Arrays

Modified from Chapter 7

Copyright © 2014 Pearson Addison-Wesley. All rights reserved.

Overview

- Introduction to Arrays
- Arrays in Functions
- Multidimensional Arrays

Copyright © 2014 Pearson Addison-Wesley. All rights reserved.

Introduction to Arrays

Modified from Section 7.1

Copyright © 2014 Pearson Addison-Wesley. All rights reserved.

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?

Copyright © 2014 Pearson Addison-Wesley. All rights reserved.

Array Declaration Syntax

- To declare an array, use the syntax:

  Type_Name Array_Name[Declared_Size];

  - Type_Name can be any type
  - Declared_Size must be a positive constant value/variable/expression
- Once declared, the array consists of the indexed variables:
  Array_Name[0] to Array_Name[Declared_Size -1]

Copyright © 2014 Pearson Addison-Wesley. All rights reserved.

Variables and Declarations

- A non-constant variable, or an expression containing non-constant variables, cannot be used to declare the size of an array

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

- This code will produce a syntax error
- Later we will see dynamic arrays which supports non-constant size

Copyright © 2014 Pearson Addison-Wesley. All rights reserved.

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 doe not produce an error message during compilation!
  - Using an out of range will cause a run-time error (segmentation fault).

Copyright © 2014 Pearson Addison-Wesley. All rights reserved.

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;
```

Copyright © 2014 Pearson Addison-Wesley. All rights reserved.

Initializing Arrays

- 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

Copyright © 2014 Pearson Addison-Wesley. All rights reserved.

Range-Based For Loops

- C++11 includes a new type of for loop, the range-based for loop, that simplifies iteration over every element in an array.
  - Syntax:

```
for (datatype varname : arrayname) {
        // varname is successively set to each
        // element in the array
}
```

  - Example:

