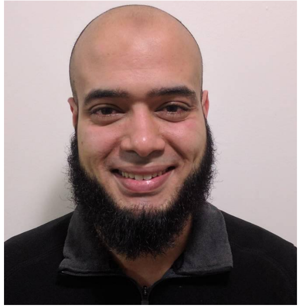
Programming4kids

2D Arrays

Mostafa Saad Ibrahim
Computer Vision Researcher @ Huawei Canada
PhD - Simon Fraser University
Bachelor / Msc - FCI Cairo University

Ex-(Software Engineer / Teaching Assistant)



Motivation

- Write a program that reads grades for students
  - 100 students
  - 20 subjects
- How can we code that?
  - Create 20 arrays grade1[100], grade2[100], …..grade20[100];
  - So impractical!
- Let’s visualize the data

Grades visualization: 7 students x 4 subjects

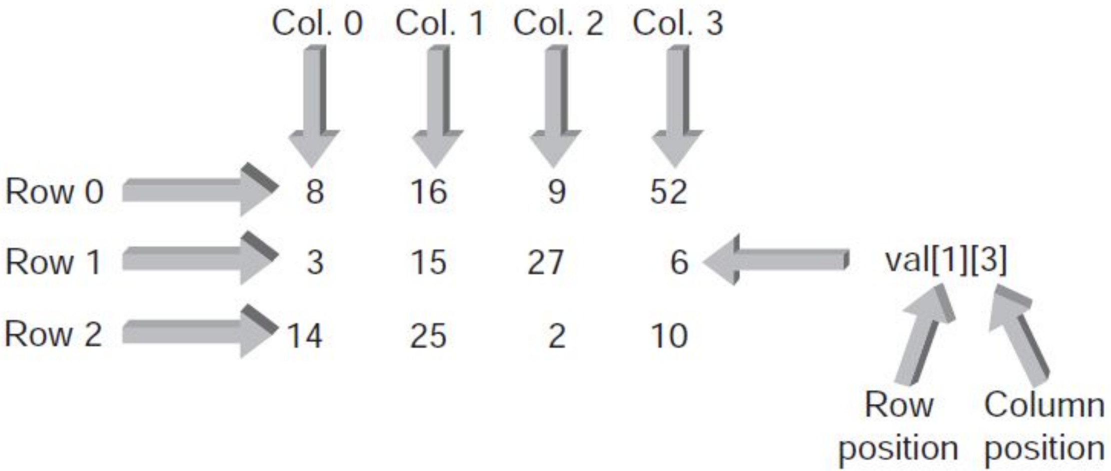
	Math	**Science**	**History**	**Arts**
Mostafa	50	33	40	30
Asmaa	35	50	44	17
Belal	30	35	50	37
Ziad	50	35	44	22
Safa	50	44	50	30
Ashraf	50	36	18	50
Mona	35	30	**47**	16

- This is called a matrix/table
  - The blue numbers
- 7 rows
  - Row 0, 1, 2, … 6
  - Row 0 for mostafa
  - Row 6 for mona
- 4 Columns
  - Column 0, 1, 2, 3
  - Column 0 for Math
- Value of table: row 6, col 2
  - 47 (Mona & History)
  - Notation: [6][2]

2D Arrays

- C++ saves our time by using 2D arrays
  - 2D = Table: rows x columns
- Same rules as 1D Arrays
- We create it as
  - double grades[7][4];
    - For 7 rows and 4 columns
  - To access in 2D arrays:
    - grades[6][2]

2D Arrays Visualization

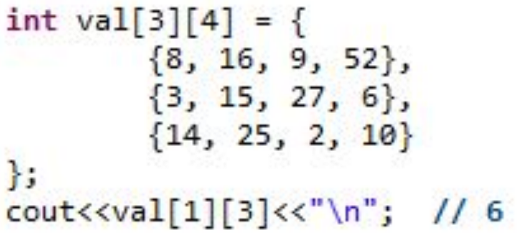


	Col. 0	Col. 1	Col. 2	Col. 3
Row 0	8	16	9	52
Row 1	3	15	27	6
Row 2	14	25	2	10

val[1][3]

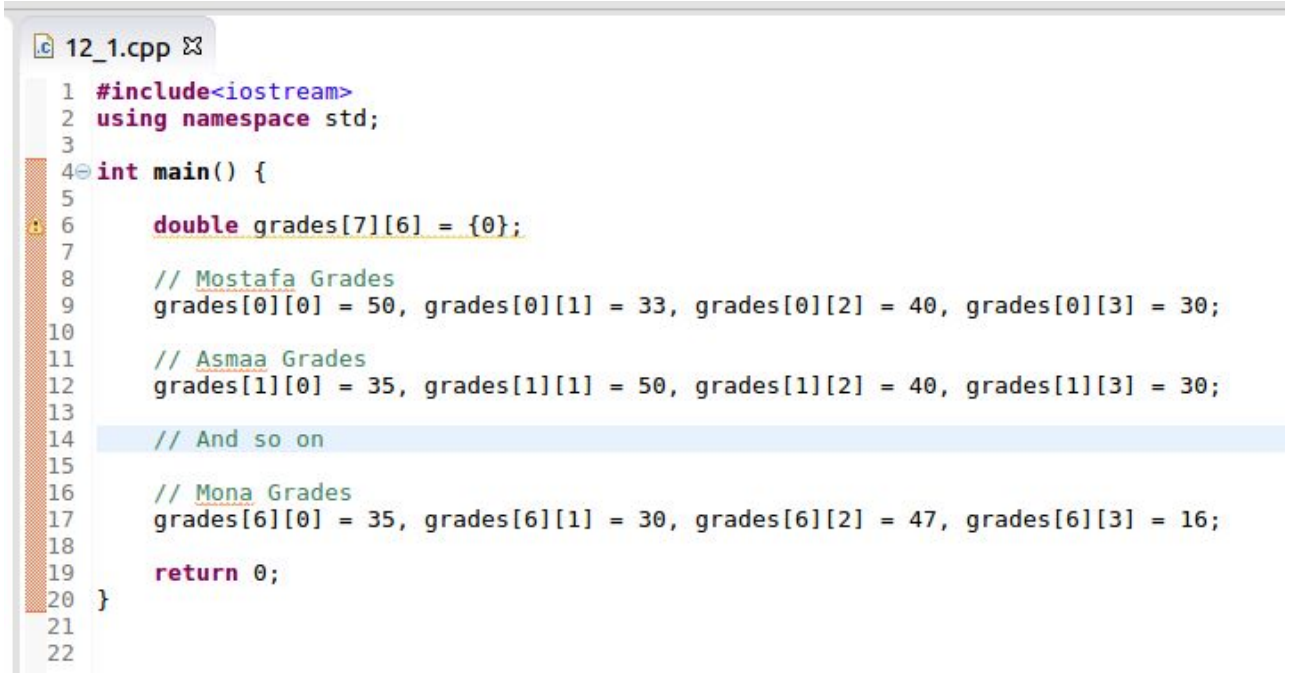
Row position

Column position



```
int val[3][4] = {
            {8, 16, 9, 52},
            {3, 15, 27, 6},
            {14, 25, 2, 10}
};
cout<<val[1][3]<<"\n";  // 6
```

