|  |
| --- |
|  |
| Data structure assignment 3 |
| **Roll no : f17-8195** |
| **usama rao**  **Section A** |
| **[Pick the date]** |

**Task 1:**

#include<iostream>

#include<conio.h>

#include<cstdio>

#include<sstream>

#include<algorithm>

using namespace std;

struct node

{

int data;

node \*left;

node \*right;

int hight;

};

node \*root = NULL;

void addnode(node \*n, node \*r);

node\* delition(node \*&root);

void CompleteBinaryTree(node \*root, node \*previous, int &compleate);

void deletenode(int d, node \*&root);

void findhight(int x, node \*root);

void insert(int d)

{

node \*n= new node;

n->data = d;

n->left = NULL;

n->right = NULL;

if (root == NULL)

{

root = n;

return;

}

else

{

addnode(n, root);

}

}

void addnode(node \*n,node \*r)

{

if (n->data < r->data)

{

if (r->left == NULL)

{

r->left =n;

}

else

{

addnode(n, r->left);

}

}

else

{

if (r->right == NULL)

{

r->right = n;

}

else

{

addnode(n, r->right);

}

}

}

void search(int x,node \*root)

{

if (root == NULL)

{

cout << "ERROR : your data not found" << endl;

return;

}

if (x < root->data)

{

search(x, root->left);

}

if (x > root->data)

{

search(x, root->right);

}

if (x == root->data)

{

cout << "your node in found : " <<root->data<< endl;

return;

}

}

void s(int x)

{

search(x, root);

}

void d(int d)

{

deletenode(d, root);

}

void deletenode(int d, node \*&root)

{

if (root == NULL)

{

cout << "ERROR : node not exit" << endl;

return;

}

if (d < root->data)

{

deletenode(d, root->left);

}

if (d > root->data)

{

deletenode(d, root->right);

}

if (d == root->data)

{

delition(root);

}

}

node\* delition(node \*&root)

{

if (root->left == NULL)

{

node \*temp = root;

root = root->right;

delete temp;

return root;

}

if (root->right == NULL)

{

node \*temp = root;

root = root->left;

delete temp;

return root;

}

node \*temp = root->right;

while (temp->left)

{

temp = temp->left;

}

temp->left = root->left;

node \*d = root;

root= root->right;

delete d;

return temp;

}

void siblings(int input, node \*root)

{

if (root == NULL)

{

cout << "ERROR : your node not Exit" << endl;

return;

}

if (input < root->data)

{

siblings(input, root->left);

}

if (input > root->data)

{

siblings(input, root->right);

}

if (input == root->data)

{

if (root->left && root->right)

{

cout << "siblings found that are " << endl;

cout << "left : " << root->left->data << endl;

cout << "right : " << root->right->data << endl;

return;

}

else{

cout << "this root not have siblings" << endl;

}

}

}

void sibling(int input)

{

siblings(input, root);

}

int hight(node \*root)

{

if (root == NULL)

{

return -1;

}

else

{

root->hight=max(hight(root->left), hight(root->right)) + 1;

return max(hight(root->left), hight(root->right)) + 1;

}

}

void h(int x)

{

cout<<"height of Root node : "<<hight(root)-1<<endl;

findhight(x, root);

}

void findhight(int x,node \*root)

{

if (x < root->data)

{

findhight(x, root->left);

}

if (x>root->data)

{

findhight(x, root->right);

}

if (x == root->data)

{

cout << "hight of perticular node : " <<root->hight<< endl;

return;

}

}

bool bst(node \*root, int minvalue, int maxvalue)

{

if (root == NULL)

{

return true;

}

else

{

if (root->data >= minvalue && root->data <= maxvalue && bst(root->left, minvalue, root->data) && bst(root->right, root->data,maxvalue))

{

return true;

}

else

{

return false;

}

}

}

int menimum(int &min);

int maximun(int &max);

void isbst()

{

int min = 0;

int max = 0;

menimum(min);

maximun(max);

if (bst(root, min, max) == 0)

{

cout << " not a bst " << endl;

