Design of Secure Computer Systems

Lab 02

Printf

This LAB will be PrintF which will emphasize on the printf function and explore the manner in which the function references memory addresses in response to its given format specification.

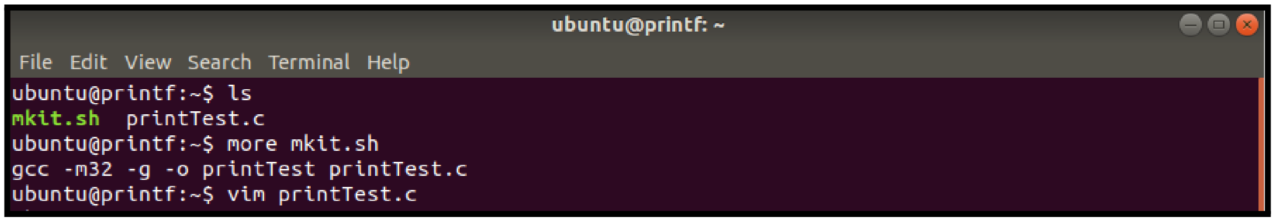
Name: Rakshita Mathur

Printf

Started the lab using the command: **labtainer printf**
After the lab started in the printf command we use **ls** to see the folders available for the lab.

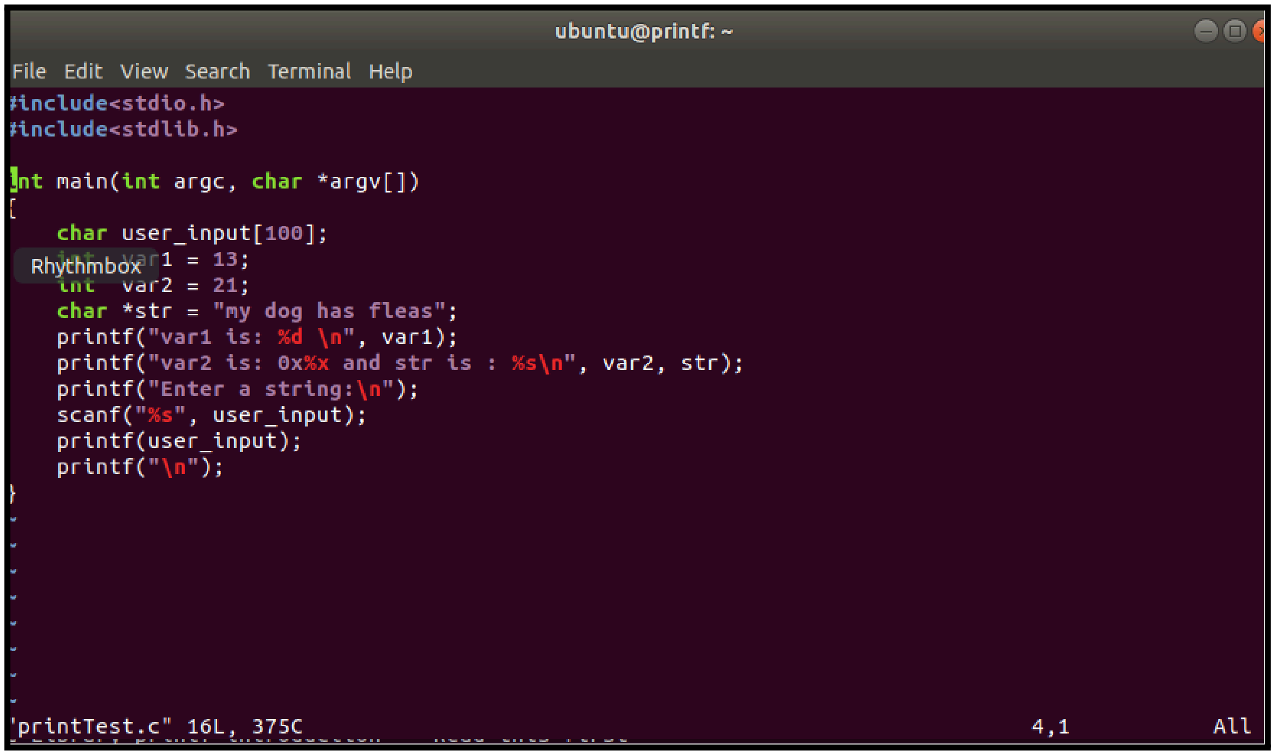
1. Reviewing the printTest.c program

Command used: **vim printTest.c**



```
ubuntu@printf:~$ ls
mkit.sh  printTest.c
ubuntu@printf:~$ more mkit.sh
gcc -m32 -g -o printTest printTest.c
ubuntu@printf:~$ vim printTest.c
```

