Programmer's Guide to GDB

While working on labs in this course you will often find yourself facing the notorious `segfault`. Segmentation faults can be particularly annoying errors because their error messages do not give any insight into what caused the error. This is where powerful command line debuggers like `gdb` and `lldb` come in handy. For this course you will mostly use `gdb` on the classroom server, but `lldb` is the default on MacOS and is extremely similar to `gdb`. In fact, the most common `gdb` commands should seem familiar since most GUI debuggers uses these same commands via buttons.

To start a debugging session run >`gdb <filename>`

where `filename` is the name of your compiled executable file, for this assignment it is `tests`. Note that your executable must be compiled with the `-g` flag, but the `Makefile` has already done this for `tests`. Syntax for commands inside the bugger is as follows `n[ame]` which means the command can be executed by typing either `n` or `name`. Common commands to know are:

- `r[un] {args}` runs the executable with the given `{args}`
  - Note: If the code encounters a runtime error (i.e. `segfault`), it will stop executing as if it had reached a breakpoint. From here you can print variables and traverse the stack frames to pin down the cause of your error.
- `b[reak] <file.c>:line` sets a breakpoint in the source code file `file.c` on line `line`
- `p[rint] <expression>` prints the evaluated expression, this can be used to print array values or dereferenced pointers
- `n[ext] <n>` if the code has reached a breakpoint, this command will step over `n` lines of code
  - If no `n` is provided the default value is 1
- `c[ontinue] <n>` if the code has reached a breakpoint, this command will continue executing until `n` breakpoints are hit
  - For example, if you set a breakpoint in a loop an called `c` you would break in the next iteration of the loop
- `u[ntil]` will continue to execute until you reach the next line of code
  - For example, if you are on the last line of a loop then calling until would stop on the next line after the loop, completing all loop iterations
- `s[tep]` will step into the next line
- `finish` will step out of the current function
- `backtrace` will print the current frames on the function call stack
- `f[rame] <n>` will switch to frame number `n`
- `q[uit]` will quit the debugging session