#include<iostream>

#include<cmath>

#include<conio.h>

using namespace std;

struct node

{

int info;

node \*left,\*right;

};

class cbt

{

private:

node \*temp,\*temp1,\*temp3,\*current;

int maxnode,mininode,hightnode;

bool flag;

int number;

public:

node \*root;

int value;

int count ;

cbt::cbt()

{

root=temp=temp1=current=NULL;

count=NULL;

number=1;

flag=true;

}

int heght(node \*temp)

{

int step=0;

while(temp->left!=NULL)

{

temp=temp->left;

step++;

}

return step;

}

void update(node \*temp,double ht)

{

if(temp->left==NULL)

{

temp1=temp;

flag=true;

number=2;

return;

}

if(temp->right==NULL)

{

temp1=temp;

flag=false;

number=2;

return;

}

if(ht>1)

update(temp->left,ht-1);

if(ht>1 && number==1)

update(temp->right,ht-1);

}

void insert(node \*current)

{

number = 1;

if (root == NULL)

{

cout << "Root " << endl;

root = new node;

root->info = value;

root->left = root->right = NULL;

current = root;

count++;

return;

}

int h = heght(root);

int y = pow(2.0, h+1) - 1;

if (count == y)

{

current = root;

while (current->left != NULL)

{

current = current->left;

}

current->left = new node;

current->left->info = value;

current->left->left = current->left->right = NULL;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << "Bcz Node R Maximum,, so adding at end of left No : " << current->info << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

count++;

return;

}

update(root, h);

//------------------------------------------------------------

if (flag == true)

{

current = temp1;

current->left = new node;

current->left->info = value;

current->left->left = current->left->right = NULL;

cout << "Left of " << current->info << endl;

count++;

return;

}

if (flag == false)

{

current = temp1;

current->right = new node;

current->right->info = value;

current->right->left = current->right->right = NULL;

cout << "right of " << current->info << endl;

count++;

return;

}

}

void swapin(int &a,int &b)

{

int tem=a;

a=b;

b=tem;

return;

}

void perculateUp(node\*temp,double ht)

{

if(ht>=1)

{

if(temp->left!=NULL)

perculateUp(temp->left,ht-1);

if(temp->right!=NULL)

perculateUp(temp->right,ht-1);

}

if(temp->left!=NULL)

if(temp->info > temp->left->info)

{

int a=temp->info;

temp->info=temp->left->info;

temp->left->info=a;

return;

}

if(temp->right!=NULL)

if(temp->info > temp->right->info)

{

int a=temp->info;

temp->info=temp->right->info;

temp->right->info=a;

return;

}

}

void perculateDown(node\*temp)

{

if(temp==NULL)

{cout<<"\tNo Tree found\n\n"; return;}

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

{

if(temp->left->info >= temp->right->info)

{

swapin(temp->info,temp->left->info); temp3=temp->left;

if(temp->left!=NULL)

perculateDown(temp->left);

}

else

{

swapin(temp->info,temp->right->info); temp3=temp->right;

if(temp->right!=NULL)

perculateDown(temp->right);

}

}

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

{

if(temp->info >=temp->left->info || temp->info < temp->left->info)

{

swapin(temp->info,temp->left->info); temp3=temp->left;

}

}

}

void inorder(node \*temp)

{

if(root==NULL)

{

cout<<"Your tree is empty"<<endl;

return;

}

if(temp->left!=NULL)

inorder(temp->left);

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

if(temp->right!=NULL)

inorder(temp->right);

return;

}

void opt()

{

cout<<"\n\n \*\*\*\*\*\*\*\*\*\*\*\*\*\* Select Option \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*.\n";

cout<<"\n Enter any of choices.\n";

cout<<"\n 1 : For Adding (inserting) node in BST.\n";

cout<<"\n 2 : For Print the whole BST .\n";

cout<<"\n 3 :for mini heap the programme .\n";

cout<<"\n 4 :for max heap the programme .\n";

cout<<"\n 5 :for exit the programme .\n";

}

};

int main()

{

cbt obj;

//int k;

char ch;

while(1)

{

obj.opt();

cin>>ch;

switch(ch)

{

case '1':

cout<<"Please put the entry in the tree "<<endl;

cin>>obj.value;

obj.insert(obj.root);

break;

case '2':

obj.inorder(obj.root);

break;

case '3':

obj.perculateUp(obj.root,obj.heght(obj.root));

break;

case '4':

obj.perculateDown(obj.root);

break;

case '5':

exit(0);

break;

default :

exit(0);

break;

}

}

}