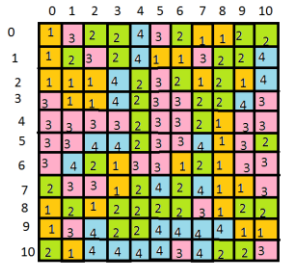
ensia The National School of Artificial Intelligence
المدرسة الوطنية العليا للذكاء الاصطناعي



Data Structure & Algorithms 1

CHAPTER 4
STATIC DATA STRUCTURE (PART2): 2D ARRAY

Sep – Dec 2023

Outline

- Introduction
  - Definition
  - Example
  - Declaration
- Fundamental Modules for 2D Array Operations
  - Read, Write
  - Applications
    - Arrangement
- 2D Array in C++
  - Definition and Declaration
  - Examples

Introduction to 2D Array

Definition

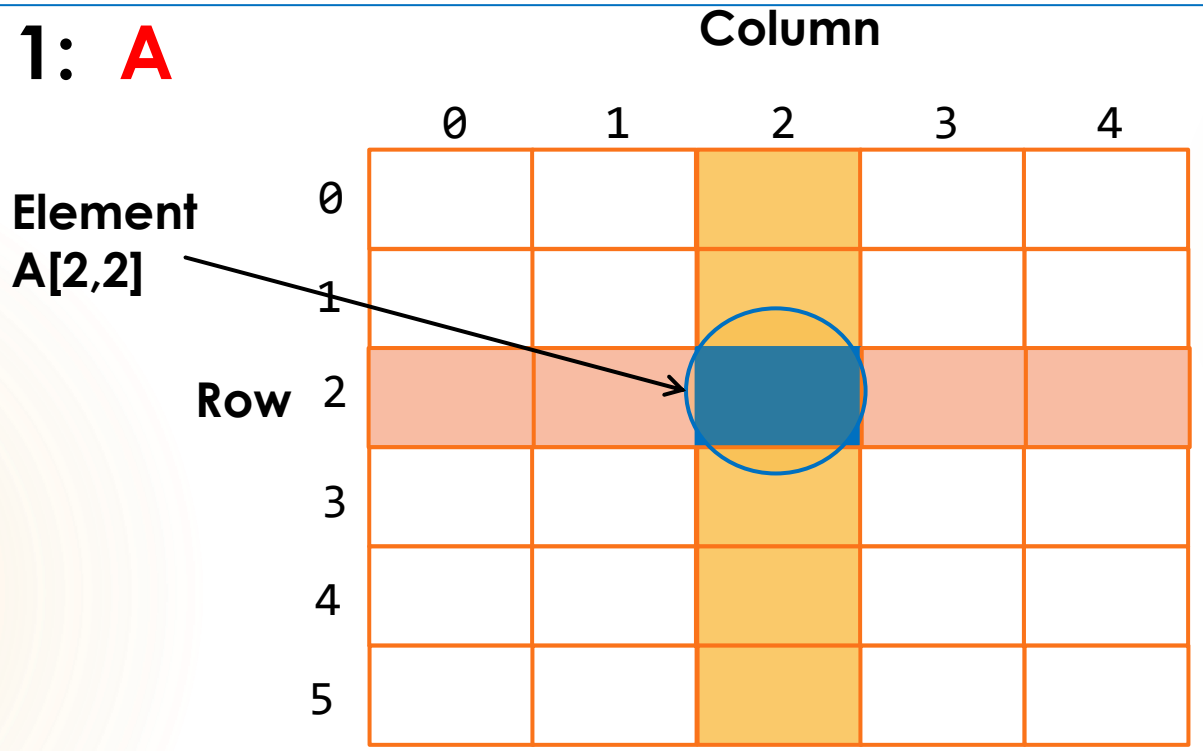
- **2D ARRAY:** is an object composed of multiple elements of the same type (row), each element, in turn, decomposed into several elements of the same type in columns.