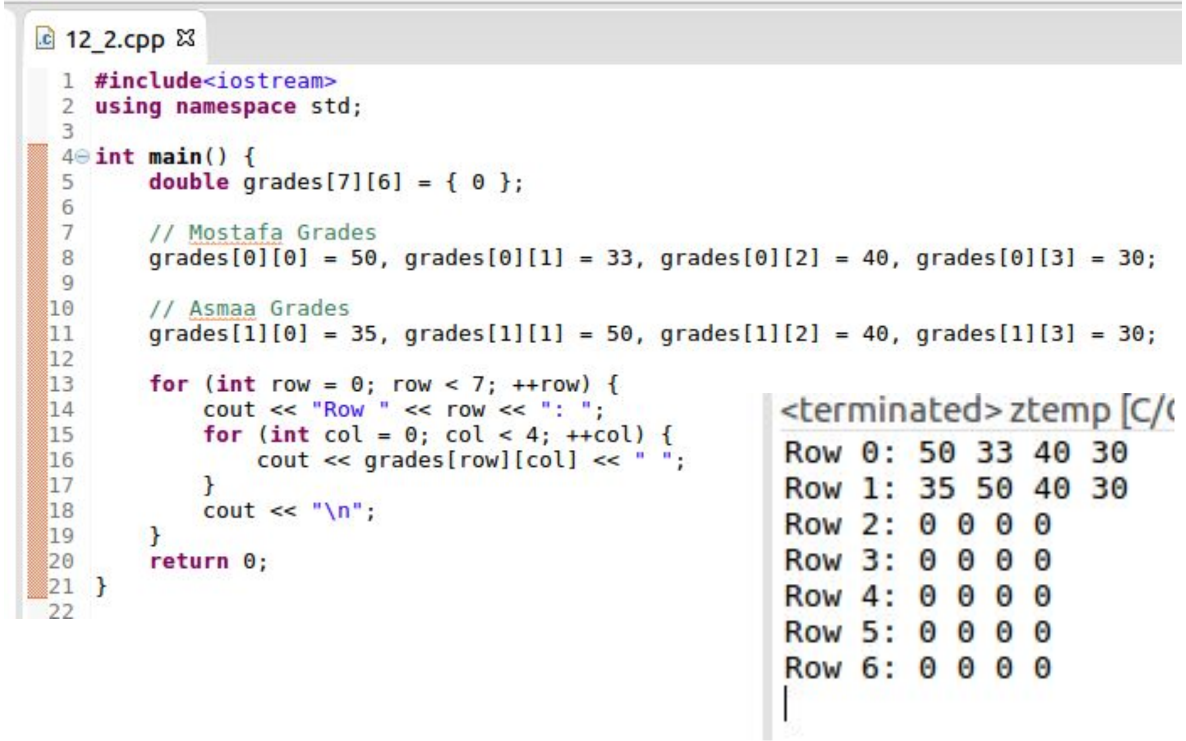
Let’s put the values



```cpp
#include<iostream>
using namespace std;

int main() {

    double grades[7][6] = {0};

    // Mostafa Grades
    grades[0][0] = 50, grades[0][1] = 33, grades[0][2] = 40, grades[0][3] = 30;

    // Asmaa Grades
    grades[1][0] = 35, grades[1][1] = 50, grades[1][2] = 40, grades[1][3] = 30;

    // And so on

    // Mona Grades
    grades[6][0] = 35, grades[6][1] = 30, grades[6][2] = 47, grades[6][3] = 16;

    return 0;
}
```

- Notice
- All mostafa data has grades[0]
- All Asmaa data has grades[1]
- All mona data has grades[6]
- Notice all inces
    - 0-6 for rows
    - 0-3 for columns

Let’s print it

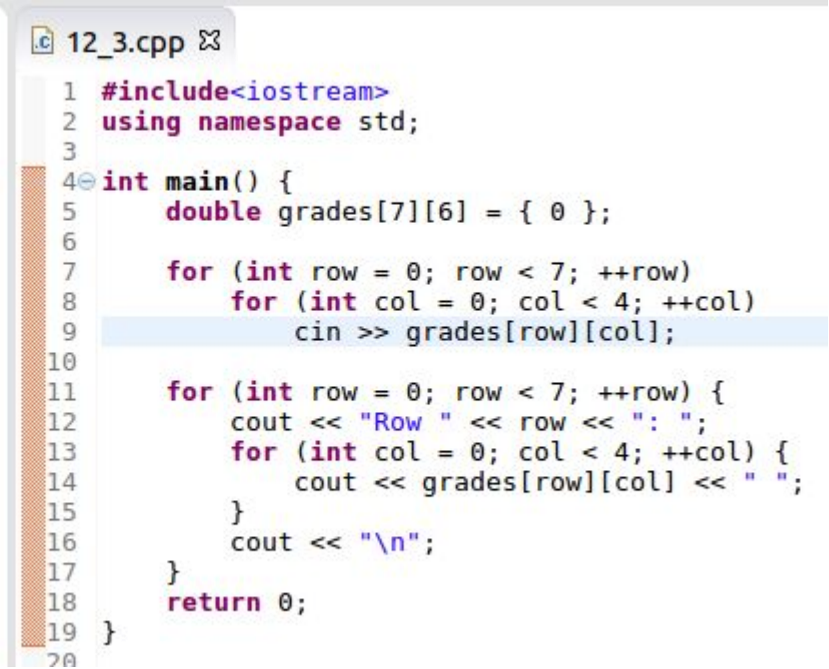


```cpp
#include<iostream>
using namespace std;

int main() {
    double grades[7][6] = { 0 };

    // Mostafa Grades
    grades[0][0] = 50, grades[0][1] = 33, grades[0][2] = 40, grades[0][3] = 30;

    // Asmaa Grades
    grades[1][0] = 35, grades[1][1] = 50, grades[1][2] = 40, grades[1][3] = 30;

    for (int row = 0; row < 7; ++row) {
        cout << "Row " << row << ": ";
        for (int col = 0; col < 4; ++col) {
            cout << grades[row][col] << " ";
        }
        cout << "\n";
    }
    return 0;
}

```

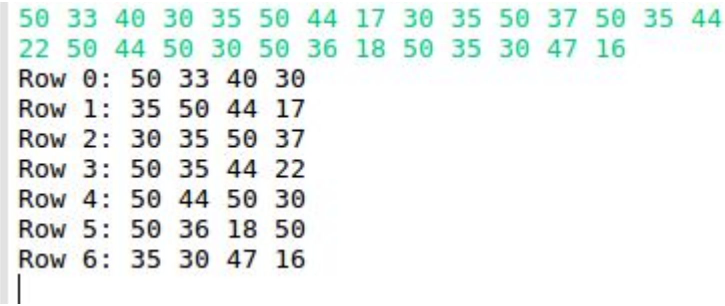
```
<terminated> ztemp [C/C
Row 0: 50 33 40 30
Row 1: 35 50 40 30
Row 2: 0 0 0 0
Row 3: 0 0 0 0
Row 4: 0 0 0 0
Row 5: 0 0 0 0
Row 6: 0 0 0 0
```

- To print
  - Loop over every row
  - Then for this row
    - Loop on its columns
- We will loop this way typically
- We can also loop on columns then loop on rows

Easier: Let’s read then print!

