CS 61 - Programming Assignment 5

Objective

The purpose of this final programming assignment is to deepen your understanding of using subroutines by implementing a recursive algorithm.

High Level Description

A string can be determined to be a palindrome (the string reads the same forwards and backwards)using a recursive algorithm. This algorithm checks if the first and last characters are the same, then if the second and second to last characters are also the same, and so on.
Implementing recursive algorithms in assembly, however, can be quite challenging. Using static memory addresses to back up registers only works for one call to a subroutine, but fails if the subroutine is called either directly or indirectly recursively. You must use a stack data structure to properly execute recursive subroutine calls.
You will write a program with a main subroutine and subroutines to get a string from the user and determine the length of a zero terminated string. Additionally, you will write a subroutine to determine if the string entered by the user is a palindrome. While this determination can be made with an iterative algorithm, in this assignment you will be required to make the determination recursively.

Before You Start Coding

The recursive version of determining if a string is a palindrome, is outlined in C/C++ below. Your implementation in assembly will be similar.

```c
bool is_palindrome(char *beg, char *end) {
 if (beg >= end) return true;        // return true if middle of string
 if (*beg != *end) return false;     // return false if chars !=
 return is_palindrome(++beg, --end); // otherwise, recurse
}
```

The main challenge of this assignment is to create a subroutine that can call itself directly, or indirectly. The proper protocol and rules of stack discipline outlined in Lab 7 will allow your subroutine to call itself.

For more details on how to properly use stacks for subroutines, see chapter 8, specifically sections 8.2 and 8.3.

Your Tasks

The assignment can be broken down into the following tasks:

1. The code template you have already been given sets up the stack and calls the main subroutine. You will write code in the main subroutine which calls a subroutine that asks the user for a string, and then calls another subroutine that counts the number of characters in the zero-terminated string. Once you have the string from the user and its length, you will call the recursive subroutine that determines if the string is a palindrome. You will write each of the subroutines mentioned above.
2. Write a subroutine that takes an address to a string that will prompt the user, and the address where that string should be stored. It returns nothing, since the user-entered string will be stored at the address of the second parameter to the subroutine. As the user enters the characters of the string, your subroutine should echo them to the screen. Once the user presses the enter key, the user input is done, and the word after the last character should be zero.
3. Write a subroutine that takes an address to a zero terminated string and returns the number of characters in that string, excluding the zero terminator. Remember to follow the subroutine protocol outlined in Lab 7, by putting the returned count in R0.
4. Write a subroutine that takes two addresses as input parameters. The first address points to the first character of the string. The second address points to the last character. The subroutine then determines if the addresses are equal, returns true if they are, and then determines if the character values at the two addresses are different, returning false if they are. These two conditions are known as the base case (or stop conditions) for the recursive algorithm. Finally, if neither of the previous conditions are true, the subroutine should call itself (recurse) passing in the first address incremented by one, and the second address decremented by one.

Hints and advice:

1. Remember to follow the subroutine protocol and stack discipline.
2. If your subroutine changes the value of a register, other than the return register R0, then back it up on the stack.
3. Use 0 for false and 1 for true when returning the result from your recursive palindrome subroutine. You can use this return value to print whether or not the string is a palindrome.
4. For the first call to the palindrome subroutine, you can determine the second address by adding the length of the string minus one to the address of the string.
5. Make sure you really understand how to debug code using the LC3Tools
6. The most common error will probably deal with not properly restoring the values to their registers, most likely because you did the increment/decrement of R6 one time too many or too little.
7. The best way to debug recursion problems is to step into the recursive palindrome method on its first call. Make sure that all the registers are properly restored to the original values when the subroutine first starts executing.

Requirements:

1. You must implement the three methods described above. You may add others, but those three are the minimum.
2. You must use the specified subroutine protocol and stack discipline for every subroutine you implement, even if they are not recursive. This requirement does not include the main subroutine. You do not need to backup and restore registers for main.
3. You must use the proper subroutine headers for each subroutine in this assignment, except for the main subroutine. The subroutine header format is specified below.

