Computer Programming I

Ming-Feng Tsai (Victor Tsai)

Dept. of Computer Science
National Chengchi University

C Arrays

Array Examples (Cont.)

- String
  - A character array
  - For example

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

  - The string "first" contains **five characters plus** a special string-termination character called the **null character**.
  - Thus, array string1 actually contains **six elements**.
  - The character constant representing **the null character is '\0'.**
  - All strings in C end with this character.

Array Examples (Cont.)

- The preceding definition is equivalent to

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

- For example

```
char string2[ 20 ];
```

  - creates a character array capable of storing a string of at most 19 characters and a terminating null character.

- The statement

```
scanf( "%s", string2 );
// read a string from stdin
```

Array Examples (Cont.)

- Ex: fig06_10.c

```c
    char string1[ 20 ]; /* reserves 20 characters */
    char string2[] = "string literal"; /* reserves 15 characters */
    int i; /* counter */

    /* read string from user into array string1 */
    printf("Enter a string: ");
    scanf( "%s", string1 ); /* input ended by whitespace character */

    /* output strings */
    printf( "string1 is: %s\nstring2 is: %s\n"
            "string1 with spaces between characters is:\n",
            string1, string2 );

    /* output characters until null character is reached */
    for ( i = 0; string1[ i ] != '\0'; i++ ) {
        printf( "%c ", string1[ i ] );
    } /* end for */

    printf( "\n" );
```

Array Examples (Cont.)

- Ex: fig06_10.c

```
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 string1 */
13    printf("Enter a string: ");
14    scanf( "%s", string1 ); /* input ended by whitespace character */
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" );
```

Array Examples (Cont.)

- Ex: fig06_10.c

```
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 string1 */
13    printf("Enter a string: ");
14    scanf( "%s", string1 ); /* input ended by whitespace character */
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" );
```

Create two strings

Array Examples (Cont.)

- Ex: fig06_10.c

```
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 string1 */
13    printf("Enter a string: ");
14    scanf( "%s", string1 ); /* input ended by whitespace character */
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" );
```

Create two strings

Array Examples (Cont.)

- Ex: fig06_10.c

```
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 string1 */
13    printf("Enter a string: ");
14    scanf( "%s", string1 ); /* input ended by whitespace character */
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" );
```

Create two strings

Read a string

Array Examples (Cont.)

- Ex: fig06_10.c

```
 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 string1 */
13    printf("Enter a string: ");
14    scanf( "%s", string1 ); /* input ended by whitespace character */
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" );
```

Create two strings

Read a string

Array Examples (Cont.)

- Ex: fig06_10.c

```
 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 string1 */
13    printf("Enter a string: ");
14    scanf( "%s", string1 ); /* input ended by whitespace character */
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" );
```

Create two strings

Read a string

string[i] != '\0'

'%c'

Array Examples (Cont.)

- Ex: fig06_10.c

```
 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 string1 */
13    printf("Enter a string: ");
14    scanf( "%s", string1 ); /* input ended by whitespace character */
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" );
```

Create two strings

Read a string

string[i] != '\0'

'%c'

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

Array Examples (Cont.)

- A static local variable exists for the duration of the program, but is visible only in the function body.
- Arrays that are static are initialized once at compile time.

Array Examples (Cont.)

- A static local variable exists for the duration of the program, but is visible only in the function body.
- Arrays that are static are initialized once at compile time.

Performance Tip 6.2

In functions that contain automatic arrays where the function is in and out of scope frequently, make the array `static` *so it's not created each time the function is called.*

Array Examples (Cont.)

- Example: fig06_11.c

```
 5 void staticArrayInit( void ); /* function prototype */
 6 void automaticArrayInit( void ); /* function prototype */
 7
 8 /* function main begins program execution */
 9 int main( void )
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    return 0; /* indicates successful termination */
19 } /* end main */
```

Array Examples (Cont.)

- Example: fig06_11.c

```
 5 void staticArrayInit( void ); /* function prototype */
 6 void automaticArrayInit( void ); /* function prototype */
 7
 8 /* function main begins program execution */
 9 int main( void )
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    return 0; /* indicates successful termination */
19 } /* end main */
```

Array Examples (Cont.)

- Example: fig06_11.c

```
 5 void staticArrayInit( void ); /* function prototype */
 6 void automaticArrayInit( void ); /* function prototype */
 7
 8 /* function main begins program execution */
 9 int main( void )
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    return 0; /* indicates successful termination */
19 } /* end main */
```

declare two functions

Array Examples (Cont.)

- Example: fig06_11.c

```c
void staticArrayInit( void )
{
    /* initializes elements to 0 first time function is called */
    static int array1[ 3 ];
    int i; /* counter */

    printf( "\nValues on entering staticArrayInit:\n" );

    /* output contents of array1 */
    for ( i = 0; i <= 2; i++ ) {
        printf( "array1[ %d ] = %d  ", i, array1[ i ] );
    } /* end for */

    printf( "\nValues on exiting staticArrayInit:\n" );

    /* modify and output contents of array1 */
    for ( i = 0; i <= 2; i++ ) {
        printf( "array1[ %d ] = %d  ", i, array1[ i ] += 5 );
    } /* end for */
} /* end function staticArrayInit */
```

```c
void automaticArrayInit( void )
{
    /* initializes elements each time function is called */
    int array2[ 3 ] = { 1, 2, 3 };
    int i; /* counter */

    printf( "\n\nValues on entering automaticArrayInit:\n" );

    /* output contents of array2 */
    for ( i = 0; i <= 2; i++ ) {
        printf("array2[ %d ] = %d  ", i, array2[ i ] );
    } /* end for */

    printf( "\nValues on exiting automaticArrayInit:\n" );

    /* modify and output contents of array2 */
    for ( i = 0; i <= 2; i++ ) {
        printf( "array2[ %d ] = %d  ", i, array2[ i ] += 5 );
    } /* end for */
} /* end function automaticArrayInit */
```

Array Examples (Cont.)

- Example: fig06_11.c

```c
void staticArrayInit( void )
{
    /* initializes elements to 0 first time function is called */
    static int array1[ 3 ];
    int i; /* counter */

    printf( "\nValues on entering staticArrayInit:\n" );

    /* output contents of array1 */
    for ( i = 0; i <= 2; i++ ) {
        printf( "array1[ %d ] = %d  ", i, array1[ i ] );
    } /* end for */

    printf( "\nValues on exiting staticArrayInit:\n" );

    /* modify and output contents of array1 */
    for ( i = 0; i <= 2; i++ ) {
        printf( "array1[ %d ] = %d  ", i, array1[ i ] += 5 );
    } /* end for */
} /* end function staticArrayInit */
```

```c
void automaticArrayInit( void )
{
    /* initializes elements each time function is called */
    int array2[ 3 ] = { 1, 2, 3 };
    int i; /* counter */

    printf( "\n\nValues on entering automaticArrayInit:\n" );

    /* output contents of array2 */
    for ( i = 0; i <= 2; i++ ) {
        printf("array2[ %d ] = %d  ", i, array2[ i ] );
    } /* end for */

    printf( "\nValues on exiting automaticArrayInit:\n" );

    /* modify and output contents of array2 */
    for ( i = 0; i <= 2; i++ ) {
        printf( "array2[ %d ] = %d  ", i, array2[ i ] += 5 );
    } /* end for */
} /* end function automaticArrayInit */
```

Array Examples (Cont.)

- Example: fig06_11.c

```c
void staticArrayInit( void )
{
    /* initializes elements to 0 first time function is called */
    static int array1[ 3 ];
    int i; /* counter */

    printf( "\nValues on entering staticArrayInit:\n" );

    /* output contents of array1 */
    for ( i = 0; i <= 2; i++ ) {
        printf( "array1[ %d ] = %d  ", i, array1[ i ] );
    } /* end for */

    printf( "\nValues on exiting staticArrayInit:\n" );

    /* modify and output contents of array1 */
    for ( i = 0; i <= 2; i++ ) {
        printf( "array1[ %d ] = %d  ", i, array1[ i ] += 5 );
    } /* end for */
} /* end function staticArrayInit */
```

```c
void automaticArrayInit( void )
{
    /* initializes elements each time function is called */
    int array2[ 3 ] = { 1, 2, 3 };
    int i; /* counter */

    printf( "\n\nValues on entering automaticArrayInit:\n" );

    /* output contents of array2 */
    for ( i = 0; i <= 2; i++ ) {
        printf("array2[ %d ] = %d  ", i, array2[ i ] );
    } /* end for */

    printf( "\nValues on exiting automaticArrayInit:\n" );

    /* modify and output contents of array2 */
    for ( i = 0; i <= 2; i++ ) {
        printf( "array2[ %d ] = %d  ", i, array2[ i ] += 5 );
    } /* end for */
} /* end function automaticArrayInit */
```

Array Examples (Cont.)

- Example: fig06_11.c

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

```
44 void automaticArrayInit( void )
45 {
46    /* initializes elements each time function is called */
47    int array2[ 3 ] = { 1, 2, 3 };
48    int i; /* counter */
49
50    printf( "\n\nValues on entering automaticArrayInit:\n" );
51
52    /* output contents of array2 */
53    for ( i = 0; i <= 2; i++ ) {
54       printf("array2[ %d ] = %d  ", i, array2[ i ] );
55    } /* end for */
56
57    printf( "\nValues on exiting automaticArrayInit:\n" );
58
59    /* modify and output contents of array2 */
60    for ( i = 0; i <= 2; i++ ) {
61       printf( "array2[ %d ] = %d  ", i, array2[ i ] += 5 );
62    } /* end for */
63 } /* end function automaticArrayInit */
```

the array1 is initialized once at compile time

Array Examples (Cont.)

- Example: fig06_11.c

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

the array1 is initialized once at compile time

```
44 void automaticArrayInit( void )
45 {
46    /* initializes elements each time function is called */
47    int array2[ 3 ] = { 1, 2, 3 };
48    int i; /* counter */
49
50    printf( "\n\nValues on entering automaticArrayInit:\n" );
51
52    /* output contents of array2 */
53    for ( i = 0; i <= 2; i++ ) {
54       printf("array2[ %d ] = %d  ", i, array2[ i ] );
55    } /* end for */
56
57    printf( "\nValues on exiting automaticArrayInit:\n" );
58
59    /* modify and output contents of array2 */
60    for ( i = 0; i <= 2; i++ ) {
61       printf( "array2[ %d ] = %d  ", i, array2[ i ] += 5 );
62    } /* end for */
63 } /* end function automaticArrayInit */
```

the array2 is initialized twice at each function call

Passing Arrays to Functions

- To pass an array argument to a function, specify the name of the array without any brackets
- C automatically passes arrays to functions by reference
  - The name of the array evaluates to the address of the first element of the array
  - Because the starting address of the array is passed, the called function knows precisely where the array is stored

Passing Arrays to Functions (Cont.)

- Example: fig06_12.c

```
 6 int main( void )
 7 {
 8    char array[ 5 ]; /* define an array of size 5 */
 9
10    printf( "    array = %p\n&array[0] = %p\n   &array = %p\n",
11            array, &array[ 0 ], &array );
12
13    return 0; /* indicates successful termination */
14 } /* end main */
```

Passing Arrays to Functions (Cont.)

