Print Decreasing

Easy

1. You are given a positive number n.
2. You are required to print the counting from n to 1.
3. You are required to not use any loops. Complete the body of print Decreasing function to achieve it.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is. Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

```
1 <= n <= 1000
```

Format

Input

A number n

Output

n
n - 1
n - 2
..
1

Example

Sample Input

```
5
```

Sample Output

```
5
4
3
2
1
```

```
// #include <iostream>

// using namespace std;

// int main(){
//      // cout<<"weeee"<<endl;
//      // cout<<"hell"<<10<<endl;
//      // cout<<"hell"<<20<<30<<endl;
//      // cout<<20<<30<<"hwll"<<endl;
//      // char s2[6]= {"hello"};
//      // cout<<s2<<endl;
//      // string s{"abcfffabc"};
//      // while (s.length()!=
0&&s.find("abc")<s.length()){
//      //     s.erase(s.find("abc"),3);
//      // }
//      // cout<<s<<endl;
//      // char sw;
//      // sw ='A'+'r'-'a';
//      // cout<<sw<<endl;
//      // string x{""};
//      // x= sw+"r";
//      // cout<<x<<endl;

//      int a{1};
//      int g{a};
//      a=4;
//      cout<<g<<endl;
//      string s{'d'};
//      cout<<s<<endl;

//      return 0;


// }
```

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

void printDecreasing(int n){
    // write your code here
    if(n==0){
        return;
    }
    cout<<n<<endl;
    printDecreasing(n-1);
    return;
}

int main(){
    int n;
    cin >> n;
    printDecreasing(n);
}
```

Print Increasing
Easy

1. You are given a positive number n.
2. You are required to print the counting from 1 to n.
3. You are required to not use any loops. Complete the body of print Increasing function to achieve it. Don't change the signature of the function.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is. Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

```
1 <= n <= 1000
```

Format

Input

A number n

Output

1
2
3
..
n

Example

Sample Input

```
5
```

Sample Output

```
1
2
3
4
5
```

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

void printIncreasing(int n){
    // write your code here
    if(n==0){
        return;
    }
    printIncreasing(n-1);
    cout<<n<<endl;
}


int main(){
    int n; cin>>n;
    printIncreasing(n);
}
```

Print Increasing Decreasing

Easy

1. You are given a positive number n.
2. You are required to print the counting from n to 1 and back to n again.
3. You are required to not use any loops. Complete the body of pdi function to achieve it. Don't change the signature of the function.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is.Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

```
1 <= n <= 1000
```

Format

Input

A number n

Output

n
n - 1
n - 2
..
1
1
2
3
..
n

Example

Sample Input

```
3
```

Sample Output

```
3
2
1
1
2
3
```

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

void printIncDec(int n){
    // write your code here
    if (n==0){
        return;
    }
    cout<<n<<endl;
    printIncDec(n-1);
    cout<<n<<endl;
}


int main(){
    int n; cin>>n;
    printIncDec(n);
}
```

Factorial

Easy

1. You are given a number n.
2. You are required to calculate the factorial of the number. Don't change the signature of factorial function.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is.Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

```
0 <= n <= 10
```

Format

Input

A number n

Output

factorial of n

Example

Sample Input

```
5
```

Sample Output

```
120
```

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

int factorial(int n){
    // write your code here
    if(n==1){
        return 1;
    }
    return n*factorial(n-1);

}


int main(){
    int n; cin>>n;
    cout<<factorial(n);
}
```

Power-linear

Easy

1. You are given a number x.
2. You are given another number n.
3. You are required to calculate x raised to the power n. Don't change the signature of power function .

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is. Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

```
1 <= x <= 10
0 <= n <= 9
```

Format

Input

A number x
A number n

Output

x raised to the power n

Example

Sample Input

```
2
5
```

Sample Output

```
32
```

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

int power(int x,int n){
   // write your code here
   if (n==0){
       return 1;
   }

   return power(x,n-1) * x;

}


int main(){
    int n,x; cin>>x>>n;
    cout<<power(x,n);
}
```

Power-logarithmic

Easy

1. You are given a number x.
2. You are given another number n.
3. You are required to calculate x raised to the power n. Don't change the signature of power function.

Note1 -> The previous version expects the call stack to be of n height. This function expects call function to be only log(n) high.

Note2 -> The online judge can't force you to write the function recursively but that is what the spirit of question is. Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

```
1 <= x <= 10
0 <= n <= 9
```

Format

Input

A number x

A number n

Output

x raised to the power n

Example

Sample Input

```
2
5
```

Sample Output

```
32
```

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

int powerLogarithmic(int x,int n){
    // write your code here
    if(n==0){
        return 1;
    }

    int pnbt = powerLogarithmic(x,n/2);//n/2 power of x

    int pn = pnbt*pnbt;

    if (n%2 == 1){     //if odd adding extra x because
        pn = pn*x;     // 9/2 is 4 not 4.5 here unless we made it
and
    }                  //here we didn't

    return pn;


}
```

```
int main(){
    int x,n; cin>>x>>n;
    cout<<powerLogarithmic(x,n);
}
```

Print Zigzag

Easy

1. Here are a few sets of inputs and outputs for your reference
Input1 -> 1
Output1 -> 1 1 1

Input2 -> 2
Output2 -> 2 1 1 1 2 1 1 1 2

Input2 -> 3
Output3 -> 3 2 1 1 1 2 1 1 1 2 3 2 1 1 1 2 1 1 1 2 3

2. Figure out the pattern and complete the recursive function pzz to achieve the above for any positive number n.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is.Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

```
1 <= n <= 10
```

Format

Input

A number n

Output

As discussed in point 1 of description

Example

Sample Input

```
3
```

Sample Output

```
3 2 1 1 1 2 1 1 1 2 3 2 1 1 1 2 1 1 1 2 3
```

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


void pzz(int n){
    // write your code here
    if(n==0){
        return;
    }
    cout<<n<<" ";
    pzz(n-1);
    cout<<n<<" ";
    pzz(n-1);
    cout<<n<<" ";


}



int main(){
    int n; cin>>n;
    pzz(n);
}
```

Tower Of Hanoi

Easy

Tower Of Hanoi
1. There are 3 towers. Tower 1 has n disks, where n is a positive number. Tower 2 and 3 are empty. 2. The disks are increasingly placed in terms of size such that the smallest disk is on top and largest disk is at bottom. 3. You are required to 3.1. Print the instructions to move the disks. 3.2. from tower 1 to tower 2 using tower 3 3.3. following the rules 3.3.1 move 1 disk at a time. 3.3.2 never place a smaller disk under a larger disk. 3.3.3 you can only move a disk at the top. Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is.Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

0 <= n <= 9 10 <= n1, n2, n3 <= 10^9 n1 != n2 != n3

Format

Input

A number n, representing number of disks A number n1, representing id of tower 1 A number n2, representing id of tower 2 A number n3, representing id of tower 3

Output

n[n1 -> n2] .. A set of instructions in above format to represent, move nth disc from n1 tower to n2 tower

Example

Sample Input

