Arrays

Rahul Deodhar

www.rahuldeodhar.com

@rahuldeodhar

rahuldeodhar@gmail.com

+91 98202 13813

Array Basics

Introduction to Arrays

- An **array** is used to process a collection of data of the same type
  - Examples: A list of names
    A list of temperatures
- Why do we need arrays?
  - Imagine keeping track of 5 test scores, or 100, or 1000 in memory
    - How would you name all the variables?
    - How would you process each of the variables?

Declaring an Array

- An array, named score, containing five variables of type int can be declared as

```
int score[ 5 ];
```

- This is like declaring 5 variables of type int:

  score[0], score[1], … , score[4]

- The value in brackets is called
  - A subscript
  - An index

The Array Variables

- The variables making up the array are referred to is
  - **Indexed variables**
  - **Subscripted variables**
  - **Elements of the array**
- The number of indexed variables in an array is the **declared size**, or **size**, of the array
  - The largest index is one less than the size
  - The first index value is zero

Array Variable Types

- An array can have indexed variables of any type

- All indexed variables in an array are of the same type
  - This is the **base type** of the array

- An indexed variable can be used anywhere an ordinary variable of the base type is used

Using [ ] With Arrays

- ❑ In an array declaration, [ ]'s enclose the <u>size</u> of the array such as this array of 5 integers:

  int score [5];

- ❑ When referring to one of the indexed variables, the [ ]'s enclose a number identifying one of the indexed variables
  - ❑ score[3] is one of the indexed variables
  - ❑ The value in the [ ]'s can be any expression that evaluates to one of the integers 0 to (size -1)

Indexed Variable Assignment

- To assign a value to an indexed variable, use the assignment operator:

```
int n = 2;
score[n + 1] = 99;
```

  - In this example, variable score[3] is assigned 99

Loops And Arrays

- for-loops are commonly used to step through arrays
  - Example:

    ```
    for (i = 0; i < 5; i++)
    {
       cout << score[i] << " off by "
              << (max – score[i]) << endl;
     }
    ```

    could display the difference between each score and the maximum score stored in an array.
  - Index size starts with 0 and ends with (size - 1)

Program Using an Array

```
//Reads in 5 scores and shows how much each
//score differs from the highest score.
#include <iostream>

int main( )
{
    using namespace std;
    int i, score[5], max;

    cout << "Enter 5 scores:\n";
    cin >> score[0];
    max = score[0];
    for (i = 1; i < 5; i++)
    {
        cin >> score[i];
        if (score[i] > max)
            max = score[i];
        //max is the largest of the values score[0],..., score[i].
    }

    cout << "The highest score is " << max << endl
         << "The scores and their\n"
         << "differences from the highest are:\n";
    for (i = 0; i < 5; i++)
        cout << score[i] << " off by "
             << (max – score[i]) << endl;

    return 0;
}
```

Sample Dialogue

```
Enter 5 scores:
5 9 2 10 6
The highest score is 10
The scores and their
differences from the highest are:
5 off by 5
9 off by 1
2 off by 8
10 off by 0
6 off by 4
```

Constants and Arrays

- Use constants to declare the size of an array
  - Using a constant allows your code to be easily altered for use on a smaller or larger set of data
    - Example:

```
const int  NUMBER_OF_STUDENTS = 50;
int score[NUMBER_OF_STUDENTS];
…
for ( i = 0; i < NUMBER_OF_STUDENTS; i++)
cout << score[i] << " off by "
     << (max – score[i]) << endl;
```

    - Only the value of the constant must be changed to make this code work for any number of students

Variables and Declarations

- ❑ Most compilers do not allow the use of a variable
  to declare the size of an array

Example:

```
cout << "Enter number of students: ";
cin >> number;
int score[number];
```

- ❑ This code is illegal on many compilers

Array Declaration Syntax

- ❑ To declare an array, use the syntax:

  ***Type_Name Array_Name*[*Declared_Size*];**

  - ❑ *Type_Name* can be any type
  - ❑ *Declared_Size* can be a constant to make your program more versatile

- ❑ Once declared, the array consists of the indexed variables:

  Array_Name[0] to *Array_Name*[Declared_Size -1]

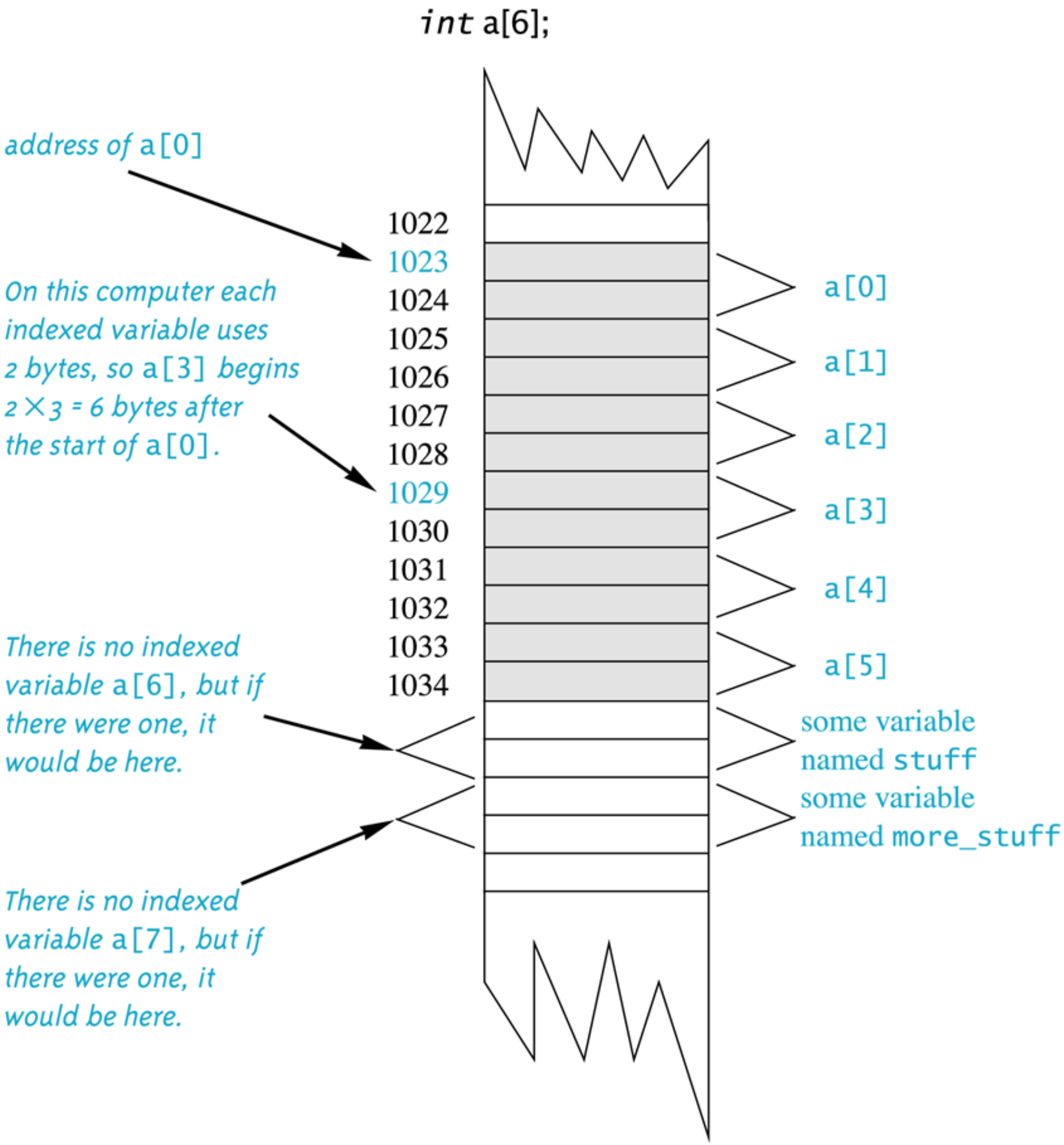
Computer Memory

- ❑ Computer memory consists of numbered locations called bytes
  - ❑ A byte's number is its **address**
- ❑ A simple variable is stored in consecutive bytes
  - ❑ The number of bytes depends on the variable's type
- ❑ A variable's address is the address of its first byte

Arrays and Memory

- Declaring the array int a[6]
  - Reserves memory for six variables of type int
  - The variables are stored one after another
  - The address of a[0] is remembered
    - The addresses of the other indexed variables is not remembered
  - To determine the address of a[3]
    - Start at a[0]
    - Count past enough memory for three integers to find a[3]

An Array in Memory



int a[6];

address of a[0]

On this computer each indexed variable uses 2 bytes, so a[3] *begins 2 × 3 = 6 bytes after the start of* a[0].

There is no indexed variable a[6], *but if there were one, it would be here.*

There is no indexed variable a[7], *but if there were one, it would be here.*

Array Index Out of Range

- A common error is using a nonexistent index
  - Index values for int a[6] are the values 0 through 5
  - An index value not allowed by the array declaration is **out of range**
  - Using an out of range index value **does not** produce an error message!

Out of Range Problems

- ❑ If an array is declared as: **int a[6];**
  and an integer is declared as: **int i = 7;**
- ❑ Executing the statement **a[i] = 238;** causes…
  - ❑ The computer to calculate the address of the illegal a[7]

    (This address could be where some other variable is stored)

    The value 238 is stored at the address calculated for a[7]
  - ❑ No warning is given!

Initializing Arrays

- To initialize an array when it is declared
  - The values for the indexed variables are enclosed in braces and separated by commas
- Example: **int children[3] = { 2, 12, 1 };**

  Is equivalent to:

```
int children[3];
children[0] = 2;
children[1] = 12;
children[2] = 1;
```

Default Values

- If too few values are listed in an initialization statement
  - The listed values are used to initialize the first of the indexed variables
  - The remaining indexed variables are initialized to a zero of the base type
  - Example: **int a[10] = {5, 5};**
    initializes a[0] and a[1] to 5 and
    a[2] through a[9] to 0

Un-initialized Arrays

- If no values are listed in the array declaration, some compilers will initialize each variable to a zero of the base type
  - DO NOT DEPEND ON THIS!

Class Work

- Can you
  - Describe the difference between **a[4]** and **int a[5]**?
  - Show the output of

```
char symbol[3] = {'a', 'b', 'c'};
for (int index = 0; index < 3; index++)
   cout << symbol[index];
```

Arrays in Functions

Arrays in Functions

- ❑ Indexed variables can be arguments to functions

Example: If a program contains these declarations:

```
int i, n, a[10];
void my_function(int n);
```

Variables a[0] through a[9] are of type int, making these calls legal:

```
my_function( a[ 0 ] );
my_function( a[ 3 ] );
my_function( a[  i ]  );
```

Indexed Variable as an Argument

```
//Illustrates the use of an indexed variable as an argument.
//Adds 5 to each employee's allowed number of vacation days.
#include <iostream>

const int NUMBER_OF_EMPLOYEES = 3;

int adjust_days(int old_days);
//Returns old_days plus 5.

int main( )
{
    using namespace std;
    int vacation[NUMBER_OF_EMPLOYEES], number;

    cout << "Enter allowed vacation days for employees 1"
         << " through " << NUMBER_OF_EMPLOYEES << ":\n";
    for (number = 1; number <= NUMBER_OF_EMPLOYEES; number++)
        cin >> vacation[number-1];

    for (number = 0; number < NUMBER_OF_EMPLOYEES; number++)
        vacation[number] = adjust_days(vacation[number]);

    cout << "The revised number of vacation days are:\n";
    for (number = 1; number <= NUMBER_OF_EMPLOYEES; number++)
        cout << "Employee number " << number
             << " vacation days = " << vacation[number-1] << endl;

    return 0;
}

int adjust_days(int old_days)
{
    return (old_days + 5);
}
```

Sample Dialogue

```
Enter allowed vacation days for employees 1 through 3:
10 20 5
The revised number of vacation days are:
Employee number 1 vacation days = 15
Employee number 2 vacation days = 25
Employee number 3 vacation days = 10
```

Arrays as Function Arguments

- ❑ A formal parameter can be for an entire array
  - ❑ Such a parameter is called an **array parameter**
    - ❑ It is not a call-by-value parameter
    - ❑ It is not a call-by-reference parameter
    - ❑ Array parameters behave much like call-by-reference parameters

Array Parameter Declaration

- An array parameter is indicated using empty brackets in the parameter list such as

```
void fill_up(int a[ ], int size);
```

Function Calls With Arrays

- If function fill_up is declared in this way:

  void fill_up(int a[ ], int size);

  and array score is declared this way:

  int score[5], number_of_scores;

  fill_up is called in this way:

  fill_up(score, number_of_scores);

Function with an Array Parameter

Function Declaration

```
void fill_up(int a[], int size);
//Precondition: size is the declared size of the array a.
//The user will type in size integers.
//Postcondition: The array a is filled with size integers
//from the keyboard.
```

Function Definition

```
//Uses iostream:
void fill_up(int a[], int size)
{
    using namespace std;
    cout << "Enter " << size << " numbers:\n";
    for (int i = 0; i < size; i++)
        cin >> a[i];
    size--;
    cout << "The last array index used is " << size << endl;
}
```

Function Call Details

- ❑ A formal parameter is identified as an array parameter by the [ ]'s with no index expression

```
void fill_up(int a[ ], int size);
```

