1. **Write a C++ program for implementing Singly Linked list.**

```
#include<iostream>
#include<cstdlib>
using namespace std;
struct node
{
        int info;
        struct node *next;
}*start;

class single_llist
{
        public:
                node* create_node(int);
                void insert_begin();
                void insert_last();
                void insert_pos();
                void delete_begin();
                void delete_last();
                void delete_pos();
                void display();
        single_list()
        {
                start=NULL;
                 }
};

int main()
{
        int choice;
        single_llist s1,s2;
        start=NULL;
        do
        {
                cout<<"-----------------"<<endl;
                cout<<"Operattions on singly linked list"<<endl;
                cout<<"-----------------"<<endl;
                cout<<"1.Insert at first"<<endl;
                cout<<"2.Insert at last"<<endl;
                cout<<"3.Insert at position"<<endl;
                cout<<"4.Delete at first"<<endl;
                cout<<"5.Delete at Last"<<endl;
                cout<<"6.Delete at position"<<endl;
                cout<<"7.Display"<<endl;
                cout<<"8.Exit"<<endl;
                cout<<"Enter your choice:";
                cin>>choice;
                switch(choice)
                {
```

```
                case 1: s1.insert_begin();
                        s1.display();
                        break;
                case 2: s1.insert_last();
                        s1.display();
                        break;
                case 3: s1.insert_pos();
                        s1.display();
                        break;
                case 4: s2.delete_begin();
                        s1.display();
                        break;
                case 5: s2.delete_last();
                        s1.display();
                        break;
                case 6: s1.delete_pos();
                        s1.display();
                        break;
                case 7:
                        s1.display();
                        break;
                case 8: exit(0);
                        break;
                default:cout<<"wrong choice...???"<<endl;
                        break;
                }
        }
        while(choice!=8);
}

node *single_llist::create_node(int value)
{
        struct node *temp, *s;
        temp=new(struct node);
        if(temp==NULL)
        {
                cout<<"Memory not allocated"<<endl;
                return 0;
        }
        else
        {
                temp->info=value;
                temp->next=NULL;
                return temp;
        }
}

void single_llist::insert_begin()
{
        int value;
```

```
cout<<"Enter the value to be inserted:";
cin>>value;
struct node *temp, *s;
temp = create_node(value);
if(start==NULL)
{
        start=temp;
        start->next=NULL;
        cout<<temp->info<<"is inserted at first in the empty list"<<endl;
}
else
{
        s=start;
        start=temp;
        start->next=s;
        cout<<temp->info<<"is inserted at first"<<endl;
}
}

void single_llist::insert_last()
{
        int value;
        cout<<"Enter the value to be inserted:";
        cin>>value;
        struct node *temp, *s;
        temp = create_node(value);
        if(start==NULL)
        {
                start=temp;
                start->next=NULL;
                cout<<temp->info<<"is inserted at last in the empty list"<<endl;
        }
        else
        {
                s=start;
                while(s->next!=NULL)
                {
                        s=s->next;
                }
                temp->next=NULL;
                s->next=temp;
                cout<<temp->info<<"is inserted at last"<<endl;
        }
}

void single_llist::insert_pos()
{
        int value, pos, counter = 0, loc = 1;
        struct node *temp, *s, *ptr;
        s = start;
```

```
        while (s != NULL)
        {
                s = s->next;
                counter++;
        }
        if (counter == 0){}
        else
        {
                cout<<"Enter the postion from "<<loc<<" to "<<counter+1<<" : ";
                cin>>pos;
                s = start;
                if(pos == 1)
                {
                cout<<"Enter the value to be inserted:";
                cin>>value;
                temp=create_node(value);
                start=temp;
                start->next=s;
                cout<<temp->info<<"is inserted at first"<<endl;
}
else if(pos>1 && pos<=counter)
{
        cout<<"Enter the value to be inserted:";
                cin>>value;
                temp=create_node(value);
                for(int i=1;i<pos;i++)
                {
                        ptr=s;
                        s=s->next;
                }
                ptr->next=temp;
                temp->next=s;
                cout<<temp->info<<"is inserted at position"<<pos<<endl;
        }
        else if(pos== counter+1)
        {
                cout<<"Enter the value to be inserted:";
                cin>>value;
                temp=create_node(value);
                while(s->next!=NULL)
                {
                        s=s->next;
                }
                temp->next=NULL;
                s->next=temp;
                cout<<temp->info<<"is inserted at last"<<endl;
        }
        else
        {
                cout<<"Position out of range...!!!"<<endl;
```

```
                }
        }
}

void single_llist::delete_begin()
{
        if(start==NULL){}
        else
        {
                struct node *s, *ptr;
                s=start;
                start=s->next;
                cout<<s->info<<"deleted from first"<<endl;
                free(s);
        }
}

void single_llist::delete_last()
{
        int i, counter=0;
        struct node *s, *ptr;
        if(start==NULL){}
        else
        {
                s=start;
                while(s!=NULL)
                {
                        s=s->next;
                        counter++;
                }
                s=start;
                if(counter==1)
                {
                        start=s->next;
                        cout<<s->info<<"Deleted from last"<<endl;
                        free(s);
                }
                else
                {
                        for(i=1;i<counter;i++)
                        {
                                ptr=s;
                                s=s->next;
                        }
                        ptr->next=s->next;
                        cout<<s->info<<"deleted from last"<<endl;
                        free(s);
                }
        }
}
```

```
void single_llist::delete_pos()
{
	int pos, i, counter = 0, loc = 1;
	struct node *s, *ptr;
	s = start;
	while (s != NULL)
	{
		s = s->next;
		counter++;
	}
	if(counter==0){}
	else
	{
		if(counter==1)
		{
			cout<<"Enter the postion [SAY"<<loc<<"]:";
			cin>>pos;
			s=start;
			if(pos==1)
			{
				start=s->next;
				cout<<s->info<<"Deleted from first"<<endl;
				free(s);
			}
			else
			cout<<"Position out of range....!!!"<<endl;
		}
		else
		{
			cout<<"Enter the position from"<<loc<<"to"<<counter<<":";
			cin>>pos;
			s=start;
			if(pos==1)
			{
				start=s->next;
				cout<<s->info<<"deleted from first"<<endl;
				free(s);
			}
			else if(pos>1 && pos<=counter)
			{
				for(i=1;i<pos;i++)
				{
					ptr=s;
					s=s->next;
				}
				ptr->next=s->next;
				if(pos==counter)
				{cout<<s->info<<"deleted from last"<<endl;
				free(s);}
				else
```

```
                    {cout<<s->info<<"deleted from postion"<<pos<<endl;
                    free(s);}
                }
                else
                cout<<"Position out of range...!!!"<<endl;
            }
        }
}

void single_llist::display()
{
        struct node *temp;
        if (start == NULL)
        cout<<"Linked list is empty....!!!"<<endl;
        else
        {
                cout<<"Linked Lsit conatains:";
                temp = start;
                while (temp != NULL)
    {
    cout<<temp->info<<" ";
    temp= temp->next;
        }

                cout<<endl;
        }
}
```

