1 write a program for summation of series 1+X+X^2+X^3+….

```
#include<iostream>
#include<math.h>
using namespace std;
int main()
{
   int x,n,i,sum=0;
   cout<<"Enter the Values of X"<<endl;
   cin>>x;
   cout<<"Enter the Values of N"<<endl;
   cin>>n;
   for(int i=0;i<=n;i++)
   {
      sum=sum+pow(x,i);
      cout<<sum<<" ";
   }
   cout<<endl;
   cout<<"The Sum of series "<<sum;
}
```

2 Write a program to find the prefix averages of the given array with time complexity O(n).

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

int main(){
   float n,a;
```

```
    float s=0;
    float avg;
    cin>>n;
    for(int i=0;i<n;i++){
        cin>>a;
        s=s+a;
        avg=s/(i+1);
        cout<<avg<<endl;
        avg=0;
    }

    return 0;
}
```

3 Write a program to find the prefix averages of the given array with time complexity O(n^2).

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

int main(){
    float n,a;
    float s=0,avg;

    cin>>n;

    for(int i=0;i<n;i++){
        cin>>a;
        s=s+a;
        avg=s/(i+1);
        cout<<avg<<endl;
```

```
    }

    return 0;
}
```

4 Problem Statement

Find the maximum element from the given array.

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

int main()
{
    int a[10],max,n,i;
    cout<<"Enter the Number of Elemenets in the Array"<<endl;
    cin>>n;
    cout<<"Enter elements"<<endl;
    for(i=0;i<n;i++)
    {
        cin>>a[i];
    }
    max=a[0];
    for(i=1;i<n;i++)
    {
        if(a[i]>max)
        {
            max=a[i];
        }
    }
    cout<<"The Maximum Number in the Array is "<<max;
```

}

5 Problem Statement

Write a program to insert an element in a given position of an array.

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

int main(){
   int n,pos,val,a[20];

   cin>>n;

   for(int i=0;i<=n-1;i++){
      cin>>a[i];
   }

   cin>>pos;
   pos=pos-1;
   cin>>val;

   for(int i=n;i>pos;i--){
      a[i]=a[i-1];
   }
   a[pos]=val;
```

```
    for(int i=0;i<=n;i++){
        cout<<a[i]<<" ";
    }


    return 0;
}
```

6 Problem Statement

WAP to insert N elements into a linked list and traverse the list.

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

int val,a[20],q[6],n;

int front=-1, rear=-1;

void insertNode(int ptr){
    if(front== -1){
        front=0;
        rear++;
        a[rear]=ptr;
    }else{
        rear++;
        a[rear]=ptr;
    }
}
```

```
int main(){

   cin>>n;

   for(int i=0;i<n;i++){
      cin>>val;
      insertNode(val);
   }
   cout<<"Linked list: ";

   for(int i=n-1;i>-1;i--){
      cout<<a[i]<<" ";
   }

   return 0;
}
```

7 Problem Statement

Implement a QUEUE using a linked list and perform 5 ENQUEUE operations, one DEQUEUE operation, and display the elements.

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

int q[16],front=-1,rear=-1,val;

void enQueue(int val){
   if(front==-1){
      front=0;
      rear++;
```

```
        q[rear]=val;
    }else{
        rear++;
        q[rear]=val;
    }
}

void deQueue(){
    cout<<"deleted from queue is : "<<q[front]<<endl;
    front++;
}

void travel(){
    cout<<"Queue elements are: ";
    for(int i=front;i<=rear;i++){
        cout<<q[i]<<" ";
    }

}

int main(){
    for(int i=0;i<5;i++){
        cin>>val;
        enQueue(val);
    }
    deQueue();
    travel();

    return 0;
}
```

8 Problem Statement

Implement STACK data structure with 5 PUSH operations, 3 POP operations and display the element using Arrays.

