APS105: Lecture 17

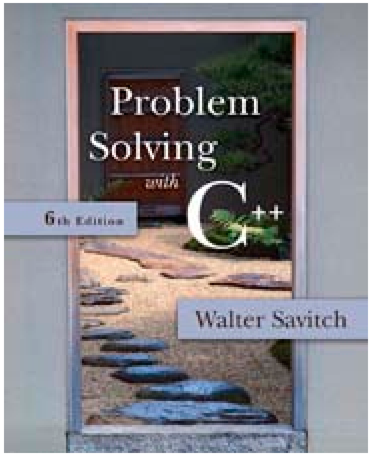
Wael Aboelsaadat

wael@cs.toronto.edu

http://ccnet3.utoronto.ca/20079/aps105h1f/

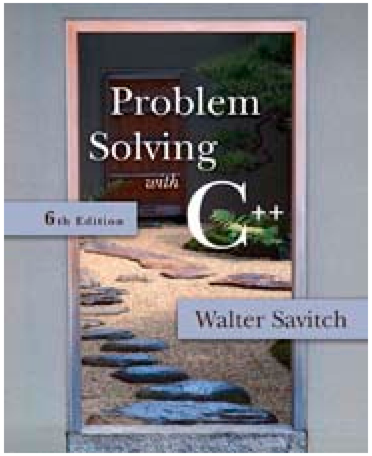
Acknowledgement: These slides are a modified version of the text book slides as supplied by Addison Wesley

Download the code shown in lecture from course website:
Handouts ➔ Lectures Source Code - Wael



7.2

Arrays in Functions



Problem Solving with C++

6th Edition

Walter Savitch

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 ]  );
```

Display 7.3

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);
  ```

Display 7.4

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

How does the function know how to access the array elements?

- To access element i, the function uses the formula
    - address in memory of element i =

      start address of array + i * element size
    - Start address of array = address of first element in array
    - E.g.

      score[2] is an indexed variable to the location identified by the above formula

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 from Display 7.4 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

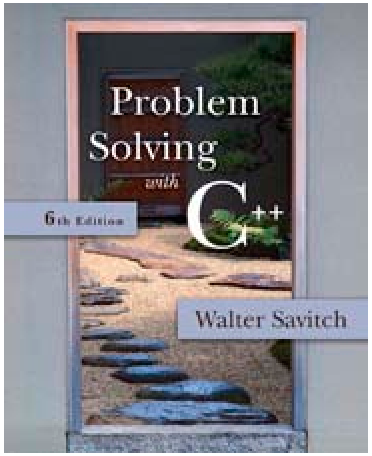
-
  ```
  double compute_average(int a[ ], int size);

  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

7.3

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 7.9 needs to know the declared size of the array

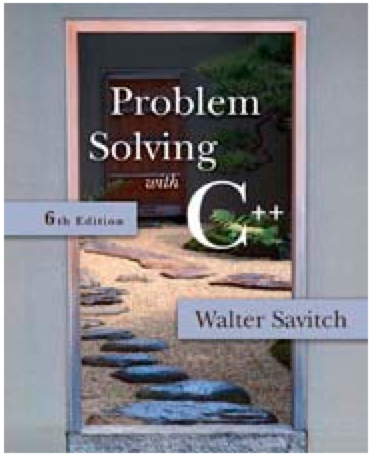
Display 7.9 (1) Display 7.9 (2) Display 7.9 (3)

Constants as Arguments

- When function fill_array (Display 7.9) 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

7.4

Multidimensional 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

Multidimensional 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);
  ```