```
3
10
11
12
```

Sample Output

```
1[10 -> 11]
2[10 -> 12]
1[11 -> 12]
3[10 -> 11]
1[12 -> 10]
2[12 -> 11]
1[10 -> 11]
```

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

void toh(int n, int t1id, int t2id, int t3id){
    // write your code here

    if (n==0){
      return;
    }


    toh(n-1,t1id,t3id,t2id);      //"faith" will print instruction
to move
                                  // n-1 disk from t1id to t3id
following all the rules

    cout<<n<<"["<<t1id<<" -> "<<t2id<<"]"<<endl;        //print
instruction
                                                        // for
moving last disk from t1id to t2id

    toh(n-1,t3id,t2id,t1id);      //"faith" will print instruction
to move
                                  // n-1 disk from t3id to t2id
following all the rules
  }
```

```
int  main() {

    int n;cin>>n;
    int n1,n2,n3;cin>>n1>>n2>>n3;
    toh(n, n1, n2, n3);
  }
```

Display Array

Easy

1. You are given a number n, representing the size of array a.
2. You are given n numbers, representing elements of array a.
3. You are required to print the elements of array from beginning to end each in a separate line.
4. For the above purpose complete the body of displayArr function. Don't change the signature.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is.
Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

```
1 <= n <= 30
0 <= n1, n2, .. n elements <= 10
```

Format

Input

A number n
n1
n2
.. n number of elements

Output

n1
n2
.. n elements

Example

Sample Input

```
5
3
1
0
7
5
```

Sample Output

```
3
1
0
7
5
```

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

void display(int arr[], int idx, int n){
    // write your code here
    //my solution

    // if(n==0){
    //     return;
    // }
    // display(arr,0,n-1);
    // cout<<arr[n-1]<<endl;

    //another diff solution
    if (idx == n){
        return;
    }
    cout<<arr[idx]<<endl;
    display(arr, idx+1,n);

}

int main(){
    int n;
    cin >> n;
    int arr[n];
    for (int i = 0; i < n; i++)
        cin >> arr[i];
    display(arr, 0, n);
}
```

Display Array In Reverse

Easy

1. You are given a number n, representing the size of array a.
2. You are given n numbers, representing elements of array a.
3. You are required to print the elements of array from end to beginning each in a separate line.
4. For the above purpose complete the body of displayArrReverse function. Don't change the signature.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is.
Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

```
1 <= n <= 30
0 <= n1, n2, .. n elements <= 10
```

Format

Input

A number n
n1
n2
.. n number of elements

Output

n1
n2
.. n elements

Example

Sample Input

```
5
3
1
0
7
5
```

Sample Output

```
5
7
0
1
3
```

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

void display(int arr[], int idx, int n){
    // write your code here
    //my solution
    // if(n == 0){
    //     return;
    // }
    // cout<<arr[n-1]<<endl;
    // display (arr, 0,n-1);

    //another solution
    if(idx == n){
        return;
    }
    display(arr, idx+1,n);
    cout<<arr[idx]<<endl;
}

int main(){
    int n;
    cin >> n;
    int arr[n];
    for (int i = 0; i < n; i++)
        cin >> arr[i];
    display(arr, 0, n);
}
```

Max Of An Array

Easy

1. You are given a number n, representing the count of elements.
2. You are given n numbers.
3. You are required to find the maximum of input.
4. For the purpose complete the body of maxOfArray function. Don't change the signature.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is. Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

```
1 <= n <= 10^4
0 <= n1, n2, .. n elements <= 10 ^9
```

Format

Input

A number n
n1
n2
.. n number of elements

Output

A number representing max

Example

Sample Input

```
6
15
30
40
4
11
9
```

Sample Output

```
40
```

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

int max(int arr[], int idx, int n){
    // write your code here
    if(idx == n-1){
        return arr[idx];
    }
    int maxe = max(arr, idx+1, n);
    if(maxe < arr[idx]){
        maxe = arr[idx];
    }
    return maxe;
}

int main(){
    int n;
    cin >> n;
    int arr[n];
    for (int i = 0; i < n; i++)
        cin >> arr[i];
    int p = max(arr, 0, n);
    cout << p << endl;
}
```

First Index

Easy

1. You are given a number n, representing the count of elements.
2. You are given n numbers.
3. You are given a number x.
4. You are required to find the first index at which x occurs in array a.
5. If x exists in array, print the first index where it is found otherwise print -1.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is.
Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

```
1 <= n <= 10^4
0 <= n1, n2, .. n elements <= 10 ^ 3
0 <= x <= 10 ^ 3
```

Format

Input

A number n
n1
n2
.. n number of elements
A number x

Output

A number representing first index of occurence of x in array a or -1 if not found at all.

Example

Sample Input

```
6
15
11
40
4
4
9
4
```

Sample Output

```
3
```

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

int firstIndex(int arr[], int idx, int x, int n){
    // write your code here
    if(idx == n){
        return -1;
    }
    if(arr[idx] == x){
        return idx;
    }
    int ans_idx = firstIndex(arr, idx+1, x, n);

    return ans_idx;
}

int main()
{
    int n;
    cin >> n;
    int d;
    int arr[n];
    for (int i = 0; i < n; i++)
        cin >> arr[i];
    cin >> d;
    int p = firstIndex(arr, 0, d, n);
    cout << p << endl;
}
```

Last Index

Easy

1. You are given a number n, representing the count of elements.
2. You are given n numbers.
3. You are given a number x.
4. You are required to find the last index at which x occurs in array a.
5. If x exists in array, print the last index where it is found otherwise print -1.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is.
Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

```
1 <= n <= 10^4
0 <= n1, n2, .. n elements <= 10 ^ 3
0 <= x <= 10 ^ 3
```

Format

Input

A number n
n1
n2
.. n number of elements
A number x

Output

A number representing last index of occurence of x in array a or -1 if not found at all.

