

Chapter 8 Multidimensional Arrays



Quiz

```
#include <iostream>
using namespace std;
int Quiz(int num)
{
    int result = 1;
    for(int i =0; i<= num; i++)
    {
        result = result * num;
        cout<<"Step "<<i<<": "<<result<<endl;
    }

    return result;
}

int main()
{
    int x = Quiz(3);
    cout<<"Final Result is: "<<x<<endl;
    return 0;
}
```



Objectives

- ❑ To give examples of representing data using two-dimensional arrays (§8.1).
- ❑ To declare two-dimensional arrays and access array elements in a two-dimensional array using row and column indexes (§8.2).
- ❑ To process two-dimensional arrays (§8.3).
- ❑ To pass two-dimensional arrays to functions (§8.4).
- ❑ To write a program for grading multiple-choice questions using two-dimensional arrays (§8.5).
- ❑ To solve the closest-pair problem using two-dimensional arrays (§8.6).
- ❑ To solve the Sudoku problem using two-dimensional arrays (§8.7).
- ❑ To declare multidimensional arrays (§8.8).



Two-dimensional Arrays

```
// Declare array ref var
```

```
elementType arrayName[rowSize][columnSize];
```

```
int matrix[5][5];
```



Two-dimensional Array Illustration

	[0]	[1]	[2]	[3]	[4]
[0]					
[1]					
[2]					
[3]					
[4]					

```
int matrix[5][5];
```

	[0]	[1]	[2]	[3]	[4]
[0]					
[1]					
[2]		7			
[3]					
[4]					

```
matrix[2][1] = 7;
```

	[0]	[1]	[2]
[0]	1	2	3
[1]	4	5	6
[2]	7	8	9
[3]	10	11	12

```
int array[][3] =  
    {1, 2, 3},  
    {4, 5, 6},  
    {7, 8, 9},  
    {10, 11, 12}  
};
```



Declaring, Creating, and Initializing Using Shorthand Notations

You can also use an array initializer to declare, create and initialize a two-dimensional array. For example,

```
int array[4][3] =  
{  
    {1, 2, 3},  
    {4, 5, 6},  
    {7, 8, 9},  
    {10, 11, 12}  
};
```

(a)

Equivalent

```
int array[4][3];  
array[0][0] = 1; array[0][1] = 2; array[0][2] = 3;  
array[1][0] = 4; array[1][1] = 5; array[1][2] = 6;  
array[2][0] = 7; array[2][1] = 8; array[2][2] = 9;  
array[3][0] = 10; array[3][1] = 11; array[3][2] = 12;
```

(b)

Initializing Arrays with Random Values

The following loop initializes the array with random values between 0 and 99:

```
for (int row = 0; row < rowSize; row++)  
{  
    for (int column = 0; column < columnSize; column++)  
    {  
        matrix[row][column] = rand() % 100;  
    }  
}
```



Printing Arrays

To print a two-dimensional array, you have to print each element in the array using a loop like the following:

```
for (int row = 0; row < rowSize; row++)  
{  
    for (int column = 0; column < columnSize; column++)  
    {  
        cout << matrix[row][column] << " ";  
    }  
    cout << endl;  
}
```



Summing All Elements

To print a two-dimensional array, you have to print each element in the array using a loop like the following:

```
for (int row = 0; row < rowSize; row++)  
{  
    for (int column = 0; column < columnSize; column++)  
    {  
        cout << matrix[row][column] << " ";  
    }  
    cout << endl;  
}
```



Summing Elements by Column

For each column, use a variable named total to store its sum. Add each element in the column to total using a loop like this:

```
for (int column = 0; column < columnSize; column++)  
{  
    int total = 0;  
    for (int row = 0; row < rowSize; row++)  
        total += matrix[row][column];  
    cout << "Sum for column " << column << " is " << total  
        << endl;  
}
```



Which row has the largest sum?

Use variables maxRow and indexOfMaxRow to track the largest sum and index of the row. For each row, compute its sum and update maxRow and indexOfMaxRow if the new sum is greater.



Passing Two-Dimensional Arrays to Functions

You can pass a two-dimensional array to a function; however, C++ requires that the column size to be specified in the function declaration. Listing 8.1 gives an example with a function that returns the sum of all the elements in a matrix.

