CS061 – Lab 6

Fun with Palindromes!

1 High Level Description

The purpose of this lab is to break down the identification of a palindrome into its most basic steps, 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 ~ Refine your subroutine with Case conversion

3.1 Exercises

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 alphabetic 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 captures a user-entered string at run-time, storing it as a null-terminated character array *(just like the .STRINGZ pseudo-op, with the big difference that that .STRINGZ requires a hard-coded character array at compile-time)*.

```
;-----------------------------------------------------------------------------------------------------------
; Subroutine: SUB_GET_STRING
; Parameter (R1): The starting address of the character array
; Postcondition: The subroutine has prompted the user to input a string,
;                terminated by the [ENTER] key (the "sentinel"), and has stored
;                the received characters in an array of characters starting at (R1).
;                the array is NULL-terminated; the sentinel character is NOT stored.
; Return Value (R5): The number of non-sentinel characters read from the user.
;                    R1 still contains the starting address of the array.
;-----------------------------------------------------------------------------------------------------------
```

This subroutine should prompt the user to enter in a string of text, ending with the [ENTER] key. The string of text will be stored starting at whatever address is specified by (R1) and will be NULL-terminated (i.e. the subroutine will store zero (x0000 = #0 ='\0') at the end of the array). The entry sentinel value itself (i.e. the newline) must not be stored in the array!

The subroutine returns in R5 the *number of non-sentinel characters* entered.

REMEMBER: no "ghost writing"! Echo each character as it is received from the user!

Example:

If the user enters: “This is really hard!”, followed by [ENTER], then the array will look like this:

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

Where the initial 'T' is stored in the address specified by R1 (this value will not be changed by the subroutine); R5 will hold the value x14 = #20 *(count the characters to confirm!)*

Test Harness:

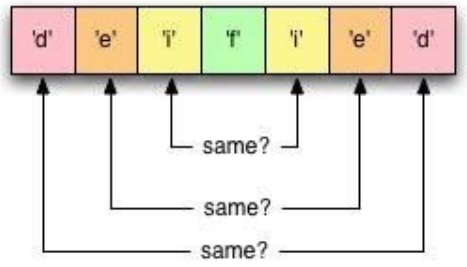
Now write a test harness *(i.e. a program that tests your subroutine to make sure it works)* that does the following:

1. R1 <- The address at which to store the array.
   Hard code this address, and reserve space there using .BLKW
   Make sure you have enough free memory starting from this address to store the maximum number of characters likely to be entered - e.g. #100.
2. Calls the subroutine
3. Calls the BIOS routine PUTS (aka Trap x22) to print the string.
   *Remember that PUTS needs the starting address of the string in **R0**, not R1!*

Exercise 02

Now add a subroutine that decides whether the newly entered string it is a palindrome or not.

```
;-------------------------------------------------------------------------------------------------------------
; Subroutine: SUB_IS_A_PALINDROME
; Parameter (R1): The starting address of a null-terminated string
; Parameter (R5): The number of characters in the array.
; Postcondition: The subroutine has determined whether the string at (R1) 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 already know the starting address of the array.
- You already 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-1]
    2. array[1] with array[n-2]
    3. array[2] with array[n-3]
    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:

Add the following steps to your original test harness:

1. Call the palindrome-checking subroutine
2. Use the return value of the subroutine to report to the user whether the string was a palindrome or not - in other words, report the result in the *test harness*, NOT in the subroutine itself. The subroutine just sets a flag *(this is a standard technique)*.

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 subroutine to ignore whitespace and punctuation, but for now we will just handle character case, so that your subroutine could at least recognize as a palindrome the string *"MadamImAdam"*. We will do this by converting the entire phrase to the same case before doing the actual palindrome check.

Write the following subroutine:

```
;-----------------------------------------------------------------------------------------------------------
; Subroutine: SUB_TO_UPPER
; Parameter (R1): 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, no output (but R1 still contains the array address, unchanged).
;-----------------------------------------------------------------------------------------------------------
```

Hints:

- Check the ASCII table to see how uppercase and lowercase letters differ in binary/hex
- Use bit-masking, not arithmetic: the conversion of a letter to uppercase can be done with a total of two lines of LC3 code.

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 a palindrome like "Racecar".

*This is one of the few cases where you will create nested subroutines: it is safe to do so here because **a)** the nested routine has no output and no return value; and **b)** you properly back up & restore only the necessary registers in all subroutines, don't you ? :)*

3.2 Submission

Demo your lab exercises to your TA ***before you leave lab***.
If you are unable to complete all exercises in lab, show your TA how far you got, and request permission to complete it after lab.
Your TA will usually give you partial credit for what you have done, and allow you to complete & demo the rest later for full credit, so long as you have worked at it seriously.
When you're done, demo it to any of the TAs or instructors in office hours ***before your next lab***.
Office hours are posted on Piazza, under the "Staff" tab.

4 So what do I know now?

... How to play games with Assembly Language :)