5. Write a program to create a WAP to store a k keys into an array of size n at the location compute using a hash function, loc=key%n, where k<=n and key takes values from [1 to m], m>n. Handle the collision using Linear probing technique.

```
#include<iostream>
#include<limits.h>
using namespace std;
void Insert(int ary[],int hFn, int Size)
{
	int element,pos,n=0;
	cout<<"Enter key element to insert\n";
	cin>>element;
	pos = element%hFn;
	while(ary[pos]!= INT_MIN)
	{
		if(ary[pos]== INT_MAX)
		break;
		pos = (pos+1)%hFn;
		n++;
		if(n==Size)
		break;
	}
	if(n==Size)
	cout<<"Hash table was full of elements\nNo Place to insert this element\n\n";
else
	ary[pos] = element;
}
void display(int ary[],int Size)
{
	int i;
	cout<<"Index\tValue\n";
	for(i=0;i<Size;i++)
	cout<<i<<"\t"<<ary[i]<<"\n";
}
int main()
{
	int Size,hFn,i,choice;
	cout<<"Enter size of hash table\n";
	cin>>Size;
	 hFn=Size;
	int ary[Size];
	for(i=0;i<Size;i++)
	ary[i]=INT_MIN;
	do{
```

```
        cout<<"Enter your choice\n";
        cout<<" 1-> Insert\n 2-> Display\n 0-> Exit\n";
        cin>>choice;
        switch(choice)
        {
        case 1:
                Insert(ary,hFn,Size);
                break;
        case 2:
                display(ary,Size);
                break;
        default:
                cout<<"Enter correct choice\n";
                break;
        }
        }while(choice);
return 0;
}
```

4. Construct a binary search tree (BST) to support the following operations.

Given a key, perform a search in the BST. If the key is found then display "key found".

