CSO ASSIGNMENT -2

QUESTION-3:

Computer	
Processor	11th Gen Intel(R) Core(TM) i5-11320H @ 3.20GHz
Memory	7875MB (1360MB used)
Operating System	Ubuntu 20.04.4 LTS
User Name	geethika (Geethika)

Operating System

Kernel	Linux 5.13.0-44-generic (x86_64)
Version	#49~20.04.1-Ubuntu SMP Wed May 18 18:44:28 UTC 2022
C Library	GNU C Library / (Ubuntu GLIBC 2.31-0ubuntu9.7) 2.31
Distribution	Ubuntu 20.04.4 LTS
Computer Name	geethika-Inspiron-14-5410

Kernel Modules

intel_powerclamp	Package Level C-state Idle Injection for Intel CPUs
coretemp	Intel Core temperature monitor

File Systems:

udev	/dev	0.00 % (3.7 GiB of 3.7 GiB)
tmpfs	/run	0.27 % (767.0 MiB of 769.1 MiB)

Processor:

4 Cores , 8 Threads

11th Gen Intel(R) Core(TM) i5-11320H @ 3.20GHz

Memory:

MemTotal	Total Memory	7875624 KiB

MemFree	Free Memory	4842536 KiB
MemAvailable		6210032 KiB

PCI DEVICES:

USB controller	Intel Corporation Device a0ed (rev 30) (prog-if 30 [XHCI])
RAM memory	Intel Corporation Device a0ef (rev 30)
Network controller	Intel Corporation Device a0f0 (rev 30)

USB DEVICES:

Linux Foundation 3.0 root hub

Sunplus Innovation Technology Inc. Integrated_Webcam_HD

Intel Corp.

Linux Foundation 2.0 root hub

Linux Foundation 3.0 root hub

Linux Foundation 2.0 root hub

BATTERY:
3-CELL BATTERY Backup of 6 hours

SENSORS:

../../BAT0/in0	Voltage	16.18V
../../nvme0/temp1	Temperature	31.85°C
../../nvme0/temp2	Temperature	31.85°C

STORAGE:

SSD: 512 GB

DMI:

Name	Inspiron 14 5410
Family	Inspiron
Vendor	Dell Inc. (Dell Computer, www.dell.com)

BENCH MARKS:
CPU ZLIB:

11th Gen Intel(R) Core(TM) i5-11320H @ 3.20GHz	8x 4500.00 MHz	1.57
PowerPC 740/750	1x 280.00 MHz	2150.60

GPU DRAWING:

11th Gen Intel(R) Core(TM) i5-11320H @ 3.20GHz	8x 4500.00 MHz	10976.66

QUESTION-4:

Given Assembly Code;

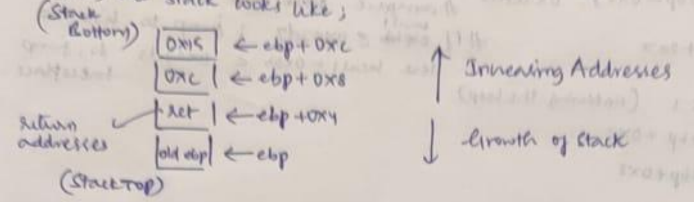
```
assemblycode:
    <+0>: push ebp
    <+1>: mov ebp,esp
    <+3>: sub esp,0x10
    <+6>: mov eax,DWORD PTR [ebp+0xc]
    <+9>: mov DWORD PTR [ebp-0x4],eax
    <+12>: mov eax,DWORD PTR [ebp+0x8]
    <+15>: mov DWORD PTR [ebp-0x8],eax
    <+18>: jmp 0x50c <asm2+31>
    <+20>: add DWORD PTR [ebp-0x4],0x1
    <+24>: add DWORD PTR [ebp-0x8],0xaf
    <+31>: cmp DWORD PTR [ebp-0x8],0xa3d3
    <+38>: jle 0x501 <asm2+20>
    <+40>: mov eax,DWORD PTR [ebp-0x4]
    <+43>: leave
    <+44>: ret
```

What does assembly code (0xc,0x15) return?

