CM2204: Advanced Programming Laboratory Worksheet (Week 8)

Prof. D. Marshall

Aims and Objectives

After working through this worksheet you should be familiar with:

- C++ constructors & destructors
- Inheritance in C++ to define new classes
- Be able to override functions in subclasses
- Understand the purpose of abstract classes and pure virtual functions

None of the work here is part of the assessed coursework for this module.

- Follow the web links for files highlighted and underscored to get code listings
- All lecture and lab class code is a available on the CM2204 Web page
- Solutions to the exercises will be released on the CM2204 Web page in **Week 9**.

C++ Initialization, cleanup & inheritance

1. (Question 1 of Chapter 6 of Thinking in C++, *Vol. 1*) Write a simple class called `Simplewith` a constructor that prints something to tell you that its been called. In `main()` make an object of your class.

2. (Question 2 of Chapter 6 of Thinking in C++, *Vol. 1*) Add a destructor to Question 1 above that prints out a message to tell you that its been called.

3. (Question 3 of Chapter 6 of Thinking in C++, *Vol. 1*) Modify Question 2 above so that the class contains an `int` member. Modify the constructor so that it takes an `int` argument that it stores in the class member. Both the constructor and destructor should print out the `int` value as part of their message, so you can see the objects as they are created and destroyed.

4. Add a *circle* class to the `Shapes.cpp` example.

5. Write a class `B` that contains a single int data member named `x`, with a function `f()` that prints out a simple message and the value of `x`. Provide a constructor that initialises `x`.

   - Write a subclass of `B` named `D`. Provide a *constructor* for `D` that calls the constructor for `B` to *initialise* `x`.
   - Provide a function `f()` in `D` that does not override `f()` in `B`, and prints out the *square* of `x`.
   - There are (at least) two ways that you can structure the code so that this function can access the value of `x` — implement both ways.
   - Is it possible to call the function `f()` defined in `B` on an object of type `D`? How?
   - Modify your code so that `f()` is overridden in `D`.
   - Make `f()` a pure `virtual` function and verify that it is no longer possible to create objects of type `B`.

6. (Question 1 of Chapter 6 of Thinking in C++, *Vol. 2*, Multiple Inheritance) The traps of Multiple Inheritance:

   - Create a base class `X` with a *single constructor* that takes an `int` argument and a member function `f()`, that takes no arguments and returns void.
   - Now inherit `X` into `Y` and `Z`, creating constructors for each of them that takes a single `int` argument.
   - Now multiply inherit `Y` and `Z` into `A`.
   - Create an object of class `A`, and call `f()` for that object.
   - Fix the problem with *explicit disambiguation*.

Further Practice

1. (Question 2 of Chapter 6 of Thinking in C++, *Vol. 2*, Multiple Inheritance) Starting with the results of Question 6 above,

   - create a *pointer* to an `X` called `px`, and assign to it the *address* of the object of type `A` you created before.
   - Fix the problem using a `virtual` base class.
   - Now fix `X` so you no longer have to call the constructor for `X` *inside* `A`.

2. (Question 3 of Chapter 6 of Thinking in C++, *Vol. 2*, Multiple Inheritance) Starting with the results of the above Further Practice Exercise 1

   - remove the explicit disambiguation for `f()`, and see if you can call `f()` through `px`.
   - Trace it to see which function gets called.
   - Fix the problem so the correct function will be called in a class hierarchy.