- ❑ An array argument does not use the [ ]'s

```
fill_up(score, number_of_scores);
```

Array Formal Parameters

- An array formal parameter is a placeholder for the argument
  - When an array is an argument in a function call, an action performed on the array parameter is performed on the array argument

  - The values of the indexed variables can be changed by the function

Array Argument Details

- What does the computer know about an array?
  - The base type
  - The address of the first indexed variable
  - The number of indexed variables
- What does a function know about an array argument?
  - The base type
  - The address of the first indexed variable

Array Parameter Considerations

- ❑ Because a function does not know the size of an array argument…
  - ❑ The programmer should include a formal parameter that specifies the size of the array
  - ❑ The function can process arrays of various sizes
    - ❑ Function fill_up (below) can be used to fill an array of *any* size:

```
fill_up(score, 5);
fill_up(time, 10);
```

const Modifier

- ❑ Array parameters allow a function to change the values stored in the array argument
- ❑ If a function should not change the values of the array argument, use the modifier **const**
- ❑ An array parameter modified with const is a **constant array parameter**
  - ❑ Example:

```
void show_the_world(const int a[ ], int size);
```

Using const With Arrays

- If const is used to modify an array parameter:
  - const is used in both the function declaration and definition to modify the array parameter
  - The compiler will issue an error if you write code that changes the values stored in the array parameter

Function Calls and const

- ❑ If a function with a constant array parameter calls another function using the const array parameter as an argument…

  - ❑ The called function must use a constant array parameter as a placeholder for the array

  - ❑ The compiler will issue an error if a function is called that does not have a const array parameter to
    accept the array argument

const Parameters Example

- **double compute_average(int a[ ], int size);**

  ```
  void show_difference(const int a[ ], int size)
  {
      double average = compute_average(a, size);
       ...
  }
  ```

- compute_average has no constant array parameter
- This code generates an error message because compute_average could change the array parameter

Returning An Array

- ❑ Recall that functions can return a value of type int, double, char, …, or a class type

- ❑ Functions cannot return arrays

- ❑ We learn later how to return a pointer to an array

Case Study: Production Graph

- Problem Definition:
  - We are writing a program for the Apex Plastic Spoon Company
  - The program will display a bar graph showing the production of each of four plants for a week
  - Each plant has separate records for each department
  - Input is entered plant by plant
  - Output shows one asterisk for each 1000 units, and production is rounded to the nearest 1,000 units

Analysis of The Problem

- Use an array named **production** to hold total production of each plant
  - Production for plant **n** is stored in production[**n-1**]

- Program must scale production to nearest 1,000 units to display asterisks in the bar

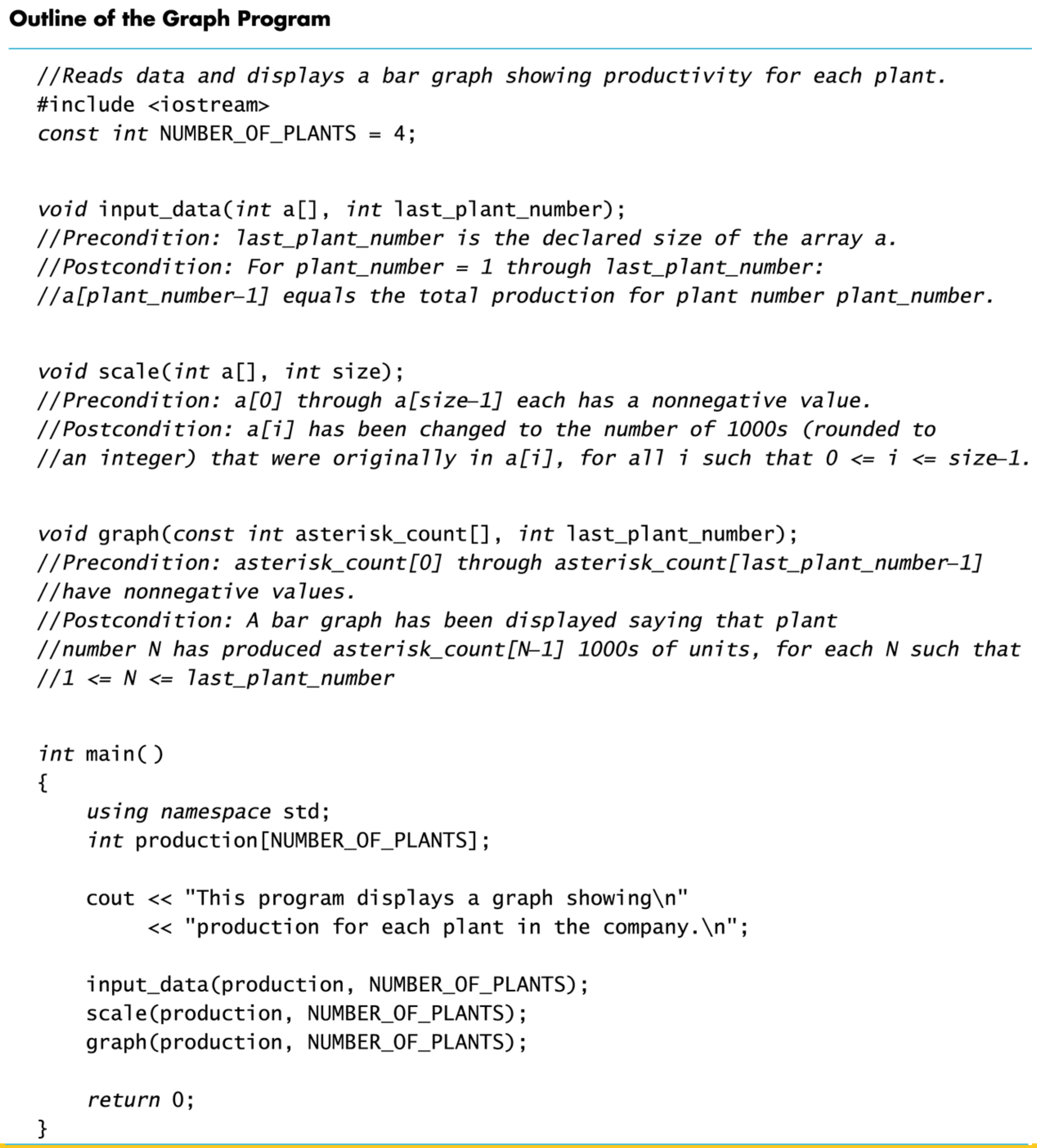
Production Graph Sub-Tasks

- ❑ Analysis leads to the following sub-tasks
  - ❑ input_data: Read input for each plant
    Set production [plant_number -1] to the total production for plant number n
  - ❑ scale: For each plant, change production[plant_number] to the correct number of asterisks
  - ❑ graph: Output the bar graph

More Analysis Details

- ❑ The entire array will be an argument for the functions we write to perform the subtasks
  - ❑ We will also include a formal parameter for the size
  - ❑ The size of the array is equal to the number of plants
  - ❑ We will use a constant for the number of plants

- ❑ The function declarations and main function for the production graph program are found in



Outline of the Graph Program

```
//Reads data and displays a bar graph showing productivity for each plant.
#include <iostream>
const int NUMBER_OF_PLANTS = 4;


void input_data(int a[], int last_plant_number);
//Precondition: last_plant_number is the declared size of the array a.
//Postcondition: For plant_number = 1 through last_plant_number:
//a[plant_number-1] equals the total production for plant number plant_number.


void scale(int a[], int size);
//Precondition: a[0] through a[size-1] each has a nonnegative value.
//Postcondition: a[i] has been changed to the number of 1000s (rounded to
//an integer) that were originally in a[i], for all i such that 0 <= i <= size-1.


void graph(const int asterisk_count[], int last_plant_number);
//Precondition: asterisk_count[0] through asterisk_count[last_plant_number-1]
//have nonnegative values.
//Postcondition: A bar graph has been displayed saying that plant
//number N has produced asterisk_count[N-1] 1000s of units, for each N such that
//1 <= N <= last_plant_number


int main( )
{
    using namespace std;
    int production[NUMBER_OF_PLANTS];

    cout << "This program displays a graph showing\n"
         << "production for each plant in the company.\n";

    input_data(production, NUMBER_OF_PLANTS);
    scale(production, NUMBER_OF_PLANTS);
    graph(production, NUMBER_OF_PLANTS);

    return 0;
}
```

Algorithm Design: input_data

- We must read all departments' data for each plant and add them to produce a plant's total
  - Algorithm for input_data:
    for plant_number is 1, 2, ..., last_plant_number

    do the following
  - Read all the data for plant number plant_number
  - Sum the numbers
  - Set production[plant_number – 1] to the total

Coding input_data

- The algorithm can be translated to C++ as:

```
void input_data(int a [ ], int last_plant_number)
{
        using namespace std;

        for (int plant_number = 1;
        plant_number <= last_plant_number;
        plant_number++)
        {
          cout << endl;
                << "Enter production for plant"
                << plant_number << endl;
         get_total( a[plant_number -1] );
        }
}
```

Testing input_data

- ❑ Each function should be tested in a program in which it is the only untested function
- ❑ Because input_data calls get_total, get_total is tested first
- ❑ Once tested, get_total can be used to test input_data

Test of Function input_data (*part 1 of 3*)

```
//Tests the function input_data.
#include <iostream>
const int NUMBER_OF_PLANTS = 4;

void input_data(int a[], int last_plant_number);
//Precondition: last_plant_number is the declared size of the array a.
//Postcondition: For plant_number = 1 through last_plant_number:
//a[plant_number-1] equals the total production for plant number plant_number.

void get_total(int& sum);
//Reads nonnegative integers from the keyboard and
//places their total in sum.

int main( )
{
    using namespace std;
    int production[NUMBER_OF_PLANTS];
    char ans;

    do
    {
        input_data(production, NUMBER_OF_PLANTS);
        cout << endl
             << "Total production for each"
             << " of plants 1 through 4:\n";
        for (int number = 1; number <= NUMBER_OF_PLANTS; number++)
        cout << production[number - 1] << " ";

        cout << endl
             << "Test Again?(Type y or n and Return): ";
        cin >> ans;
    }while ( (ans != ’N’) && (ans != ’n’) );

    cout << endl;

    return 0;
}
```

Test of Function input_data (*part 2 of 3*)

```
//Uses iostream:
void input_data(int a[], int last_plant_number)
{
    using namespace std;
    for (int plant_number = 1;
                   plant_number <= last_plant_number; plant_number++)
    {
        cout << endl
             << "Enter production data for plant number "
             << plant_number << endl;
        get_total(a[plant_number - 1]);
    }
}


//Uses iostream:
void get_total(int& sum)
{
    using namespace std;
    cout << "Enter number of units produced by each department.\n"
         << "Append a negative number to the end of the list.\n";

    sum = 0;
    int next;
    cin >> next;
    while (next >= 0)
    {
        sum = sum + next;
        cin >> next;
    }

    cout << "Total = " << sum << endl;
}
```

Test of Function input_data (*part 3 of 3*)

Sample Dialogue

```
Enter production data for plant number 1
Enter number of units produced by each department.
Append a negative number to the end of the list.
1 2 3 -1
Total = 6

Enter production data for plant number 2
Enter number of units produced by each department.
Append a negative number to the end of the list.
0 2 3 -1
Total = 5

Enter production data for plant number 3
Enter number of units produced by each department.
Append a negative number to the end of the list.
2 -1
Total = 2

Enter production data for plant number 4
Enter number of units produced by each department.
Append a negative number to the end of the list.
-1
Total = 0

Total production for each of plants 1 through 4:
6 5 2 0
Test Again?(Type y or n and Return): n
```

Test Data for input_data

- Remember that input_data should be tested
  - With a plant that contains no production figures
  - With a plant having only one production figure
  - With a plant having more than one figure
  - With zero and non-zero production figures

Algorithm for scale

❑ Scale changes the value of the indexed variable to show the whole number of asterisks to print

❑ Scale is called using
scale (production, NUMBER_OF_PLANTS);

and its algorithm is

```
for (int index = 0; index < size; index++)
  Divide the value of a[index] by 1,000 and round the result to the nearest integer
```

Coding scale

- ❑ The code for scale, below, uses a function named round that must be defined as well

```
void scale(int a[ ], int size)
{
    for (int index = 0; index < size; index++)
     a[index] = round (a[index] / 1000.0);
}
```

Why not 1000?

