Use of GNU Debugger (GDB) for Reverse Engineering of C Programs in a Linux Platform

Instructor: Dr. Natarajan Meghanathan, Jackson State University

Demo and Hardcopy Submission: You need to show a demo of the working of Phase 2 and 3 of the project to me before the above deadline and submit a hardcopy of the report (for all the three phases) in class at the above date/time.

The purpose of the project is to familiarize students with the GNU Debugger (GDB) and its use for reverse engineering. This will be done in Ubuntu OS with the use of simple C programs. If you need to set up Ubuntu as a virtual machine, follow the instructions provided for the same at http://143.132.8.23/cms/tues/docs/CSC439-AIS-Spring2013/Project-2-Email-Security-PGP.pdf

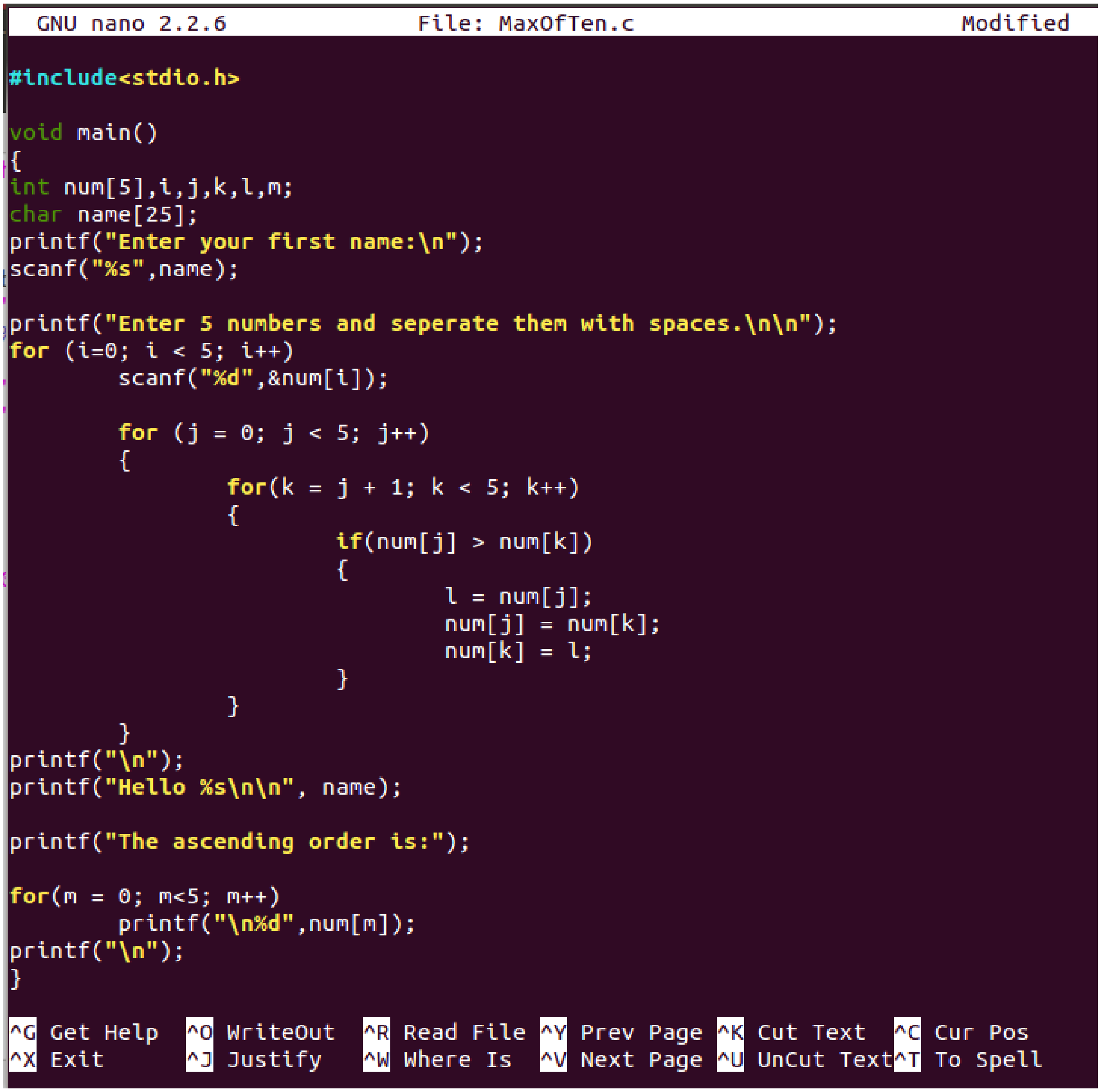
This project is divided into three phases. The first phase will introduce you to the GNU debugger (GDB) and the next two phases will require you to explore things built on top of the basics that you learn in Phase 1.

Phase 1: Introduction to the GNU Debugger (GDB)
25 points

What to submit:

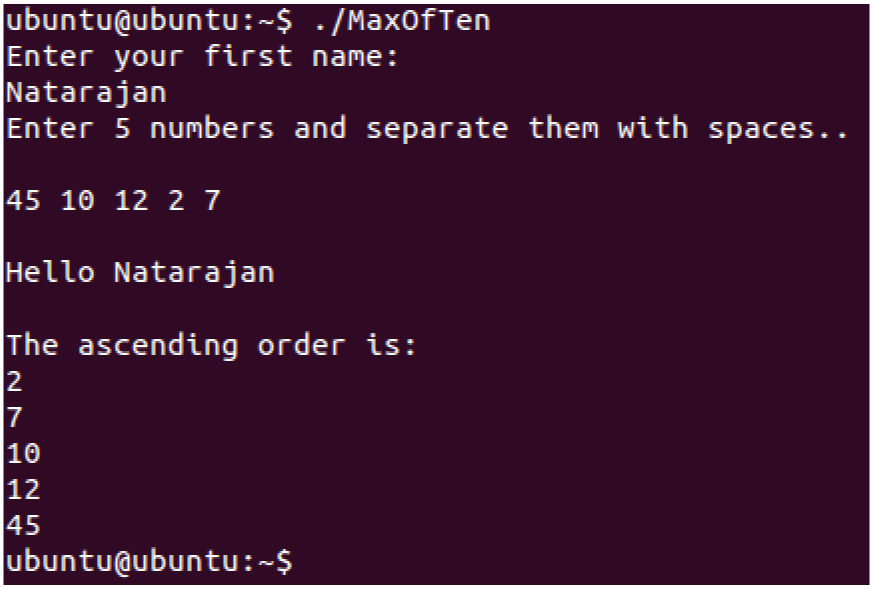
Screenshots of **ALL** the practice steps. Indicate the Step # in your screenshots.

1. Open a terminal and create the following C program **MaxOfTen.c** using an editor of your choice. I recommend using pico or vi. Type in the following code, save it and exit from the editor.



