CS264 Laboratory Session 5

Dr. Edgar Galván

Tuesday 14^{th} November 2017

Deadline: All solutions to be submitted by 5pm Thursday 23^{rd} 2017

1 Lab objectives

In this lab you will design and implement some classes in C++ that use some of the inheritance techniques discussed in lectures. However before starting on the programming you will be given a walk through of the Eclipse CDT integrated development environment. This will involve showing you how to (i) create a new c++ project, (ii) how to add a new class to that project, (iii) use the debugger within the IDE, and (iv) add your new project to git.

Source: *C++ in the Lab: Lab Manuals to Accompany C++ How to Program 4ed. Chapter 9. Deitel, Deitel & Nieto. Prentice Hall, NJ.*

Learning Outcomes
Having completed this lab you should know how to create and manage C++ project with Eclipse CDT and git and use the debugger within Eclipse CDT to debug a program in execution.
You will also be able to create inheritance hierachies in C++. You should understand the details of object construction and destruction within such hierarchies.

2 Questions:

Step 0.1: For this week's exercises you should create a sub-directory off of the top level `Labs` directory called `Lab5`. For this lab you will be provided with skelton code which can be downloaded from CS264 moodle webpage. You should start by downloading and unpacking either the zip file into your Lab5 subdirectory. This should create two subfolder, one for each question. Be sure to commit at least after each exercise with appropriate commit messages. Be sure to commit at least after each exercise with appropriate commit messages. For each of the problems given below write a C++ program that provides a solution that problem. Each box provides a filename to use. Please ensure that you use those filenames. Failure to do so can result in loss of marks. **You should include a comment at the top of each source file with your full name followed by your surname, the latter in capital letters (e.g., Edgar GALVAN) and student number (if you have one).**

Remember:

- **Add source files to your git repository and commit changes regularly.**
- **All commits should be accompanied by messages that would allow a lecturer or demonstrator to understand the purpose of the commit.**

- **Comment your code.**
- **Use proper indentation for function and control structures.**

Exercise 1: Develop a `Racecar` class that inherits publicly for a class Car, which represents a car by its maximum speed, the number of engine valves, its colour and its name. A `Racecar` is distinguished by its gearbox (the number of gears it has), its sponsor and the presence of a parachute.

To complete this problem you should use the template code in the Exercise1 directory as a starting point. Comments in each of the files will guide you through the functionality and definitions that need to be added. The `driver.cpp` file contains a main method that you can use to test your solution. The result of running this code should look something like the following:

```
chevy:
Car:  Chevrolette is black and has a 4-valve engine.  MAX SPEED = 95 mph.

f1:
Car:  Ferrari is red and has a 40-valve engine.  MAX SPEED = 220 mph.
Ferrari also has 7 gears and is sponsored by Bug2Bug.
Ferrari has used its parachute.
```

Exercise 2: Develop a class hierarchy of `Vehicle`s. Create two classes, `Taxi` and `Truck`, that inherit publicly from class `Vehicle`. A `Taxi` should contain a data member that indicates whether it is carrying passengers. A `Truck` should contain a data member that indicates whether it is carrying cargo. Add the necessary member functions to manipulate and access class data. Write a driver that prints a `Truck` object and a `Taxi` object to the screen (using an overloaded stream-insertion operator).

To complete this problem you should use the template code in the Exercise2 directory as a starting point. Comments in each of the files will guide you through the functionality and definitions that need to be added. The `driver.cpp` file contains a main method that provides a partial implementation of the driver code mentioned above. The result of running this code should look something like the following:

```
Vehicle
         Number of doors:  2
         Number of cylinders:  6
         Transmission type:  3
         Colour:  blue
         Fuel level:  14.6

Taxi
         Number of doors:  4
         Number of cylinders:  6
         Transmission type:  5
         Colour:  yellow
         Fuel level:  3.3
         The taxi currently has no customers

Truck
         Number of doors:  2
         Number of cylinders:  16
         Transmission type:  8
         Colour:  black
         Fuel level:  7.54
         The truck is currently carrying cargo.
```