Outline

fig06_03.c

(1 of 2)

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

for loop initializes each array element separately

for loop outputs all array elements

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

Outline

fig06_03.c

(2 of 2)

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

initializer list initializes all array elements simultaneously

Outline

fig06_04.c

(1 of 2)

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

Outline

fig06_04.c

(2 of 2)

Common Programming Error 6.2

Forgetting to initialize the elements of an array whose elements should be initialized.

Common Programming Error 6.3

Providing more initializers in an array initializer list than there are elements in the array is a syntax error.

Outline

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

`#define` directive tells compiler to replace all instances of the word **`SIZE`** with **`10`**

`fig06_05.c`

(1 of 2)

`SIZE` is replaced with **`10`** by the compiler, so array **`s`** has 10 elements

`for` loop initializes each array element separately

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

Outline

fig06_05.c

(2 of 2)

Software Engineering Observation 6.1

Defining the size of each array as a symbolic constant makes programs more scalable.

Good Programming Practice 6.1

Use only uppercase letters for symbolic constant names. This makes these constants stand out in a program and reminds you that symbolic constants are not variables.

Good Programming Practice 6.2

In multiword symbolic constant names, use underscores to separate the words for readability.

Outline

fig06_06.c

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

initializer list initializes all array elements simultaneously

for loop adds each element of the array to variable **total**

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

Outline

```
1  /* Fig. 6.7: fig06_07.c
2     Student poll program */
3  #include <stdio.h>
4  #define RESPONSE_SIZE 40 /* define array sizes */
5  #define FREQUENCY_SIZE 11
6
7  /* function main begins program execution */
8  int main( void )
9  {
10    int answer; /* counter to loop through 40 responses */
11    int rating; /* counter to loop through frequencies 1-10 */
12
13    /* initialize frequency counters to 0 */
14    int frequency[ FREQUENCY_SIZE ] = { 0 };
15
16    /* place the survey responses in the responses array */
17    int responses[ RESPONSE_SIZE ] = { 1, 2, 6, 4, 8, 5, 9, 7, 8, 10,
18         1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6,
19         5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };
20
21    /* for each answer, select value of an element of array responses
22         and use that value as subscript in array frequency to
23         determine element to increment */
24    for ( answer = 0; answer < RESPONSE_SIZE; answer++ ) {
25       ++frequency[ responses [ answer ] ];
26    } /* end for */
27
```

`#define` directives create symbolic constants

`frequency` array is defined with 11 elements

`responses` array is defined with 40 elements and its elements are initialized

subscript of **`frequency`** array is given by value in **`responses`** array

`fig06_07.c`

(1 of 2)

```
28    /* display results */
29    printf( "%s%17s\n", "Rating", "Frequency" );
30
31    /* output the frequencies in a tabular format */
32    for ( rating = 1; rating < FREQUENCY_SIZE; rating++ ) {
33       printf( "%6d%17d\n", rating, frequency[ rating ] );
34    } /* end for */
35
36    return 0; /* indicates successful termination */
37
38 } /* end main */
```

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

Outline

fig06_07.c

(2 of 2)

Common Programming Error 6.6

Referring to an element outside the array bounds.

Error-Prevention Tip 6.1

When looping through an array, the array subscript should never go below 0 and should always be less than the total number of elements in the array (size – 1). Make sure the loop-terminating condition prevents accessing elements outside this range.

Outline

fig06_08.c

(1 of 2)

```
1  /* Fig. 6.8: fig06_08.c
2     Histogram printing program */
3  #include <stdio.h>
4  #define SIZE 10
5
6  /* function main begins program execution */
7  int main( void )
8  {
9     /* use initializer list to initialize array n */
10    int n[ SIZE ] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
11    int i; /* outer for counter for array elements */
12    int j; /* inner for counter counts *s in each histogram bar */
13
14    printf( "%s%13s%17s\n", "Element", "Value", "Histogram" );
15
16    /* for each element of array n, output a bar of the histogram */
17    for ( i = 0; i < SIZE; i++ ) {
18       printf( "%7d%13d        ", i, n[ i ]) ;
19
20       for ( j = 1; j <= n[ i ]; j++ ) { /* print one bar */
21          printf( "%c", '*' );
22        } /* end inner for */
23
24       printf( "\n" ); /* end a histogram bar */
25    } /* end outer for */
26
27    return 0; /* indicates successful termination */
28
29 } /* end main */
```

nested **for** loop prints n[**i**] asterisks on the **i**th line

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

Outline

fig06_08.c

(2 of 2)