Function floor

- Function round, called by scale, uses the floor function from the cmath library
  - The floor function returns the first whole number less than its argument:

    floor (3.4) returns 3
    floor (3.9) returns 3

  - Adding 0.5 to the argument for floor is how round performs its task

    floor (3.4 + 0.5) returns 3
    floor (3.9 + 0.5) returns 4

Testing scale

- To test scale
  - First test round
  - Scale should be tested with arguments that
    - Are 0
    - Round up
    - Round down

The Function scale (*part 1 of 2*)

```
//Demonstration program for the function scale.
#include <iostream>
#include <cmath>

void scale(int a[], int size);
//Precondition: a[0] through a[size–1] each has a nonnegative value.
//Postcondition: a[i] has been changed to the number of 1000s (rounded to
//an integer) that were originally in a[i], for all i such that 0 <= i <= size–1.

int round(double number);
//Precondition: number >= 0.
//Returns number rounded to the nearest integer.

int main( )
{
    using namespace std;
    int some_array[4], index;

    cout << "Enter 4 numbers to scale: ";
    for (index = 0; index < 4; index++)
        cin >> some_array[index];

    scale(some_array, 4);

    cout << "Values scaled to the number of 1000s are: ";
    for (index = 0; index < 4; index++)
        cout << some_array[index] << " ";
    cout << endl;

    return 0;
}

void scale(int a[], int size)
{
    for (int index = 0; index < size; index++)
        a[index] = round(a[index]/1000.0);
}
```

The Function scale (*part 2 of 2*)

```
//Uses cmath:
int round(double number)
{
    using namespace std;
    return static_cast<int>(floor(number + 0.5));
}
```

Sample Dialogue

```
Enter 4 numbers to scale: 2600 999 465 3501
Values scaled to the number of 1000s are: 3 1 0 4
```

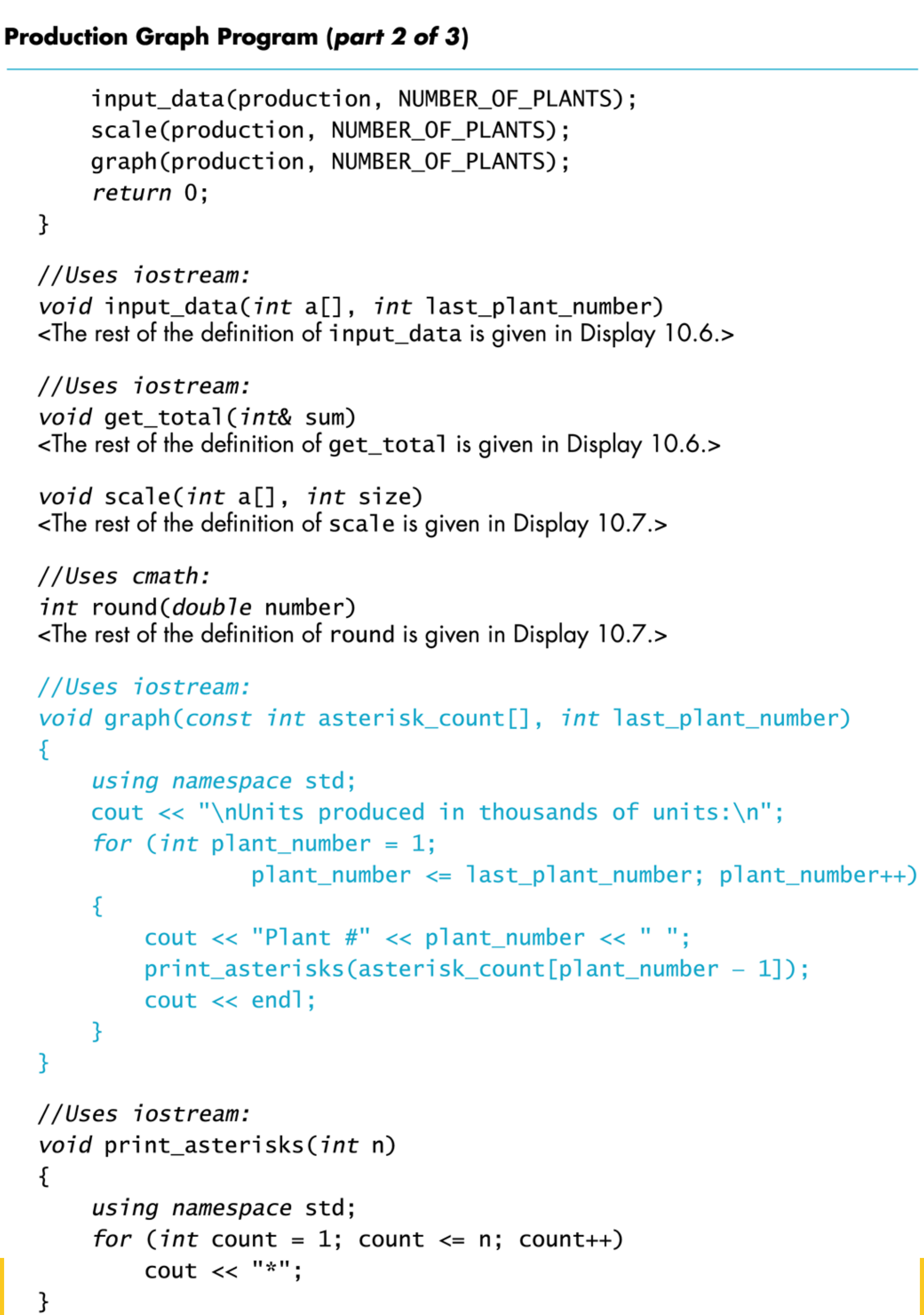
Function graph

- ❑ The design of graph is quite straightforward and not included here

- ❑ The complete program to produce the bar graph is found in

Production Graph Program (*part 1 of 3*)

```cpp
//Reads data and displays a bar graph showing productivity for each plant.
#include <iostream>
#include <cmath>
const int NUMBER_OF_PLANTS = 4;

void input_data(int a[], int last_plant_number);
//Precondition: last_plant_number is the declared size of the array a.
//Postcondition: For plant_number = 1 through last_plant_number:
//a[plant_number-1] equals the total production for plant number plant_number.

void scale(int a[], int size);
//Precondition: a[0] through a[size-1] each has a nonnegative value.
//Postcondition: a[i] has been changed to the number of 1000s (rounded to
//an integer) that were originally in a[i], for all i such that 0 <= i <= size-1.

void graph(const int asterisk_count[], int last_plant_number);
//Precondition: asterisk_count[0] through asterisk_count[last_plant_number-1]
//have nonnegative values.
//Postcondition: A bar graph has been displayed saying that plant
//number N has produced asterisk_count[N-1] 1000s of units, for each N such that
//1 <= N <= last_plant_number

void get_total(int& sum);
//Reads nonnegative integers from the keyboard and
//places their total in sum.

int round(double number);
//Precondition: number >= 0.
//Returns number rounded to the nearest integer.

void print_asterisks(int n);
//Prints n asterisks to the screen.

int main( )
{
    using namespace std;
    int production[NUMBER_OF_PLANTS];

    cout << "This program displays a graph showing\n"
         << "production for each plant in the company.\n";
```



Production Graph Program (*part 2 of 3*)

```
    input_data(production, NUMBER_OF_PLANTS);
    scale(production, NUMBER_OF_PLANTS);
    graph(production, NUMBER_OF_PLANTS);
    return 0;
}

//Uses iostream:
void input_data(int a[], int last_plant_number)
<The rest of the definition of input_data is given in Display 10.6.>

//Uses iostream:
void get_total(int& sum)
<The rest of the definition of get_total is given in Display 10.6.>

void scale(int a[], int size)
<The rest of the definition of scale is given in Display 10.7.>

//Uses cmath:
int round(double number)
<The rest of the definition of round is given in Display 10.7.>

//Uses iostream:
void graph(const int asterisk_count[], int last_plant_number)
{
    using namespace std;
    cout << "\nUnits produced in thousands of units:\n";
    for (int plant_number = 1;
                plant_number <= last_plant_number; plant_number++)
    {
        cout << "Plant #" << plant_number << " ";
        print_asterisks(asterisk_count[plant_number - 1]);
        cout << endl;
    }
}

//Uses iostream:
void print_asterisks(int n)
{
    using namespace std;
    for (int count = 1; count <= n; count++)
        cout << "*";
}
```

Production Graph Program (*part 3 of 3*)

Sample Dialogue

```
This program displays a graph showing
production for each plant in the company.

Enter production data for plant number 1
Enter number of units produced by each department.
Append a negative number to the end of the list.
2000 3000 1000 -1
Total = 6000

Enter production data for plant number 2
Enter number of units produced by each department.
Append a negative number to the end of the list.
2050 3002 1300 -1
Total = 6352

Enter production data for plant number 3
Enter number of units produced by each department.
Append a negative number to the end of the list.
5000 4020 500 4348 -1
Total = 13868

Enter production data for plant number 4
Enter number of units produced by each department.
Append a negative number to the end of the list.
2507 6050 1809 -1
Total = 10366

Units produced in thousands of units:
Plant #1 ******
Plant #2 ******
Plant #3 **************
Plant #4 **********
```

Class Work

- ❑ Can you

  - ❑ Write a function definition for a function called one_more, which has a formal parameter for an array of integers and increases the value of each array element by one. Are other formal parameters needed?

Programming with Arrays

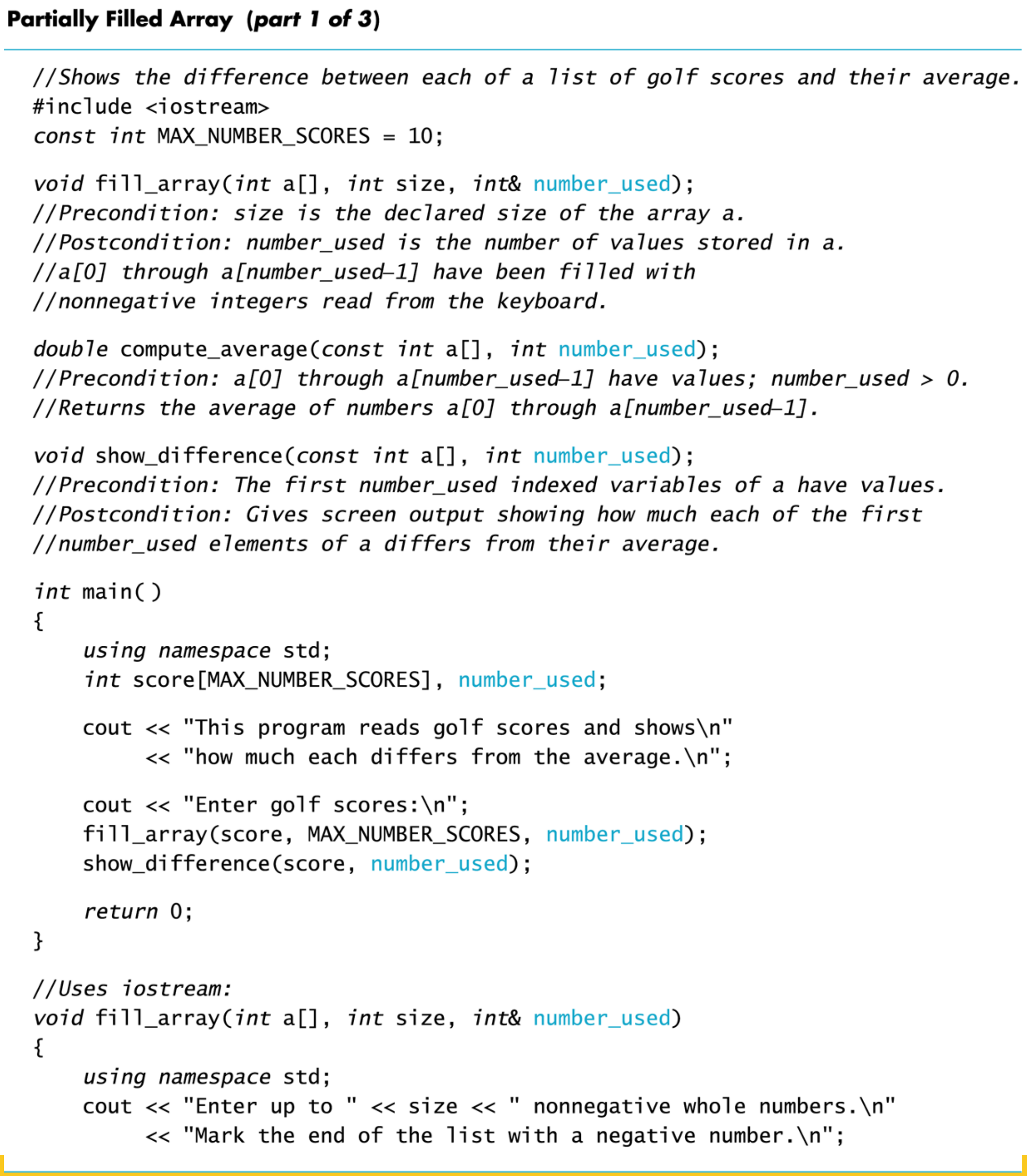
Programming With Arrays

- The size needed for an array is changeable
  - Often varies from one run of a program to another
  - Is often not known when the program is written

- A common solution to the size problem
  - Declare the array size to be the largest that could be needed
  - Decide how to deal with partially filled arrays