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

int s[10],t=-1,val;

void push(int val){
   t++;
   s[t]=val;
}

void pop(){
   cout<<"The popped element is "<<s[t]<<endl;
   t--;
}

void travel(){
   if(t>=0){
      cout<<"Stack elements are:";
      for(int i=t;i>=0;i--){
         cout<<s[i]<<" ";
      }
   }
}

int main(){
```

```
    for (int i=0;i<5;i++){
        cin>>val;
        push(val);
    }
    pop();
    pop();

    travel();

    return 0;
}
```

9 Problem Statement

WAP to implement the insertion sort.

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

int val,n;

void insertionsort(int *a){
    int i,j,t;
    for(int i=0;i<n;i++){
        t=a[i];
        j=i-1;
        while(j>=0 && a[j]>t){
            a[j+1]=a[j];
            j=j-1;
        }
```

```
        a[j+1]=t;
    }
}

int main(){

    cout<<"Enter the number of elements: ";
    cin>>n;

    int *a=(int*)malloc(n*(sizeof(int)));

    cout<<"Enter elements:"<<endl;
    cout<<"Array before Sorting: ";

    for(int i=0;i<n;i++){
        cin>>val;
        a[i]=val;
        cout<<val<<" ";
    }
    cout<<endl;

    insertionsort(a);

    cout<<"Array after Sorting: ";
    for(int i=0;i<n;i++){
        cout<<a[i]<<" ";
    }
    return 0;
}
```

10 Problem Statement

Write an algorithm to find the position of a specific value in an array using Linear Search.

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

int n,val,key;

int main(){
   cin>>n;
   int *a=(int*)malloc(n*(sizeof(int)));
   cout<<"enter the elements in the array"<<endl;
   for(int i=0;i<n;i++){
      cin>>val;
      a[i]=val;
   }

   cout<<"enter the seraching element in the array"<<endl;
   cin>>key;

   for(int i=0;i<n;i++){
      if(a[i]==key){
         cout<<key<<" is present at location:"<<i+1;
      }
   }

   return 0;
}
```

11 Implement Merge Sort

```
#include<iostream>
using namespace std;
void swapping(int &a, int &b) {
   int temp;
   temp = a;
   a = b;
   b = temp;
}
void display(int *array, int size) {
   for(int i = 0; i<size; i++)
      cout << array[i] << " ";
   cout << endl;
}
void merge(int *array, int l, int m, int r) {
   int i, j, k, nl, nr;
   nl = m-l+1; nr = r-m;
   int larr[nl], rarr[nr];

   for(i = 0; i<nl; i++)
      larr[i] = array[l+i];
   for(j = 0; j<nr; j++)
      rarr[j] = array[m+1+j];
   i = 0; j = 0; k = l;

   while(i < nl && j<nr) {
      if(larr[i] <= rarr[j]) {
         array[k] = larr[i];
         i++;
      }else{
```

```
      array[k] = rarr[j];
      j++;
    }
    k++;
  }
  while(i<nl) {
    array[k] = larr[i];
    i++; k++;
  }
  while(j<nr) {
    array[k] = rarr[j];
    j++; k++;
  }
}
void mergeSort(int *array, int l, int r) {
  int m;
  if(l < r) {
    int m = l+(r-l)/2;

    mergeSort(array, l, m);
    mergeSort(array, m+1, r);
    merge(array, l, m, r);
  }
}
int main() {
  int n;
  cin >> n;
  int arr[n];
  for(int i = 0; i<n; i++) {
    cin >> arr[i];
  }
```

```
  cout << "Array before Sorting: ";
  display(arr, n);
  mergeSort(arr, 0, n-1);
  cout << "Array after Sorting: ";
  display(arr, n);
}
```

12 Implement Quick Sort

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

void swap(int *a,int *b){
   int t =*a;
   *a=*b;
   *b=t;
}
void printarr(int arr[],int size){
   for(int i=0;i<size;i++){
      cout<<arr[i]<<" ";
   }
   cout<<endl;
}

int partition(int arr[],int l,int h){
   int pivot = arr[h];
   int i = (l-1);

   for(int j=l;j<h;j++){
      if(arr[j]<=pivot){
```

```
            i++;
            swap(&arr[i],&arr[j]);

        }
    }
    swap(&arr[i+1],&arr[h]);
    return (i+1);

}

void Quicksort(int arr[],int l ,int h){
    if(l<h){
    int pi=partition(arr , l , h);

    Quicksort(arr,l,pi-1);

    Quicksort(arr,pi+1,h);
    }
}
int main(){
    int n;
    cin>>n;
    int arr[n];
    for(int i=0 ;i<n;i++){
        cin>>arr[i];
    }
    int m= sizeof(arr)/sizeof(arr[0]);

    Quicksort(arr,0,m-1);
    printarr(arr,m);
}
```