}

else

{

cout << " its a bst " << endl;

}

}

int menimum(int &min )

{

node\*temp = root;

while (temp->left)

{

temp = temp->left;

}

return min=temp->data;

}

int maximun(int &max)

{

node\*temp = root;

while (temp->right)

{

temp = temp->right;

}

return max=temp->data;

}

void countleafnodes(node \*r,int& count)

{

if (r != NULL)

{

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

{

count++;

}

}

if (r == NULL)

{

return;

}

else

{

countleafnodes(r->left,count);

countleafnodes(r->right,count);

}

}

void countleaf()

{

int count = 0;

countleafnodes(root, count);

cout << "leave nodes are : "<<count << endl;

}

void iscompleate()

{

node \*previous = NULL;

int compleate=0;

CompleteBinaryTree(root, previous, compleate);

if (compleate==0)

{

cout << "its not a compleate binary tree " << endl;

}

else

{

cout << "its a compleate binary tree " << endl;

}

}

void CompleteBinaryTree(node \*root,node \*previous,int &compleate)

{

if (root->left == NULL && root->right != NULL) //why exception thrown on this condion?

{

compleate = 0;

}

if (root == NULL)

{

return;

}

else

{

previous = root;

CompleteBinaryTree(root->left, previous,compleate);

CompleteBinaryTree(root->right, previous,compleate);

}

}

int main()

{

int x=0;

cout << "press -1 for stop inserting : " << endl;

while (x != -1)

{

cout << "enter no : " ;

cin >> x;

insert(x);

}

cout << endl;

cout << "enter node which you want to find hight : ";

cin >> x;

h(x);

cout << "enetr node of which you want to deleate : ";

cin >> x;

d(x);

cout << "enetr node of which you want to see siblings : ";

cin >> x;

sibling(x);

countleaf();

cout << "is bst : "; isbst();

//cout << "is complete binary tree <<endl : " << iscompleate();

cout << "enter node which you want to find" << endl;

cin >> x;

s(x);

//countleaf();

cout << "enetr node of which you want to see siblings : ";

cin >> x;

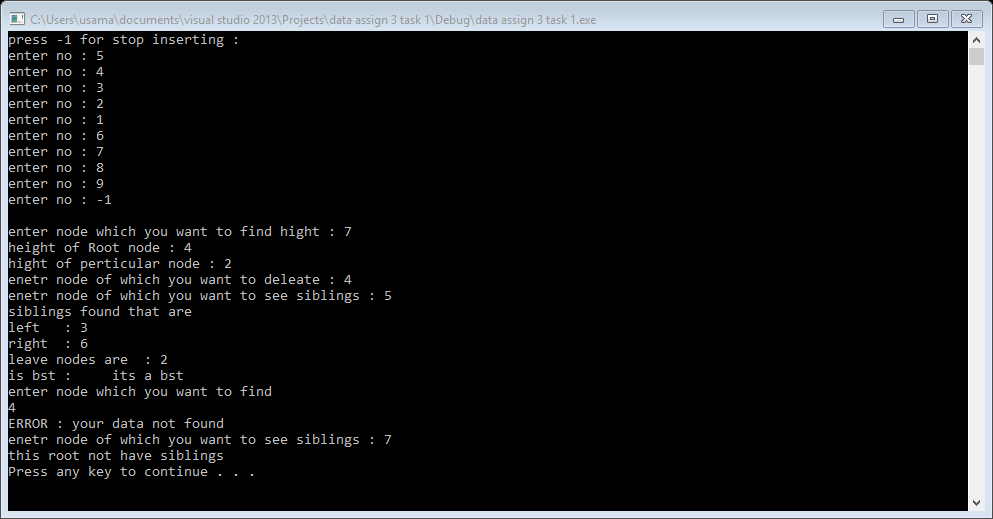
sibling(x);