Example

Sample Input

```
6
15
```

```
11
40
4
4
9
4
```

Sample Output

```
4
```

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

int lastIndex(int arr[], int idx, int x, int n){
    // write your code here

    if(idx == n){
        return -1;
    }

    int ans_idx = lastIndex(arr,idx+1,x,n);

    if((ans_idx == -1)&&(arr[idx] == x )){
        return idx;
    }else{
        return ans_idx;
    }
}

int main(){
    int n;
    cin >> n;
    int d;
    int arr[n];
    for (int i = 0; i < n; i++)
        cin >> arr[i];
    cin >> d;
    int p = lastIndex(arr, 0, d, n);
    cout << p << endl;
}
```

All Indices Of Array

Easy

1. You are given a number n, representing the count of elements.
2. You are given n numbers.
3. You are given a number x.
4. You are required to return the all indices at which x occurs in array a.

5. Return an array of appropriate size which contains all indices at which x occurs in array
6. If no such element exist print "NO OUTPUT" a.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is. Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

1 <= n <= 10^4 0 <= n1, n2, .. n elements <= 10 ^ 3 0 <= x <= 10 ^ 3

Format

Input

A number n n1 n2 .. n number of elements A number x

Output

Return the array of indices from the allIndices function. Display is managed for you.

Example

Sample Input

```
6
15
11
40
4
4
9
4
```

Sample Output

```
3
4
```

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

vector<int> allIndex(vector<int>& arr, int idx, int data, int
count)
{

    if(idx == arr.size()){
      if(count == 0){
        cout<<"NO OUTPUT"<<endl;
      }
      vector <int> ans(count);
      return ans;
    }
    vector <int> indess;
    if(arr[idx] == data){
      indess = allIndex(arr, idx+1, data, count + 1);
      indess[count] = idx;
    }else{
      indess = allIndex(arr, idx+1, data, count);

    }

    return indess;

}

int main() {
  int n ;
  cin >> n;
  vector<int> arr;
  for (int i = 0; i < n; i++) {
    int d;
    cin >> d;
    arr.push_back(d);
  }
  int data;
  cin >> data;
  vector<int> ans = allIndex(arr, 0, data, 0);
  if (ans.size() == 0) {
    cout <<endl;
    return 0;
  }
  for (int ele : ans) cout << ele << endl;

  return 0;
}
```

Get Subsequence

Easy

1. You are given a string str. 2. Complete the body of getSS function - without changing signature - to calculate all subsequences of str. Use sample input and output to take idea about subsequences. Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is. Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

0 <= str.length <= 20

Format

Input

A string str

Output

Contents of the arraylist containing subsequences as shown in sample output

Example

Sample Input

```
abc
```

Sample Output

```
[, c, b, bc, a, ac, ab, abc]
```

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

vector<string> gss(string s){
    // write your code here
    if(s.length() == 0){
        vector<string> base;
        base.push_back("");
        return base;
    }
    char fc = s[0];
    string rss = s.substr(1,s.length()-1);
    vector <string> faith = gss(rss);

    vector <string> result ;
    for(auto s1 : faith) {
        result.push_back(""+s1);
    }
    for(auto s1 : faith) {
        result.push_back(fc+s1);
    }
    return result;
}

int main(){
    string s;
    cin >> s;
    vector<string> ans = gss(s);
    int cnt = 0;
    cout << "[";
    for (string str : ans){
        if (cnt != ans.size() - 1)
            cout << str << ", ";
        else
            cout << str;
        cnt++;
    }
    cout << "]";
}
```

Get Kpc

Easy

1. You are given a string str. The string str will contains numbers only, where each number stands for a key pressed on a mobile phone.
2. The following list is the key to characters map :
   0 -> .;
   1 -> abc
   2 -> def
   3 -> ghi
   4 -> jkl
   5 -> mno
   6 -> pqrs
   7 -> tu
   8 -> vwx
   9 -> yz
3. Complete the body of getKPC function - without changing signature - to get the list of all words that could be produced by the keys in str.
Use sample input and output to take idea about output.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is. Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

```
0 <= str.length <= 10
str contains numbers only
```

Format

Input

A string str

Output

Contents of the arraylist containing words as shown in sample output

Example

Sample Input

```
78
```

Sample Output

```
[tv, tw, tx, uv, uw, ux]
```

```
#include<iostream>
#include<string>
#include<vector>
using namespace std;
//edited by me at 28 dec
vector <string> kcm;
bool flag = false;

void fill_kcm (){
    if(flag){
        return ;
    }
    kcm.push_back(".;");   //0
    kcm.push_back("abc"); //1
    kcm.push_back("def"); //2
    kcm.push_back("ghi"); //3
    kcm.push_back("jkl"); //4
    kcm.push_back("mno"); //5
    kcm.push_back("pqrs");//6
    kcm.push_back("tu");  //7
    kcm.push_back("vwx"); //8
    kcm.push_back("yz");  //9
     flag = true;

 }


vector<string> getKPC(string s){
    //Write your code here
    fill_kcm();



    if(s.length() == 0){
        vector <string> base{""};
        return base;
    }
    char fc = s[0];
    string rss = s.substr(1,s.length()-1);
    vector<string> faith = getKPC(rss);

    vector<string> result ;
    for (int i{0}; i < ( (kcm[fc-'0']).length() ); i++){
        char cfc = ((kcm[fc - '0'])[i]);
        for(int j{};j<faith.size();j++){
            result.push_back(cfc+faith[j]);
        }
    }

    return result;

}


int main(){
    string s;
    cin >> s;
    vector<string> ans = getKPC(s);
    int cnt = 0;

    cout << "[";
    for (string str : ans){
        if (cnt != ans.size() - 1)
            cout << str << ", ";
        else
            cout << str;
        cnt++;
    }
    cout << "]";
}
```

Get Stair Paths

Easy

1. You are given a number n representing number of stairs in a staircase.
2. You are standing at the bottom of staircase. You are allowed to climb 1 step, 2 steps or 3 steps in one move.
3. Complete the body of getStairPaths function - without changing signature - to get the list of all paths that can be used to climb the staircase up.

Use sample input and output to take idea about output.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is. Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

```
0 <= n <= 10
```

Format

Input

A number n

Output

Contents of the arraylist containing paths as shown in sample output

Example

Sample Input

```
3
```

Sample Output

```
[111, 12, 21, 3]
```

```
#include<iostream>
#include<vector>
#include<string>
using namespace std;

vector<string> get_stair_paths(int n){
    //Write your code here
//my solution
    if (n == 0){
        vector <string> base {""};
        return base;
    }

    // vector <vector<string>> vov ; //for storing required
faith(s)

