array parameter, there is a local variable in the called function that is used to "walk" through the array.

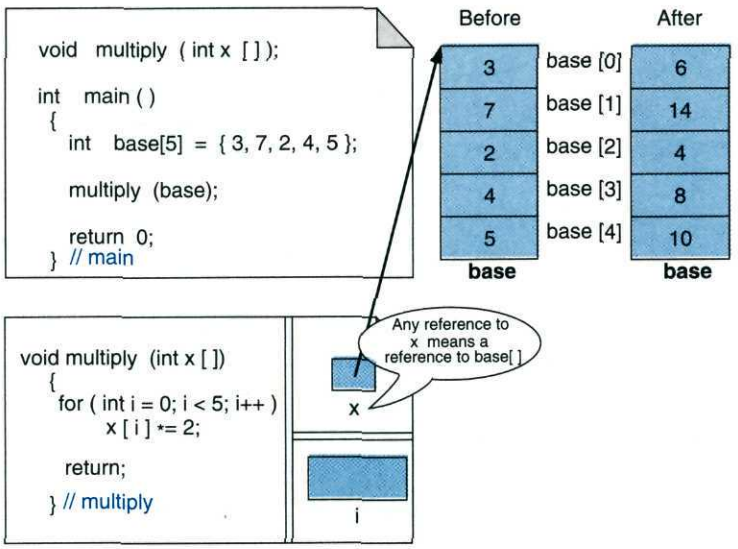


Figure 8-13 Changing values in arrays

8-4 ARRAY APPLICATIONS

In this section we discuss three common array applications.

FREQUENCY ARRAYS

A **frequency array** shows the number of elements with an identical value found in a series of numbers. For example, suppose we have taken a sample of 100 values between 0 and 19. We want to know how many of the values are 0, how many are 1, how many are 2, and so forth up through 19.

We can read these numbers into an array called `numbers`. We then create an array of 20 elements that will show the frequency of each number in the series. This design is shown in Figure 8-14.

With the data structure shown in Figure 8-14 in mind, how do we write the application? Since we know that there are exactly 100 elements, we can use a *for* loop to examine each value in the array. But how can we relate the value in `numbers` to a location in the frequency?

One way to do it is to assign the value from the data array to an index and then use the index to access the frequency array. The code for this technique is shown below.

```
f = numbers[i];
frequency [f]++;
```

Since an index is an expression, however, we can simply use the value from our data array as the index for the frequency array. This concept is shown in the following

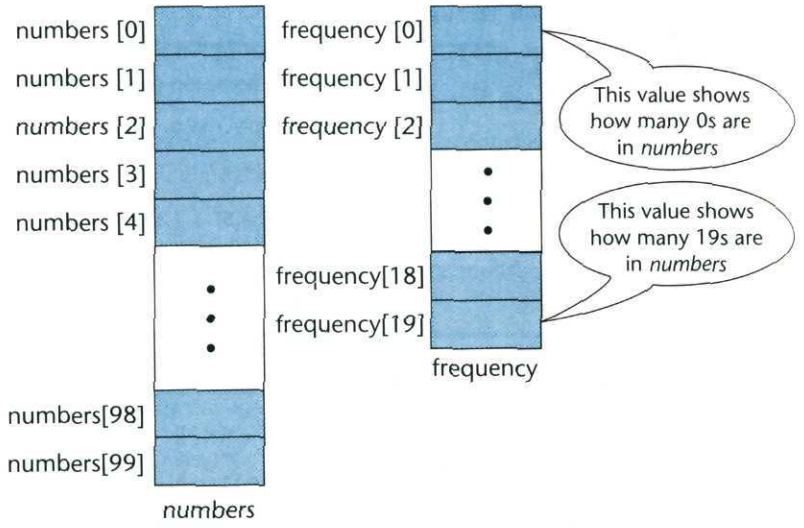


Figure 8-14 Frequency array

example in which the value of `numbers[i]` is determined first, and that value is then used to index into `frequency`.

```
frequency [numbers [i]]++;
```

The complete function is shown in Program 8-4 as `makeFrequency`. The function first initializes the frequency array and then scans the data array to count the number of occurrences of each value.

