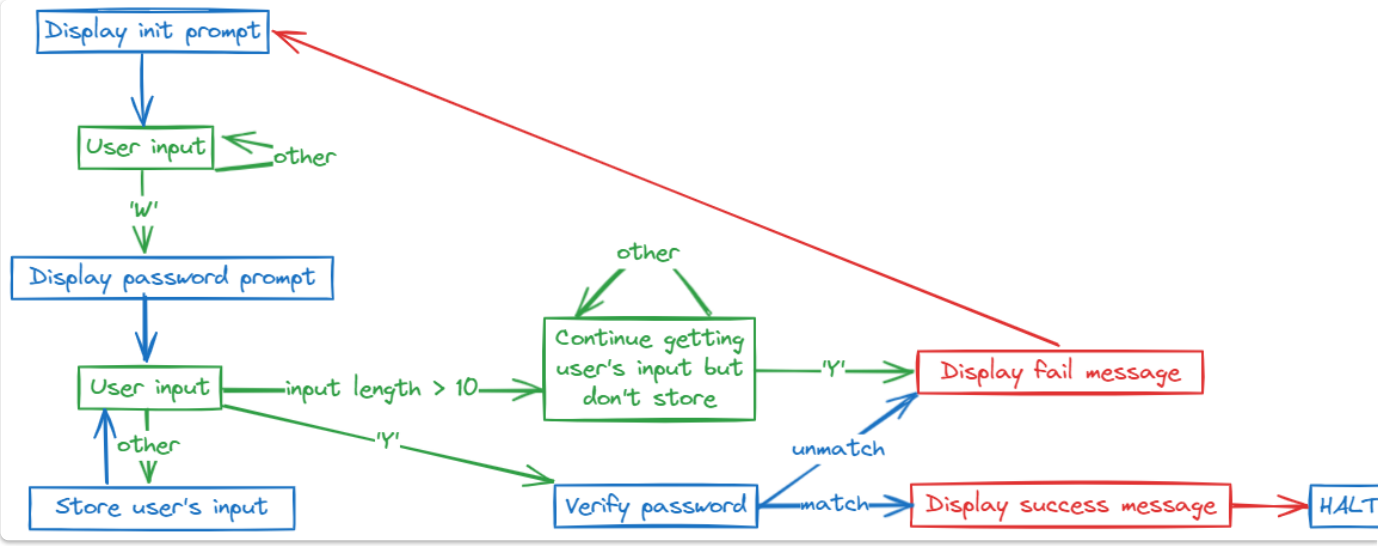
lab5

实验目的

1. 学习使用LC-3汇编语言处理字符串的输入与输出。
2. 进一步学习汇编语言中的函数调用。

实验方法

流程图



- 在内存中留出10个连续位置存储用户输入的密码，验证密码时可直接使用LAB3写的`strcmp`函数。
- 由于正确密码的长度是10， 故若用户在输入`'Y'`之前已经输入了超过10个字符，则这轮肯定失败，故无需继续储存用户的输入，只需继续接收用户的输入而不做任何事情，等用户输入`'Y'`后直接跳到失败状态。

打印PROMPT

预先设置好的prompt用`.STRINGZ`放到内存中：

```
INIT_STR .STRINGZ   "Welcome to the bank system! Type 'W' to withdraw some fund."
PROMPT_STR .STRINGZ "Please input your password:"
SUCCESS_STR .STRINGZ    "Success!"
INCORRECT_STR_1 .STRINGZ    "Incorrect password! "
INCORRECT_STR_2 .STRINGZ    " attempt(s) remains."
FAIL_STR .STRINGZ   "Fails.\n"
```

先用`LEA`指令将要打印的prompt的首地址载入`R0`，再调用`PUTS`来打印:

```
LEA     R0,     INIT_STR        ; print init prompt
PUTS
```

等待用户输入`'W'`

内存里预先存入`-'W'`：

```
W .FILL #-87  ; inverse of the ascii code of 'W'
```

每次将用户输入的字符与`'W'`相比较，不相同则继续让用户输入。

```
            LD  R1, W               ; R1 <- -'W'
INPUT_W     IN                      ; wait for the user to enter 'W'
            ADD R0, R0, R1          ; R0 <- R0 - 'W'
            BRnp  INPUT_W           ; input != 'W'
```

处理用户输入的密码

在内存中分配10个连续位置用于存储用户输入的密码，第11个位置填`'\0'`以方便字符串比较

```
USER_ENTERED .BLKW  10  ; store the user-entered password here
NULL .FILL  #0
```