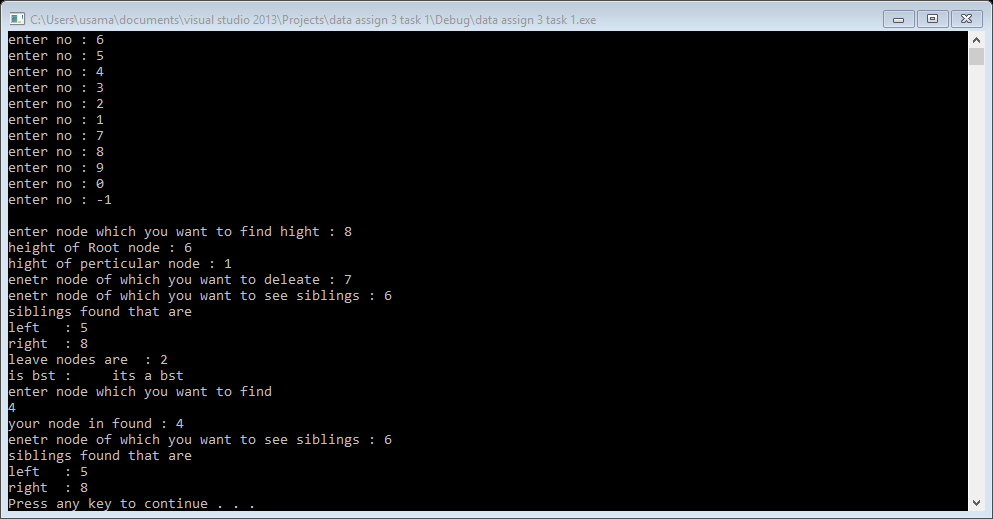
system("pause");

return 0;

}

**Result:**

****

****

**Task 2:**

#include<iostream>

#include<conio.h>

#include<cstdio>

#include<sstream>

#include<algorithm>

using namespace std;

struct node

{

int data;

node \*left;

node \*right;

int hight;

};

node \*root = NULL;

void addnode(node \*n, node \*r);

node\* delition(node \*&root);

void CompleteBinaryTree(node \*root, node \*previous, int &compleate);

void deletenode(int d, node \*&root);

void findhight(int x, node \*root);

void insert(int d)

{

node \*n= new node;

n->data = d;

n->left = NULL;

n->right = NULL;

if (root == NULL)

{

root = n;

return;

}

else

{

addnode(n, root);

}

}

void addnode(node \*n,node \*r)

{

if (n->data < r->data)

{

if (r->left == NULL)

{

r->left =n;

}

else

{

addnode(n, r->left);

}

}

else

{

if (r->right == NULL)

{

r->right = n;

}

else

{

addnode(n, r->right);

}

}

}

void search(int x,node \*root)

{

if (root == NULL)

{

cout << "ERROR : your data not found" << endl;

return;

}

if (x < root->data)

{

search(x, root->left);

}

if (x > root->data)

{

search(x, root->right);

}

if (x == root->data)

{

cout << "your node in found : " <<root->data<< endl;

return;

}

}

void s(int x)

{

search(x, root);

}

void d(int d)

{

deletenode(d, root);

}

void deletenode(int d, node \*&root)

{

if (root == NULL)

{

cout << "ERROR : node not exit" << endl;

return;

}

if (d < root->data)

{

deletenode(d, root->left);

}

if (d > root->data)

{

deletenode(d, root->right);

}

if (d == root->data)

{

delition(root);

}

}

node\* delition(node \*&root)

{

if (root->left == NULL)

{

node \*temp = root;

root = root->right;

delete temp;

return root;

}

if (root->right == NULL)

{

node \*temp = root;

root = root->left;

delete temp;

return root;

}

node \*temp = root->right;

while (temp->left)

{

temp = temp->left;

}

temp->left = root->left;

node \*d = root;

root= root->right;

delete d;

return temp;

}

void siblings(int input, node \*root)

{

if (root == NULL)

{

cout << "ERROR : your node not Exit" << endl;

return;

}

if (input < root->data)

{

siblings(input, root->left);

}

if (input > root->data)

{

siblings(input, root->right);

}

if (input == root->data)

{

if (root->left && root->right)

