**CIS 21JA - Lab Assignment 8**

In addition to covering string instructions and 2D arrays, this last lab is also a review of concepts that are covered this quarter. Class notes from earlier modules can be helpful.

**Overview**

Write a program that works with Fibonacci numbers.   
The program fills a 2D array with Fibonacci numbers up to the max count of numbers that the user chooses.  
Then it allows the user to search for a number in the array.

**Background**  
By definition, the Fibonacci sequence starts with the numbers 0 and 1.  
Subsequent numbers are found by adding the previous 2 numbers.  
For example, the 3rd, 4th, and 5th numbers in the sequence are: 0 + 1 = 1, 1 + 1 = 2, 2 + 1 = 3.  
Here are the first 10 numbers: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34

**Program**  
Follow these ordered steps to complete your program:

1. Create 2 constants called NUMROW and NUMCOL that specify the size of the 2D array.  
Make sure these are constants (not memory variables) and define them at the top of the program so they're easily found. To test your program you will need to set these 2 constants to different values, so the easier it is to find them, the faster you can test.

2. Write 2 macros: - a macro that accepts a character as input and prints the character  
 - a macro that accepts a text string as input and prints the text string.  
You can write additional macros if you see the need for them.  
  
3. In the .data section, define a 2D array that is NUMROW and NUMCOL size.  
Make sure you use these constants to define the array, don't use immediate values like 3 or 5. During testing, when you set NUMROW and NUMCOL to different values, the 2D array should change size accordingly.

Example of the 2D array with NUMROW = 3, NUMCOL = 3:  
     0  1  1  
     2  3  5  
     8 13  21  
  
Example of 2D array with NUMROW = 2, NUMCOL = 4:  
    0  1  1  2  
    3  5  8  13

The 2D array has unsigned WORD data, therefore the largest array size is 24 Fibonacci numbers.  
You can assume that the NUMROW and NUMCOL will not specify more than 24 elements in the array.

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

4. Write the main procedure that does 3 tasks:  
A. Loop to call a getCount procedure to ask the user how many Fibonacci numbers he/she wants to see

- The getCount procedure accepts input data through registers and it returns data through registers

- The 2 data values it returns are: the number that the user enters, and a boolean to indicate whether

the number is valid

The loop in main to ask for user input will end when the boolean indicates that the number is valid.

B. Call the procedure fillArray to fill the array with Fibonacci numbers, up to the max count from the user

- The fillArray procedure accepts data through the stack because it will need to use multiple registers

- The fillArray procedure fills the array with Fibonacci numbers and then it calls a printArray procedure to

print all the numbers.

C. Call the procedure search to let the user search for a particular number in the array

5. Write the getCount procedure    (passing data through registers)

- prompt the user and read in the max count of number

- if (max count < 1) or (max count > size of array), return a boolean to indicate invalid

- otherwise return a boolean to indicate valid  
  Remember that the size of the array is dependent on the NUMCOL and NUMROW values

6. Write the fillArray procedure    (passing data through the stack)

- use *string instructions* to fill the 2D array with Fibonacci numbers, up to the max count from the user

- if the max count is less than the array size, don't go above the max count

- when the array is filled, call a printArray procedure to print all the numbers that are filled

-  the printArray procedure also accepts data through the stack

7. Write the printArray procedure     (passing data through the stack, called by fillArray - not by main)

- use *string instructions* to print the data in the array, up to the max count

- the output must correspond to the NUMCOL and NUMROW that make up the array

- for example, if NUMCOL and NUMROW are 3, then the output should show 3 rows and 3 columns, but

only up to the max count of numbers

If the max count is 5:   
           0  1  1  
           2  3

If the max count is 7:  
           0  1  1  
           2  3  5  
           8   
   -  the printed numbers are separated by a space, and you don't have to align the numbers into columns

8. Write the search procedure       (passing data through the stack)

- loop and prompt for a target number until the user enters -1

- use *string instructions* to find the target number

- print the value  -1 if the target is not found

- otherwise print the row and column indexes where the target is found, in the format  [row][col]

**Testing**  
Test the program by running it with a set NUMROW and NUMCOL, then change only the NUMROW and NUMCOL values, run the program again to see that the array has changed accordingly.

Sample output  
Sample 1: user input in blue, NUMCOL = 5, NUMROW = 5  
  How many numbers? 29  
  Must be between 1 and 25  
  How many numbers? 19  
  0 1 1 2 3  
  5 8 13 21 34  
  55 89 144 233 377  
  610 987 1597 2584  
  Target number? (-1 to end search): 3  
  [0][4]  
  Target number? (-1 to end search): 610  
  [3][0]  
  Target number? (-1 to end search): 144  
  [2][2]  
  Target number? (-1 to end search): 133  
  -1  
  Target number? (-1 to end search): -1  
  Press any key to continue . . .  
  
Sample 2: NUMCOL = 8, NUMROW = 3  
  How many numbers? 30  
  Must be between 1 and 24  
  How many numbers? 12  
  0 1 1 2 3 5 8 13  
  21 34 55 89  
  Target number? (-1 to end search): 3  
  [0][4]  
  Target number? (-1 to end search): 44  
  -1  
  Target number? (-1 to end search): 55  
  [1][2]  
  Target number? (-1 to end search): -1  
  Press any key to continue . . .  
  
**Additional requirements**- Use string instructions whenever you need to walk an array and access data.  
   Points will be deducted if your code to walk an array takes more instructions than using a string instruction.  
- Procedures should pass data through register or through the stack as required.  
  Pass addresses of text strings to any procedure that needs to print them.  
- Except for main, no other procedure should directly access data defined in .data by using their names.  
- The program should have the exact number of procedures as required, and they must be called as specified.  
- Other than the 2D and text strings, you should not use any other memory variables.  
- Use bit wise instructions that would be faster than multiply or divide by powers of 2.  
- As usual, documentation is needed.