# Introduction to Programming in C++
# Seventh Edition

# Objectives

- Open Visual Studio
- Get your book

- Declare and initialize a one-dimensional array
- Enter data into a one-dimensional array
- Display the contents of a one-dimensional array
- Pass a one-dimensional array to a function
- Calculate the total and average of the values in a one-dimensional array

# C++

- Complete project 5 last day
- Open Visual Studio
- Get your book

- Arrays and Multi Dimensional Arrays
- Friday We will be put into groups to begin the final project.

# Objectives (cont'd.)

- Search a one-dimensional array
- Access an individual element in a one-dimensional array
- Find the highest value in a one-dimensional array
- Explain the bubble sort algorithm
- Use parallel one-dimensional arrays

# Arrays

- A **simple variable** (also called a **scalar variable**) is unrelated to any other variable in memory
- Sometimes variables are related to each other
- Easier and more efficient to treat these as a group
- A group of related variables with the same data type is referred to as an **array**

# Arrays (cont'd.)

- Storing data in an array increases the efficiency of a program
  - Data can be accessed from internal memory faster than it can be from a file on a disk
  - Once stored in an array, data can be used as many times as necessary without having to enter it again
- Variables in an array can be used like any other
- Most commonly used arrays in business applications are one-dimensional and two-dimensional

# One-Dimensional Arrays

- Variables in an array are stored in consecutive locations in computer's internal memory
- Each variable in an array has the same name and data type
- You distinguish one variable in a **one-dimensional array** from another variable in the same array by using a unique integer, called a subscript
- A **subscript** indicates a variable's position in the array and is assigned by the computer when the array is created

# One-Dimensional Arrays (cont'd.)

- First variable in a one-dimensional array is assigned a subscript of 0, second a subscript of 1, and so on
- You refer to a variable in an array by the array's name immediately followed by a subscript enclosed in square brackets (e.g., *sales[0]*)
- The last subscript in an array is always one less than the total number of variables in the array, since the subscripts begin at 0

# One-Dimensional Arrays (cont'd.)

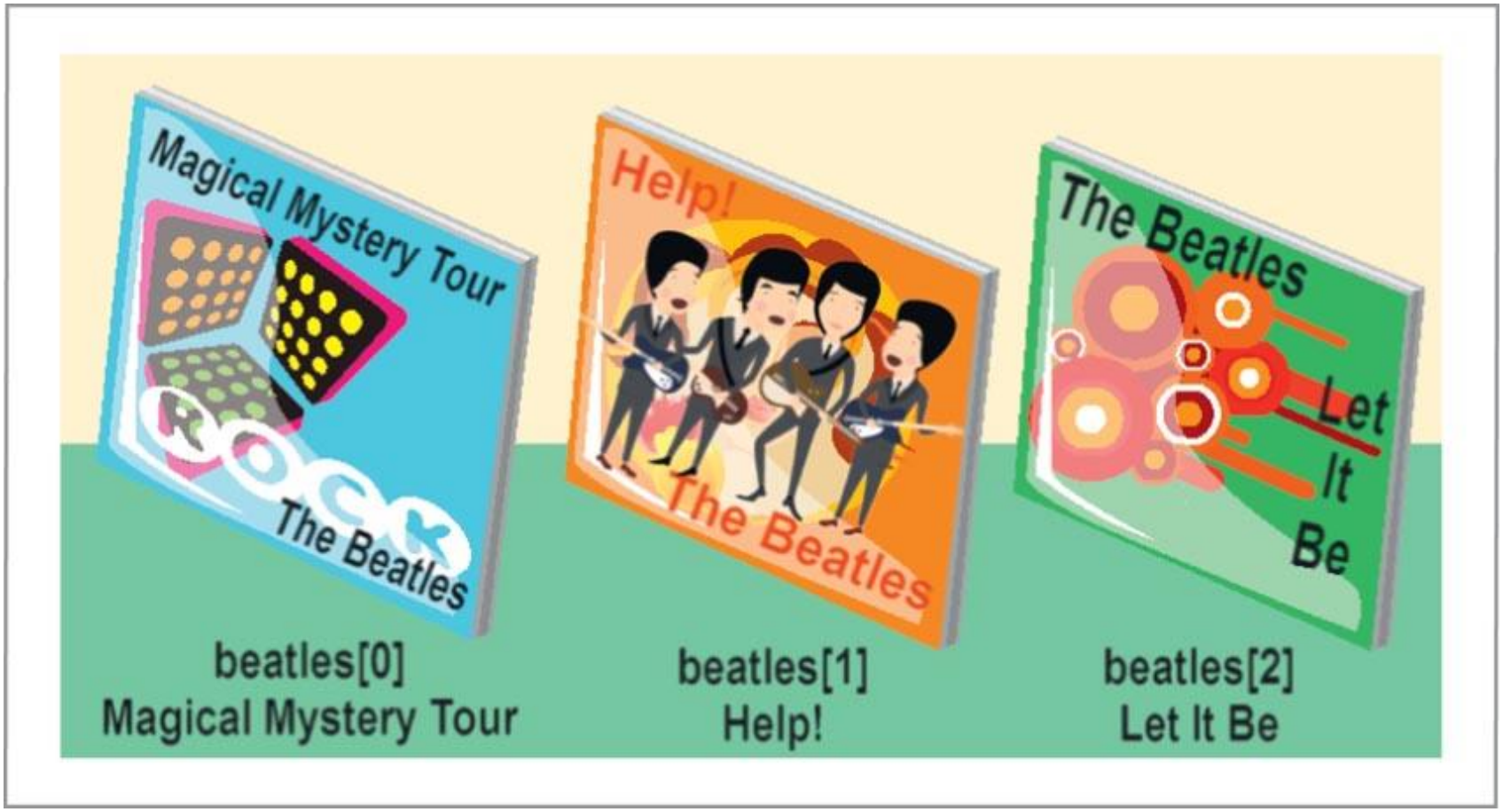



Figure 11-1 Illustration of naming conventions for one-dimensional `beatles` array

# One-Dimensional Arrays (cont'd.)

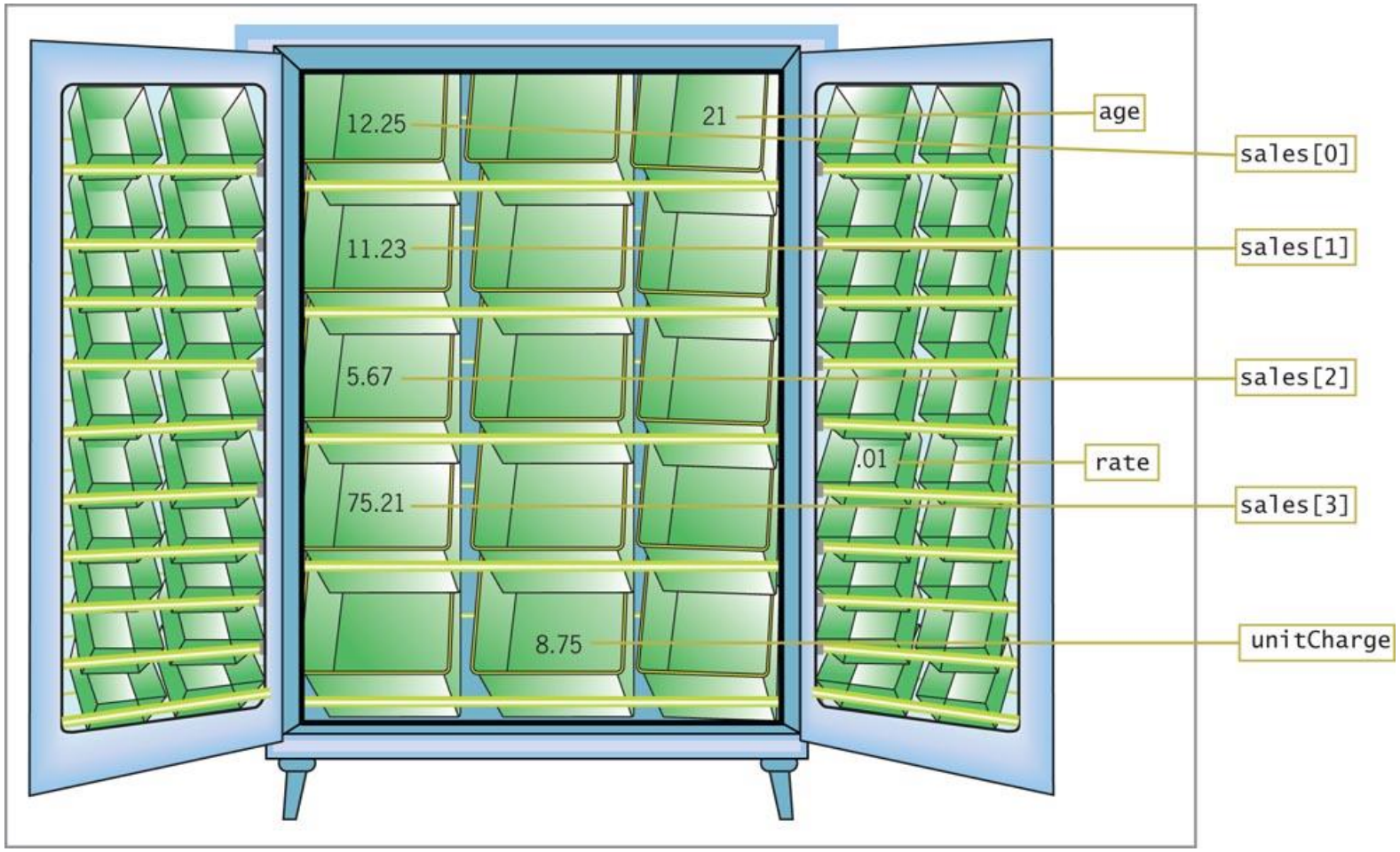



Figure 11-2 Illustration of naming conventions for one-dimensional `sales` array

# One-Dimensional Arrays (cont'd.)

**Problem specification**

The XYZ Company wants a program that allows the user to enter the sales made in each of its four sales regions. The program then should display the amounts on the computer screen.

| Input | Processing | Output |
|---|---|---|
| sales (made in each of 4 regions) | Processing items:<br>array (4 elements)<br>subscript (counter: 0 to 3) | sales (made in each of 4 regions) |

Algorithm:

1. repeat for (subscript from 0 to 3)
   enter the sales into array[subscript]
   end repeat
2. repeat for (subscript from 0 to 3)
   display the sales stored in array[subscript]
   end repeat

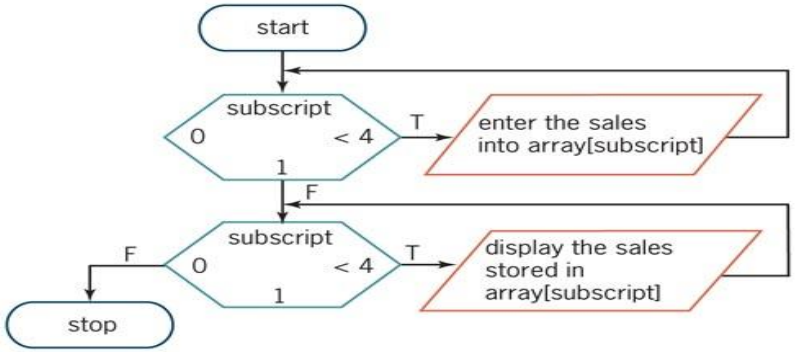



Figure 11-3 Problem specification and IPO chart for the XYZ Company's sales program

# Declaring and Initializing a One-Dimensional Array

- Must declare an array before you can use it
- Also good programming practice to initialize array variables
- Syntax for declaring and initializing a one-dimensional array is:

  *dataType arrayName* [*numberOfElements*] = {*initialValues*};

  - *dataType* is the type of data that the array variables (**elements**) will store
  - *arrayName* is name of array (same rules for naming an array as for naming a variable)

# Declaring and Initializing a One-Dimensional Array (cont'd.)

- *numberOfElements* is an integer specifying the size of the array (enclosed in square brackets)
- You may initialize array elements by entering one or more values, separated by commas, in braces
- Assigning initial values is referred to as **populating the array**
- The values used to populate an array should have the same data type as the array variables
  - Otherwise, implicit type conversion is performed

# Declaring and Initializing a One-Dimensional Array (cont'd.)

- Most C++ compilers initialize uninitialized numeric array elements to 0.0 or 0 (depending on data type)
- Automatic initialization is only done if you provide at least one value in the *initialValues* section
- Be sure to include an appropriate number of initial values
  - Providing too many will cause a compiler error or incorrectly assign adjacent memory locations

# Declaring and Initializing a One-Dimensional Array (cont'd.)

**HOW TO** Declare and Initialize a One-Dimensional Array

Syntax

*dataType arrayName*[*numberOfElements*] = {*initialValues*};

Example 1

```
char letters[3] = {'A', 'B', 'C'};
```

declares and initializes a three-element `char` array named `letters`

Example 2

```
double sales[4] = {0.0, 0.0, 0.0, 0.0};
```

or

```
double sales[4] = {0.0};
```

declares and initializes a four-element `double` array named `sales`; each element is initialized to 0.0

Example 3

```
int numbers[6] = {12, 0, 0, 0, 0, 0};
```

or

```
int numbers[6] = {12};
```

declares and initializes a six-element `int` array named `numbers`; the first element is initialized to 12, whereas the others are initialized to 0

Figure 11-4 How to declare and initialize a one-dimensional array

# Entering Data into a One-Dimensional Array

- You can use an assignment statement or the extraction operator to enter data into an array element
- Syntax of assignment statement is:

  *arrayName*[*subscript*] = *expression*;

  – *expression* can include any combination of constants, variables, and operators

  – Data type of *expression* must match data type of array; otherwise, implicit type conversion will occur

# Entering Data into a One-Dimensional Array (cont'd.)

**HOW TO** Use an Assignment Statement to Assign Data to a One-Dimensional Array

Syntax

*arrayName*[*subscript*] = *expression*;

Example 1

```
letters[1] = 'Y';
```

assigns the letter Y to the second element in the `letters` array

Example 2

```
int subscript = 0;
while (subscript < 4)
{
      sales[subscript] = 0.0;
      subscript += 1;
}     //end while
```

assigns the `double` number 0.0 to each of the four elements in the `sales` array; provides another means of initializing the array

*(continues)*

Figure 11-5 How to use an assignment statement to assign data to a one-dimensional array

# Entering Data into a One-Dimensional Array (cont'd.)

*(continued)*

Example 3

```
for (int x = 1; x <= 6; x += 1)
     numbers[x - 1] = pow(x, 2);
//end for
```

assigns the squares of the numbers from 1 through 6 to the six-element `numbers` array

Example 4

```
int increase = 0;
cout << "Enter increase amount: ";
cin >> increase;
for (int x = 0; x < 6; x += 1)
     numbers[x] += increase;
//end for
```

assigns, to each element in the six-element `numbers` array, the sum of the element's current value plus the value stored in the `increase` variable

Figure 11-5 How to use an assignment statement to assign data to a one-dimensional array (cont.)

# Entering Data into a One-Dimensional Array (cont'd.)

**HOW TO** Use the Extraction Operator to Store Data in a One-Dimensional Array

Syntax

```
cin >> arrayName[subscript];
```

Example 1

```
cin >> letters[0];
```

stores the user's entry in the first element in the `letters` array

*(continues)*

Figure 11-6 How to use the extraction operator to store data in a one-dimensional array

# Entering Data into a One-Dimensional Array (cont'd.)

*(continued)*

Example 2

```
for (int sub = 0; sub < 4; sub += 1)
{
      cout << "Enter the sales for Region ";
      cout << sub + 1 << ": ";
      cin >> sales[sub];
}     //end for
```

stores the user's entries in the four-element `sales` array

Example 3

```
int x = 0;
while (x < 6)
{
      cout << "Enter an integer: ";
      cin >> numbers[x];
      x += 1;
}     //end while
```

stores the user's entries in the six-element `numbers` array

Figure 11-6 How to use the extraction operator to store data in a one-dimensional array (cont.)

# Displaying the Contents of a One-Dimensional Array

- To display the contents of an array, you need to access each of its elements
- Use a loop along with a counter variable that keeps track of each subscript in the array

# Displaying the Contents of a One-Dimensional Array (cont'd.)

**HOW TO** Display the Contents of a One-Dimensional Array

Example 1

```
int x = 0;
while (x < 3)
{
      cout << letters[x] << endl;
      x += 1;
}     //end while
```

displays the contents of the three-element `letters` array

*(continues)*

Figure 11-7 How to display the contents of a one-dimensional array

# Displaying the Contents of a One-Dimensional Array (cont'd.)

*(continued)*

Example 2

```
for (int sub = 0; sub < 4; sub += 1)
{
     cout << "Sales for Region " << sub + 1 << ": $";
     cout << sales[sub] << endl;
}    //end for
```

displays the contents of the four-element `sales` array

Example 3

```
int x = 0;
do   //begin loop
{
     cout << numbers[x] << endl;
     x += 1;
}  while (x < 6);
```

displays the contents of the six-element `numbers` array

Figure 11-7 How to display the contents of a one-dimensional array (cont.)

