CHAPTER 6
Arrays

Objectives

• To describe why an array is necessary in programming (§6.1).
• To learn how to declare an array (§§6.2.1).
• To access array elements using indexed variables (§6.2.2).
• To initialize the values in an array (§6.2.3).
• To develop and invoke functions with array arguments (§§6.3–6.4).
• To search elements using the linear (§6.5.1) or binary search algorithm (§6.5.2).
• To sort an array using the selection sort (§6.6.1)
• (Optional) To sort an array using the insertion sort algorithms (§6.6.2).
• To declare and create two-dimensional arrays (§6.7).
• (Optional) To declare and create multidimensional arrays (§6.8).
6.1 Introduction

<Side Remark: why array?>
<Side Remark: array>

Often you will have to store a large number of values during the execution of a program. Suppose, for instance, that you want to read one hundred numbers, compute their average, and find out how many numbers are above the average. Your program first reads the numbers and computes their average, and then compares each number with the average to determine whether it is above the average. The numbers must all be stored in variables in order to accomplish this task. You have to declare one hundred variables and repeatedly write almost identical code one hundred times. From the standpoint of practicality, it is impossible to write a program this way. An efficient, organized approach is needed. C++ and all other high-level languages provide a data structure, the array, which stores a fixed-size sequential collection of elements of the same type.

6.2 Array Basics

An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type. Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], ..., and numbers[99] to represent individual variables. This section introduces how to declare array variables, create arrays, and process arrays using indexed variables.

6.2.1 Declaring Arrays

To declare an array, you need to specify its element type and size using the following syntax:

```c
dataType arrayName[arraySize];
```

The arraySize must be an integer greater than zero. For example, the following statement declares an array of ten double values:

```c
double myList[10];
```

The compiler allocates ten double elements for array myList. Figure 6.1 illustrates the array with sample element values.
double myList[10];

<table>
<thead>
<tr>
<th>Index</th>
<th>Element Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5.6</td>
</tr>
<tr>
<td>1</td>
<td>4.5</td>
</tr>
<tr>
<td>2</td>
<td>3.3</td>
</tr>
<tr>
<td>3</td>
<td>13.2</td>
</tr>
<tr>
<td>4</td>
<td>4.0</td>
</tr>
<tr>
<td>5</td>
<td>34.33</td>
</tr>
<tr>
<td>6</td>
<td>34.0</td>
</tr>
<tr>
<td>7</td>
<td>45.45</td>
</tr>
<tr>
<td>8</td>
<td>99.993</td>
</tr>
<tr>
<td>9</td>
<td>11123</td>
</tr>
</tbody>
</table>

Array element at index 5

Element value

Figure 6.1
The array myList has ten elements of double type and int indices from 0 to 9.

NOTE
C++ requires that the array size used to declare an array must be a constant expression. For example, the following code is illegal:

```c++
int size = 4;
double myList[size]; // Wrong
```

But it would be OK, if size is a constant as follow:

```c++
const int size = 4;
double myList[size]; // Correct
```

***END NOTE

NOTE

.Side Remark: arbitrary initial values>
When an array is created, its elements are assigned with arbitrary values.

TIP

.Side Remark: declaring together>
If arrays have the same element type, they can be declared together, as follows:

```c++
datatype arrayName1[size1], arrayName2[size2], ..., arrayNamen[sizen];
```

The variables are separated by commas. For example,

```c++
double list1[10], list2[25];
```
6.2.2 Array Indexed Variables

The array elements are accessed through the index. Array indices are 0-based; that is, they start from 0 to arraySize-1. In the example in Figure 6.1, myList holds ten double values and the indices are from 0 to 9.

Each element in the array is represented using the following syntax, known as an indexed variable:

```
arrayName[index];
```

For example, myList[9] represents the last element in the array myList.

CAUTION
Some languages use parentheses to reference an array element, as in myList(9). But C++ uses brackets, as in myList[9].

CAUTION
C++ does not check array’s boundary. So, accessing array elements using subscripts beyond the boundary (e.g., myList[-1] and myList[11]) does not not cause syntax errors, but the operating system might report a memory access violation.

After an array is declared, an indexed variable can be used in the same way as a regular variable. For example, the following code adds the values in myList[0] and myList[1] to myList[2].

```
myList[2] = myList[0] + myList[1];
```

The following loop assigns 0 to myList[0], 1 to myList[1], ..., and 9 to myList[9]:

```
for (int i = 0; i < 10; i++)
```
```
myList[i] = i;
```

6.2.3 Array Initializers

C++ has a shorthand notation, known as the array initializer, which combines declaring an array and initializing in one statement using the following syntax:

```
dataType arrayName[arraySize] = {value0, value1, ..., valuek};
```
For example,

```c
double myList[4] = {1.9, 2.9, 3.4, 3.5};
```

This statement declares and initializes the array `myList` with four elements, which is equivalent to the statements shown below:

```c
double myList[4];
myList[0] = 1.9;
myList[1] = 2.9;
myList[2] = 3.4;
myList[3] = 3.5;
```

**CAUTION**

Using an array initializer, you have to declare, and initialize the array all in one statement. Splitting it would cause a syntax error. Thus the next statement is wrong:

```c
double myList[4];
myList = {1.9, 2.9, 3.4, 3.5};
```

***End of CAUTION***

**NOTE:**

*Side Remark: implicit size*

C++ allows you to omit the array size when declaring and creating an array using an initializer. For example, the following declaration is fine:

```c
double myList[] = {1.9, 2.9, 3.4, 3.5};
```

C++ automatically figures out how many elements are in the array.

***End of NOTE***

**NOTE:**

*Side Remark: partial initialization*

C++ allows you to initialize a part of the array. For example, the following statement assigns values 1.9, 2.9 to the first two elements of the array. The other two elements will be set to zero. Note that if an array is declared, but not initialized, all its elements will contain “garbage”, like all other local variables.

```c
double myList[4] = {1.9, 2.9};
```

***END NOTE***

6.2.4 Initializing Character Arrays
You can initialize a character array using an initializer just like any other arrays. For example, the following code initializes array `city`:
C++ allows you to initialize a character array simply with a string. For example,

```cpp
char city[] = {'D', 'a', 'l', 'l', 'a', 's'};
```

This statement is equivalent to the preceding statement, except that C++ adds the character '\0', called the null terminator, to indicate the end of the string, as shown in Figure 6.2. Recall that a character that begins with the back slash symbol (\) is an escape character.

![Figure 6.2](image)

**Figure 6.2**

A character array can be initialized with a string.

### 6.2.5 Processing Arrays

When processing array elements, you will often use a for loop. Here are the reasons why:

- All of the elements in an array are of the same type. They are evenly processed in the same fashion by repeatedly using a loop.
- Since the size of the array is known, it is natural to use a for loop.