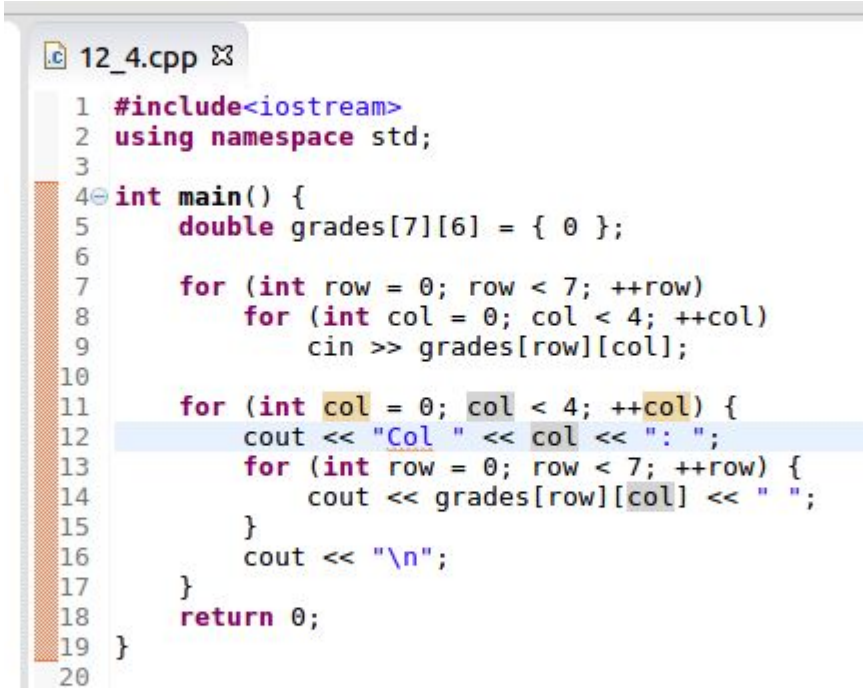


```cpp
#include<iostream>
using namespace std;

int main() {
    double grades[7][6] = { 0 };

    for (int row = 0; row < 7; ++row)
        for (int col = 0; col < 4; ++col)
            cin >> grades[row][col];

    for (int row = 0; row < 7; ++row) {
        cout << "Row " << row << ": ";
        for (int col = 0; col < 4; ++col) {
            cout << grades[row][col] << " ";
        }
        cout << "\n";
    }
    return 0;
}
```



```
50 33 40 30 35 50 44 17 30 35 50 37 50 35 44
22 50 44 50 30 50 36 18 50 35 30 47 16
Row 0: 50 33 40 30
Row 1: 35 50 44 17
Row 2: 30 35 50 37
Row 3: 50 35 44 22
Row 4: 50 44 50 30
Row 5: 50 36 18 50
Row 6: 35 30 47 16
```

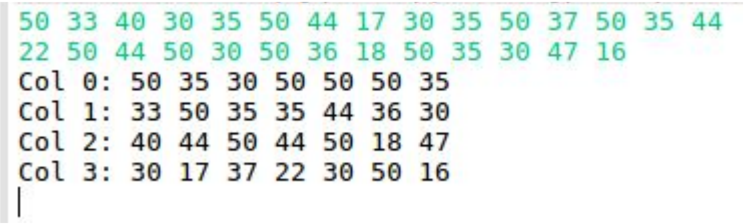
Column Row Order



12_4.cpp

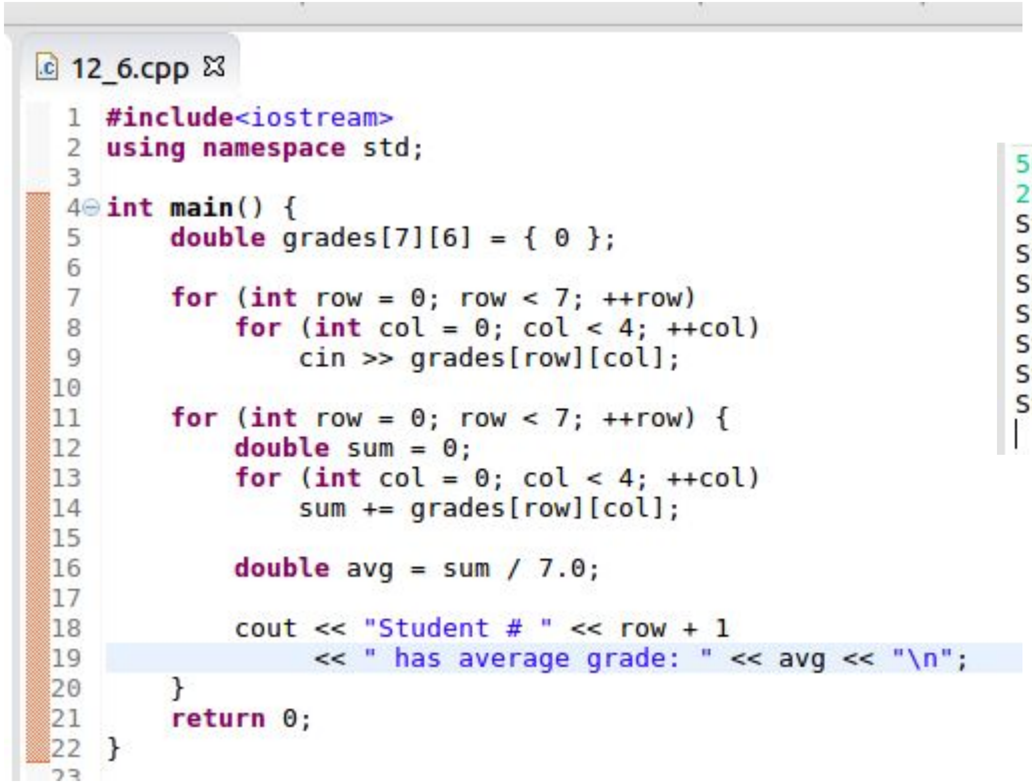
```cpp
#include<iostream>
using namespace std;

int main() {
    double grades[7][6] = { 0 };

    for (int row = 0; row < 7; ++row)
        for (int col = 0; col < 4; ++col)
            cin >> grades[row][col];

    for (int col = 0; col < 4; ++col) {
        cout << "Col " << col << ": ";
        for (int row = 0; row < 7; ++row) {
            cout << grades[row][col] << " ";
        }
        cout << "\n";
    }
    return 0;
}
```

