Lab 04: Debugging Code

# Objectives:

* Learn about the C Compiler messages
* Become familiar with various types of Compiler errors
* Learn coding practices to help avoid unintentional errors

# Starting Point:

* [lab04-1\_1.c](https://drive.google.com/open?id=1pmdsWtPCre7AMFbF3S6qZXCSqha6NZmR)
* [lab04-1\_2.c](https://drive.google.com/open?id=19pgKa9C6ePLNWgtoWzyyd5AdEIQSgTnt)
* [lab04-1\_3.c](https://drive.google.com/open?id=1BaYZLzXVglGsSG5T_FXaDXzpXPkr27jS)
* [lab04-1\_4.c](https://drive.google.com/open?id=1XQw7AvTO4kovjCy58e7TnUnHA_8j5Lyz)
* [lab04-1\_5.c](https://drive.google.com/open?id=1noVjlpo1tSbUNf6QnJ6jEr_KJQ_4I2qY)
* [lab04-2\_1.c](https://drive.google.com/open?id=1DywHakoZ6y5abTa4MmYBjLZBPhjcOupv)
* [lab04-2\_2.c](https://drive.google.com/open?id=1UTdeipRyyBOcZQpGaWPfaC0myQeIECSy)
* [lab04-2\_3.c](https://drive.google.com/open?id=1JwETfsyf7Hr-AU6-1TrgwnTPBeMhVVGv)
* [lab04-2\_4.c](https://drive.google.com/open?id=1ziAwd7RK8PLpxRWtm5cFfP3yJ668jtdA)
* [lab04-2\_5.c](https://drive.google.com/open?id=1iHOJSbg6c8yQGnTd0Ckjrf8HFFfd3lZI)
* [lab04-3.c](https://drive.google.com/open?id=1SouvM8P4DvZpYlyKeCpmmBZlkSnF84Yg)

# Turn-In:

**Upload one .pdf file** containing the following before the start of lab next week

(Before Lab 05).

The fixed source code for each of the three parts at the end of your lab report

1. Screenshots of all your code running
2. Answers to the questions at the bottom of the lab manual

**Upload one .zip** file of your **lab04** folder. Title this **firstname\_lastname\_lab04.zip**. Before zipping, delete all generated .exe files (not the **ds4rd.exe**).

# Process:

In this lab, you will learn more about the GCC Compiler’s debugging messages.

Knowing how to read the debugging messages for errors and warning can be a

great skill to know when your program is not functioning correctly.

## C Compiler:

By now you have used the C compiler to create several programs. Often, you will find

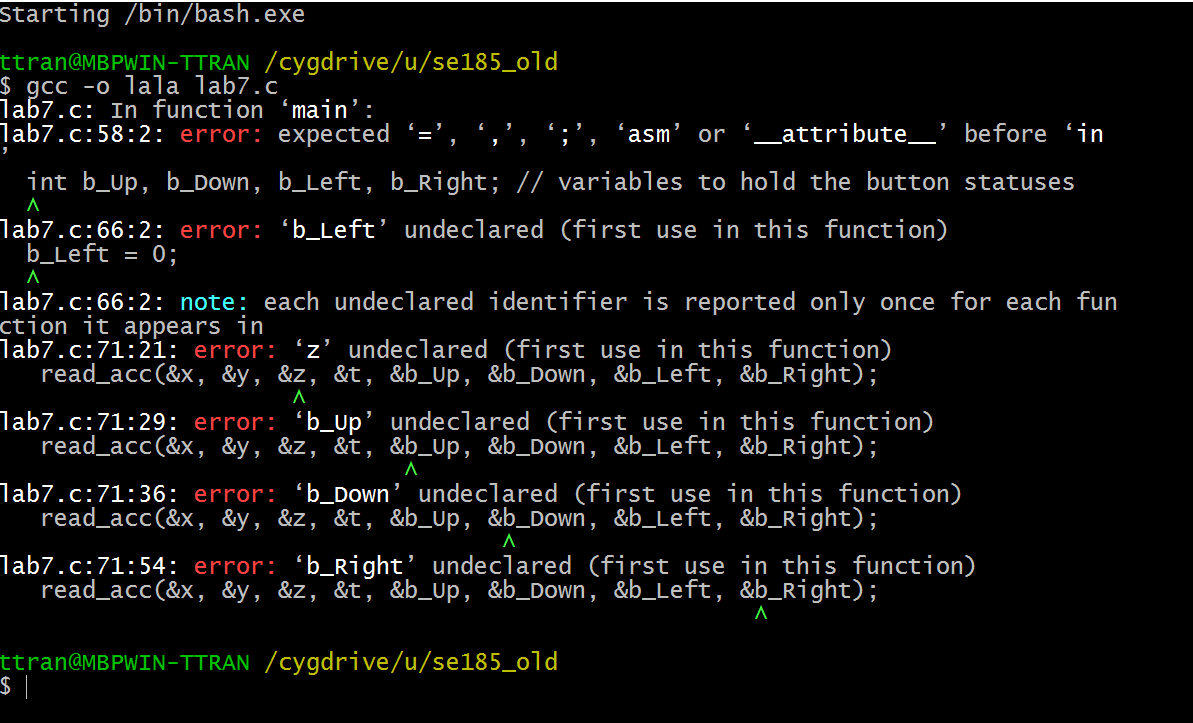
that the programs you create either do not compile correctly or the result doesn’t match

your expectations when you run it. You may have seen this when you made errors in

your code. In this lab, you will examine several different programs that have either a