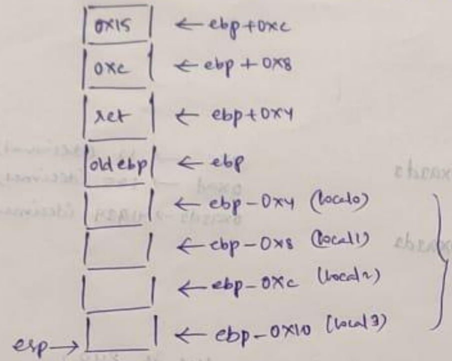
4. assembly code (0xc, 0x15) implies
push 0x15
push 0xc
Call assemblycode

<+0> : push ebp # pushes the Basepointer onto the Stack
<+1> : mov ebp ,esp # moves existing stackpointer (esp) into ebp → callee saved
(current stack position)

Now; the stack looks like ;



<+3> : sub esp, 0x10 # allocates 16 bytes of space on the stack



<+6> : mov eax, DWORD PTR [ebp + 0xc] # moving the value in the address (ebp + 0xc) into eax $\boxed{eax = 0x15}$

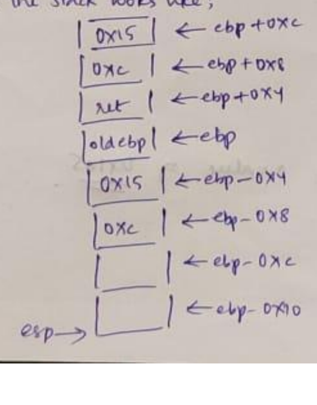
<+9> : mov DWORD PTR [ebp − 0x4], eax # moving contents in eax to [ebp − 0x4] $\boxed{local\ 0 = 0x15}$

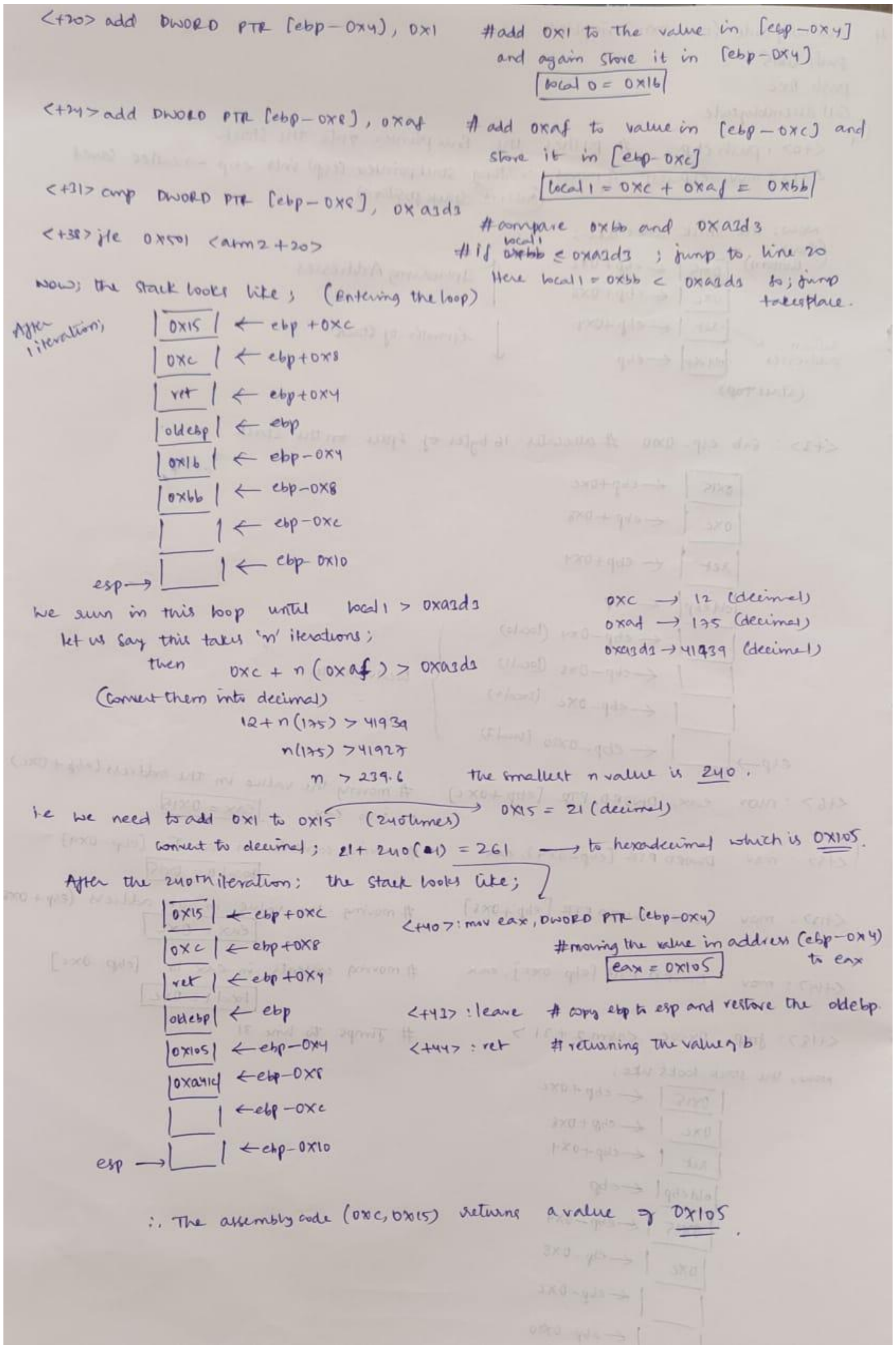
<+12> : mov eax, DWORD PTR [ebp + 0x8] # moving the value in the address (esp + 0x8) $\boxed{eax = 0xc}$