Assume the array is declared as follows:

```cpp
const int ARRAY_SIZE = 10;
double myList[ARRAY_SIZE];
```

Here are some examples of processing arrays:

1. (Initializing arrays with random values) The following loop initializes the array myList with random values between 0 and 99:

   ```cpp
   for (int i = 0; i < ARRAY_SIZE; i++)
   {
       myList[i] = rand() % 100;
   }
   ```

2. (Printing arrays) To print an array, you have to print each element in the array using a loop like the following:

   ```cpp
   for (int i = 0; i < ARRAY_SIZE; i++)
   {
       cout << myList[i] << " ";
   }
   ```

   **TIP:**

   `<Side Remark: print character array>`

   For a character array, it can be printed using one print statement. For example, the following code displays Dallas:
char city[] = "Dallas";
cout << city;

***END TIP

3. (Copying arrays) Can you copy array using a syntax like this?

    list = myList;

This is not allowed in C++. You have to copy individual elements from one array to the other as follows:

    for (int i = 0; i < ARRAY_SIZE; i++)
        list[i] = myList[i];

4. (Summing all elements) Use a variable named total to store the sum. Initially total is 0. Add each element in the array to total using a loop like this:

    double total = 0;
    for (int i = 0; i < ARRAY_SIZE; i++)
        total += myList[i];

5. (Finding the largest element) Use a variable named max to store the largest element. Initially max is myList[0]. To find the largest element in the array myList, compare each element in myList with max, update max if the element is greater than max.

    double max = myList[0];
    for (int i = 1; i < ARRAY_SIZE; i++)
        if (myList[i] > max) max = myList[i];

6. (Finding the smallest index of the largest element) Often you need to locate the largest element in an array. If an array has more than one largest element, find the smallest index of such an element. Suppose the array myList is {1, 5, 3, 4, 5, 5}. So, the largest element is 5 and the smallest index for 5 is 1. Use a variable named max to store the largest element and a variable named indexOfMax to denote the index of the largest element. Initially max is myList[0] and indexOfMax is 0. Compare each element in myList with max, update max and indexOfMax if the element is greater than max.

    double max = myList[0];
    int indexOfMax = 0;
    for (int i = 1; i < ARRAY_SIZE; i++)
        if (myList[i] > max)
            max = myList[i];
        else
            indexOfMax = i;
What is the consequence if `myList[i] > max` is replaced by `myList[i] >= max`?

6.2.6 Example: Testing Arrays

This section presents a program that reads six integers, finds the largest of them, and counts its occurrences. Suppose that you entered 3, 5, 2, 5, 5, 5; the program finds that the largest is 5 and the occurrence count for 5 is 4.

An intuitive solution is to first read the numbers and store them in an array, then find the largest number in the array, and finally count the occurrences of the largest number in the array. The program is given in Listing 6.1.

```
#include <iostream>
using namespace std;

int main()
{
    const int TOTAL_NUMBERS = 6;
    int numbers[TOTAL_NUMBERS];

    // Read all numbers
    for (int i = 0; i < TOTAL_NUMBERS; i++)
    {
        cout << "Enter a number: ";
        cin >> numbers[i];
    }

    // Find the largest
    int max = numbers[0];
    for (int i = 1; i < TOTAL_NUMBERS; i++)
    {
        if (max < numbers[i])
            max = numbers[i];
    }

    // Find the occurrence of the largest number
    int count = 0;
    for (int i = 0; i < TOTAL_NUMBERS; i++)
    {
        if (numbers[i] == max) count++;
    }
}
```
// Display the result
    cout << "The array is ";
    for (int i = 0; i < TOTAL_NUMBERS; i++)
    {
        cout << numbers[i] << " ";
    }
    cout << "The largest number is " << max;
    cout << "The occurrence count of the largest number is " << count;
    return 0;
}

<Output>
Enter a number: 5
Enter a number: 2
Enter a number: 5
Enter a number: 5
The array is 3 5 2 5 5 5
The largest number is 5
The occurrence count of the largest number is 4
<End Output>

The program declares an array of six integers (line 7) and enters the element from the keyboard (lines 10-14). It finds the largest number in the array (lines 17-22), counts its occurrences (lines 25-29), and displays the result (lines 32-40). To display the array, you need to display each element in the array using a loop.

Without using the numbers array, you would have to declare a variable for each number entered, because all the numbers are compared to the largest number to count its occurrences after it is found.

CAUTION

<Side Remark: off-by-one error>
Programmers often mistakenly reference the first element in an array with index 1, so that the index of the tenth element becomes 10. This is called the off-by-one error.

***End NOTE

TIP

<side remark: debugging in IDE>
If you use an IDE such as C++Builder and Visual C++.NET, please refer to Learning C++ Effectively with C++Builder/Visual C++.NET in the supplements.
This supplement shows you how to use a debugger to inspect arrays.

6.2.7 Example: Assigning Grades

This example writes a program that reads student scores, gets the best score, and then assigns grades based on the following scheme:

Grade is A if score is >= best - 10;
Grade is B if score is >= best - 20;
Grade is C if score is >= best - 30;
Grade is D if score is >= best - 40;
Grade is F otherwise.

The program prompts the user to enter the total number of students, then prompts the user to enter all of the scores, and concludes by displaying the grades.

The program reads the scores, then finds the best score, and finally assigns grades to the students based on the preceding scheme. For simplicity, assume that there are five students. Listing 6.2 gives the solution to the problem.

Listing 6.2 AssignGrade.cpp (Assigning Grades)

```cpp
#include <iostream>
using namespace std;

int main()
{
    // Maximum number of students
    const int NUMBER_OF_STUDENTS = 5;
    int scores[NUMBER_OF_STUDENTS]; // Array scores
    int best = 0; // The best score
    char grade; // The grade

    // Read scores and find the best score
    for (int i = 0; i < NUMBER_OF_STUDENTS; i++)
    {
        cout << "Please enter a score: ";
        cin >> scores[i];
        if (scores[i] > best)
            best = scores[i];
    }

    // Assign and display grades
```

---

**Side Remark line 7: constant**
**Side Remark line 8: declare array**
**Side Remark line 16: enter score**
**Side Remark line 18: update best**
**Side Remark line 25: assign grade**
**Side Remark line 35: display output**
for (int i = 0; i < NUMBER_OF_STUDENTS; i++)
{
    if (scores[i] >= best - 10)
        grade = 'A';
    else if (scores[i] >= best - 20)
        grade = 'B';
    else if (scores[i] >= best - 30)
        grade = 'C';
    else if (scores[i] >= best - 40)
        grade = 'D';
    else
        grade = 'F';

    cout << "Student " << i << " score is " << scores[i] << " and grade is " << grade << "\n";
}

return 0;

<Output>
Please enter a score: 4
Please enter a score: 40
Please enter a score: 50
Please enter a score: 60
Please enter a score: 70
Student 0 score is 4 and grade is F
Student 1 score is 40 and grade is C
Student 2 score is 50 and grade is B
Student 3 score is 60 and grade is A
Student 4 score is 70 and grade is A
<End Output>

The program declares scores as an array of int type in order to store the students’ scores (line 8).

The array is not needed to find the best score, but it is needed to keep all of the scores so that grades can be assigned later on, and it is needed when scores are printed along with the students’ grades.

6.3 Passing Arrays to Functions
Just as you can pass single values to a function, you can also pass an entire array to a function. Listing 6.3 gives an example to demonstrate how to declare and invoke this type of functions.

Listing 6.3 PassArrayDemo.cpp (Passing Arrays)

***PD: Please add line numbers in the following code***
***Layout: Please layout exactly. Don’t skip the space. This is true for all source code in the book. Thanks, AU.***

<Side Remark line 4: function prototype>
<Side Remark line 8: declare array>
#include <iostream>
using namespace std;

void printArray(int list[], int arraySize); // function prototype

int main()
{
    int numbers[5] = {1, 4, 3, 6, 8};
    printArray(numbers, 5);

    return 0;
}

void printArray(int list[], int arraySize)
{
    for (int i = 0; i < arraySize; i++)
    {
        cout << list[i] << " ";
    }
}

<Output>
1 4 3 6 8
<End Output>

In the function header (line 14), int list[] indicates that the parameter is an integer array of any size. So you can pass any integer array to invoke this function (line 9).

Note that the parameter names in function prototypes can be omitted. So the function prototype may be declared without the parameter name list and arraySize as follows:

void printArray(int [], int arraySize); // function prototype

NOTE

<side remark: passing size along with array>

Normally when you pass an array to a function, you should also pass its size in another argument. So the function knows how many elements are in the functions. Otherwise, you will have to hard code this into the function or declare it in a global variable. Neither is flexible or robust.