- Example: fig06_12.c

```
 6 int main( void )
 7 {
 8    char array[ 5 ]; /* define an array of size 5 */
 9
10    printf( "    array = %p\n&array[0] = %p\n   &array = %p\n",
11            array, &array[ 0 ], &array );
12
13    return 0; /* indicates successful termination */
14 } /* end main */
```

Passing Arrays to Functions (Cont.)

- Example: fig06_12.c

```
 6 int main( void )
 7 {
 8    char array[ 5 ]; /* define an array of size 5 */
 9
10    printf( "    array = %p\n&array[0] = %p\n   &array = %p\n",
11            array, &array[ 0 ], &array );
12
13    return 0; /* indicates successful termination */
14 } /* end main */
```

print the memory address of the array

Passing Arrays to Functions (Cont.)

- Example: fig06_12.c

```
 6 int main( void )
 7 {
 8    char array[ 5 ]; /* define an array of size 5 */
 9
10    printf( "    array = %p\n&array[0] = %p\n   &array = %p\n",
11            array, &array[ 0 ], &array );
12
13    return 0; /* indicates successful termination */
14 } /* end main */
```

print the memory address of the array

```
    array = 0x7fff5fbfe7d0
&array[0] = 0x7fff5fbfe7d0
   &array = 0x7fff5fbfe7d0
```

Passing Arrays to Functions (Cont.)

- For a function to receive an array through a function call, the function's parameter list must specify that an array will be received.
- For example, the function header for function modifyArray (that we called earlier in this section) might be written as

```
void modifyArray( int b[], int size )
```

- The size of the array is not required between the array brackets

Passing Arrays to Functions (Cont.)

- Example: fig06_13.c

```
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( void )
12 {
13    int a[ SIZE ] = { 0, 1, 2, 3, 4 }; /* initialize a */
14    int i; /* counter */
```

```
26    /* pass array a to modifyArray by reference */
27    modifyArray( a, SIZE );
28
29    printf( "The values of the modified array are:\n" );
```

```
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 ] );
```

Passing Arrays to Functions (Cont.)

- Example: fig06_13.c

```
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( void )
12 {
13    int a[ SIZE ] = { 0, 1, 2, 3, 4 }; /* initialize a */
14    int i; /* counter */
```

```
26    /* pass array a to modifyArray by reference */
27    modifyArray( a, SIZE );
28
29    printf( "The values of the modified array are:\n" );
```

```
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 ] );
```

Passing Arrays to Functions (Cont.)

- Example: fig06_13.c

```
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( void )
12 {
13    int a[ SIZE ] = { 0, 1, 2, 3, 4 }; /* initialize a */
14    int i; /* counter */
```

define a symbolic constant

```
26    /* pass array a to modifyArray by reference */
27    modifyArray( a, SIZE );
28
29    printf( "The values of the modified array are:\n" );
```

```
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 ] );
```

Passing Arrays to Functions (Cont.)

- Example: fig06_13.c

```
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( void )
12 {
13    int a[ SIZE ] = { 0, 1, 2, 3, 4 }; /* initialize a */
14    int i; /* counter */
```

define a symbolic constant

```
26    /* pass array a to modifyArray by reference */
27    modifyArray( a, SIZE );
28
29    printf( "The values of the modified array are:\n" );
```

```
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 ] );
```

Passing Arrays to Functions (Cont.)

- Example: fig06_13.c

```
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( void )
12 {
13    int a[ SIZE ] = { 0, 1, 2, 3, 4 }; /* initialize a */
14    int i; /* counter */
```

define a symbolic constant

equals to
int a[5] = {0, 1, 2, 3, 4}

```
26    /* pass array a to modifyArray by reference */
27    modifyArray( a, SIZE );
28
29    printf( "The values of the modified array are:\n" );
```

```
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 ] );
```

Passing Arrays to Functions (Cont.)

- Example: fig06_13.c

```
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( void )
12 {
13    int a[ SIZE ] = { 0, 1, 2, 3, 4 }; /* initialize a */
14    int i; /* counter */
```

define a symbolic constant

equals to
int a[5] = {0, 1, 2, 3, 4}

```
26    /* pass array a to modifyArray by reference */
27    modifyArray( a, SIZE );
28
29    printf( "The values of the modified array are:\n" );
```

```
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 ] );
```

Passing Arrays to Functions (Cont.)

- Example: fig06_13.c

```
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( void )
12 {
13    int a[ SIZE ] = { 0, 1, 2, 3, 4 }; /* initialize a */
14    int i; /* counter */
```

define a symbolic constant

equals to
int a[5] = {0, 1, 2, 3, 4}

```
26    /* pass array a to modifyArray by reference */
27    modifyArray( a, SIZE );
28
29    printf( "The values of the modified array are:\n" );
```

pass the address of a[] to the **modifyArray** function

```
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 ] );
```

Passing Arrays to Functions (Cont.)

- Example: fig06_13.c

```
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( void )
12 {
13    int a[ SIZE ] = { 0, 1, 2, 3, 4 }; /* initialize a */
14    int i; /* counter */
```

define a symbolic constant

equals to
int a[5] = {0, 1, 2, 3, 4}

```
26    /* pass array a to modifyArray by reference */
27    modifyArray( a, SIZE );
28
29    printf( "The values of the modified array are:\n" );
```

pass the address of a[] to the **modifyArray** function

```
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 ] );
```

Passing Arrays to Functions (Cont.)

- Example: fig06_13.c

```
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( void )
12 {
13    int a[ SIZE ] = { 0, 1, 2, 3, 4 }; /* initialize a */
14    int i; /* counter */
```

define a symbolic constant

equals to
int a[5] = {0, 1, 2, 3, 4}

```
26    /* pass array a to modifyArray by reference */
27    modifyArray( a, SIZE );
28
29    printf( "The values of the modified array are:\n" );
```

pass the address of a[] to the **modifyArray** function

```
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 ] );
```

pass the value of a[3] to the **modifyElement** function

Passing Arrays to Functions (Cont.)

- Example: fig06_13.c

```
49 void modifyArray( int b[], int size )
50 {
51     int j; /* counter */
52
53     /* multiply each array element by 2 */
54     for ( j = 0; j < size; j++ ) {
55         b[ j ] *= 2;
56     } /* end for */
57 } /* end function modifyArray */
```

```
61 void modifyElement( int e )
62 {
63     /* multiply parameter by 2 */
64     printf( "Value in modifyElement is %d\n", e *= 2 );
65 } /* end function modifyElement */
```

Passing Arrays to Functions (Cont.)

- Example: fig06_13.c

```
49 void modifyArray( int b[], int size )
50 {
51     int j; /* counter */
52
53     /* multiply each array element by 2 */
54     for ( j = 0; j < size; j++ ) {
55         b[ j ] *= 2;
56     } /* end for */
57 } /* end function modifyArray */
```

```
61 void modifyElement( int e )
62 {
63     /* multiply parameter by 2 */
64     printf( "Value in modifyElement is %d\n", e *= 2 );
65 } /* end function modifyElement */
```

Passing Arrays to Functions (Cont.)

- Example: fig06_13.c

```
49 void modifyArray( int b[], int size )
50 {
51     int j; /* counter */
52
53     /* multiply each array element by 2 */
54     for ( j = 0; j < size; j++ ) {
55         b[ j ] *= 2;
56     } /* end for */
57 } /* end function modifyArray */
```

receive the address of the array

```
61 void modifyElement( int e )
62 {
63     /* multiply parameter by 2 */
64     printf( "Value in modifyElement is %d\n", e *= 2 );
65 } /* end function modifyElement */
```

Passing Arrays to Functions (Cont.)

- Example: fig06_13.c

```
49 void modifyArray( int b[], int size )
50 {
51     int j; /* counter */
52
53     /* multiply each array element by 2 */
54     for ( j = 0; j < size; j++ ) {
55         b[ j ] *= 2;
56     } /* end for */
57 } /* end function modifyArray */
```

receive the address of the array

```
61 void modifyElement( int e )
62 {
63     /* multiply parameter by 2 */
64     printf( "Value in modifyElement is %d\n", e *= 2 );
65 } /* end function modifyElement */
```

Passing Arrays to Functions (Cont.)

- Example: fig06_13.c

```
49 void modifyArray( int b[], int size )
50 {
51     int j; /* counter */
52
53     /* multiply each array element by 2 */
54     for ( j = 0; j < size; j++ ) {
55         b[ j ] *= 2;
56     } /* end for */
57 } /* end function modifyArray */
```

receive the address of the array

```
61 void modifyElement( int e )
62 {
63     /* multiply parameter by 2 */
64     printf( "Value in modifyElement is %d\n", e *= 2 );
65 } /* end function modifyElement */
```

receive the value of the variable

Passing Arrays to Functions (Cont.)

- There may be situations in your programs in which a function should not be allowed to modify array elements.
- C provides the type qualifier **const** to prevent modification of array values in a function.

Passing Arrays to Functions (Cont.)

- Example: fig06_14.c

```
 5 void tryToModifyArray( const int b[] ); /* function prototype */
 6
 7 /* function main begins program execution */
 8 int main( void )
 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    return 0; /* indicates successful termination */
16 } /* end main */
```

Passing Arrays to Functions (Cont.)

- Example: fig06_14.c

```
5  void tryToModifyArray( const int b[] ); /* function prototype */
6
7  /* function main begins program execution */
8  int main( void )
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     return 0; /* indicates successful termination */
16 } /* end main */
```

Passing Arrays to Functions (Cont.)

- Example: fig06_14.c

```
5  void tryToModifyArray( const int b[] ); /* function prototype */
6
7  /* function main begins program execution */
8  int main( void )
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     return 0; /* indicates successful termination */
16 } /* end main */
```

declare the received array is a constant array

Passing Arrays to Functions (Cont.)

- Example: fig06_14.c

```
20 void tryToModifyArray( const int b[] )
21 {
22    b[ 0 ] /= 2; /* error */
23    b[ 1 ] /= 2; /* error */
24    b[ 2 ] /= 2; /* error */
25 } /* end function tryToModifyArray */
```

Passing Arrays to Functions (Cont.)

- Example: fig06_14.c

```
20 void tryToModifyArray( const int b[] )
21 {
22    b[ 0 ] /= 2; /* error */
23    b[ 1 ] /= 2; /* error */
24    b[ 2 ] /= 2; /* error */
25 } /* end function tryToModifyArray */
```

Passing Arrays to Functions (Cont.)

- Example: fig06_14.c

```
20 void tryToModifyArray( const int b[] )
21 {
22    b[ 0 ] /= 2; /* error */
23    b[ 1 ] /= 2; /* error */
24    b[ 2 ] /= 2; /* error */
25 } /* end function tryToModifyArray */
```

Passing Arrays to Functions (Cont.)

- Example: fig06_14.c

```
20 void tryToModifyArray( const int b[] )
21 {
22    b[ 0 ] /= 2; /* error */
23    b[ 1 ] /= 2; /* error */
24    b[ 2 ] /= 2; /* error */
25 } /* end function tryToModifyArray */
```

cause the compiler error:
l-value specifies a const object

Sorting Arrays

- Sorting data (i.e., placing the data into a particular order such as ascending or descending) is one of the most important computing applications.
- Here we discuss what is perhaps the simplest known sorting scheme.

