

# Chapter 6

## Arrays

© Copyright 2007 by Deitel & Associates, Inc. and  
Pearson Education Inc. All Rights Reserved.

# Chapter 6 - Arrays

## 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

## Objectives

- In this chapter, you will learn:
  - To introduce the array data structure.
  - To understand the use of arrays to store, sort and search lists and tables of values.
  - To understand how to define an array, initialize an array and refer to individual elements of an array.
  - To be able to pass arrays to functions.
  - To understand basic sorting techniques.
  - To be able to define and manipulate multiple subscript 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:
    - $c[0], c[1] \dots c[n-1]$

Name of array (Note<sup>5</sup> 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

## Example: Array Declaration and Initialization

### METHOD - 1:

```
int c[12] = {-45,6,0,72,1543,-89,0,62,-3,1,6453,78};
```

### METHOD - 2:

```
int c[12];  
C[0] = -45;  
C[1] = 6;  
C[2] = 0;  
C[3] = 72;  
.....  
C[11] = 78;
```

### METHOD - 3:

```
int c[12], i;  
for (i=0; i <= 11; i++)  
{  
    printf("Enter element - %d", i);  
    scanf("%d", &c[i]);  
}
```

## 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 ]
```

## 6.2 Arrays

| Operators |    |    |                 |    |    | Associativity | Type           |
|-----------|----|----|-----------------|----|----|---------------|----------------|
| []        | () |    |                 |    |    | left to right | highest        |
| ++        | -- | !  | ( <i>type</i> ) |    |    | right to left | unary          |
| *         | /  | %  |                 |    |    | left to right | multiplicative |
| +         | -  |    |                 |    |    | left to right | additive       |
| <         | <= | >  | >=              |    |    | left to right | relational     |
| ==        | != |    |                 |    |    | left to right | equality       |
| &&        |    |    |                 |    |    | left to right | logical and    |
|           |    |    |                 |    |    | left to right | logical or     |
| ?:        |    |    |                 |    |    | right to left | conditional    |
| =         | += | -- | *=              | /= | %= | right to left | assignment     |
| ,         |    |    |                 |    |    | left to right | comma          |

**Fig. 6.2** Operator precedence.



## 6.3 Defining Arrays

- When defining arrays, specify
  - Name
  - Type of array
  - Number of elements  
`arrayType arrayName[ numberOfElements ];`
  - Examples:  
`int c[ 10 ];`  
`float myArray[ 3284 ];`
- Defining multiple arrays of same type
  - Format similar to regular variables
  - Example:  
`int b[ 100 ], x[ 27 ];`

## 6.4 Examples Using Arrays

- Initializers

```
int a[ 5 ] = { 10, 20, 30, 40, 50 };
```

- If not enough initializers, rightmost elements become 0

```
int a[ 5 ] = { 0 }; // All elements are 0
```

```
int a[ 5 ] = { 10 }; // First element is 10, other elements are 0
```

- If too many elements, then a syntax error is produced
- C arrays have no bounds checking

- If size omitted, initializers determine it

```
int a[ ] = { 10, 20, 30, 40, 50 };
```

- 5 initializers, therefore compiler knows that array has 5 elements

**fig06\_03.c**

```
1 /* Fig. 6.3: fig06_03.c
2    initializing an array */
3 #include <stdio.h>
4
5 /* function main begins program execution */
6 int main()
7 {
8     int n[ 10 ]; /* n is an array of 10 integers */
9     int i;      /* counter */
10
11     /* initialize elements of array n to 0 */
12     for ( i = 0; i < 10; i++ ) {
13         n[ i ] = 0; /* set element at location i to 0 */
14     } /* end for */
15
16     printf( "%s%13s\n", "Element", "Value" );
17
18     /* output contents of array n in tabular format */
19     for ( i = 0; i < 10; i++ ) {
20         printf( "%7d%13d\n", i, n[ i ] );
21     } /* end for */
22
23     return 0; /* indicates successful termination */
24
25 } /* end main */
```

| Element | Value |
|---------|-------|
| 0       | 0     |
| 1       | 0     |
| 2       | 0     |
| 3       | 0     |
| 4       | 0     |
| 5       | 0     |
| 6       | 0     |
| 7       | 0     |
| 8       | 0     |
| 9       | 0     |



Outline

**Program Output**

## 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

### fig06\_04.c

```
1  /* Fig. 6.4: fig06_04.c
2     Initializing an array with an initializer list */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     /* use initializer list to initialize array n */
9     int n[ 10 ] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
10    int i; /* counter */
11
12    printf( "%s%13s\n", "Element", "Value" );
13
14    /* output contents of array in tabular format */
15    for ( i = 0; i < 10; i++ ) {
16        printf( "%7d%13d\n", i, n[ i ] );
17    } /* end for */
18
19    return 0; /* indicates successful termination */
20
21 } /* end main */
```

| Element | Value |
|---------|-------|
| 0       | 32    |
| 1       | 27    |
| 2       | 64    |
| 3       | 18    |
| 4       | 95    |
| 5       | 14    |
| 6       | 90    |
| 7       | 70    |
| 8       | 60    |
| 9       | 37    |



Outline

**Program Output**



## Outline

### fig06\_05.c

```
1  /* Fig. 6.5: fig06_05.c
2     Initialize the elements of array s to the even integers from 2 to 20 */
3  #include <stdio.h>
4  #define SIZE 10
5
6  /* function main begins program execution */
7  int main()
8  {
9     /* symbolic constant SIZE can be used to specify array size */
10    int s[ SIZE ]; /* array s has 10 elements */
11    int j;         /* counter */
12
13    for ( j = 0; j < SIZE; j++ ) { /* set the values */
14        s[ j ] = 2 + 2 * j;
15    } /* end for */
16
17    printf( "%s%13s\n", "Element", "Value" );
18
19    /* output contents of array s in tabular format */
20    for ( j = 0; j < SIZE; j++ ) {
21        printf( "%7d%13d\n", j, s[ j ] );
22    } /* end for */
23
24    return 0; /* indicates successful termination */
25
26 } /* end main */
```



| Element | Value |
|---------|-------|
| 0       | 2     |
| 1       | 4     |
| 2       | 6     |
| 3       | 8     |
| 4       | 10    |
| 5       | 12    |
| 6       | 14    |
| 7       | 16    |
| 8       | 18    |
| 9       | 20    |



Outline

**Program Output**



## Outline

fig06\_06.c

```
1  /* Fig. 6.6: fig06_06.c
2     Compute the sum of the elements of the array */
3  #include <stdio.h>
4  #define SIZE 12
5
6  /* function main begins program execution */
7  int main()
8  {
9     /* use initializer list to initialize array */
10    int a[ SIZE ] = { 1, 3, 5, 4, 7, 2, 99, 16, 45, 67, 89, 45 };
11    int i;          /* counter */
12    int total = 0; /* sum of array */
13
14    /* sum contents of array a */
15    for ( i = 0; i < SIZE; i++ ) {
16        total += a[ i ];
17    } /* end for */
18
19    printf( "Total of array element values is %d\n", total );
20
21    return 0; /* indicates successful termination */
22
23 } /* end main */
```

```
Total of array element values is 383
```

**Program Output**



## Outline

### fig06\_07.c (Part 1 of 2)