4. The subroutine that asks for and gets user input should be at a label called get_user_string.
5. In the get_user_string subroutine the user should be prompted with the string `"Enter a string: "` to get the user string.
6. The get_user_string subroutine should take two parameters. The first is the address of the user prompt string and should be in R1. The second is the address where the user string should be stored and should be in R2.
7. The get_user_string subroutine does not return a value.
8. The subroutine that computes the length of a zero terminated string should be called strlen.
9. The strlen subroutine takes one parameter, the address of the string. This address should be in R1.
10. The strlen subroutine will return the number of non-zero characters in the string at the address given in R1. This length will be returned using R0.
11. The recursive subroutine that determines if a string is a palindrome should be called palindrome.
12. The palindrome subroutine shall take two parameters. The first parameter is the address of the first letter of the string stored in R1. The second parameter is the address of the last character of the string stored in R2.
13. The palindrome subroutine shall return 1 if the passed in string is a palindrome, and 0 otherwise. This return value shall be stored in R0.
14. The palindrome subroutine shall be able to handle strings upto 100 characters in length.
15. Once the string has been processed by the palindrome subroutine, the main subroutine shall print out `"The string is a palindrome"` if the string was a palindrome and the subroutine returned 1, and `"The string is not a palindrome"` otherwise.

Hint: If you use the code templated provided as is, most of these requirements will already be met.

Subroutine Headers

The following headers have been provided to you in the template - you must adhere to them exactly!

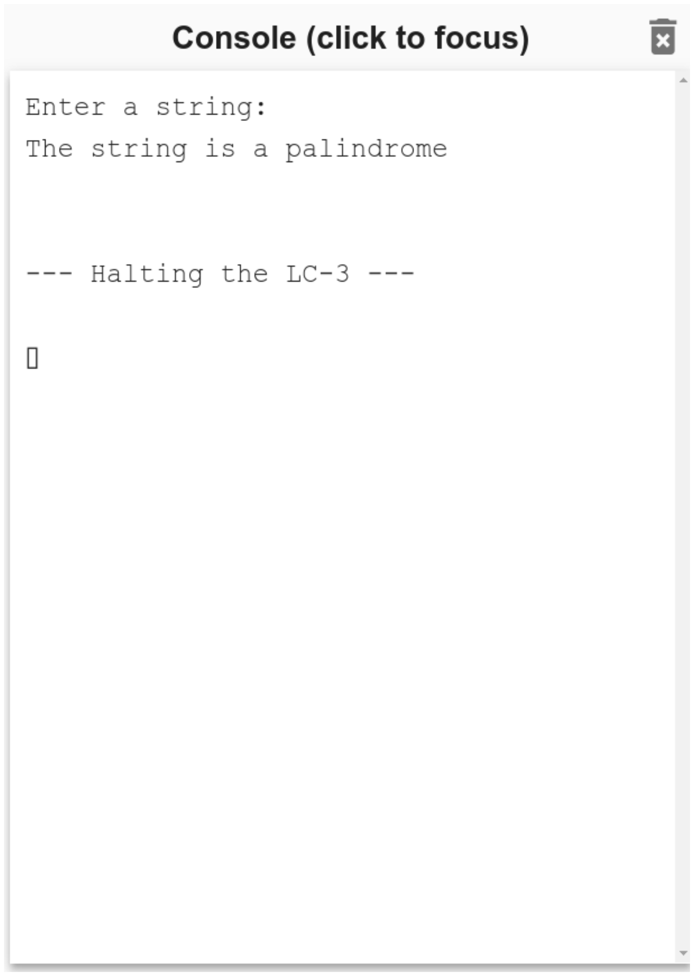
```
;------------------------------------------------------------------------------
; (subroutine name) - (Describe what it does in 1 or 2 lines)
;
; parameter: R1 - (describe input parameter)
; parameter: R2 - (describe input parameter if it exists, else omit this line.)
;
; returns: (describe value returned in R0, or say “nothing”)
;------------------------------------------------------------------------------
```

Replace the text in parentheses as described, without the parentheses. For example:

```
;------------------------------------------------------------------------------
; strlen - compute the length of a zero terminated string
;
; parameter: R1 - the address of a zero terminated string
;
; returns: The length of the string
;------------------------------------------------------------------------------
```