Performance Tip 6.4

Often, the simplest algorithms perform poorly. Their virtue is that they are easy to write, test and debug. More complex algorithms are often needed to realize maximum performance.

Sorting Arrays (Cont.)

- Example: fig06_15.c

```
10    int a[ SIZE ] = { 6, 2, 10, 4, 8, 89, 12, 68, 45, 37 };
11    int pass; /* passes counter */
12    int i; /* comparisons counter */
13    int hold; /* temporary location used to swap array elements */
```

```
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       } /* end inner for */
37    } /* end outer for */
```

Sorting Arrays (Cont.)

- Example: fig06_15.c

```
10    int a[ SIZE ] = { 6, 2, 10, 4, 8, 89, 12, 68, 45, 37 };
11    int pass; /* passes counter */
12    int i; /* comparisons counter */
13    int hold; /* temporary location used to swap array elements */
```

```
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       } /* end inner for */
37    } /* end outer for */
```

Sorting Arrays (Cont.)

- Example: fig06_15.c

```
10    int a[ SIZE ] = { 6, 2, 10, 4, 8, 89, 12, 68, 45, 37 };
11    int pass; /* passes counter */
12    int i; /* comparisons counter */
13    int hold; /* temporary location used to swap array elements */
```

declare an array with some integers

```
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       } /* end inner for */
37    } /* end outer for */
```

Sorting Arrays (Cont.)

- Example: fig06_15.c

```
10    int a[ SIZE ] = { 6, 2, 10, 4, 8, 89, 12, 68, 45, 37 };
11    int pass; /* passes counter */
12    int i; /* comparisons counter */
13    int hold; /* temporary location used to swap array elements */
```

declare an array with some integers

```
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       } /* end inner for */
37    } /* end outer for */
```

Sorting Arrays (Cont.)

- Example: fig06_15.c

```
10    int a[ SIZE ] = { 6, 2, 10, 4, 8, 89, 12, 68, 45, 37 };
11    int pass; /* passes counter */
12    int i; /* comparisons counter */
13    int hold; /* temporary location used to swap array elements */
```

declare an array with some integers

```
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       } /* end inner for */
37    } /* end outer for */
```

Bubble sort algorithm for sorting the integers

Sorting Arrays (Cont.)

- Example: fig06_15.c

```
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       } /* end inner for */
37    } /* end outer for */
```

pass = 1

hold = 0

i

6	2	10	4	8	89	12	68	45	37

Sorting Arrays (Cont.)

- Example: fig06_15.c

```
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       } /* end inner for */
37    } /* end outer for */
```

pass = 1

hold = 0

i

6	2	10	4	8	89	12	68	45	37

Sorting Arrays (Cont.)

- Example: fig06_15.c

```
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       } /* end inner for */
37    } /* end outer for */
```

pass = 1

hold = 0

i

6	2	10	4	8	89	12	68	45	37

Sorting Arrays (Cont.)

- Example: fig06_15.c

```
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       } /* end inner for */
37    } /* end outer for */
```

pass = 1

hold = 0

i

6	2	10	4	8	89	12	68	45	37

Sorting Arrays (Cont.)

- Example: fig06_15.c

```
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       } /* end inner for */
37    } /* end outer for */
```

pass = 1

hold = 6

i

6	2	10	4	8	89	12	68	45	37

Sorting Arrays (Cont.)

- Example: fig06_15.c

```
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       } /* end inner for */
37    } /* end outer for */
```

pass = 1

hold = 6

i

6	2	10	4	8	89	12	68	45	37

Sorting Arrays (Cont.)

- Example: fig06_15.c

```
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       } /* end inner for */
37    } /* end outer for */
```

pass = 1

hold = 6

i

2	2	10	4	8	89	12	68	45	37

Sorting Arrays (Cont.)

- Example: fig06_15.c

```
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       } /* end inner for */
37    } /* end outer for */
```

pass = 1

hold = 6

i

2	2	10	4	8	89	12	68	45	37

Sorting Arrays (Cont.)

- Example: fig06_15.c

```
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       } /* end inner for */
37    } /* end outer for */
```

pass = 1

hold = 6

i

2	6	10	4	8	89	12	68	45	37

Sorting Arrays (Cont.)

- Example: fig06_15.c

```
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       } /* end inner for */
37    } /* end outer for */
```

pass = 1

hold = 6

i

2	6	10	4	8	89	12	68	45	37

Sorting Arrays (Cont.)

- Example: fig06_15.c

```
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       } /* end inner for */
37    } /* end outer for */
```

pass = 1

hold = 6

i

2	6	10	4	8	89	12	68	45	37

Sorting Arrays (Cont.)

- Example: fig06_15.c

```
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       } /* end inner for */
37    } /* end outer for */
```

pass = 1

hold = 6

i

2	6	10	4	8	89	12	68	45	37

Sorting Arrays (Cont.)

- Example: fig06_15.c

```
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       } /* end inner for */
37    } /* end outer for */
```

pass = 1

hold = 6

i

2	6	10	4	8	89	12	68	45	37

Sorting Arrays (Cont.)

- Example: fig06_15.c

```
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       } /* end inner for */
37    } /* end outer for */
```

pass = 1

hold = 6

i

2	6	10	4	8	89	12	68	45	37

Sorting Arrays (Cont.)

- Example: fig06_15.c

```
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       } /* end inner for */
37    } /* end outer for */
```

pass = 1

hold = 6

i

2	6	10	4	8	89	12	68	45	37

Sorting Arrays (Cont.)

- Example: fig06_15.c

```
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       } /* end inner for */
37    } /* end outer for */
```

pass = 1

hold = 6

i

2	6	10	4	8	89	12	68	45	37

Sorting Arrays (Cont.)

- Example: fig06_15.c

```
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       } /* end inner for */
37    } /* end outer for */
```

pass = 1

hold = 10

i

2	6	10	4	8	89	12	68	45	37

Sorting Arrays (Cont.)

- Example: fig06_15.c

```
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       } /* end inner for */
37    } /* end outer for */
```

pass = 1

hold = 10

i

2	6	10	4	8	89	12	68	45	37

Sorting Arrays (Cont.)

- Example: fig06_15.c

```
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       } /* end inner for */
37    } /* end outer for */
```

pass = 1

hold = 10

i

2	6	4	4	8	89	12	68	45	37

Sorting Arrays (Cont.)

- Example: fig06_15.c

```
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       } /* end inner for */
37    } /* end outer for */
```

pass = 1

hold = 10

i

2	6	4	4	8	89	12	68	45	37

Sorting Arrays (Cont.)

- Example: fig06_15.c

```
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       } /* end inner for */
37    } /* end outer for */
```

pass = 1

hold = 10

i

2	6	4	10	8	89	12	68	45	37

Sorting Arrays (Cont.)

- Example: fig06_15.c

```
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       } /* end inner for */
37    } /* end outer for */
```

pass = 1

hold = 10

i

Sorting Arrays (Cont.)

- For each pass, the largest value is guaranteed to sink to the bottom element of the array.

Sorting Arrays (Cont.)

- For each pass, the largest value is guaranteed to sink to the bottom element of the array.

pass = 1

Sorting Arrays (Cont.)

- For each pass, the largest value is guaranteed to sink to the bottom element of the array.

pass = 1

2 | 6 | 4 | 8 | 10 | 12 | 68 | 45 | 37 | 89

pass = 2

2 | 4 | 6 | 8 | 10 | 12 | 45 | 37 | 68 | 89

Sorting Arrays (Cont.)

- For each pass, the largest value is guaranteed to sink to the bottom element of the array.

pass = 1

pass = 2

pass = 3

Sorting Arrays (Cont.)

- For each pass, the largest value is guaranteed to sink to the bottom element of the array.

pass = 1

pass = 2

pass = 3

...

Sorting Arrays (Cont.)

- For each pass, the largest value is guaranteed to sink to the bottom element of the array.

pass = 1

pass = 2

pass = 3

...

pass = 10

Case Study: Computing Mean, Median and Mode Using Arrays

- Computers are commonly used for survey data analysis to compile and analyze the results of surveys and opinion polls.

Case Study: Computing Mean, Median and Mode Using Arrays (Cont.)

- Example: fig06_16.c

```
/* function prototypes */
void mean( const int answer[] );
void median( int answer[] );
void mode( int freq[], const int answer[] ) ;
void bubbleSort( int a[] );
void printArray( const int a[] );

/* function main begins program execution */
int main( void )
{
    int frequency[ 10 ] = { 0 }; /* initialize array frequency */

    /* initialize array response */
    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 };
```

Case Study: Computing Mean, Median and Mode Using Arrays (Cont.)

- Example: fig06_16.c

```
/* function prototypes */
void mean( const int answer[] );
void median( int answer[] );
void mode( int freq[], const int answer[] ) ;
void bubbleSort( int a[] );
void printArray( const int a[] );

/* function main begins program execution */
int main( void )
{
    int frequency[ 10 ] = { 0 }; /* initialize array frequency */

    /* initialize array response */
    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 };
```

Case Study: Computing Mean, Median and Mode Using Arrays (Cont.)

- Example: fig06_16.c

```
/* function prototypes */
void mean( const int answer[] );
void median( int answer[] );
void mode( int freq[], const int answer[] ) ;
void bubbleSort( int a[] );
void printArray( const int a[] );

/* function main begins program execution */
int main( void )
{
    int frequency[ 10 ] = { 0 }; /* initialize array frequency */

    /* initialize array response */
    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 };
```

declare an array with 99 responses of data

Case Study: Computing Mean, Median and Mode Using Arrays (Cont.)

- Example: fig06_16.c

```c
void mean( const int answer[] )
{
    int j; /* counter for totaling array elements */
    int total = 0; /* variable to hold sum of array elements */

    printf( "%s\n%s\n%s\n", "********", "  Mean", "********" );

    /* total response values */
    for ( j = 0; j < SIZE; j++ ) {
        total += answer[ j ];
    } /* end for */

    printf( "The mean is the average value of the data\n"
            "items. The mean is equal to the total of\n"
            "all the data items divided by the number\n"
            "of data items ( %d ). The mean value for\n"
            "this run is: %d / %d = %.4f\n\n",
            SIZE, total, SIZE, (  double  ) total / SIZE );
} /* end function mean */
```

Case Study: Computing Mean, Median and Mode Using Arrays (Cont.)

- Example: fig06_16.c

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

Case Study: Computing Mean, Median and Mode Using Arrays (Cont.)

- Example: fig06_16.c

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

Case Study: Computing Mean, Median and Mode Using Arrays (Cont.)

- Example: fig06_16.c

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

use a for loop to sum up all responses

Case Study: Computing Mean, Median and Mode Using Arrays (Cont.)

- Example: fig06_16.c

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

use a for loop to sum up all responses

total / SIZE to calculate mean

Case Study: Computing Mean, Median and Mode Using Arrays (Cont.)

- Example: fig06_16.c

```
********
  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
```

Case Study: Computing Mean, Median and Mode Using Arrays (Cont.)