```
1 /* Fig. 6.7: fig06_07.c
2     Student poll (ANKET) program */
3 #include <stdio.h>
4 #define RESPONSE_SIZE 200 /* define array sizes */
5 #define FREQUENCY_SIZE 10
6
7 /* function main begins program execution */
8 int main()
9 {
10     int answer; /* counter */
11     int rating; /* counter */
12
13     /* initialize frequency counters to 0 */
14     int frequency[ FREQUENCY_SIZE ] = { 0 };
15
16     /* place survey responses in array responses */
17     int responses[ RESPONSE_SIZE ] = { 1, 2, 6, 4, 8, 5, 9, 7, 8, 10,
18         1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6,
19         5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };
20
```

Outline

fig06\_07.c (Part 2 of 2)

```

21  /* for each answer, select value of an element of array responses
22     and use that value as subscript in array frequency to
23     determine element to increment */
24  for ( answer = 0; answer < RESPONSE_SIZE; answer++ ) {
25      ++frequency[ responses [ answer ] ];
26  } /* end for */
27
28  /* display results */
29  printf( "%s%17s\n", "Rating", "Frequency" );
30
31  /* output frequencies in tabular format */
32  for ( rating = 1; rating < FREQUENCY_SIZE; rating++ ) {
33      printf( "%6d%17d\n", rating, frequency[ rating - 1 ] );
34  } /* end for */
35
36  return 0; /* indicates successful termination */
37
38 } /* end main */

```

| Rating | Frequency |
|--------|-----------|
| 1      | 2         |
| 2      | 2         |
| 3      | 2         |
| 4      | 2         |
| 5      | 5         |
| 6      | 11        |
| 7      | 5         |
| 8      | 7         |
| 9      | 1         |
| 10     | 3         |

Program Output



## Outline

### fig06\_08.c (Part 1 of 2)

```
1 /* Fig. 6.8: fig06_08.c
2 Histogram printing program */
3 #include <stdio.h>
4 #define SIZE 10
5
6 /* function main begins program execution */
7 int main()
8 {
9     /* use initializer list to initialize array n */
10    int n[ SIZE ] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
11    int i; /* outer counter */
12    int j; /* inner counter */
13
14    printf( "%s%13s%17s\n", "Element", "Value", "Histogram" );
15
16    /* for each element of array n, output a bar in histogram */
17    for ( i = 0; i < SIZE; i++ ) {
18        printf( "%7d%13d", i, n[ i ] );
19
20        for ( j = 1; j <= n[ i ]; j++ ) { /* print one bar */
21            printf( "%c", '*' );
22        } /* end inner for */
23
```

```

24     printf( "\n" ); /* start next line of output */
25 } /* end outer for */
26
27 return 0; /* indicates successful termination */
28
29 } /* end main */

```



## Outline

fig06\_08.c (Part 2 of 2)

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     | *         |



## Outline

### fig06\_09.c (Part 1 of 2)

```
1  /* Fig. 6.9: fig06_09.c
2     Roll a six-sided die 6000 times */
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <time.h>
6  #define SIZE 6
7
8  /* function main begins program execution */
9  int main()
10 {
11     int face;           /* random number with value 1 - 6 */
12     int roll;          /* roll counter */
13     int frequency[ SIZE ] = { 0 }; /* initialize array to 0 */
14
15     srand( time( NULL ) ); /* seed random-number generator */
16
17     /* roll die 6000 times */
18     for ( roll = 1; roll <= 6000; roll++ ) {
19         face = rand() % 6 + 1;
20         ++frequency[ face - 1 ]; /* replaces 26-line switch of Fig. 5.8 */
21     } /* end for */
22
23     printf( "%s%17s\n", "Face", "Frequency" );
24
```

```
25  /* output frequency elements 1-6 in tabular format */
26  for ( face = 1; face < SIZE; face++ ) {
27      printf( "%4d%17d\n", face, frequency[ face - 1 ] );
28  } /* end for */
29
30  return 0; /* indicates successful termination */
31
32 } /* end main */
```

| Face | Frequency |
|------|-----------|
| 1    | 1029      |
| 2    | 951       |
| 3    | 987       |
| 4    | 1033      |
| 5    | 1010      |
| 6    | 990       |



## Outline

**fig06\_09.c (Part 2  
of 2)**

**Program Output**



**fig06\_10.c (Part 1  
of 2)**

```
1 /* Fig. 6.10: fig06_10.c
2    Treating character arrays as strings */
3 #include <stdio.h>
4
5 /* function main begins program execution */
6 int main()
7 {
8     char string1[ 20 ];          /* reserves 20 characters */
9     char string2[ ] = "string literal"; /* reserves 15 characters */
10    int i;                      /* counter */
11
12    /* read string from user into array string2 */
13    printf("Enter a string: ");
14    scanf( "%s", string1 );
15
16    /* output strings */
17    printf( "string1 is: %s\nstring2 is: %s\n"
18           "string1 with spaces between characters is:\n",
19           string1, string2 );
20
21    /* output characters until null character is reached */
22    for ( i = 0; string1[ i ] != '\0'; i++ ) {
23        printf( "%c ", string1[ i ] );
24    } /* end for */
25
```

```
26 printf( "\n" );
27
28 return 0; /* indicates successful termination */
29
30 } /* end main */
```

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



## Outline

26

**fig06\_10.c (Part 2  
of 2)**

## Example: Statistical Calculations

- Get student scores from user (store into an array)
- Calculate the followings

$$\text{Avg } (\bar{x}) = \frac{\sum_{i=1}^N X_i}{N} \quad \text{Variance} = \frac{\sum_{i=1}^N (x_i - \bar{x})^2}{N - 1}$$

$$\text{Standard deviation} = \sqrt{\text{Variance}}$$

$$\text{Absolute deviation} = \frac{\sum_{i=1}^N |x_i - \bar{x}|}{N}$$

## Example: Statistical Calculations

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h> // fabs,sqrt
#define MAXSTUDENTS 100

int main()
{
    int score[MAXSTUDENTS];
    int N = 0; // Number of students
    float avg, variance, std_dev, abs_dev;
    float total = 0.0, sqr_total = 0.0, abs_total = 0.0;
    int i = 0;

    printf("How many students are there ? ");
    scanf("%d", &N);
    for (i = 0; i < N; i++)
    {
        printf("Enter grade of student # %d : ", i + 1);
        scanf("%d", &score[i]);
        total = total + score[i];
    }
}
```

## Example: Statistical Calculations (cont.)

```
avg = total / N;

for (i = 0; i < N; i++)
{
    sqr_total = sqr_total + (score[i] - avg) * (score[i] - avg);
    abs_total = abs_total + fabs(score[i] - avg);
}
variance = sqr_total / (N - 1);
std_dev = sqrt(variance);
abs_dev = abs_total / N;

printf("Average          = %f\n", avg);
printf("Variance          = %f\n", variance);
printf("Standard deviation = %f\n", std_dev);
printf("Absolute deviation = %f\n", abs_dev);

system("pause");
return 0;
}
```

