CSCI 2461

Debugging with GDB and Valgrind

KATE HALUSHKA & JONATHAN LEE 2022, LINNEA DIERKSHEIDE 2021,
JOHN SHEPHERD 2018, PHIL LOPREIATO, NEEL SHAH 2014

What is GDB?

GNU Project debugger

Used for debugging your C programs

How to use GDB

- GDB is already installed in the SEAS shell
- In case you wanted GDB (and valgrind) on a linux based system:
  - sudo apt-get install gdb valgrind

How to use GDB

- When you compile a C program with gcc, you can add flags like -o
  - ex: gcc example.c -o example
  - To execute this, run ./example
- When you want to use gdb, add the -g flag
  - ex: gcc -g example.c -o example
  - To execute this with gdb, run gdb ./example

How to use GDB

- This will not immediately run your program, but that is OK!
- If you just want to run your program in gdb, you can now type run (r)
- However, you will usually be using gdb when something is broken and you don't want to just run straight through your program

GDB: Breakpoints

- Breakpoints are a way of telling C to stop the program at a certain point
- You can examine memory once the program is stopped at a point
- Check contents of variables at a point in your code
- Great for debugging!
- Similar to using Java visualizer but more powerful

GDB: Breakpoints

- Set breakpoint
  - break <line num>
- Set breakpoint on function
  - break <func name>
- List of breakpoints
  - info breakpoints
- Remove breakpoints
  - clear <breakpoint num> ← retrieved from info
- Skip breakpoints
  - ignore <breakpoint num> ← retrieved from info
- You can also just type b instead of break

GDB: Breakpoints

- What if I have more than 1 file in my program?
  - Use this format to make a breakpoint for a specific file:
  - b filename.ext:line
  - example: b hello_world.c:25

GDB: Examining Code

- So, we hit a breakpoint. Now what?
- We can "step" through, line by line, by typing "next" or "n"
  - Tip: you can repeat commands in GDB just by hitting enter
- You can continue until the next breakpoint or the end of the program by typing "continue" or "c"
- Use "step" to go into a function
  - Use "finish" or "fin" to jump to the end of the function you are currently in
- Use "backtrace" or "bt" to see the stack, especially useful for a seg fault!

GDB: Examining Code

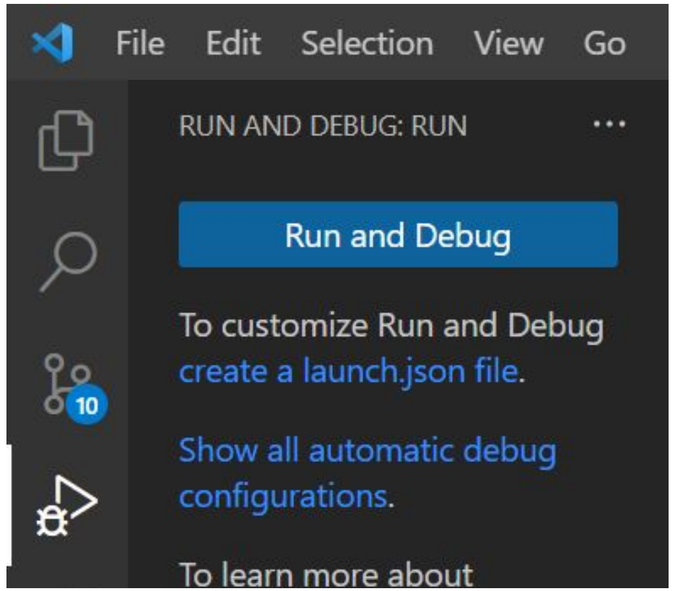
- You can also see the state of your data at this point in the program
- Use "print" or "p" and then the variable name you want to inspect
- You can also print all of your local variables with "info locals" or "i lo"
- Same with args, "info args" or "i args"

