1. Write a program that detects whether a given 2x2 matrix is identity or not. You need to take the matrix as an input from the user.

$$\begin{array}{c|cc} & 0 & 1 \\ \hline 0 & 1 & 0 \\ 1 & 0 & 1 \end{array}$$

Identity matrix is a matrix that has 1 in its main diagonal and all the other entries are 0.

First, see that you need to focus on the following points:

1. Check main diagonal entries to see if they are 1 or not

$$\begin{array}{c|cc} & 0 & 1 \\ \hline 0 & \boxed{1} & 0 \\ 1 & 0 & \boxed{1} \end{array}$$

2. Check the entries other than main diagonal to see if they are 0 or not.

$$\begin{array}{c|cc} & 0 & 1 \\ \hline 0 & 1 & \boxed{0} \\ 1 & \boxed{0} & 1 \end{array}$$

3. Both 1 and 2 conditions must hold otherwise the matrix will not be an identity matrix.

You already know to iterate a 2D matrix you need two loops nested.

```
For(int j=0;j<2;j++)
{
        For(int j=0;j<2;j++)
        {
                Cout<<arr[i][j];
        }
}
```

This is the place where you need to insert a logic in which you will be checking diagonal or non-diagonal terms.

Next step is to analyze how you will get to know that the cells you are iterating are diagonal cells or non-diagonal cells.

Analyze the index to see what trend is being followed.

	0	1
0	Arr[0][0]	Arr[0][1]
1	Arr[1][0]	Arr[1][1]

Analyze the index of diagonal terms (shown in green above). All the diagonal terms have same integer as row and column index. i.e. [0][0] and [1][1]. If you draw another matrix as an example (e.g. 3x3) you will notice that the same trend is assumed. Thus in your nested loops, try to think whether you need to use arr[i][i] or arr[j][j]. Take a short example and dry run to see what seems correct (using i or j).

```
For(int j=0;j<2;j++)
{
        For(int j=0;j<2;j++)
        {
                If(arr[?][?]==1)
                        //do something
        }
}
```

For non-diagonal terms, the case is easy to handle. As you already have devised a way to check diagonal terms. The else part of the same condition is actually your non-diagonal terms. That is, if the cell is a diagonal term, then it will be handle by the above if statement. Otherwise, if this is not true then it is a *must thing* that the cell is a non-diagonal term. Because, any cell can either be a diagonal cell or a non-diagonal one.

To summarize, you can iterate all terms using arr[i][j].. in these terms, the diagonal terms can be identified by same row and column index.

2. Write a program that finds an equilibrium index of an array. An index of an array is said to be equilibrium index if the sum of the elements on its lower indexes is equal to the sum of elements at higher indexes. For Example:

A[] = {-7, 1, 5, 2, -4, 3, 0} Output : 2 is an equilibrium index, because: A[0] + A[1] + A[2] = A[4] + A[5] + A[6] = 1

First understand the problem statement. Equilibrium index can lie anywhere in your array (not necessarily at the mid only). E.g.

0	4	1	0	1	5	0
0	1	2	3	4	5	5

Left values sum:
0+4+1+0=5

Right values sum:
0+5=5

You need to design a logic in which you will be able to identify the cell at which following condition holds:

The sum of value of cells on left side = The sum of value of cells on right side

You already know that if you run one loop till the size of an array, you will be able to iterate through each cell.

0	4	1	0	1	5	0
0	1	2	3	4	5	6

```
for(int j=0;j<7;j++)//7 iterations
{
        cout<<arr[j];
}
```

But, in this task you need to perform three tasks in each iteration.

1. Find left value sum
2. Find right value sum
3. Compare to see if 1 and 2 are equal or not.

Lets now see how to perform 1 and 2:

0	4	1	0	1	5	0
0	1	2	3	4	5	6

The number of cells to consider while computing left and right sum in each iteration will not be same..The count will vary…e.g. consider first few steps of dry run..

	⬇					
0	4	1	0	1	5	0
0	1	2	3	4	5	6

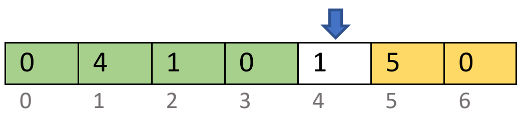
LEFT SUM	RIGHT SUM
0 Arr[0]	1+0+1+5+0=7 Arr[2]+Arr[3]…+Arr[6]

		⬇				
0	4	1	0	1	5	0
0	1	2	3	4	5	6

LEFT SUM	RIGHT SUM
0+4=4 Arr[0]+arr[1]	0+1+5+0=6 Arr[3]+Arr[4]…+Arr[6]

			⬇			
0	4	1	0	1	5	0
0	1	2	3	4	5	6

LEFT SUM	RIGHT SUM
0+4+1=5 Arr[0]+arr[1]+arr[2]	1+5+0=6 Arr[4]+Arr[5]+Arr[6]



LEFT SUM	RIGHT SUM
0+4+1+0=5 Arr[0]+Arr[1]+Arr[2]+Arr[3]	5+0=5 Arr[5]+Arr[6]

Stop here!
Equilibrium index found

Here, you can report value (1) or index (4)!

Analyze the indexes involved in left and right sum:

Iteration#	LEFT SUM	RIGHT SUM
1	0 Arr[0]	1+0+1+5+0=7 Arr[2]+Arr[3]+arr[4]+arr[5]+Arr[6]
2	0+4=4 Arr[0]+arr[1]	0+1+5+0=6 Arr[3]+Arr[4]+arr[5]+Arr[6]
3	0+4+1=5 Arr[0]+arr[1]+arr[2]	1+5+0=6 Arr[4]+Arr[5]+Arr[6]
4	0+4+1+0=5 Arr[0]+Arr[1]+Arr[2]+Arr[3]	5+0=5 Arr[5]+Arr[6]

In the first iteration, left index goes from 0 to 0. (1 value considered)

Right index goes from 2 to size i.e. 6. (5 values considered for sum)

In the second iteration, left index goes from 0 to 1 (2 values considered for sum)

Right index goes from 3 to size i.e. 6. (4 values considered for sum)

In the second iteration, left index goes from 0 to 2 (3 values considered for sum)

Right index goes from 4 to size i.e. 6. (3 values considered for sum)

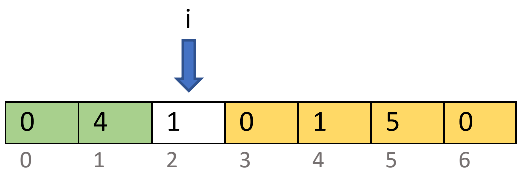
In the second iteration, left index goes from 0 to 3 (4 values considered for sum)

Right index goes from 5 to size i.e. 6. (2 values considered for sum)

That is, if we declare one variable to deal with left index, then its value will start from 0 and will go till size. But, for right index, the situation is different. Your left side is eventually getting increased whereas the right side will eventually shrink in size (w.r.t number of elements to consider for sum).

Hint: you might need two loops (nested), because at each iteration you further need a mechanism to iterate the right side cells and find their sum.

You need to devise a formula or equation to initialize and end your loops.



Left index moves from 0 to i
Right index moves from i+1 to size