```
int arr[ ] = {2, 4, 6, 8};
for (int x : arr)
        cout << x;
```

  - The example code outputs 2 4 6 8

Copyright © 2014 Pearson Addison-Wesley. All rights reserved.

Arrays in Functions

Modified from Section 7.2

Copyright © 2014 Pearson Addison-Wesley. All rights reserved.

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

Copyright © 2014 Pearson Addison-Wesley. All rights reserved.

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 (assume ‘score’ is an array declared previously)

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

Copyright © 2014 Pearson Addison-Wesley. All rights reserved.

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
- How can a function know the number of elements of an array argument?
  - Include a formal parameter that specifies the size of the array

Copyright © 2014 Pearson Addison-Wesley. All rights reserved.

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
  - Example: void show_the_world(const int a[ ], int size);
  - The modifier is used in both the function declaration and definition to modify the array parameter
  - The modifier is not used in function calls
  - The compiler will issue an error if you write code that changes the values stored in the array parameter

Copyright © 2014 Pearson Addison-Wesley. All rights reserved.

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

Copyright © 2014 Pearson Addison-Wesley. All rights reserved.

Multidimensional Arrays

Modified from Section 7.4

Copyright © 2014 Pearson Addison-Wesley. All rights reserved.

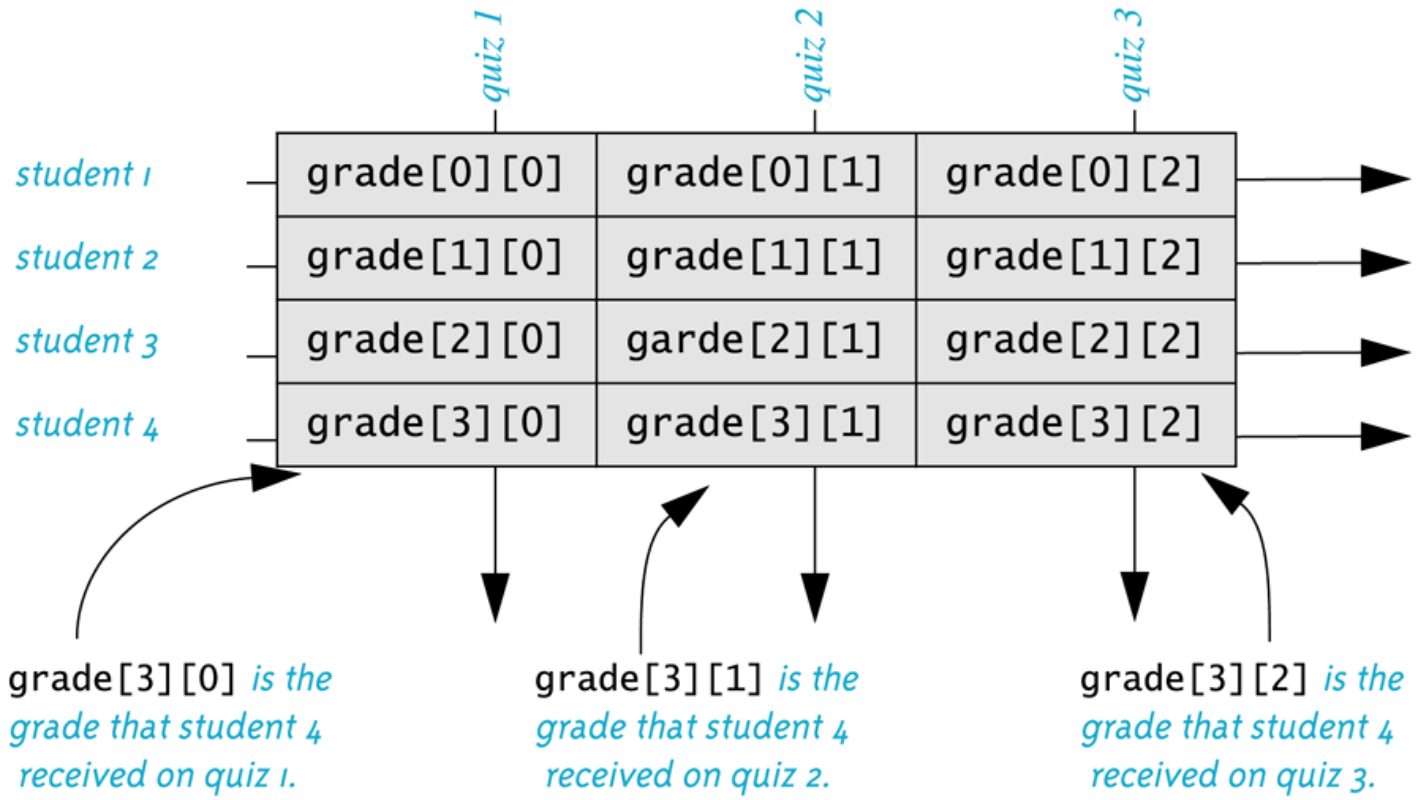
Multi-Dimensional Arrays

- C++ allows arrays with multiple index values
  - Declaration syntax:
    data_type array_name[size1][size2]....[sizeN];
  - Example: int grade[4][3];
    - Two index values are needed to access an int in the array
    - The first ranges from 0 to 3
    - The second ranges from 0 to 2
    - Each index is enclosed in its own brackets
    - grade can be visualized as an array of 4 rows and 3 columns
  - grade is actually an array of size 4
    - grade's base type is an array of 3 integers
    - grade[i]’s base type is integer

Copyright © 2014 Pearson Addison-Wesley. All rights reserved.

Index Values of grade (Display 7.15)

The Two-Dimensional Array grade



Copyright © 2014 Pearson Addison-Wesley. All rights reserved.

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 print_grades(const int grade[ ][3], int size);

Copyright © 2014 Pearson Addison-Wesley. All rights reserved.

Program Example: 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 students
    - 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

Copyright © 2014 Pearson Addison-Wesley. All rights reserved.

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

Display 7.14 (1-3)

Display 7.15

Display 7.16

Copyright © 2014 Pearson Addison-Wesley. All rights reserved.

Display 7.14 (1/3)

Back Next

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.>
```

Copyright © 2014 Pearson Addison-Wesley. All rights reserved.

Display 7.14 (2/3)

Back Next

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]
    }
}
```

Copyright © 2014 Pearson Addison-Wesley. All rights reserved.

Display 7.14 (3/3)

Back Next

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
```

Copyright © 2014 Pearson Addison-Wesley. All rights reserved.

Displays 7.14 & 7.15

```
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]
    }
}
```

grade[0][0]	grade[0][1]	grade[0][2]
grade[1][0]	grade[1][1]	grade[1][2]
grade[2][0]	garde[2][1]	grade[2][2]
grade[3][0]	grade[3][1]	grade[3][2]

Copyright © 2014 Pearson Addison-Wesley. All rights reserved.