You can pass a primitive data type variable or an array to a function. However, there are important differences between them.

<Side Remark: pass by value>

- Passing a variable of a primitive type means that the value of the variable is passed to a formal parameter. Changing the value of the local parameter inside the function does
not affect the value of the variable outside the function. This is pass by value.

*Side Remark: pass by reference*

- Passing an array variable means that the starting address of the array is passed to the formal parameter. The parameter inside the function references to the same array that is passed to the function. No new arrays are created. This is pass by reference.

Listing 6.4 gives an example that demonstrates the differences between pass by value and pass by reference.

**Listing 6.4 ModifyArray.cpp (Pass By References Demo)**

```cpp
#include <iostream>
using namespace std;

void m(int, int[]);

int main()
{
  int x = 1; // x represents an int value
  int y[10]; // y represents an array of int values
  int y[0] = 1; // Initialize y[0]

  m(x, y); // Invoke m with arguments x and y
  cout << "x is " << x << endl;
  cout << "y[0] is " << y[0] << endl;

  return 0;
}

void m(int number, int numbers[])
{
  number = 1001; // Assign a new value to number
  numbers[0] = 5555; // Assign a new value to numbers[0]
}
```

*Output*

```
x is 1
y[0] is 5555
```

*End Output*

You will see that after function m is invoked, x remains 1, but y[0] is 5555. This is because y and numbers reference to the same array, although y and numbers are independent variables. When invoking m(x, y), the values of x and y are passed to number and numbers. Since y contains the reference value to the
array, numbers now contains the same reference value to the same array.

.Side Remark: const array
Passing arrays by reference makes sense for performance reasons. If an array is passed by value, all its elements must be copied into a new array. For large arrays, it could take some time and additional memory space. However, passing arrays by reference could lead to errors if your function changes the array accidentally. To prevent it from happening, you can put the const keyword before the array parameter to tell the compiler that the array cannot be changed. The compiler will report errors if the code in the function attempts to modify the array.

Listing 6.5 gives an example that declares a const array argument list in the function p (line 4). The function attempts to modify the first element in the array in line 7. This error is detected by the compiler, as shown in the sample output.

Listing 6.5 ConstArrayDemo.cpp (Const Array Arguments)
***PD: Please add line numbers in the following code***
***Layout: Please layout exactly. Don’t skip the space. This is true for all source code in the book. Thanks, AU.
.Side Remark line 4: const array argument
.Side Remark line 7: attempt to modify

```cpp
#include <iostream>
using namespace std;

void p(const int list[], int arraySize)
{
    // Modify array accidentally
    list[0] = 100; // Compile error!
}

int main()
{
    int numbers[5] = {1, 4, 3, 6, 8};
    p(numbers, 5);

    return 0;
}
```

Output

error C2166: l-value specifies const object

End Output

Output

E2024 Cannot modify a const object

End Output
Both Visual C++.NET and C++Builder report the error. L-value means the left value on the left side of an assignment statement. Since the l-value is declared const, it cannot be changed.

**NOTE**

*side remark: cascading const parameters*

If you define a const parameter in a function f1 and this parameter is passed to another function f2, then the corresponding parameter in function f2 should be declared const for consistency. Consider the following code:

```cpp
void f2(int list[], int size)
{
    // do something
}

void f1(const int list[], int size)
{
    // Do something
    f2(list, size);
}
```

The compiler reports an error, because list is const in f1 and it is passed to f2, but it is not const in f2. The function declaration for f2 should be

```cpp
void f2(const int list[], int size)
```

***End NOTE***

### 6.4 Modifying Arrays in Functions

You can declare a function to return a primitive type value. For example,

```cpp
// Return the sum of the elements in the list
int sum(int list[], int size)
```

Can you return an array from a function using a similar syntax? For example, you may attempt to declare a function that returns a new array that is a reversal of an array as follows:

```cpp
// Return the reversal of list
int[] reverse(const int list[], int size)
```

This is not allowed in C++. However, you can circumvent this restriction by passing two array arguments in the function, as follows:
The program is given in Listing 6.6.

Listing 6.6 ReverseArray.cpp (Reversing an Array)

```cpp
#include <iostream>
using namespace std;

void reverse(const int list[], int newList[], int size)
{
    for (int i = 0, j = size - 1; i < size; i++, j--)
    {
        newList[j] = list[i];
    }
}

void printArray(const int list[], int size)
{
    for (int i = 0; i < size; i++)
        cout << list[i] << " ";
}

int main()
{
    int size = 6;
    int list[] = {1, 2, 3, 4, 5, 6};
    int newList[6];
    reverse(list, newList, size);
    cout << "The original array: ";
    printArray(list, 6);
    cout << endl;

    cout << "The reversed array: ";
    printArray(newList, 6);
    cout << endl;

    return 0;
}
```
The reverse function (lines 7-10) uses a loop to copy the first element, second, ..., and so on in the original array to the last element, second last, ..., into the new array, as shown in the following diagram.

To invoke this function (line 25), you have to pass two arguments. The first argument is the original array whose contents are not changed in the function. The second argument is the new array whose contents are changed in the function.

6.5 Searching Arrays

**Side Remark: linear search**
**Side Remark: binary search**

Searching is the process of looking for a specific element in an array; for example, discovering whether a certain score is included in a list of scores. Searching is a common task in computer programming. There are many algorithms and data structures devoted to searching. In this section, two commonly used approaches are discussed, linear search and binary search.

6.5.1 The Linear Search Approach

The linear search approach compares the key element key sequentially with each element in the array. The function continues to do so until the key matches an element in the array or the array is exhausted without a match being found. If a match is made, the linear search returns the index of the element in the array that matches the key. If no match is found, the search returns -1. The linearSearch function in Listing 6.7 gives the solution:

**Listing 6.7 LinearSearch.cpp (Linear Search)**

```cpp
***PD: Please add line numbers in the following code***

```int```
```
t```

linearSearch(`int list[]`, int key, int arraySize)
{
    ```for```

    ```for``` (```int i = 0; i < arraySize; i++)```
        ```if``` (key == list[i])
            ```return``` i;
```}
```
```

```return``` -1;
```

Please trace the function using the following statements:
The linear search function compares the key with each element in the array. The elements in the array can be in any order. On average, the algorithm will have to compare half of the elements in an array before finding the key if it exists. Since the execution time of a linear search increases linearly as the number of array elements increases, linear search is inefficient for a large array.

6.5.2 The Binary Search Approach

Binary search is the other common search approach for a list of values. For binary search to work, the elements in the array must already be ordered. Without loss of generality, assume that the array is in ascending order. The binary search first compares the key with the element in the middle of the array. Consider the following three cases:

- If the key is less than the middle element, you only need to continue to search for the key in the first half of the array.
- If the key is equal to the middle element, the search ends with a match.
- If the key is greater than the middle element, you only need to continue to search for the key in the second half of the array.

Clearly, the binary search function eliminates half of the array after each comparison. Suppose that the array has \( n \) elements. For convenience, let \( n \) be a power of 2. After the first comparison, there are \( n/2 \) elements left for further search; after the second comparison, there are \( (n/2)/2 \) elements left for further search. After the \( k \)th comparison, there are \( n/2^k \) elements left for further search. When \( k = \log_2 n \), only one element is left in the array, and you only need one more comparison. Therefore, in the worst case, you need \( \log_2 n + 1 \) comparisons to find an element in the sorted array when using the binary search approach. For a list of 1024 (\( 2^{10} \)) elements, binary search requires only eleven comparisons in the worst case, whereas a linear search would take 1024 comparisons in the worst case.