- **Insert an element into a binary search tree.**
- **Delete an element from a binary search tree.**

Display the tree using inorder, preorder and post order traversal methods(a).

```
#include<iostream>
#include<cstdlib>
using namespace std;
struct node
{
        int info;
        struct node*left;
        struct node*right;
}*root;
class BST
{
        public:
        void find(int, node **,node **);
        void insert(node *,node *);

        void case_a(node *,node *);
        void case_b(node *,node *);
        void case_c(node *,node *);
        void preorder(node *);
        void inorder(node *);
        void postorder(node *);
        void display(node *,int);
        BST()
        {
                root=NULL;
        }
};
int main()
{
        int choice,num;
        BST bst;
        node *temp;
        while(1)
        {
                cout<<"------"<<endl;
                cout<<"Operations on BST"<<endl;
                cout<<"------"<<endl;
                cout<<"1.Insert Element"<<endl;
                cout<<"2.Inorder Traversal"<<endl;
                cout<<"3.Preorder Traversal"<<endl;
                cout<<"4.Postorder Traversal"<<endl;
                cout<<"5.Display"<<endl;
                cout<<"6.Quit"<<endl;
                cout<<"Enter your choice:";
```

```
                cin>>choice;
                switch(choice)
                {
                        case 1:
                                temp=new node;
                                cout<<"Enter the number to be inserted:";
                                cin>>temp->info;
                                bst.insert(root,temp);
                                break;
                        case 2:
                                cout<<"Inorder Traversal of BST:"<<endl;
                                bst.inorder(root);
                                cout<<endl;
                                break;
                        case 3:
                                cout<<"Preorder Traversal of BST:"<<endl;
                                bst.preorder(root);
                                cout<<endl;
                                break;
                        case 4:
                                cout<<"Postorder Traversal of BST:"<<endl;
                                bst.postorder(root);
                                cout<<endl;
                                break;
                        case 5:
                                cout<<"Display BST:"<<endl;
                                bst.display(root,1);
                                cout<<endl;
                                break;
                        case 7:
                                exit(1);
                        default:
                                cout<<"Wrong choice"<<endl;
                }
        }
}
void BST::find(int item, node **par,node **loc)
{
        node *ptr, *ptrsave;
        if(root==NULL)
        {
                *loc=NULL;
                *par=NULL;
                return;
        }
        if(item==root->info)
        {
                *loc=root;
                *par=NULL;
                return;
```

```
        }
        if(item<root->info)
        ptr=root->left;
        else
        ptr=root->right;
        ptrsave=root;
        while(ptr!=NULL)
        {
                if(item==ptr->info)
                {
                        *loc=ptr;
                        *par=ptrsave;
                        return;
                }
                ptrsave = ptr;
                if (item < ptr->info)
                ptr = ptr->left;
                else
                ptr = ptr->right;
        }
*loc=NULL;
*par=ptrsave;
}
void BST::insert(node *tree,node*newnode)
{
        if(root==NULL)
        {
                root=new node;
                root->info=newnode->info;
                root->left=NULL;
                root->right=NULL;
                cout<<"Root Node is Added"<<endl;
                return;
        }
        if(tree->info==newnode->info)
        {
                cout<<"Element already in the tree"<<endl;
                return;
        }
        if(tree->info>newnode->info)
        {
                if(tree->left!=NULL)
                {
                        insert(tree->left,newnode);
                }
                else
                {
                        tree->left=newnode;
                        (tree->left)->left=NULL;
                        (tree->left)->right=NULL;
```

```
                cout<<"Node Added To Left"<<endl;
                return;
            }
        }
        else
        {
            if(tree->right!=NULL)
            {
                insert(tree->right,newnode);
            }
            else
            {
            tree->right = newnode;
            (tree->right)->left = NULL;
            (tree->right)->right = NULL;
            cout<<"Node Added To Right"<<endl;
            return;
            }
        }
}

void BST::case_a(node *par, node *loc )
{
        if (par == NULL)
        {
            root = NULL;
        }
        else
        {
            if (loc == par->left)
            par->left = NULL;
        else
            par->right = NULL;
        }
}

void BST::case_b(node *par, node *loc)
{
        node *child;
        if (loc->left != NULL)
        child = loc->left;
 else
        child = loc->right;
        if(par==NULL)
        {
            root=child;
        }
        else
        {
            if(loc==par->left)
```

```
                par->left=child;
                else
                par->right=child;
        }
}

void BST::case_c(node *par, node *loc)
{
        node *ptr, *ptrsave, *suc, *parsuc;
        ptrsave = loc;
        ptr = loc->right;
        while (ptr->left != NULL)
        {
                ptrsave = ptr;
                ptr = ptr->left;
        }
        suc = ptr;
        parsuc = ptrsave;
        if (suc->left == NULL && suc->right == NULL)
        case_a(parsuc, suc);
        else
        case_b(parsuc, suc);
        if(par==NULL)
        {
                root=suc;
        }
        else
        {
                if(loc==par->left)
                par->left=suc;
                else
                par->right=suc;
        }
        suc->left=loc->left;
        suc->right=loc->right;
}
void BST::preorder(node *ptr)
{
        if (root == NULL)
        {
                cout<<"Tree is empty"<<endl;
                return;
        }
        if (ptr != NULL)
        {
                cout<<ptr->info<<" ";
                preorder(ptr->left);
                preorder(ptr->right);
        }
}
```

