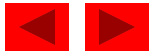
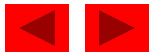
Chapter 6 - Arrays

Outline



6.1 Introduction

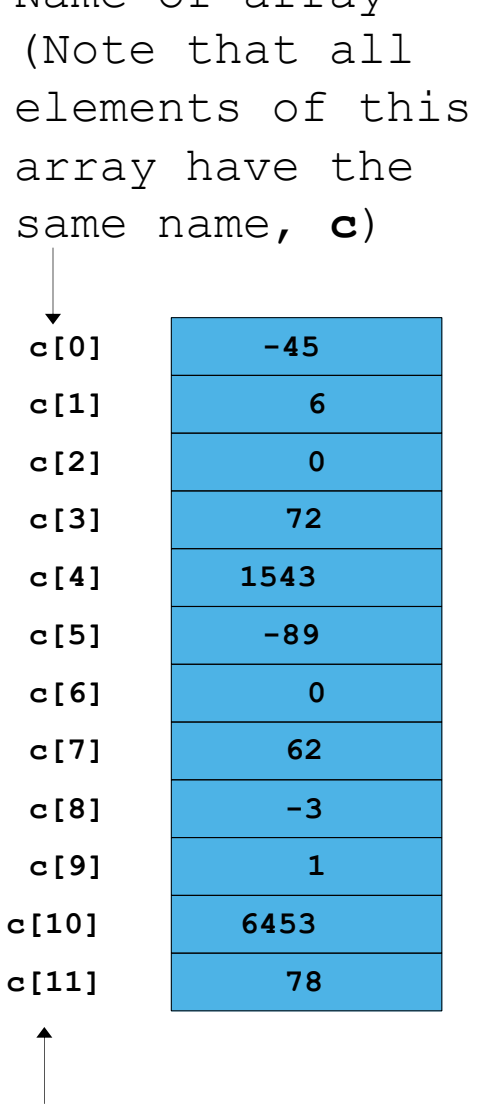
- Arrays
  - Structures of related data items
  - Static entity – same size throughout program
  - Dynamic data structures discussed in Chapter 12



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]**



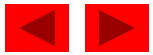
6.2 Arrays

- 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

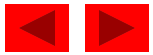
```
arrayType arrayName[ numberOfElements ];
```

  - 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 };
```

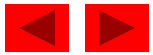
  - If not enough initializers, rightmost elements become **0**

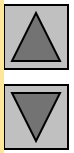
```
int n[ 5 ] = { 0 }
```

    - 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





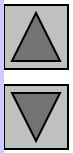
Outline

1. Initialize array

2. Loop

3. Print

```
1  /* Fig. 6.8: fig06_08.c
2     Histogram printing program */
3  #include <stdio.h>
4  #define SIZE 10
5
6  int main()
7  {
8     int n[ SIZE ] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
9     int i, j;
10
11    printf( "%s%13s%17s\n", "Element", "Value", "Histogram" );
12
13    for ( i = 0; i <= SIZE - 1; i++ ) {
14       printf( "%7d%13d        ", i, n[ i ]) ;
15
16       for ( j = 1; j <= n[ i ]; j++ )   /* print one bar */
17          printf( "%c", '*' );
18
19       printf( "\n" );
20    }
21
22    return 0;
23 }
```



Outline

Program Output

```
Element         Value         Histogram
      0            19         ********************
      1             3         ***
      2            15         ***************
      3             7         *******
      4            11         ***********
      5             9         *********
      6            13         *************
      7             5         *****
      8            17         *****************
      9             1         *
```

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' };
```

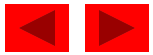
  - Can access individual characters

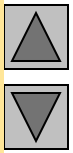
```
string1[ 3 ] is character 's'
```

  - Array name is address of array, so & not needed for scanf

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

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





Outline

```
1  /* Fig. 6.10: fig06_10.c
2     Treating character arrays as strings */
3  #include <stdio.h>
4
5  int main()
6  {
7     char string1[ 20 ], string2[] = "string literal";
8     int i;
9
10    printf(" Enter a string: ");
11    scanf( "%s", string1 );
12    printf( "string1 is: %s\nstring2: is %s\n"
13            "string1 with spaces between characters is:\n",
14            string1, string2 );
15
16    for ( i = 0; string1[ i ] != '\0'; i++ )
17       printf( "%c ", string1[ i ] );
18
19    printf( "\n" );
20    return 0;
21 }
```

1. Initialize strings

2. Print strings

2.1 Define loop

2.2 Print characters individually

2.3 Input string

3. Print string

Program Output

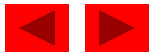
```
Enter a string: Hello there
string1 is: Hello
string2 is: string literal
string1 with spaces between characters is:
H e l l o
```

