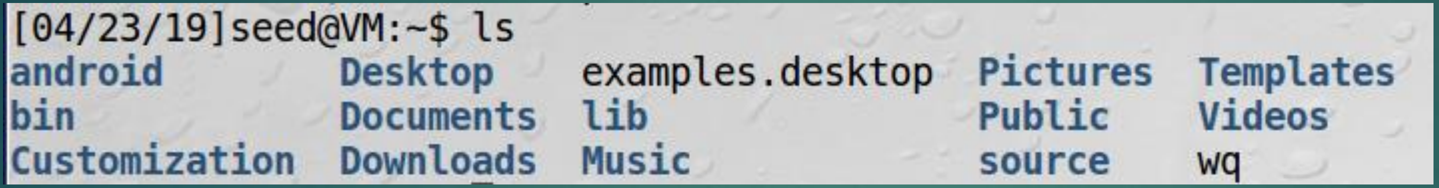
Stack Tracing

JUAN BERMUDEZ

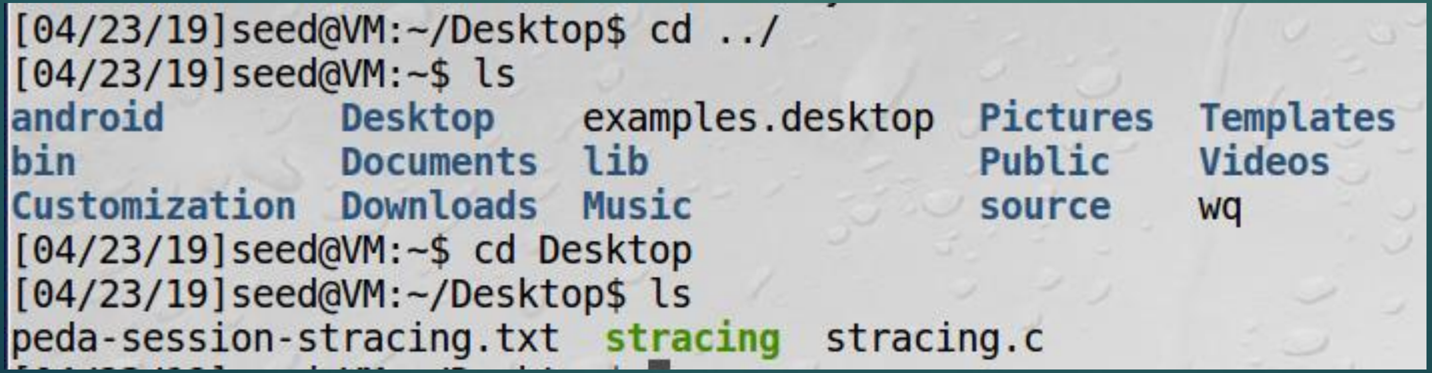
Running a program on linux

- 1. find the directory of the program:
  use the command <ls> to list the files of where your program is



```
[04/23/19]seed@VM:~$ ls
android        Desktop    examples.desktop  Pictures  Templates
bin            Documents  lib               Public    Videos
Customization  Downloads  Music             source    wq
```

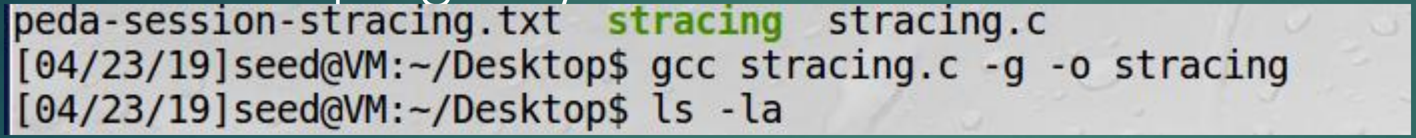
- The program is in Desktop and we need to execute it from it's location, use <cd Desktop>. The name of the c program is stracing.c
  .



```
[04/23/19]seed@VM:~/Desktop$ cd ../
[04/23/19]seed@VM:~$ ls
android        Desktop    examples.desktop  Pictures  Templates
bin            Documents  lib               Public    Videos
Customization  Downloads  Music             source    wq
[04/23/19]seed@VM:~$ cd Desktop
[04/23/19]seed@VM:~/Desktop$ ls
peda-session-stracing.txt  stracing  stracing.c
```

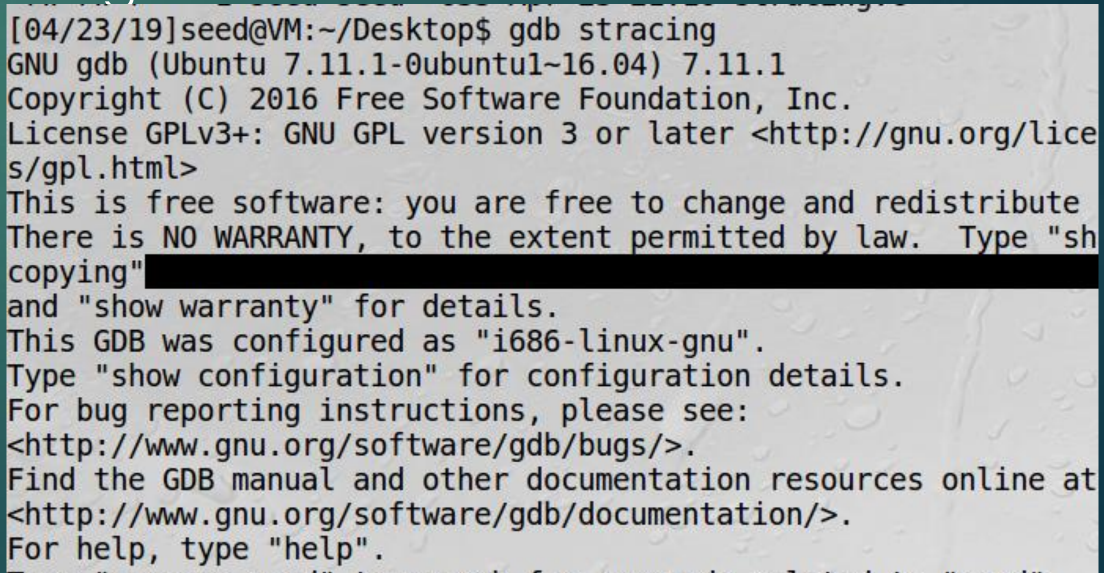
Compile the program and Run in in gdb debugger