Expected/ Sample output

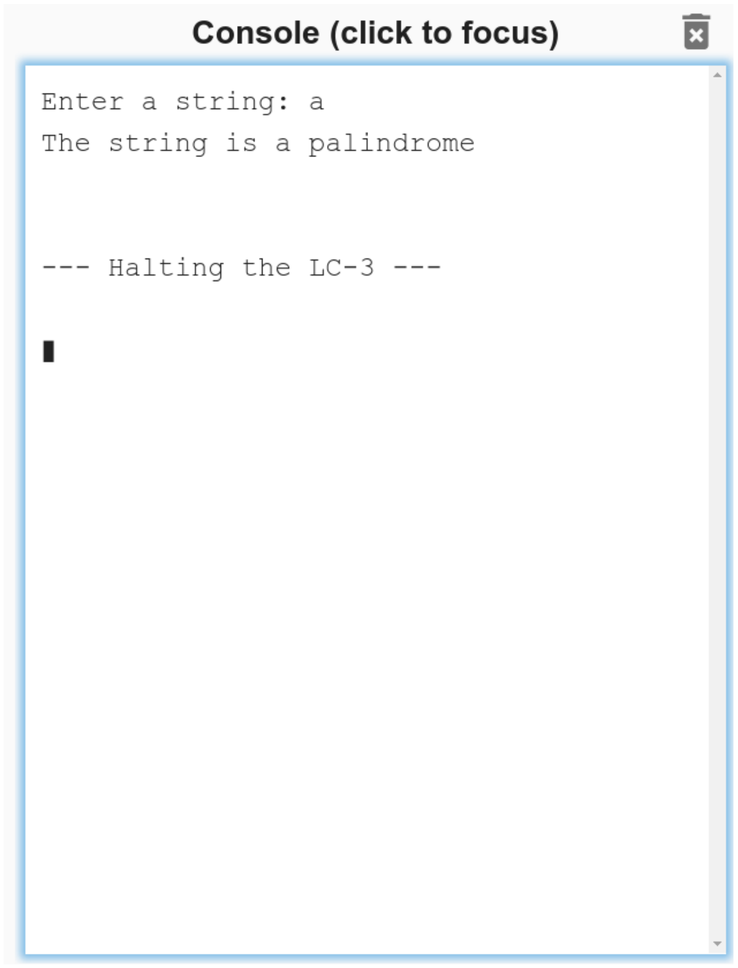
Examples



```
Enter a string:
The string is a palindrome


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

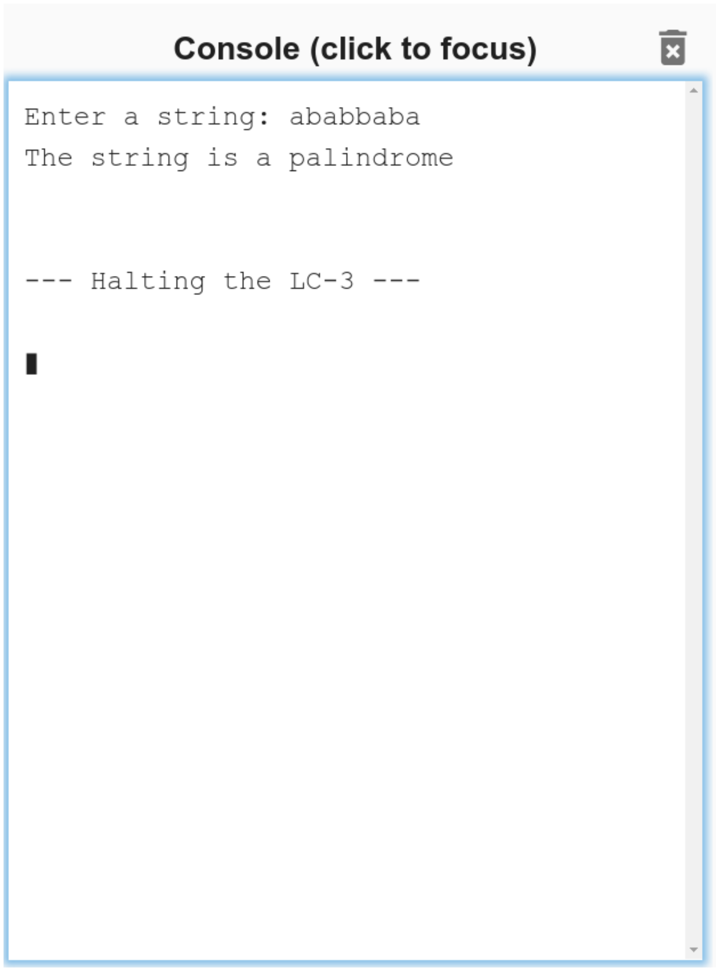
1. The empty string is a palindrome