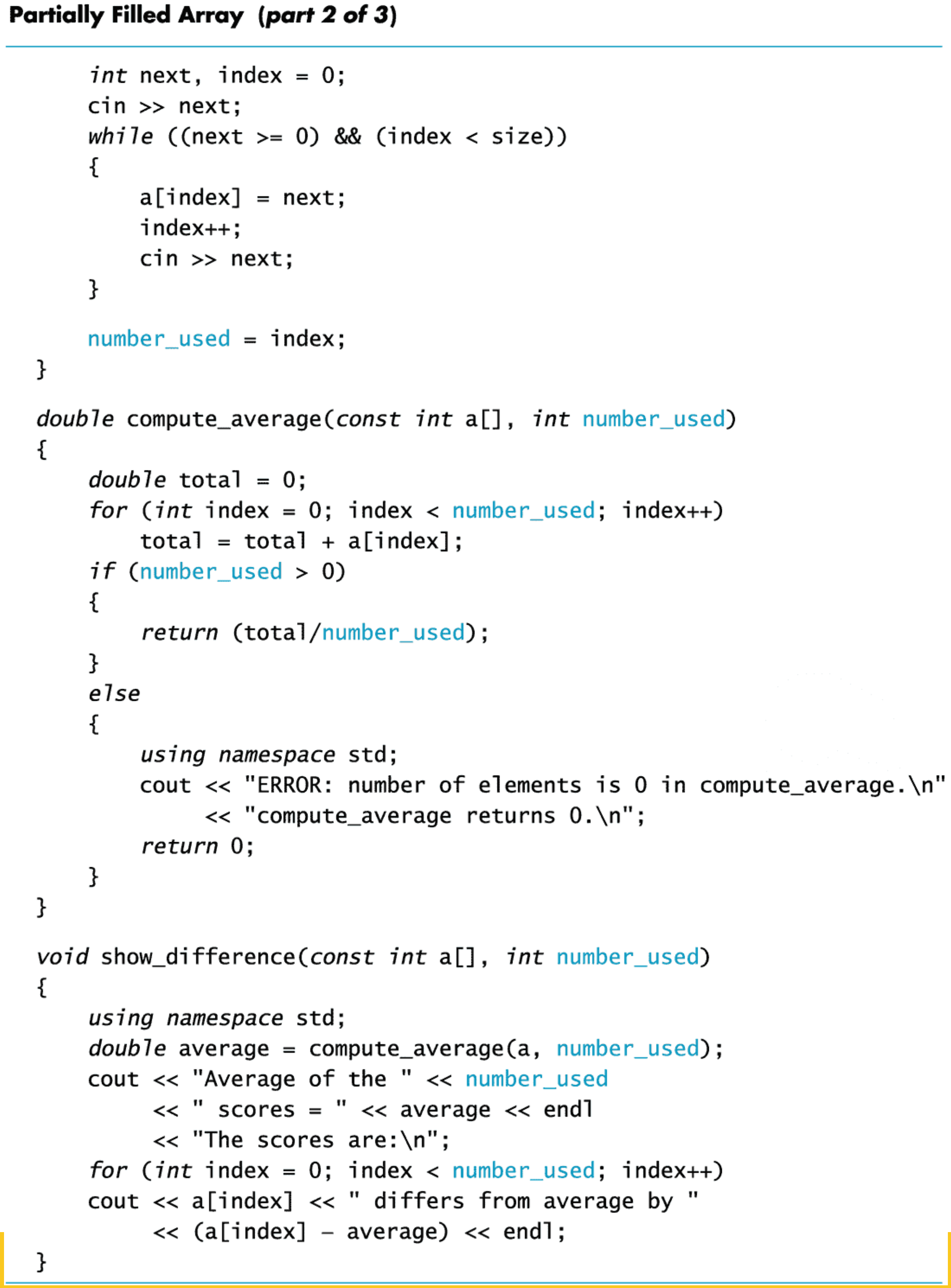
Partially Filled Arrays

- When using arrays that are partially filled
  - Functions dealing with the array may not need to know the declared size of the array, only how many elements are stored in the array
  - A parameter, ***number_used,*** may be sufficient to ensure that referenced index values are legal
  - A function such as fill_array in Display 10.9 needs to know the declared size of the array



Partially Filled Array (*part 1 of 3*)

```
//Shows the difference between each of a list of golf scores and their average.
#include <iostream>
const int MAX_NUMBER_SCORES = 10;

void fill_array(int a[], int size, int& number_used);
//Precondition: size is the declared size of the array a.
//Postcondition: number_used is the number of values stored in a.
//a[0] through a[number_used–1] have been filled with
//nonnegative integers read from the keyboard.

double compute_average(const int a[], int number_used);
//Precondition: a[0] through a[number_used–1] have values; number_used > 0.
//Returns the average of numbers a[0] through a[number_used–1].

void show_difference(const int a[], int number_used);
//Precondition: The first number_used indexed variables of a have values.
//Postcondition: Gives screen output showing how much each of the first
//number_used elements of a differs from their average.

int main( )
{
    using namespace std;
    int score[MAX_NUMBER_SCORES], number_used;

    cout << "This program reads golf scores and shows\n"
         << "how much each differs from the average.\n";

    cout << "Enter golf scores:\n";
    fill_array(score, MAX_NUMBER_SCORES, number_used);
    show_difference(score, number_used);

    return 0;
}

//Uses iostream:
void fill_array(int a[], int size, int& number_used)
{
    using namespace std;
    cout << "Enter up to " << size << " nonnegative whole numbers.\n"
         << "Mark the end of the list with a negative number.\n";
```



Partially Filled Array *(part 2 of 3)*

```
    int next, index = 0;
    cin >> next;
    while ((next >= 0) && (index < size))
    {
        a[index] = next;
        index++;
        cin >> next;
    }

    number_used = index;
}

double compute_average(const int a[], int number_used)
{
    double total = 0;
    for (int index = 0; index < number_used; index++)
        total = total + a[index];
    if (number_used > 0)
    {
        return (total/number_used);
    }
    else
    {
        using namespace std;
        cout << "ERROR: number of elements is 0 in compute_average.\n"
             << "compute_average returns 0.\n";
        return 0;
    }
}

void show_difference(const int a[], int number_used)
{
    using namespace std;
    double average = compute_average(a, number_used);
    cout << "Average of the " << number_used
         << " scores = " << average << endl
         << "The scores are:\n";
    for (int index = 0; index < number_used; index++)
    cout << a[index] << " differs from average by "
         << (a[index] - average) << endl;
}
```

Partially Filled Array (*part 3 of 3*)

Sample Dialogue

```
This program reads golf scores and shows
how much each differs from the average.
Enter golf scores:
Enter up to 10 nonnegative whole numbers.
Mark the end of the list with a negative number.
69 74 68 -1
Average of the 3 scores = 70.3333
The scores are:
69 differs from average by -1.33333
74 differs from average by 3.66667
68 differs from average by -2.33333
```

Constants as Arguments

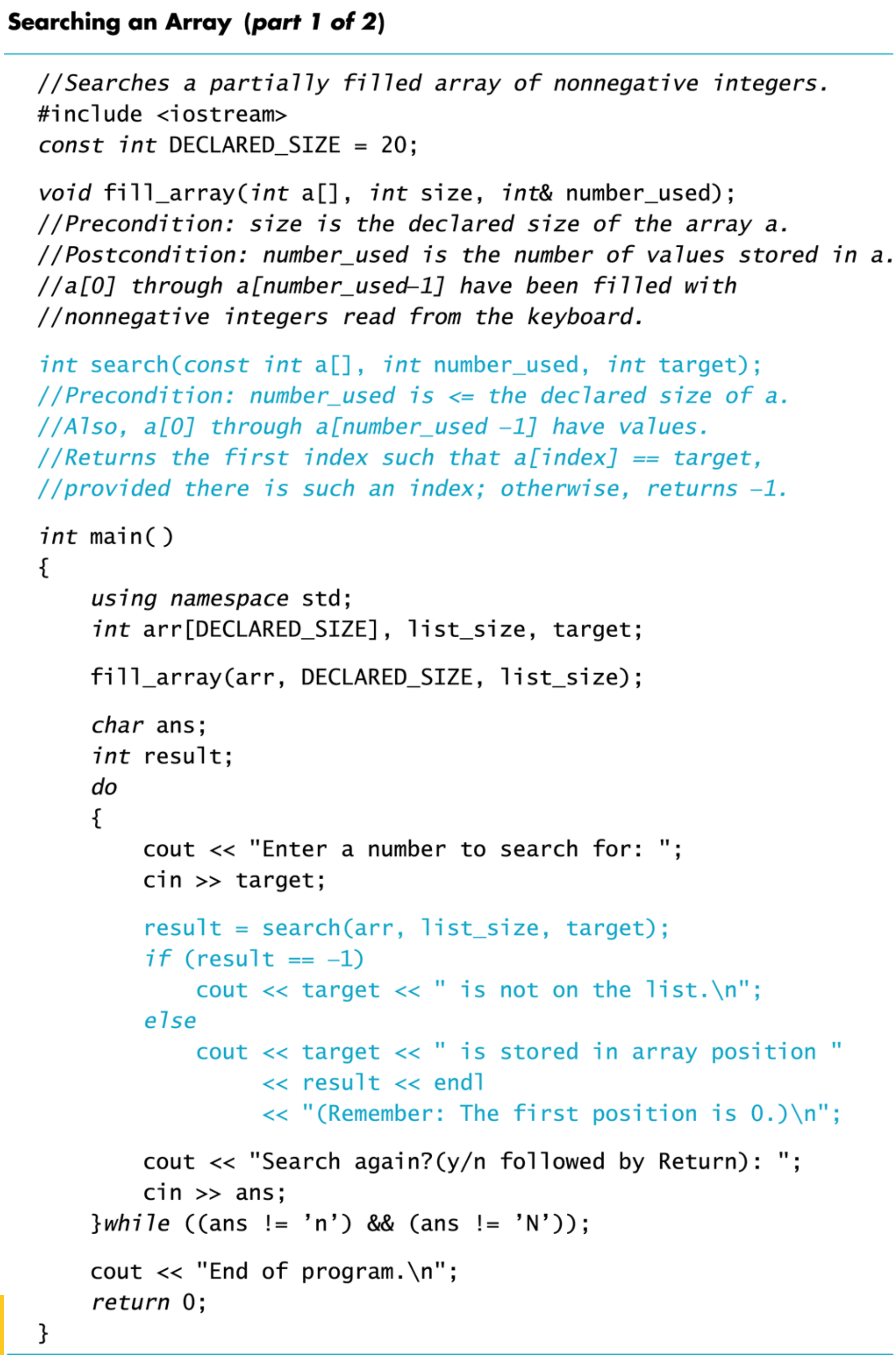
- When function fill_array (Display 10.9) is called MAX_NUMBER_SCORES is used as an argument
  - Can't MAX_NUMBER_SCORES be used directly without making it an argument?
    - Using MAX_NUMBER_SCORES as an argument makes it clear that fill_array requires the array's declared size
    - This makes fill_array easier to be used in other programs

Searching Arrays

- A **sequential search** is one way to search an array for a given value
  - Look at each element from first to last to see if the target value is equal to any of the array elements
  - The index of the target value can be returned to indicate where the value was found in the array
  - A value of -1 can be returned if the value was not found

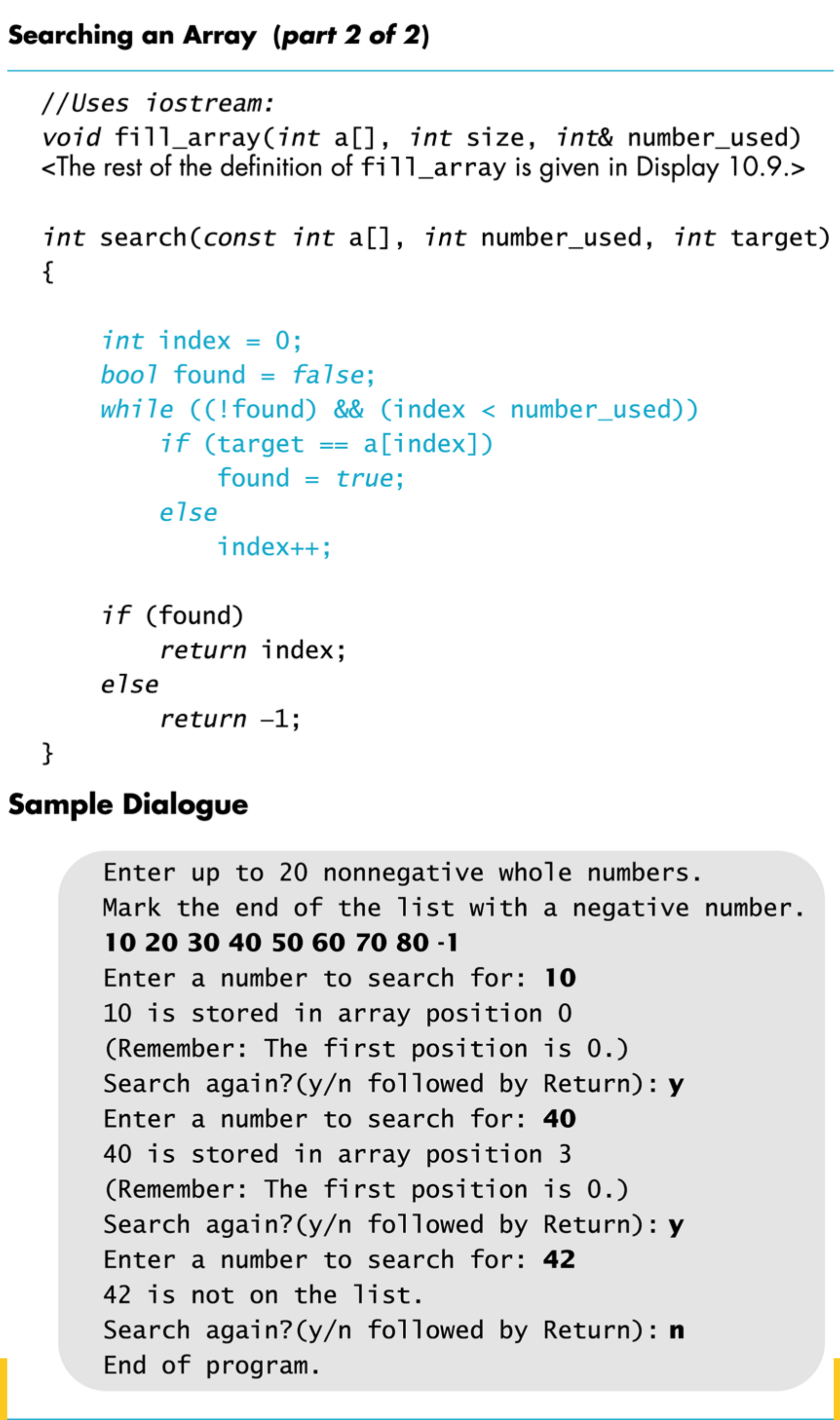
The search Function

- The search function of Display 10.10…
  - Uses a while loop to compare array elements to the target value
  - Sets a variable of type bool to true if the target value is found, ending the loop
  - Checks the boolean variable when the loop ends to see if the target value was found
  - Returns the index of the target value if found, otherwise returns -1