The portion of the array being searched shrinks by half after each comparison. Let \( \text{low} \) and \( \text{high} \) denote, respectively, the first index and last index of the array that is currently being searched. Initially, \( \text{low} = 0 \) and \( \text{high} = \text{list.length}-1 \). Let \( \text{mid} \) denote the index of the middle element. So \( \text{mid} = (\text{low} + \text{high})/2 \). Figure 6.3 shows how to find key 11 in the list \{2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79\} using binary search.
Figure 6.3
Binary search eliminates half of the list from further consideration after each comparison.

The binary search returns the index of the search key if it is contained in the list. Otherwise, it returns -(insertion point + 1). The insertion point is the point at which the key would be inserted into the list. For example, the insertion point for key 5 is 2, so the binary search returns -3; the insertion point for key 51 is 7, so the binary search returns -8.

You know how the binary approach works. The task now is to implement it in C++, as shown in Listing 6.8.

Listing 6.8 BinarySearch.cpp (Binary Search)

```cpp
int binarySearch(int list[], int key, int arraySize)
{
    int low = 0;
    int high = arraySize - 1;
    while (high >= low)
    {
        int mid = (low + high) / 2;
        if (key < list[mid])
            high = mid - 1;
        else if (key == list[mid])
            return mid;
        else
            low = mid + 1;
    }
    return -low - 1;
}
```

You start to compare the key with the middle element in the list whose low index is 0 and high index is list.length-1. If key < list[mid], set the high index to mid-1; if key == list[mid], a match is found and return mid; if key > list[mid], set the low index to mid+1. Continue the search until low > high or a match is found. If low > high, return -(low+1), where low is the insertion point.

What happens if (high >= low) in line 7 is replaced by (high > low)? The search would miss a possible matching element.
Consider a list with just one element. The search would miss the element.

Does the function still work if there are duplicate elements in the list? Yes, as long as the elements are sorted in increasing order in the list. The function returns the index of one of the matching element if the element is in the list.

Please trace the program using the following statements:

```cpp
int list[] = {2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79};
int i = binarySearch(list, 2, 13); // returns 0
int j = binarySearch(list, 11, 13); // returns 4
int k = binarySearch(list, 12, 13); // returns -6
```

**NOTE**
Linear search is useful for finding an element in a small array or an unsorted array, but it is inefficient for large arrays. Binary search is more efficient, but requires that the array be pre-sorted.

### 6.6 Sorting Arrays
Sorting, like searching, is also a common task in computer programming. It would be used, for instance, if you wanted to display the grades from Listing 6.2, “Assigning Grades,” in alphabetical order. Many different algorithms have been developed for sorting. This section introduces two simple, intuitive sorting algorithms: selection sort and insertion sort.

#### 6.6.1 Selection Sort
Suppose that you want to sort a list in ascending order. Selection sort finds the largest number in the list and places it last. It then finds the largest number remaining and places it next to last, and so on until the list contains only a single number. Figure 6.4 shows how to sort the list \{2, 9, 5, 4, 8, 1, 6\} using selection sort.

***Same as Fig5.10 in intro5e p191***
Select 9 (the largest) and swap it with 6 (the last) in the list

Select 8 (the largest) and swap it with 1 (the last) in the remaining list

Select 6 (the largest) and swap it with 1 (the last) in the remaining list

Select 5 (the largest) and swap it with 4 (the last) in the remaining list

4 is the largest and last in the list. No swap is necessary

Select 2 (the largest) and swap it with 1 (the last) in the remaining list

Since there is only one number remaining in the list, sort is completed

The number 9 is now in the correct position and thus no longer needs to be considered.

The number 8 is now in the correct position and thus no longer needs to be considered.

The number 6 is now in the correct position and thus no longer needs to be considered.

The number 5 is now in the correct position and thus no longer needs to be considered.

The number 4 is now in the correct position and thus no longer needs to be considered.

The number 2 is now in the correct position and thus no longer needs to be considered.

**Figure 6.4**

Selection sort repeatedly selects the largest number and swaps it with the last number in the list.

You know how the selection sort approach works. The task now is to implement it in C++. For beginners, it is difficult to develop a complete solution on the first attempt. You may start to write the code for the first iteration to find the largest element in the list and swap it with the last element, and then observe what would be different for the second iteration, the third, and so on. The insight this gives will enable you to write a loop that generalizes all the iterations.

***6E: Add the labels same as in intro5e if the following are in the same page.***

The solution can be described as follows:

```cpp
for (int i = list.length - 1; i > 0; i--)
    select the largest element in list[0..i];
    swap the largest with list[i], if necessary;
    // list[i] is in its correct position.
    // The next iteration apply on list[0..i-1]
```

Listing 6.9 implements the solution.
void selectionSort(double list[], int arraySize)
{
    for (int i = arraySize - 1; i >= 1; i--)
    {
        // Find the maximum in the list[0..i]
        double currentMax = list[0];
        int currentMaxIndex = 0;
        for (int j = 1; j <= i; j++)
        {
            if (currentMax < list[j])
            {
                currentMax = list[j];
                currentMaxIndex = j;
            }
        }
        // Swap list[i] with list[currentMaxIndex] if necessary;
        if (currentMaxIndex != i)
        {
            list[currentMaxIndex] = list[i];
            list[i] = currentMax;
        }
    }
}

The selectionSort(double list[]) function sorts any array of double elements. The function is implemented with a nested for loop. The outer loop (with the loop control variable i) (line 4) is iterated in order to find the largest element in the list, which ranges from list[0] to list[i], and exchange it with the current last element, list[i].

The variable i is initially list.length-1. After each iteration of the outer loop, list[i] is in the right place. Eventually, all the elements are put in the right place; therefore, the whole list is sorted.

Please trace the function with the following statements:

double list[] = {1, 9, 4.5, 6.6, 5.7, -4.5};
selectionSort(list, 6);

6.6.2 (Optional) Insertion Sort
Suppose that you want to sort a list in ascending order. The insertion-sort algorithm sorts a list of values by repeatedly inserting a new element into a sorted sublist until the whole list is sorted. Figure 6.5 shows how to sort the list {2, 9, 5, 4, 8, 1, 6} using insertion sort.
Step 1: Initially, the sorted sublist contains the first element in the list. Insert 9 to the sublist.

Step 2: The sorted sublist is {2, 9}. Insert 5 to the sublist.

Step 3: The sorted sublist is {2, 5, 9}. Insert 4 to the sublist.

Step 4: The sorted sublist is {2, 4, 5, 9}. Insert 8 to the sublist.

Step 5: The sorted sublist is {2, 4, 5, 8, 9}. Insert 1 to the sublist.

Step 6: The sorted sublist is {1, 2, 4, 5, 8, 9}. Insert 6 to the sublist.

Step 7: The entire list is now sorted.

**Figure 6.5**

Insertion sort repeatedly inserts a new element into a sorted sublist.

The algorithm can be described as follows:

```java
for (int i = 1; i < list.length; i++) {
    insert list[i] into a sorted sublist list[0..i-1] so that list[0..i] is sorted.
}
```

To insert list[i] into list[0..i-1], save list[i] into a temporary variable, say currentElement. Move list[i-1] to list[i] if list[i-1] > currentElement, move list[i-2] to list[i-1] if list[i-2] > currentElement, and so on, until list[i-k] <= currentElement. Assign currentElement to list[i-k+1]. For example, to insert 4 into {2, 5, 9} in Step 3 in Figure 6.6, move list[2] (9) to list[3] since 9 > 4, move list[1] (5) to list[2] since 5 > 4. Finally move currentElement (4) to list[1], as shown in Figure 6.14.