There is a potentially serious problem with this function. Can you see what it is? Think about our discussion of what happens if the index gets out of range (see page 338). What if one of the numbers in our data is greater than 19? We will be destroying some other part of our program! To protect against this possibility, each data value should be tested to make sure that it is within the indexing range of *frequency*.

HISTOGRAMS

A **histogram** is a pictorial representation of a frequency array. Instead of printing the values of the elements to show the frequency of each number, we can print a histogram in the form of a bar chart. For example, Figure 8-15 is a histogram for a set of numbers in the range 0...19. In this example, asterisks (*) are used to build the bar. Each asterisk represents one occurrence of the data value.

Let's write a program that builds a frequency array for data values in the range 0...19 and then prints their histogram. The data are read from a file. To provide flexibility, the `getData` function may only partially fill the array. The function that loads it also guards against too much data. The design for the program is shown in Figure 8-16, and the code used is shown in Program 8-4.

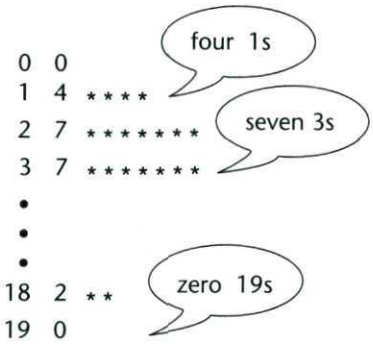


Figure 8-15 Frequency histogram

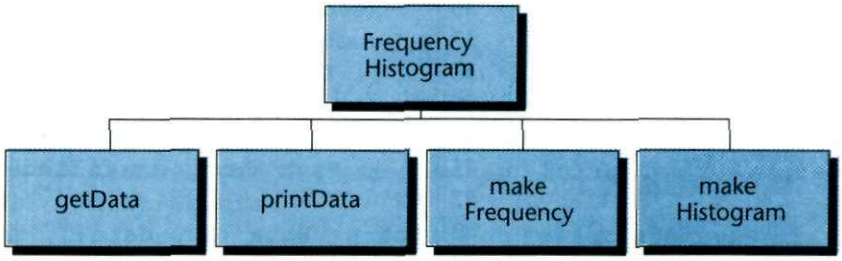


Figure 8-16 Histogram program design

Program 8-4 Frequency and histogram

```
/* Read data from a file into an array.
   Build frequency array & print the data with histogram.
      Written by:
      Date:
*/
#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;

const int cMAX_ELMNTS  = 100;
const int cANLYS_RANGE =  19;

   int getData (int  numbers[], int size, int range);

   void printData     (const int numbers[], int size,
                       int lineSize);
   void makeFrequency (int numbers[], int size,
                       int frequency[], int range);
   void makeHistogram (int frequency[], int range);

```

Program 8-4 Frequency and histogram *(continued)*

```
22 int main ()
23 {
24    int size;
25    int nums[cMAX_ELMNTS];
26    int frequency[cANLYS_RANGE + 1];
27
28    size = getData (nums, cMAX_ELMNTS, cANLYS_RANGE);
29    printData (nums, size, 10);
30
31    makeFrequency(nums,size,frequency,cANLYS_RANGE);
32    makeHistogram(frequency, cANLYS_RANGE);
33    return 0;
34 } // main
35 /* ================== getData ==================
36    Read data from file into array. The array
37    does not have to be completely filled.
38       Pre   data is an empty array
39             size is maximum elements in array
40             range is highest value that can be accepted
41       Post  Array is filled--Return number of elements
42 */
43 int getData (int data[], int size, int range)
44 {
45    ifstream  fsData;
46    fsData.open("histogrm.dat");
47    if (!fsData)
48       cerr << "Error opening file\a\a\n", exit (100);
49
50    int dataIn;
51    int loader = 0;
52    while (loader < size
53        && (fsData >> dataIn))
54       if (dataIn >= 0 && dataIn <= range)
55          data[loader++] = dataIn;
56       else
57          cout << "Data point " << dataIn
58               << " invalid. Ignored. \n";
59
60 // Test to see what stopped while
61    if (loader == size && (fsData >> dataIn))
62       // More data in file
63      cout << "\nToo much data. Process what read.\n";
64    return loader;
65 } // getData
66 /* ================== printData ==================
67    Print the data as a two-dimensional array.
68       Pre   data: a filled array
69             size: size of array to be printed
70            lineSize: max elements printed on a line
```