- Type the following command:
  gcc stracing.c –g –o stracing (-g is for debugging and – o is to name to program)



```
peda-session-stracing.txt  stracing  stracing.c
[04/23/19]seed@VM:~/Desktop$ gcc stracing.c -g -o stracing
[04/23/19]seed@VM:~/Desktop$ ls -la
```

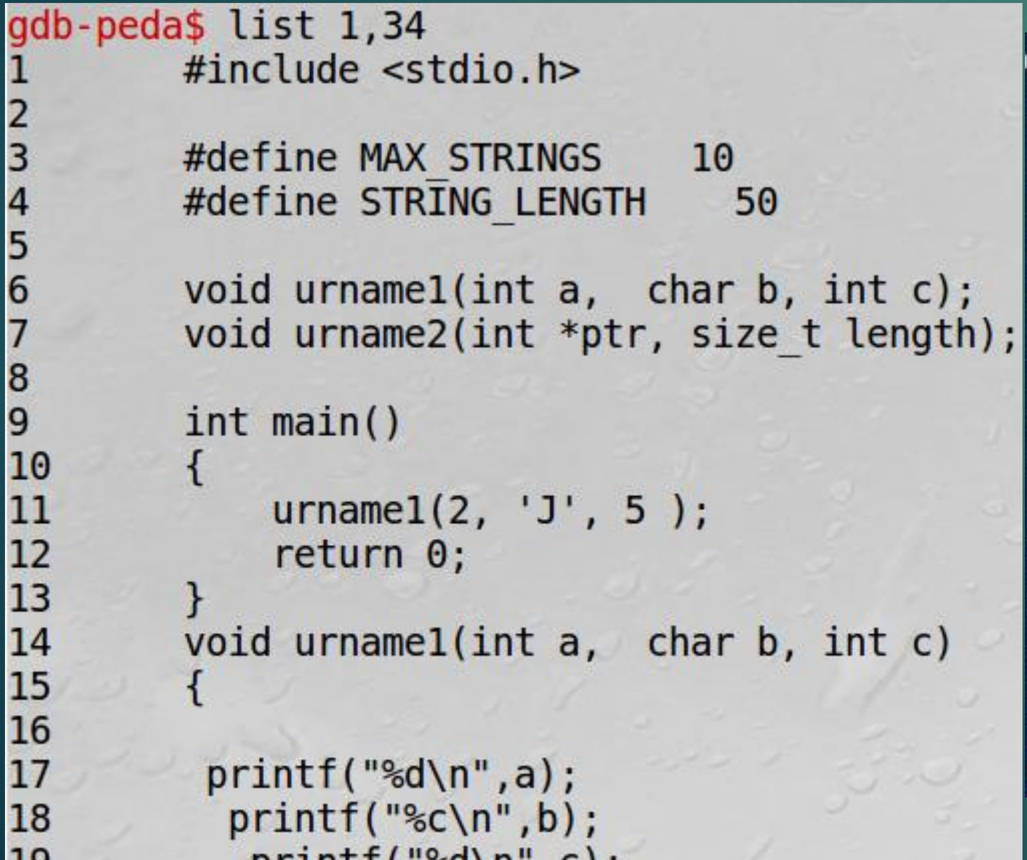
- The following command opens the program in gdb
  gdb stracing



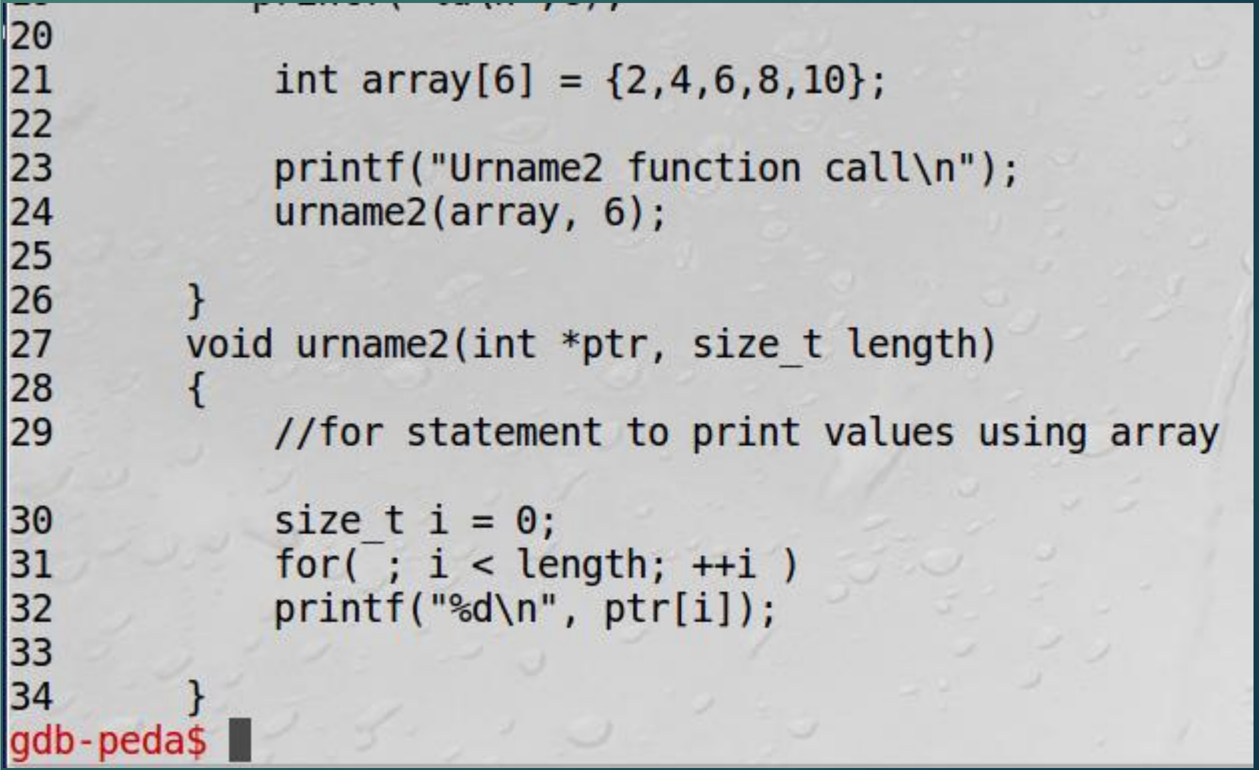
```
[04/23/19]seed@VM:~/Desktop$ gdb stracing
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.04) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/lice
s/gpl.html>
This is free software: you are free to change and redistribute
There is NO WARRANTY, to the extent permitted by law.  Type "sh
copying"[redacted]
and "show warranty" for details.
This GDB was configured as "i686-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".
```

