CS231, DLDCA, Lab 3 (Part 1)

Sanoj S Vijendra, 22B0916

Contents

1 Question 1(a)

Q :

- This program asks you to enter three or more numbers, and accepts only certain sequences of numbers.
- Your task is to find out what sequences are accepted.

Ans :

```
0000000000401146 <main>:
  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>
  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>
```

Here, first we see that on line number *40115f*, $<$ *printf@plt* $>$ command is given. This will invoke the printf (output) function. After that, we observe that the values of registers *[rbp-0x4], [rbp-0xc], [rbp-0xd]* are initialised to $0, 0$ and 1 respectively (note that *[rbp-0x4]* means address which have offset of -4 from rbp). Now a jump occurs and program moves to the line number *4011df*.

Now, lets move to the part where $<\_$ *isoc99_scanf@plt* $>$ is called. This is used to scan the input given by the user. It return output 1 to *eax* if a number gets scanned i.e. input is given .

```
  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>
```

Lets, ignore initial lines and focus on last three lines only. Here, program checks if value stored in *eax* is 1 or not. If it is 1 (i.e. some input is given) then program will jump to line number *401178* else it will continue. This will form a loop.

```
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>
```

If no input is given then program compare value stored in *[rbp-0xc]*(we'll analyze later what this stores) with 2 and if value of *[rbp-0xc]* is greater than 2 then it moves to line number *401225*, else it calls the function $< puts@plt >$.
We had left what happens if *eax* returns 1 above. So, lets see from line number *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>
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]
```

Here, we can see that 1 is added to the value stored in the *[rbp-0xc]* each time $< \_\_isoc99\_scanf@plt >$ is called and input is given. So, we can conclude that value in *[rbp-0xc]* stores the number of terms entered by the user.
Now, a comparision is done between 1 and *[rbp-0xc]*. Next line says that, if value stored in *[rbp-0xc]* is less than or equal to 1 then it will jump to line number *4011c7*, else program continues as it is. Lets, assume for time being that value was greater than 1. Then, value stored at *[rbp-0x8]* will be moved to *eax* which will be then moved to *[rbp-0x14]*. In cut-short we can say that value of *[rbp-0x8]* is copied to *[rbp-0x14]*. Also, *[rbp-0x4]* was set to 0 initially. So, *eax* is re-stored to 0.

```
40118b: 48 98                 cdqe   \\ change eax to rax (16bit t0 32 bit)
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    \\ arithmetic right shift with first 31 digits
40119c: c1 e8 1f              shr    eax,0x1f   \\ logical right shift with first 31 bits repla
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
```

This complete set of instructions stores the numbers entered, in an array and finally at the last step takes their difference. Lets, see. An array with base pointer is present at *[rbp-0x1c]* and it stores the input values provided and previous difference. *[rbp-0x4]* stores the current index number(wich is 0 or 1 or 2). This index number is copied to *eax*. Now, *edx* is loaded with previous *eax* (converted to *rax*)+1. Then sar and shr are operated on *eax* which means now it stores only msb of *edx*. After operating next 4 lines, we'll get that *edx* has its lsb(which is 1 as eax was either 0 or 1 or 2) in it which is then loaded to *eax*. This eax is then converted to *rax*.
So, next declared *eax* stores the next number in it. Also, *ecx* had stored previous number in itself. Now, difference of *ecx* and *eax* is taken. Then, this difference is moved to *[rbp-0x8]*.

```
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
```

First, two line says if number of numbers entered is less than or equal to 2 then jump to line number *4011c7* will occur. Then *[rbp-0x14]* (which contains previous difference) and is compared with current difference. If they are unequal then 0 is loaded to *[rbp-0xd]*. (We had seen above that *[rbp-0x8]* is copied to *[rbp-0x14]*, line number *401182*). *[rbp-0xd]*'s value will be used by the program during producing output.

From these analysis, we can conclude that the program is running in an loop until - (i) number of numbers entered are less than 3, (ii) *[rbp-0x14]* and *[rbp-0x8]* are not equal. So, after n-loops, *[rbp-0x8]* will store the difference of n^{th} and $(n-1)^{th}$ terms and *[rbp-0x14]* will store difference of $(n-1)^{th}$ and $(n-2)^{th}$ terms. And these two differences are checked if they are equal or not. If, they are not equal then value of *[rbp-0xd]* is set to 0 which means sequence is no longer an A.P. So, we can conclude that program is checking whether entered sequence is an A.P. or not.
Also, by running the program we observe that:-

- if numbers entered are in A.P. the program will continue unless it is terminated by *Ctrl+D*.
- if less than 3 input is given then promgram says " *You have not entered enough numbers, try again*".
- if sequence is AP then output is *YES*, else *NO*.

2 Question 1(b)

Q :

- This program asks you to input a number (say x), and calculates and displays func(x).
- func(x) is a recursive function.
- Your task is to find the recurrence relation it satisfies (optionally, you can also find a closed form expression for func(x)).

Ans :

