

CSC 211: Computer Programming

Multidimensional Arrays, fstream

Michael Conti

Department of Computer Science and Statistics
University of Rhode Island

Summer 2025

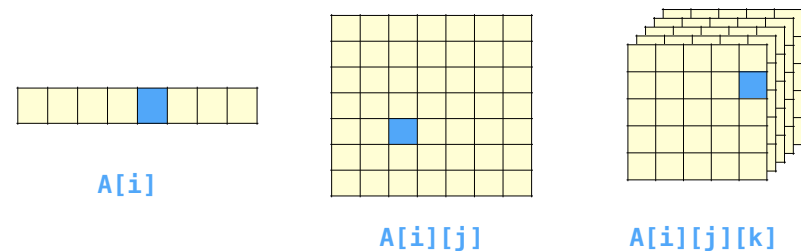


Multidimensional Arrays

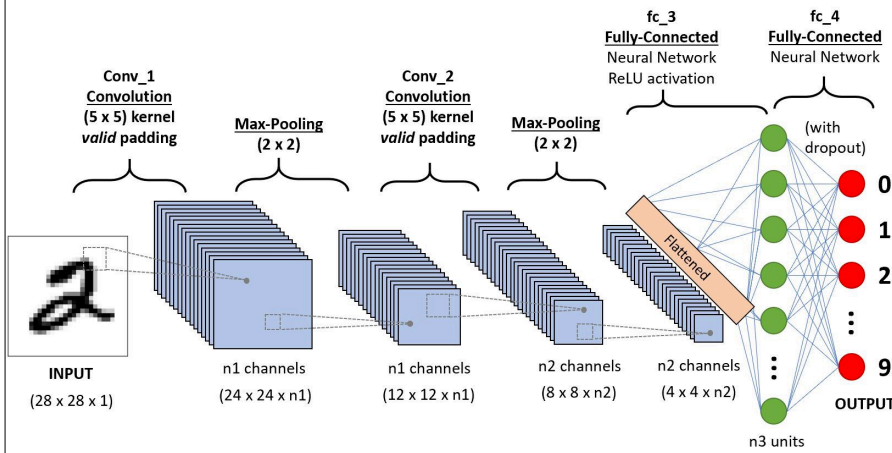
Arrays, of any dimension, are **statically allocated** in memory with a size calculated at compile time. That is, their size is **fixed** and **cannot** be changed later.

Multidimensional Arrays

- Generalization of **arrays** to multiple dimensions
 - ✓ e.g. matrices, tensors
- Each element can be accessed using its corresponding **indices**



Modern machine learning



<https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

5

Declaration of 2D arrays

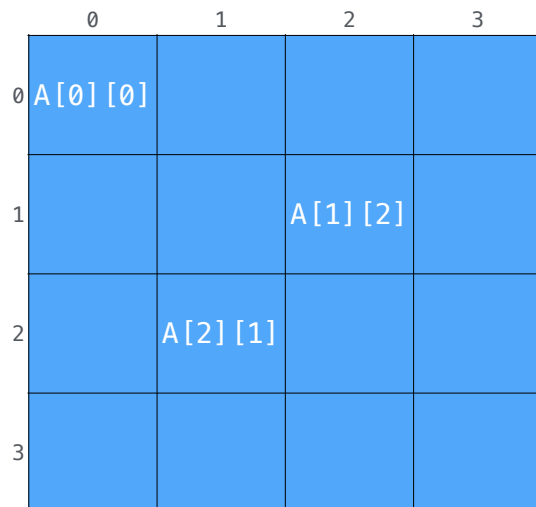
```
// array declaration by specifying size
int matrix1[10][10];

// can also declare an array of
// user specified size
int n = 8;
int matrix2[n][n];

// can declare and initialize elements
double matrix3[2][2];
matrix3 = { {10.0, 20.0}, {30.0, 40.0} };
```

6

Indexing 2D arrays



7

Indexing 2D arrays

- Individual elements can be accessed by using the **subscription operator []**

```
int matrix2[3][3];

for (int i = 0 ; i < 3 ; i ++ ) {
    for (int j = 0 ; j < 3 ; j ++ ) {
        matrix[i][j] = (j + 1) + i * 3;
    }
}
```

8

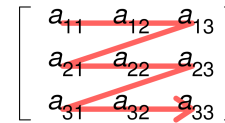
How are these arrays stored in memory?

- In computing, **row-major** order and **column-major** order are two methods for storing multidimensional arrays as contiguous blocks of memory
 - ✓ row-major order is used in C, C++, Objective-C (for C-style arrays), PL/I, Pascal, Speakeasy, SAS, ...
 - ✓ column-major order is used in Fortran, MATLAB, GNU Octave, S-Plus, R, Julia, ...
- Alternatively, neither row-major or column-major approaches are also used (non-contiguous blocks)
 - ✓ Java, C#, CLI, .Net, Scala, Swift, Python, Lua, ...