- We can also see it from the columns perspective
    - ○ Note: This is slower :)



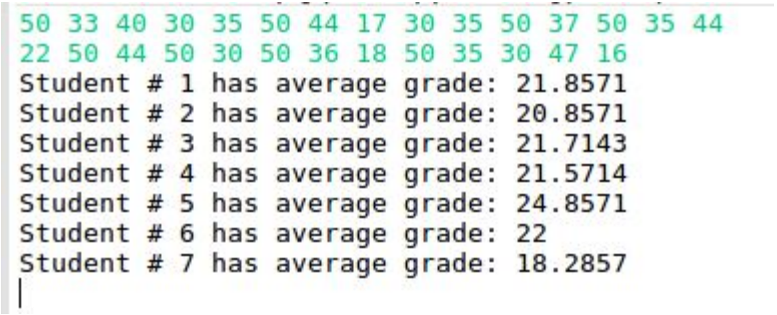
```
50 33 40 30 35 50 44 17 30 35 50 37 50 35 44
22 50 44 50 30 50 36 18 50 35 30 47 16
Col 0: 50 35 30 50 50 50 35
Col 1: 33 50 35 35 44 36 30
Col 2: 40 44 50 44 50 18 47
Col 3: 30 17 37 22 30 50 16
```

Let’s compute average grade per student



12_6.cpp

```cpp
#include<iostream>
using namespace std;

int main() {
    double grades[7][6] = { 0 };

    for (int row = 0; row < 7; ++row)
        for (int col = 0; col < 4; ++col)
            cin >> grades[row][col];

    for (int row = 0; row < 7; ++row) {
        double sum = 0;
        for (int col = 0; col < 4; ++col)
            sum += grades[row][col];

        double avg = sum / 7.0;

        cout << "Student # " << row + 1
             << " has average grade: " << avg << "\n";
    }
    return 0;
}
```



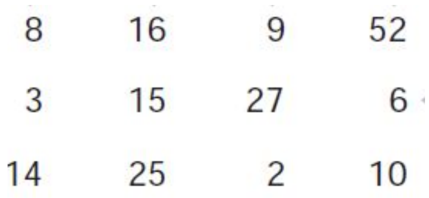
```
50 33 40 30 35 50 44 17 30 35 50 37 50 35 44
22 50 44 50 30 50 36 18 50 35 30 47 16
Student # 1 has average grade: 21.8571
Student # 2 has average grade: 20.8571
Student # 3 has average grade: 21.7143
Student # 4 has average grade: 21.5714
Student # 5 has average grade: 24.8571
Student # 6 has average grade: 22
Student # 7 has average grade: 18.2857
```

Multidimensional Arrays

- What if we have 5 years. For each year, we have 100 students and 20 subjects? How to represent?
  - 5 Arrays, each one is 2D array [100][20]
  - Not convenient
- C++: double grades[5][100][20];
  - grades[2][70][8];
  - Grade for the 3rd year, student #71, 9th subject
  - This is 2 * 70 * 8 double numbers
- You can do bigger arrays
  - Int results[10][10][10][10][10][10];
    - This is 1000,000 numbers. Be careful.

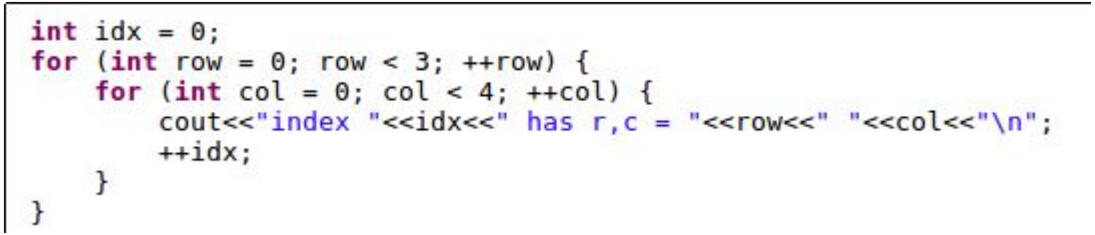
Flatten an array

- To flatten array, means convert to 1D array
- You simply put values from rows in order
- E.g. array 1D now is:
    - **8 16 9 52** 3 15 27 6 **14 25 2 10**



8	16	9	52
3	15	27	6
14	25	2	10

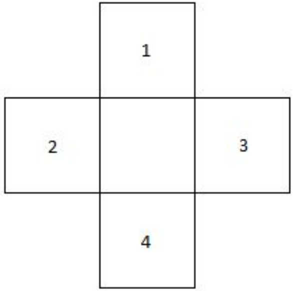
- Let say the 2D array is 3x4. Then new 1D array has length 12 also
    - If we have position (i, j) in 2D array, what is index in 1D array?
    - If we have index in 1D array, what is the position (i, j) in 2D array?
    - Find a simple formula for each of them. Use a code to enumerate

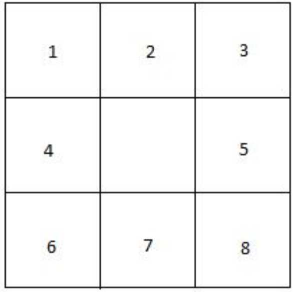


```
int idx = 0;
for (int row = 0; row < 3; ++row) {
    for (int col = 0; col < 4; ++col) {
        cout<<"index "<<idx<<" has r,c = "<<row<<" "<<col<<"\n";
        ++idx;
    }
}
```

Position neighbours

- For a position (i, j)
    - Sometimes we 4 neighbours
        - **up, right, down, left**
    - Sometimes we need 8 neighbours
        - **up, right, down, left**, up right, up left, down right, down left
        - Given (i, j), can u use a loop of 8 steps and print theses 8 positions, elegantly?

