

Chapter 4 - Arrays

Outline

- 4.1 Introduction
- 4.2 Arrays
- 4.3 Declaring Arrays
- 4.4 Examples Using Arrays
- 4.5 Passing Arrays to Functions
- 4.6 Sorting Arrays
- 4.7 Case Study: Computing Mean, Median and Mode Using Arrays
- 4.8 Searching Arrays: Linear Search and Binary Search
- 4.9 Multiple-Subscripted Arrays



4.1 Introduction

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



4.2 Arrays

- Array
 - Consecutive group of memory locations
 - Same name and type (**int**, **char**, etc.)
- To refer to an element
 - Specify array name and position number (index)
 - Format: arrayname[position number]
 - First element at position 0
- N-element array **c**

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

 - Nth element as position N-1



4.2 Arrays

- Array elements like other variables
 - Assignment, printing for an integer array **c**

```
c[ 0 ] = 3;
cout << c[ 0 ];
```
- Can perform operations inside subscript


```
c[ 5 - 2 ] same as c[3]
```



4.2 Arrays

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

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

Position number of the element within array `c`



4.3 Declaring Arrays

- When declaring arrays, specify
 - Name
 - Type of array
 - Any data type
 - Number of elements
 - `type arrayName [arraySize] ;`

```
int c[ 10 ]; // array of 10 integers
float d[ 3284 ]; // array of 3284 floats
```
- Declaring multiple arrays of same type
 - Use comma separated list, like regular variables


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



4.4 Examples Using Arrays

- Initializing arrays
 - For loop
 - Set each element
 - Initializer list
 - Specify each element when array declared


```
int n[ 5 ] = { 1, 2, 3, 4, 5 };
```
 - If not enough initializers, rightmost elements 0
 - If too many syntax error
 - To set every element to same value


```
int n[ 5 ] = { 0 };
```
 - If array size omitted, initializers determine size


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

 - 5 initializers, therefore 5 element array



```

1 // Fig. 4.3: fig04_03.cpp
2 // Initializing an array.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <iomanip>
9
10 using std::setw;
11
12 int main()
13 {
14     int n[ 10 ]; // n is an array
15
16     // initialize elements of array n
17     for ( int i = 0; i < 10; i++ )
18         n[ i ] = 0; // set element at location i to 0
19
20     cout << "Element" << setw( 13 ) << "Value" << endl;
21
22     // output contents of array n in tabular format
23     for ( int j = 0; j < 10; j++ )
24         cout << setw( 7 ) << j << setw( 13 ) << n[ j ] << endl;
25

```

8


 Outline

fig04_03.cpp
(1 of 2)

Declare a 10-element array of integers.

Initialize array to 0 using a for loop. Note that the array has elements n[0] to n[9].

© 2003 Prentice Hall, Inc. All rights reserved.

```
26     return 0; // indicates successful termination
27
28 } // end main
```

Element	Value
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0



Outline

9

fig04_03.cpp
(2 of 2)

fig04_03.cpp
output (1 of 1)

© 2003 Prentice Hall, Inc.
All rights reserved.

```
1 // Fig. 4.4: fig04_04.cpp
2 // Initializing an array with a declaration.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <iomanip>
9
10 using std::setw;
11
12 int main()
13 {
14     // use initializer list to initialize array n
15     int n[ 10 ] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
16
17     cout << "Element" << setw( 13 ) << "Value" << endl;
18
19     // output contents of array n in tabular format
20     for ( int i = 0; i < 10; i++ )
21         cout << setw( 7 ) << i << setw( 13 ) << n[ i ] << endl;
22
23     return 0; // indicates successful termination
24
25 } // end main
```

Note the use of the initializer list.



Outline

10

fig04_04.cpp
(1 of 1)

© 2003 Prentice Hall, Inc.
All rights reserved.

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





  [Outline](#) 11

fig04_04.cpp
output (1 of 1)

© 2003 Prentice Hall, Inc.
All rights reserved.

4.4 Examples Using Arrays

- Array size
 - Can be specified with constant variable (**const**)
 - `const int size = 20;`
 - Constants cannot be changed
 - Constants must be initialized when declared
 - Also called named constants or read-only variables

© 2003 Prentice Hall, Inc. All rights reserved.  

```

1 // Fig. 4.5: fig04_05.cpp
2 // Initialize array s to the even integers from 2 to 20.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <iomanip>
9
10 using std::setw;
11
12 int main()
13 {
14     // constant variable can be used to
15     const int arraySize = 10;
16
17     int s[ arraySize ]; // array s has 10 e
18
19     for ( int i = 0; i < arraySize; i++ ) /
20         s[ i ] = 2 + 2 * i;
21
22     cout << "Element" << setw( 13 ) << "Value"
23

```

Note use of **const** keyword. Only **const** variables can specify array sizes.

The program becomes more scalable when we set the array size using a **const** variable. We can change **arraySize**, and all the loops will still work (otherwise, we'd have to update every loop in the program).



Outline

13

fig04_05.cpp
(1 of 2)

© 2003 Prentice Hall, Inc.
All rights reserved.

```

24 // output contents of array s in tabular format
25 for ( int j = 0; j < arraySize; j++ )
26     cout << setw( 7 ) << j << setw( 13 ) << s[ j ] << endl;
27
28 return 0; // indicates successful termination
29
30 } // end main

```

Element	Value
0	2
1	4
2	6
3	8
4	10
5	12
6	14
7	16
8	18
9	20



Outline

14

fig04_05.cpp
(2 of 2)

fig04_05.cpp
output (1 of 1)

© 2003 Prentice Hall, Inc.
All rights reserved.

```

1 // Fig. 4.6: fig04_06.cpp
2 // Using a properly initialized constant variable.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 int main()
9 {
10     const int x = 7; // initialized constant variable
11
12     cout << "The value of constant variable x is: "
13         << x << endl;
14
15     return 0; // indicates successful termination
16
17 } // end main

