Capitolo 4 - Arrays

Outline

4.1 Introduction

- Arrays
  - Structures of related data items
  - Static entity - same size throughout program
- A few types
  - C-like, pointer-based arrays
  - C++, arrays as objects

4.2 Arrays

- Array
  - Consecutive group of memory locations
  - Same name and type
- To refer to an element, specify
  - Array name and position number
- Format: *arrayname[position number]*
  - First element at position 0
  - **n** element array **c**:

```
c[ 0 ], c[ 1 ]...c[ n - 1 ]
```

- Array elements are like normal variables

```
c[ 0 ] =  3;
cout << c[ 0 ];
```

- Performing operations in subscript. If **x = 3**,

```
c[ 5 - 2 ] == c[ 3 ] == c[ x ]
```

4.2 Arrays

Name of array (Note that all elements of this array have the same name, **c**)

c[0]	**-45**
c[1]	**6**
c[2]	**0**
c[3]	**72**
c[4]	**1543**
c[5]	**-89**
c[6]	**0**
c[7]	**62**
c[8]	**-3**
c[9]	**1**
c[10]	**6453**
c[11]	**78**

Position number of the element within array **c**

4.3 Declaring Arrays

- Declaring arrays - specify:
  - Name
  - Type of array
  - Number of elements
  - Examples

```
int c[ 10 ];
float hi[ 3284 ];
```

- Declaring multiple arrays of same type
  - Similar format as other variables
  - Example

```
int b[ 100 ], x[ 27 ];
```

4.4 Examples Using Arrays

- Initializers

```
int n[ 5 ] = { 1, 2, 3, 4, 5 };
```

  – If not enough initializers, rightmost elements become 0
  – If too many initializers, a syntax error is generated

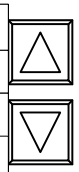
```
int n[ 5 ] = { 0 }
```

  – Sets all the elements to **0**

- If size omitted, the initializers determine it

```
int n[] = { 1, 2, 3, 4, 5 };
```

  – 5 initializers, therefore **n** is a 5 element array



Outline

1. Initialize array using a declaration

2. Define loop

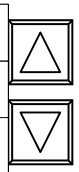
3. Print out each array element

```
// Fig. 4.4: fig04_04.cpp
// Initializing an array with a declaration
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

using std::setw;

int main()
{
   int n[ 10 ] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };

   cout << "Element" << setw( 13 ) << "Value" << endl;

   for ( int i = 0; i < 10; i++ )
      cout << setw( 7 ) << i << setw( 13 ) << n[ i ] << endl;

   return 0;
}
```

Notice how they array is declared and elements referenced.

Program Output

```
Element        Value
      0           32
      1           27
      2           64
      3           18
      4           95
      5           14
      6           90
      7           70
      8           60
      9           37
```



Outline

1. Initialize `const` variable

2. Attempt to modify variable

```
// Fig. 4.7: fig04_07.cpp
// A const object must be initialized

int main()
{
   const int x;  // Error: x must be initialized

   x = 7;        // Error:

   return 0;
}
```

Notice that **`const`** variables must be initialized because they cannot be modified later.

Program Output

```
Fig04_07.cpp:
Error E2304 Fig04_07.cpp 6: Constant variable 'x' must be
   initialized in function main()
Error E2024 Fig04_07.cpp 8: Cannot modify a const object in
   function main()
*** 2 errors in Compile ***
```

4.4 Examples Using Arrays

- Strings
  - Arrays of characters
  - All strings end with **`null`** (**`'\0'`**)
  - Examples:

```
char string1[] = "hello";
char string1[] = { 'h', 'e', 'l', 'l', 'o',
                   '\0' };
```

  - Subscripting is the same as for a normal array

    `String1[ 0 ]` is **`'h'`**

    `string1[ 2 ]` is **`'l'`**

- Input from keyboard

```
char string2[ 10 ];
cin >> string2;
```

  - Takes user input
  - Side effect: if too much text entered, data written beyond array

```
// Fig. 4 12: fig04 12.cpp
// Treating character arrays as strings
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

int main()
{
   char string1[ 20 ], string2[] = "string literal";

   cout << "Enter a string: ";
   cin >> string1;
   cout << "string1 is: " << string1
        << "\nstring2 is: " << string2
        << "\nstring1 with spaces between characters is:\n";

   for ( int i = 0; string1[ i ] != '\0'; i++ )
      cout << string1[ i ] << ' ';

   cin >> string1;  // reads "there"
   cout << "\nstring1 is: " << string1 << endl;

   cout << endl;
   return 0;
}
```

Inputted strings are separated by whitespace characters. **"there"** stayed in the buffer.

