Password Verification

郑悟强 PB22051082

2023.12.15

1 Purpose

Develop a password verification program for a hypothetical bank system using LC-3 assembly language. This program should validate user passwords during sensitive operations, like withdrawing funds, with a limit of three attempts.

2 Program Design

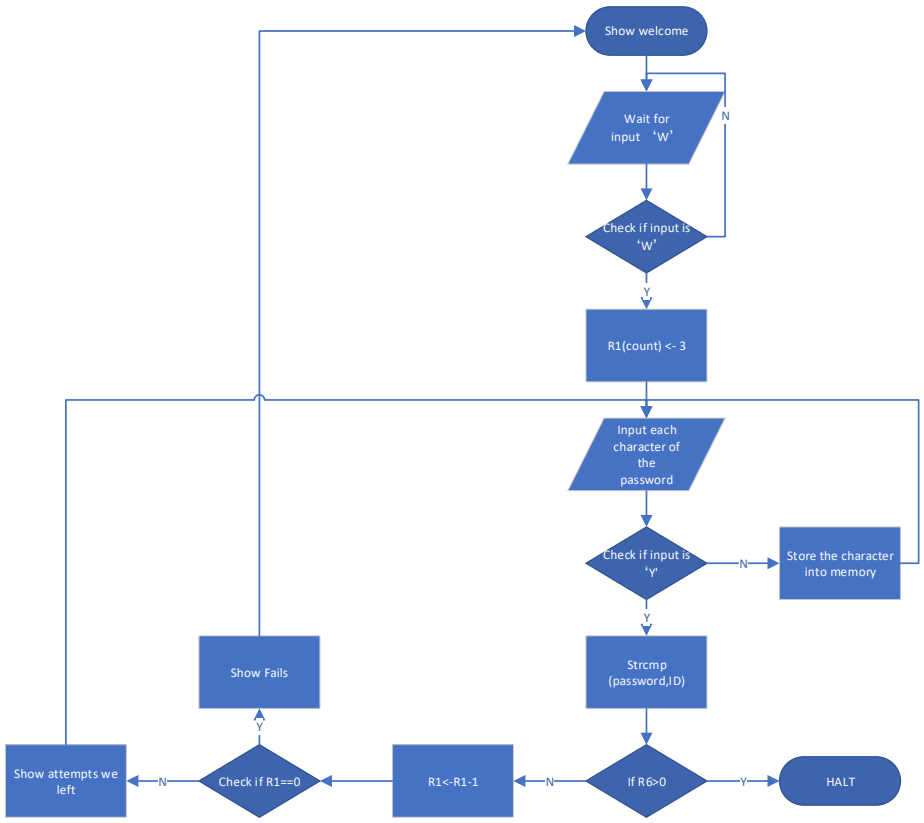
2.1 Instructions

Step1. Initial Prompt: On starting, display *Welcome to the bank system! Type 'W' to withdraw some fund.* Wait for the user to input 'W'.
Step2. Password Input: Once 'W' is entered, prompt *Please input your password:* .
Step3. Password Verification:
• The correct password is your student ID (*PB22051082*). After entering the password, type 'Y' to submit.
• Users get three attempts to enter the correct password.
• Display Success! for a correct password or *Incorrect password! X attempt(s) remain.* for an incorrect attempt, where X is the number of remaining attempts.
Step4. Attempt Limit: After three incorrect attempts, display *Fails.* and restart from step 1, which means the prompt Welcome ... will be output again and the user should call for a new job.
Step5. Successful Entry: On correct entry, the program should HALT immediately.

2.2 Strcmp Function

Similar to lab3:
Step1. R2, R3 store the address of the two strings.
Step2. LDR R4, R2, #0, LDR R5, R3, #0.
Step3. R5 <- R4 - R5 (2's complement code : R5 takes the inverse and add 1).
Step4.
if R5 == 0
compare if R2,R3 have both come to the end ,if so R6 <- 1, RET. else if R2 comes to the end but R3 doesn't , R6 <- 0 RET. else R2++, R3++, return to Step2.
else R6 <- 1, RET.

2.3 Flow Chart



3 Test Evidence

To begin with, the program will wait for the user to input 'W'.

Console (click to focus)

```
Welcome to the bank system! Type 'W' to withdraw some fund.
Input a character>
```

If we input other characters, it will keep on asking the user to input 'W' until we input 'W'.

