Chapter 6
Arrays

Associate Prof. Yuh-Shyan Chen
Dept. of Computer Science and Information Engineering
National Chung-Cheng University

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
  - An element array named c: `c[0], c[1]...c[n-1]`
6.2 Arrays (II)

- Array elements are like normal variables
  \[ c[0] = 3; \]
  \[ \text{printf( "%d", c[0] );} \]

- Perform operations in subscript. If \( x = 3 \),

6.3 Declaring Arrays

- When declaring arrays, specify
  - Name
  - Type of array
  - Number of elements
    \[ \text{arrayType arrayName[ numberOfElements ];} \]
    \[ \text{int c[10];} \]
    \[ \text{float myArray[3284];} \]

- Declaring multiple arrays of same type
  - Format similar to regular variables
    \[ \text{int b[100], x[27];} \]

6.4 Examples Using Arrays

- Initializers
  \[ \text{int n[5] = \{1, 2, 3, 4, 5\};} \]
  - If not enough initializers, rightmost elements become 0
  - If too many, syntax error
    \[ \text{int n[5] = \{0\};} \]
  - C arrays have no bounds checking

- If size omitted, initializers determine it
  \[ \text{int n[]} = \{1, 2, 3, 4, 5\}; \]
  - 5 initializers, therefore 5 element array

---

```c
/* Fig. 6.8: fig06_08.c */
/* Histogram printing program */
/* Function Header */
/* Function main */

int main()
{
    int n[SIZE] = {19, 3, 15, 7, 11, 9, 13, 5, 17, 1};

    printf( "%s%13s%17s
", "Element", "Value", "Histogram");
    printf( "%7d%13d
", i, n[i] );

    for ( j = 1; j <= n[i]; j++ )   /* print one bar */
        printf( "%c", '*' );

    printf( "\n" );
    return 0;
}
```
### 6.4 Examples Using Arrays (II)

- **Character arrays**
  - String "hello" is really a static array of characters
  - Character arrays can be initialized using string literals
    ```c
    char string1[] = "first";  
    char string1[] = ( 'f', 'i', 'r', 's', 't', '\0' );
    ```
  - null character \'\0\' terminates strings
  - string1 actually has 6 elements

---

### 6.4 Examples Using Arrays (III)

- **Character arrays (continued)**
  - Access individual characters
    - `string1[3]` is character 's'
  - Array name is address of array, so & not needed for `scanf`
    ```c
    scanf( "%s", string2 );
    ```
    - Reads characters until whitespace encountered
    - Can write beyond end of array, be careful

---

Here is the code snippet:

```c
#include <stdio.h>

int main()
{
  char string1[20], string2[20] = "string literal";
  int i;

  printf(" Enter a string: ");
  scanf( "%s", string1 );

  printf("string1 is: %s
string2: is %s
string1 with spaces between characters is:", string1, string2);

  for ( i = 0; string1[ i ] != '\0'; i++ )
    printf( "%c ", string1[ i ] );

  printf( "\n" );
  return 0;
}
```

**Enter a string: Hello there**

```
string1 is: Hello
string2: is string literal
string1 with spaces between characters is: H e l l o
```
6.5 Passing Arrays to Functions

- **Passing arrays**
  - Specify array name without brackets
    ```c
    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**

---

6.5 Passing Arrays to Functions (II)

- **Function prototype**
  ```c
  void modifyArray(int b[], int arraySize);
  ```
- **Parameter names optional in prototype**
  ```c
  int b[] could be simply int []
  int arraySize could be simply int
  ```

---

6.5 Passing Arrays to Functions

- **Passing array elements**
  - Passed by call-by-value
  - Pass subscripted name (i.e., `myArray[3]`) to function
### 3.1 Function definitions

```c
void modifyArray( int b[], int size )
{
    int j;
    for ( j = 0; j <= size - 1; j++ )
        b[ j ] *= 2;
}

void modifyElement( int e )
{
    printf( "Value in modifyElement is %d\n", e *= 2 );
}
```

**Effects of passing entire array call by reference:**

- The values of the original array are:
  - 0 1 2 3 4
- The values of the modified array are:
  - 0 2 4 6 8

