Week 7 Exercises : Simple Functions

Mar 5th 2012

1 Approach

Always read *all* of this document before you start the first exercise. Do this before the Tuesday lab session so that you know what your workload for the week will be. If you are aiming for a high grade then do both the basic exercises and those marked with a star. If you are having difficulty with the pace on the course then skip the starred exercises. The exercises are now roughly divided into three; these match the Tuesday / Wednesday / Thursday lab sessions. Make sure you approach the work in these batches: on a given day ask the questions that you need to understand how to do those exercises — finish them between classes.

2 Exercises

TUESDAY starts with some simple functions and looks at the difference between local and global variables.

1. Write a function that takes a single integer argument and returns a single integer result. The function should calculate the square of the number. Write a program that inputs integers from the console and outputs their squares until 0 is input.

2. Write a function that takes a single integer argument and returns a single integer result. The function should return the biggest value it has been passed since the beginning of the program. Initially the biggest value can be assumed to be zero. Write a program that inputs integers from the console and outputs the biggest number so far.

3. * Write a function that takes a single integer argument. The integer result on each call will be the argument passed in two calls before the current call. The function can return zeros for the first and second call, before it starts to return the values it has remembered. Write a program to test the function.
   Example of execution: (function name is *f*)

```
cout << "output: " << f(4) << " " << f(1) << " " << f(6) << " " << f(8);
output: 0 0 4 1
```

4. * Write a function that takes two integer values as arguments, one signed and one unsigned. The signed integer will be used to fill a memory as described in the previous exercise. The unsigned value will be the latency, i.e. the number of previous calls ago that the number should be returned from.

WEDNESDAY continues with the problem of identifying pieces of code that make good functions.

5. Functions with no arguments and a void return can still perform actions, e.g. input and output. Pick one shape drawing exercise from two week's ago and identify parts of it that can be moved into a function.

6. Functions with arguments, but still with a void return can perform parameterised actions - sequences of input and output that change depending on the argument. Pick an exercise from two weeks ago with a suitable piece of code and convert it to a function.

7. * Combine the drawing primitives from two week's ago into a single program. Put each primitive in a separate function using an argument instead of an input to control the size of the shape. Write a simple text menu in *main* that inputs the shape name and parameters before calling the correct function.

8. * Write a function that creates a 2D array of chars. You will need to find out how to create 2D arrays, using loops and the dynamic memory allocation that you saw last week. To test it write a function that fills it with a pattern of characters, and another function that outputs the contents to the screen.

THURSDAY continues the theme of identifying good candidate functions by looking at improvements to exercises from previous weeks.

9. Take your sorting program from last week's Exercise 4. Write a sorting program that operates on arbitrary sized data. It will input a size from the console, and allocate an array of that many integers. The array will be filled with random values and then sorted. You need to decide which parts of this problem will make appropriate functions.

10. * The type of sort that you have implemented so far is called an insertion sort. Lookup the definition of a bubble-sort and write a program with two functions, one to perform insertion sorts and one for a bubble sort. You are going to investigate which of these is quicker using the `time` function.