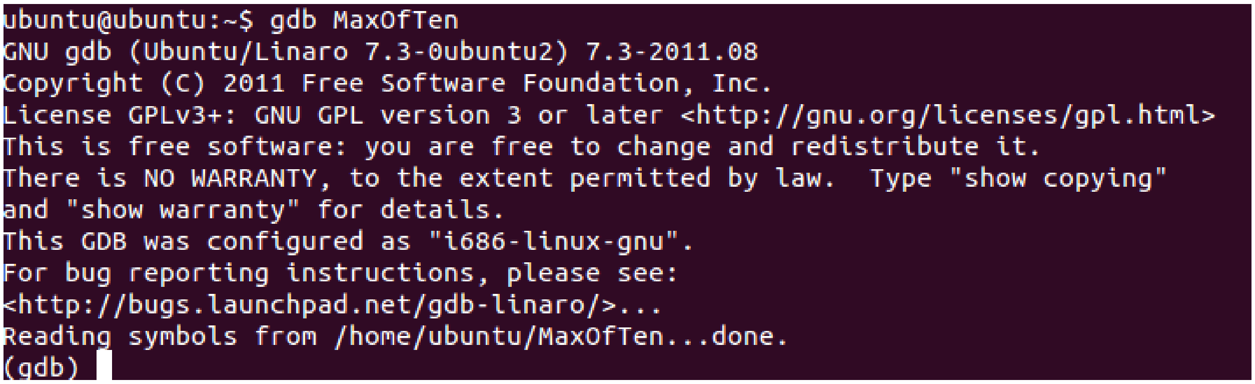
```c
#include<stdio.h>

void main()
{
int num[5],i,j,k,l,m;
char name[25];
printf("Enter your first name:\n");
scanf("%s",name);

printf("Enter 5 numbers and seperate them with spaces.\n\n");
for (i=0; i < 5; i++)
        scanf("%d",&num[i]);

        for (j = 0; j < 5; j++)
        {
                for(k = j + 1; k < 5; k++)
                {
                        if(num[j] > num[k])
                        {
                                l = num[j];
                                num[j] = num[k];
                                num[k] = l;
                        }
                }
        }
printf("\n");
printf("Hello %s\n\n", name);

printf("The ascending order is:");

for(m = 0; m<5; m++)
        printf("\n%d",num[m]);
printf("\n");
}
```

2. Now, compile the program using **gcc -g -o MaxOfTen MaxOfTen.c**. After compiling, run the program using **./MaxOfTen** The program will require you to enter your first name. Do so and press **enter**. It will then require you to enter 5 integers in the terminal. Separate the integers with spaces. After typing in the integers, press **enter** and the program will sort the integers in ascending order and print the result to the screen.



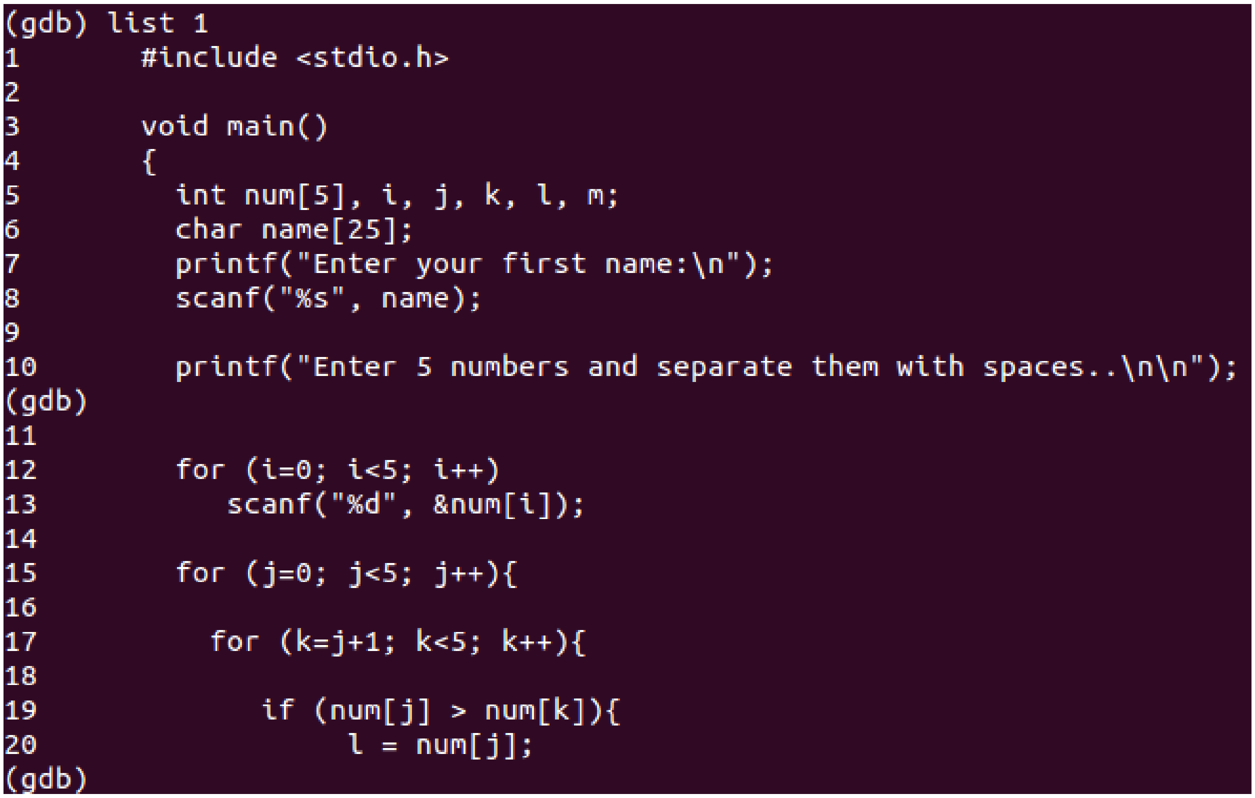
```
ubuntu@ubuntu:~$ ./MaxOfTen
Enter your first name:
Natarajan
Enter 5 numbers and separate them with spaces..

45 10 12 2 7

Hello Natarajan

The ascending order is:
2
7
10
12
45
ubuntu@ubuntu:~$
```

3. Next, we will load the program into gdb so we can analyze through our code.
Load the program using **gdb MaxOfTen**



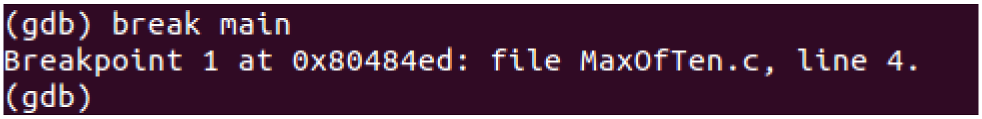
```
ubuntu@ubuntu:~$ gdb MaxOfTen
GNU gdb (Ubuntu/Linaro 7.3-0ubuntu2) 7.3-2011.08
Copyright (C) 2011 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /home/ubuntu/MaxOfTen...done.
(gdb) 
```

4. Type in **list 1** and press **enter** to list the program. If all the lines do not show up at once, you can keep pressing the **enter** key to list all the code lines.