**Effects of passing array element call by value:**

- The value of `a[3]` is 6
- Value in `modifyElement` is 12
- The value of `a[3]` is 6

### 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, 4, 5
  - 1 is the mode

```c
#include <stdio.h>
define SIZE 99

void mean( const int answer[] );
void median( int answer[] );
void mode( int answer[], const int frequency[] );
void bubbleSort( int answer[] );
void printArray( const int answer[] );

int main()
{
    int frequency[ 10 ] = { 0 };
    int response[ SIZE ] =
    { 6, 7, 8, 9, 8, 7, 8, 9, 8, 9,
      7, 8, 9, 5, 9, 8, 7, 8, 7, 8,
      6, 7, 8, 9, 3, 9, 8, 7, 8, 7,
      7, 8, 9, 8, 9, 8, 9, 7, 8, 9,
      6, 7, 8, 7, 8, 7, 9, 8, 9, 2,
      7, 8, 9, 8, 9, 8, 9, 7, 5, 3,
      5, 6, 7, 2, 5, 3, 9, 4, 6, 4,
      7, 8, 9, 6, 8, 7, 8, 9, 7, 8,
      7, 4, 4, 2, 5, 3, 8, 7, 5, 6,
      4, 5, 6, 1, 6, 5, 7, 8, 7 };

    mean( response );
    median( response );
    mode( frequency, response );
    return 0;
}

void mean( const int answer[] )
{
    int j, total = 0;
    printf( "%s\n%s\n%s\n%s\n", "********", "  Mean", "********", "\n" );
    for ( j = 0; j <= SIZE - 1; j++ )
        total += answer[ j ];
    printf( "The mean is the average value of the data\nitems. The mean is equal to the total of\nall the data items divided by the number\nof data items ( %d ). The mean value for\nthis run is: %d / %d = %.4f\n\n", SIZE, total, SIZE, ( double ) total / SIZE );
}

void median( int answer[] )
{
    printf( "\n%s\n%s\n%s\n%s\n", "********", " Median", "********", "\n" );
    printArray( answer );
    bubbleSort( answer );
    printArray( answer );
    printf( "The median is element %d of\nthe sorted %d element array.\nFor this run the median is %d\n\n", SIZE / 2, SIZE, answer[ SIZE / 2 ];
}
```
void mode( int freq[], const int answer[] )
{
    int rating, j, h, largest = 0, modeValue = 0;

    printf( "********
    Mode
********" );

    for ( rating = 1; rating <= 9; rating++ )
        freq[ rating ] = 0;

    for ( j = 0; j <= SIZE - 1; j++ )
        ++freq[ answer[ j ] ];

    printf( "%s%11s%19s

%54s
%54s

", "Response", "Frequency", "Histogram",
1 "1    1    2    2", "5    0    5    0    5" );

    for ( rating = 1; rating <= 9; rating++ )
    {
        printf( "%8d%11d          ", rating, freq[ rating ] );
        if ( freq[ rating ] > largest ) {
            largest = freq[ rating ];
            modeValue = rating;
        }
        for ( h = 1; h <= freq[ rating ]; h++ )
            printf( "*" );
    }

    printf( "The mode is the most frequent value.
" For this run the mode is %d which occurred %d times.
", modeValue, largest );
}

void bubbleSort( int a[] )
{
    int pass, j, hold;

    for ( pass = 1; pass <= SIZE - 1; pass++ )
    {
        for ( j = 0; j <= SIZE - 2; j++ )
        {
            if ( a[ j ] > a[ j + 1 ] ) {
                hold = a[ j ];
                a[ j ] = a[ j + 1 ];
                a[ j + 1 ] = hold;
            }
        }
    }
}

void printArray( const int a[] )
{
    int j;

    for ( j = 0; j <= SIZE - 1; j++ )
    {
        if ( j % 20 == 0 )
            printf( "\n" );
        printf( "%2d", a[ j ] );
    }
}

********
Mean
********
The mean is the average value of the data items. The mean is equal to the total of all the data items divided by the number of data items (99). The mean value for this run is: 681 / 99 = 6.8788