PassTwoDimensionalArray

PassTwoDimensionalArray

```

#include <iostream>
using namespace std;

const int COLUMN_SIZE = 4;

int sum(const int a[][COLUMN_SIZE], int rowSize)
{
    int total = 0;
    for (int row = 0; row < rowSize; row++)
    {
        for (int column = 0; column < COLUMN_SIZE; column++)
        {
            total += a[row][column];
        }
    }

    return total;
}

int main()
{
    const int ROW_SIZE = 3;
    int m[ROW_SIZE][COLUMN_SIZE];
    cout << "Enter " << ROW_SIZE << " rows and "
        << COLUMN_SIZE << " columns: " << endl;
    for (int i = 0; i < ROW_SIZE; i++)
        for (int j = 0; j < COLUMN_SIZE; j++)
            cin >> m[i][j];

    cout << "\nSum of all elements is " << sum(m, ROW_SIZE) << endl;

    return 0;
}

```



Example: Grading Multiple-Choice Test

Students' Answers to the Questions:

0 1 2 3 4 5 6 7 8 9

Student 0	A	B	A	C	C	D	E	E	A	D
Student 1	D	B	A	B	C	A	E	E	A	D
Student 2	E	D	D	A	C	B	E	E	A	D
Student 3	C	B	A	E	D	C	E	E	A	D
Student 4	A	B	D	C	C	D	E	E	A	D
Student 5	B	B	E	C	C	D	E	E	A	D
Student 6	B	B	A	C	C	D	E	E	A	D
Student 7	E	B	E	C	C	D	E	E	A	D

Key

Key to the Questions:

0 1 2 3 4 5 6 7 8 9

D B D C C D A E A D

GradeExam

GradeExam

```

#include <iostream>
using namespace std;

int main()
{
    const int NUMBER_OF_STUDENTS = 8;
    const int NUMBER_OF_QUESTIONS = 10;

    // Students' answers to the questions
    char answers[NUMBER_OF_STUDENTS][NUMBER_OF_QUESTIONS] =
    {
        {'A', 'B', 'A', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
        {'D', 'B', 'A', 'B', 'C', 'A', 'E', 'E', 'A', 'D'},
        {'E', 'D', 'D', 'A', 'C', 'B', 'E', 'E', 'A', 'D'},
        {'C', 'B', 'A', 'E', 'D', 'C', 'E', 'E', 'A', 'D'},
        {'A', 'B', 'D', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
        {'B', 'B', 'E', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
        {'B', 'B', 'A', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
        {'E', 'B', 'E', 'C', 'C', 'D', 'E', 'E', 'A', 'D'}
    };

    // Key to the questions
    char keys[] = {'D', 'B', 'D', 'C', 'C', 'D', 'A', 'E', 'A', 'D'};

    // Grade all answers
    for (int i = 0; i < NUMBER_OF_STUDENTS; i++)
    {
        // Grade one student
        int correctCount = 0;
        for (int j = 0; j < NUMBER_OF_QUESTIONS; j++)
        {
            if (answers[i][j] == keys[j])
                correctCount++;
        }

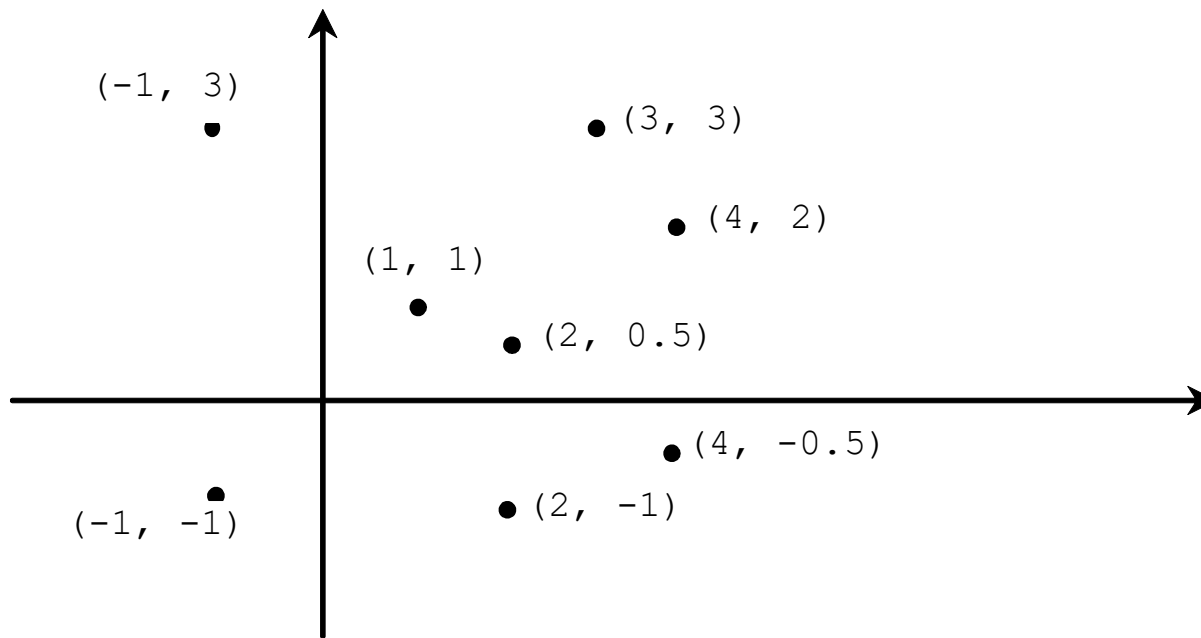
        cout << "Student " << i << "'s correct count is " <<
            correctCount << endl;
    }

    return 0;
}

