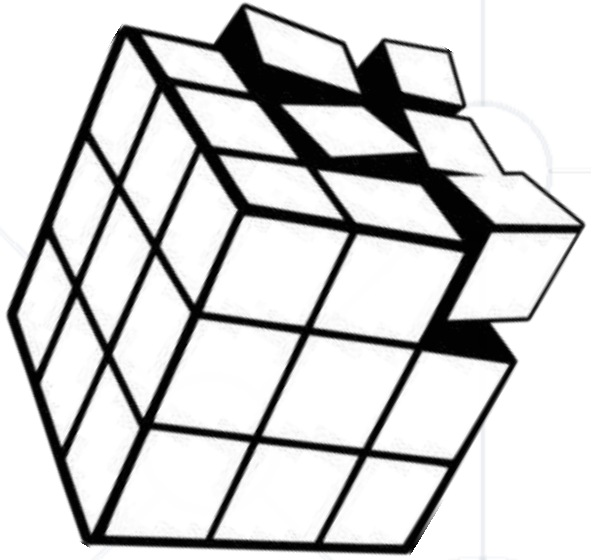
Multidimensional Arrays



SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg/

Have a Question?

Software University

sli.do

#cpp-advanced

Table of Contents

Multidimensional Arrays

Definition and Usage

What is Multidimensional Array?

- Array is a **systematic arrangement of** similar elements
- Multidimensional arrays **have more than one** dimension
  - They are just normal arrays which are indexed differently
- Most-common usage: making a **matrix / table**

ROWS	COLS				
	[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
	[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
	[2][0]	[2][1]	[2][2]	[2][3]	[2][4]

Col Index

Row Index

Accessing Multidimensional Arrays

- Accessing

Index of row

Index of column

```
int element = matrix[1][0];
```

1st element of the 2nd row

- Accessing elements is done with **one indexer per dimension**
- Multidimensional arrays represent a **rows with values**
- The rows represent the first dimension and the columns - the second

Declaring Multidimensional Arrays

- Declaring: add a `size` for each additional dimension

```
int matrix[2][3];
```

```
int matrix[][3];
```

First dimension can omit size if it is a function parameter

Using Multidimensional Arrays

Software University

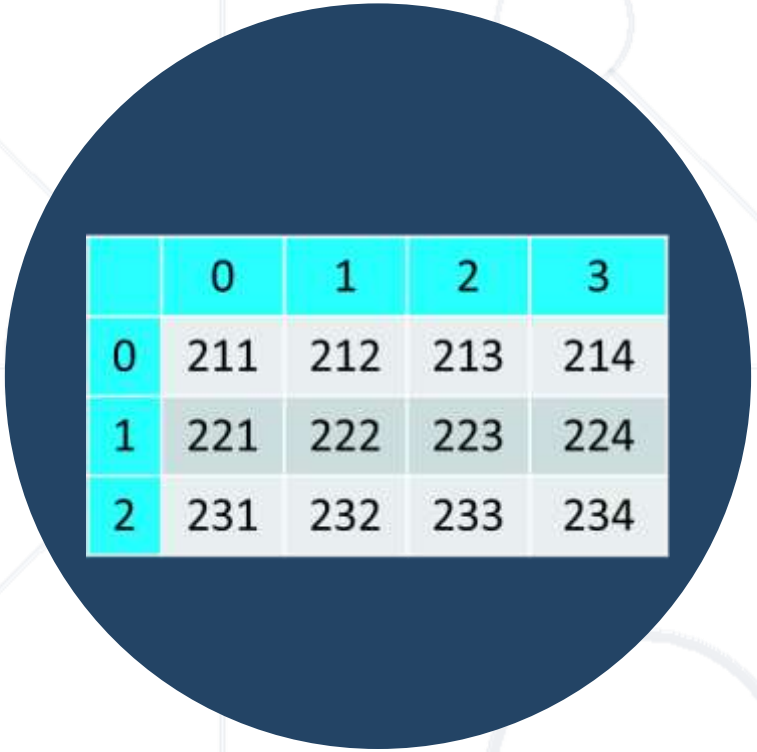
- Each **n**-dimention is an array with **(n - 1)** dimensions

```
int matrix[][3] =
{
  { 11, 12, 13 },
  { 21, 22, 23 }
};
```

If no initializer {} brackets, values are undefined

If more elements than initialized, others are defaults

```
int cube[2][3][4] =
{
  { {111, 112, 113, 114}, {121, 122, 123, 124}, {131, 132, 133, 134} },
  { {211, 212, 213, 214}, {221, 222, 223, 224}, {231, 232, 233, 234} }
};
```



Reading and Printing Matrices

Matrices and Higher Dimensions

Reading a Matrix

Software University

```
int main()
{
    int a[5][5];
    int row, col;
    cin >> row >> col;

    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            cin >> a[i][j];
        }
    }

    return 0;
}
```

