Laboratory 7  Debugging

7.1 Objectives

- To be able to put debugging statements into a program.
- To be able to detect and correct logic errors in programs.
- To understand functions better.

7.2 Prerequisites

You should know how to write a loop before you start this lab.

7.3 Correspondence to Text

This lab does not correspond well to an exact section in the text, but sections 2.8 and 5.9 cover some of the material in this lab.

7.4 Introduction

So far we have written many programs, but if the program doesn’t work the first time, there has been no way to determine what went wrong. This lab will give you several hints for how to debug your programs, but the best method is still careful design and tracing through your program before beginning your coding.

7.5 Syntax Errors

The first kind of error that you might run into when programming is syntax errors. Finding syntax errors is a matter of a lot of experience in some cases, but also paying close attention to syntax error messages. If the message says “parse error, line 30” that means that somewhere on or before line 30 there is an error that prevented the compiler from translating the C++ code into object code. Let’s look at a program full of syntax errors and try to figure out some of the problems.

- Copy lab7.cpp. lab7.cpp is the triathlon program that we wrote in one lab.
- Compile the program.

You should get errors similar to the ones below:

lab7.cpp:34: unterminated string or character constant
lab7.cpp:23: possible real start of unterminated constant
This message usually means that you have forgotten an end quote. But the error message is also
telling you that it found the end at line 34, but that it may have started at line 23.

- Find the missing end quote, fix it, and recompile.

Now, depending on what compiler you use, you might actually get more errors than you had
before. This is because some errors mask other errors; that is, because the C++ compiler was
looking for an end quote, it skipped over some of the other errors. You should see something
similar to the following error list now:

lab7.cpp: In function ‘int main(...)’:
lab7.cpp:22: no match for ‘_I0_istream_withassign & << float &’
lab7.cpp:28: warning: implicit declaration of function ‘int CalcLegTime(...)’

Let’s look at the error messages one by one. The first line, lab7.cpp: In function ‘int
main(...)’:, just means that the error messages listed below are in the function named main().
It is not an error message itself. The next error, no match for or no member function istream
usually means that a << is backwards. The next warning message, implicit declaration,
means that the compiler hasn’t found a declaration for CalcLegTime. Why is this?

- Fix the error and warning messages by changing lines 14 and 22. Remember that C++ is
case-sensitive.
- Compile the program again.

7.6 Debugging

Once you have successfully compiled your program, you are not through. Now, you need to get
your program working correctly. If it doesn’t, you need to find out why. Your instructor may
tell you about a debugger that you can use that will help you step through program. But even
without a debugger, there are many steps you can take to find where your program is going wrong.
Often, adding debugging statements in the form of couts can help you pinpoint a problem. A
debugging statement is simply a statement that allows you to output what the program is doing
at a particular point, and where the program is.

7.6.1 Digression on Libraries

In order to see how some of this works, we are going to look at a relatively simple program, but
one that seems complex because it uses three different files. A library consists of a header file and
its associated function definition or implementation file. Header files as you know typically have
declarations of constants and functions; the definitions for these functions (i.e., the function header
and body) are in other (compiled) files. This header file has a .h extension; the implementation
file has a .cpp extension. To see how the program that we will use works, run the program temp.
Your instructor will tell you where to locate this file.
• Run temp, and enter 32.0 and 'A' to convert the temperature 32.0 degrees F to 0 degrees Celsius.

• Copy the files temp.cpp, heat.cpp, and heat.h.

heat.h is a header file that contains the declarations for the function definition contained in heat.cpp. These two files together are a library the header file and the implementation. The heat.cpp must be compiled separately from “your” file (temp.cpp). This is so that the heat library can be reused by many programs (just as you reuse iostream and math in many programs).

• Compile the library into object code. Refer to the directions that your instructor gave you for lab 3 for how to do this. After compiling, you should have an object file possibly named heat.o which will remain in your library for linking (possibly with other object files) later. The process of linking will create an executable file.

