Report Lab-3

Chandrakant Pradhan(22b1023)

September 2023

Contents

1 Question-1

1.1 Program-1

Sequences that are accepted by the program:
This program takes input as a sequence of numbers and then prints ”yes” if the numbers are in arithmetic sequence and prints ”no” otherwise.

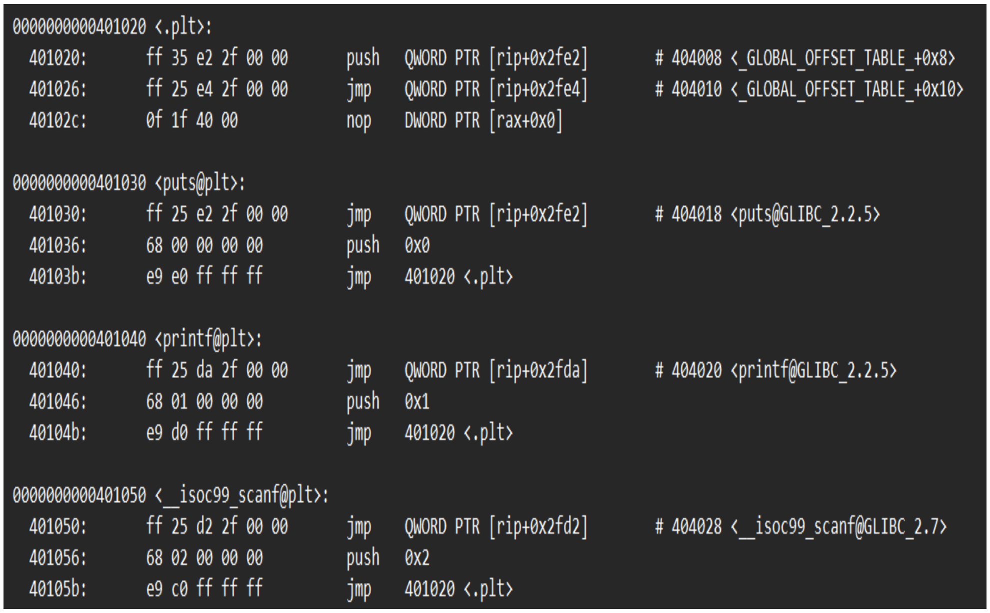
1.1.1 How I reached this conclusion:

In order to analyze the binary file we first disassemble it by using the Linux command:
objdump -DM intel program1 > output.asm

Now we see many sections, following are the important section that tells us what the program actually does:

- .plt
- .main

1.1.2 .plt

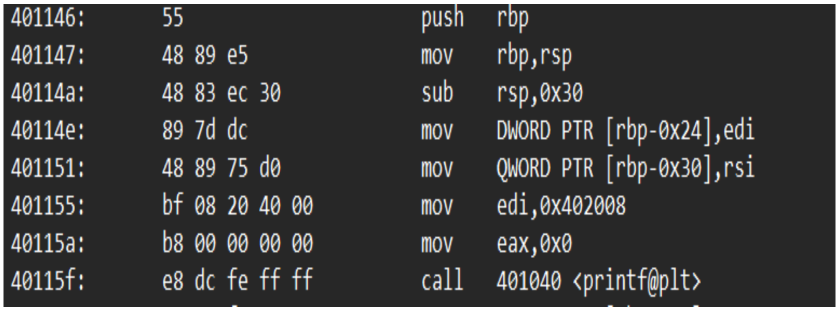


```
0000000000401020 <.plt>:
  401020:       ff 35 e2 2f 00 00       push   QWORD PTR [rip+0x2fe2]        # 404008 <_GLOBAL_OFFSET_TABLE_+0x8>
  401026:       ff 25 e4 2f 00 00       jmp    QWORD PTR [rip+0x2fe4]        # 404010 <_GLOBAL_OFFSET_TABLE_+0x10>
  40102c:       0f 1f 40 00             nop    DWORD PTR [rax+0x0]

0000000000401030 <puts@plt>:
  401030:       ff 25 e2 2f 00 00       jmp    QWORD PTR [rip+0x2fe2]        # 404018 <puts@GLIBC_2.2.5>
  401036:       68 00 00 00 00          push   0x0
  40103b:       e9 e0 ff ff ff          jmp    401020 <.plt>

0000000000401040 <printf@plt>:
  401040:       ff 25 da 2f 00 00       jmp    QWORD PTR [rip+0x2fda]        # 404020 <printf@GLIBC_2.2.5>
  401046:       68 01 00 00 00          push   0x1
  40104b:       e9 d0 ff ff ff          jmp    401020 <.plt>

0000000000401050 <__isoc99_scanf@plt>:
  401050:       ff 25 d2 2f 00 00       jmp    QWORD PTR [rip+0x2fd2]        # 404028 <__isoc99_scanf@GLIBC_2.7>
  401056:       68 02 00 00 00          push   0x2
  40105b:       e9 c0 ff ff ff          jmp    401020 <.plt>
```

The role of this block is that it provides the c function **puts**, **scanf**, and **printf** to be used in the code. First, we jump to the Global Offset Table (GOT) entry at a particular address like **[rip+0x2fe2]** which stores the required function and then pushes a value like, **0x0** onto the stack. This is typically used for passing arguments to functions. Then we again jump to .plt.

