**SD ASSIGNMENT 1**

**(BST)**

**NAME : KUNAL GUPTA**

**GR.NO: 17U427 ROLL NO:23**

**BATCH: A1**

**CODE:**

#include<iostream>

#include<stdlib.h>

using namespace std;

#define size 30

class node //class node

{

public:

node \*left,\*right;

int data;

node() //constructor

{

left=right=NULL;

}

node(int x)

{

data=x;

left=right=NULL;

}

};

class bst //class bst

{

node \*root;

void inorder1(node \*temp);

void rec\_search(node \*,int x);

node \*mirror(node \*);

node \*delete\_node(node \*,int);

int heightp(node \*);

void level(node \*);

public:

void insert(int x);

void insert\_rec(int x);

int search(int x);

void searchrec(int x)

{

rec\_search(root,x);

}

bst() //constructor

{

root=NULL;

}

void inorder();

void dele();

void displayMirror();

void height()

{

cout<<"Height:"<<heightp(root);

}

void min();

void max();

void levelp()

{

level(root);

}

};

void bst::insert(int x)

{

node \*curr,\*par;

node \*temp=new node(x);

if(root==NULL)

{

root=temp;

}

else

{

curr=root;

while(curr!=NULL)

{

par=curr;

if(curr->data>temp->data)

{

curr=curr->left;

}

else

{

curr=curr->right;

}

}

if(par->data<temp->data)

{

par->right=temp;

}

else

{

par->left=temp;

}

}

}

void bst::inorder()

{

inorder1(root);

}

void bst::inorder1(node \*temp)

{

if(temp!=NULL)

{

inorder1(temp->left);

cout<<" "<<temp->data;

inorder1(temp->right);

}

}

int bst::search(int x)

{

node \*curr;

if(root==NULL)

return 0;

curr=root;

while(curr!=NULL)

{

if(curr->data==x)

return 1;

if(curr->data>x)

curr=curr->left;

if(curr->data<x)

curr=curr->right;

}

return 0;

}

void bst::rec\_search(node \*T,int x)

{

if(T!=NULL)

{

rec\_search(T->left,x);

if(T->data==x)

{

cout<<"Search Found\n";

}

rec\_search(T->right,x);

}

}

node\* bst::mirror(node\* t)

{

node\* temp;

if (t==NULL)

return NULL;

else

{

temp = new node(t->data);

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

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

return temp;

}

}

void bst::displayMirror()

{

node \*temproot;

temproot = mirror(root);

cout << "Inorder traversal of mirrored BST" << endl;

inorder1(temproot);

}

void bst::dele()

{

int i;

cout<<"Enter the data to delete:";

cin>>i;

root=delete\_node(root,i);

}

node\* bst::delete\_node(node \*T,int j)

{

node \*temp;

if(T==NULL)

{

cout<<"Delete key not found...\n";

return T;

}

if(j<T->data)

{

T->left=delete\_node(T->left,j);

return T;

}

if(j>T->data)

{

T->right=delete\_node(T->right,j);

return T;

}

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

{

temp=T;

free(temp);

cout<<"Deleted "<<j<<"successfully(leaf node)\n";

return NULL;

}

if(T->left==NULL)

{

temp=T;

T=T->right;

free(temp);

cout<<"Deleted "<<j<<" successfully(node with one child ->right)\n";

return T;

}

if(T->right==NULL)

{

temp=T;

T=T->left;

free(temp);

cout<<"Deleted "<<j<<" successfully(node with one child ->left)\n";

return T;

}

temp=T->right;

while(temp->left!=NULL)

temp=temp->left;

T->data = temp->data;

T->right = delete\_node(T->right,temp->data);

cout<<"Deleted "<<j<<" successfully (node with two children)"<<endl;

return T;

}

void bst::min()//minimum value of node

{

node \*p=root;

while(p->left!=NULL)

{

p=p->left;

}

cout<<"Minimum of Tree:"<<p->data;

}

void bst::max()

{

node \*p=root;

while(p->right!=NULL)

{

p=p->right;

}

cout<<"Maximum of Tree:"<<p->data;

}

int bst::heightp(node \*T) // height of tree

{

if(T==NULL)

return 0;

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

return 1;

int lh=heightp(T->left);

int rh=heightp(T->right);

if(lh>rh)

return lh+1;

else

return rh+1;

}

class queue //class queue

{

int front,rear;

node \*data[size];

public:

queue();

int isfull();

int isEmpty();

void enqueue(node \*entry);

node \* dequeue();

};

queue::queue() //constructor

{

rear=front=-1;

}

int queue::isfull()

{

if(front==size-1)

return 1;

return 0;

}

int queue::isEmpty()

{

if(front==-1)

{

return 1;

}

return 0;

}

void queue::enqueue(node \*entry)

{

if (rear==-1 && front==-1)

{

rear=front=0;

data[rear]=entry;

}

else

{

rear++;

data[rear] = entry;

}

}

node \*queue::dequeue()

{

node \*temp;

if(front==rear)

{

temp=data[front];

front=rear=-1;

}

else

{

temp=data[front];

front++;

}

return temp;

