GDB Tutorial

CS2200

What is GDB?

- “GNU Debugger”
- A debugger for several languages including C
- Allows you to inspect what the program is doing at certain point of its execution
- Makes it easier to fix errors like segmentation faults
- http://sourceware.org/gdb/current/onlinedocs/gdb/ (online manual link)

Additional Steps when compiling

- Normally you would compile a program like

```
gcc [flags] <source files> -o <output file>
```

- Now you add a -g option to enable built in debugging support (which GDB needs)

```
gcc [other flags] -g <source files> -o <output file>
```

Starting up GDB

- Just try “gdb” or “gdb ./filename” or incase you have a file that takes in arguments “gdb ./filename --argument1 argument2” in the terminal. You’ll get a prompt that looks:

```
(gdb)
```

- If you did not specify a file name, do so now

```
(gdb) file prog1.x
```

- Note: prog1.x is the name of the executable file

Some Tips

- GDB has an interactive shell, much like the terminal, it can recall history with the arrow keys, auto complete works with the TAB key and some you can use the initials for most commands
- Use the help command if you want to know more about a particular function

```
(gdb) help [command]
```

Running the Program

- To run the program, just use:

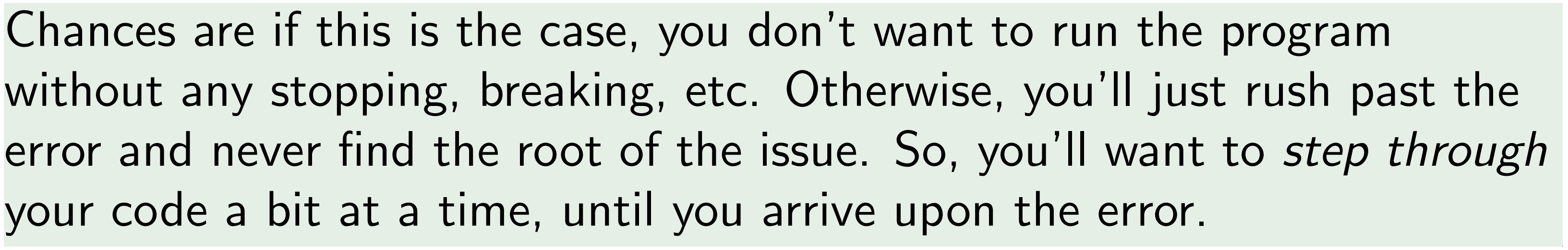
```
(gdb) run
```

- If there are no serious flaws (The program did not get seg faults, etc.), it should run fine too
- If there were some issues, you should get some useful information like

```
Program received signal SIGSEGV, Segmentation fault.
0x0000000000400524 in sum_array_region (arr=0x7fffc902a270, r1=2, c1=5,
r2=4, c2=6) at sum-array-region2.c:12
```

What if I have bugs?

- So you got the program working successfully, but what if it is not doing what you wanted it to do?



Chances are if this is the case, you don't want to run the program without any stopping, breaking, etc. Otherwise, you'll just rush past the error and never find the root of the issue. So, you'll want to *step through* your code a bit at a time, until you arrive upon the error.

- Let us look at the commands to set break points

Setting Breakpoints

- Breakpoints can be used to stop the program run in the middle, at a designated point. The simplest way is the command "break". This sets a break point at a specified file-line pair

```
(gdb) break file1.c:6
```

- This sets a breakpoint at line 6 of file1.c
- Tip: You can set as many breakpoints as you want in a file

Breakpoints with Functions

- You can also break at a particular function. Suppose you have a function my_func

```
int my_func(int a, char *b);
```

- You can break anytime this function is called:

```
(gdb) break my_func
```

- Tip: You can set conditional breakpoints like:

```
(gdb) break file1.c:6 if i >= ARRAYSIZE
```

“Continue” and “Step”

- Once the breakpoints have been set, you can use the “run” command to begin execution. This time it should stop where you told it to
- You can continue execution to the next breakpoint by using the “continue” command (using run again will start from the beginning)

```
(gdb) continue
```

- You can execute line-by-line by using the step command

```
(gdb) step
```

The “next” command

- Similar to step the next command single-steps, except this one treats each subroutine as a single line of code

```
(gdb) next
```

Tip

Typing “`step`” or “`next`” a lot of times can be tedious. If you just press ENTER, gdb will repeat the same command you just gave it. You can do this a bunch of times.

Printing values

- It is useful to be able to see the value of your variables when the code is being executed. You can do this by using the print command

```
(gdb) print my_var
```

- You can also view the hex value by using print/x:

```
(gdb) print/x my_var
```

Setting Watchpoints

- Similar to breakpoints, watchpoints also interrupt the flow of the program. However, they pause the program when the value of a watched variable changes.

```
(gdb) watch my_var
```

- Tip: If you have multiple variables with the same name, the printed value depends on the scope of the variable.

Extra Commands

- backtrace – Produces stack trace of function calls that lead to a seg fault
- finish – runs until the current function is finished
- delete – deletes a specific breakpoint
- info breakpoints – shows information about all breakpoints