Arrays in C

Example:

```c
int a[50];

a[15] = a[7];
```

In C:

```c
a[15] = a[7];
```

In MIPS?

There are no “arrays” in MIPS.

E.g. `$s0` holds starting address of array `a[ ]` in Memory.

**NOTE:** You cannot transfer data directly between memory addresses.

How to translate this into MIPS?

```mips
lw $t0, 28($s0) # a[7]
sw $t0, 60($s0) # a[15]
```

Another Example

```c
m = a[i]; // C instruction
```

```mips
sll $t0, $s2, 2 # offset = i * 4
add $t0, $s0, $t0 # base + offset
lw $s1, 0($t0)
```

How to manipulate single bytes in Memory?

Recall “lw” and “sw”. There is also a load byte “lb” and a store byte “sb” instruction.

1 word = 4 bytes

Strings in C (COMP 206)

- stored as consecutive bytes (essentially the same as an array of char)
- ASCII coded
- terminated with null char (0 in ASCII, we write ‘\0’)

```c
char *str; // Declare a pointer to a string.
str = "COMP 273";
```

1 word = 4 bytes

Better picture of what's going on....
Count the number of chars in a string (C)

```c
char *str;  // Declare a pointer to a string.
// str is an address (a 32 bit number).
int ct = 0;
str = "COMP 273";
while ( *(str + ct) != '\0' ){ // coming soon in COMP 206
    ct++;
}
```

C CODE

```c
str = "COMP 273";
while ( *(str + ct) != \"\0\"){
    ct++;
}
```

MIPS CODE

```mips
# load the address where string begins
# initialize ct to 0 (use a register)
loop: # compute address of Memory byte to examine next
# load that byte into a register
# if that byte is \"\0\", branch to exit
# increment ct
# jump back to "loop"
exit:
```

Q: How to get data into and out of Memory?

A: 1) "assembler directives"
2) "system calls"

Assembler Directives (Example)

```assembly
.data
str : .asciiz "I love COMP 273"
.text
.globl main
main:
```

str is a label that aids in programming. Think of it as a label for an address (similar to the "Exit" labels that we saw in conditional branches earlier).
More Assembler Directives

y0 : .word -17
b0 : .byte 0xd, 62, -3 # signed
b1 : .byte 250 # out of range
arr0 : .space 1400
y1 : .word 0x2c24

Example: swap

C code

tmp = y0;
y0 = y1;
y1 = tmp;

MIPS code

This code assumes that the variables are already in registers.

move $t0, $s0
move $s0, $s1           # "move" is a pseudoinstruction
move $s1, $t0

Q: How to get data into and out of Memory?

A: 1) "assembler directives"
   2) "system calls"

Example: print a string

la $a0, str
li $v0, 4          # li is a pseudoinstruction "load immediate"
# ori $v0, $zero, 4 is the real instruction
syscall

System calls ("syscall" instruction) uses the console.

syscall

This instruction uses registers $2, $4, $5 which you can also write $v0 and $a0, $a1, respectively.
Example: read a string from console

li $v0, 8      # code for reading string is 8
add $a0, $zero, $s1      # $s1 specifies address
                     # where string will start
la $t0, sizeBuffer        # specifies a buffer size (see A2)
lw $a1, 0($t0)          # load that buffer size.
syscall

The OS/kernel stops the program and waits for a string to be typed into
the console (hitting "enter" signals the end of the string, or max length is
reached). The string is then written from the buffer into Memory starting
at address specified by $s1. Only the string is written (not the whole
buffer size). Then the program continues.

The addresses and strings are all in Memory.

The strings below are also stored in Memory

"I love assembly language"
"even more than I love Java or C."
"I am so glad that I am taking COMP 273"
"because I am learning so much."

Assignment 2 posted today
Task: manipulate an array of string references (addresses).

Assignment 2: two parts
1) read in a list of strings from the console (loop)
   - store the strings in Memory
   - store the addresses of the strings in an array in Memory
     (this array is a list)
2) manipulate the list of strings using "move to front"
   - user enters an index i, and the i-th string address is
     moved to the front

[EDITED Feb 21] It is important to understand where your
variables are in Memory. Note we use assembler
directives to assign Memory for:
- maxLengthString       (integer i.e. 1 word)
- stringReferenceArray  (5 words)
- strings                (100 bytes)
- prompts    e.g. "enter maximum length of a string:"
                "enter a string:"
                "move to front index:"

The following slide shows how they are layed out, starting
at address 0x10010000. Note in MARS the addresses
increase to right and down (opposite from slides).