Purpose: Passing and using 2D arrays in functions.

Problem 1: Declare a matrix (2D array of integers) of size ROWS by COLUMN using syntax given below; declare the ROWS and COLS in global space so that we can use them in different functions.

```
const int ROWS = 3;       //Declared in global space. All the functions can use this constant.

const int COLS = 4;       //Declared in global space. All the functions can use this constant.




int matrix[ROWS][COLS] = {0};//(Declared in main) to initialize all the elements to zero.
```

Now print the matrix in following format. Code to print the matrix is given below

```
0     0     0     0

0     0     0     0

0     0     0     0
```

Code to print the matrix:

```
for(int r = 0 ; r < ROWS ; r++)      // It will traverse all the rows

{

        for(int c = 0 ; c < COLS ; c++)     // It will traverse all the cells in per row

        {

                cout<<matrix[r][c]<<"\t"; // matrix[r][c] is to access element (r,c).

        }

        cout<<endl;

}
```

Run and check the output of your program.
Now update the initialization of your matrix such that it has following values in it. Code for this initialization is given below:

```
10     22     23     94

34     68     90      11

29     64     83      27
```

Code to initialize matrix to specific numbers:

```
const int ROWS = 3;
```

```
const int COLS = 4;



int matrix[ROWS][COLS] = {{10,22,23,94},{34,68,90,11},{29,64,83,27}};
```

Now run your program and see the result.

Problem 2: Write body of a function PrintArray below your main function. Following is the syntax for PrintArray function; notice that we are specifying the size of second dimension with the matrix.

```
void PrintArray( int myArray[][COLS] , int my2DArrayRows , int my2DArrayCols )

{

        for(int r = 0 ; r < my2DArrayRows ; r++)            // It will traverse all the rows

        {

                for(int c = 0 ; c < my2DArrayCols ; c++)  // It will traverse all the cells in per row

                {

                        cout<<myArray[r][c]<<"\t";          // matrix[r][c] is to access element (r,c).

                }

                cout<<endl;

        }

}
```

Write the prototype of PrintArray in the start of your cpp file (below “using namespace std;”). Following is the syntax:

```
void PrintArray( int [][COLS] , int , int );
```

Now update your main function such that instead of displaying the matrix from main it calls PrintArray function. Following is the syntax for function call:

```
PrintArray(matrix, ROWS, COLS);          // Calling PrintArray function with its parameters.
```

Problem 3: Write a function InputArray with following prototype:

```
void InputArray( int [][COLS] , int , int );
```

Update your main as code given below:

```
void main()

{



```

```
        int matrix[ROWS][COLS] = {{10,22,23,94},{34,68,90,11},{29,64,83,27}};



        InputArray(matrix, ROWS, COLS);

        PrintArray(matrix, ROWS, COLS);// Calling PrintArray function with its parameters.

}
```

Now run and test your program.

Problem 4: Write a function TestMatrixAddition with following prototype:

```
Void TestMatrixAddition();
```

Your TestMatrixAddition function should do following:

- Declare three matrices A, B and C and initialize them to zero.
- Input a matrix A from user (using InputMatrix function)
- Input a matrix B from user (using InputMatrix function)
- Call a function AddMatrix (prototype and description given below)
- Output matrix A, B and A+B

Sample Output is given below:

```
A =

5       3

8       1



B =

3       4

1       5



A + B =

8       7

9       6
```

AddMatrix takes three matrices A, B and Result as parameters, add the matrices and save the result in Result matrix. Remember two matrices having different sizes cannot be added so you will not call AddMatrix in this situation and print an error message. Prototype of AddMatrix is given below:

```
void AddMatrix( int A[][COLS] , int B[][COLS] ,int Result[][COLS] ,int , int );
```

Update your main as code given below and test your MatrixAddition.

```
void main()

{

        TestMatrixAddition();

}
```

For all the exercises below, you will use your InputArray and PrintArray functions for input and output, write a Test function and call the test function from main as we did in MatrixAddition exercise.