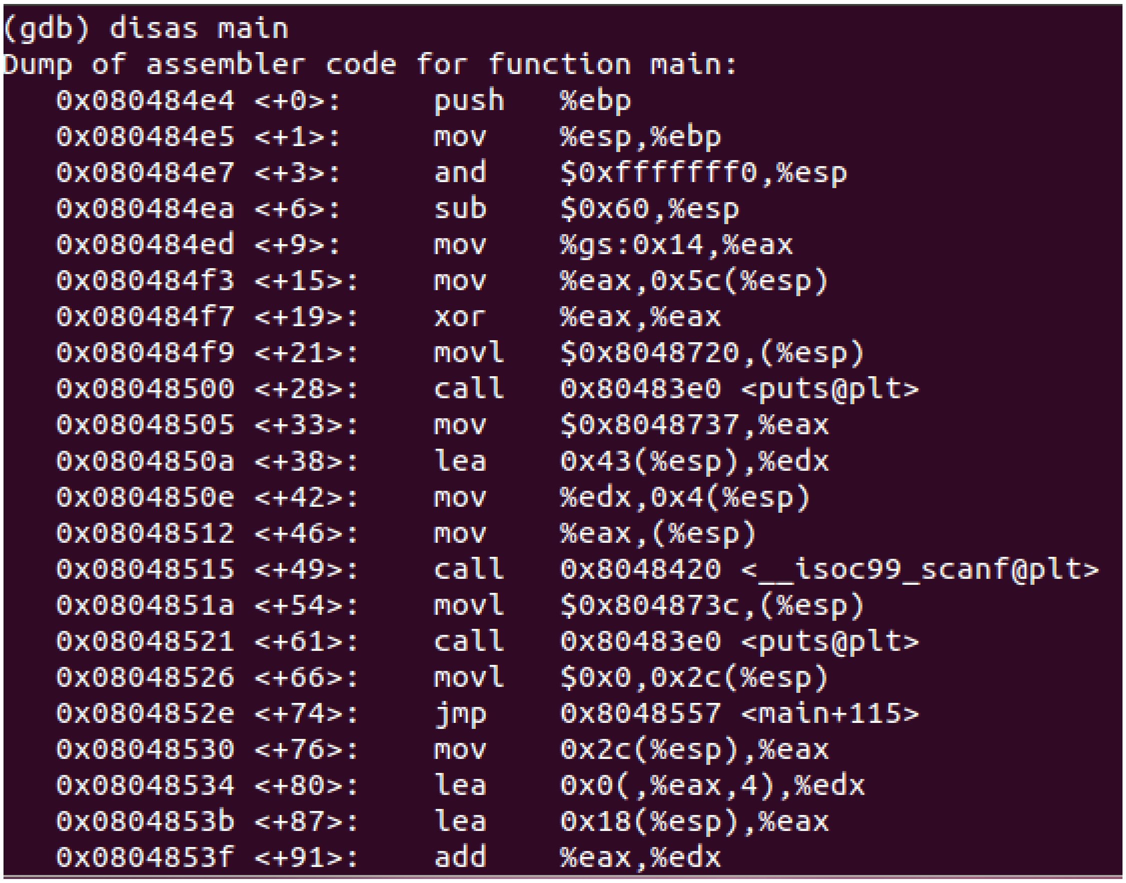
```
(gdb) list 1
1       #include <stdio.h>
2
3       void main()
4       {
5         int num[5], i, j, k, l, m;
6         char name[25];
7         printf("Enter your first name:\n");
8         scanf("%s", name);
9
10        printf("Enter 5 numbers and separate them with spaces..\n\n");
(gdb)
11
12        for (i=0; i<5; i++)
13           scanf("%d", &num[i]);
14
15        for (j=0; j<5; j++){
16
17          for (k=j+1; k<5; k++){
18
19             if (num[j] > num[k]){
20                  l = num[j];
(gdb)
```

5. Set a breakpoint at main so that we can be able to step through the program execution. To set a breakpoint, type the command **break main** and hit **enter**.



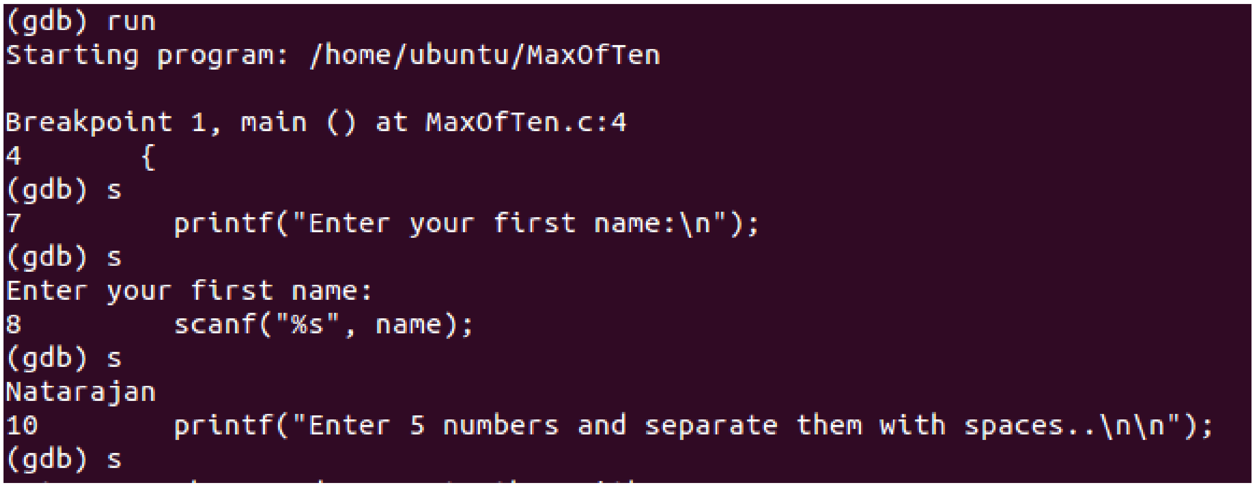
```
(gdb) break main
Breakpoint 1 at 0x80484ed: file MaxOfTen.c, line 4.
(gdb)
```

6. Disassemble the main function to get the assembly code equivalence. We are not going to do any assembly programming, this is just to take a peek into what's going on in the CPU. Use **disas main** to do this. Keep pressing the enter key until it reaches the end of assembler dump.



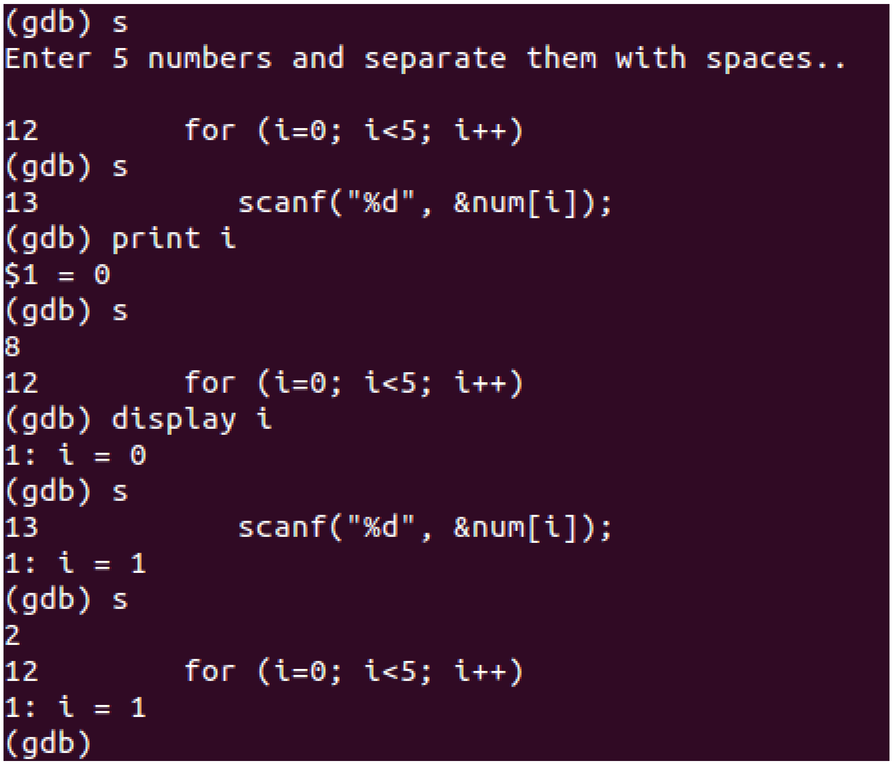
```
(gdb) disas main
Dump of assembler code for function main:
   0x080484e4 <+0>:     push   %ebp
   0x080484e5 <+1>:     mov    %esp,%ebp
   0x080484e7 <+3>:     and    $0xfffffff0,%esp
   0x080484ea <+6>:     sub    $0x60,%esp
   0x080484ed <+9>:     mov    %gs:0x14,%eax
   0x080484f3 <+15>:    mov    %eax,0x5c(%esp)
   0x080484f7 <+19>:    xor    %eax,%eax
   0x080484f9 <+21>:    movl   $0x8048720,(%esp)
   0x08048500 <+28>:    call   0x80483e0 <puts@plt>
   0x08048505 <+33>:    mov    $0x8048737,%eax
   0x0804850a <+38>:    lea    0x43(%esp),%edx
   0x0804850e <+42>:    mov    %edx,0x4(%esp)
   0x08048512 <+46>:    mov    %eax,(%esp)
   0x08048515 <+49>:    call   0x8048420 <__isoc99_scanf@plt>
   0x0804851a <+54>:    movl   $0x804873c,(%esp)
   0x08048521 <+61>:    call   0x80483e0 <puts@plt>
   0x08048526 <+66>:    movl   $0x0,0x2c(%esp)
   0x0804852e <+74>:    jmp    0x8048557 <main+115>
   0x08048530 <+76>:    mov    0x2c(%esp),%eax
   0x08048534 <+80>:    lea    0x0(,%eax,4),%edx
   0x0804853b <+87>:    lea    0x18(%esp),%eax
   0x0804853f <+91>:    add    %eax,%edx
```