<+15> : mov DWORD PTR [ebp − 0xc], eax # moving contents in eax to [ebp − 0xc] $\boxed{local\ 1 = 0xc}$

<+18> : jmp 0x50c <asm 2 + 31> # Jumps to line 31

Now; The stack looks like ;





```
<+20> add DWORD PTR [ebp-0x4], 0x1      #add 0x1 to the value in [ebp-0x4]
                                         and again store it in [ebp-0x4]
                                         | local 0 = 0x16 |

<+24> add DWORD PTR [ebp-0xc], 0xaf     # add 0xaf to value in [ebp-0xc] and
                                         store it in [ebp-0xc]
                                         | local 1 = 0xc + 0xaf = 0xbb |

<+31> cmp DWORD PTR [ebp-0xc], 0xa3d3
                                         #compare 0xbb and 0xa3d3
<+38> jle 0x501 <arm2+20>               #if local1 ≤ 0xa3d3 ; jump to line 20
                                         Here local1 = 0xbb < 0xa3d3 so; jump
                                                                  takesplace.
```

Now; the stack looks like; (Entering the loop)

```
After
iteration;   | 0x15   | ← ebp +0xc
             | 0xc    | ← ebp+0x8
             | ret    | ← ebp+0x4
             | oldebp | ← ebp
             | 0x16   | ← ebp-0x4
             | 0xbb   | ← ebp-0x8
             |        | ← ebp-0xc
      esp →  |        | ← ebp-0x10
```

We run in this loop until local1 > 0xa3d3

let us say this takes 'n' iterations;

then 0xc + n(0xaf) > 0xa3d3

0xc → 12 (decimal)
0xaf → 175 (decimal)
0xa3d3 → 41939 (decimal)

(Convert them into decimal)

$12 + n(175) > 41939$

$n(175) > 41927$

$n > 239.6$ the smallest n value is $\underline{240}$.

i.e we need to add 0x1 to 0x15 (240times) → 0x15 = 21 (decimal)

convert to decimal; 21 + 240(1) = 261 ⟶ to hexadecimal which is $\underline{\underline{0x105}}$.