1.1.3 .main

1.1.4 code before printing

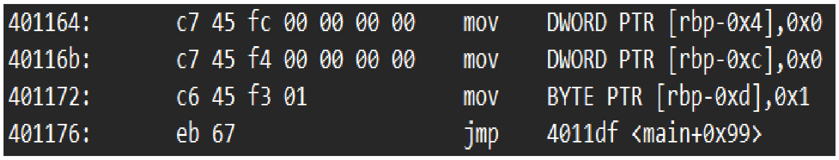


```
401146:      55                   push   rbp
401147:      48 89 e5             mov    rbp,rsp
40114a:      48 83 ec 30          sub    rsp,0x30
40114e:      89 7d dc             mov    DWORD PTR [rbp-0x24],edi
401151:      48 89 75 d0          mov    QWORD PTR [rbp-0x30],rsi
401155:      bf 08 20 40 00       mov    edi,0x402008
40115a:      b8 00 00 00 00       mov    eax,0x0
40115f:      e8 dc fe ff ff       call   401040 <printf@plt>
```

First rbp is pushed into the stack, then the stack pointer address is stored in it, then the stack pointer is moved down to allocate 48 bytes of memory, this must be some kind of space to hold the input numbers. Then the value stored in registers edi and rsi are stored at positions [**rbp-0x24**] and [**rbp-0x30**] respectively.

Then edi is loaded with an address of formatted string ”Enter three or more numbers (Terminate with CTRL+D):” and eax is loaded with zero so that the string stored in edi is printed when printf is called. Then printf is called and the string is printed.

1.1.5 code after printing



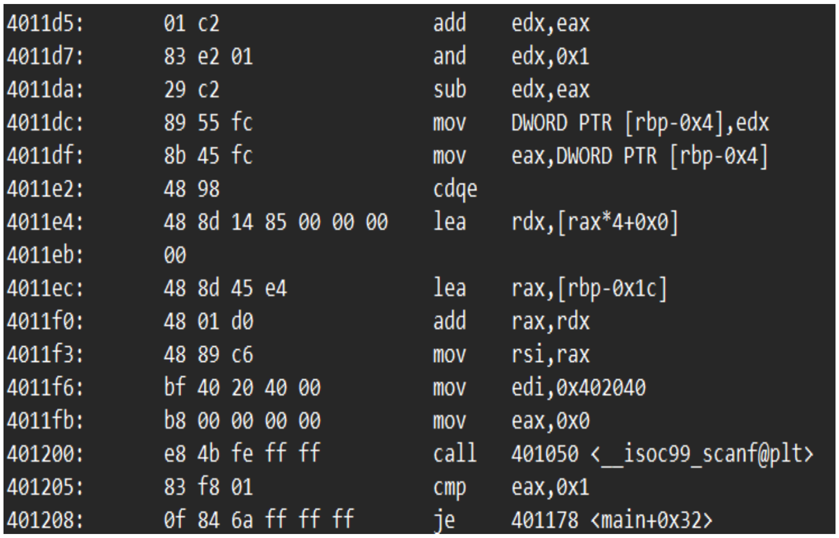
```
401164:      c7 45 fc 00 00 00 00    mov    DWORD PTR [rbp-0x4],0x0
40116b:      c7 45 f4 00 00 00 00    mov    DWORD PTR [rbp-0xc],0x0
401172:      c6 45 f3 01             mov    BYTE PTR [rbp-0xd],0x1
401176:      eb 67                   jmp    4011df <main+0x99>
```

Then we store three variables and initialise them

- [**rbp-0x4**] initialised to 0 (consider this as some kind of offset o)
- [**rbp-0xc**] initialised to 0 (consider this as the number of inputs given call s)
- [**rbp-0xd**] initialized to 1 (consider this as a flag call f)

The reason for the above variables would be understood later.