```



Problem: Finding Two Points Nearest to Each Other



	x	y
0	-1	3
1	-1	-1
2	1	1
3	2	0.5
4	2	-1
5	3	3
6	4	2
7	4	-0.5

FindNearestPoints

FindNearestPoints

```

#include <iostream>
#include <cmath>
using namespace std;

// Compute the distance between two points (x1, y1) and (x2, y2)
double getDistance(double x1, double y1, double x2, double y2)
{
    return sqrt((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1));
}

int main()
{
    const int NUMBER_OF_POINTS = 8;

    // Each row in points represents a point
    double points[NUMBER_OF_POINTS][2];

    cout << "Enter " << NUMBER_OF_POINTS << " points: ";
    for (int i = 0; i < NUMBER_OF_POINTS; i++)
        cin >> points[i][0] >> points[i][1];

    // p1 and p2 are the indices in the points array
    int p1 = 0, p2 = 1; // Initial two points
    double shortestDistance = getDistance(points[p1][0], points[p1][1],
        points[p2][0], points[p2][1]); // Initialize shortestDistance

    // Compute distance for every two points
    for (int i = 0; i < NUMBER_OF_POINTS; i++)
    {
        for (int j = i + 1; j < NUMBER_OF_POINTS; j++)
        {
            double distance = getDistance(points[i][0], points[i][1],
                points[j][0], points[j][1]); // Find distance

            if (shortestDistance > distance)
            {
                p1 = i; // Update p1
                p2 = j; // Update p2
                shortestDistance = distance; // Update shortestDistance
            }
        }
    }

    // Display result
    cout << "The closest two points are " <<
        "(" << points[p1][0] << ", " << points[p1][1] << ") and (" <<
        points[p2][0] << ", " << points[p2][1] << ")" << endl;

    return 0;
}

```



Case Study: Sudoku

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6							
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

The objective is to fill the grid (see the left figure) so that every row, every column, and every 3×3 box contain the numbers 1 to 9, as shown in the right figure.

Case Study: Sudoku

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6							
			4	1	9			5
				8			7	9

5	3	1	2	7	4	8	9	
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6							
			4	1	9			5
				8			7	9

5	3	1	2	7	4	9		
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6							
			4	1	9			5
				8			7	9

Rule 1: Fill in an empty cell from the first to the last.

Rule 2: Fill in a smallest number possible.

Rule 3: If no number can fill in a cell, backtrack.



What is Sudoku?

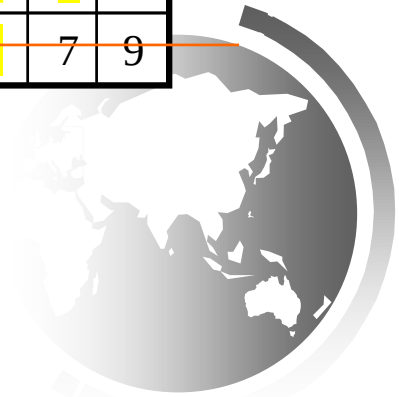
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6							
			4	1	9			5
				8			7	9



Every row contains the numbers 1 to 9

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6							
			4	1	9			5
				8			7	9

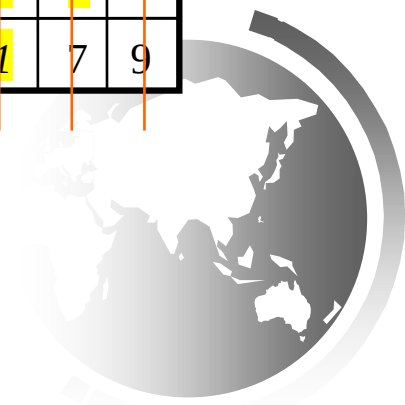
5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9



Every column contains the numbers 1 to 9

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6							
			4	1	9			5
				8			7	9

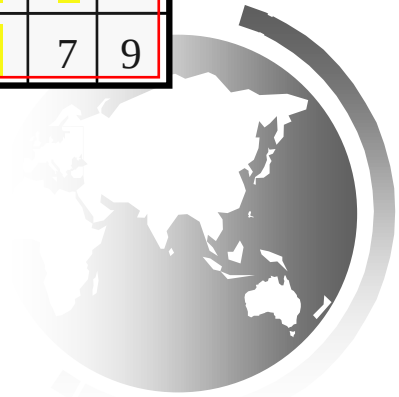
5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9



