Lab a

Q1

#include <iostream>

using namespace std;

template <typename t>

class queue

{

public:

t data;

queue<t> \*next;

queue(t d)

{

this->next = NULL;

this->data = d;

}

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

{

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

if (q == NULL)

{

q = n;

return;

}

else

{

queue<t> \*temp = q;

while (temp->next != NULL)

{

temp = temp->next;

}

temp->next = n;

}

}

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

{

if (s != NULL)

{

queue<t> \*a = s;

s = s->next;

delete a;

}

}

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

{

if (s != NULL)

{

return s->data;

}

return NULL;

}

bool isempty(queue<t> \*&s)

{

return s == NULL ? true : false;

}

};

template <typename t>

class tree

{

public:

t data;

tree<t> \*right;

tree<t> \*left;

tree(t d)

{

this->data = d;

this->left = NULL;

this->right = NULL;

}

void insert(tree<t> \*&root, t d)

{

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

if (root == NULL)

{

root = n;

return;

}

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

q->push(q, root);

while (!q->isempty(q))

{

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

q->pop(q);

if (f->left == NULL)

{

f->left = n;

return;

}

if (f->right == NULL)

{

f->right = n;

return;

}

if (f->left)

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

if (f->right)

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

}

}

void level\_order(tree<t> \*root)

{

cout << "\nlevel order\n";

if (root == NULL)

{

cout << "NULL";

return;

}

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

q->push(q, root);

q->push(q, NULL);

while (!q->isempty(q))

{

tree<t> \*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

bool search(tree<t> \*root, t c)

{

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

q->push(q, root);

while (!q->isempty(q))

{

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

q->pop(q);

if (f->data == c)

{

return true;

}

if (f->left)

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

if (f->right)

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

}

return false;

}

// part b

void height(tree<t> \*root, int h, int &ans, t key)

{

if (root == NULL)

return;

if (root->data == key)

{

ans = h;

return;

}

height(root->left, h + 1, ans, key);

height(root->right, h + 1, ans, key);

}

int height\_of\_a\_given\_node(tree<t> \*root, t key)

{

int ans = -1;

height(root, 0, ans, key);

return ans;

}

// part c

void depth(tree<t> \*root, int h, int &ans, bool &c, t key)

{

if (root == NULL)

{

if (c)

{

c = false;

ans = h;

}

return;

}

if (root->data == key)

{

c = true;

}

if (c)

{

depth(root->left, h + 1, ans, c, key);

depth(root->right, h + 1, ans, c, key);

}

else

{

depth(root->left, 0, ans, c, key);

depth(root->right, 0, ans, c, key);

}

}

int depth\_of\_a\_given\_node(tree<t> \*root, t key)

{

int ans = -1;

bool c = false;

depth(root, 0, ans, c,key);

return ans;

}

// part d

void solve(tree<t>\* root,int& i)

{

if(root==NULL) return;

solve(root->left,i);

i++;

solve(root->right,i);

}

int number\_of\_node(tree<t> \* root)

{

int i=0;

solve(root,i);

return i;

}

bool complete\_bst(tree<t> \* root,int i,int n)

{

if(root==NULL) return true;

if(i>=n)

{

return false;

}

return complete\_bst(root->left,2\*i+1,n) && complete\_bst(root->right,2\*i+2,n);

}

};

int main()

{

tree<char> \*root = NULL;

int n;

cout << "enter the size of charater array : ";

cin >> n;

char arr[n];

cout << "enter the data : ";

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

{

cin >> arr[i];

root->insert(root, arr[i]);

}

root->level\_order(root);

int t;

cout<<"enter 1 for search(0 to skip): ";

cin>>t;

while(t)

{

char c;

cout<<"enter the key for search : ";

cin>>c;

if(root->search(root,c))

{

cout<<"the key is present .\n";

}

else cout<<"the key is not present .\n";

cout<<"enter 0 to discontinue searching(any other number to continue : )";

cin>>t;

}

cout<<"enter 1 for computing height of a specific node(0 to skip) : ";

cin>>t;

while(t)

{

char c;

cout<<"enter the key for search : ";

cin>>c;

cout<<"the height of the given node is: "<<root->height\_of\_a\_given\_node(root,c)<<"\n";

cout<<"enter 0 to discontinue (any other number to continue : )";

cin>>t;

}

cout<<"enter 1 for computing depth of a specific node(0 to skip) : ";

cin>>t;

while(t)

{

char c;

cout<<"enter the key for search : ";

cin>>c;

cout<<"the height of the given node is: "<<root->depth\_of\_a\_given\_node(root,c)<<"\n";

cout<<"enter 0 to discontinue (any other number to continue : )";

cin>>t;

}

int k=root->number\_of\_node(root);

if(root->complete\_bst(root,0,n))

{

cout<<"its a complete binary tree .";

}

else cout<<"its not a complete binary tree .";