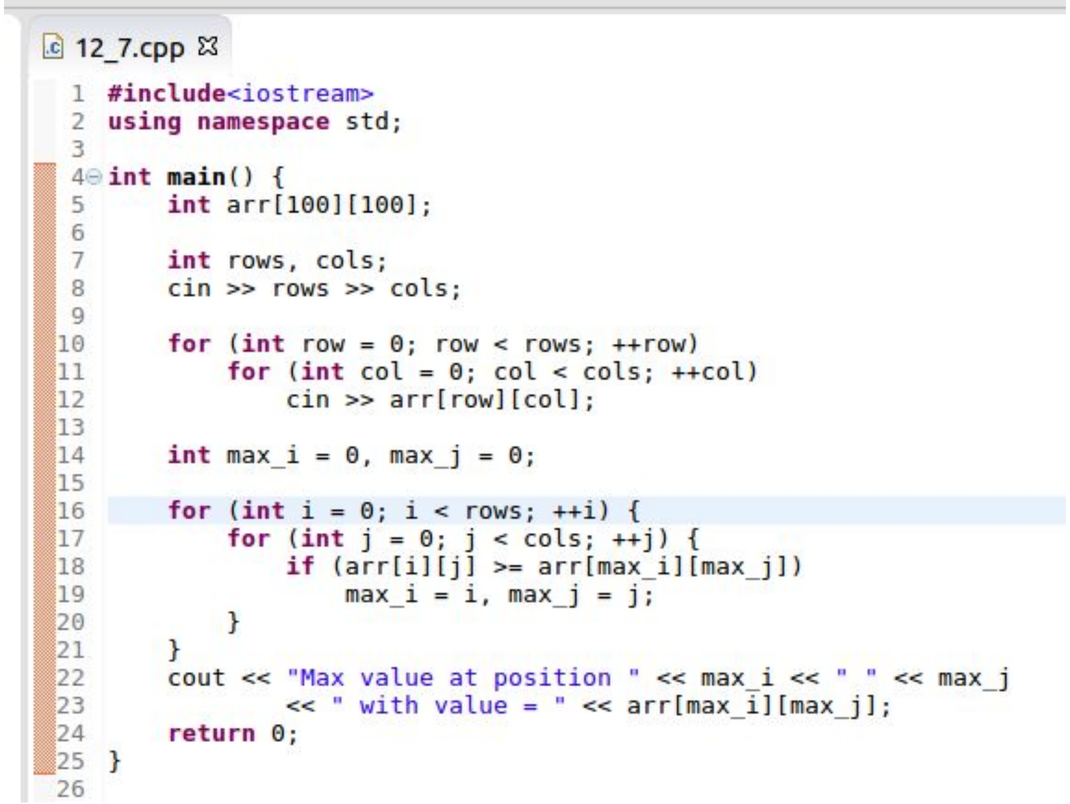




Practice: Max value

- Read 2 integers for the rows and columns of a matrix (<= 100). Then read rows x cols integer value. Find the position of maximum value in the array. If there are several ones, find the last occurance
- Input:
  - 3 4
  - 1 5 1 10
  - 2 10 3 4
  - 1 10 **10** 7
- Output
  - Max value at position 2 2 with value = 10

Practice: Max value

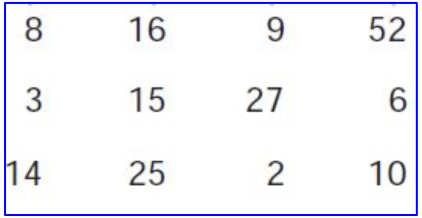


```cpp
#include<iostream>
using namespace std;

int main() {
    int arr[100][100];

    int rows, cols;
    cin >> rows >> cols;

    for (int row = 0; row < rows; ++row)
        for (int col = 0; col < cols; ++col)
            cin >> arr[row][col];

    int max_i = 0, max_j = 0;

    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            if (arr[i][j] >= arr[max_i][max_j])
                max_i = i, max_j = j;
        }
    }
    cout << "Max value at position " << max_i << " " << max_j
            << " with value = " << arr[max_i][max_j];
    return 0;
}
```

- Using >= finds last occurance

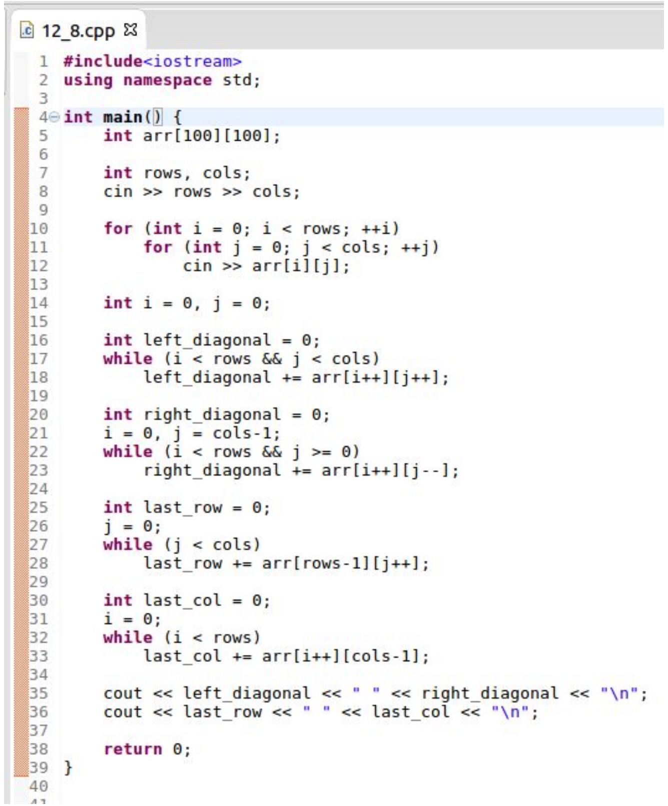
Practice: Special print

- Read 2 integers for the rows and columns of a matrix (<= 100). Then read rows x cols integer value.
- Print the following 4 values
  - The sum of the left diagonal & The sum of the right diagonal
  - The sum of the last row & The sum of the last column
- Input: 3 4
  - 8 16 9 52
  - 3 15 27 6
  - 14 25 2 10
- Output
  - 25 104
  - 51 68



8	16	9	52
3	15	27	6
14	25	2	10

Practice: Special print

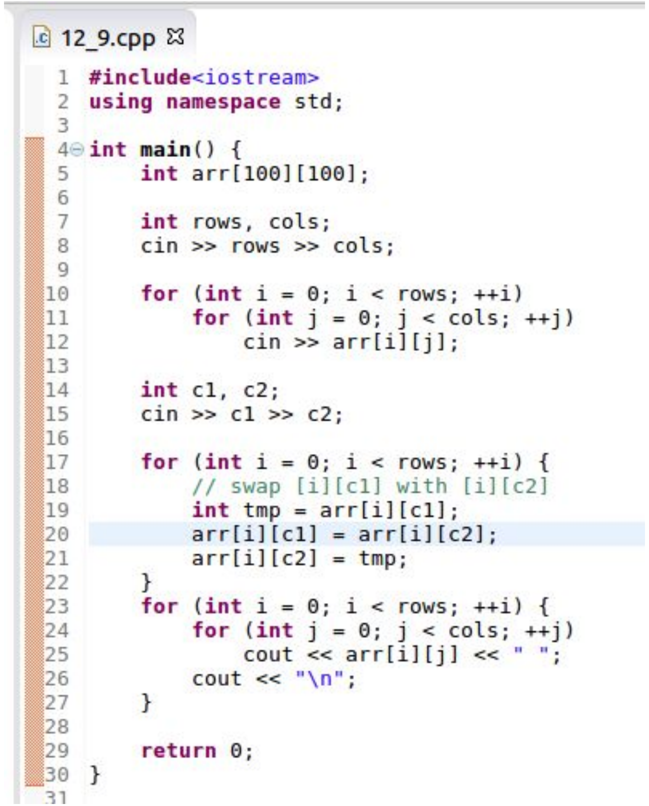


```cpp
#include<iostream>
using namespace std;

int main() {
    int arr[100][100];

    int rows, cols;
    cin >> rows >> cols;

    for (int i = 0; i < rows; ++i)
        for (int j = 0; j < cols; ++j)
            cin >> arr[i][j];

    int i = 0, j = 0;

    int left_diagonal = 0;
    while (i < rows && j < cols)
        left_diagonal += arr[i++][j++];

    int right_diagonal = 0;
    i = 0, j = cols-1;
    while (i < rows && j >= 0)
        right_diagonal += arr[i++][j--];

    int last_row = 0;
    j = 0;
    while (j < cols)
        last_row += arr[rows-1][j++];

    int last_col = 0;
    i = 0;
    while (i < rows)
        last_col += arr[i++][cols-1];

    cout << left_diagonal << " " << right_diagonal << "\n";
    cout << last_row << " " << last_col << "\n";

    return 0;
}
```

