

# CISC181 Introduction to Computer Science

Dr. McCoy

Lecture 12  
October 8, 2009

## 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



## 4.9 Multiple-Subscripted Arrays

- To initialize
  - Default of 0
  - Initializers grouped by row in braces

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



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



## 4.9 Multiple-Subscripted Arrays

- Referenced like normal
  - `cout << b[ 0 ][ 1 ];`
  - 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 ] );`



## NOTE PROBLEM WITH NEXT FUNCTION

- Normally when an array is passed as an argument to a function, you also pass the number of elements in that array.
- When an array with multiple subscripts is passed to a function – it is passed as an array (giving no "number" of elements) of arrays (whose size it given).
- But, in the following function the max index is left off (the function only works for one size array).

```
1 // fig. 4.20: fig04_20.cpp
2 // initializing multidimensional arrays.
3 #include <iostream>
4 using namespace std;
5 using namespace std;
6 void printArray( int [][ 3 ] );
7
8 int main()
9 {
10     int array1[ 2 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 } };
11     int array2[ 2 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 } };
12     int array3[ 2 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 } };
13
14     cout << "values in array1 by row are: " << endl;
15     printArray( array1 );
16
17     cout << "values in array2 by row are: " << endl;
18     printArray( array2 );
19
20     cout << "values in array3 by row are: " << endl;
21     printArray( array3 );
22
23     return 0; // indicates successful termination
24 } // 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.

Outline  
fig04\_20.cpp  
(1 of 2)

© 2001 Pearson Education, Inc. All rights reserved.

```

25 // function to output array with two rows
26 void printarray( int a[2][3] )
27 {
28     for ( int i = 0; i < 2; i++ ) //
29     {
30         for ( int j = 0; j < 3; j++ ) // output column values
31             cout << a[i][j] << " ";
32     }
33     cout << endl; // output new line of output
34 } // end function printarray
35
36 int main()
37 {
38     // initialize array by row
39     int a[2][3] = { { 1, 2, 3 }, { 4, 5, 6 } };
40     // output array by row
41     printarray( a );
42     // output array by column
43     printarray( a );
44     // output array by row
45     printarray( a );
46     // output array by column
47     printarray( a );
48 } // end function main

```

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

Output:  
 values in array by row are:  
 1 2 3  
 4 5 6  
 values in array by row are:  
 1 2 3  
 4 5 6  
 values in array by row are:  
 1 2 3  
 4 5 6

### 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	82	81
Student1	88	85

## NOTE POOR PROGRAMMING STYLE

- Lack of documentation with function prototypes.
- Lack of variable names in function prototypes.
- NOTE: Can't at all tell what function does by looking at the prototype. That is WRONG.

```

1 // fig. 4.20: fig4_20.cpp
2 // multiple-subscripted array example.
3 #include <iostream>
4
5 using namespace
6 std;
7 using namespace
8 std;
9 using namespace
10 std;
11
12 #include <iostream>
13 using namespace
14 std;
15 using namespace
16 std;
17
18 const int students = 10; // number of students
19 const int exams = 10; // number of exams
20
21 // function prototypes
22 int minmax( int [][2], int, int );
23 int minmax( int [][2], int, int );
24 double average( int [][2], int );
25 void printarray( int [][2], int, int );
26

```

```

26 int main()
27 {
28     // initialize student grades for three students (rows)
29     int studentGrades[3][10] = {
30         { 77, 88, 84, 76 },
31         { 84, 85, 80, 78 },
32         { 79, 86, 84, 81 } };
33
34     // output array studentGrades
35     cout << "the array looks:\n";
36     printarray( studentGrades, students, exams );
37
38     // determine min/max and average grade values
39     int minmax( int [][2], int, int );
40     int minmax( int [][2], int, int );
41     int minmax( int [][2], int, int );
42     int minmax( int [][2], int, int );
43
44     cout << "the array organization:\n";
45 }

```

```

46 // calculate average grade for each student
47 for ( int person = 0; person < students; person++ )
48     cout << "the average grade for student " << person
49         << " is:\n";
50     double average( studentGrades[ person ], exams );
51     cout << endl;
52
53 return 0; // indicates successful program
54 } // end main
55
56 // find min/max grade
57 int minmax( int grades[2][ exams ], int person )
58 {
59     int lowestGrade = 100; // initialize to highest possible grade
60
61     for ( int i = 0; i < exams; i++ )
62     {
63         for ( int j = 0; j < exams; j++ )
64             if ( grades[ person ][ j ] < lowestGrade )
65                 lowestGrade = grades[ person ][ j ];
66     }
67
68     return lowestGrade;
69 } // end function minmax

```

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

```

99 // find maximum grade
100 void find_max_grade( int grades[ ][ ], int num_s, int num_g, int num_c )
101 {
102     int highest_grade = 0; // initialize to lowest possible grade
103     for ( int k = 0; k < num_g; k++ )
104     {
105         for ( int j = 0; j < num_c; j++ )
106         {
107             if ( grades[ k ][ j ] > highest_grade )
108                 highest_grade = grades[ k ][ j ];
109         }
110     }
111     return highest_grade;
112 } // end function find_max_grade
113
114 } // end function main
115

```

9g04\_13.cpp (4 of 6)

```

97 // determine average grade for particular student
98 double average( int student_id[ ], int num_s )
99 {
100     int total = 0;
101
102     // sum all grades for one student
103     for ( int k = 0; k < num_s; k++ )
104         total += student_id[ k ];
105
106     return static_cast<double>( total ) / num_s; // average
107 } // end function average
108

```

9g04\_13.cpp (5 of 6)

```

99 // return the array
100 void print_array( int grades[ ][ ], int num_s, int num_g, int num_c )
101 {
102     // use C++ for initialization and output column heads
103     cout << endl << "   ";
104     for ( int k = 0; k < num_g; k++ )
105         cout << "  " << k << "  ";
106
107     // output grades in tabular format
108     for ( int k = 0; k < num_g; k++ )
109     {
110         // output label for row
111         cout << "Student " << k << " : ";
112
113         // output one grade for one student
114         for ( int j = 0; j < num_c; j++ )
115             cout << " " << grades[ k ][ j ] << " ";
116     }
117 } // end function print_array
118

```

9g04\_13.cpp (6 of 6)

```

the array is:
student_id[0] 77 88 88 78
student_id[1] 88 87 88 78
student_id[2] 75 88 88 88

lowest grade: 66
highest grade: 88
the average grade for student 0 is 74.00
the average grade for student 1 is 87.00
the average grade for student 2 is 84.75

```

9g04\_13.cpp output(3 of 3)

### 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

9g04\_13.cpp

### 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
      - $2^5 > 30$

9g04\_13.cpp