CS 261 Recitation 3: GDB Practice

In order to get credit for the recitation, you need to be checked off by the end of recitation. For non-zero recitations, you can earn a maximum of 3 points for recitation work completed outside of recitation time, but you must finish this recitation before the next recitation. For extenuating circumstance, contact your Instructor.

**Group work** and **individual work** are highlighted

**Recitation 3 Grade Breakdown:**
- Part 1: GDB practice 10 pts
- Part 2: Continue working on Recitation 2

**Download and unzip the start code for this recitation:**
`wget https://classes.engr.oregonstate.edu/eecs/winter2024/cs261-020/recitations/rec3.zip`

**Part 1: GDB practice**

**Step 0 (Optional): GDB Setup**
If you prefer a more informational GDB interface (see below) with register values, source code, assembly code, stack information, etc., you may run the following script:

```
In your home directory, type:
python /nfs/farm/classes/eecs/spring2021/cs161-001/public_html/gdb/set_up.py
```
Answer ‘y’ to the question:

Once setup successfully, you will have a .gdb folder and a .gdbinit file under your home directory, and you can verify it with:

```bash
ls .gdb
cat .gdbinit
```

**Step 1: Demo**

Before you start working on using GDB yourself, your TAs will demonstrate how to use GDB to debug a buggy program, `buggy-pointers.c`. Watch carefully as they explain how to run the program with GDB and how to use GDB’s various commands, and follow along yourself in the source code as they debug.

To use GDB, you need to compile your program with a debugging \(-g\) flag:

```bash
gcc --std=c99 -g [some_program.c] -o [exe_name]
```

Then, run gdb with the executable:

```bash
gdb ./[exe_name]
```

Some important commands to watch for during the demo (links to command documentation are included):

- `run (r)` – starts your program from the beginning. Command line arguments to your program can be specified with the `run` command.

```bashun --args [args]
```
• **break** (b) – 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**.

  break [file_name]: [line_num]
  break [function_name]

• **list** (l) – prints out the lines of source code near the one currently being executed or near a specified location.

• **print** (p) – prints out the GDB value stored in a specified variable, etc.

  print [var_name or function_name]
  You may also print out the address of a specified variable.

  print &[var_name]

• **step** (s) – 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** (n) – 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.

  As you may notice, each statement may contain multiple assembly instructions. You may also run those assembly instructions one by one by “next instruction” or “ni”

• **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**.

  watch [var_name]

• **continue** (c) – tells GDB to resume normal execution of the program from the line of code where it’s currently stopped until the next breakpoint, or the end of the program.

• **backtrace** (bt) – prints a backtrace, which is the sequence of function calls (called **frames**) that brought the program to the current line of code being executed.

• **x/100wx** [address or register] – read memory

  x – Examine

  100 – 100 values

  w – sized as word (w, 4 bytes) / b – 1 byte / g – 8 bytes

  x – In hexadecimal (x) / d – decimal
(5 pts) Step 2: Use GDB to resolve the segmentation fault

Try to use GDB 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. If you were able to successfully debug the program, make sure to describe all of the bugs you found and how you fixed them in list_sort.txt.

(5 pts) Step 3: Use GDB to resolve the logic error

Try to use GDB 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. If you were able to successfully debug the program, make sure to describe all of the bugs you found and how you fixed them in array_sort.txt.

Part 2: Continue working on Recitation 2

If you were unable to complete recitation 2 last week, you may use the rest of the time to complete it : )

Make sure you get checked off by showing them the output of your program and your group work before the end of your recitation section.

For backup purposes, please submit your work for this recitation (including all documents/text files for group work, and programs) to TEACH.