```
############################################################
#           cs315 Week 4 - part 2
#
#   ->  Arrays (static & dynamic)
#
############################################################

Concept of arrays should be familiar from previous programming courses (e.g. Java, C++, Python)

Three pieces of information are needed for an array:
    1) base address
    2) length
    3) number of bytes per element (also referred to as 'element size')

Keeping track of both base address and length is absolutely necessary in MIPS
    * base and length are two separate values
    * without a base address, we don't know where in memory the array starts
    * without a length, we don’t know where in memory the array ends
    * if either the base address or the length is unknown, we cannot safely work with the array
    * the element size may be assumed if we know what data type is in the array

Ex: we may know that a given array is an array of words and may therefore assume that its element size is 4 (i.e. 4 bytes per word)

Index address calculation
    * to access an element in an array, we must know its memory address (index address)
    * address can be calculated given the array base address, the index, and the element size
        i = b + s * n

    i -> index address (i.e. address of element at index n)
    b -> array base address
    s -> element size in bytes
    n -> index number (starting from 0)

Note:   do not mix arithmetic in different bases (i.e. do not add hex with decimal)
        remember that addresses are in base 16

Ex: for an array of words starting at address 0x10010040, calculate the address of index 3
    b -> 0x1001 0040    # base address given in problem
    s ->  4             # words are 4 bytes each
    n ->  3             # index number given in problem

    i = b + s * n = b + 410 * 310 = b + 12 = b + 0x0000 000c = 0x1001 0040 + 0x0000 000C = 0x1001 004C
                                         ^               ^
                                    in base 10      in base 16

Ex: for an array of doubles starting at address 0x1001 0594, calculate the address of index 2
    b -> 0x1001 0594
    s -> 8              # doubles are 2 words (8 bytes)
    n -> 2
    i = b + s * n = b + 8 * 2 = b + 16 = b + 0x0000 0010 = 0x1001 0594 + 0x0000 0010 = 0x1001 05A4
                                     ^               ^
                                in base 10      in base 16


~ Static arrays vs. Dynamic arrays

Static arrays:
    * located in static memory
    * declared in a .data section
    * array length is known before the program starts running (at 'compile time')
    * will be the same length every time the program runs
```

```
Dynamic arrays
    * located in dynamic memory
    * allocated (created) with system call 9
    * NOT declared anywhere
    * length is not known until after program is running (at 'run time')
    * may not be the same for each program run

Static arrays:
    Declaring an array with n number of words
    [label]:    .word   w0, w1, w2, ..., w9     # declare static array of size 10 of words (32 bits each) AND  stores w0, w1, w2, ..., w9 in successive word locations
    [label]:    .word   w:n                     # declare static array of size 10 of words and initialize them to n

Ex:
    myFirstArray:   .word 7, 8, 9, 10, 11       # array with length 5 ([7 8 9 10 11])
    mySecondArray:  .word 0:3                   # array with length 3 ([0 0 0])

Static array base addresses:
    * base address is bound to the array's label
    * retrieved with 'la' command (load address)

Ex: if myArray is a static array, its base address can be loaded into $a1 with 'la $a1, myArray'

Dynamic arrays
    * created using system call 9 (dynamic allocation)
    * load $a0 before syscall   # $a0 specifies array size in BYTES
    * read $v0 after syscall    # base address is returned in $v0
    * the system call is the ONLY time we will be given the base address (DO NOT LOSE IT!)
    * length of the dynamic array must be recorded
    * store length in a word (Ex: myArrayLength)

* How to save a dynamic array's base address:
    * declare a static word variable to hold the base address
    * store the base address at the word after the system call allocates the array

Ex: create a dynamic array of 10 words and store its base
        .data
myBaseHolder:   .word   0       # declare static word to hold base
        .text
    ...
    li $v0, 9               # specifies system call 9
    li $a0, 40              # 10 words requires 40 bytes
    syscall                 # system call will return base address in $v0

    lw $t9, myBaseHolder    # load address of myBaseHolder
    sw $v0, 0($t9)          # store returned base address at myBaseHolder


* How to retrieve a dynamic array's base address? read the base address from the word where it has been stored

NOTE: The address of the word IS NOT the base address of the array!

Reading the stored base address is a two-step process
    1) load the address of the word holding the base
    2) read from that address to retrieve the base

Ex: assume a base address has been stored in a word named myBaseHolder
    la $t9, myBaseHolder    # load address of myBaseHolder into $t9

    lw $t0, 0($t9)          # load value from myBaseHolder into $t0

Thus:
    # $t9 has the address of myBaseHolder
    # $t0 has the base address of the array
```

Using arrays:
    * static and Dynamic arrays are used the same way
    * once created, the only difference between static and dynamic arrays is 'where' they are in memory (i.e. static memory or dynamic memory)

Address calculation:
    * calculated using i = b + s * n

Write code to do calculation (i.e. multiply s and n, then add result to b)
    Ex: given an array of words whose base address is in $t0, calculate the address of the index number specified in $t1

```
# $t0 -  b (base address)
# $t1 -  n (index number)
# $t2 -  s (element size)
# $t3 -  i (index address to be calculated)

li $t2, 4           # load 's'
mul $t3, $t2, $t1   # $t3 <-- 's' * 'n'
add $t3, $t0, $t3   # $t3 <-- 'b' + 's' * 'n'
```