Imperial College London – Department of Computing
Second Year Computing Laboratory
Credit Suisse C/C++ Workshop – Part 1
October 2013

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!

Exercise 1

This exercise should give you some practice with using data types, constants, general program flow and passing parameters by reference.

- Write a C++ program that asks the user for two values representing the width and length of a rectangle. The width should be a whole number between 1 and 10000 inclusive. The length should be a whole number between 1 and 20000 inclusive. Within your program write a function (returning void) `CalculateRectangleMetrics` that accepts two by value parameters for the width and length of the rectangle and two by reference parameters, one using the pointer notation, the other the reference notation. The by-reference parameter which is a pointer should yield the perimeter of the rectangle and the by-reference parameter which is a C++ reference should yield the area of the rectangle. Once you have the perimeter and area values, print them out to the screen to display to the user.
- Think carefully about the datatypes you could use for this and where constants come into play.
- For the more adventurous, try to make your program a little more polished by validating that the user-supplied width and length are valid values.

Exercise 2

This exercise will give you some practice in iterating over an array using pointers.

- If you open the .cpp supplied file for this exercise you will see an array of 10 ints set up for you.
- Now write a function GetIntegerArrayMetrics that returns a bool. The function should have two 'input' parameters:
    1. A pointer to the start of the array
    2. The length of the array

- It should also take 4 by-reference parameters used for output:
    1. The minimum value in the array
    2. The index in the array of that minimum value
    3. The maximum value in the array, and
    4. The index in the array of that maximum value.

- It is good practice to check that the supplied pointer parameter is not 'NULL'. So check that first before doing anything — if it is 'NULL' then make the function return 'false'. If all goes well, return 'true'. When calling the function, check for that return value, and if 'true' print out the answers retrieved, else print out an error message telling the user that something went wrong.

- Within the implementation for GetIntegerArrayMetrics, use a pointer to iterate over the array of integers looking for the min and max values. Be careful not to iterate over the end of the array (you can use the supplied array size parameter to help you with this).

Exercise 3

This exercise will again give you some practice in iterating over an array using pointers and amending values in that array.

- If you open the .cpp file supplied for this exercise you will see a `char` array set up that contains a string expressing your indifference to C++. We simply can't have that so your job is to write a function ChangeHateToLove that returns '`void`'. It should take one `char*` parameter. The function should use a `char*` variable to iterate through the array of chars and detect and change instances of the word 'hate' to 'love'. Whilst iterating you can ensure you don't go beyond the bounds of the array by testing that the character at the current position doesn't equal '\0' (this is because a string is a special array of chars that has this string sentinel character as its last element).
- Note that you will have to ensure that any 'h' you find is followed by 'a', 't' and 'e' before you can change it — so be careful!
- Be aware too that the word 'hate' also exists in the word 'whatever' — we don't want the 'hate' in there being changed. So, try to come up with a test that will prevent that from happening. As a suggestion, just check that the word 'hate' exists with no other letters immediately after it. (You could also check that no letters immediately precede it, but for this exercise the 'after' check will suffice).