Searching an Array ***(part 1 of 2)***

```
//Searches a partially filled array of nonnegative integers.
#include <iostream>
const int DECLARED_SIZE = 20;

void fill_array(int a[], int size, int& number_used);
//Precondition: size is the declared size of the array a.
//Postcondition: number_used is the number of values stored in a.
//a[0] through a[number_used-1] have been filled with
//nonnegative integers read from the keyboard.

int search(const int a[], int number_used, int target);
//Precondition: number_used is <= the declared size of a.
//Also, a[0] through a[number_used -1] have values.
//Returns the first index such that a[index] == target,
//provided there is such an index; otherwise, returns -1.

int main( )
{
    using namespace std;
    int arr[DECLARED_SIZE], list_size, target;

    fill_array(arr, DECLARED_SIZE, list_size);

    char ans;
    int result;
    do
    {
        cout << "Enter a number to search for: ";
        cin >> target;

        result = search(arr, list_size, target);
        if (result == -1)
            cout << target << " is not on the list.\n";
        else
            cout << target << " is stored in array position "
                 << result << endl
                 << "(Remember: The first position is 0.)\n";

        cout << "Search again?(y/n followed by Return): ";
        cin >> ans;
    }while ((ans != 'n') && (ans != 'N'));

    cout << "End of program.\n";
    return 0;
}
```



Searching an Array (*part 2 of 2*)

```
//Uses iostream:
void fill_array(int a[], int size, int& number_used)
<The rest of the definition of fill_array is given in Display 10.9.>

int search(const int a[], int number_used, int target)
{

    int index = 0;
    bool found = false;
    while ((!found) && (index < number_used))
        if (target == a[index])
            found = true;
        else
            index++;

    if (found)
        return index;
    else
        return -1;
}
```

Sample Dialogue

```
Enter up to 20 nonnegative whole numbers.
Mark the end of the list with a negative number.
10 20 30 40 50 60 70 80 -1
Enter a number to search for: 10
10 is stored in array position 0
(Remember: The first position is 0.)
Search again?(y/n followed by Return): y
Enter a number to search for: 40
40 is stored in array position 3
(Remember: The first position is 0.)
Search again?(y/n followed by Return): y
Enter a number to search for: 42
42 is not on the list.
Search again?(y/n followed by Return): n
End of program.
```

Program E.g.: Sorting an Array

- ❑ Sorting a list of values is very common task
  - ❑ Create an alphabetical listing
  - ❑ Create a list of values in ascending order
  - ❑ Create a list of values in descending order
- ❑ Many sorting algorithms exist
  - ❑ Some are very efficient
  - ❑ Some are easier to understand

The Selection Sort Algorithm

- ❑ When the sort is complete, the elements of the array are ordered such that

  a[0] < a[1] < … < a [number_used -1]

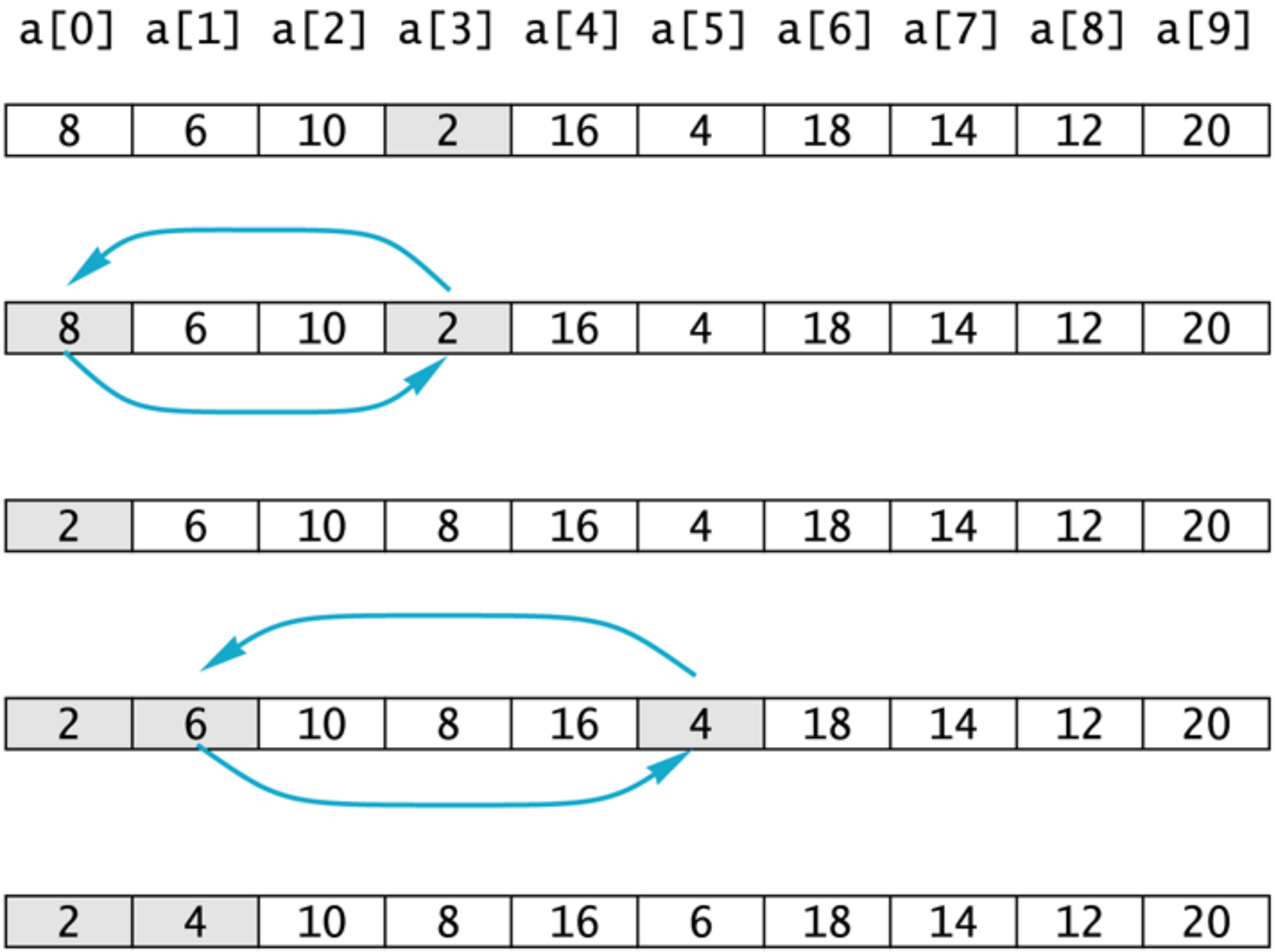
  - ❑ This leads to an outline of an algorithm:

```
for (int index = 0; index < number_used; index++)
    place the indexth smallest element in a[index]
```

Sort Algorithm Development

- One array is sufficient to do our sorting
  - Search for the smallest value in the array
  - Place this value in a[0], and place the value that was in a[0] in the location where the smallest was found
  - Starting at a[1], find the smallest remaining value swap it with the value currently in a[1]
  - Starting at a[2], continue the process until the array is sorted

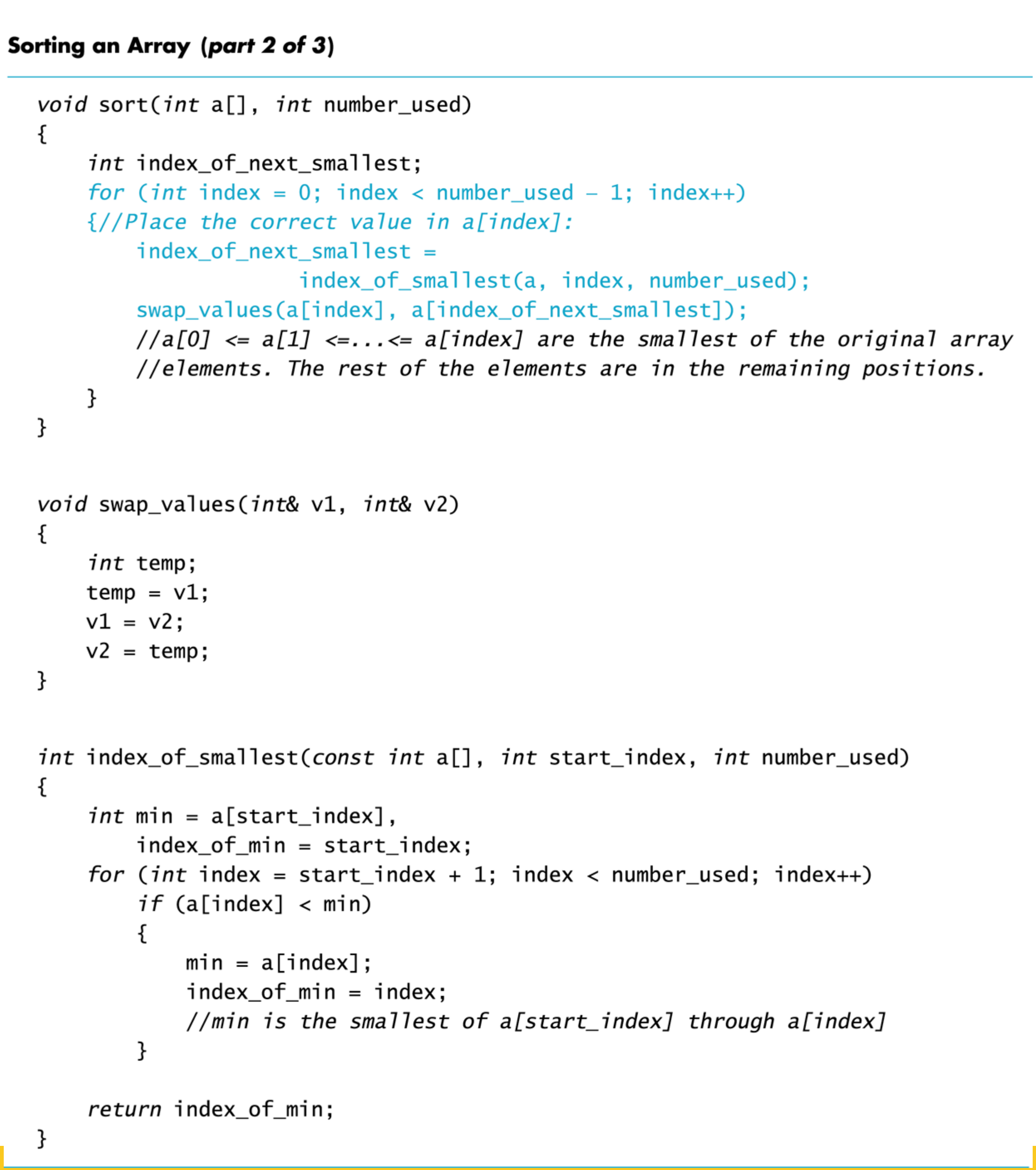
Selection Sort



a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
8	6	10	2	16	4	18	14	12	20
8	6	10	2	16	4	18	14	12	20
2	6	10	8	16	4	18	14	12	20
2	6	10	8	16	4	18	14	12	20
2	4	10	8	16	6	18	14	12	20

Sorting an Array (*part 1 of 3*)

```
//Tests the procedure sort.
#include <iostream>

void fill_array(int a[], int size, int& number_used);
//Precondition: size is the declared size of the array a.
//Postcondition: number_used is the number of values stored in a.
//a[0] through a[number_used – 1] have been filled with
//nonnegative integers read from the keyboard.

void sort(int a[], int number_used);
//Precondition: number_used <= declared size of the array a.
//The array elements a[0] through a[number_used – 1] have values.
//Postcondition: The values of a[0] through a[number_used – 1] have
//been rearranged so that a[0] <= a[1] <= ... <= a[number_used – 1].

void swap_values(int& v1, int& v2);
//Interchanges the values of v1 and v2.

int index_of_smallest(const int a[], int start_index, int number_used);
//Precondition: 0 <= start_index < number_used. Referenced array elements have
//values.
//Returns the index i such that a[i] is the smallest of the values
//a[start_index], a[start_index + 1], ..., a[number_used – 1].

int main( )
{
    using namespace std;
    cout << "This program sorts numbers from lowest to highest.\n";

    int sample_array[10], number_used;
    fill_array(sample_array, 10, number_used);
    sort(sample_array, number_used);

    cout << "In sorted order the numbers are:\n";
    for (int index = 0; index < number_used; index++)
        cout << sample_array[index] << " ";
    cout << endl;

    return 0;
}

//Uses iostream:
void fill_array(int a[], int size, int& number_used)
 <The rest of the definition of fill_array is given in Display 10.9.>
```



