CS240
Lab 7 Assignment
Introduction to the GDB Debugger

Inspect and run a small C program on a Linux machine. NOTE: Do not run your C programs directly on your own Mac unless you are using *wx* (virtual machine) or if you are logged in remotely using ***ssh***.

Create the following file *prime.c* with emacs:

```
/* CS 240 program to check if a number is prime */
#include <stdio.h>

int test_prime(int num) {
  int i;
  int prime=1; //assume prime initially

  for (i=2; i<=num/2; ++i) {
    if (num%i == 0) {
      prime = 0; // set to not prime
      break;
    }
  }

  if (prime) {
    printf("%d is prime\n", num);
  } else {
    printf("%d is not prime\n", num);
  }
  return 0;
}

int main() {
  int test1 = 5;
  int test2 = 12;
  test_prime(test1);
  test_prime(test2);
  return 0;
}
```

2. In order to run programs under gdb, they should be compiled with debugging symbols turned on (-g option). Usually we provide a Makefile with recipes to compile, but it's useful to learn some common compiler options directly. To compile an executable called *prime* from the C code in *prime.c*, with **all** **w**arnings and debu**g**ging symbols enabled, using the 19**99 C** language **st**an**d**ard and **64**-bit code, run:

    $ ***gcc -Wall --std=c99 -m64 -g -o prime prime.c***

3. Run the program:

    $ ***./prime***

   and you should see the following output:

```
5 is prime
12 is not  prime
```

4. Now, run the program under gdb:

    $ ***gdb prime***

```
GNU gdb (GDB) Red Hat Enterprise Linux (7.2-90.el6)
```

Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/cs240/test/prime...done.

You can enter gdb commands commands at the prompt to perform various actions. Observe and verify the output as indicated:

(gdb) ***run***

Starting program: /home/yourname/gdb-example
5 is prime
12 is not prime
Program exited with code 021.

(gdb) ***quit***

So, ***run*** is used to execute the program, and ***quit*** is used to exit gdb.

5. The gdb debugger also allows you to walk through the program while it is running so that you can trace its steps carefully. Start another gdb session:

$ ***gdb prime***

The ***break*** command sets a breakpoint (an address/instruction in the program where gdb should pause execution). Breakpoints can be set at the beginning of a function or at specific lines in program file. There are many things that can be done with breakpoints, such as making them conditional or temporary.

6. Set a breakpoint so the program pauses at the beginning of the main function:

(gdb) ***break main***

Breakpoint 1 at 0x400554: file prime.c, line 24.

7. Run the program, and observe that it pauses execution at the breakpoint:

(gdb) ***run***

Starting program: /home/jherbst/prime
Breakpoint 1, main () at prime.c:24
24 int test1 = 5;

The highlighted line above is the next statement to be executed when the program is resumed (the first statement in the *main()* function).

8. The ***print*** command displays the value of variables or expressions within the scope of the current frame. So, since *test1* is declared in main, you can print its value at this point:

(gdb) ***print test1***

$1 = some value

The ***$1*** represents the variable. The current value is not meaningful because the statement initializing the value to 5 has not yet been executed.

9. Execute a single statement by doing a step:

(gdb) ***step***

25 int test2 = 12;

10. Now display *test1* again:

(gdb) ***print test1***

$2 = 5

11. Try displaying a variable outside the current *frame/scope* (***num*** is a local variable inside the function ***test_prime()***, so it is not understood the function:

(gdb) ***print num***

No symbol "num" in current context.

12. Execute another statement:

(gdb) ***step***

26 test_prime(test1);

The ***step*** and ***next*** commands are both used to make gdb move forward in the program. For statements that do not involve functions, the two commands are identical and merely make gdb execute one statement. For statements that involve a function, however, the two commands are different: ***next*** tells gdb to execute the entire function, while ***step*** tells gdb to move inside the function.

13. Entering ***next*** at this point should execute the entire function ***test_prime(test1)***:

(gdb) ***next***

5 is prime
27 test_prime(test2);

14. Start to step through the second invocation of ***test_prime()***:

(gdb) ***step***

test_prime (num=12) at prime.c:6
6 int prime=1; //assume prime initially

15. Now that you are within the ***test_prime()*** function, you can also change the current context with the ***up*** or ***down*** commands (this doesn't change the point at which you are executing the program, but instead allows you to display values defined within a different context or frame):

(gdb) ***up***

#1 0x0000000000400576 in main () at prime.c:27
27 test_prime(test2);

16. Use the ***info*** command to display information about the current frame:

(gdb) ***info locals***

test1 = 5
test2 = 12

17. Go back ***down*** to the ***test_prime()*** frame, and display ***info*** about the args (arguments) in the current frame:

(gdb) ***down***

#0 test_prime (num=12) at gdb-example.c:6
6 int prime=1; //assume prime initially

(gdb) ***info args***

num=12

18. Another convenience provided by gdb is to ***list*** a small segment of the code around where the program is currently stopped so you can see which statements have been executed and which ones are about to be:

(gdb) ***list***

```
1       /* CS 240 program to check if a number is prime */
2       #include <stdio.h>
3
4       int test_prime(int num) {
5         int i;
6         int prime=1; //assume prime initially
7
8         for (i=2; i<=num/2;++i) {
```

```
9           if (num%i == 0) {
10            prime = 0; // set to not prime
```

19. To finish the program, enter ***cont*** to continue execution to the end, and then ***quit*** to exit gdb:

(gdb) ***cont***

```
Continuing.
12 is not  prime
Program exited with code 021.
```

(gdb) ***quit***

NOTE: most of the commands in gdb can be shortened to a single letter (as long as the shortened version can uniquely select the desired command).