**Figure 6.6**

A new element is inserted into a sorted sublist.
The algorithm can be expanded and implemented in Listing 6.10.

**Listing 6.10 InsertionSort.cpp (Sorting Numbers)**

```cpp
void insertionSort(double list[], int arraySize)
{
    for (int i = 1; i < arraySize; i++)
    {
        /* insert list[i] into a sorted sublist list[0..i-1] so that
           list[0..i] is sorted. */
        double currentElement = list[i];
        int k;
        for (k = i - 1; k >= 0 && list[k] > currentElement; k--)
        {
            list[k + 1] = list[k];
        }
        // Insert the current element into list[k+1]
        list[k + 1] = currentElement;
    }
}
```

The `insertionSort(double list[])` function sorts any array of double elements. The function is implemented with a nested for loop. The outer loop (with the loop control variable `i`) (line 4) is iterated in order to obtain a sorted sublist, which ranges from `list[0]` to `list[i]`. The inner loop (with the loop control variable `k`) inserts `list[i]` into the sublist from `list[0]` to `list[i-1]`.

Please trace the function with the following statements:

```cpp
double list[] = {1, 9, 4.5, 6.6, 5.7, -4.5};
insertionSort(list, 6);
```

### 6.7 Two-Dimensional Arrays

Thus far, you have used one-dimensional arrays to model linear collections of elements. You can use a two-dimensional array to represent a matrix or a table.

#### 6.7.1 Declaring Two-Dimensional Arrays

Here is the syntax for declaring a two-dimensional array:

```cpp
dataType arrayName[rowSize][columnSize];
```

As an example, here is how you would declare a two-dimensional array variable `matrix` of int values:

```cpp
int matrix[5][5];
```

Two subscripts are used in a two-dimensional array, one for the row, and the other for the column. As in a one-dimensional array, the index for each subscript is of the `int` type and starts from 0, as shown in Figure 6.7(a).
The index of each subscript of a two-dimensional array is an int value starting from 0.

To assign the value 7 to a specific element at row 2 and column 1, as shown in Figure 6.7(b), you can use the following:

```
matrix[2][1] = 7;
```

**CAUTION**

It is a common mistake to use `matrix[2, 1]` to access the element at row 2 and column 1. In C++, each subscript must be enclosed in a pair of square brackets.

You can also use an array initializer to declare, create, and initialize a two-dimensional array. For example, the following code in (a) creates an array with the specified initial values, as shown in Figure 6.7(c). This is equivalent to the code in (b).

```
int array[4][3] = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9},
    {10, 11, 12}
};
```

```
int array[4][3];
array[0][0] = 1; array[0][1] = 2; array[0][2] = 3;
array[1][0] = 4; array[1][1] = 5; array[1][2] = 6;
array[2][0] = 7; array[2][1] = 8; array[2][2] = 9;
array[3][0] = 10; array[3][1] = 11; array[3][2] = 12;
```

6.7.2 Processing Two-Dimensional Arrays

Suppose an array `matrix` is declared as follows:

```
const int rowSize = 10;
const int columnSize = 10;
int matrix[rowSize][columnSize];
```

Here are some examples of processing two-dimensional arrays:

1. (Initializing arrays with random values) The following loop initializes the array with random values between 0 and 99:

```
for (int row = 0; row < rowSize; row++)
    for (int column = 0; column < columnSize; column++)
        matrix[row][column] = rand() % 100;
```
2. (Printing arrays) To print a two-dimensional array, you have to print each element in the array using a loop like the following:

```cpp
for (int row = 0; row < rowSize; row++)
    for (int column = 0; column < columnSize; column++)
        cout << matrix[row][column] << " ";
    cout << endl;
```

3. (Summing all elements) Use a variable named total to store the sum. Initially total is 0. Add each element in the array to total using a loop like this:

```cpp
int total = 0;
for (int row = 0; row < rowSize; row++)
    for (int column = 0; column < columnSize; column++)
        total += matrix[row][column];
```

4. (Summing elements by column) For each column, use a variable named total to store its sum. Add each element in the column to total using a loop like this:

```cpp
for (int column = 0; column < columnSize; column++)
    int total = 0;
    for (int row = 0; row < rowSize; row++)
        total += matrix[row][column];
    cout << "Sum for column " << column << " is " << total << endl;
```

6. (Which row has the largest sum?) Use variables maxRow and indexOfMaxRow to track the largest sum and index of the row. For each row, compute its sum and update maxRow and indexOfMaxRow if the new sum is greater.

```cpp
int maxRow = 0;
int indexOfMaxRow = 0;
// Get sum of the first row in maxRow
for (int column = 0; column < columnSize; column++)
    maxRow += matrix[0][column];
for (int row = 1; row < rowSize; row++)
    int totalOfThisRow = 0;
    for (int column = 0; column < columnSize; column++)
        totalOfThisRow += matrix[row][column];
    if (totalOfThisRow > maxRow)
        maxRow = totalOfThisRow;
        indexOfMaxRow = row;
    
cout << "Row " << indexOfMaxRow << " has the maximum sum" << " of " << maxRow << endl;
```

6.7.3 Example: Grading a Multiple-Choice Test
This example presents a program that grades multiple-choice tests. Suppose there are eight students and ten questions, and the answers are stored in a two-dimensional array. Each row records a student’s answers to the questions. For example, the following array stores the test.

***PD: Same as in intro5e p199 (unnumbered figure)***

Students’ Answers to the Questions:

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>Student 0</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>C</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>A</td>
<td>D</td>
</tr>
<tr>
<td>Student 1</td>
<td>D</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>A</td>
</tr>
<tr>
<td>Student 2</td>
<td>E</td>
<td>D</td>
<td>A</td>
<td>C</td>
<td>B</td>
<td>E</td>
<td>E</td>
<td>A</td>
<td>D</td>
</tr>
<tr>
<td>Student 3</td>
<td>C</td>
<td>B</td>
<td>A</td>
<td>E</td>
<td>D</td>
<td>C</td>
<td>E</td>
<td>E</td>
<td>A</td>
</tr>
<tr>
<td>Student 4</td>
<td>A</td>
<td>B</td>
<td>D</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>E</td>
<td>A</td>
<td>D</td>
</tr>
<tr>
<td>Student 5</td>
<td>B</td>
<td>B</td>
<td>E</td>
<td>C</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>E</td>
<td>A</td>
</tr>
<tr>
<td>Student 6</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>C</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>E</td>
<td>A</td>
</tr>
<tr>
<td>Student 7</td>
<td>E</td>
<td>B</td>
<td>E</td>
<td>C</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>E</td>
<td>A</td>
</tr>
</tbody>
</table>

The key is stored in a one-dimensional array, as follows:

Key to the Questions:

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>Key</td>
<td>D</td>
<td>B</td>
<td>D</td>
<td>C</td>
<td>C</td>
<td>D</td>
<td>A</td>
<td>E</td>
<td>A</td>
</tr>
</tbody>
</table>

Your program grades the test and displays the result. The program compares each student’s answers with the key, counts the number of correct answers, and displays it. Listing 6.11 gives the program.