Notice how string elements are referenced like arrays.

Outline

1. Initialize strings

2. Print strings

2.1 Define loop

2.2 Print characters individually

2.3 Input string

ing

Program Output

```
Enter a string: Hello there
string1 is: Hello
string2 is: string literal
string1 with spaces between characters is:
H e l l o
string1 is: there
```

4.5 Passing Arrays to Functions

- Specify the name without any brackets
  - To pass array **`myArray`** declared as

    ```
    int myArray[ 24 ];
    ```

    to function **`myFunction`**, a function call would resemble

    ```
    myFunction( myArray, 24 );
    ```

  - Array size is usually passed to function
- Arrays passed call-by-reference
  - Value of name of array is address of the first element
  - Function knows where the array is stored
    - Modifies original memory locations
- Individual array elements passed by call-by-value
  - pass subscripted name (i.e., **`myArray[ 3 ]`**) to function

4.5 Passing Arrays to Functions

- Function prototype:

```
void modifyArray( int b[], int arraySize );
```

  - Parameter names optional in prototype
    - **`int b[]`** could be simply **`int []`**
    - **`int arraysize`** could be simply **`int`**

```
// Fig. 4.14: fig04_14.cpp
// Passing arrays and individual array elements to functions
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

using std::setw;

void modifyArray( int [], int );  // appears strange
void modifyElement( int );

int main()
{
   const int arraySize = 5;
   int i, a[ arraySize ] = { 0, 1, 2, 3, 4 };

   cout << "Effects of passing entire array call-by-reference:"
        << "\n\nThe values of the original array are:\n";

   for ( i = 0; i < arraySize; i++ )
      cout << setw( 3 ) << a[ i ];

   cout << endl;

   // array a passed call-by-reference
   modifyArray( a, arraySize );

   cout << "The values of the modified array are:\n";
```

Functions can modify entire arrays. Individual array elements are not modified by default.

No parameter names in function prototype.

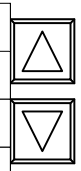
```
The values of the original array are:
  0  1  2  3  4
The values of the modified array are:
  0  2  4  6  8
```

Outline

1. Define function to take in arrays

1.1 Initialize arrays

2. Modify the array (call by reference)



Outline

2.1 Modify an element (call by value)

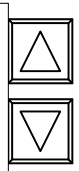
3. Print changes.

3.1 Function

```
32
33    for ( i = 0; i < arraySize; i++ )
34       cout << setw( 3 ) << a[ i ];
35
36    cout << "\n\n\n"
37         << "Effects of passing array element call-by-value:"
38         << "\n\nThe value of a[3] is " << a[ 3 ] << '\n';
39
40    modifyElement( a[ 3 ] );
41
42    cout << "The value of a[3] is " << a[ 3 ] << endl;
43
44    return 0;
45 }
46
47 // In function modifyArray, "b" points to the original
48 // array "a" in memory.
49 void modifyArray( int b[], int sizeOfArray )
50 {
51    for ( int j = 0; j < sizeOfArray; j++ )
52       b[ j ] *= 2;
53 }
54
55 // In function modifyElement, "e" is a local
56 // array element a[ 3 ] passed from main.
57 void modifyElement( int e )
58 {
59    cout << "Value in modifyElement is "
60         << ( e *= 2 ) << endl;
61 }
```

Parameter names required in function definition

```
Effects of passing array element call-by-value:


The value of a[3] is 6

Value in modifyElement is 12

The value of a[3] is 6
```



Program Output

```
Effects of passing entire array call-by-reference:

The values of the original array are:
  0  1  2  3  4
The values of the modified array are:
  0  2  4  6  8


Effects of passing array element call-by-value:

The value of a[3] is 6
Value in modifyElement is 12
The value of a[3] is 6
```