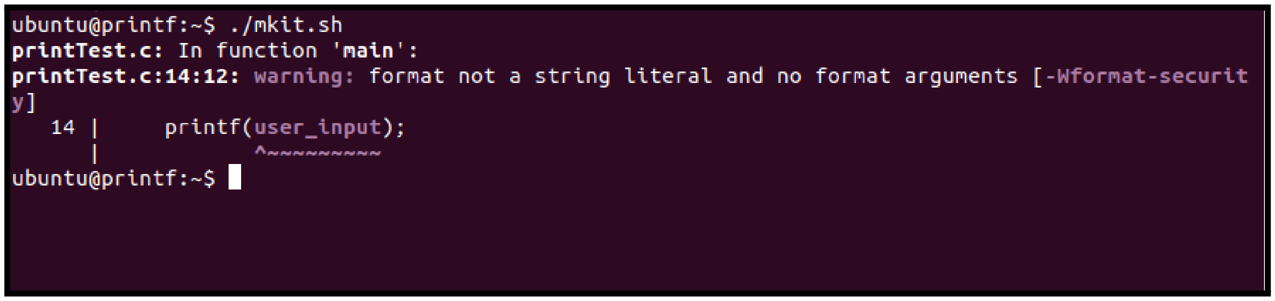
We observed the c program that has the main function body with three variables and the formatting characters. The syntax of printf has the first parameter is a format string that contains literal text to be displayed and one and more than one or more conversion specifications that determine how any remaining parameters are displayed.



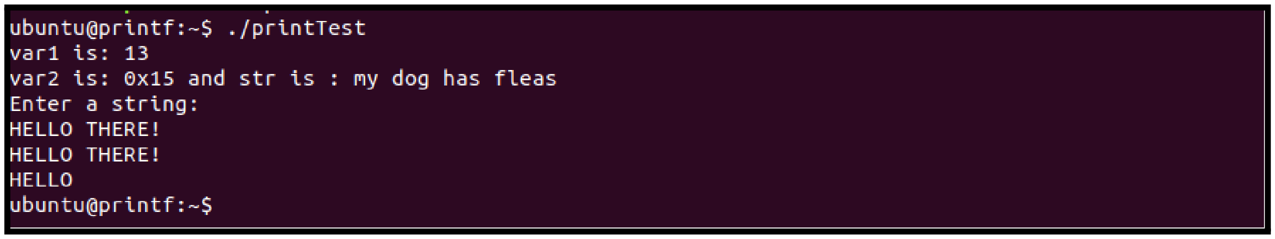
```
#include<stdio.h>
#include<stdlib.h>

int main(int argc, char *argv[])
{
    char user_input[100];
    int var1 = 13;
    int  var2 = 21;
    char *str = "my dog has fleas";
    printf("var1 is: %d \n", var1);
    printf("var2 is: 0x%x and str is : %s\n", var2, str);
    printf("Enter a string:\n");
    scanf("%s", user_input);
    printf(user_input);
    printf("\n");
}
~
~
~
~
~
~
~
~
"printTest.c" 16L, 375C                                        4,1           All
```

1. Run printTest

Compiling the code using the **./mkit** command and running the program using the **./printTest** command to observe the output.



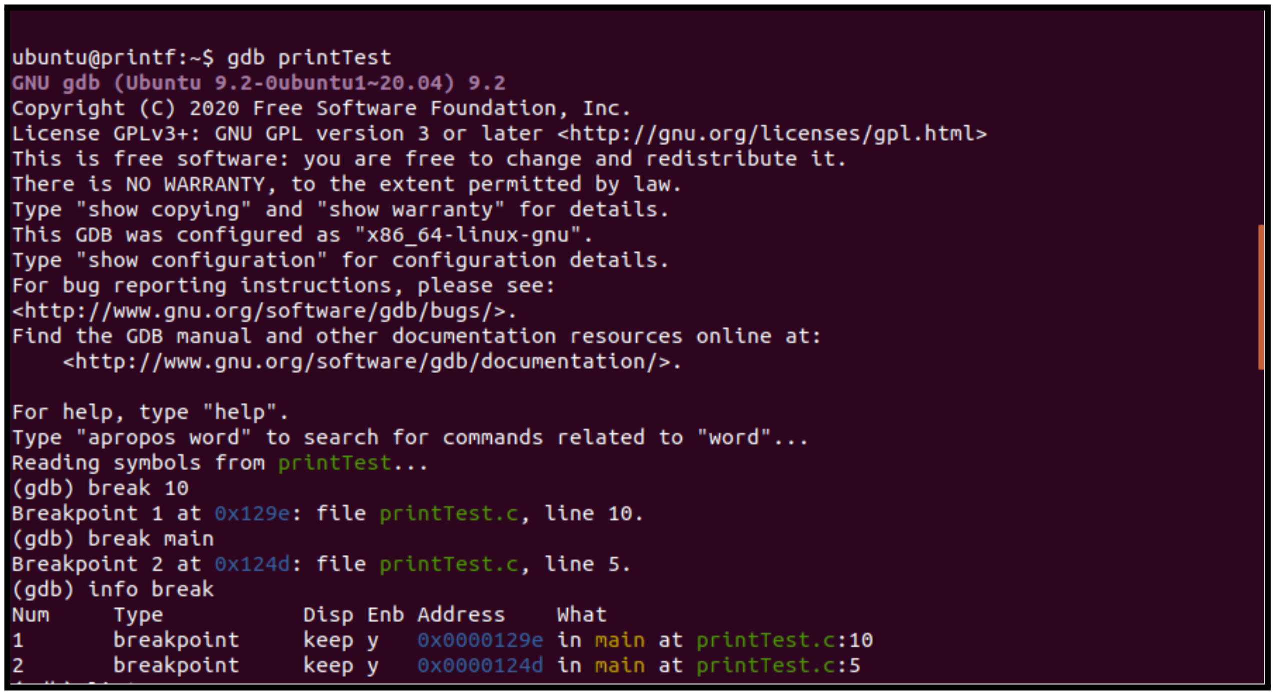
```
ubuntu@printf:~$ ./mkit.sh
printTest.c: In function 'main':
printTest.c:14:12: warning: format not a string literal and no format arguments [-Wformat-securit
y]
   14 |     printf(user_input);
      |            ^~~~~~~~~~
ubuntu@printf:~$
```



```
ubuntu@printf:~$ ./printTest
var1 is: 13
var2 is: 0x15 and str is : my dog has fleas
Enter a string:
HELLO THERE!
HELLO THERE!
HELLO
ubuntu@printf:~$
```

