CLEMSON School of COMPUTING

Introduction
This week's lab will introduce you to a debugging tool called `gdb.` As your programs begin to get more complicated as you use arrays, functions, and pointers, you may encounter run-time errors that can be frustrating to debug using `printf` statements.

Lab Objectives

By successfully completing today's lab, you will be able to:

- Use the `gdb` utility to determine where run-time errors are occurring within a C program.
- Explain how to use the `gdb` commands to debug a program

Prior to Lab

- You should be familiar with how to write a multi-file C program and use arrays.
- You should be familiar with declaring a pointer variable, modifying where that pointer variable points to, and dereferencing a pointer.

Deadline and Collaboration Policy

- This assignment is due by 11:00 PM on Friday (11/2/2018) via Canvas.
    - More instructions on what and how to submit are included at the end of this document.
- You should write your solutions to this lab by yourself. In this lab, **you should not talk about specific code to anyone but a course instructor or lab teaching assistant. If you speak with a TA at the TA help desk, you should document the TA's name in the academic honesty code header.**
- Your zyBooks chapters and lecture slides are available resources you can use to assist you with this lab.

Lab Instructions

`Segmentation fault`

Ah, those two words strike fear in C programmers everywhere! Why? A segmentation fault (or segfault) is an example of a run-time error where the computer's hardware is notifying the operating system that a program is trying to access a restricted area of memory. In a course such as CPSC 1010-1011, these errors are usually due to attempting to access memory outside the bounds of an array or trying to dereference a null pointer. Chances are that you've already seen this error message and then had to use a ton of `printf` statements to try and narrow down the location of the error within your programming. *Detecting run-time errors in a C program that has compiled with no errors or warnings is often the most time-consuming part of programming!*

Good news! The purpose of today's lab is to introduce you to a tool that can help save you time in identifying where your errors are!

The `gdb` tool is known as a debugger, or a computer software program/tool that assists a programmer in finding errors within their computer programs. This tool will allow you to execute a program, stop at various points within the program (known as **breakpoints**), print values of variables at specific points, and other useful tools. For most of the students in early courses in computer science, the `gdb` tool is incredibly helpful in finding the statement of code that is where your run-time error is occurring.

The `gdb` tool will be used as a command-line program. To launch `gdb`, you simply need to run the `gdb` command. However, you also need to prepare your program for use by the `gdb` debugger. The gcc compiler has a `-g` flag option that compiles your program with debugging information—this information isn't normally needed if your program compiles and runs without problems. Thus, you only need to use the `-g` option when you are intending on using the `gdb` debugger. Assume that you have a program called `myProgram.c` that you wish to compile into an executable object file called `myProgram.out`

```
prompt % gcc -g Wall myProgram.c -o myProgram.out
```

Once you've compiled your program using the `-g` flag, you can load your program into `gdb` by running the following command:

```
prompt % gdb ./myProgram.out
```

When the `gdb` tool launches, you will see several lines of text which includes copyright information, version number, legal disclaimers related to warranties, links for reporting bugs, and a link for additional information on the `gdb` tool. You will see a cursor prompt next to `(gdb)`. To quit `gdb`, simply type "`quit`", "`q`", or press Control + 'D' and you will return to the Linux command line prompt. However, there is another way to launch the `gdb` tool:

```
prompt % gdb -tui ./myProgram.out
```

The version above will launch the `gdb` tool, but split the screen to show the source code in the upper-half and the `gdb` prompt on the lower half (called the "text user interface"). If your top-half says "No source available", type the "`list`" command at the `gdb` prompt to load your source code. Note, you can also enable the text user interface mode when you are in the `gdb` tool using the `tui` enable command.

Here's a breakdown of useful commands within `gdb`:

Command	What it does
`q` (or `quit`)	Exit the gdb tool
`tui enable`	Activate the TUI mode
`tui disable`	Return to the console prompt mode of `gdb`
`b main` (or `break main`)	Causes the execution of the program to pause at the start of the main function
`b 15` (or `break 15`)	Will cause the program execution to stop at line 15 (or another number that you specify)
`r` (or `run`)	Start the current program that's loaded
`n` (or `next`)	Run the next line of course code

Command	What it does
`c` (or `continue`)	Continue without stopping until the next breakpoint, error, or the program finishes
`p x` (or `print x`)	Print to the screen the current value of the variable x (you can change x to be whichever variable name you want)
`d x` (`display x`)	Display the current value of x at each `gdb` command prompt
`r`	Restart the currently running program using the previous command line

See the additional resources section for other references that can be helpful.

Assignment

The purpose of today's lab will be to have you investigate a program that compiles, but does not run successfully. When you run the program, you should encounter the "Segmentation fault" or "Bus error" messages. A "Bus error" message is a hardware message that can happen when the operating system cannot allocate the memory you've requested. Typically, this can happen when you are allocating too much memory using the malloc command. Bus errors also happen when a process tries to access memory that's not a valid memory address. One way to think of this: segfaults indicate that there's an invalid access attempt to a valid memory location while a bus error indicates an attempt to access an invalid memory address.

This week's lab is largely exploratory!

Lab Exercise

Download the `files.zip` file from Canvas into your Lab 10 directory. You will find two .C program files.

- Compile and try to run `example1.c`. It will segfault.
  - Open up the file `example1-tutorial.pdf` and follow the step-by-step instructions on how the `gdb` tool can be used to figure out where the segmentation fault occurs.
  - Take this step seriously! This step is intended on helping you get familiar with another command-line tool, which, as you already know, takes time and practice.
- Compile and run `example2.c`.
  - Use the `gdb` tool to isolate where the error(s) occurs.
  - Answer the questions in the file `lab10.text`
- Fix the error or errors in `example2.c` and include good function comments.
  - Each function should include a purpose statement, and a rough description of how it operates, making sure to touch on parameters and return values.
  - Within each function, make sure to add comments about
    - errors you've fixed (how you fixed them, and what was wrong)
    - what the code is doing every so often, or whenever you think it needs explanation.
      - (A simple `printf` statement doesn't require comments, but a large block of them and some mathematical operations on variables would – check coding style guidelines from the recent labs if you need help with this, or ask a TA)
- Answer the questions on Canvas under Lab 10 Quiz.

Submission Guidelines

- Submit your corrected `example2.c` program to Canvas assignment associated for this lab by 11:00PM on Friday (10/26/2018) and answer the questions under Lab 10 Quiz on Canvas by that date/time. You should verify within Canvas to make sure your submission was uploaded correctly and in its entirety. You must include the academic honesty header and *explicitly* list the names of TAs that you asked for help outside of lab.

Grading Rubric

- If you are not present and attending lab on Tuesday and Thursday, you will not receive credit for this lab.
- If you do not check in your code with a TA on Thursday, you will not receive credit for this portion of the lab.
- If your assignment is not submitted fully to Canvas by the due date, you will not receive credit for that portion of the lab.
- If you do not complete the Quiz on Canvas by the assigned date/time you will not receive credit for this portion of the lab.
- Your assignment will be graded out of 100 points. The approximate grading distribution is:

Program `example2.c` runs correctly without errors or warnings	30 points
Proper code formatting, commenting, headers using good programming conventions	15 points
Canvas Lab 10 Quiz questions	55 points

Additional Resources

- Professor Hochrine's Guide to GDB
  http://people.cs.clemson.edu/~chochri/Courses/Documents/gdb-basics.pdf

- Beejs guide
  https://beej.us/guide/bggdb/

- Quick Reference Guide (on Canvas)