

Chapter 9 - Arrays

Outline

1. Introduction
2. Arrays
3. Declaring Arrays
4. Examples Using Arrays
5. Passing Arrays to Functions
6. Multiple-Subscripted Arrays



Objectives

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



Arrays

Name of array (Note³ that all elements of this array have the same name, c)

- Array
 - Group of consecutive memory locations
 - Same name and type
- To refer to an element, specify
 - Array name
 - Position number
- Format:
 - arrayname* [*position number*]
 - First element at position 0
 - n element array named c:
 - c[0], c[1]...c[n - 1]

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78

Position number of the element within array c



Arrays

- Array elements are like normal variables

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

- Perform operations in subscript. If x equals 3

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



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];`



Examples Using Arrays

- Initializers

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

- If not enough initializers, rightmost elements become 0

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

- All elements 0

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

- If size omitted, initializers determine it

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

- 5 initializers, therefore 5 element array





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

Element	value
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0



Outline

Examples Using Arrays

- Character arrays
 - String “first” is really a static array of characters
 - Character arrays can be initialized using string literals

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

 - Null character '\0' terminates strings
 - `string1` actually has 6 elements
 - It is equivalent to

```
char string1[] = { 'f', 'i', 'r', 's', 't', '\0' };
```
 - Can access individual characters

```
string1[3] is character 's'
```
 - Array name is address of array, so `&` not needed for `scanf`

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

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





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

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



Outline



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

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



Outline



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

Total of array element values is 383

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



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`





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



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



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



Effects of passing entire array by reference:

The values of the original array are:

0 1 2 3 4

The values of the modified array are:

0 2 4 6 8

Effects of passing array element by value:

The value of a[3] is 6

Value in modifyElement is 12

The value of a[3] is 6

Multiple-Subscripted Arrays

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

	Column	Column 1	Column	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Array name Row subscript Column subscript



Multiple-Subscripted Arrays

- Initialization

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

1	2
3	4

- Initializers grouped by row in braces

- If not enough, unspecified elements set to zero

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

1	0
3	4

- Referencing elements

- Specify row, then column

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





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



Outline

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

Values in array1 by row are:

1 2 3

4 5 6

Values in array2 by row are:

1 2 3

4 5 0

Values in array3 by row are:

1 2 0

4 0 0



Outline

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



Outline

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



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



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



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



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

The array is:

	[0]	[1]	[2]	[3]
studentGrades[0]	77	68	86	73
studentGrades[1]	96	87	89	78
studentGrades[2]	70	90	86	81

Lowest grade: 68

Highest grade: 96

The average grade for student 0 is 76.00

The average grade for student 1 is 87.50

The average grade for student 2 is 81.75