```

Proper initialization of
const variable.

```
The value of constant variable x is: 7
```



Outline

fig04_06.cpp
(1 of 1)

fig04_06.cpp
output (1 of 1)

© 2003 Prentice Hall, Inc.
All rights reserved.

```

1 // Fig. 4.7: fig04_07.cpp
2 // A const object must be initialized
3
4 int main()
5 {
6     const int x; // Error: x must be initialized
7
8     x = 7; // Error: cannot modify a const variable
9
10    return 0; // indicates successful termination
11
12 } // end main

```

Uninitialized **const** results
in a syntax error. Attempting
to modify the **const** is
another error.

```

d:\cphtp4_examples\ch04\Fig04_07.cpp(6) : error C2734: 'x' :
const object must be initialized if not extern
d:\cphtp4_examples\ch04\Fig04_07.cpp(8) : error C2166:
l-value specifies const object

```



Outline

fig04_07.cpp
(1 of 1)

fig04_07.cpp
output (1 of 1)

© 2003 Prentice Hall, Inc.
All rights reserved.


```

1 // Fig. 4.8: fig04_08.cpp
2 // Compute the sum of the elements of the array.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 int main()
9 {
10     const int arraySize = 10;
11
12     int a[ arraySize ] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
13
14     int total = 0;
15
16     // sum contents of array a
17     for ( int i = 0; i < arraySize; i++ )
18         total += a[ i ];
19
20     cout << "Total of array element values is " << total << endl;
21
22     return 0; // indicates successful termination
23
24 } // end main

```

```
Total of array element values is 55
```



Outline

17

fig04_08.cpp
(1 of 1)

fig04_08.cpp
output (1 of 1)

© 2003 Prentice Hall, Inc.
All rights reserved.

```

1 // Fig. 4.9: fig04_09.cpp
2 // Histogram printing program.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <iomanip>
9
10 using std::setw;
11
12 int main()
13 {
14     const int arraySize = 10;
15     int n[ arraySize ] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
16
17     cout << "Element" << setw( 13 ) << "Value"
18         << setw( 17 ) << "Histogram" << endl;
19
20     // for each element of array n, output a bar
21     for ( int i = 0; i < arraySize; i++ ) {
22         cout << setw( 7 ) << i << setw( 13 )
23             << n[ i ] << setw( 9 );
24
25         for ( int j = 0; j < n[ i ]; j++ ) // print one bar
26             cout << " * ";

```

Prints asterisks corresponding
to size of array element,
n[i].



Outline

18

fig04_09.cpp
(1 of 2)

© 2003 Prentice Hall, Inc.
All rights reserved.

```

27
28     cout << endl; // start next line of output
29
30 } // end outer for structure
31
32 return 0; // indicates successful termination
33
34 } // end main

```

Element	Value	Histogram
0	19	*****
1	3	***
2	15	*****
3	7	*****
4	11	*****
5	9	*****
6	13	*****
7	5	*****
8	17	*****
9	1	*



Outline

19

fig04_09.cpp
(2 of 2)

fig04_09.cpp
output (1 of 1)

© 2003 Prentice Hall, Inc.
All rights reserved.

```

1 // Fig. 4.10: fig04_10.cpp
2 // Roll a six-sided die 6000 times.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <iomanip>
9
10 using std::setw;
11
12 #include <cstdlib>
13 #include <ctime>
14
15 int main()
16 {
17     const int arraySize = 7;
18     int frequency[ arraySize ] = { 0 };
19
20     srand( time( 0 ) ); // seed random-number
21
22     // roll die 6000 times
23     for ( int roll = 1; roll <= 6000; roll++ )
24         ++frequency[ 1 + rand() % 6 ]; // replac
25                                     // of Fig

```

Remake of old program to roll dice. An array is used instead of 6 regular variables, and the proper element can be updated easily (without needing a **switch**).

This creates a number between 1 and 6, which determines the index of **frequency[]** that should be incremented.



Outline

20

fig04_10.cpp
(1 of 2)

© 2003 Prentice Hall, Inc.
All rights reserved.

```

26
27 cout << "Face" << setw( 13 ) << "Frequency" << endl;
28
29 // output frequency elements 1-6 in tabular format
30 for ( int face = 1; face < arraySize; face++ )
31     cout << setw( 4 ) << face
32         << setw( 13 ) << frequency[ face ] << endl;
33
34 return 0; // indicates successful termination
35
36 } // end main

```

Face	Frequency
1	1003
2	1004
3	999
4	980
5	1013
6	1001



Outline

21

fig04_10.cpp
(2 of 2)

fig04_10.cpp
output (1 of 1)

© 2003 Prentice Hall, Inc.
All rights reserved.

```

1 // Fig. 4.11: fig04_11.cpp
2 // Student poll program.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <iomanip>
9
10 using std::setw;
11
12 int main()
13 {
14     // define array sizes
15     const int responseSize = 40; // size of array responses
16     const int frequencySize = 11; // size of array frequency
17
18     // place survey responses in array responses
19     int responses[ responseSize ] = { 1, 2, 6, 4, 8, 5, 9, 7, 8,
20         10, 1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7,
21         5, 6, 6, 5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };
22
23     // initialize frequency counters to 0
24     int frequency[ frequencySize ] = { 0 };
25

```



Outline

22

fig04_11.cpp
(1 of 2)

© 2003 Prentice Hall, Inc.
All rights reserved.

```

26 // for each answer, select value of an element of array
27 // responses and use that value as subscript in array
28 // frequency to determine element to increment
29 for ( int answer = 0; answer < responseSize; answer++ )
30     ++frequency[ responses[answer] ];
31
32 // display results
33 cout << "Rating" << setw( 17 ) <<
34
35 // output frequencies in tabular form
36 for ( int rating = 1; rating < frequencySize; rating++ )
37     cout << setw( 6 ) << rating
38         << setw( 17 ) << frequency[ rating ] << endl;
39
40 return 0; // indicates successful termination
41
42 } // end main

```

responses[answer] is the rating (from 1 to 10). This determines the index in frequency[] to increment.



