Arrays

First step is to reserve sufficient space for the array.

Array elements are accessed via their addresses in memory, which is convenient if you’ve given the `.space` directive a suitable label.

```
.data
list:   .word 2, 3, 5, 7, 11, 13, 17, 19, 23, 29
size:   .word 10
...

lw    $t3, size
la    $t1, list    # get array address
li    $t2, 0       # set loop counter

print_loop:
    beq    $t2, $t3, print_loop_end  # check for array end

    lw    $a0, ($t1)   # print value at the array pointer
    li    $v0, 1
    syscall

    addi   $t2, $t2, 1  # advance loop counter
    addi   $t1, $t1, 4  # advance array pointer
    j    print_loop  # repeat the loop

print_loop_end:
```
Array Example

This is part of the palindrome example from the course website:

```
.data
string_space: .space 1024
...
# prior to the loop, $t1 is set to the address of the first
# char in string_space, and $t2 is set to the last one

  test_loop:
      bge $t1, $t2, is_palin  # if lower pointer >= upper
      #   pointer, yes

      lb $t3, ($t1)            # grab the char at lower ptr
      lb $t4, ($t2)            # grab the char at upper ptr
      bne $t3, $t4, not_palin  # if different, it's not

      addi $t1, $t1, 1          # advance lower ptr
      addi $t2, $t2, -1         # advance upper ptr
      j  test_loop              # repeat the loop
```
Example 1: Array Traversal in C

```c
// PrintList.c
#include <stdio.h>

int main() {
    int Sz = 10;
    int Array[10] = {1, 1, 2, 3, 5, 8, 13, 21, 34, 55};

    int Pos = 0;
    while ( Pos < Sz ) {
        printf("%3d:  %d\n", Pos, Array[Pos]);
        ++Pos;
    }
}
```
Example 1: Array Traversal in MIPS

```assembly
# PrintList.asm
.data
Sz: .word 10
Array: .word 1, 1, 2, 3, 5, 8, 13, 21, 34, 55
NL: .asciiz "\n"

.text
main:
    lw $s7, Sz # get size of list
    move $s1, $zero # set counter for # of elems printed
    move $s2, $zero # set offset from Array

print_loop:
    bge $s1, $s7, print_loop_end # stop after last elem is printed
    lw $a0, Array($s2) # print next value from the list
    li $v0, 1
    syscall
    la $a0, NL # print a newline
    li $v0, 4
    syscall

    addi $s1, $s1, 1 # increment the loop counter
    addi $s2, $s2, 4 # step to the next array elem
    j print_loop # repeat the loop

print_loop_end:
```
Example 2: C Bubblesort

```c
int main() {

    int Sz = 10;
    int List[10] = {17, 5, 92, 87, 41, 10, 23, 55, 72, 36};

    int Stop,   // $s3: upper limit for pass
         Curr,   // $s0: index of current value in comparison
         Next,   // $s1: index of successor to current value
         Temp;   // $s2: temp storage for swap

    for (Stop = Sz - 1; Stop > 0; Stop--) {
        for (Curr = 0; Curr < Stop; Curr++) {
            Next = Curr + 1;
            if ( List[Curr] > List[Next] ) {
                Temp    = List[Curr];
                List[Curr] = List[Next];
                List[Next] = Temp;
            }
        }
    }
}
```
Example 2: Analysis

```c
int main() {
    ...
    int Stop, Curr,
        Next,
        Temp;
    for (Stop = Sz - 1; Stop > 0; Stop--) {
        for (Curr = 0; Curr < Stop; Curr++) {
            Next = Curr + 1;
            if ( L[Curr] > L[Next] ) {
                Temp    = L[Curr];
                L[Curr] = L[Next];
                L[Next] = Temp;
            }
        }
    }
}
```

- data declarations as before
- $s3$: upper limit for pass
- $s0$: counter for inner loop
- $s1$: offset of current elem
- no need for these
- $t7$: current value
- $t8$: next value

We need to map arguments and variables to registers, and identify any additional registers needed.
Example 2: MIPS Bubblesort

.data
Sz:    .word  10
List:   .word  17, 5, 92, 87, 41, 30, 23, 55, 72, 36

.text
main:
################################################################# bubble_sort
   lw   $s3, Sz          # set outer loop limit
   addi  $s3, $s3, -1

outer:                                 # outer bubble-sort loop
   bge  $zero, $s3, outer_end
   li    $s0, 0          # set inner loop counter
   li    $s1, 0          # set current element offset

   ## inner loop goes here ##

   addi  $s3, $s3, -1      # decrement outer loop limit
   j     outer              # restart outer loop

outer_end:
Example 2: MIPS Bubblesort

```mips
## see preceding slide for surrounding code
inner:                           # inner bubble-sort loop
    bge $s0, $s3, inner_end

    lw  $t7, List($s1)            # get current element
    lw  $t8, List + 4($s1)        # get next element

    ble $t7, $t8, no_swap
    sw  $t8, List($s1)
    sw  $t7, List + 4($s1)

    no_swap:
        addi $s1, $s1, 4
        addi $s0, $s0, 1               # increment inner loop counter
        j     inner                   # restart inner loop

inner_end:
```