CS264 Laboratory Session 6

Dr. John McDonald

17^{th} November 2015

Deadline: Solutions to be submitted by 6pm 1^{st} December 2015.

1 Lab objectives

In this lab you will start to work will continue with designing and implementing some classes in C++ that use inheritance. Further to this you will also use polymorphism invoke behaviours over a group of different concrete classes derived from the same abstract base class.

Finally you will implement some generic functions and develop some programs using containers from the STL.
Reference: 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 will be able to create inheritance hierarchies in C++ and use polymorphism to avoid complicate selection statements. You will also be able to write simple generic functions and classes. You will also be able to use a subset of containers provided by the STL and use iterators to access elements in a container independent fashion.

2 Questions:

For each of the problems given below write a C++ program that provides a solution. Each box provides a filename to use (or in certain cases multiple filenames). Please ensure that you use those filenames. Failure to do so can result in loss of marks.

Step 0.1: For this week's exercises you should create eclipse projects for each exercise. Each project should be checked into a subdirectory of a top level directory called `Lab6`. Be sure to commit at least after each exercise with appropriate commit messages.

Remember:

- **Add source files to subversion and to 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 1.1: On the course website you will find a link to code to be downloaded for this lab. You should download this code and unpack it to its own directory. Next create an Eclipse project in that directory. This project should automatically import the .cpp and .h files.
The code provides a partial implementation of a `Vehicle` hierarchy. Your task is to apply polymorphism to this hierarchy. From the existing code develop an *abstract* base class that includes the vehicle's name, colour, number of doors, number of cylinders, transmission type and fuel level. Add a member function named `horn` that displays the sound made by the `Vehicle`'s horn. The `print` member functions and the `horn` member function should both be virtual functions; `horn` should be a pure virtual function. Class `Taxi` and class `Truck` should both be derived from `Vehicle`.
Complete the driver to test the class hierarchy. This driver instantiates one object of type `Taxi` and one object of type `Truck`. Insert those objects into a "container" – a vector of base-class pointers. For each object in the vector, call virtual functions `horn` and `print`.

Exercise 1.2: Rewrite the driver program such that all the `Taxi` and `Truck` objects are create dynamically (i.e. using `new`). Remember to delete these objects before the program terminates.

Exercise 1.3: Modify the program to include a new `ParkingLot` class. A header file for this class is included in the download from the previous exercise (i.e. your task is to provide the implementation file, `ParkingLot.cpp`). This class should use a vector of `Vehicless` to store each vehicle parked. Modify the driver program to place 10 `Vehicles` in the parking lot. Modify the `for` loop to honk the horn and to display information about each `Vehicle`.

Exercise 1.4: Review the notes from the course on creating template functions. Based on this write a template function, `print`, that takes as input an array and a variable that gives the number of elements in the array. The function should then step through the array printing out each element and separating the elements by a single space. The function should use templates in that the type of the array should be a template. You should provide a driver function that demonstrates the function on an array of doubles and an array of strings.

Exercise 1.5: Add a second `print` function that takes as input a single STL container. The function should perform the same function, printing out each element of the container. To do this it should use a `const_iterator` (which is essentially just an `iterator` where you can't change the object to which it points).
Note: This exercise requires you to deal with a idiosyncrasy of C++ related to how it figures out what a templated name actually refers to. (For those interested in the details you should have a look at the explanation at this link.) In short the issue will cause the following function to cause an error :

```
template <typename T>
void foo(vector<T> v){
  vector<T>::iterator i; //ERROR: compiler can't determine this is a typename
}
```

To fix this issue you have to explicitly tell the compiler that the iterator is a type as follows :

```
template <typename T>
void foo(vector<T> v){
  typename vector<T>::iterator i; //All is good again!
}
```