**1.Write a C++ program to implement 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 search();

void display();

single\_llist()

{

start = NULL;

}

};

.0

int main()

{

int choice;

single\_llist sl,s2;

start = NULL;

do

{

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.Search"<<endl;

cout<<"8.Display"<<endl;

cout<<"9.Exit "<<endl;

cout<<"Enter your choice :";

cin>>choice;

switch(choice)

{

case 1: sl.insert\_begin();

sl.display();

break;

case 2: sl.insert\_last();

sl.display();

break;

case 3: sl.insert\_pos();

sl.display();

break;

case 4: s2.delete\_begin();

sl.display();

break;

case 5: s2.delete\_last();

sl.display();

break;

case 6: sl.delete\_pos();

sl.display();

break;

case 7:sl.search();

sl.display();

break;

case 8:sl.display();

break;

case 9:exit(0);

break;

default:cout<<"Wrong choice...???"<<endl;

break;

}

}

while(choice != 9);

}

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<<"Positon 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 postion 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::search()

{

int value, loc = 0, pos = 0, counter = 0;

struct node \*s;

s = start;

while (s != NULL)

{

s = s->next;

counter++;

}

if (start == NULL){}

else

{

cout<<"Enter the value to be searched : ";

cin>>value;

struct node \*s;

s = start;

while (s != NULL)

{

pos++;

if (s->info == value)

{

loc++;

if(loc == 1)

cout<<"Element "<<value<<" is found at position "<<pos;

else if(loc <= counter)

cout<<" , "<<pos;

}

s = s->next;

}

cout<<endl;

if (loc == 0)

cout<<"Element "<<value<<" not found in the list"<<endl;

}

}

void single\_llist::display()

{

struct node \*temp;

if (start == NULL)

cout<<"Linked list is empty...!!!"<<endl;

else

{

cout<<"Linked list contains : ";

temp = start;

while (temp != NULL)

{

cout<<temp->info<<" ";

temp = temp->next;

}

cout<<endl;

}

}

**2.Write a C++ program to implement doubly linked list.**

#include<stdio.h>

#include<stdlib.h>

struct node

{

struct node \*prev;

struct node \*next;

int data;

};

struct node \*head;

void insertion\_beginning();

void insertion\_last();

void insertion\_specified();

void deletion\_beginning();

void deletion\_last();

void deletion\_specified();

void display();

void search();

int main ()

{

int choice =0;

while(choice != 9)

{

printf("\nChoose one option from the following list ...\n");

printf("\n1.Insert in begining\n2.Insert at last\n3.Insert at any random location\n4.Delete from Beginning\n5.Delete from last\n6.Delete the node after the given data\n7.Search\n8.Show\n9.Exit\n");

printf("\nEnter your choice?\n");

scanf("\n%d",&choice);

switch(choice)

{

case 1:

insertion\_beginning();

break;

case 2:

insertion\_last();

break;

case 3:

insertion\_specified();

break;

case 4:

deletion\_beginning();

break;

case 5:

deletion\_last();

break;

case 6:

deletion\_specified();

break;

case 7: search();

break;

case 8: display();

break;

case 9: exit(0);

break;

default: printf("Please enter valid choice..");

}

}

}

void insertion\_beginning()

{

struct node \*ptr;

int item;

ptr = (struct node \*)malloc(sizeof(struct node));

if(ptr == NULL)

{

printf("\nOVERFLOW");

}

else

{

printf("\nEnter Item value");

scanf("%d",&item);

if(head==NULL)

{

ptr->next = NULL;

ptr->prev=NULL;

ptr->data=item;

head=ptr;

}

else

{

ptr->data=item;

ptr->prev=NULL;

ptr->next = head;

head->prev=ptr;

head=ptr;

}

printf("\nNode inserted\n");

}

}

void insertion\_last()

{

struct node \*ptr,\*temp;

int item;

ptr = (struct node \*) malloc(sizeof(struct node));

if(ptr == NULL)