5. Observing Calling conventions with gdb

Run the program in gdb: **gdb printTest**

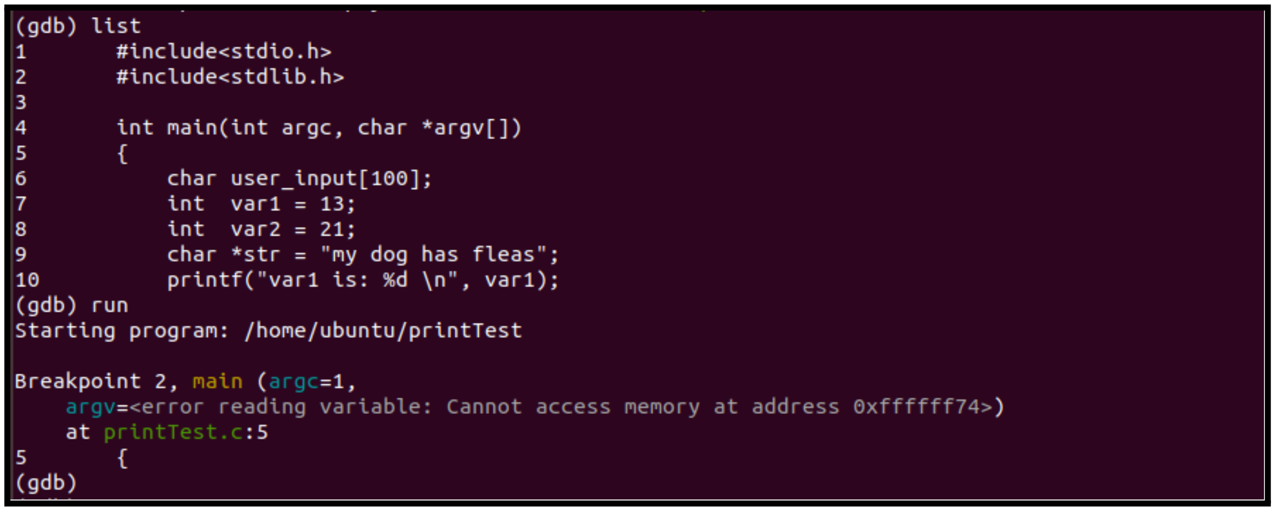


```
ubuntu@printf:~$ gdb printTest
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from printTest...
(gdb) break 10
Breakpoint 1 at 0x129e: file printTest.c, line 10.
(gdb) break main
Breakpoint 2 at 0x124d: file printTest.c, line 5.
(gdb) info break
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x0000129e in main at printTest.c:10
2       breakpoint     keep y   0x0000124d in main at printTest.c:5
```

