/\*PROGRAM SHOWING VARIOUS OPERATIONS ON BINARY SEARCH TREE\*/

#include<iostream>

#include<conio.h>

#include<stdlib.h>

using namespace std;

int max(int value1,int value2)

{

return((value1>value2)?value1:value2);

}

class BSTnode

{

public:

int data;

BSTnode \*left,\*right;

BSTnode(int x)

{

data=x;

left=right=NULL;

}

};

class BST

{

BSTnode \*root;

BSTnode\*delet1(BSTnode \*T,int);

int height1(BSTnode \*T);

BSTnode \*search1(BSTnode \*, int);

BSTnode \*mirror1(BSTnode \*);

public:

BST()

{

root=NULL;

}

BSTnode \*find(int x);

void insert(int x);

void create();

void level\_wise();

void mirror()

{

root=mirror1(root);

}

int height()

{

return(height1(root));

}

int mindata();

};

int BST::mindata()

{

BSTnode \*T;

T=root;

while(T->left!=NULL)

T=T->left;

if(T==NULL)

return -1;

return T->data;

}

int BST::height1(BSTnode \*T)

{

int x;

if(T==NULL)

return 0;

x=1+max(height1(T->left),height1(T->right));

return x;

}

class Q

{

BSTnode \*data[30];

int R,F;

public:

Q()

{

R=F=-1;

}

void init()

{

R=F=-1;

}

int empty()

{

if(R==-1)

return 1;

return 0;

}

void insert(BSTnode \*p)

{

if(empty())

R=F=0;

else

R=R+1;

data[R]=p;

}

BSTnode \*Delete()

{

BSTnode \*p=data[F];

if(R==F)

R=F=-1;

else F=F+1;

return(p);

}

};

BSTnode \*BST::mirror1(BSTnode \*T)

{

BSTnode \*temp;

if(T==NULL)

return NULL;

temp=T->left;

T->left=mirror1(T->right);

T->right=mirror1(temp);

return(T);

}

void BST::insert(int x)

{

BSTnode \*p,\*q,\*r;

r=new BSTnode(x);

if(root==NULL)

{

root=r;

return;

}

p=root;

while(p!=NULL)

{

q=p;

if(x>p->data)

p=p->right;

else

p=p->left;

}

if(x>q->data)

q->right=r;

else

q->left=r;

}

void BST::create()

{

int n,x,i;

root=NULL;

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

cin>>n;

cout<<"\n Enter tree values:";

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

{

cin>>x;

insert(x);

}

}

BSTnode \* BST::find(int x)

{

BSTnode \*T;

T=root;

while(T!=NULL)

{

if(x==T->data)

return(T);

if(x>T->data)

T=T->right;

else

T=T->left;

}

return(NULL);

}

void BST::level\_wise()

{

Q q1,q2;

BSTnode \*p1,\*p2;

BSTnode \*T=root;

if(T==NULL)

return;

q1.insert(T);

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

while(!q1.empty())

{

cout<<"\n";

q2.init();

while(!q1.empty())

{

p1=q1.Delete();

if(p1->left!=NULL)

{

q2.insert(p1->left);

cout<<" "<<p1->left->data;

}

if(p1->right!=NULL)

{

q2.insert(p1->right);

cout<<" "<<p1->right->data;

}

}

q1=q2;

}

}

int main()

{

BST bst;

BSTnode \*p;

int x,op;

do

{

cout<<"\n1)Create\n2)Number of nodes in longest path\n3)Search";

cout<<"\n4)Level wise traversal\n5)Minimum data in the tree";

cout<<"\n6)Insert\n7)Mirror";

cout<<"\n8)Quit";

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

cin>>op;

switch(op)

{

case 1:

bst.create();

break;

case 2:

cout<<"\n Number of nodes:"<<bst.height();

break;

case 3:

cout<<"\n Enter the key to searched:";

cin>>x;

p=bst.find(x);

if(p==NULL)

cout<<"\n \*\*\*\*\*\*\*Not Found\*\*\*\*\*\*\*\*";

else

cout<<"\n \*\*\*\*\*\*\*Found\*\*\*\*\*\*\*\*";

break;

case 4:

bst.level\_wise();

break;

case 5:

cout<<"\n Minimum data:"<<bst.mindata();

break;

case 6:

cout<<"\n Enter the data to be inserted:";

cin>>x;

bst.insert(x);

break;

case 7:

cout<<"\n Tree data before mirroring:";

bst.level\_wise();

bst.mirror();

cout<<"\n Tree data after mirror:";

bst.level\_wise();

break;

}

}

while(op!=8);

}

Output

/\*

1)Create

2)Number of nodes in longest path

3)Search

4)Level wise traversal

5)Minimum data in the tree

6)Insert

7)Mirror

8)Quit

Enter your choice:1

Enter the number of nodes:5

Enter tree values:10 8 2 9 15

1)Create

2)Number of nodes in longest path

3)Search

4)Level wise traversal

5)Minimum data in the tree

6)Insert

7)Mirror

8)Quit

Enter your choice:2

Number of nodes:3

1)Create

2)Number of nodes in longest path

3)Search

4)Level wise traversal

5)Minimum data in the tree

6)Insert

7)Mirror

8)Quit

Enter your choice:3

Enter the key to searched:8

\*\*\*\*\*\*\*Found\*\*\*\*\*\*\*\*

1)Create

2)Number of nodes in longest path

3)Search

4)Level wise traversal

5)Minimum data in the tree

6)Insert

7)Mirror

8)Quit

Enter your choice:4

10

8 15

2 9

1)Create

2)Number of nodes in longest path

3)Search

4)Level wise traversal

5)Minimum data in the tree

6)Insert

7)Mirror

8)Quit

Enter your choice:5

Minimum data:2

1)Create

2)Number of nodes in longest path

3)Search

4)Level wise traversal

5)Minimum data in the tree

6)Insert

7)Mirror

8)Quit

Enter your choice:6

Enter the data to be inserted:56

1)Create

2)Number of nodes in longest path

3)Search

4)Level wise traversal

5)Minimum data in the tree

6)Insert

7)Mirror

8)Quit

Enter your choice:4

10

8 15

2 9 56

1)Create

2)Number of nodes in longest path

3)Search

4)Level wise traversal

5)Minimum data in the tree

6)Insert

7)Mirror

8)Quit

Enter your choice:7

Tree data before mirroring:

10

8 15

2 9 56

Tree data after mirror:

10

15 8

56 9 2

1)Create

2)Number of nodes in longest path

3)Search

4)Level wise traversal

5)Minimum data in the tree

6)Insert

7)Mirror

8)Quit

Enter your choice:8

Process returned 0 (0x0) execution time : 87.187 s

Press any key to continue.