## Example: Statistical Calculations (Output)

**How many students are there ? 4**

**Enter grade of student # 1 : 92**

**Enter grade of student # 2 : 62**

**Enter grade of student # 3 : 70**

**Enter grade of student # 4 : 51**

**Average = 68.750000**

**Variance = 300.916656**

**Standard deviation = 17.346949**

**Absolute deviation = 12.250000**



## Outline

### fig06\_11.c (Part 1 of 3)

```
1  /* Fig. 6.11: fig06_11.c
2     Static arrays are initialized to zero */
3  #include <stdio.h>
4
5  void staticArrayInit( void );    /* function prototype */
6  void automaticArrayInit( void ); /* function prototype */
7
8  /* function main begins program execution */
9  int main()
10 {
11     printf( "First call to each function:\n" );
12     staticArrayInit();
13     automaticArrayInit();
14
15     printf( "\n\nSecond call to each function:\n" );
16     staticArrayInit();
17     automaticArrayInit();
18
19     return 0; /* indicates successful termination */
20
21 } /* end main */
22
```



## Outline

fig06\_11.c (Part 2  
of 3)

```
23 /* function to demonstrate a static local array */
24 void staticArrayInit( void )
25 {
26     /* initializes elements to 0 first time function is called */
27     static int array1[ 3 ];
28     int i; /* counter */
29
30     printf( "\nValues on entering staticArrayInit: \n" );
31
32     /* output contents of array1 */
33     for ( i = 0; i <= 2; i++ ) {
34         printf( "array1[ %d ] = %d ", i, array1[ i ] );
35     } /* end for */
36
37     printf( "\nValues on exiting staticArrayInit: \n" );
38
39     /* modify and output contents of array1 */
40     for ( i = 0; i <= 2; i++ ) {
41         printf( "array1[ %d ] = %d ", i, array1[ i ] += 5 );
42     } /* end for */
43
44 } /* end function staticArrayInit */
45
```





## Outline

### fig06\_11.c (Part 3 of 3)

```
46 /* function to demonstrate an automatic local array */
47 void automaticArrayInit( void )
48 {
49     /* initializes elements each time function is called */
50     int array2[ 3 ] = { 1, 2, 3 };
51     int i; /* counter */
52
53     printf( "\n\nValues on entering automaticArrayInit:\n" );
54
55     /* output contents of array2 */
56     for ( i = 0; i <= 2; i++ ) {
57         printf("array2[ %d ] = %d ", i, array2[ i ] );
58     } /* end for */
59
60     printf( "\n\nValues on exiting automaticArrayInit:\n" );
61
62     /* modify and output contents of array2 */
63     for ( i = 0; i <= 2; i++ ) {
64         printf( "array2[ %d ] = %d ", i, array2[ i ] += 5 );
65     } /* end for */
66
67 } /* end function automaticArrayInit */
```



First call to each function:

Values on entering staticArrayInit:

array1[ 0 ] = 0 array1[ 1 ] = 0 array1[ 2 ] = 0

Values on exiting staticArrayInit:

array1[ 0 ] = 5 array1[ 1 ] = 5 array1[ 2 ] = 5

Values on entering automaticArrayInit:

array2[ 0 ] = 1 array2[ 1 ] = 2 array2[ 2 ] = 3

Values on exiting automaticArrayInit:

array2[ 0 ] = 6 array2[ 1 ] = 7 array2[ 2 ] = 8

Second call to each function:

Values on entering staticArrayInit:

array1[ 0 ] = 5 array1[ 1 ] = 5 array1[ 2 ] = 5

Values on exiting staticArrayInit:

array1[ 0 ] = 10 array1[ 1 ] = 10 array1[ 2 ] = 10

Values on entering automaticArrayInit:

array2[ 0 ] = 1 array2[ 1 ] = 2 array2[ 2 ] = 3

Values on exiting automaticArrayInit:

array2[ 0 ] = 6 array2[ 1 ] = 7 array2[ 2 ] = 8

## 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

fig06\_12.c

```
1  /* Fig. 6.12: fig06_12.c
2     The name of an array is the same as &array[ 0 ] */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main()
7  {
8     char array[ 5 ]; /* define an array of size 5 */
9
10    printf( "    array = %p\n&array[0] = %p\n"
11           "    &array = %p\n",
12           array, &array[ 0 ], &array );
13
14    return 0; /* indicates successful termination */
15
16 } /* end main */
```

```
    array = 0012FF78
&array[0] = 0012FF78
    &array = 0012FF78
```

**Program Output**



## Outline

### fig06\_13.c (Part 1 of 3)

```
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  /* function prototypes */
7  void modifyArray( int b[ ], int size );
8  void modifyElement( int e );
9
10 /* function main begins program execution */
11 int main()
12 {
13     int a[ SIZE ] = { 0, 1, 2, 3, 4 }; /* initialize a */
14     int i; /* counter */
15
16     printf( "Effects of passing entire array by reference: \n\nThe "
17            "values of the original array are: \n" );
18
19     /* output original array */
20     for ( i = 0; i < SIZE; i++ ) {
21         printf( "%3d", a[ i ] );
22     } /* end for */
23
24     printf( "\n" );
25
```



## Outline

### fig06\_13.c (Part 2 of 3)

```
26  /* pass array a to modifyArray by reference */
27  modifyArray( a, SIZE );
28
29  printf( "The values of the modified array are:\n" );
30
31  /* output modified array */
32  for ( i = 0; i < SIZE; i++ ) {
33      printf( "%3d", a[ i ] );
34  } /* end for */
35
36  /* output value of a[ 3 ] */
37  printf( "\n\nEffects of passing array element "
38          "by value:\n\nThe value of a[3] is %d\n", a[ 3 ] );
39
40  modifyElement( a[ 3 ] ); /* pass array element a[ 3 ] by value */
41
42  /* output value of a[ 3 ] */
43  printf( "The value of a[ 3 ] is %d\n", a[ 3 ] );
44
45  return 0; /* indicates successful termination */
46
47 } /* end main */
48
```

**fig06\_13.c (Part 3  
of 3)**

```
49 /* in function modifyArray, "b" points to the original array "a"
50    in memory */
51 void modifyArray( int b[ ], int size )
52 {
53     int j; /* counter */
54
55     /* multiply each array element by 2 */
56     for ( j = 0; j < size; j++ ) {
57         b[ j ] *= 2;
58     } /* end for */
59
60 } /* end function modifyArray */
61
62 /* In function modifyElement, "e" is a local copy of array element
63    a[ 3 ] passed from main */
64 void modifyElement( int e )
65 {
66     /* multiply parameter by 2 */
67     printf( "Value in modifyElement is %d\n", e *= 2 );
68 } /* end function modifyElement */
```





Effects of passing entire array 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 by value:

The value of a[3] is 6

Value in modifyElement is 12

The value of a[3] is 6



**Notice that the original element remains unchanged!!!**



## Outline

### fig06\_14.c (Part 1 of 2)

```
1  /* Fig. 6.14: fig06_14.c
2     Demonstrating the const type qualifier with arrays */
3  #include <stdio.h>
4
5  void tryToModifyArray( const int b[ ] ); /* function prototype */
6
7  /* function main begins program execution */
8  int main()
9  {
10     int a[ ] = { 10, 20, 30 }; /* initialize a */
11
12     tryToModifyArray( a );
13
14     printf("%d %d %d\n", a[ 0 ], a[ 1 ], a[ 2 ] );
15
16     return 0; /* indicates successful termination */
17
18 } /* end main */
19
```



## Outline

fig06\_14.c (Part 2 of 2)

Program Output

```
20 /* in function tryToModifyArray, array b is const, so it cannot be
21    used to modify the original array a in main. */
22 void tryToModifyArray( const int b[ ] )
23 {
24     b[ 0 ] /= 2; /* error */
25     b[ 1 ] /= 2; /* error */
26     b[ 2 ] /= 2; /* error */
27 } /* end function tryToModifyArray */
```

Compiling...