{

printf("\nOVERFLOW");

}

else

{

printf("\nEnter value");

scanf("%d",&item);

ptr->data=item;

if(head == NULL)

{

ptr->next = NULL;

ptr->prev = NULL;

head = ptr;

}

else

{

temp = head;

while(temp->next!=NULL)

{

temp = temp->next;

}

temp->next = ptr;

ptr ->prev=temp;

ptr->next = NULL;

}

}

printf("\nnode inserted\n");

}

void insertion\_specified()

{

struct node \*ptr,\*temp;

int item,loc,i;

ptr = (struct node \*)malloc(sizeof(struct node));

if(ptr == NULL)

{

printf("\n OVERFLOW");

}

else

{

temp=head;

printf("Enter the location");

scanf("%d",&loc);

for(i=0;i<loc;i++)

{

temp = temp->next;

if(temp == NULL)

{

printf("\n There are less than %d elements", loc);

return;

}

}

printf("Enter value");

scanf("%d",&item);

ptr->data = item;

ptr->next = temp->next;

ptr -> prev = temp;

temp->next = ptr;

temp->next->prev=ptr;

printf("\nnode inserted\n");

}

}

void deletion\_beginning()

{

struct node \*ptr;

if(head == NULL)

{

printf("\n UNDERFLOW");

}

else if(head->next == NULL)

{

head = NULL;

free(head);

printf("\nnode deleted\n");

}

else

{

ptr = head;

head = head -> next;

head -> prev = NULL;

free(ptr);

printf("\nnode deleted\n");

}

}

void deletion\_last()

{

struct node \*ptr;

if(head == NULL)

{

printf("\n UNDERFLOW");

}

else if(head->next == NULL)

{

head = NULL;

free(head);

printf("\nnode deleted\n");

}

else

{

ptr = head;

if(ptr->next != NULL)

{

ptr = ptr -> next;

}

ptr -> prev -> next = NULL;

free(ptr);

printf("\nnode deleted\n");

}

}

void deletion\_specified()

{

struct node \*ptr, \*temp;

int val;

printf("\n Enter the data after which the node is to be deleted : ");

scanf("%d", &val);

ptr = head;

while(ptr -> data != val)

ptr = ptr -> next;

if(ptr -> next == NULL)

{

printf("\nCan't delete\n");

}

else if(ptr -> next -> next == NULL)

{

ptr ->next = NULL;

}

else

{

temp = ptr -> next;

ptr -> next = temp -> next;

temp -> next -> prev = ptr;

free(temp);

printf("\nnode deleted\n");

}

}

void display()

{

struct node \*ptr;

printf("\n printing values...\n");

ptr = head;

while(ptr != NULL)

{

printf("%d\n",ptr->data);

ptr=ptr->next;

}

}

void search()

{

struct node \*ptr;

int item,i=0,flag;

ptr = head;

if(ptr == NULL)

{

printf("\nEmpty List\n");

}

else

{

printf("\nEnter item which you want to search?\n");

scanf("%d",&item);

while (ptr!=NULL)

{

if(ptr->data == item)

{

printf("\nitem found at location %d ",i+1);

flag=0;

break;

}

else

{

flag=1;

}

i++;

ptr = ptr -> next;

}

if(flag==1)

{

printf("\nItem not found\n");

}

}

}

**3.Write a C++ program to split the linked list into two halves such that the element ‘e’ should be the first element of second list.**

#include<iostream>

using namespace std;

struct Node{

int value;

struct Node \*next;

};

struct Node\* head = NULL;

struct Node\* sHead = NULL;

struct Node\* temp = NULL;

void insert(int new\_data){

struct Node\* new\_node = new Node(); //(struct Node\*)malloc(sizeof(struct Node));

new\_node->value = new\_data;

new\_node->next = head;

head = new\_node;

}

int n;

int ele;

int splitIndex;

int main(){

int i;

cout<<"Enter number of elements you want in the list\t";

cin>>n;

cout<<"Enter elements :" <<endl;

for(i=0;i<n;i++){

cin>>ele;

insert(ele);