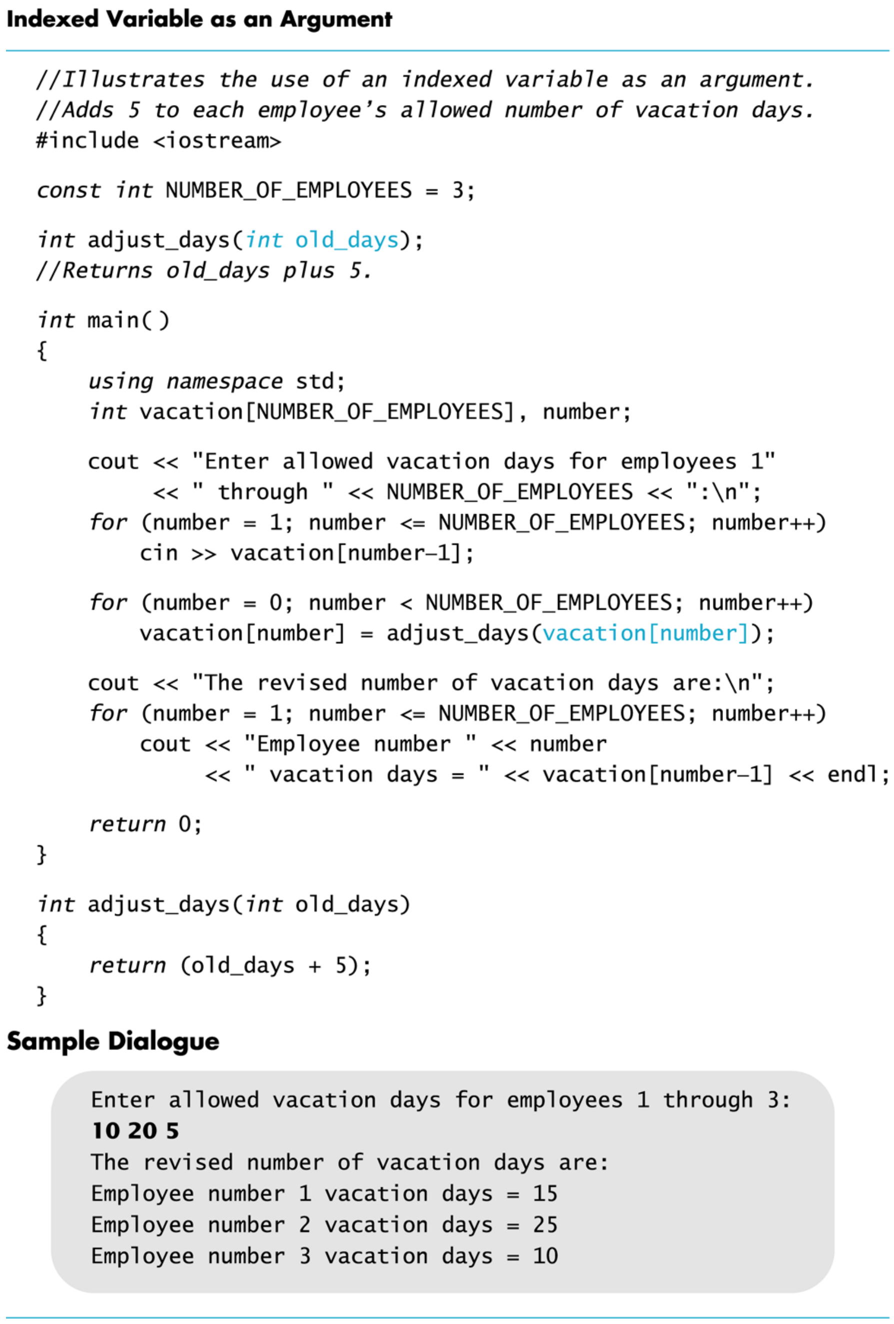
- The base type of a multi-dimensional array must be completely specified in the parameter declaration