FIG06\_14.C

fig06\_14.c(24) : error C2166: l-value specifies const object

fig06\_14.c(25) : error C2166: l-value specifies const object

fig06\_14.c(26) : error C2166: l-value specifies const object



**const definition causes the Compiler give errors!!**

## 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



## Outline

### fig06\_15.c (Part 1 of 3)

```
1  /* Fig. 6.15: fig06_15.c
2     This program sorts an array's values into ascending order */
3  #include <stdio.h>
4  #define SIZE 10
5
6  /* function main begins program execution */
7  int main()
8  {
9     /* initialize a */
10    int a[ SIZE ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
11    int i;    /* inner counter */
12    int pass; /* outer counter */
13    int hold; /* temporary location used to swap array elements */
14
15    printf( "Data items in original order\n" );
16
17    /* output original array */
18    for ( i = 0; i < SIZE; i++ ) {
19        printf( "%4d", a[ i ] );
20    } /* end for */
21
```

**fig06\_15.c (Part 2  
of 3)**

```
22  /* bubble sort */
23  /* loop to control number of passes */
24  for ( pass = 1; pass < SIZE; pass++ ) {
25
26      /* loop to control number of comparisons per pass */
27      for ( i = 0; i < SIZE - 1; i++ ) {
28
29          /* compare adjacent elements and swap them if first
30             element is greater than second element */
31          if ( a[ i ] > a[ i + 1 ] ) {
32              hold = a[ i ];
33              a[ i ] = a[ i + 1 ];
34              a[ i + 1 ] = hold;
35          } /* end if */
36
37      } /* end inner for */
38
39  } /* end outer for */
40
41  printf( "\nData items in ascending order\n" );
42
```



## Outline

**fig06\_15.c (Part 3  
of 3)**

```
43  /* output sorted array */
44  for ( i = 0; i < SIZE; i++ ) {
45      printf( "%4d", a[ i ] );
46  } /* end for */
47
48  printf( "\n" );
49
50  return 0; /* indicates successful termination */
51
```

Data items in original order

2 6 4 8 10 12 89 68 45 37

Data items in ascending order

2 4 6 8 10 12 37 45 68 89

**Program Output**

## 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

### fig06\_16.c (Part 1 of 8)

```
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  /* function prototypes */
8  void mean( const int answer[ ] );
9  void median( int answer[ ] );
10 void mode( int freq[ ], const int answer[ ] );
11 void bubbleSort( int a[ ] );
12 void printArray( const int a[ ] );
13
14 /* function main begins program execution */
15 int main()
16 {
17     int frequency[ 10 ] = { 0 }; /* initialize array frequency */
18
```



## Outline

### fig06\_16.c (Part 2 of 8)

```
19  /* initialize array response */
20  int response[ SIZE ] =
21      { 6, 7, 8, 9, 8, 7, 8, 9, 8, 9,
22        7, 8, 9, 5, 9, 8, 7, 8, 7, 8,
23        6, 7, 8, 9, 3, 9, 8, 7, 8, 7,
24        7, 8, 9, 8, 9, 8, 9, 7, 8, 9,
25        6, 7, 8, 7, 8, 7, 9, 8, 9, 2,
26        7, 8, 9, 8, 9, 8, 9, 7, 5, 3,
27        5, 6, 7, 2, 5, 3, 9, 4, 6, 4,
28        7, 8, 9, 6, 8, 7, 8, 9, 7, 8,
29        7, 4, 4, 2, 5, 3, 8, 7, 5, 6,
30        4, 5, 6, 1, 6, 5, 7, 8, 7 };
31
32  /* process responses */
33  mean( response );
34  median( response );
35  mode( frequency, response );
36
37  return 0; /* indicates successful termination */
38
39 } /* end main */
40
```



fig06\_16.c (Part 3  
of 8)

```

41 /* calculate average of all response values */
42 void mean( const int answer[ ] )
43 {
44     int j;          /* counter */
45     int total = 0; /* variable to hold sum of array elements */
46
47     printf( "%s\n%s\n%s\n", "*****", "  Mean", "*****" );
48
49     /* total response values */
50     for ( j = 0; j < SIZE; j++ ) {
51         total += answer[ j ];
52     } /* end for */
53
54     printf( "The mean is the average value of the data\n"
55            "items. The mean is equal to the total of\n"
56            "all the data items divided by the number\n"
57            "of data items ( %d ). The mean value for\n"
58            "this run is: %d / %d = %.4f\n\n",
59            SIZE, total, SIZE, ( double ) total / SIZE );
60 } /* end function mean */
61

```



## Outline

**fig06\_16.c (Part 4  
of 8)**

```
62 /* sort array and determine median element's value */
63 void median( int answer[ ] )
64 {
65     printf( "\n%s\n%s\n%s\n%s",
66             "*****", " Median", "*****",
67             "The unsorted array of responses is" );
68
69     printArray( answer ); /* output unsorted array */
70
71     bubbleSort( answer ); /* sort array */
72
73     printf( "\n\nThe sorted array is" );
74     printArray( answer ); /* output sorted array */
75
76     /* display median element */
77     printf( "\n\nThe median is element %d of\n"
78             "the sorted %d element array.\n"
79             "For this run the median is %d\n\n",
80             SIZE / 2, SIZE, answer[ SIZE / 2 ] );
81 } /* end function median */
82
```



## Outline

**fig06\_16.c (Part 5  
of 8)**

```
83 /* determine most frequent response */
84 void mode( int freq[ ], const int answer[ ] )
85 {
86     int rating;          /* counter */
87     int j;              /* counter */
88     int h;              /* counter */
89     int largest = 0;    /* represents largest frequency */
90     int modeValue = 0; /* represents most frequent response */
91
92     printf( "\n%s\n%s\n%s\n",
93            "*****", "  Mode", "*****" );
94
95     /* initialize frequencies to 0 */
96     for ( rating = 1; rating <= 9; rating++ ) {
97         freq[ rating ] = 0;
98     } /* end for */
99
100    /* summarize frequencies */
101    for ( j = 0; j < SIZE; j++ ) {
102        ++freq[ answer[ j ] ];
103    } /* end for */
104
```



## Outline

### fig06\_16.c (Part 6 of 8)

```

105  /* output headers for result columns */
106  printf( "%s%11s%19s\n\n%54s\n%54s\n\n",
107          "Response", "Frequency", "Histogram",
108          "1  1  2  2", "5  0  5  0  5" );
109
110  /* output results */
111  for ( rating = 1; rating <= 9; rating++ ) {
112      printf( "%8d%11d      ", rating, freq[ rating ] );
113
114      /* keep track of mode value and largest frequency value */
115      if ( freq[ rating ] > largest ) {
116          largest = freq[ rating ];
117          modeValue = rating;
118      } /* end if */
119
120      /* output histogram bar representing frequency value */
121      for ( h = 1; h <= freq[ rating ]; h++ ) {
122          printf( "*" );
123      } /* end inner for */
124
125      printf( "\n" ); /* being new line of output */
126  } /* end outer for */
127