# Coding the XYZ Company's Sales Program

- Log in to you computer
- Open Visual Studio
- Go to my website Project 6
- Answer the following questions 6.4, 6.5, 6.6
- When finished with all the problems raise your hand and I will verify. This is a test and it is worth 25 points
- Objective

Students should understand how to use an array and use it in a program.

# Coding the XYZ Company's Sales Program (cont'd.)

| IPO chart information | C++ instructions |
|---|---|
| **Input** | |
| sales (made in each of 4 regions) | the sales will be entered into the array |
| **Processing** | |
| array (4 elements) | `double sales[4] = {0.0};` |
| subscript counter (0 to 3) | declared and initialized in the for clause |
| **Output** | |
| sales (made in each of 4 regions) | displayed from the array by the for loop |
| **Algorithm** | |
| 1. repeat for (each of the 4 array elements)<br>enter the sales into the current array element<br><br>end repeat | `for (int sub = 0, sub < 4; sub += 1)`<br>`{`<br>`cout << "Enter the sales for Region ";`<br>`cout << sub + 1 << ": ";`<br>`cin >> sales[sub];`<br>`} //end for` |
| 2. repeat for (each of the 4 array elements)<br>display the sales stored in the current array element<br>end repeat | `for (int sub = 0, sub < 4; sub += 1)`<br>`{`<br>`cout << "Sales for Region " << sub + 1 << ": $";`<br>`cout << sales[sub] << endl;`<br>`} //end for` |

Figure 11-8 IPO chart information and C++ instructions for the XYZ Company's sales program

# Coding the XYZ Company's Sales Program (cont'd.)

```
 1 //XYZ Company.cpp
 2 //displays the contents of an array
 3 //Created/revised by <your name> on <current date>
 4
 5 #include <iostream>
 6 #include <iomanip>
 7 using namespace std;
 8
 9 int main()
10 {
11     //declare array
12     double sales[4] = {0.0};
13
14     //store data in the array
15     for (int sub = 0; sub < 4; sub += 1)
16     {
17          cout << "Enter the sales for Region ";
18          cout << sub + 1 << ": ";
19          cin >> sales[sub];
20     }    //end for
21
22     //display the contents of the array
23     cout << fixed << setprecision(2) << endl << endl;
24     for (int sub = 0; sub < 4; sub += 1)
25     {
26          cout << "Sales for Region " << sub + 1 << ": $";
27          cout << sales[sub] << endl;
28     }    //end for
29
30     //system("pause");
31     return 0;
32 }   //end of main function
```

array declaration

stores data in the array

displays the contents of the array

displays the contents of the array

your C++ development tool may require this statement

Figure 11-9 The XYZ Company's sales program

# Coding the XYZ Company's Sales Program (cont'd.)

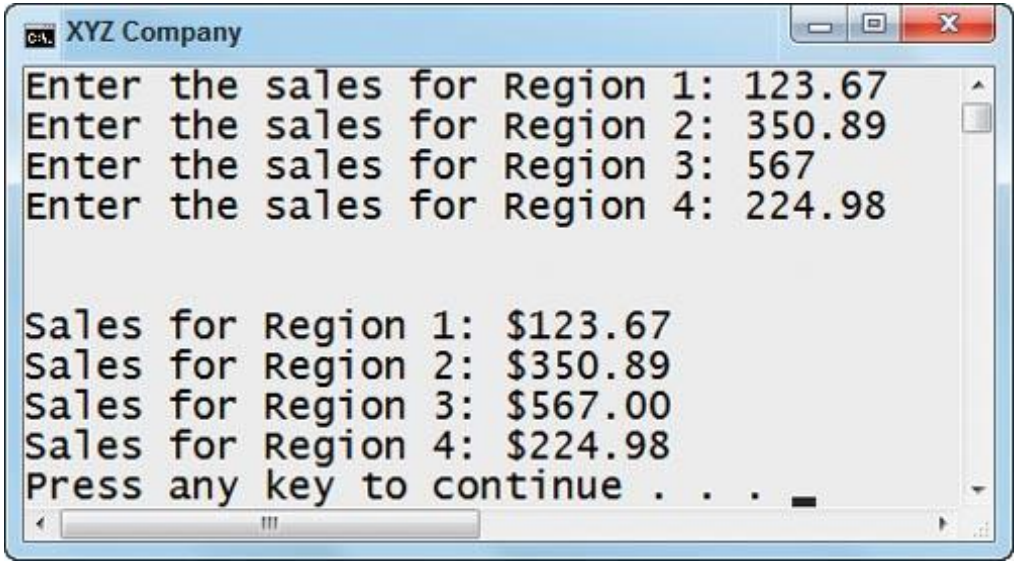



Figure 11-10 Sample run of XYZ Company's sales program

# Coding the XYZ Company's Sales Program (cont'd.)


Figure 11-11 Desk-check table after the array declaration statement is processed


Figure 11-12 Desk-check table after the *initialization* argument on line 15 is processed

# Coding the XYZ Company's Sales Program (cont'd.)

| sales[0] | sales[1] | sales[2] | sales[3] | sub |
|---|---|---|---|---|
| ~~0.0~~ | 0.0 | 0.0 | 0.0 | 0 |
| 123.67 | | | | |

Figure 11-13 Desk-check table after Region 1's sales entered in array

| sales[0] | sales[1] | sales[2] | sales[3] | sub |
|---|---|---|---|---|
| ~~0.0~~ | ~~0.0~~ | 0.0 | 0.0 | ~~0~~ |
| 123.67 | 350.89 | | | 1 |

Figure 11-14 Desk-check table after Region 2's sales entered in array

| sales[0] | sales[1] | sales[2] | sales[3] | sub |
|---|---|---|---|---|
| ~~0.0~~ | ~~0.0~~ | ~~0.0~~ | 0.0 | ~~0~~ |
| 123.67 | 350.89 | 567.0 | | ~~1~~ |
| | | | | 2 |

Figure 11-15 Desk-check table after Region 3's sales entered in array

# Coding the XYZ Company's Sales Program (cont'd.)

| sales[0] | sales[1] | sales[2] | sales[3] | sub |
|---|---|---|---|---|
| ~~0.0~~ | ~~0.0~~ | ~~0.0~~ | ~~0.0~~ | ~~0~~ |
| 123.67 | 350.89 | 567.0 | 224.98 | ~~1~~ |
| | | | | ~~2~~ |
| | | | | 3 |

Figure 11-16 Desk-check table after Region 4's sales entered in array

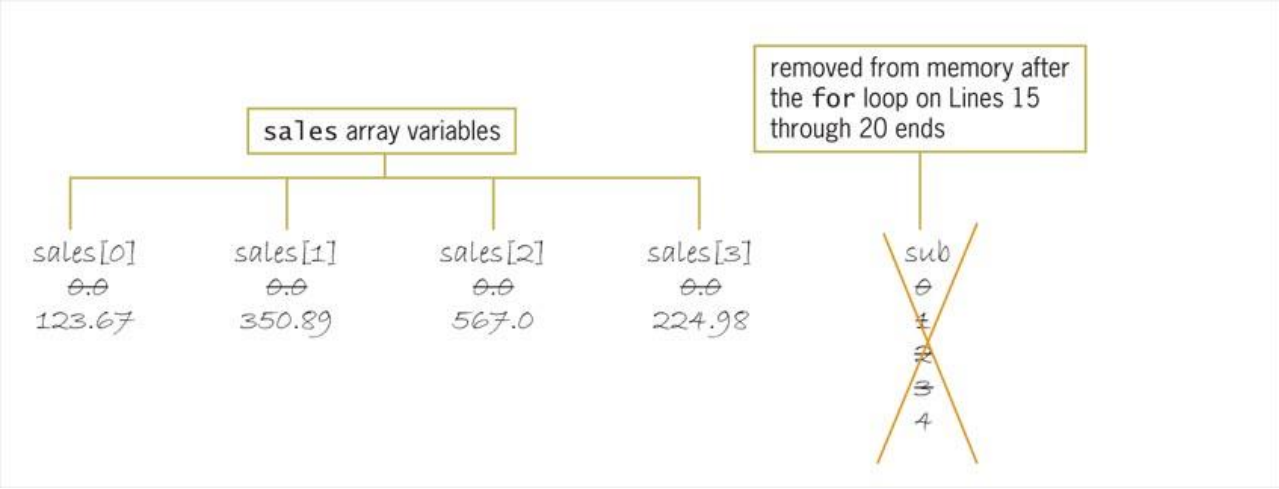



Figure 11-17 Desk-check table after `for` loop on lines 15-20 ends

# Coding the XYZ Company's Sales Program (cont'd.)

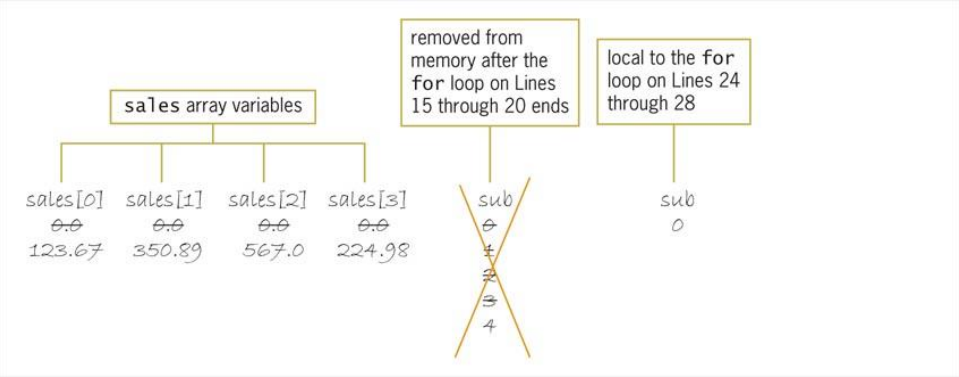



Figure 11-18 Desk-check table after the *initialization* argument on line 23 is processed

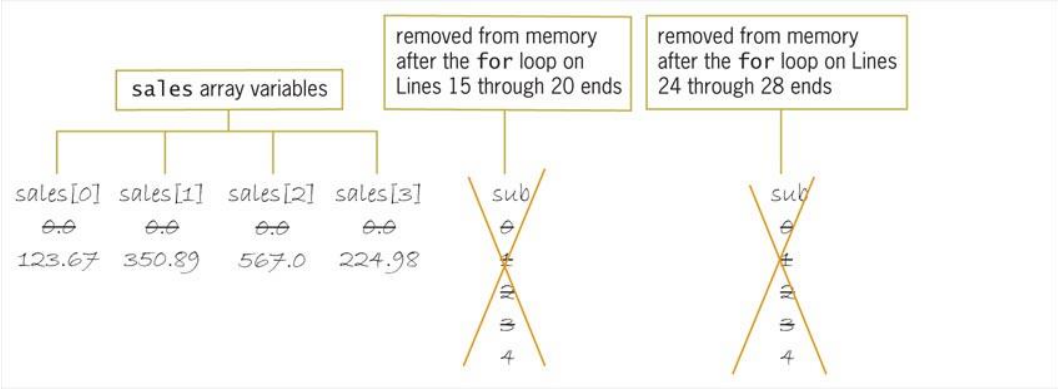



Figure 11-19 Desk-check table after the `for` loop on lines 23 through 27 ends

# Passing a One-Dimensional Array to a Function

- You can pass an array to a function by including the array's name as the actual argument
- Unless specified otherwise, scalar variables in C++ are passed *by value*
- Arrays, however, are passed *by reference* by default, because it is more efficient
- Passing an array *by value* would require copying the entire array, which could be very large

# Passing a One-Dimensional Array to a Function (cont'd.)

- Passing an array by reference allows the computer to pass the address of only the first array element
  - Since elements are stored in contiguous memory locations, computer can use this address to locate remaining elements in the array
- Indicate that you are passing an array by entering formal parameter's name and data type, followed by empty square brackets
  - Address-of operator (&) is not needed in function header or function prototype, since arrays are passed *by reference* by default

# Passing a One-Dimensional Array to a Function (cont'd.)

```
1 //Modified XYZ Company.cpp
2 //displays the contents of an array
3 //Created/revised by <your name> on <current date>
4
5 #include <iostream>
6 #include <iomanip>
7 using namespace std;
8
9 //function prototype
10 void displayArray(double dollars[], int numElements);
11
12 int main()
13 {
14     //declare array
15     double sales[4] = {0.0};
16
```

the name is optional

the name is optional

Figure 11-20 XYZ Company's modified sales program

# Passing a One-Dimensional Array to a Function (cont'd.)

```
17      //store data in the array
18      for (int sub = 0; sub < 4; sub += 1)
19      {
20           cout << "Enter the sales for Region ";
21           cout << sub + 1 << ": ";
22           cin >> sales[sub];
23      }    //end for
24
25      //display the contents of the array
26      displayArray(sales, 4);
27
28      //system("pause");
29      return 0;
30  }   //end of main function
31
32  //*****function definitions*****
33  void displayArray(double dollars[], int numElements)
34  {
35      cout << fixed << setprecision(2) << endl << endl;
36      for (int sub = 0; sub < numElements; sub += 1)
37      {
38           cout << "Sales for Region " << sub + 1 << ": $";
39           cout << dollars[sub] << endl;
40      }    //end for
41  }   //end of displayArray function
```

if your C++ development tool requires this statement, delete the two forward slashes

Figure 11-20 XYZ Company's modified sales program (cont'd.)

# Passing a One-Dimensional Array to a Function (cont'd.)

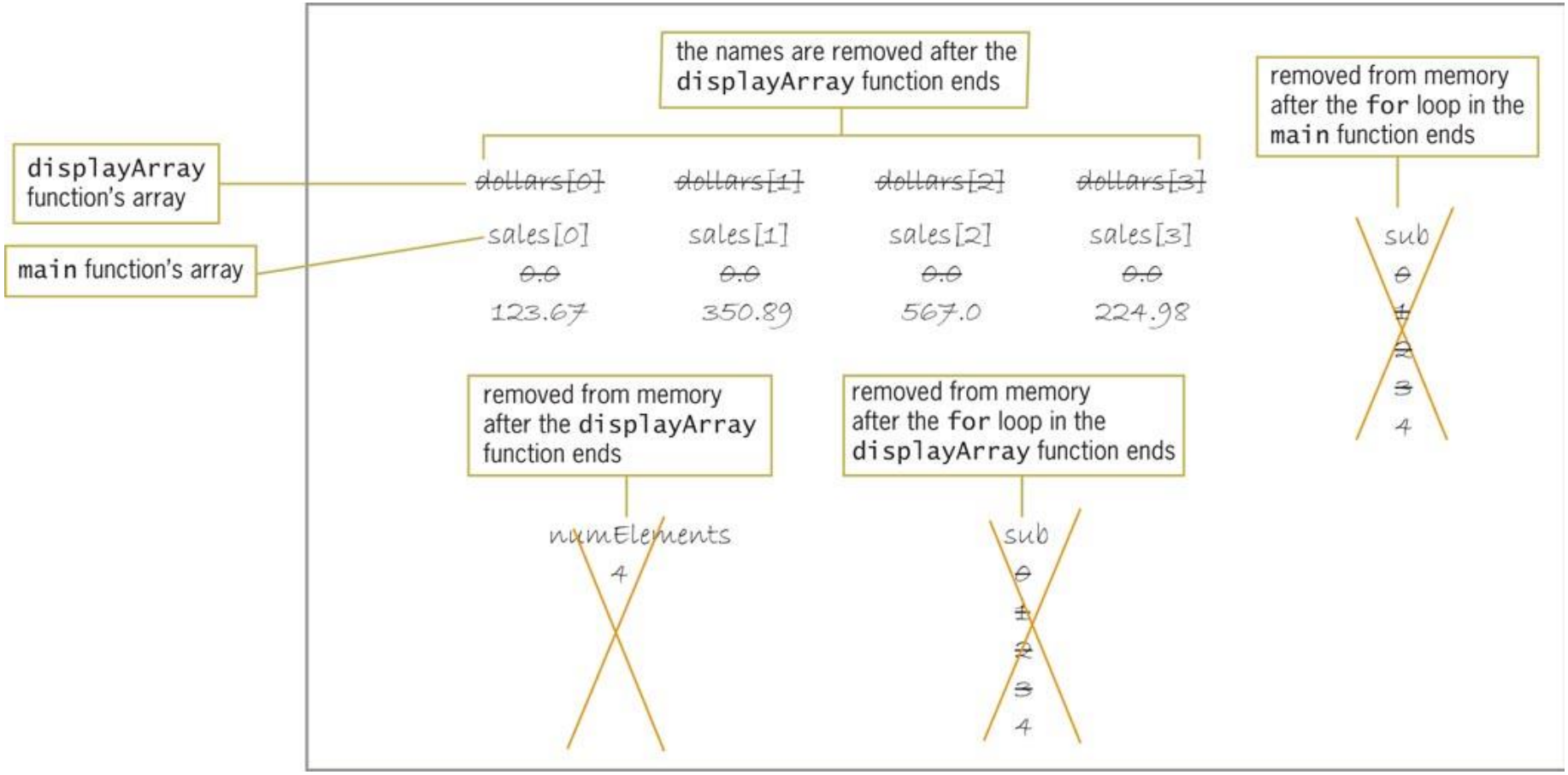



Figure 11-21 Completed desk-check table for the XYZ Company's modified sales program

# The Moonbucks Coffee Program—Calculating a Total and Average

- Program displays total and average number of pounds of coffee used in a 12-month period by the Moonbucks Coffee Company
- Stores monthly usage amount in a 12-element double array named `pounds`
- Uses a program-defined value-returning function named `getTotal` to calculate total usage for the year

# The Moonbucks Coffee Program (cont'd.)

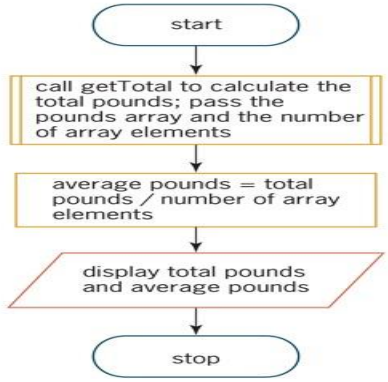
**Problem specification**
The store manager at Moonbucks Coffee wants a program that displays both the total and average number of pounds of coffee used during a 12-month period. Last year, the pounds of coffee used each month were as follows: 400.5, 450, 475.5, 336.5, 457, 325, 220.5, 276, 300, 320.5, 400.5, 415. The program will use two value-returning functions: `main` and `getTotal`. The `getTotal` function will calculate the total number of pounds of coffee used.