• Compile and link your source with the heat.o file (or whatever your object file is called). Your instructor will tell you how to do this. You may then run your program; it should perform in the same way as the temp that you ran from the public directory.

• Run it and use the same values. You should get the same answers.

7.6.2 Finding the problem

This program looks as if it runs correctly. In fact, I took this program directly from a textbook (not yours), so one would think it would run correctly. However, there is an error in it. Your goal during this lab is to find the error and fix it.

• Test all of the options with the values 32.0F, 0.0C, and 273K. All of these should be equivalent. Test all six options and determine which are incorrect.

• Answer the questions under section Checking Conversion in lab 7 questions.

7.7 Debugging Statements

7.7.1 “x is 0” debugging statements

There are many ways to find an error. Tracing through a program by hand is one of the best. We’ve done that before. In this lab, we will use cout statements to help debug “programmatically” (by adding statements in the program itself). It is also often useful to know exactly where you are in the program and the value of variables at a particular point in a program. This technique might help us debug the temperature program that we have. There are two conversions that are done incorrectly: C and D. There are several reasons that the output could be incorrect. One of the following must be true: The output could be incorrect because the input is read in incorrectly; because the values are being passed to the functions incorrectly; because the calculation in the function is being done incorrectly; because the value is being passed back from the function.
incorrectly; or because the value is being corrupted after it has returned from the function. To check each of these, we will add `cout` statements at various points in the program to locate the problem.

- Copy the program `heat.cpp` to `heat2.cpp` and `temp.cpp` to `temp2.cpp` so that you do not destroy your old versions. This is a good debugging tip. When you start to debug a program, do not modify the original. Instead, copy it to another name, and modify the new file. Now, you'll modify `heat2.cpp` and `temp2.cpp` until you find the problem; then, you'll go back and fix `heat.cpp` and `temp.cpp`.

- Check the `if` clauses for ’c’ and ’d’ in the function `Convert` (in the file `temp2.cpp`) to ensure that the right functions are being called. Add a statement in these clauses to display the value of the returned expression. To do this, assign the value returned to a variable `temp` that you declare in the function `Convert`. Within the `if` clauses, you will need to add curly braces since you’re making a compound statement. The code should now be:

```cpp
float Convert (float Temperature, char Conversion)
{
    float temp; // for debugging purposes
    Conversion = toupper(Conversion);
    if (Conversion == 'A')
        return FahrToCelsius(Temperature);
    else if (Conversion == 'B')
        return CelsiusToFahr(Temperature);
    else if (Conversion == 'C')
    {
        temp = FahrToKelvin(Temperature);
        cout << "C: Value returned is " << temp << endl;
        return temp;
    }
    else if (Conversion == 'D')
    {
        temp = KelvinToFahr(Temperature);
        cout << "D: Value returned is " << temp << endl;
        return temp;
    }
    else
        <rest of if statement>
}
```

This will display the value returned by the function so that you can check to see if it’s correct there, and if only later it becomes incorrect.
Next, edit heat2.cpp, the functions FahrToKelvin and KelvinToFahr. First, display the input value (maybe it’s not getting passed correctly). Next, display the value that is being returned (maybe the calculation is being done correctly, but it’s getting passed back incorrectly). Finally, let’s perform the calculation in a different way (maybe the calculation is being done incorrectly). When converting from Kelvin to Fahrenheit temperatures, we could do the conversion from K to C then from C to F. Let’s cout that also. So, the function KelvinToFahr in heat2.cpp should now be:

```cpp
{
    // the next statement ensures that the value was input and passed correctly
    cout << "The beginning value in Kelvin to Fahr is " << KTemp << endl;
    // next, print the value to be returned to ensure its correctness
    cout << "The ending value is " << (KTemp + 273) * 1.8 + 32.0 << endl;
    // the next 2 statements do the calculation differently as a check.
    cout << "The celsius equivalent is " << KelvinToCelsius (KTemp) << endl;
    cout << "The Fahr equivalent of that is " << CelsiusToFahr (KelvinToCelsius (KTemp)) << endl;
    return (KTemp + 273) * 1.8 + 32.0; // this was previously the only line
}
```

Modify the function KelvinToFahr and FahrToKelvin to include the couts specified above (you will need to modify the FahrToKelvin couts to display the Ftemp not the KTemp for the first cout, and so on for the next three couts). In both cases, the return statement should not be modified.