```
Enter a string: a
The string is a palindrome


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

2. All one character strings are palindromes

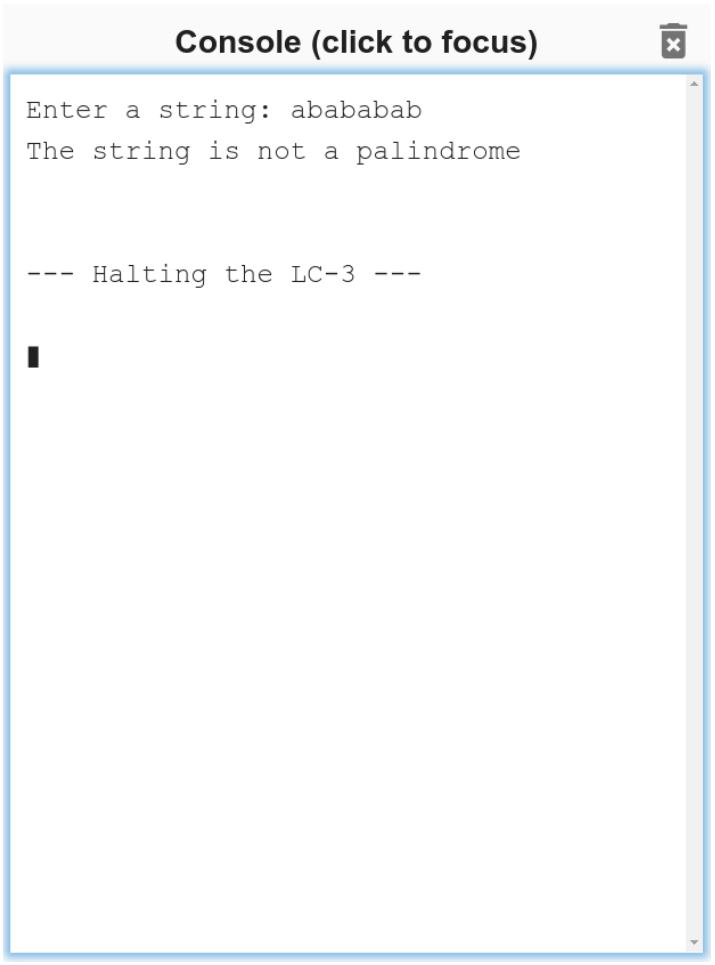

Console (click to focus)

```
Enter a string: ababbaba
The string is a palindrome


--- Halting the LC-3 ---

▮
```

