CSCI-21 Assignment #4 – working with arrays – due 3/21/16

In the Settings menu of SPIM, set Bare Machine OFF, Allow Pseudo Instructions ON, Load Exception File OFF, Delayed Branches ON, Delayed Loads ON, Mapped IO OFF. In QtSpim, you may have a problem caused by the branch delay ON and branch macros, so be careful with your macro use with branches and use only those macros that actually work (no internal branches within the macro).

A carefully thought-out and debugged pseudocode design will cut the time it takes to do these exercises in half. I suggest you also write and debug them in C or C++ first, then translate them into assembler.

Exercise 1 – Conversion to lower case

Declare a string in the data section, for example:

```
        .data
string: .asciiz     "ABCDEFG"
```

Write a program that prints the string, converts the string to all lower case characters and then prints the string again. Do this conversion by adding 0x20 to each character in the string. (See Appendix B to figure out why this works.)

Assume that the data comprises only uppercase alphabetical characters, with no spaces or punctuation.

Exercises 2 and 3 -- Capitalization (2 versions)
Declare a string in the data section such as:

```
        .data
string: .asciiz "in a  hole in the   ground there lived a hobbit"
```

Notice there are extra spaces in this string. Write a program that capitalizes the first letter of each word, so that after running your program the data will look like this:

```
        .data
string: .asciiz "In A  Hole In The   Ground There Lived A Hobbit"
```

Easy version (exercise 2): assume that the data comprises only lower case characters and spaces. There may, however, be several spaces in a row (as in "`the    ground`" above). Be sure to capitalize only the first letter of the words.

Medium-hard version (exercise 3): Copy and modify your program so that it assumes that the data comprises only upper and lower case characters and spaces, and alters a character only if it is lower case, and follows a space or is the first character on the line.

For both versions, print the strings before and after translation by using the syscall print string service. For more info on syscall services, see pages 21 and 22 in the text.

Exercise 4:

Prompt the user for a string and then parse the string exchanging the case of each character. You may assume the string comprises only lower- and upper-case alphabetic characters and spaces, with no punctuation. Change all upper-case letters to lower case and all lower-case letters to upper case. For example, if the user enters this:

```
The Quick BrOwn foX
```

change it to this:

```
tHE qUICK bRoWN FOx
```

and print the resulting string immediately below the old (input) string. Do not re-print the input string.

Exercise 5:

Prompt the user for a series of 10 integers and hold them in an integer array. Then, after the entries are complete, pass through the array finding and printing the largest and smallest entries. Do not track the largest and smallest values as they are being entered. For example, if the user enters this series (you will have to prompt for these on separate input lines):

```
12   8    11   32   20   1    29   6    19   12
```

print this:

```
Smallest: 1
Largest: 32
```

Exercise 6:

Prompt the user for a series of 10 integers and hold them in an integer array. Print the array with tabs between the elements, then reverse the elements in the array and print the integers again (with tabs) in the resulting reversed order. Do not just pass through the array in reverse order to print it. Exchange the actual data elements in the array.

For example, if the user enters this series (again, you will prompt for these on separate input lines):

```
12   8    11   32   20   1    29   6    19   12
```

print this (on two lines as shown):

```
12   8    11   32   20   1    29   6    19   12
12   19   6    29   1    20   32   11   8    12
```

Exercise 7:

Prompt the user for a series of 10 integers and hold them in an integer array. Print the array with tabs between the elements, then sort the elements in the array and print the integers again (with tabs) in the resulting sorted order. You may use any simple $O(n^2)$ sort, like bubble sort or ripple sort.

For example, if the user enters this series (again, you will prompt for these on separate input lines):

```
12    8     11    32    20    1     29    6     19    12
```

print this (on two lines as shown):

```
12    8     11    32    20    1     29    6     19    12
1     6     8     11    12    12    19    20    29    32
```

You may combine exercises 5, 6 and 7 into one program, prompting once and then (if needed) copying the values into a second array to work on them.