- Accessing an element of the array is done by specifying the object's name followed, in **square brackets** `[r , c]`, by the row index and column index separated by a comma

Introduction to 2D Array

Examples

- **Example 1: A**



- A: is an array of 6 rows and 5 columns

Introduction to 2D Array

Examples

- `A = T[2,3]`
- `B = T[i,j]`
- `C = T[x + k, z DIV 2]`
- `T[2,3] = k`
- `T[I,j] = 13`
- `T[i DIV 2, j DIV x] = M + n`

Declaration of 2D Array

- **Remark**: The number of rows and columns in the array constitutes the size of the array.
  - It is crucial not to confuse between the maximum size used during the declaration of the array and the actual size of the array.
- The declaration is practically identical to that of one-dimensional arrays.

Declaration of 2D Array

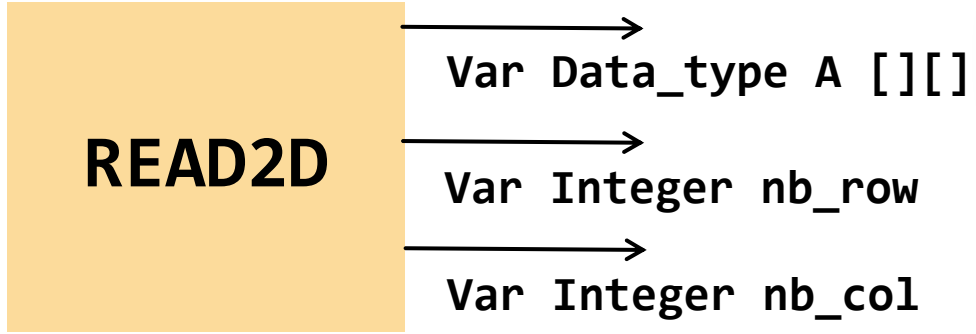
- **FORMAT:**

```
Element_type  ArrayName [rowCount] [columnCount]
```

- **Example:**
  - `Char ticTacToe [3][3]` *// Represents a Tic-Tac-Toe game board (X O)*
  - `Integer pixels [1024][768]` *// Represents a basic grayscale image with 1024x768 resolution*
  - `Integer grades [5][3]` *// Represents grades of 5 students in 3 modules*
  - `Boolean attendance[30][5];` *// Represents the attendance for a class with 30 students and 5 days*