    // for(int i{};i <= n-1; i++){
    //     vov.push_back(get_stair_paths(i));
    // }

    vector <string> result;
    char to_add{'1'};

    for(int i{n-1};i>=0;i--){
        if(to_add >'3'){
            break;
        }
        vector <string> faith = get_stair_paths(i);
        for(auto s:faith){
            result.push_back(to_add+s);
        }
        to_add++;
    }




   //sir solution
    // if (n == 0){
    //     vector <string> base {""};
    //     return base;
    // }else if(n<0){
    //     vector <string> base;
    //     return base;
    // }
    // vector <string> path1 = get_stair_paths(n-1);
    // vector <string> path2 = get_stair_paths(n-2);
    // vector <string> path3 = get_stair_paths(n-3);

    // vector <string> result;
    // for(auto s:path1){
    //     result.push_back('1'+s);
    // }
    // for(auto s:path2){
    //     result.push_back('2'+s);
    // }
    // for(auto s:path3){
    //     result.push_back('3'+s);
    // }

    return result;

}

int main(){
    int n;
    cin >> n;
    vector<string> ans = get_stair_paths(n);
    int cnt = 0;

    cout << '[';
    for (string str : ans){
        if (cnt != ans.size() - 1)
            cout << str << ", ";
        else
            cout << str;
        cnt++;
    }
    cout << ']';
}
```

Get Maze Paths
Easy

1. You are given a number n and a number m representing number of rows and columns in a maze.
2. You are standing in the top-left corner and have to reach the bottom-right corner. Only two moves are allowed 'h' (1-step horizontal) and 'v' (1-step vertical).
3. Complete the body of getMazePath function - without changing signature - to get the list of all paths that can be used to move from top-left to bottom-right.
Use sample input and output to take idea about output.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is. Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

```
0 <= n <= 10
0 <= m <= 10
```

Format

Input

A number n

A number m

Output

Contents of the arraylist containing paths as shown in sample output

Example

Sample Input

```
3
3
```

Sample Output

```
[hhvv, hvhv, hvvh, vhhv, vhvh, vvhh]
```

```
#include<iostream>
#include<vector>

using namespace std;

// sr - source row
// sc - source column
// dr - destination row
// dc - destination column
vector <string> getMazePaths(int sr, int sc, int dr, int dc) {
    //sir solution
    if((sr == dr)&&(sc == dc)){
        vector <string> base{""};
        return base ;
    }

    //Paths After one steps horizontal
    vector <string> pah;
    if(sc < dc){
        pah = getMazePaths(sr,sc+1,dr,dc);

    }
    //Paths After one steps vertical
    vector <string> pav;
    if(sr < dr){
        pav = getMazePaths(sr+1,sc,dr,dc);
    }

    vector <string> result ;

    for(auto s: pah){
        result.push_back('h'+s);
    }
```

```cpp
    for(auto s: pav){
        result.push_back('v'+s);
    }

    //my solution
    // if((sr == dr)&&(sc == dc)){
    //     vector <string> base{""};
    //     return base ;
    // }else if (sr > dr){
    //     vector <string> base;
    //     return base ;
    // }else if (sc > dc){
    //     vector <string> base;
    //     return base ;
    // }

    // //Paths After one steps horizontal
    // vector <string> pah = getMazePaths(sr,sc+1,dr,dc);

    // //Paths After one steps vertical
    // vector <string> pav = getMazePaths(sr+1,sc,dr,dc);

    // vector <string> result ;

    // for(auto s: pah){
    //     result.push_back('h'+s);
    // }

    // for(auto s: pav){
    //     result.push_back('v'+s);
    // }

    return result;
}

void display(vector<string>& arr){
    cout << "[";
    for(int i = 0;i < arr.size();i++){
        cout << arr[i];
        if(i < arr.size() -1) cout << ", ";
    }

    cout << "]"<<endl;
}

int main() {
    int n,m; cin >> n >> m;
    vector<string> ans = getMazePaths(0,0,n-1,m-1);
    display(ans);

    return 0;
}
```

Get Maze Path With Jumps

Easy

1. You are given a number n and a number m representing number of rows and columns in a maze.
2. You are standing in the top-left corner and have to reach the bottom-right corner.
3. In a single move you are allowed to jump 1 or more steps horizontally (as h1, h2, ..), or 1 or more steps vertically (as v1, v2, ..) or 1 or more steps diagonally (as d1, d2, ..).
4. Complete the body of getMazePath function - without changing signature - to get the list of all paths that can be used to move from top-left to bottom-right.
Use sample input and output to take idea about output.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is. Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

```
0 <= n <= 10
0 <= m <= 10
```

Format

Input

A number n
A number m

Output

Contents of the arraylist containing paths as shown in sample output

Example

Sample Input

```
2
2
```

Sample Output

```
[h1v1, v1h1, d1]
```

```
#include<iostream>
#include<vector>
#include<string>

using namespace std;

vector<string> get_maze_paths(int sr, int sc, int dr, int dc) {
    //Write your code here

    if((sr == dr)&&(sc == dc)){
        vector <string> base {""};
        return base;
    }

    vector <string> result;

    //horizontal


    int hor{dc-sc};//maximun length of step
```

```
for( int i {1}; i<=hor;i++){
    //after ith steps in horizontal direction
    vector <string> paih;
    paih = get_maze_paths(sr, sc+i, dr ,dc);

    for(auto s: paih){
        char temp_c = '0'+i;
        string temp_s = temp_c + s;
        result.push_back('h'+ temp_s);
    }

}


//vertical


int vor {dr-sr};//maximun length of step

for( int i {1}; i<= vor;i++){
    //after ith steps in vertical direction
    vector <string> paiv;
    paiv = get_maze_paths(sr+i, sc, dr ,dc);
    for(auto s: paiv){
        char temp_c = '0'+i;
        string temp_s = temp_c + s;
        result.push_back('v'+ temp_s);
    }

}

//diagonal


int dor1{dc-sc};
int dor2{dr-sr};
int dor = (dor1<dor2) ? dor1 : dor2;//maximun length of step

for( int i{1}; i <= dor; i++){
    //after ith steps in diagonal direction
    vector <string> paih;
    paih = get_maze_paths(sr+i, sc+i, dr ,dc);

    for(auto s: paih){
        char temp_c = '0'+i;
        string temp_s = temp_c + s;
        result.push_back('d'+ temp_s);
    }

