|  |  |  |
| --- | --- | --- |
| |  | | --- | | Lab 11 – Debugging and GDB  CPSC 2311- Fall 2020 | |  |

# Introduction

The goal of the lab is to give you practice with debugging and GDB: The GNU Project Debugger. GDB is a powerful tool, understanding how to use it is a good skill to have. You should have learned about GDB in either CPSC 1010 or CPSC 1020 if not both.

**Due:** Sunday, November 15, 2020, midnight.

# Lab Instructions

You will first watch a video that walks you through how to use GDB.

<https://www.youtube.com/watch?v=uhIt8YqtmuQ&feature=youtu.be>

**Part 1: 15% of grade**

I have provided you with a input.txt, driver.c, functions.c, and functions.h files. You should compile these files. You will find these files have at least 1 error and several warnings. These are compile time errors and warnings. Warnings are a unacceptable as errors. You **must** clear all errors and warnings.

**Part 2: 50% of grade**

Once you have cleared the warnings and error, compile the code using the flag -g, run it with the input.txt file. You should get a segfault. This is a runtime error. Set breakpoints to places the error could possibly be. Rerun using GDB to find the segfault. You **must post several screenshots** showing the use of GDB. After clearing the runtime errors complete part 3. Even if you figure out the error before using GDB, you **must** demonstrate the use of GDB to find the error or you will see a deduction of 50 points. I purposely did not make the errors hard to find, but you must use GDB in this lab. So play around with it and post **several** screenshots of your use of GDB. Knowing how to use GDB will help you in future classes.

**Part 3: 35% of grade**

Now that you have cleared the compile time and runtime errors, you will add a couple functions to this code, as well as, add the necessary code to the driver to produce the appropriate output, shown toward the end of the document. The functions are described below:

**int calculateVal(int\*\* mat, int size);** - This function will return a sum of all values of the matrix **except** those that are located on the left and right diagonals. As an example, consider the following 5 X 5 matrix:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
| 0 | 2 | 2 | 2 | 2 | 2 |
| 1 | 2 | 2 | 2 | 2 | 2 |
| 2 | 2 | 2 | 2 | 2 | 2 |
| 3 | 2 | 2 | 2 | 2 | 2 |
| 4 | 2 | 2 | 2 | 2 | 2 |

The values in red, highlighted in blue are the left diagonal and the values in red highlighted in yellow are the right diagonal. The 2 in the middle, highlighted in purple, is in both categories. This function will calculate the values of all the remaining numbers in the matrix. (32 for this matrix).

The function **calculateVal** will call functions **isRightDiagonal** and **isLeftDiagonal**.

**bool isRightDiagonal(int size, int row, int col);** - This function returns true if a given element in the defined 2D array is part of the right diagonal. This can be written with one line of code using a ternary operator. Whether you write this function in 1 line of code or not, you **must** use a ternary operator in this function.

**bool isLeftDiagonal(int row, int col);** - Like the function above, this function returns true if a given element in the defined 2D array is part of the left diagonal. Also like the function above, it can be written with one line of code using a ternary operator. Whether you write this function in 1 line of code or not, you **must** use a ternary operator in this function.

**HINT:** Use the matrix above to help you determine the algorithm for the isRightDiagonal and isLeftDiagonal funtions. I strongly suggest you list out the matrix element locations that make up both the left and right diagonals. You should see a pattern forming and can deduce the algorithm for these functions.

Example: mat[0][0]

mat[1][1]

etc.

**Here is a sample output for the input.txt file given to you.**

**The format of your output must be like the following:**

01 01 01 01 01

01 01 01 01 01

01 01 01 01 01

01 01 01 01 01

01 01 01 01 01

Total = 16

You should test your program using several test files.

**You must also include a makefile with your code that will include make (that will compile the program), make run that will run the program passing input.txt on the command line and make clean.**

You must add a header to all files similar to the following:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

\*Your name \*

\*CPSC2310 Fall 2020 \*

\*UserName: \*

\*Instructor: Dr. Yvon Feaster \*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Submission Instructions

**tar zip all files naming the tarred file <username>lab11.tar.gz ; submit it to handin in the lab11 folder.** ([http://handin.cs.clemson.edu)](http://handin.cs.clemson.edu)/)

**You must check your tarred files after submission to handin. If any of your files are corrupt or missing you will receive a 0 on this lab.**