********
Median
********
The unsorted array of responses is:

7 8 9 8 7 8 9 8 9 7 8 9 5 9 8 7 8 7 8
6 7 8 9 3 9 8 7 8 7 7 8 9 8 9 8 9 7 8 9
6 7 8 7 8 7 9 8 9 2 7 8 9 8 9 8 9 7 8 9
5 6 7 2 5 3 9 4 6 4 7 8 9 6 8 7 8 9 7 8
7 4 4 2 5 3 8 7 5 6 4 5 6 1 6 5 7 8 7

The sorted array is:

1 2 2 2 3 3 3 3 4 4 4 4 4 5 5 5 5 5 5 5
5 6 6 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7 7
7 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9

The median is element 49 of the sorted 99 element array.
For this run the median is 7

********
Mode
********
Response  Frequency  Histogram
1    1    2    2
5    0    5    0    5
1          1          *
2          3          ***
3          4          ****
4          5          *****
5          8          ********
6          9          *********
7         23          ***********************
8         27          ***************************
9         19          *******************

The mode is the most frequent value.
For this run the mode is 8 which occurred 27 times.
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

  - 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 \( \log_2 \) number of elements
      - 30 element array takes at most 5 steps
        \( 2^{5} > 30 \)

6.9 Multiple-Subscripted Arrays

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

```
Row 0 | Column 1 | Column 2 | Column 3
-----|---------|---------|---------
Row 1 | a[0][1] | a[1][1] | a[2][1]  
 Row 2 | a[0][2] | a[1][2] | a[2][2]  |
```

- Initialization
  - \( \text{int } b[2][2] = \{ \{ 1, 2 \}, \{ 3, 4 \} \}; \)
  - Initializers grouped by row in braces
  - If not enough, unspecified elements zero
    \( \text{int } b[2][2] = \{ \{ 1 \}, \{ 3, 4 \} \}; \)

- Referencing elements
  - Specify row, then column
    \( \text{printf("\%d", b[0][1]);} \)
Array passing rules

- The first subscript of a multiple-subscripted array is not required either
- But all subsequent subscripts are required

```c
/* Fig. 6.22: fig06_22.c */
/* Double subscripted array example */

#include <stdio.h>

#define STUDENTS 3
#define EXAMS 4

int minimum( const int grades[][ EXAMS ], int, int );
int maximum( const int grades[][ EXAMS ], int, int );
double average( const int grades[], int );
void printArray( const int grades[][ EXAMS ], int, int );

int main()
{...
```

Each row is a particular student, each column is the grades on the exam.

```c
12   int student;
13   for ( student = 0; student <= STUDENTS - 1; student++ )
14       printf( "The average grade for student %d is %.2f
", student, average( grades[ student ], EXAMS ) );
15   return 0;
16 }
17
18 /* Find the minimum grade */
int minimum( const int grades[][ EXAMS ], int pupils, int tests )
{...
```

```c
65   int i, total = 0;
66   for ( i = 0; i <= tests - 1; i++ )
67       total += grades[ i ];
68   return ( double ) total / tests;
69 }
70
71 /* Print the array */
void printArray( const int grades[][ EXAMS ], int pupils, int tests )
{...
```

```c
43    int student;
44    for ( student = 0; student <= STUDENTS - 1; student++ )
45        printf( "The average grade for student %d is %.2f
", student, average( grades[ student ], EXAMS ) );
46    return 0;
47 }
48
49 /* Find the maximum grade */
int maximum( const int grades[][ EXAMS ], int pupils, int tests )
{...
```

```c
91   return highGrade;
92 }
93
94 /* Determine the average grade for a particular exam */
double average( const int grades[], int tests )
{...
```
The array is:

<table>
<thead>
<tr>
<th></th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
</tr>
</thead>
<tbody>
<tr>
<td>studentGrades[0]</td>
<td>77</td>
<td>68</td>
<td>86</td>
<td>73</td>
</tr>
<tr>
<td>studentGrades[1]</td>
<td>96</td>
<td>87</td>
<td>89</td>
<td>78</td>
</tr>
<tr>
<td>studentGrades[2]</td>
<td>70</td>
<td>90</td>
<td>86</td>
<td>81</td>
</tr>
</tbody>
</table>

Content grade: 00
Student grade: 90

The average grade for student 0 is 76.00
The average grade for student 1 is 87.50
The average grade for student 2 is 81.75