13 Implement Dijkstra's single-source shortest-path algorithm.

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

int minimumDist(int dist[], bool Tset[])
{
	int min=INT_MAX,index;

	for(int i=0;i<6;i++)
	{
		if(Tset[i]==false && dist[i]<=min)
		{
			min=dist[i];
			index=i;
		}
	}
	return index;
}

void Dijkstra(int graph[6][6],int src)
{
	int dist[6];
	bool Tset[6];


	for(int i = 0; i<6; i++)
	{
		dist[i] = INT_MAX;
```

```
		Tset[i] = false;
	}

	dist[src] = 0;

	for(int i = 0; i<6; i++)
	{
		int m=minimumDist(dist,Tset);
		Tset[m]=true;
		for(int i = 0; i<6; i++)
		{

			if(!Tset[i] && graph[m][i] && dist[m]!=INT_MAX &&
dist[m]+graph[m][i]<dist[i])
				dist[i]=dist[m]+graph[m][i];
		}
	}
	cout<<"Distance from Source"<<endl;
	for(int i = 0; i<6; i++)
	{

		char str=65+i;
		cout<<dist[i]<<endl;
	}
}

int main()
{
	int graph[6][6];
		for(int i=0;i<6;i++)
   {
```

```
    for(int j=0;j<6;j++)
    {
       cin>>graph[i][j];
    }
  }
        Dijkstra(graph,0);
        return 0;
}
```

14 Implement all pairs Floyd-Warshal's shortest-path algorithms.

```
#include <iostream>
using namespace std;
void floyds(int b[5][5])
{
  int i, j, k;
  for (k = 0; k < 5; k++)
  {
    for (i = 0; i < 5; i++)
    {
      for (j = 0; j < 5; j++)
      {
        if ((b[i][k] * b[k][j] != 0) && (i != j))
        {
          if ((b[i][k] + b[k][j] < b[i][j]) || (b[i][j] == 0))
          {
            b[i][j] = b[i][k] + b[k][j];
          }
        }
      }
    }
```

```
    }
    cout<<"All pairs shortest path"<<endl;
    for (i = 0; i < 5; i++)
    {
        for (j = 0; j < 5; j++)
        {
            cout<<b[i][j]<<" ";
        }
        cout<<endl;

    }
}
int main()
{
    int b[5][5];
    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j < 5; j++)
        {
            cin>>b[i][j];
        }
    }
    floyds(b);
}
```

15 Implement 0/1 Knapsack problem

```
#include<iostream>
#include<cmath>
using namespace std;
int knapsack01(int W,int N,int v[],int w[]) {
   int DP[N+1][W+1];
   for(int i=0;i<N+1;i++) DP[i][0] = 0;
   for(int i=0;i<W+1;i++) DP[0][i] = 0;

   for(int i=1;i<N+1;i++){
      for(int j=1;j<W+1;j++){
         if(w[i-1] <= j){
            DP[i][j] = max(v[i-1]+DP[i-1][j-w[i-1]],DP[i-1][j]);
         }
         else{
                                 DP[i][j] = DP[i-1][j];
         }
      }
   }
   return DP[N][W];
}
int main()
{
           int N;
           cin>>N;
           int w[N],v[N];
           for(int i=0;i<N;i++)
           {
                    cin>>w[i]>>v[i];
           }
```

```
        int W;
        cin>>W;
        int profit=knapsack01(W,N,v,w);
        cout<<"The maximum value of items that can be put into knapsack is ";
        cout<<profit;
}
```

16 A dog has to move from starting point to it's end point where 1 represent free space and 0 represented blocked area.Print the path from 0,0 to n-1, n-1 if exists or print -1;

Gray blocks are blocked so that dog cant move to that area.

the above image is the solution or way from staring point to end point.