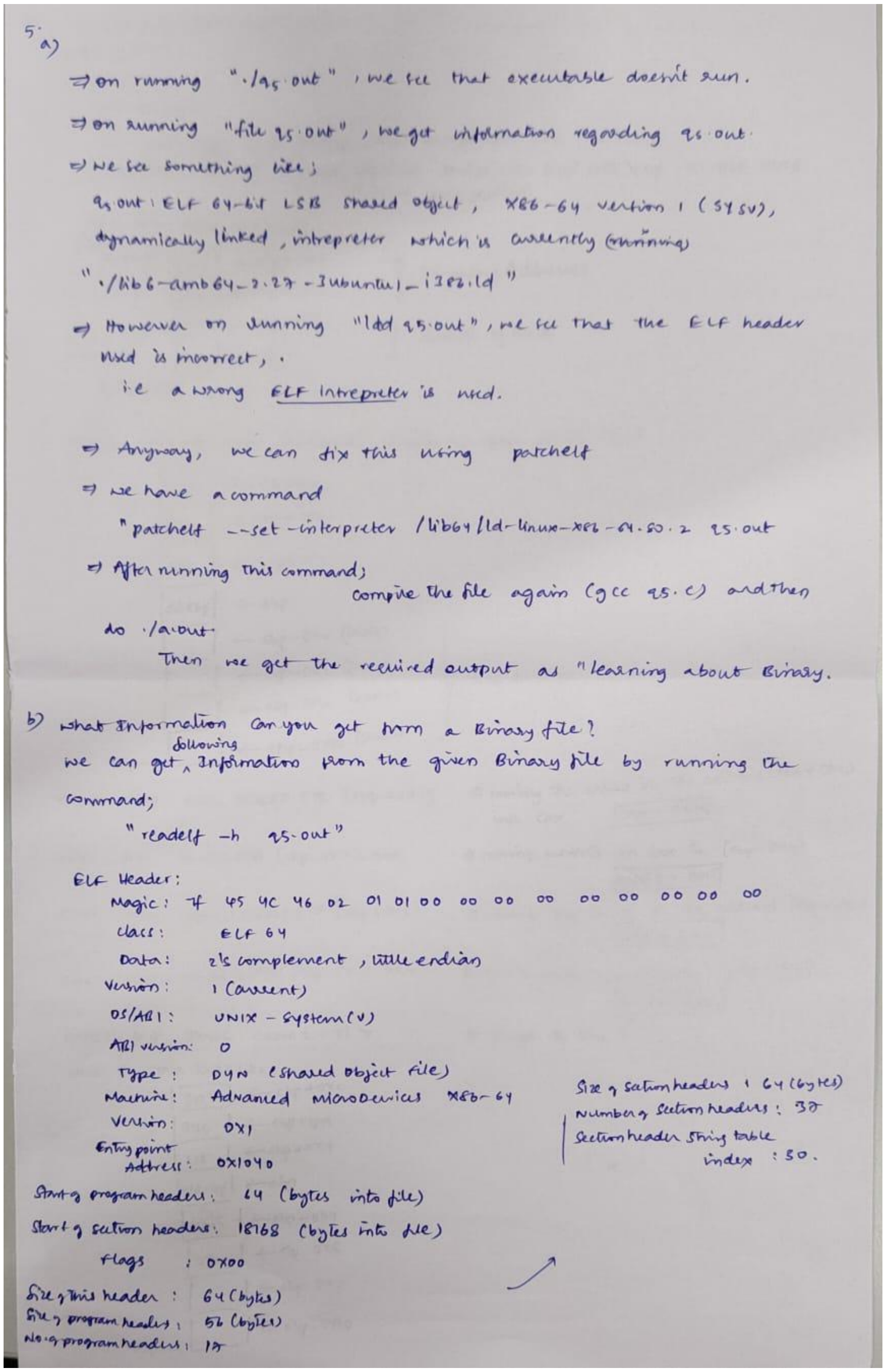
After the 240th iteration; the stack looks like;

```
             | 0x15   | ← ebp+0xc       <+40>: mov eax, DWORD PTR [ebp-0x4]
             | 0xc    | ← ebp+0x8               #moving the value in address (ebp-0x4)
             | ret    | ← ebp+0x4               | eax = 0x105 |          to eax
             | oldebp | ← ebp           <+43> : leave   # copy ebp to esp and restore the oldebp.
             | 0x105  | ← ebp-0x4       <+44> : ret     # returning the value of b
             | 0xa41c | ← ebp-0x8
             |        | ← ebp-0xc
      esp →  |        | ← ebp-0x10
```

∴ The assembly code (0xc, 0x15) returns a value → $\underline{\underline{0x105}}$.

The assembly code(0xc,0x15) returns a value of 0x105.

QUESTION-5:



5.
a)

⇒ on running "./q5.out", we see that executable doesn't run.

⇒ on running "file q5.out", we get information regarding q5.out.

⇒ we see something like;

q5.out : ELF 64-bit LSB shared object, x86-64 version 1 (SYSV), dynamically linked, intrepreter which is currently (running) "./lib6-amb64-2.27-3ubuntu1-i386.ld"

⇒ However on running "ldd q5.out", we see that the ELF header used is incorrect, .

i.e a wrong ELF Intrepreter is used.

⇒ Anyway, we can fix this using patchelf

⇒ we have a command

"patchelf --set-interpreter /lib64/ld-linux-x86-64.so.2 q5.out

⇒ After running this command;

compile the file again (gcc q5.c) and then do ./a.out.

Then we get the required output as "learning about Binary.

b) what Information can you get from a Binary file?

we can get following Information from the given Binary file by running the command;

"readelf -h q5.out"

```
ELF Header:
    Magic:  7f 45 4C 46 02 01 01 00 00 00 00 00 00 00 00 00
    class:          ELF 64
    Data:           2's complement, little endian
    Version:        1 (current)
    OS/ABI:         UNIX - System (V)
    ABI version:    0
    Type:           DYN (shared object file)
    Machine:        Advanced MicroDevices x86-64
    Version:        0x1
    Entry point
      Address:      0x1040
Start of program headers:  64 (bytes into file)
Start of section headers:  18168 (bytes into file)
    Flags           : 0x00
Size of this header    : 64 (bytes)
Size of program headers : 56 (bytes)
No. of program headers : 13
Size of section headers : 64 (bytes)
Number of section headers : 37
Section header string table
                index : 30.
```