Outline

23

fig04_11.cpp
(2 of 2)

© 2003 Prentice Hall, Inc.
All rights reserved.

Rating	Frequency
1	2
2	2
3	2
4	2
5	5
6	11
7	5
8	7
9	1
10	3



Outline

24

fig04_11.cpp
output (1 of 1)

© 2003 Prentice Hall, Inc.
All rights reserved.

4.4 Examples Using Arrays

- Strings (more in ch. 5)
 - Arrays of characters
 - All strings end with **null** ('\0')
 - Examples
 - `char string1[] = "hello";`
 - **N**ull character implicitly added
 - `string1` has 6 elements
 - `char string1[] = { 'h', 'e', 'l', 'l', 'o', '\0' };`
 - Subscripting is the same
 - `string1[0]` is 'h'
 - `string1[2]` is 'l'



4.4 Examples Using Arrays

- Input from keyboard
 - `char string2[10];`
 - `cin >> string2;`
 - Puts user input in string
 - Stops at first whitespace character
 - Adds **n**ull character
 - If too much text entered, data written beyond array
 - We want to avoid this (section 5.12 explains how)
- Printing strings
 - `cout << string2 << endl;`
 - Does not work for other array types
 - Characters printed until **n**ull found



```

1 // Fig. 4_12: fig04_12.cpp
2 // Treating character arrays as strings.
3 #include <iostream>
4
5 using std::cout;
6 using std::cin;
7 using std::endl;
8
9 int main()
10 {
11     char string1[ 20 ], //
12     char string2[] = "string literal";
13
14     // read string from user into string1
15     cout << "Enter the string \\\\";
16     cin >> string1; // reads "hello" [space terminates input]
17
18     // output strings
19     cout << "string1 is: " << string1
20         << "\\nstring2 is: " << string2;
21
22     cout << "\\nstring1 with spaces between characters is:\\n";
23

```

Two different ways to declare strings. `string2` is initialized, and its size determined automatically.

Examples of reading strings from the keyboard and printing them out.

```

24 // output characters until null character is reached
25 for ( int i = 0; string1[ i ] != '\\0'; i++ )
26     cout << string1[ i ] << " ";
27
28 cin >> string1; // reads "there"
29 cout << "\\nstring1 is: " << string1 << endl;
30
31 return 0; // indicates successful termination
32
33 } // end main

```

Can access the characters in a string using array notation. The loop ends when the null character is found.

```

Enter the string "hello there": hello there
string1 is: hello
string2 is: string literal
string1 with spaces between characters is:
h e l l o
string1 is: there

```

4.4 Examples Using Arrays

- Recall static storage (chapter 3)
 - If **static**, local variables save values between function calls
 - Visible only in function body
 - Can declare local arrays to be static
 - Initialized to zero

```
static int array[3];
```
- If not static
 - Created (and destroyed) in every function call



```

1 // Fig. 4.13: fig04_13.cpp
2 // Static arrays are initialized to zero.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 void staticArrayInit( void ); // function prototype
9 void automaticArrayInit( void ); // function prototype
10
11 int main()
12 {
13     cout << "First call to each function:\n";
14     staticArrayInit();
15     automaticArrayInit();
16
17     cout << "\n\nSecond call to each function:\n";
18     staticArrayInit();
19     automaticArrayInit();
20     cout << endl;
21
22     return 0; // indicates successful termination
23
24 } // end main
25

```



```

26 // function to demonstrate a static array
27 void staticArrayInit( void )
28 {
29     // initializes elements to 0 first time function is called
30     static int array1[ 3 ];
31
32     cout << "\nValues on entering staticArrayInit:\n";
33
34     // output contents of array1
35     for ( int i = 0; i < 3; i++ )
36         cout << "array1[" << i << "] = " << array1[ i ] << " ";
37
38     cout << "\nValues on exiting staticArrayInit:\n";
39
40     // modify and output contents of array1
41     for ( int j = 0; j < 3; j++ )
42         cout << "array1[" << j << "] = "
43             << ( array1[ j ] += 5 ) << " ";
44
45 } // end function staticArrayInit
46

```

Static array, initialized to zero on first function call.

Array data is changed; the modified values stay.



```

47 // function to demonstrate an automatic local array
48 void automaticArrayInit( void )
49 {
50     // initializes elements each time function is called
51     int array2[ 3 ] = { 1, 2, 3 };
52
53     cout << "\n\nValues on entering automaticArrayInit:\n";
54
55     // output contents of array2
56     for ( int i = 0; i < 3; i++ )
57         cout << "array2[" << i << "] = " << array2[ i ] << " ";
58
59     cout << "\n\nValues on exiting automaticArrayInit:\n";
60
61     // modify and output contents of array2
62     for ( int j = 0; j < 3; j++ )
63         cout << "array2[" << j << "] = "
64             << ( array2[ j ] += 5 ) << " ";
65
66 } // end function automaticArrayInit

```

Automatic array, recreated with every function call.

Although the array is changed, it will be destroyed when the function exits and the changes will be lost.