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

Display 7.3

Back

Next



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);
}
```

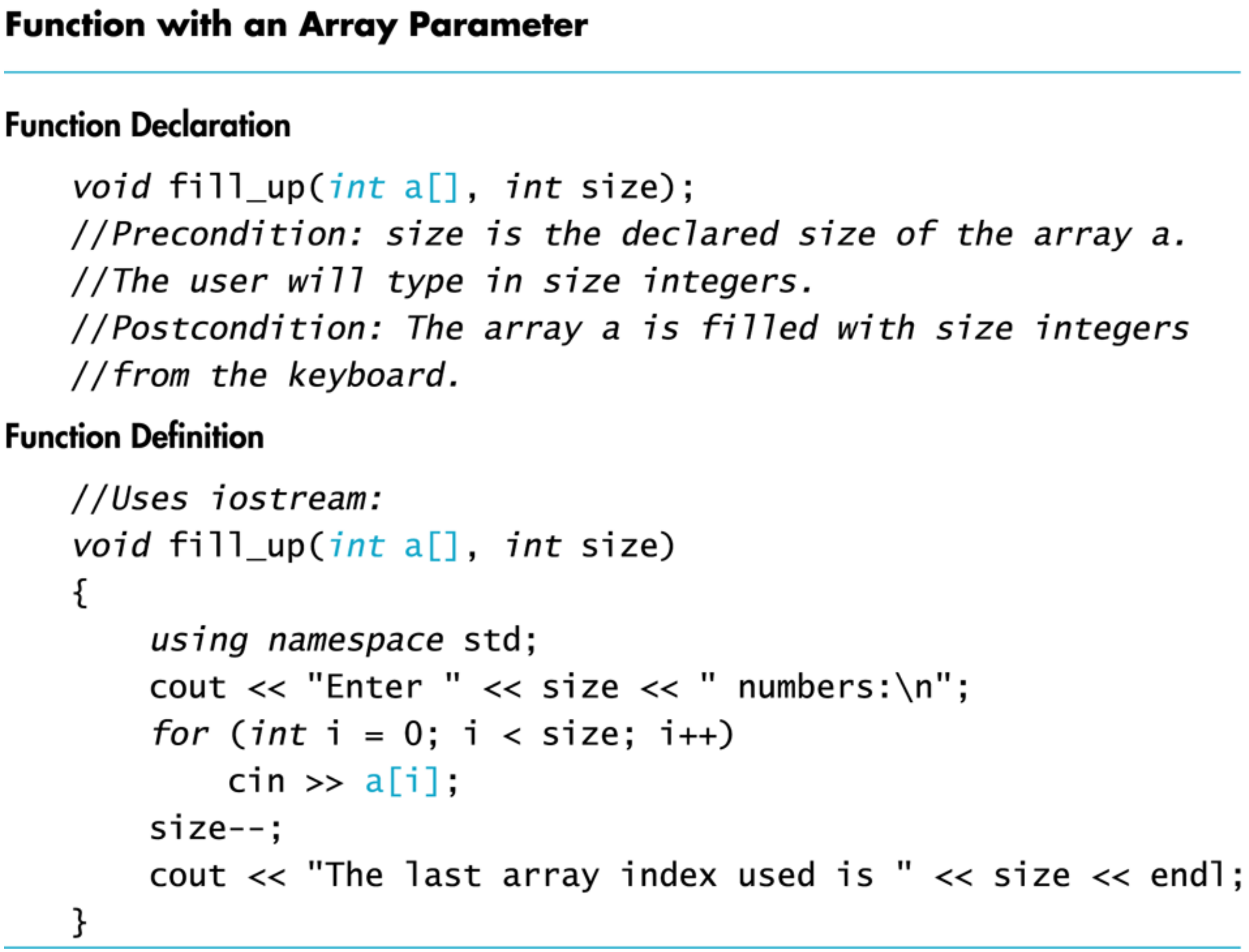
Sample Dialogue

```
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
```

Display 7.4

Back

Next



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.
```