View the source code in gdb

- Type the command to list the source code of the program:
  list 1,34



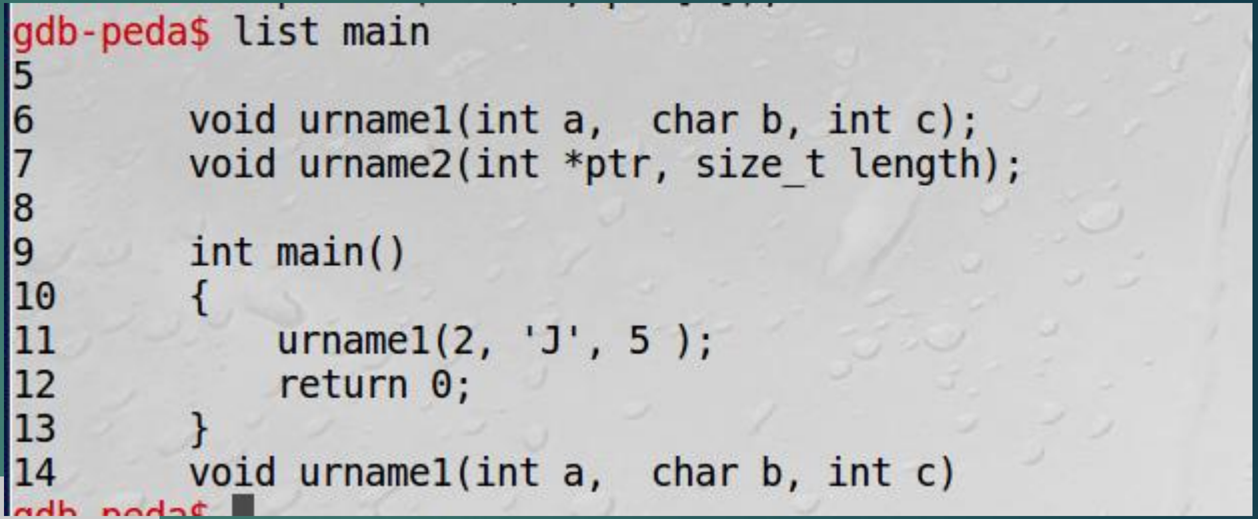
```
gdb-peda$ list 1,34
1       #include <stdio.h>
2
3       #define MAX_STRINGS    10
4       #define STRING_LENGTH    50
5
6       void urname1(int a,  char b, int c);
7       void urname2(int *ptr, size_t length);
8
9       int main()
10      {
11          urname1(2, 'J', 5 );
12          return 0;
13      }
14      void urname1(int a,  char b, int c)
15      {
16
17       printf("%d\n",a);
18        printf("%c\n",b);
19         printf("%d\n",c);
```



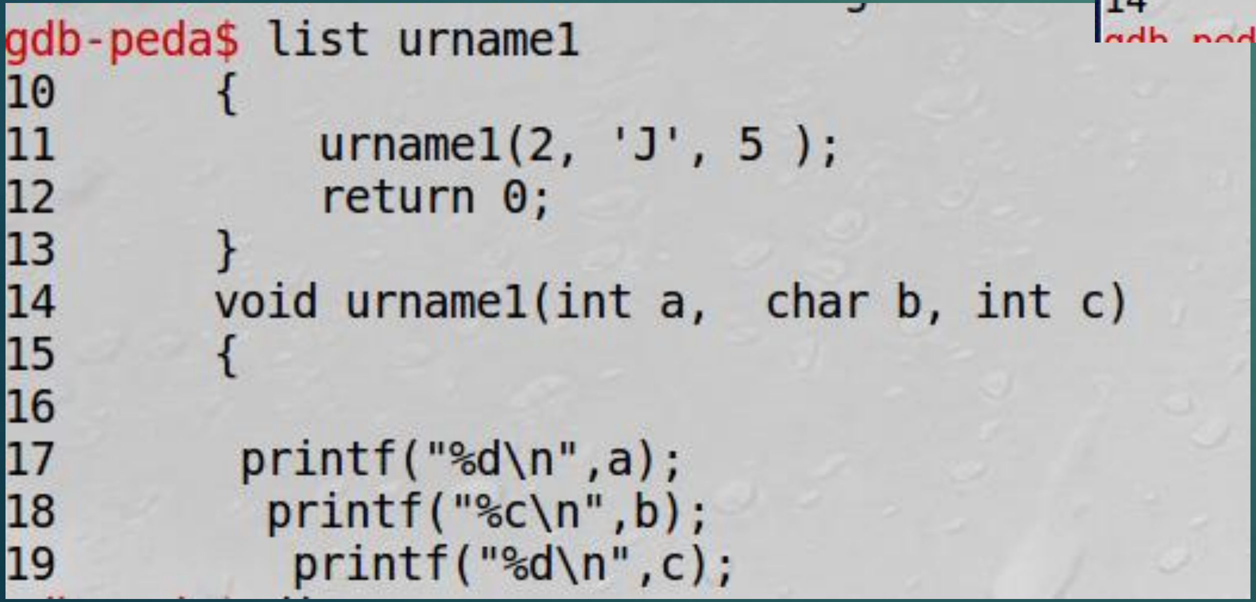
```
20
21              int array[6] = {2,4,6,8,10};
22
23              printf("Urname2 function call\n");
24              urname2(array, 6);
25
26          }
27          void urname2(int *ptr, size_t length)
28          {
29              //for statement to print values using array

30              size_t i = 0;
31              for( ; i < length; ++i )
32              printf("%d\n", ptr[i]);
33
34          }
gdb-peda$ 
```