```

**fig06\_16.c (Part 7  
of 8)**

```
128  /* display the mode value */
129  printf( "The mode is the most frequent value.\n"
130         "For this run the mode is %d which occurred"
131         " %d times.\n", modeValue, largest );
132} /* end function mode */
133
134/* function that sorts an array with bubble sort algorithm */
135void bubbleSort( int a[ ] )
136{
137  int pass; /* counter */
138  int j;    /* counter */
139  int hold; /* temporary location used to swap elements */
140
141  /* loop to control number of passes */
142  for ( pass = 1; pass < SIZE; pass++ ) {
143
144      /* loop to control number of comparisons per pass */
145      for ( j = 0; j < SIZE - 1; j++ ) {
146
147          /* swap elements if out of order */
148          if ( a[ j ] > a[ j + 1 ] ) {
149              hold = a[ j ];
150              a[ j ] = a[ j + 1 ];
151              a[ j + 1 ] = hold;
152          } /* end if */
153
```

**fig06\_16.c (Part 8  
of 8)**

```
154     } /* end inner for */
155
156 } /* end outer for */
157
158} /* end function bubbleSort */
159
160/* output array contents (20 values per row) */
161void printArray( const int a[ ] )
162{
163     int j; /* counter */
164
165     /* output array contents */
166     for ( j = 0; j < SIZE; j++ ) {
167
168         if ( j % 20 == 0 ) { /* begin new line every 20 values */
169             printf( "\n" );
170         } /* end if */
171
172         printf( "%2d", a[ j ] );
173     } /* end for */
174
175} /* end function printArray */
```





\*\*\*\*\*

### 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

```
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
```

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 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 8
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
```



**Program Output  
(continued)**

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

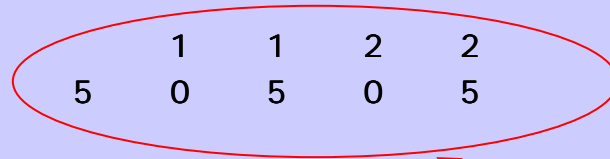
\*\*\*\*\*

Mode

\*\*\*\*\*

Response    Frequency

Hi stogram



| Response | Frequency | Hi stogram |
|----------|-----------|------------|
| 1        | 1         | *          |
| 2        | 3         | ***        |
| 3        | 4         | ****       |
| 4        | 5         | *****      |
| 5        | 8         | *****      |
| 6        | 9         | *****      |
| 7        | 23        | *****      |
| 8        | 27        | *****      |
| 9        | 19        | *****      |

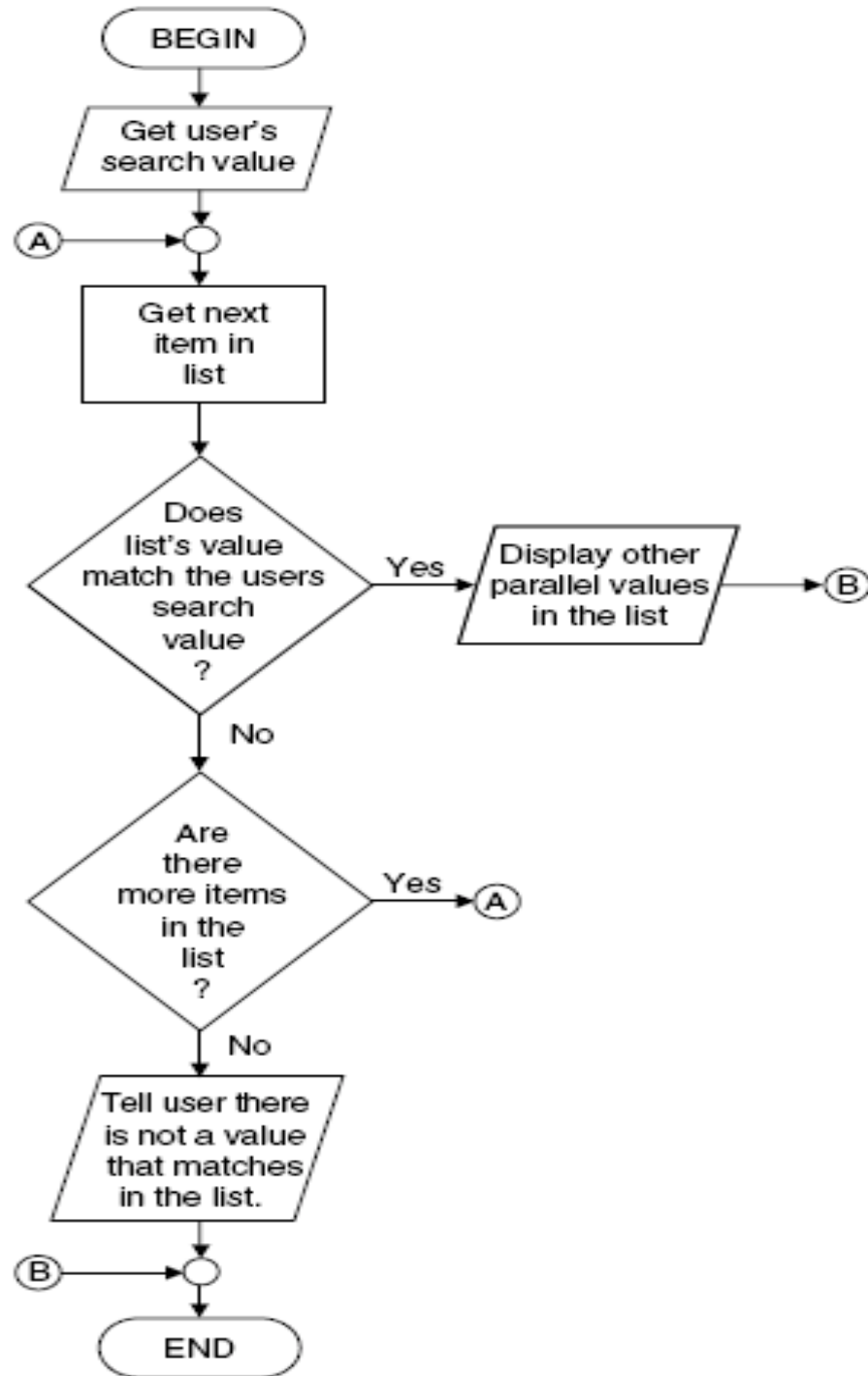
**Histogram Scales  
5, 10, 15, 20, 25**

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

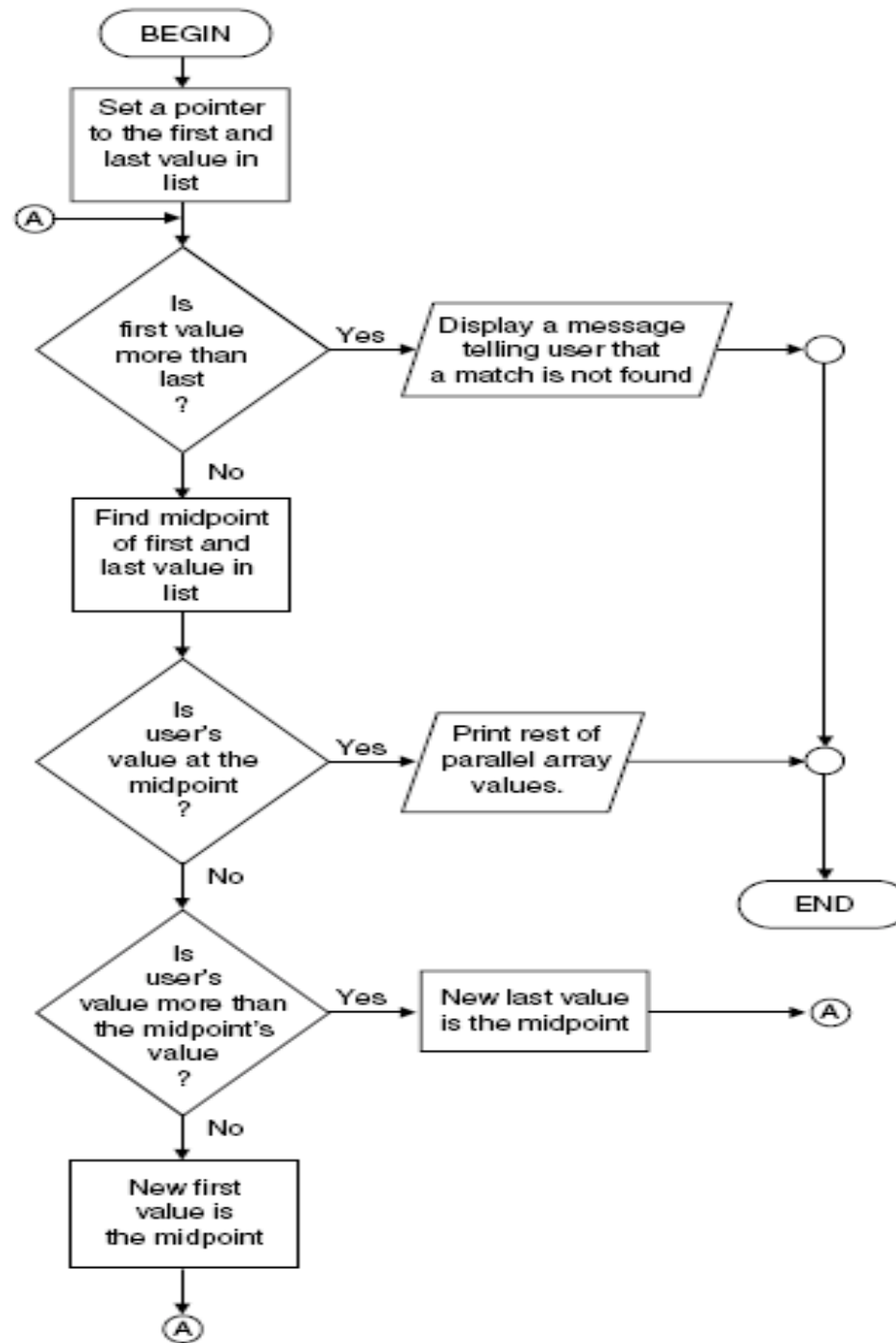
# Linear search



## 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

# Binary search





## Outline

### fig06\_18.c (Part 1 of 3)

```
1  /* Fig. 6.18: fig06_18.c
2     Linear search of an array */
3  #include <stdio.h>
4  #define SIZE 100
5
6  /* function prototype */
7  int linearSearch( const int array[ ], int key, int size );
8
9  /* function main begins program execution */
10 int main()
11 {
12     int a[ SIZE ]; /* create array a */
13     int x;         /* counter */
14     int searchKey; /* value to locate in a */
15     int element;  /* variable to hold location of searchKey or -1 */
16
17     /* create data */
18     for ( x = 0; x < SIZE; x++ ) {
19         a[ x ] = 2 * x;
20     } /* end for */
21
22     printf( "Enter integer search key:\n" );
23     scanf( "%d", &searchKey );
24
```

**fig06\_18.c (Part 2  
of 3)**

```
25  /* attempt to locate searchKey in array a */
26  element = linearSearch( a, searchKey, SIZE );
27
28  /* display results */
29  if ( element != -1 ) {
30      printf( "Found value in element %d\n", element );
31  } /* end if */
32  else {
33      printf( "Value not found\n" );
34  } /* end else */
35
36  return 0; /* indicates successful termination */
37
38 } /* end main */
39
40 /* compare key to every element of array until the location is found
41    or until the end of array is reached; return subscript of element
42    if key or -1 if key is not found */
43 int linearSearch( const int array[ ], int key, int size )
44 {
45     int n; /* counter */
46
47     /* loop through array */
48     for ( n = 0; n < size; ++n ) {
49
```



  
  
Outline

```
50     if ( array[ n ] == key ) {  
51         return n; /* return location of key */  
52     } /* end if */  
53  
54 } /* end for */  
55  
56 return -1; /* key not found */  
57  
58 } /* end function linearSearch */
```

```
Enter integer search key:  
36  
Found value in element 18
```

```
Enter integer search key:  
37  
Value not found
```

fig06\_18.c (Part 3  
of 3)

**Program Output**



## Outline

### fig06\_19.c (Part 1 of 5)

```
1  /* Fig. 6.19: fig06_19.c
2     Binary search of an array */
3  #include <stdio.h>
4  #define SIZE 15
5
6  /* function prototypes */
7  int binarySearch( const int b[ ], int searchKey, int low, int high );
8  void printHeader( void );
9  void printRow( const int b[ ], int low, int mid, int high );
10
11 /* function main begins program execution */
12 int main()
13 {
14     int a[ SIZE ]; /* create array a */
15     int i;         /* counter */
16     int key;       /* value to locate in array a */
17     int result;    /* variable to hold location of key or -1 */
18
19     /* create data */
20     for ( i = 0; i < SIZE; i++ ) {
21         a[ i ] = 2 * i;
22     } /* end for */
23
24     printf( "Enter a number between 0 and 28: " );
25     scanf( "%d", &key );
26
```



## Outline

### fig06\_19.c (Part 2 of 5)

```
27  printHeader();
28
29  /* search for key in array a */
30  result = binarySearch( a, key, 0, SIZE - 1 );
31
32  /* display results */
33  if ( result != -1 ) {
34      printf( "\n%d found in array element %d\n", key, result );
35  } /* end if */
36  else {
37      printf( "\n%d not found\n", key );
38  } /* end else */
39
40  return 0; /* indicates successful termination */
41
42 } /* end main */
43
44 /* function to perform binary search of an array */
45 int binarySearch( const int b[ ], int searchKey, int low, int high )
46 {
47     int middle; /* variable to hold middle element of array */
48
```



## Outline

### fig06\_19.c (Part 3 of 5)

```
49  /* loop until low subscript is greater than high subscript */
50  while ( low <= high ) {
51
52      /* determine middle element of subarray being searched */
53      middle = ( low + high ) / 2;
54
55      /* display subarray used in this loop iteration */
56      printRow( b, low, middle, high );
57
58      /* if searchKey matched middle element, return middle */
59      if ( searchKey == b[ middle ] ) {
60          return middle;
61      } /* end if */
62
63      /* if searchKey less than middle element, set new high */
64      else if ( searchKey < b[ middle ] ) {
65          high = middle - 1; /* search low end of array */
66      } /* end else if */
67
68      /* if searchKey greater than middle element, set new low */
69      else {
70          low = middle + 1; /* search high end of array */
71      } /* end else */
72
73  } /* end while */
74
```



## Outline

### fig06\_19.c (Part 4 of 5)