GDB: Common Commands

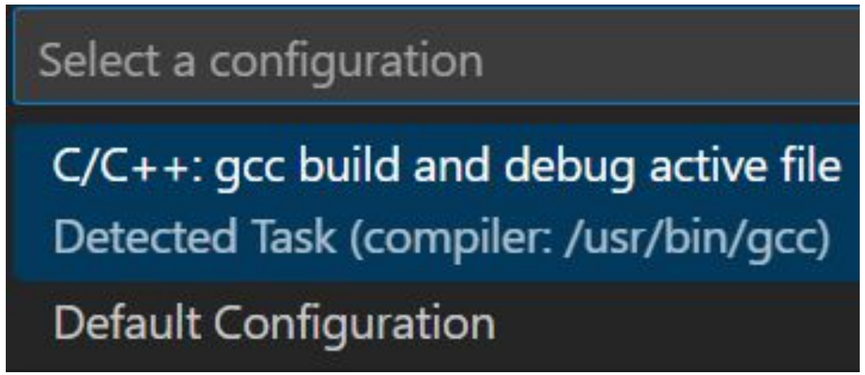
- run (r)
- break (b)
- next (n)
- continue (c)
- step (s)
- finish (fin)
- print (p)
- info locals (i lo)
- info args (i args)
- backtrace (bt)
- quit (q)
- There are many more! Just do a simple google search for gdb commands

For VS Code Users

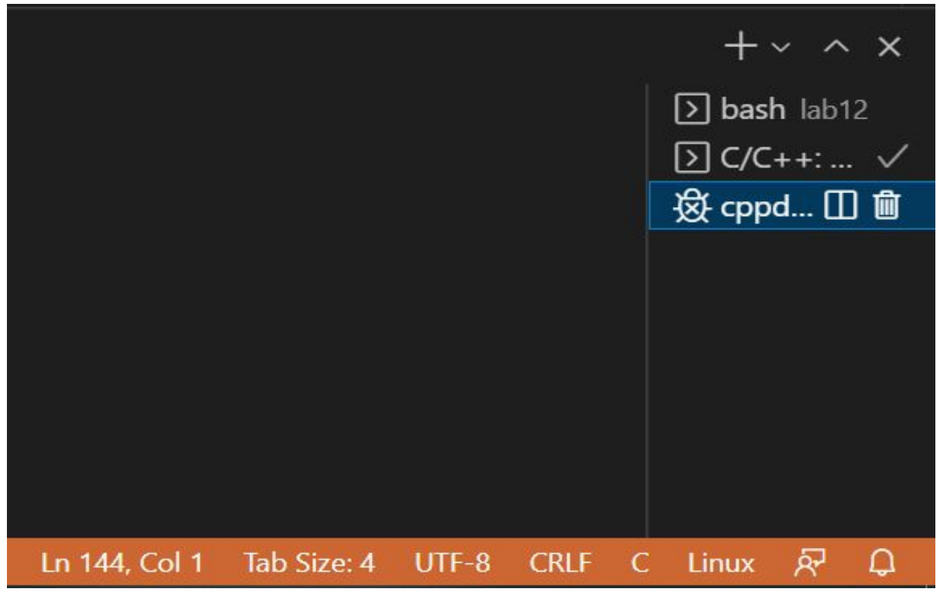
- Using GDB in Visual Studio Code (on shell.seas.gwu.edu).
- See the full setup instructions here:
- https://gw-cs2461-2022.github.io/tutorials/VSCode_GDB_Tutorial.pdf



1) Open debug panel



2) Debug active file



3) Select the debugging terminal

Using VS Code Debugger (GDB)