}
```

```
    return result;
}

void display(vector<string>& arr){
    cout << "[";
    for(int i = 0;i < arr.size();i++){
        cout << arr[i];
        if(i < arr.size() -1) cout << ", ";
    }

    cout << "]"<<endl;
}


int main() {
    int n,m; cin >> n >> m;
    vector<string> ans = get_maze_paths(0,0,n-1,m-1);
    display(ans);

    return 0;
}
```

Print Subsequence

Easy

1. You are given a string str.
2. Complete the body of printSS function - without changing signature - to calculate and print all subsequences of str.
Use sample input and output to take idea about subsequences.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is. Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

```
0 <= str.length <= 7
```

Format

Input

A string str

Output

Subsequences of str in order hinted by Sample output

Example

Sample Input

```
yvTA
```

Sample Output

```
yvTA
yvT
yvA
yv
yTA
yT
yA
y
vTA
vT
vA
v
TA
T
A
```

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

void printSS(string ques, string ans){
    // write your code here
    if (ques.length() == 0){
        cout<<ans<<endl;
        return ;
    }

    char ch = ques[0];
    string roq = ques.substr(1,ques.length()-1);

    printSS(roq, ans+ch);
    printSS (roq, ans+"");
}

int main(){
    string str;
    cin >> str;
    printSS(str, "");
}
```

Print Kpc

Easy

1. You are given a string str. The string str will contains numbers only, where each number stands for a key pressed on a mobile phone. 2. The following list is the key to characters map 0 -> .; 1 -> abc 2 -> def 3 -> ghi 4 -> jkl 5 -> mno 6 -> pqrs 7 -> tu 8 -> vwx 9 -> yz 3. Complete the body of printKPC function - without changing signature - to print the list of all words that could be produced by the keys in str. Use sample input and output to take idea about output. Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is. Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

0 <= str.length <= 10 str contains numbers only

Format

Input

A string str

Output

Words that can be produced by pressed keys indictated by str in order hinted by Sample output

Example

Sample Input

```
78
```

Sample Output

```
tv
tw
tx
uv
uw
ux
```

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

string codes[] = {".;", "abc", "def", "ghi", "jkl", "mno", "pqrs",
"tu", "vwx", "yz"};

void printKPC(string ques, string asf){
    // write your code here
    if(ques.length() == 0){
        cout<<asf<<endl;
        return ;
    }

    char ch = ques[0];
    string roq = ques.substr(1,ques.length()-1);

    int chn = ch-'0';
    for(int i{}; i < codes[chn].length(); i++){
        char c = (codes[chn])[i];
        printKPC(roq, asf+c);
    }

    return ;

}

int main(){
    string str;
    cin >> str;
    printKPC(str, "");
}
```

Print Stair Paths

Easy

1. You are given a number n representing number of stairs in a staircase.
2. You are standing at the bottom of staircase. You are allowed to climb 1 step, 2 steps or 3 steps in one move.
3. Complete the body of printStairPaths function - without changing signature - to print the list of all paths that can be used to climb the staircase up.

Use sample input and output to take idea about output.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is. Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

```
0 <= n <= 10
```

Format

Input

A number n

Output

Print paths (one path in each line) in order hinted by Sample output

Example

Sample Input

```
3
```

Sample Output

```
111
12
21
3
```

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

void printStairPaths(int n, string psf){
    // write your code here

    //sir solution
    if(n == 0){
        cout <<psf<<endl;
        return;
    }else if (n < 0){
        return;
    }

    printStairPaths(n-1,psf+"1");
    printStairPaths(n-2,psf+"2");
    printStairPaths(n-3,psf+"3");

    //my solution
    // if(n == 0){
    //     cout <<psf<<endl;
    //     return;
    // }

    // int nol = n<3 ? n : 3;//number of loops
    // for(int i{1};i <= nol; i++){
    //     string e{""};
    //     char c  = '0'+i;
    //     string p = e + c;  //converting to string
    //     printStairPaths(n-i,psf+p);
    // }
    return ;
}

int main(){

    int n;
    cin >> n;
    printStairPaths(n, "");
}
```

Print Maze Paths

Easy

1. You are given a number n and a number m representing number of rows and columns in a maze.
2. You are standing in the top-left corner and have to reach the bottom-right corner. Only two moves are allowed 'h' (1-step horizontal) and 'v' (1-step vertical).
3. Complete the body of pri tMazePath function - without changing signature - to print the list of all paths that can be used to move from top-left to bottom-right.

Use sample input and output to take idea about output.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is. Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

```
0 <= n <= 10
0 <= m <= 10
```

Format

Input

A number n
A number m

Output

Print paths (one path in each line) in order hinted by Sample output

Example

Sample Input

```
2
2
```

Sample Output

```
hv
vh
```

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

void printMazePaths(int sr, int sc, int dr, int dc, string psf){
    // write your code here

    //call smart base case normal

    if((dr == sr) && (dc == sc)){
        cout<<psf<<endl;
        return ;
    }

    if(sc < dc){
        printMazePaths(sr, sc+1, dr, dc, psf + "h");
    }
    if(sr < dr){
        printMazePaths(sr+1, sc, dr, dc, psf + "v");
    }

    return ;
}

int main(){
    int n;
    int m;
    cin >> n >> m;
    printMazePaths(0, 0, n - 1, m - 1, "");
}
```

Print Maze Paths With Jumps
Easy

1. You are given a number n and a number m representing number of rows and columns in a maze.
2. You are standing in the top-left corner and have to reach the bottom-right corner.
3. In a single move you are allowed to jump 1 or more steps horizontally (as h1, h2, ..), or 1 or more steps vertically (as v1, v2, ..) or 1 or more steps diagonally (as d1, d2, ..).
4. Complete the body of printMazePath function - without changing signature - to print the list of all paths that can be used to move from top-left to bottom-right.
Use sample input and output to take idea about output.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is. Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

```
0 <= n <= 5
0 <= m <= 5
```

Format

Input

A number n
A number m

Output

Print paths (one path in each line) in order hinted by Sample output

Example

Sample Input