To view and print function code

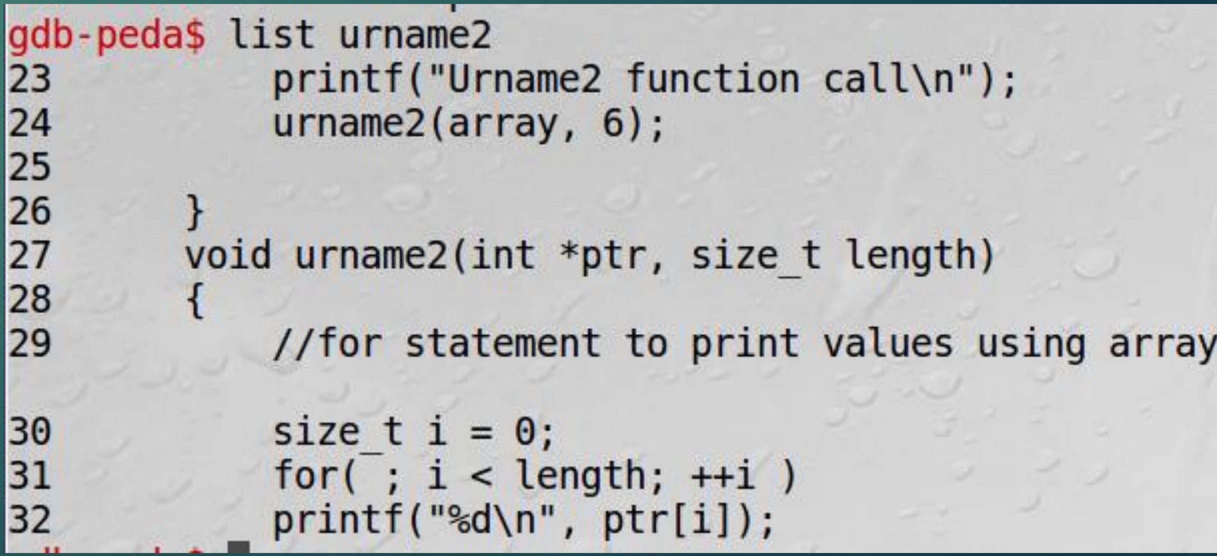
- Type < list functionName> like so, to view the contents of the function



```
gdb-peda$ list main
5
6       void urname1(int a,  char b, int c);
7       void urname2(int *ptr, size_t length);
8
9       int main()
10      {
11          urname1(2, 'J', 5 );
12          return 0;
13      }
14      void urname1(int a,  char b, int c)
```



```
gdb-peda$ list urname1
10      {
11          urname1(2, 'J', 5 );
12          return 0;
13      }
14      void urname1(int a,  char b, int c)
15      {
16
17       printf("%d\n",a);
18        printf("%c\n",b);
19         printf("%d\n",c);
```

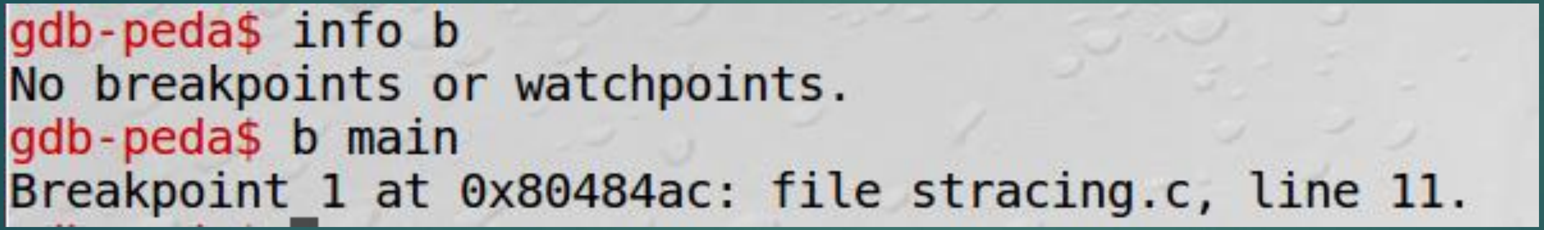


```
gdb-peda$ list urname2
23              printf("Urname2 function call\n");
24              urname2(array, 6);
25
26          }
27          void urname2(int *ptr, size_t length)
28          {
29              //for statement to print values using array

30              size_t i = 0;
31              for( ; i < length; ++i )
32              printf("%d\n", ptr[i]);
```

Check for break points

- Type < info b> to show break points, if any

- Type <b main> to see the breakpoints of main, it will be at 11 because urname2 function is called

