Obviously when all numbers are declared as 1 gives 20/100 by mistake this result is found:

number = 1;//int.Parse(Console.ReadLine());

That means any number given “No” result is met just 2 times;

We can initiate 3 types of m3xs:

* Char [,] represented with 0 and 1:

for (int i = 0; i < 8; i++)

{

number = 1;//int.Parse(Console.ReadLine());// input

binaryStr = Convert.ToString(number, 2).PadLeft(8,'0');  
//convert to specifik number

int posFix = 7;

for (int j = 0; j < 8; j++)

{

char[] tempChar = binaryStr.ToCharArray();

binCharArr[i, posFix - j] = tempChar[j];  
// posFix-j fix position in the m3x

}

}

* Int[,] through bitwise operations :

for (int row = 0; row < 8; row++)

{

int number = int.Parse(Console.ReadLine());

for (int col = 0; col < 8; col++)

{ //matrix[row, col] = (number >> col) & 1=1;

matrix[row, col] = (number >> col) & 1;

}

}

* Bool[,]:

Test and logical problem Pillar can occupy col 7 or 0 that divides non existing to existing matrix and could produce result such as – 7 -0 or No (just these two are possible since we have another statement “Leftmost” or “leftmost among possable”)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |  |
| 0 |  |  |  |  |  |  |  |  | n0 = 0 |
| 1 |  | ■ |  |  |  |  |  |  | n1 = 64 |
| 2 |  |  |  |  |  |  |  |  | n2 = 0 |
| 3 |  |  |  |  | ■ |  |  |  | n3 = 8 |
| 4 |  |  |  |  |  |  |  |  | n4 = 0 |
| 5 |  |  |  |  | ■ | ■ |  |  | n5 = 12 |
| 6 | ■ | ■ | ■ |  |  |  |  |  | n6 = 224 |
| 7 |  |  |  |  |  |  |  |  | n7 = 0 |

Interesting Note is the way this matrix is represented and it is because of actual representation of the number for example 224 -> 1110000 which is related the way bits are extracted.