3. ababbaba is a palindrome

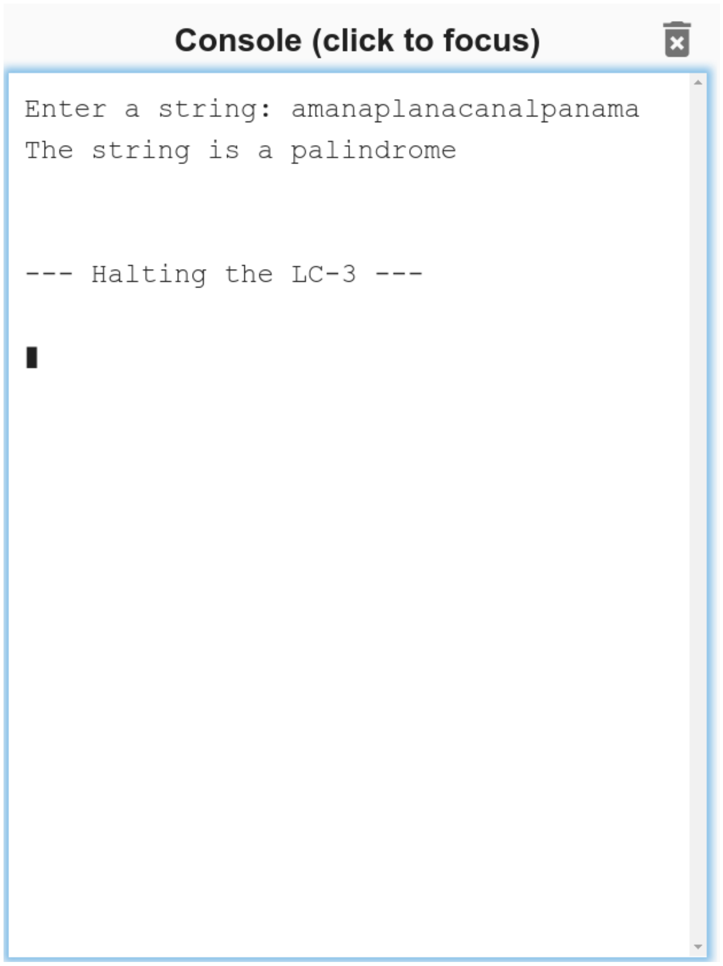

Console (click to focus)

```
Enter a string: abababab
The string is not a palindrome


--- Halting the LC-3 ---

▮
```

4. abababab is not a palindrome

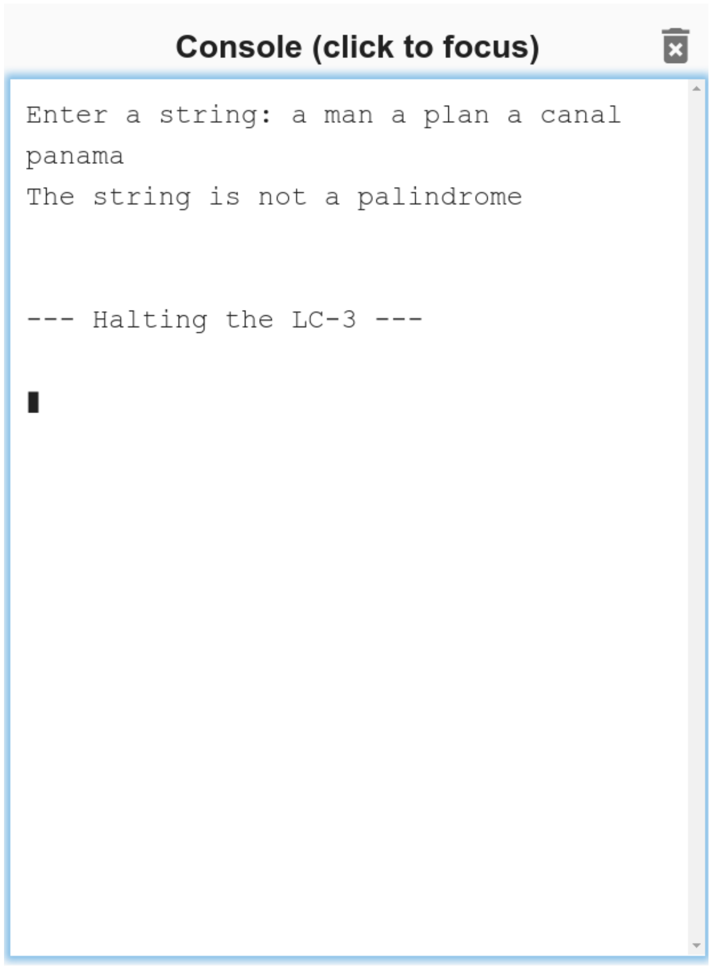

Console (click to focus)

```
Enter a string: amanaplanacanalpanama
The string is a palindrome


--- Halting the LC-3 ---

▮
```

5. “a man a plan a canal panama”, without spaces is a palindrome


Console (click to focus)

```
Enter a string: a man a plan a canal
panama
The string is not a palindrome


--- Halting the LC-3 ---

▮
```

6. “a man a plan a canal panama”, with spaces, is not a palindrome

Submission Instructions

Submit ("Upload") your **assignment5.asm** file *(and ONLY that file!)* to the Programming Assignment 5 folder in Gradescope: the Autograder will run & report your grade within a minute or so.
You may submit as many times as you like - your grade will be that of your last submission.
If you wish to set your grade to a previous submission with a higher score, you may open your "Submission history" and "Activate" any other submission - that's the one we will see.