7. Enter the **run** command to begin executing the program. Notice that the program will break at main. It shows the breakpoint at main, and below it is the line of code that will be executed next. Press **s** 3 times on your keyboard to step through the program. After pressing the third s, the pointer is going to hold and require you to type in your first name. Press **enter** after keying in your first name. The next instruction shown will print the string that requires you to enter the integers you are going to work with. Press **s** to step through this.



```
(gdb) run
Starting program: /home/ubuntu/MaxOfTen

Breakpoint 1, main () at MaxOfTen.c:4
4       {
(gdb) s
7         printf("Enter your first name:\n");
(gdb) s
Enter your first name:
8         scanf("%s", name);
(gdb) s
Natarajan
10        printf("Enter 5 numbers and separate them with spaces..\n\n");
(gdb) s
```

8. Press **s** again to continue stepping through the program. Here, we will monitor the variable **i** to see how it increments as the for loop is executed. Step again through the program by typing **s**, and then type in **print i** to see the beginning value of **i** which is 0.



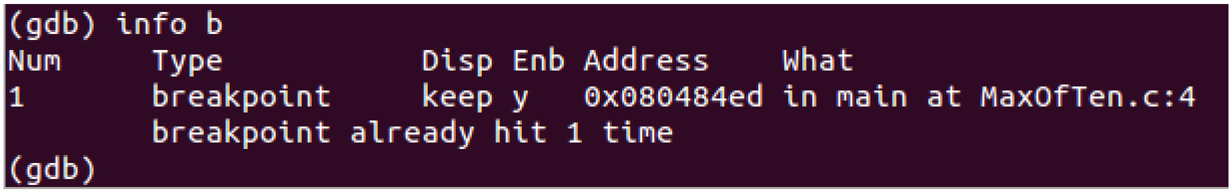
```
(gdb) s
Enter 5 numbers and separate them with spaces..

12        for (i=0; i<5; i++)
(gdb) s
13            scanf("%d", &num[i]);
(gdb) print i
$1 = 0
(gdb) s
8
12        for (i=0; i<5; i++)
(gdb) display i
1: i = 0
(gdb) s
13            scanf("%d", &num[i]);
1: i = 1
(gdb) s
2
12        for (i=0; i<5; i++)
1: i = 1
(gdb)
```

Press **s** again. The pointer is going to wait for you to type in an integer. Type in your first integer and press **enter**.

Now, you will have to show the value of i after every iteration of the for loop. You can do this using the **display i** command. This way, the value of **i** will be displayed after every iteration without you having to type the print i command.

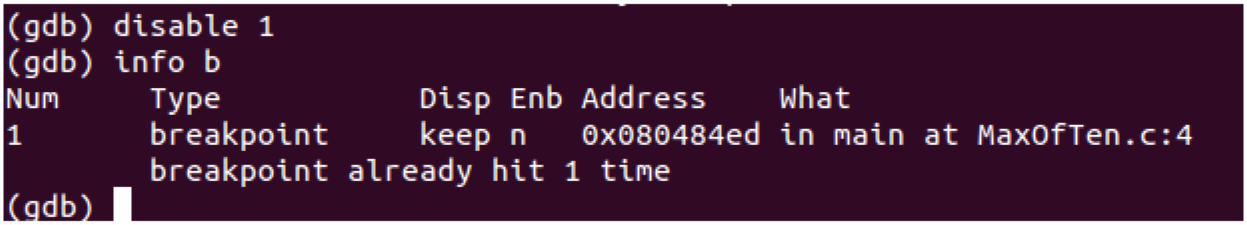
9. Step through the program using **s** and notice that the value of **i** will be printed each time you step. Keep stepping through the program and entering the integers until it reads the 5th one. Remember to take the screenshot of each step. After entering the 5th integer, press **s** and the program will move to the next for loop because the value of **i** is now 5, and this causes the program to finish the first loop and set up the next loop for execution. Take screenshot covering all i values.

10. Type **info b** to print information about existing break points.



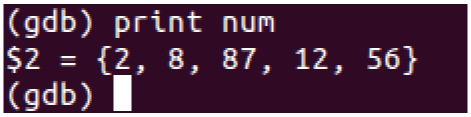
```
(gdb) info b
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x080484ed in main at MaxOfTen.c:4
        breakpoint already hit 1 time
(gdb)
```

11. Disable the breakpoint using the command **disable 1**. The number 1 in the command represents the breakpoint number because you can set more than one breakpoint. Print the breakpoint information again and notice that the character under **Enb** has changed from **y** to **n**.



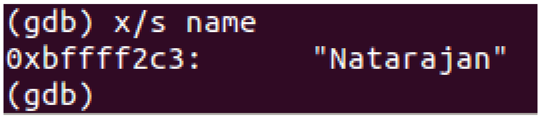
```
(gdb) disable 1
(gdb) info b
Num     Type           Disp Enb Address    What
1       breakpoint     keep n   0x080484ed in main at MaxOfTen.c:4
        breakpoint already hit 1 time
(gdb) 
```

12. Show the content of the array variable **num** using **print num**.



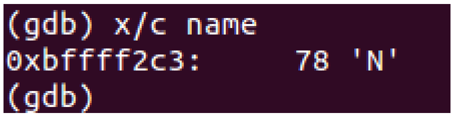
```
(gdb) print num
$2 = {2, 8, 87, 12, 56}
(gdb) 
```

13. Examine the name variable as a string using **x/s name**. This command will print your first name in string format. The 4-byte hexadecimal value to the left is the virtual memory address where the string is stored.



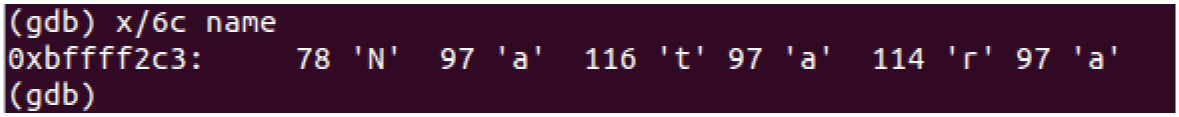
```
(gdb) x/s name
0xbffff2c3:      "Natarajan"
(gdb)
```

14. Examine a character using **x/c name**. This prints the first character in the name variable.



```
(gdb) x/c name
0xbffff2c3:     78 'N'
(gdb)
```

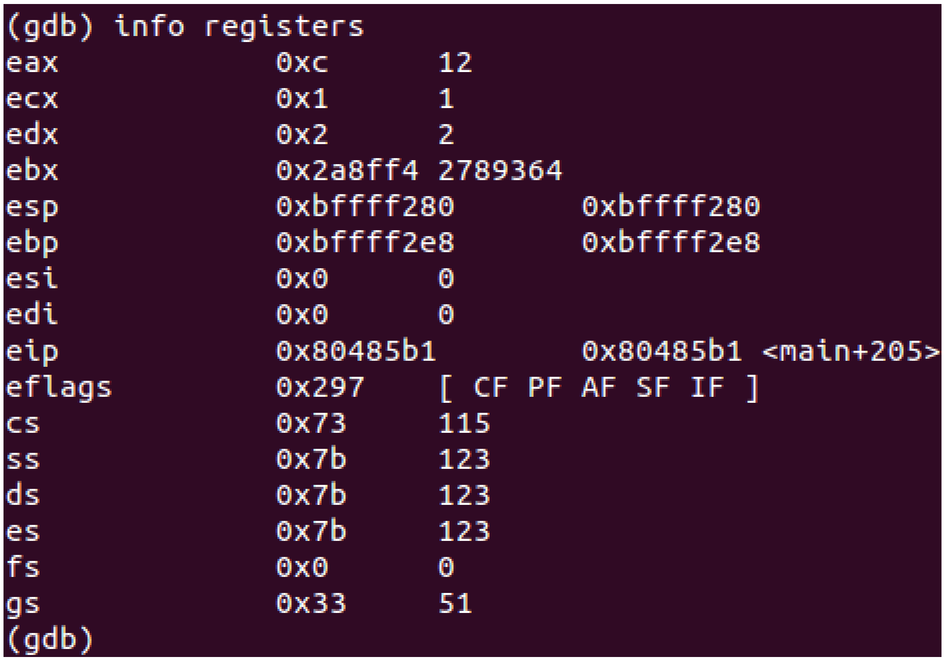
Increase the number of characters you want to print using **x/nc name** where **n** is an integer that represents how many characters you want to print. In the screenshot below, the integer 6 was used and this printed out 6 characters and their ASCII values.



```
(gdb) x/6c name
0xbffff2c3:     78 'N'  97 'a'  116 't' 97 'a'  114 'r' 97 'a'
(gdb)
```