- Example: fig06_16.c

```c
void median( int answer[] )
{
   printf( "\n%s\n%s\n%s\n%s",
           "********", " Median", "********",
           "The unsorted array of responses is" );

   printArray( answer ); /* output unsorted array */

   bubbleSort( answer ); /* sort array */

   printf( "\n\nThe sorted array is" );
   printArray( answer ); /* output sorted array */

   /* display median element */
   printf( "\n\nThe median is element %d of\n"
           "the sorted %d element array.\n"
           "For this run the median is %d\n\n",
           SIZE / 2, SIZE, answer[ SIZE / 2 ] );
} /* end function median */
```

Case Study: Computing Mean, Median and Mode Using Arrays (Cont.)

- Example: fig06_16.c

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

Case Study: Computing Mean, Median and Mode Using Arrays (Cont.)

- Example: fig06_16.c

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

sort the array first

Case Study: Computing Mean, Median and Mode Using Arrays (Cont.)

- Example: fig06_16.c

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

sort the array first

Case Study: Computing Mean, Median and Mode Using Arrays (Cont.)

- Example: fig06_16.c

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

sort the array first

the median is located at the position of (SIZE/2)

Case Study: Computing Mean, Median and Mode Using Arrays (Cont.)

- Example: fig06_16.c

```
********
 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 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 9

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

Case Study: Computing Mean, Median and Mode Using Arrays (Cont.)

- Example: fig06_16.c

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

Case Study: Computing Mean, Median and Mode Using Arrays (Cont.)

- Example: fig06_16.c

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

Case Study: Computing Mean, Median and Mode Using Arrays (Cont.)

- Example: fig06_16.c

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

initialize an array named freq to record the frequency

Case Study: Computing Mean, Median and Mode Using Arrays (Cont.)

- Example: fig06_16.c

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

initialize an array named freq to record the frequency

Case Study: Computing Mean, Median and Mode Using Arrays (Cont.)

- Example: fig06_16.c

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

initialize an array named freq to record the frequency

record the frequency

Case Study: Computing Mean, Median and Mode Using Arrays (Cont.)

- Example: fig06_16.c

```
109     for ( rating = 1; rating <= 9; rating++ ) {
110         printf( "%8d%11d          ", rating, freq[ rating ] );
111
112         /* keep track of mode value and largest frequency value */
113         if ( freq[ rating ] > largest ) {
114             largest = freq[ rating ];
115             modeValue = rating;
116         } /* end if */
117
118         /* output histogram bar representing frequency value */
119         for ( h = 1; h <= freq[ rating ]; h++ ) {
120             printf( "*" );
121         } /* end inner for */
122
123         printf( "\n" ); /* being new line of output */
124     } /* end outer for */
125
126     /* display the mode value */
127     printf( "The mode is the most frequent value.\n"
128             "For this run the mode is %d which occurred"
129             " %d times.\n", modeValue, largest );
```

Case Study: Computing Mean, Median and Mode Using Arrays (Cont.)

- Example: fig06_16.c

```
109     for ( rating = 1; rating <= 9; rating++ ) {
110         printf( "%8d%11d           ", rating, freq[ rating ] );
111
112         /* keep track of mode value and largest frequency value */
113         if ( freq[ rating ] > largest ) {
114             largest = freq[ rating ];
115             modeValue = rating;
116         } /* end if */
117
118         /* output histogram bar representing frequency value */
119         for ( h = 1; h <= freq[ rating ]; h++ ) {
120             printf( "*" );
121         } /* end inner for */
122
123         printf( "\n" ); /* being new line of output */
124     } /* end outer for */
125
126     /* display the mode value */
127     printf( "The mode is the most frequent value.\n"
128             "For this run the mode is %d which occurred"
129             " %d times.\n", modeValue, largest );
```

Case Study: Computing Mean, Median and Mode Using Arrays (Cont.)

- Example: fig06_16.c

```
109     for ( rating = 1; rating <= 9; rating++ ) {
110         printf( "%8d%11d           ", rating, freq[ rating ] );
111
112         /* keep track of mode value and largest frequency value */
113         if ( freq[ rating ] > largest ) {
114             largest = freq[ rating ];
115             modeValue = rating;
116         } /* end if */
117
118         /* output histogram bar representing frequency value */
119         for ( h = 1; h <= freq[ rating ]; h++ ) {
120             printf( "*" );
121         } /* end inner for */
122
123         printf( "\n" ); /* being new line of output */
124     } /* end outer for */
125
126     /* display the mode value */
127     printf( "The mode is the most frequent value.\n"
128             "For this run the mode is %d which occurred"
129             " %d times.\n", modeValue, largest );
```

choose the value with the largest frequency as the mode

Case Study: Computing Mean, Median and Mode Using Arrays (Cont.)

- Example: fig06_16.c

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

Searching Array

- It may be necessary to determine whether an array contains a value that matches a certain **key value**.
- The process of finding a particular element of an array is called **searching**.
- In this section we discuss two searching techniques—the simple **linear search** technique and the more efficient (but more complex) **binary search** technique.

Searching Array (Cont.)

- Example: fig06_18.c

```
 7 int linearSearch( const int array[], int key, int size );
 8
 9 /* function main begins program execution */
10 int main( void )
11 {
12    int a[ SIZE ]; /* create array a */
13    int x; /* counter for initializing elements 0-99 of array a */
14    int searchKey; /* value to locate in array 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 */
```

Searching Array (Cont.)

- Example: fig06_18.c

```
 7 int linearSearch( const int array[], int key, int size );
 8
 9 /* function main begins program execution */
10 int main( void )
11 {
12    int a[ SIZE ]; /* create array a */
13    int x; /* counter for initializing elements 0-99 of array a */
14    int searchKey; /* value to locate in array 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 */
```

Searching Array (Cont.)

- Example: fig06_18.c

```
 7 int linearSearch( const int array[], int key, int size );
 8
 9 /* function main begins program execution */
10 int main( void )
11 {
12    int a[ SIZE ]; /* create array a */
13    int x; /* counter for initializing elements 0-99 of array a */
14    int searchKey; /* value to locate in array 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 */
```

initialize an array

Searching Array (Cont.)

- Example: fig06_18.c

```
22    printf( "Enter integer search key:\n" );
23    scanf( "%d", &searchKey );
24
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 */
```

Searching Array (Cont.)

- Example: fig06_18.c

```
22    printf( "Enter integer search key:\n" );
23    scanf( "%d", &searchKey );
24
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 */
```

Searching Array (Cont.)

- Example: fig06_18.c

```
22    printf( "Enter integer search key:\n" );
23    scanf( "%d", &searchKey );
24
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 */
```

enter a key to search

Searching Array (Cont.)

- Example: fig06_18.c

```
22    printf( "Enter integer search key:\n" );
23    scanf( "%d", &searchKey );
24
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 */
```

enter a key to search

Searching Array (Cont.)

- Example: fig06_18.c

```
22    printf( "Enter integer search key:\n" );
23    scanf( "%d", &searchKey );
24
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 */
```

enter a key to search

invoke linearSearch()

Searching Array (Cont.)

- Example: fig06_18.c

```
42 int linearSearch( const int array[], int key, int size )
43 {
44     int n; /* counter */
45
46     /* loop through array */
47     for ( n = 0; n < size; ++n ) {
48
49         if ( array[ n ] == key ) {
50             return n; /* return location of key */
51         } /* end if */
52     } /* end for */
53
54     return -1; /* key not found */
55 } /* end function linearSearch */
```

Searching Array (Cont.)

- Example: fig06_18.c

```
42 int linearSearch( const int array[], int key, int size )
43 {
44     int n; /* counter */
45
46     /* loop through array */
47     for ( n = 0; n < size; ++n ) {
48
49         if ( array[ n ] == key ) {
50             return n; /* return location of key */
51         } /* end if */
52     } /* end for */
53
54     return -1; /* key not found */
55 } /* end function linearSearch */
```

Searching Array (Cont.)

- Example: fig06_18.c

```
42 int linearSearch( const int array[], int key, int size )
43 {
44    int n; /* counter */
45
46    /* loop through array */
47    for ( n = 0; n < size; ++n ) {
48
49       if ( array[ n ] == key ) {
50          return n; /* return location of key */
51       } /* end if */
52    } /* end for */
53
54    return -1; /* key not found */
55 } /* end function linearSearch */
```

the key part of linear search

Searching Array (Cont.)

- The linear searching method works well for small or unsorted arrays.
- However, for large arrays **linear searching** is **inefficient**.
- If the array is sorted, the high-speed binary search technique can be used.

Searching Array (Cont.)

- Example: fig06_19.c

```
 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( void )
13 {
14    int a[ SIZE ]; /* create array a */
15    int i; /* counter for initializing elements 0-14 of array a */
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 */
```

Searching Array (Cont.)

- Example: fig06_19.c

```
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( void )
13 {
14    int a[ SIZE ]; /* create array a */
15    int i; /* counter for initializing elements 0-14 of array a */
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 */
```

Searching Array (Cont.)

- Example: fig06_19.c

```
 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( void )
13 {
14    int a[ SIZE ]; /* create array a */
15    int i; /* counter for initializing elements 0-14 of array a */
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 */
```

declare three functions

Searching Array (Cont.)

- Example: fig06_19.c

```
 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( void )
13 {
14    int a[ SIZE ]; /* create array a */
15    int i; /* counter for initializing elements 0-14 of array a */
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 */
```

declare three functions

Searching Array (Cont.)

- Example: fig06_19.c

```
 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( void )
13 {
14    int a[ SIZE ]; /* create array a */
15    int i; /* counter for initializing elements 0-14 of array a */
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 */
```

declare three functions

initialize an array

Searching Array (Cont.)

- Example: fig06_19.c

```
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 */
```

Searching Array (Cont.)

- Example: fig06_19.c

```
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 */
```

Searching Array (Cont.)

- Example: fig06_19.c

```
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 */
```

print the indexes

Searching Array (Cont.)

- Example: fig06_19.c

```
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 */
```

print the indexes

Searching Array (Cont.)

- Example: fig06_19.c

```
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 */
```

print the indexes

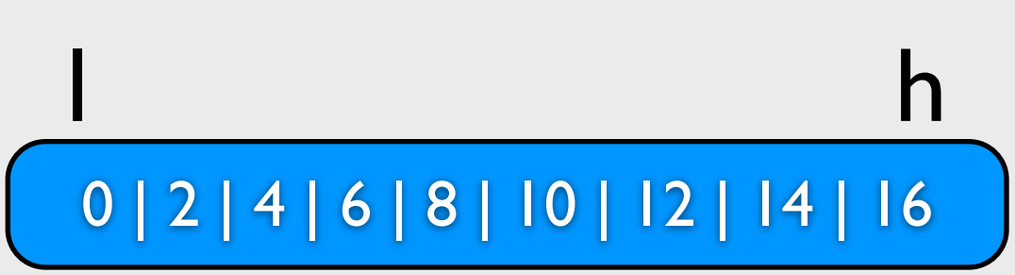
invoke binarySearch()

Searching Array (Cont.)

- Example: fig06_19.c