First call to each function:

```
Values on entering staticArrayInit:
array1[0] = 0 array1[1] = 0 array1[2] = 0
Values on exiting staticArrayInit:
array1[0] = 5 array1[1] = 5 array1[2] = 5
```

```
Values on entering automaticArrayInit:
array2[0] = 1 array2[1] = 2 array2[2] = 3
Values on exiting automaticArrayInit:
array2[0] = 6 array2[1] = 7 array2[2] = 8
```

Second call to each function:

```
Values on entering staticArrayInit:
array1[0] = 5 array1[1] = 5 array1[2] = 5
Values on exiting staticArrayInit:
array1[0] = 10 array1[1] = 10 array1[2] = 10
```

```
Values on entering automaticArrayInit:
array2[0] = 1 array2[1] = 2 array2[2] = 3
Values on exiting automaticArrayInit:
array2[0] = 6 array2[1] = 7 array2[2] = 8
```

4.5 Passing Arrays to Functions

- Specify name without brackets
 - To pass array `myArray` to `myFunction`

```
int myArray[ 24 ];
myFunction( myArray, 24 );
```
 - Array size usually passed, but not required
 - Useful to iterate over all elements



4.5 Passing Arrays to Functions

- Arrays passed-by-reference
 - Functions can modify original array data
 - Value of name of array is address of first element
 - Function knows where the array is stored
 - Can change original memory locations
- Individual array elements passed-by-value
 - Like regular variables
 - `square(myArray[3]);`



4.5 Passing Arrays to Functions

- Functions taking arrays
 - Function prototype
 - `void modifyArray(int b[], int arraySize);`
 - `void modifyArray(int [], int);`
 - Names optional in prototype
 - Both take an integer array and a single integer
 - No need for array size between brackets
 - Ignored by compiler
 - If declare array parameter as **const**
 - Cannot be modified (compiler error)
 - `void doNotModify(const int []);`



```

1 // Fig. 4.14: fig04_14.cpp
2 // Passing arrays and individual array elements to functions.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <iomanip>
9
10 using std::setw;
11
12 void modifyArray( int [], int ); // appears strange
13 void modifyElement( int );
14
15 int main()
16 {
17     const int arraySize = 5;           // size of array a
18     int a[ arraySize ] = { 0, 1, 2, 3, 4 }; // initialize a
19
20     cout << "Effects of passing entire array by reference:"
21          << "\n\nThe values of the original array are:\n";
22
23     // output original array
24     for ( int i = 0; i < arraySize; i++ )
25         cout << setw( 3 ) << a[ i ];

```

Syntax for accepting an array in parameter list.



```

26     cout << endl;
27
28     // pass array a to modifyArray by reference.
29     modifyArray( a, arraySize );
30
31     cout << "The values of the modified array are:\n";
32
33     // output modified array
34     for ( int j = 0; j < arraySize; j++ )
35         cout << setw( 3 ) << a[ j ];
36
37     // output value of a[ 3 ]
38     cout << "\n\n"
39          << "Effects of passing array element by value:"
40          << "\n\nThe value of a[3] is ";
41
42     // pass array element a[ 3 ] by value
43     modifyElement( a[ 3 ] );
44
45     // output value of a[ 3 ]
46     cout << "The value of a[3] is " << a[ 3 ] << endl;
47
48     return 0; // indicates successful termination
49
50 } // end main
51

```

Pass array name (a) and size to function. Arrays are passed by reference.

Pass a single array element by value; the original cannot be modified.



```

52
53 // in function modifyArray, "b" points to
54 // the original array "a" in memory
55 void modifyArray( int b[], int sizeofArray )
56 {
57     // multiply each array element by 2
58     for ( int k = 0; k < sizeofArray; k++ )
59         b[ k ] *= 2;
60
61 } // end function modifyArray
62
63 // in function modifyElement, "e" is a local
64 // array element a[ 3 ] passed from main
65 void modifyElement( int e )
66 {
67     // multiply parameter by 2
68     cout << "Value in modifyElement is "
69         << ( e *= 2 ) << endl;
70
71 } // end function modifyElement

```

Although named **b**, the array points to the original array **a**. It can modify **a**'s data.

Individual array elements are passed by value, and the originals cannot be changed.

fig04_14.cpp
(3 of 3)

Effects of passing entire array 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 by value:

The value of a[3] is 6

Value in modifyElement is 12

The value of a[3] is 6

fig04_14.cpp
output (1 of 1)

```

1 // Fig. 4.15: fig04_15.cpp
2 // Demonstrating the const type qualifier.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 void tryToModifyArray( const int [] ); //
9
10 int main()
11 {
12     int a[] = { 10, 20, 30 };
13
14     tryToModifyArray( a );
15
16     cout << a[ 0 ] << ' ' << a[ 1 ] << ' ' << a[ 2 ] << '\n';
17
18     return 0; // indicates successful termination
19
20 } // end main
21

```

Array parameter declared as **const**. Array cannot be modified, even though it is passed by reference.



Outline

41

fig04_15.cpp
(1 of 2)

© 2003 Prentice Hall, Inc.
All rights reserved.

```

22 // In function tryToModifyArray, "b" cannot be used
23 // to modify the original array "a" in main.
24 void tryToModifyArray( const int b[] )
25 {
26     b[ 0 ] /= 2; // error
27     b[ 1 ] /= 2; // error
28     b[ 2 ] /= 2; // error
29
30 } // end function tryToModifyArray

```

```

d:\cphttp4_examples\ch04\Fig04_15.cpp(26) : error C2166:
l-value specifies const object
d:\cphttp4_examples\ch04\Fig04_15.cpp(27) : error C2166:
l-value specifies const object
d:\cphttp4_examples\ch04\Fig04_15.cpp(28) : error C2166:
l-value specifies const object

```



Outline

42

fig04_15.cpp
(2 of 2)

fig04_15.cpp
output (1 of 1)

© 2003 Prentice Hall, Inc.
All rights reserved.

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:
 - Go left to right, and exchange elements as necessary
 - One pass for each element
 - Original: 3 4 2 7 6
 - Pass 1: 3 2 4 6 7 (elements exchanged)
 - Pass 2: 2 3 4 6 7
 - Pass 3: 2 3 4 6 7 (no changes needed)
 - Pass 4: 2 3 4 6 7
 - Pass 5: 2 3 4 6 7
 - Small elements "bubble" to the top (like 2 in this example)



4.6 Sorting Arrays

- Swapping variables

```
int x = 3, y = 4;
y = x;
x = y;
```

- What happened?