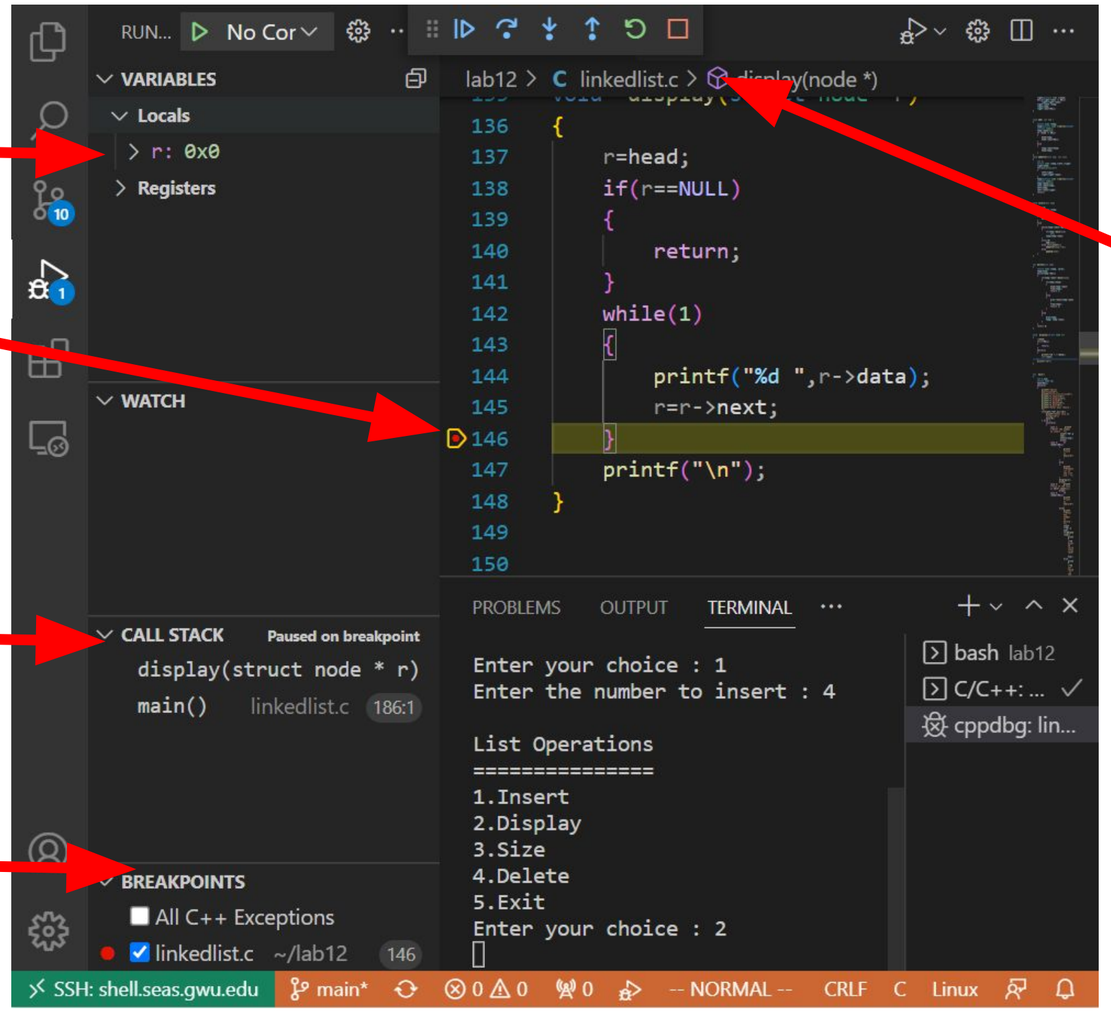
- Using the debugger:

Variable values

Set breakpoints

Call Stack

View breakpoints



- Continue
- Step Over
- Step Into
- Step Out
- Restart
- Stop

Valgrind

- C allows the programmer all the power
- Memory management is left up to you
- malloc (realloc, calloc, etc.) gives you memory for use
- It is left up to you to tell the computer you are no longer using this memory (free), unlike other languages such as Java
- Num_of_mallocs = num_of_frees
- Data malloc'd should likely be free'd at the end of its scope (more specifically, its lifespan)

Valgrind

- Manually - count mallocs and count frees, if equal then you likely have no problem
- Or allow Valgrind to do the work for you
- valgrind ./example
  - works basically the same as gdb
- If you have missed frees, use the sizes and number of blocks to deduce where the issues are

Valgrind

- Valgrind has some options (flags) that you can use to get more info. about memory leaks, errors, etc.
  - Ex. valgrind -flag ./example
- -v or --verbose, tells you more info
- -s, shows error list
- --leak-check=full, each leak will be detailed
- more online, sometimes it will give you suggestions of what to use!

Exercise

- Download linkedlist.c ("Lab 12 Files" zip)
- Can download exercise in shell.seas.gwu.edu

```
wget https://gw-cs2461-2022.github.io/lectures/lab12.zip
unzip lab12.zip
cd lab12
```

- Fix the errors and memory leak(s) using GDB and valgrind
- There are 5 bugs and 1 memory leak