CMPS 351: Spring 2015 Program #5
Date assigned: Thursday, April 2, 2015 Due: Thursday, April 30, 2015

For this program, you will write MIPS assembly language functions to get and set the integer data in a one-dimensional array. Your functions should take at least two parameters; the array being accessed and the cell to access. For the setter function, you will also need the data to store in the selected cell. The array has the format where the length of the array is stored as the first element in the array and all of the data is found in the subsequent cells.

Since it is possible to request access to cells outside the array, you will also need to do some error checking. You will first want to make sure that the provided pointer appears to indicate a valid data structure. This means that the array should be within data memory. Since you will be working with an array of integers, you can also check to make sure that the end of the array is still inside the valid region of memory. Before you attempt to access the physical memory, you should also perform a bounds check to ensure that the user has requested a cell that is represented by the current array.

Your program will not need to interact with the console. Instead, you will use a test suite that has been provided for you. The test suite will call your functions (get / set) with the parameter ($a0 contains the address of an array, $a1 contains the cell number and $a2 may contain some data) and will wait for your function to return its results in $v0 if necessary. The test suite will also tell you whether your function has performed the expected actions. Finally, the test suite will also test to make sure that your program can handle several types of malformed data structures and requests.

To get your program to run, you will need to include the provided function. However, you should submit your code without the extra function. You can combine your code to that of the necessary function by running the batch script provided on Moodle with your file name as a parameter (ex: P5Combine.bat "pde1373 – Program #5.asm") or by using drag & drop. The script will place the result in a file named output.asm that can be loaded into QTSPIM and run as normal. You will need to rerun this script every time you edit your program so that the output file has your latest code. Alternatively, you can use an equivalent command to concatenate the code from each of the files and place it in an output file.
Unix: cat <Program #5 Function Name>.asm "Program #5 - Test Suite.asm" > <output>.asm

Your program should include appropriate comments indicating what the code should be doing and what registers are being used for. After displaying the results, your program should exit cleanly. Please include your name and CLID in the program headers and include your CLID in the file names. Your programs should be turned in through Moodle before class starts on the due date. You should test your programs using the QT SPIM simulator to ensure their functionality before submitting them.

Expected output:

```
----------Starting functionality tests.----------
Test #1 passed:  Testing the read function for a single element array.
Test #2 passed:  Testing the read function for a 2 element array.
Test #3 passed:  Testing the read function for the second element of a 2
element array.
Test #4 passed:  Repeating test #2 to make sure nothing changed.
Test #5 passed:  Repeating test #3 to make sure nothing changed.
Test #6 passed:  Checking the 16th element of a 33 element array.
Test #7 passed:  Checking the 8th element of a 33 element array.
Test #8 passed:  Checking the 33rd element of a 33 element array.
Test #9 passed:  Checking the 1st element of a 33 element array.
Test #10 passed:  Setting the 1st element of a single element array.
```

```
Test #11 passed:  Setting the 1st element of a 2 element array.
Test #12 passed:  Setting the 2nd element of a 2 element array.
Test #13 passed:  Repeating test #11.
Test #14 passed:  Repeating test #12.
Test #15 passed:  Testing the get and set functions with a large data set.
Test #16 passed:  Testing the get and set functions with a large data set.
Test #17 passed:  Testing the get and set functions with a large data set.
Test #18 passed:  Testing the get and set functions with a large data set.
Test #19 passed:  Testing the get and set functions with a large data set.
Test #20 passed:  Testing the get and set functions with a large data set.
Test #20 passed:  Testing the get and set functions with a large data set.
Test #22 passed:  Testing the get and set functions with a large data set.
Test #23 passed:  Testing the get and set functions with a large data set.

----------Starting parameter checking tests.----------
Test #101 passed:  Null pointer check. (get)
Test #102 passed:  Null pointer check. (set)
Test #103 passed:  Array pointer below zero. (get)
Test #104 passed:  Array pointer below zero. (set)
Test #105 passed:  Pointer in text range. (get)
Test #106 passed:  Pointer in text range. (set)
Test #107 passed:  Pointer too high for dynamic data memory. (get)
Test #108 passed:  Pointer too high for dynamic data memory. (set)
Test #109 passed:  Pointer is not word aligned. (get)
Test #110 passed:  Pointer is not word aligned. (get)
Test #111 passed:  Pointer is not word aligned. (get)
Test #112 passed:  Pointer is not word aligned. (set)
Test #113 passed:  Pointer is not word aligned. (set)
Test #114 passed:  Pointer is not word aligned. (set)
Test #115 passed:  Negative array length. (get)
Test #116 passed:  Negative array length. (set)
Test #117 passed:  Zero array length. (get)
Test #118 passed:  Zero array length. (set)
Test #119 passed:  Array extends outside of memory. (get)
Test #120 passed:  Array extends outside of memory. (set)
Test #121 passed:  Array wraps around to the beginning of memory. (get)
Test #122 passed:  Array wraps around to the beginning of memory. (set)
Test #123 passed:  Array is inside of the memory range, but attempts to
cross from the heap to the stack. (get)
Test #124 passed:  Array is inside of the memory range, but attempts to
cross from the heap to the stack. (set)
Test #125 passed:  Array barely stays below the stack. (get)
Test #127 passed:  Array index is negative. (get)
Test #128 passed:  Array index is negative. (set)
Test #129 passed:  Array index is too big for the array. (get)
Test #130 passed:  Array index is too big for the array. (set)
Test #131 passed:  Array index is negative. (get)
Test #132 passed:  Array index is negative. (set)
----------Testing completed.----------
```

Objectives:

1. To practice building functions in the MIPS assembly language.
2. To introduce and practice working with arrays.
3. To introduce and practice working with pointers.

4. To introduce and practice working with objects.