**`main` function**

| **IPO chart information** | **C++ instructions** |
|---|---|
| **Input** | |
| array (12 elements) | `double pounds[12] = {400.5, 450.0, 475.5, 336.5, 457.0, 325.0, 220.5, 276.0, 300.0, 320.5, 400.5, 415.0};` |
| **Processing** | |
| none | |
| **Output** | |
| total pounds | `double total = 0.0;` |
| average pounds | `double average = 0.0;` |
| **Algorithm** | |
| 1. call the getTotal function to calculate the total pounds; pass the pounds array and the number of array elements | `total = getTotal(pounds, 12);` |
| 2. calculate the average pounds by dividing the total pounds by the number of array elements | `average = total / 12;` |
| 3. display the total pounds and average pounds | `cout << "Total pounds: " << total << endl;`<br>`cout << "Average pounds: " << average << endl;` |

Figure 11-22 Problem specification, IPO chart information, and C++ instructions for the Moonbucks Coffee program

# The Moonbucks Coffee Program (cont'd.)




**getTotal function**

**IPO chart information**

**Input**

array (formal parameter)
number of elements (formal parameter)

**Processing**

subscript (counter: 0 to one less than the number of elements)

**Output**

total pounds (accumulator)

**Algorithm**

1. repeat for (subscript from 0 to one less than the number of elements)
   add the array[subscript] value to the total pounds
   end repeat
2. return the total pounds

**C++ instructions**

```
double poundsUsed[]
int numElements

declared and initialized in the for clause

double totalUsed = 0.0;

for (int sub = 0; sub < numElements;
sub += 1)
    totalUsed += poundsUsed[sub];

//end for
return totalUsed;
```

Figure 11-22 Problem specification, IPO chart information, and C++ instructions for the Moonbucks Coffee program (cont'd.)

# The Moonbucks Coffee Program (cont'd.)

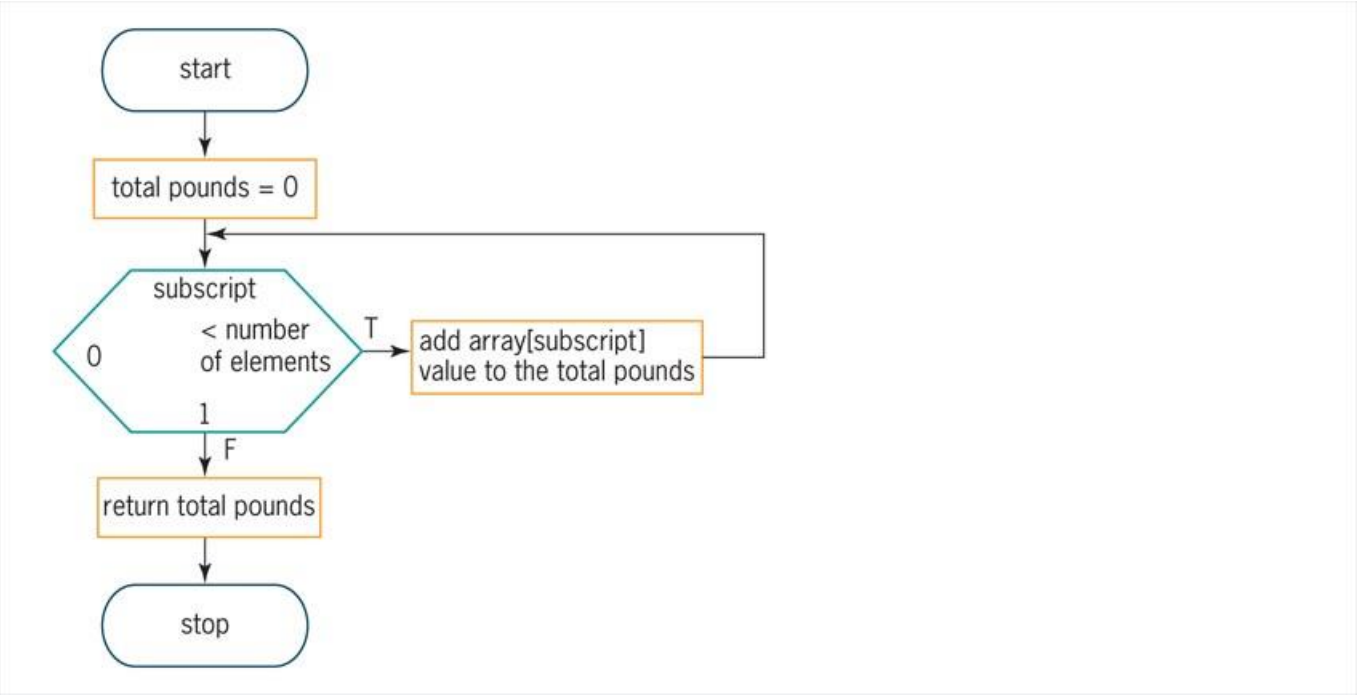



Figure 11-22 Problem specification, IPO chart information, and C++ instructions for the Moonbucks Coffee program (cont'd.)

# The Moonbucks Coffee Program (cont'd.)

```
1  //Moonbucks Coffee.cpp
2  //Displays the total and average number of pounds
3  //of coffee used during a 12-month period
4  //Created/revised by <your name> on <current date>
5
6  #include <iostream>
7  using namespace std;
8
9  //function prototype
10 double getTotal(double poundsUsed[], int numElements);
11
12 int main()
13 {
14    //declare array
15    double pounds[12] = {400.5, 450.0,
16           475.5, 336.5, 457.0, 325.0, 220.5,
17           276.0, 300.0, 320.5, 400.5, 415.0};
18    //declare variables
19    double total   = 0.0;
20    double average = 0.0;
21
22    //calculate the total and average pounds used
23    total = getTotal(pounds, 12);
24    average = total / 12;
25
```

Figure 11-23 Moonbucks Coffee program

# The Moonbucks Coffee Program (cont'd.)

```
26      //display the total and average pounds used
27      cout << "Total pounds: " << total << endl;
28      cout << "Average pounds: " << average << endl;
29
30      system("pause");
31      return 0;
32  }   //end of main function
33
34  //*****function definitions*****
35  double getTotal(double poundsUsed[], int numElements)
36  {
37      double totalUsed = 0.0;      //accumulator
38
39      //accumulate the pounds used
40      for (int sub = 0; sub < numElements; sub += 1)
41          totalUsed += poundsUsed[sub];
42      //end for
43
44      return totalUsed;
45  }   //end of getTotal function
```

your C++ development tool may not require this statement

Figure 11-23 Moonbucks Coffee program (cont'd.)

# The Moonbucks Coffee Program (cont'd.)

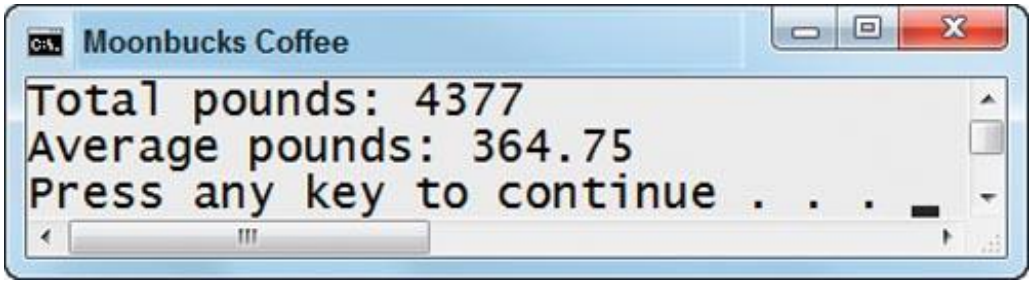



Figure 11-24 Result of running the Moonbucks Coffee program

# The JK Motors Program—Searching an Array

- Program displays number of employees whose salary is greater than an amount entered by user
- Stores employees' salaries in a 10-element `int` array named `salaries`
- Uses a loop to search through `salaries` array and a selection structure to compare salary in current element with salary entered by user
  - Increments a counter if the current salary is greater than the one entered by the user

# The JK Motors Program—Searching an Array (cont'd.)

**Problem specification**

The payroll manager at JK Motors wants a program that displays the number of employees who earn more than a specific amount, which he will enter. The company employs 10 people. Their annual salaries are as follows: 23000, 26000, 34000, 21000, 54000, 45000, 36000, 80000, 75000, 34000. The program will use only the `main` function. It will use a sentinel value to end the program.

**IPO chart information**

**Input**

array (10 elements)

salary to search for

**Processing**

subscript (counter: 0 to 9)

**Output**

number earning over the salary to search for (counter)

**Algorithm**

1. enter the salary to search for
2. repeat while (the salary to search for is greater than or equal to 0)
   - repeat for (subscript from 0 to 9)
     - if (the array[subscript] value is greater than the salary to search for)
       - add 1 to the number earning over the salary to search for
     - end if
   - end repeat

**C++ instructions**

```
int salaries[10] = {23000, 26000,
34000, 21000, 54000, 45000,
36000, 80000, 75000, 34000};
int searchFor = 0;
```

declared and initialized in the for clause

```
int numEarnOver = 0;
```

```
cout << "Salary to search for "
<< "(negative number to end): ";
cin >> searchFor;
while (searchFor >= 0)
{
    for (int sub = 0; sub < 10;
    sub += 1)
      if (salaries[sub] > searchFor)

          numEarnOver += 1;

      //end if
    //end for
```

Figure 11-25 Problem specification, IPO chart information, and C++ instructions for the JK motors program

# The JK Motors Program—Searching an Array (cont'd.)

```
display the number earning over
the salary to search for

enter the salary to search for

reset to 0 the number earning over
the salary to search for
end repeat
```

```
cout << "Number of employees earning "
<< "more than $" << searchFor << ": "
<< numEarnOver << endl << endl;

cout << "Salary to search for "
<< "(negative number to end): ";
cin >> searchFor;
numEarnOver = 0;

}   //end while
```

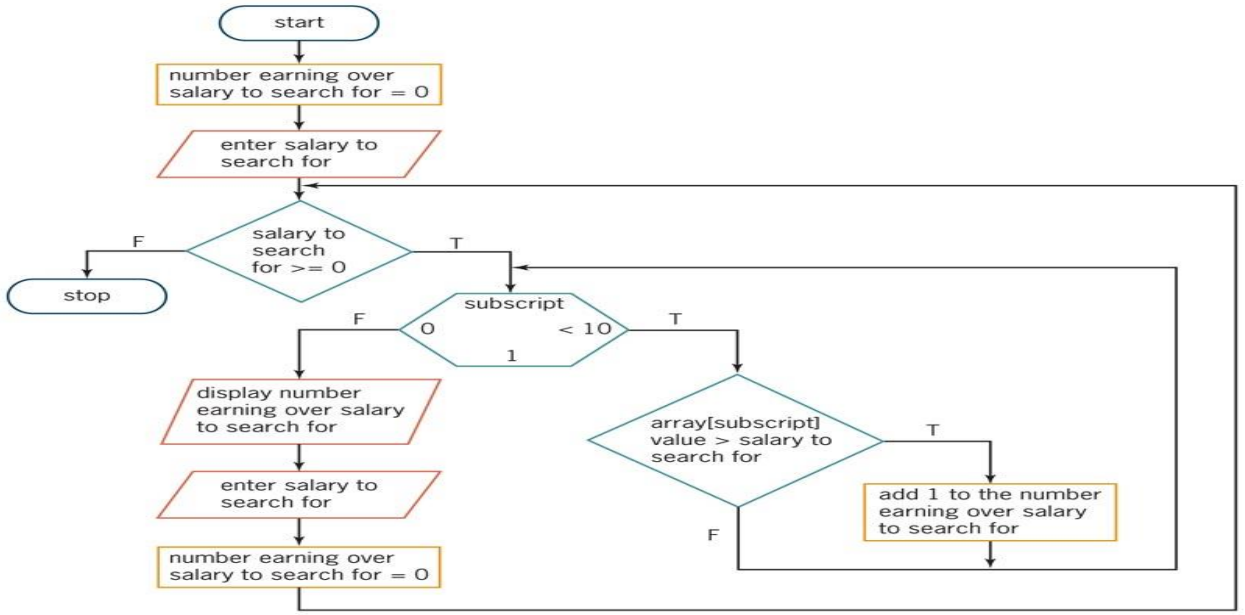



Figure 11-25 Problem specification, IPO chart information, and C++ instructions for the JK motors program (cont'd.)

# The JK Motors Program—Searching an Array (cont'd.)

```
 1 //JK Motors.cpp - displays the number of employees
 2 //whose salary is greater than a specific amount
 3 //Created/revised by <your name> on <current date>
 4
 5 #include <iostream>
 6 using namespace std;
 7
 8 int main()
 9 {
10     //declare array and variables
11     int salaries[10] = {23000, 26000,
12                         34000, 21000,
13                         54000, 45000,
14                         36000, 80000,
15                         75000, 34000};
16     int searchFor   = 0;
17     int numEarnOver = 0;          //counter
18
19     //get salary to search for
20     cout << "Salary to search for "
21          << "(negative number to end): ";
22     cin >> searchFor;
23
24     while (searchFor >= 0)
25     {
26          //search the array
27          for (int sub = 0; sub < 10; sub += 1)
28               if (salaries[sub] > searchFor)
29                    numEarnOver += 1;
30               //end if
31          //end for
32
33          //display the search results
34          cout << "Number of employees earning "
35          << "more than $" << searchFor << ": "
36          << numEarnOver << endl << endl;
37
```

Figure 11-26 JK motors program

# The JK Motors Program—Searching an Array (cont'd.)

```
38          //get another salary to search for
39          cout << "Salary to search for "
40               << "(negative number to end): ";
41          cin >> searchFor;
42          numEarnOver = 0;
43      }   //end while
44      //system("pause");
45      return 0;
46  }   //end of main function
```

your C++ development tool may require this statement

Figure 11-26 JK motors program (cont'd.)

