Lab 6: Software Testing and Debugging

Overview

The purpose of these exercises is threefold:

1. to give you practice in identifying and correcting ***syntax errors*** in programs,
2. to give you practice in identifying and correcting obvious ***runtime errors*** in programs, and
3. to give you practice in devising ***software tests***.

The first problem asks you to compile a program, look at the error messages produced by the compiler, and then read the program and correct the mistakes. The faster you can do this, the more efficient you will become at writing syntactically correct software. You do not need to know what the program is supposed to do because the errors are not in logic but in using the language properly.

The second problem asks you to compile a program, which will compile without error, and then run it. When it runs, there will be obvious errors. You will not need to know what the program is supposed to do because the program's behavior will be unacceptable no matter what it is. For example, it might loop forever or produce no output or produce garbled output. You will have access to the source code so that you can correct the mistake(s) or at least suggest what is wrong.

The third problem asks you to test a program to determine whether it has any faults, and if so, to determine what they are. You will not be able to see the source code of the program, but you will be able to run it. This type of testing is called ***black box testing***, because the program is essentially like a black box, an object whose inside is invisible to you. You will be given a precise description of what the program is supposed to do, which is called a requirements specification. This specification can be used to decide what tests to use on the program.

A Bit About Software Testing

A ***test*** is an input to a program. A ***test case*** is an input together with the expected output. Running a program on a test input is meaningless unless you know what the output for that test input should be. For example, if the program's requirements specification states that the program computes the positive square root of any non-negative number, then a test case could be (input = 25.0, expected output = 5.0), because 5.0 is the positive square root of 25.0. Therefore, your job as a program tester is to devise a suitable set of test cases. Such a set is often called a ***test suite***.

Usually, before you start testing the program, you would create the list of all test cases – inputs and expected outputs – and put that list into a file, perhaps into a spreadsheet. When you run the program on a test, either it passes or fails the test. If a program fails the test, we say that the test ***exposed an error*** in the program. A spreadsheet might contain information such as when you tested it, whether the error was reported and corrected, and so on.

If a program passes all of your tests, you have to decide whether the program is error-free. Just because it passed all of the tests does not mean that it has no errors. It might mean that your test cases were not very good. If you think your tests were thorough enough to detect all possible errors in the program, then you can conclude that the program is error-free. You might decide that the tests were not thorough enough and continue testing further.

How can you decide whether the tests were thorough enough? Many people have spent much time trying to answer this question, and there are hundreds of scientific articles about the subject. We cannot begin to tackle that problem in very much depth here. However, there is some common sense that you can apply.

If the program's specification says that for all inputs that satisfy some condition C, the output is supposed to be X, and for inputs that satisfy a different condition D, the output should be Y, then clearly any good set of test cases would include some inputs that satisfy condition C and some that satisfy condition D. This type of thinking will suffice in the exercises that you have to do today.

Exercises

This is a three-part lab.

Problem 1. There is a directory named `syntaxbugs` in the `cs136labs/lab06` directory. Within it are five files named `syntaxbug01.cpp`, `syntaxbug02.cpp`, ..., `syntaxbug05.cpp`. None of these compile without error. Study the error messages and find a fix, then make sure your corrected file compiles. Name the corrected version of `syntaxbug0x.cpp` `syntaxbug0x_fixed.cpp`, x=1,2,3,4,5. In the corrected program, put a comment line like this:

```
/* ERROR FIXED HERE: */
```

to the right of the code on the line that you fixed. If there is more than one line, put this on each such line. Each corrected program is worth 0.5 out of 10 points for this lab.

Problem 2. There is a directory named `runtimebugs` in the `cs136labs/lab06` directory. Within it are three files named `runtimebug01.cpp`, `runtimebug02.cpp`, and `runtimebug03.cpp`. Each of these compiles without error but does not run correctly. Determine what is wrong with each and try to correct the code. Make sure that your corrected code compiles and runs correctly. ***Follow the same instructions as in Problem 1 and add comment lines to indicate where you fixed the program.*** Once it is correct, name the corrected version `runtimebug0x_fixed.cpp`, x =1,2,3. Each corrected program is worth 1 out of 10 points for this lab.

Problem 3. The third directory is named `softwaretesting`. It contains the following files:

program_specification A precise description of what the program `classifytriangle` is supposed to do.
classifytriangle1 A version of `classifytriangle` that does not work correctly.
classifytriangle2 A version of `classifytriangle` that does not work correctly.
classifytriangle3 A version of `classifytriangle` that does not work correctly.
angleof A program that can be useful in checking whether the output of `classifytriangle` is correct.

Your job is to find tests of the three `classifytriangle` programs that expose their errors. Each of these programs has one bug, i.e., a single mistake in its code that causes one or more inputs to result in incorrect output. Therefore, for each it is enough to find a single test case that exposes that bug. When you find the test case, your job is to characterize what the program is doing wrong. In other words, it is not enough to say, for example, that the program does not work when given the input (5,12,15). You also have to say something like, "it appears that the program sometimes says a triangle is acute when it is not", or "it appears that it sometimes accepts inputs that are not valid triangles." You will get partial credit if you find a bug but cannot describe it accurately. You will get full credit if you describe it accurately as well.

In case you do not remember your geometry, a triangle is ***acute*** if all angles in it are less than 90 degrees, it is ***obtuse*** if there is a single angle greater than 90 degrees, and it is a ***right*** triangle if there is 90 degree angle in it. These are mutually exclusive possibilities. A triangle is ***scalene*** if no two sides are equal, ***isosceles*** if exactly two sides are equal, and ***equilateral*** if all three sides are equal. The program `angleof` accepts command line arguments like `triangleclassify`.

```
    angleof a b c
```

will output the number of degrees in the angle between the sides whose lengths are `a` and `b`. You might find this useful when you need to know whether the `classifytriangle` program correctly identified the type of triangle.

For this problem, you should create a single file named `triangletests`. For each of the buggy `classifytriangle` programs, write the test case that exposes an error and a description of what you believe is the error. There should be three short paragraphs in this file. Each correctly identified error is worth 1.5 out of 10 points for this lab.

What and How to Submit

Submit your work, however complete it is, by the end of today's lab, i.e., before the end of the class at 2:00 P.M. The instructions are just like the previous lab's:

1. Create a directory in
   `/data/biocs/b/student.accounts/cs135_sw/cs136labs/lab06/submissions`
   whose name is your *username*.
2. Put all of your files into this directory. Make sure they are named correctly. There should be five files from Problem 1, three from Problem 2, and one from Problem 3. You will lose 5% of the grade if you misname the file!
3. Change the permission on the directory that you created so that no one else can read it. You do this with the commands
   ```
   $ cd /data/biocs/b/student.accounts/cs135_sw/cs136labs/lab05/submissions
   $ chmod 700 username
   ```

Remember to add the comment lines described in the problems above. ***There are absolutely no extensions to the deadline for this lab.***