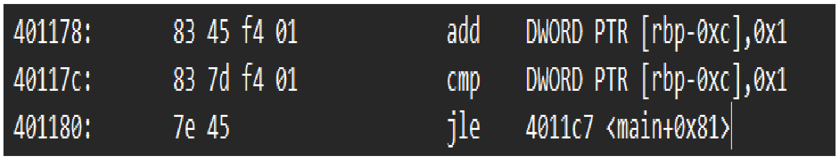
1.1.6 Block 4011df



```
4011d5:       01 c2                   add    edx,eax
4011d7:       83 e2 01                and    edx,0x1
4011da:       29 c2                   sub    edx,eax
4011dc:       89 55 fc                mov    DWORD PTR [rbp-0x4],edx
4011df:       8b 45 fc                mov    eax,DWORD PTR [rbp-0x4]
4011e2:       48 98                   cdqe
4011e4:       48 8d 14 85 00 00 00    lea    rdx,[rax*4+0x0]
4011eb:       00
4011ec:       48 8d 45 e4             lea    rax,[rbp-0x1c]
4011f0:       48 01 d0                add    rax,rdx
4011f3:       48 89 c6                mov    rsi,rax
4011f6:       bf 40 20 40 00          mov    edi,0x402040
4011fb:       b8 00 00 00 00          mov    eax,0x0
401200:       e8 4b fe ff ff          call   401050 <__isoc99_scanf@plt>
401205:       83 f8 01                cmp    eax,0x1
401208:       0f 84 6a ff ff ff       je     401178 <main+0x32>
```

Then we jump to the address 4011df. There **[rbp-0x4]** i.e offset is loaded in eax, then it is sign extended so that rax = eax , then rdx = rax*4 and rax is loaded with a new quantity **[rbp-0x1c]** which holds the address of previous input that was stored in the buffer. Then rax becomes rax + rdx which is equivalent to saying that **rax = previous address + offset*4**, we would soon see how offset works. Then the updated address is loaded in rsi and edi is loaded with some address and eax is loaded with zero to make the syscall and then scanf is called which would store the input in the updated address in the buffer. Now if the input given to the scanf was a number then eax becomes 1 and it jumps to then address 401178 which is the main logic of the code , else eax would be zero and we go ahead and check the further code.

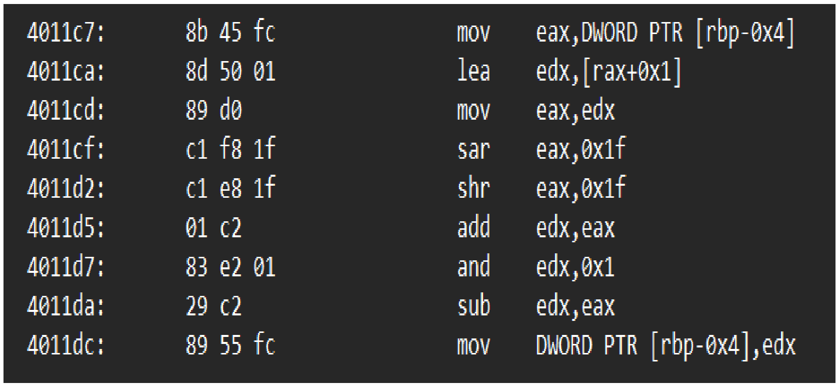
1.1.7 Block 401178



```
401178:       83 45 f4 01             add    DWORD PTR [rbp-0xc],0x1
40117c:       83 7d f4 01             cmp    DWORD PTR [rbp-0xc],0x1
401180:       7e 45                   jle    4011c7 <main+0x81>
```

This is the code at 401178. This increments the **[rbp-0xc]** i.e our size variable s by 1 as we got an input as a number and then check if the size variable was greater than 1, if not then it jumps to a very interesting piece of code, the address 4011c7, else it continues the further code.

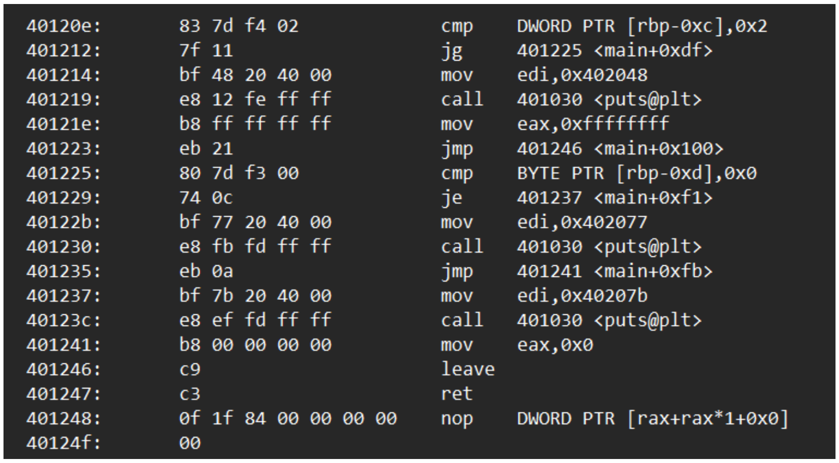
1.1.8 Block 4011c7 : Increment logic



```
4011c7:     8b 45 fc          mov    eax,DWORD PTR [rbp-0x4]
4011ca:     8d 50 01          lea    edx,[rax+0x1]
4011cd:     89 d0             mov    eax,edx
4011cf:     c1 f8 1f          sar    eax,0x1f
4011d2:     c1 e8 1f          shr    eax,0x1f
4011d5:     01 c2             add    edx,eax
4011d7:     83 e2 01          and    edx,0x1
4011da:     29 c2             sub    edx,eax
4011dc:     89 55 fc          mov    DWORD PTR [rbp-0x4],edx
```