Listing 6.11 GradeExam.cpp (Grading Exams)

```cpp
#include <iostream>
using namespace std;

int main()
{
    const int NUMBER_OF_STUDENTS = 8;
    const int NUMBER_OF_QUESTIONS = 10;

    // Students' answers to the questions
    char answers[NUMBER_OF_STUDENTS][NUMBER_OF_QUESTIONS] = {
        {'A', 'B', 'A', 'C', 'C', 'D', 'E', 'A', 'D'},
        {'D', 'B', 'A', 'B', 'C', 'A', 'E', 'A', 'D'},
        {'E', 'D', 'A', 'C', 'B', 'E', 'E', 'A', 'D'},
        {'C', 'B', 'A', 'E', 'D', 'C', 'E', 'E', 'A'},
        {'A', 'B', 'D', 'C', 'C', 'D', 'E', 'A', 'D'},
        {'B', 'B', 'E', 'C', 'C', 'D', 'E', 'E', 'A'},
        {'B', 'B', 'A', 'C', 'C', 'D', 'E', 'E', 'A'},
        {'E', 'B', 'E', 'C', 'C', 'D', 'E', 'E', 'A'}
    };
```
The statement in lines 10–20 declares and initializes a two-dimensional array of characters.

The statement in line 23 declares and initializes an array of char values.

Each row in the array answers stores a student’s answer, which is graded by comparing it with the key in the array keys. The result is displayed immediately after a student’s answer is graded.

6.7.4 (Optional) Example: Computing Taxes

Listing 5.4, ComputeTaxWithFunction.cpp, simplified Listing 3.4, ComputeTaxWithSelectionStatement.cpp. Listing 5.4 can be further improved using arrays. For each filing status, there are six tax rates. Each rate is applied to a certain amount of taxable income. For example, from the taxable income of
$400,000 for a single filer, $6,000 is taxed at 10%, (27950 – 6000) at 15%, (67700 – 27950) at 27%, (141250 – 67700) at 30%, (307050 – 141250) at 35%, and (400000 – 307050) at 38.6%. The six rates are the same for all filing statuses, which can be represented in the following array:

```csharp
double rates[] = {0.10, 0.15, 0.27, 0.30, 0.35, 0.386};
```

The brackets for each rate for all the filing statuses can be represented in a two-dimensional array as follows:

```csharp
int brackets[4][5] = 
{ 
    {6000, 27950, 67700, 141250, 307050}, // Single filer
    {12000, 46700, 112850, 171950, 307050}, // married jointly
    {6000, 23350, 56425, 85975, 153525}, // married separately
    {10000, 37450, 96700, 156600, 307050} // head of household
};
```

Suppose the taxable income is $400,000 for single filers, the tax can be computed as follows:

```
brackets[0][0] * rates[0] +
(brackets[0][1] – brackets[0][0]) * rates[1] +
(brackets[0][2] – brackets[0][1]) * rates[2] +
(brackets[0][3] – brackets[0][2]) * rates[3] +
(brackets[0][4] – brackets[0][3]) * rates[4] +
(400000 – brackets[0][4]) * rates[5]
```

Listing 6.12 gives the solution to the program.

```
#include <iostream>
#include <iomanip>
using namespace std;

double computeTax(int, double);

int main()
{
    // Prompt the user to enter filing status
    cout << "(0-single filer, 1-married jointly,\n" <<
        "2-married separately, 3-head of household)\n" <<
        "Enter the filing status: ";
    int status;
    cin >> status;

    // Prompt the user to enter taxable income
```
cout << "Enter the taxable income: ";
double income;
cin >> income;

// Compute and display the result
cout << "Tax is " << fixed << setprecision(2) << computeTax(status, income);
return 0;

// double computeTax(int status, double income)

const int BRACKET_SIZE = 6;
double rates[] = {0.10, 0.15, 0.27, 0.30, 0.35, 0.386};

int brackets[4][5] = {
    {6000, 27950, 67700, 141250, 307050}, // Single filer
    {12000, 46700, 112850, 171950, 307050}, // Married jointly
    {6000, 23350, 56425, 85975, 153525}, // Married separately
    {10000, 37450, 96700, 156600, 307050} // Head of household
};

double tax = 0; // Tax to be computed

// Compute tax in the first bracket
if (income <= brackets[status][0])
    return tax = income * rates[0]; // Done
else
    tax = brackets[status][0] * rates[0];

// Compute tax in the 2nd, 3rd, 4th, and 5th brackets, if needed
for (int i = 1; i < BRACKET_SIZE; i++)
{
    if (income > brackets[status][i])
        tax += (brackets[status][i] - brackets[status][i - 1]) * rates[i];
    else
        tax += (income - brackets[status][i - 1]) * rates[i];
}
// Compute tax in the last (i.e., 6th) bracket
return tax += (income - brackets[status][4]) * rates[5];

<Output>
(0-single filer, 1-married jointly,
2-married separately, 3-head of household
Enter the filing status: 0
Enter the taxable income: 12534
Tax is 1580.10
<End Output>
The computeTax function computes the tax for the taxable income of a given filing status. The tax for the first bracket (0 to \( \text{brackets[status][0]} \)) is computed in lines 44-47. The taxes for the second, third, fourth, and fifth brackets are computed in the loop in lines 50-60. The tax for the last bracket is computed in line 63.

6.8 (Optional) Multidimensional Arrays

In the preceding section, you used a two-dimensional array to represent a matrix or a table. Occasionally, you will need to represent \( n \)-dimensional data structures. In C++, you can create \( n \)-dimensional arrays for any integer \( n \).

The way to declare two-dimensional array can be generalized to declare \( n \)-dimensional array for \( n \geq 3 \). For example, the following syntax declares a three-dimensional array \( \text{scores} \).

```cpp
double scores[10][5][2];
```

6.8.1 Example: Computing Student Scores

Listing 6.13 gives a program that calculates the total score for the students in a class. Suppose the scores are stored in a three-dimensional array named \( \text{scores} \). The first index in \( \text{scores} \) refers to a student, the second refers to an exam, and the third refers to a part of the exam. Suppose there are seven students, five exams, and each exam has two parts: a multiple-choice part and a programming part. \( \text{scores[i][j][0]} \) represents the score on the multiple-choice part for the \( i \)'s student on the \( j \)'s exam. \( \text{scores[i][j][1]} \) represents the score on the programming part for the \( i \)'s student on the \( j \)'s exam. The program processes the \( \text{scores} \) array for all the students. For each student, it adds the two scores from all exams to \text{totalScore} and displays \text{totalScore}. Your program displays the total score for each student, as shown in the sample output.

Listing 6.13 TotalScore.cpp (Computing Student Scores)

```cpp
#include <iostream>
using namespace std;

double computeTax(int, double);

int main()
{
   const int NUMBER_OF_STUDENTS = 7;
   const int NUMBER_OF_EXAMS = 5;
   const int NUMBER_OF_PARTS_IN_EXAM = 2;

   double scores;
   // [NUMBER_OF_STUDENTS][NUMBER_OF_EXAMS][NUMBER_OF_PARTS_IN_EXAM]
```
To understand this example, it is essential to know how data in the three-dimensional array are interpreted. scores[0] is a two-dimensional array that stores all the exam scores for the first student. scores[0][0] is {7.5, 20.5}, a one-dimensional array, which stores two scores for two parts of the first student’s first exam. scores[0][0][0] is 7.5, which is the score for the first part of the first student’s first exam. scores[5] is a two-dimensional array that stores all the exam scores for the sixth student. scores[5][4] is {16, 6.5}, a one-dimensional array, which stores two scores for two parts of the sixth student’s fifth exam. scores[5][4][1] is 6.5, which is the score for the second part of the sixth student’s fifth exam.

The statement in lines 12–22 declares and initializes a three-dimensional array of double values.
The scores for each student are added in lines 16–18, and the result is displayed in lines 29–30. The for loop in line 25 process the scores for all the students.

