Question 2

#include <iostream>

#include<vector>

using namespace std;

template <typename t>

class queue

{

public:

t data;

queue<t> \*next;

queue(t d)

{

this->data = d;

this->next = NULL;

}

void push(queue<t> \*&s, t d)

{

queue<t> \*n = new queue<t>(d);

if (s == NULL)

{

s = n;

return;

}

else

{

queue<t> \*temp = s;

while (temp->next != NULL)

{

temp = temp->next;

}

temp->next = n;

}

}

void pop(queue<t> \*&s)

{

if (s != NULL)

{

queue<t> \*to = s;

s = s->next;

delete to;

}

}

t top(queue<t> \*&s)

{

if (s != NULL)

{

return s->data;

}

return NULL;

}

bool isempty(queue<t> \*s)

{

return s == NULL ? true : false;

}

};

class tree

{

public:

int data;

tree \*left;

tree \*right;

tree(int d)

{

this->left = NULL;

this->right = NULL;

this->data = d;

}

tree \*insert(tree \*root)

{

int d;

cout << "enter the data (-1 for NULL) : ";

cin >> d;

if (d == -1)

{

return NULL;

}

root = new tree(d);

cout << "for the left of " << d << " ";

root->left = insert(root->left);

cout << "for the right of " << d << " ";

root->right = insert(root->right);

return root;

}

void delete\_deepest(tree \*&root, tree \*temp)

{

queue<tree \*> \*q = NULL;

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

{

root = NULL;

return;

}

q->push(q, root);

while (!q->isempty(q))

{

tree \*f = q->top(q);

q->pop(q);

if (f == temp)

{

delete temp;

return;

}

if (f->left)

{

if (f->left == temp)

{

f->left = NULL;

delete temp;

return;

}

else

q->push(q, f->left);

}

if (f->right)

{

if (f->right == temp)

{

f->right = NULL;

delete temp;

return;

}

else

q->push(q, f->right);

}

}

}

void del(tree \*&root, int key)

{

if (root->data == key && root->left == NULL && root->right == NULL)

{

root = NULL;

return;

}

tree \*target = NULL;

tree \*deepest = NULL;

queue<tree \*> \*q = NULL;

q->push(q, root);

while (!q->isempty(q))

{

tree \*f = q->top(q);

q->pop(q);

if (f->data == key)

{

target = f;

}

deepest = f;

if (f->left)

{

q->push(q, f->left);

}

if (f->right)

{

q->push(q, f->right);

}

}

if (target == NULL)

{

cout << "key is not present \n";

return;

}

if (target == deepest)

{

delete\_deepest(root, target);

return;

}

int x = deepest->data;

delete\_deepest(root, deepest);

if (root == NULL)

return;

target->data = x;

}

void level\_order(tree \*&root)

{

if (root == NULL)

{

cout << "NULL";

return;

}

queue<tree \*> \*q = NULL;

q->push(q, root);

q->push(q, NULL);

while (!q->isempty(q))

{

tree \*f = q->top(q);

q->pop(q);

if (f == NULL)

{

if (!q->isempty(q))

{

q->push(q, NULL);

}

cout << "\n";

}

else

{

cout << f->data << " ";

if (f->left)

{

q->push(q, f->left);

}

if (f->right)

{

q->push(q, f->right);

}

}

}

}

void del\_k(tree \* & root,int k)

{

if(root==NULL) return;

queue<tree \*> \*q = NULL;

q->push(q, root);

vector<int> a;

q->push(q, NULL);

int c=0;

while (!q->isempty(q))

{

tree \*f = q->top(q);

q->pop(q);

if (f == NULL)

{

if (!q->isempty(q))

{

q->push(q, NULL);

}

c++;

}

else

{

if(c==k)

{

a.push\_back(f->data);

}

if (f->left)

{

q->push(q, f->left);

}

if (f->right)

{

q->push(q, f->right);

}

}

}

for(int i=0;i<a.size();i++)

{

root->del(root,a[i]);

}

}

void find\_n(tree \* root, tree \* & t, int key)

{

if(root==NULL)

{

return;

}

if(root->data==key)

{

t=root;

return;

}

find\_n(root->left,t,key);

find\_n(root->right,t,key);

}

tree \* find\_node(tree \* root,int key)

{

tree \* ans=NULL;

find\_n(root,ans,key);

return ans;

}

};

int main()

{

tree \*root = NULL;

root = root->insert(root);

cout << "\n level order: \n";

root->level\_order(root);

int t;

cout << "enter the data of the nodes whoese subtree wants to display (-1 to discontinue ) ";

cin>>t;

while (t != -1)

{

tree \* f=root->find\_node(root,t);

if(f!=NULL)

{

cout << "\n level order of a paricular subtree : \n";

root->level\_order(f);

}

else cout<<"\nthe node is not present .";

cout << "\nenter the data of the nodes whoese subtree wants to display (-1 to discontinue ) ";

cin>>t;