```
3
3
```

Sample Output

```
h1h1v1v1
h1h1v2
h1v1h1v1
h1v1v1h1
h1v1d1
h1v2h1
h1d1v1
h2v1v1
h2v2
v1h1h1v1
v1h1v1h1
v1h1d1
v1h2v1
v1v1h1h1
v1v1h2
v1d1h1
v2h1h1
v2h2
d1h1v1
d1v1h1
d1d1
d2
```

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


    void printMazePaths(int sr, int sc, int dr, int dc, string
psf) {
        // write your code here
        if((sr == dr) && (sc == dc)){
            cout <<psf<<endl;  //path so far
            return ;
        }

        int hmax = dc - sc;
        for(int i{1}; i <= hmax; i++){
            char c = '0'+i;
            printMazePaths(sr, sc+i, dr, dc, psf+"h"+c);
        }


        int vmax = dr - sr;
        for(int i{1}; i <= vmax; i++){
            char c = '0'+i;
            printMazePaths(sr+i, sc, dr, dc, psf+"v"+c);
        }

        int dmax = (dc - sc) < (dr-sr) ? (dc - sc) : (dr-sr);
        for(int i{1}; i <= dmax; i++){
            char c = '0'+i;
            printMazePaths(sr+i, sc+i, dr, dc, psf+"d"+c);
        }
        return;

    }

int main() {
        int n ;
        int m ;cin>>n>>m;
        printMazePaths(0, 0, n - 1, m - 1, "");
    }
```

Print Permutations

Easy

1. You are given a string str.
2. Complete the body of printPermutations function - without changing signature - to calculate and print all permutations of str.
Use sample input and output to take idea about permutations.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is.
Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

`0 <= str.length <= 7`

Format

Input

A string str

Output

Permutations of str in order hinted by Sample output

Example

Sample Input

```
abc
```

Sample Output

```
abc
acb
bac
bca
cab
cba
```

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

void printPermutations(string str, string asf){
    // write your code here
    if(str.length() == 0){
        cout<<asf<<endl;
        return;
    }

    for(int i{}; i < str.length(); i++){
        char c = str[i];
        string temp = str;
        string ros = temp.erase(i,1);
        printPermutations( ros , asf+c);
    }

    return ;
}

int main(){
    string str;
    cin>>str;
    printPermutations(str,"");

}
```

Print Encodings

Easy

1. You are given a string str of digits. (will never start with a 0)
2. You are required to encode the str as per following rules
   1 -> a
   2 -> b
   3 -> c
   ..
   25 -> y
   26 -> z
3. Complete the body of printEncodings function - without changing signature - to calculate and print all encodings of str.

Use the input-output below to get more understanding on what is required
123 -> abc, aw, lc
993 -> iic
013 -> Invalid input. A string starting with 0 will not be passed.
103 -> jc
303 -> No output possible. But such a string maybe passed. In this case print nothing.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is.
Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

```
0 <= str.length <= 10
```

Format

Input

A string str

Output

Permutations of str in order hinted by Sample output

Example

Sample Input

```
655196
```

Sample Output

```
feeaif
feesf
```

```
#include <iostream>
#include<string>
#include <sstream>
using namespace std;

//codes global
//char codes[]
{'0','a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','
p','q','r','s','t','u','v','w','x','y','z'};
string codes[]
{"0","a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","
p","q","r","s","t","u","v","w","x","y","z"};

void printEncoding(string str, string asf){
    // write your code here
    if(str.length() == 0) {
        cout << asf << endl;
        return;
    }
    //taking single number

    //taking option which is a string(numerical)
    //and converting to actual number
    string option1 = str.substr(0,1);
    stringstream oss1(option1);
    int x1{};
    oss1 >> x1;

    //taking remaning question(str)
    string rs1 = str.substr( 1, str.length() - 1 );
    if( (x1 > 0) && (x1 < 27) ) {
        printEncoding(rs1, asf + codes[x1]);
    }

    if(str.length() >= 2) {
        //taking two number

        //taking option which is a string(numerical)
        //and converting to actual number
        string option2 = str.substr(0,2);
        stringstream oss2(option2);
        int x2{};
        oss2 >> x2;

        //taking remaning question(str)
        string rs2 = str.substr( 2, str.length() - 2 );
        if( (x2 > 9) && (x2 < 27) ) {
            printEncoding(rs2, asf + codes[x2]);
        }
    }

    return;
}

int main(){
    string str;
    cin>>str;
    printEncoding(str,"");

}
```

Flood Fill

Easy

1. You are given a number n, representing the number of rows.
2. You are given a number m, representing the number of columns.
3. You are given n*m numbers, representing elements of 2d array a. The numbers can be 1 or 0 only.
4. You are standing in the top-left corner and have to reach the bottom-right corner.
Only four moves are allowed 't' (1-step up), 'l' (1-step left), 'd' (1-step down) 'r' (1-step right). You can only move to cells which have 0 value in them. You can't move out of the boundaries or in the cells which have value 1 in them (1 means obstacle)
5. Complete the body of floodfill function - without changing signature - to print all paths that can be used to move from top-left to bottom-right.

Note1 -> Please check the sample input and output for details
Note2 -> If all four moves are available make moves in the order 't', 'l', 'd' and 'r'

Constraints

```
1 <= n <= 10
1 <= m <= 10
e1, e2, .. n * m elements belongs to set (0, 1)
```

Format

Input

A number n
A number m
e11
e12..
e21
e22..
.. n * m number of elements

Output

trrrdlt
tlldrt
.. and so on

Example

Sample Input

```
3 3
0 0 0
1 0 1
0 0 0
```

Sample Output

```
rddr
```

```
#include<iostream>
#include<vector>

using namespace std;

void floodfill(vector<vector<int>> maze,int sr,int sc,string
asf,vector < vector < bool >> visited) {

    int rl = maze.size();
    int cl = maze[0].size();

    if (sr < 0 || sc < 0 || sr == rl || sc == cl|| maze[sr][sc] ==
1 || visited[sr][sc]  == true){
        return;
    }
    if ((sr==(rl-1))&&(sc==(cl-1))){
        cout<<asf<<endl;
        return;
    }
    visited [sr][sc] = true;
    floodfill(maze , sr-1,sc,asf +"t",visited);
    floodfill(maze , sr,sc-1,asf +"l",visited);
    floodfill(maze , sr+1,sc,asf +"d",visited);
    floodfill(maze , sr,sc+1,asf +"r",visited);
    visited [sr][sc] = false;



    return;
}

int main() {
    int n, m;
    cin >> n >> m;
    vector < vector < int >> arr(n, vector < int > (m));

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

    vector < vector < bool >> visited(n, vector < bool > (m));
    floodfill(arr ,0,0,"",visited);
    return 0;
}
```

Target Sum Subsets

Easy

1. You are given a number n, representing the count of elements.
2. You are given n numbers.
3. You are given a number "tar".
4. Complete the body of printTargetSumSubsets function - without changing signature - to calculate and print all subsets of given elements, the contents of which sum to "tar". Use sample input and output to get more idea.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is. Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