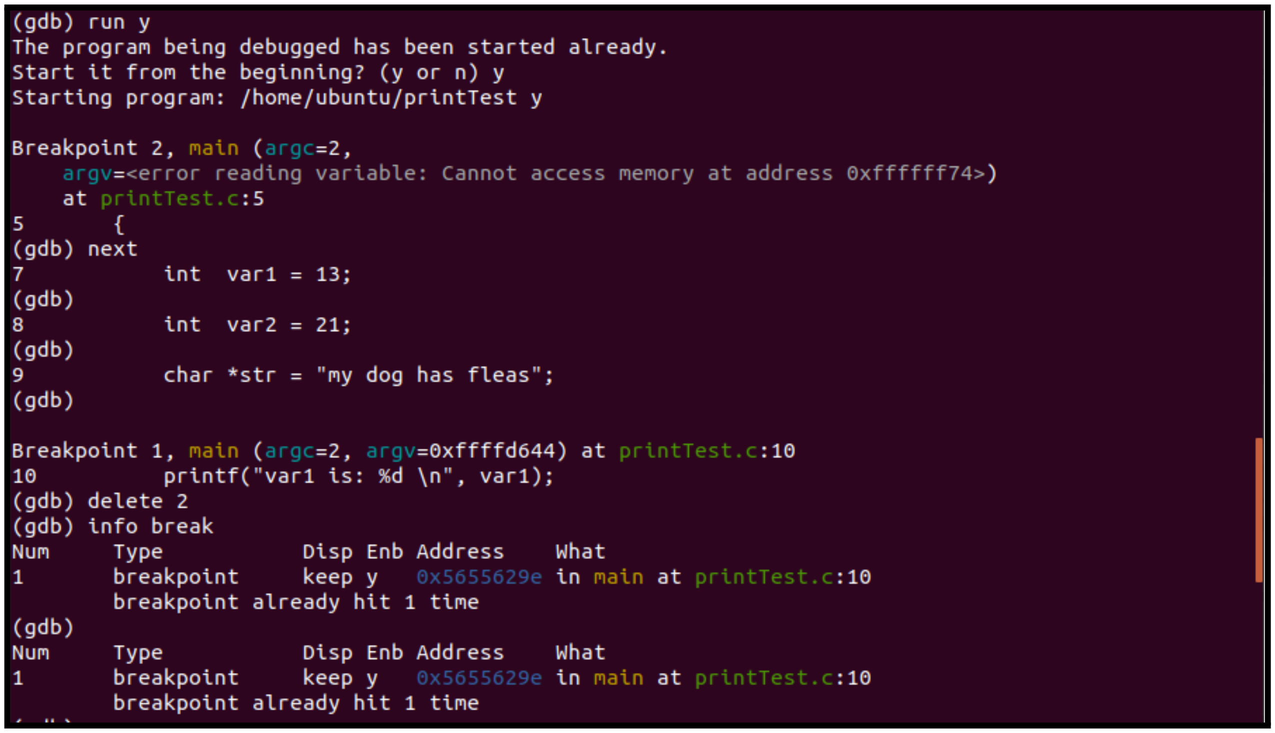
Listing the program with the list command at setting the breakpoint at the line of the first printf statement and run:

break <number>

run



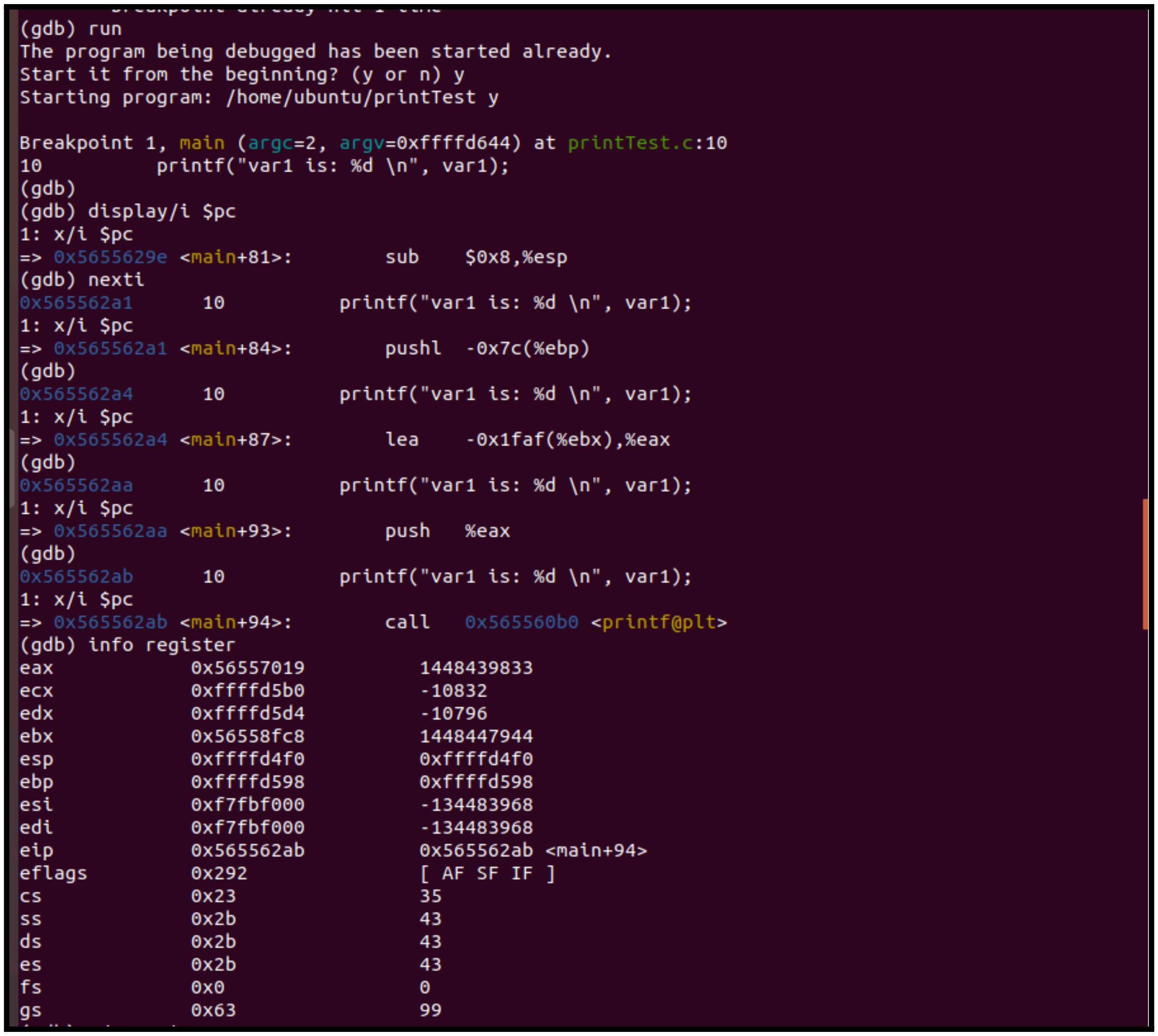
```
(gdb) list
1       #include<stdio.h>
2       #include<stdlib.h>
3
4       int main(int argc, char *argv[])
5       {
6           char user_input[100];
7           int  var1 = 13;
8           int  var2 = 21;
9           char *str = "my dog has fleas";
10          printf("var1 is: %d \n", var1);
(gdb) run
Starting program: /home/ubuntu/printTest

Breakpoint 2, main (argc=1,
    argv=<error reading variable: Cannot access memory at address 0xffffff74>)
    at printTest.c:5
5       {
(gdb)
```



```
(gdb) run y
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/ubuntu/printTest y

Breakpoint 2, main (argc=2,
    argv=<error reading variable: Cannot access memory at address 0xffffff74>)
    at printTest.c:5
5       {
(gdb) next
7           int  var1 = 13;
(gdb)
8           int  var2 = 21;
(gdb)
9           char *str = "my dog has fleas";
(gdb)

Breakpoint 1, main (argc=2, argv=0xffffd644) at printTest.c:10
10          printf("var1 is: %d \n", var1);
(gdb) delete 2
(gdb) info break
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x5655629e in main at printTest.c:10
        breakpoint already hit 1 time
(gdb)
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x5655629e in main at printTest.c:10
        breakpoint already hit 1 time
```