```
44 int binarySearch( const int b[], int searchKey, int low, int high )
45 {
46     int middle; /* variable to hold middle element of array */
47
48     /* loop until low subscript is greater than high subscript */
49     while ( low <= high ) {
50
51         /* determine middle element of subarray being searched */
52         middle = ( low + high ) / 2;
53
54         /* display subarray used in this loop iteration */
55         printRow( b, low, middle, high );
56
57         /* if searchKey matched middle element, return middle */
58         if ( searchKey == b[ middle ] ) {
59             return middle;
60         } /* end if */
61
62         /* if searchKey less than middle element, set new high */
63         else if ( searchKey < b[ middle ] ) {
64             high = middle - 1; /* search low end of array */
65         } /* end else if */
66
67         /* if searchKey greater than middle element, set new low */
68         else {
69             low = middle + 1; /* search high end of array */
70         } /* end else */
71
72     } /* end while */
73
74     return -1; /* searchKey not found */
75 } /* end function binarySearch */
```

searchKey = 4

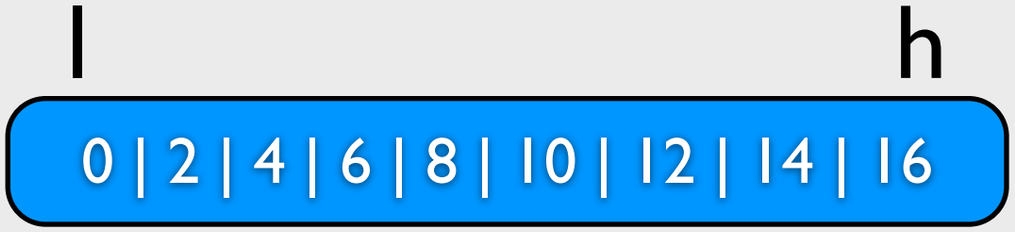


Searching Array (Cont.)

- Example: fig06_19.c

```
44 int binarySearch( const int b[], int searchKey, int low, int high )
45 {
46    int middle; /* variable to hold middle element of array */
47
48    /* loop until low subscript is greater than high subscript */
49    while ( low <= high ) {
50
51       /* determine middle element of subarray being searched */
52       middle = ( low + high ) / 2;
53
54       /* display subarray used in this loop iteration */
55       printRow( b, low, middle, high );
56
57       /* if searchKey matched middle element, return middle */
58       if ( searchKey == b[ middle ] ) {
59          return middle;
60       } /* end if */
61
62       /* if searchKey less than middle element, set new high */
63       else if ( searchKey < b[ middle ] ) {
64          high = middle - 1; /* search low end of array */
65       } /* end else if */
66
67       /* if searchKey greater than middle element, set new low */
68       else {
69          low = middle + 1; /* search high end of array */
70       } /* end else */
71
72    } /* end while */
73
74    return -1; /* searchKey not found */
75 } /* end function binarySearch */
```

searchKey = 4

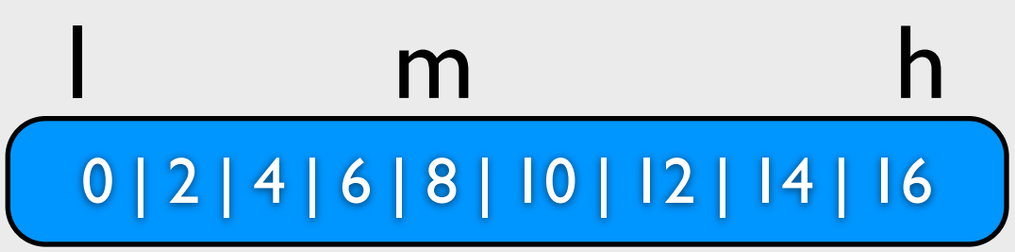


Searching Array (Cont.)

- Example: fig06_19.c

```
44 int binarySearch( const int b[], int searchKey, int low, int high )
45 {
46    int middle; /* variable to hold middle element of array */
47
48    /* loop until low subscript is greater than high subscript */
49    while ( low <= high ) {
50
51       /* determine middle element of subarray being searched */
52       middle = ( low + high ) / 2;
53
54       /* display subarray used in this loop iteration */
55       printRow( b, low, middle, high );
56
57       /* if searchKey matched middle element, return middle */
58       if ( searchKey == b[ middle ] ) {
59          return middle;
60       } /* end if */
61
62       /* if searchKey less than middle element, set new high */
63       else if ( searchKey < b[ middle ] ) {
64          high = middle - 1; /* search low end of array */
65       } /* end else if */
66
67       /* if searchKey greater than middle element, set new low */
68       else {
69          low = middle + 1; /* search high end of array */
70       } /* end else */
71
72    } /* end while */
73
74    return -1; /* searchKey not found */
75 } /* end function binarySearch */
```

searchKey = 4

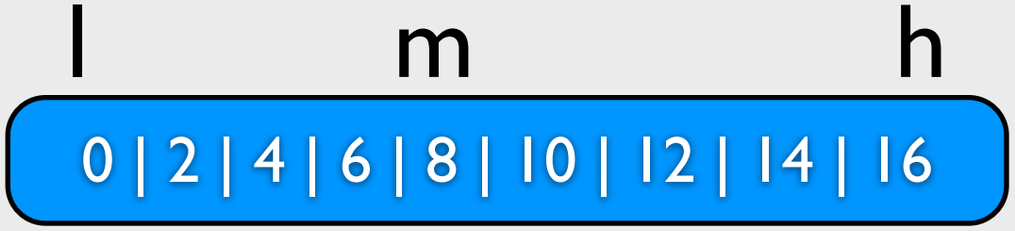


Searching Array (Cont.)

- Example: fig06_19.c

```
44 int binarySearch( const int b[], int searchKey, int low, int high )
45 {
46    int middle; /* variable to hold middle element of array */
47
48    /* loop until low subscript is greater than high subscript */
49    while ( low <= high ) {
50
51       /* determine middle element of subarray being searched */
52       middle = ( low + high ) / 2;
53
54       /* display subarray used in this loop iteration */
55       printRow( b, low, middle, high );
56
57       /* if searchKey matched middle element, return middle */
58       if ( searchKey == b[ middle ] ) {
59          return middle;
60       } /* end if */
61
62       /* if searchKey less than middle element, set new high */
63       else if ( searchKey < b[ middle ] ) {
64          high = middle - 1; /* search low end of array */
65       } /* end else if */
66
67       /* if searchKey greater than middle element, set new low */
68       else {
69          low = middle + 1; /* search high end of array */
70       } /* end else */
71
72    } /* end while */
73
74    return -1; /* searchKey not found */
75 } /* end function binarySearch */
```

searchKey = 4

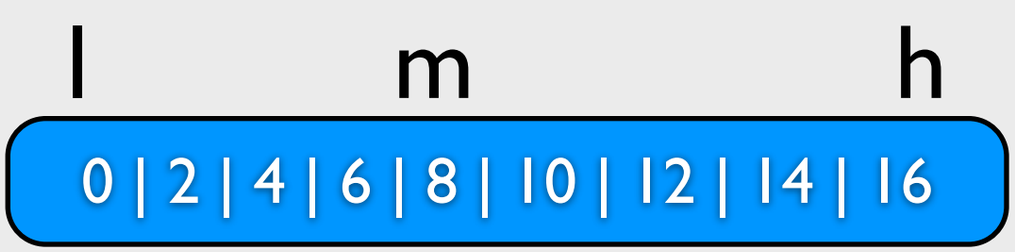


Searching Array (Cont.)

- Example: fig06_19.c

```
44 int binarySearch( const int b[], int searchKey, int low, int high )
45 {
46     int middle; /* variable to hold middle element of array */
47
48     /* loop until low subscript is greater than high subscript */
49     while ( low <= high ) {
50
51         /* determine middle element of subarray being searched */
52         middle = ( low + high ) / 2;
53
54         /* display subarray used in this loop iteration */
55         printRow( b, low, middle, high );
56
57         /* if searchKey matched middle element, return middle */
58         if ( searchKey == b[ middle ] ) {
59             return middle;
60         } /* end if */
61
62         /* if searchKey less than middle element, set new high */
63         else if ( searchKey < b[ middle ] ) {
64             high = middle - 1; /* search low end of array */
65         } /* end else if */
66
67         /* if searchKey greater than middle element, set new low */
68         else {
69             low = middle + 1; /* search high end of array */
70         } /* end else */
71
72     } /* end while */
73
74     return -1; /* searchKey not found */
75 } /* end function binarySearch */
```

searchKey = 4

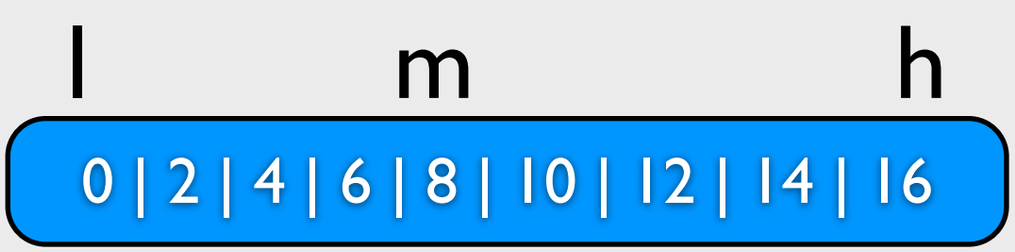


Searching Array (Cont.)

- Example: fig06_19.c

```
44 int binarySearch( const int b[], int searchKey, int low, int high )
45 {
46    int middle; /* variable to hold middle element of array */
47
48    /* loop until low subscript is greater than high subscript */
49    while ( low <= high ) {
50
51       /* determine middle element of subarray being searched */
52       middle = ( low + high ) / 2;
53
54       /* display subarray used in this loop iteration */
55       printRow( b, low, middle, high );
56
57       /* if searchKey matched middle element, return middle */
58       if ( searchKey == b[ middle ] ) {
59          return middle;
60       } /* end if */
61
62       /* if searchKey less than middle element, set new high */
63       else if ( searchKey < b[ middle ] ) {
64          high = middle - 1; /* search low end of array */
65       } /* end else if */
66
67       /* if searchKey greater than middle element, set new low */
68       else {
69          low = middle + 1; /* search high end of array */
70       } /* end else */
71
72    } /* end while */
73
74    return -1; /* searchKey not found */
75 } /* end function binarySearch */
```

searchKey = 4



Searching Array (Cont.)

- Example: fig06_19.c

```
44 int binarySearch( const int b[], int searchKey, int low, int high )
45 {
46    int middle; /* variable to hold middle element of array */
47
48    /* loop until low subscript is greater than high subscript */
49    while ( low <= high ) {
50
51       /* determine middle element of subarray being searched */
52       middle = ( low + high ) / 2;
53
54       /* display subarray used in this loop iteration */
55       printRow( b, low, middle, high );
56
57       /* if searchKey matched middle element, return middle */
58       if ( searchKey == b[ middle ] ) {
59          return middle;
60       } /* end if */
61
62       /* if searchKey less than middle element, set new high */
63       else if ( searchKey < b[ middle ] ) {
64          high = middle - 1; /* search low end of array */
65       } /* end else if */
66
67       /* if searchKey greater than middle element, set new low */
68       else {
69          low = middle + 1; /* search high end of array */
70       } /* end else */
71
72    } /* end while */
73
74    return -1; /* searchKey not found */
75 } /* end function binarySearch */
```

searchKey = 4

l h m

0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16

Searching Array (Cont.)

- Example: fig06_19.c

```
44 int binarySearch( const int b[], int searchKey, int low, int high )
45 {
46    int middle; /* variable to hold middle element of array */
47
48    /* loop until low subscript is greater than high subscript */
49    while ( low <= high ) {
50
51       /* determine middle element of subarray being searched */
52       middle = ( low + high ) / 2;
53
54       /* display subarray used in this loop iteration */
55       printRow( b, low, middle, high );
56
57       /* if searchKey matched middle element, return middle */
58       if ( searchKey == b[ middle ] ) {
59          return middle;
60       } /* end if */
61
62       /* if searchKey less than middle element, set new high */
63       else if ( searchKey < b[ middle ] ) {
64          high = middle - 1; /* search low end of array */
65       } /* end else if */
66
67       /* if searchKey greater than middle element, set new low */
68       else {
69          low = middle + 1; /* search high end of array */
70       } /* end else */
71
72    } /* end while */
73
74    return -1; /* searchKey not found */
75 } /* end function binarySearch */
```

searchKey = 4

l h m

0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16

Searching Array (Cont.)

- Example: fig06_19.c

```
44 int binarySearch( const int b[], int searchKey, int low, int high )
45 {
46     int middle; /* variable to hold middle element of array */
47
48     /* loop until low subscript is greater than high subscript */
49     while ( low <= high ) {
50
51         /* determine middle element of subarray being searched */
52         middle = ( low + high ) / 2;
53
54         /* display subarray used in this loop iteration */
55         printRow( b, low, middle, high );
56
57         /* if searchKey matched middle element, return middle */
58         if ( searchKey == b[ middle ] ) {
59             return middle;
60         } /* end if */
61
62         /* if searchKey less than middle element, set new high */
63         else if ( searchKey < b[ middle ] ) {
64             high = middle - 1; /* search low end of array */
65         } /* end else if */
66
67         /* if searchKey greater than middle element, set new low */
68         else {
69             low = middle + 1; /* search high end of array */
70         } /* end else */
71
72     } /* end while */
73
74     return -1; /* searchKey not found */
75 } /* end function binarySearch */
```

searchKey = 4

l h m

0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16

Searching Array (Cont.)

- Example: fig06_19.c

```
44 int binarySearch( const int b[], int searchKey, int low, int high )
45 {
46    int middle; /* variable to hold middle element of array */
47
48    /* loop until low subscript is greater than high subscript */
49    while ( low <= high ) {
50
51       /* determine middle element of subarray being searched */
52       middle = ( low + high ) / 2;
53
54       /* display subarray used in this loop iteration */
55       printRow( b, low, middle, high );
56
57       /* if searchKey matched middle element, return middle */
58       if ( searchKey == b[ middle ] ) {
59          return middle;
60       } /* end if */
61
62       /* if searchKey less than middle element, set new high */
63       else if ( searchKey < b[ middle ] ) {
64          high = middle - 1; /* search low end of array */
65       } /* end else if */
66
67       /* if searchKey greater than middle element, set new low */
68       else {
69          low = middle + 1; /* search high end of array */
70       } /* end else */
71
72    } /* end while */
73
74    return -1; /* searchKey not found */
75 } /* end function binarySearch */
```

searchKey = 4

l h m

0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16

Searching Array (Cont.)

- Example: fig06_19.c

```
44 int binarySearch( const int b[], int searchKey, int low, int high )
45 {
46    int middle; /* variable to hold middle element of array */
47
48    /* loop until low subscript is greater than high subscript */
49    while ( low <= high ) {
50
51       /* determine middle element of subarray being searched */
52       middle = ( low + high ) / 2;
53
54       /* display subarray used in this loop iteration */
55       printRow( b, low, middle, high );
56
57       /* if searchKey matched middle element, return middle */
58       if ( searchKey == b[ middle ] ) {
59          return middle;
60       } /* end if */
61
62       /* if searchKey less than middle element, set new high */
63       else if ( searchKey < b[ middle ] ) {
64          high = middle - 1; /* search low end of array */
65       } /* end else if */
66
67       /* if searchKey greater than middle element, set new low */
68       else {
69          low = middle + 1; /* search high end of array */
70       } /* end else */
71
72    } /* end while */
73
74    return -1; /* searchKey not found */
75 } /* end function binarySearch */
```

searchKey = 4

l m h

0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16

Searching Array (Cont.)

- Example: fig06_19.c

```
44 int binarySearch( const int b[], int searchKey, int low, int high )
45 {
46    int middle; /* variable to hold middle element of array */
47
48    /* loop until low subscript is greater than high subscript */
49    while ( low <= high ) {
50
51       /* determine middle element of subarray being searched */
52       middle = ( low + high ) / 2;
53
54       /* display subarray used in this loop iteration */
55       printRow( b, low, middle, high );
56
57       /* if searchKey matched middle element, return middle */
58       if ( searchKey == b[ middle ] ) {
59          return middle;
60       } /* end if */
61
62       /* if searchKey less than middle element, set new high */
63       else if ( searchKey < b[ middle ] ) {
64          high = middle - 1; /* search low end of array */
65       } /* end else if */
66
67       /* if searchKey greater than middle element, set new low */
68       else {
69          low = middle + 1; /* search high end of array */
70       } /* end else */
71
72    } /* end while */
73
74    return -1; /* searchKey not found */
75 } /* end function binarySearch */
```

searchKey = 4

l m h

0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16

Searching Array (Cont.)

- Example: fig06_19.c

```
44 int binarySearch( const int b[], int searchKey, int low, int high )
45 {
46     int middle; /* variable to hold middle element of array */
47
48     /* loop until low subscript is greater than high subscript */
49     while ( low <= high ) {
50
51         /* determine middle element of subarray being searched */
52         middle = ( low + high ) / 2;
53
54         /* display subarray used in this loop iteration */
55         printRow( b, low, middle, high );
56
57         /* if searchKey matched middle element, return middle */
58         if ( searchKey == b[ middle ] ) {
59             return middle;
60         } /* end if */
61
62         /* if searchKey less than middle element, set new high */
63         else if ( searchKey < b[ middle ] ) {
64             high = middle - 1; /* search low end of array */
65         } /* end else if */
66
67         /* if searchKey greater than middle element, set new low */
68         else {
69             low = middle + 1; /* search high end of array */
70         } /* end else */
71
72     } /* end while */
73
74     return -1; /* searchKey not found */
75 } /* end function binarySearch */
```

searchKey = 4

l m h

0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16

Searching Array (Cont.)

- Example: fig06_19.c

```
44 int binarySearch( const int b[], int searchKey, int low, int high )
45 {
46    int middle; /* variable to hold middle element of array */
47
48    /* loop until low subscript is greater than high subscript */
49    while ( low <= high ) {
50
51       /* determine middle element of subarray being searched */
52       middle = ( low + high ) / 2;
53
54       /* display subarray used in this loop iteration */
55       printRow( b, low, middle, high );
56
57       /* if searchKey matched middle element, return middle */
58       if ( searchKey == b[ middle ] ) {
59          return middle;
60       } /* end if */
61
62       /* if searchKey less than middle element, set new high */
63       else if ( searchKey < b[ middle ] ) {
64          high = middle - 1; /* search low end of array */
65       } /* end else if */
66
67       /* if searchKey greater than middle element, set new low */
68       else {
69          low = middle + 1; /* search high end of array */
70       } /* end else */
71
72    } /* end while */
73
74    return -1; /* searchKey not found */
75 } /* end function binarySearch */
```

searchKey = 4

l m h

Searching Array (Cont.)

- Example: fig06_19.c

```
44 int binarySearch( const int b[], int searchKey, int low, int high )
45 {
46    int middle; /* variable to hold middle element of array */
47
48    /* loop until low subscript is greater than high subscript */
49    while ( low <= high ) {
50
51       /* determine middle element of subarray being searched */
52       middle = ( low + high ) / 2;
53
54       /* display subarray used in this loop iteration */
55       printRow( b, low, middle, high );
56
57       /* if searchKey matched middle element, return middle */
58       if ( searchKey == b[ middle ] ) {
59          return middle;
60       } /* end if */
61
62       /* if searchKey less than middle element, set new high */
63       else if ( searchKey < b[ middle ] ) {
64          high = middle - 1; /* search low end of array */
65       } /* end else if */
66
67       /* if searchKey greater than middle element, set new low */
68       else {
69          low = middle + 1; /* search high end of array */
70       } /* end else */
71
72    } /* end while */
73
74    return -1; /* searchKey not found */
75 } /* end function binarySearch */
```

searchKey = 4

l m h

0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16

Searching Array (Cont.)

- Example: fig06_19.c

```
44 int binarySearch( const int b[], int searchKey, int low, int high )
45 {
46     int middle; /* variable to hold middle element of array */
47
48     /* loop until low subscript is greater than high subscript */
49     while ( low <= high ) {
50
51         /* determine middle element of subarray being searched */
52         middle = ( low + high ) / 2;
53
54         /* display subarray used in this loop iteration */
55         printRow( b, low, middle, high );
56
57         /* if searchKey matched middle element, return middle */
58         if ( searchKey == b[ middle ] ) {
59             return middle;
60         } /* end if */
61
62         /* if searchKey less than middle element, set new high */
63         else if ( searchKey < b[ middle ] ) {
64             high = middle - 1; /* search low end of array */
65         } /* end else if */
66
67         /* if searchKey greater than middle element, set new low */
68         else {
69             low = middle + 1; /* search high end of array */
70         } /* end else */
71
72     } /* end while */
73
74     return -1; /* searchKey not found */
75 } /* end function binarySearch */
```

searchKey = 4

m l h

0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16

Searching Array (Cont.)

- Example: fig06_19.c

```
44 int binarySearch( const int b[], int searchKey, int low, int high )
45 {
46    int middle; /* variable to hold middle element of array */
47
48    /* loop until low subscript is greater than high subscript */
49    while ( low <= high ) {
50
51       /* determine middle element of subarray being searched */
52       middle = ( low + high ) / 2;
53
54       /* display subarray used in this loop iteration */
55       printRow( b, low, middle, high );
56
57       /* if searchKey matched middle element, return middle */
58       if ( searchKey == b[ middle ] ) {
59          return middle;
60       } /* end if */
61
62       /* if searchKey less than middle element, set new high */
63       else if ( searchKey < b[ middle ] ) {
64          high = middle - 1; /* search low end of array */
65       } /* end else if */
66
67       /* if searchKey greater than middle element, set new low */
68       else {
69          low = middle + 1; /* search high end of array */
70       } /* end else */
71
72    } /* end while */
73
74    return -1; /* searchKey not found */
75 } /* end function binarySearch */
```

searchKey = 4

m l h

0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16

Searching Array (Cont.)

- Example: fig06_19.c