}

cout<<"\nList of elements : "<<endl;

Node \*t;

t = head;

while(t != NULL){

cout<<t->value<<"\t";

t = t->next;

}

cout<<"\n\nEnter the position you want the list to split ";

cin>>splitIndex;

while(splitIndex < 0 || splitIndex > n-1){

cout<<"Invalid position. Try again."<<endl;

cin>>splitIndex;

}

temp = head;

for(i=0;i<=splitIndex;i++){

if(i==splitIndex-1){

Node \*tN;

tN = temp->next;

sHead = tN;

temp->next = NULL;

break;

}

temp = temp->next;

}

temp = head;

if(temp == NULL){

cout<<"\nFirst list is empty"<<endl;

}else{

cout<<"\n\nFirst list element "<<endl;

while(temp != NULL){

cout<<temp->value<<"\t";

temp = temp->next;

}

}

temp = sHead;

if(temp == NULL){

cout<<"\nSecond list is empty"<<endl;

}else{

cout<<"\n\nSecond list elements "<<endl;

while(temp != NULL){

cout<<temp->value<<"\t";

temp = temp->next;

}

}

return 0;

}

**4.Find the subset of a given set S = {S1,S2,S3,………,Sn} OF ‘n’ positive integers whose sum is equal to a given positive integer d.**