- Both x and y are 3!
- Need a temporary variable

- Solution

```
int x = 3, y = 4, temp = 0;
temp = x; // temp gets 3
x = y;    // x gets 4
y = temp; // y gets 3
```



```
1 // Fig. 4.16: fig04_16.cpp
2 // This program sorts an array's values into ascending order.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <iomanip>
9
10 using std::setw;
11
12 int main()
13 {
14     const int arraySize = 10; // size of array a
15     int a[ arraySize ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
16     int hold; // temporary location used to swap array elements
17
18     cout << "Data items in original order\n";
19
20     // output original array
21     for ( int i = 0; i < arraySize; i++ )
22         cout << setw( 4 ) << a[ i ];
23
```



47

```

24 // bubble sort
25 // loop to control number of passes
26 for ( int pass = 0; pass < arraySize - 1; pass++ )
27
28 // loop to control number of comparisons per pass
29 for ( int j = 0; j < arraySize - 1; j++ )
30
31 // compare side-by-side elements and swap
32 // first element is greater than second element
33 if ( a[ j ] > a[ j + 1 ] )
34     hold = a[ j ];
35     a[ j ] = a[ j + 1 ];
36     a[ j + 1 ] = hold;
37
38 } // end if
39

```

Do a pass for each element in the array.

fig04_16.cpp (2 of 3)

If the element on the left (index j) is larger than the element on the right (index j + 1), then we swap them. Remember the need of a temp variable.

© 2003 Prentice Hall, Inc. All rights reserved.

48

```

40 cout << "\nData items in ascending order\n";
41
42 // output sorted array
43 for ( int k = 0; k < arraySize; k++ )
44     cout << setw( 4 ) << a[ k ];
45
46 cout << endl;
47
48 return 0; // indicates successful termination
49
50 } // end main

```

Outline

fig04_16.cpp (3 of 3)

fig04_16.cpp output (1 of 1)

```

Data items in original order
 2  6  4  8 10 12 89 68 45 37
Data items in ascending order
 2  4  6  8 10 12 37 45 68 89

```

© 2003 Prentice Hall, Inc. All rights reserved.

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

- Mean
 - Average (sum/number of elements)
- Median
 - Number in middle of sorted list
 - 1, 2, 3, 4, 5 (3 is median)
 - If even number of elements, take average of middle two
- Mode
 - Number that occurs most often
 - 1, 1, 1, 2, 3, 3, 4, 5 (1 is mode)



```

1 // Fig. 4.17: fig04_17.cpp
2 // This program introduces the topic of survey data analysis.
3 // It computes the mean, median, and mode of the data.
4 #include <iostream>
5
6 using std::cout;
7 using std::endl;
8 using std::fixed;
9 using std::showpoint;
10
11 #include <iomanip>
12
13 using std::setw;
14 using std::setprecision;
15
16 void mean( const int [], int );
17 void median( int [], int );
18 void mode( int [], int [], int );
19 void bubbleSort( int[], int );
20 void printArray( const int[], int );
21
22 int main()
23 {
24     const int responseSize = 99; // size of array responses
25

```



```

26  int frequency[ 10 ] = { 0 }; // initialize array frequency
27
28  // initialize array responses
29  int response[ responseSize ] =
30      { 6, 7, 8, 9, 8, 7, 8, 9, 8, 9,
31        7, 8, 9, 5, 9, 8, 7, 8, 7, 8,
32        6, 7, 8, 9, 3, 9, 8, 7, 8, 7,
33        7, 8, 9, 8, 9, 8, 9, 7, 8, 9,
34        6, 7, 8, 7, 8, 7, 9, 8, 9, 2,
35        7, 8, 9, 8, 9, 8, 9, 7, 5, 3,
36        5, 6, 7, 2, 5, 3, 9, 4, 6, 4,
37        7, 8, 9, 6, 8, 7, 8, 9, 7, 8,
38        7, 4, 4, 2, 5, 3, 8, 7, 5, 6,
39        4, 5, 6, 1, 6, 5, 7, 8, 7 };
40
41  // process responses
42  mean( response, responseSize );
43  median( response, responseSize );
44  mode( frequency, response, responseSize );
45
46  return 0; // indicates successful termination
47
48 } // end main
49

```



```

50 // calculate average of all response values
51 void mean( const int answer[], int arraySize )
52 {
53     int total = 0;
54
55     cout << "*****\n Mean\n*****\n";
56
57     // total response values
58     for ( int i = 0; i < arraySize; i++ )
59         total += answer[ i ];
60
61     // format and output results
62     cout << fixed << setprecision( 4 );
63
64     cout << "The mean is the average value of the data\n"
65           << "items. The mean is equal to the total sum of
66           << "all the data items divided by the number
67           << "of data items (" << arraySize
68           << "). The mean value for\nthis run is: "
69           << total << " / " << arraySize << " = "
70           << static_cast< double >( total ) / arraySize
71           << "\n\n";
72
73 } // end function mean
74

```

We cast to a double to get decimal points for the average (instead of an integer).



```

75 // sort array and determine median element's value
76 void median( int answer[], int size )
77 {
78     cout << "\n*****\n Median\n*****\n"
79         << "The unsorted array of
80
81     printArray( answer, size ); //
82
83     bubbleSort( answer, size ); // sort array
84
85     cout << "\n\nThe sorted array is";
86     printArray( answer, size ); // output sorted array
87
88     // display median element
89     cout << "\n\nThe median is element " << size / 2
90         << " of\nthe sorted " << size
91         << " element array.\nFor this run the median is "
92         << answer[ size / 2 ] << "\n\n";
93
94 } // end function median
95

