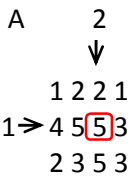
Level 2: Matrices

Introduction

When we go to a post office we see mail boxes lined up in rows and columns. Each mail box has mails in it and is referred with a number. The mail boxes can also be referred without using these numbers on them. If we refer to a row and a column it will correspond to a mail box. If we ask our friend to find the mail box at the 3rd row and the 5th column, he or she will easily locate it. Just like the mail boxes in a post office, a matrix is composed of boxes which are referred by the row and the column numbers.



The matrix A above has 3 rows and 4 columns, so it has 12 boxes with numbers in them. In the matrix above, the referred position is row 1, column 2.

Notice that just like as in arrays, the indices start from 0.

We can define a matrix as follows:

```
int myMatrix[3][4];
```

defines a matrix called 'myMatrix' with 3 rows and 4 columns. It is similar to the arrays; in addition matrices have two dimensional hence we need to add one more bracket when defining.

We can read from and write to the matrix locations in the same way we do in arrays. For instance,

```
cout << myMatrix[1][2] << endl;
```

outputs the value at position (1, 2) – row 1, column 2 – whereas,

```
myMatrix[1][2] = 100;
```

puts the value '100' in the same location.

Example: Read two integers *n* and *m* then a matrix of size (*n*, *m*) from the input file *matrix.txt*. Square the values in each position and write it to *square.txt*. The size of the matrix will not be more than (100, 100).

Sample:

matrix.txt:	*square.txt*:
3 4	1 4 4 1
1 2 2 1	16 25 25 9
4 5 5 3	4 9 25 9
2 3 5 3	

Solution:

```
#include <fstream>
using namespace std;

int main()
{
  ifstream fin("matrix.txt");
  ofstream fout("square.txt");
  int matrix[100][100];
  int n, m;

  fin >> n >> m;
  // read the matrix from the input file
  for (int row = 0; row < n; ++row)
    for (int col = 0; col < m; ++col)
      fin >> matrix[row][col];

  // write matrix to the output file
  for (int row = 0; row < n; ++row)
  {
    for (int col = 0; col < m; ++col)
      fout << matrix[row][col] * matrix[row][col] << " ";
    fout << endl;             // put enter at the end of each row
  }
}
```

Example: Read two integers *n* and *m* then a matrix of size (*n*, *m*) from the input file *matrix.txt*. Horizontally flip the matrix and write it to *flip.txt*. The size of the matrix will not be more than (100, 100).

Sample:

matrix.txt:	*flip.txt*:
3 4	1 2 2 1
1 2 2 1	3 5 5 4
4 5 5 3	3 5 3 2
2 3 5 3	

Solution:

```
#include <fstream>
using namespace std;

int main()
```

```
{
  ifstream fin("matrix.txt");
  ofstream fout("flip.txt");
  int matrix[100][100];
  int n, m;

  fin >> n >> m;
  // read the matrix from the input file
  for (int row = 0; row < n; ++row)
    for (int col = 0; col < m; ++col)
      fin >> matrix[row][col];

  // make flip operation
  for (int row = 0; row < n; ++row)
    for (int col = 0; col < m / 2; ++col)
    {
      int temp = matrix[row][col];
      matrix[row][col] = matrix[row][m - col - 1];
      matrix[row][m - col - 1] = temp;
    }

  // write matrix to the output file
  for (int row = 0; row < n; ++row)
  {
    for (int col = 0; col < m; ++col)
      fout << matrix[row][col] << " ";
    fout << endl;
  }
}
```

Exercise 1: Read two integers *n* and *m* then a matrix of size (*n*, *m*) from the input file *matrix.txt*. Vertically flip the matrix and write it to *flip.txt*. The size of the matrix will not be more than (100, 100).

Sample:

matrix.txt:	*flip.txt*:
3 4	2 3 5 3
1 2 2 1	4 5 5 3
4 5 5 3	1 2 2 1
2 3 5 3	

* *Exercise 2*: Read an integer *n* then a matrix of size (*n*, *n*) from the input file *matrix.txt*. Flip the matrix 90 degrees clockwise and write it to *flip.txt*. The size of the matrix will not be more than (100, 100).

Sample:

matrix.txt:	*flip.txt*:
4	9 2 4 1
1 2 2 1	8 3 5 2
4 5 5 3	7 5 5 2

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

Exercise 3: Read two integers *n* and *m* then two matrices of size (*n*, *m*) from the input file *matrix.txt*. Add the values at the same positions of these matrices, and write the summation matrix to *sum.txt*. The size of the matrices will not be more than (100, 100).

