Lab 2: Reserving Lab Report

20200977 Han Youngtae

Problem 1

The first problem is to determine whether the input string matches a pre-stored string. Using gdb, we can check the pre-stored string, which is "CSE3030@Sogang"

```
0x404038 <msg>: "CSE3030@Sogang"
(gdb) x/s 0x401176
```

The table under is the detailed explanation for the assembly code chunk by chunk.

```
Dump of assembler code for function main:
   0x0000000000401136 <+0>:     sub    $0x28,%rsp
   0x000000000040113a <+4>:     mov    $0x402004,%edi
   0x000000000040113f <+9>:     callq  0x401030 <puts@plt>
```

↑ [Stack] Allocating space on the stack for the local variables.

```
   0x0000000000401144 <+14>:    mov    %rsp,%rsi
   0x0000000000401147 <+17>:    mov    $0x402018,%edi
   0x000000000040114c <+22>:    mov    $0x0,%eax
   0x0000000000401151 <+27>:    callq  0x401040 <__isoc99_scanf@plt>
```

↑ [scanf] Call the scanf to get the input string.

```
   0x0000000000401156 <+32>:    mov    $0x0,%eax
   0x000000000040115b <+37>:    movslq %eax,%rdx
   0x000000000040115e <+40>:    movzbl 0x404038(%rdx),%edx
   0x0000000000401165 <+47>:    test   %dl,%dl
   0x0000000000401167 <+49>:    je     0x401176 <main+64>
   0x0000000000401169 <+51>:    movslq %eax,%rcx
   0x000000000040116c <+54>:    cmp    %dl,(%rsp,%rcx,1)
   0x000000000040116f <+57>:    jne    0x401193 <main+93>
   0x0000000000401171 <+59>:    add    $0x1,%eax
   0x0000000000401174 <+62>:    jmp    0x40115b <main+37>
   0x0000000000401176 <+64>:    mov    $0x1,%eax
   0x000000000040117b <+69>:    test   %eax,%eax
   0x000000000040117d <+71>:    je     0x40119a <main+100>
   0x000000000040117f <+73>:    mov    $0x40201d,%edi
   0x0000000000401184 <+78>:    callq  0x401030 <puts@plt>
   0x0000000000401189 <+83>:    mov    $0x0,%eax
   0x000000000040118e <+88>:    add    $0x28,%rsp
   0x0000000000401192 <+92>:    retq
   0x0000000000401193 <+93>:    mov    $0x0,%eax
   0x0000000000401198 <+98>:    jmp    0x40117b <main+69>
```

↑ [Check whether the strings are identical]

Pre-stored string is stored to the register %edx. As the program runs, the value stored in the %eax increases by 1, and accordingly, the pre-ordered string excluding the first letter is stored in the %edx. (e.g. CSE3030@Sogang -> SE3030@Sogang -> E3030@Sogang)

If the programs runs until the end of the string, %dl becomes 0x0 because the string is NULL. Therefore, by the code <main+47>, the program jumps to the <main+64> and print the succes message.

However, if the characters of strings are not identical, the program jumps to the <main+93> by the code <main+57>. Then %eax become 0x0 and by <main+71> it jumps to the <main+100> and print the failure message.

```
   0x0000000000401189 <+83>:    mov    $0x0,%eax
   0x000000000040118e <+88>:    add    $0x28,%rsp
   0x0000000000401192 <+92>:    retq
```

↑ [Return]

After printing the failure or succes message, initialize the return value as 0, return the stack space and fishish the program.

```
   0x000000000040119a <+100>:   mov    $0x402038,%edi
   0x000000000040119f <+105>:   callq  0x401030 <puts@plt>
   0x00000000004011a4 <+110>:   jmp    0x401189 <main+83>
End of assembler dump.
```

```
(gdb) x/s 0x402038
0x402038:       "No, that is not the input I want!"
```

↑ [Print Failure Message]

If the input is not same with the pre-stored string, print the failure message and jump to the return codes (<main+83>)

Problem 2

The second problem is to find three integers that satisfy three conditions, which is as below.

