6

C Arrays

OBJECTIVES

In this chapter you will learn:

- To use the array data structure to represent lists of values.
- To define an array, initialize an array and refer to individual elements of an array.
- To define symbolic constants.

Outline

6.1 Introduction
6.2 Arrays
6.3 Defining Arrays
6.4 Array Examples

6.1 Introduction

- **Arrays**
  - **Structures of related data items**
  - **Static entity – same size throughout program**
  - **Dynamic data structures discussed in Chapter 12**

6.2 Arrays

- **Array**
  - **Group of consecutive memory locations**
  - **Same name and type**
- **To refer to an element, specify**
  - **Array name**
  - **Position number**
- **Format:**

  arrayname`[` *position number* `]`
  - **First element at position `0`**
  - **`n` element array named `c`:**
    - `c[ 0 ], c[ 1 ]...c[ n - 1 ]`

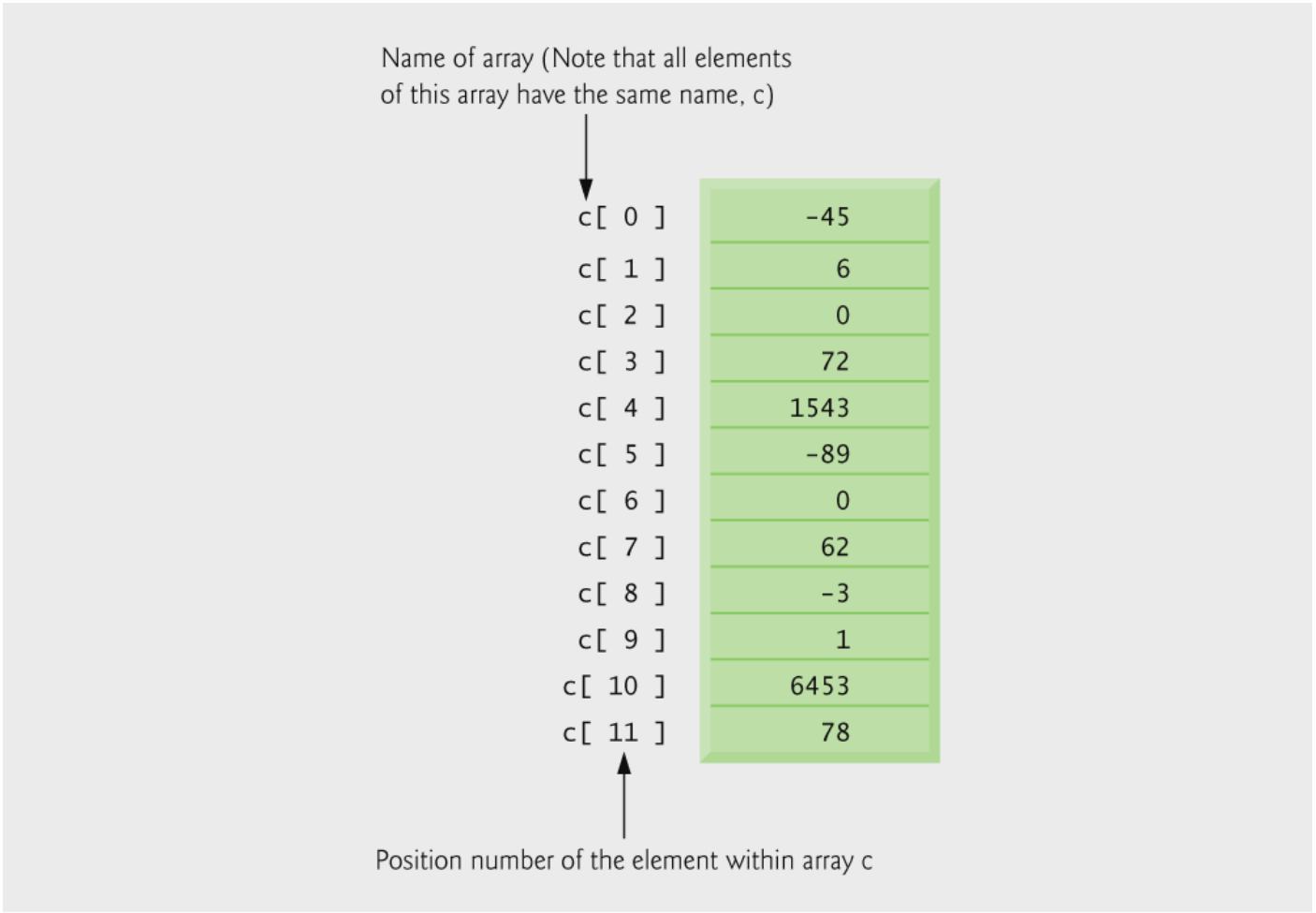

Fig. 6.1 | 12-element array.

6.2 Arrays

- **Array elements are like normal variables**

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

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

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

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.

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

Fig. 6.2 | Operator precedence.

6.3 Defining Arrays

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

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

  - Examples:

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

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

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

6.4 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_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)

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 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

```c
/* Fig. 6.5: fig06_05.c
   Initialize the elements of array s to the even integers from 2 to 20 */
#include <stdio.h>
#define SIZE 10 /* maximum size of array */

/* function main begins program execution */
int main( void )
{
   /* symbolic constant SIZE can be used to specify array size */
   int s[ SIZE ]; /* array s has SIZE elements */
   int j; /* counter */

   for ( j = 0; j < SIZE; j++ ) { /* set the values */
      s[ j ] = 2 + 2 * j;
   } /* end for */

   printf( "%s%13s\n", "Element", "Value" );

   /* output contents of array s in tabular format */
   for ( j = 0; j < SIZE; j++ ) {
      printf( "%7d%13d\n", j, s[ j ] );
   } /* end for */

   return 0; /* indicates successful termination */

} /* 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)

Common Programming Error 6.4

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