6.5 Passing Arrays to Functions

- Passing arrays
  - 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

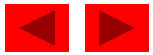


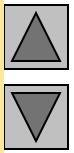
6.5 Passing Arrays to Functions

- 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`**





Outline

1. Function definitions

2. Pass array to a function

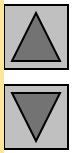
2.1 Pass array element to a function

3. Print

```
1  /* Fig. 6.13: fig06 13.c
2     Passing arrays and individual array elements to functions */
3  #include <stdio.h>
4  #define SIZE 5
5
6  void modifyArray( int [], int );  /* appears strange */
7  void modifyElement( int );
8
9  int main()
10 {
11    int a[ SIZE ] = { 0, 1, 2, 3, 4 }, i;
12
13    printf( "Effects of passing entire array call "
14            "by reference:\n\nThe values of the "
15            "original array are:\n" );
16
17    for ( i = 0; i <= SIZE - 1; i++ )
18       printf( "%3d", a[ i ] );
19
20    printf( "\n" );
21    modifyArray( a, SIZE );  /* passed call by reference */
22    printf( "The values of the modified array are:\n" );
23
24    for ( i = 0; i <= SIZE - 1; i++ )
25       printf( "%3d", a[ i ] );
26
27    printf( "\n\n\nEffects of passing array element call "
28            "by value:\n\nThe value of a[3] is %d\n", a[ 3 ] );
29    modifyElement( a[ 3 ] );
30    printf( "The value of a[ 3 ] is %d\n", a[ 3 ] );
31    return 0;
32 }
```

Entire arrays passed call-by-reference, and can be modified

Array elements passed call-by-value, and cannot be modified



Outline

3.1 Function definitions

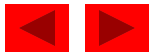
```
33
34 void modifyArray( int b[], int size )
35 {
36    int j;
37
38    for ( j = 0; j <= size - 1; j++ )
39       b[ j ] *= 2;
40 }
41
42 void modifyElement( int e )
43 {
44    printf( "Value in modifyElement is %d\n", e *= 2 );
45 }
```

Program Output

```
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
```

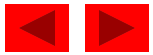
6.6 Sorting Arrays

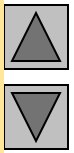
- Sorting data
  - Important computing application
  - Virtually every organization must sort some data
- Bubble sort (sinking sort)
  - Several passes through the array
  - Successive pairs of elements are compared
    - If increasing order (or identical), no change
    - If decreasing order, elements 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





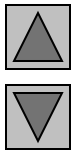
Outline

1. Function prototypes

1.1 Initialize array

2. Call functions `mean`, `median`, and `mode`

```
1  /* Fig. 6.16: fig06 16.c
2     This program introduces the topic of survey data analysis.
3     It computes the mean, median, and  mode of the data */
4  #include <stdio.h>
5  #define SIZE 99
6
7  void mean( const int [] );
8  void median( int [] );
9  void mode( int [], const int [] ) ;
10 void bubbleSort( int [] );
11 void printArray( const int [] );
12
13 int main()
14 {
15    int frequency[ 10 ] = { 0 };
16    int response[ SIZE ] =
17       { 6, 7, 8, 9, 8, 7, 8, 9, 8, 9,
18         7, 8, 9, 5, 9, 8, 7, 8, 7, 8,
19         6, 7, 8, 9, 3, 9, 8, 7, 8, 7,
20         7, 8, 9, 8, 9, 8, 9, 7, 8, 9,
21         6, 7, 8, 7, 8, 7, 9, 8, 9, 2,
22         7, 8, 9, 8, 9, 8, 9, 7, 5, 3,
23         5, 6, 7, 2, 5, 3, 9, 4, 6, 4,
24         7, 8, 9, 6, 8, 7, 8, 9, 7, 8,
25         7, 4, 4, 2, 5, 3, 8, 7, 5, 6,
26         4, 5, 6, 1, 6, 5, 7, 8, 7 };
27
28    mean( response );
29    median( response );
30    mode( frequency, response );
31    return 0;
32 }
```



Outline

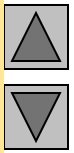
```
33
34 void mean( const int answer[] )
35 {
36    int j, total = 0;
37
38    printf( "%s\n%s\n%s\n", "********", "  Mean", "********" );
39
40    for ( j = 0; j <= SIZE - 1; j++ )
41       total += answer[  j  ];
42
43    printf( "The mean is the average value of the data\n"
44            "items. The mean is equal to the total of\n"
45            "all the data items divided by the number\n"
46            "of data items ( %d ). The mean value for\n"
47            "this run is: %d / %d = %.4f\n\n",
48            SIZE, total, SIZE, (  double  ) total / SIZE );
49 }
50
51 void median( int answer[] )
52 {
53    printf( "\n%s\n%s\n%s\n%s",
54            "********", " Median", "********",
55            "The unsorted array of responses is" );
56
57    printArray( answer );
58    bubbleSort( answer );
59    printf( "\n\nThe sorted array is" );
60    printArray( answer );
61    printf( "\n\nThe median is element %d of\n"
62            "the sorted %d element array.\n"
63            "For this run the median is %d\n\n",
64            SIZE / 2, SIZE, answer[ SIZE / 2 ] );
```