同上，内存里预先存入`-'Y'`：

```
Y .FILL #-89  ; inverse of the ascii code of 'Y'
```

`R3`指向下一个字符存入的位置，`R4`指向上面分配的那段内存位置结束的地方。用户每次输入一个字符，若是`'Y'`则跳到`CHECK`模块；否则，若`R3`已超过`R4`，说明用户正在输入长度超过10的密码，一定是错的，跳到`EXCEED`模块；再否则，存储用户输入的字符，递增`R3`，继续让用户输入。

```
            LD  R6, Y               ; R6 <- -'Y'
INPUT       LEA R3, USER_ENTERED    ; R3: addr to store the next input char
            ADD R4, R3, #9          ; R4: end of USER_ENTERED
            NOT R4, R4
            ADD R4, R4, #1          ; 2's complement of negative of R4
LOOP_INPUT  IN                      ; input
            ADD R5, R0, R6          ; R5 <- R0 - 'Y'
            BRz CHECK               ; input == 'Y', end
            ADD R5, R3, R4          ; R5 <- R3 - R4
            BRp EXCEED              ; R3 exceeds the bound, the input is definitely
wrong
            STR R0, R3, #0          ; store user's input char
```

```
            ADD R3, R3, #1          ; increment R3
            BR LOOP_INPUT           ; continue loop
```

`EXCEED`模块只是不断地接受用户输入的字符而不存储，直到用户输入`'Y'`后，直接跳转到`INCORRECT`模块。

```
EXCEED      IN
            ADD R5, R0, R6          ; R5 <- R0 - 'Y'
            BRz INCORRECT           ; input == 'Y', directly branch to INCORRECT
            BR EXCEED
```

验证密码

内存里预先存入正确密码：

```
ANS .STRINGZ    "PB22000197"  ; answer of the password
```

只需比较`ANS`和`USER_ENTERED`两个位置存储的字符串是否一致，以下程序与LAB3完全相同：

```
CHECK       LEA R0, ANS             ; addr of the next char of ans
            LEA R1, USER_ENTERED    ; addr of the next char of user-entered
LOOP_CHECK  LDR R3, R0, #0          ; R3 <- char of ans
            LDR R4, R1, #0          ; R4 <- char of user-entered
            NOT R4, R4
            ADD R4, R4, #1
            ADD R4, R3, R4          ; R4 <- R3 - R4
            BRnp INCORRECT          ; unmatching, incorrect
            ADD R3, R3, #0          ; set cc according to R3
            BRz  SUCCESS            ; both reaches '\0', success
            ADD R0, R0, #1          ; increment addr
            ADD R1, R1, #1          ; increment addr
            BR LOOP_CHECK
```

输入密码正确

打印成功消息并退出程序：

```
SUCCESS     LEA R1, SUCCESS_STR     ; print success message
            PUTS
            HALT
```

输入密码不正确

输入密码的整体过程中用`R2`记录剩余尝试的次数，初始化为3：

```
AND R2, R2, #0          ; clear R2
ADD R2, R2, #3          ; R2 <- 3 (total # of attempts)
```

`INCORRECT`模块中首先递减`R2`，若`R2`降为0，则直接跳转到`FAIL`模块：

```
ADD R2, R2, #-1         ; decrement # of attempts
BRz FAIL                ; no attempt left, failed
```

否则先打印`"Incorrect password! "`，然后打印`R2`的值（内存`ZERO`处已预先存入`x30`），再打印`" attempt(s) remain."`。

```
LEA     R0,     INCORRECT_STR_1 ; print the first half of the incorrect message
PUTS
LD      R0,     ZERO            ; R0 <- x30
ADD R0, R0, R2          ; R0 <- # of attempts left
OUT                     ; print the # of attempts left
LEA R0, INCORRECT_STR_2 ; print the second half of the incorrect message
PUTS
```

最后继续让用户尝试：

```
BR  INPUT               ; continue attempt
```

失败

打印失败消息，并回到初始状态：

```
FAIL        LEA R0, FAIL_STR        ; print fail message
            PUTS
            BR  INIT                ; go back to the init state
```

