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

```
0 41
1 97
2 91
3 89
4 89
5 100
6 76
7 83
8 91
9 79
```

Two-dimensional Array

Matrix

```
0 81
1 97
2 91
3 89
4 89
5 100
6 76
7 83
8 91
9 79
```

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++)
    - for (j = 0; j < 4; j++)
      - printf("%f\n", test_score[i][j]);

- An array element can be used wherever a variable is legal
  - for (i = 0; i < 10; i++)
    - score = test[i] * weight + 5;
  - for (i = 0; i < 10; i++)
    - test[i] = test[i] + 10;
**Static Array Initialization**

Compile-time initialization

```c

```

**Dynamic Array Sizing**

Compiler counts the number of elements

```
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

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

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

**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 */
array = 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
```

```
\[ \begin{array}{c|c|c}
0 & A & B \\
1 & D & E \\
2 & G & H \\
3 & J & K \\
\end{array} \]
\[ \begin{array}{c|c|c|c}
0 & A & B & C \\
1 & D & E & F \\
2 & G & H & I \\
3 & J & K & L \\
\end{array} \]
```

H is at RM address 7
nrows = ncols = 3
A \times 3 + 1 = 7
\[ \begin{array}{ccc|ccc|ccc}
0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\
0 & B & C & A & D & E & F & G & H \\
1 & D & E & F & G & H & I & J & K \\
2 & G & H & I & J & K & L & A & B \\
3 & J & K & L & A & B & C & D & E \\
\end{array} \]