3. Define function `mean`

3.1 Define function `median`

3.1.1 Sort Array

3.1.2 Print middle element



Outline

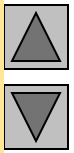
3.2 Define function `mode`

3.2.1 Increase `frequency[]` depending on `response[]`

```
65 }
66
67 void mode( int freq[], const int answer[] )
68 {
69    int rating, j, h, largest = 0, modeValue = 0;
70
71    printf( "\n%s\n%s\n%s\n",
72            "********", "  Mode", "********" );
73
74    for ( rating = 1; rating <= 9; rating++ )
75       freq[ rating ] = 0;
76
77    for ( j = 0; j <= SIZE - 1; j++ )
78       ++freq[ answer[ j ] ];
79
80    printf( "%s%11s%19s\n\n%54s\n%54s\n\n",
81            "Response", "Frequency", "Histogram",
82            "1    1    2    2", "5    0    5    0    5" );
83
84    for ( rating = 1; rating <= 9; rating++ ) {
85       printf( "%8d%11d          ", rating, freq[ rating ] );
86
87       if ( freq[ rating ] > largest ) {
88          largest = freq[ rating ];
89          modeValue = rating;
90       }
91
92       for ( h = 1; h <= freq[ rating ]; h++ )
93          printf( "*" );
94
```

Notice how the subscript in **`frequency[]`** is the value of an element in **`response[]`** (**`answer[]`**)

Print stars depending on value of **`frequency[]`**



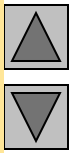
Outline

3.3 Define `bubbleSort`

3.3 Define `printArray`

```
95       printf( "\n" );
96    }
97
98    printf( "The mode is the most frequent value.\n"
99            "For this run the mode is %d which occurred"
100           " %d times.\n", modeValue, largest );
101}
102
103 void bubbleSort( int a[] )
104 {
105   int pass, j, hold;
106
107   for ( pass = 1; pass <= SIZE - 1; pass++ )
108
109      for ( j = 0; j <= SIZE - 2; j++ )
110
111         if ( a[ j ] > a[ j + 1 ] ) {
112            hold = a[ j ];
113            a[ j ] = a[ j + 1 ];
114            a[ j + 1 ] = hold;
115         }
116 }
117
118 void printArray( const int a[] )
119 {
120   int j;
121
122   for ( j = 0; j <= SIZE - 1; j++ ) {
123
124      if ( j % 20 == 0 )
125         printf( "\n" );
```

Bubble sort: if elements out of order, swap them.

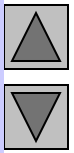


Outline

```
126
127       printf( "%2d", a[ j ] );
128    }
129 }
```

Program Output

```
********
 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 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

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
```



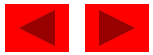
Outline

Program Output

```
********
  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.
```

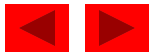
6.8 Searching Arrays: Linear Search and Binary Search

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



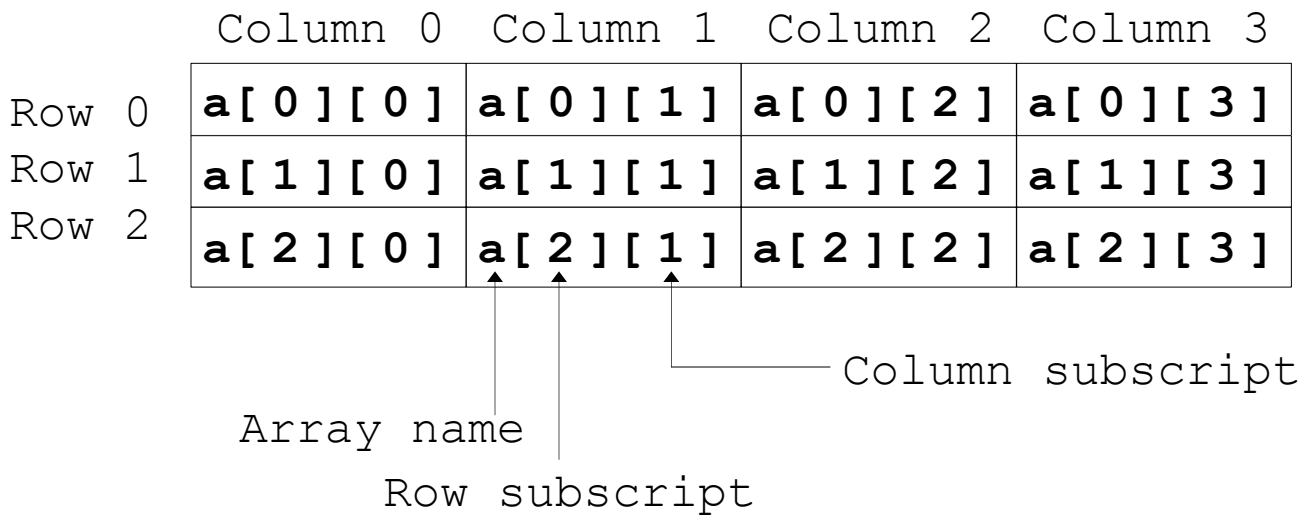
6.8 Searching Arrays: Linear Search and Binary Search