Program 8-4 Frequency and histogram *(continued)*

```
71          Post  The data have been printed
72  */
73  void printData (const int data[], int size, int lineSize)
74  {
75     cout << endl << endl;
76     for (int i = 0, numPrinted = 0; i < size; i++)
77        {
78         numPrinted++;
79         cout << setw(3) << data[i];
80         if (numPrinted >= lineSize)
81            {
82             cout << endl;
83             numPrinted = 0;
84            } // if
85        } // for
86     cout << endl << endl;
87     return;
88  } // printData
89  /* ================ makeFrequency =================
90     Analyze the data in nums and build their frequency
91     distribution.
92        Pre    nums: array of data for analysis
93               size: size of array containing data
94               frequency: accumulation array
95               range: maximum value of data
96        Post   Frequency array has been built
97  */
98  void makeFrequency (int nums[],       int size,
99                      int frequency[], int range)
100 {
101    // First initialize the frequency array
102    for (int i = 0; i <= range; i++)
103       frequency [i] = 0;
104
105    // Scan numbers and build frequency array
106    for (int i = 0; i < size; i++)
107       frequency [nums [i]]++;
108    return;
109 } // makeFrequency
110 /* ================ makeHistogram =================
111    Print a histogram representing analyzed data.
112       Pre   freq contains value count
113             range max data value & max array index
114       Post  histogram has been printed
115 */
116 void makeHistogram (int freq[], int range)
117 {
118    for (int i = 0; i <= range; i++)
119       {
```

Program 8-4 Frequency and histogram *(continued)*

```
120         cout << setw(3) << i << setw(3) << freq[i];
121         for (int j = 1; j <= freq[i]; j++)
122             cout << "*";
123         cout << endl;
124       } // for i
125   return;
126 } // makeHistogram
127 // ================ End of Program ================
```

```
Results:
Data point 20 invalid. Ignored.
Data point 25 invalid. Ignored.

 1  2  3  4  5  6  7  8  7 10
 2 12 13 13 15 16 17 18 17  7
 3  4  6  8 10  2  4  6  8 10
 4  3  5  7  1  3  7  7 11 13
 5 10 11 12 13 16 18 11 12  7
 6  1  2  2  3  3  3  4  4  4
 7  7  8  7  6  5  4  1  2  2
 8 11 11 13 13 13 17 17  7  7
13 17 17 15 15

 0  0
 1  4 ****
 2  7 *******
 3  7 *******
 4  8 ********
 5  4 ****
 6  5 *****
 7 12 ************
 8  5 *****
 9  0
10  4 ****
11  5 *****
12  3 ***
13  8 ********
14  0
15  3 ***
16  2 **
17  6 ******
18  2 **
19  0
```