Key Terms
**PD: Please place terms in two columns same as in intro5e.**

- array 176
- array initializer 178
- binary search 193
- const array
- index 177
- indexed variable 177
- insertion sort 191
- linear search 193
- multidimensional array 196
- selection sort 191

Chapter Summary

- An array represents a list of value of the same type. An array is created using the syntax `dataType arrayName[size].`
- Each element in the array is represented using the syntax `arrayName[index].` An index must be an integer or an integer expression. Array index is 0-based, meaning that the index for the first element is 0.
- Programmers often mistakenly reference the first element in an array with index 1, so that the index of the tenth element becomes 10. This is called the index off-by-one error.
- C++ has a shorthand notation, known as the array initializer, which combines creating an array and initializing in one statement using the syntax: `dataType arrayName[] = {value0, value1, ..., valuek}.`
- When you pass an array argument to a function, you are actually passing the reference of the array; that is, the called function can modify the elements in the caller’s original array.
- When you pass an array argument to a function, often you should also pass the size in another argument. So the function knows how many elements are in the functions.
- You can specify const array parameters to prevent arrays from being modified accidentally.
- You can use arrays of arrays to form multidimensional arrays. For example, a two-dimensional array is declared as an array of arrays using the syntax `dataType arrayName[size1][size2].`

Review Questions
Section 6.2 Array Basics

6.1
How do you declare and create an array?

6.2
How do you access elements of an array? Can you copy an array \( \text{a} \) to \( \text{b} \) using \( \text{b} = \text{a} \)?

6.3
Is memory allocated when an array is declared? What is the printout of the following code?

```cpp
int numbers[30];
cout << "number[0] is " << numbers[0];
cout << "number[30] is " << numbers[30];
```

6.4
Indicate true or false for the following statements:

- Every element in an array has the same type.
- The array size is fixed after it is created.
- The array size used to declare an array must be a constant expression.

6.5
Which of the following statements are valid array declarations?

```cpp
double d[30];
char r[30];
int i[] = {3, 4, 3, 2};
float f[] = {2.3, 4.5, 6.6};
```

6.6
What is the array index type? What is the lowest index?

6.7
What is the representation of the third element in an array named \( a \)?

6.8
What happens when your program attempts to access an array element with an invalid index?

6.9
Identify and fix the errors in the following code:

```cpp
***PD: Please add line numbers in the following code***
int main()
{
    double r[100];
    for (int i = 0; i < 100; i++)
        r[i] = rand() % 100;
}
```

Section 6.3 Passing Arrays to Functions

6.10
When an array is passed to a function, a new array is created and passed to the function. Is this true?

6.11
Show the output of the following two programs:
6.12
How do you prevent the array from being modified accidentally in a function?

Section 6.5 Searching Arrays
6.13
Use Figure 6.3 as an example to show how to apply the binary search approach to search for key 10 and key 12 in list {2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79}.

Section 6.6 Sorting Arrays
6.14
Use Figure 6.4 as an example to show how to apply the selection sort approach to sort {3.4, 5, 3, 3.5, 2.2, 1.9, 2}.

6.15
Use Figure 6.5 as an example to show how to apply the insertion sort approach to sort {3.4, 5, 3, 3.5, 2.2, 1.9, 2}.

6.16
How do you modify the selectionSort function in Listing 6.9 to sort numbers in decreasing order?

6.17
How do you modify the insertionSort function in Listing 6.10 to sort numbers in decreasing order?

Section 6.7 Two-dimensional Arrays
6.18
Declare and create a 4 × 5 int matrix.

6.19
What is the output of the following code?
```cpp
int array[5][6];
int x[] = {1, 2};
array[0][1] = x[1];
cout << "array[0][1] is " << array[0][1];
```
Which of the following statements are valid array declarations?

- `int r[2];`
- `int x[];`
- `int y[3][];`

**Programming Exercises**

**Section 6.2 Array Basics**

6.1  
*(Analyzing input)* Write a program that reads ten numbers, computes their average, and finds out how many numbers are above the average.

6.2  
*(Alternative solution to Listing 6.1, “Testing Arrays”)* The solution of Listing 6.1 counts the occurrences of the largest number by comparing each number with the largest. So you have to use an array to store all the numbers. Another way to solve the problem is to maintain two variables, `max` and `count`. `max` stores the current max number, and `count` stores its occurrences. Initially, assign the first number to `max` and 1 to `count`. Compare each subsequent number with `max`. If the number is greater than `max`, assign it to `max` and reset `count` to 1. If the number is equal to `max`, increment `count` by 1. Use this approach to rewrite Listing 6.1.

6.3  
*(Reversing the numbers entered)* Write a program that reads ten integers and displays them in reverse order in which they were read.

6.4  
*(Analyzing scores)* Write a program that reads an unspecified number of scores and determines how many scores are above or equal to the average and how many scores are below the average. Enter a negative number to signify the end of the input. Assume that the maximum number of scores is 100.

6.5**  
*(Printing distinct numbers)* Write a program that reads in ten numbers and displays distinct numbers (i.e., if a number appears multiple times, it is displayed only once). Hint: Read a number and store it to an array if it is new. If the number is already in the array, discard it. After the input, the array contains the distinct numbers.

6.6*  
*(Revising Listing 4.10 “Printing Prime Numbers”)* Listing 4.10 determines whether a number `n` is prime by checking whether `2, 3, 4, 5, 6, ..., n/2` is a divisor. If a
divisor is found, n is not prime. A more efficient approach to determine whether n is prime is to check whether any of the prime numbers less than or equal to $\sqrt{n}$ can divide n evenly. If not, n is prime.

Rewrite Listing 4.10 to display the first fifty prime numbers using this approach. You need to use an array to store the prime numbers and later use them to check whether they are possible divisors for n.

6.7*
(Counting single digits) Write a program that generates one hundred random integers between 0 and 9 and displays the count for each number. Hint: Use \texttt{rand()} \% 10 to generate a random integer between 0 and 9. Use an array of ten integers, say \texttt{counts}, to store the counts for the number of 0’s, 1’s, ..., 9’s.

Sections 6.3–6.4
6.8
(Averaging an array) Write two overloaded functions that return the average of an array with the following headers:

```c
int average(int array[]);
double average(double array[]);
```

Use \{1, 2, 3, 4, 5, 6\} and \{6.0, 4.4, 1.9, 2.9, 3.4, 3.5\} to test the functions.

6.9
(Finding the smallest element) Write a function that finds the smallest element in an array of integers. Use \{1, 2, 4, 5, 10, 100, 2, -22\} to test the function.

6.10
(Finding the index of the smallest element) Write a function that returns the index of the smallest element in an array of integers. If there are more than one such elements, return the smallest index. Use \{1, 2, 4, 5, 10, 100, 2, -22\} to test the function.

6.11* (Computing deviation) Exercise 5.21 computes the standard deviation of numbers. This exercise uses a different but equivalent formula to compute the standard deviation of n numbers.

***PD: Same as on P207 in intro5e

\[
\text{mean} = \frac{\sum_{i=1}^{n} x_i}{n} = \frac{x_1 + x_2 + ... + x_n}{n}, \quad \text{deviation} = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \text{mean})^2}{n-1}}
\]

To compute deviation with this formula, you have to store the individual numbers using an array, so that they can be used after the mean is obtained. Use \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\} to test the function.

6.12* (Reversing an array) Write a function that reverses an array using the following header:
void reverse(int source[], int target[])

Sections 6.5-6.6

6.13
(Finding the sales amount) Rewrite Listing 4.6, FindSalesAmount.cpp, using the binary search approach. Since the sales amount is between 1 and COMMISSION_SOUGHT/0.08, you can use a binary search to improve Listing 4.6.