# The JK Motors Program—Searching an Array (cont'd.)

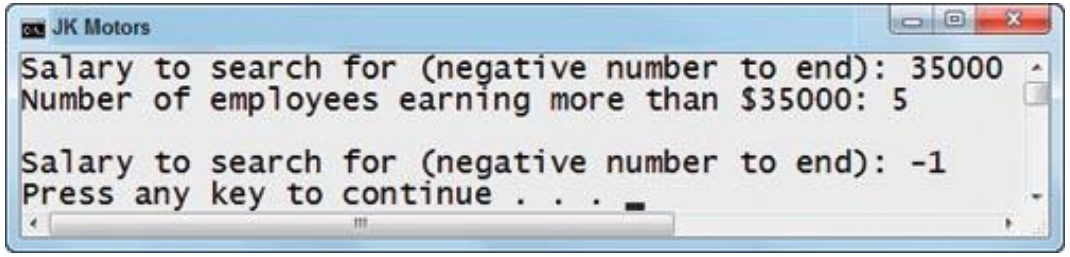



Figure 11-27 Sample run of the JK motors program

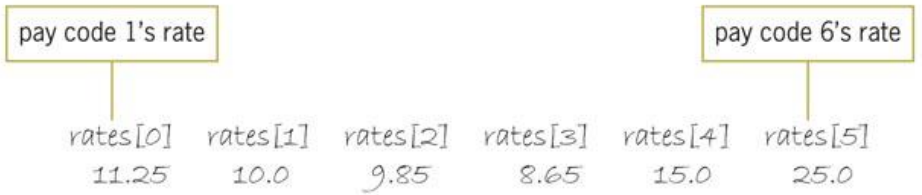
# The Hourly Rate Program—Accessing an Individual Element

- Program uses a six-element array to store hourly rates, each associated with a specific pay code
- Program prompts user to enter a pay code and determines whether the pay code is valid
- Pay code must be between 1 and 6, inclusive
- If pay code is valid, program uses pay code to display appropriate hourly rate from array
- If the pay code is not valid, program displays the message “Invalid pay code”

# The Hourly Rate Program—Accessing an Individual Element (cont'd.)

**Problem specification**

Create a program that displays the hourly rate associated with the pay code entered by the user. The pay codes and hourly rates are as follows: 1, $11.25; 2, $10; 3, $9.85; 4, $8.65; 5, $15; 6, $25. The program should store the hourly rates in a six-element `double` array named `rates`. The array is illustrated below. The hourly rate for pay code 1 is stored in the first array element, whose subscript is 0. The hourly rate for pay code 2 is stored in the array element whose subscript is 1, and so on. Notice that the pay code is always one number more than the subscript of its corresponding hourly rate in the array.




| IPO chart information | C++ instructions |
|---|---|
| **Input** | |
| pay code | `int code = 0;` |
| **Processing** | |
| array [6 elements] | `double rates[6] = {11.25, 10.0, 9.85, 8.65, 15.0, 25.0};` |
| **Output** | |
| hourly rate | displayed from the array |

Figure 11-28 Problem specification, IPO chart information, and C++ instructions for the hourly rate program

# The Hourly Rate Program—Accessing an Individual Element (cont'd.)

**Algorithm**

1. enter the pay code
2. if (the pay code is greater than or equal to 1 and less than or equal to 6)
   - display the hourly rate from the array, using the pay code minus 1 as the subscript
   - else
   - display "Invalid pay code" message
   - end if

```
cout << "Pay code (1 - 6): ";
cin >> code;
if (code >= 1 && code <= 6)

    cout << "Hourly rate: $" <<
    rates[code - 1] << endl;

else
    cout << "Invalid pay code" << endl;

//end if
```

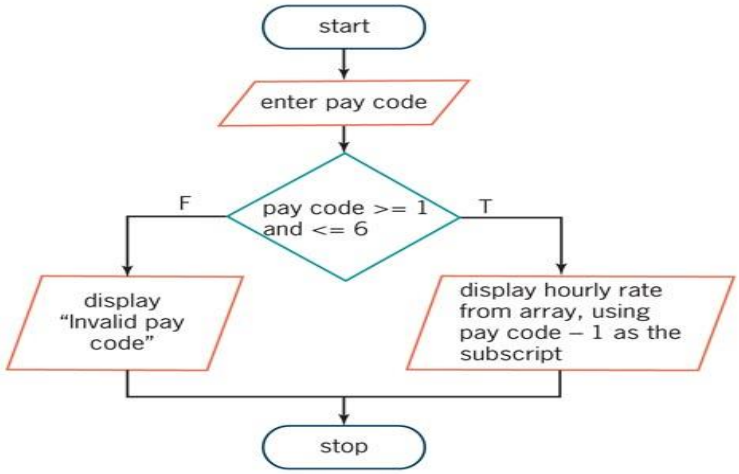



Figure 11-28 Problem specification, IPO chart information, and C++ instructions for the hourly rate program (cont'd.)

# The Hourly Rate Program—Accessing an Individual Element (cont'd.)

```
 1 //Hourly Rate.cpp - displays the hourly rate
 2 //associated with the pay code entered by the user
 3 //Created/revised by <your name> on <current date>
 4
 5 #include <iostream>
 6 #include <iomanip>
 7 using namespace std;
 8
 9 int main()
10 {
11     //declare array and variable
12     double rates[6] = {11.25, 10.0, 9.85,
13                        8.65, 15.0, 25.0};
14     int code = 0;
15
16     //display hourly rate with two decimal places
17     cout << fixed << setprecision(2);
18
19     //get pay code
20     cout << "Pay code (1 - 6): ";
21     cin >> code;
22     if (code >= 1 && code <= 6)
23         cout << "Hourly rate: $" <<
24             rates[code - 1] << endl;
25     else
26         cout << "Invalid pay code" << endl;
27     //end if
28
29     //system("pause");
30     return 0;
31 }   //end of main function
```

your C++ development tool may require this statement

Figure 11-29 Hourly rate program

# The Hourly Rate Program—Accessing an Individual Element (cont'd.)

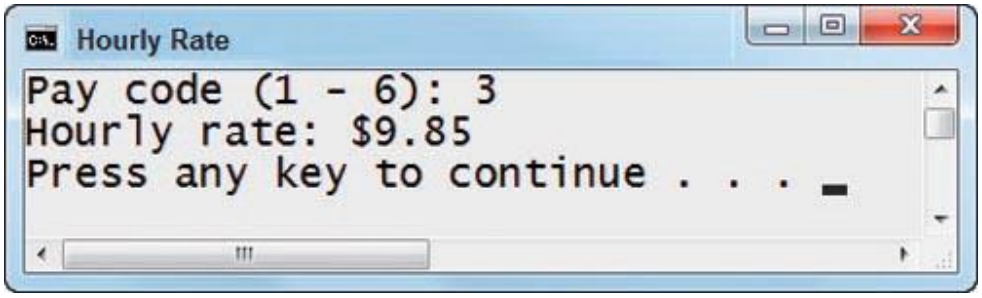



Figure 11-30 Sample run of the hourly rate program

# The Random Numbers Program—Finding the Highest Value

- Program's `main` function assigns random numbers between 1 and 100 to a five-element `int` array named `randNums`
- Calls a program-defined void function to display contents of array
- Then calls a program-defined value-returning function to determine highest number in array
- `main` function then display's highest value

# The Random Numbers Program (cont'd.)

**Problem specification**
Create a program that assigns random integers from 1 through 100 to a five-element array. The program should display both the contents of the array and the highest number stored in the array. Use a program-defined void function to display the array's contents. Use a program-defined value-returning function to determine the highest number in the array.

**main function**

| IPO chart information | C++ instructions |
|---|---|
| **Input** | |
| random number (5 from 1 to 100) | generated by the program and stored in the array |
| **Processing** | |
| array (5 elements) | `int randNums[5] = {0};` |
| subscript counter (0 to 4) | declared and initialized in the for clause |
| **Output** | |
| random number (5 from 1 to 100) | displayed by the displayArray function |
| highest value in the array | determined by the getHighest function |

Figure 11-31 Problem specification, IPO chart information, and C++ instructions for the random numbers program

# The Random Numbers Program (cont'd.)

**Algorithm**

1. initialize the random number generator

```
srand(static_cast<int>(time(0)));
```

2. repeat for (subscript counter from 0 to 4)
   generate a random number and store it in the current array element
   end repeat

```
for (int sub = 0; sub < 5; sub += 1)

   randNums[sub] = 1 + rand()
   % (100 - 1 + 1);

//end for
```

3. call the displayArray function to display the contents of the array, pass the array and the number of elements

```
displayArray(randNums, 5);
```

4. call the getHighest function to determine the highest number in the array, pass the array and the number of elements, then display the highest number

```
cout << endl << "Highest number: "
<< getHighest(randNums, 5) << endl;
```

Figure 11-31 Problem specification, IPO chart information, and C++ instructions for the random numbers program (cont'd.)

# The Random Numbers Program (cont'd.)

**displayArray function**

| IPO chart information | C++ instructions |
|---|---|
| **Input** | |
| array (5 elements) | `int numbers[]` (formal parameter) |
| number of elements | `int numElements` (formal parameter) |
| **Processing** | |
| subscript counter (0 to the number of elements) | declared and initialized in the for clause |
| **Output** | |
| array (5 elements) | displayed from the array by the for loop |

**Algorithm**

```
repeat for (subscript counter
from 0 to the number of elements)
      display the contents of the
      current array element
end repeat
```

```
for (int sub = 0; sub <
numElements; sub += 1)
   cout << numbers[sub] << endl;

//end for
```

Figure 11-31 Problem specification, IPO chart information, and C++ instructions for the random numbers program (cont'd.)

# The Random Numbers Program (cont'd.)

**getHighest function**

**IPO chart information**

**Input**

array (5 elements)
number of elements

**Processing**

subscript counter

**Output**

highest number

**Algorithm**

1. repeat while (the subscript counter is less than the number of elements)
   if (the current array element's value is greater than the highest number)
   assign the current array element's value as the highest number
   end if
   add 1 to the subscript counter
   end repeat
2. return highest number

**C++ instructions**

```
int numbers[] (formal parameter)
int numElements (formal parameter)

int x = 1;

int high = numbers[0];

while (x < numElements)
{
   if (numbers[x] > high)

      high = numbers[x];

   //end if
   x += 1;
}  //end while
return high;
```

Figure 11-31 Problem specification, IPO chart information, and C++ instructions for the random numbers program (cont'd.)

# The Random Numbers Program (cont'd.)

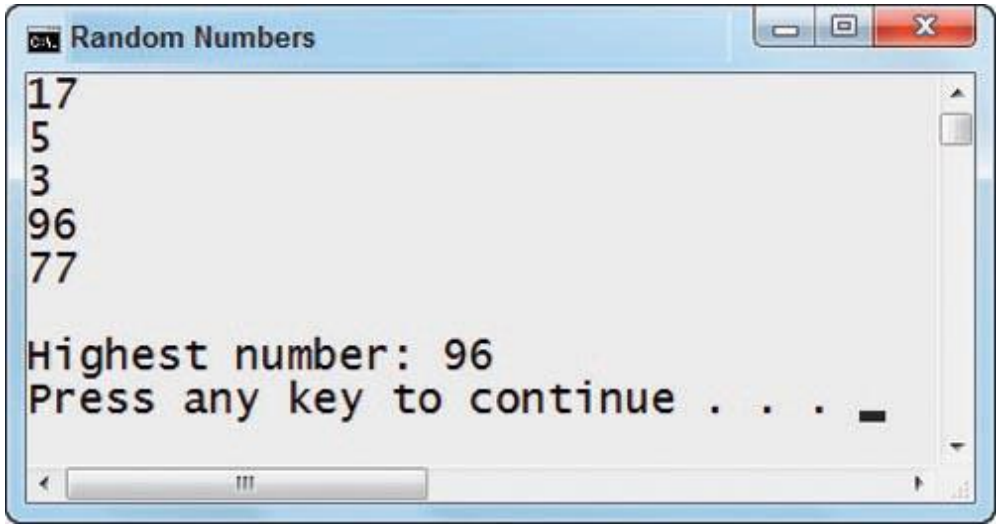



Figure 11-32 Sample run of the random numbers program

# The Random Numbers Program (cont'd.)

```
1  //Random Numbers.cpp - displays the highest
2  //random number stored in an array
3  //Created/revised by <your name> on <current date>
4
5  #include <iostream>
6  #include <ctime>
7  using namespace std;
8
9  //function prototypes
10 void displayArray(int numbers[], int numElements);
11 int getHighest(int numbers[], int numElements);
12
13 int main()
14 {
15    //declare array
16    int randNums[5] = {0};
17
18    //initialize random number generator
19    srand(static_cast<int>(time(0)));
20    //assign random integers from 1
21    //through 100 to the array
22    for (int sub = 0; sub < 5; sub += 1)
23         randNums[sub] = 1 + rand() % (100 - 1 + 1);
24    //end for
25
```

Figure 11-33 Random numbers program

# The Random Numbers Program (cont'd.)

```
26      //display array
27      displayArray(randNums, 5);
28
29      //display highest number in the array
30      cout << endl << "Highest number: "
31           << getHighest(randNums, 5) << endl;
32
33      system("pause");
34      return 0;
35  }   //end of main function
36
37  //*****function prototype*****
38  void displayArray(int numbers[], int numElements)
39  {
40      for (int sub = 0; sub < numElements; sub += 1)
41           cout << numbers[sub] << endl;
42      //end for
43  }   //end of displayArray function
44
45  int getHighest(int numbers[], int numElements)
46  {
47      //assign first element's value
48      //to the high variable
49      int high = numbers[0];
50
51      //begin the search with the second element
52      int x = 1;
53
```

your C++ development tool may not require this statement

Figure 11-33 Random numbers program (cont'd.)

# The Random Numbers Program (cont'd.)

```
54      //search for highest number
55      while (x < numElements)
56      {
57           if (numbers[x] > high)
58                high = numbers[x];
59           //end if
60           x += 1;
61      }    //end while
62
63      return high;
64 }    //end of getHighest function
```

Figure 11-33 Random numbers program (cont'd.)

