(May 10) - Last Day to withdraw from class.

"W"

- Arrays / Strings
- Pointers

"Suddenly ad scope"

```c
#define five 5
```

PUBLIC

// Declarations and function definitions

IMPLEMENTATION

"static"

C code

PRAGMA MY_CODE Level WARNING "MAYBE OK"

OTHER CREATORS

IMPLEMENTATION CODE

-- TEST UTILITIES (UART TEST)

void main(int)

```c
void main(void)
```
Public function in file (prototype)

Private function in file (prototype and implementation)

Pseudo-globals variable in file 'main'
```c
for (int i = 0; i < MAXEXP; i++) {
    for (int j = 0; j < MAXEXP; j++) {
        ...
    }
}
```

```c
int ADC_init(void) {
    ADcName = \n    ADcValue = ADcInit;  
    
    \[\] \text{WRAPPEP Function:}
    \] GetADCValue (vuln) {
        return (ADcValue);
    }
}
```
Arrays and Strings

Gabriel Hugh Elkaim
Spring 2013
Arrays

Definition

Arrays are variables that can store many items of the same type. The individual items known as elements, are stored sequentially and are uniquely identified by the array index (sometimes called a subscript).

- Arrays:
  - May contain any number of elements
  - Elements must be of the same type
  - The index is zero based
  - Array size (number of elements) must be specified at declaration

Syntax Example

Arrays are declared much like ordinary variables:

Syntax

```c
type arrayName[size];
```

- `size` refers to the number of elements
- `size` must be a constant integer

Example

```c
int a[10]; // An array that can hold 10 integers
char s[25]; // An array that can hold 25 characters
```
Arrays

How to Initialize an Array at Declaration

Arrays may be initialized with a list when declared:

Syntax

\[
\text{type} \ arrayName[size] = \{item_1, \ldots, item_n\};
\]

- The items must all match the \textit{type} of the array

Example

\[
\begin{align*}
\text{int} & \ a[5] = \{10, 20, 30, 40, 50\}; \\
\text{char} & \ b[5] = \{'a', 'b', 'c', 'd', 'e'\};
\end{align*}
\]

Arrays

How to Use an Array

Arrays are accessed like variables, but with an index:

Syntax

\[
arrayName[index]
\]

- \textit{index} may be a variable or a constant
- The first element in the array has an index of 0
- C does not provide any bounds checking

Example

\[
\begin{align*}
\text{int} & \ i, a[10]; \ //An \ array \ that \ can \ hold \ 10 \ integers \\
\text{for} & \ (i = 0; i < 10; i++) \ { \\
& \ a[i] = 0; \ //Initialize \ all \ array \ elements \ to \ 0 \\
} \\
& \ a[4] = 42; \ //Set \ fifth \ element \ to \ 42
\end{align*}
\]
Arrays

Creating Multidimensional Arrays

Add additional dimensions to an array declaration:

Syntax

```
type arrayName[size_1]...[size_n];
```

- Arrays may have any number of dimensions
- Three dimensions tend to be the largest used in common practice

Example

```
int a[10][10];        //10x10 array for 100 integers
float b[10][10][10];  //10x10x10 array for 1000 floats
```

Arrays

Initializing Multidimensional Arrays at Declaration

Arrays may be initialized with lists within a list:

Syntax

```
type arrayName[size_0]...[size_n] = 
  {{item,...,item},
   ...,
   {item,...,item}};
```

Example

```
char a[3][3] = {{'X', 'O', 'X'},
               {'O', 'O', 'X'},
               {'X', 'X', 'O'}};
int b[2][2][2] = {{{0, 1},{2, 3}},{{4, 5},{6, 7}}};
```
Arrays

Visualizing 2-Dimensional Arrays

```c
int a[3][3] = {
    {0, 1, 2},
    {3, 4, 5},
    {6, 7, 8}
};
```

Row, Column

```
a[y][x]
```

<table>
<thead>
<tr>
<th>Row 0</th>
<th>Row 1</th>
<th>Row 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a[0][0] = 0;</td>
<td>a[1][0] = 3;</td>
<td>a[2][0] = 6;</td>
</tr>
<tr>
<td>a[0][1] = 1;</td>
<td>a[1][1] = 4;</td>
<td>a[2][1] = 7;</td>
</tr>
<tr>
<td>a[0][2] = 2;</td>
<td>a[1][2] = 5;</td>
<td>a[2][2] = 8;</td>
</tr>
</tbody>
</table>

Arrays

Visualizing 3-Dimensional Arrays

```c
int a[2][2][2] = {
    {{0, 1}, {2, 3}},
    {{4, 5}, {6, 7}}
};
```