Every 3×3 box contains the numbers 1 to 9

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6							
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9



Checking Whether a Solution Is Correct

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6							
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

CheckSudokuSolution

CheckSudokuSolution

```

#include <iostream>
using namespace std;

void readASolution(int grid[][9]);
bool isValid(const int grid[][9]);
bool isValid(int i, int j, const int grid[][9]);

int main()
{
    // Read a Sudoku puzzle
    int grid[9][9];
    readASolution(grid);

    cout << (isValid(grid) ? "Valid solution" : "Invalid solution");

    return 0;
}

/** Read a Sudoku puzzle from the keyboard */
void readASolution(int grid[][9])
{
    cout << "Enter a Sudoku puzzle solution:" << endl;
    for (int i = 0; i < 9; i++)
        for (int j = 0; j < 9; j++)
            cin >> grid[i][j];
}

// Check whether the fixed cells are valid in the grid
bool isValid(const int grid[][9])
{
    for (int i = 0; i < 9; i++)
        for (int j = 0; j < 9; j++)
            if (grid[i][j] < 1 || grid[i][j] > 9 ||
                !isValid(i, j, grid))
                return false;

    return true; // The fixed cells are valid
}

```

```

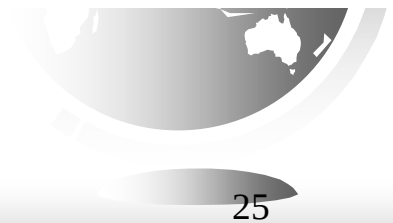
// Check whether grid[i][j] is valid in the grid
bool isValid(int i, int j, const int grid[][9])
{
    // Check whether grid[i][j] is valid at the i's row
    for (int column = 0; column < 9; column++)
        if (column != j && grid[i][column] == grid[i][j])
            return false;

    // Check whether grid[i][j] is valid at the j's column
    for (int row = 0; row < 9; row++)
        if (row != i && grid[row][j] == grid[i][j])
            return false;

    // Check whether grid[i][j] is valid in the 3-by-3 box
    for (int row = (i / 3) * 3; row < (i / 3) * 3 + 3; row++)
        for (int col = (j / 3) * 3; col < (j / 3) * 3 + 3; col++)
            if (row != i && col != j && grid[row][col] == grid[i][j])
                return false;

    return true; // The current value at grid[i][j] is valid
}

```



Multidimensional Arrays

In the preceding section, you used a two-dimensional array to represent a matrix or a table. Occasionally, you will need to represent n -dimensional data structures. In C++, you can create n -dimensional arrays for any integer n .

The way to declare two-dimensional array can be generalized to declare n -dimensional array for $n \geq 3$. For example, the following syntax declares a three-dimensional array scores.

double scores[10][5][2];



Problem: Guessing Birthday

Listing 4.4, `GuessBirthday.cpp`, gives a program that guesses a birthday. The program can be simplified by storing the numbers in five sets in a three-dimensional array, and it prompts the user for the answers using a loop.



GuessBirthdayUsingArray

GuessBirthdayUsingArray

```

#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    int day = 0; // Day to be determined
    char answer;

    int dates[5][4][4] = {
        {{ 1, 3, 5, 7},
         { 9, 11, 13, 15},
         {17, 19, 21, 23},
         {25, 27, 29, 31}},
        {{ 2, 3, 6, 7},
         {10, 11, 14, 15},
         {18, 19, 22, 23},
         {26, 27, 30, 31}},
        {{ 4, 5, 6, 7},
         {12, 13, 14, 15},
         {20, 21, 22, 23},
         {28, 29, 30, 31}},
        {{ 8, 9, 10, 11},
         {12, 13, 14, 15},
         {24, 25, 26, 27},
         {28, 29, 30, 31}},
        {{16, 17, 18, 19},
         {20, 21, 22, 23},
         {24, 25, 26, 27},
         {28, 29, 30, 31}}};

    for (int i = 0; i < 5; i++)
    {
        cout << "Is your birthday in Set" << (i + 1) << "?" << endl;
        for (int j = 0; j < 4; j++)
        {
            for (int k = 0; k < 4; k++)
                cout << setw(3) << dates[i][j][k] << " ";
            cout << endl;
        }
        cout << "\nEnter N/n for No and Y/y for Yes: ";
        cin >> answer;
        if (answer == 'Y' || answer == 'y')
            day += dates[i][0][0];
    }

    cout << "Your birthday is " << day << endl;

    return 0;
}

```

