CS061 – Lab 07
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 ~ Make it extra awesome just for the fun of it!

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

1. Write the following subroutine:

```
;-----------------------------------------------------------------------------------------------------------
; 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 a #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**

Test Harness:

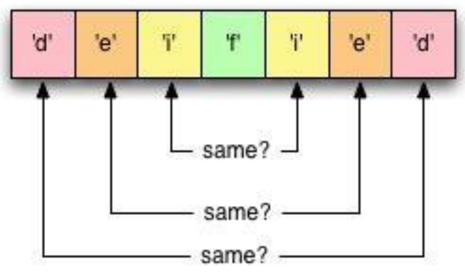
Write a test harness (i.e. a program that tests your subroutine to make sure it works) that does the following:

i. R0 <- Some address at which to store the array
ii. Calls the subroutine
iii. 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

Write 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 returned a flag indicating such.
; 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. …
  5. When will you stop? (Hint: NOT after n comparisons)
- Return that the string is not a palindrome (i.e. return 0) if you ever get a mismatch during comparison.
- If the two ends you are comparing meet in the middle (i.e. when you get to the green square in the Figure above) and you have not yet returned false, then the string must be a palindrome because the first half mirrors the second half.

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 (optional - but it will give you a head start for lab 08!):

The subroutine from Exercise 02 does not ignore whitespace, punctuation, and sees uppercase letters as different from lowercase.

Enhance your subroutine from Exercise 02 so that it ignores whitespace, punctuation, and is case-insensitive.

Hint: Check the ASCII table in your book (which you brought to lab…RIGHT?) or www.asciitable.com to see how uppercase and lowercase letters differ in binary as well as how space and punctuation characters are coded.