Common Programming Error 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.

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

```c
/* Fig. 6.7: fig06_07.c
   Student poll program */
#include <stdio.h>
#define RESPONSE_SIZE 40 /* define array sizes */
#define FREQUENCY_SIZE 11

/* function main begins program execution */
int main( void )
{
   int answer; /* counter to loop through 40 responses */
   int rating; /* counter to loop through frequencies 1-10 */

   /* initialize frequency counters to 0 */
   int frequency[ FREQUENCY_SIZE ] = { 0 };

   /* place the survey responses in the responses array */
   int responses[ RESPONSE_SIZE ] = { 1, 2, 6, 4, 8, 5, 9, 7, 8, 10,
        1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6,
        5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };

   /* for each answer, select value of an element of array responses
        and use that value as subscript in array frequency to
        determine element to increment */
   for ( answer = 0; answer < RESPONSE_SIZE; answer++ ) {
      ++frequency[ responses [ answer ] ];
   } /* end for */

```

`#define` directives create symbolic constants

`fig06_07.c`

(1 of 2)

`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

```
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)

Good Programming Practice 6.3

Strive for program clarity. Sometimes it may be worthwhile to trade off the most efficient use of memory or processor time in favor of writing clearer programs.

Performance Tip 6.1

Sometimes performance considerations far outweigh clarity considerations.

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.

Error-Prevention Tip 6.2

Programs should validate the correctness of all input values to prevent erroneous information from affecting a program's calculations.

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)

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)

```c
/* Fig. 6.11: fig06_11.c
   Static arrays are initialized to zero */
#include <stdio.h>

void staticArrayInit( void );    /* function prototype */
void automaticArrayInit( void ); /* function prototype */

/* function main begins program execution */
int main( void )
{
   printf( "First call to each function:\n" );
   staticArrayInit();
   automaticArrayInit();

   printf( "\n\nSecond call to each function:\n" );
   staticArrayInit();
   automaticArrayInit();

   return 0; /* indicates successful termination */

} /* end main */

```

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
```


Common Programming Error 6.8

Assuming that elements of a local `static` array are initialized to zero every time the function in which the array is defined is called.