Imperial College London – Department of Computing Second Year Computing Laboratory

Credit Suisse C/C++ Workshop – Part 2

October 2015

Note — using 'make'

The following exercises will obviously involve editing and compiling C++ source code. To be able to do this more easily there is a utility called 'make' which allows multiple source files to be compiled and then linked together to form the .exe. Details of how to use it can be found here:
http://mrbook.org/tutorials/make/
Alternatively ask one of the lab assistants or look at your make slides in C from last year.

Exercise 1

This exercise is nice and simple and should ease you back into the start of term! It should give you practice in writing classes, inheritance and polymorphism.

- Create an abstract base class `Creature` with one function declared in it called `MakeNoise()`, which returns `void`. Note the word abstract !
- Create three classes Monkey, Duck and Pig derived from the base class. Each of those classes should implement MakeNoise() appropriately and the effect of calling the function should be to print out to the console the appropriate animal noise.
- Your `main()` should have only one `Creature* pCreature` variable declared (i.e. a pointer to a `Creature`) which should be used to call `MakeNoise()` on each animal.
- Note, when writing the classes, create a separate .h file for the class definition and one .cpp file for the class implementation. Your .h file for each class should start something like this (substitute the appropriate `FILENAME`):
```
#ifndef FILENAME_H
#define FILENAME_H
// class declaration goes in here
......
// and ends with:
#endif
```

- This ensures that you do not get 'multiple definition' errors for the classes should the .h files get included more than once in a .cpp file. These are commonly referred to as 'include guards'. Different teams will have different conventions for how they are used – and even spelled – but they're extremely common practice.

Exercise 2

This exercise should give you practice in class design, inheritance, encapsulation and polymorphism.

- The University Bookshop sells Books, CDs and DVDs. And in this example, only 2 items of each! If you open up `Lab2_ex2.cpp` you will see what the bookshop is selling. DVDs and CDs have associated with them a running time (in mins). Additionally DVDs have a format (either PAL or NTSC — an enum would be good for this) and CDs have associated with them the number of tracks on the CD. Books have associated with them a number of pages. All the items on sale have a unit price in pence and a title.

- Unfortunately, the cash-strapped university, in an underhand attempt to raise more money, have decided to tax sales on these items. A tax will be placed on each DVD sold equivalent to 2p for every minute of running time. For CDs, the tax is 5p for every track. For books, it is 4p for every whole set of 10 pages.

- Your task now is to print out the title of each item along with its new price in pence (i.e. the unit price + tax).

- Think carefully about your class design. What commonality is there among the classes you are creating?

- As a hint, you will need as a bare minimum two classes, `Shop` and `SaleableItem`. The `Shop` object should hold a collection of `SaleableItem` pointers. It is recommended for this exercise that you implement this collection as a `std::vector`. This is a very useful class available in the Standard Template Library and is available if you `#include <vector>`. So, declare a member variable `saleableItems_` of type `std::vector<std::shared_ptr<SaleableItem»` in Shop (we're using shared_ptr to make memory management easier) When you want to add to the collection, just call `saleableItems_.push_back(pSaleableItem);` (where `pSaleableItem` is a `shared_ptr<SaleableItem>`). When you want to iterate through the collection you can do this: `for( auto& item : saleableItems_ )`

  ```
  item->doSomething();
  ```

- To print out the details you can use `std::cout`, which means you will need `#include <iostream>` in the appropriate place(s).

- Ensure you have one .h file per class defintion and one .cpp file per class implementation. As previously, remember to guard your header files with `#ifdef`.
- And don't forget to clean up any memory . . . :-).