{

cout << "siblings found that are " << endl;

cout << "left : " << root->left->data << endl;

cout << "right : " << root->right->data << endl;

return;

}

else{

cout << "this root not have siblings" << endl;

}

}

}

void sibling(int input)

{

siblings(input, root);

}

/\*

int Max(node \*root)

{

if (root->right->hight > root->left->hight)

{

return (root->right->hight);

}

else

{

return (root->left->hight);

}

}\*/

int hight(node \*root)

{

if (root == NULL)

{

return -1;

}

else

{

root->hight=max(hight(root->left), hight(root->right)) + 1;

return max(hight(root->left), hight(root->right)) + 1;

}

}

void h(int x)

{

cout<<"height of Root node : "<<hight(root)-1<<endl;

findhight(x, root);

}

void findhight(int x,node \*root)

{

if (x < root->data)

{

findhight(x, root->left);

}

if (x>root->data)

{

findhight(x, root->right);

}

if (x == root->data)

{

cout << "hight of perticular node : " <<root->hight<< endl;

return;

}

}

bool bst(node \*root, int minvalue, int maxvalue)

{

if (root == NULL)

{

return true;

}

else

{

if (root->data >= minvalue && root->data <= maxvalue && bst(root->left, minvalue, root->data) && bst(root->right, root->data,maxvalue))

{

return true;

}

else

{

return false;

}

}

}

int menimum(int &min);

int maximun(int &max);

void isbst()

{

int min = 0;

int max = 0;

menimum(min);

maximun(max);

if (bst(root, min, max) == 0)

{

cout << " not a bst " << endl;

}

else

{

cout << " its a bst " << endl;

}

}

int menimum(int &min )

{

node\*temp = root;

while (temp->left)

{

temp = temp->left;

}

return min=temp->data;

}

int maximun(int &max)

{

node\*temp = root;

while (temp->right)

{

temp = temp->right;

}

return max=temp->data;

}

void countleafnodes(node \*r,int& count)

{

if (r != NULL)

{

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

{

count++;

}

}

if (r == NULL)

{

return;

}

else

{

countleafnodes(r->left,count);

countleafnodes(r->right,count);

}

}

void countleaf()

{

int count = 0;

countleafnodes(root, count);

cout << "leave nodes are : "<<count << endl;

}

void iscompleate()

{

node \*previous = NULL;

int compleate=0;

CompleteBinaryTree(root, previous, compleate);

if (compleate==0)

{

cout << "its not a compleate binary tree " << endl;

}

else

{

cout << "its a compleate binary tree " << endl;

}

}

void CompleteBinaryTree(node \*root,node \*previous,int &compleate)

{

if (root->left == NULL && root->right != NULL) //why exception thrown on this condion?

{

compleate = 0;

}

if (root == NULL)

{

return;

}

else

{

previous = root;

CompleteBinaryTree(root->left, previous,compleate);

CompleteBinaryTree(root->right, previous,compleate);

}

}

int main()

{

int x=0;

cout << "press -1 for stop inserting : " << endl;

while (x != -1)

{

cout << "enter no : " ;

cin >> x;

insert(x);

}

cout << endl;

cout << "enter node which you want to find hight : ";

cin >> x;

h(x);

cout << "enetr node of which you want to deleate : ";

cin >> x;

d(x);

cout << "enetr node of which you want to see siblings : ";

cin >> x;

sibling(x);

countleaf();

cout << "is bst : "; isbst();

//cout << "is complete binary tree <<endl : " << iscompleate();

cout << "enter node which you want to find" << endl;

cin >> x;

s(x);

//countleaf();

cout << "enetr node of which you want to see siblings : ";

cin >> x;