完整代码

```
            .ORIG x3000
INIT        LEA R0,     INIT_STR        ; print init prompt
            PUTS

            LD  R1, W               ; R1 <- -'W'
INPUT_W     IN                      ; wait for the user to enter 'W'
            ADD R0,     R0,     R1          ; R0 <- R0 - 'W'
            BRnp  INPUT_W           ; input != 'W'

            LEA R0, PROMPT_STR      ; print password prompt
```

```
            PUTS

            LD  R6,     Y               ; R6 <- -'Y'
            AND R2,     R2,     #0          ; clear R2
            ADD R2, R2, #3          ; R2 <- 3 (total # of attempts)

; wait for the user to enter the password
INPUT       LEA R3, USER_ENTERED    ; R3: addr to store the next input char
            ADD R4, R3, #9          ; R4: end of the locations to store the user-
enterdd password
            NOT R4, R4
            ADD R4, R4, #1          ; 2's complement of negative of R4
LOOP_INPUT  IN                      ; input
            ADD R5,     R0, R6          ; R5 <- R0 - 'Y'
            BRz CHECK               ; input == 'Y', end
            ADD R5, R3, R4          ; R5 <- R3 - R4
            BRp EXCEED              ; R3 exceeds the bound, the input is definitely
wrong
            STR R0,     R3, #0          ; store user's input char
            ADD R3, R3, #1          ; increment R3
            BR LOOP_INPUT           ; continue loop

EXCEED      IN
            ADD R5, R0, R6          ; R5 <- R0 - 'Y'
            BRz INCORRECT           ; input == 'Y', directly branch to INCORRECT
            BR EXCEED

; check
CHECK       LEA R0, ANS             ; addr of the next char of ans
            LEA R1, USER_ENTERED    ; addr of the next char of user-entered
LOOP_CHECK  LDR R3,     R0,     #0          ; R3 <- char of ans
            LDR R4, R1, #0          ; R4 <- char of user-entered
            NOT R4, R4
            ADD R4, R4, #1
            ADD R4, R3, R4          ; R4 <- R3 - R4
            BRnp INCORRECT          ; unmatching, incorrect
            ADD R3, R3, #0          ; set cc according to R3
            BRz  SUCCESS            ; both reaches '\0', success
            ADD R0, R0, #1          ; increment addr
            ADD R1, R1, #1          ; increment addr
            BR LOOP_CHECK

INCORRECT   ADD R2, R2, #-1         ; decrement # of attempts
            BRz FAIL                ; no attempt left, failed
            LEA R0,     INCORRECT_STR_1 ; print the first half of the incorrect
message
```

```
            PUTS
            LD  R0,     ZERO            ; R0 <- x30
            ADD R0, R0, R2          ; R0 <- # of attempts left
            OUT                     ; print the # of attempts left
            LEA R0, INCORRECT_STR_2 ; print the second half of the incorrect message
            PUTS
            BR  INPUT               ; continue attempt

SUCCESS     LEA R1, SUCCESS_STR     ; print success message
            PUTS
            HALT

FAIL        LEA R0, FAIL_STR        ; print fail message
            PUTS
            BR  INIT                ; go back to the init state


            INIT_STR .STRINGZ   "Welcome to the bank system! Type 'W' to withdraw some
fund."
            PROMPT_STR .STRINGZ "Please input your password:"
            SUCCESS_STR .STRINGZ        "Success!"
            INCORRECT_STR_1 .STRINGZ    "Incorrect password! "
            INCORRECT_STR_2 .STRINGZ    " attempt(s) remain."
            FAIL_STR .STRINGZ   "Fails.\n"
            W .FILL     #-87  ; inverse of the ascii code of 'W'
            Y .FILL #-89  ; inverse of the ascii code of 'Y'
            SAVE .BLKW  1
            ANS .STRINGZ        "PB22000197"  ; answer of the password
            USER_ENTERED .BLKW  10  ; store the user-entered password here
            NULL .FILL  #0
            ZERO .FILL  x30

           .END
```

程序测试结果

链接：https://rec.ustc.edu.cn/share/d169b2d0-9f3f-11ee-bafb-af0ea963b77e

---

问题讨论

1. 我没有自己定义函数, 而是直接调用了`OUT`, `PUTS`, `IN`这些trap routines来完成输入输出。
2. 我没有使用递归函数，有了栈也不会使用，因为按照我的实验思路并不需要它。

3. 如”实验方法”中所述，我将这些prompt预先放到内存中。
4. 如程序运行结果的视频中所展示的那样，程序完全可以应对非常长的密码输入，因为当用户输入的密码长度超过10时，程序不再存储用户的输入了，只是继续接受用户输入而不做任何事，所以不存在内存溢出的问题。
5. 原本我打算在10个字符的连续内存位置里循环地存储用户的输入，但后来发现这样会导致验证密码时只考虑了用户的最后10个输入，例如若用户输入`2333PB22000197Y`，程序也会判对，但实际上用户输入的密码是不对的。后来我发现当用户输入了超过10个字符时，这轮的密码一定是错的，那么也就无需继续存储和验证了；但是为运行效果起见又要等用户输入`'Y'`了才能判断结果，因此写了一个新的模块`EXCEED`，这个模块只循环地接收用户输入而不做任何事情，当用户输入`'Y'`后，它直接跳转到`INCORRECT`模块。

实验总结

通过本次实验，我：

1. 掌握了使用LC-3汇编语言输入和输出字符串的方法。
2. 进一步巩固了函数调用的有关知识。