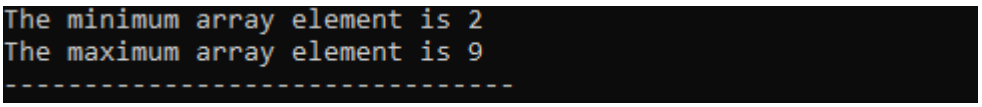
```
void BST::inorder(node *ptr)
 {
        if(root==NULL)
        {
                cout<<"Tree is empty"<<endl;
                return;
         }
         if(ptr!=NULL)
         {
                inorder(ptr->left);
                cout<<ptr->info<<" ";
                inorder(ptr->right);
         }
}

void BST::postorder(node *ptr)
{
        if(root==NULL)
        {
                cout<<"Tree is empty"<<endl;
                return;
        }
        if(ptr!=NULL)
        {
                postorder(ptr->left);
                postorder(ptr->right);
                cout<<ptr->info<<" ";
        }
}

void BST::display(node *ptr, int level)
{
        int i;
        if (ptr != NULL)
        {
                display(ptr->right, level+1);
                cout<<endl;
                if (ptr == root)
                cout<<"Root->: ";
                 else
                {
                        for (i = 0;i < level;i++)
                        cout<<" ";
                }
                cout<<ptr->info;
                display(ptr->left, level+1);
        }
```

8. Finding minimum and maximum from given unsorted array by using divide and conquer method.

```
#include <iostream>
using namespace std;
void findMinAndMax(int arr[], int low, int high, int &min, int &max)
{
	if (low == high)
	{
		if (max < arr[low])
		{
			max = arr[low];
		}
		if (min > arr[high])
		{
			min = arr[high];
		}
	return;
	}

	if (high - low == 1)
	{
		if (arr[low] < arr[high])
		{
			if (min > arr[low])
			{
				min = arr[low];
			}

			if (max < arr[high])
			{
				max = arr[high];
			}
		}
		else {
			if (min > arr[high])
			{
				min = arr[high];
			}

		if (max < arr[low])
		{
			max = arr[low];
		}
```

```
        }
        return;
}
int mid = (low + high) / 2;
findMinAndMax(arr, low, mid, min, max);
findMinAndMax(arr, mid + 1, high, min, max);
}

int main()
{
        int arr[] = { 7, 2, 9, 3, 6, 7, 8, 4 };
        int n = sizeof(arr) / sizeof(arr[0]);
        int max = arr[0], min = arr[0];
        findMinAndMax(arr, 0, n - 1, min, max);
        cout << "The minimum array element is " << min << endl;
        cout << "The maximum array element is " << max;
        return 0;
}
```

OUTPUT:



