

# CHAPTER 3

## ARRAYS

Dr. Shady Yehia Elmashad



# Outline

- 1. Introduction**
- 2. Arrays**
- 3. Declaring Arrays**
- 4. Examples Using Arrays**
- 5. Multidimensional Arrays**
- 6. Multidimensional Arrays Examples**
- 7. Examples Using Strings**
- 8. Passing Arrays to Functions**
- 9. Sorting Arrays**
- 10. Case Study: Computing Mean, Median and Mode Using Arrays**
- 11. Searching Arrays: Linear Search and Binary Search**
- 12. Multiple-Subscripted Arrays Example**

## 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**:  
 $c[ 0 ], c[ 1 ] \dots c[ n - 1 ]$
- Array elements are like normal variables  
 $c[ 0 ] = 3;$   
 $\text{cout} \ll c[ 0 ];$
- Performing operations in subscript. If  $x = 3$ ,  
 $c[ 5 - 2 ] == c[ 3 ] == c[ x ]$

# 2. Arrays

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

↓

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

↑

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

# 3. Declaring Arrays

```
array_type array_name [ no. of element ] ;
```

Example:

```
int X [ 5 ] ;
```

```
X [ 0 ] = 1 ; X [ 1 ] = 10 ; X [ 2 ] = 2 ;
```

```
X [ 3 ] = 8 ; X [ 4 ] = 12 ;
```

Or

```
int X [ 5 ] = { 1, 10, 2, 8, 12 } ;
```

**X**

<i>x[0]</i>	1
<i>x[1]</i>	10
<i>x[2]</i>	2
<i>x[3]</i>	8
<i>x[4]</i>	12

# 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. 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

Notice how the array is declared and elements referenced.

```
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     int n[ 10 ] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
15
16     cout << "Element" << setw( 13 ) << "Value" << endl;
17
18     for ( int i = 0; i < 10; i++ )
19         cout << setw( 7 ) << i << setw( 13 ) << n[ i ] << endl;
20
21     return 0;
22 }
```

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

## Program Output



## 4. Examples Using Arrays

What is the O/P?

```
# include < iostream.h >
void main ( ) {
int list [ 10 ] = { 0 };
int h;
for ( h = 0; h < 5; h ++ )
list [ 2 * h + 1 ] = h + 2 ;
for ( h = 0; h < 10; h ++ )
cout << list [ h ] ;
}
```

```
list [ 1 ] = 2
list [ 3 ] = 3
list [ 5 ] = 4
list [ 7 ] = 5
list [ 9 ] = 6
```

Output: 0203040506

## 4. Examples Using Arrays

What is the O/P?

```
# include < iostream.h >
void main ( ) {
int  list [ 10 ] = { 2, 1, 2, 1, 1, 2, 3, 2, 1,
2 };
cout << list [ 2 ] ;
cout << list [ list [ 2 ] ] ;
cout << list [ list [ 2 ] + list [ 3 ] ] ;
cout << list [ list [ list [ 2 ] ] ] ;
}
```

Output:            2 2 1 2

## 4. Examples Using Arrays

### What is the O/P?

```
# include < iostream.h >
void main ( ) {
int list [ 10 ] = { 2, 1, 2, 4, 1, 2, 0, 2, 1, 2 };
int line [ 10 ];
for ( int h = 0 ; h < 10 ; h ++ )
line [ h ] = list [ 9 - h ]
for ( int k = 0 ; k < 10 ; k ++ ) {
cout << list [ k ] << endl << line [ k ];
}
}
```

Output:                    2124120212  
                                 2120214212

# 5. Multidimensional Arrays

```
array_type array_name [no. of rows] [no. of columns];
```

- Example:

```
float Salary [ 3 ] [ 2 ] = { 12.0, 6.0, 9.0, 0.0, 10.5, 5.2 } ;
```

- OR

```
float Salary [ 3 ] [ 2 ] = { {12.0, 6.0} , {9.0, 0.0} , {10.5, 5.2} } ;
```

Salary [ 0 ] [ ]	12.0	6.0
Salary [ 1 ] [ ]	9.0	0.0
Salary [ 2 ] [ ]	10.5	5.2
	Salary [ ] [ 0 ]	Salary [ ] [ 1 ]

# 5. Multidimensional Arrays

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

	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>

- 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

# 5. Multidimensional 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

## 6. Examples: Adding Two Matrices

```
# include < iostream.h >
void main ( ) {
int A [3] [5], B [3] [5], C [3] [5];    // Program to add two integer matrices
for ( int i = 0 ; i < 3 ; i ++ ) {      // A(3,5) and B (3,5) in a matrix C (3,5)
    for ( int j = 0 ; j < 5 ; j ++ ) {
        cin >> A [ i ] [ j ] ; } }
for ( int i = 0 ; i < 3 ; i ++ ) {
    for ( int j = 0 ; j < 5 ; j ++ ) {
        cin >> B [ i ] [ j ] ; } }
for ( int i = 0 ; i < 3 ; i ++ ) {
    for ( int j = 0 ; j < 5 ; j ++ ) {
        C [ i ] [ j ] = A [ i ] [ j ] + B [ i ] [ j ] ;
        cout << C [ i ] [ j ] ; } }
}
```