***compilation error****,* an error during the compilation “gcc …” step, or an ***unexpected***

***value****,* a result was given that was not expected,error.



* An example of a compile error.

## Part 1: Compiler Errors:

1. You have likely noticed when you try to compile programs that it sometimes goes wrong; the compiler was unable to compile your source code into a working program.

Luckily, the compiler will often output helpful messages which we can use to find what is wrong with our source code, which helps us fix these issues.

1. Try compiling **lab04-1\_1.c,** which will fail to compile. Let’s take a look at the output.

For example, you should see a line with the message:

|  |
| --- |
| **lab04-1\_1.c:29:5:** error: expected ‘**;**’ before ‘**scanf**’  scanf("%d", &i);  ^~~~~ |

This tells us the compiler failed at **line 29**, **5 characters in**, in **lab04-1\_1.c**, and was expecting a semicolon somewhere before that line. Checking our code, we can see that there is indeed a missing semicolon after our **printf** on   
**line 29**. Add this missing semicolon.

Take a look at the rest of the output and fix the errors. It may be a good idea to fix just one error at a time. After you fix one error, save and re-compile your code to see what other errors may exist.

**Comment out lines you changed and put the correct code below.**

**For new lines added, put a comment stating what you’ve added.**

Compile, run, and test to take make sure you’ve fixed all issues.

1. Now do the same for **lab04-1\_2.c**, **lab04-1\_3.c**, **lab04-1\_4.c,** and **lab04-1\_5.c.** Take a look at the output from the compilation errors and try to fix the issues.
2. Include a copy of the fixed source code and correct output (i.e. screenshot) in your lab report.

## Part 2: Unintended Results (Logic Errors):

1. Sometimes your program will compile successfully, but it doesn’t behave or give you the output you may expect. These kind of problems are a bit more difficult to find and fix.
2. Compile and run **lab04-2\_1.c**. If you run it a few times, you may notice the output isn’t always correct for the number you input. Open **lab04-2\_1.c** and try to find where the problem is coming from. Once you find the problem(s), comment out the line and add the corrected line of code below the commented line. Compile, run, and make sure the program has been corrected.
3. Do the same for **lab04-2\_2.c, lab04-2\_3.c, lab04-2\_4.c,** and **lab04-2\_5.c.** There are comments at the top of each program that describe what the program should do.

**Remember to comment out lines with the problems and add the corrected line below it. Make sure to compile and test again to make sure the problem has been fixed. Include the corrected code and screenshots of the code running in your lab report.**

## Part 3: Putting It All Together:

Now that you have investigated compiler errors and unintended results, you should be familiar with some of the common issues that can appear when writing C code.

1. Download the **lab04\_3.c** code linked above.
2. This program will have a mixture of compiler issues and unintended results. As with the two sections above, please go through the code and fix any issues that you think you see. Remember to **comment out lines with the problems and add the corrected line below it**.

**Hint**: There is a GCC compiler flag, **“-Wall”**, that may be useful.   
  
This flag will tell the compiler to warn you of potential issues in your code. These issues do not cause compile errors, but may cause issues when you run your code. The warnings are often the result of unintended results issues in code. It is good practice to run your code with the **“-Wall”** flag to try and find issues. Please note that the warnings can sometimes be ignored, as you may want to use a certain type of behavior that **“-Wall”** thinks is bad. An example of this is not using the return value of a function. We often ignore the return value of scanf() and “-Wall” may give you a warning when you turn it on for this behavior.

1. Include a copy of the fixed source code and program output (i.e. screenshot) in your lab report

## Questions:

**For each** program in **Part 1 and 2** that you fixed, answer the following:

1. What changes did you have to make to fix the program? Please list the line number with the changes that you made.
2. What kind of issue caused this problem?
   1. Example: missing semicolon, wrong variable types, missing brackets, etc.

**For Part 3:** What is the purpose of the **“-Wall”** flag? Do you have to fix all of the messages that it gives you?

## Ending the Lab Session:

1. Be sure you will be able to have access to your code and data when you work on your lab report.
2. Leave your source code on your U: drive.
3. Log off before leaving your workstation in the lab!