1. 64<x<=96
2. 512<z<=560
3. $y = \frac{x+2z}{3}$

After analize the assembly code, I found the proper answer using python and choose the answer ($x=87,\ y=371,\ z=513$).

```
solutions = []

for z in range(513, 561):
    for x in range(65, 97):
        y = (2*z+x)/3
        if y.is_integer():
            solutions.append((x, int(y), z))

for solution in solutions:
    print(solution)
```

```
(66, 364, 513)
(69, 365, 513)
(72, 366, 513)
(75, 367, 513)
(78, 368, 513)
(81, 369, 513)
(84, 370, 513)
(87, 371, 513)
(90, 372, 513)
(93, 373, 513)
(96, 374, 513)
(67, 365, 514)
(70, 366, 514)
(73, 367, 514)
(76, 368, 514)
```

The table below is the detailed explanation for the assembly code chunk by chunk.

```
Dump of assembler code for function main:
   0x0000000000401136 <+0>:     sub    $0x18,%rsp
   0x000000000040113a <+4>:     mov    $0x402004,%edi
   0x000000000040113f <+9>:     callq  0x401030 <puts@plt>
   0x0000000000401144 <+14>:    lea    0x4(%rsp),%rcx
   0x0000000000401149 <+19>:    lea    0x8(%rsp),%rdx
   0x000000000040114e <+24>:    lea    0xc(%rsp),%rsi
```

↑ [Stack] Allocating space on the stack for the local variables.

```
   0x0000000000401153 <+29>:    mov    $0x402018,%edi
   0x0000000000401158 <+34>:    mov    $0x0,%eax
   0x000000000040115d <+39>:    callq  0x401040 <__isoc99_scanf@plt>
```

↑ [scanf] Call the scanf to get the input integers.

```
   0x00000000004011c5 <+143>:   mov    $0x402021,%edi
   0x00000000004011ca <+148>:   callq  0x401030 <puts@plt>
   0x00000000004011cf <+153>:   jmp    0x4011b4 <main+126>
End of assembler dump.
```

↑ Success message

```
   0x00000000004011aa <+116>:   mov    $0x402040,%edi
   0x00000000004011af <+121>:   callq  0x401030 <puts@plt>
```

↑ Failure message

```
   0x00000000004011a6 <+112>:   test   %ecx,%ecx
   0x00000000004011a8 <+114>:   jne    0x4011c5 <main+143>
```

↑ [Goal]

For the program to be able to jump to the succes message<main+143>, the register %ecx must be 0x0.

```
   0x0000000000401162 <+44>:    mov    0xc(%rsp),%edx
   0x0000000000401166 <+48>:    cmp    $0x40,%edx
   0x0000000000401169 <+51>:    jle    0x4011be <main+136>
   0x000000000040116b <+53>:    mov    $0x1,%ecx
   0x0000000000401170 <+58>:    cmp    $0x60,%edx
   0x0000000000401173 <+61>:    jle    0x40117a <main+68>
```

↑ [Condition 1]

The variable x should be larger than 0x40 (64) and smaller or equal than 0x60 (96)

```
   0x0000000000401175 <+63>:    mov    $0x0,%ecx
   0x000000000040117a <+68>:    mov    0x4(%rsp),%eax
   0x000000000040117e <+72>:    cmp    $0x200,%eax
   0x0000000000401183 <+77>:    jg     0x40118a <main+84>
   0x0000000000401185 <+79>:    mov    $0x0,%ecx
   0x000000000040118a <+84>:    cmp    $0x230,%eax
   0x000000000040118f <+89>:    jle    0x401196 <main+96>
   0x0000000000401191 <+91>:    mov    $0x0,%ecx
```

↑ [Condition 2]

The variable z should be larger than 0x200 (512) and smaller or equal than 0x230 (560)

```
   0x0000000000401196 <+96>:    mov    0x8(%rsp),%esi
   0x000000000040119a <+100>:   mov    %esi,%edi
   0x000000000040119c <+102>:   sub    %edx,%edi
   0x000000000040119e <+104>:   sub    %esi,%eax
   0x00000000004011a0 <+106>:   add    %eax,%eax
   0x00000000004011a2 <+108>:   cmp    %eax,%edi
   0x00000000004011a4 <+110>:   jne    0x4011aa <main+116>
```