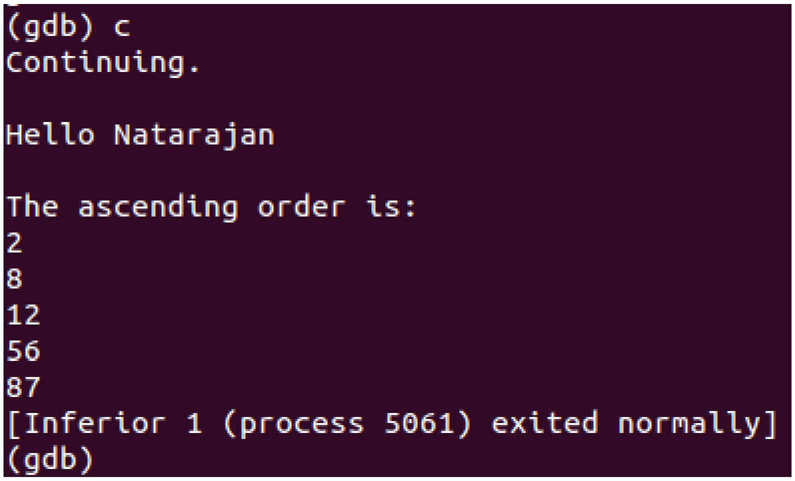
Remember that the **MaxOfTen** program is still running. But we are able to pause the program and examine its execution.

15. Examine the processor registers using the command **info registers**. This will print the CPU registers and their contents.



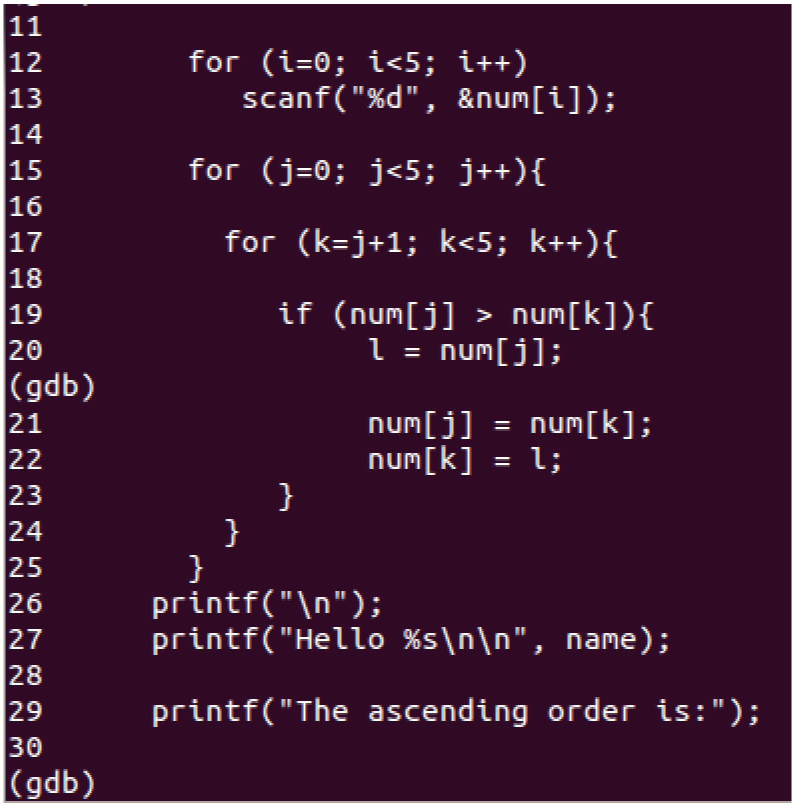
```
(gdb) info registers
eax            0xc      12
ecx            0x1      1
edx            0x2      2
ebx            0x2a8ff4 2789364
esp            0xbffff280       0xbffff280
ebp            0xbffff2e8       0xbffff2e8
esi            0x0      0
edi            0x0      0
eip            0x80485b1        0x80485b1 <main+205>
eflags         0x297    [ CF PF AF SF IF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x33     51
(gdb)
```

16. Press **c** to continue and finish up the execution of the program. Type **quit** and press **enter** to quit gdb.



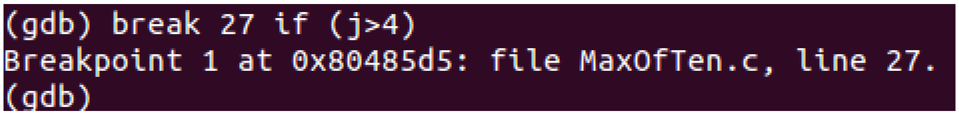
```
(gdb) c
Continuing.

Hello Natarajan

The ascending order is:
2
8
12
56
87
[Inferior 1 (process 5061) exited normally]
(gdb)
```

17. Next, you will set a conditional breakpoint i.e. there will be a breakpoint if some condition is met. You will set a condition that will break the execution once the value of **j > 4**. Then you will modify the **name** string in which the name you entered was stored. To do this, load the program into gdb again. Use **list 1** to list the code. Keep pressing **enter** till the entire code has been listed.



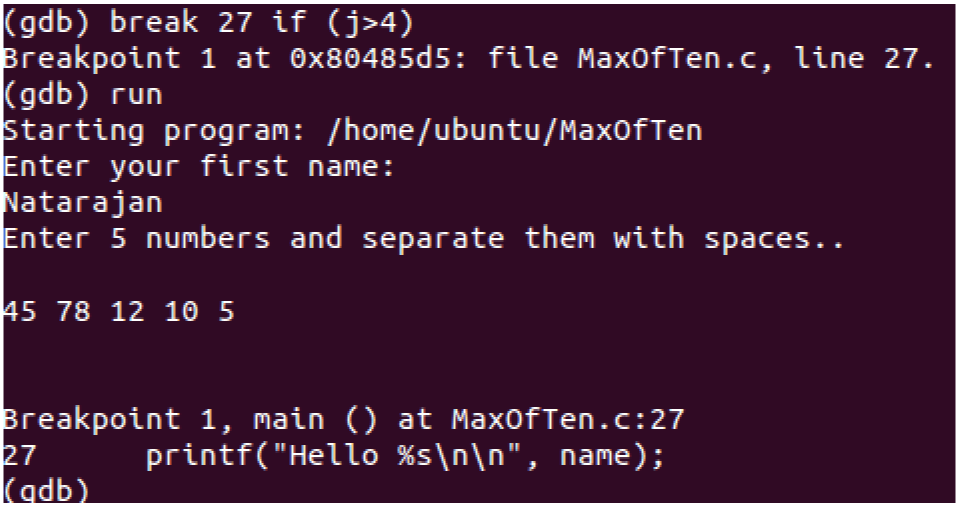
```
11
12          for (i=0; i<5; i++)
13             scanf("%d", &num[i]);
14
15          for (j=0; j<5; j++){
16
17            for (k=j+1; k<5; k++){
18
19               if (num[j] > num[k]){
20                    l = num[j];
(gdb)
21                    num[j] = num[k];
22                    num[k] = l;
23               }
24            }
25          }
26        printf("\n");
27        printf("Hello %s\n\n", name);
28
29        printf("The ascending order is:");
30
(gdb)
```

18. Break at the line that you have the **printf("Hello %s\n\n", name);** statement. From the screenshot above, the statement is at line 27. Use the command **break 27 if (j > 4)** and hit enter.



```
(gdb) break 27 if (j>4)
Breakpoint 1 at 0x80485d5: file MaxOfTen.c, line 27.
(gdb)
```

19. Run the program, enter you first name as required and press **enter**. Next, enter any 5 integers and press **enter**.