10. Write a C++ program for solving the N-Queen's Problem using backtracking.

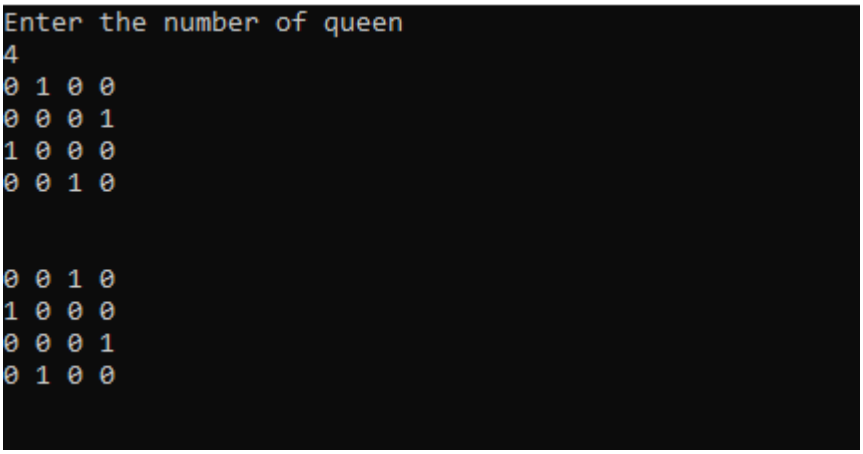
//program to solve the n queen problem
//grid[][] is represent the 2-d array with value(0 and 1) for grid[i][j]=1 means queen i are placed at j column.
//we can take any number of queen , for this time we take the atmost 10 queen (grid[10][10]).

```
#include<iostream>
using namespace std;
int grid[10][10];
void print(int n)
{
	for (int i = 0;i <= n-1; i++)
	{
		for (int j = 0;j <= n-1; j++)
		{
			cout <<grid[i][j]<< " ";
		 }
		cout<<endl;
	}
	cout<<endl;
	cout<<endl;
}

bool isSafe(int col, int row, int n)
{
	for (int i = 0; i < row; i++)
	{
		if (grid[i][col])
		 {
			 return false;
		}
	}
	for (int i = row,j = col;i >= 0 && j >= 0; i--,j--)
	{
		if (grid[i][j])
		{
			return false;
		}
	}

	for (int i = row, j = col; i >= 0 && j < n; j++, i--)
	{
		if (grid[i][j])
```

```
                {
                        return false;
                }
        }
         return true;
}

bool solve (int n, int row)
{
        if (n == row)
        {
                print(n);
                return true;
         }

        bool res = false;
        for (int i = 0;i <=n-1;i++)
         {
                if (isSafe(i, row, n))
                 {
                         grid[row][i] = 1;
                         res = solve(n, row+1) || res;
                        grid[row][i] = 0;
                 }
         }
        return res;
}

int main()
{
        ios_base::sync_with_stdio(false);
        cin.tie(NULL);
        int n;
        cout<<"Enter the number of queen"<<endl;
        cin >> n;
        for (int i = 0;i < n;i++)
        {
                for (int j = 0;j < n;j++)
                {
                        grid[i][j] = 0;
                }
        }

        bool res = solve(n, 0);
```

```
        if(res == false)
        {
                cout << -1 << endl; //if there is no possible solution
        }
        else
        {
                cout << endl;
         }
    return 0;
}
```

OUTPUT:



11. Write a program to implement Breadth First Search for undirected graph(BFS).

```
#include<iostream>
#include <list>
 using namespace std;
class Graph
{
	int V;
	list<int> *adj;
	public:
		Graph(int V);
		void addEdge(int v, int w);
		void BFS(int s);
};
 Graph::Graph(int V)
{
	this->V = V;
	adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
	adj[v].push_back(w);
}

void Graph::BFS(int s)
{
	bool *visited = new bool[V];
	for(int i = 0; i < V; i++)
	visited[i] = false;
	list<int> queue;
	visited[s] = true;
	queue.push_back(s);
	list<int>::iterator i;
	while(!queue.empty())
	{
		s = queue.front();
		cout << s << " ";
		queue.pop_front();
		for (i = adj[s].begin(); i != adj[s].end(); ++i)
		{
		if (!visited[*i])
		{
			visited[*i] = true;
```

```
                        queue.push_back(*i);
                }
                }
        }
}

int main()
{
        Graph g(10);
        g.addEdge(3, 4);
        g.addEdge(3, 5);
        g.addEdge( 4,2);
        g.addEdge(4, 9);
        cout << "Following is Breadth First Traversal "<< "(starting from vertex 2) \n";
        g.BFS(3);
         return 0;
}
```

OUTPUT:

```
Following is Breadth First Traversal (starting from vertex 2)
3 4 5 2 9
--------------------------------
```

12. Write a program to implement Depth First Search for undirected graph(DFS).

```
#include <bits/stdc++.h>
using namespace std;
class Graph
{
	public:
		map<int, bool> visited;
		map<int, list<int> > adj;
		void addEdge(int v, int w);
		void DFS(int v);
};
void Graph::addEdge(int v, int w)
{
	adj[v].push_back(w);
	adj[w].push_back(v);
}
void Graph::DFS(int v)
{
	visited[v] = true;
	cout << v << " ";
	list<int>::iterator i;
	for (i = adj[v].begin(); i != adj[v].end(); ++i)
	if (!visited[*i])
	DFS(*i);
}
int main()
{
	Graph g;
	g.addEdge(0, 1);
	g.addEdge(0, 2);
	g.addEdge(1, 2);
	g.addEdge(2, 0);
	g.addEdge(2, 3);
	g.addEdge(3, 3);
	cout << "Following is Depth First Traversal"<<" (starting from vertex 2) \n";
	g.DFS(2);
	return 0;
}
```

OUTPUT:

```
Following is Depth First Traversal (starting from vertex 2)
2 0 1 3
--------------------------------
```