# The Random Numbers Program (cont'd.)

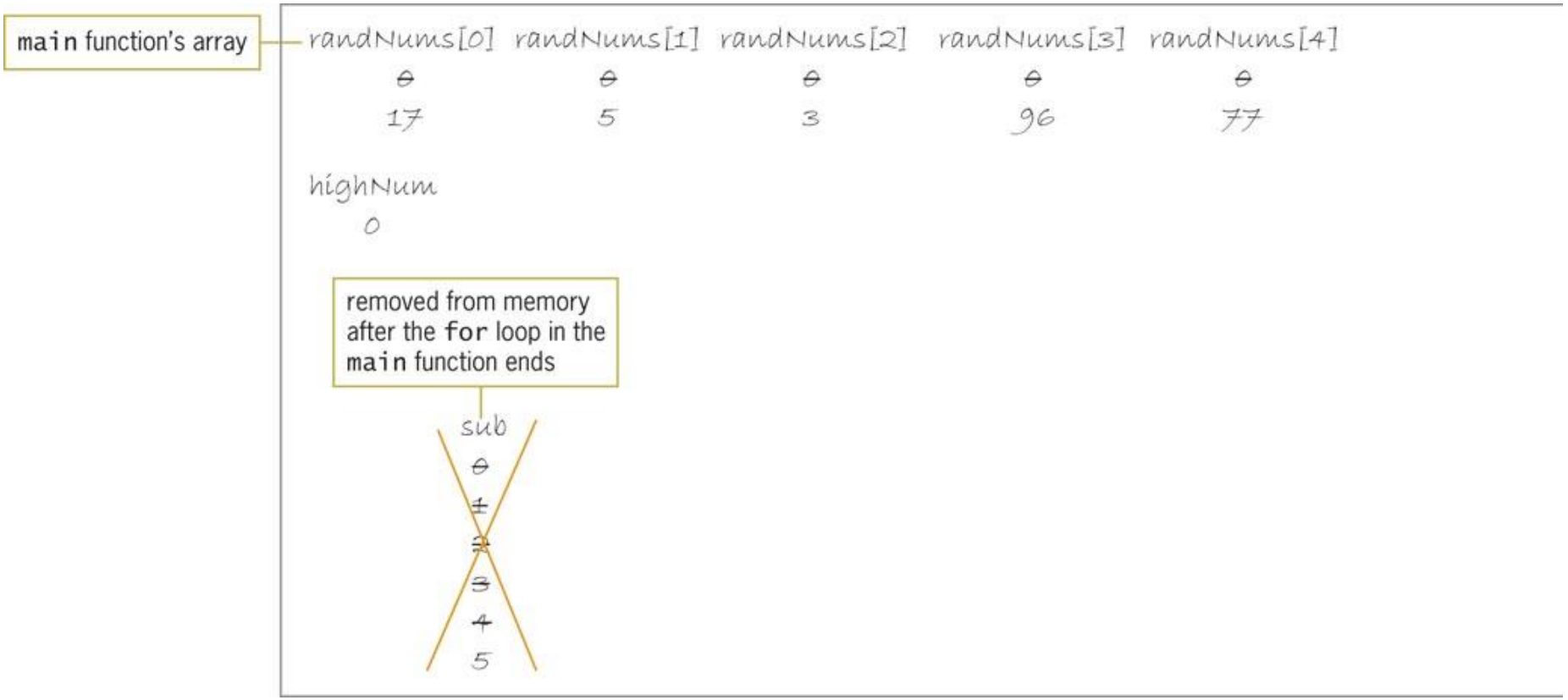



Figure 11-34 Desk-check table after the `for` loop on lines 24 through 26 ends

# The Random Numbers Program (cont'd.)

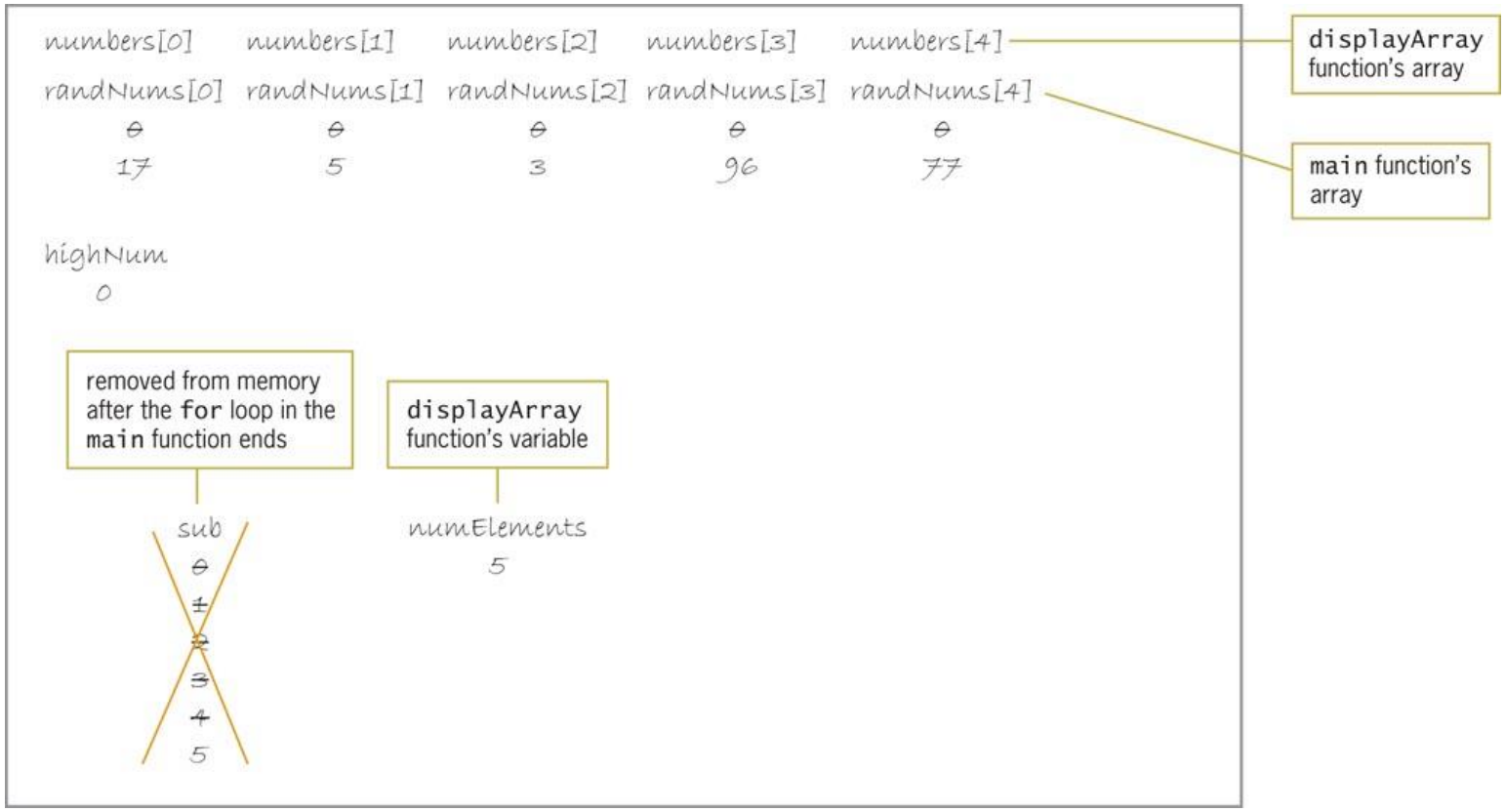



Figure 11-35 Desk-check table after the `displayArray` function header is processed

# The Random Numbers Program (cont'd.)

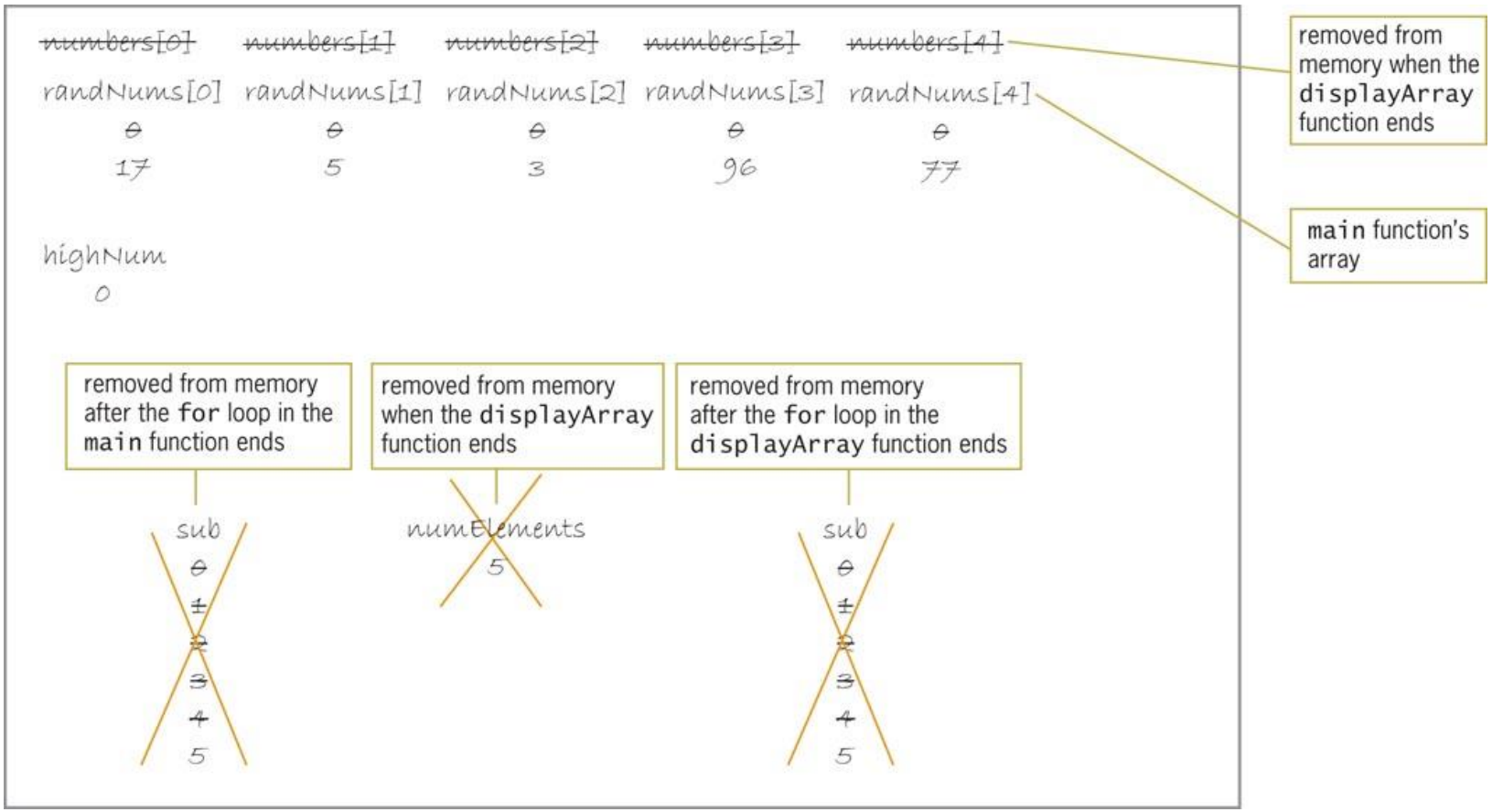



Figure 11-36 Desk-check table after the `displayArray` function ends

# The Random Numbers Program (cont'd.)

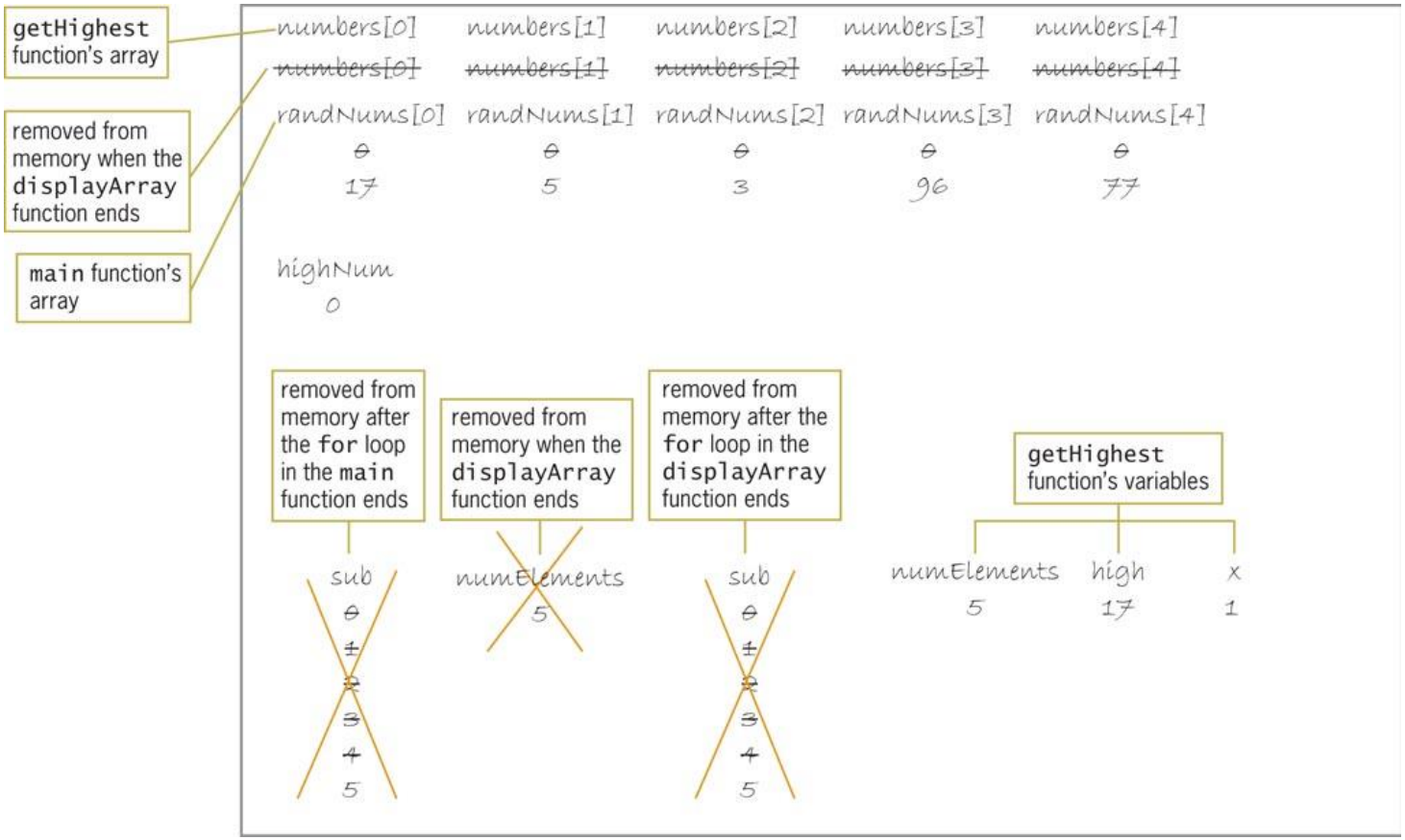



Figure 11-37 Desk-check table after the declaration statements on lines 52 and 55 are processed

# The Random Numbers Program (cont'd.)

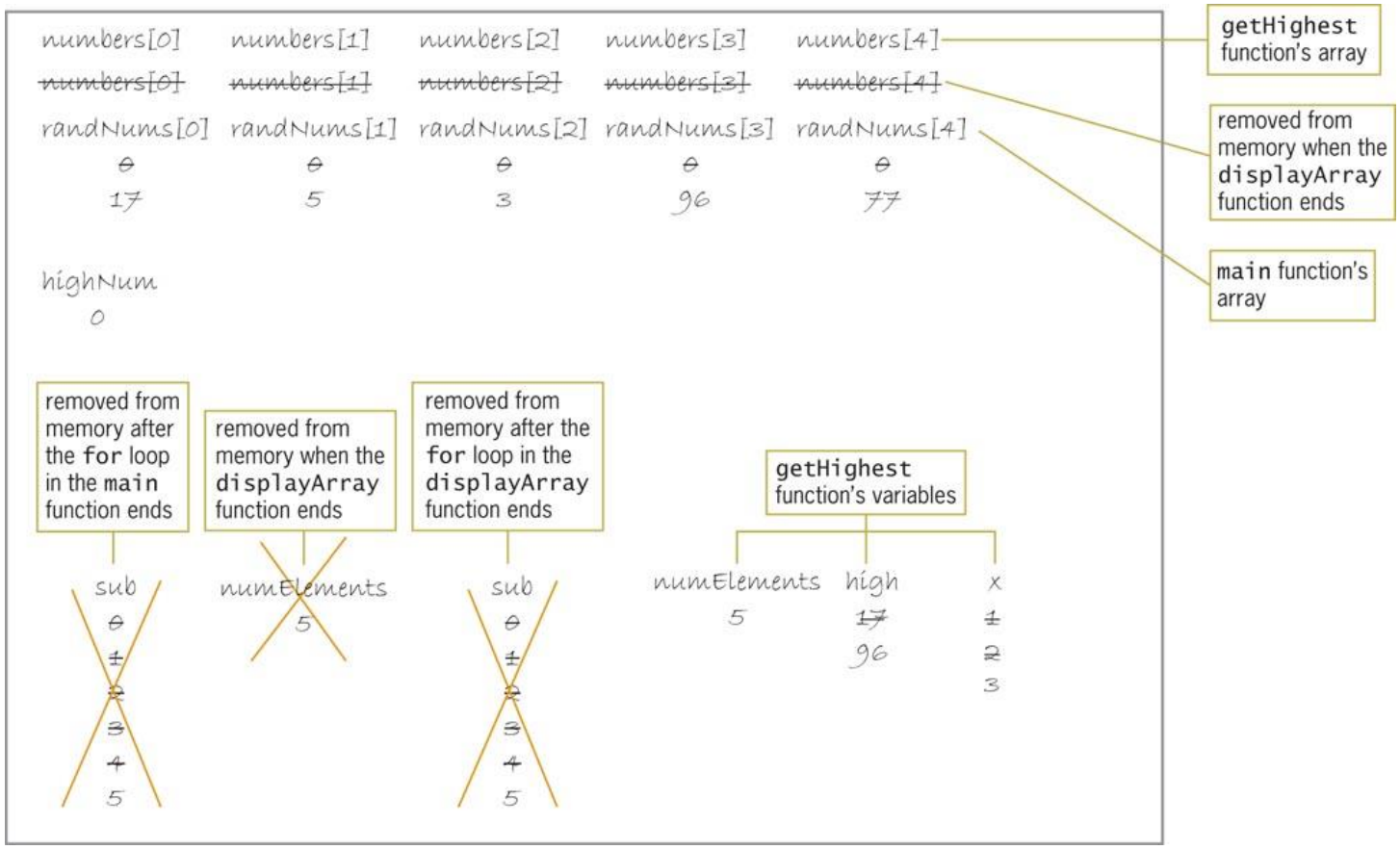



Figure 11-38 Desk-check table showing the fourth element's value entered in the `high` variable