Basic Modules for 2D Array

PROCEDURE



Role: read a 2D array with number of rows and number of columns

Basic Modules for 2D Array

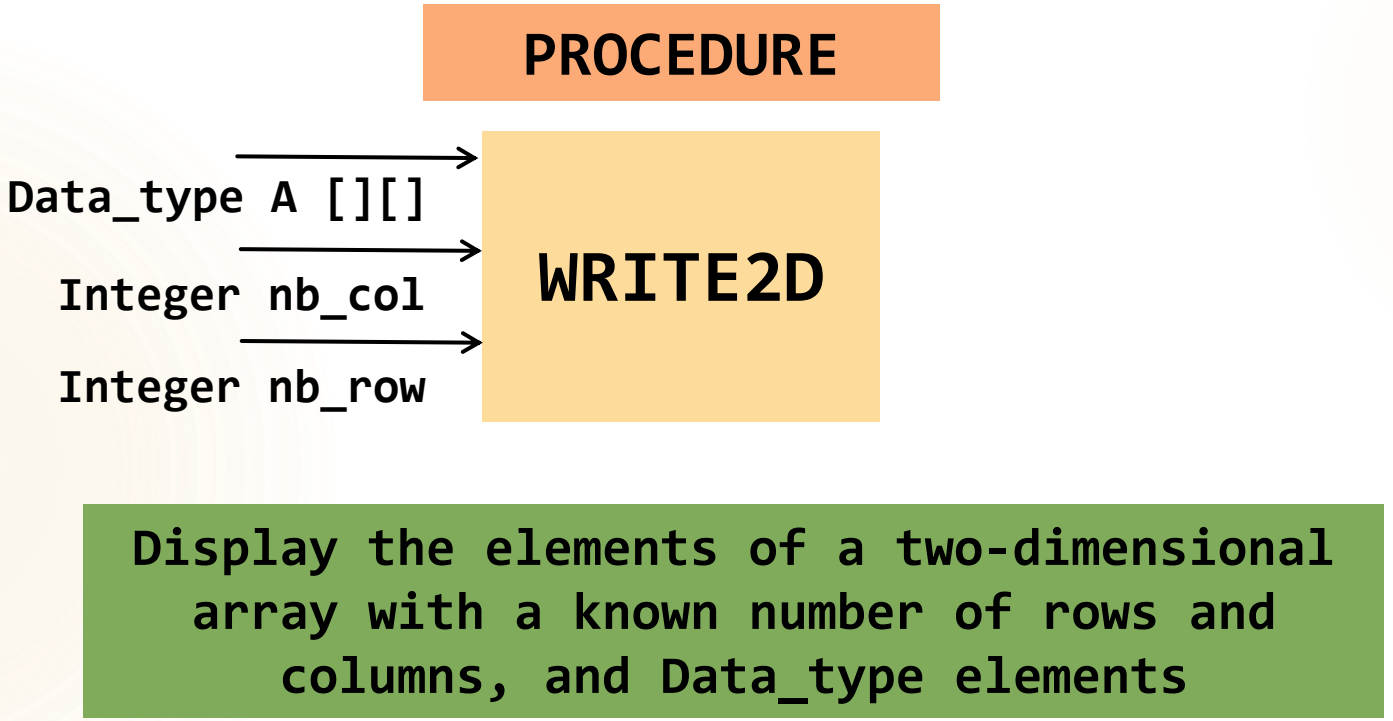
ANALYSIS

- The number of rows (nb_row) of the array is read.
- The number of columns (nb_col) of the array is read.

- We vary i = 0, 1, 2, .., nb_row-1, and in each iteration:
  - We vary j = 0, 1, 2, .., nb_col-1 and in each iteration:
    - We read the element A[i,j] of the array.

```
Procedure READ2D (Var Data_type A[][], Var Integer
nb_row, Var Integer nb_col)
Variable Integer i, j
BEGIN
    WRITE ('Enter the number of rows')
    READ (nb_row)
    WRITE ('Enter the number of columns')
    READ (nb_col)
    FOR i FROM 0 TO nb_row-1 DO
        FOR j FROM 0 TO nb_col-1 DO
            WRITE ('A[', i, ',', j, '] = ')
            READ (A[i,j])
        END FOR
    END FOR
END
```

Basic Modules for 2D Array



Basic Modules for 2D Array

ANALYSIS

- We vary i = 0, 1, 2, .., nb_row-1, and in each iteration:
  - We vary j = 0, 1, 2, .., nb_col-1, and in each iteration:
    - We write the element A[i,j] of the array.

```
Procedure WRITE2D (Data_type A[], Integer
nb_row, Integer nb_col)
Variable Integer i, j
BEGIN
    FOR i FROM 0 TO nb_row-1 DO
        FOR j FROM 0 TO nb_col-1 DO
            WRITE (A[i,j], ' |')
        END FOR
        WRITE ('\n')
    END FOR
END
```

Basic Modules for 2D Array

Application 1: Searching for an element in a 2D array

Provide the solution that allows searching for a given value **V** in an array of a maximum of 10 row and 10 columns of integers

Modular Decomposition:

- We need a module SearchV2D
- Basic modules:
  - READ2D and WRITE2D

Module SearchV2D
Analysis:

- Initialization of a variable Found to false;
- Initialization of a variable i;
- While (i < nb_row) AND Found = False)
  - Initialization of a variable j;
  - While (j < nb_col) AND Found = False)
  - Compare an element A[I,j] with the value V
    - If A[I,j] equals V, set Found to True
- Assignment: **SearchV2D** = Found

Basic Modules for 2D Array

Application 1: Searching for an element in a 2D array

```
Boolean Function SearchV2D (Data_type A[][], Integer nb_row, Intger nb_col,
Data_type V)
Variable Integer i, j
          Boolean Found
BEGIN
    Found = False
    i = 0
    WHILE (i < nb_row) AND (Found == False) DO
          j = 0
          WHILE (j < nb_col) AND (Found == False) DO
              IF A[i,j] == V THEN
                   Found = True
               END IF
               j = j + 1
          END WHILE
          i = i + 1
    END WHILE
    SeachV2D = Found
END
```

Basic Modules for 2 D Array

Application 2: Sum of Columns

- Given a 2D array A, store the cumulative sums of the elements in each column of the same column in a 1D array SCOL.

