6

C Arrays

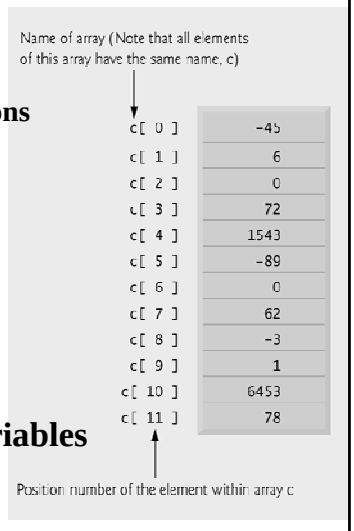
6.2 Arrays

- **Array**
  - **Group of consecutive memory locations**
  - **Same name and type**
- **To refer to an element, specify**
  - **Array name + position number**
    arrayname **[** *position number* **]**
  - **First element at position `0`**
  - **`n` element array named `c`:**
    - `c[ 0 ], c[ 1 ]...c[ n – 1 ]`
- **Array elements are like normal variables**

```
c[ 0 ] =  3;
printf( "%d", c[ 0 ] );
```

  - **Perform operations in subscript. If `x` equals `3`**

```
c[ 5 - 2 ] == c[ 3 ] == c[ x ]
```



Common Programming Error 6.1

It is important to note the difference between the "seventh element of the array" and "array element seven." Because array subscripts begin at 0, the "seventh element of the array" has a subscript of 6, while "array element seven" has a subscript of 7 and is actually the eighth element of the array. This is a source of "off-by-one" errors.

6.3 Defining Arrays

- **When defining arrays, specify**
  - **Name**
  - **Type of array**
  - **Number of elements**

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

  - **Examples:**

```
int c[ 10 ];
float myArray[ 3284 ];
```

- **Defining multiple arrays of same type**
  - **Format similar to regular variables**
  - **Example:**

```
int b[ 100 ], x[ 27 ];
```

6.4 Array Examples

- **Initializers**

```
int n[ 5 ] = { 1, 2, 3, 4, 5 };
```

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

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

    - **All elements 0**
  - **If too many initializers, a syntax error occurs**
  - **C arrays have no bounds checking**
- **If size omitted, initializers determine it**

```
int n[ ] = { 1, 2, 3, 4, 5 };
```

  - **5 initializers, therefore 5 element array**

Outline

fig06_04.c

(1 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

```
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 Errors

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

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

```
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 */
```

```
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 */
```

Outline

fig06_05.c

(1 of 2)

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

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)

Common Programming Errors

6.4: Ending a `#define` or `#include` preprocessor directive with a semicolon. Remember that preprocessor directives are not C statements.

6.5: Assigning a value to a symbolic constant in an executable statement is a syntax error. A symbolic constant is not a variable. No space is reserved for it by the compiler as with variables that hold values at execution time.

```
2    Initialize the elements of array s to the even
3  #include <stdio.h>
4  #define SIZE 10 /* maximum size of array */
5
6  /* function main begins program execution */
7  int main( void )
```

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_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 */
```

frequency array is defined with 7 elements
Reset to 0 at once !!

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

Outline

fig06_09.c

(2 of 2)

```
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
```

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)

Common Programming Error 6.7

Not providing `scanf` with a character array large enough to store a string typed at the keyboard can result in destruction of data in a program and other runtime errors. This can also make a system susceptible to worm and virus attacks.

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.

```
24 void staticArrayInit( void )
25 {
26    /* initializes elements to 0
27    static int array1[ 3 ];
28    int i; /* counter */
```

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

```
47 void automaticArrayInit( void )
48 {
49    /* initializes elements each time
50    int array2[ 3 ] = { 1, 2, 3 };
51    int i; /* counter */
```

automatic array is recreated every time `automaticArrayInit` is called

6.5 Passing Arrays to Functions

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

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

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

6.5 Passing Arrays to Functions

- **Function prototype**

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

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

Outline

fig06_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]**

Outline

fig06_13.c

(3 of 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
```

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.