# 6. Examples: Multiply Two Matrices

```
# include < iostream.h >
void main ( ) {
float A [4] [5], B [5] [7], C [4] [7] ; // Program to add two float matrices
for ( int i = 0 ; i < 4 ; i ++ ) { // A(4,5) and B (5,7) in a matric C (4,7)
    for ( int j = 0 ; j < 5 ; j ++ ) {
        cin >> A [ i ] [ j ] ; } }
for ( int i = 0 ; i < 5 ; i ++ ) {
    for ( int j = 0 ; j < 7 ; j ++ ) {
        cin >> B [ i ] [ j ] ; } }
for ( int i = 0 ; i < 4 ; i ++ ) {
    for ( int j = 0 ; j < 7 ; j ++ ) {
        C [ i ] [ j ] = 0 ;
        for ( int k = 0 ; k < 5 ; j ++ ) {
            C [ i ] [ j ] = C [ i ] [ j ] + A [ i ] [ k ] * B [ k ] [ j ] ;
        } } }
}
```



## 6. Examples: Adding All Elements of an Array

```
# include < iostream.h >
void main ( ) {
int X [5] [5] ;          /* This program reads an array
(5*5) from the user and sum its elements */
int total = 0 ;
for ( int i = 0 ; i < 5 ; i ++ ) {
    for ( int j = 0 ; j < 5 ; j ++ ) {
        cin >> X [ i ] [ j ] ; } }
for ( int i = 0 ; i < 5 ; i ++ ) {
    for ( int j = 0 ; j < 5 ; j ++ ) {
        total = total + X [ i ] [ j ] ; } }
cout << " The summation of the array elements is " <<
total << endl ;
}
```

## 6. Examples: Adding the diagonal Elements of an Array

```
# include < iostream.h >
void main ( ) {
int X [5] [5];           // This program reads an array (5*5)
int sumdiagonal = 0 ;   // from the user and sum its diagonal elements
for ( int i = 0 ; i < 5 ; i ++ ) {
    for ( int j = 0 ; j < 5 ; j ++ ) {
        cin >> X [ i ] [ j ] ; } }
for ( int i = 0 ; i < 5 ; i ++ ) {
    for ( int j = 0 ; j < 5 ; j ++ ) {
        if ( i == j ) {
            sumdiagonal = sumdiagonal + X [ i ] [ j ] ;
        } } }
cout << " The summation of the diagonal elements is " <<
sumdiagonal ;
}
```

## 6. Examples: Min. Value in an Array and Its Location

```
# include < iostream.h >
void main ( ) {
int X [5] [5];           // This program reads an array (5*5)
int min, mr = 0, mc = 0; // from the user and find the min. value and its location
for ( int i = 0 ; i < 5 ; i ++ ) {
    for ( int j = 0 ; j < 5 ; j ++ ) {
        cin >> X [ i ] [ j ] ; } }
min = X [ 0 ] [ 0 ] ;
for ( int i = 0 ; i < 5 ; i ++ ) {
    for ( int j = 0 ; j < 5 ; j ++ ) {
        if ( min > X [ i ] [ j ] ) {
            min = X [ i ] [ j ] ; mr = i ; mc = j ;
        } } }
cout << " The min. value in this array is " << min << " , " ;
cout << " and its location is " << mr << " , " << mc ;
}
```

## 6. Examples: Array Transpose

```
# include < iostream.h >
void main ( )
{
int X [5] [5], Y [5] [5];
/* This program reads an array (5*5) from the user and changing
its rows into columns ( Transpose )*/
for ( int i = 0 ; i < 5 ; i ++ ) {
for ( int j = 0 ; j < 5 ; j ++ ) {
Y [i] [j] = X [j] [i];
cout << X [i] [j];
}
}
}
```

# 7. Examples Using Strings

- 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



## Outline

### 1. Initialize strings

### 2. Print strings

#### 2.1 Define loop

#### 2.2 Print characters individually

#### 2.3 Input string

### 3. Print string

## Program Output

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

Notice how string elements are referenced like arrays.

```
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 ], string2[] = "string literal";
12
13     cout << "Enter a string: ";
14     cin >> string1;
15     cout << "string1 is: " << string1
16         << "\nstring2 is: " << string2
17         << "\nstring1 with spaces between characters is:\n";
18
19     for ( int i = 0; string1[ i ] != '\0'; i++ )
20         cout << string1[ i ] << ' ';
21
22     cin >> string1; // reads "there"
23     cout << "\nstring1 is: " << string1 << endl;
24
25     cout << endl;
26     return 0;
27 }
```

```
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
```

## 8. 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

## 8. 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`





## Outline



### 1. Define function to take in arrays

#### 1.1 Initialize arrays

### 2. Modify the array (call by reference)

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
```

```
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;  
18     int i, a[ arraySize ] = { 0, 1, 2, 3, 4 };  
19  
20     cout << "Effects of passing entire array call-by-reference:"  
21         << "\n\nThe values of the original array are:\n";  
22  
23     for ( i = 0; i < arraySize; i++ )  
24         cout << setw( 3 ) << a[ i ];  
25  
26     cout << endl;  
27  
28     // array a passed call-by-reference  
29     modifyArray( a, arraySize );  
30  
31     cout << "The values of the modified array are:\n";
```



## Outline



### 2.1 Modify an element (call by value)

### 3. Print changes.

### 3.1 Function Definitions

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