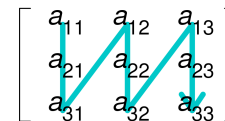
9

Row-major and column-major order

Row-major order



Column-major order



	0	1	2	3
0	1	2	3	4
1	8	6	4	2
2	10	20	30	40
3	5	7	9	11



10

Question

- How many bytes are these arrays using in memory?

```
int array[100000];
```

```
int matrix[1000][1000];
```

```
double tensor[1000][1000][1000];
```

11

Question

- Write a program that reads in the value of n, and prints the identity matrix of size n x n?

12

Multidimensional arrays and functions

- The first array size need not be specified
- The second (and any subsequent) must be given
- Example:

```
int foo(int list[][100], int rows, int cols);
```

size is required so the compiler can calculate the memory addresses of individual elements

<https://stackoverflow.com/questions/12813494/why-do-we-need-to-specify-the-column-size-when-passing-a-2d-array-as-a-parameter>

13

Multidimensional arrays and functions

- Variable sized 2D arrays are not very well supported by the built-in components of C and C++
- Need to know size of 2D array by compile time in function parameter list
- Can get around this by setting a max size of 2D in as parameter

14

Multidimensional arrays and functions

- Function printMatrix expects 5x5 matrix
- Relevant data is 3x4
- Only iterate over row (3) x col (4) to manipulate matrix data

```
void printMatrix(int m1[][5] int row, int col
```

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

15

Multidimensional vectors and functions

- Can also use vectors

```
void printMatrix(vector< vector<int> > m1){  
    m1.size() // gets number of rows  
    m1[0].size() // gets number of columns  
}
```

16

Question

- Write a void function that adds two (N×N) 2D matrices together where $1 < N \leq 10$ to `std::cout`.

1	2	3		1	2	3		2	4	6
4	5	6		1	2	3		5	7	9
7	8	9		1	2	3		8	10	12

M1 + M2 = M3

17

fstream

fstream

- `fstream` stands for file stream and is used for handling files in C++
- Provides classes for reading from and writing to files:
 - `ifstream` (input file stream)
 - `ofstream` (output file stream)
 - `fstream` (for both input and output)
- Common operations: open, close, read, write, etc.

19

fstream

```
#include <iostream>
#include <fstream>
#include <vector>

int main() {
    // Open the file
    std::ifstream inputFile("matrix.txt");

    // Check if file opened successfully
    if (!inputFile.is_open()) {
        std::cerr << "Error: Failed to open the file." << std::endl;
        return 1;
    }

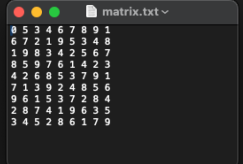
    // 2D vector to store the matrix
    std::vector<std::vector<int>> matrix;

    // Read the matrix from the file
    for (int i = 0; i < 9; ++i) {
        std::vector<int> row;
        for (int j = 0; j < 9; ++j) {
            int num;
            inputFile >> num;
            row.push_back(num);
        }
        matrix.push_back(row);
    }

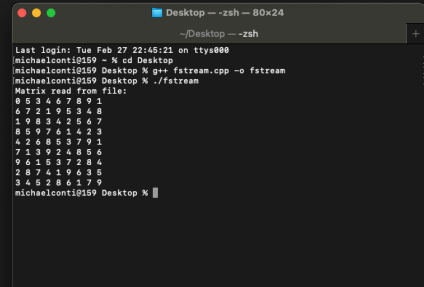
    // Close the file
    inputFile.close();

    // Display the matrix
    std::cout << "Matrix read from file:" << std::endl;
    for (const auto& row : matrix) {
        for (int num : row) {
            std::cout << num << " ";
        }
        std::cout << std::endl;
    }

    return 0;
}
```



```
0 5 3 4 6 7 8 9 1
6 7 2 1 9 5 3 4 8
1 9 3 4 2 5 6 7
8 5 9 7 6 1 4 2 3
4 2 8 8 5 3 7 9 1
7 1 2 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
```



```
Last login: Tue Feb 27 22:45:21 on ttys000
michaelcont@1959 ~ % cd Desktop
michaelcont@1959 Desktop % g++ fstream.cpp -o fstream
michaelcont@1959 Desktop % ./fstream
Matrix read from file:
0 5 3 4 6 7 8 9 1
6 7 2 1 9 5 3 4 8
1 9 3 4 2 5 6 7
8 5 9 7 6 1 4 2 3
4 2 8 8 5 3 7 9 1
7 1 2 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
michaelcont@1959 Desktop %
```

20