

# Arrays & Vectors

Week 5

Sayatbek Orazbekov

[s.orazbekov@astanait.edu.kz](mailto:s.orazbekov@astanait.edu.kz)

# 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> |

Diagram illustrating the structure of a multiple-subscripted array. The array is represented as a table with rows and columns. The first subscript (row) is labeled "Row subscript" and the second subscript (column) is labeled "Column subscript". The array name "a" is indicated by an arrow pointing to the first subscript.

# 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 |

Row 0

Row 1

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

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

# Multiple-Subscripted Arrays

- Referenced like normal

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

- Outputs 0

- Cannot reference using commas

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

- Syntax error

|   |   |
|---|---|
| 1 | 0 |
| 3 | 5 |

- 1 // Fig. 3.22: 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.

- 28
- 29 *// function to output array with two rows and three columns*
- 30 `void printArray( int a[][ 3 ] )`
- 31 `{`
- 32 `for ( int i = 0; i < 2; i++ ) {` *// for each row*
- 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

# 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 | 1     | 85    |
| Student1 | 89    | 80    |

- 1 // Fig. 3.23: 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 `<< "\n\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 termination
• 52
• 53 } // end main
• 54
• 55 // find minimum grade
• 56 int minimum( int grades[][ exams ], int pupils, int tests )
• 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

*“Well, I’ll eat it,” said Alice, “and if it makes me grow larger, I can reach the key; and if it makes me grow smaller, I can creep under the door; so either way I’ll get into the garden.”*

*Lewis Carroll, Alice’s Adventures in Wonderland*

# VECTORS

## Array

- cannot change the length

## Vector

- the same purpose as arrays
  - ✓ except can change length while the program is running

Like an array, a vector has a base type, and like an array, a vector stores a collection of values of its base type.



# Vectors

Library:

```
#include <vector>
```

Declaration:

```
vector <base_type> name;
```

Example:

```
vector <int> v;
```

```
vector <int> v(10);
```

# Vectors

- ❑ To add an element to a vector for the first time, you normally use the member function ***push\_back***.

Example:

```
int main() {  
  
    vector<double> sample;  
    sample[0]=1;  
    sample.push_back(0.0);  
    sample.push_back(1.1);  
    sample.push_back(2.2);  
  
}
```

# Vectors

- The number of elements in a vector is called the **size of the vector**.
- The member function **size** can be used to determine how many elements are in a vector.

Example:

```
for (int i = 0; i < sample.size( ); i++){  
    cout << sample[i] << endl;  
}
```

// Demonstrating C++ Standard Library class template vector.

```
#include <iostream>
using std::cout;
using std::cin;
using std::endl;

#include <iomanip>
using std::setw;

#include <vector>
using std::vector;

void outputVector( const vector< int > & ); // display the vector
void inputVector( vector< int > & ); // input values into the vector

int main()
{
    vector< int > integers1( 7 ); // 7-element vector< int >
    vector< int > integers2( 10 ); // 10-element vector< int >
```

```

// print integers1 size and contents
cout << "Size of vector integers1 is " << integers1.size()
      << "\nvector after initialization:" << endl;
outputVector( integers1 );

// print integers2 size and contents
cout << "\nSize of vector integers2 is " << integers2.size()
      << "\nvector after initialization:" << endl;
outputVector( integers2 );

// input and print integers1 and integers2
cout << "\nEnter 17 integers:" << endl;
inputVector( integers1 );
inputVector( integers2 );

cout << "\nAfter input, the vectors contain:\n"
      << "integers1:" << endl;
outputVector( integers1 );
cout << "integers2:" << endl;
outputVector( integers2 );

```

```

// use inequality (!=) operator with vector objects
cout << "\nEvaluating: integers1 != integers2" << endl;

if ( integers1 != integers2 )
    cout << "integers1 and integers2 are not equal" << endl;

// create vector integers3 using integers1 as an
// initializer; print size and contents
vector< int > integers3( integers1 ); // copy constructor

cout << "\nSize of vector integers3 is " << integers3.size()
    << "\nvector after initialization:" << endl;
outputVector( integers3 );

// use overloaded assignment (=) operator
cout << "\nAssigning integers2 to integers1:" << endl;
integers1 = integers2; // integers1 is larger than integers2

cout << "integers1:" << endl;
outputVector( integers1 );
cout << "integers2:" << endl;
outputVector( integers2 );