Outline

fig06_09.c

(1 of 2)

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

for loop uses one array to track number of times each number is rolled instead of using 6 variables and a **switch** statement

Outline

```
22
23    printf( "%s%17s\n", "Face", "Frequency" );
24
25    /* output frequency elements 1-6 in tabular format */
26    for ( face = 1; face < SIZE; face++ ) {
27       printf( "%4d%17d\n", face, frequency[ face ] );
28    } /* end for */
29
30    return 0; /* indicates successful termination */
31
32 } /* end main */
```

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

fig06_09.c

(2 of 2)

6.4 Array Examples

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

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

    - **Null character `'\0'` terminates strings**
    - `string1` **actually has 6 elements**

It is equivalent to

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

  - **Can access individual characters**

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

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

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

    - **Reads characters until whitespace encountered**
    - **Be careful not to write past end of array, as it is possible to do so**

Outline

fig06_10.c

(1 of 2)

```
1  /* Fig. 6.10: fig06_10.c
2     Treating character arrays as strings */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main( void )
7  {
8     char string1[ 20 ]; /* reserves 20 characters */
9     char string2[] = "string literal"; /* reserves 15 characters */
10    int i; /* counter */
11
12    /* read string from user into array 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" );
27
28    return 0; /* indicates successful termination */
29
30 } /* end main */
```

string2 array is defined with one element for each character, so 15 elements including null character **/0**

for loop prints characters of **string1** array with spaces in between

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

Outline

fig06_10.c

(2 of 2)

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 is not created each time the function is called.

Outline

fig06_11.c

(1 of 4)

```
1  /* Fig. 6.11: fig06_11.c
2     Static arrays are initialized to zero */
3  #include <stdio.h>
4
5  void staticArrayInit( void );    /* function prototype */
6  void automaticArrayInit( void ); /* function prototype */
7
8  /* function main begins program execution */
9  int main( 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
19    return 0; /* indicates successful termination */
20
21 } /* end main */
22
```

Outline

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

`static` array is created only once, when **`staticArrayInit`** is first called

fig06_11.c

(2 of 4)

Outline

```
45
46 /* function to demonstrate an automatic local array */
47 void automaticArrayInit( void )
48 {
49    /* initializes elements each time function is called */
50    int array2[ 3 ] = { 1, 2, 3 };
51    int i; /* counter */
52
53    printf( "\n\nValues on entering automaticArrayInit:\n" );
54
55    /* output contents of array2 */
56    for ( i = 0; i <= 2; i++ ) {
57       printf("array2[ %d ] = %d  ", i, array2[ i ] );
58    } /* end for */
59
60    printf( "\nValues on exiting automaticArrayInit:\n" );
61
62    /* modify and output contents of array2 */
63    for ( i = 0; i <= 2; i++ ) {
64       printf( "array2[ %d ] = %d  ", i, array2[ i ] += 5 );
65    } /* end for */
66
67 } /* end function automaticArrayInit */
```