4.6 Sorting Arrays

- Sorting data
  - Important computing application
  - Virtually every organization must sort some data
    - Massive amounts must be sorted
- Bubble sort (sinking sort)
  - Several passes through the array
  - Successive pairs of elements are compared
    - If increasing order (or identical), no change
    - If decreasing order, elements exchanged
  - Repeat these steps for every element

4.6 Sorting Arrays

- Example:
  - Original: 3 4 2 6 7
  - Pass 1: 3 2 4 6 7
  - Pass 2: 2 3 4 6 7
  - Small elements "bubble" to the top

4.7 Case Study: Computing Mean, Median and Mode Using Arrays

- Mean
  - Average
- Median
  - Number in middle of sorted list
  - 1, 2, 3, 4, 5 (3 is median)
- Mode
  - Number that occurs most often
  - 1, 1, 1, 2, 3, 3, 4, 5 (1 is mode)

```
// Fig. 4.17: fig04 17.cpp
// This program introduces the topic of survey data analysis.
// It computes the mean, median, and mode of the data.
#include <iostream>

using std::cout;
using std::endl;
using std::ios;

#include <iomanip>

using std::setw;
using std::setiosflags;
using std::setprecision;

void mean( const int [], int );
void median( int [], int );
void mode( int [], int [], int );
void bubbleSort( int[], int );
void printArray( const int[], int );

int main()
{
   const int responseSize = 99;
   int frequency[ 10 ] = { 0 },
       response[ responseSize ] =
          { 6, 7, 8, 9, 8, 7, 8, 9, 8, 9,
            7, 8, 9, 5, 9, 8, 7, 8, 7, 8,
            6, 7, 8, 9, 3, 9, 8, 7, 8, 7,
            7, 8, 9, 8, 9, 8, 9, 7, 8, 9,
            6, 7, 8, 7, 8, 7, 9, 8, 9, 2,
            7, 8, 9, 8, 9, 8, 9, 7, 5, 3,
            5, 6, 7, 2, 5, 3, 9, 4, 6, 4,
```

Outline

1. Function prototypes

1.1 Initialize array

2. Call functions `mean`, `median`, and `mode`

```
34              7, 8, 9, 6, 8, 7, 8, 9, 7, 8,
35              7, 4, 4, 2, 5, 3, 8, 7, 5, 6,
36              4, 5, 6, 1, 6, 5, 7, 8, 7 };
37
38    mean( response, responseSize );
39    median( response, responseSize );
40    mode( frequency, response, responseSize );
41
42    return 0;
43 }
44
45 void mean( const int answer[], int arraySize )
46 {
47    int total = 0;
48
49    cout << "********\n  Mean\n********\n";
50
51    for ( int j = 0; j < arraySize; j++ )
52       total += answer[ j ];
53
54    cout << "The mean is the average value of the data\n"
55         << "items. The mean is equal to the total of\n"
56         << "all the data items divided by the number\n"
57         << "of data items (" << arraySize
58         << "). The mean value for\nthis run is: "
59         << total << " / " << arraySize << " = "
60         << setiosflags( ios::fixed | ios::showpoint )
61         << setprecision( 4 )
62         << static cast< double >( total ) / arraySize << "\n\n";
63 }
64
65 void median( int answer[], int size )
66 {
67    cout << "\n********\n Median\n********\n"
```


3. Define function `mean`

3.1 Define function `median`

```
        << "The unsorted array of responses is";

   printArray( answer, size );
   bubbleSort( answer, size );
   cout << "\n\nThe sorted array is";
   printArray( answer, size );
   cout << "\n\nThe median is element " << size / 2
        << " of\nthe sorted " << size
        << " element array.\nFor this run the median is "
        << answer[ size / 2 ] << "\n\n";
}

void mode( int freq[], int answer[], int size )
{
   int rating, largest = 0, modeValue = 0;

   cout << "\n********\n  Mode\n********\n";

   for ( rating = 1; rating <= 9; rating++ )
      freq[ rating ] = 0;

   for ( int j = 0; j < size; j++ )
      ++freq[ answer[ j ] ];

   cout << "Response"<< setw( 11 ) << "Frequency"
        << setw( 19 ) << "Histogram\n\n" << setw( 55 )
        << "1    1    2    2\n" << setw( 56 )
        << "5    0    5    0    5\n\n";
```

Notice how the subscript in **`frequency[]`** is the value of an element in **`response[]`** (**`answer[]`**).