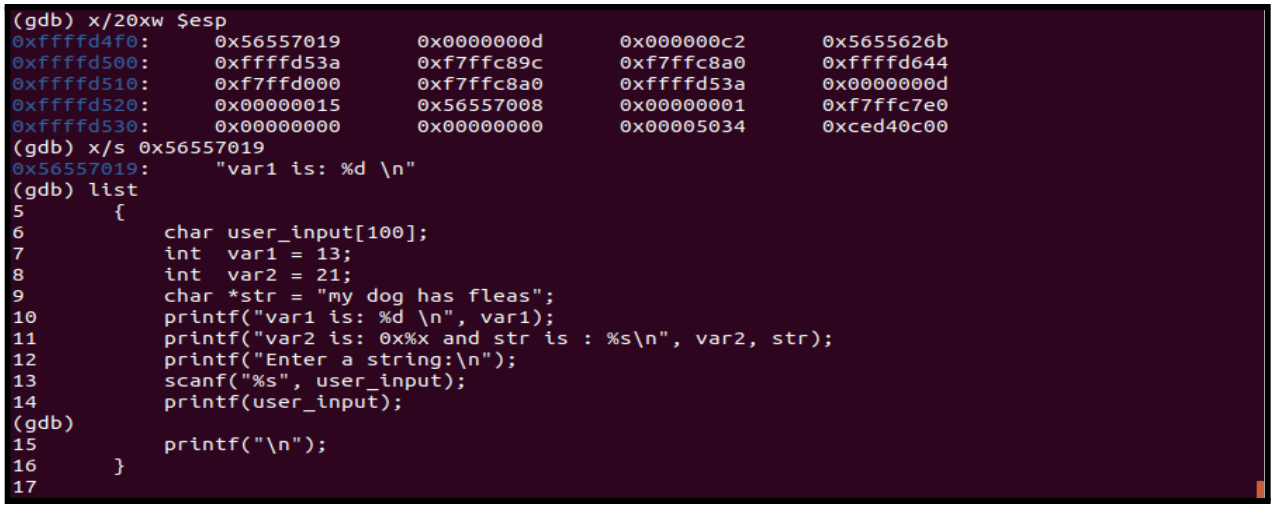
The program will break just before the call to printf. But not close enough for our purposes, so we will view the disassembly of the machine instructions so that we can advance execution to just before the actual call.

To display the disassembly of the current instruction: **display/i $pc**



```
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/ubuntu/printTest y

Breakpoint 1, main (argc=2, argv=0xffffd644) at printTest.c:10
10          printf("var1 is: %d \n", var1);
(gdb)
(gdb) display/i $pc
1: x/i $pc
=> 0x5655629e <main+81>:        sub    $0x8,%esp
(gdb) nexti
0x565562a1      10          printf("var1 is: %d \n", var1);
1: x/i $pc
=> 0x565562a1 <main+84>:        pushl  -0x7c(%ebp)
(gdb)
0x565562a4      10          printf("var1 is: %d \n", var1);
1: x/i $pc
=> 0x565562a4 <main+87>:        lea    -0x1faf(%ebx),%eax
(gdb)
0x565562aa      10          printf("var1 is: %d \n", var1);
1: x/i $pc
=> 0x565562aa <main+93>:        push   %eax
(gdb)
0x565562ab      10          printf("var1 is: %d \n", var1);
1: x/i $pc
=> 0x565562ab <main+94>:        call   0x565560b0 <printf@plt>
(gdb) info register
eax            0x56557019          1448439833
ecx            0xffffd5b0          -10832
edx            0xffffd5d4          -10796
ebx            0x56558fc8          1448447944
esp            0xffffd4f0          0xffffd4f0
ebp            0xffffd598          0xffffd598
esi            0xf7fbf000          -134483968
edi            0xf7fbf000          -134483968
eip            0x565562ab          0x565562ab <main+94>
eflags         0x292               [ AF SF IF ]
cs             0x23                35
ss             0x2b                43
ds             0x2b                43
es             0x2b                43
fs             0x0                 0
gs             0x63                99
```