Rubric

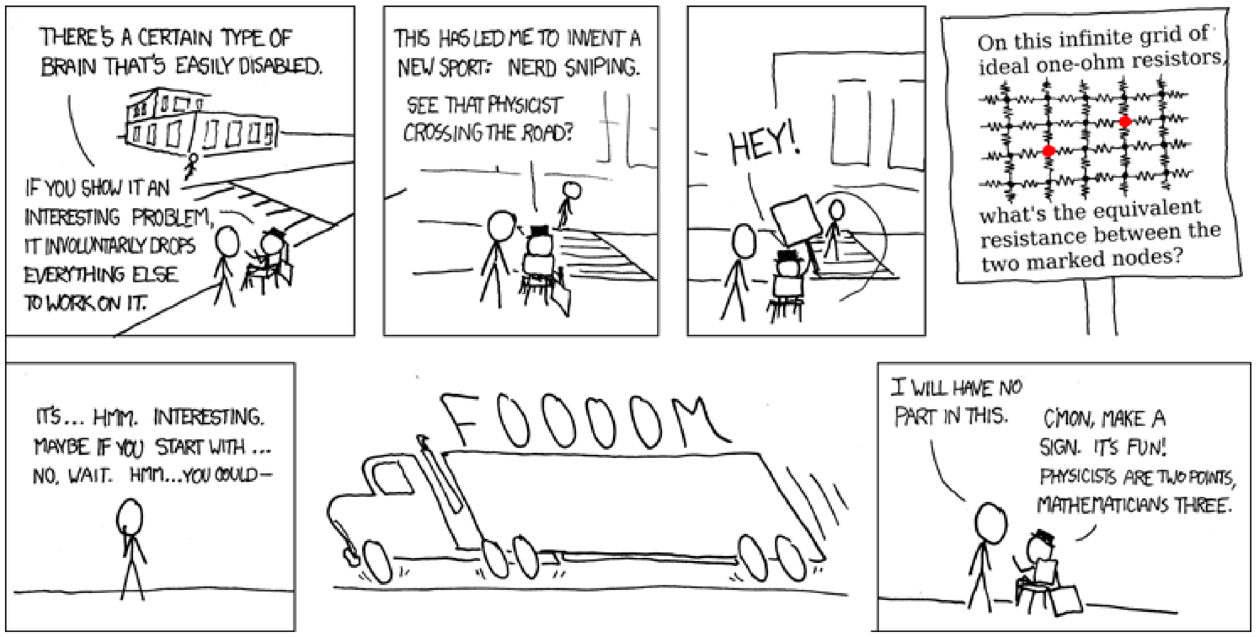
- To pass the assignment, you need a score >= 80.
  The autograder will run several tests on your code, and assign a grade for each.
  But certain errors *(run-time errors, incorrect usage of I/O routines, missing newlines, etc.)* may cause ALL tests to fail which leads to a 0/100! So submit early and study the autograder report carefully!!

- If you pass all tests you will receive a grade of 70/100

- The last 30 points are given for following all requirements, including using subroutine headers (5 points) and the palindrome subroutine using recursion (25 points). Notice that not using recursion in the palindrome subroutine will result in a maximum score of 75 points. This score is below the minimum passing grade.

- **You must use the template we provide** - if you make <u>*any*</u> changes to the provided starter code, the autograder may not be able to interpret the output, resulting in a grade of 0.

One last XKCD comic for the quarter!


THERE'S A CERTAIN TYPE OF BRAIN THAT'S EASILY DISABLED.
IF YOU SHOW IT AN INTERESTING PROBLEM, IT INVOLUNTARILY DROPS EVERYTHING ELSE TO WORK ON IT.
THIS HAS LED ME TO INVENT A NEW SPORT: NERD SNIPING.
SEE THAT PHYSICIST CROSSING THE ROAD?
HEY!
On this infinite grid of ideal one-ohm resistors, what's the equivalent resistance between the two marked nodes?
IT'S... HMM. INTERESTING. MAYBE IF YOU START WITH ... NO, WAIT. HMM...YOU COULD—
FOOOOM
I WILL HAVE NO PART IN THIS.
C'MON, MAKE A SIGN. IT'S FUN! PHYSICISTS ARE TWO POINTS, MATHEMATICIANS THREE.

Source: http://xkcd.com/356/