automatic array is recreated every time **`automaticArrayInit`** is called

`fig06_11.c`

(3 of 4)

```
First call to each function:

Values on entering staticArrayInit:
array1[ 0 ] = 0  array1[ 1 ] = 0  array1[ 2 ] = 0
Values on exiting staticArrayInit:
array1[ 0 ] = 5  array1[ 1 ] = 5  array1[ 2 ] = 5

Values on entering automaticArrayInit:
array2[ 0 ] = 1  array2[ 1 ] = 2  array2[ 2 ] = 3
Values on exiting automaticArrayInit:
array2[ 0 ] = 6  array2[ 1 ] = 7  array2[ 2 ] = 8

Second call to each function:

Values on entering staticArrayInit:
array1[ 0 ] = 5  array1[ 1 ] = 5  array1[ 2 ] = 5
Values on exiting staticArrayInit:
array1[ 0 ] = 10  array1[ 1 ] = 10  array1[ 2 ] = 10

Values on entering automaticArrayInit:
array2[ 0 ] = 1  array2[ 1 ] = 2  array2[ 2 ] = 3
Values on exiting automaticArrayInit:
array2[ 0 ] = 6  array2[ 1 ] = 7  array2[ 2 ] = 8
```

Outline

fig06_11.c

(4 of 4)

6.5 Passing Arrays to Functions

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

```
int myArray[ 24 ];
myFunction( myArray, 24 );
```

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

6.5 Passing Arrays to Functions

- **Function prototype**

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

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

Performance Tip 6.3

Passing arrays by reference makes sense for performance reasons. If arrays were passed by value, a copy of each element would be passed. For large, frequently passed arrays, this would be time consuming and would consume considerable storage for the copies of the arrays.

Outline

fig06_12.c

```
1  /* Fig. 6.12: fig06_12.c
2     The name of an array is the same as &array[ 0 ] */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main( 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
15 } /* end main */
```

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

Outline

fig06_13.c

(1 of 3)

```
1  /* Fig. 6.13: fig06_13.c
2     Passing arrays and individual array elements to functions */
3  #include <stdio.h>
4  #define SIZE 5
5
6  /* function prototypes */
7  void modifyArray( int b[], int size );
8  void modifyElement( int e );
9
10 /* function main begins program execution */
11 int main( void )
12 {
13    int a[ SIZE ] = { 0, 1, 2, 3, 4 }; /* initialize a */
14    int i; /* counter */
15
16    printf( "Effects of passing entire array by reference:\n\nThe "
17            "values of the original array are:\n" );
18
19    /* output original array */
20    for ( i = 0; i < SIZE; i++ ) {
21       printf( "%3d", a[ i ] );
22    } /* end for */
23
24    printf( "\n" );
25
26    /* pass array a to modifyArray by reference */
27    modifyArray( a, SIZE );
28
29    printf( "The values of the modified array are:\n" );
30
```

Function prototype indicates function will take an array

Array **a** is passed to **modifyArray** by passing only its name

Outline

fig06_13.c

(2 of 3)

```
31    /* output modified array */
32    for ( i = 0; i < SIZE; i++ ) {
33       printf( "%3d", a[ i ] );
34    } /* end for */
35
36    /* output value of a[ 3 ] */
37    printf( "\n\n\nEffects of passing array element "
38            "by value:\n\nThe value of a[3] is %d\n", a[ 3 ] );
39
40    modifyElement( a[ 3 ] ); /* pass array element a[ 3 ] by value */
41
42    /* output value of a[ 3 ] */
43    printf( "The value of a[ 3 ] is %d\n", a[ 3 ] );
44
45    return 0; /* indicates successful termination */
46
47 } /* end main */
48
49 /* in function modifyArray, "b" points to the original array "a"
50    in memory */
51 void modifyArray( int b[], int size )
52 {
53    int j; /* counter */
54
55    /* multiply each array element by 2 */
56    for ( j = 0; j < size; j++ ) {
57       b[ j ] *= 2;
58    } /* end for */
59
60 } /* end function modifyArray */
```

Array element is passed to **modifyElement** by passing **a[3]**

```
61
62 /* in function modifyElement, "e" is a local copy of array element
63    a[ 3 ] passed from main */
64 void modifyElement( int e )
65 {
66    /* multiply parameter by 2 */
67    printf( "Value in modifyElement is %d\n", e *= 2 );
68 } /* end function modifyElement */
```

```
Effects of passing entire array by reference:

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


Effects of passing array element by value:

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

Outline

fig06_13.c

(3 of 3)

Outline

fig06_14.c

(1 of 2)