```
#include<iostream>
using namespace std;
bool isSafe(int r,int c,int n,int **maze)
{
   if( r < 0 || r >= n )
   return false;
   if( c < 0 || c >= n )
   return false;
   if(maze[r][c])
   return true;
   return false;
```

```
}
bool solvemaze(int ** maze,int i,int j,int ** soln,int n)
{
   if(i == n-1 && j == n-1)
   {
      soln[i][j]=1;
      return true;
   }
   if(isSafe(i,j,n,maze))
   {
      soln[i][j]=1;
      if(solvemaze(maze,i+1,j,soln,n))
      return true;
      if(solvemaze(maze,i,j+1,soln,n))
      return true;
      soln[i][j]=0;
   }
   return false;
}
int main()
{
   int n;
   cin>>n;
   int** maze = new int*[n];
   for(int i = 0; i < n; ++i)
   maze[i] = new int[n];
   int** soln = new int*[n];
   for(int i = 0; i < n; ++i)
   soln[i] = new int[n];
   for(int i=0;i<n;i++)
   {
```

```
        for(int j=0;j<n;j++)
        {
            cin>>maze[i][j];
            soln[i][j]=0;
        }
    }
    if(solvemaze(maze,0,0,soln,n))
    {
        for(int i=0;i<n;i++)
        {
            for(int j=0;j<n;j++)
            cout<<soln[i][j]<<" ";
            cout<<endl;
        }
    }
    else
    {
        cout<<"-1";
    }
}
```

17 In chess, each step of a knight comprises stepping by two squares horizontally and one square vertically, or by one square horizontally and two squares vertically. A knight making one step from location (0,0) of an infinite chess board would finish up at one of the following eight locations: (1,2), (−1,2), (1,−2), (−1,−2), (2,1), (−2,1), (2,−1), (−2,−1).

Beginning from location (0,0), what is the minimum number of steps needed for a knight to get to some other arbitrary location (x,y)?

```
#include <bits/stdc++.h>
```

```
using namespace std;
#define ll long long int
ll knight_move(ll x, ll y)
{
    ll a, b, z, c, d;
    x = abs(x), y = abs(y);
    if (x < y) a = x, x = y, y = a;
    if (x == 2 && y == 2) return 4;
    if (x == 1 && y == 0) return 3;
    if (y == 0 || (y << 1) < x){
        c = y & 1;
        a = x - (c << 1), b = a & 3;
        return ((a - b) >> 1) + b + c;
    }
    else
    {
        d = x - ((x - y) >> 1);
        z = ((d % 3) != 0), c = (x - y) & 1;
        return ((d / 3) * 2) + c + (z * 2 * (1 - c));
    }
}
int main()
{
    string str;
    while(getline(cin,str))
    {
        if(str == "END") break;
        ll a,b;
        stringstream IN(str);
        IN >> a >> b;
        cout<< knight_move(a,b)<<endl;
```

```
    }
}
```

18 Implement Graph using Adjacency List. Given the number of vertices (V) and the edges, create an undirected graph using an adjacency list. Print the adjacency list of each vertex.

```
#include <bits/stdc++.h>
using namespace std;
void addEdge(vector<int> adj[], int s, int d)
{
    adj[s].push_back(d);
    adj[d].push_back(s);
}
void printGraph(vector<int> adj[], int V)
{
    for (int d = 0; d < V; ++d)
    {
        cout << "Adjacency list of vertex "
             << d<<endl;
        cout<<"head ";
    for (auto x : adj[d])
        cout << "-> " << x;
        cout<<endl;
    }
}
int main()
{
    int u,v,V;
    cin>>V;
    int E;
```

```
    cin>>E;
    vector<int> adj[V];
    for(int i=0;i<E;i++)
    {
        cin>>u>>v;
        addEdge(adj, u, v);
    }
    printGraph(adj, V);
}
```

19 Write a program to perform a breadth-first search in a graph.