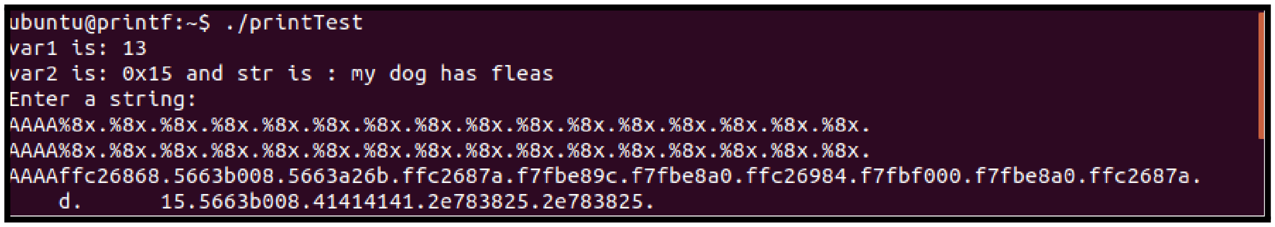
Then use the **nexti** instruction to advance execution to the next instruction. Repeatedly press the Return key to keep stepping until you reach the call to printf@plt Now the program is really just about to call printf. Look at twenty words on the stack as hexadecimal values: **x/20xw $esp**



```
(gdb) x/20xw $esp
0xffffd4f0:     0x56557019      0x0000000d      0x000000c2      0x5655626b
0xffffd500:     0xffffd53a      0xf7ffc89c      0xf7ffc8a0      0xffffd644
0xffffd510:     0xf7ffd000      0xf7ffc8a0      0xffffd53a      0x0000000d
0xffffd520:     0x00000015      0x56557008      0x00000001      0xf7ffc7e0
0xffffd530:     0x00000000      0x00000000      0x00005034      0xced40c00
(gdb) x/s 0x56557019
0x56557019:     "var1 is: %d \n"
(gdb) list
5       {
6               char user_input[100];
7               int  var1 = 13;
8               int  var2 = 21;
9               char *str = "my dog has fleas";
10              printf("var1 is: %d \n", var1);
11              printf("var2 is: 0x%x and str is : %s\n", var2, str);
12              printf("Enter a string:\n");
13              scanf("%s", user_input);
14              printf(user_input);
(gdb)
15              printf("\n");
16      }
17
```

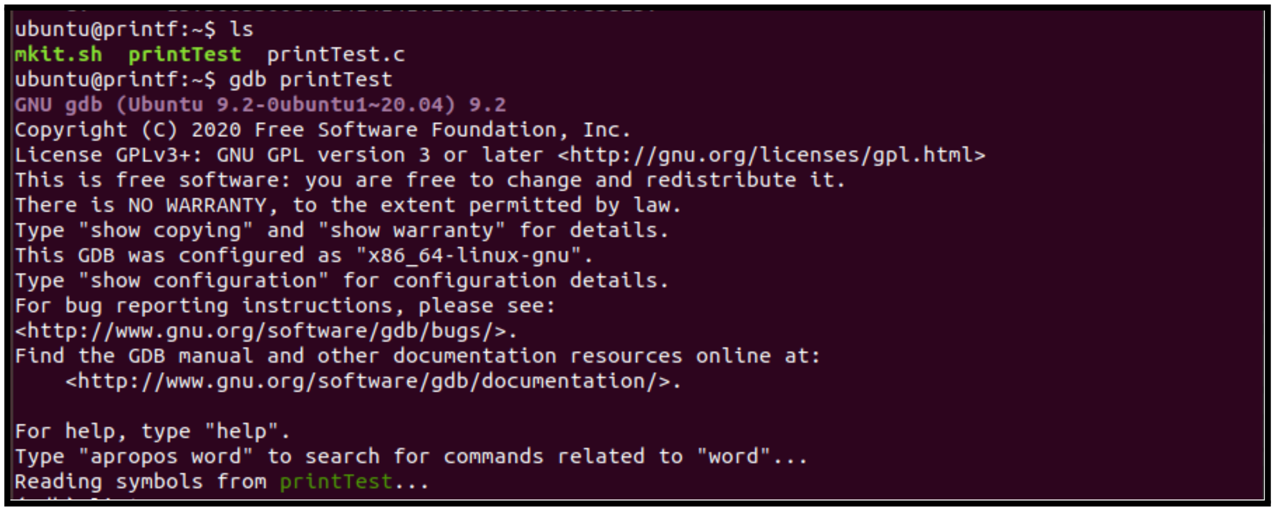
6. User input in format strings

Compiling the code and running the printf command with multiple 8 digit hexadecimal values.
Command used for entering a string:
AAAA%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.