Practice: Swap 2 columns

- Read integers N, M, then Read **matrix** NxM. Then read 2 indices of columns. Swap the 2 columns together. Print the new matrix.
- Input: 3 4
    - 8 16 9 52
    - 3 15 27 6
    - 14 25 2 10
    - **0 3**
- Output
    - 52 16 9 8
    - 6 15 27 3
    - 10 25 2 14

Practice: Swap 2 columns

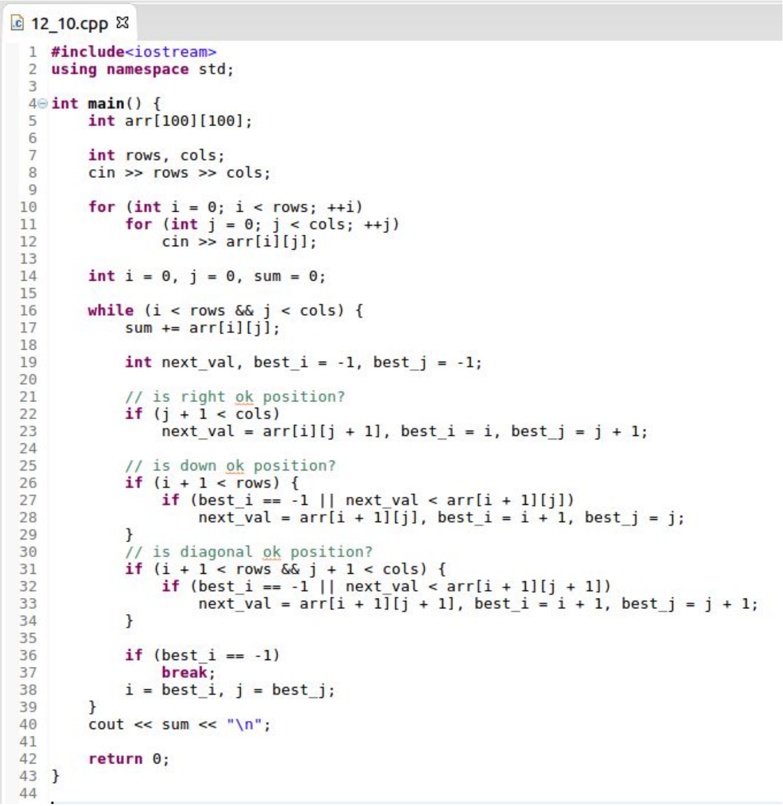


```cpp
#include<iostream>
using namespace std;

int main() {
    int arr[100][100];

    int rows, cols;
    cin >> rows >> cols;

    for (int i = 0; i < rows; ++i)
        for (int j = 0; j < cols; ++j)
            cin >> arr[i][j];

    int c1, c2;
    cin >> c1 >> c2;

    for (int i = 0; i < rows; ++i) {
        // swap [i][c1] with [i][c2]
        int tmp = arr[i][c1];
        arr[i][c1] = arr[i][c2];
        arr[i][c2] = tmp;
    }
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j)
            cout << arr[i][j] << " ";
        cout << "\n";
    }

    return 0;
}
```

Practice: Greedy Robot

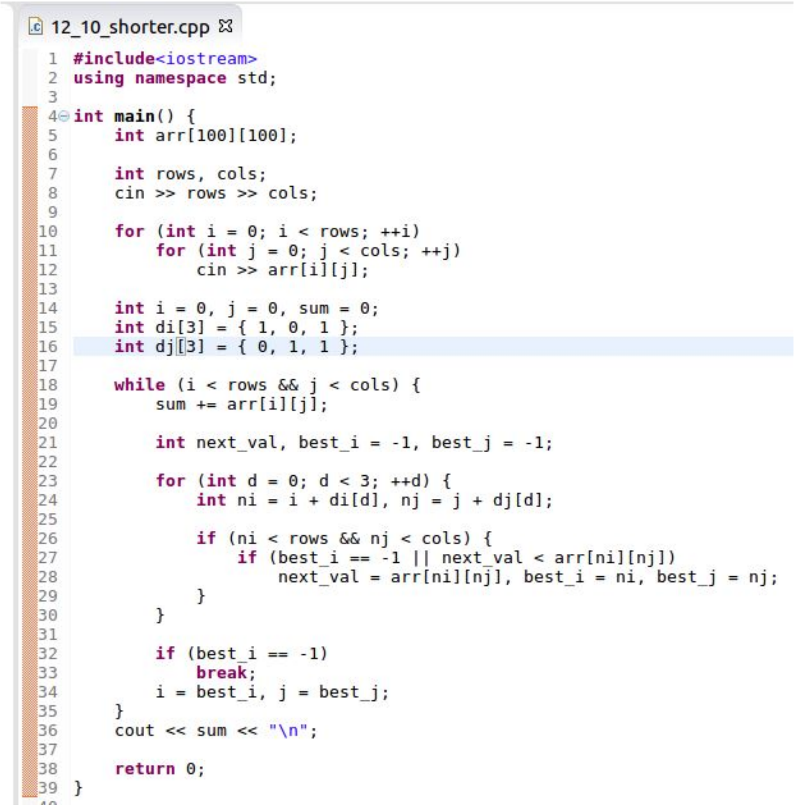
- Read integers N, M, then Read **matrix** NxM. All values are *distinct*. A robot starts at cell (0, 0). Take the value in the current cell and moves. It can move only one step to either: Right, Bottom or the diagonal. It always selects the cell that has maximum value. Print the total values the robot collects