# The Random Numbers Program (cont'd.)

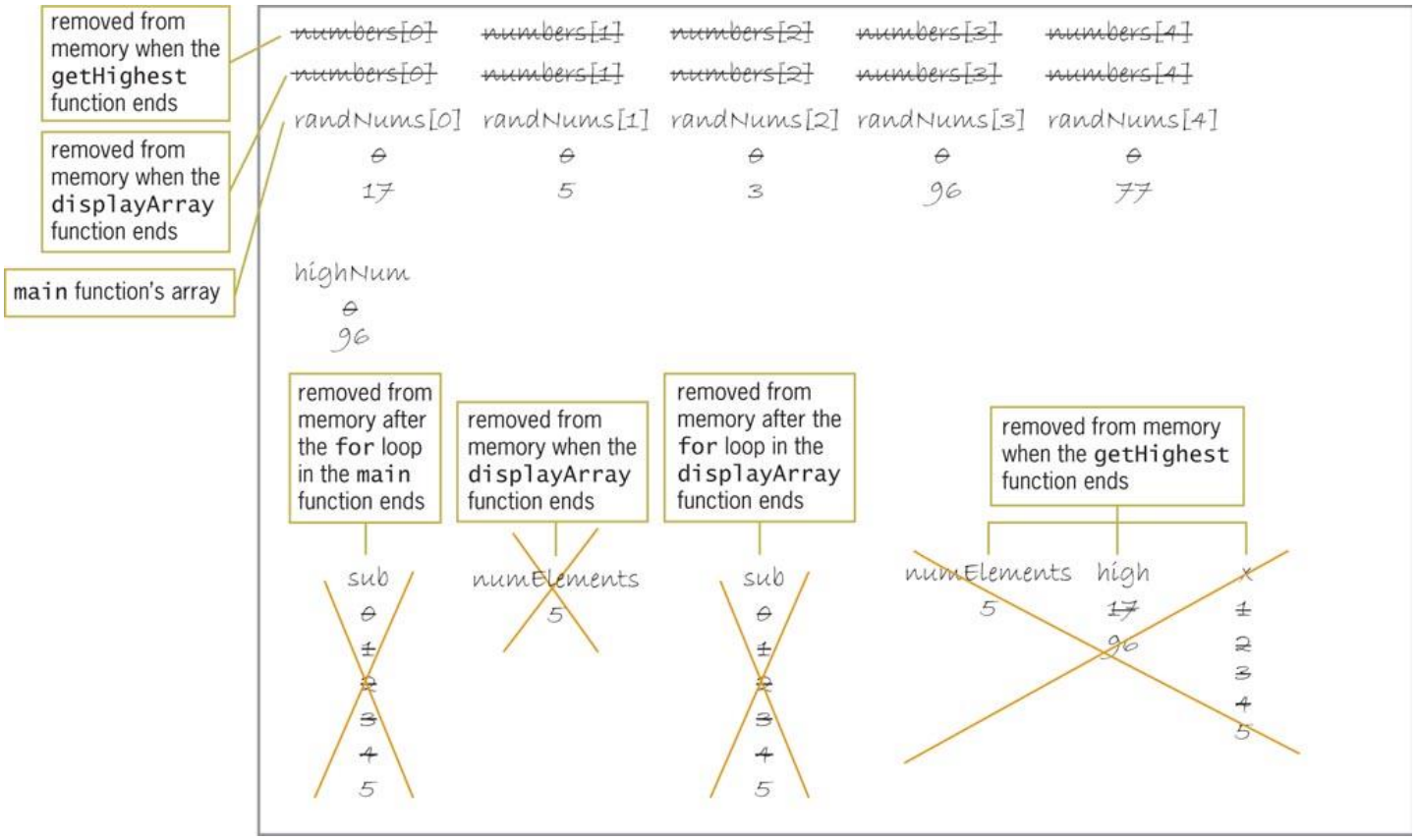



Figure 11-39 Completed desk-check table for random numbers program

# Sorting the Data Stored in a One-Dimensional Array

- You sometimes need to arrange the contents of an array in either ascending or descending order
- Arranging data in a specific order is called **sorting**
- When a one-dimensional array is sorted in ascending order, first element contains smallest value and last element contains largest value
- Conversely, when sorted in descending order, first element contains largest value and last element contains smallest value

# Sorting the Data Stored in a One-Dimensional Array (cont'd.)

- Many different types of sorting algorithms
- **Bubble sort** provides a quick and easy way to sort items stored in an array, as long as the number of items is relatively small (e.g., fewer than 50)
- Works by comparing adjacent array elements and swapping ones that are out of order
- Continues comparing and swapping until data in the array are sorted

# Sorting the Data Stored in a One-Dimensional Array (cont'd.)

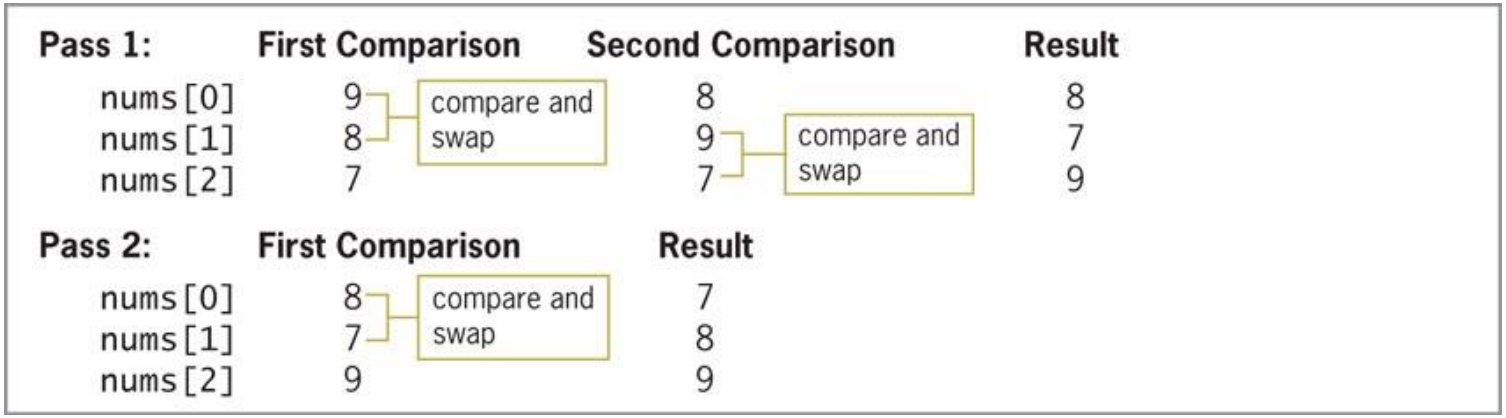



Figure 11-40 Array values before, during, and after the bubble sort

# Sorting the Data Stored in a One-Dimensional Array (cont'd.)

```
 1  //Bubble Sort.cpp - uses the bubble sort to
 2  //sort the contents of a one-dimensional array
 3  //in ascending order
 4  //Created/revised by <your name> on <current date>
 5
 6  #include <iostream>
 7  using namespace std;
 8
 9  int main()
10  {
11     int numbers[4] = {23, 46, 12, 35};
12     int sub        = 0;   //keeps track of subscripts
13     int temp       = 0;   //variable used for swapping
14     int maxSub     = 3;   //maximum subscript
15     int lastSwap   = 0;   //position of last swap
16     char swap      = 'Y'; //indicates if a swap was made
17
18     //repeat loop instructions as long as a swap was made
19     while (swap == 'Y')
20     {
21          swap = 'N';    //assume no swaps are necessary
22
23          sub = 0;      //begin comparing with first
24                        //array element
25
```

Figure 11-41 Bubble sort program

# Sorting the Data Stored in a One-Dimensional Array (cont'd.)

```
26          //compare adjacent array elements to determine
27          //whether a swap is necessary
28          while (sub < maxSub)
29          {
30               if (numbers[sub] > numbers[sub + 1])
31               {
32                    //a swap is necessary
33                    temp = numbers[sub];
34                    numbers[sub] = numbers[sub + 1];
35                    numbers[sub + 1] = temp;
36                    swap = 'Y';
37                    lastSwap = sub;
38               } //end if
39               sub += 1; //increment subscript
40          } //end while
41
42          maxSub = lastSwap; //reset maximum subscript
43     } //end while
44
45     //display sorted array
46     for (int x = 0; x < 4; x += 1)
47          cout << numbers[x] << endl;
48     //end for
49
50     system("pause");
51     return 0;
52  }  //end of main function
```

your C++ development tool may not require this statement

Figure 11-41 Bubble sort program (cont'd.)

# Sorting the Data Stored in a One-Dimensional Array (cont'd.)

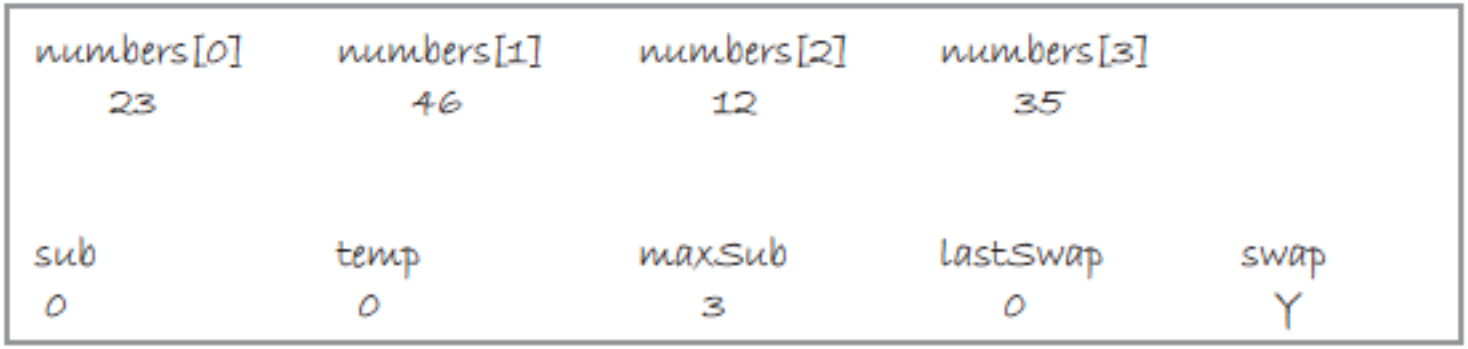



Figure 11-42 Desk-check table after the declaration statements on lines 11 through 16 are processed