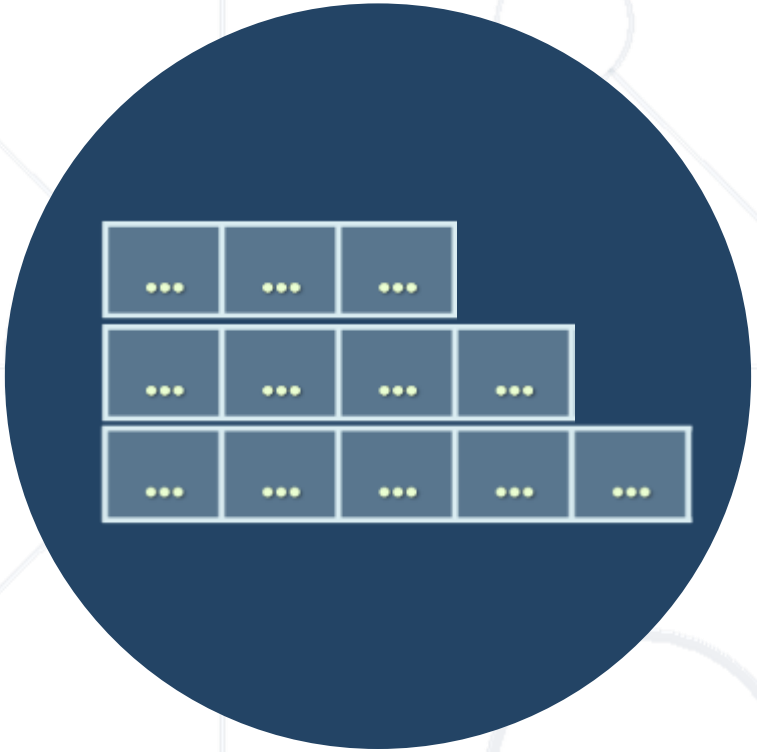
Printing a Matrix

Software University

```
int main()
{
    int a[5][5];
    int row, col;
    cin >> row >> col;

    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            cin >> a[i][j];
        }
     }
```

```
for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            cout << a[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```



Passing Arrays to Methods

Passing Arrays to Methods

- Arrays can be **passed to methods**

```
void foo(int arr[3][5])
```

```
void foo(int arr [][5])
```

- The first dimension **could be skipped**

"Multidimensional" Containers

Software University

- We know `std::vector` can contain any type
  - Any type with a default constructor
  - `int`, `double`, `char`, `string`, even another `std::vecto`
- Often containers will contain other containers
- **Example:** a vector of vectors (2D), a vector of vector of vectors (3D)
  - Element access is the same code as with multidimensional arrays
  - **Note**: no row-major order (not contiguous in memory)

std::array & std::vector

Software University

- Multidimensional arrays could be created with
  - `std::array`
  - `std::vector`

- If we know the needed size in advance we use `std::array`
- Arrays' data is allocated on the stack

std::array Matrix

Software University

```
const int rows = 3;
const int cols = 5;
// create an empty matrix
std::array<std::array<int, cols>, rows> matrix;

//initialize a matrix
std::array<std::array<int, cols>, rows> matrix
{
  { 0, 1, 2, 3, 4 },
  { 1, 2, 3, 4, 5 },
  { 2, 3, 4, 5, 6 }
};
```

std::vector Matrix

- If we don't know the size we use a `std::vector`

```
//create an empty matrix
std::vector<std::vector<int>> matrix;

//initialize a matrix
std::vector<std::vector<int>, rows> matrix
{
  { 0, 1, 2, 3, 4 },
  { 1, 2, 3 }
  { 2, 3, 4, 5, 6, 7, 8 }
};
```

When we have vectors - the matrix can have any size

Working with 2D std::vector

Software University

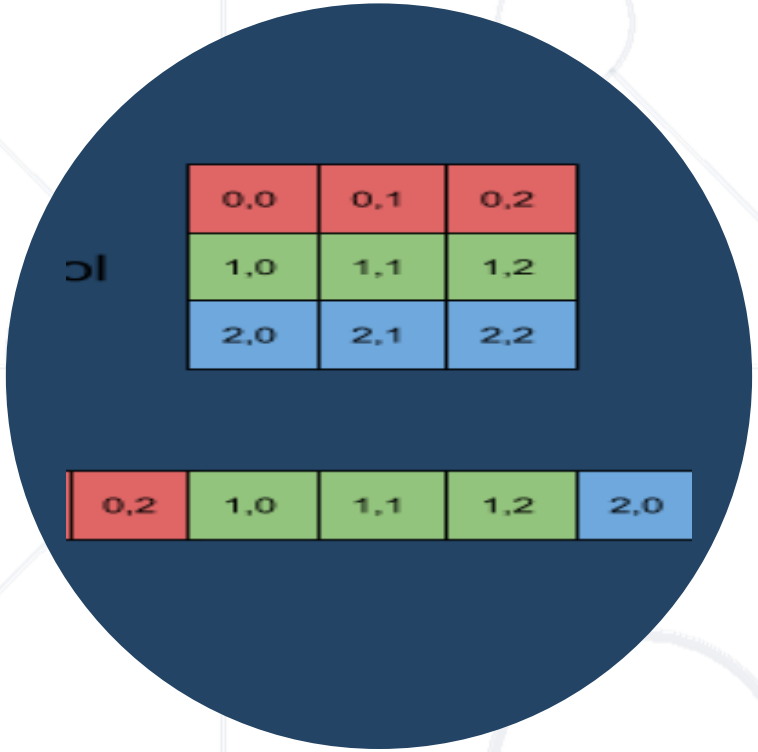
- Working with 2D **std::vector** when dealing with **methods**
- A method can return a populated matrix