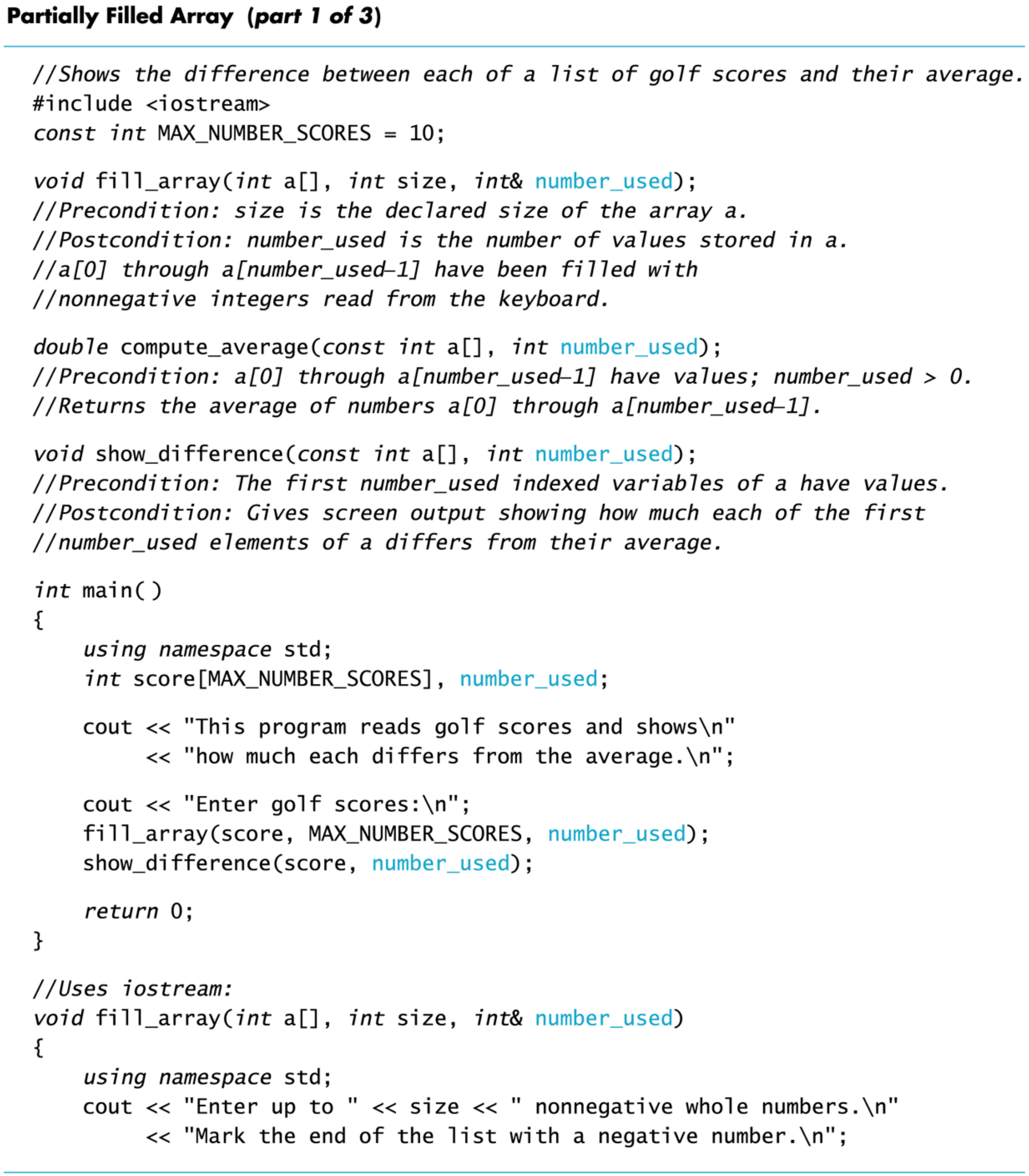
Function Definition

```
//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;
}
```

Display 7.9 (1/3)

