The GNU Debugger (GDB)

The GNU Debugger is a powerful debugging tool that will be critical to Lab 2 and Lab 3 and is a useful tool to know as a programmer moving forward. There are tutorials and reference sheets available on the course webpage, but the following tutorial should get you started with the basics:

GDB Tutorial:

1) **Download calculator.c from the class webpage if you didn't already have it from Section 1:**
   
   ```
   > wget https://courses.cs.washington.edu/courses/cse351/20su/sections/01/code/calculator.c
   ```

2) **Compile the file with debugging symbols** (-g flag):
   
   ```
   > gcc -g -o calculator calculator.c
   ```

3) **Load the binary (executable) into GDB. This will spit out a bunch of information (e.g. version, license).**
   
   ```
   > gdb calculator
   ```

4) **Inside of GDB, use the run command (run or just r) to execute your program. By default, this will continue until an error or breakpoint is encountered or your program exits.**
   
   a. Command-line arguments can be passed as additional arguments to run:
      
      ```gdb run 3 4 +```
   
   b. To step through the program starting at `main()` instead, use the start command (start or just sta):
      
      ```gdb start```

5) **To view source code while debugging, use the list command (list or just l).**
   
   a. You can give list a function name ("list <function>") to look at the beginning of a function.
      
      ```gdb list main```
   
   b. You can give list a line number ("list <line>") to look at the lines around that line number, or provide a specific range ("list <start>, <end>").
      
      ```gdb list 45```
      ```gdb list 10, 15```
   
   c. "list" will display the next 10 lines of code after whatever was last displayed and "list --" will display the previous 10 lines of code before whatever was last displayed.

6) **To view assembly code while debugging, use the disassemble command (disassemble or just disas).**
   
   a. "disas" will display the disassembly of the current function that you are in.
   
   ```gdb disas main```
   ```gdb disas print_operation```

7) **Create breakpoints using the break command (break or b)**
   
   a. A breakpoint will stop program execution before the shown instruction has been executed!
   
   ```gdb break main```
   ```gdb break 34```
   ```gdb break *0x4005d5```
   
   b. You can create a breakpoint at a function name, source code line number, or assembly instruction address. The following all break at the same place:
      
      ```(gdb) break main```
      ```(gdb) break 34```
      ```(gdb) break *0x4005d5```
   
   c. Each break point has an associated number. You can view your breakpoints using the info command (info or just i) and then enable (enable or just en) or disable (disable or just dis) specific ones.
      
      ```gdb info break```
      ```gdb disable 3```
Navigating source code within GDB is done while program execution is started (**run** or **start**), but halted (e.g. at a breakpoint).

a. Use the **next** command to execute the next # of lines of **source** code and then break again. This will complete ("**step over**") any function calls found in the lines of code.

   (gdb) next
   (gdb) next 4

b. Use the **step** command to execute the next # of lines of **source** code and then break again. This will step **into** any function calls found in the lines of code.

   (gdb) step
   (gdb) step 4

c. Use the "**next instruction**" command to execute the next # of lines of **assembly** code and then break again. This will complete ("**step over**") any function calls.

   (gdb) nexti
   (gdb) nexti 4

d. Use the "**step instruction**" command to execute the next # of lines of **assembly** code and then break again. This will step **into** any function calls.

   (gdb) stepi
   (gdb) stepi 4

e. Use the finish command to step **out** of the current function call.

f. Use the continue command (**continue** or just **c**) to resume continuous program execution (until next breakpoint is reached or your program terminates).

You can print the current value of variables or expressions using the print command (**print** or just **p**):

a. The print command can take an optional format specifier: /x (hex), /d (decimal), /u (unsigned), /t (binary), /c (char), /f (float)

   (gdb) print /t argc
   (gdb) print /x argv
   (gdb) print /d argc*2+5
   (gdb) print /x $rax

b. The display command (**display** or just **disp**) is similar, but causes the expression to print in the specified format **every time** the program stops.

You can terminate the current program run using the kill command (**kill** or just **k**). This will allow you to restart execution (run or start) with your breakpoints intact.

You can exit GDB by either typing **Ctrl-D** or using the quit command (**quit** or just **q**)