```

Sort array by passing it to a function. This keeps the program modular.



```

96 // determine most frequent response
97 void mode( int freq[], int answer[], int size )
98 {
99     int largest = 0; // represents largest frequency
100    int modeValue = 0; // represents most frequent response
101
102    cout << "\n*****\n Mode\n*****\n";
103
104    // initialize frequencies to 0
105    for ( int i = 1; i <= 9; i++ )
106        freq[ i ] = 0;
107
108    // summarize frequencies
109    for ( int j = 0; j < size; j++ )
110        ++freq[ answer[ j ] ];
111
112    // output headers for result columns
113    cout << "Response" << setw( 11 ) << "Frequency"
114        << setw( 19 ) << "Histogram\n\n" << setw( 55 )
115        << "1 1 2 2\n" << setw( 56 )
116        << "5 0 5 0 5\n\n";
117

```



```

118 // output results
119 for ( int rating = 1; rating <= 9; rating++ ) {
120     cout << setw( 8 ) << rating << "
121         << freq[ rating ] << "
122
123     // keep track of mode value and
124     if ( freq[ rating ] > largest ) {
125         largest = freq[ rating ];
126         modeValue = rating;
127
128     } // end if
129
130     // output histogram bar representing frequency value
131     for ( int k = 1; k <= freq[ rating ]; k++ )
132         cout << ' *';
133
134     cout << '\n'; // begin new line of output
135
136 } // end outer for
137
138 // display the mode value
139 cout << "The mode is the most frequent value.\n"
140     << "For this run the mode is " << modeValue
141     << " which occurred " << largest << " times." << endl;
142
143 } // end function mode

```

The mode is the value that occurs most often (has the highest value in freq).



```

144
145 // function that sorts an array with bubble sort algorithm
146 void bubbleSort( int a[], int size )
147 {
148     int hold; // temporary location used to swap elements
149
150     // loop to control number of passes
151     for ( int pass = 1; pass < size; pass++ )
152
153         // loop to control number of comparisons per pass
154         for ( int j = 0; j < size - 1; j++ )
155
156             // swap elements if out of order
157             if ( a[ j ] > a[ j + 1 ] ) {
158                 hold = a[ j ];
159                 a[ j ] = a[ j + 1 ];
160                 a[ j + 1 ] = hold;
161
162             } // end if
163
164 } // end function bubbleSort
165

```



```

166 // output array contents (20 values per row)
167 void printArray( const int a[], int size )
168 {
169     for ( int i = 0; i < size; i++ ) {
170
171         if ( i % 20 == 0 ) // begin new line every 20 values
172             cout << endl;
173
174             cout << setw( 2 ) << a[ i ];
175
176     } // end for
177
178 } // end function printArray

```



```

*****
  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 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 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 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

```





```

*****
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
 - Start at one end, go to other
 - Useful for small and unsorted arrays
 - Inefficient
 - If search key not present, examines every element



4.8 Searching Arrays: Linear Search and Binary Search

- Binary search
 - Only used with sorted arrays
 - Compare middle element with key
 - If equal, match found
 - If key < middle
 - Repeat search on first half of array
 - If key > middle
 - Repeat search on last half
 - Very fast
 - At most N steps, where $2^N > \#$ of elements
 - 30 element array takes at most 5 steps
 - $2^5 > 30$



```

1 // Fig. 4.19: fig04_19.cpp
2 // Linear search of an array.
3 #include <iostream>
4
5 using std::cout;
6 using std::cin;
7 using std::endl;
8
9 int linearSearch( const int [], int, int ); // prototype
10
11 int main()
12 {
13     const int arraySize = 100; // size of array a
14     int a[ arraySize ]; // create array a
15     int searchKey; // value to locate in a
16
17     for ( int i = 0; i < arraySize; i++ ) // create some data
18         a[ i ] = 2 * i;
19
20     cout << "Enter integer search key: ";
21     cin >> searchKey;
22
23     // attempt to locate searchKey in array a
24     int element = linearSearch( a, searchKey, arraySize );
25

```

Takes array, search key, and array size.



Outline

fig04_19.cpp
(1 of 2)

```
26 // display results
27 if ( element != -1 )
28     cout << "Found value in element " << element << endl;
29 else
30     cout << "Value not found" << endl;
31
32 return 0; // indicates successful termination
33
34 } // end main
35
36 // compare key to every element of array until location is
37 // found or until end of array is reached; return subscript of
38 // element if key or -1 if key not found
39 int linearSearch( const int array[], int key, int sizeOfArray )
40 {
41     for ( int j = 0; j < sizeOfArray; j++ )
42
43         if ( array[ j ] == key ) // if found,
44             return j;           // return location of key
45
46     return -1; // key not found
47
48 } // end function linearSearch
```



Outline

63

fig04_19.cpp
(2 of 2)

© 2003 Prentice Hall, Inc.
All rights reserved.

```
Enter integer search key: 36
Found value in element 18

Enter integer search key: 37
Value not found
```



Outline

64

fig04_19.cpp
output (1 of 1)

© 2003 Prentice Hall, Inc.
All rights reserved.


```

1 // Fig. 4.20: fig04_20.cpp
2 // Binary search of an array.
3 #include <iostream>
4
5 using std::cout;
6 using std::cin;
7 using std::endl;
8
9 #include <iomanip>
10
11 using std::setw;
12
13 // function prototypes
14 int binarySearch( const int [], int, int, int, int );
15 void printHeader( int );
16 void printRow( const int [], int, int, int, int );
17
18 int main()
19 {
20     const int arraySize = 15; // size of array a
21     int a[ arraySize ];      // create array a
22     int key;                 // value to locate in a
23
24     for ( int i = 0; i < arraySize; i++ ) // create some data
25         a[ i ] = 2 * i;
26

```



Outline

65



fig04_20.cpp
(1 of 6)

© 2003 Prentice Hall, Inc.
All rights reserved.