Problem 5: Trace of a matrix is sum of entities of its main diagonal if the matrix is square and undefined otherwise. Write a function which takes a matrix and returns its trace.

```
int TraceOfMatrix( int [][COLS] , int , int );
```

For example, if matrix A is given below (Main diagonal is highlighted in yellow):

-1	2	7	0
3	5	-8	4
1	2	7	-3
4	-2	1	0

Trace of A = -1+5+7+0 = 11.

Help**:** In your TestTraceOfMatrix function, call TraceOfMatrix if matrix is square and print error message otherwise.

```
//You need only single loop to traverse the diagonal

for(int i = 0 , i < ROWS-1 ; i++)

{

        cout<<matrix[i][i];  //matrix (0,0) , (1,1), (2,2) and so on

}
```

Problem 6: A square matrix in which all the entries below the main diagonal are zero is called upper triangular. Write a function IsMatrixUpperTriangular which takes a matrix and returns true if the matrix is upper triangular and false otherwise.

```
bool IsMatrixUpperTriangular( int [][COLS] , int , int );
```

For example matrix A shown below is upper triangular while matrix B is not upper triangular.

-1	2	7	0
0	5	0	-1
0	0	7	0
0	0	0	0

Matrix A

-1	2	7	0
0	5	0	-1
-9	0	7	0
0	0	0	0

Matrix B

Help**:** Notice the indices of blue cells which you need to traverse and device the nested loop accordingly.

(0,0)	2	7	0
(1,0)	(1,1)	0	-1
(2,0)	(2,1)	(2,2)	0
(3,0)	(3,1)	(3,2)	(3,3)

Problem 7: TicTacToe.cpp file contains a program which declares a TicTacToe board (a 2-D array of characters), takes input symbols (either 'X' or 'O') from user and prints the Tic-Tac-Toe board. A user 'O' wins Tic-Tac-Toe if any one of the three columns of the board has all 'O's in it (for now, we are not handling Rows and Diagonal check).

X X O X O O X O X	X X O X O O O X O	O X O X O O X O X
Result: X wins	Result: O wins	Result: No one wins

Write a function IsSymbolWinner which takes tic-tac-toe board and a symbol 'X' or 'O' from main and returns true if the player with that symbol wins or not.

```
Bool IsSymbolWinner( char [][COLS] , int , int, char symbol);
```

Update main such that after displaying the board, it announces the result (using your function).

Problem 8: Declare a friend list which will be containing names of your friends (2-D array of characters having names of 5 friends where each name can be at max 50 characters long). Following is the syntax:

```
const int FRIENDS_COUNT = 5;          // Total number of friends
```

```
const int NAME_LENGHT = 50;              // Each name can be 50 characters long at max.



char myFriendList[FRIENDS_COUNT][NAME_LENGHT] = {0};  // All the names initializes to null character.

// myFrientList is 5 by 50 array. First dimension of this list is your 5 friends and second dimension is name of each friend which can be 50 characters long at max.
```

Now read names of all your friends from console. Following is the syntax

```
for(int i=0 ; i<FRIENDS_COUNT ; i++)

{

        cout<<"Friend "<<i+1<<":\t";

        cin.getline(myFriendList[i],NAME_LENGHT);      //myFriendList[i] is the name of ith friend and its max length can be 50.

}
```

Now display your friend list using following syntax

```
for(int i=0 ; i<FRIENDS_COUNT ; i++)

{

        cout<<"Friend "<<i+1<<":\t";

        cout<<myFriendList[i]<<endl; //Display ith friends name which is a c-string.

}
```

Now you know how we can access name of ith friend (which is a c-string). Using your find string code which we did in CStrings exercises, update your program such that it takes a string from user and finds it in your friend list and displays all the friends who have that string in their name.
//myFriendList[i][j] gives jth character of ith friend's name
Sample input and output is shown below:

```
Friends Name:

Imran Khan

Javed Miandad

Wasim Akram

Aqib Javed

Saqlain Mushtaq
```

Enter friend which you want to search in list: Javed

Search Result:

Javed Miandad

Aqib Javed

Problem 9: If A is a square matrix, then the transpose of A, denoted by A^T, is defined to be the matrix that results from interchanging the rows and columns of A; that is, the first column of A^T is the first row of A, the second column of A^T is the second row of A, and so forth.