```
44 int binarySearch( const int b[], int searchKey, int low, int high )
45 {
46     int middle; /* variable to hold middle element of array */
47
48     /* loop until low subscript is greater than high subscript */
49     while ( low <= high ) {
50
51         /* determine middle element of subarray being searched */
52         middle = ( low + high ) / 2;
53
54         /* display subarray used in this loop iteration */
55         printRow( b, low, middle, high );
56
57         /* if searchKey matched middle element, return middle */
58         if ( searchKey == b[ middle ] ) {
59             return middle;
60         } /* end if */
61
62         /* if searchKey less than middle element, set new high */
63         else if ( searchKey < b[ middle ] ) {
64             high = middle - 1; /* search low end of array */
65         } /* end else if */
66
67         /* if searchKey greater than middle element, set new low */
68         else {
69             low = middle + 1; /* search high end of array */
70         } /* end else */
71
72     } /* end while */
73
74     return -1; /* searchKey not found */
75 } /* end function binarySearch */
```

searchKey = 4

m l h

0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16

Searching Array (Cont.)

- Example: fig06_19.c

```
44 int binarySearch( const int b[], int searchKey, int low, int high )
45 {
46     int middle; /* variable to hold middle element of array */
47
48     /* loop until low subscript is greater than high subscript */
49     while ( low <= high ) {
50
51         /* determine middle element of subarray being searched */
52         middle = ( low + high ) / 2;
53
54         /* display subarray used in this loop iteration */
55         printRow( b, low, middle, high );
56
57         /* if searchKey matched middle element, return middle */
58         if ( searchKey == b[ middle ] ) {
59             return middle;
60         } /* end if */
61
62         /* if searchKey less than middle element, set new high */
63         else if ( searchKey < b[ middle ] ) {
64             high = middle - 1; /* search low end of array */
65         } /* end else if */
66
67         /* if searchKey greater than middle element, set new low */
68         else {
69             low = middle + 1; /* search high end of array */
70         } /* end else */
71
72     } /* end while */
73
74     return -1; /* searchKey not found */
75 } /* end function binarySearch */
```

searchKey = 4

m l h

0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16

Searching Array (Cont.)

- Example: fig06_19.c

```
44 int binarySearch( const int b[], int searchKey, int low, int high )
45 {
46    int middle; /* variable to hold middle element of array */
47
48    /* loop until low subscript is greater than high subscript */
49    while ( low <= high ) {
50
51       /* determine middle element of subarray being searched */
52       middle = ( low + high ) / 2;
53
54       /* display subarray used in this loop iteration */
55       printRow( b, low, middle, high );
56
57       /* if searchKey matched middle element, return middle */
58       if ( searchKey == b[ middle ] ) {
59          return middle;
60       } /* end if */
61
62       /* if searchKey less than middle element, set new high */
63       else if ( searchKey < b[ middle ] ) {
64          high = middle - 1; /* search low end of array */
65       } /* end else if */
66
67       /* if searchKey greater than middle element, set new low */
68       else {
69          low = middle + 1; /* search high end of array */
70       } /* end else */
71
72    } /* end while */
73
74    return -1; /* searchKey not found */
75 } /* end function binarySearch */
```

searchKey = 4

m

l h

0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16

Searching Array (Cont.)

- Example: fig06_19.c

```
44 int binarySearch( const int b[], int searchKey, int low, int high )
45 {
46    int middle; /* variable to hold middle element of array */
47
48    /* loop until low subscript is greater than high subscript */
49    while ( low <= high ) {
50
51       /* determine middle element of subarray being searched */
52       middle = ( low + high ) / 2;
53
54       /* display subarray used in this loop iteration */
55       printRow( b, low, middle, high );
56
57       /* if searchKey matched middle element, return middle */
58       if ( searchKey == b[ middle ] ) {
59          return middle;
60       } /* end if */
61
62       /* if searchKey less than middle element, set new high */
63       else if ( searchKey < b[ middle ] ) {
64          high = middle - 1; /* search low end of array */
65       } /* end else if */
66
67       /* if searchKey greater than middle element, set new low */
68       else {
69          low = middle + 1; /* search high end of array */
70       } /* end else */
71
72    } /* end while */
73
74    return -1; /* searchKey not found */
75 } /* end function binarySearch */
```

searchKey = 4

m

l h

0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16

Searching Array (Cont.)

- Example: fig06_19.c

```
44 int binarySearch( const int b[], int searchKey, int low, int high )
45 {
46    int middle; /* variable to hold middle element of array */
47
48    /* loop until low subscript is greater than high subscript */
49    while ( low <= high ) {
50
51       /* determine middle element of subarray being searched */
52       middle = ( low + high ) / 2;
53
54       /* display subarray used in this loop iteration */
55       printRow( b, low, middle, high );
56
57       /* if searchKey matched middle element, return middle */
58       if ( searchKey == b[ middle ] ) {
59          return middle;
60       } /* end if */
61
62       /* if searchKey less than middle element, set new high */
63       else if ( searchKey < b[ middle ] ) {
64          high = middle - 1; /* search low end of array */
65       } /* end else if */
66
67       /* if searchKey greater than middle element, set new low */
68       else {
69          low = middle + 1; /* search high end of array */
70       } /* end else */
71
72    } /* end while */
73
74    return -1; /* searchKey not found */
75 } /* end function binarySearch */
```

searchKey = 4

m

l h

0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16

Searching Array (Cont.)

- Example: fig06_19.c

Searching Array (Cont.)

- Example: fig06_19.c

```
Enter a number between 0 and 28: 4

Subscripts:
  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
  0   2*  4
      4*
```

Searching Array (Cont.)

- Example: fig06_19.c

```
Enter a number between 0 and 28: 4

Subscripts:
  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
  0   2*  4
          4*
```

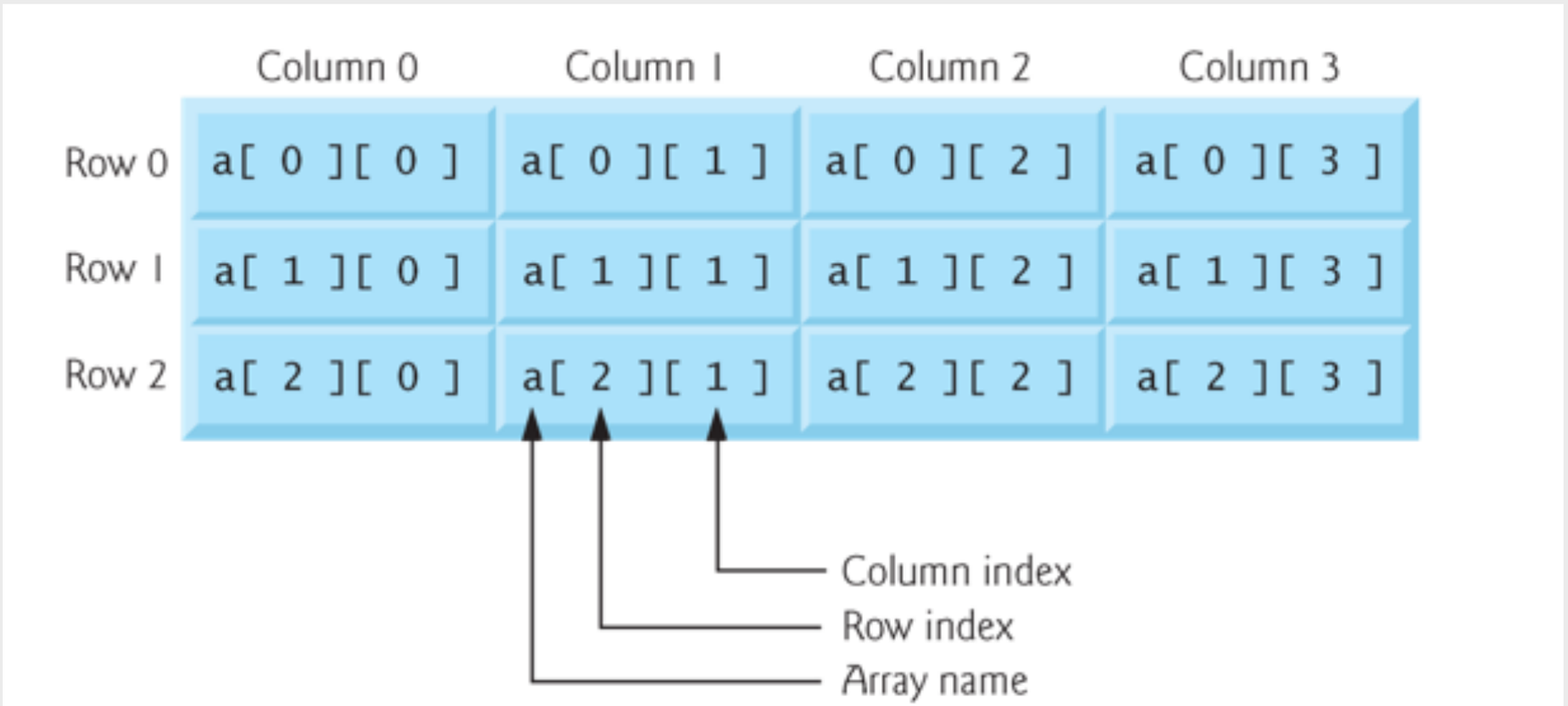
```
Enter a number between 0 and 28: 25

Subscripts:
  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*
```

Multiple-Subscripted Arrays

- Arrays in C can have **multiple subscripts**.
- A common use of multiple-subscripted arrays (also called **multidimensional arrays**) is to represent tables of values consisting of information arranged in rows and columns.

Multiple-Subscripted Arrays (Cont.)



Multiple-Subscripted Arrays (Cont.)

Common Programming Error 6.9

Referencing a double-subscripted array element as `a[ x, y ]` *instead of* `a[ x ][ y ]`. *C interprets* `a[ x, y ]` *as* `a[ y ]`, *and as such it does not cause a compilation error.*

Multiple-Subscripted Arrays (Cont.)

- A multiple-subscripted array can be initialized when it's defined, much like a single-subscripted array.
- For example, a double-subscripted array int **`b[2][2]`** could be defined and initialized with

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

Multiple-Subscripted Arrays (Cont.)

- If there are not enough initializers for a given row, the remaining elements of that row are initialized to 0. Thus,

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

- **`b[0][0]`** to 1, **`b[0][1]`** to 0, **`b[1][0]`** to 3 and **`b[1][1]`** to 4.

Multiple-Subscripted Arrays (Cont.)

- Example: fig06_21.c

```c
void printArray( const int a[][ 3 ] ); /* function prototype */

/* function main begins program execution */
int main( void )
{
   /* initialize array1, array2, array3 */
   int array1[ 2 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 } };
   int array2[ 2 ][ 3 ] = { 1, 2, 3, 4, 5 };
   int array3[ 2 ][ 3 ] = { { 1, 2 }, { 4 } };

   printf( "Values in array1 by row are:\n" );
   printArray( array1 );

   printf( "Values in array2 by row are:\n" );
   printArray( array2 );

   printf( "Values in array3 by row are:\n" );
   printArray( array3 );
   return 0; /* indicates successful termination */
} /* end main */
```

Multiple-Subscripted Arrays (Cont.)

- Example: fig06_21.c

```c
void printArray( const int a[][ 3 ] ); /* function prototype */

/* function main begins program execution */
int main( void )
{
   /* initialize array1, array2, array3 */
   int array1[ 2 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 } };
   int array2[ 2 ][ 3 ] = { 1, 2, 3, 4, 5 };
   int array3[ 2 ][ 3 ] = { { 1, 2 }, { 4 } };

   printf( "Values in array1 by row are:\n" );
   printArray( array1 );

   printf( "Values in array2 by row are:\n" );
   printArray( array2 );

   printf( "Values in array3 by row are:\n" );
   printArray( array3 );
   return 0; /* indicates successful termination */
} /* end main */
```

Multiple-Subscripted Arrays (Cont.)

- Example: fig06_21.c