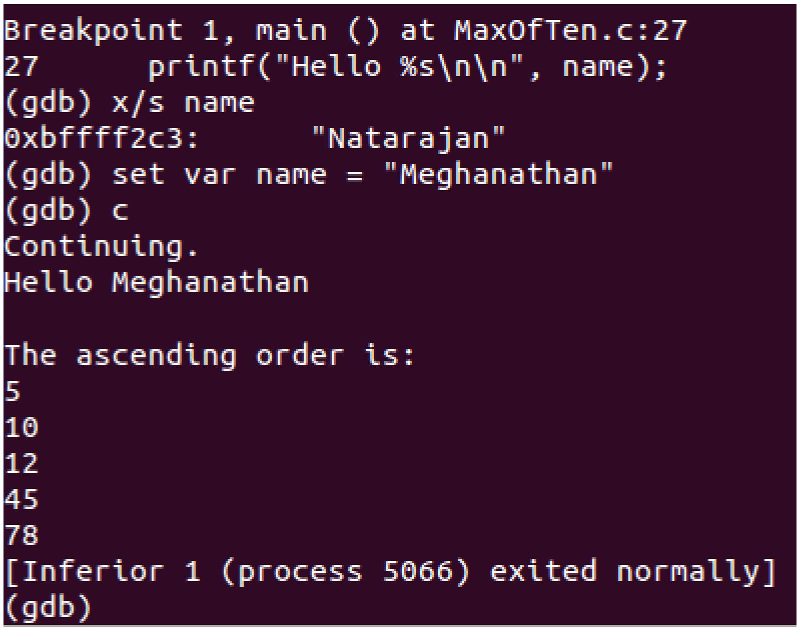


```
(gdb) break 27 if (j>4)
Breakpoint 1 at 0x80485d5: file MaxOfTen.c, line 27.
(gdb) run
Starting program: /home/ubuntu/MaxOfTen
Enter your first name:
Natarajan
Enter 5 numbers and separate them with spaces..

45 78 12 10 5


Breakpoint 1, main () at MaxOfTen.c:27
27        printf("Hello %s\n\n", name);
(gdb)
```

20. Now at the break point, enter **x/s name** to print the current value of the variable **name**. Change the value of the variable **name** from your first name to your last name. To do this, type in the command set var name = "Meghanathan"   where Meghanathan should be switched with your last name.

Examine the content of the variable **name** again and see that it has been changed.

Type **c** to continue and observe that your last name will be printed instead of your first name.



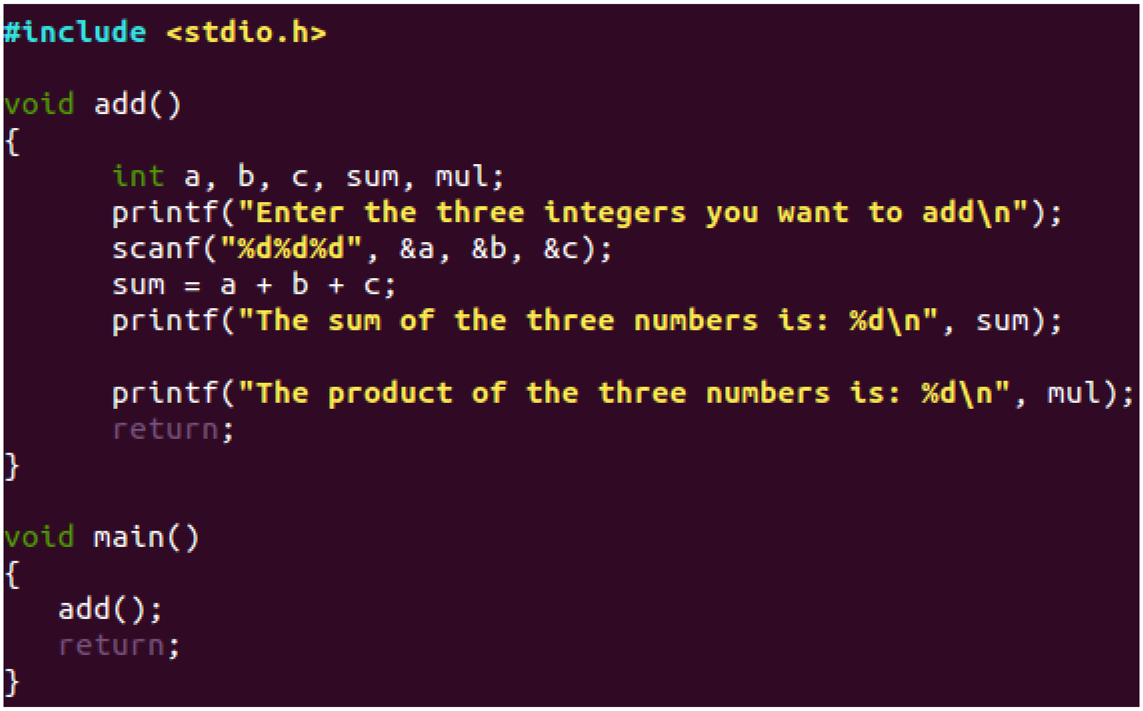
```
Breakpoint 1, main () at MaxOfTen.c:27
27      printf("Hello %s\n\n", name);
(gdb) x/s name
0xbffff2c3:      "Natarajan"
(gdb) set var name = "Meghanathan"
(gdb) c
Continuing.
Hello Meghanathan

The ascending order is:
5
10
12
45
78
[Inferior 1 (process 5066) exited normally]
(gdb)
```

Type **quit** and hit **enter** to quit gdb.

Phase 2: Exploring GDB on Your Own Tasks
35 points

What to Submit (Phase 2): Screenshots of **ALL** the tasks. Indicate the task # in your screenshots.

1. Type and compile the simple code in the screenshot below. Name the program whatever you want.



```
#include <stdio.h>

void add()
{
      int a, b, c, sum, mul;
      printf("Enter the three integers you want to add\n");
      scanf("%d%d%d", &a, &b, &c);
      sum = a + b + c;
      printf("The sum of the three numbers is: %d\n", sum);

      printf("The product of the three numbers is: %d\n", mul);
      return;
}

void main()
{
   add();
   return;
}
```

2. Load the program into gdb. You need the executable to load with gd. So, make sure to compile your C program.
3. List all the code in gdb.
4. Disassemble the main function.
5. Disassemble the add function.
6. Set a breakpoint at the add function.
7. Run the program.
8. In the execution of the program, modify and double the value of a, b, and c. So for example, if your initial values of a, b, and c were 2, 4, and 8, you should double and make them 4, 8, and 16. Make sure you do this after the values of a, b, and c have been added and stored in sum.
9. Set the value of mul to be the product of the doubled values of a, b, and c.
10. Finish the execution of the program.

Hint: If you run the program and enter 2, 4, and 8 as the input values of a, b, and c. The value of **sum** printed in the program should be 14 which is 2 + 4 + 8, and the value of **mul** printed in the program should be 512 which is 4 * 8 * 16.