}

void bst::level(node \*T) //levelwise display

{

node \*n;

queue q1;

q1.enqueue(T);

while(!q1.isEmpty())

{

n=q1.dequeue();

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

if(n->left!=NULL)

{

q1.enqueue(n->left);

}

if(n->right!=NULL)

{

q1.enqueue(n->right);

}

}

}

int main() //main function

{

bst b;

int a,ch,c,data,res;

do

{

cout<<"\n--------------------------------------------------\n";

cout<<"\n1.Insert\n2.Inoder\n3.Search\n4.Recursive search\n5.Delete\n6.Mirror\N7.Height\n8.Minimum\n9.Maximum\n10.Levelwise Display\n11.Exit\n\n";

cout<<"Enter your choice :\n";

cin>>ch;

switch(ch)

{

case 1:

do

{

cout<<"Enter the data:";

cin>>c;

b.insert(c);

cout<<"Do you want to insert 1/0:";

cin>>a;

}while(a);

break;

case 2:

b.inorder();

break;

case 3:

cout<<"Enter the data to search:";

cin>>data;

res=b.search(data);

if(res==1)

{

cout<<"Record found !!\n";

}

else

{

cout<<"Record not found !!\n";

}

break;

case 4:

cout<<"Enter the data to search:";

cin>>data;

b.searchrec(data);

break;

case 5:

b.dele();

break;

case 6:

b.displayMirror();

break;

case 7:

b.height();

break;

case 8:

b.min();

break;

case 9:

b.max();

break;

case 10:

cout<<"hi";

b.levelp();

break;

}

}while(ch!=11);

return 0;

}

**/\* OUTPUT :**

--------------------------------------------------

1.Insert

2.Inoder

3.Search

4.Recursive search

5.Delete

6.Mirror

7.Height

8.Minimum

9.Maximum

10.Levelwise Display

11.Exit

Enter your choice :

1

Enter the data:5

Do you want to insert 1/0:1

Enter the data:8

Do you want to insert 1/0:1

Enter the data:3

Do you want to insert 1/0:1

Enter the data:4

Do you want to insert 1/0:1

Enter the data:6

Do you want to insert 1/0:1

Enter the data:7

Do you want to insert 1/0:0

--------------------------------------------------

1.Insert

2.Inoder

3.Search

4.Recursive search

5.Delete

6.Mirror

7.Height

8.Minimum

9.Maximum

10.Levelwise Display

11.Exit

Enter your choice :

2

3 4 5 6 7 8

--------------------------------------------------

1.Insert

2.Inoder

3.Search

4.Recursive search

5.Delete

6.Mirror

7.Height

8.Minimum

9.Maximum

10.Levelwise Display

11.Exit

Enter your choice :

3

Enter the data to search:5

Record found !!

--------------------------------------------------

1.Insert

2.Inoder

3.Search

4.Recursive search

5.Delete

6.Mirror

7.Height

8.Minimum

9.Maximum

10.Levelwise Display

11.Exit

Enter your choice :

4

Enter the data to search:7

Search Found

--------------------------------------------------

1.Insert

2.Inoder

3.Search

4.Recursive search

5.Delete

6.Mirror

7.Height

8.Minimum

9.Maximum

10.Levelwise Display

11.Exit

Enter your choice :

5

Enter the data to delete:5

Deleted 6 successfully(node with one child ->right)

Deleted 5 successfully (node with two children)

--------------------------------------------------

1.Insert

2.Inoder

3.Search

4.Recursive search

5.Delete

6.Mirror

7.Height

8.Minimum

9.Maximum

10.Levelwise Display

11.Exit

Enter your choice :

2

3 4 6 7 8

--------------------------------------------------

1.Insert

2.Inoder

3.Search

4.Recursive search

5.Delete

6.Mirror

7.Height

8.Minimum

9.Maximum

10.Levelwise Display

11.Exit

Enter your choice :

6

Inorder traversal of mirrored BST

8 7 6 4 3

--------------------------------------------------

1.Insert

2.Inoder

3.Search

4.Recursive search

5.Delete

6.Mirror

7.Height

8.Minimum

9.Maximum

10.Levelwise Display

11.Exit

Enter your choice :

7

Height:3

--------------------------------------------------

1.Insert

2.Inoder

3.Search

4.Recursive search

5.Delete

6.Mirror

7.Height

8.Minimum

9.Maximum

10.Levelwise Display

11.Exit

Enter your choice :

8

Minimum of Tree:3

--------------------------------------------------

1.Insert

2.Inoder

3.Search

4.Recursive search

5.Delete

6.Mirror

7.Height

8.Minimum

9.Maximum

10.Levelwise Display

11.Exit

Enter your choice :

9

Maximum of Tree:8

--------------------------------------------------

1.Insert

2.Inoder

3.Search

4.Recursive search

5.Delete

6.Mirror

7.Height

8.Minimum

9.Maximum

10.Levelwise Display

11.Exit

Enter your choice :

10

hi 6 3 8 4 7

--------------------------------------------------

1.Insert

2.Inoder

3.Search

4.Recursive search

5.Delete

6.Mirror

7.Height

8.Minimum

9.Maximum

10.Levelwise Display

11.Exit

Enter your choice :

11

\*/