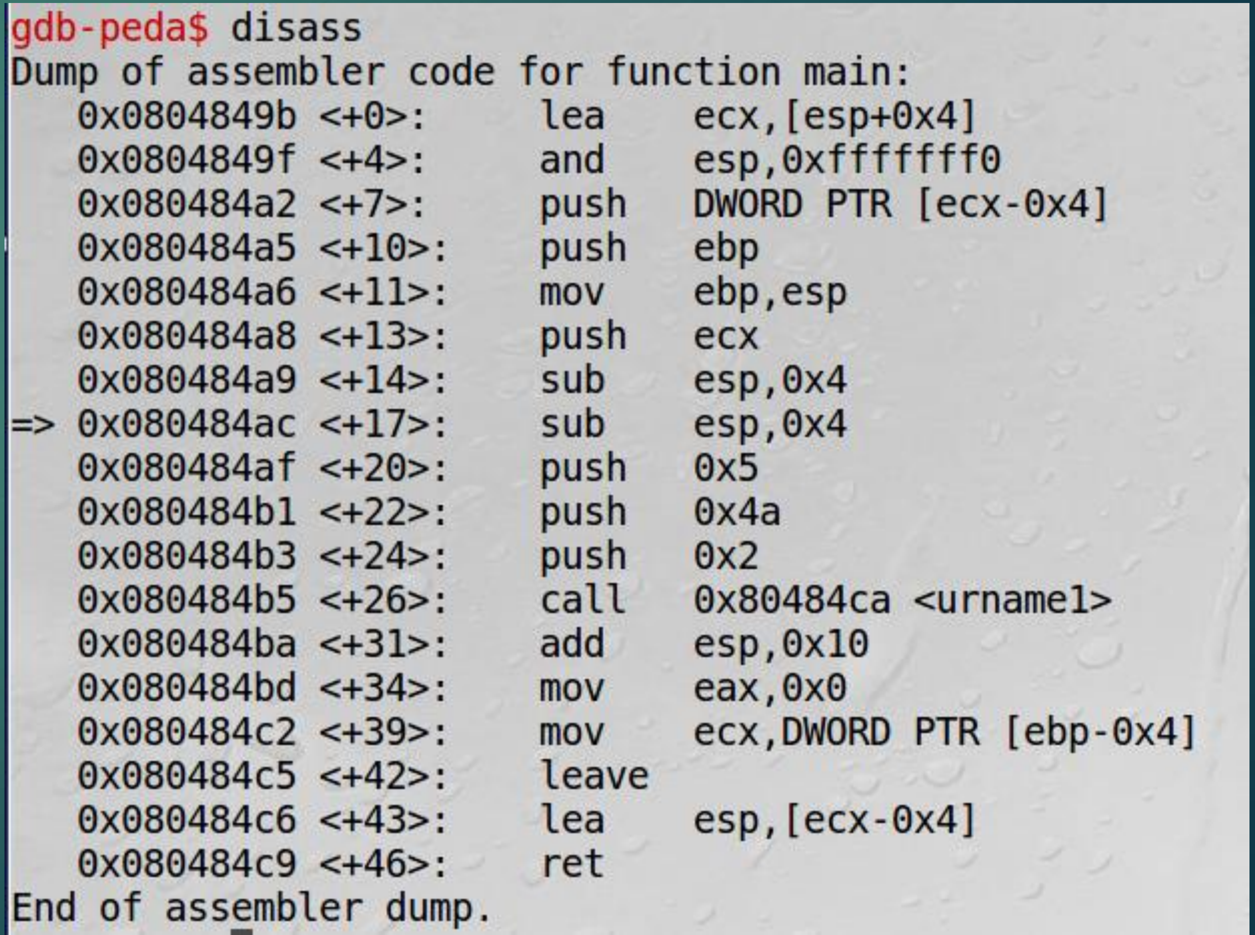


```
gdb-peda$ info b
No breakpoints or watchpoints.
gdb-peda$ b main
Breakpoint 1 at 0x80484ac: file stracing.c, line 11.
```

Showing the values in assembly language

- Type <disass> to show the assembly code, this is the equivalent of the source code.

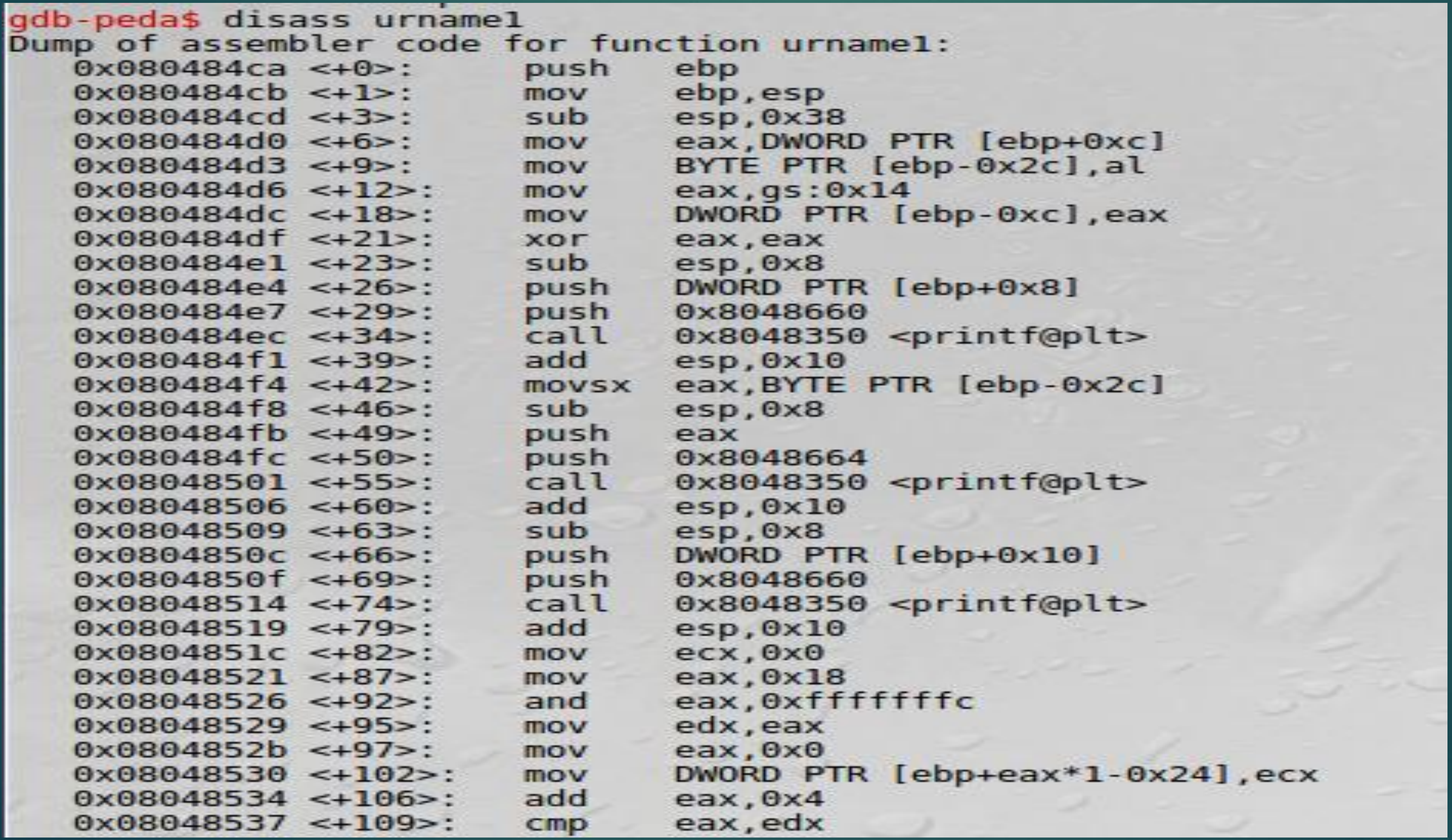
  The following shows the memory locations of the stacks as they go in sequence throughout the program:



```
gdb-peda$ disass
Dump of assembler code for function main:
   0x0804849b <+0>:     lea    ecx,[esp+0x4]
   0x0804849f <+4>:     and    esp,0xfffffff0
   0x080484a2 <+7>:     push   DWORD PTR [ecx-0x4]
   0x080484a5 <+10>:    push   ebp
   0x080484a6 <+11>:    mov    ebp,esp
   0x080484a8 <+13>:    push   ecx
   0x080484a9 <+14>:    sub    esp,0x4
=> 0x080484ac <+17>:    sub    esp,0x4
   0x080484af <+20>:    push   0x5
   0x080484b1 <+22>:    push   0x4a
   0x080484b3 <+24>:    push   0x2
   0x080484b5 <+26>:    call   0x80484ca <urname1>
   0x080484ba <+31>:    add    esp,0x10
   0x080484bd <+34>:    mov    eax,0x0
   0x080484c2 <+39>:    mov    ecx,DWORD PTR [ebp-0x4]
   0x080484c5 <+42>:    leave
   0x080484c6 <+43>:    lea    esp,[ecx-0x4]
   0x080484c9 <+46>:    ret
End of assembler dump.
```

