Recitation 03

Debugging with gdb

Preface

Preparing for Recitation 03

Pulling the third assignment

- Inside your recitations repository in your VM
  - Run **git pull upstream master**
  - Make sure you have an r03 directory

Today's agenda

- We will cover in recitation
  - Debugging with gdb
- What you will do tonight
  - R03
    - Use gdb to fix a buggy program
    - Write a README.md file explaining how you used gdb

Getting started with GDB

How to use it and why you should

What is debugging?

- Just because your code compiles doesn't mean it does what you want
  - It could loop forever, crash, or otherwise just not work correctly
  - Writing tests helps you find out that your code doesn't work correctly, but you might need more help figuring out *why* your code doesn't work
- A debugger can help you by providing a number of helpful tools
  - In this class we will be using gdb, the **G**NU **deb**ugger
  - GDB lets you
    - Run your program
    - Stop your program at a certain point
    - Examine what your program is doing
    - Change things within your program to see if it helps

How do you use GDB?

- Add the -g flag when you compile with gcc
  - This flag tells GCC to include debugging information that GDB can use
- Run your program with GDB
  - Run `gdb ./your_program`
  - You will then be given an interactive shell where you can issue commands to GDB

Some common GDB commands

Short Name	Long Name	What do it do?
r	run	Begins executing the program – you can specify arguments after the word run
s	step	Runs the next line, going inside functions and running their code too
n	next	Runs the next line, counting called functions as a single line
p	print	Prints the value of an expression or variable
l	list	Prints out source code
q	quit	Exit GDB

Some more advanced GDB commands

Short Name	Long Name	What do it do?
b	break	Sets a breakpoint at a specified location (either a function name or line number)
c	continue	Continues executing after being stopped by a breakpoint
bt	backtrace	Prints out information on the callstack
f	frame	Prints information on the current frame / allows you to change frames
i	info	Prints out helpful information (try info args and info locals)

Debugging an infinite loop

- Set a breakpoint inside the loop
  - Or just run it and hit control-c
- List the code
  - This is so you can see the loop condition
- Step over the code
- Check the values involved in the loop condition
  - Are they changing the right way? Are the variables changing at all?

Debugging a crash

- Run your program
- Use bt to see the call stack
  - You can also use where to see where you were last running
- Use frame to go to where your code was last running
- Use list to see the code that ran
- Check the locals and args to see if they are bad