Sorting an Array (*part 2 of 3*)

```
void sort(int a[], int number_used)
{
    int index_of_next_smallest;
    for (int index = 0; index < number_used - 1; index++)
    {//Place the correct value in a[index]:
        index_of_next_smallest =
                     index_of_smallest(a, index, number_used);
        swap_values(a[index], a[index_of_next_smallest]);
        //a[0] <= a[1] <=...<= a[index] are the smallest of the original array
        //elements. The rest of the elements are in the remaining positions.
    }
}


void swap_values(int& v1, int& v2)
{
    int temp;
    temp = v1;
    v1 = v2;
    v2 = temp;
}


int index_of_smallest(const int a[], int start_index, int number_used)
{
    int min = a[start_index],
        index_of_min = start_index;
    for (int index = start_index + 1; index < number_used; index++)
        if (a[index] < min)
        {
            min = a[index];
            index_of_min = index;
            //min is the smallest of a[start_index] through a[index]
        }

    return index_of_min;
}
```

Sorting an Array (*part 3 of 3*)

Sample Dialogue

```
This program sorts numbers from lowest to highest.
Enter up to 10 nonnegative whole numbers.
Mark the end of the list with a negative number.
80 30 50 70 60 90 20 30 40 -1
In sorted order the numbers are:
20 30 30 40 50 60 70 80 90
```

Class Work

- ❑ Can you

  - ❑ Write a program that will read up to 10 letters into an array and write the letters back to the screen in the reverse order?

    abcd should be output as dcba

    Use a period as a sentinel value to mark the end of input

Arrays and Classes

Arrays and Classes

- ❑ Arrays can use structures or classes as their base types
  - ❑ Example:

```
struct WindInfo
     {
     double velocity;
     char direction;
     }

     WindInfo   data_point[10];
```

Accessing Members

- ❑ When an array's base type is a structure or a class…
  - ❑ Use the dot operator to access the members of an indexed variable
  - ❑ Example:

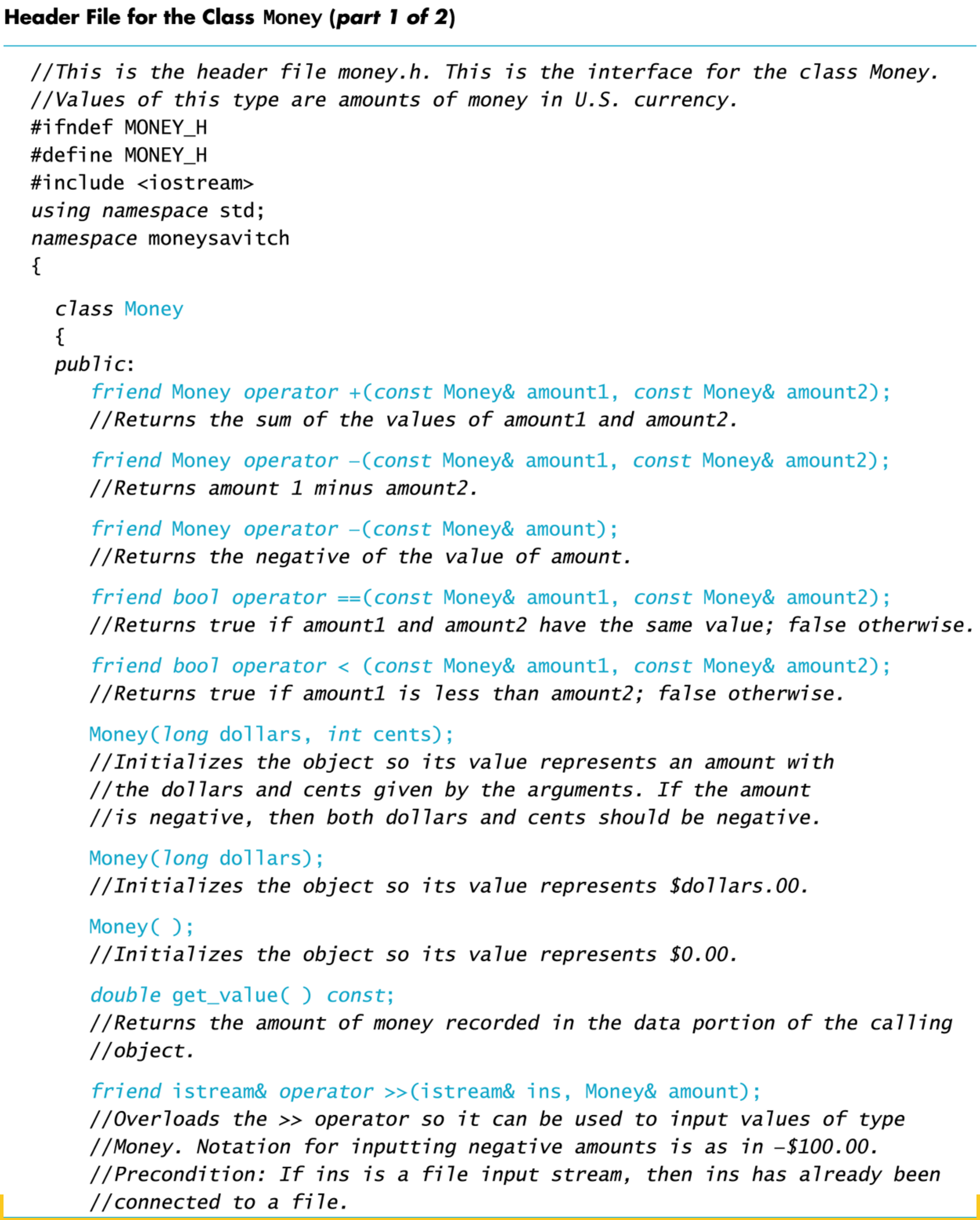
```
for (i = 0; i < 10; i++)
      {
       cout << "Enter velocity: ";
       cin >> data_point[i].velocity;
       …
      }
```

An Array of Money

- The Money class of Chapter 8 can be the base type for an array
- When an array of classes is declared
  - The default constructor is called to initialize the indexed variables
- An array of class Money is demonstrated in

Display 10.14 (1)

Display 10.14 (2)



Header File for the Class Money (*part 1 of 2*)

```
//This is the header file money.h. This is the interface for the class Money.
//Values of this type are amounts of money in U.S. currency.
#ifndef MONEY_H
#define MONEY_H
#include <iostream>
using namespace std;
namespace moneysavitch
{

  class Money
  {
  public:
    friend Money operator +(const Money& amount1, const Money& amount2);
    //Returns the sum of the values of amount1 and amount2.

    friend Money operator –(const Money& amount1, const Money& amount2);
    //Returns amount 1 minus amount2.

    friend Money operator –(const Money& amount);
    //Returns the negative of the value of amount.

    friend bool operator ==(const Money& amount1, const Money& amount2);
    //Returns true if amount1 and amount2 have the same value; false otherwise.

    friend bool operator < (const Money& amount1, const Money& amount2);
    //Returns true if amount1 is less than amount2; false otherwise.

    Money(long dollars, int cents);
    //Initializes the object so its value represents an amount with
    //the dollars and cents given by the arguments. If the amount
    //is negative, then both dollars and cents should be negative.

    Money(long dollars);
    //Initializes the object so its value represents $dollars.00.

    Money( );
    //Initializes the object so its value represents $0.00.

    double get_value( ) const;
    //Returns the amount of money recorded in the data portion of the calling
    //object.

    friend istream& operator >>(istream& ins, Money& amount);
    //Overloads the >> operator so it can be used to input values of type
    //Money. Notation for inputting negative amounts is as in –$100.00.
    //Precondition: If ins is a file input stream, then ins has already been
    //connected to a file.
```

Header File for the Class Money (*part 2 of 2*)

```
     friend ostream& operator <<(ostream& outs, const Money& amount);
     //Overloads the << operator so it can be used to output values of type
     //Money. Precedes each output value of type Money with a dollar sign.
     //Precondition: If outs is a file output stream, then outs has already been
     //connected to a file.
  private:
     long all_cents;
  };
}//namespace moneysavitch
#endif //MONEY_H
```

Program Using an Array of Objects (*part 1 of 2*)

```
//Reads in 5 amounts of money and shows how much each
//amount differs from the largest amount.
#include <iostream>
#include "money.h"

int main( )
{
    using namespace std;
    using namespace moneysavitch;
    Money amount[5], max;
    int i;

    cout << "Enter 5 amounts of money:\n";
    cin >> amount[0];
    max = amount[0];
    for (i = 1; i < 5; i++)
    {
        cin >> amount[i];
        if (max < amount[i])
            max = amount[i];
        //max is the largest of amount[0],..., amount[i].
    }

    Money difference[5];
    for (i = 0; i < 5; i++)
        difference[i] = max - amount[i];

    cout << "The highest amount is " << max << endl;
    cout << "The amounts and their\n"
         << "differences from the largest are:\n";
    for (i = 0; i < 5; i++)
    {
        cout << amount[i] << " off by "
             << difference[i] << endl;
    }

    return 0;
}
```

Program Using an Array of Objects (*part 2 of 2*)

Sample Dialogue

```
Enter 5 amounts of money:
$5.00 $10.00 $19.99 $20.00 $12.79
The highest amount is $20.00
The amounts and their
differences from the largest are:
$5.00 off by $15.00
$10.00 off by $10.00
$19.99 off by $0.01
$20.00 off by $0.00
$12.79 off by $7.21
```

Arrays as Structure Members

- A structure can contain an array as a member
  - Example:

```
struct Data
    {
    double time[10];
    int distance;
    }

    Data my_best;
```

  - my_best contains an array of type double

Accessing Array Elements

❑ To access the array elements within a structure

  ❑ Use the dot operator to identify the array within the structure

  ❑ Use the [ ]'s to identify the indexed variable desired

Example: **my_best.time[i]**
references the ith indexed variable of the variable time in the structure my_best

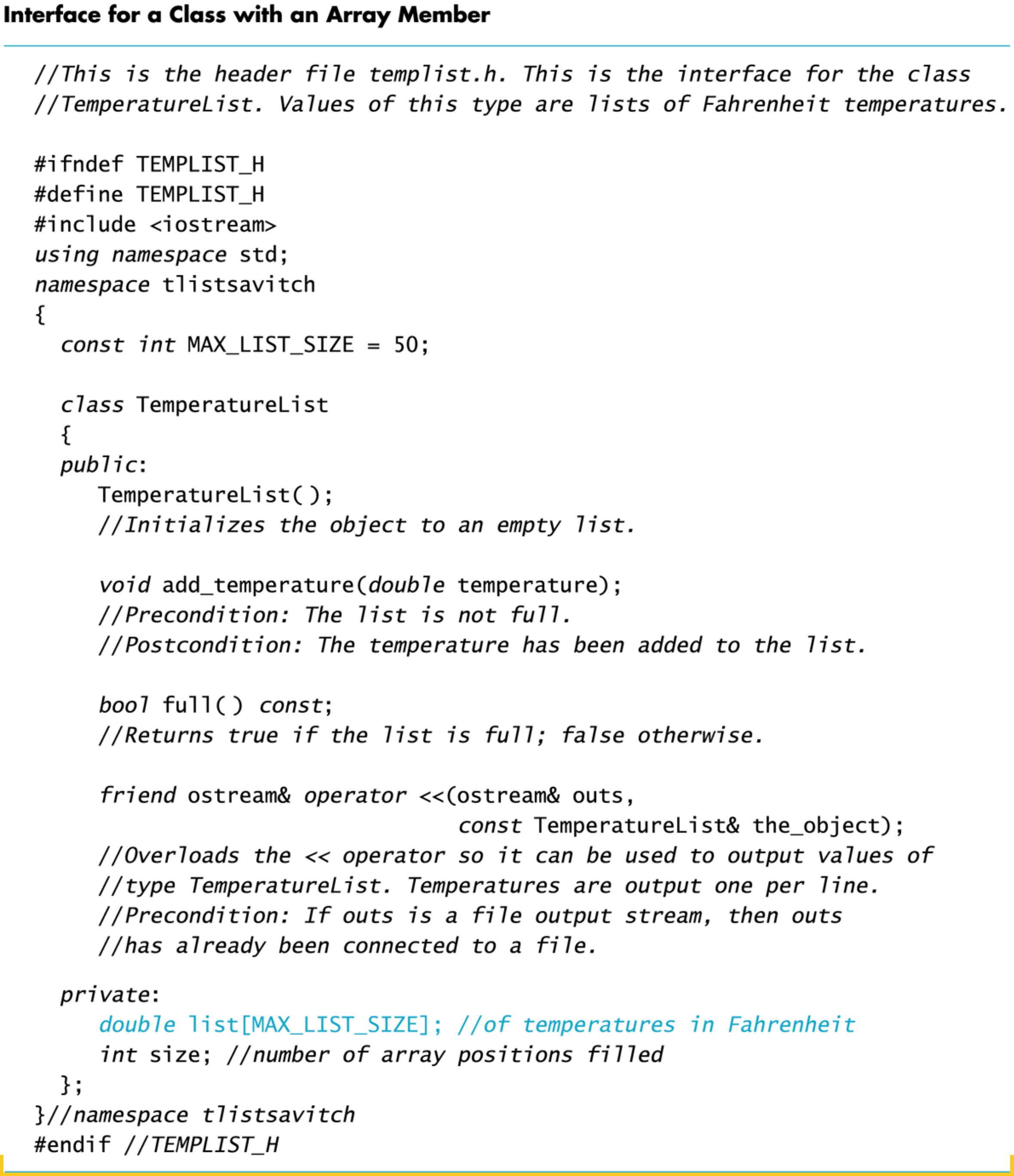
Arrays as Class Members

- Class TemperatureList includes an array
  - The array, named list, contains temperatures
  - Member variable size is the number of items stored

```
class TemperatureList
 {
   public:
         TemperatureList( );
         //Member functions
     private:
         double  list [MAX_LIST_SIZE];
         int size;
  }
```