**#include<iostream>**

**using namespace std;**

**int s[10],d,n,set[10],count=0;**

**void display(int);**

**int flag = 0;**

**int main()**

**{**

**int subset(int,int);**

**int i;**

**cout<<"ENTER THE NUMBER OF THE ELEMENTS IN THE SET : ";**

**cin>>n;**

**cout<<"ENTER THE SET OF VALUES : ";**

**for(i=0;i<n;i++)**

**cin>>s[i];**

**cout<<"ENTER THE SUM : ";**

**cin>>d;**

**cout<<"THE PROGRAM OUTPUT IS: ";**

**subset(0 , 0);**

**if(flag == 0)**

**cout<<"There is no solution";**

**}**

**int subset(int sum,int i)**

**{**

**if(sum == d)**

**{**

**flag = 1;**

**display(count);**

**return 0;**

**}**

**if(sum>d || i>=n)**

**return 1;**

**else**

**{**

**set[count]=s[i];**

**count++;**

**subset(sum+s[i],i+1);**

**count--;**

**subset(sum,i+1);**

**}**

**}**

**void display(int count)**

**{**

**int i;**

**cout<<"\t{";**

**for(i=0;i<count;i++)**

**cout<<set[i];**

**cout<<"}";**

**}**

**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;**

**}**

**6.Write a program to Insert into and Delete from a Binary Search Tree.**

**# 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 del(int);**

**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.Delete Element "<<endl;**

**cout<<"3.Inorder Traversal"<<endl;**

**cout<<"4.Preorder Traversal"<<endl;**

**cout<<"5.Postorder Traversal"<<endl;**

**cout<<"6.Display"<<endl;**

**cout<<"7.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:**

**if (root == NULL)**

**{**

**cout<<"Tree is empty, nothing to delete"<<endl;**

**continue;**

**}**

**cout<<"Enter the number to be deleted : ";**

**cin>>num;**

**bst.del(num);**

**break;**

**case 3:**

**cout<<"Inorder Traversal of BST:"<<endl;**

**bst.inorder(root);**

**cout<<endl;**

**break;**

**case 4:**

**cout<<"Preorder Traversal of BST:"<<endl;**

**bst.preorder(root);**

**cout<<endl;**

**break;**

**case 5:**

**cout<<"Postorder Traversal of BST:"<<endl;**

**bst.postorder(root);**

**cout<<endl;**

**break;**

**case 6:**

**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::del(int item)**

**{**

**node \*parent, \*location;**

**if (root == NULL)**

**{**

**cout<<"Tree empty"<<endl;**

**return;**

**}**

**find(item, &parent, &location);**

**if (location == NULL)**

**{**

**cout<<"Item not present in tree"<<endl;**

**return;**

**}**

**if (location->left == NULL && location->right == NULL)**

**case\_a(parent, location);**

**if (location->left != NULL && location->right == NULL)**

**case\_b(parent, location);**

**if (location->left == NULL && location->right != NULL)**

**case\_b(parent, location);**

**if (location->left != NULL && location->right != NULL)**

**case\_c(parent, location);**

**free(location);**

**}**

**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);**

**}**

**}**

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

**#include <iostream>**

**using namespace std;**

**void MinMax(int arr[], int low, int high, int &min, int &max)**

**{**

**if (low == high)**

**{**

**if (max < arr[low]) { // comparison 1**

**max = arr[low];**

**}**

**if (min > arr[high]) { // comparison 2**

**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;**

**MinMax(arr, low, mid, min, max);**

**MinMax(arr, mid + 1, high, min, max);**

**}**

**int main()**

**{**

**int i, n, arr[50];**

**cout<<"Enter the number of elements : ";**

**cin>>n;**

**for( i = 0; i < n; i++ )**

**{**

**cout<<"Enter the element : ";**

**cin>>arr[i];**

**}**

**int max = arr[0], min = arr[0];**

**MinMax(arr, 0, n - 1, min, max);**

**cout<<"The minimum array element is "<<min<<endl;**

**cout<<"The maximum array element is "<<max;**

**}**

**8.Create a program to merge sort using divide and** conquer **array.**

**#include <iostream>**

**using namespace std;**

**void Merge(int \*a, int low, int high, int mid)**

**{**

**int i, j, k, temp[high-low+1];**

**i = low;**

**k = 0;**

**j = mid + 1;**

**while (i <= mid && j <= high)**

**{**

**if (a[i] < a[j])**

**{**

**temp[k] = a[i];**

**k++;**

**i++;**

**}**

**else**

**{**

**temp[k] = a[j];**

**k++;**

**j++;**

**}**

**}**

**while (i <= mid)**

**{**

**temp[k] = a[i];**

**k++;**

**i++;**

**}**

**while (j <= high)**

**{**

**temp[k] = a[j];**

**k++;**

**j++;**

**}**

**for (i = low; i <= high; i++)**

**{**

**a[i] = temp[i-low];**

**}**

**}**

**void MergeSort(int \*a, int low, int high)**

**{**

**int mid;**

**if (low < high)**

**{**

**mid=(low+high)/2;**

**MergeSort(a, low, mid);**

**MergeSort(a, mid+1, high);**

**Merge(a, low, high, mid);**

**}**

**}**

**int main()**

**{**

**int n, i;**

**cout<<"\nEnter the number of data element to be sorted: ";**

**cin>>n;**

**int arr[n];**

**for(i = 0; i < n; i++)**

**{**

**cout<<"Enter element : ";**

**cin>>arr[i];**

**}**

**MergeSort(arr, 0, n-1);**

**// Printing the sorted data.**

**cout<<"\nSorted Data ";**

**for (i = 0; i < n; i++)**

**cout<<"->"<<arr[i];**

**return 0;**

**}**

**9.Write a C++ program for solving the N-Queen’s Problem using backtracking.**

#include<iostream>

using namespace std;

int grid[10][10];

//print the solution

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;

}

//function for check the position is safe or not

//row is indicates the queen no. and col represents the possible positions

bool isSafe(int col, int row, int n) {

//check for same column

for (int i = 0; i < row; i++) {

if (grid[i][col]) {

return false;

}

}

//check for upper left diagonal

for (int i = row,j = col;i >= 0 && j >= 0; i--,j--) {

if (grid[i][j]) {

return false;

}

}

//check for upper right diagonal

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;

//recursive call solve(n, row+1) for next queen (row+1)

res = solve(n, row+1) || res;

//if res ==false then backtracking will occur

//by assigning the grid[row][i] = 0

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;

}

**10.Write a program to implement breadth first search for undirected graph (BFS).**

#include<iostream>

#include <list>

using namespace std;

// This class represents a directed graph using

// adjacency list representation

class Graph

{

int V; // No. of vertices

// Pointer to an array containing adjacency

// lists

list<int> \*adj;

public:

Graph(int V); // Constructor

// function to add an edge to graph

void addEdge(int v, int w);

// prints BFS traversal from a given source s

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); // Add w to v’s list.

