

Outline

- 6.1 Introduction**
- 6.2 Arrays**
- 6.3 Declaring Arrays**
- 6.4 Examples Using Arrays**
- 6.5 Passing Arrays to Functions**
- 6.6 Sorting Arrays**
- 6.7 Case Study: Computing Mean, Median and Mode Using Arrays**
- 6.8 Searching Arrays**
- 6.9 Multiple-Subscripted Arrays**

6.1 Introduction

- Arrays
 - Structures of related data items
 - Static entity – same size throughout program
 - Dynamic data structures discussed in Chapter 12

6.2 Arrays

- Array
 - Group of consecutive memory locations
 - Same name and type
- To refer to an element, specify
 - Array name
 - Position number
- Format:
 - arrayname [position number]*
 - First element at position 0
 - **n** element array named **c**:
 - `c[0], c[1]...c[n - 1]`

Name of array
(Note that all elements of this array have the same name, **c**)

↓

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78

↑
Position number
of the element
within array **c**

6.2 Arrays

- Array elements are like normal variables

```
c[ 0 ] = 3;
```

```
printf( "%d", c[ 0 ] );
```

- Perform operations in subscript. If **x** equals 3

```
c[ 5 - 2 ] == c[ 3 ] == c[ x ]
```

6.3 Declaring Arrays

- When declaring arrays, specify

- Name
- Type of array
- Number of elements

```
arrayType arrayName[ numberOfElements ];
```

- Examples:

```
int c[ 10 ];
```

```
float myArray[ 3284 ];
```

- Declaring multiple arrays of same type

- Format similar to regular variables
- Example:

```
int b[ 100 ], x[ 27 ];
```

6.4 Examples Using Arrays

- Initializers

```
int n[ 5 ] = { 1, 2, 3, 4, 5 };
```

- If not enough initializers, rightmost elements become 0

```
int n[ 5 ] = { 0 }
```

- All elements 0
 - If too many a syntax error is produced syntax error
 - C arrays have no bounds checking
- If size omitted, initializers determine it

```
int n[ ] = { 1, 2, 3, 4, 5 };
```

- 5 initializers, therefore 5 element array

6.4 Examples Using Arrays

- Character arrays
 - String `"first"` is really a static array of characters
 - Character arrays can be initialized using string literals

```
char string1[] = "first";
```

- Null character `'\0'` terminates strings
- `string1` actually has 6 elements
 - It is equivalent to

```
char string1[] = { 'f', 'i', 'r', 's', 't', '\0' };
```

- Can access individual characters
- Array name is address of array, so `&` not needed for `scanf`

```
string1[3] is character 's'
```

```
scanf( "%s", string2 );
```

- Reads characters until whitespace encountered
- Can write beyond end of array, be careful

6.5 Passing Arrays to Functions

- Passing arrays
 - To pass an array argument to a function, specify the name of the array without any brackets

```
int myArray[ 24 ] ;  
myFunction ( myArray , 24 ) ;
```
 - Array size usually passed to function
 - Arrays passed call-by-reference
 - Name of array is address of first element
 - Function knows where the array is stored
 - Modifies original memory locations
- Passing array elements
 - Passed by call-by-value
 - Pass subscripted name (i.e., **myArray [3]**) to function

6.5 Passing Arrays to Functions

- Function prototype

```
void modifyArray( int b[], int arraySize );
```

- Parameter names optional in prototype
 - `int b[]` could be written `int []`
 - `int arraySize` could be simply `int`

6.6 Sorting Arrays

- Sorting data
 - Important computing application
 - Virtually every organization must sort some data
- Bubble sort (sinking sort)
 - Several passes through the array
 - Successive pairs of elements are compared
 - If increasing order (or identical), no change
 - If decreasing order, elements exchanged
 - Repeat
- Example:
 - original: 3 4 2 6 7
 - pass 1: 3 2 4 6 7
 - pass 2: 2 3 4 6 7
 - Small elements "bubble" to the top

6.7 Case Study: Computing Mean, Median and Mode Using Arrays

- Mean – average
- Median – number in middle of sorted list
 - 1, 2, 3, 4, 5
 - 3 is the median
- Mode – number that occurs most often
 - 1, 1, 1, 2, 3, 3, 4, 5
 - 1 is the mode

6.8 Searching Arrays: Linear Search and Binary Search

- Search an array for a *key value*
- Linear search
 - Simple
 - Compare each element of array with key value
 - Useful for small and unsorted arrays

6.8 Searching Arrays: Linear Search and Binary Search

- Binary search
 - For sorted arrays
 - Compares **middle** element with **key**
 - If equal, match found
 - If **key** < **middle**, looks in first half of array
 - If **key** > **middle**, looks in last half
 - Repeat
 - Very fast; at most n steps, where $2^5 > \text{number of elements}$
 - 30 element array takes at most 5 steps
 - $2^5 > 30$ so at most 5 steps

6.9 Multiple-Subscripted Arrays

- Multiple subscripted arrays
 - Tables with rows and columns (**m** by **n** array)
 - Like matrices: specify row, then column

	Column 0	Column 1	Column 2	Column 3
Row 0	<code>a[0][0]</code>	<code>a[0][1]</code>	<code>a[0][2]</code>	<code>a[0][3]</code>
Row 1	<code>a[1][0]</code>	<code>a[1][1]</code>	<code>a[1][2]</code>	<code>a[1][3]</code>
Row 2	<code>a[2][0]</code>	<code>a[2][1]</code>	<code>a[2][2]</code>	<code>a[2][3]</code>

Diagram illustrating the structure of a multiple-subscripted array. The array is represented as a table with rows and columns. The labels "Row 0", "Row 1", and "Row 2" are on the left. The labels "Column 0", "Column 1", "Column 2", and "Column 3" are at the top. The elements are shown as `a[row][column]`. Arrows point from the labels "Array name", "Row subscript", and "Column subscript" to the corresponding parts of the element `a[2][1]` in the table.

6.9 Multiple-Subscripted Arrays

- Initialization

- `int b[2][2] = { { 1, 2 }, { 3, 4 } };`

- Initializers grouped by row in braces

- If not enough, unspecified elements set to zero

```
int b[2][2] = { { 1 }, { 3, 4 } };
```

- Referencing elements

- Specify row, then column

```
printf( "%d", b[ 0 ][ 1 ] );
```

1	2
3	4

1	0
3	4