}

cout << "enter the data for delete (-1 to discontinue ) ";

cin>>t;

while (t != -1)

{

root->del(root, t);

cout << "\n level order after deletion : \n";

root->level\_order(root);

cout << "\nenter the data for delete (-1 to discontinue ) ";

cin>>t;

}

cout << "enter the of the level whose all nodes to be deleted (-1 to discontinue ) ";

cin>>t;

while(t!=-1 && root!=NULL)

{

root->del\_k(root, t);

cout << "\n level order after deletion : \n";

root->level\_order(root);

cout << "\nenter the value of level (-1 to discontinue ) ";

cin>>t;

}

if(root==NULL)

{

cout<<"ALL ELEMENTS ARE DELETED .";

}

}

Question 3

#include <iostream>

#include <vector>

using namespace std;

template <typename t>

class queue

{

public:

t data;

queue<t> \*next;

queue(t d)

{

this->data = d;

this->next = NULL;

}

void push(queue<t> \*&s, t d)

{

queue<t> \*n = new queue<t>(d);

if (s == NULL)

{

s = n;

return;

}

else

{

queue<t> \*temp = s;

while (temp->next != NULL)

{

temp = temp->next;

}

temp->next = n;

}

}

void pop(queue<t> \*&s)

{

if (s != NULL)

{

queue<t> \*to = s;

s = s->next;

delete to;

}

}

t top(queue<t> \*&s)

{

if (s != NULL)

{

return s->data;

}

return NULL;

}

bool isempty(queue<t> \*s)

{

return s == NULL ? true : false;

}

};

class bst

{

void solve(bst \*root, int &ans)

{

if (root == NULL)

return;

ans++;

solve(root->left, ans);

solve(root->right, ans);

}

void solve1(bst \*root, int &ans)

{

if (root == NULL)

return;

ans += root->data;

solve1(root->left, ans);

solve1(root->right, ans);

}

int balance\_factor(bst \*root)

{

if (root == NULL)

return -1;

return 1 + balance\_factor(root->left) + balance\_factor(root->right);

}

void avl(bst \*root, bool &c)

{

if (root == NULL)

{

return;

}

avl(root->left, c);

if (balance\_factor(root) < -1 || balance\_factor(root) > 1)

{

c = false;

return;

}

avl(root->right, c);

}

void convert(bst \*&root, int &s)

{

if (root == NULL)

return;

convert(root->right, s);

s += root->data;

root->data = s - root->data;

convert(root->left, s);

}

public:

int data;

bst \*right;

bst \*left;

bst(int d)

{

this->data = d;

this->right = NULL;

this->left = NULL;

}

bst \*insert\_into\_bst(bst \*root, int d)

{

if (root == NULL)

{

bst \*n = new bst(d);

return n;

}

if (root->data < d)

{

root->right = insert\_into\_bst(root->right, d);

}

else

{

root->left = insert\_into\_bst(root->left, d);

}

return root;

}

void insert(bst \*&root)

{

int d;

cout << "enter the data: (-1 to stop) ";

cin >> d;

while (d != -1)

{

root = insert\_into\_bst(root, d);

cout << "enter the data: (-1 to stop) ";

cin >> d;

}

}

void inorder(bst \*root)

{

if (root == NULL)

return;

inorder(root->left);

cout << root->data << " ";

inorder(root->right);

}

void level\_order(bst \*&root)

{

if (root == NULL)

{

cout << "NULL";

return;

}

queue<bst \*> \*q = NULL;

q->push(q, root);

q->push(q, NULL);

while (!q->isempty(q))

{

bst \*f = q->top(q);

q->pop(q);

if (f == NULL)

{

if (!q->isempty(q))

{

q->push(q, NULL);

}

cout << "\n";

}

else

{

cout << f->data << " ";

if (f->left)

{

q->push(q, f->left);

}

if (f->right)

{

q->push(q, f->right);

}

}

}

}

// part a

int number\_of\_node(bst \*root)

{

int ans = 0;

solve(root, ans);

return ans;

}

// part b

int weight(bst \*root)

{

int ans = 0;

solve1(root, ans);

return ans;

}

// part d

bool check\_avl(bst \*root)

{

bool c = true;

avl(root, c);

return c;

}

// part e

void convert\_into\_greater\_bst(bst \*&root)

{

int s = 0;

convert(root, s);

return;

}

};

int main()

{

bst \*root = NULL;

root->insert(root);

cout << "inorder:" << endl;

root->inorder(root);

cout << "\nnumber of nodes : " << root->number\_of\_node(root);

cout << "\nweight of tree : " << root->weight(root);

if (root->check\_avl(root))

{

cout << "\nthe tree is avl .";

}

else

cout << "\nthe tree is not avl.";

cout << "\n";

root->convert\_into\_greater\_bst(root);

cout << "inorder:" << endl;

root->inorder(root);

}