Arrays

Hour 12

- Objectives
  - Defining arrays
  - Indexing arrays
  - Arrays and for loops
  - Pointers and arrays
  - Static array initialization and sizing (aka unsized arrays)
  - Storing arrays in memory

Copyright © 1998-2002 Delroy A. Brinkerhoff. All Rights Reserved.

Arrays

Simple structured data

- An array is an ordered collection of variables, each of the same type, referenced by one name and a subscript
  - subscripts range from 0..size-1
  - C does not check array bounds

- Array definitions include the data type and array size
  - bracketed values are dimensional sizes
  - each dimension is individually bracketed

- Name of the array, without a subscript, is a constant address (the address of the first element)

```c
int test[10]; /* 10 ints */
float test_score[10][4]; /* 40 floats */
double class_score[10][4][5]; /* 200 doubles */
```
One-dimensional Array

Vector

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>41</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>97</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>91</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>89</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>89</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>76</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>83</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>91</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>79</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Two-dimensional Array

Matrix

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>41</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>97</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>91</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>89</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>89</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>76</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>83</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>91</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>79</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- test[0] at index 0
- test[5] at index 5
- test_score[0][0] at index 0,0
- test_score[2][3] at index 2,3
- test_score[3][2] at index 3,2
- test_score[9][3] at index 9,3
Three-dimensional Array

Solid

class_score[0][2][3]
class_score[0][0][0]
class_score[2][4][1]
class_score[4][3][0]

Array Syntax

Using arrays

- Arrays are often used with loops
  - `for (i = 0; i < 10; i++)`
    - `test[i] = 100; /* initialize test */`
  - `for (i = 0; i < 10; i++)`
    - `printf("%d\n", test[i]);`
  - `for (i = 0; i < 10; i++)`
    - `score = test[i] * weight + 5;`
- An array element can be used wherever a variable is legal
  - `for (i = 0; i < 10; i++)`
    - `test_score[i][j] = test_score[i][j] + 10;`
### Static Array Initialization

**Compile-time initialization**

```c
int month_len[12] =
{ 31, 28, 31, 30,
  31, 30, 31, 31,
  30, 31, 30, 31
};

int month_len[] =
{ 31, 28, 31, 30,
  31, 30, 31, 31,
  30, 31, 30, 31
};
```

```c
int test_scores[5][4]=
{ { 95, 98, 97, 96 },
  { 79, 89, 79, 85 },
  { 99, 98, 99, 99 },
  { 90, 89, 83, 86 },
  { 75, 72, 79, 69 }
};
```

### Dynamic Array Sizing

**Compiler counts the number of elements**

- **General format**
  - `type name [] = { ... }; /* do static initialization */`
  - `int number = sizeof(name) / sizeof(type);`
    - `sizeof(name)` is the total number of bytes occupied by the array
    - `sizeof(type)` is the number of bytes in type
    - the quotient is the number of elements in the array

- **Example**

  ```c
  int month_len[] =
  { 31, 28, 31, 30,
    31, 30, 31, 31,
    30, 31, 30, 31
  };

  int number = sizeof(month_len)/sizeof(int);
  ```
Pointer Assignment & Operator Usage

Comparing pointers to arrays

Therefore:
- *pointer == 0
- pointer[0] == 0, pointer[1] == 1, ... pointer[99] == 99
- *(pointer +1) == 1

```c
int array[100];
int* pointer;

for (i = 0; i < 100; i++)
    array[i] = i;
pointer = array;
```

Arrays and Pointers

A strong relationship

```c
char array[100]; /* array variable */
char* pointer; /* pointer variable*/

pointer = array; /* common */

array[i] == *(array+i) == pointer[i] == *(pointer+i)
&array[i] == array+i == pointer+i

pointer = &array[0]; /* uncommon */
pointer = &array[i]; /* okay: 0 ≤ i < 10*/

pointer++; /* okay; next char */
array++; /* illegal */
aarray = pointer; /* illegal */
```
Arrays and Memory Mapping

Storing arrays in memory

- Multidimensional arrays are stored as single dimensional arrays in memory
- Two ways to store:
  - row-major order: stored by rows
  - column-major order: stored by columns
- C/C++ use row major
  - RMaddr = i * ncols + j

H is at i = 2, j = 1
ncols = 3
2 * 3 + 1 = 7

H is at RM address 7

\[ \text{RMaddr} = i \times \text{ncols} + j \]