A

4	-5	3	6	0	1
3	2	5	-1	2	4
1	4	-3	0	3	2
2	4	1	1	9	0
1	3	0	4	5	7

SCOL

11	8	6	10	20	14

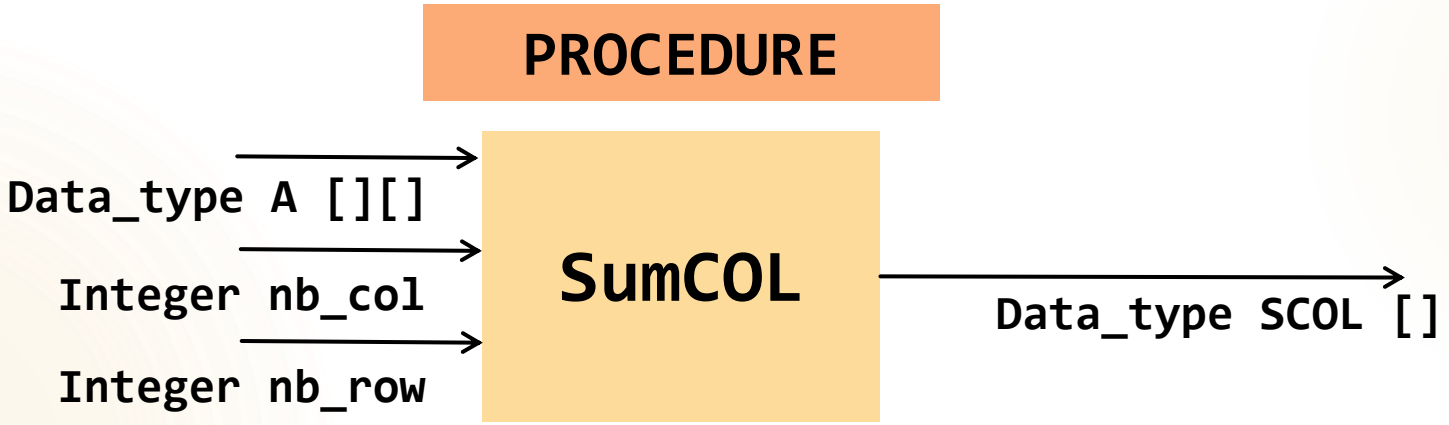
Basic Modules for 2D Array

Application 2: Sum of Columns

Modular breakdown:

- The module READ2D, which allows us to read a 2D array.
- The module SumCOL, which calculates the sum of a column and stores it in a cell of a 1D array.
- A module that displays the array containing the results of the sum of each column, WRITE1D.
- An optional module, WRITE2D, which displays the array containing the elements of the initial 2D array.

Basic Modules for 2D Array



Calculates the cumulative sum for each column, considering all its elements, and stores it in a 1D array R.

Basic Modules for 2D Array

Application 2: Sum of Columns

Construction of the SumCOL Module:

Analysis:

- For each column j,
  - • We initialize sumC to 0 (which will contain the sum of the elements in the column)
  - • For each row i
    - □ We accumulate the elements A[i, j] into sumC
- • We store the cumulative sum (sumC) in the SCOL[j]

Basic Modules for 2D Array

Application 2: Sum of Columns

```
Procedure SumCOL (Data_type A[][], Integer nb_row, Intger nb_col, Var Data_type SCOL[])
Variable Integer i, j, sumC
BEGIN
    FOR j FROM 0 TO nb_col-1 DO
        sumC = 0
        FOR i FROM 0 TO nb_row-1 DO
            sumC = sumC + A[i,j]
        END FOR
        SCOL[j] = sumC
    END FOR
END
```

Basic Modules for 2D Array

Application 2: Sum of Columns

Main Algorithm

Analysis:

- The procedure READ2D is called to read the size of the array and its elements.
- WRITE2D is called for verification (optional).
- The SumCOL procedure is called to perform the sums (cumulative sums) of the columns and place them in a one-dimensional array.
- WRITE1D is called to display the contents of the SCOL array.