```
1  /* Fig. 6.14: fig06_14.c
2     Demonstrating the const type qualifier with arrays */
3  #include <stdio.h>
4
5  void tryToModifyArray( const int b[] ); /* function prototype */
6
7  /* function main begins program execution */
8  int main( 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
16    return 0; /* indicates successful termination */
17
18 } /* end main */
19
20 /* in function tryToModifyArray, array b is const, so it cannot be
21    used to modify the original array a in main. */
22 void tryToModifyArray( const int b[] )
23 {
24    b[ 0 ] /= 2;    /* error */
25    b[ 1 ] /= 2;    /* error */
26    b[ 2 ] /= 2;    /* error */
27 } /* end function tryToModifyArray */
```

const qualifier tells compiler that array cannot be changed

Any attempts to modify the array will result in errors

```
Compiling...
FIG06_14.C
fig06_14.c(24) : error C2166: l-value specifies const object
fig06_14.c(25) : error C2166: l-value specifies const object
fig06_14.c(26) : error C2166: l-value specifies const object
```

Outline

fig06_14.c

(2 of 2)

Software Engineering Observation 6.3

The `const` type qualifier can be applied to an array parameter in a function definition to prevent the original array from being modified in the function body. This is another example of the principle of least privilege. Functions should not be given the capability to modify an array unless it is absolutely necessary.

6.6 Sorting Arrays

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

Outline

fig06_15.c

(1 of 2)

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

Outline

```
29          /* compare adjacent elements and swap them if first
30          element is greater than second element */
31          if ( a[ i ] > a[ i + 1 ] ) {
32             hold = a[ i ];
33             a[ i ] = a[ i + 1 ];
34             a[ i + 1 ] = hold;
35          } /* end if */
36
37       } /* end inner for */
38
39    } /* end outer for */
40
41    printf( "\nData items in ascending order\n" );
42
43    /* output sorted array */
44    for ( i = 0; i < SIZE; i++ ) {
45       printf( "%4d", a[ i ] );
46    } /* end for */
47
48    printf( "\n" );
49
50    return 0; /* indicates successful termination */
51 }
```

If any two array elements are out of order, the function swaps them

fig06_15.c

(2 of 2)

```
Data items in original order
   2   6   4   8  10  12  89  68  45  37
Data items in ascending order
   2   4   6   8  10  12  37  45  68  89
```

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

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

Outline

fig06_16.c

(1 of 6)

```
1  /* Fig. 6.16: fig06_16.c
2     This program introduces the topic of survey data analysis.
3     It computes the mean, median and mode of the data */
4  #include <stdio.h>
5  #define SIZE 99
6
7  /* function prototypes */
8  void mean( const int answer[] );
9  void median( int answer[] );
10 void mode( int freq[], const int answer[] ) ;
11 void bubbleSort( int a[] );
12 void printArray( const int a[] );
13
14 /* function main begins program execution */
15 int main( void )
16 {
17    int frequency[ 10 ] = { 0 }; /* initialize array frequency */
18
19    /* initialize array response */
20    int response[ SIZE ] =
21       { 6, 7, 8, 9, 8, 7, 8, 9, 8, 9,
22         7, 8, 9, 5, 9, 8, 7, 8, 7, 8,
23         6, 7, 8, 9, 3, 9, 8, 7, 8, 7,
24         7, 8, 9, 8, 9, 8, 9, 7, 8, 9,
25         6, 7, 8, 7, 8, 7, 9, 8, 9, 2,
26         7, 8, 9, 8, 9, 8, 9, 7, 5, 3,
27         5, 6, 7, 2, 5, 3, 9, 4, 6, 4,
28         7, 8, 9, 6, 8, 7, 8, 9, 7, 8,
29         7, 4, 4, 2, 5, 3, 8, 7, 5, 6,
30         4, 5, 6, 1, 6, 5, 7, 8, 7 };
```

Outline

fig06_16.c

(2 of 6)