```

27     cout << "Enter a number between 0 and 28: ";
28     cin >> key;
29
30     printHeader( arraySize );
31
32     // search for key in array a
33     int result =
34         binarySearch( a, key, 0, arraySize - 1, arraySize );
35
36     // display results
37     if ( result != -1 )
38         cout << '\n' << key << " found in array element "
39             << result << endl;
40     else
41         cout << '\n' << key << " not found" << endl;
42
43     return 0; // indicates successful termination
44 } // end main
45
46

```



Outline

66



fig04_20.cpp
(2 of 6)

© 2003 Prentice Hall, Inc.
All rights reserved.

```

47 // function to perform binary search of an array
48 int binarySearch( const int b[], int searchKey, int low,
49 int high, int size )
50 {
51 int middle;
52
53 // loop until low subscript is greater than high
54 while ( low <= high ) {
55
56 // determine middle element of subarray being searched
57 middle = ( low + high ) / 2;
58
59 // display subarray used in this loop iteration
60 printRow( b, low, middle, high, size );
61

```

Determine middle element



Outline

67

fig04_20.cpp
(3 of 6)

© 2003 Prentice Hall, Inc.
All rights reserved.

```

62 // if searchKey matches middle element, return middle
63 if ( searchKey == b[ middle ] ) // match
64 return middle;
65
66 else
67
68 // if searchKey less than middle element
69 // set new high element
70 if ( searchKey < b[ middle ] )
71 high = middle - 1; // search high end
72
73 // if searchKey greater than middle element
74 // set new low element
75 else
76 low = middle + 1; // search low end
77
78 }
79 return -1; // searchKey not found
80
81 } // end function binarySearch

```

Use the rule of binary search:
If key equals middle, match

If less, search low end

If greater, search high end

Loop sets low, middle and high dynamically. If searching the high end, the new low is the element above the middle.



Outline

68

fig04_20.cpp
(4 of 6)

© 2003 Prentice Hall, Inc.
All rights reserved.

```

82
83 // print header for output
84 void printHeader( int size )
85 {
86     cout << "\nSubscripts:\n";
87
88     // output column heads
89     for ( int j = 0; j < size; j++ )
90         cout << setw( 3 ) << j << ' ';
91
92     cout << '\n'; // start new line of output
93
94     // output line of - characters
95     for ( int k = 1; k <= 4 * size; k++ )
96         cout << '-';
97
98     cout << endl; // start new line of output
99
100 } // end function printHeader
101

```



```

102 // print one row of output showing the current
103 // part of the array being processed
104 void printRow( const int b[], int low, int mid,
105              int high, int size )
106 {
107     // loop through entire array
108     for ( int m = 0; m < size; m++ )
109
110         // display spaces if outside current subarray range
111         if ( m < low || m > high )
112             cout << "  ";
113
114         // display middle element marked with a *
115         else
116
117             if ( m == mid ) // mark middle value
118                 cout << setw( 3 ) << b[ m ] << '*';
119
120         // display other elements in subarray
121         else
122             cout << setw( 3 ) << b[ m ] << ' ';
123
124     cout << endl; // start new line of output
125
126 } // end function printRow

```



Enter a number between 0 and 28: 6

Subscripts:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

0 2 4 6 8 10 12 14* 16 18 20 22 24 26 28
0 2 4 6* 8 10 12

6 found in array element 3

Enter a number between 0 and 28: 25

Subscripts:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

0 2 4 6 8 10 12 14* 16 18 20 22 24 26 28
16 18 20 22* 24 26 28
24 26* 28
24*

25 not found



Outline

71



fig04_20.cpp
output (1 of 2)

© 2003 Prentice Hall, Inc.
All rights reserved.

Enter a number between 0 and 28: 8

Subscripts:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

0 2 4 6 8 10 12 14* 16 18 20 22 24 26 28
0 2 4 6* 8 10 12
8 10* 12
8*

8 found in array element 4



Outline

72



fig04_20.cpp
output (2 of 2)

© 2003 Prentice Hall, Inc.
All rights reserved.

4.9 Multiple-Subscripted Arrays

- Multiple subscripts
 - `a[i][j]`
 - Tables with rows and columns
 - Specify row, then column
 - “Array of arrays”
 - `a[0]` is an array of 4 elements
 - `a[0][0]` is the first element of that array

	Column 0	Column 1	Column 2	Column 3
Row 0	<code>a[0][0]</code>	<code>a[0][1]</code>	<code>a[0][2]</code>	<code>a[0][3]</code>
Row 1	<code>a[1][0]</code>	<code>a[1][1]</code>	<code>a[1][2]</code>	<code>a[1][3]</code>
Row 2	<code>a[2][0]</code>	<code>a[2][1]</code>	<code>a[2][2]</code>	<code>a[2][3]</code>

Array name Column subscript
 Row subscript



4.9 Multiple-Subscripted Arrays

- To initialize
 - Default of 0
 - Initializers grouped by row in braces
- ```
int b[2][2] = { { 1, 2 }, { 3, 4 } };
```

|   |   |
|---|---|
| 1 | 2 |
| 3 | 4 |

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

|   |   |
|---|---|
| 1 | 0 |
| 3 | 4 |



## 4.9 Multiple-Subscripted Arrays

- Referenced like normal

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

|   |   |
|---|---|
| 1 | 0 |
| 3 | 4 |

- Outputs 0
- Cannot reference using commas

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

- Syntax error

- Function prototypes

- Must specify sizes of subscripts
  - First subscript not necessary, as with single-scripted arrays
- `void printArray( int [][] 3 ] );`



