CSCI251 Advanced Programming

Spring 2024

Lab 4 for Week 8 & 9

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. (20 points) Buggy inheritance: Fix Debug-A.cpp. The run of this program, with input as shown (Blob etc.), should be:

```
Enter painting's title Blob
Enter artist Degas
Enter painting's title Blob
Enter artist Alice
Enter painting's title Blob
Enter artist Bob
Enter painting's title Blob
Enter artist Picasso

Blob by Degas value $25000
Blob by Alice value $400
Blob by Bob value $400
Blob by Picasso value $25000
```

2. (0 point) Why will the following fail to compile?

```
class B
{
public:
virtual void X() = 0;

};

class D : B
{
public:
virtual void X() {cout << "D object" << endl;}
};
int main()
{
B objB;
}
```

Task 2: Relationships (20 points)

1. (20 points) **A Box in a Box in a Box**

Write code in Box.cpp to represent storing a collection of Boxes, i.e. Box objects, within a Box object. You will need functionality to add a Box to a Box. You can play around with this

idea a fair bit, and potentially include limits on the number of Boxes or the size and capacity of Boxes to make sure that a Box cannot be added to itself.

2. (0 point) **Consider A-Class.cpp.**
(a) What concept does it illustrate?
(b) How are the classes related?
(c) What happens if either the worker or company is deleted in main, after the Contract is set up but before the output is displayed?
(d) What happens if we add an additional company and add a contract between John and the new company? How would we manage this situation, or the case where a new worker wants to work for Bell?

Task 3: Inheritance, abstraction and polymorphism (20 points)

1. (10 points) An Animal hierarchy:
(a) Modify the provided *Cat* class so it is derived from a base class *Animal*. Write the base class *Animal*. Make another subclass of *Animal* and make sure there is some functionality distinguishing them. Write a main function to illustrate creating instances of all three classes.
(b) Make the base class *Animal* abstract, and modify the derived classes if necessary to make sure they are not abstract classes. The *Animal* instance can be commented out in main for the version you submit.
(c) Write driver code containing a vector of pointers to *Animal* in which you store multiple different types within the *Animal* hierarchy, and demonstrate adding different types of *Animal* to this vector.

2. (10 points) A Transportation hierarchy.
For each of the classes below you should provide sufficient functionality to test your understanding of the relationships between the classes. You should, in particular, see how you can use constructors across classes. The code should be able to compile but you don't need to store data or have other within each class at this point.
(a) Define a base class *Transportation*.
(b) Define three derived classes, *SeaTransport*, *LandTransport*, and *AirTransport*.
(c) Inheritance can occur across multiple levels. Define Car and Canoe as classes derived from appropriate classes.
(d) It's possible to have multiple inheritance in C++. Define Hovercraft as derived from two appropriate classes.

Task 4: Libraries (20 points)

1. (0 point) The files *driver.cpp*, *mylibrary.cpp*, and *mylibrary.h* form a program.
(a) Produce an executable from those files.

```
g++ driver.cpp mylibrary.cpp -o M1
```

(b) Convert the non-driver cpp file into a static library libcode.a.

```
g++ -c mylibrary.cpp
```

```
ar -crv libcode.a mylibrary.o
```

(c) Generate an executable.

```
g++ driver.cpp libcode.a -o M2
```

(d) Convert the non-driver cpp file into a shared (dynamically linked) library.

```
LD_LIBRARY_PATH=.
export LD_LIBRARY_PATH
g++ -fpic -shared -o libcode.so mylibrary.cpp
```

(e) Generate an executable.

```
g++ -I. -L. driver.cpp -lcode -o M3
```

2. (10 points) How do the 3 executables differ in size? (put your answer in a text file)

3. (10 points) Which of the 3 executables run without the corresponding library being present? (put your answer in a text file)

Task 5: The Three Little Software Engineers (20 points)

Carry out these tasks for *The Three Little Pigs*, a story in *3Pigs.txt*. Ideally you should be discussing this with other people.

1. Write a timeline identifying the events in the story.
2. Identify appropriate objects/classes and attributes/behaviours for those classes.
3. Sketch a class diagram showing the classes and appropriate relationships. This may be a somewhat iterative process with multiple versions.
4. Write code to implement the identified structures.
5. Write *main()* function(s) which when run will approximate the events of the story, or similar stories. It can be interactive if you like, to cover variations on the stories.