- You will also need to add the statement:
  ```cpp
  #include <iostream> // included only for debugging purposes
  ```
  at the beginning of the file heat2.cpp. This was not in the file before because no couts or cins occur in the program.

- Compile the program. Remember that heat2.cpp and temp2.cpp should be compiled together into one executable.

- Run the program. Use the value 32F and 273K to test the conversions (because we know what the answer should be for those values). When I did this, my output for the Kelvin to Fahrenheit conversion was:

This program converts a temperature between the scales Fahrenheit, Celsius, and Kelvin.

Please enter the temperature you wish to convert: 273

Please enter:
  A - to convert Fahrenheit to Celsius.
  B - to convert Celsius to Fahrenheit.
C - to convert Fahrenheit to Kelvin.
D - to convert Kelvin to Fahrenheit.
E - to convert Kelvin to Celsius.
F - to convert Celsius to Kelvin.

Your Choice ? D
The beginning value in Kelvin to Fahr is 273
The ending value is 1014.8
The celsius equivalent is 0
The Fahr equivalent of that is 32
D: Value returned is 1014.8

-- The converted temperature is 1014.8.

Your output may be slightly different because your couts may be slightly different. However, you should have the same numbers, and should be able to figure out the problem based on our debugging statements. Remember, there is a limited number of possibilities for the problem. It is one of the following: The output could be incorrect because the input is read in incorrectly; because the values are being passed to the functions incorrectly; because the calculation in the function is being done incorrectly; because the value is being passed back from the function incorrectly; or because the value is being corrupted after it has returned from the function. We’ve checked each of these.

- Find out what the problem is, and fix it.
- Go through the same process for testing the ’C’ option, converting Fahrenheit to Kelvin.
- Fix the Fahrenheit to Kelvin problem.
- Save the program, recompile it, and test it.
- Modify the original program, recompile it, and test it. In heat.cpp and temp.cpp (the original files), do not use any additional cout statements; we added these just to figure out where the problem was. Now that you know that, just fix the problem.

7.8 Synthesis

- Copy the program lab7end.cpp. Name it the same thing in your directory. All this program does is perform several calculations. The program reads in two numbers (x and y) and calculates: $x^2 - x! + x^y$
x! is read “x factorial” and is an abbreviated way of writing: $x * (x-1) * (x-2) * (x-3) * ... * 2 * 1$. For example, $4! = 4 * 3 * 2 * 1 = 24$. Thus, if $x$ were 4 and $y$ were 5, the equation would be: $4^2 - 4! + 4^5 = 16 - 24 + 1024 = 1016$.

- Copy the program to another name as well (e.g., lab7end2.cpp). There are two logic errors. There is nothing wrong with any of the variable types (i.e., all ints should remain ints and
all floats should remain floats). One note about types: this program has many variables of type double. This is simply a type of float. When you see it, think float (you can even change them all to floats if you want, and the program will still work correctly once you’ve fixed the logic errors which have nothing to do with the types of the variables or the parameters).

- Add cout debugging statements to lab7end2.cpp, and compile and test lab7end2.cpp until you have found the errors. You might also find it helpful to hand trace the program to find the errors (getting a printout of the file may help you do this).

- Fix the errors and test lab7end2.cpp.

- Fix the errors in lab7end.cpp, and compile and test it. There should be no debugging statements in it, but the errors should be corrected.