```
75     return -1;    /* searchKey not found */
76
77 } /* end function binarySearch */
78
79 /* Print a header for the output */
80 void printHeader( void )
81 {
82     int i; /* counter */
83
84     printf( "\nSubscripts: \n" );
85
86     /* output column head */
87     for ( i = 0; i < SIZE; i++ ) {
88         printf( "%3d ", i );
89     } /* end for */
90
91     printf( "\n" ); /* start new line of output */
92
93     /* output line of - characters */
94     for ( i = 1; i <= 4 * SIZE; i++ ) {
95         printf( "-" );
96     } /* end for */
97
98     printf( "\n" ); /* start new line of output */
99 } /* end function printHeader */
100
```

**fig06\_19.c (Part 5  
of 5)**

```
101/* Print one row of output showing the current
102 part of the array being processed. */
103void printRow( const int b[ ], int low, int mid, int high )
104{
105 int i; /* counter */
106
107 /* loop through entire array */
108 for ( i = 0; i < SIZE; i++ ) {
109
110     /* display spaces if outside current subarray range */
111     if ( i < low || i > high ) {
112         printf( "    " );
113     } /* end if */
114     else if ( i == mid ) { /* display middle element */
115         printf( "%3d*", b[ i ] ); /* mark middle value */
116     } /* end else if */
117     else { /* display other elements in subarray */
118         printf( "%3d ", b[ i ] );
119     } /* end else */
120
121 } /* end for */
122
123 printf( "\n" ); /* start new line of output */
124} /* end function printRow */
```

**Program Output**

Enter a number between 0 and 28: 25

Subscri pts:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

-----

0 2 4 6 8 10 12 14\* 16 18 20 22 24 26 28  
 16 18 20 22\* 24 26 28  
 24 26\* 28  
 24\*

25 not found

Enter a number between 0 and 28: 8

Subscri pts:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

-----

0 2 4 6 8 10 12 14\* 16 18 20 22 24 26 28  
 0 2 4 6\* 8 10 12  
 8 10\* 12  
 8\*

8 found in array element 4



## Outline

Enter a number between 0 and 28: 6

Subscri pts:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

-----  
0 2 4 6 8 10 12 14\* 16 18 20 22 24 26 28

0 2 4 6\* 8 10 12

6 found i n array el ement 3

**Program Output  
(continued)**



## 6.9 Multiple-Subscripted Arrays

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

|       | Col umn     | Col umn 1   | Col umn     | Col umn 3   |
|-------|-------------|-------------|-------------|-------------|
| Row 0 | a[ 0 ][ 0 ] | a[ 0 ][ 1 ] | a[ 0 ][ 2 ] | a[ 0 ][ 3 ] |
| Row 1 | a[ 1 ][ 0 ] | a[ 1 ][ 1 ] | a[ 1 ][ 2 ] | a[ 1 ][ 3 ] |
| Row 2 | a[ 2 ][ 0 ] | a[ 2 ][ 1 ] | a[ 2 ][ 2 ] | a[ 2 ][ 3 ] |

Col umn subscript

Array name Row subscript

## 6.9 Multiple-Subscripted Arrays

- Initialization

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

|   |   |
|---|---|
| 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 | 0 |
| 3 | 4 |

- Referencing elements

- Specify row, then column

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

Outline**fig06\_21.c (Part 1 of 2)**

```
1  /* Fig. 6.21: fig06_21.c
2     Initializing multidimensional arrays */
3  #include <stdio.h>
4
5  void printArray( const int a[][ 3 ] ); /* function prototype */
6
7  /* function main begins program execution */
8  int main()
9  {
10     /* initialize array1, array2, array3 */
11     int array1[ 2 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 } };
12     int array2[ 2 ][ 3 ] = { 1, 2, 3, 4, 5 };
13     int array3[ 2 ][ 3 ] = { { 1, 2 }, { 4 } };
14
15     printf( "Values in array1 by row are:\n" );
16     printArray( array1 );
17
18     printf( "Values in array2 by row are:\n" );
19     printArray( array2 );
20
21     printf( "Values in array3 by row are:\n" );
22     printArray( array3 );
23
24     return 0; /* indicates successful termination */
25
26 } /* end main */
27
```



## Outline

fig06\_21.c (Part 2  
of 2)

```
28 /* function to output array with two rows and three columns */
29 void printArray( const int a[ ][ 3 ] )
30 {
31     int i; /* counter */
32     int j; /* counter */
33
34     /* loop through rows */
35     for ( i = 0; i <= 1; i++ ) {
36
37         /* output column values */
38         for ( j = 0; j <= 2; j++ ) {
39             printf( "%d ", a[ i ][ j ] );
40         } /* end inner for */
41
42         printf( "\n" ); /* start new line of output */
43     } /* end outer for */
44
45 } /* end function printArray */
```

```
Values in array1 by row are:
1 2 3
4 5 6
Values in array2 by row are:
1 2 3
4 5 0
Values in array3 by row are:
1 2 0
4 0 0
```

**Program Output**

## Example: Adding Two Matrices

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int a[2][2] = {{10, 15}, {20, 5}}; /* Matrix a */
    int b[2][2] = {{25, 5}, { 6, 0}}; /* Matrix b */
    int c[2][2]; /* Matrix c */
    int i, j;
    printf ("RESULTING ADDITION MATRIX \n\n");

    for(i=0; i<2; i++) {
        for(j=0; j<2; j++) {
            c[i][j] = a[i][j] + b[i][j];
            printf ("%d\t", c[i][j]);
        }
        printf ("\n"); /* new line */
    }

    system("PAUSE");
    return 0;
}
```

## Example: Adding Two Matrices (Output)

RESULTING ADDITION MATRIX

35            20

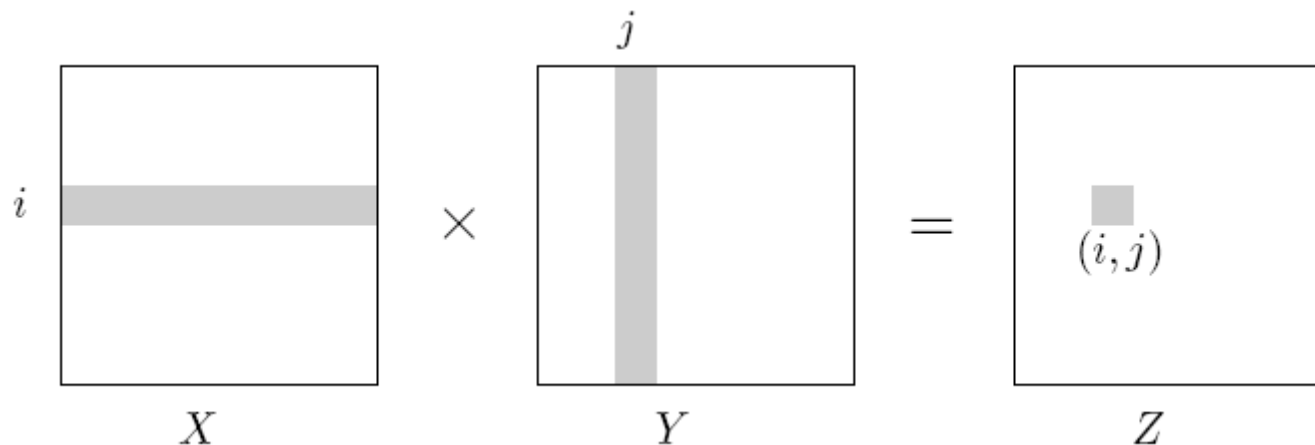
26            5

Devam etmek için bir tuşa basın . . .