```
31
32    /* process responses */
33    mean( response );
34    median( response );
35    mode( frequency, response );
36
37    return 0; /* indicates successful termination */
38
39 } /* end main */
40
41 /* calculate average of all response values */
42 void mean( const int answer[] )
43 {
44    int j; /* counter for totaling array elements */
45    int total = 0; /* variable to hold sum of array elements */
46
47    printf( "%s\n%s\n%s\n", "********", "  Mean", "********" );
48
49    /* total response values */
50    for ( j = 0; j < SIZE; j++ ) {
51       total += answer[ j ];
52    } /* end for */
53
54    printf( "The mean is the average value of the data\n"
55            "items. The mean is equal to the total of\n"
56            "all the data items divided by the number\n"
57            "of data items ( %d ). The mean value for\n"
58            "this run is: %d / %d = %.4f\n\n",
59            SIZE, total, SIZE, (  double  ) total / SIZE );
60 } /* end function mean */
```

Outline

fig06_16.c

(3 of 6)

```
61
62 /* sort array and determine median element's value */
63 void median( int answer[] )
64 {
65    printf( "\n%s\n%s\n%s\n%s",
66            "********", " Median", "********",
67            "The unsorted array of responses is" );
68
69    printArray( answer ); /* output unsorted array */
70
71    bubbleSort( answer ); /* sort array */
72
73    printf( "\n\nThe sorted array is" );
74    printArray( answer ); /* output sorted array */
75
76    /* display median element */
77    printf( "\n\nThe median is element %d of\n"
78            "the sorted %d element array.\n"
79            "For this run the median is %d\n\n",
80            SIZE / 2, SIZE, answer[ SIZE / 2 ] );
81 } /* end function median */
82
83 /* determine most frequent response */
84 void mode( int freq[], const int answer[] )
85 {
86    int rating; /* counter for accessing elements 1-9 of array freq */
87    int j; /* counter for summarizing elements 0-98 of array answer */
88    int h; /* counter for diplaying histograms of elements in array freq */
89    int largest = 0; /* represents largest frequency */
90    int modeValue = 0; /* represents most frequent response */
```

Once the array is sorted, the median will be the value of the middle element

```
91
92    printf( "\n%s\n%s\n%s\n",
93            "********", "  Mode", "********" );
94
95    /* initialize frequencies to 0 */
96    for ( rating = 1; rating <= 9; rating++ ) {
97       freq[ rating ] = 0;
98    } /* end for */
99
100   /* summarize frequencies */
101   for ( j = 0; j < SIZE; j++ ) {
102      ++freq[ answer[ j ] ];
103   } /* end for */
104
105   /* output headers for result columns */
106   printf( "%s%11s%19s\n\n%54s\n%54s\n\n",
107           "Response", "Frequency", "Histogram",
108           "1    1    2    2", "5    0    5    0    5" );
109
110   /* output results */
111   for ( rating = 1; rating <= 9; rating++ ) {
112      printf( "%8d%11d          ", rating, freq[ rating ] );
113
114      /* keep track of mode value and largest frequency value */
115      if ( freq[ rating ] > largest ) {
116         largest = freq[ rating ];
117         modeValue = rating;
118      } /* end if */
```

Outline

fig06_16.c

(4 of 6)

```
119
120      /* output histogram bar representing frequency value */
121      for ( h = 1; h <= freq[ rating ]; h++ ) {
122         printf( "*" );
123      } /* end inner for */
124
125      printf( "\n" ); /* being new line of output */
126   } /* end outer for */
127
128   /* display the mode value */
129   printf( "The mode is the most frequent value.\n"
130           "For this run the mode is %d which occurred"
131           " %d times.\n", modeValue, largest );
132  } /* end function mode */
133
134 /* function that sorts an array with bubble sort algorithm */
135 void bubbleSort( int a[] )
136 {
137   int pass; /* pass counter */
138   int j;    /* comparison counter */
139   int hold; /* temporary location used to swap elements */
140
141   /* loop to control number of passes */
142   for ( pass = 1; pass < SIZE; pass++ ) {
143
144      /* loop to control number of comparisons per pass */
145      for ( j = 0; j < SIZE - 1; j++ ) {
146
```

Outline

fig06_16.c

(5 of 6)

