CSC 211: Computer Programming
Multidimensional Arrays

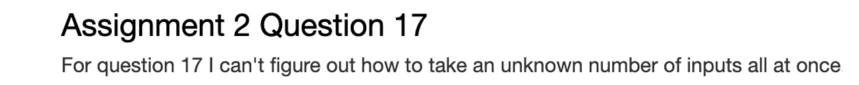
Michael Conti

Department of Computer Science and Statistics
University of Rhode Island

Summer 2023

THINK BIG WE DO℠

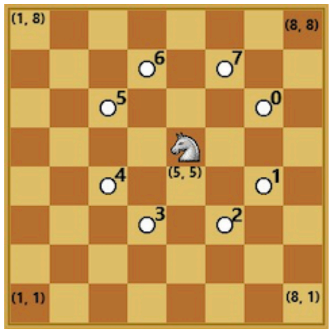
Administrative Announcements



Assignment 2 Question 17

For question 17 I can't figure out how to take an unknown number of inputs all at once

```
while (std::cin >> move) {
    switch (move){
        case 0:
            x += 2;
            y += 1;
            break;
```

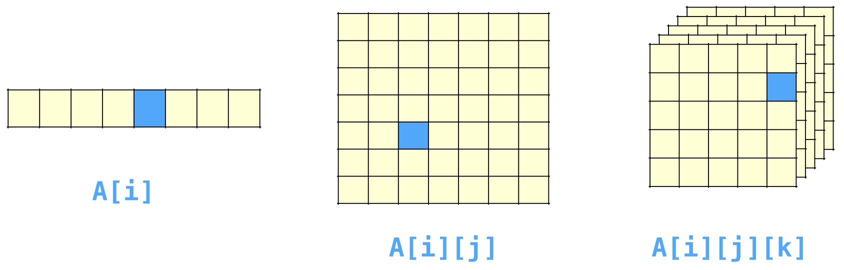


echo 3 4 0 3 3 6 6 1 5 5 4 | ./main_1

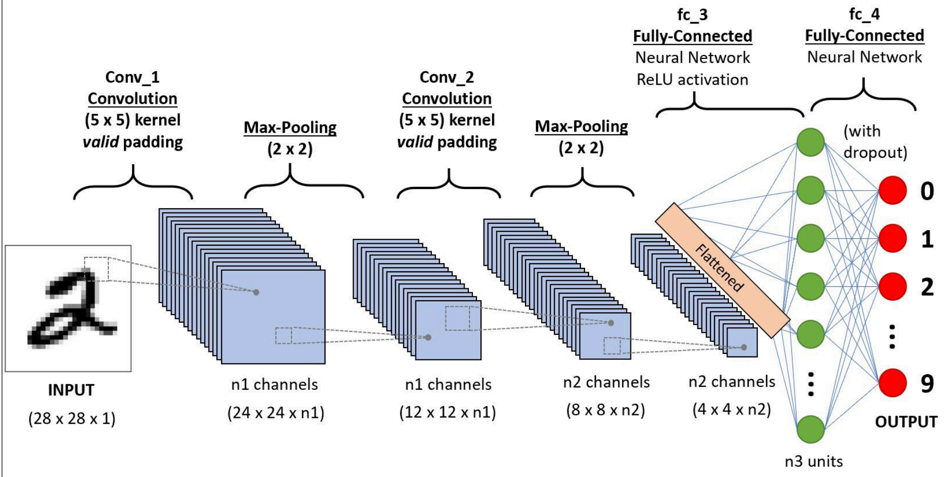
Arrays, of any dimension, are statically allocated in memory with a size calculated at compile time. That is, their size is **fixed** and **cannot** be changed later.

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

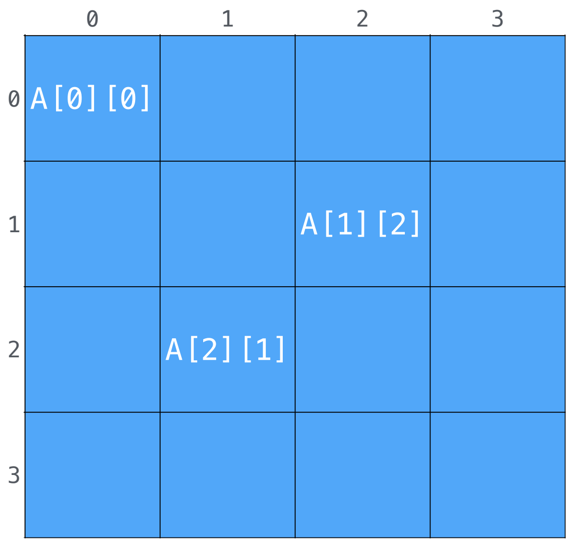
Declaration of 2D arrays

```
// 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} };
```

Indexing 2D arrays



Indexing 2D arrays

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

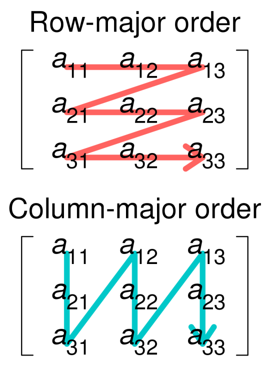
```
int matrix2[3][3];

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

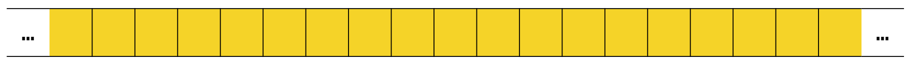
How are these arrays stored in memory?

- 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, …

Row-major and column-major order


Row-major order
a_{11} a_{12} a_{13}
a_{21} a_{22} a_{23}
a_{31} a_{32} a_{33}
Column-major order
a_{11} a_{12} a_{13}
a_{21} a_{22} a_{23}
a_{31} a_{32} a_{33}

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



Question

- How many bytes are these arrays using in memory?

```
int array[100000];


int matrix[1000][1000];


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

Question

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

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

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

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

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
}
```

Question

- Write a function that adds two (NxN) 2D matrices together where 1 < N <= 10.

$$\underset{M1}{\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{bmatrix}} + \underset{M2}{\begin{bmatrix}1 & 2 & 3\\ 1 & 2 & 3\\ 1 & 2 & 3\end{bmatrix}} = \underset{M3}{\begin{bmatrix}2 & 4 & 6\\ 5 & 7 & 9\\ 8 & 10 & 12\end{bmatrix}}$$