```
1 <= n <= 30
0 <= n1, n2, .. n elements <= 20
0 <= tar <= 50
```

Format

Input

Input Format
A number n
n1
n2
.. n number of elements
A number tar

Output

Comma separated elements of the subset, the contents of which add to "tar"
.. all such subsets, each in a single line (the elements of each subset should be comma separated)

Example

Sample Input

```
5
10
20
30
40
50
60
```

Sample Output

```
10, 20, 30, .
10, 50, .
20, 40, .
```

```
#include <iostream>
#include<string>
#include <vector>
//#include <sstream>
using namespace std;

    // set is the subset
    // sos is sum of subset
    // tar is target
```

```
  void printTargetSumSubsets(vector<int> arr, int idx, string set,
int sos, int tar){
    //write your code here
    if(idx == arr.size()){
      if(sos == tar){
        cout<<set+"."<<endl;
      }
      return;
    }

    // int fnum = arr[idx];
    // stringstream ss ;
    // ss << fnum ;

    // string fnumstr ;
    // ss >> fnumstr;
    if (sos>tar){
      return ;
    }
    string fnumstr  = to_string(arr[idx]);
    printTargetSumSubsets(arr,idx+1,set+fnumstr+",
",sos+arr[idx],tar);
    printTargetSumSubsets(arr,idx+1,set,sos,tar);

    return ;
}


int main(){
  int n{};
  cin>>n;

  vector<int> arr;

  int number {};
  for(int i{};i<n;i++) {
    cin>>number;
    arr.push_back(number);
  }

  int tar{};
  cin>>tar;

  printTargetSumSubsets(arr,0,"",0,tar);

  return 0;
}
```

N Queens
Easy

1. You are given a number n, the size of a chess board.
2. You are required to place n number of queens in the n * n cells of board such that no queen can kill another.
Note - Queens kill at distance in all 8 directions
3. Complete the body of printNQueens function - without changing signature - to calculate and print all safe configurations of n-queens. Use sample input and output to get more idea.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is.
Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

```
1 <= n <= 10
```

Format

Input

A number n

Output

Safe configurations of queens as suggested in sample output

Example

Sample Input

```
4
```

Sample Output

```
0-1, 1-3, 2-0, 3-2, .
0-2, 1-0, 2-3, 3-1, .
```

```
#include<bits/stdc++.h>
// #include<iostream>
// #include<vector>
// #include<string>

using namespace std;


bool ifplacingqueenissafehere(vector<vector<int>> chess,int
row,int col){

  for (int i{row-1} , j{col} ;i>=0;i--){
    if(chess[i][j] == 1){
      return false;
    }
  }

  for (int i{row-1},j{col-1};i>=0 && j>=0;i--,j--){
    if(chess[i][j] == 1){
      return false;
    }
  }

  for (int i{row-1},j{col+1};i>=0 && j<chess.size();i--,j++){
    if(chess[i][j] == 1){
```

```cpp
        return false;
      }
    }

    return true;
}
void printNQueens(vector<vector<int>> chess,string qsf,int row){
    //write your code here
    if ( row == chess.size()){
      cout << qsf + "." << endl;
      return ;
    }
//traversing through coloums
    for (int col {};col < chess[0].size();col++){
      if(ifplacingqueenissafehere(chess ,row ,col) == true){
        chess [row][col] = 1;
        printNQueens(chess , qsf + to_string(row) +
"-"+to_string(col)+", ",row+1);
        chess [row][col] = 0;
      }
    }
      return ;
}

int main(){
    int n;
    cin >> n;
    vector<vector<int>> chess(n , vector<int> (n));

    printNQueens(chess,"",0);

}
```

Knights Tour
Easy

1. You are given a number n, the size of a chess board.
2. You are given a row and a column, as a starting point for a knight piece.
3. You are required to generate the all moves of a knight starting in (row, col) such that knight visits all cells of the board exactly once.
4. Complete the body of printKnightsTour function - without changing signature - to calculate and print all configurations of the chess board representing the route of knight through the chess board. Use sample input and output to get more idea.

Note -> When moving from (r, c) to the possible 8 options give first precedence to (r - 2, c + 1) and move in clockwise manner to explore other options.
Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is. Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Constraints

```
n = 5
0 <= row < n
0 <= col < n
```

Format

Input

A number n
A number row
A number col

Output

All configurations of the chess board representing route of knights through the chess board starting in (row, col)
Use displayBoard function to print one configuration of the board.

Example

Sample Input