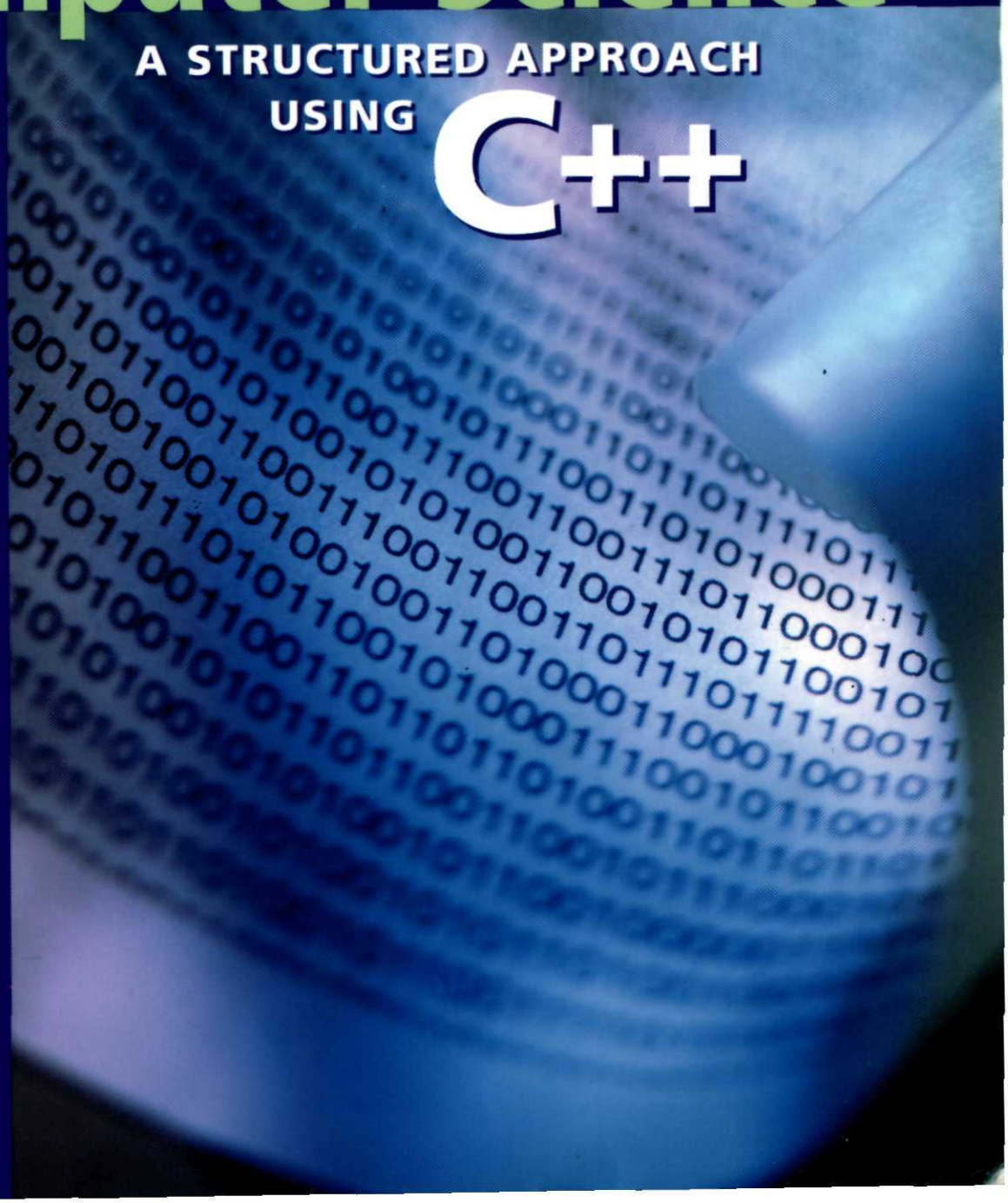
RANDOM NUMBER PERMUTATIONS

A random number permutation is a set of random numbers in which no numbers are repeated. For example, given a random number permutation of 10 numbers, the values from 0 to 9 would all be included with no duplicates.

Behrouz A. Forouzan
Richard F. Gilberg

Computer Science

A STRUCTURED APPROACH USING C++



SECOND EDITION