## Example: Multiplying Two Matrices (1)

- The product of two  $n \times n$  matrices  $X$  and  $Y$  is a third  $n \times n$  matrix  $Z = X \cdot Y$ , with  $(i,j)^{\text{th}}$  entry

$$Z_{ij} = \sum_{k=1}^N X_{ik} Y_{kj}$$



## Example: Multiplying Two Matrices (2)

- Example: X and Y are two 2x2 matrices.
- Z is also a 2x2 matrix ( $Z = X \cdot Y$ )

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, \quad Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

$$XY = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$



## Example: Multiplying Two Matrices (3)

```
#include <stdio.h>
#include <stdlib.h>
#define N 2
#define M 4
#define L 3

int main()
{
    int a[N][M] = {{8, 5, -6, 7},
                  {0, 2, 1, 4}}; /* Matrix a */

    int b[M][L] = {{3, -9, 1},
                  {2, 5, 8},
                  {-2, 4, 0},
                  {1, 7, 6}}; /* Matrix b */

    int c[N][L]; /* Matrix c */

    int i,j,k;
```

## Example: Multiplying Two Matrices (4)

```
/* First, compute the multiplication */
for (i = 0; i < N; i++)
{
    for (j = 0; j < L; j++)
    {
        c[i][j] = 0;
        for (k = 0; k < M; k++)
            c[i][j] = c[i][j] +
                a[i][k] * b[k][j];
    }
}
```

## Example: Multiplying Two Matrices (5)

```
/* Now, display the resulting matrix */  
  
printf ("RESULTING MULTIPLICATION MATRIX \n\n");  
  
for(i=0; i<N; i++)  
{  
    for(j=0; j<L; j++)  
        printf ("%d\t", c[i][j]);  
  
    printf ("\n"); /* new line */  
}  
  
system("PAUSE");  
return 0;  
}
```

## Example: Multiplying Two Matrices (Output)

RESULTING MULTIPLICATION MATRIX

|    |     |    |
|----|-----|----|
| 53 | -22 | 90 |
| 6  | 42  | 40 |

Devam etmek için bir tuşa basın . . .

  
Outline  


```
1  /* Fig. 6.22: fig06_22.c
2     Double-subscripted array example */
3  #include <stdio.h>
4  #define STUDENTS 3
5  #define EXAMS 4
6
7  /* function prototypes */
8  int minimum( const int grades[ ][ EXAMS ], int pupils, int tests );
9  int maximum( const int grades[ ][ EXAMS ], int pupils, int tests );
10 double average( const int setOfGrades[ ], int tests );
11 void printArray( const int grades[ ][ EXAMS ], int pupils, int tests );
12
13 /* function main begins program execution */
14 int main()
15 {
16     int student; /* counter */
17
18     /* initialize student grades for three students (rows) */
19     const int studentGrades[ STUDENTS ][ EXAMS ] =
20         { { 77, 68, 86, 73 },
21           { 96, 87, 89, 78 },
22           { 70, 90, 86, 81 } };
23
```

fig06\_22.c (Part 1  
of 6)

**fig06\_22.c (Part 2  
of 6)**

```
24  /* output array studentGrades */
25  printf( "The array is:\n" );
26  printArray( studentGrades, STUDENTS, EXAMS );
27
28  /* determine smallest and largest grade values */
29  printf( "\n\nLowest grade: %d\nHighest grade: %d\n",
30         minimum( studentGrades, STUDENTS, EXAMS ),
31         maximum( studentGrades, STUDENTS, EXAMS ) );
32
33  /* calculate average grade for each student */
34  for ( student = 0; student <= STUDENTS - 1; student++ ) {
35      printf( "The average grade for student %d is %.2f\n",
36             student, average( studentGrades[ student ], EXAMS ) );
37  } /* end for */
38
39  return 0; /* indicates successful termination */
40
41 } /* end main */
42
43 /* Find the minimum grade */
44 int minimum( const int grades[ ][ EXAMS ], int pupils, int tests )
45 {
46     int i;           /* counter */
47     int j;           /* counter */
48     int lowGrade = 100; /* initialize to highest possible grade */
49
```

**fig06\_22.c (Part 3  
of 6)**

```
50  /* loop through rows of grades */
51  for ( i = 0; i < pupils; i++ ) {
52
53      /* loop through columns of grades */
54      for ( j = 0; j < tests; j++ ) {
55
56          if ( grades[ i ][ j ] < lowGrade ) {
57              lowGrade = grades[ i ][ j ];
58          } /* end if */
59
60      } /* end inner for */
61
62  } /* end outer for */
63
64  return lowGrade; /* return minimum grade */
65
66 } /* end function minimum */
67
68 /* Find the maximum grade */
69 int maximum( const int grades[ ][ EXAMS ], int pupils, int tests )
70 {
71     int i;          /* counter */
72     int j;          /* counter */
73     int highGrade = 0; /* initialize to lowest possible grade */
74
```

**fig06\_22.c (Part 4  
of 6)**

```
75  /* loop through rows of grades */
76  for ( i = 0; i < pupils; i++ ) {
77
78      /* loop through columns of grades */
79      for ( j = 0; j < tests; j++ ) {
80
81          if ( grades[ i ][ j ] > highGrade ) {
82              highGrade = grades[ i ][ j ];
83          } /* end if */
84
85      } /* end inner for */
86
87  } /* end outer for */
88
89  return highGrade; /* return maximum grade */
90
91 } /* end function maximum */
92
93 /* Determine the average grade for a particular student */
94 double average( const int setOfGrades[ ], int tests )
95 {
96     int i;          /* counter */
97     int total = 0; /* sum of test grades */
98
```



**fig06\_22.c (Part 5  
of 6)**

```
99  /* total all grades for one student */
100  for ( i = 0; i < tests; i++ ) {
101      total += setOfGrades[ i ];
102  } /* end for */
103
104  return ( double ) total / tests; /* average */
105
106} /* end function average */
107
108/* Print the array */
109void printArray( const int grades[ ][ EXAMS ], int pupils, int tests )
110{
111  int i; /* counter */
112  int j; /* counter */
113
114  /* output column heads */
115  printf( "                [0] [1] [2] [3]" );
116
117  /* output grades in tabular format */
118  for ( i = 0; i < pupils; i++ ) {
119
120      /* output label for row */
121      printf( "\nstudentGrades[%d] ", i );
122
```

**fig06\_22.c (Part 6  
of 6)**

```
123     /* output grades for one student */
124     for ( j = 0; j < tests; j++ ) {
125         printf( "%-5d", grades[ i ][ j ] );
126     } /* end inner for */
127
128 } /* end outer for */
129
130} /* end function printArray */
```

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