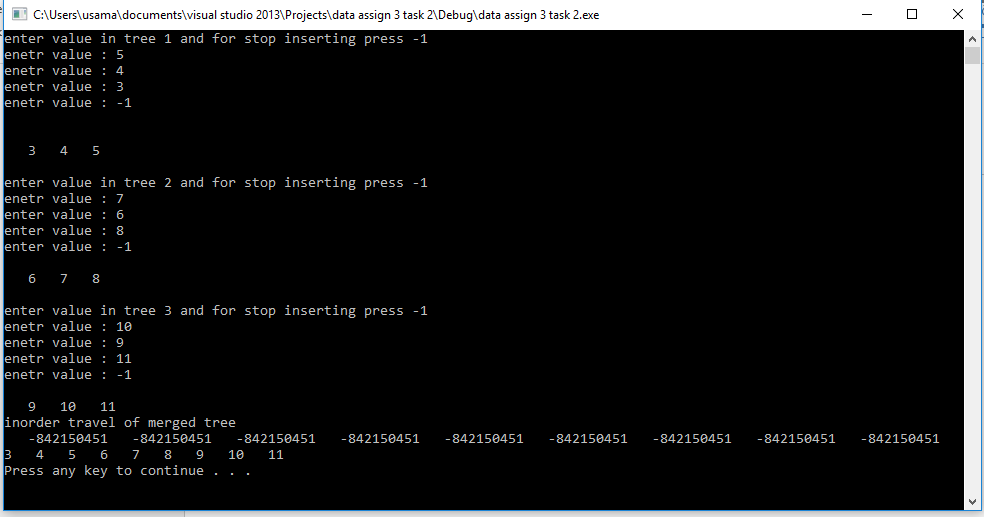
sibling(x);

system("pause");

return 0;

}

**Result:**



**Task 3:**

#include<iostream>

#include<conio.h>

#include<algorithm>

using namespace std;

struct avlnode

{

int data;

avlnode \*left;

avlnode \*right;

int hight;

};

avlnode \*root = NULL;

int hight(avlnode \*p)

{

if (p == NULL)

{

return -1;

}

else

{

return p->hight;

}

}

avlnode \*SingleRightRotation(avlnode \*&p)

{

avlnode \*t = p->left;

p->left = t->right;

t->right = p;

p->hight = max(hight(p->left), hight(p->right)) + 1;

t->hight = max(hight(p->left), hight(p->right)) + 1;

return t; ////////////////////////////

}

avlnode \*SingleLeftRotation(avlnode \*&p)

{

avlnode \*t = p->right;

p->right = t->left;

t->left = p;

p->hight = max(hight(p->left), hight(p->right)) + 1;

t->hight = max(hight(p->left), hight(p->right)) + 1;

return t; ///////////////////////////

}

avlnode \*DoubleRightLeftRotation(avlnode \*&p)

{

p->right = SingleRightRotation(p->right);

return SingleLeftRotation(p);

}

avlnode \*DoubleLeftRightRotation(avlnode \*&p)

{

p->left = SingleLeftRotation(p->left);

return SingleRightRotation(p);

}

avlnode \*insert(int d, avlnode \*&t)

{

if (t == NULL)

{

t = new avlnode;

t->left = NULL;

t->right = NULL;

t->data = d;

t->hight = 0;

}

if (d < t->data)

{

/////insert(d, t->left);

t->left = insert(d, t->left);

if (hight(t->left) - hight(t->right) == 2)

{

if (d < t->left->data)

{

/////SingleRightRotation(t);

t=SingleRightRotation(t);

}

else

{

//////DoubleLeftRightRotation(t);

t = DoubleLeftRightRotation(t);

}

}

}

if (d>t->data)

{

/////insert(d, t->right);

t->right=insert(d, t->right);

if (hight(t->right) - hight(t->left) == 2)

{

if (d > t->right->data)

{

/////SingleLeftRotation(t);

t=SingleLeftRotation(t);

}

else

{

////DoubleRightLeftRotation(t);

t = DoubleRightLeftRotation(t);

}

}

}

t->hight = max(hight(t->left), hight(t->right)) + 1;

return t;

}

void in(int d)

{

insert(d, root);

}

void inoder(avlnode \*t)

{

if (t == NULL)

{

return;

}

else

{

inoder(t->left);

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

inoder(t->right);

}

}

int main()

{

int x = 0;

cout << "press -1 for stop inserting : " << endl;

while (x != -1)