Console (click to focus)

```
Welcome to the bank system! Type 'W' to withdraw some fund.
Input a character> Y

Input a character> T

Input a character> W
Please input your password:
```

Then the user needs to input the password. (The correct answer is my student ID: PB22051082)

If we input wrong password, the programm will warn us that we are wrong, and show the left attempts we have. For example, the 3 times we input are:

•PB220510Y

•PA22051082Y

•PB220510822Y

```
Console (click to focus)
Welcome to the bank system! Type 'W' to withdraw some fund.
Input a character> Y

Input a character> T

Input a character> W
Please input your password:
Incorrect password! 2 attempt(s) remain.
Incorrect password! 1 attempt(s) remain.
Incorrect password! 0 attempt(s) remain.
 Fails.
Welcome to the bank system! Type 'W' to withdraw some fund.
Input a character> 
```

We fail the three attempts, so the program starts from step1 again.

This time we try the correct answer (PB22051082Y):

```
Console (click to focus)
Welcome to the bank system! Type 'W' to withdraw some fund.
Input a character> W
Please input your password:
 Success!

--- Halting the LC-3 ---
```

4 Procedure

4.1 The question about PASS address been covered

Question:

When I tried my program, I found that when I input a password longer than the ID for the first time (eg. PB220510822Y), the next time, whatever I input, the program wwould consider it wrong.

Cause:

When I input the password, I will store the characters into PASS address, which I .BLKW 10 characters. But the following address are all empty, so the characters will keep on inputing. When the strcmp function starts to work, it will keep on comparing, so, it will find that, the two string's length was different, so the rsult may be wrong.

Solution:

When I finished input (Which means input a 'Y'), before comparing, the program will store a 0 into the end of PASS location. So the password string will have an end for comparing. **Code:**

```
;then we need to compare the password with the ID
COMPARE AND R0, R0, #0
        STR R0, R3, #0      ;before we start to compare, we should add a \0 at the end of the password
        JSR STRCMP
```

!	▶ **x30E8**	x0050	80	*PASS .BLKW 11*
!	▶ **x30E9**	x0042	66	*PASS .BLKW 11*
!	▶ **x30EA**	x0032	50	*PASS .BLKW 11*
!	▶ **x30EB**	x0032	50	*PASS .BLKW 11*
!	▶ **x30EC**	x0030	48	*PASS .BLKW 11*
!	▶ **x30ED**	x0035	53	*PASS .BLKW 11*
!	▶ **x30EE**	x0031	49	*PASS .BLKW 11*
!	▶ **x30EF**	x0030	48	*PASS .BLKW 11*
!	▶ **x30F0**	x0038	56	*PASS .BLKW 11*
!	▶ **x30F1**	x0032	50	*PASS .BLKW 11*
!	▶ **x30F2**	x0000	0	*PASS .BLKW 11*
!	▶ **x30F3**	x0032	50	
!	▶ **x30F4**	x0032	50	
!	▶ **x30F5**	x0032	50	

the location of x30F2 is the 0 add to the end.

4.2 How to output line feed in the console

Question:
If we just let the string output on the console, they will pile together, and it's not tidy and hard to read. So, we need to print line feed.
Solution:
We can get R0 10 (the ASCII code of line feed) and use TRAP x21 to print.

5 Discussion Questions

5.1 Do you use function definition/call in your program, why or why not?

I use the strcmp function to compare the two strings.
Because we have made this function in lab3, I can just change a few parameter to use it, it's convenient and will make the code more tiny and logical.

5.2 Do you use a recursive function in your program, why or why not? If not, will you use this trick when the stack mechanism is provided?

I don't use this.
Because the recursive function is convenient when we tackle with some loop questions for which we can only find the recurrence relation between the elements.
But in this program, we just need to cope with an easy loop, about the attempts we left. The recursive function isn't convenient here.
If we face some recurrence problem, I will prefer to use this.

5.3 How do you store these present prompts? If you use a recursive function, can you conclude how many parts should a typical program assembled?

I choose to use *.STRINGZ* to store the promots.
If I use recursive function, I will divide a program into *1.initialize variable, 2.recursive tackle the problem, 3.output the result* three parts.

5.4 Assess the security of your program with potential vulnerability scenarios. For example, what if the user types a super long password to your program?

It will make no sense.
Because I store the password at the end of my address, from x30E8, the address following are all empty. Unless the password will long enough to the location of xFFFF (It's unpractical. 53015 characters)
Or we can try this way to solve this problem:
First we will use a register to store the length of the correct answer. And use a count to store the length of the password we have inputted, if the length has been to an upper bond we set before, we will just stop inputting and consider it wrong.

5.5 Share challenges faced during development and how they were resolved.

In the section4 *Procedure*.