Practice: Greedy Robot



```cpp
#include<iostream>
using namespace std;

int main() {
    int arr[100][100];

    int rows, cols;
    cin >> rows >> cols;

    for (int i = 0; i < rows; ++i)
        for (int j = 0; j < cols; ++j)
            cin >> arr[i][j];

    int i = 0, j = 0, sum = 0;

    while (i < rows && j < cols) {
        sum += arr[i][j];

        int next_val, best_i = -1, best_j = -1;

        // is right ok position?
        if (j + 1 < cols)
            next_val = arr[i][j + 1], best_i = i, best_j = j + 1;

        // is down ok position?
        if (i + 1 < rows) {
            if (best_i == -1 || next_val < arr[i + 1][j])
                next_val = arr[i + 1][j], best_i = i + 1, best_j = j;
        }
        // is diagonal ok position?
        if (i + 1 < rows && j + 1 < cols) {
            if (best_i == -1 || next_val < arr[i + 1][j + 1])
                next_val = arr[i + 1][j + 1], best_i = i + 1, best_j = j + 1;
        }

        if (best_i == -1)
            break;
        i = best_i, j = best_j;
    }
    cout << sum << "\n";

    return 0;
}
```

Practice: Greedy Robot - Shorter

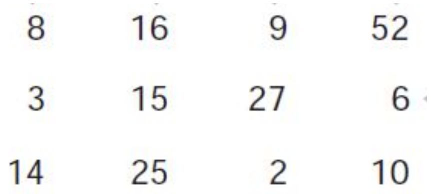


```cpp
#include<iostream>
using namespace std;

int main() {
    int arr[100][100];

    int rows, cols;
    cin >> rows >> cols;

    for (int i = 0; i < rows; ++i)
        for (int j = 0; j < cols; ++j)
            cin >> arr[i][j];

    int i = 0, j = 0, sum = 0;
    int di[3] = { 1, 0, 1 };
    int dj[3] = { 0, 1, 1 };

    while (i < rows && j < cols) {
        sum += arr[i][j];

        int next_val, best_i = -1, best_j = -1;

        for (int d = 0; d < 3; ++d) {
            int ni = i + di[d], nj = j + dj[d];

            if (ni < rows && nj < cols) {
                if (best_i == -1 || next_val < arr[ni][nj])
                    next_val = arr[ni][nj], best_i = ni, best_j = nj;
            }
        }

        if (best_i == -1)
            break;
        i = best_i, j = best_j;
    }
    cout << sum << "\n";

    return 0;
}
```

- In last code we tried 3 positions
  - (i+1, j), (i, j+1), (i+1)
  - The shift from (i, j) is
  - (1, 0), (0, 1), (1, 1)
- What if we coded the shifts in 2 arrays di, dj and used them
  - Then we stop all this copy/paste
- This is called **direction array**
  - Simple trick for cleaner code when u want to move to your **neighbours**

Practice: Flatten array

- Let Say we have matrix of ROWS x COLS
  - 1D here: 8 16 9 52 3 15 **27** 6 14 25 2 10
- To convert from (i, j) in matrix to 1D array
  - i * COLS + j
  - (1, 2) ⇒ 1 * 4 + 2 = 6
- To convert from index in 1D array to (i, j) in matrix
  - i = idx/COLS, j = idx%COLS
  - Idx = 6 ⇒ (6/4, 6%4) = (1, 2)
  - Why? Idx = i * COLS + j
    - Idx / COLS   = (i * COLS + j)/COLS   = i + 0, as j < COLS
    - Idx % COLS = (i * COLS + j)%COLS = 0 + j, as j < COLS and (i*COLS)%COLS = 0



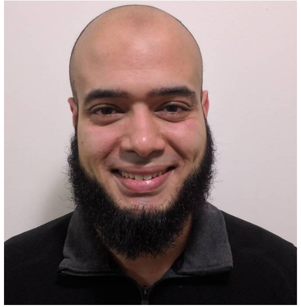
8	16	9	52
3	15	27	6
14	25	2	10

Programming4kids

2D Arrays

Mostafa Saad Ibrahim
Computer Vision Researcher @ Huawei Canada
PhD - Simon Fraser University
Bachelor / Msc - FCI Cairo University

Ex-(Software Engineer / Teaching Assistant)



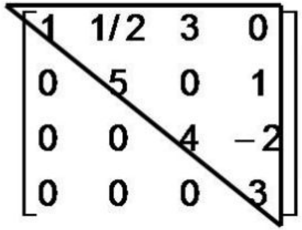
Homework 1: Smaller row?

- Read integers N, M, then Read **matrix** NxM. Then read Q, for q integers. Each query is 2 integers for 2 rows indices
- Compare the 2 rows and print **YES** if first row < 2nd one for all row values
- Input ⇒ Output
  - 3 4
  - 8 16 9 52
  - 3 15 27 6
  - 14 25 29 10
  - 3
  - 1 2 ⇒ NO
  - 2 3 ⇒ YES
  - 1 3 ⇒ NO

Homework 2: Triangular matrix

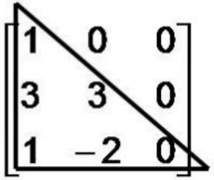
- Read integer N, then Read **Square** matrix NxN. Then, print 2 values. The sum of the **upper** triangle matrix and the **lower** triangle.
- Input
  - 3
  - 8 16 9
  - 3 15 27
  - 14 25 29
- Output
  - 94 (8+15+29+3+25+14)
  - 104 (8+15+29+16+27+9)

Upper triangular matrix



$$\begin{bmatrix} 1 & 1/2 & 3 & 0 \\ 0 & 5 & 0 & 1 \\ 0 & 0 & 4 & -2 \\ 0 & 0 & 0 & 3 \end{bmatrix}$$

Lower triangular matrix



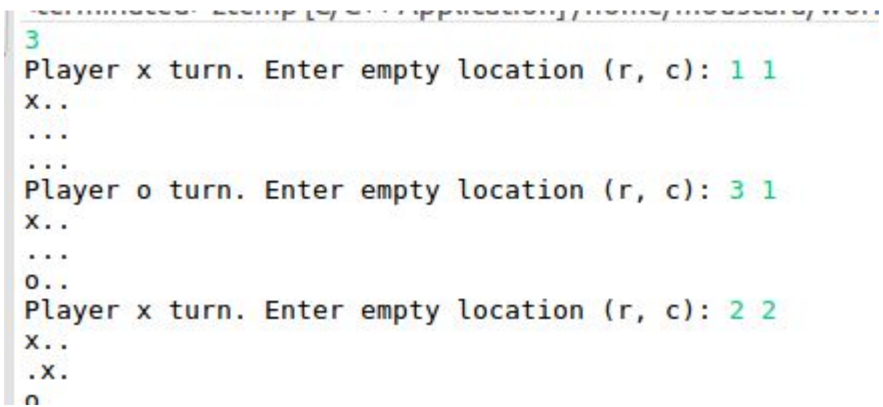
$$\begin{bmatrix} 1 & 0 & 0 \\ 3 & 3 & 0 \\ 1 & -2 & 0 \end{bmatrix}$$