```
147         /* swap elements if out of order */
148         if ( a[ j ] > a[ j + 1 ] ) {
149            hold = a[ j ];
150            a[ j ] = a[ j + 1 ];
151            a[ j + 1 ] = hold;
152         } /* end if */
153
154      } /* end inner for */
155
156   } /* end outer for */
157
158 } /* end function bubbleSort */
159
160 /* output array contents (20 values per row) */
161 void printArray( const int a[] )
162 {
163   int j; /* counter */
164
165   /* output array contents */
166   for ( j = 0; j < SIZE; j++ ) {
167
168      if ( j % 20 == 0 ) { /* begin new line every 20 values */
169         printf( "\n" );
170      } /* end if */
171
172      printf( "%2d", a[ j ] );
173   } /* end for */
174
175 } /* end function printArray */
```

Outline

fig06_16.c

(6 of 6)

Outline

(1 of 2)

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


********
 Median
********
The unsorted array of responses is
 6 7 8 9 8 7 8 9 8 9 7 8 9 5 9 8 7 8 7 8
 6 7 8 9 3 9 8 7 8 7 7 8 9 8 9 8 9 7 8 9
 6 7 8 7 8 7 9 8 9 2 7 8 9 8 9 8 9 7 5 3
 5 6 7 2 5 3 9 4 6 4 7 8 9 6 8 7 8 9 7 8
 7 4 4 2 5 3 8 7 5 6 4 5 6 1 6 5 7 8 7

The sorted array is
 1 2 2 2 3 3 3 3 4 4 4 4 4 5 5 5 5 5 5 5
 5 6 6 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7 7
 7 7 7 7 7 7 7 7 7 7 7 7 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
```

(continued on next slide…)

(continued from previous slide…)

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

********
  Mode
********
Response  Frequency          Histogram

                                    1    1    2    2
                               5    0    5    0    5

      1          1          *
      2          3          ***
      3          4          ****
      4          5          *****
      5          8          ********
      6          9          *********
      7         23          ***********************
      8         27          ***************************
      9         19          *******************
The mode is the most frequent value.
For this run the mode is 8 which occurred 27 times.
```

Outline

(2 of 2)

6.8 Searching Arrays

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

Outline

fig06_18.c

(1 of 3)

```
1  /* Fig. 6.18: fig06_18.c
2     Linear search of an array */
3  #include <stdio.h>
4  #define SIZE 100
5
6  /* function prototype */
7  int linearSearch( const int array[], int key, int size );
8
9  /* function main begins program execution */
10 int main( 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 */
21
```

```
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 */
35
36    return 0; /* indicates successful termination */
37
38 } /* end main */
39
40 /* compare key to every element of array until the location is found
41    or until the end of array is reached; return subscript of element
42    if key or -1 if key is not found */
43 int linearSearch( const int array[], int key, int size )
44 {
45    int n; /* counter */
46
```

Outline

fig06_18.c

(2 of 3)

Outline

