CSC 150

Prelab #5: Debugging

Debugging is an important task during software construction. After programs are compiled, they may not work as expected the first time you run them. Debugging is the process of finding runtime or logical errors in your program, so that it does exactly what you expect from the program.

Runtime errors are errors that occur during the execution of your program. For instance, division by zero gives you a runtime error.

Logical errors are mistakes that produce erroneous results. For instance, if you are supposed to add two values or expressions in some statement of a program, but instead you multiply them, the program will produce unexpected results.

To find a runtime error or logical error, programmers usually insert additional **`cout`** statements in the code to find it. These statements will display the state of variables at various points of execution, and even identify where in the program it is currently executing. You must remove them later after you have fixed the program. Visual Studio has a debugger for finding runtime or logical errors without the need to insert **`cout`** statements. You will learn how to use the debugger in the Debugging Lab. The debugger is used after a program build is created. That is, the program should not have syntax errors and should be able to compile and link. When you use the debugger, you usually set one or more breakpoints (points where you want the program to stop) and let the debugger step through the code following these breakpoints.

In Visual Studio, a program in running mode is switched to break mode when the system finds a runtime error. When that happens, a C++ window pops up. It tells you more or less the type of error. A program with a runtime error throws or generates an exception. You will need to fix the program and run it again. You must stop debugging (click **Stop Debugging** on the Debug menu) to exit the break mode of your program. A program is in break mode when a program is suspended from execution at some breakpoint.

Testing is another task in software development that works closely with debugging. It is important to test your program with several sample runs or test cases. If you don't test your program, then you don't need to debug it, and you will not know how well your code satisfies the user requirements. If your program does not do exactly what is expected on the sample runs, then some logical or runtime error(s) need to be identified and fixed in your code. After you fix your code, you must test the program again. Testing techniques are not part of CSC 150. However, test cases or sample runs must be written or identified before you code your program. You have seen that the lab activities and programming assignments have several sample runs, each testing a different functionality or behavior of your program. You should test your programs with values at the extremes of what is valid input, values outside the valid ranges, and several values within the desired range to ensure all results and error handling work as designed.

R**ead the debugging section (Section 5.5) on page 120 of the course textbook, then complete the next project.**

Create a project and add the file prelab5.cpp to it. This program is intended to display a menu and allow the user to perform various simple math operations. The actual operations are simulated by statements displayed, don't be concerned about them. The problem with this program is that it doesn't flow correctly.

Some of the problems are really pretty obvious. Don't just jump in and fix them. Use the debugging process to find them.

Using the debugger tools and cout statements you insert into the code, track the value of the variable ***choice***. Fix the first problem that shows up. Then test each menu option, find a fix a couple of problems related to the menu itself.

There is nothing to turn in for this prelab.