CS061 – Programming Assignment 07

Objective

The purpose of this assignment is to make use of your familiarity with subroutines and LC3 programming talent to create a program that makes use of advanced looping and counting to build and traverse an array of arrays.

High Level Description

Write three subroutines:

1. INPUT_SENTENCE
2. FIND_LONGEST_WORD
3. PRINT_ANALYSIS

Your Tasks

This programming assignment consists of writing three subroutines and a testbed:

```
;-----------------------------------------------------------------------------------------------------------
; Main code block
; Description: Tests functionality of the system by calling the written subroutines
;-----------------------------------------------------------------------------------------------------------
```

This is the regular old non-subroutine part of your program. It serves as a testbed by calling the subroutines you wrote.

```
.ORIG x3000
; Instructions
   ; load parameters for INPUT_SENTENCE
   ; call INPUT_SENTENCE
   ; load parameters for FIND_LONGEST WORD
   ; call FIND_LONGEST_WORD
   ; load parameters for PRINT_ANALYSIS
   ; call PRINT_ANALYSIS
   HALT
; Local Data
   ; whatever local data you need
.END
```

```
;-----------------------------------------------------------------------------------------------------------
; Subroutine: INPUT_SENTENCE
; Input (R1): The address of where to store the array of words
; Postcondition: The subroutine has collected an ENTER-terminated string of words from
;                the user and stored them in consecutive memory locations, starting at (R1).
; Return Value: None
;-----------------------------------------------------------------------------------------------------------
```

This subroutine is used to store an array of words (not just an array of characters).
Do not store the input as a string of space-separated characters.

Example: If R1 == x5000 and the user enters

```
The quick brown fox[ENTER]
```

Then the subroutine should store the following at x5000:

'T'	'h'	'e'	0	'q'	'u'	'i'	'c'	'k'	0	'b'	'r'	'o'	'w'	'n'	0	'f'	'o'	'x'	0	0

Note the null termination at the end of every word, and the final extra null at the end of the list

```
;---------------------------------------------------------------------------------------------------------------
; Subroutine: FIND_LONGEST_WORD
; Input (R1): The address of the array of words
; Postcondition: The subroutine has located the longest word in the array of words
; Return value (R2): The address of the beginning of the longest word
;---------------------------------------------------------------------------------------------------------------
```

This subroutine starts at the beginning of the array of words, scans the length of each word, and returns the address of the beginning of the longest.

In the example given above, the (first) longest word is "quick", so the subroutine would return x5004 in R2 (i.e. the address of the 'q' in "quick")

```
;---------------------------------------------------------------------------------------------------------------
; Subroutine: PRINT_ANALYSIS
; Input (R1): The address of the beginning of the array of words
; Input (R2): The address of the longest word
; Postcondition: The subroutine has printed out a list of all the words entered as well as the
;                longest word in the sentence.
; Return Value: None
;---------------------------------------------------------------------------------------------------------------
```

This subroutine takes R1 and R2 as parameters. It uses R1 to print out each word of the sentence and R2 to print out the longest word.

Example (and you are officially awesome if you make the output fancy like this):

```
The words in the sentence include: {"The", "quick", "brown", "fox"}
The longest word is: "quick"
```

Uh…help?

We are building an array of words.
A word is an array-of-characters. Thus, we are building an array of array-of-characters. To facilitate not being horribly confused, write out the pseudocode/logic/C++ for how to traverse a two-dimensional array and find the longest item.

<u>Input Validation?</u>
None, don't worry about it . Assume perfectly valid input.

<u>What if there is a tie for longest word?</u>
You are welcome to choose any of the following:

1. Use the first longest word
2. Use the last longest word
3. If you're *really* awesome, figure out how to return an *array* of addresses of the longest words and modify PRINT_ANALYSIS so that it looks like this:
   `The longest words are: {"quick", "brown"}`
   (+1 extra credit point for this)

<u>How do I store an array of words?</u>
Store all the characters of a word, followed by #0 to null-terminate the single word *(this is exactly how .STRINGZ works).*
At the end of the array, store another #0 to null-terminate the array of words. Thus, at the end of

your array, you would have two consecutive #0’s.

Rubric

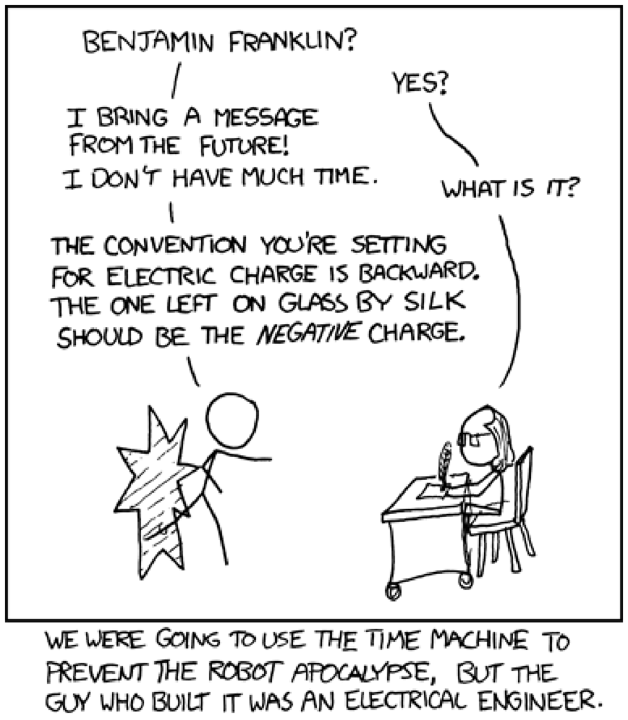
Does anyone even read this part? It’s rather important

- Code does not assemble: (-10 points)
- No header: (-10 points)
- Lacking complete headers on subroutines (-2 points)
- Ghost typing (-1 point)
- Ignoring/not-following specifications: (-??? points depending on severity)

- Well commented code (+2 points)
- Correct input of each word (+2 points)
- Correct storing of the array of words (+2 points)
- Correct output of individual words (+2 points)
- Correct output of longest word (+2 points)

Note: “Correct” means “as stated in the functional specifications”.
Deviation from specs will almost always result in a loss of points.

Comics?!Sweet!!


BENJAMIN FRANKLIN?
I BRING A MESSAGE FROM THE FUTURE! I DON'T HAVE MUCH TIME.
YES?
WHAT IS IT?
THE CONVENTION YOU'RE SETTING FOR ELECTRIC CHARGE IS BACKWARD. THE ONE LEFT ON GLASS BY SILK SHOULD BE THE *NEGATIVE* CHARGE.
WE WERE GOING TO USE THE TIME MACHINE TO PREVENT THE ROBOT APOCALYPSE, BUT THE GUY WHO BUILT IT WAS AN ELECTRICAL ENGINEER.

Source: http://xkcd.com/567/