Outline

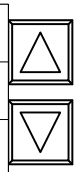
3.1 Define function `median`

3.1.1 Sort Array

3.1.2 Print middle element

3.2 Define function `mode`

3.2.1 Increase `frequency[]` depending on `response[]`



Outline

3.3 Define `bubbleSort`

```
   for ( rating = 1; rating <= 9; rating++ ) {
      cout << setw( 8 ) << rating << setw( 11 )
           << freq[ rating ] << "          ";

      if ( freq[ rating ] > largest ) {
         largest = freq[ rating ];
         modeValue = rating;
      }

      for ( int h = 1; h <= freq[ rating ]; h++ )
         cout << '*';

      cout << '\n';
   }

   cout << "The mode is the most frequent value.\n"
        << "For this run the mode is " << modeValue
        << " which occurred " << largest << " times." << endl;
}

void bubbleSort( int a[], int size )
{
   int hold;

```

Print stars depending on value of **`frequency[]`**

```
   for ( int pass = 1; pass < size; pass++ )

      for ( int j = 0; j < size - 1; j++ )

         if ( a[ j ] > a[ j + 1 ] ) {
            hold = a[ j ];
            a[ j ] = a[ j + 1 ];
            a[ j + 1 ] = hold;
         }
}

void printArray( const int a[], int size )
{
   for ( int j = 0; j < size; j++ ) {

      if ( j % 20 == 0 )
         cout << endl;

      cout << setw( 2 ) << a[ j ];
   }
}
```

Bubble sort: if elements out of order, swap them.

Outline

3.3 Define `bubbleSort`

3.3 Define `printArray`

4. Program Output

```
********
  Mean
********
The mean is the average value of the data
items. The mean is equal to the total of
all the data items divided by the number
of data items (99). The mean value for
this run is: 681 / 99 = 6.8788


********
 Median
********
The unsorted array of responses is
 6 7 8 9 8 7 8 9 8 9 7 8 9 5 9 8 7 8 7 8
 6 7 8 9 3 9 8 7 8 7 7 8 9 8 9 8 9 7 8 9
 6 7 8 7 8 7 9 8 9 2 7 8 9 8 9 8 9 7 5 3
 5 6 7 2 5 3 9 4 6 4 7 8 9 6 8 7 8 9 7 8
 7 4 4 2 5 3 8 7 5 6 4 5 6 1 6 5 7 8 7

The sorted array is
 1 2 2 2 3 3 3 3 4 4 4 4 4 5 5 5 5 5 5 5
 5 6 6 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7 7
 7 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8
 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9

The median is element 49 of
the sorted 99 element array.
For this run the median is 7
```

Outline

Program Output

```
********
  Mode
********
Response  Frequency        Histogram

                                   1    1    2    2
                              5    0    5    0    5

       1          1          *
       2          3          ***
       3          4          ****
       4          5          *****
       5          8          ********
       6          9          *********
       7         23          ***********************
       8         27          ***************************
       9         19          *******************
The mode is the most frequent value.
For this run the mode is 8 which occurred 27 times.
```

4.8 Searching Arrays: Linear Search and Binary Search

- Search array for a key value
- Linear search
  - Compare each element of array with key value
  - Useful for small and unsorted arrays
- Binary search
  - Can only be used on sorted arrays
  - Compares middle element with key
    - If equal, match found
    - If key < middle, repeat search through the first half of the array
    - If key > middle, repeat search through the last half of the array
  - Very fast; at most **n** steps, where $2^{n} >$ # of elements
    - 30 element array takes at most 5 steps
      - $2^{5} > 30$

4.9 Multiple-Subscripted Arrays

- Multiple subscripts - tables with rows, columns
  - Like matrices: specify row, then column.