Back

Next



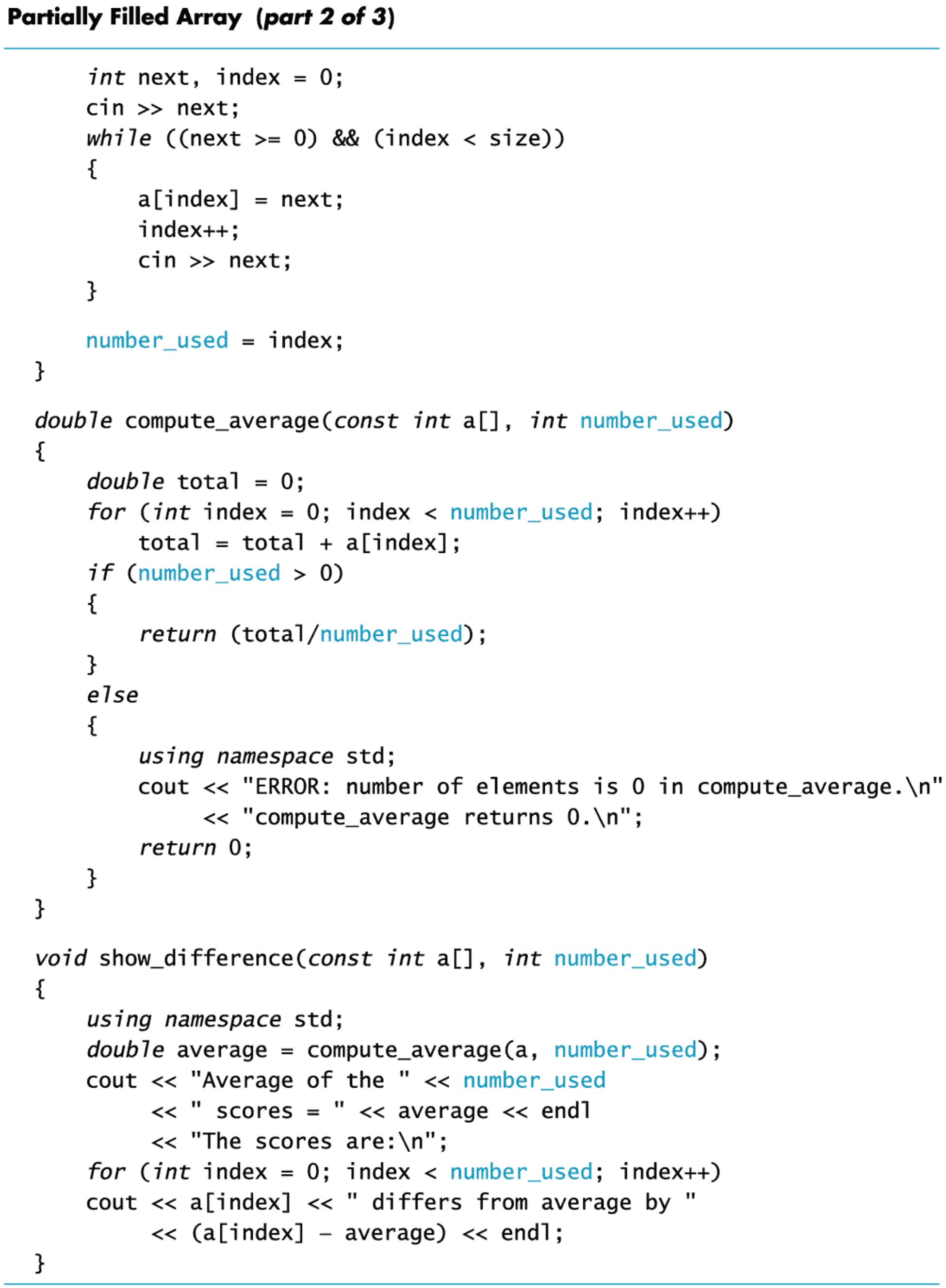
Partially Filled Array ***(part 1 of 3)***

```
//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";
```

Display 7.9 (2/3)