```
#include<iostream>
using namespace std;
struct queue
{
    int size;
    int f;
    int r;
    int *arr;
};
int isEmpty(struct queue *q)
{
    if(q->r == q->f)
    {
        return 1;
    }
    return 0;
}
```

```
int isFull(struct queue *q)
{
   if(q->r == q->size - 1)
   {
      return 1;
   }
   return 0;
}
void enqueue(struct queue *q, int val)
{
   if(isFull(q))
   {
      printf("The Queue is Full.");
   }
   else
   {
      q->r++;
      q->arr[q->r] = val;
   }
}
int dequeue(struct queue *q)
{
   int a = -1;
   if(isEmpty(q))
   {
      printf("Queue is Empty.\n");
   }
   else
   {
      q->f++;
      a = q->arr[q->f];
```

```
    }
    return a;
}
int main()
{
    struct queue q;
    q.size = 400;
    q.f = q.r = 0;
    q.arr = (int *)malloc(q.size*sizeof(int));
    int node;
    int i;
    int n;
    cin>>n;
    int a[n][n];
    int visited[n];
    for(i=0;i<n;i++)
    {
        visited[i]=0;
    }
    int k1=0,k2=0;
    for(i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
            a[i][j]=0;
        }
    }
    while(k1!=-1 && k2!=-1)
    {
        cin>>k1>>k2;
        if(k1!=-1 && k2!=-1)
```

```
    {
        a[k1][k2]=1;
    }
  }
  int key1;
  cin>>key1;
  visited[key1] = 1;
  enqueue(&q, key1);
  while(!isEmpty(&q))
  {
    int node = dequeue(&q);
    for(int j = 0; j<n; j++)
    {
      if(a[node][j] == 1 && visited[j] == 0)
      {
        visited[j] = 1;
        enqueue(&q, j);
      }
    }
    cout<<node<<" ";
  }
   return 0;
}
```

20 There are n cities and there are roads in between some of the cities. Somehow all the roads are damaged simultaneously. We have to repair the roads to connect the cities again. There is a fixed cost to repair a particular road. Find out the minimum cost to connect all the cities by repairing roads.

```
#include <iostream>
#include <vector>
#include <limits>
using namespace std;
int minnode(int n, int keyval[], bool mstset[])
{
   int mini = numeric_limits<int>::max();
   int mini_index;
   for (int i = 0; i < n; i++)
   {
      if (mstset[i] == false && keyval[i] < mini)
      {
         mini = keyval[i], mini_index = i;
      }
   }
   return mini_index;
}
void findcost(int n1, int n2, vector<vector<int>> city)
{
   int parent[n1];
   int keyval[n1];
   bool mstset[n1];
   for (int i = 0; i < n1; i++)
   {
      keyval[i] = numeric_limits<int>::max();
      mstset[i] = false;
   }
   parent[0] = -1;
   keyval[0] = 0;
   for (int i = 0; i < n1 - 1; i++)
```

```
    {
        int u = minnode(n1, keyval, mstset);
        mstset[u] = true;
        for (int v = 0; v < n1; v++)
        {
            if (city[u][v] && mstset[v] == false &&
                city[u][v] < keyval[v])
            {
                keyval[v] = city[u][v];
                parent[v] = u;
            }
        }
    }
    int cost = 0;
    for (int i = 1; i < n1; i++)
        cost += city[parent[i]][i];
    cout << cost << endl;
}
int main()
{
    int n1, n2;
    cin >> n1 >> n2;
    int a;
    vector<vector<int>> v;
    for (int i = 0; i < n1; i++)
    {
        vector<int> p;
        for (int j = 0; j < n2; j++)
        {
            cin >> a;
            p.push_back(a);
```

```
        }
        v.push_back(p);
    }
    findcost(n1, n2, v);
    return 0;
}
```

21 Write a program to perform a depth-first search in a graph.

```
#include<iostream>
using namespace std;
int n,i,j;
int visited[999];
int graph[999][999];
void dfs(int m)
{
    int j;
    cout<<m<<" ";
    visited[m]=1;
    for(j=0;j<n;j++)
    {
        if(graph[m][j]==1 && visited[j]==0)
        {
            dfs(j);
        }
    }
}
int main()
{
    cin>>n;
    for(i=0;i<n;i++)
```

```
    {
        for(j=0;j<n;j++)
        {
            cin>>graph[i][j];
        }
    }
    for(i=0;i<n;i++)
    {
        visited[i]=0;
    }
    int key;
    cin>>key;
    dfs(key);
}
```

22 Write a program to solve N Queen's Problem using Backtracking.