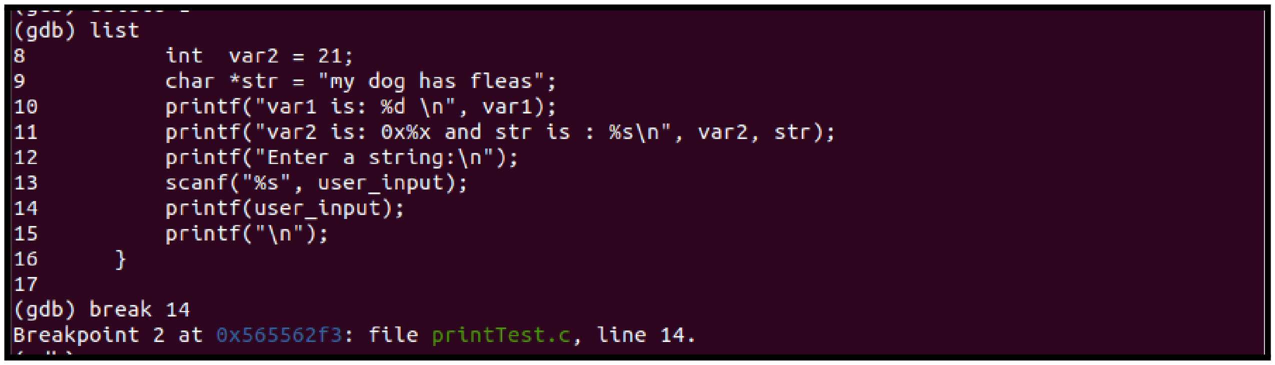
```
ubuntu@printf:~$ ./printTest
var1 is: 13
var2 is: 0x15 and str is : my dog has fleas
Enter a string:
AAAA%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.
AAAA%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.
AAAAffc26868.5663b008.5663a26b.ffc2687a.f7fbe89c.f7fbe8a0.ffc26984.f7fbf000.f7fbe8a0.ffc2687a.
       d.      15.5663b008.41414141.2e783825.2e783825.
```

Running gdc on printTest.c



```
ubuntu@printf:~$ ls
mkit.sh  printTest  printTest.c
ubuntu@printf:~$ gdb printTest
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 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-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from printTest...
```

We observed by adding the breakpoints as we did before.



```
(gdb) list
8           int  var2 = 21;
9           char *str = "my dog has fleas";
10          printf("var1 is: %d \n", var1);
11          printf("var2 is: 0x%x and str is : %s\n", var2, str);
12          printf("Enter a string:\n");
13          scanf("%s", user_input);
14          printf(user_input);
15          printf("\n");
16      }
17
(gdb) break 14
Breakpoint 2 at 0x565562f3: file printTest.c, line 14.
```

