```
##################################################################################
# Ethan West, CS 2318-253, Assignment 2 Part 1 Program B           #
##################################################################################
# MIPS assembly program to carry out the following tasks, in order:
#   - Allocate a global array (i.e., space in the data segment) enough
#     for storing 4 integers and initialize the array (from 1st to 4th
#     element) with 888, 111, 333 and 222 at the same time (i.e., DON'T
#     first allocate uninitialized space for array and later write code
#     to put the values into array).
#
#   - Display a labeled output about the array's initial contents
#     (in the order from 1st to 4th element).
#
#   - IMPORTANT (for the purpose of this exercise):
#     -  You are to load the values of the array elements from memory
#        and use those values to generate the labeled output. (You are
#        not to simply display a hard-coded string(s) and values
#        showing what the contents of the array should look like.)
#
#   - Re-order the values in the array so that the contents of the array
#     in memory (from 1st to 4th element) eventually becomes 333, 222, 888
#     and 111, using the following operations in the order listed (to not
#     defeat the goals of this exercise, you must NOT change the specified
#     operations and order, even if  doing so will accomplish the same
#     effect more efficiently):
#
#     - Swap the contents in memory of the 1st and 3rd elements of the
#       initial array.
#
#       - NOTE: Contents of the array in memory (from 1st to 4th
#         element) after this first swapping operation should be 333,
#         111, 888 and 222.
#     - Swap the contents in memory of the 2nd and 4th elements of the
#       array that results after the preceding first swap.
#
#       - NOTE: Contents of the array in memory (from 1st to 4th
#         element) after this second swapping operation should
#         now and finally be 333, 222, 888 and 111.
#
#   - IMPORTANT (for the purpose of this exercise):
#     - When performing each of the three swap operations above, you can
#       re-use (where expedient) the array's base address in register
#       (loaded when performing the display of the array's initial
#       contents)but you MUST re-load the values of the associated
#       array elements fresh from memory (i.e., assuming no knowledge
#       that certain values might have already existed in some registers
#       due to prior operations).
#
#   - Display a labeled output about the array's contents (in the order
#     from 4th to 1st element) after the 2 swapping operations above.
#
```

```
#     - NOTE: The contents of the array's elements are to appear (when
#       displayed) in the order from 4th to 1st element and not from 1st
#       to 4th element.
#
#   - (IN CASE YOU WONDER: Order in which the eventual contents of the
#     array should appear in this output -->  111, 888, 222 and 333.)
#
#   - IMPORTANT (for the purpose of this exercise):
#     - When displaying the after-swap labeled output, you can re-use
#       the array's base address in register (loaded when performing
#       prior operations) but you MUST re-load the values of the array
#       elements fresh from memory (i.e., assuming no knowledge that
#       certain values might have already existed in some registers
#       due to prior operations).
#
#   - CAUTION:
#     - Too many past students regretted having points taken off for
#       not labeling output.
#
############################## data segment ##################################

            .data
intArr:         .word 888, 111, 333, 222    # global int array of size 4 initialized
                            #  t0 888, 111, 333, 222 (from 1 - 4)
indexZero:      .asciiz "[0]: "         # String helpers for use in labeling
indexOne:       .asciiz "[1]: "
indexTwo:       .asciiz "[2]: "
indexThree:     .asciiz "[3]: "

            .text
            .globl main
main:
            la $t0, intArr          # $t0 has adress of intArr

            # BEGIN_(Printing of initial order of array)
            lw $t1, 0($t0)          # $t1 has adress of intArr[0]
            li $v0, 4
            la $a0, indexZero       # Print "[0]: " lable
            syscall
            li $v0, 1
            move $a0, $t1           # Print intArr[0]
            syscall

            li $v0, 11          # new line
            li $a0, '\n'
            syscall

            lw $t2, 4($t0)          # $t2 has adress of intArr[1]
            li $v0, 4
            la $a0, indexOne        # Print "[1]: " lable
            syscall
            li $v0, 1
            move $a0, $t2           # Print intArr[1]
            syscall

            li $v0, 11          # new line
            li $a0, '\n'
            syscall

            lw $t3, 8($t0)          # $t3 has adress of intArr[2]
            li $v0, 4
            la $a0, indexTwo        # Print "[2]: " lable
            syscall
            li $v0, 1
            move $a0, $t3           # Print intArr[2]
            syscall

            li $v0, 11          # new line
            li $a0, '\n'
            syscall

            lw $t4, 12($t0)         # $t4 has adress of intArr[3]
            li $v0, 4
            la $a0, indexThree      # Print "[3]: " lable
            syscall
            li $v0, 1
            move $a0, $t4           # Print intArr[3]
            syscall
            # END_(Printing of initial order of array)

            li $v0, 11  # new line
            li $a0, '\n'
            syscall

            # BEGIN_(Swap 1st and 3rd elements)
            sw $t1, 8($t0)
            sw $t3, 0($t0)
            # END_(Swap 1st and 3rd elements)

            # BEGIN_(Swap 2nd and 4th elements)
            sw $t4, 4($t0)
            sw $t2, 12($t0)
            # END_(Swap 2nd and 4th elements)

            li $v0, 11  # new line
            li $a0, '\n'
            syscall

            # BEGIN_(Printing of final order of array)
            lw $t4, 12($t0)         # $t4 has adress of intArr[3]
            li $v0, 4
            la $a0, indexThree      # Print "[3]: " lable
            syscall
            li $v0, 1
            move $a0, $t4           # Print intArr[3]
            syscall

            li $v0, 11  # new line
            li $a0, '\n'
            syscall

            lw $t3, 8($t0)          # $t3 has adress of intArr[2]
            li $v0, 4
            la $a0, indexTwo        # Print "[2]: " lable
            syscall
            li $v0, 1
            move $a0, $t3           # Print intArr[2]
            syscall

            li $v0, 11  # new line
            li $a0, '\n'
            syscall

            lw $t2, 4($t0)          # $t2 has adress of intArr[1]
            li $v0, 4
            la $a0, indexOne        # Print "[1]: " lable
            syscall
            li $v0, 1
            move $a0, $t2           # Print intArr[1]
            syscall

            li $v0, 11          # new line
            li $a0, '\n'
            syscall

            lw $t1, 0($t0)          # $t1 has adress of intArr[0]
            li $v0, 4
            la $a0, indexZero       # Print "[0]: " lable
            syscall
            li $v0, 1
            move $a0, $t1           # Print intArr[0]
            syscall

            li $v0, 11          # new line
            li $a0, '\n'
            syscall
            # END_(Printing of final order of array)

            li $v0, 10      # graceful exit
            syscall
#################################################


```