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Column subscript

Array name

Row subscript

- Initialize

```
int b[ 2 ][ 2 ] = { { 1, 2 }, { 3, 4 } };
```

1	2
3	4

  - Initializers grouped by row in braces

```
int b[ 2 ][ 2 ] = { { 1 }, { 3, 4 } };
```

1	0
3	4

4.9 Multiple-Subscripted Arrays

- Referenced like normal

```
cout << b[ 0 ][ 1 ];
```

  - Will output the value of **0**
  - Cannot reference with commas

```
cout << b( 0, 1 );
```

    - Will try to call function **b**, causing a syntax error

```
// Fig. 4.23: fig04 23.cpp
// Double-subscripted array example
#include <iostream>

using std::cout;
using std::endl;
using std::ios;

#include <iomanip>

using std::setw;
using std::setiosflags;
using std::setprecision;

const int students = 3;   // number of students
const int exams = 4;      // number of exams

int minimum( int [][ exams ], int, int );
int maximum( int [][ exams ], int, int );
double average( int [], int );
void printArray( int [][ exams ], int, int );

int main()
{
   int studentGrades[ students ][ exams ] =
          { { 77, 68, 86, 73 },
            { 96, 87, 89, 78 },
            { 70, 90, 86, 81 } };

   cout << "The array is:\n";
   printArray( studentGrades, students, exams );
   cout << "\n\nLowest grade: "
        << minimum( studentGrades, students, exams )
```

Each row is a particular student, each column is the grades on the exam.

Outline

1. Initialize variables

1.1 Define functions to take double scripted arrays

1.2 Initialize `studentgrades[][]`

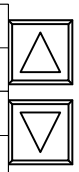
2. Call functions `minimum`, `maximum`, and `average`

```
34          << "\nHighest grade: "
35          << maximum( studentGrades, students, exams ) << '\n';
36
37     for ( int person = 0; person < students; person++ )
38        cout << "The average grade for student " << person << " is "
39             << setiosflags( ios::fixed | ios::showpoint )
40             << setprecision( 2 )
41             << average( studentGrades[ person ], exams ) << endl;
42
43     return 0;
44 }
45
46 // Find the minimum grade
47 int minimum( int grades[][ exams ], int pupils, int tests )
48 {
49     int lowGrade = 100;
50
51     for ( int i = 0; i < pupils; i++ )
52
53        for ( int j = 0; j < tests; j++ )
54
55           if ( grades[ i ][ j ] < lowGrade )
56              lowGrade = grades[ i ][ j ];
57
58     return lowGrade;
59 }
60
61 // Find the maximum grade
62 int maximum( int grades[][ exams ], int pupils, int tests )
63 {
64     int highGrade = 0;
65
66     for ( int i = 0; i < pupils; i++ )
```

Outline

2. Call functions `minimum`, `maximum`, and `average`

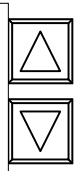
3. Define functions



Outline

3. Define functions

```
67
68       for ( int j = 0; j < tests; j++ )
69
70          if ( grades[ i ][ j ] > highGrade )
71             highGrade = grades[ i ][ j ];
72
73    return highGrade;
74 }
75
76 // Determine the average grade for a particular student
77 double average( int setOfGrades[], int tests )
78 {
79    int total = 0;
80
81    for ( int i = 0; i < tests; i++ )
82       total += setOfGrades[ i ];
83
84    return static cast< double >( total ) / tests;
85 }
86
87 // Print the array
88 void printArray( int grades[][ exams ], int pupils, int tests )
89 {
90    cout << "                  [0]  [1]  [2]  [3]";
91
92    for ( int i = 0; i < pupils; i++ ) {
93       cout << "\nstudentGrades[" << i << "] ";
94
95       for ( int j = 0; j < tests; j++ )
96          cout << setiosflags( ios::left ) << setw( 5 )
97               << grades[ i ][ j ];
98    }
99 }
```



Outline

Program Output

```
The array is:
                 [0]  [1]  [2]  [3]
studentGrades[0] 77   68   86   73
studentGrades[1] 96   87   89   78
studentGrades[2] 70   90   86   81

Lowest grade: 68
Highest grade: 96
The average grade for student 0 is 76.00
The average grade for student 1 is 87.50
The average grade for student 2 is 81.75
```