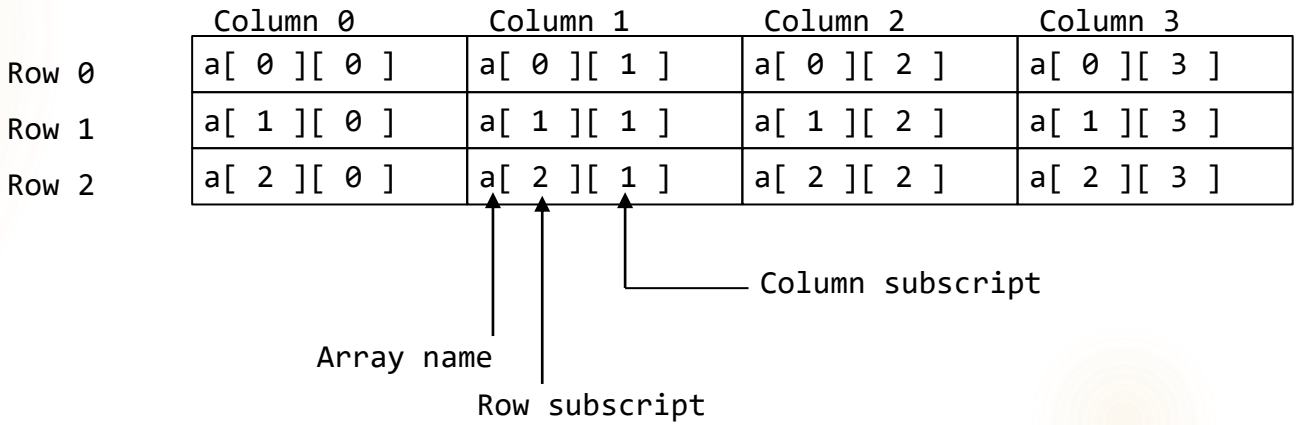
Basic Modules for 2D Array

Application 2: Sum of Columns

```
Algorithm Application_2

Constant MaxR = 10, MaxC = 8
Variable Integer A[MaxR][MaxC], nb_row, nb_col, R[MaxC]

Procedure READ2D WRITE2D, SumCOL,WRITE1D
        …. Body ….

BEGIN
   READ2D(A, nb_row, nb_col)
   SumCOL(A, nb_row, nb_col, R)
   WRITE1D(R, nb_col)
END
```

2D Array in C++

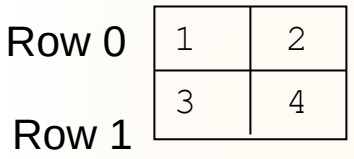
- Multiple subscripts
  - **`a[ i ][ j ]`**
  - Tables with rows and columns
  - Specify row, then column
  - "Array of arrays"
    - **`a[0]`** is an array of 4 elements
    - **`a[0][0]`** is the first element of that array



2D Array in C++

- 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 } };
```

1	0
3	4

2D Array in C++

- 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 ] );`**

```cpp
// Fig. 4.22: fig04_22.cpp
// Initializing multidimensional arrays.
#include <iostream>

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

void printArray( int [][ 3 ] );

int main()
{
   int array1[ 2 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 } };
   int array2[ 2 ][ 3 ] = { 1, 2, 3, 4, 5 };
   int array3[ 2 ][ 3 ] = { { 1, 2 }, { 4 } };

   cout << "Values in array1 by row are:" << endl;
   printArray( array1 );

   cout << "Values in array2 by row are:" << endl;
   printArray( array2 );

   cout << "Values in array3 by row are:" << endl;
   printArray( array3 );

   return 0;  // indicates successful termination

} // 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.

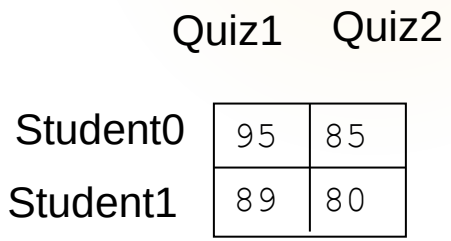
```
28
29 // function to output array with two rows
30 void printArray( int a[][ 3 ] )
31 {
32    for ( int i = 0; i < 2; i++ ) {    // f
33
34       for ( int j = 0; j < 3; j++ )   // output column values
35          cout << a[ i ][ j ] << ' ';
36
37       cout << endl;  // start new line of output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

```
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
```

2D Array in C++

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



```
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
```

```
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         << "\nHighest grade: "
40         << maximum( studentGrades, students, exams ) << '\n';
41
42    cout << fixed << setprecision( 2 );
43
```

```
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 pupi
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
```

```
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
```

```
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
```