First the quantity **[rbp-0x4]** i.e offset is loaded in eax, then edx = eax + 1, then eax = edx then the following two lines basically make eax = 1 if it was odd, else if it was even then eax becomes 0. Then edx = eax + edx i.e edx = edx + (0 or 1) then edx and 1 is done which check the parity of edx and then edx = edx - eax and then this edx is loaded in **[rbp-0x4]** as the new offset. If analyzed carefully we can say that if the initial offset was zero then after this step it becomes 1, and if it was 1 initially then offset becomes zero. I call this block of code as the **increment logic**. After this again we go to block 4011df and take input the number and if ctrl D is not given then we again go to block 401178 and it goes on.We can see that this is forming a loop, now when ctrl D is hit we don't go to 401178, instead we move ahead and do the following:

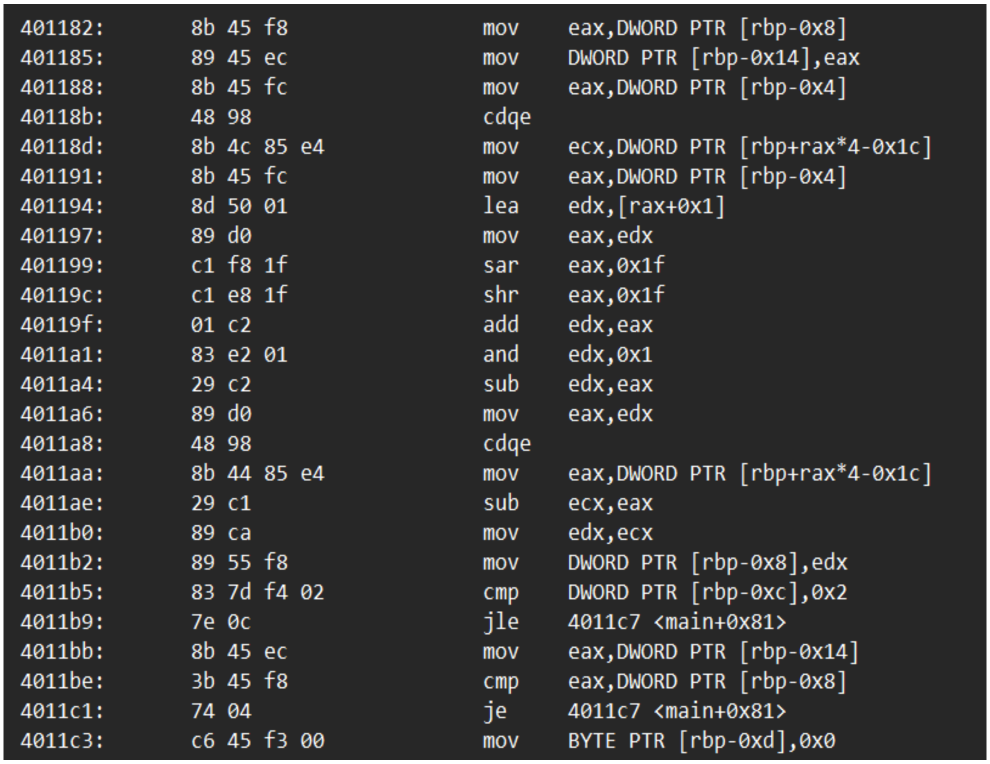
1.1.9 Exit logic



```
40120e:     83 7d f4 02             cmp    DWORD PTR [rbp-0xc],0x2
401212:     7f 11                   jg     401225 <main+0xdf>
401214:     bf 48 20 40 00          mov    edi,0x402048
401219:     e8 12 fe ff ff          call   401030 <puts@plt>
40121e:     b8 ff ff ff ff          mov    eax,0xffffffff
401223:     eb 21                   jmp    401246 <main+0x100>
401225:     80 7d f3 00             cmp    BYTE PTR [rbp-0xd],0x0
401229:     74 0c                   je     401237 <main+0xf1>
40122b:     bf 77 20 40 00          mov    edi,0x402077
401230:     e8 fb fd ff ff          call   401030 <puts@plt>
401235:     eb 0a                   jmp    401241 <main+0xfb>
401237:     bf 7b 20 40 00          mov    edi,0x40207b
40123c:     e8 ef fd ff ff          call   401030 <puts@plt>
401241:     b8 00 00 00 00          mov    eax,0x0
401246:     c9                      leave
401247:     c3                      ret
401248:     0f 1f 84 00 00 00 00    nop    DWORD PTR [rax+rax*1+0x0]
40124f:     00
```

Here first [**rbp-0xc**] is checked, if number of entry is greater than two and we already know that ctrl d is pressed then it goes to 401225 where it compares the flag to 0, if the flag is zero then it prints no and makes eax =0 and then leave and return, else if flag is 1 then it prints yes, makes eax = 0 and then leave and return. If number of entry less than two then print the string ”Enter more than three numbers”

