

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. You may have learned about GDB in either CPSC 1010 or CPSC 1020 if not both.

This is an individual lab, you are not allowed to get help from anyone but your lab TAs.

Due: 02/13/23 11:59pm

Lab Instructions

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

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

<https://www.youtube.com/watch?v=xQ0ONbt-qPs>

Part 1: 15% of grade

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

Part 2: 35% 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. I purposely did not make the error hard to find. You have probably already spotted one or more of the error(s) causing the segfault. Do not fix it yet. Use GDB to find the error(s). Set breakpoints to places the error(s) could possibly be. Rerun the program using GDB to pinpoint 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(s) or you will see a deduction of 35 points. Play around with GDB posting **several** screenshots of your use of GDB. Knowing how to use GDB will help you in future classes.

Part 3: 50% 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 value 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** functions. I strongly suggest you list 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
```

Notice the output is 2 digits.
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 Spring 2023         *
*UserName:                     *
*Instructor: Dr. Yvon Feaster *
*****/
```

Submission Instructions

We will be submitting this lab via gradescope. The following files should be submitted (make sure the names of your code files are exact):

driver.c

functions.h

functions.c

makefile

A **pdf** containing your gdb screenshots and/or descriptions

Contact Eliza with any Gradescope-specific questions (esorber@clermson.edu or Teams message)

In your make target, name your executable <LAB7>. If you name it differently, the autograder won't be able to find your executable and you will lose points on the Make command test case. Also, make sure that you don't submit this executable to gradescope. You should ONLY submit the following files:

driver.c

functions.c

functions.h

makefile

screenshots.pdf (You can name this file whatever you'd like, but please submit it as a pdf so that the TAs can view it in Gradescope (.docx doesn't work))