Print assembly of urname1

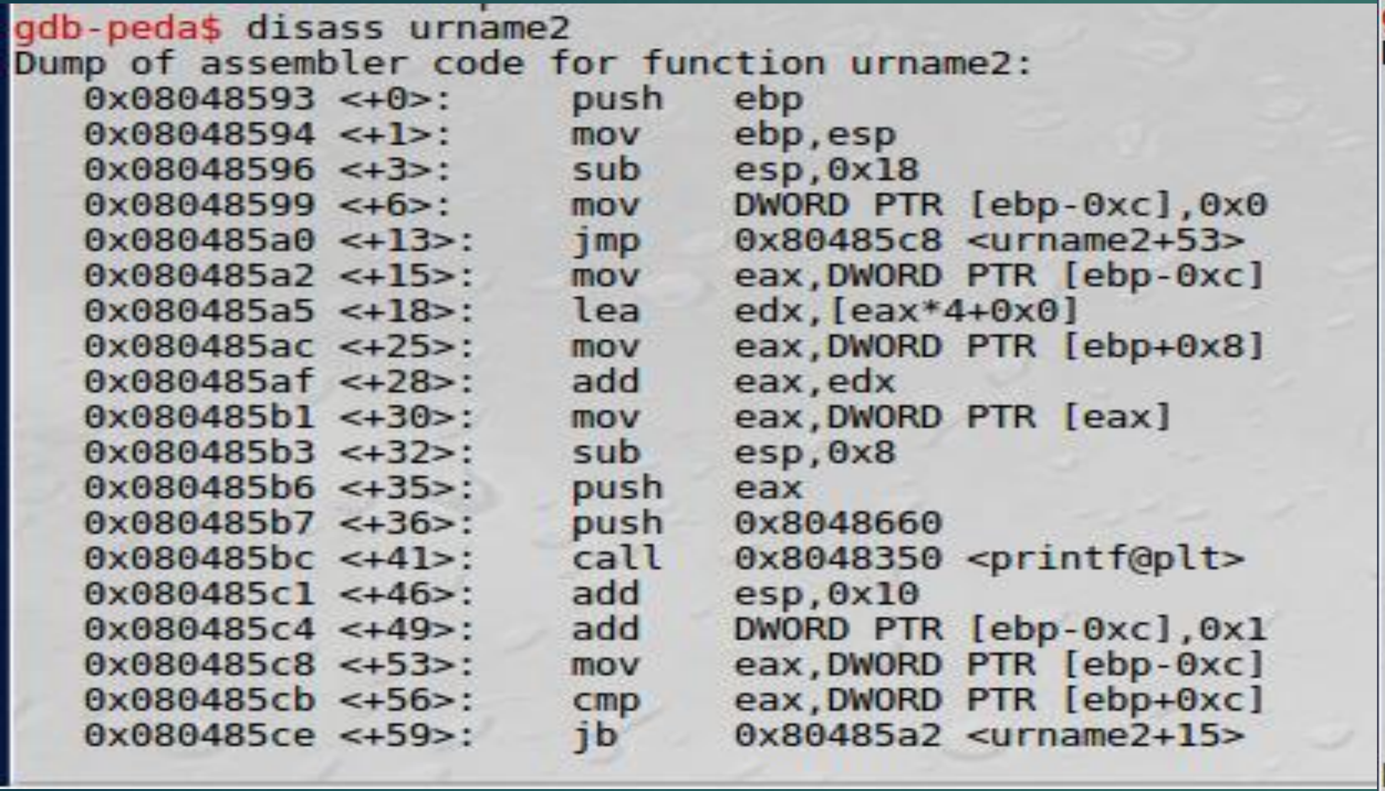
- Type <disass urname1>



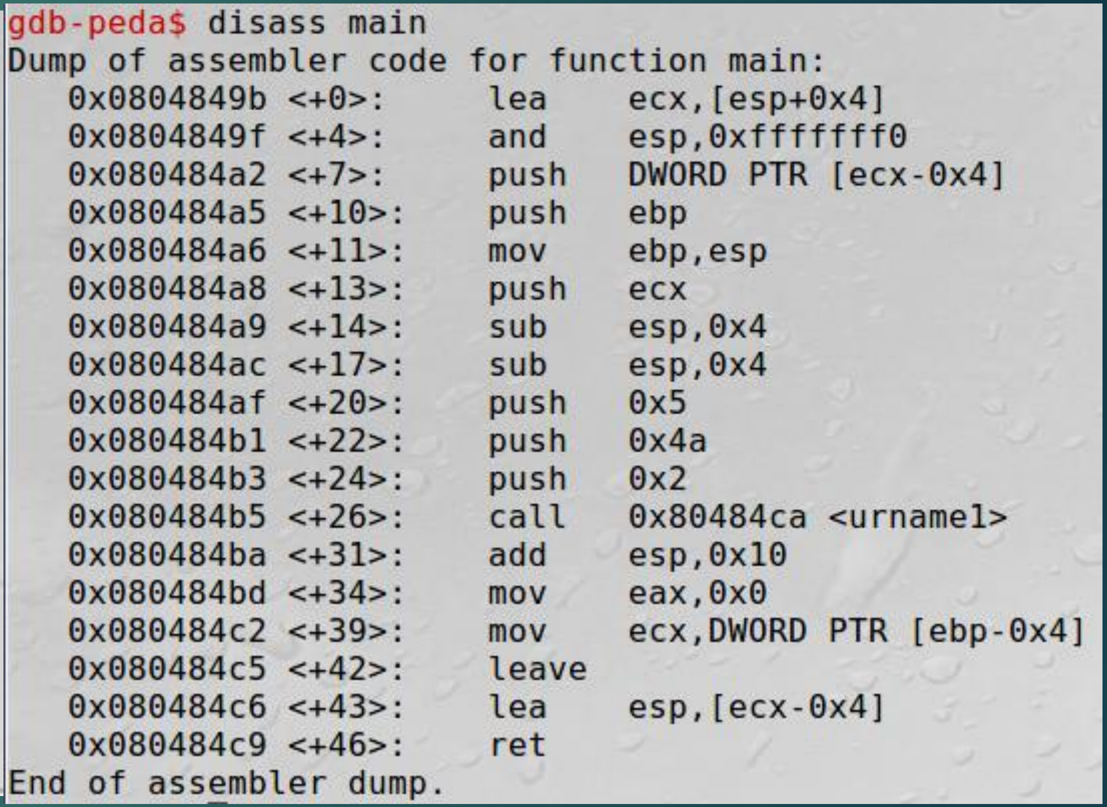
```
gdb-peda$ disass urname1
Dump of assembler code for function urname1:
   0x080484ca <+0>:     push   ebp
   0x080484cb <+1>:     mov    ebp,esp
   0x080484cd <+3>:     sub    esp,0x38
   0x080484d0 <+6>:     mov    eax,DWORD PTR [ebp+0xc]
   0x080484d3 <+9>:     mov    BYTE PTR [ebp-0x2c],al
   0x080484d6 <+12>:    mov    eax,gs:0x14
   0x080484dc <+18>:    mov    DWORD PTR [ebp-0xc],eax
   0x080484df <+21>:    xor    eax,eax
   0x080484e1 <+23>:    sub    esp,0x8
   0x080484e4 <+26>:    push   DWORD PTR [ebp+0x8]
   0x080484e7 <+29>:    push   0x8048660
   0x080484ec <+34>:    call   0x8048350 <printf@plt>
   0x080484f1 <+39>:    add    esp,0x10
   0x080484f4 <+42>:    movsx  eax,BYTE PTR [ebp-0x2c]
   0x080484f8 <+46>:    sub    esp,0x8
   0x080484fb <+49>:    push   eax
   0x080484fc <+50>:    push   0x8048664
   0x08048501 <+55>:    call   0x8048350 <printf@plt>
   0x08048506 <+60>:    add    esp,0x10
   0x08048509 <+63>:    sub    esp,0x8
   0x0804850c <+66>:    push   DWORD PTR [ebp+0x10]
   0x0804850f <+69>:    push   0x8048660
   0x08048514 <+74>:    call   0x8048350 <printf@plt>
   0x08048519 <+79>:    add    esp,0x10
   0x0804851c <+82>:    mov    ecx,0x0
   0x08048521 <+87>:    mov    eax,0x18
   0x08048526 <+92>:    and    eax,0xfffffffc
   0x08048529 <+95>:    mov    edx,eax
   0x0804852b <+97>:    mov    eax,0x0
   0x08048530 <+102>:   mov    DWORD PTR [ebp+eax*1-0x24],ecx
   0x08048534 <+106>:   add    eax,0x4
   0x08048537 <+109>:   cmp    eax,edx
```

Print assembly for urname2

- Type <disass urname2> likewise to print the assembly code for main, that is if you want to separate the code by function calls