}

void Graph::BFS(int s)

{

// Mark all the vertices as not visited

bool \*visited = new bool[V];

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

visited[i] = false;

// Create a queue for BFS

list<int> queue;

// Mark the current node as visited and enqueue it

visited[s] = true;

queue.push\_back(s);

// 'i' will be used to get all adjacent

// vertices of a vertex

list<int>::iterator i;

while(!queue.empty())

{

// Dequeue a vertex from queue and print it

s = queue.front();

cout << s << " ";

queue.pop\_front();

// Get all adjacent vertices of the dequeued

// vertex s. If a adjacent has not been visited,

// then mark it visited and enqueue it

for (i = adj[s].begin(); i != adj[s].end(); ++i)

{

if (!visited[\*i])

{

visited[\*i] = true;

queue.push\_back(\*i);

}

}

}

}

// Driver program to test methods of graph class

int main()

{

// Create a graph given in the above diagram

Graph g(4);

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 Breadth First Traversal "<< "(starting from vertex 2) \n";

g.BFS(2);

return 0;

}

**11.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;

}

**12.Write a program to implement Min Heap.**

#include <iostream>

#include <conio.h>

using namespace std;

void min\_heap(int \*a, int m, int n){

int j, t;

t= a[m];

j = 2 \* m;

while (j <= n) {

if (j < n && a[j+1] < a[j])

j = j + 1;

if (t < a[j])

break;

else if (t >= a[j]) {

a[j/2] = a[j];

j = 2 \* j;

}

}

a[j/2] = t;

return;

}

void build\_minheap(int \*a, int n) {

int k;

for(k = n/2; k >= 1; k--) {

min\_heap(a,k,n);

}

}

int main() {

int n, i;

cout<<"enter no of elements of array\n";

cin>>n;

int a[30];

for (i = 1; i <= n; i++) {

cout<<"enter element"<<" "<<(i)<<endl;

cin>>a[i];

}

build\_minheap(a, n);

cout<<"Min Heap\n";

for (i = 1; i <= n; i++) {

cout<<a[i]<<endl;

}

getch();

}

**13.Write a program to implement Max Heap Sort.**

#include <iostream>

using namespace std;

void MaxHeapify (int a[], int i, int n)

{

int j, temp;

temp = a[i];

j = 2\*i;

while (j <= n)

{

if (j < n && a[j+1] > a[j])

j = j+1;

if (temp > a[j])

break;

else if (temp <= a[j])

{

a[j/2] = a[j];

j = 2\*j;

}

}

a[j/2] = temp;

return;

}

void HeapSort(int a[], int n)

{

int i, temp;

for (i = n; i >= 2; i--)

{

temp = a[i];

a[i] = a[1];

a[1] = temp;

MaxHeapify(a, 1, i - 1);

}

}

void Build\_MaxHeap(int a[], int n)

{

int i;

for(i = n/2; i >= 1; i--)

MaxHeapify(a, i, n);

}

int main()

{

int n, i,arr[100];

cout<<"\nEnter the number of data element to be sorted: ";

cin>>n;

n++;

for(i=1;i<n;i++)

{

cout<<"Enter element"<<i<<":";

cin>>arr[i];

}

Build\_MaxHeap(arr, n-1);

HeapSort(arr, n-1);

cout<<"\nSorted Data ";

for (i = 1; i < n; i++)

cout<<" "<<arr[i];

return 0;

}