Homework 3: Find mountains

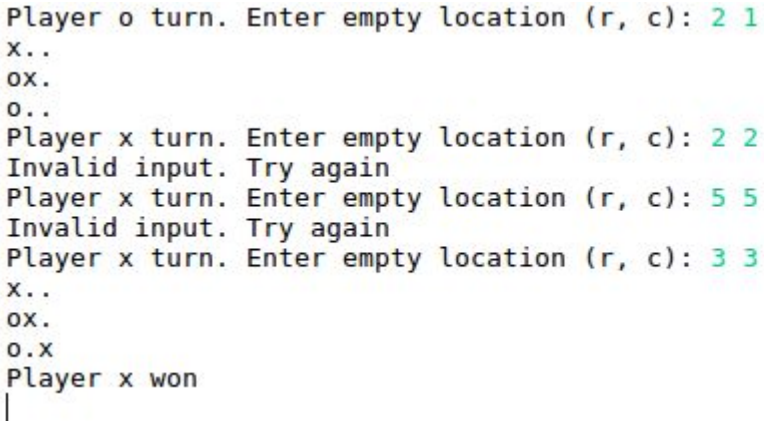
- Read integers N, M, then Read **matrix** NxM. Print all positions that are mountain. Position is mountain if its value > 8 neighbours values
- Input
  - 3 3
  - 8 6 1
  - 3 2 9
  - 1 6 4
- Output
  - 0 0 (8 > 6, 3, 2)
  - 1 2 (9 > 1, 2, 5, 4, 6)

Homework 4 : NxN tic-tac-toe

- Read integer N for the dimension of tic-tac-toe (3 <= N <= 9). Then run a game of 2 users who keep playing till one of them wins or tie. Print the grid after each round. Checkout below



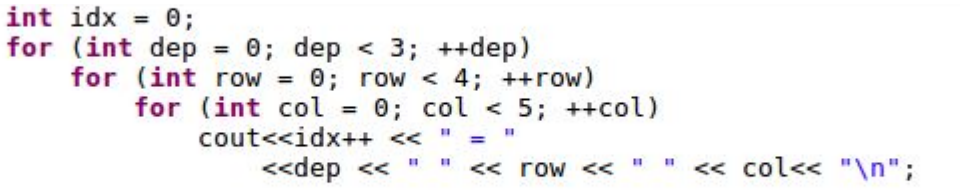
```
3
Player x turn. Enter empty location (r, c): 1 1
x..
...
...
Player o turn. Enter empty location (r, c): 3 1
x..
...
o..
Player x turn. Enter empty location (r, c): 2 2
x..
.x.
o
```



```
Player o turn. Enter empty location (r, c): 2 1
x..
ox.
o..
Player x turn. Enter empty location (r, c): 2 2
Invalid input. Try again
Player x turn. Enter empty location (r, c): 5 5
Invalid input. Try again
Player x turn. Enter empty location (r, c): 3 3
x..
ox.
o.x
Player x won
```

Homework 5: Flatten 3D Array

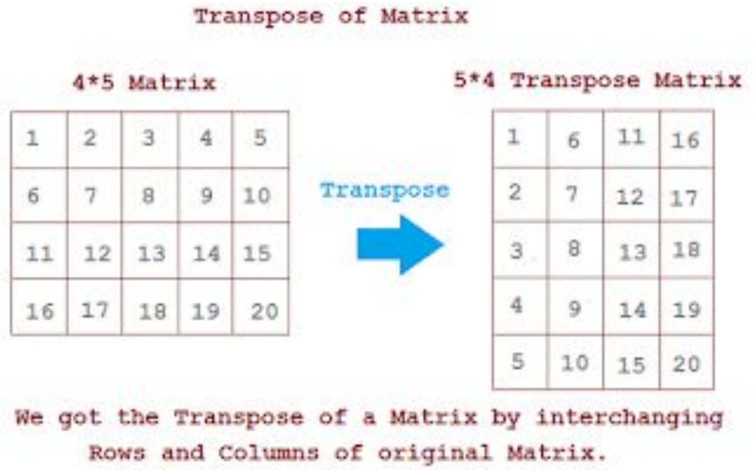
- Read 3 numbers: DEPTH, ROWS, COLS the dimensions of 3D array
- Then read integer either 1 (convert 3D to 1D) or 2 (1D to 3D)
- If input was 1, then read 3 integers d, r, c then convert to position in 1D array
- If input was 2, then read 1 integer position, then convert to 3D array position
- Try to generalize if we have e.g. 6D array
- Input ⇒ Outputs
    - 3 4 5   1   1 0 0   ⇒ 20
    - 3 4 5   2   20   ⇒ 1 0 0
    - 3 4 5   1   1 1 1   ⇒ 26
    - 3 4 5   1   2 3 2   ⇒ 57
    - 3 4 5   1   2 0 0   ⇒ 40
    - 3 4 5   2   59   ⇒ 2 3 4



```
int idx = 0;
for (int dep = 0; dep < 3; ++dep)
    for (int row = 0; row < 4; ++row)
        for (int col = 0; col < 5; ++col)
            cout<<idx++ << " = "
                <<dep << " " << row << " " << col<< "\n";
```

Homework 6: Transpose

- Read integers N, M, then Read **matrix** NxM. Compute another array, the transpose
- Input/output as in image



Transpose of Matrix

4*5 Matrix

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

Transpose

5*4 Transpose Matrix

1	6	11	16
2	7	12	17
3	8	13	18
4	9	14	19
5	10	15	20

We got the Transpose of a Matrix by interchanging Rows and Columns of original Matrix.

Homework 7: Active Robot

- Read integers N, M represents a matrix. A robot start at cell (0, 0). Read integer K, then K commands. Each command is 2 values
  - Direction from 1 to 4: up, right, down, left
  - Steps: a number to number steps to take in the direction. Steps [1, 10^10]
  - If the robot hits the wall during the move, it **circulates** in the matrix.
  - For every command, print where is the robot now
- Input
  - 3 4   4   **2 1**   **3 2**   **4 2**   **1 3**
    - 2 1 means to right 1 step - 3 2 means down 2 steps
- Output
  - (0, 1)   (2,1)   (2, 3)   (2, 3)

Homework 8: How many primes

- Read integers N, M, then Read **matrix** NxM. Then read integer Q, for Q queries. Each queries is a grid with **top left** (i, j) and # rows & # cols
  - So read 4 integers for i j r c
- For each query, print how many prime numbers in the requested grid.
- Input ⇒ Output
  - 3 4
  - 8 2 9 5
  - 3 2 27 6
  - 7 8 29 22
  - 2
  - 1 0 2 2 ⇒ 3 (primes 3, 2, 7 in rectangle (0, 1) (2, 1))
  - 0 1 2 3 ⇒ 3 (primes 2, 5, 2 in rectangle (0, 1) (1, 3))

تم بحمد الله

علمكم الله ما ينفعكم

ونفعكم بما تعلمتم

وزادكم علماً