```
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 copy of
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 }
```



## Outline

### 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

# 9. 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

# 9. 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

# 10. 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)



## Outline



### 1. Function prototypes

#### 1.1 Initialize array

### 2. Call functions mean, median, and 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::ios;
9
10 #include <iomanip>
11
12 using std::setw;
13 using std::setiosflags;
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;
25     int frequency[ 10 ] = { 0 },
26         response[ responseSize ] =
27         { 6, 7, 8, 9, 8, 7, 8, 9, 8, 9,
28           7, 8, 9, 5, 9, 8, 7, 8, 7, 8,
29           6, 7, 8, 9, 3, 9, 8, 7, 8, 7,
30           7, 8, 9, 8, 9, 8, 9, 7, 8, 9,
31           6, 7, 8, 7, 8, 7, 9, 8, 9, 2,
32           7, 8, 9, 8, 9, 8, 9, 7, 5, 3,
33           5, 6, 7, 2, 5, 3, 9, 4, 6, 4,
```



### 3. Define function mean

#### 3.1 Define function median

```
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"
```





## 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 []

```
68     << "The unsorted array of responses is";
69
70     printArray( answer, size );
71     bubbleSort( answer, size );
72     cout << "\n\nThe sorted array is";
73     printArray( answer, size );
74     cout << "\n\nThe median is element " << size / 2
75         << " of\nthe sorted " << size
76         << " element array.\nFor this run the median is "
77         << answer[ size / 2 ] << "\n\n";
78 }
79
80 void mode( int freq[], int answer[], int size )
81 {
82     int rating, largest = 0, modeValue = 0;
83
84     cout << "\n*****\n  Mode\n*****\n";
85
86     for ( rating = 1; rating <= 9; rating++ )
87         freq[ rating ] = 0;
88
89     for ( int j = 0; j < size; j++ )
90         ++freq[ answer[ j ] ];
91
92     cout << "Response"<< setw( 11 ) << "Frequency"
93         << setw( 19 ) << "Histogram\n\n" << setw( 55 )
94         << "1    1    2    2\n" << setw( 56 )
95         << "5    0    5    0    5\n\n";
```

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



```
96
97 for ( rating = 1; rating <= 9; rating++ ) {
98     cout << setw( 8 ) << rating << setw( 11 )
99         << freq[ rating ] << "          ";
100
101     if ( freq[ rating ] > largest ) {
102         largest = freq[ rating ];
103         modeValue = rating;
104     }
105
106     for ( int h = 1; h <= freq[ rating ]; h++ )
107         cout << '*';
108
109     cout << '\n';
110 }
111
112 cout << "The mode is the most frequent value.\n"
113     << "For this run the mode is " << modeValue
114     << " which occurred " << largest << " times." << endl;
115 }
116
117 void bubbleSort( int a[], int size )
118 {
119     int hold;
```

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



## Outline



### 3.3 Define bubbleSort

### 3.3 Define printArray

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

```
121 for ( int pass = 1; pass < size; pass++ )
122
123     for ( int j = 0; j < size - 1; j++ )
124
125         if ( a[ j ] > a[ j + 1 ] ) {
126             hold = a[ j ];
127             a[ j ] = a[ j + 1 ];
128             a[ j + 1 ] = hold;
129         }
130 }
131
132 void printArray( const int a[], int size )
133 {
134     for ( int j = 0; j < size; j++ ) {
135
136         if ( j % 20 == 0 )
137             cout << endl;
138
139         cout << setw( 2 ) << a[ j ];
140     }
141 }
```



## 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 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



## 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.

```

# 10. 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  $\log_2 n$  steps, where  $2^{\log_2 n} > \#$  of elements
      - 30 element array takes at most 5 steps
- $2^5 > 30$



## Outline



### 1. Initialize variables

#### 1.1 Define functions to take double scripted arrays

#### 1.2 Initialize studentgrades [ ] [ ]

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

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

```
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::ios;
8
9 #include <iomanip>
10
11 using std::setw;
12 using std::setiosflags;
13 using std::setprecision;
14
15 const int students = 3; // number of students
16 const int exams = 4; // number of exams
17
18 int minimum( int [][] exams , int, int );
19 int maximum( int [][] exams , int, int );
20 double average( int [], int );
21 void printArray( int [][] exams , int, int );
22
23 int main()
24 {
25     int studentGrades[ students ][ exams ] =
26         { { 77, 68, 86, 73 },
27           { 96, 87, 89, 78 },
28           { 70, 90, 86, 81 } };
29
30     cout << "The array is:\n";
31     printArray( studentGrades, students, exams );
32     cout << "\n\nLowest grade: "
33         << minimum( studentGrades, students, exams )
```



## Outline



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

**3. Define functions**

```
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++ )
```





### 3. Define functions

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

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

**Program Output**