Plane, Row, Column

```
a[z][y][x]
```

<table>
<thead>
<tr>
<th>Plane 0</th>
<th>Plane 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a[0][0][0] = 0;</td>
<td>a[0][0][0] = 0;</td>
</tr>
<tr>
<td>a[0][0][1] = 1;</td>
<td>a[0][0][1] = 1;</td>
</tr>
<tr>
<td>a[0][1][0] = 2;</td>
<td>a[0][1][0] = 2;</td>
</tr>
<tr>
<td>a[0][1][1] = 3;</td>
<td>a[0][1][1] = 3;</td>
</tr>
<tr>
<td>a[1][0][0] = 4;</td>
<td>a[1][0][0] = 4;</td>
</tr>
<tr>
<td>a[1][0][1] = 5;</td>
<td>a[1][0][1] = 5;</td>
</tr>
<tr>
<td>a[1][1][0] = 6;</td>
<td>a[1][1][0] = 6;</td>
</tr>
<tr>
<td>a[1][1][1] = 7;</td>
<td>a[1][1][1] = 7;</td>
</tr>
</tbody>
</table>
Arrays
Example of Array Processing

/************************************************************/
* Print out 0 to 90 in increments of 10
************************************************************/
int main(void)
{
    int i = 0;
    int a[10] = {0,1,2,3,4,5,6,7,8,9};
    while (i < 10)
    {
        a[i] *= 10;
        printf("%d\n", a[i]);
        i++;
    }
    while (1);
}

Strings
Character Arrays and Strings

- **Strings**: are arrays of char whose last element is a null character \"\0\" with an ASCII value of 0. C has no native string data type, so strings must always be treated as character arrays.
- Are enclosed in double quotes "string"
- Are terminated by a null character '\0'
- Must be manipulated as arrays of characters (treated element by element)

May be initialized with a string literal
Strings
Creating a String Character Array

Strings are created like any other array of `char`:

**Syntax**

```
char arrayName[length];
```

- `length` must be one larger than the length of the string to accommodate the terminating null character `\0`
- A `char` array with `n` elements holds strings with `n-1 char`

**Example**

```c
char str1[10]; //Holds 9 characters plus '\0'
char str2[6];  //Holds 5 characters plus '\0'
```

---

Strings
How to Initialize a String at Declaration

Character arrays may be initialized with string literals:

**Syntax**

```
char arrayName[] = "Microchip";
```

- Array size is not required
- Size automatically determined by length of string
- NULL character ' \0 ' is automatically appended

**Example**

```c
char str1[] = "Microchip"; //10 chars "Microchip\0"
char str2[6] = "Hello";    //6 chars "Hello\0"
//Alternative string declaration - size required
char str3[4] = {'P', 'I', 'C', '\0'};
```
Strings

How to Initialize a String in Code

In code, strings must be initialized element by element:

Syntax

```c
arrayName[0] = char1;
arrayName[1] = char2;
...;
arrayName[n] = '\0';
```

- Null character '\0' must be appended manually

Example

```c
str[0] = 'H';
str[1] = 'e';
str[2] = 'l';
str[3] = 'l';
str[4] = 'o';
str[5] = '\0';
```

Strings

Comparing Strings

- Strings cannot be compared using relational operators (==, !=, etc.)
- Must use standard C library string manipulation functions
  - `strcmp()` returns 0 if strings equal

Example

```c
char str[] = "Hello";

if (!strcmp(str, "Hello"))
    printf("The string is \"%s\".\n", str);
```
Functions
Array Parameters

- Arrays are passed by reference rather than by value for greater efficiency
- A pointer to the array, rather than the array itself, is passed to the function

```
This declaration...
void WriteLCD(char greetings[]){...}
```

...is equivalent to this declaration.
```
void WriteLCD(char *greetings){...}
```

Exercise 10
Arrays
Lab 10
Arrays

• Open the lab Project:

On the class website
/Examples/Lab10.zip -> Load “Lab10.X”

1 Open MPLAB®X and select Open Project Icon (Ctrl + Shift + O)
Open the Project listed above.

If you already have a project open in MPLAB X, close it by “right clicking” on the open project and selecting “Close”

Exercise 10
Arrays

Solution: Step 1

/*############################################################################
# STEP 1: Create two initialized arrays with 10 elements each named array1 and
#         array2 (you may use the pre-defined constant ARRAY_SIZE as part of
#         the array declaration).
#         The arrays should be initialized with the following values:
#         + array1: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
#         + array2: 9, 8, 7, 6, 5, 4, 3, 2, 1, 0
#         Note: the elements are all of type int
############################################################################*/

// array1 declaration & definition
int array1[ARRAY_SIZE] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

// array2 declaration & definition
int array2[ARRAY_SIZE] = {9, 8, 7, 6, 5, 4, 3, 2, 1, 0};
Exercise 10

Arrays

Solution: Step 2

```c
/*STEP 2: Pass the two arrays you declared above (array1 & array2) to the
function add_function() (see its definition below). Store the
result of the function call in the array result[]. The idea here is
to add each corresponding element of array1 and array2 and store the
result in result[]. In other words, add the first element of
array1[] to the first element of array2[] and store the result in
the first element of result[]. Next add the second elements...*/
result[i] = add_function(array1[i], array2[i]);
i++;
```

Exercise 10

Conclusions

- Arrays may be used to store a group of related variables of the same type under a common name
- Individual elements are accessed by using the array index in conjunction with the array name
- Arrays may be used in many places that an ordinary variable would be used
Questions?