SYSC 2006 Foundations of Imperative Programming

Lab 6

Learning Strategies

The demand for TAs is always highest in the last portion of the lab; if you need specific help, take the initiative and ask for help at the start of the lab.

***Always track your time*. If you cannot finish this lab in the allotted time, you need to study more before the next lab or assignment.**

Objective

- Pass-by-Reference with Structures
- Modular Programs
- Testing Strategies

Agenda

1. **Lab Introduction by the TA**: Unless your past performance is A's, you are expected to gather round the TA for a brief introduction of the lab. The TA will identify the key points to learn, probable error spots, and suggested work strategies. TAs are not necessarily experienced lecturers so please help the process by gathering quickly and being quiet. Be at your best behaviour for these 15 minutes.
2. **Activity A**
3. **Activity B**
4. **[Optional] Demo**: Ask a TA to give you feedback on your lab. There is no mark for this demo. The purpose is for your learning benefit. Anyone with a grade below B should always ask for feedback.
5. **Submission:** Submit your **source** files as well as your stack frame diagram JPG on the course webpage.

Activity A (1 ½ hour)

You are provided with a header file that defines some key data types for managing information about a class of students. You are required to write the implementation of this student library as well as write a test driver demonstrating the correct behavior of all library functions.

Aside: The above paragraph – along with the header file – should be enough words for you to do the job. Give it a try. If you need more help, follow the steps listed below. The steps illustrate incremental code development. When writing test code, remember to test your code on more than one piece of data.

1. Write the first version of your test driver. Create a separate file called **lab6.c**, include the header file and write a mostly-empty main() function. Compile this program. We now have a starting base.
2. Let's now write-and-test the simplest structure: NAME.

a. In the main() function, declare a variable of NAME called name. Hard-code the initialization of this variable name. Use any reasonable values for the initial contents. Compile (and run) this program. It will not do anything much but you could step through it with the debugger to confirm the initial values.
b. Create another file called student.c. Write the implementation of the easiest function, printName(…). In the test driver, put a corresponding call of this function. Now when you compile and run this program, it will actually do something (it will print the values of your name variable).
c. Copy this procedure for the ADDRESS and STUDENT structure types as well as their respective printxxx() functions.
d. Now is the difficult part: Instead of printing the structures, you want to input their values from the console. As before, work one-by-one through scanName(), scanfAddress() and then scanStudent().

Activity B (1/2 hour)

In the upcoming exams, you will have to write stack frame diagrams for program fragments. Practice now by drawing the complete stack frame diagram for your code in Activity B. To submit, take a picture with your cell phone, and save it as a JPG.