```c
5  void printArray( const int a[][ 3 ] ); /* function prototype */
6
7  /* function main begins program execution */
8  int main( void )
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    return 0; /* indicates successful termination */
24 } /* end main */
```

The first subscript is not required, but all subsequent subscripts are required.

Multiple-Subscripted Arrays (Cont.)

- Example: fig06_21.c

```c
void printArray( const int a[][ 3 ] ); /* function prototype */

/* function main begins program execution */
int main( void )
{
   /* initialize array1, array2, array3 */
   int array1[ 2 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 } };
   int array2[ 2 ][ 3 ] = { 1, 2, 3, 4, 5 };
   int array3[ 2 ][ 3 ] = { { 1, 2 }, { 4 } };

   printf( "Values in array1 by row are:\n" );
   printArray( array1 );

   printf( "Values in array2 by row are:\n" );
   printArray( array2 );

   printf( "Values in array3 by row are:\n" );
   printArray( array3 );
   return 0; /* indicates successful termination */
} /* end main */
```

The first subscript is not required, but all subsequent subscripts are required.

Multiple-Subscripted Arrays (Cont.)

- Example: fig06_21.c

```c
5  void printArray( const int a[][ 3 ] ); /* function prototype */
6
7  /* function main begins program execution */
8  int main( void )
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    return 0; /* indicates successful termination */
24 } /* end main */
```

The first subscript is not required, but all subsequent subscripts are required.

declare three 2-dim arrays

Multiple-Subscripted Arrays (Cont.)

- Example: fig06_21.c

```c
void printArray( const int a[][ 3 ] )
{
    int i; /* row counter */
    int j; /* column counter */

    /* loop through rows */
    for ( i = 0; i <= 1; i++ ) {

        /* output column values */
        for ( j = 0; j <= 2; j++ ) {
            printf( "%d ", a[ i ][ j ] );
        } /* end inner for */

        printf( "\n" ); /* start new line of output */
    } /* end outer for */
} /* end function printArray */
```

Multiple-Subscripted Arrays (Cont.)

- Example: fig06_21.c

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

Multiple-Subscripted Arrays (Cont.)

- Example: fig06_21.c

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

use 2-level for loop to visit a 2-dim array

Multiple-Subscripted Arrays (Cont.)

- Example: fig06_21.c

```
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
```

Multiple-Subscripted Arrays (Cont.)

- Example: fig06_22.c

```
16    int student; /* 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
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; student++ ) {
35       printf( "The average grade for student %d is %.2f\n",
36          student, average( studentGrades[ student ], EXAMS ) );
37    } /* end for */
```

Multiple-Subscripted Arrays (Cont.)

- Example: fig06_22.c

```
16    int student; /* 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
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; student++ ) {
35       printf( "The average grade for student %d is %.2f\n",
36          student, average( studentGrades[ student ], EXAMS ) );
37    } /* end for */
```

Multiple-Subscripted Arrays (Cont.)

- Example: fig06_22.c

```
16    int student; /* 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
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; student++ ) {
35       printf( "The average grade for student %d is %.2f\n",
36          student, average( studentGrades[ student ], EXAMS ) );
37    } /* end for */
```

each row represents a student and each column represents a grade on one of the four exams

Multiple-Subscripted Arrays (Cont.)

- Example: fig06_22.c

```
16    int student; /* 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
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; student++ ) {
35       printf( "The average grade for student %d is %.2f\n",
36          student, average( studentGrades[ student ], EXAMS ) );
37    } /* end for */
```

each row represents a student and each column represents a grade on one of the four exams

Multiple-Subscripted Arrays (Cont.)

- Example: fig06_22.c

```
16    int student; /* 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
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; student++ ) {
35       printf( "The average grade for student %d is %.2f\n",
36          student, average( studentGrades[ student ], EXAMS ) );
37    } /* end for */
```

each row represents a student and each column represents a grade on one of the four exams

print the minimum and maximum grades

Multiple-Subscripted Arrays (Cont.)

- Example: fig06_22.c

```
16    int student; /* 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
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; student++ ) {
35       printf( "The average grade for student %d is %.2f\n",
36          student, average( studentGrades[ student ], EXAMS ) );
37    } /* end for */
```

each row represents a student and each column represents a grade on one of the four exams

print the minimum and maximum grades

Multiple-Subscripted Arrays (Cont.)

- Example: fig06_22.c

```
16    int student; /* 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
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; student++ ) {
35       printf( "The average grade for student %d is %.2f\n",
36          student, average( studentGrades[ student ], EXAMS ) );
37    } /* end for */
```

each row represents a student and each column represents a grade on one of the four exams

print the minimum and maximum grades

calculate the average scores of each student

Multiple-Subscripted Arrays (Cont.)

- Example: fig06_22.c

```
43 int minimum( const int grades[][ EXAMS ], int pupils, int tests )
44 {
45    int i; /* student counter */
46    int j; /* exam counter */
47    int lowGrade = 100; /* initialize to highest possible grade */
48
49    /* loop through rows of grades */
50    for ( i = 0; i < pupils; i++ ) {
51
52       /* loop through columns of grades */
53       for ( j = 0; j < tests; j++ ) {
54
55          if ( grades[ i ][ j ] < lowGrade ) {
56             lowGrade = grades[ i ][ j ];
57          } /* end if */
58       } /* end inner for */
59    } /* end outer for */
60
61    return lowGrade; /* return minimum grade */.
62 } /* end function minimum */
```

Multiple-Subscripted Arrays (Cont.)

- Example: fig06_22.c

```
43 int minimum( const int grades[][ EXAMS ], int pupils, int tests )
44 {
45    int i; /* student counter */
46    int j; /* exam counter */
47    int lowGrade = 100; /* initialize to highest possible grade */
48
49    /* loop through rows of grades */
50    for ( i = 0; i < pupils; i++ ) {
51
52       /* loop through columns of grades */
53       for ( j = 0; j < tests; j++ ) {
54
55          if ( grades[ i ][ j ] < lowGrade ) {
56             lowGrade = grades[ i ][ j ];
57          } /* end if */
58       } /* end inner for */
59    } /* end outer for */
60
61    return lowGrade; /* return minimum grade */.
62 } /* end function minimum */
```

Multiple-Subscripted Arrays (Cont.)

- Example: fig06_22.c

```
43 int minimum( const int grades[][ EXAMS ], int pupils, int tests )
44 {
45    int i; /* student counter */
46    int j; /* exam counter */
47    int lowGrade = 100; /* initialize to highest possible grade */
48
49    /* loop through rows of grades */
50    for ( i = 0; i < pupils; i++ ) {
51
52       /* loop through columns of grades */
53       for ( j = 0; j < tests; j++ ) {
54
55          if ( grades[ i ][ j ] < lowGrade ) {
56             lowGrade = grades[ i ][ j ];
57          } /* end if */
58       } /* end inner for */
59    } /* end outer for */
60
61    return lowGrade; /* return minimum grade */.
62 } /* end function minimum */
```

find out the lowest grade

Multiple-Subscripted Arrays (Cont.)

- Example: fig06_22.c

```
65 int maximum( const int grades[][ EXAMS ], int pupils, int tests )
66 {
67    int i; /* student counter */
68    int j; /* exam counter */
69    int highGrade = 0; /* initialize to lowest possible grade */
70
71    /* loop through rows of grades */
72    for ( i = 0; i < pupils; i++ ) {
73
74       /* loop through columns of grades */
75       for ( j = 0; j < tests; j++ ) {
76
77          if ( grades[ i ][ j ] > highGrade ) {
78             highGrade = grades[ i ][ j ];
79          } /* end if */
80       } /* end inner for */
81    } /* end outer for */
82
83    return highGrade; /* return maximum grade */
84 } /* end function maximum */
```

Multiple-Subscripted Arrays (Cont.)

- Example: fig06_22.c

```
65 int maximum( const int grades[][ EXAMS ], int pupils, int tests )
66 {
67    int i; /* student counter */
68    int j; /* exam counter */
69    int highGrade = 0; /* initialize to lowest possible grade */
70
71    /* loop through rows of grades */
72    for ( i = 0; i < pupils; i++ ) {
73
74       /* loop through columns of grades */
75       for ( j = 0; j < tests; j++ ) {
76
77          if ( grades[ i ][ j ] > highGrade ) {
78             highGrade = grades[ i ][ j ];
79          } /* end if */
80       } /* end inner for */
81    } /* end outer for */
82
83    return highGrade; /* return maximum grade */
84 } /* end function maximum */
```

Multiple-Subscripted Arrays (Cont.)

- Example: fig06_22.c

```
65 int maximum( const int grades[][ EXAMS ], int pupils, int tests )
66 {
67    int i; /* student counter */
68    int j; /* exam counter */
69    int highGrade = 0; /* initialize to lowest possible grade */
70
71    /* loop through rows of grades */
72    for ( i = 0; i < pupils; i++ ) {
73
74       /* loop through columns of grades */
75       for ( j = 0; j < tests; j++ ) {
76
77          if ( grades[ i ][ j ] > highGrade ) {
78             highGrade = grades[ i ][ j ];
79          } /* end if */
80       } /* end inner for */
81    } /* end outer for */
82
83    return highGrade; /* return maximum grade */
84 } /* end function maximum */
```

find out the highest grade

Multiple-Subscripted Arrays (Cont.)

- Example: fig06_22.c

```
87 double average( const int setOfGrades[], int tests )
88 {
89    int i; /* exam counter */
90    int total = 0; /* sum of test grades */
91
92    /* total all grades for one student */
93    for ( i = 0; i < tests; i++ ) {
94       total += setOfGrades[ i ];
95    } /* end for */
96
97    return ( double ) total / tests; /* average */
98 } /* end function average */
```

Multiple-Subscripted Arrays (Cont.)

- Example: fig06_22.c

```
87 double average( const int setOfGrades[], int tests )
88 {
89    int i; /* exam counter */
90    int total = 0; /* sum of test grades */
91
92    /* total all grades for one student */
93    for ( i = 0; i < tests; i++ ) {
94       total += setOfGrades[ i ];
95    } /* end for */
96
97    return ( double ) total / tests; /* average */
98 } /* end function average */
```

Multiple-Subscripted Arrays (Cont.)

- Example: fig06_22.c

```
87 double average( const int setOfGrades[], int tests )
88 {
89    int i; /* exam counter */
90    int total = 0; /* sum of test grades */
91
92    /* total all grades for one student */
93    for ( i = 0; i < tests; i++ ) {
94       total += setOfGrades[ i ];
95    } /* end for */
96
97    return ( double ) total / tests; /* average */
98 } /* end function average */
```

sum up all grades

Multiple-Subscripted Arrays (Cont.)

- Example: fig06_22.c

```
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
```

Vim Tips

- A minimalist Vim plugin manager
  - vim-plug
- Useful links
  - Vim Awesome
  - 優秀 Vim 外掛幫你打造完美 IDE