```

1 // Fig. 4.22: fig04_22.cpp
2 // Initializing multidimensional arrays.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 void printArray(int [][] 3]);
9
10 int main()
11 {
12 int array1[2][3] = { { 1, 2, 3 }, { 4, 5, 6 } };
13 int array2[2][3] = { 1, 2, 3, 4, 5 };
14 int array3[2][3] = { { 1, 2 }, { 4 } };
15
16 cout << "Values in array1 by row are:" << endl;
17 printArray(array1);
18
19 cout << "Values in array2 by row are:" << endl;
20 printArray(array2);
21
22 cout << "Values in array3 by row are:" << endl;
23 printArray(array3);
24
25 return 0; // indicates successful termination
26
27 } // end main

```

Note the format of the prototype.

Note the various initialization styles. The elements in **array2** are assigned to the first row and then the second.

fig04\_22.cpp  
(1 of 2)

Outline

```

28
29 // function to output array with two rows
30 void printArray(int a[][3])
31 {
32 for (int i = 0; i < 2; i++) { // f
33
34 for (int j = 0; j < 3; j++) // output column values
35 cout << a[i][j] << ' ';
36
37 cout << endl; // start new line of output
38
39 } // end outer for structure
40
41 } // end function printArray

```

For loops are often used to iterate through arrays. Nested loops are helpful with multiple-subscripted arrays.



Outline

fig04\_22.cpp  
(2 of 2)

fig04\_22.cpp  
output (1 of 1)

```

Values in array1 by row are:
1 2 3
4 5 6
Values in array2 by row are:
1 2 3
4 5 0
Values in array3 by row are:
1 2 0
4 0 0

```

© 2003 Prentice Hall, Inc.  
All rights reserved.

## 4.9 Multiple-Subscripted Arrays

- Next: program showing initialization
  - After, program to keep track of students grades
  - Multiple-subscripted array (table)
  - Rows are students
  - Columns are grades

|          | Quiz1 | Quiz2 |
|----------|-------|-------|
| Student0 | 95    | 85    |
| Student1 | 89    | 80    |



```

1 // Fig. 4.23: fig04_23.cpp
2 // Double-subscripted array example.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7 using std::fixed;
8 using std::left;
9
10 #include <iomanip>
11
12 using std::setw;
13 using std::setprecision;
14
15 const int students = 3; // number of students
16 const int exams = 4; // number of exams
17
18 // function prototypes
19 int minimum(int [][] exams, int, int);
20 int maximum(int [][] exams, int, int);
21 double average(int [], int);
22 void printArray(int [][] exams, int, int);
23

```



Outline

79



fig04\_23.cpp  
(1 of 6)

© 2003 Prentice Hall, Inc.  
All rights reserved.

```

24 int main()
25 {
26 // initialize student grades for three students (rows)
27 int studentGrades[students][exams] =
28 { { 77, 68, 86, 73 },
29 { 96, 87, 89, 78 },
30 { 70, 90, 86, 81 } };
31
32 // output array studentGrades
33 cout << "The array is:\n";
34 printArray(studentGrades, students, exams);
35
36 // determine smallest and largest grade values
37 cout << "\n\nLowest grade: "
38 << minimum(studentGrades, students, exams)
39 << "\n\nHighest grade: "
40 << maximum(studentGrades, students, exams) << '\n';
41
42 cout << fixed << setprecision(2);
43

```



Outline

80



fig04\_23.cpp  
(2 of 6)

© 2003 Prentice Hall, Inc.  
All rights reserved.



```

44 // calculate average grade for each student
45 for (int person = 0; person < students; person++)
46 cout << "The average grade for student " << person
47 << " is "
48 << average(studentGrades[person], exams)
49 << endl;
50
51 return 0; // indicates successful termin
52
53 } // end main
54
55 // find minimum grade
56 int minimum(int grades[][exams], int pupils)
57 {
58 int lowGrade = 100; // initialize to highest possible grade
59
60 for (int i = 0; i < pupils; i++)
61
62 for (int j = 0; j < tests; j++)
63
64 if (grades[i][j] < lowGrade)
65 lowGrade = grades[i][j];
66
67 return lowGrade;
68
69 } // end function minimum

```

Determines the average for one student. We pass the array/row containing the student's grades. Note that `studentGrades [ 0 ]` is itself an array.



```

70
71 // find maximum grade
72 int maximum(int grades[][exams], int pupils, int tests)
73 {
74 int highGrade = 0; // initialize to lowest possible grade
75
76 for (int i = 0; i < pupils; i++)
77
78 for (int j = 0; j < tests; j++)
79
80 if (grades[i][j] > highGrade)
81 highGrade = grades[i][j];
82
83 return highGrade;
84
85 } // end function maximum
86

```



```

87 // determine average grade for particular student
88 double average(int setOfGrades[], int tests)
89 {
90 int total = 0;
91
92 // total all grades for one student
93 for (int i = 0; i < tests; i++)
94 total += setOfGrades[i];
95
96 return static_cast< double >(total) / tests; // average
97
98 } // end function maximum

```



Outline

83



fig04\_23.cpp  
(5 of 6)

© 2003 Prentice Hall, Inc.  
All rights reserved.

```

99
100 // Print the array
101 void printArray(int grades[][exams], int pupils, int tests)
102 {
103 // set left justification and output column heads
104 cout << left << " [0] [1] [2] [3]";
105
106 // output grades in tabular format
107 for (int i = 0; i < pupils; i++) {
108
109 // output label for row
110 cout << "\nstudentGrades[" << i << "] ";
111
112 // output one grades for one student
113 for (int j = 0; j < tests; j++)
114 cout << setw(5) << grades[i][j];
115
116 } // end outer for
117
118 } // end function printArray

```



Outline

84



fig04\_23.cpp  
(6 of 6)

© 2003 Prentice Hall, Inc.  
All rights reserved.

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



Outline

85

**fig04\_23.cpp  
output (1 of 1)**