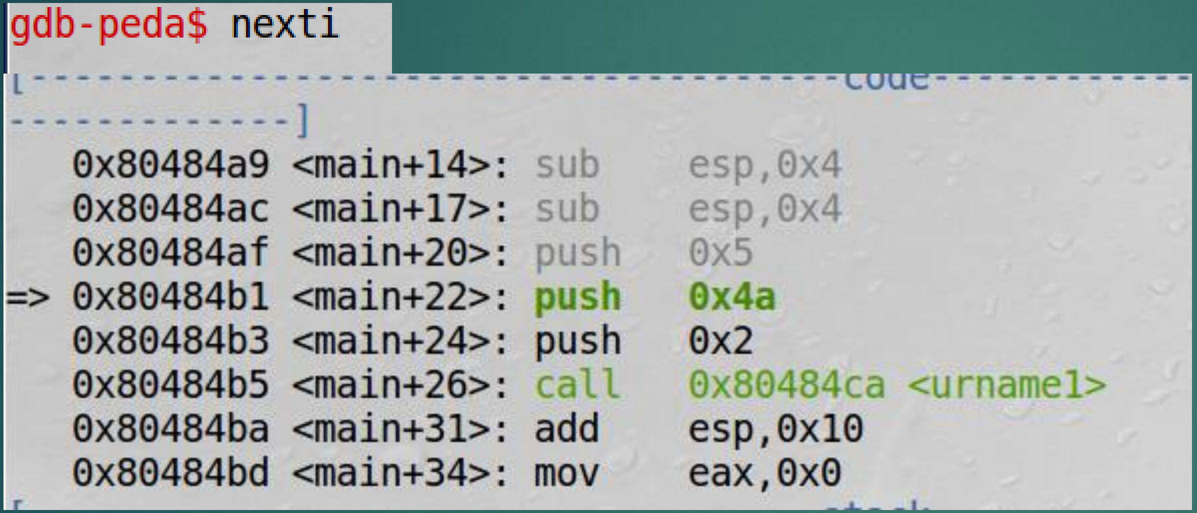
```
gdb-peda$ disass urname2
Dump of assembler code for function urname2:
   0x08048593 <+0>:     push   ebp
   0x08048594 <+1>:     mov    ebp,esp
   0x08048596 <+3>:     sub    esp,0x18
   0x08048599 <+6>:     mov    DWORD PTR [ebp-0xc],0x0
   0x080485a0 <+13>:    jmp    0x80485c8 <urname2+53>
   0x080485a2 <+15>:    mov    eax,DWORD PTR [ebp-0xc]
   0x080485a5 <+18>:    lea    edx,[eax*4+0x0]
   0x080485ac <+25>:    mov    eax,DWORD PTR [ebp+0x8]
   0x080485af <+28>:    add    eax,edx
   0x080485b1 <+30>:    mov    eax,DWORD PTR [eax]
   0x080485b3 <+32>:    sub    esp,0x8
   0x080485b6 <+35>:    push   eax
   0x080485b7 <+36>:    push   0x8048660
   0x080485bc <+41>:    call   0x8048350 <printf@plt>
   0x080485c1 <+46>:    add    esp,0x10
   0x080485c4 <+49>:    add    DWORD PTR [ebp-0xc],0x1
   0x080485c8 <+53>:    mov    eax,DWORD PTR [ebp-0xc]
   0x080485cb <+56>:    cmp    eax,DWORD PTR [ebp+0xc]
   0x080485ce <+59>:    jb     0x80485a2 <urname2+15>
```



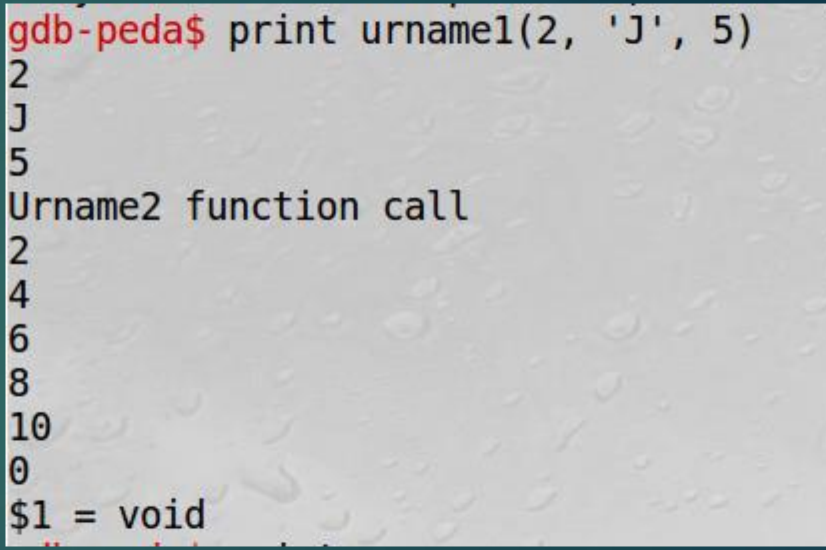
```
gdb-peda$ disass main
Dump of assembler code for function main:
   0x0804849b <+0>:     lea    ecx,[esp+0x4]
   0x0804849f <+4>:     and    esp,0xfffffff0
   0x080484a2 <+7>:     push   DWORD PTR [ecx-0x4]
   0x080484a5 <+10>:    push   ebp
   0x080484a6 <+11>:    mov    ebp,esp
   0x080484a8 <+13>:    push   ecx
   0x080484a9 <+14>:    sub    esp,0x4
   0x080484ac <+17>:    sub    esp,0x4
   0x080484af <+20>:    push   0x5
   0x080484b1 <+22>:    push   0x4a
   0x080484b3 <+24>:    push   0x2
   0x080484b5 <+26>:    call   0x80484ca <urname1>
   0x080484ba <+31>:    add    esp,0x10
   0x080484bd <+34>:    mov    eax,0x0
   0x080484c2 <+39>:    mov    ecx,DWORD PTR [ebp-0x4]
   0x080484c5 <+42>:    leave
   0x080484c6 <+43>:    lea    esp,[ecx-0x4]
   0x080484c9 <+46>:    ret
End of assembler dump.
```

Use nexti to move to the next instruction to be executed

- Type <nexti> to point to the next register and see an operation



```
gdb-peda$ nexti
[----------------------------------------code-------------
--------------]
   0x80484a9 <main+14>: sub    esp,0x4
   0x80484ac <main+17>: sub    esp,0x4
   0x80484af <main+20>: push   0x5
=> 0x80484b1 <main+22>: push   0x4a
   0x80484b3 <main+24>: push   0x2
   0x80484b5 <main+26>: call   0x80484ca <urname1>
   0x80484ba <main+31>: add    esp,0x10
   0x80484bd <main+34>: mov    eax,0x0
```



```
gdb-peda$ print urname1(2, 'J', 5)
2
J
5
Urname2 function call
2
4
6
8
10
0
$1 = void
```

- We can also see the value of the register by printing its contents,
  this will print the contents of the function call <print urname(2, ‘J’, 5)> Here the value of the urname function is $1