1.1.10 Block after 401178 : main logic



```
401182:       8b 45 f8                mov    eax,DWORD PTR [rbp-0x8]
401185:       89 45 ec                mov    DWORD PTR [rbp-0x14],eax
401188:       8b 45 fc                mov    eax,DWORD PTR [rbp-0x4]
40118b:       48 98                   cdqe
40118d:       8b 4c 85 e4             mov    ecx,DWORD PTR [rbp+rax*4-0x1c]
401191:       8b 45 fc                mov    eax,DWORD PTR [rbp-0x4]
401194:       8d 50 01                lea    edx,[rax+0x1]
401197:       89 d0                   mov    eax,edx
401199:       c1 f8 1f                sar    eax,0x1f
40119c:       c1 e8 1f                shr    eax,0x1f
40119f:       01 c2                   add    edx,eax
4011a1:       83 e2 01                and    edx,0x1
4011a4:       29 c2                   sub    edx,eax
4011a6:       89 d0                   mov    eax,edx
4011a8:       48 98                   cdqe
4011aa:       8b 44 85 e4             mov    eax,DWORD PTR [rbp+rax*4-0x1c]
4011ae:       29 c1                   sub    ecx,eax
4011b0:       89 ca                   mov    edx,ecx
4011b2:       89 55 f8                mov    DWORD PTR [rbp-0x8],edx
4011b5:       83 7d f4 02             cmp    DWORD PTR [rbp-0xc],0x2
4011b9:       7e 0c                   jle    4011c7 <main+0x81>
4011bb:       8b 45 ec                mov    eax,DWORD PTR [rbp-0x14]
4011be:       3b 45 f8                cmp    eax,DWORD PTR [rbp-0x8]
4011c1:       74 04                   je     4011c7 <main+0x81>
4011c3:       c6 45 f3 00             mov    BYTE PTR [rbp-0xd],0x0
```

Here first [**rbp-0x8**] which stores the difference of previous two inputs(we would see how) is loaded in eax and then eax is loaded to [**rbp-0x14**](consider this as a temp var that stores the previous difference), now eax is loaded with [**rbp-0x4**] the current offset and then sign extension so rax = eax, now the line **DWORD PTR** [**rbp+rax*4-0x1c**] says that ecx holds some value stored in array corresponding to current offset, this must be our new number.
Now again eax is loaded with current offset and a bunch of same calculation are done which are same as in block 4011c7, which just inverts the offset, so at line 4011a6 eax holds the inverted offset and then it is sign extended to rax then the eax is loaded with **DWORD PTR** [**rbp+rax*4-0x1c**] which would be the number corresponding to the previous offset, so we can say that this was the

previous number stored.
Finally the difference between ecx and eax is made and stored in **[rbp-0x8]** which would be the new difference.
Now if the number of elements are greater than 2 then the values of the previous difference and current difference are compared, if they are the same then jump to increment block else the flag becomes 0. If the number of elements were less than or equal to zero then we would have just jumped to the increment block.

1.1.11 Conclusion

In this program as we are inputting the numbers, the previous difference was maintained and current difference is calculated, if they were equal then the flag remained 1, because it was initialized with value 1, if the difference was not same then flag become 0 for rest of the program and when we hit ctrl-D if the flag was 1 ”Yes” was printed else ”No” was printed.

1.2 Program-2

1.2.1 What is the function?

The function that was given in program 2 takes a number as an input, if the number is 0 then it returns 1, else if a number n is given then it returns

$$F(n) = \sum_{k=1}^{n} F(k-1) * F(n-k) \tag{1}$$

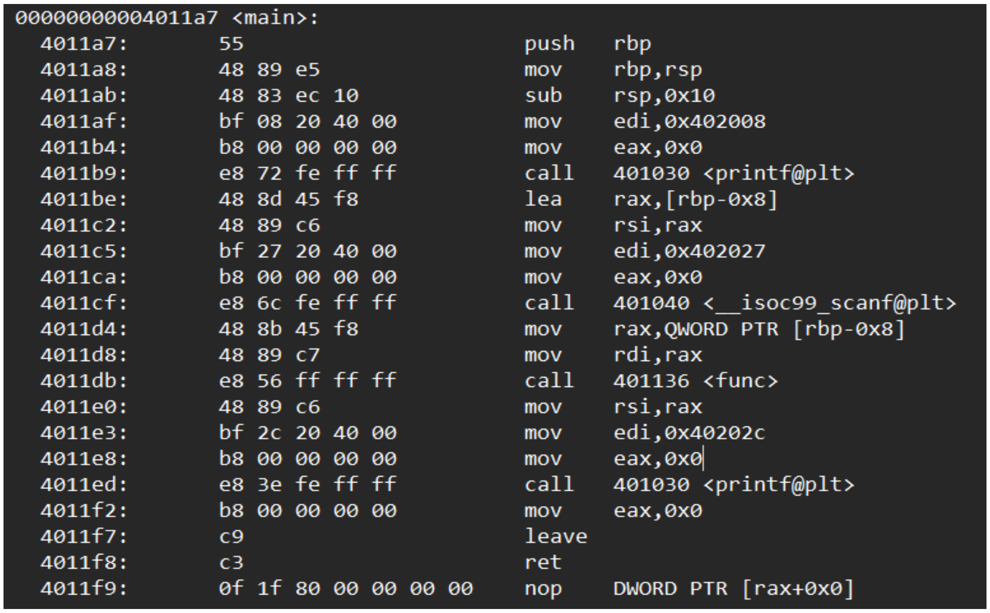
1.2.2 How I got to this result ?