```
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>
```

Here, we see that on line number *4011b9*, $< printf@plt >$ is called which will invoke the output string. Next, we observe that on line number *4011cf*, $< \_\_isoc99\_scanf@plt >$ is called. This is used to take input from the user. The input is stored in *[rbp-0x8]* which is copied to *rax* which is then copied to *rdi*. So, in cut-short input is stored in *rdi* register.
After, this program calls, function $< func >$. We'll later see what does this $< func >$ do. After that $< printf@plt$ is called again and this time it will be used to display the result.
Now, lets move to the $< func >$ part.

```
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>
.
.
.
```

```
4011a1: 48 8b 5d f8           mov    rbx,QWORD PTR [rbp-0x8]
4011a5: c9                    leave
4011a6: c3                    ret
```

Here, input *rdi* is copied to *[rpb-0x28]* and it is compared to 0. If *rdi* is not 0 then it will jump to line number *401151*, else program will continue. Lets, consider for time being that *rdi* is 0. Then, value 1 is copied to *eax*, then jump occurs to line number *4011a1*. Line number *4011a1* to *4011a6* is used to return the value '1'. This is the base case for the recursion.
Now, lets see what happens if *rdi* is not equal to 0.

```
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
```

We can see that the value in the register *[rbp-0x18]* is initialised to 0. Also, value in register *[rbp-0x20]* is initialised to 1. Now, a jump occurs to line number *401193*.
Here, *[rbp-0x20]* is copied to *rax*. Then, it is compared to *[rbp-0x28]*, which stores the value of input given to the function $< func >$. If *[rbp-0x28]* is greater or equal to *[rbp-0x20]* then a jump to line number *401163* will occur, else value of *[rbp-0x18]* will be copied to *rax*. So, lets see what happens if jump occurs.
[rbp-0x20] is loaded in *rax*. Then value of *rax* is decreased by 1 and this value is loaded back to *rdi*, which will be used as input for the following $< func >$ call. Value returned by this function will be stored in *rax* itself. So, to use this value later, *rax* is copied to *rbx*. Then, the original input *[rbp-0x28]* is copied into *rax* and value of *[rbp-0x20]* is subtracted. Then again value of *rax* is copied to *rdi* to give input to following $< func >$ call. Value returned by this will also be stored in *rax*. Now, the values in *rax* and *rbx* are multiplied. This result is then added to *[rbp-0x18]* which was initially initialised to 0. Also, value of *[rbp-0x20]* is increased by 1. After this, again we'll move to top of this paragraph with new values in each register.
In nutshell, this section is following a *for loop*, where *[rbp-0x20]* is initialised to 1 and is incremented by 1 in each loop. Loop exits when its value becomes greater than the value of *[rbp-0x20]* (which stores initial input value for $< func >$). Inside the loop, function are recursively called with their input values as "*[rbp-0x20]* $- 1$" and "*[rbp-0x28]* $-$ *[rbp-0x20]*" respectively. Results returned by these two functions are then multiplied, and value of *[rbp-0x18]* is incremented by this product.

Finally, when loop condition is over-ruled then *[rbp-0x18]* is loaded in *rax* which is returned by the $< func >$. In short, what loop does is :-

```
for (int k = 1; k <= n; ++k)
{
    p = p + f(n - k) * f(k - 1);
}
```

So, overall we can conclude that the given program returns 1 as the answer if input is 0 or else it returns a recursive function f. So, overall this function will be given as :-

$$f(x) = \begin{cases} 1 & \text{if } x = 0 \\ f(x-1) \cdot f(0) + f(n-2) \cdot f(1) + ... + f(0) \cdot f(n-1) & \text{if } x \geq 1 \end{cases} \tag{1}$$

This recursive relation is very famous and its result is well known *Catalan Number*. So, the function becomes:

$$f(x) = \begin{cases} 1 & \text{if } x = 0 \\ \frac{\binom{2x}{x}}{x+1} & \text{if } x \geq 1 \end{cases} \tag{2}$$