{

cout << "enter no : ";

cin >> x;

if (x == -1)

{

break;

}

in(x);

}

cout<<"inoder of avl tree : "<< endl;

inoder(root);

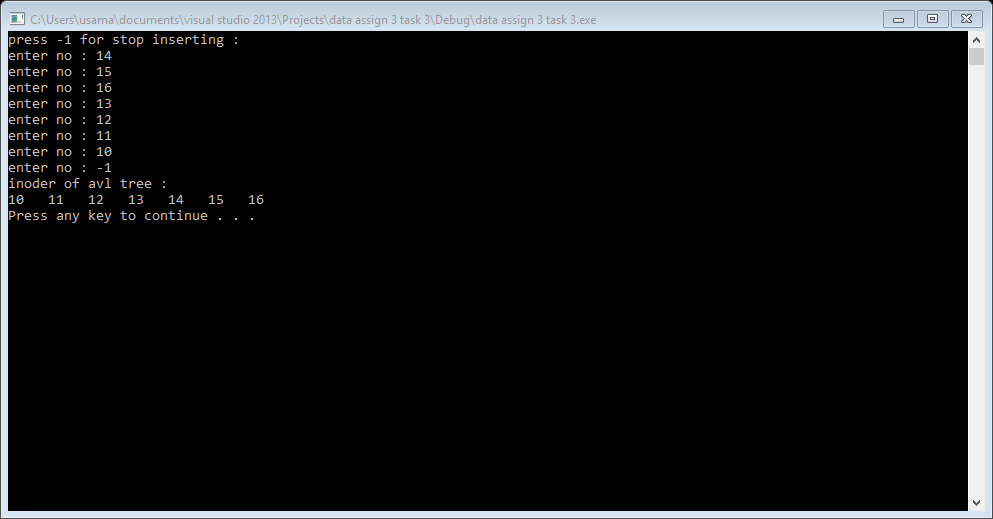
cout << endl;

system("pause");

return 0;

}

**Result:**

****

Task 4:

#include<iostream>

#include<conio.h>

using namespace std;

struct node

{

int data;

node \*left;

node \*right;

};

node \*root = NULL;

class queue

{

node \*array[10];

int front;

int rear;

public:

queue()

{

front = rear = -1;

}

bool isempty()

{

return (front == -1 && rear == -1);

}

bool isfull()

{

return ((rear + 1) % 10 == front);

}

void push(node \*x)

{

if (isfull())

{

cout << "ERROR : queue is full " <<endl;

return;

}

if (isempty())

{

front = rear = 0;

}

else

{

rear = (rear + 1) % 10;

}

array[rear] = x;

}

node \*pop()

{

if (isempty())

{

cout << "ERROR : Queue is empty " << endl;

return NULL;

}

if (front == rear)

{

front = rear = -1;

}

else

{

node \*c = array[front];

front = (front + 1) % 10;

return c;

}

}

};

void addnode(node \*n, node \*r);

void insert(int d)

{

node \*n = new node;

n->data = d;

n->left = NULL;

n->right = NULL;

if (root == NULL)

{

root = n;

return;

}

else

{

addnode(n, root);

}

}

void addnode(node \*n, node \*r)

{

if (n->data < r->data)

{

if (r->left == NULL)

{

r->left = n;

}

else

{

addnode(n, r->left);

}

}

else

{

if (r->right == NULL)

{

r->right = n;

}

else

{

addnode(n, r->right);

}

}

}

void levelorder(node \*root)

{

if (root == NULL)

{

return;

}

queue q;

q.push(root);

while (!q.isempty())

{

node \*current = q.pop();

//q.pop();

cout << current->data<<" ";

if (current->left)

{

q.push(current->left);

}

if (current->right)

{

q.push(current->right);

}

}

}

int main()

{

int x = 0;

cout << "press -1 for stop inserting : " << endl;

while (x != -1)

{

cout << "enter no : ";

cin >> x;

if (x == -1)

{

break;

}

insert(x);

}

cout << endl;

levelorder(root);

cout << endl;

system("pause");

return 0;

}