Phase 3: Using gdb to obtain a password read into memory from a protected file

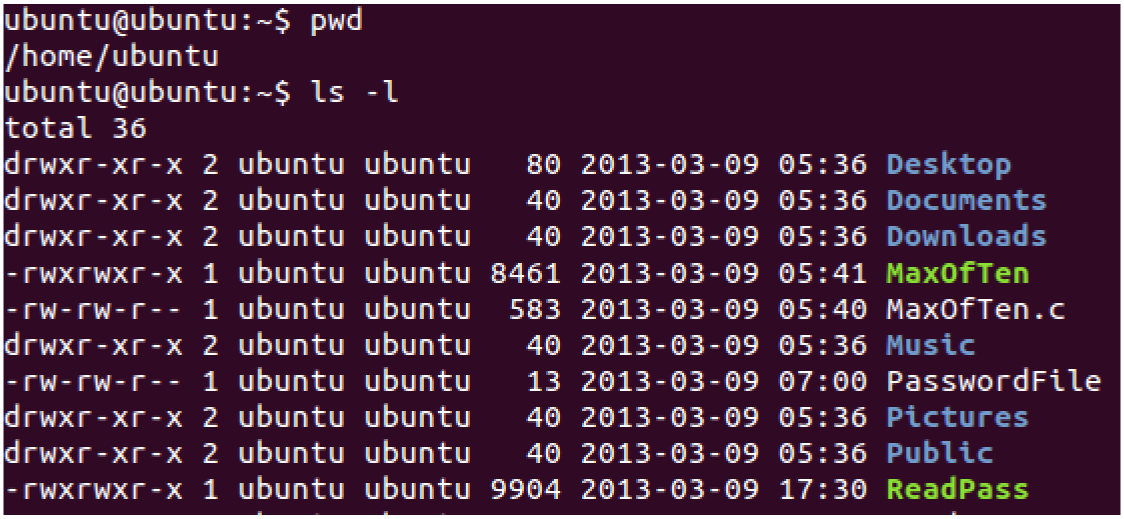
The objective of this phase of the project is to gain access to a password-protected program. This will be achieved by loading the program into gdb and breaking the execution process in order to peek into the stack and copy the password as soon as it is loaded from a file. Your goal will be to find out the correct password value stored in the PasswordFile (i.e., mimic as if you are not looking at the PasswordFile; but through the execution stack using gdb), enter it when prompted for the user password and have the program print that you entered the correct password. The executable file of the password reading program ReadPass is provided to you in JSU Blackboard site for the course as well as emailed to you. You need to transfer it to your virtual machine. You can create the PasswordFile as a simple text file in your present Linux working directory.

<u>Tasks</u>
<u>40 points</u>

What to Submit (Phase 3): Screenshots of **<u>ALL</u>** the tasks. Indicate the task # in your screenshots.

Password to be stored in PasswordFile for each student: Your password will be the name of your home city and a 5-digit zip code appended to it. For example, if your home city is Clinton and the zip code is 39056, your password will be Clinton39056.

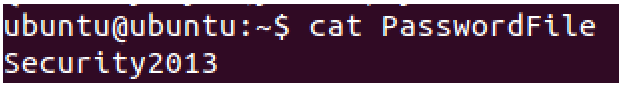
1. Briefly explain how you were able to get the executable of the password reading program **ReadPass** from JSU Blackboard or your email to the present working directory of your virtual machine. Also, show that your directory now has the ReadPass executable using the **ls -l** command. Make sure you save the downloaded executable as **ReadPass**



```
ubuntu@ubuntu:~$ pwd
/home/ubuntu
ubuntu@ubuntu:~$ ls -l
total 36
drwxr-xr-x 2 ubuntu ubuntu   80 2013-03-09 05:36 Desktop
drwxr-xr-x 2 ubuntu ubuntu   40 2013-03-09 05:36 Documents
drwxr-xr-x 2 ubuntu ubuntu   40 2013-03-09 05:36 Downloads
-rwxrwxr-x 1 ubuntu ubuntu 8461 2013-03-09 05:41 MaxOfTen
-rw-rw-r-- 1 ubuntu ubuntu  583 2013-03-09 05:40 MaxOfTen.c
drwxr-xr-x 2 ubuntu ubuntu   40 2013-03-09 05:36 Music
-rw-rw-r-- 1 ubuntu ubuntu   13 2013-03-09 07:00 PasswordFile
drwxr-xr-x 2 ubuntu ubuntu   40 2013-03-09 05:36 Pictures
drwxr-xr-x 2 ubuntu ubuntu   40 2013-03-09 05:36 Public
-rwxrwxr-x 1 ubuntu ubuntu 9904 2013-03-09 17:30 ReadPass
```

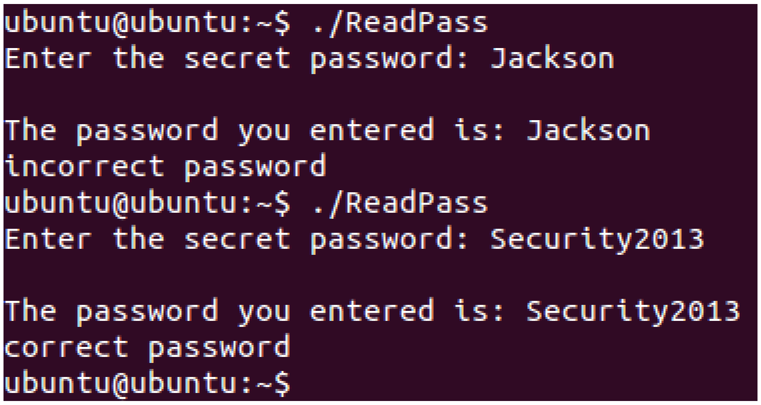
2. Create a password file (named **PasswordFile**) using an editor (e.g., pico) of your choice. Type in the password assigned to each of you in the beginning of the project description. In this description, I enter the password **Security2013** into the file. Note that the S in the text is in

uppercase. Save the password in the file **PasswordFile** and exit. Use the cat command to print the contents of the PasswordFile and capture it as a screenshot.



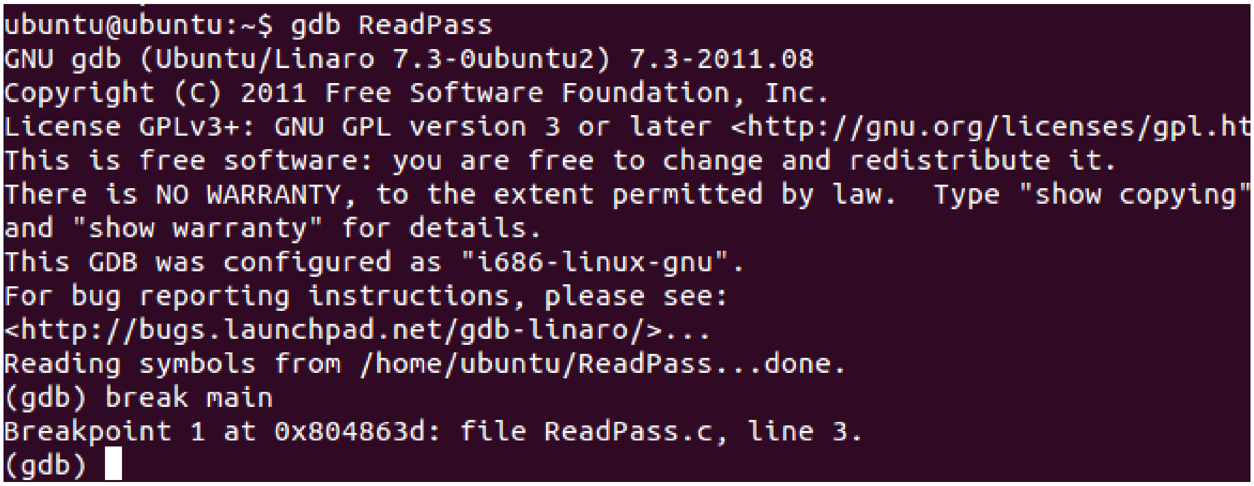
```
ubuntu@ubuntu:~$ cat PasswordFile
Security2013
```

3. Run the program using **./ReadPass** This program will ask you to enter the secret password. When it requires the password, type in **Jackson**. Jackson is not the correct password. The purpose of typing it is to see what the program does when the password is incorrect. Now run the program again and this time enter the password you saved in the PasswordFile (in my case, it is Security2013).



```
ubuntu@ubuntu:~$ ./ReadPass
Enter the secret password: Jackson

The password you entered is: Jackson
incorrect password
ubuntu@ubuntu:~$ ./ReadPass
Enter the secret password: Security2013

The password you entered is: Security2013
correct password
ubuntu@ubuntu:~$
```

4. Load the program into gdb using **gdb ReadPass**. Break the program at the main function using the command **break main**.