- 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 $2^n >$ 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



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

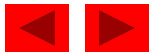
    `int b[ 2 ][ 2 ] = { { 1 }, { 3, 4 } };`

1	2
3	4

1	0
3	4

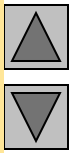
- Referencing elements
  - Specify row, then column

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



```c
/* Fig. 6.22: fig06 22.c
   Double-subscripted array example */
#include <stdio.h>
#define STUDENTS 3
#define EXAMS 4

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

int main()
{
   int student;
   const int studentGrades[ STUDENTS ][ EXAMS ] =
      { { 77, 68, 86, 73 },
        { 96, 87, 89, 78 },
        { 70, 90, 86, 81 } };

   printf( "The array is:\n" );
   printArray( studentGrades, STUDENTS, EXAMS );
   printf( "\n\nLowest grade: %d\nHighest grade: %d\n",
           minimum( studentGrades, STUDENTS, EXAMS ),
           maximum( studentGrades, STUDENTS, EXAMS ) );

   for ( student = 0; student <= STUDENTS - 1; student++ )
      printf( "The average grade for student %d is %.2f\n",
              student,
              average( studentGrades[ student ], EXAMS ) );

   return 0;
}
```

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



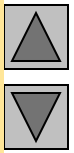
Outline

1. Initialize variables

1.1 Define functions to take double scripted

`studentgrades[][]`

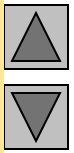
2. Call functions `minimum`, `maximum`, and `average`



Outline

3. Define functions

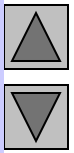
```
33
34 /* Find the minimum grade */
35 int minimum( const int grades[][ EXAMS ],
36              int pupils, int tests )
37 {
38    int i, j, lowGrade = 100;
39
40    for ( i = 0; i <= pupils - 1; i++ )
41       for ( j = 0; j <= tests - 1; j++ )
42          if ( grades[ i ][ j ] < lowGrade )
43             lowGrade = grades[ i ][ j ];
44
45    return lowGrade;
46 }
47
48 /* Find the maximum grade */
49 int maximum( const int grades[][ EXAMS ],
50              int pupils, int tests )
51 {
52    int i, j, highGrade = 0;
53
54    for ( i = 0; i <= pupils - 1; i++ )
55       for ( j = 0; j <= tests - 1; j++ )
56          if ( grades[ i ][ j ] > highGrade )
57             highGrade = grades[ i ][ j ];
58
59    return highGrade;
60 }
61
62 /* Determine the average grade for a particular exam */
63 double average( const int setOfGrades[], int tests )
64 {
```



Outline

3. Define functions

```
65    int i, total = 0;
66
67    for ( i = 0; i <= tests - 1; i++ )
68       total += setOfGrades[ i ];
69
70    return ( double ) total / tests;
71 }
72
73 /* Print the array */
74 void printArray( const int grades[][ EXAMS ],
75                  int pupils, int tests )
76 {
77    int i, j;
78
79    printf( "                [0]  [1]  [2]  [3]" );
80
81    for ( i = 0; i <= pupils - 1; i++ ) {
82       printf( "\nstudentGrades[%d] ", i );
83
84       for ( j = 0; j <= tests - 1; j++ )
85          printf( "%-5d", grades[ i ][ j ] );
86    }
87 }
```



Outline

Program Output

```
The array is:
                  [0]   [1]   [2]   [3]
studentGrades[0] 77    68    86    73
studentGrades[1] 96    87    89    78
studentGrades[2] 70    90    86    81

Lowest grade: 68
Highest grade: 96
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
```