```
#include<bits/stdc++.h>
using namespace std;
int N;
void printSol(vector<vector<int>>board)
{
    for(int i = 0;i<N;i++)
    {
        for(int j = 0;j<N;j++)
        {
            cout<<board[i][j]<<" ";
        }
        cout<<"\n";
    }
```

```
}
bool isSafe(int row,int col,vector<bool>rows,vector<bool>left_digonals,vector<bool>Right_digonals)
{
   if(rows[row] == true || left_digonals[row+col] == true || Right_digonals[col-row+N-1] == true)
   {
      return false;
   }
   return true;
}
bool solve(vector<vector<int>>& board,int
col,vector<bool>rows,vector<bool>left_digonals,vector<bool>Right_digonals)
{
        if(col>=N)
        {
                return true;
        }
        for(int i = 0;i<N;i++)
        {
                if(isSafe(i,col,rows,left_digonals,Right_digonals) == true)
                {
                        rows[i] = true;
                        left_digonals[i+col] = true;
                        Right_digonals[col-i+N-1] = true;
                        board[i][col] = 1;
                        if(solve(board,col+1,rows,left_digonals,Right_digonals) == true)
                        {
                                return true;
                        }
                        rows[i] = false;
                        left_digonals[i+col] = false;
                        Right_digonals[col-i+N-1] = false;
```

```
                board[i][col] = 0;
            }
        }
            return false;
}
int main()
{
   cin>>N;
   vector<vector<int>>board(N,vector<int>(N,0));
   vector<bool>rows(N,false);
   vector<bool>left_digonals(2*N-1,false);
   vector<bool>Right_digonals(2*N-1,false);
   bool ans = solve(board , 0, rows,left_digonals,Right_digonals);
   if(ans == true){
      printSol(board);
   }
   else
   {
      cout<<"Solution Does not Exist\n";
   }
}
```

23 Write a program to perform a breadth-first search in a graph.

```
#include<iostream>
using namespace std;
struct queue
{
   int size;
   int f;
   int r;
```

```
    int *arr;
};
int isEmpty(struct queue *q)
{
    if(q->r == q->f)
    {
        return 1;
    }
    return 0;
}
int isFull(struct queue *q)
{
    if(q->r == q->size - 1)
    {
        return 1;
    }
    return 0;
}
void enqueue(struct queue *q, int val)
{
    if(isFull(q))
    {
        printf("The Queue is Full.");
    }
    else
    {
        q->r++;
        q->arr[q->r] = val;
    }
}
int dequeue(struct queue *q)
```

```
{
    int a = -1;
    if(isEmpty(q))
    {
        printf("Queue is Empty.\n");
    }
    else
    {
        q->f++;
        a = q->arr[q->f];
    }
    return a;
}
int main()
{
    struct queue q;
    q.size = 400;
    q.f = q.r = 0;
    q.arr = (int *)malloc(q.size*sizeof(int));
    int node;
    int i;
    int n;
    cin>>n;
    int a[n][n];
    int visited[n];
    for(i=0;i<n;i++)
    {
        visited[i]=0;
    }
    int k1=0,k2=0;
    for(i=0;i<n;i++)
```

```
{
   for(int j=0;j<n;j++)
   {
      a[i][j]=0;
   }
}
while(k1!=-1 && k2!=-1)
{
   cin>>k1>>k2;
   if(k1!=-1 && k2!=-1)
   {
      a[k1][k2]=1;
   }
}
int key1;
cin>>key1;
visited[key1] = 1;
enqueue(&q, key1);
while(!isEmpty(&q))
{
   int node = dequeue(&q);
   for(int j = 0; j<n; j++)
   {
      if(a[node][j] == 1 && visited[j] == 0)
      {
         visited[j] = 1;
         enqueue(&q, j);
      }
   }
   cout<<node<<" ";
}
```

```
    return 0;
}
```

24 Implement Brute-force approach for pattern matching

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

void search(char* txt,char* pat)
{
   int n,m,i,j;
   n=strlen(txt);
   m=strlen(pat);
   for(i=0;i<=n-m;i++){
      for(j=0;j<m;j++){
         if(txt[i+j]!=pat[j]) break;
      }
   if(j==m) cout<<i;
   }
}

int main()
{
   char txt[100],pat[50];
   cin>>txt;
   cin>>pat;
   search(txt,pat);
   return 0;
}
```