	Rules: `n/a`	
Files to turn in : `Errors.hpp Errors.cpp`		
Remarks : `Errors.hpp is given, but you must complete it`		
Forbidden functions : `malloc - free`		

Your first mission will be to implement the error reporting system. Errors will have to respect the following inheritance hierarchy:

- std::exception
  - NasaError
    - LifeCriticalError
    - MissionCriticalError
    - CommunicationError
    - UserError

The method getComponent() of the exceptions should return the component name given in second parameter of their constructor. However, CommunicationError's getComponent method should always return "CommunicationDevice".

The getComponent() must have to following prototype:

```
1 std::string const &getComponent() const;
```

II.2 Exercise 1

	Exercise : 01	points : 4
Exercise 1		
Turn-in directory: `(cpp_d14a)/ex01`		
Compiler: `g++`	Compilation flags: `-W -Wall -Werror -std=c++03`	
Makefile: `No`	Rules: `n/a`	
Files to turn in : `Makefile Errors.cpp Errors.hpp(from ex_0), BaseComponent.hpp Engine.cpp, Engine.hpp Oxygenator.cpp, Oxygenator.hpp AtmosphereRegulator.cpp, AtmosphereRegulator.hpp WaterReclaimer.cpp, WaterReclaimer.hpp`		
Remarks : `Your objective is to be able to run 'make test' without error`		
Forbidden functions : `malloc - free`		

Now that you have created your classes, it's time to use it! The NASA has prepared some unit tests (RoverUnitTests.cpp) to ensure that all components are working properly, but mainly that all errors are verified.

To do this you are given the prototype files for each component of the Rover. As they are prototypes, the errors aren't implemented and it's up to you to ensure correct compilation and execution of 'make test'.

`All of the files are in the subject`

`You can modify all files, except RoverUnitTest.cpp`

Chapter III

Communication

III.1 Exercise 2

	Exercise : 02	points : 3
Exercise 02		
Turn-in directory: `(cpp_d14a)/ex02`		
Compiler: `g++`	Compilation flags: `-W -Wall -Werror -std=c++03`	
Makefile: `No`	Rules: `n/a`	
Files to turn in : `Errors.hpp (from part_1/ex_1), Errors.cpp CommunicationDevice.hpp, CommunicationDevice.cpp CommunicationAPI.hpp, CommunicationAPI.cpp`		
Remarks : `n/a`		
Forbidden functions : `malloc - free`		

Now, you will have to implement a CommunicationDevice. It will be used for communication between Houston and Mars.

You will have to use the CommunicationAPI and handle all its errors according the following:

- If sendMessage throws a standard exception, you should just print the error on the standard error output
- If receiveMessage throws a standard exception, you should also print the error on the standard error output, and the message should be "INVALID MESSAGE"
- If a standard exception is thrown in CommunicationDevice's constructor, you should catch it and throw a CommunicationError with the error preceded by "Error: " (example: "Error: userName should be at least 1 char.")
- Same thing for startMission, but with "LogicError: ", "RuntimeError: ", and "Error: " prefixes, repectively for std::logic_error, std::runtime_error and std::exception

Chapter IV

What about destructor?

IV.1 ScopedPtr

IV.1.1 Exercise 3

	Exercise : 03	points : 4
Exercise 3		
Turn-in directory: `(cpp_d14a)/ex03`		
Compiler: `g++`	Compilation flags: `-W -Wall -Werror -std=c++03`	
Makefile: `No`	Rules: `n/a`	
Files to turn in : `SimplePtr.hpp, SimplePtr.cpp, BaseComponent.cpp, BaseComponent.hpp`		
Remarks : `n/a`		
Forbidden functions : `malloc - free`		

The aim of this exercise is to design a generic class to ensure dynamically allocated rover's components deletion. For instance:

```
#include <stdexcept>

int main()
{
    try {
        // Use your auto delete here
        SimplePtr regulator(new AtmosphereRegulator);
        SimplePtr reclaimer(new WaterReclaimer);

        // The code above shouldn't be changed.
        throw std::runtime_error("An error occured here!");
    }
    catch (...) { }

    return 0;
```

```
16 }
```

SimplePtr.hpp is in the subject files, and does not need to be changed

IV.2 RefPtr

IV.2.1 Exercise 4

	Exercise : 04	points : 5
Exercise 4		
Turn-in directory: `(cpp_d14a)/ex04`		
Compiler: `g++`	Compilation flags: `-W -Wall -Werror -std=c++03`	
Makefile: `No`	Rules: `n/a`	
Files to turn in : `RefPtr.hpp, RefPtr.cpp, BaseComponent.hpp, BaseComponent.cpp`		
Remarks : `n/a`		
Forbidden functions : `malloc - free`		

Our ScopedPtr is nice, but we can't make a copy of it. Let's implement a RefPtr that can be stored, copied and still takes care of the delete!

`You are allowed to make any change necessary in the given class`

`Think about copy and assignment`

This code should construct only one Oxygenator and delete it.

```
#include <stdexcept>
#include <cassert>

#include "RefPtr.hpp"
#include "Oxygenator.hpp"

int
main()
{
    try {
        RefPtr oxygenator = new Oxygenator;
        BaseComponent *raw = oxygenator.get();
        RefPtr other(raw);
        RefPtr useless;
        RefPtr lastOne;
```

```
16          lastOne = other;
17          assert(lastOne.get() == raw);
18          (void)useless;
19          throw std::runtime_error("An error occured here!");
20      }
21      catch(...) { }
22
23      return 0;
24  }
```