LAB 3 : GDB

GDB(or the GNU Debugger) is a command-line debugger. Interaction is via command line using text-based commands. Using GDB, we can set breakpoints, step through the code, and see variable values and so on.

Basic gdb commands:

Note: There isn't a GDB build for M1 Mac. You can use the lldb commands instead. In places where the lldb command isn't mentioned, use the gdb command(it means it has the same syntax as the gdb command). Refer to the gdb to lldb command map link for further information.

1. **Running gdb**

   gdb **<name of the executable for your program>**

   lldb **<name of the executable>**

   Note: This is not the .cpp or the .o file, instead it is the name of the compiled program. Example: From lab1, if you did g++ hello.cpp -o myExec, to run it in debugging mode, the command will be gdb **myExec.**

   You can see a gdb prompt like this **(gdb)** on the terminal. This is where we type the gdb commands. The gdb is awaiting these commands/instructions and the program has not started running yet.

   Task: Create an executable for primes.cpp program and run gdb.

2. **Running the program -** run (shorthand: r)

   After step 1, you are in the (gdb) prompt. If the program you are debugging requires any command-line arguments, you specify them to the run command.

   (gdb) run arg1 arg2 …

   Task: Run the program using the above command. primes.cpp takes one command line argument n to generate prime numbers till n

3. **Creating and Deleting breakpoints -** break (shorthand: b)

   These are used to pause the program's execution wherever you want to examine the program state at that point. These get numbered sequentially.
   To set a breakpoint at the beginning of a function:

   (gdb) break <name of the function>

   To set a breakpoint at line **x** in **hello.cpp**:

   (gdb) break hello.cpp:x

   To keep track of breakpoint numbers:

   (gdb) info break (lldb) breakpoint list or br l

To delete a breakpoint numbered **n** or to clear a breakpoint at line number x:

(gdb) delete **n** (shorthand: d) (lldb) br del **n**

(gdb) clear **x**

4. **Info** - Provides information about breakpoints, local variables, parameters of a function

   (gdb) info breakpoints or (lldb) breakpoint list or br l

   (gdb) info locals or (lldb) frame v

   (gdb) info args

   Task: Create a breakpoint at line number 15 and at the main function. Explore the above info commands to check what gets displayed. Clear/Delete these breakpoints

5. **List -** list (shorthand: l)

   To list/display the program code block either from the beginning, or with a range around where you are currently stopped.

   (gdb) list **<function_name>**

6. **Print value of variables -** print (shorthand: p)

   (gdb) print variablename

   Note: If you see the error "No symbol xxxx in current context", or the error, "optimized out", you probably aren't in a place in the program where you can read the variable.
   Some Advanced Usage Tips: p/x variablename can help you print the data in hexadecimal format. If you want to check data over, say, 16 bytes of memory, you can use p/16xb. Data will be in reverse order.

7. **Next** - next (shorthand: n)

   The next command steps to the next program line and *completely runs functions*.
   Note: If you have a function and you use next, the function will run to completion, and gdb execution will stop at the next line after the function call (unless there is a breakpoint inside the function).

8. **Step -** step (shorthand: s)

   This is similar to next, but it will step into functions. This means that it will attempt to go to the first line in a function if there is a function called on the current line.

9. **Finish -** finish

   Runs a function completely. It is helpful if you accidentally step into a function. **Note:** If you do accidentally step into a function, you can use the finish command to finish the function immediately and go back to the next line after the function.

Tasks:

A. Run the program to display prime numbers till 10 using the executable you created. What error do you notice?

B. List the contents of main and primeList functions

C. Create a breakpoint at the main function. Display the breakpoint information. Delete the breakpoint.

D. Run the program in debug mode(Refer **Task 1 and 2** at the beginning of assignment). Use Ctrl+C to be able to enter the next command.

E. Create a breakpoint at line number 21.

F. Use the next command until you arrive at line number 25. If you enter into any of the inbuilt functions, use the command finish to get out of it.

G. Use the step command when you are at line 25. Which function does this enter into?

H. Step into the isPrime function in your code. Try to figure out what is giving the error you noticed in Part(A). You can create breakpoints, print the variable values using the print command and so on. Make necessary changes in code.

I. Get the code running! :)

Something Useful!

Backtrace Command: gdb has the ability to give you a backtrace (or a "stack trace") of your program's execution at any given point. This is useful for locating segmentation faults. If a program named **myProgram** segfaults while running in GDB during execution of a function named **myFunc**, gdb will print the following information:

Program received signal SIGSEGV, Segmentation fault.0x0000000000400ac1 in **myFunc** (fp=fp@entry=0x603010, nread=nread@entry=0) at **myProgram.cpp:51** 51 if (strlen(unusedptr) == MAX_FRAG_LEN)

This backtrace command to get a full stack trace of the program's execution to understand when the error occurred:

```
(gdb) backtrace
```

```
#0  0x0000000000400ac1 in myFunc (fp=fp@entry=0x603010, nread=nread@entry=0) at myProgram.cpp:51

#1  0x0000000000400bd7 in myFunc2 (fp=fp@entry=0x603010, arr=arr@entry=0x7fffffff4cb0, maxfrags=maxfrags@entry=5000) at myProgram.cpp:66

#2  0x00000000004010ed in main (argc=<optimized out>, argv=<optimized out>) at myProgram.cpp:211
```

Finally, you see that the error occurred from the code in myProgram.cpp, on line 51. All of this is helpful information to go on if you're trying to debug the segfault.