Sample:

matrix.txt:

```
3 4
1 2 2 1
4 5 5 3
2 3 5 3

9 8 7 6
1 0 4 -1
3 3 3 3
```

sum.txt:

```
10 10 9 7
5 5 9 2
5 6 8 6
```

Matrix Initialization

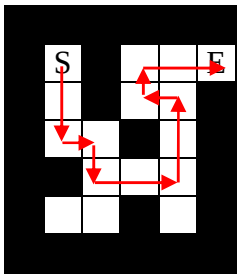
Matrices can be initialized while defining as in arrays:

```
int matrix[3][4] = {{1, 2, 2, 1}, {4, 5, 5, 3}, {2, 3, 5, 3}};
```

Matrices as Mazes and Graphs

Matrices can be used for different purposes than just numbers such as representing mazes and graphs.

* *Exercise 4*: You will be given a map of a maze and a path. Check if the path is a correct way to the exit. You will start at position (1, 1), and exit is at (1, *m*–1). An example maze of size (7, 6) is given below. Starting and exit locations are shown as 'S' and 'E'. A path is shown with red color to the exit.



Sample:

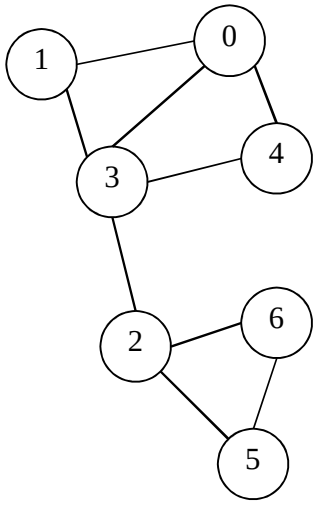
maze.txt:	*output*:
7 6 1 1 1 1 1 1 1 0 1 0 0 0 1 0 1 0 0 1 1 0 0 1 0 1 1 1 0 0 0 1 1 0 0 1 0 1 1 1 1 1 1 1 ddrdrruulurr	Right path!

You will first read the size of the maze (*n*, *m*). Then the maze will be given as a matrix size of (*n*, *m*); 1's are walls, 0's are empty locations. The size of the matrix will not be more than (100, 100). The path is given as a string composed of four different letters; 'l' means 'left', 'r' means 'right', 'u' means 'up', and 'd' means 'down'.

You will write '`Right path!`' if the path leads to the exit from the starting location, otherwise write '`Wrong path!`' on the screen.

The following exercise demonstrates how matrices can be used to represent graphs.

Exercise 5: You will be given a map of cities and roads connecting them. We know that there is at least one path from a city to any other city. Check if is it possible to travel passing all the roads just once with a car. An example map is given below:



Each city has a number on it and roads are always two-way as shown in the figure.

Sample:

map.txt:	*output*:
7 0 1 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 1 0 1 1 1 1 1 0 1 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 1 0	Can be toured!

You will first read the number of cities *n* then the map will be given as a matrix of size (*n*, *n*) from the input file *map.txt*. There will not be more than 50 cities. In the matrix, '`1`' indicates that there is a road between the cities of the corresponding row and column. For instance, position (0, 1) has '`1`' which means cities 0 and 1 have a road between them. Note that matrix is symmetric since roads are two-way. That is position (1, 0) has also '`1`'. On the map above we can find a tour as 0-1-3-4-0-3-2-5-6. If there is a tour output '`Can be toured!`' on the screen, otherwise '`Cannot be toured!`'.

Character Matrix

Remember that we defined a matrix as follows:

```
int myMatrix[20][10];
```

This is in fact an integer matrix, that is, it is composed of integer values. We can also define a matrix composed of characters as follows:

```
char myMatrix[20][10];
```

Example: In exercise 4, the maze was given as an integer matrix. For instance, we can more intuitively represent it in a character matrix; the input can be as follows:

Sample:

maze.txt:	*output*:
7 6 ###### #.#... #.#..# #..#.# ##...# #..#.# ###### ddrdrruulurr	Right path!

In this representation, '`#`' corresponds to walls and '`.`' corresponds to empty locations. We don't need to put space between characters. We can read the file almost the same way:

```
int n, m;
char maze[100][100];
string path;

fin >> n >> m;
// read the maze from the input file
for (int row = 0; row < n; ++row)
  for (int col = 0; col < m; ++col)
    fin >> maze[row][col];
fin >> path;
```

We can also use matrices to manipulate shapes. The following exercise demonstrates how matrices can be used to manipulate shapes.

* *Exercise 6*: write a program that reads a number $n \leq 20$ from the keyboard and prints the spiral of asterisks on the screen as shown below.

Sample:

n = 5

```
*****
*
* ***
*   *
*****
```

n = 8

```
********
*
* ******
* *    *
* *  * *
* **** *
*      *
********
```

Acknowledgement

Thanks to Dr. Omer Kilavuz for his help in writing this chapter.