A =	A^T
1 -2 4 3 7 0 -5 8 6	1 3 -5 -2 7 8 4 0 6

(We are handling only square matrices).
Write a program TransposeMatrix which takes transpose of a matrix (in place i.e. without using any extra/temp matrix).
Hint**:** We do not need to work on diagonal elements. Traverse just like upper triangular matrix and swap Matrix(i,j) with Matrix(j,i).

Problem 10: Implement two different functions, one for inputting a 2D array and one for outputting.Test the two functions via menu. Make sure that the input and output take place such that each row of the 2D array occurs on a different line.

voidInputArray(bool arr[][COLSIZE],inttotalRows); //here COLSIZE is declared as a constant
voidOutputArray(bool arr[][COLSIZE],inttotalRows); //here COLSIZE is declared as a constant

Now set the COLSIZE to 60 and ROW to 20 and use the rand() function to input column entries into the matrix. The matrix entries can only be zero or one.
To output a matrix your OutputArray() function should first clear the screen and then display the matrix.

NOTE: If you do not want to fix the column size, you can allocate a matrix of MAXCOL and MAXROW and use only currentColSize and currentRowSize

Problem 11: For your game you need 2 different 2D arrays, one matrix to keep the current state (generation) of the game, and the other one to keep the updated state. After the updated state has

been computed, the previous state would be overwritten with the updated state and the new updated state will be calculated.
You have to write a function to copy a source array into the destination array,and a function that updates the matrix, with respect to the following rules.

1. Any live cell with less than two live neighbors dies, as if caused by under-population.
2. Any live cell with two or three live neighbors, lives on to the next generation.
3. Any live cell with more than three live neighbors dies, as if by overcrowding.
4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

The initial pattern,the one generated through the functionInputArrays(), constitutes the *seed* of the system. The first generation is created by applying the above rules simultaneously**** to every cell in the seed—births and deaths occur simultaneously****, and the discrete moment at which this happens is sometimes called a *tick* (in other words, each generation is a pure function of the preceding one). The rules continue to be applied repeatedly to create further generations.
*The ones and zeros in the matrix represent life and death, respectively.

void Copy2Darray(bool destination[][COLSIZE],bool source[][COLSIZE],inttotalRows); //here COLSIZE is declared a constant.

void Update2DArray(boolcurrentState[][COLSIZE],boolprevStat[][COLSIZE],inttotalRows); //here COLSIZE is declared a constant.

**Because the births and deaths that transform one generation to the next must all take effect simultaneously. Thus, when computing a new generation, new births anddeaths in that generation don't impact other births and deaths in that generation.To keep the two generations separate, you will need to work on two arrays—one for the current generation, and a second that allows you to compute and store the next generation without changing the current one.
You can use the Sleep() function between the transition of two generations (states) so that you can monitor the transformation.

Problem 12: Include the "mygraphics.h" file in your code at the beginning. You have to compile your code as:

```
g++ -o mycode.exe mygraphics.cpp mycode.cpp myconsole.cpp-lgdi32
```

Note: we have included the built in library gdi32.lib by using the -l switch.

Now change the color of the pixels at (2,0), (2,1)….(2,50) to red by using:

```
COLORREF redColor = RGB(255,0,0);
mySetPixel(x,y,redColor)  //set the values of x,y coordinates yourself.
```

Repeat the same for green, blue, magenta, violet and white colors.

Note: you can easily draw horizontal or vertical lines using these pixel level commands.