# Sorting the Data Stored in a One-Dimensional Array (cont'd.)

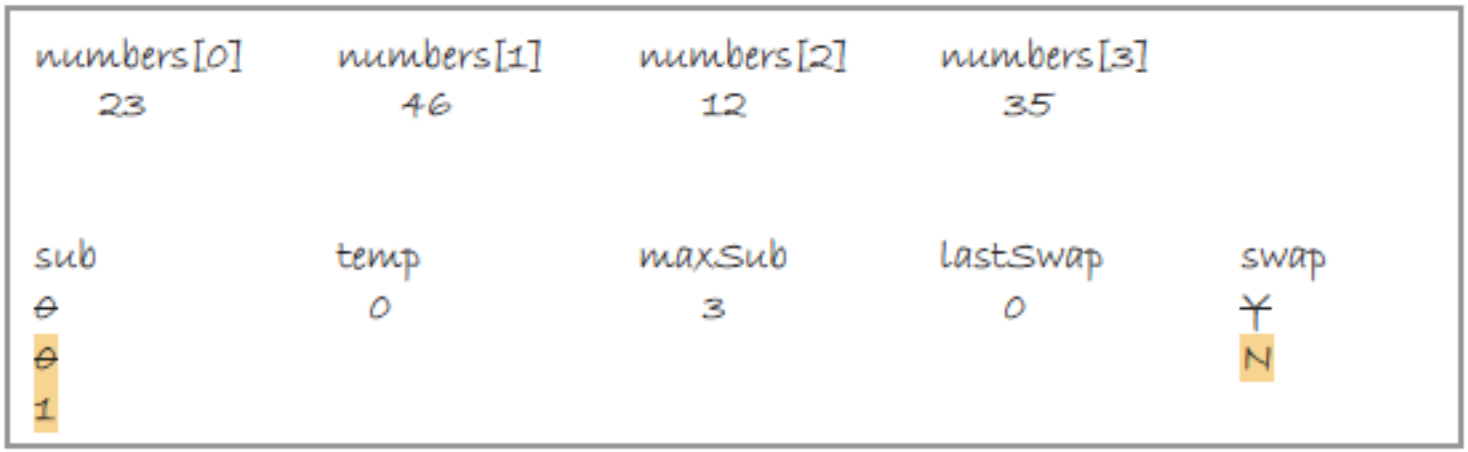



Figure 11-43 Desk-check table after nested loop is processed first time

# Sorting the Data Stored in a One-Dimensional Array (cont'd.)

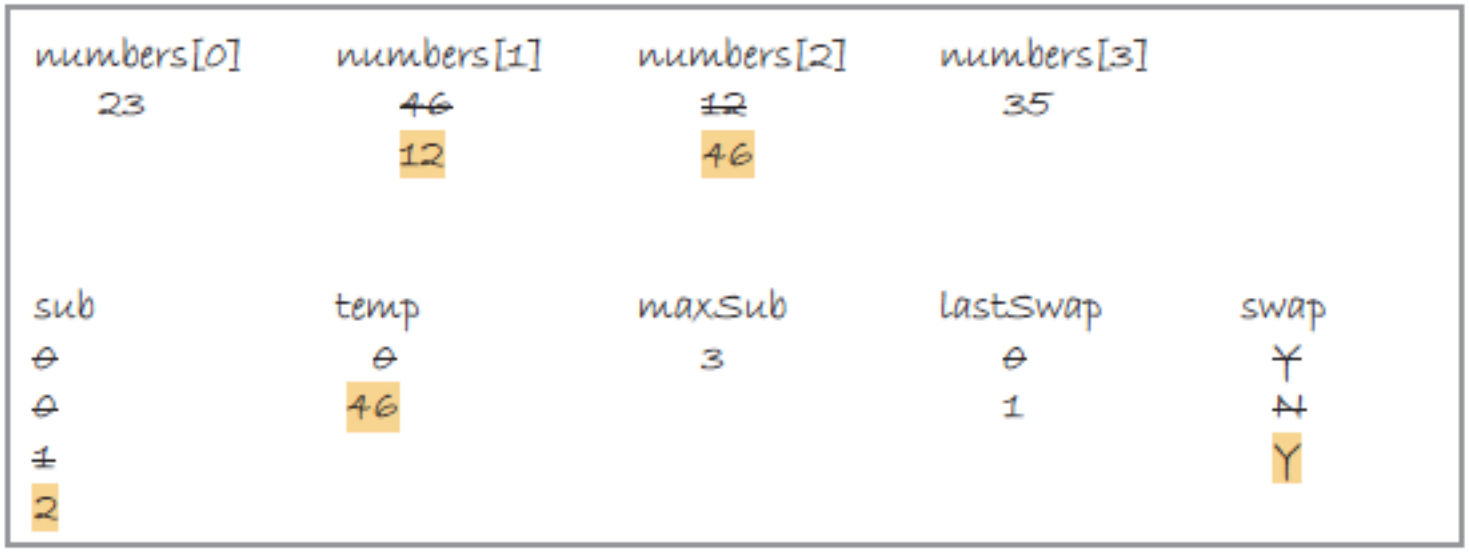



Figure 11-44 Desk-check table after nested loop is processed second time

# Sorting the Data Stored in a One-Dimensional Array (cont'd.)

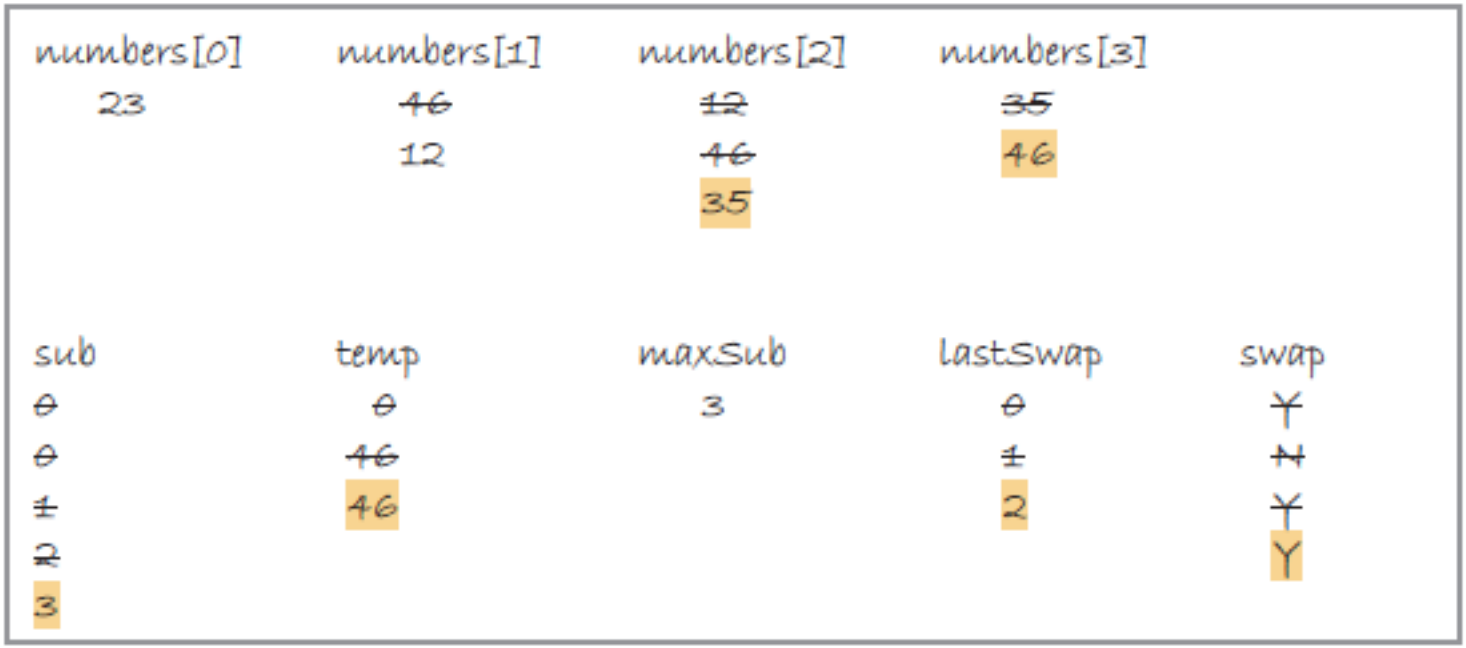



Figure 11-45 Desk-check table after nested loop is processed third time

# Sorting the Data Stored in a One-Dimensional Array (cont'd.)

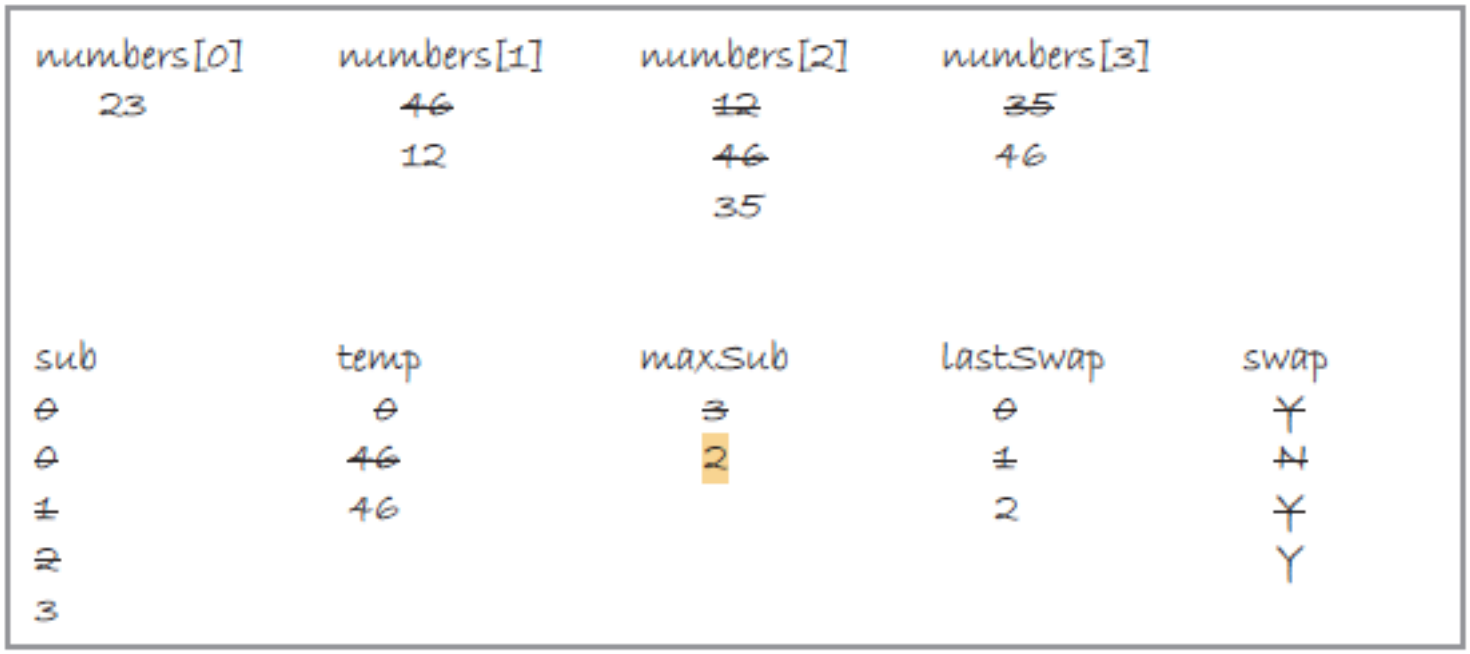



Figure 11-46 Desk-check table after outer loop is processed first time

# Sorting the Data Stored in a One-Dimensional Array (cont'd.)

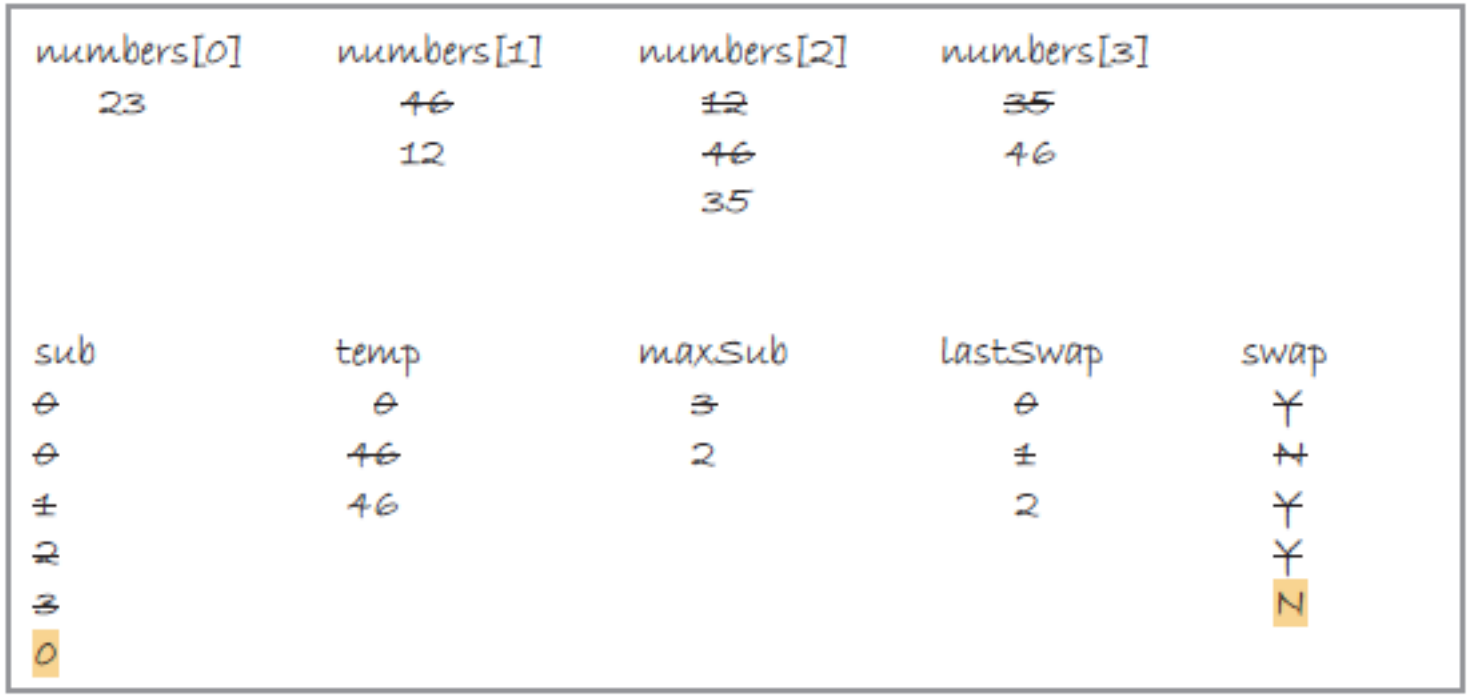



Figure 11-47 Desk-check table after the instructions on lines 21 and 23 are processed

# Sorting the Data Stored in a One-Dimensional Array (cont'd.)

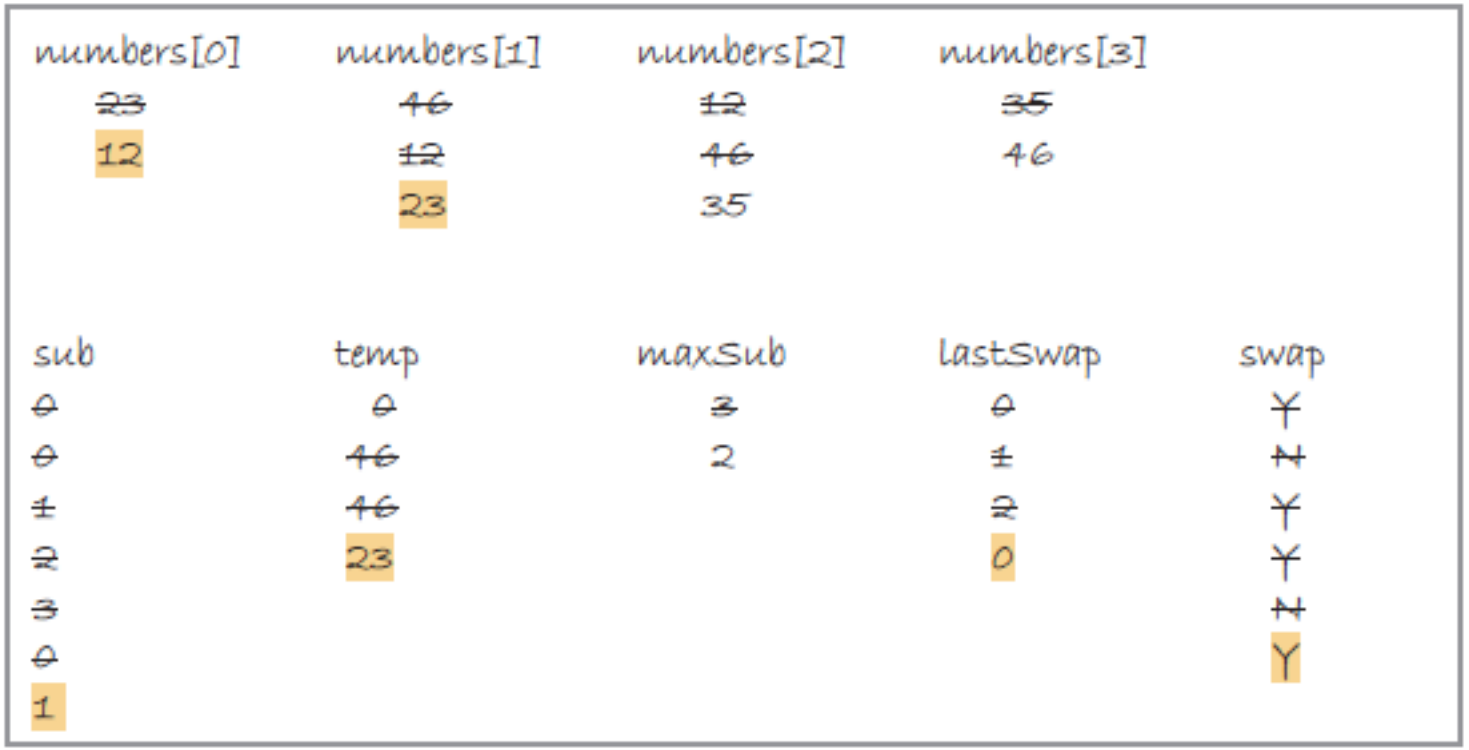
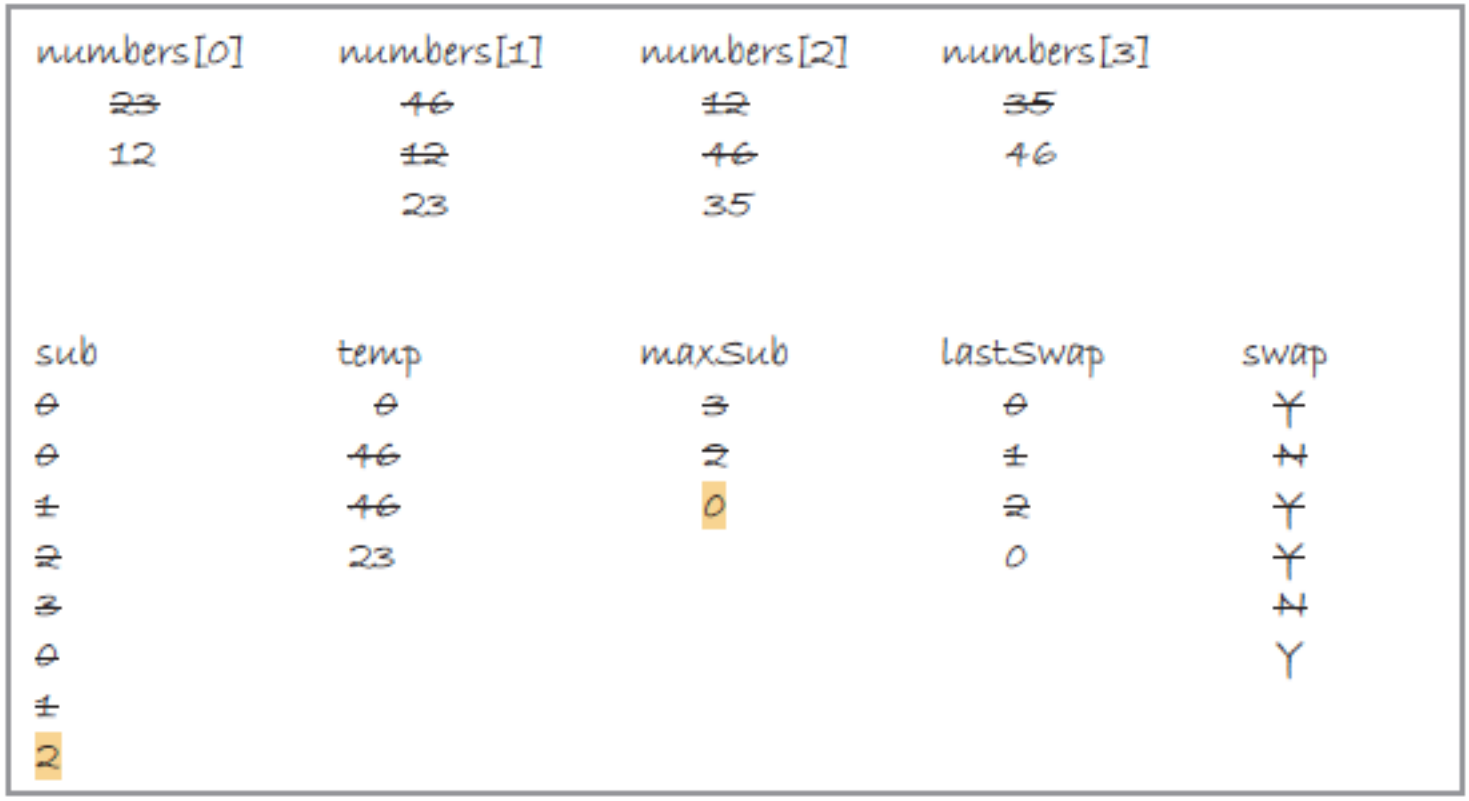



Figure 11-48 Desk-check table after the instructions in the nested loop are processed

# Sorting the Data Stored in a One-Dimensional Array (cont'd.)



| numbers[0] | numbers[1] | numbers[2] | numbers[3] |
|---|---|---|---|
| ~~23~~ | ~~46~~ | ~~12~~ | ~~35~~ |
| 12 | ~~12~~ | ~~46~~ | 46 |
| | 23 | 35 | |

| sub | temp | maxSub | lastSwap | swap |
|---|---|---|---|---|
| ~~0~~ | ~~0~~ | ~~3~~ | ~~0~~ | ~~Y~~ |
| ~~0~~ | ~~46~~ | ~~2~~ | ~~1~~ | ~~N~~ |
| ~~1~~ | ~~46~~ | 0 | ~~2~~ | ~~Y~~ |
| ~~2~~ | 23 | | 0 | ~~Y~~ |
| ~~3~~ | | | | ~~N~~ |
| ~~0~~ | | | | Y |
| ~~1~~ | | | | |
| 2 | | | | |

Figure 11-49 Desk-check table after the instructions in the nested loop are processed again