Similar to previous question we first disassemble the program and again we look at important parts of code. Again there is a .plt section which is exact same as the previous program , then there are two important block of codes :

- func
- main

We would first analyze the main program and then analyze the func portion.

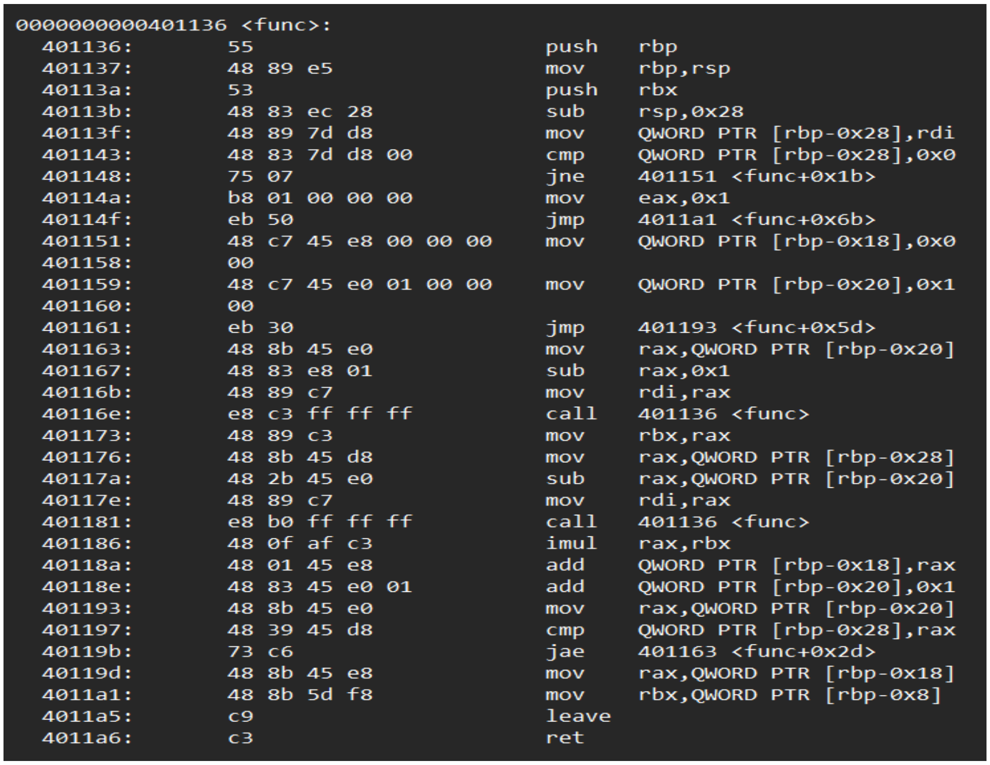
1.2.3 Analysis of Main



```
00000000004011a7 <main>:
  4011a7:       55                      push   rbp
  4011a8:       48 89 e5                mov    rbp,rsp
  4011ab:       48 83 ec 10             sub    rsp,0x10
  4011af:       bf 08 20 40 00          mov    edi,0x402008
  4011b4:       b8 00 00 00 00          mov    eax,0x0
  4011b9:       e8 72 fe ff ff          call   401030 <printf@plt>
  4011be:       48 8d 45 f8             lea    rax,[rbp-0x8]
  4011c2:       48 89 c6                mov    rsi,rax
  4011c5:       bf 27 20 40 00          mov    edi,0x402027
  4011ca:       b8 00 00 00 00          mov    eax,0x0
  4011cf:       e8 6c fe ff ff          call   401040 <__isoc99_scanf@plt>
  4011d4:       48 8b 45 f8             mov    rax,QWORD PTR [rbp-0x8]
  4011d8:       48 89 c7                mov    rdi,rax
  4011db:       e8 56 ff ff ff          call   401136 <func>
  4011e0:       48 89 c6                mov    rsi,rax
  4011e3:       bf 2c 20 40 00          mov    edi,0x40202c
  4011e8:       b8 00 00 00 00          mov    eax,0x0
  4011ed:       e8 3e fe ff ff          call   401030 <printf@plt>
  4011f2:       b8 00 00 00 00          mov    eax,0x0
  4011f7:       c9                      leave
  4011f8:       c3                      ret
  4011f9:       0f 1f 80 00 00 00 00    nop    DWORD PTR [rax+0x0]
```