↑ [Condition 3]

y-x must be equal to 2*(z-y)

```
   0x00000000004011af <+121>:   callq  0x401030 <puts@plt>
   0x00000000004011b4 <+126>:   mov    $0x0,%eax
   0x00000000004011b9 <+131>:   add    $0x18,%rsp
   0x00000000004011bd <+135>:   retq
```

↑ [Return]

After printing the failure or succes message, initialize the return value as 0, return the stack space and fishish the program.

Problem 3

The third problem is to transform 10 to 401 using operations three times. The input integers are matched with different operations respectively. The match between an integer and an operation is as follow.

integer	jump to	operation
0	<main+18>	x=x+1
1	<main+21>	x=x
2	<main+77>	x=x-1
3	<main+82>	x=2*x
4	<main+18>	x=x+1
5	<main+91>	x=10
6	<main+21>	x=x
7	<main+86>	x=x**2
>7	<main+21>	x=x

By input 3, 7, and 0 sequentially, we can transform 10 to 401 as 10 → 20 → 400 → 401. The table below is the detailed explanation for the assembly code chunk by chunk. For convenience, let **%ebp as X**, and **%dbx as i**.

```
Dump of assembler code for function main:
   0x0000000000401136 <+0>:     push   %rbp
   0x0000000000401137 <+1>:     push   %rbx
   0x0000000000401138 <+2>:     sub    $0x18,%rsp
```

↑ [Stack]

Allocating space on the stack for the local variables.

```
   0x000000000040113c <+6>:     mov    $0xa,%ebp
   0x0000000000401141 <+11>:    mov    $0x0,%ebx
```

↑ [Initialization]

Initialize x=10, i=0

```
   0x000000000040114e <+24>:    cmp    $0x2,%ebx
   0x0000000000401151 <+27>:    jg     0x401198 <main+98>
```

↑ [Exit condition]

If i become larger than 2, that is receive three integers, jump to the <main+98> to determine whether success or fail.

```
   0x0000000000401198 <+98>:    cmp    $0x191,%ebp
   0x000000000040119e <+104>:   je     0x4011b6 <main+128>
   0x00000000004011a0 <+106>:   mov    $0x402038,%edi
   0x00000000004011a5 <+111>:   callq  0x401030 <puts@plt>
   0x00000000004011aa <+116>:   mov    $0x0,%eax
   0x00000000004011af <+121>:   add    $0x18,%rsp
   0x00000000004011b3 <+125>:   pop    %rbx
   0x00000000004011b4 <+126>:   pop    %rbp
   0x00000000004011b5 <+127>:   retq
   0x00000000004011b6 <+128>:   mov    $0x40201b,%edi
   0x00000000004011bb <+133>:   callq  0x401030 <puts@plt>
   0x00000000004011c0 <+138>:   jmp    0x4011aa <main+116>
End of assembler dump.
```

↑ [Determining and return]

If X is equal to 0x191(401), jump to <main+128> and print success message. But if not, do not jump and print failure message(<main+111>). After prints one of the messages, return and finishe the probram.

X=X+1

```
   0x0000000000401148 <+18>:    add    $0x1,%ebp
```

X=X (no changes for X and only increase i by 1)

```
   0x000000000040114b <+21>:    add    $0x1,%ebx
```

X=X-1

```
   0x0000000000401183 <+77>:    sub    $0x1,%ebp
```

X=2*X

```
   0x0000000000401188 <+82>:    add    %ebp,%ebp
```

X=X**2

```
   0x000000000040118c <+86>:    imul   %ebp,%ebp
```

X=10

```
   0x0000000000401191 <+91>:    mov    $0xa,%ebp
```

↑ [Operations]

Operations for the **X**.

```
   0x0000000000401175 <+63>:    cmp    $0x7,%eax
   0x0000000000401178 <+66>:    ja     0x40114b <main+21>
   0x000000000040117a <+68>:    mov    %eax,%eax
   0x000000000040117c <+70>:    jmpq   *0x402060(,%rax,8)
```

↑ [Matching]

Matching input integers to the operations respectively.