```
5
2
0
```

Sample Output

```
25 2 13 8 23
12 7 24 3 14
1 18 15 22 9
6 11 20 17 4
19 16 5 10 21

19 2 13 8 21
12 7 20 3 14
1 18 15 22 9
6 11 24 17 4
25 16 5 10 23

25 2 13 8 19
12 7 18 3 14
```

```
1 24 15 20 9
6 11 22 17 4
23 16 5 10 21

19 2 13 8 25
12 7 18 3 14
1 20 15 24 9
6 11 22 17 4
21 16 5 10 23

21 2 17 8 19
12 7 20 3 16
1 22 13 18 9
6 11 24 15 4
23 14 5 10 25

23 2 17 8 25
12 7 24 3 16
1 22 13 18 9
6 11 20 15 4
21 14 5 10 19

25 2 17 8 23
12 7 24 3 16
1 18 13 22 9
6 11 20 15 4
19 14 5 10 21

19 2 17 8 21
12 7 20 3 16
1 18 13 22 9
6 11 24 15 4
25 14 5 10 23

25 2 15 8 19
16 7 18 3 14
1 24 11 20 9
6 17 22 13 4
23 12 5 10 21

19 2 15 8 25
16 7 18 3 14
1 20 11 24 9
6 17 22 13 4
21 12 5 10 23

21 2 15 8 19
16 7 20 3 14
1 22 11 18 9
6 17 24 13 4
23 12 5 10 25
```

```
23 2 15 8 25
16 7 24 3 14
1 22 11 18 9
6 17 20 13 4
21 12 5 10 19

23 2 13 8 21
14 7 22 3 12
1 24 9 20 17
6 15 18 11 4
25 10 5 16 19

21 2 13 8 23
14 7 22 3 12
1 20 9 24 17
6 15 18 11 4
19 10 5 16 25

25 2 13 8 19
14 7 18 3 12
1 24 9 20 17
6 15 22 11 4
23 10 5 16 21

19 2 13 8 25
14 7 18 3 12
1 20 9 24 17
6 15 22 11 4
21 10 5 16 23

21 2 11 16 19
12 17 20 3 10
1 22 7 18 15
6 13 24 9 4
23 8 5 14 25

23 2 11 16 25
12 17 24 3 10
1 22 7 18 15
6 13 20 9 4
21 8 5 14 19

23 2 11 16 21
12 17 22 3 10
1 24 7 20 15
6 13 18 9 4
25 8 5 14 19

21 2 11 16 23
12 17 22 3 10
1 20 7 24 15
6 13 18 9 4
```

```
19 8 5 14 25

21 2 9 14 19
10 15 20 3 8
1 22 5 18 13
16 11 24 7 4
23 6 17 12 25

23 2 9 14 25
10 15 24 3 8
1 22 5 18 13
16 11 20 7 4
21 6 17 12 19

25 2 9 14 23
10 15 24 3 8
1 18 5 22 13
16 11 20 7 4
19 6 17 12 21

19 2 9 14 21
10 15 20 3 8
1 18 5 22 13
16 11 24 7 4
25 6 17 12 23

23 2 7 12 21
8 13 22 17 6
1 24 3 20 11
14 9 18 5 16
25 4 15 10 19

21 2 7 12 23
8 13 22 17 6
1 20 3 24 11
14 9 18 5 16
19 4 15 10 25

25 2 7 12 23
8 13 24 17 6
1 18 3 22 11
14 9 20 5 16
19 4 15 10 21

19 2 7 12 21
8 13 20 17 6
1 18 3 22 11
14 9 24 5 16
25 4 15 10 23

25 4 15 10 23
14 9 24 5 16
```

```
1 18 3 22 11
8 13 20 17 6
19 2 7 12 21
```

```
19 4 15 10 21
14 9 20 5 16
1 18 3 22 11
8 13 24 17 6
25 2 7 12 23
```

```
25 4 15 10 19
14 9 18 5 16
1 24 3 20 11
8 13 22 17 6
23 2 7 12 21
```

```
19 4 15 10 25
14 9 18 5 16
1 20 3 24 11
8 13 22 17 6
21 2 7 12 23
```

```
21 6 17 12 19
16 11 20 7 4
1 22 5 18 13
10 15 24 3 8
23 2 9 14 25
```

```
23 6 17 12 25
16 11 24 7 4
1 22 5 18 13
10 15 20 3 8
21 2 9 14 19
```

```
25 6 17 12 23
16 11 24 7 4
1 18 5 22 13
10 15 20 3 8
19 2 9 14 21
```

```
19 6 17 12 21
16 11 20 7 4
1 18 5 22 13
10 15 24 3 8
25 2 9 14 23
```

```
25 8 5 14 19
6 13 18 9 4
1 24 7 20 15
12 17 22 3 10
23 2 11 16 21
```

```
19 8 5 14 25
6 13 18 9 4
1 20 7 24 15
12 17 22 3 10
21 2 11 16 23

21 8 5 14 19
6 13 20 9 4
1 22 7 18 15
12 17 24 3 10
23 2 11 16 25

23 8 5 14 25
6 13 24 9 4
1 22 7 18 15
12 17 20 3 10
21 2 11 16 19

21 12 5 10 19
6 17 20 13 4
1 22 11 18 9
16 7 24 3 14
23 2 15 8 25

23 12 5 10 25
6 17 24 13 4
1 22 11 18 9
16 7 20 3 14
21 2 15 8 19

23 12 5 10 21
6 17 22 13 4
1 24 11 20 9
16 7 18 3 14
25 2 15 8 19

21 12 5 10 23
6 17 22 13 4
1 20 11 24 9
16 7 18 3 14
19 2 15 8 25

21 14 5 10 19
6 11 20 15 4
1 22 13 18 9
12 7 24 3 16
23 2 17 8 25

23 14 5 10 25
6 11 24 15 4
1 22 13 18 9
12 7 20 3 16
```

```
21 2 17 8 19

25 14 5 10 23
6 11 24 15 4
1 18 13 22 9
12 7 20 3 16
19 2 17 8 21

19 14 5 10 21
6 11 20 15 4
1 18 13 22 9
12 7 24 3 16
25 2 17 8 23

23 16 5 10 21
6 11 22 17 4
1 24 15 20 9
12 7 18 3 14
25 2 13 8 19

21 16 5 10 23
6 11 22 17 4
1 20 15 24 9
12 7 18 3 14
19 2 13 8 25

25 16 5 10 23
6 11 24 17 4
1 18 15 22 9
12 7 20 3 14
19 2 13 8 21

19 16 5 10 21
6 11 20 17 4
1 18 15 22 9
12 7 24 3 14
25 2 13 8 23

23 10 5 16 21
6 15 22 11 4
1 24 9 20 17
14 7 18 3 12
25 2 13 8 19

21 10 5 16 23
6 15 22 11 4
1 20 9 24 17
14 7 18 3 12
19 2 13 8 25

25 10 5 16 19
6 15 18 11 4
```

```
1 24 9 20 17
14 7 22 3 12
23 2 13 8 21

19 10 5 16 25
6 15 18 11 4
1 20 9 24 17
14 7 22 3 12
21 2 13 8 23
```

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

//function to display the 2-d array
void display(vector<vector<int>> &chess){
    for(int i=0;i<chess.size();i++){
        for(int j=0;j<chess.size();j++){
            cout << chess[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;
}

void printKnightsTour(vector<vector<int>> &chess,int n,int r,int
c,int upcomingMove){
  //write your code here
    if(r < 0 || c < 0 || r >= n || c >= n || chess[r][c] != 0 ){
        return;
    }

    if(upcomingMove == 25 ){
        chess[r][c] = upcomingMove;
        display(chess);
        chess[r][c] = 0;
        return;
    }

    chess[r][c] = upcomingMove;
        printKnightsTour(chess,n,r-2,c+1,upcomingMove + 1);
        printKnightsTour(chess,n,r-1,c+2,upcomingMove + 1);
        printKnightsTour(chess,n,r+1,c+2,upcomingMove + 1);
        printKnightsTour(chess,n,r+2,c+1,upcomingMove + 1);
        printKnightsTour(chess,n,r+2,c-1,upcomingMove + 1);
        printKnightsTour(chess,n,r+1,c-2,upcomingMove + 1);
        printKnightsTour(chess,n,r-1,c-2,upcomingMove + 1);
        printKnightsTour(chess,n,r-2,c-1,upcomingMove + 1);
        chess[r][c] = 0;
    return ;

}
```

```cpp
int main(){
    int n{};
    cin >>n;

    vector<vector <int> > chess (n,vector<int>(n));
    int r{};
    int c{};
    cin>>r;
    cin>>c;
    printKnightsTour(chess, n,r,c, 1);


    return 0;
}
```