```
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 */
```

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

fig06_15.c

(2 of 2)

```
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

```
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
```

6.8 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
```

Outline

fig06_18.c

(2 of 3)

```
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

(3 of 3)

```
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

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

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

6.8 Searching Arrays

- **Binary search**
  - **For sorted arrays only**
  - **Compares `middle` element with `key`**
    - **If equal, match found**
    - **If `key < middle`, looks in first half of array**
    - **If `key > middle`, looks in last half**
    - **Repeat**
  - **Very fast; at most n steps, where 2^n > number of elements**
    - **30 element array takes at most 5 steps**

      2^5 > 30 so at most 5 steps

Outline

fig06_19.c

(1 of 6)

```
1  /* Fig. 6.19: fig06_19.c
2     Binary search of an array */
3  #include <stdio.h>
4  #define SIZE 15
5
6  /* function prototypes */
7  int binarySearch( const int b[], int searchKey, int low, int high );
8  void printHeader( void );
9  void printRow( const int b[], int low, int mid, int high );
10
11 /* function main begins program execution */
12 int main( 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 */
23
24    printf( "Enter a number between 0 and 28: " );
25    scanf( "%d", &key );
26
27    printHeader();
28
29    /* search for key in array a */
30    result = binarySearch( a, key, 0, SIZE - 1 );
```

```
31
32    /* display results */
33    if ( result != -1 ) {
34       printf( "\n%d found in array element %d\n", key, result );
35    } /* end if */
36    else {
37       printf( "\n%d not found\n", key );
38    } /* end else */
39
40    return 0; /* indicates successful termination */
41
42 } /* end main */
43
44 /* function to perform binary search of an array */
45 int binarySearch( const int b[], int searchKey, int low, int high )
46 {
47    int middle; /* variable to hold middle element of array */
48
49    /* loop until low subscript is greater than high subscript */
50    while ( low <= high ) {
51
52       /* determine middle element of subarray being searched */
53       middle = ( low + high ) / 2;
54
55       /* display subarray used in this loop iteration */
56       printRow( b, low, middle, high );
57
```

Outline

fig06_19.c

(2 of 6)


```
58       /* if searchKey matched middle element, return middle */
59       if ( searchKey == b[ middle ] ) {
60          return middle;
61       } /* end if */
62
63       /* if searchKey less than middle element, set new high */
64       else if ( searchKey < b[ middle ] ) {
65          high = middle - 1; /* search low end of array */
66       } /* end else if */
67
68       /* if searchKey greater than middle element, set new low */
69       else {
70          low = middle + 1; /* search high end of array */
71       } /* end else */
72
73    } /* end while */
74
75    return -1;   /* searchKey not found */
76
77 } /* end function binarySearch */
78
79 /* Print a header for the output */
80 void printHeader( void )
81 {
82    int i; /* counter */
83
84    printf( "\nSubscripts:\n" );
85
```

If value is found, return its index

If value is too high, search the left half of array

If value is too low, search the right half of array

Outline

fig06_19.c

(3 of 6)

Outline

fig06_19.c

(4 of 6)

```
86    /* output column head */
87    for ( i = 0; i < SIZE; i++ ) {
88       printf( "%3d ", i );
89    } /* end for */
90
91    printf( "\n" ); /* start new line of output */
92
93    /* output line of - characters */
94    for ( i = 1; i <= 4 * SIZE; i++ ) {
95       printf( "-" );
96    } /* end for */
97
98    printf( "\n" ); /* start new line of output */
99 } /* end function printHeader */
100
101 /* Print one row of output showing the current
102    part of the array being processed. */
103 void printRow( const int b[], int low, int mid, int high )
104 {
105    int i; /* counter for iterating through array b */
106
```

Outline

fig06_19.c

(5 of 6)

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

```
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*

25 not found
```

(continued on next slide…)

(continued from previous slide…)

```
Enter a number between 0 and 28: 8

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
                  8  10* 12
                  8*

8 found in array element 4
```

```
Enter a number between 0 and 28: 6

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

6 found in array element 3
```

Outline

fig06_19.c

(6 of 6)

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 ] );
    ```

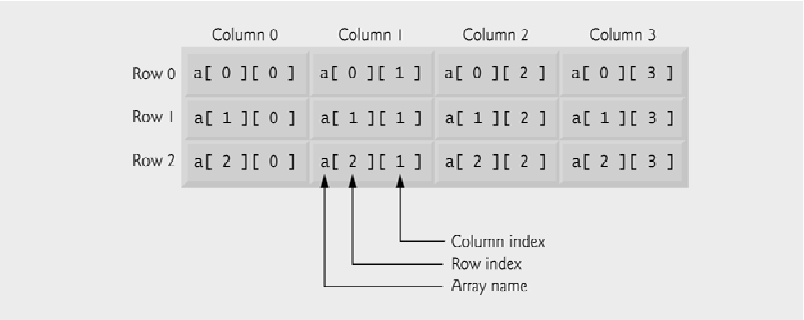


Fig. 6.20 | Double-subscripted array with three rows and four columns.

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.

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

Outline

fig06_21.c

(2 of 2)

```
28 /* function to output array with two rows and three columns */
29 void printArray( const int a[][ 3 ] )
30 {
31    int i; /* 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
```