Problem 4

The last problem is to find proper string that satisfies conditions behind.

1. length of the string is 11
2. the string consists of lowercase alphabet letters.
3. the string is palindrome (symmetric)
4. the string is consists of exactly 5 different letters.

Among the strings that satisfy the four conditions above, I choose "**abcdeaedcba**". The table below is the detailed explanation for the assembly code chunk by chunk.

```
Dump of assembler code for function main:
   0x0000000000401156 <+0>:     push   %rbp
   0x0000000000401157 <+1>:     push   %rbx
   0x0000000000401158 <+2>:     sub    $0x48,%rsp
   0x000000000040115c <+6>:     mov    $0x402004,%edi
   0x0000000000401161 <+11>:    callq  0x401030 <puts@plt>
   0x0000000000401166 <+16>:    lea    0x20(%rsp),%rsi
   0x000000000040116b <+21>:    mov    $0x402018,%edi
   0x0000000000401170 <+26>:    mov    $0x0,%eax
   0x0000000000401175 <+31>:    callq  0x401060 <__isoc99_scanf@plt>
   0x000000000040117a <+36>:    lea    0x20(%rsp),%rdi
```

↑ [Allocating Stack and Receive input]

Allocating space on the stack and receiving input. Save the input to the **0x20(%rsp)**.

```
   0x000000000040117f <+41>:    callq  0x401040 <strlen@plt>
   0x0000000000401184 <+46>:    mov    %eax,%ebx
```

↑ [strlen]

Using strlen funciton, calculate the length of the string and save it to the **%ebx**.

```
   0x0000000000401186 <+48>:    cmp    $0xb,%eax
   0x0000000000401189 <+51>:    je     0x4011ae <main+88>
```

↑ [Condition 1]

The length of the string must be equal to 0xb(11)

```
   0x00000000004011bc <+102>:   movslq %ecx,%rax
   0x00000000004011bf <+105>:   movzbl 0x20(%rsp,%rax,1),%eax
   0x00000000004011c4 <+110>:   lea    -0x61(%rax),%edx
   0x00000000004011c7 <+113>:   cmp    $0x19,%dl
   0x00000000004011ca <+116>:   jbe    0x4011d1 <main+123>
```

↑ [Condition 2]

The string is consist of lowercase alphabet letters.

```
   0x00000000004011cc <+118>:   mov    $0x0,%ebp
   0x00000000004011d1 <+123>:   mov    %ebx,%edx
   0x00000000004011d3 <+125>:   sub    %ecx,%edx
   0x00000000004011d5 <+127>:   sub    $0x1,%edx
   0x00000000004011d8 <+130>:   movslq %edx,%rdx
   0x00000000004011db <+133>:   cmp    %al,0x20(%rsp,%rdx,1)
   0x00000000004011df <+137>:   je     0x4011e6 <main+144>
```

↑ [Condition 3]

The string is palindrome, which means symmetric.

```
  0x00000000004011ef <+153>:   cmpb   $0x0,(%rsp,%rdx,1)
--Type <return> to continue, or q <return> to quit---
  0x00000000004011f3 <+157>:   jne    0x4011b5 <main+95>
  0x00000000004011f5 <+159>:   movb   $0x1,(%rsp,%rdx,1)
  0x00000000004011f9 <+163>:   add    $0x1,%esi
  0x00000000004011fc <+166>:   jmp    0x4011b5 <main+95>
  0x00000000004011fe <+168>:   cmp    $0x5,%esi
  0x0000000000401201 <+171>:   jne    0x401207 <main+177>
```

↑ [Condition 4]

The string must be consist of exactly 5 different letters.

Final Result

```
cse20200977@cspro:~/cs_lab2$ ./check.py
[*] 2-1: O
[*] 2-2: O
[*] 2-3: O
[*] 2-4: O
```