CSE31 : Lab #2 – GDB & Pointers

This lab contains several parts. To ensure you get full credit, make sure you read this lab carefully and follow the instructions precisely.

Overview

This lab will require use of gdb to debug and fix the errors. We will also be checking your familiarity with gdb and the commands needed to do the debugging.

(Reference) Reading

P&H : 3.1-3.2
K&R : 5.1-5.5

(Exercise) GDB

Download gdb command reference handout from the assignment page. Answer the following questions:

Q1. How do you run a program in gdb?

Q2. How do you pass command line arguments to a program when using gdb?

Q3. How do you set a breakpoint in a program?

Q4. How do you set a breakpoint which only occurs when a set of conditions is true (some variables have a certain value)?

Q5. How do you execute the next line of C code in the program after a break?

Q6. If the next line is a function call, you'll execute the call in one step. How do you execute the C code, line by line, inside the function call?

Q7. How do you continue running the program after breaking?

Q8. How can you see the value of a variable (or even an expression) in gdb?

Q9. How do you configure gdb so it prints the value of a variable after every step?

Q10. How do you print a list of all variables and their values in the current function?

Q11. How do you exit out of gdb?

(Exercise) Debugging

Use the program given in appendTest.c for this exercise from the assignment page. You can compile and run the program using various inputs. You will notice after appending a few strings together that it does not always produce the correct output. You can exit the program using ctrl-C. Note that you need to compile it using –g flag to get the debugging information. You can start gdb in two ways :

1. In EMACS, type M-x gdb, then type gdb <filename>
2. Run gdb <filename> from the command line

Load appendTest into gdb and set a breakpoint in the append function before running it. When the debugger gets to the append function, step through each instruction line by line along with the values of the variables. The values of the pointers s1 and s2 are interest to us. Fix the error so append works correctly as intended. Hint: Think of how C represents strings.

Q12. What is the bug causing append to not work correctly?

(Exercise) Segmentation Faults

Recall what causes segmentation fault and bus errors from lecture. Common cause is an invalid pointer or address that is being dereferenced by the C program. Use the program average.c from the assignment page for this exercise. The program is intended to find the average of all the numbers inputted by the user. Currently, it has a bus error if the input exceeds one number.

Load average into gdb with all the appropriate information and run it. Gdb will trap on the segmentation fault and give you back the prompt. First find where the program execution ended by using backtrace (bt as shortcut) which will print out a stack trace. Find the exact line that caused the segmentation fault.

Q13. What line caused the segmentation fault?

Q14. How do you fix the line so it works properly?

You can recompile the code and run the program again. The program now reads all the input values but the average calculated is still incorrect. Use gdb to fix the program by looking at the output of read_values. To do this, either set a breakpoint using the line number or set a breakpoint in the read_values function. Then continue executing to the end of the function and view the values being returned. (To run until the end of the current function, use the finish command).

Q15. What is the bug? How do you fix it?

What to hand in

When you are done with this lab assignment, you are ready to submit your work. Make sure you have done the following ***before*** you press Submit:

- Answers to Q1-Q15.
- Attach fixed appendTest.c and average.c
- List of collaborators