# Sorting the Data Stored in a One-Dimensional Array (cont'd.)

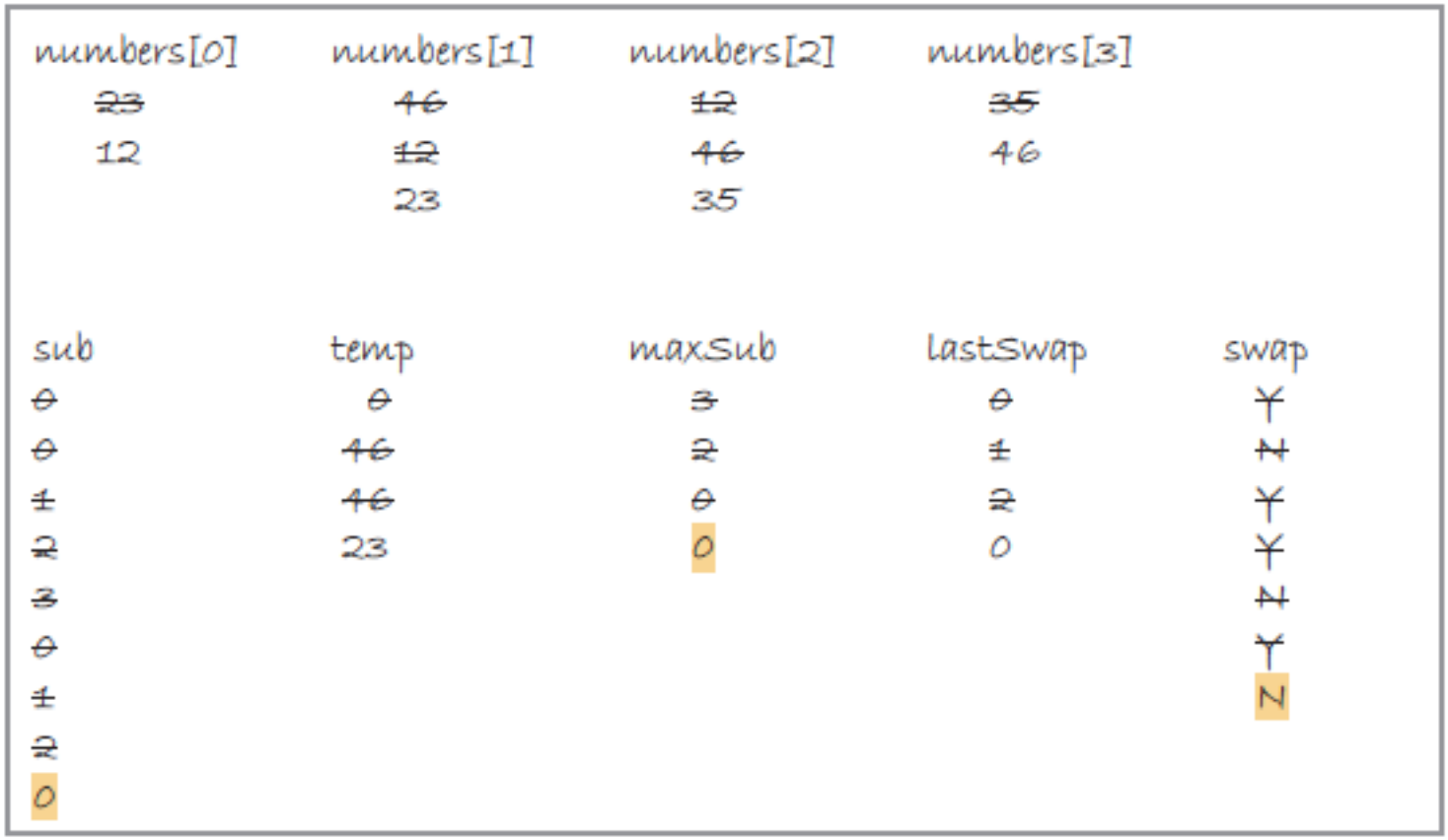



Figure 11-50 Current status of the desk-check table

# Sorting the Data Stored in a One-Dimensional Array (cont'd.)

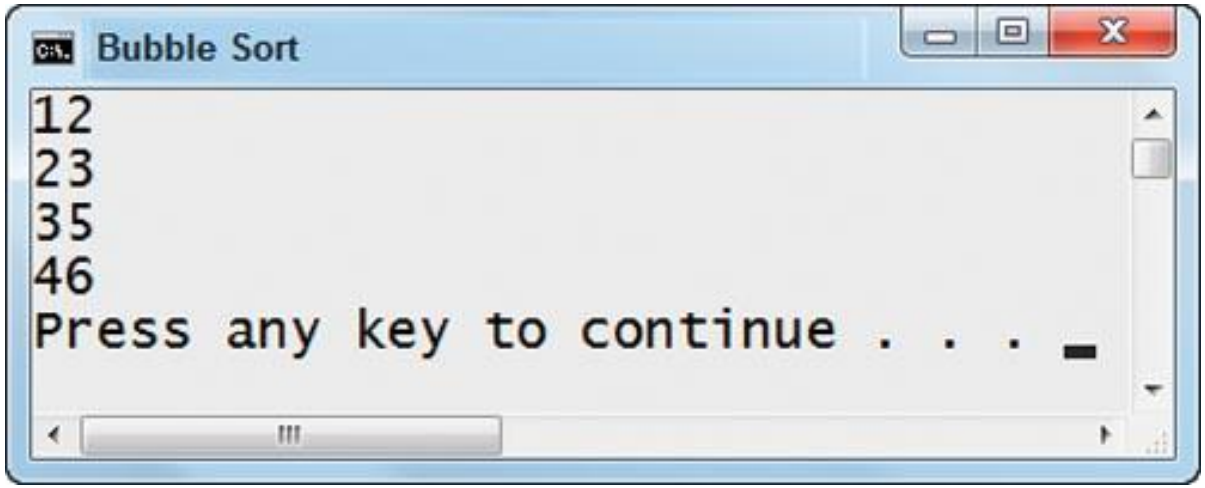



Figure 11-51 Result of running the bubble sort program

# Parallel One-Dimensional Arrays

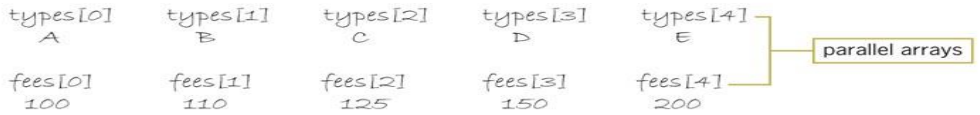
- Program for a motorcycle club displays annual fee associated with membership type entered by user
- Program uses two parallel one-dimensional arrays
  - `char` array named `types`: stores the five membership types
  - `int` array named `fees`: stores annual fee associated with each type
- Two arrays are referred to as **parallel arrays** if their elements are related by their position in the arrays

# Parallel One-Dimensional Arrays (cont'd.)

**Problem specification**

The members of a local motorcycle club are required to pay an annual fee based on their membership type. Create a program that displays a member's annual fee and membership type. The membership types and associated fees are shown here. Use a one-dimensional `char` array named `types` to store the membership types. Use a one-dimensional `int` array named `fees` to store the annual fees.

| Membership type | Annual fee |
|---|---|
| A | 100 |
| B | 110 |
| C | 125 |
| D | 150 |
| E | 200 |




**Input**

membership type
(A, B, C, D, or E)

**Processing**

```
Processing items:
    types array (5 elements)
    fees array (5 elements)
    subscript (counter: 0 to 4)

Algorithm:
1. enter the membership type
2. repeat while (the subscript is less than 5 and
   the membership type has not been located in
   the types array)
        add 1 to the subscript
   end repeat
3. if (the subscript is less than 5)
        display types[subscript] and fees[subscript]
   else
        display "Invalid membership type"
   end if
```

**Output**

fee
membership type

Figure 11-52 Problem specification and IPO chart information for the club membership program

# Parallel One-Dimensional Arrays (cont'd.)

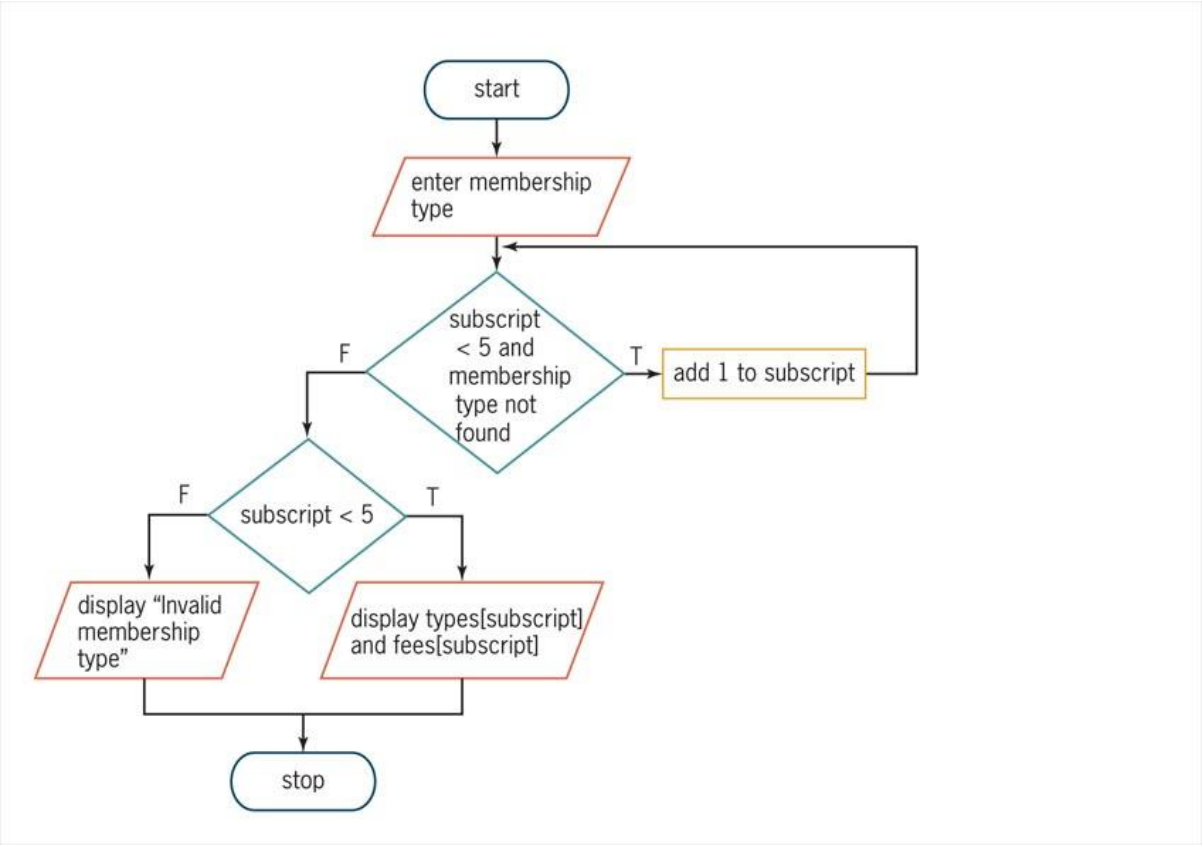



Figure 11-52 Problem specification and IPO chart information for the club membership program (cont.)

# Parallel One-Dimensional Arrays (cont'd.)

| IPO chart information | C++ instructions |
|---|---|
| **Input** | |
| membership type (A, B, C, D, or E) | `char memberType = ' ';` |
| **Processing** | |
| types array (5 elements) | `char types[5] = {'A', 'B', 'C', 'D', 'E'};` |
| fees array (5 elements) | `int fees[5] = {100, 110, 125, 150, 200};` |
| subscript (counter: 0 to 4) | `int sub = 0;` |
| **Output** | |
| fee | from the fees array |
| membership type | from the types array |

Figure 11-53 IPO chart information and C++ instructions for the club membership program

# Parallel One-Dimensional Arrays (cont'd.)

| Algorithm | |
|---|---|
| 1. enter the membership type | `cout << "Membership type (A, B, C, D, or E): ";`<br>`cin >> memberType;`<br>`memberType = toupper(memberType);` |
| 2. repeat while (the subscript is less than 5 and the membership type has not been located in the types array)<br>add 1 to the subscript<br>end repeat | `while (sub < 5 && types[sub] != memberType)`<br>`sub += 1;`<br>`//end while` |
| 3. if (the subscript is less than 5)<br>display types[subscript] and fees[subscript]<br>else<br>display "Invalid membership type"<br>end if | `if (sub < 5)`<br>`cout << "Annual fee for membership type " << types[sub] << ": $" << fees[sub] << endl;`<br>`else`<br>`cout << "Invalid membership type" << endl;`<br>`//end if` |

Figure 11-53 IPO chart information and C++ instructions for the club membership program (cont.)

# Parallel One-Dimensional Arrays (cont'd.)

```
1  //Club Membership.cpp - displays the annual
2  //membership fee
3  //Created/revised by <your name> on <current date>
4
5  #include <iostream>
6  using namespace std;
7
8  int main()
9  {
10     //declare arrays
11     char types[5]   = {'A', 'B', 'C', 'D', 'E'};
12     int fees[5]     = {100, 110, 125, 150, 200};
13     //declare variables
14     char memberType = ' ';
15     int sub         = 0;
16
17     //get type to search for
18     cout << "Membership type (A, B, C, D, or E): ";
19     cin >> memberType;
20     memberType = toupper(memberType);
21
22     //locate the position of the membership
23     //type in the types array
24     while (sub < 5 && types[sub] != memberType)
25           sub += 1;
26     //end while
27
```

parallel one-dimensional arrays

Figure 11-54 Club membership program

# Parallel One-Dimensional Arrays (cont'd.)

```
28      //if the membership type was located in the
29      //types array, display the membership type
30      //and the corresponding fee
31      if (sub < 5)
32            cout << "Annual fee for membership type "
33            << types[sub] << ": $" << fees[sub] << endl;
34      else
35            cout << "Invalid membership type" << endl;
36      //end if
37
38      system("pause");
39      return 0;
40  }   //end of main function
```

your C++ development tool may not require this statement

Figure 11-54 Club membership program (cont'd.)

# Parallel One-Dimensional Arrays (cont'd.)

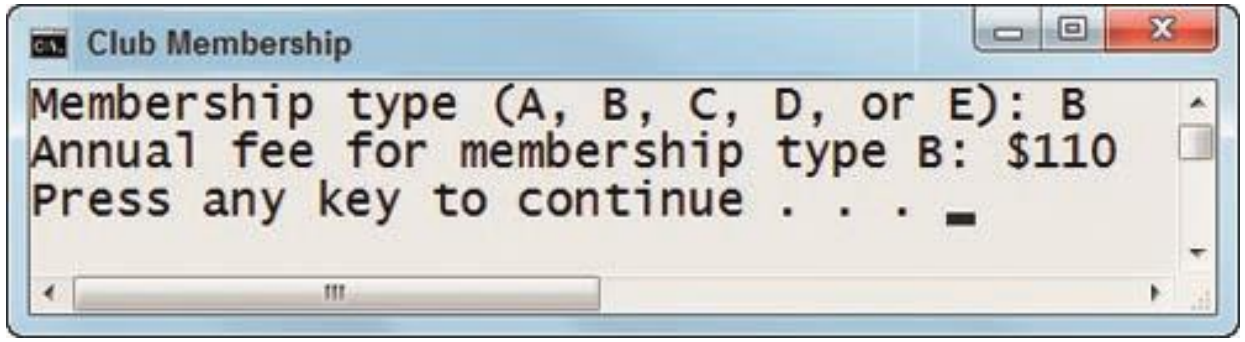



Figure 11-55 Sample run of the club membership program

# Summary

- An array is a group of variables that have the same name and data type
- One- and two-dimensional arrays are the most common types
- Arrays are often used to store related data in internal memory: more efficient to access than from disk
- Must declare an array before using it
- After declaration, you can use an assignment statement or extraction operator to enter data into it

# Summary (cont'd.)

- Each element of a one-dimensional array is assigned a unique number, called a subscript
- First element is assigned a subscript of 0, second a subscript of 1, and so on
- Last subscript of a one-dimensional array is always one number less than the number of elements
- You refer to an element by the array's name followed by the subscript in square brackets
- Parallel arrays are two or more arrays whose elements are related by their position in the arrays

# Lab 11-1: Stop and Analyze

- Study the code in Figure 11-56 and then answer the questions
- The `domestic` array contains the amounts the company sold domestically during the months January through June
- The `international` array contains the amounts the company sold internationally during the same period

# Lab 11-2: Plan and Create

- Plan and create a program for Penelope Havert

**Problem specification**

Penelope Havert wants a program that allows her to enter the monthly rainfall amounts for the previous year. The program then should allow her to either display the monthly rainfall amounts on the screen or calculate and display the total annual rainfall amount. In this program, you will use two program-defined void functions named `displayMonthly` and `displayTotal`. Both functions will be passed the contents of a one-dimensional array named `rainfall`, along with the number of array elements. The `rainfall` array will contain the 12 monthly rainfall amounts entered by the user. Void functions are appropriate in this case because neither function needs to return a value to the statement that called it.

Figure 11-57 Problem specification for Lab 11-2

# Lab 11-3: Modify

- Modify the program in Lab 11-2
- Change the void `displayTotal` function with a value-returning function named `getTotal`
  - Function should calculate total rainfall and return value to `main` function to be displayed
- `main` function should display an error message when user enters a menu choice other than 1, 2, or 3 and should then display the menu again
- Test the program appropriately

# Lab 11-4: Desk-Check

- Desk-check the code in Figure 11-60 using the data shown below
- What will the `for` loop on Lines 31 through 34 display on the screen?

| Student | Midterm | Final |
|---|---|---|
| 1 | 90 | 100 |
| 2 | 88 | 68 |
| 3 | 77 | 75 |
| 4 | 85 | 85 |
| 5 | 45 | 32 |

# Final Project

- C++ final project
- Go over rubric for final
- Your partner
- Design Process

- Objective

The final is a culmination of everything you have learned through the class hitting every standard that Scott Community College expected you to learn through the course

# Lab 11-5: Debug

- Open the program in the Lab11-5.cpp file
- Debug the program