Week 6 Recitation Exercise

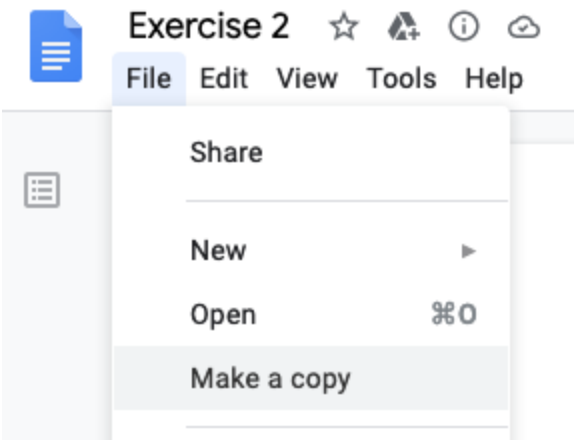
CS 261 – 10 points

In this exercise, you will explore how to use two important debugging tools, GDB and Valgrind. GDB is a debugger that allows you to see what's happening "inside" a program as it executes. We can use GDB to trace down program crashes and any other kind of bug. Valgrind (or rather a tool within Valgrind called Memcheck) is a memory error detector. We can use Valgrind/Memcheck to understand when and how we're making mistakes related to memory usage. This recitation exercise will specifically involve working in a small group to debug some broken programs using both GCB and Valgrind.

Follow the instructions below, and when you're finished, make sure to submit your completed exercise on Canvas. Please submit one copy per group (as long as all group members' names are listed below, it's fine if just one person submits). Remember that this exercise will be graded based on effort, not correctness, so don't worry if you don't get all the right answers. Do make sure you *try* to get the right answers though.

Step #1: Organize your group and create your submission document

Your TA will assign you to a small group of students. Among your group, select one group member to record your group's answers for this exercise. This group member should make a personal copy of this document so they are able to edit it:



Step #2: Add your names

Next, your group's recorder should add the names and OSU email addresses of all of your group's members in the table below:

Group member name	OSU email address

Step #3: Grab some buggy array code

Now, take a look at the code in the following GitHub repo:

https://github.com/osu-cs261-f21/recitation-6

Clone this repo onto your own machine or onto one of the ENGR servers, but note that you won't be able to push changes back to the repo, since your access to it is read-only. If you'd like to be able to push any changes you make to the code back to GitHub, you'll have to make a fork first and then clone and work with the fork.

Ultimately, you'll want to be able to run this code on the ENGR servers, since GDB and Valgrind are already installed there, so make sure you're able to get the code onto those machines.

Step #4: Watch the first GDB/Valgrind demo video

In case you've never used GDB or valgrind before, I've prepared a couple tutorial videos based on some of the code in the recitation repository. The first tutorial video walks through how to use GDB and Valgrind to debug the program **`buggy-pointers.c`**. You can find that tutorial video here:

https://media.oregonstate.edu/media/t/1_2s6m3klv

Feel free to follow along with the tutorial to debug your own copy of **`buggy-pointers.c`**.

Step #5: Watch the second GDB/Valgrind demo video

The second tutorial video walks through how to use GDB and Valgrind to debug the program **`buggy-factorial.c`**. You can find that tutorial video here:

https://media.oregonstate.edu/media/t/1_k1xlf75p

Feel free to follow along with the tutorial to debug your own copy of **`buggy-factorial.c`**.

Step #6: Use GDB and Valgrind to debug a program

Now that you've seen a few examples where GDB and Valgrind were used to identify bugs in different programs, it's time for you to do some debugging on your own. In particular, try to use GDB and Valgrind to debug the program **`buggy-list-sort.c`**.

The program in **`buggy-list-sort.c`** specifically generates a short linked list containing random integers and tries to sort it using bubble sort. However, something in the code is broken, and the program crashes with a segmentation fault before the sorting is complete.

Try to use GDB and/or Valgrind to diagnose and fix the problem. For your reference, here's a list of some of the GDB commands that were demonstrated in the demo videos above (with links to documentation on each command included):

- `run` – starts your program from the beginning. Command line arguments to your program can be specified with the `run` command.
- `break` – tells gdb to pause the execution of your program once it reaches a specified line in your source code. This is called setting a ***breakpoint***.
- `list` – prints out the lines of source code near the one currently being executed or near a specified location.
- `print` – prints out the value stored in a specified variable, etc.

- `step` – tells gdb to execute the very next line of code when it's paused at a breakpoint. If the next line of code is inside a function call, the `step` command enters that function.
- `next` – a lot like the `step` command; tells gdb to execute the very next line of code when it's paused at a breakpoint. However, if the next line of code is inside a function call, the `next` command runs that function without entering into it.
- `watch` – tells gdb to pause whenever the value of a specified variable changes and to print out the change in that variable's value. This is called setting a ***watchpoint***.
- `continue` – tells gdb to resume normal execution of the program from the line of code where it's currently stopped until the next breakpoint, watchpoint, etc.
- `backtrace` – prints a backtrace, which is the sequence of function calls (called ***frames***) that brought the program to the current line of code being executed.

When you're done debugging, explain the *process* you used to debug the program in the space provided below. If you were able to successfully debug the program, make sure to describe all of the bugs you found and how you fixed them.

[Describe your debugging process here.]

Step #7: Debug another program with GDB and Valgrind

Finally, In particular, try to use GDB and Valgrind to debug the program **`buggy-array-sort.c`**. This program generates a small array of random integers and tries to sort it using insertion sort. However, something in the program is broken, and the array is not correctly sorted. Again, try to use GDB and/or Valgrind to diagnose and fix the problem. When you're done debugging, explain the *process* you used to debug the program in the space provided below. If you were able to successfully debug the program, make sure to describe all of the bugs you found and how you fixed them.

[Describe your debugging process here.]

Submission and wrap-up

Once you've completed this exercise, download your completed worksheet as a PDF and submit that PDF on Canvas (refer to the image below to help find how to download a PDF from Google Docs). Again, please submit one copy of your completed

worksheet for your entire group. It's OK if just one member submits on Canvas. All group members will receive the same grade as long as their names and email addresses are included above.