```
std::vector<std::vector<int>> readMatrix()
```

- A method can accept the 2D std::matrix as a normal function parameter

```
void foo(std::vector<std::vector<int>> matrix); // makes a copy
```

```
void foo(std::vector<std::vector<int>>& matrix); // passed by reference
```



Row-Major Order in Multidimensional Arrays

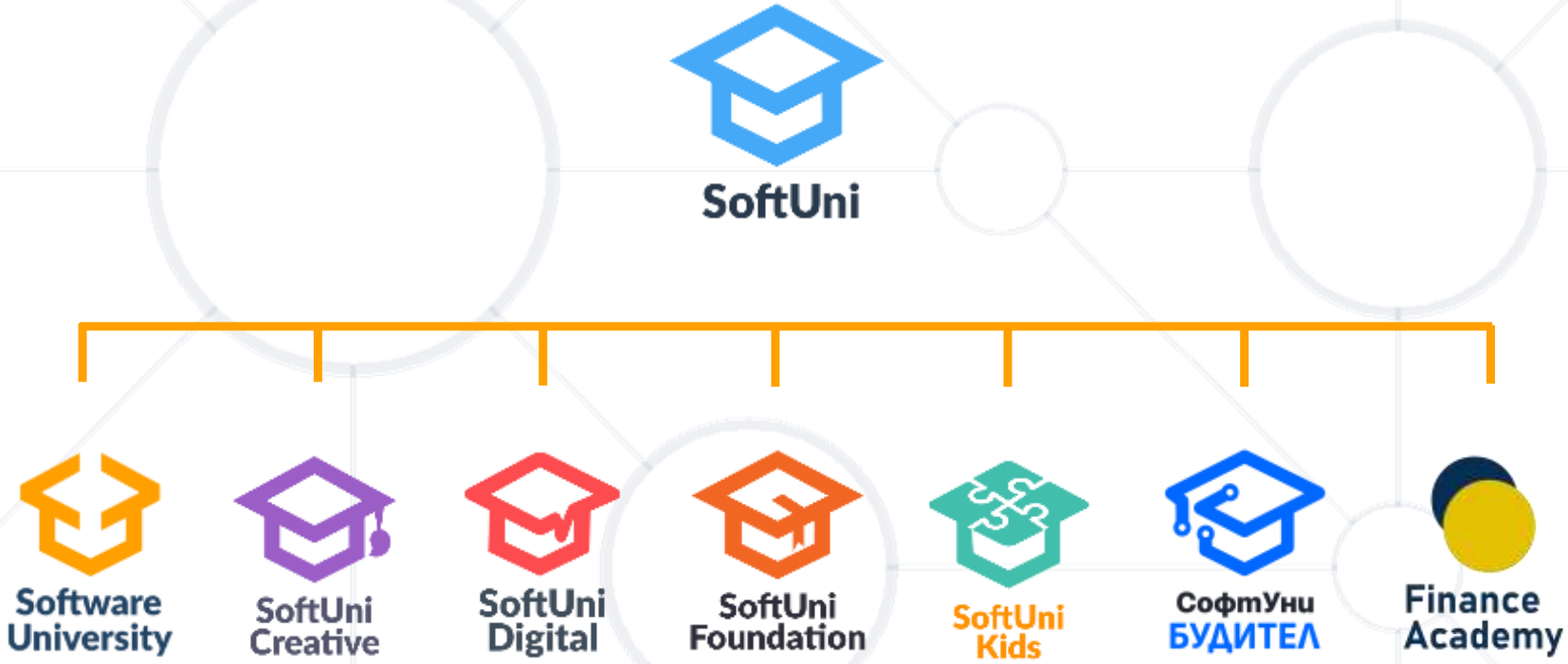
Row-Major Programming Language

- In row-major order
  - The consecutive elements of a **row reside next to each other**, whereas the same holds true for **consecutive elements of a column in column-major order**
- C++ is **Row-Major Based Programming Language**

Summary

- Multidimensional arrays
  - Have **more than one** dimension
  - Two-dimensional arrays are like tables with **rows** and **columns**
  - Most-common usage: making a matrix or a table
- C++ is Row-Major Based Programming Language

Questions?



SoftUni Diamond Partners

Software University

SmartIT

Coca-Cola HBC Bulgaria

SUPER HOSTING .BG

createX

Postbank

Решения за твоето утре

DRAFT KINGS

VIVACOM

AMBITIONED

INDEAVR

Serving the high achievers

SOFTWARE GROUP

PHAR VISION

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg