Exercise 03 : Using GDB

On our course homepage, you'll find several small programs you can use to help you get familiar with gdb. You can download them via the course page. Or, if you're already working on a common platform system, you can use the following curl commands to copy these files to your local directory. You're getting three of these programs as source files, and, for one, you're just getting a pre-compiled executable (secret) that should be able to run on a common platform machine.

```
mkdir some-place-to-work
cd some-place-to-work
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise03/buggy.c
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise03/loopy.c
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise03/secret
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise03/jumpy.c
```

When you compile these programs, remember to use the -g option to the compiler, to tell it to include debugging information in your executable. So, you might compile the first program like:

```
gcc -Wall -std=c99 -g buggy.c -o buggy
```

For this exercise, you just have to answer some questions based on debugging these programs. There's a Moodle Quiz you'll fill out to answer the questions (so, this exercise looks like a quiz in Moodle, but it's classified as an exercise in the gradebook).

Run each of these programs in gdb to answer the following:

1. Compile the **buggy.c** program and run it inside gdb. It should crash when you run it, but gdb will let you examine the state of the program when it crashed. Look at a stack trace. What function was the program running in when it crashed?

2. Compile the **loopy.c** program and run it inside gdb. Let it run for a second, and it will get stuck in an infinite loop. Stop the program and examine its state. You can see it is running in a function called h( ). The function h( ) gets called several times during execution. Look at a backtrace and report what parameter value is passed to h( ) that causes it to get stuck in an infinite loop.

3. There's a program called **secret.c.** I'm not giving you its source code, just a pre-built executable. You can still run the program and debug it, but you'll have to run it on one of the common platform systems (that's what it's compiled for). It should run for you on one of the Linux systems in a public lab, or you can remotely log in on remote.eos.ncsu.edu.

   Before you can run it, you'll need to mark it as executable with the following command:

   ```
   chmod +x secret
   ```

   You don't get the source code for the **secret.c** program, but I will show you one of the functions:

   ```
   int loop() {
      int j = 0;
      int i = 0;
      while ( i == 0 ) {
         j++;
      }

      return j;
   }
   ```

   The secret program prints out a message (a secret color), but first, it calls the loop( ) function shown above. You can see, this is a bad function; it gets stuck in an infinite loop and never returns.

Run the **secret** program in gdb, wait a moment to give it a chance to get stuck in this loop, then interrupt its execution (with ctrl-C) while it's in the loop. With gdb, you can change variable values as a program is running. Use this to change the value of variable **i** so that the loop will terminate. Resume execution and the program should print out the secret color then terminate. What color it?

4. When you run the **jumpy.c** on a common platform system, it should print out the following output:

```
I don't like 83
I like 77
I like 93
I like 86
I don't like 49
```

   Each output line is printed by a randomly selected function, either edith(), tyler(), melonie() or aaron(). By just looking at the source code and the output, you can tell some things about who might have printed each line. For example, the first line of output could have been printed by melonie(), tyler() or edith(). Of course, tracing this program in gdb will make it easy to figure out exactly who printed each line. You can do this by either putting a breakpoint at each of the four functions and then seeing which one of them is called, or you can just put a breakpoint in main and use the step and next commands to watch what happens as main executes.

   Figure out which function prints each of the lines and fill that answer in the quiz.

There's nothing to submit for this exercise. When you complete the Moodle quiz for exercise 03, that will be your way of showing that you did the exercise.