6.14
(Timing execution) Write a program that randomly generates an array of 100000 integers and a key. Estimate the execution time of invoking the linearSearch function in Listing 6.7. Sort the array and estimate the execution time of invoking the binarySearch function in Listing 6.8. You may use the following code template to obtain the execution time:

```cpp
long startTime = time(0);
perform the task;
long endTime = time(0);
long executionTime = endTime - startTime;
```

6.15*
(Revising selection sort) In §6.5, you used selection sort to sort an array. The selection sort function repeatedly finds the largest number in the current array and swaps it with the last number in the array. Rewrite this example by finding the smallest number and swapping it with the first number in the array.

6.16**
(Bubble sort) Write a sort function that uses the bubble-sort algorithm. The bubble-sort algorithm makes several passes through the array. On each pass, successive neighboring pairs are compared. If a pair is in decreasing order, its values are swapped; otherwise, the values remain unchanged. The technique is called a bubble sort or sinking sort because the smaller values gradually "bubble" their way to the top and the larger values sink to the bottom.

The algorithm can be described as follows:

```cpp
bool changed = true;
do {
    changed = false;
    for (int j = 0; j < listSize - 1; j++)
    if (list[j] > list[j + 1])
        swap list[j] with list[j + 1];
    changed = true;
    } while (changed);
```

Clearly, the list is in increasing order when the loop terminates. It is easy to show that the do loop executes at most listSize - 1 times.
Use \{6.0, 4.4, 1.9, 2.9, 3.4, 2.9, 3.5\} to test the function.

6.17**
(Sorting students) Write a program that prompts the user to enter the number of students, and student names and their scores, and prints student names in decreasing order of their scores.

Section 6.7 Two-dimensional Arrays

6.18*
(Summing all the numbers in a matrix) Write a function that sums all the integers in a matrix of integers. Use \{{1, 2, 4, 5}, \{6, 7, 8, 9\}, \{10, 11, 12, 13\}, \{14, 15, 16, 17\}\} to test the function.

6.19*
(Summing the major diagonal in a matrix) Write a function that sums all the integers in the major diagonal in a matrix of integers. Use \{{1, 2, 4, 5}, \{6, 7, 8, 9\}, \{10, 11, 12, 13\}, \{14, 15, 16, 17\}\} to test the function.

6.20*
(Sorting students on grades) Rewrite Listing 6.11, GradeExam.cpp, to display the students in increasing order of the number of correct answers.

6.21*
(Computing the weekly hours for each employee) Suppose the weekly hours for all employees are stored in a two-dimensional array. Each row records an employee’s seven-day work hours with seven columns. For example, the following array stores the work hours for eight employees. Write a program that displays employees and their total hours in decreasing order of the total hours.

<table>
<thead>
<tr>
<th>Su</th>
<th>M</th>
<th>T</th>
<th>W</th>
<th>H</th>
<th>F</th>
<th>Sa</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee 0</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>Employee 1</td>
<td>7</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Employee 2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Employee 3</td>
<td>9</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Employee 4</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>Employee 5</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Employee 6</td>
<td>3</td>
<td>7</td>
<td>4</td>
<td>8</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>Employee 7</td>
<td>6</td>
<td>3</td>
<td>5</td>
<td>9</td>
<td>2</td>
<td>7</td>
</tr>
</tbody>
</table>

6.22
(Adding two matrices) Write a function to add two matrices. The header of the function is as follows:

\[
\text{void addMatrix(int a[[]], int b[][], int c[][])}
\]

In order to be added, two matrices must have the same dimensions and the same or compatible types of elements. As shown below, two matrices are added by adding the two elements of the arrays with the same index:
6.23**

(Multiplying two matrices) Write a function to multiply two matrices. The header of the function is as follows:

```c
void multiplyMatrix(int a[][], int b[][], int c[][[]])
```

To multiply matrix `a` by matrix `b`, the number of columns in `a` must be the same as the number of rows in `b`, and the two matrices must have elements of the same or compatible types. Let `c` be the result of the multiplication, and `a`, `b`, and `c` are denoted as follows:

\[
\begin{bmatrix}
a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\
a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\
a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\
a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\
a_{51} & a_{52} & a_{53} & a_{54} & a_{55}
\end{bmatrix}\times
\begin{bmatrix}
b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\
b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\
b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \\
b_{41} & b_{42} & b_{43} & b_{44} & b_{45} \\
b_{51} & b_{52} & b_{53} & b_{54} & b_{55}
\end{bmatrix}=
\begin{bmatrix}
c_{11} & c_{12} & c_{13} & c_{14} & c_{15} \\
c_{21} & c_{22} & c_{23} & c_{24} & c_{25} \\
c_{31} & c_{32} & c_{33} & c_{34} & c_{35} \\
c_{41} & c_{42} & c_{43} & c_{44} & c_{45} \\
c_{51} & c_{52} & c_{53} & c_{54} & c_{55}
\end{bmatrix}
\]

Where \( c_{ij} = a_{i1} \times b_{1j} + a_{i2} \times b_{2j} + a_{i3} \times b_{3j} + a_{i4} \times b_{4j} + a_{i5} \times b_{5j} \).

6.24*

(TicTacToe board) Write a program that randomly fills in 0s and 1s into a TicTacToe board, prints the board, and finds the rows, columns, or diagonals with all 0s or 1s. Use a two-dimensional array to represent a TicTacToe board. Here is a sample run of the program:

```
001
001
001
All 0's on row 0
All 1's on row 2
All 1's on column 2
```

6.25**

(Checker board) Write a program that randomly fills in 0s and 1s into an 8 \times 8 checker board, prints the board, and finds the rows, columns, or diagonals with all 0s or 1s. Use a two-dimensional array to represent a checker board. Here is a sample run of the program:

```
10101000
10101000
11100111
10101000
10101000
00100000
00100000
00100000
```
6.26***
(Playing a TicTacToe game) In a game of TicTacToe, two players take turns marking an available cell in a $3 \times 3$ grid with their respective tokens (either X or O). When one player has placed three tokens in a horizontal, vertical, or diagonal row on the grid, the game is over and that player has won. A draw (no winner) occurs when all the cells on the grid have been filled with tokens and neither player has achieved a win. Create a program for playing TicTacToe, as follows:

1. The program prompts the first player to enter an X token, and then prompts the second player to enter an O token. Whenever a token is entered, the program refreshes the board and determines the status of the game (win, draw, or unfinished).

2. To place a token, prompt the user to enter the row and the column for the token.

6.27***
(LCM) Write a program that prompts the user to enter two integers and finds their least common multiple. The least common multiple (LCM) of two numbers is the smallest number that is a multiple of both. For example, the LCM for 8 and 12 is 24, for 15 and 25 is 75, and for 120 and 150 is 600. There are many ways to find the LCM. In this exercise, you will use the approach described as follows:

To find the LCM of two numbers, first create a prime factor table for each number. The first column of the table consists of all the prime factors and the second column tracks the occurrence of the corresponding prime factor in the number. For example, the prime factors for 120 are 2, 2, 2, 3, 5, so the prime factor table for number 120 is shown as follows:

<table>
<thead>
<tr>
<th>prime factors for 120</th>
<th># of occurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

The prime factors for 150 are 2, 3, 5, 5, so the prime factor table for number 150 is shown as follows:

<table>
<thead>
<tr>
<th>prime factors for 120</th>
<th># of occurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

The LCM of the two numbers consists of the factors with the largest occurrence in the two numbers. So the LCM for 120 and 150 is $2 \times 2 \times 2 \times 3 \times 5 \times 5$, where 2 appears three times in 120, 3 one time in 120, and 5 two times in 150.
Hint: The prime factor table can be represented using a two-dimensional array. Write a function named `getPrimeFactors(int number)` that returns a two-dimensional array for the prime factor table.