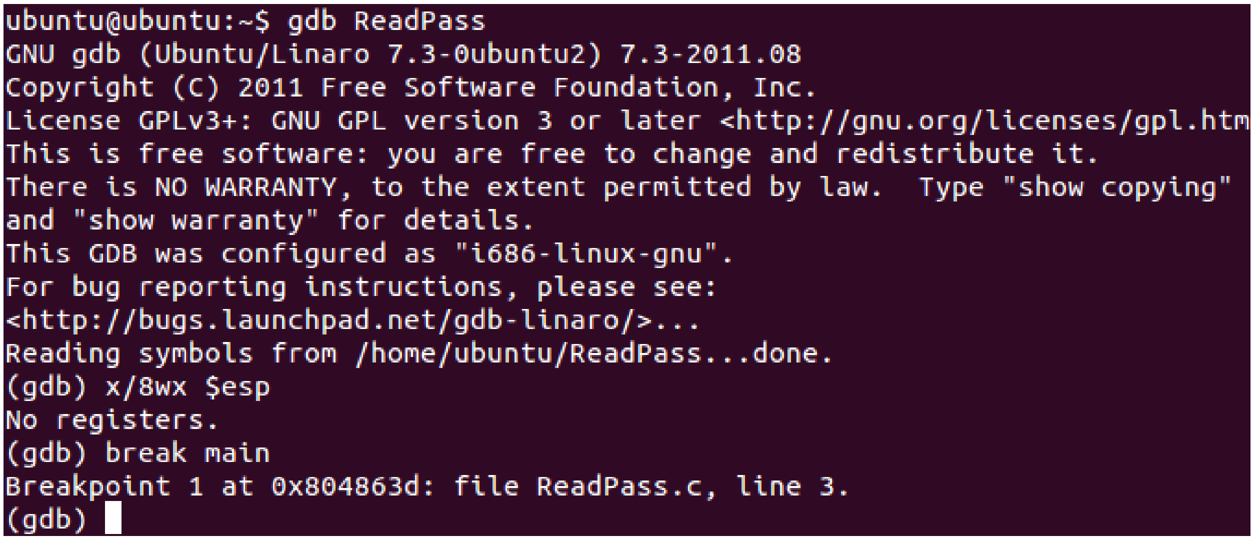


```
ubuntu@ubuntu:~$ gdb ReadPass
GNU gdb (Ubuntu/Linaro 7.3-0ubuntu2) 7.3-2011.08
Copyright (C) 2011 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.ht
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /home/ubuntu/ReadPass...done.
(gdb) break main
Breakpoint 1 at 0x804863d: file ReadPass.c, line 3.
(gdb) 
```

5. Use the **list** command to see the contents of the source code. Find out which variable in the program is used to store the password retrieved from the PasswordFile. What is the variable name?

6. Run the program using the command **run.** Keep pressing **s** to run the program line by line after the breakpoint. Once the execution is past the line where the password is retrieved from the file, use the appropriate gdb command to find out the value of the variable that stores the value of the password retrieved from the file. Keep pressing **s** and when it comes to the point where the program asks the user to enter a password, enter the password that you believe you have figured out (i.e., retrieved from the file). The program will validate the password you entered and print out whether your entry is correct or not correct.

7. Load the program again into gdb using **gdb ReadPass**. Examine the stack using the command **x/8wx $esp** and hit **enter**. Notice the result "No registers". This means that the program has not been run and therefore nothing has been loaded into the registers. Break the program at the main function using the command **break main**.

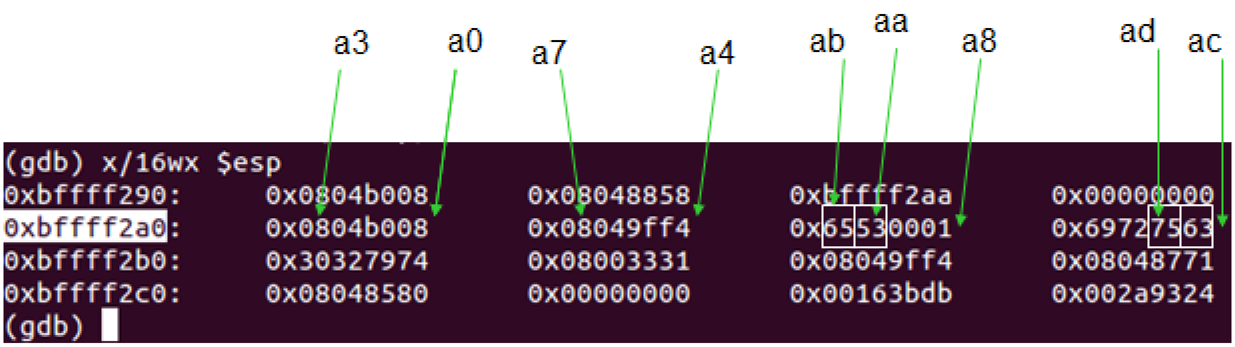


```
ubuntu@ubuntu:~$ gdb ReadPass
GNU gdb (Ubuntu/Linaro 7.3-0ubuntu2) 7.3-2011.08
Copyright (C) 2011 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.htm
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /home/ubuntu/ReadPass...done.
(gdb) x/8wx $esp
No registers.
(gdb) break main
Breakpoint 1 at 0x804863d: file ReadPass.c, line 3.
(gdb) 
```

8. Now that you have figured out the name of the variable that stores the password retrieved from the file, find its virtual memory address and take a screenshot of the command used along with the memory address value displayed.

9. Examine the contents of the stack using the **x/8wx $esp** command and with different integer values **(like 8, 12, 16, etc)** until you can obtain a good range of virtual memory addresses that encompass the virtual memory address you found in task 8. Now, through these virtual memory addresses and the hexadecimal values of the contents stored in these addresses, extract the value of the password retrieved from the file. Next page, I show a screenshot of the strategy I use to retrieve the contents Security2013.

In my case, the starting virtual memory address of the variable that stores the password retrieved from the file is **0xbffff2aa**. As I indicate in the following screenshot, the values in the stack are the hexadecimal ASCII values of the characters in the string 'Security2013' read from the reverse order. Repeat the same for your password (note that the starting virtual memory address for your password variable may be different or same - this is what you have to figure out from Task 8) and provide the appropriate screenshots.



```
(gdb) x/16wx $esp
0xbffff290:     0x0804b008      0x08048858      0xbffff2aa      0x00000000
0xbffff2a0:     0x0804b008      0x08049ff4      0x65530001      0x69727563
0xbffff2b0:     0x30327974      0x08003331      0x08049ff4      0x08048771
0xbffff2c0:     0x08048580      0x00000000      0x00163bdb      0x002a9324
(gdb) 
```

53 at 0xbffff2aa is the hexadecimal ASCII value of ‘S’ [decimal ASCII value is 83]
65 at 0xbffff2ab is the hexadecimal ASCII value of ‘e’ [decimal ASCII value is 101]
63 at 0xbffff2ac is the hexadecimal ASCII value of ‘c’ [decimal ASCII value is 99]
75 at 0xbffff2ad is the hexadecimal ASCII value of ‘u’ [decimal ASCII value is 117]

and so on………