Here first the rbp, the base pointer is pushed into the stack and then the rsp, stack pointer is loaded into the rbp then the stack pointer is decremented, creating a 16 bit space where variables could be stored.
Then edi is loaded with an address of a formatted string and eax is loaded with 0 and then printf is called which print ”Enter a non-negative integer:” to the terminal.Now the address **[rbp-0x8]** is stored in rax and then rsi is loaded with rax , then edi and eax are loaded with some constants for the syscall, then the scanf is called **which would store the inputted value inside the address that is stored in rsi i.e at the address [rbp-0x8]**.Then the value from this address is again moved to the register rdi and the function is called, this effectively means that the inputted number is given as argument to this function.
Now the value that would be returned by the function would be stored at rax. Again move rax into rsi, load the constants inside edi and eax and then printf is called which would print the value inside rsi , then eax would be made zero then return to the main.

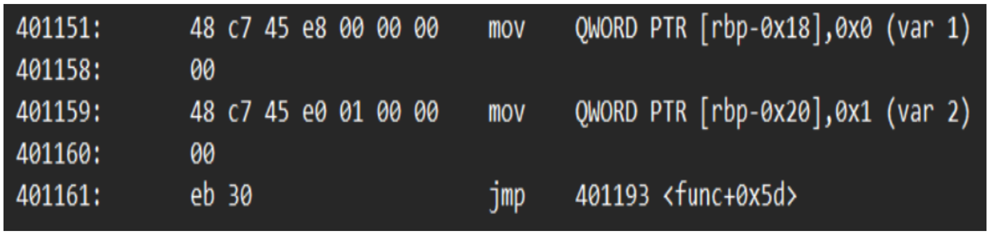
1.2.4 Analysis of Func



```
0000000000401136 <func>:
  401136:       55                      push   rbp
  401137:       48 89 e5                mov    rbp,rsp
  40113a:       53                      push   rbx
  40113b:       48 83 ec 28             sub    rsp,0x28
  40113f:       48 89 7d d8             mov    QWORD PTR [rbp-0x28],rdi
  401143:       48 83 7d d8 00          cmp    QWORD PTR [rbp-0x28],0x0
  401148:       75 07                   jne    401151 <func+0x1b>
  40114a:       b8 01 00 00 00          mov    eax,0x1
  40114f:       eb 50                   jmp    4011a1 <func+0x6b>
  401151:       48 c7 45 e8 00 00 00    mov    QWORD PTR [rbp-0x18],0x0
  401158:       00
  401159:       48 c7 45 e0 01 00 00    mov    QWORD PTR [rbp-0x20],0x1
  401160:       00
  401161:       eb 30                   jmp    401193 <func+0x5d>
  401163:       48 8b 45 e0             mov    rax,QWORD PTR [rbp-0x20]
  401167:       48 83 e8 01             sub    rax,0x1
  40116b:       48 89 c7                mov    rdi,rax
  40116e:       e8 c3 ff ff ff          call   401136 <func>
  401173:       48 89 c3                mov    rbx,rax
  401176:       48 8b 45 d8             mov    rax,QWORD PTR [rbp-0x28]
  40117a:       48 2b 45 e0             sub    rax,QWORD PTR [rbp-0x20]
  40117e:       48 89 c7                mov    rdi,rax
  401181:       e8 b0 ff ff ff          call   401136 <func>
  401186:       48 0f af c3             imul   rax,rbx
  40118a:       48 01 45 e8             add    QWORD PTR [rbp-0x18],rax
  40118e:       48 83 45 e0 01          add    QWORD PTR [rbp-0x20],0x1
  401193:       48 8b 45 e0             mov    rax,QWORD PTR [rbp-0x20]
  401197:       48 39 45 d8             cmp    QWORD PTR [rbp-0x28],rax
  40119b:       73 c6                   jae    401163 <func+0x2d>
  40119d:       48 8b 45 e8             mov    rax,QWORD PTR [rbp-0x18]
  4011a1:       48 8b 5d f8             mov    rbx,QWORD PTR [rbp-0x8]
  4011a5:       c9                      leave
  4011a6:       c3                      ret
```

First the base pointer is loaded in the stack, rsp address is loaded into rbp , also rbx is pushed in the stack, then rsp is decremented to store 40 bits of variables , then the input value which is stored in rdi is loaded in address [**rbp-0x28**] , then if the input value is 0 then program skips jump to 401151 and makes eax = 1 then jumps to 4011a1 where rbx is loaded with [**rbp-0x8**] and then leave and return , this eax value would then be 1 which makes rax value to be 1 and then the main prints the value inside rax.Else if the input is not 0 then jump to address 401151. Now would analyse block 401151.

