CSCI251 Advanced Programming

Spring 2023

Lab 3 for Week 6 & 7

General Note

In these series of laboratory exercises you will be learning and practicing various programming concepts enabled by C++ programming language. Mostly, these concepts will be introduced through simple and sometimes complex examples. You are encouraged to think about how and why things work the way they do. The exercises are not set up to trip you, but you need to think carefully.

In order to focus on the task at hand, you can use an integrated development environment (IDE) to write, compile and test your code. In one of the notes provided on Moodle, you can find the steps required to install Code::Blocks on several OS (e.g. Windows, Linux, MacOS). You are encouraged to follow the instructions and install it for your operating system. The operating system on which you develop your code should not matter. All that needs to happen is that your code is C++17-compliant. The system on which your code will be tested has g++ 7.5 installed and your code must work on that system (i.e. **capa.its.uow.edu.au**).

In order to test that your code is compliant and will work on **capa.its.uow.edu.au**, you will need to do a secure login to capa and test your code. The appropriate option required to ensure that you are compiling against C++17 is the -std=c++17 added to your command line instruction. For example:

g++ file1.cpp file2.cpp file3.cpp -std=c++17 -o output_exec

Get familiar with your IDE and ensure that the build option is set to use c++17.

Use sensible names for your variables and insert comments to let the reader know what your code is doing. If your code contains no comments you will lose marks.

Code Submission

For each exercise, name your code as instructed. Each Lab will be due for submission at midnight of Weeks 3, 5, 7, 9 and 11. You will also be reminded accordingly as the session progresses. Submission points will be set up on Moodle. For each task you might save the code file(s) like:

t1.cpp: only one file.

t1-1.cpp: if multiple subtasks

t1.zip: if there are multiple files

At the end, please submit a (big) zip _le that includes all task for the lab, and name it like: “familyname_studentId_lab#.zip”. The symbol # represents the lab number. For example, in Lab 1 (week 2&3), the submission can be named as “Jordan_12345_lab1.zip”.

Task 1 Debugging (20 points)

1. (10 points) Debug: Debug-A.cpp. The output should be

```
ID # 111
Name: Alice Anteater
Salary: $23.45
```

2. (10 points) Debug: Debug-B.cpp. The output should be as follows:

```
Given name : Alice
Id. number : 12321
Hat type : Beret
Hat size : M
Given name : Bob
Id. number : 2324
Hat type : Trilby
Hat size : S
```

Task 2 Cat Class (20 points)

We provide you the file Cat.cpp, you will do following tasks:
1. Add a constructor.
2. Add a destructor.
3. Add a copy constructor.
4. Add code to show the addresses of the contents of a Cat object. This is just to see how the internals are located, relatively.
5. Add code to show the value of the content of a Cat object.
6. Extend the main function to demonstrate the use of the class.

Task 3 A Bonus class (20 points)

Create a piece of code Bonus.cpp. This should contain a class Staff with the following:

1. Data fields holding a staff number, last name, first name, base salary, sales made, staff class, and the bonus for which the person is eligible.
2. Constant static fields holding the bonus rates per sale, as a percentage of salary, in accordance with the following array:

Sales	Class A	Class B	Class C
$0-20$	0.03	0.02	0.005
$21-50$	0.05	0.035	0.015
$51+$	0.075	0.055	0.04

3. Include a static function that displays the bonus table.
4. Include a function *setFields()* to set all the field values on the basis of the staff number, last and first names, base salary, sales made and staff class.

5. Include a function *computeBonus()* to determine the bonus earnt.
6. Include a function *display()* to display all the information for a staff member.
7. In the *main()* function you should display the Bonus table, construct a couple of Staff objects, use *setFields()* for each, passing whatever fields you want to demonstrate the functionality and finally run the *display()* function for each. Note that after *setFields()* runs the bonus should have been determined.

In writing the code you should consider carefully whether data fields and functions should be public or private. You should call the fields by appropriate names.

There is an example of output in Sample.txt.

There is no need to make the output particular ash in terms of aligning columns exactly and so on.

Task 4 Setting up a bigger example (20 points)

Write appropriately placed code supporting the classes and functionality described below. They should be in files *Four.cpp*, *Library.cpp*, and *Library.h*.

1. Write a class, Money, to hold the name of the country a currency is associated with, number of dollar units, and the exchange rate (to global dollars).
2. Write a class, City, holding city information, specifically a city name, country, latitude and longitude.
3. Each class should have a constructor that takes arguments to set the field values.
4. Create a *friend* function that tests if a given City and a given Money are associated with the same country.
5. Write a short *main()* function to test the classes and the *friend* function applied to them. You shouldn't get input from the user.

Task 5 Composition (20 points)

Make a copy of your solutions from Task 4 (Setting up...), and call the files *Five.cpp*, *LibraryC.cpp* and *LibraryC.h*.
1. Put a Money object inside the City class.
2. The City constructor should call the Money constructor.
3. Write display functions in Money and City. The City one should use the Money one.
4. Write a *main()* to demonstrate the functionality you have added.