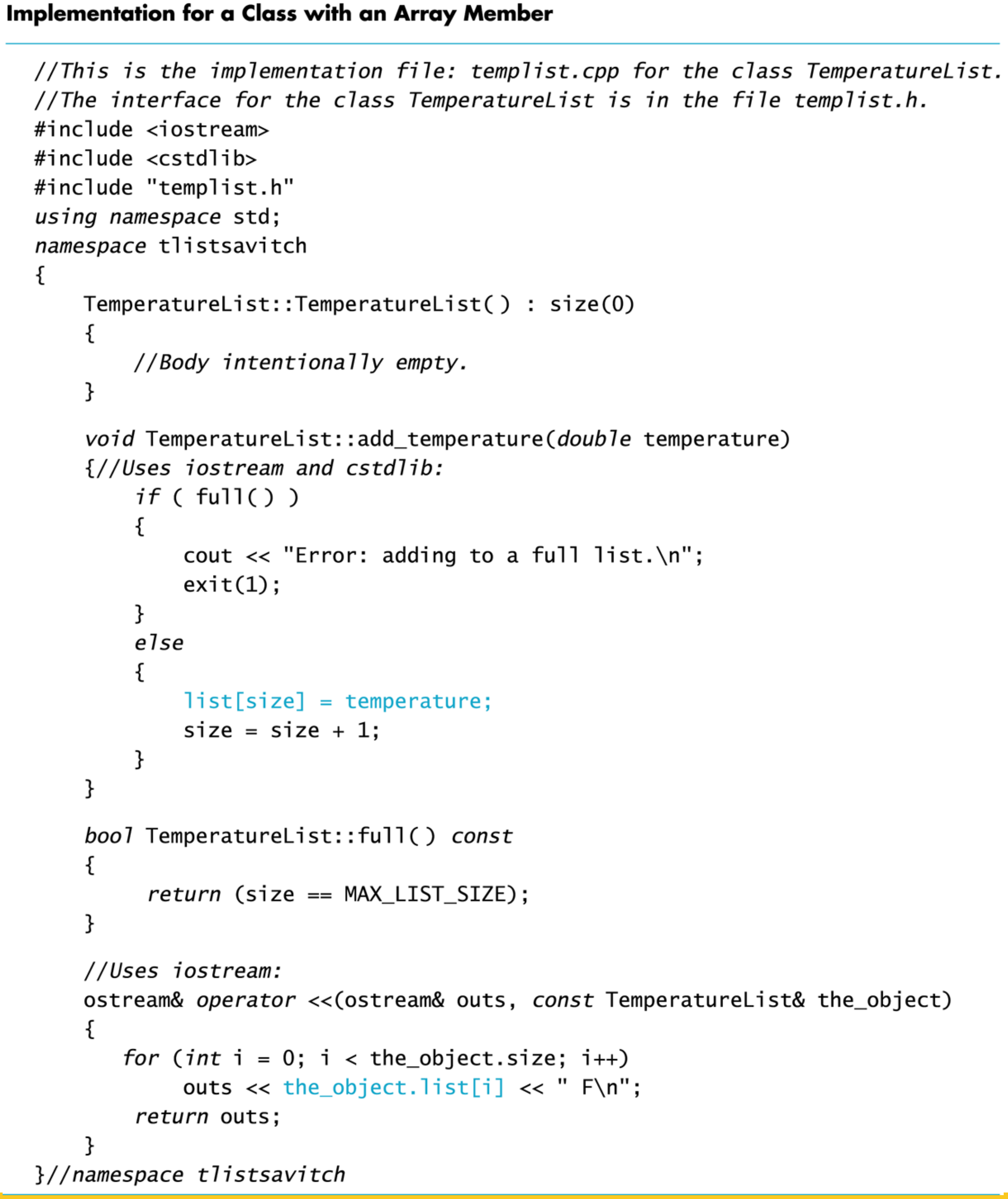
Overview of TemperatureList

- To create an object of type TemperatureList:
  - TemperatureList my_data;
- To add a temperature to the list:
  - My_data.add_temperature(77);
    - A check is made to see if the array is full
- << is overloaded so output of the list is
  - cout << my_data;



Interface for a Class with an Array Member

```
//This is the header file templist.h. This is the interface for the class
//TemperatureList. Values of this type are lists of Fahrenheit temperatures.

#ifndef TEMPLIST_H
#define TEMPLIST_H
#include <iostream>
using namespace std;
namespace tlistsavitch
{
  const int MAX_LIST_SIZE = 50;

  class TemperatureList
  {
  public:
     TemperatureList( );
     //Initializes the object to an empty list.

     void add_temperature(double temperature);
     //Precondition: The list is not full.
     //Postcondition: The temperature has been added to the list.

     bool full( ) const;
     //Returns true if the list is full; false otherwise.

     friend ostream& operator <<(ostream& outs,
                                 const TemperatureList& the_object);
     //Overloads the << operator so it can be used to output values of
     //type TemperatureList. Temperatures are output one per line.
     //Precondition: If outs is a file output stream, then outs
     //has already been connected to a file.

  private:
     double list[MAX_LIST_SIZE]; //of temperatures in Fahrenheit
     int size; //number of array positions filled
  };
}//namespace tlistsavitch
#endif //TEMPLIST_H
```



Implementation for a Class with an Array Member

```
//This is the implementation file: templist.cpp for the class TemperatureList.
//The interface for the class TemperatureList is in the file templist.h.
#include <iostream>
#include <cstdlib>
#include "templist.h"
using namespace std;
namespace tlistsavitch
{
    TemperatureList::TemperatureList( ) : size(0)
    {
        //Body intentionally empty.
    }

    void TemperatureList::add_temperature(double temperature)
    {//Uses iostream and cstdlib:
        if ( full( ) )
        {
            cout << "Error: adding to a full list.\n";
            exit(1);
        }
        else
        {
            list[size] = temperature;
            size = size + 1;
        }
    }

    bool TemperatureList::full( ) const
    {
         return (size == MAX_LIST_SIZE);
    }

    //Uses iostream:
    ostream& operator <<(ostream& outs, const TemperatureList& the_object)
    {
       for (int i = 0; i < the_object.size; i++)
            outs << the_object.list[i] << " F\n";
        return outs;
    }
}//namespace tlistsavitch
```

Class Work

- ❑ Can you
  - ❑ Declare an array as a member of a class?
  - ❑ Declare an array of objects of a class?
  - ❑ Write code to call a member function of an element
    in an array of objects of a class?
  - ❑ Write code to access an element of an array of integers that is a member of a class?

Multi-Dimensional Arrays

Multi-Dimensional Arrays

- ❑ C++ allows arrays with multiple index values
  - ❑ **char page [30] [100];**
    declares an array of characters named page
    - ❑ page has two index values:
      The first ranges from 0 to 29
      The second ranges from 0 to 99
  - ❑ Each index in enclosed in its own brackets
  - ❑ Page can be visualized as an array of
    30 rows and 100 columns

Index Values of page

- ❑ The indexed variables for array page are
  page[0][0], page[0][1], ..., page[0][99]
  page[1][0], page[1][1], ..., page[1][99]

  ...

  page[29][0], page[29][1], ... , page[29][99]

- ❑ page is actually an array of size 30
  - ❑ page's base type is an **array of 100 characters**

Multi-D Array Parameters

- Recall that the size of an array is not needed when declaring a formal parameter:

```
  void display_line(const char a[ ], int size);
```

- The base type of a multi-dimensional array must be completely specified in the parameter declaration

```
void display_page(const char page[ ] [100],
int size_dimension_1);
```

Program E.g.: Grading Program

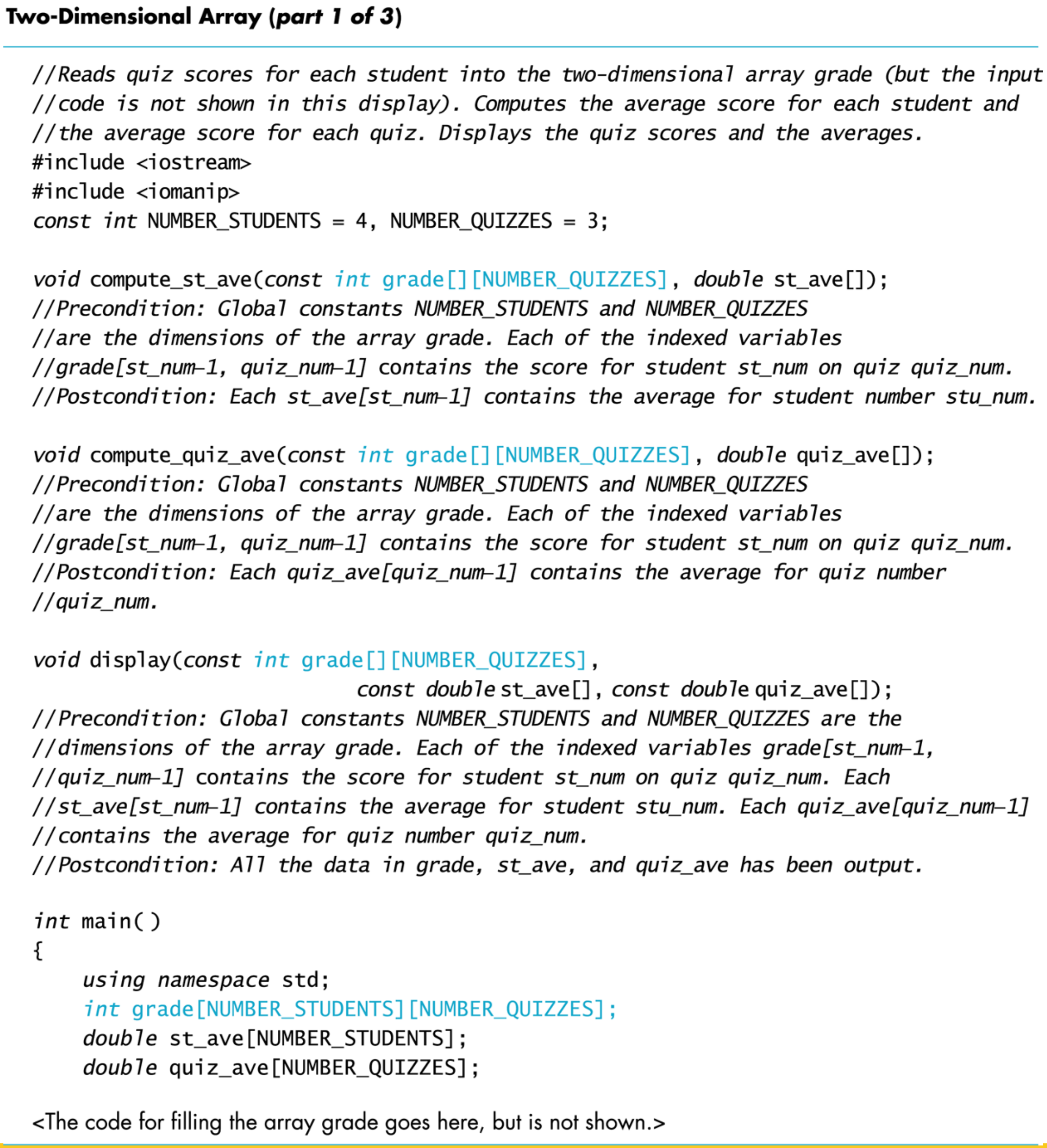
- Grade records for a class can be stored in a two-dimensional array
  - For a class with 4 students and 3 quizzes the array could be declared as

```
int grade[4][3];
```

    - The first array index refers to the number of a student
    - The second array index refers to a quiz number
- Since student and quiz numbers start with one, we subtract one to obtain the correct index

Grading Program: average scores

- The grading program uses one-dimensional arrays to store…
  - Each student's average score
  - Each quiz's average score
- The functions that calculate these averages use global constants for the size of the arrays
  - This was done because the functions seem to be particular to this program