1.2.5 Block 401151

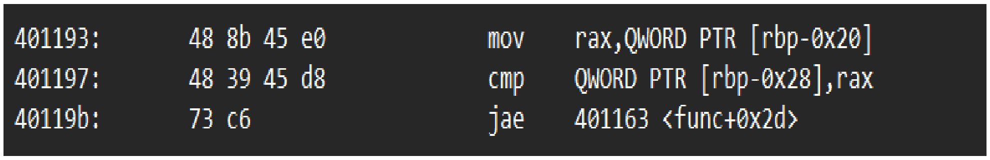


```
401151:       48 c7 45 e8 00 00 00    mov    QWORD PTR [rbp-0x18],0x0 (var 1)
401158:       00
401159:       48 c7 45 e0 01 00 00    mov    QWORD PTR [rbp-0x20],0x1 (var 2)
401160:       00
401161:       eb 30                   jmp    401193 <func+0x5d>
```

Following things happens:

- Value 0 is stored in address [**rbp-0x18**] (call this variable ans)
- Value 1 is stored in address [**rbp-0x20**] (call this variable i)
- jump to 401193 address

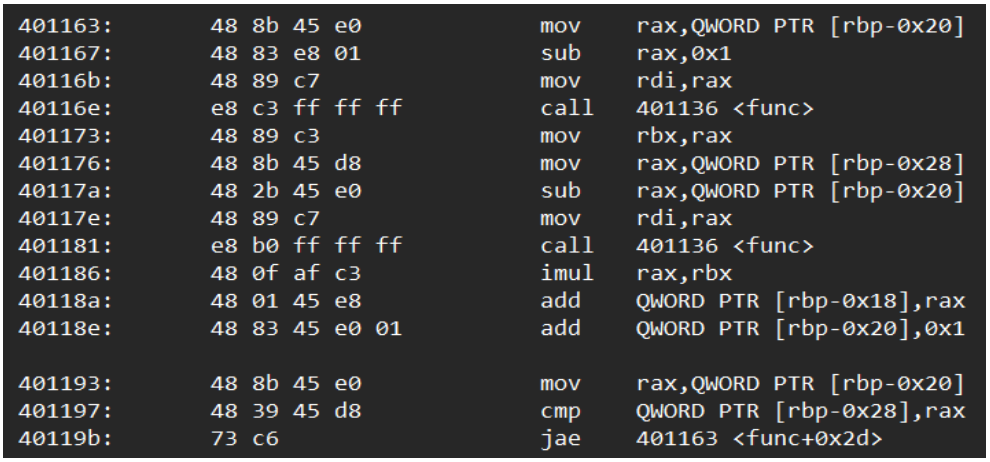
1.2.6 Block 401193



```
401193:       48 8b 45 e0             mov    rax,QWORD PTR [rbp-0x20]
401197:       48 39 45 d8             cmp    QWORD PTR [rbp-0x28],rax
40119b:       73 c6                   jae    401163 <func+0x2d>
```

- rax is loaded with [**rbp-0x20**]
- If its value is less than or equal to input value then jump to 401163
- Else continue for the termination of the code

1.2.7 Block 401163



```
401163:       48 8b 45 e0              mov    rax,QWORD PTR [rbp-0x20]
401167:       48 83 e8 01              sub    rax,0x1
40116b:       48 89 c7                 mov    rdi,rax
40116e:       e8 c3 ff ff ff           call   401136 <func>
401173:       48 89 c3                 mov    rbx,rax
401176:       48 8b 45 d8              mov    rax,QWORD PTR [rbp-0x28]
40117a:       48 2b 45 e0              sub    rax,QWORD PTR [rbp-0x20]
40117e:       48 89 c7                 mov    rdi,rax
401181:       e8 b0 ff ff ff           call   401136 <func>
401186:       48 0f af c3              imul   rax,rbx
40118a:       48 01 45 e8              add    QWORD PTR [rbp-0x18],rax
40118e:       48 83 45 e0 01           add    QWORD PTR [rbp-0x20],0x1

401193:       48 8b 45 e0              mov    rax,QWORD PTR [rbp-0x20]
401197:       48 39 45 d8              cmp    QWORD PTR [rbp-0x28],rax
40119b:       73 c6                    jae    401163 <func+0x2d>
```

Following things happen:

- First rax = i and then rax = rax - 1 and rdi = rax
- Then control flow goes to 4001136 to the start of function with argument rdi = i-1
- Then the value returned to rax is stored in rbx
- Then rax is loaded with [**rbp-0x28**] i.e rax = input and then rax = rax-i i.e rax = input - i
- Then again the functio is called and its value is stored in rax.
- Then rax = rax * rbx then ans = ans + rax i.e ans = ans + f(i-1) * f(input -i)
- Then i = i + 1
- Then again block 401193 is executed if the value of i is less than or equal to input value , if yes then again loop is done else we exit the loop

1.2.8 Conclusion

- If the value given to function is 0 then it returns 1
- If given value n then it loops from 1 to n and in each iteration it adds f(i-1)*f(n-i) to the ans where i is the iterator and ans is the value to be returned
- Hence we got the answer what the function does