```



```

// use equality (==) operator with vector objects
cout << "\nEvaluating: integers1 == integers2" << endl;

if ( integers1 == integers2 )
    cout << "integers1 and integers2 are equal" << endl;
// use square brackets to create rvalue
cout << "\nintegers1[5] is " << integers1[ 5 ];

// use square brackets to create lvalue
cout << "\n\nAssigning 1000 to integers1[5]" << endl;
integers1[ 5 ] = 1000;
cout << "integers1:" << endl;
outputVector( integers1 );

// attempt to use out-of-range subscript
cout << "\nAttempt to assign 1000 to integers1.at( 15 )" << endl;
integers1.at( 15 ) = 1000; // ERROR: out of range
return 0;
} // end main

```

```

// output vector contents
void outputVector( const vector< int > &array )
{
    size_t i; // declare control variable
    for ( i = 0; i < array.size(); i++ )
    {
        cout << setw( 12 ) << array[ i ];
        if ( ( i + 1 ) % 4 == 0 ) // 4 numbers per row of output
            cout << endl;
    } // end for

    if ( i % 4 != 0 )
        cout << endl;
} // end function outputVector

// input vector contents
void inputVector( vector< int > &array )
{
    for ( size_t i = 0; i < array.size(); i++ )
        cin >> array[ i ];
} // end function inputVector

```



# Two / Three / Multi Dimensioned arrays using vector

- A two dimensional array is a vector of vectors.
- The vector constructor can initialize the length of the array and set the initial value.
- Example of a vector of vectors to represent a two dimensional array:
  - `vector< vector<int> > vI2Matrix(3, vector<int>(2,0));`

```

#include <iostream>
#include <vector>
using namespace std;
int main() {
    // Declare size of two dimensional array and initialize.
    vector< vector<int> > vI2Matrix(3, vector<int>(2,0));
    vI2Matrix[0][0] = 0;
    vI2Matrix[0][1] = 1;
    vI2Matrix[1][0] = 10;
    vI2Matrix[1][1] = 11;
    vI2Matrix[2][0] = 20;
    vI2Matrix[2][1] = 21;
    cout << "Loop by index:" << endl;
    int ii, jj;
    for(ii=0; ii < 3; ii++) {
        for(jj=0; jj < 2; jj++) {
            cout << vI2Matrix[ii][jj] << endl;
        }
    }
    return 0;
}

```

Loop by index:

0  
1  
10  
11  
20  
21

# Two / Three / Multi Dimensioned arrays using vector

- A three dimensional vector would be declared as:

```
#include <iostream>
#include <vector>
using namespace std;
void main() {
    // Vector length of 3 initialized to 0
    vector<int> vI1Matrix(3,0);
    // Vector length of 4 initialized to hold another
    // vector vI1Matrix which has been initialized to 0
    vector< vector<int> > vI2Matrix(4, vI1Matrix);
```

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<vector<vector<int>>> vI3Matrix(2, vector<vector<int>>(3, vector<int>(4,0)));
    for(int kk=0; kk<4; kk++) {
        for(int jj=0; jj<3; jj++) {
            for(int ii=0; ii<2; ii++) {
                cout << vI3Matrix[ii][jj][kk] << endl;
            }
        }
    }
    return 0;
}
```

# Quiz

- 1. What is vector?
  - Benefits of vector over array.
  - Difference between array and vector.
  - Which one is better for what purpose?
- 2. Write a program that create a 2-dimensional vector *vec1*.
  - initialize all elements of a vector to 1.
  - then output content of the vector *vec1*.

# Readings:

- **C++ How to Program, By H. M. Deitel**
  - Chapter 7. Arrays and Vectors