Back

Next



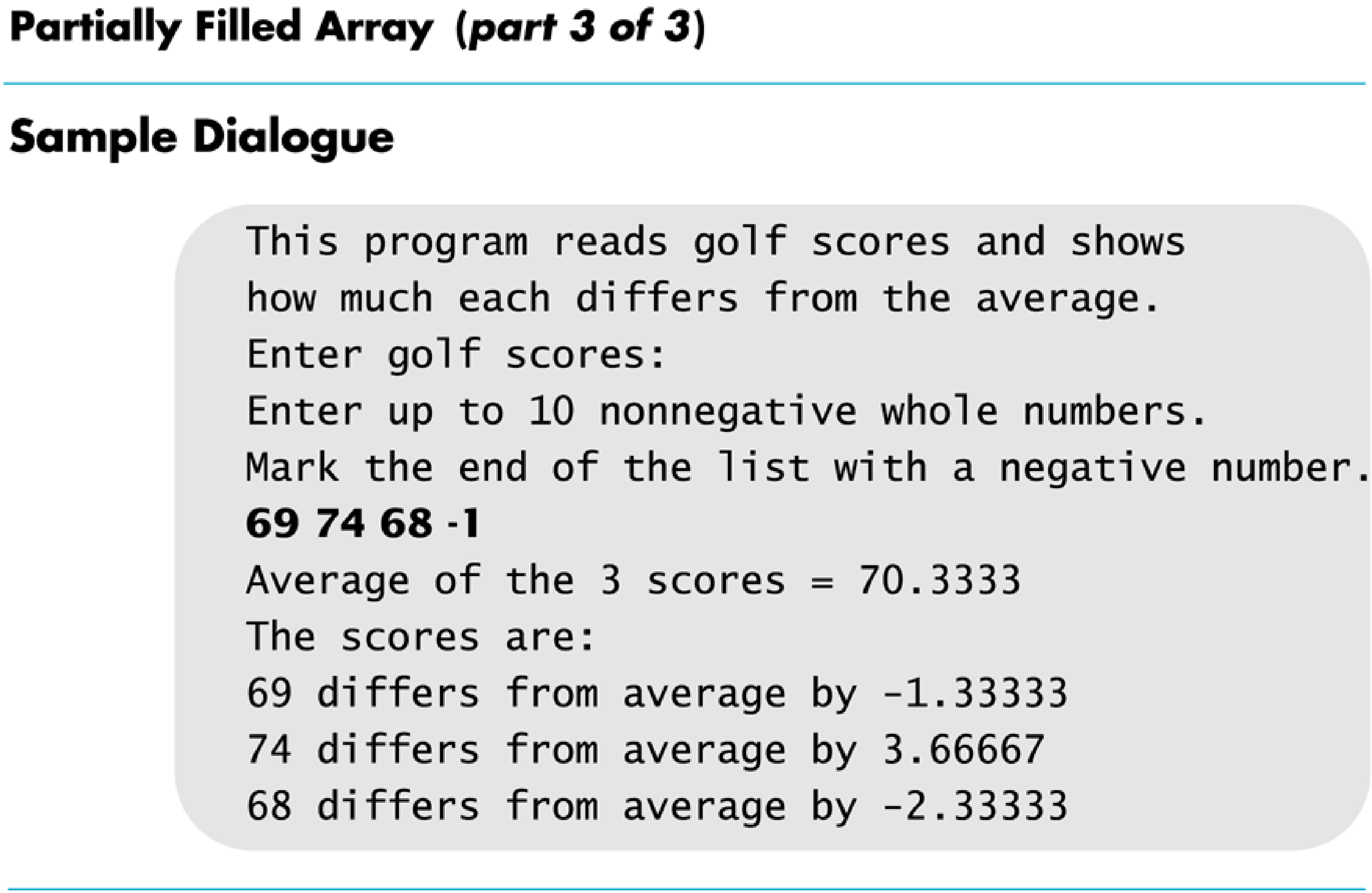
Partially Filled Array ***(part 2 of 3)***

```
        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;
    }
```

Display 7.9 (3/3)

Back

Next



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
```