CS061 – Lab 08
Fun with Palindromes!

1 High Level Description

The purpose of this lab is to break down the identification of palindromes into their most atomic components and implement a palindrome checker in LC3.

2 Our Objectives for This Week

1. Exercise 01 ~ Capture a string of text and store it
2. Exercise 02 ~ Check to see if it's a palindrome
3. Exercise 03 ~ Case conversion

What is a Palindrome?

In case you didn't already know, a palindrome is a word or phrase that is spelled the same forwards as backwards. Such words include:

- "racecar"
- "madam"
- "deified"
- "tacocat"

Phrases can be palindromes too (see Exercise 03)! For example, the following are all palindromes (with the assumption that anything except alphabet characters are ignored)

- "live not on evil"
- "So many dynamos"
- "Are we not drawn onward, we few, drawn onward to new era"

Exercise 01

Write the following subroutine, which allows a user to enter a string at run-time (unlike the .STRINGZ pseudo-op, which stores "hard-coded" strings at compile-time).

```
;-------------------------------------------------------------------------------------------------------
; Subroutine: SUB_GET_STRING
; Parameter (R0): The address of where to start storing the string
; Postcondition: The subroutine has allowed the user to input a string,
;                terminated by the [ENTER] key, and has stored it in an array
;                that starts at (R0) and is NULL-terminated.
; Return Value: R5 The number of non-sentinel characters read from the user
;-------------------------------------------------------------------------------------------------------
```

This subroutine should prompt the user to enter in a string of text, which will be terminated by the [ENTER] key. The string of text will be stored starting at whatever address is specified by (R0) and will be NULL-terminated (i.e. The subroutine will store zero (#0) at the end of the array). The subroutine should not store the sentinel value (i.e. the newline character) in the array. The subroutine returns the number of non-sentinel characters entered in R5.

Example:

If the user enters: "This is really hard!", then the array will look like this:

'T'	'h'	'i'	's'	' '	'i'	's'	' '	'r'	'e'	'a'	'l'	'l'	'y'	' '	'h'	'a'	'r'	'd'	'!'	0

R5 will hold the value #20 = x14

Test Harness:

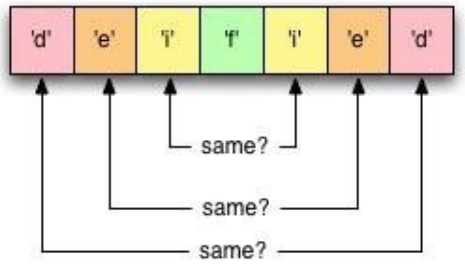
Now write a test harness (i.e. a program that tests your subroutine to make sure it works) that does the following:

1. R0 <- Some address at which to store the array *(make sure you have enough free memory starting from this address to store the number of characters likely to be entered)*.
2. Calls the subroutine
3. Immediately calls PUTS (aka: Trap x22) to print the string (which can be done since R0 still holds the address of the start of a null-terminated string)

Exercise 02

Now add the following subroutine:

```
;-------------------------------------------------------------------------------------------------------------------
; Subroutine: SUB_IS_A_PALINDROME
; Parameter (R0): The address of a string
; Parameter (R5): The number of characters in the array.
; Postcondition: The subroutine has determined whether the string at (R0) is
;                       a palindrome or not, and returned a flag to that effect.
; Return Value: R4 {1 if the string is a palindrome, 0 otherwise}
;-------------------------------------------------------------------------------------------------------------------
```



Hints:

- You know the starting address of the array
- You know how many characters are in the array
- Thus, you can calculate the address of the last character of the array
- If the array has n characters, compare
    1. array[0] with array[n]
    2. array[1] with array[n-1]
    3. array[2] with array[n-2]
    4. …
- At what point can you decide that the string *IS* a palindrome?
  At what point can you decide that the string is *NOT* a palindrome?
  Hint: in NEITHER case is the answer "after n comparisons"

Test Harness:

Write a test harness that does the following (you can reuse code from Ex1):

1. Prompts the user to type in a string, which will be analyzed
2. Obtains the string from the user
3. Calls the palindrome-checking subroutine
4. Uses the return value of the subroutine to print to the user whether the string was a palindrome or not

Exercise 03:

The subroutine from Exercise 02 would not recognize a phrase such as *"Madam, I'm Adam"* as a palindrome. It would be fairly simple to rework our palindrome subroutine to ignore whitespace, punctuation and case, but for now we will just handle case:

Write the following subroutine:

```
;-------------------------------------------------------------------------------------------------------------
; Subroutine: SUB_TO_UPPER
; Parameter (R0): Starting address of a null-terminated string
; Postcondition: The subroutine has converted the string to upper-case in-place
;                i.e. the upper-case string has replaced the original string
; No return value.
;-------------------------------------------------------------------------------------------------------------
```

Hints:

- Check the ASCII table (www.asciitable.com) to see how uppercase and lowercase letters differ in binary
- The conversion of a letter to uppercase can be done with a total of two lines of LC3 code. Look at the difference in the hexadecimal values of a lowercase vs. an uppercase letter.
- Use bit-masking.

Test Harness:

Instead of writing a separate test harness for this subroutine, you can just add a call to it inside your is_palindrome subroutine from exercise 2, and test it with the palindrome like "Racecar"