Used: **display/i $pc**

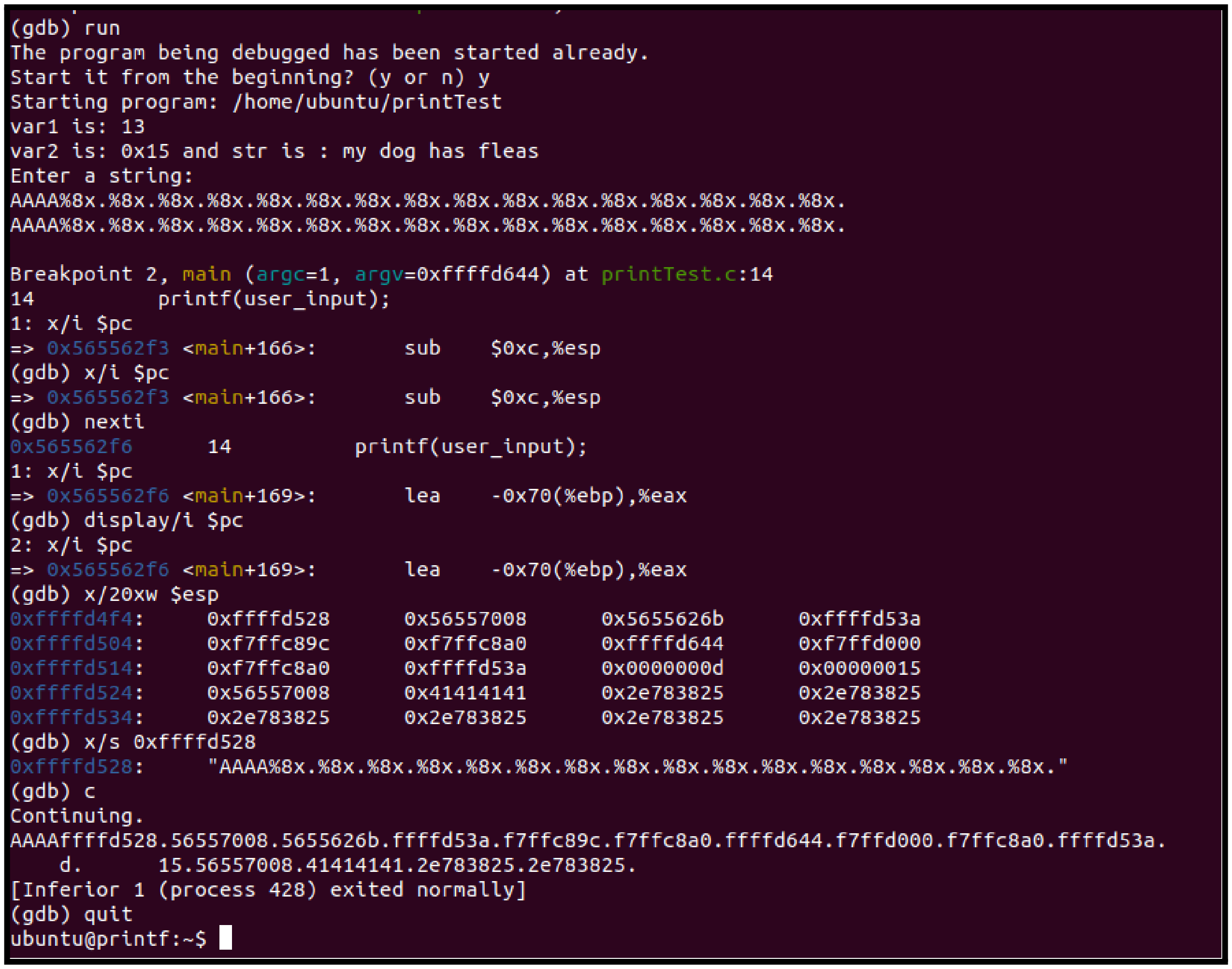
nexti

<return>...

to step to the call to **printf@plt** and then display the stack content.

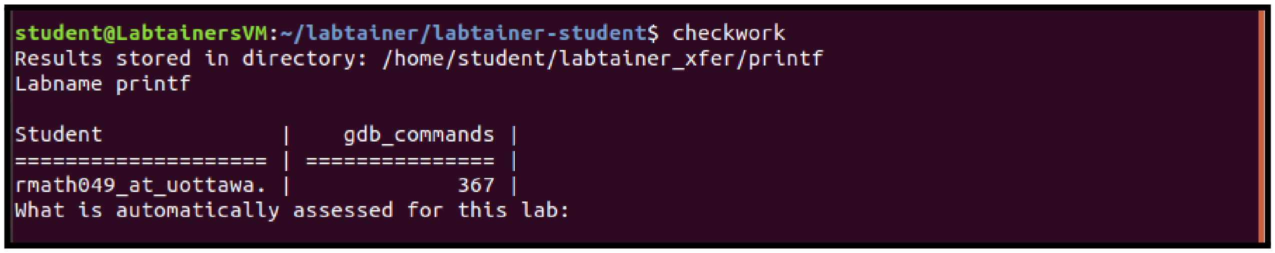
x/20x2 $esp

To find the first (and only) parameter to the printf statement and confirm it is the address of our user-provided format string we use the command: **x/s <address>**



```
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/ubuntu/printTest
var1 is: 13
var2 is: 0x15 and str is : my dog has fleas
Enter a string:
AAAA%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.
AAAA%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.

Breakpoint 2, main (argc=1, argv=0xffffd644) at printTest.c:14
14          printf(user_input);
1: x/i $pc
=> 0x565562f3 <main+166>:       sub    $0xc,%esp
(gdb) x/i $pc
=> 0x565562f3 <main+166>:       sub    $0xc,%esp
(gdb) nexti
0x565562f6      14          printf(user_input);
1: x/i $pc
=> 0x565562f6 <main+169>:       lea    -0x70(%ebp),%eax
(gdb) display/i $pc
2: x/i $pc
=> 0x565562f6 <main+169>:       lea    -0x70(%ebp),%eax
(gdb) x/20xw $esp
0xffffd4f4:     0xffffd528      0x56557008      0x5655626b      0xffffd53a
0xffffd504:     0xf7ffc89c      0xf7ffc8a0      0xffffd644      0xf7ffd000
0xffffd514:     0xf7ffc8a0      0xffffd53a      0x0000000d      0x00000015
0xffffd524:     0x56557008      0x41414141      0x2e783825      0x2e783825
0xffffd534:     0x2e783825      0x2e783825      0x2e783825      0x2e783825
(gdb) x/s 0xffffd528
0xffffd528:     "AAAA%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x.%8x."
(gdb) c
Continuing.
AAAAffffd528.56557008.5655626b.ffffd53a.f7ffc89c.f7ffc8a0.ffffd644.f7ffd000.f7ffc8a0.ffffd53a.
    d.      15.56557008.41414141.2e783825.2e783825.
[Inferior 1 (process 428) exited normally]
(gdb) quit
ubuntu@printf:~$
```

Used checkwork command to check the lab work



```
student@LabtainersVM:~/labtainer/labtainer-student$ checkwork
Results stored in directory: /home/student/labtainer_xfer/printf
Labname printf

Student              |    gdb_commands |
==================== | =============== |
rmath049_at_uottawa. |             367 |
What is automatically assessed for this lab:
```