Two-Dimensional Array (*part 1 of 3*)

```
//Reads quiz scores for each student into the two-dimensional array grade (but the input
//code is not shown in this display). Computes the average score for each student and
//the average score for each quiz. Displays the quiz scores and the averages.
#include <iostream>
#include <iomanip>
const int NUMBER_STUDENTS = 4, NUMBER_QUIZZES = 3;

void compute_st_ave(const int grade[][NUMBER_QUIZZES], double st_ave[]);
//Precondition: Global constants NUMBER_STUDENTS and NUMBER_QUIZZES
//are the dimensions of the array grade. Each of the indexed variables
//grade[st_num-1, quiz_num-1] contains the score for student st_num on quiz quiz_num.
//Postcondition: Each st_ave[st_num-1] contains the average for student number stu_num.

void compute_quiz_ave(const int grade[][NUMBER_QUIZZES], double quiz_ave[]);
//Precondition: Global constants NUMBER_STUDENTS and NUMBER_QUIZZES
//are the dimensions of the array grade. Each of the indexed variables
//grade[st_num-1, quiz_num-1] contains the score for student st_num on quiz quiz_num.
//Postcondition: Each quiz_ave[quiz_num-1] contains the average for quiz number
//quiz_num.

void display(const int grade[][NUMBER_QUIZZES],
                             const double st_ave[], const double quiz_ave[]);
//Precondition: Global constants NUMBER_STUDENTS and NUMBER_QUIZZES are the
//dimensions of the array grade. Each of the indexed variables grade[st_num-1,
//quiz_num-1] contains the score for student st_num on quiz quiz_num. Each
//st_ave[st_num-1] contains the average for student stu_num. Each quiz_ave[quiz_num-1]
//contains the average for quiz number quiz_num.
//Postcondition: All the data in grade, st_ave, and quiz_ave has been output.

int main( )
{
    using namespace std;
    int grade[NUMBER_STUDENTS][NUMBER_QUIZZES];
    double st_ave[NUMBER_STUDENTS];
    double quiz_ave[NUMBER_QUIZZES];

<The code for filling the array grade goes here, but is not shown.>
```

Two-Dimensional Array (*part 2 of 3*)

```
    compute_st_ave(grade, st_ave);
    compute_quiz_ave(grade, quiz_ave);
    display(grade, st_ave, quiz_ave);
    return 0;
}


void compute_st_ave(const int grade[][NUMBER_QUIZZES], double st_ave[])
{
    for (int st_num = 1; st_num <= NUMBER_STUDENTS; st_num++)
    {//Process one st_num:
        double sum = 0;
        for (int quiz_num = 1; quiz_num <= NUMBER_QUIZZES; quiz_num++)
            sum = sum + grade[st_num-1][quiz_num-1];
        //sum contains the sum of the quiz scores for student number st_num.
        st_ave[st_num-1] = sum/NUMBER_QUIZZES;
        //Average for student st_num is the value of st_ave[st_num-1]
    }
}

void compute_quiz_ave(const int grade[][NUMBER_QUIZZES], double quiz_ave[])
{
    for (int quiz_num = 1; quiz_num <= NUMBER_QUIZZES; quiz_num++)
    {//Process one quiz (for all students):
        double sum = 0;
        for (int st_num = 1; st_num <= NUMBER_STUDENTS; st_num++)
            sum = sum + grade[st_num-1][quiz_num-1];
        //sum contains the sum of all student scores on quiz number quiz_num.
        quiz_ave[quiz_num-1] = sum/NUMBER_STUDENTS;
        //Average for quiz quiz_num is the value of quiz_ave[quiz_num-1]
    }
}
```

Two-Dimensional Array (*part 3 of 3*)

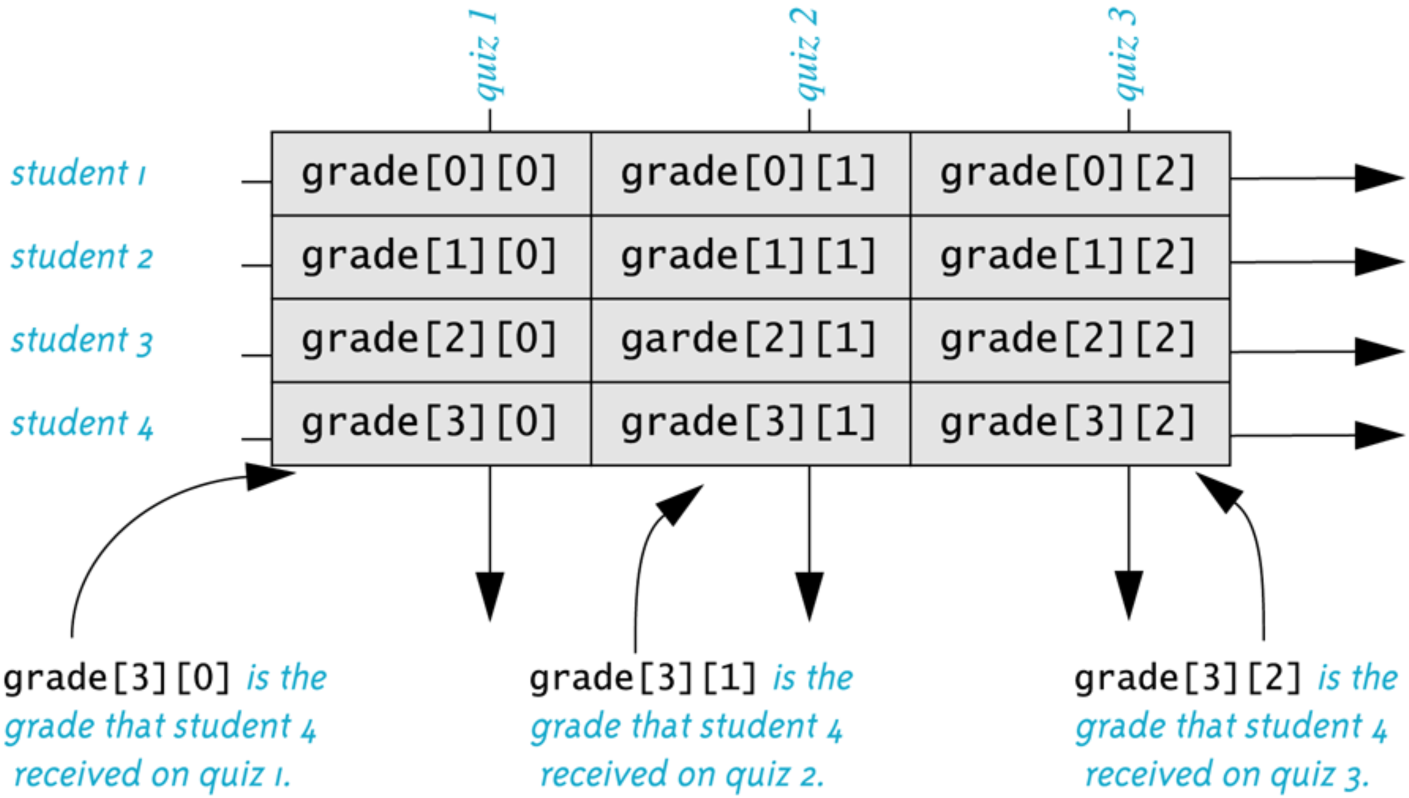
```
//Uses iostream and iomanip:
void display(const int grade[][NUMBER_QUIZZES],
                         const double st_ave[], const double quiz_ave[])
{
    using namespace std;
    cout.setf(ios::fixed);
    cout.setf(ios::showpoint);
    cout.precision(1);

    cout << setw(10) << "Student"
         << setw(5) << "Ave"
         << setw(15) << "Quizzes\n";
    for (int st_num = 1; st_num <= NUMBER_STUDENTS; st_num++)
    {//Display for one st_num:
        cout << setw(10) << st_num
             << setw(5) << st_ave[st_num-1] << " ";
        for (int quiz_num = 1; quiz_num <= NUMBER_QUIZZES; quiz_num++)
            cout << setw(5) << grade[st_num-1][quiz_num-1];
        cout << endl;
    }

    cout << "Quiz averages = ";
    for (int quiz_num = 1; quiz_num <= NUMBER_QUIZZES; quiz_num++)
        cout << setw(5) << quiz_ave[quiz_num-1];
    cout << endl;
}
```

Sample Dialogue

<The dialogue for filling the array grade is not shown.>

```
Student    Ave          Quizzes
      1   10.0          10   10   10
      2    1.0           2    0    1
      3    7.7           8    6    9
      4    7.3           8    4   10
 Quiz averages =       7.0  5.0  7.5
```

The Two-Dimensional Array grade



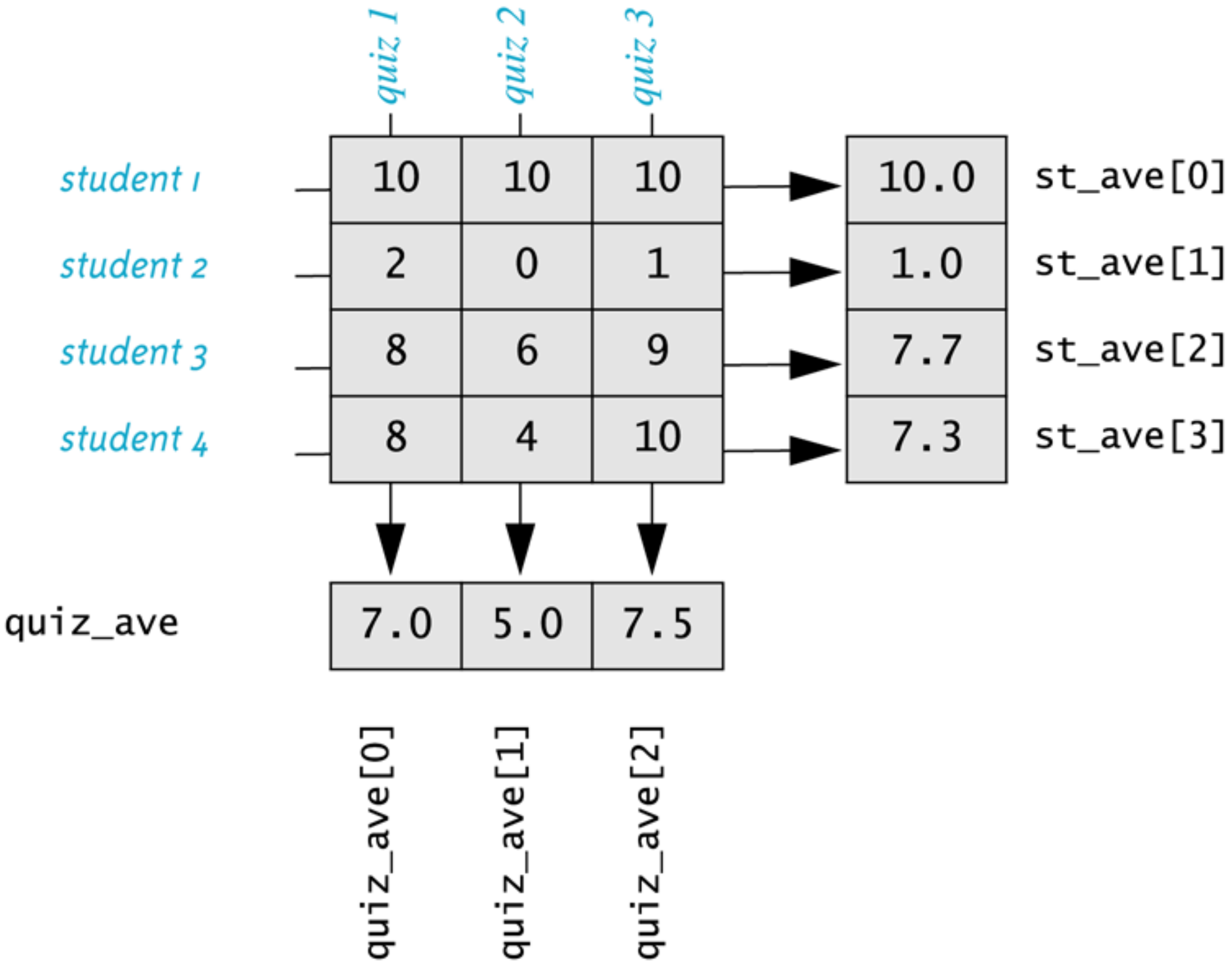
	quiz 1	*quiz 2*	*quiz 3*
student 1	`grade[0][0]`	`grade[0][1]`	`grade[0][2]`
student 2	`grade[1][0]`	`grade[1][1]`	`grade[1][2]`
student 3	`grade[2][0]`	`garde[2][1]`	`grade[2][2]`
student 4	`grade[3][0]`	`grade[3][1]`	`grade[3][2]`

`grade[3][0]` *is the grade that student 4 received on quiz 1.*

`grade[3][1]` *is the grade that student 4 received on quiz 2.*

`grade[3][2]` *is the grade that student 4 received on quiz 3.*

The Two-Dimensional Array grade (Another View)



	quiz 1	quiz 2	quiz 3		
student 1	10	10	10	10.0	st_ave[0]
student 2	2	0	1	1.0	st_ave[1]
student 3	8	6	9	7.7	st_ave[2]
student 4	8	4	10	7.3	st_ave[3]
quiz_ave	7.0	5.0	7.5		
	quiz_ave[0]	quiz_ave[1]	quiz_ave[2]		

Section 10.5 Conclusion

- Can you

  - Write code that will fill the array a(declared below) with numbers typed at the keyboard? The numbers will be input fiver per line, on four lines.

```
int a[4][5];
```

Home Work

- ❑ Write a function to get input into a 2-D array.