```
47    /* loop through array */
48    for ( n = 0; n < size; ++n ) {
49
50       if ( array[ n ] == key ) {
51          return n; /* return location of key */
52       } /* end if */
53
54    } /* end for */
55
56    return -1; /* key not found */
57
58 } /* end function linearSearch */
```

Linear search algorithm searches through every element in the array until a match is found

fig06_18.c

(3 of 3)

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

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

6.9 Multiple-Subscripted Arrays

- **Multiple subscripted arrays**
  - **Tables with rows and columns (`m` by `n` array)**
  - **Like matrices: specify row, then column**
- **Initialization**
  - `int b[ 2 ][ 2 ] = { { 1, 2 }, { 3, 4 } };`
  - **Initializers grouped by row in braces**
  - **If not enough, unspecified elements set to zero**

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

- **Referencing elements**
  - **Specify row, then column**

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

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 syntax error.

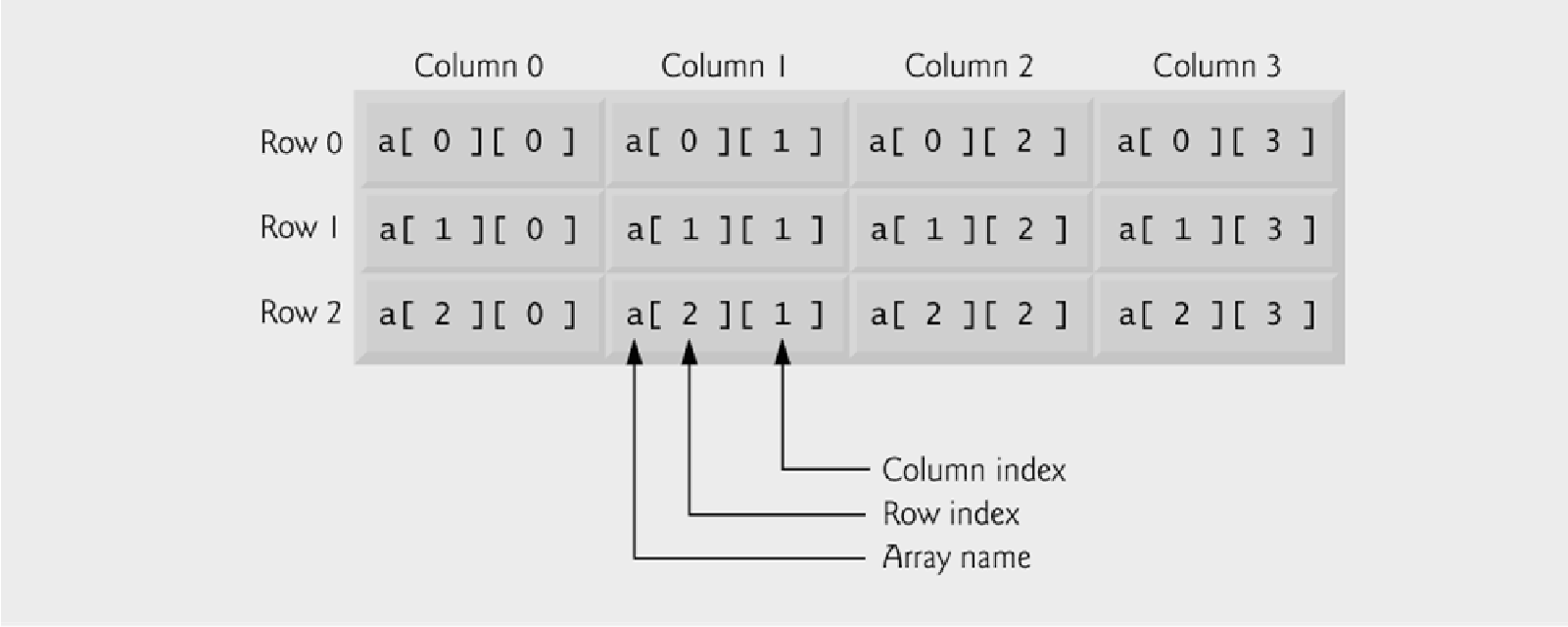

Fig. 6.20 | Double-subscripted array with three rows and four columns.

Outline

fig06_21.c

(1 of 2)

```
1  /* Fig. 6.21: fig06_21.c
2     Initializing multidimensional arrays */
3  #include <stdio.h>
4
5  void printArray( const int a[][ 3 ] ); /* function prototype */
6
7  /* function main begins program execution */
8  int main( 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
24    return 0; /* indicates successful termination */
25
26 } /* end main */
27
```

array1 is initialized with both rows full

array2 and **array3** are initialized only partially

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

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

Outline

fig06_21.c

(2 of 2)