**Practical No.5**

**Binary Search Tree**

**Problem Statement:-**

Implement binary search tree and perform following operations:

a) Insert (Handle insertion of duplicate entry) b) Delete c) Search d) Display tree (Traversal) e) Display - Depth of tree f) Display - Mirror image g) Create a copy h) Display all parent nodes with their child nodes i) Display leaf nodes j) Display tree level wise

(Note: Insertion, Deletion, Search and Traversal are compulsory, from rest of operations, perform Any three)

**Program Code:-**

#include<iostream>

using namespace std;

typedef struct tnode

{

int data;

struct tnode\*left;

struct tnode\*right;

}tnode;

typedef struct node

{

struct tnode\*x;

struct node \*next;

}node;

class queue

{

node \*front,\*rear;

public:

queue()

{

front=NULL;

rear=NULL;

}

int isempty()

{

if(front==NULL)

return 1;

return 0;

}

void enque(tnode \*i)

{

node \*p;

p=new node();

p->x=i;

p->next=NULL;

if(front==NULL)

{

front=p;

rear=p;

}

else

{

rear->next=p;

rear=rear->next;

}

}

tnode \*deque()

{

node \*p;

tnode \*temp;

p=front;

temp=front->x;

if(front==rear)

{

front=NULL;

rear=NULL;

}

else

{

front=front->next;

}

delete p;

return temp;

}

};

class tree

{

tnode \*t;

public:

tree()

{

t=NULL;

}

tnode \*insert(int x)

{

tnode \*p,\*q,\*r;

p=new tnode();

p->data=x;

p->left=NULL;

p->right=NULL;

if(t==NULL)

return p;

q=t;

r=t;

while(r!=NULL)

{

q=r;

if(x<r->data)

r=r->left;

else

r=r->right;

}

if(x<q->data)

q->left=p;

else

q->right=p;

return t;

}

tnode \*create()

{

int n,i,key;

cout<<" \n Enter the number of nodes - ";

cin>>n;

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

{

cout<<" \n Enter the data -";

cin>>key;

t=insert(key);

}

return t;

}

void inorder(tnode \*t)

{

if(t!=NULL)

{

inorder(t->left);

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

inorder(t->right);

}

}

tnode\* search(int key)

{

tnode \*s=t;

while(s!=NULL)

{

if(s->data==key)

return t;

else if(s->data<key)

s=s->right;

else

s=s->left;

}

return NULL;

}

tnode \*find\_min(tnode \*r)

{

while(r->left!=NULL)

{

r=r->left;

}

return r;

}

tnode \*del(tnode \*t,int key)

{

tnode \*temp;

if(t==NULL)

{

return NULL;

}

if(key<t->data)

{

t->left=del(t->left,key);

return t;

}

if(key>t->data)

{

t->right=del(t->right,key);

return t;

}

//element found

//no child

if(t->left==NULL&t->right==NULL)

{

temp=t;

delete temp;

return NULL;

}

//one child

if(t->left!=NULL&&t->right==NULL)

{

temp=t;

t=t->left;

delete temp;

return t;

}

if(t->left==NULL&&t->right!=NULL)

{

temp=t;

t=t->right;

delete temp;

return t;

}

//both child present

temp=find\_min(t->right);

t->data=temp->data;

t->right=del(t->right,temp->data);

return t;

}

tnode \*mirror(tnode \*t)

{

tnode \*temp;

if(t==NULL)

{

return NULL;

}

temp=t->left;

t->left=mirror(t->right);

t->right=mirror(temp);

return t;

}

tnode\* copy(tnode \*T)

{

tnode \*P;

P=NULL;

if(T!=NULL)

{

P=new tnode();

P->data=T->data;

P->left=copy(T->left);

P->right=copy(T->right);

}

return P;

}

int height(tnode \*T)

{

int hl,hr;

if(T==NULL)

return 0;

if(T->left==NULL && T->right==NULL)

return 0;

hl=height(T->left);

hr=height(T->right);

if(hl>hr)

return 1+hl;

else

return 1+hr;

}

void leaf(tnode \*T)

{

if(T==NULL)

return ;

if(T->left==NULL && T->right==NULL)

{ cout<<"\t"<<T->data;

}

leaf(T->left);

leaf(T->right);

}

void parent(tnode \*T)

{

if(T==NULL)

return ;

if(T->left!=NULL && T->right==NULL)

{

cout<<"\t"<<T->data;

cout<<"\t"<<T->left->data;

cout<<"\n";

}

if(T->left==NULL && T->right!=NULL)

{

cout<<"\t"<<T->data;

cout<<"\t"<<T->right->data;

cout<<"\n";

}

if(T->left!=NULL && T->right!=NULL)

{

cout<<"\t"<<T->data;

cout<<"\t"<<T->left->data<<"\t"<<T->right->data;

cout<<"\n";

}

parent(T->left);

parent(T->right);

}

void level\_wise()

{

tnode \*t1;

queue q1;

if(t==NULL)

return;

q1.enque(t);

cout<<"\n"<<t->data;

while(q1.isempty()!=1)

{

cout<<"\n";

queue q2;

while(q1.isempty()!=1)

{

t1=q1.deque();

if(t1->left!=NULL)

{

q2.enque(t1->left);

cout<<" "<<t1->left->data;

}

if(t1->right!=NULL)

{

q2.enque(t1->right);

cout<<" "<<t1->right->data;

}

}

q1=q2;

}

}

};

int main()

{

int choice,key, cnt;

tnode \*root,\*result, \*rt;

tree t;

do

{

cout<<" \n Main menu "

"\n 1.Create "

"\n 2.Insert "

"\n 3.Display "

"\n 4.Search "

"\n 5.Delete "

"\n 6.Mirror image "

"\n 7.create copy "

"\n 8.Find Depth "

"\n 9.Minimum "

"\n 10.Display Tree Level-wise "

"\n 11.Display Leaf nodes "

"\n 12.Display parent node with child nodes "

"\n 13.Exit \n Enter your choice - ";

cin>>choice;

switch(choice)

{

case 1:root=t.create();

break;

case 2:cout<<"\n Enter the number to insert - ";

cin>>key;

root=t.insert(key);

break;

case 3:cout<<"Binary tree :-";

t.inorder(root);

break;

case 4:cout<<" \n Enter the node to search -";

cin>>key;

result=t.search(key);

if(result==NULL)

{

cout<<"\n Element "<<key<<" not present"<<endl;

}

else

{

cout<<"\n Element "<<key<<" is present"<<endl;

}

break;

case 5:cout<<"\n Enter the node to delete -";

cin>>key;

result=t.del(root,key);

root=result;

cout<<"\n Element deleted successfully!!"<<endl;

break;

case 6:root=t.mirror(root);

cout<<"\n Mirror image of the binary tree is :-"<<endl;

t.inorder(root);

break;

break;

case 7: cout<<"\n Copied tree - ";

rt=t.copy(root);

t.inorder(rt);

break;

case 8:cnt=t.height(root);

cout<<"\n Height of tree -"<<cnt;

break;

case 9:result=t.find\_min(root);

cout<<"\n Minimum is "<<result->data<<endl;

break;

case 10:cout <<"\n Level wise display :-"<<endl;

t.level\_wise();

break;

case 11:cout <<"\n Leaf nodes are :-"<<endl;

t.leaf(root);

break;

case 12:cout <<"\n Parent node with child nodes are :-"<<endl;

t.parent(root);

break;

case 13:return 0;

default:cout<<"\n Invalid choice !! Please enter your choice again."<<endl;

}

}while(choice!=13);

}