**CIS 21JA - Lab 8**

New material: 1D and 2D array, string instruction  
Review material: nested procedure call

**Overview**

Write a program that reads in multiple text strings from the user to store in a 2D array. Then the program lets the user search for a letter in the strings.

**Program**  
Download the lab8.asm file, which already has the following:

1. Two constants called ROWS and COLS.  
 ROWS and COLS specify the size of the 2D array:   
 - an array of max ROWS number of strings, currently set to 3  
 - each string has a max length of COLS character, currently set to 11  
To test your program you will need to set these 2 constants to different values.

2. A macro that accepts a text string *address* as input and prints the text string.

Fill in lab8.asm with the following:  
  
1. In the .data section, define a 2D array that is ROWS by COLS size.

Make sure you use the constants to define the array. During testing, when you change ROWS and COLS

to different values, the 2D array should change size accordingly.

Other than the 2D array and text strings, the .data section should not contain any other memory variable.

2. Fill in the main procedure of lab8.asm so it has 3 steps:

Step 1. Call a fillArr procedure, pass arguments through registers

* The HLL pseudocode for the call is:   
   num of strings = fillArr(2D array, prompt string for a string)

Step 2. Check the return value ‘num of strings’:

* End the program if num of strings is 0 (the user didn’t input any string into the 2D array)
* Continue to step 3 if there is at least 1 input string

Step 3. Fill in the body of the letter loop to:

* Call the findCount procedure, pass arguments through the stack
* The HLL pseudocode for the call is:   
   count = findCount(2D array, num of strings, prompt string for a letter)  
   print count

3. Write the fillArr procedure (pass data in registers)

* Loop to prompt the user and read in the string  
  - Make sure to use the macro for the prompt string  
  - Store the string in consecutive rows of the 2D array, one string per row, starting with the first row  
  - The loop ends when there are ROWS number of strings, or when the user input is only the Enter key
* When the loop is finished, call the printArr procedure (passing data through the stack)  
  - The HLL pseudocode for the call is: printArr(2D array, num of strings)

4. Write the printArr procedure (pass data through the stack)

* Loop num of strings time:   
  - Print the string at the current row by using the macro  
  - Go to the next row

 5. Write the findCount procedure (pass data through the stack)

* Prompt the user for a letter and read it in as a character (make sure to print the character)  
  Use the macro to print the prompt.
* Loop through the 2D array:  
  - use *string instruction* to find the letter. There should be no nested loop in the code.  
  - increment the count for each instance of the letter  
  - the loop should only go through the number of strings in the array. If the array has empty rows because there are fewer strings than the max ROWS, don’t force the code to go through the empty rows.
* Return the count

6. Extra Credit (5pts) – Optional:

Step 1: call printLoc

* In the findCount procedure above, after incrementing the count for each instance of the letter, add a call to the printLoc procedure
* The printLoc procedure accepts arguments through the stack.   
  - The procedure prints the location of the found letter in the format: [row index][column index]  
  - The HLL pseudocode for the call is:   
   printLoc(addr of found letter, start addr of array)

Step 2: write printLoc (pass data through the stack)

* Subtract: the address of the found letter - the start addr of the 2D array
* Use the last 2 problems of lab 8 exercise as examples to help you find the row index and column index of the found letter  
  (Hint: you’re essentially doing the reverse of the exercise problems:  
  - you’re finding *the row and column indices from a location*  
  - the exercises find *the location from the row and column indices*)
* Invoke a printIndex macro to print: [row index][column index] format

Step 3: write the printIndex macro

* Accept 1 number (the index)
* Print ‘[‘ , then print the index, then print ‘]’

Special note: when passing the index value to the macro, don’t store the index value in eax to pass it. Use some other register.

**Additional requirements, don't miss them:**

* Procedures should receive input data through register or through the stack as required.
* Except for main, no other procedure should directly access variables defined in .data  
  But constants can be directly accessed by any procedure since the assembler substitutes the constants directly in the code during translation.

**Testing**  
Test the program by running it with different ROWS and COLS values. Here are 2 sample test runs.

Sample output  
Sample 1: user input in blue, ROWS = 8, COLS = 41  
Enter a string: hello there

Enter a string: yep

Enter a string: nope

Enter a string: stop

Enter a string: <Enter key> ; loop ends before 8 times max

hello there ; output of printArr

yep

nope

stop

Enter target letter: e

[0][1] print index is EC and not required

[0][8]

[0][10]

[1][1]

[2][3]

Total count: 5

Enter target letter: y

[1][0]

Total count: 1

Enter target letter: z

Total count: 0

Enter target letter: <control-c>

Sample 2: ROWS = 3, COLS = 11  
Enter a string: 1234

Enter a string: 5

Enter a string: 67 ; loop ends at max 3 ROWS

1234

5

67

Enter target letter: a

Total count: 0

Enter target letter: <control-c>