}

Q2

#include<iostream>

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->data=d;

this->left=NULL;

this->right=NULL;

}

void level\_order(tree \* root)

{

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<<endl;

}

else

{

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

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

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

}

}

}

};

int search(int in[],int s,int e,int key)

{

for(int i=s;i<=e;i++)

{

if(in[i]==key) return i;

}

return -1;

}

int \* const\_lv(int in[],int s,int e,int lv[],int n)

{

int \* ans=new int [e-s+1];

int k=0;

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

{

if(search(in,s,e,lv[i])!=-1) ans[k++]=lv[i];

}

return ans;

}

tree \* const\_bt(int in[],int s,int e,int n, int lv[])

{

if(s>e) return NULL;

tree \* root = new tree(lv[0]);

if(s==e) return root;

int index=search(in,s,e,lv[0]);

int \*ll=const\_lv(in,s,index-1,lv,n);

int \*rl=const\_lv(in,index+1,e,lv,n);

root->left=const\_bt(in,s,index-1,index-s,ll);

root->right=const\_bt(in,index+1,e,e-index,rl);

return root;

}

int main()

{

int n;

cout<<"enter the lenght of the inorder or levelorder array : ";

cin>>n;

int in[n],lv[n];

cout<<"enter the inorder : ";

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

{

cin>>in[i];

}

cout<<"\nenter the levelorder : ";

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

{

cin>>lv[i];

}

tree \* root= const\_bt(in,0,n-1,n,lv);

root->level\_order(root);

}

Q3

#include<iostream>

using namespace std;

//treaded tree

class ttree

{

public:

int data;

ttree \* left, \* right;

bool lthread;

bool rthread;

ttree(int d)

{

this->data=d;

this->left=NULL;

this->right=NULL;

lthread=true;

rthread=true;

}

ttree \* insert(ttree \*root,int key)

{

ttree \* ptr=root;

ttree \* par=NULL;

while(ptr!=NULL)

{

if(key==ptr->data)

{

cout<<"duplicate key : ";

return root;

}

par=ptr;

if(key<ptr->data)

{

if(ptr->lthread==false)

{

ptr=ptr->left;

}

else break;

}

else

{

if(ptr->rthread==false)

{

ptr=ptr->right;

}

else break;

}

}

ttree \* n=new ttree(key);

if(par==NULL)

{

root=n;

}

else if(par->data>key)

{

n->left=par->left;

n->right=par;

par->lthread=false;

par->left=n;

}

else

{

n->right=par->right;

n->left=par;

par->rthread=false;

par->right=n;

}

return root;

}

ttree \* inorder\_successor(ttree \* root)

{

if(root->rthread==true)

{

return root->right;

}

root=root->right;

while(root->lthread==false)

{

root=root->left;

}

return root;

}

void inorder(ttree \* root)

{

if(root==NULL){ cout<<"tree empty\n"; return;}

ttree \* ptr=root;

while(ptr->lthread==false)

{

ptr=ptr->left;

}

while(ptr!=NULL)

{

cout<<ptr->data<<" ";

ptr=inorder\_successor(ptr);

}

}

};

int main()

{

ttree \* root=NULL;

int n;

cout<<"ENTER THE NUMBER OF NODES : ";

cin>>n;

cout<<"ENTER THE NODES : ";

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

{

int d;

cin>>d;

root =root -> insert(root, d);

}

cout<<"inorder traversal : ";

root->inorder(root);

}

Q4

#include <iostream>

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> \*& q)

{

return q==NULL ?true : false;

}

};

class bst

{

public:

int data;

bst \*right;

bst \*left;

bst(int d)

{

this->data = d;

this->right = NULL;

this->left = NULL;

}

// tree \* insert(tree \* root)

// {

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

// int d;

// 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);

// }

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

{

bst \* temp=root;

while(temp)

{

if(temp->data==d) {cout<<"\nduplicate data : "; return;}

if(temp->left)

{

if(temp->data>d)

{

temp=temp->left;

continue;

}

}

if(temp->left==NULL)

{

if(temp->data>d)

{

bst \* n=new bst(d);

temp->left=n;

return;

}

}

if(temp->right)

{

if(temp->data<d)

{

temp=temp->right;

continue;

}

}

if(temp->right==NULL)

{

if(temp->data<d)

{

bst \* n=new bst(d);

temp->right=n;

return;

}

}

}

}

void insert(bst \*&root)

{

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

int d;

cin >> d;

if(d==-1) {return;}

root=new bst (d);

while(d!=-1)

{

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

int d;

cin >> d;

if(d==-1) return;

insert\_into\_bst(root,d);

}

}

void level\_order(bst \* root)

{

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<<endl;

}

else

{

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

if(f->left)

{

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

}

if(f->right)

{

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

}

}

}

}

void inorder(bst \* root)

{

if(root==NULL) return ;

inorder(root->left);

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

inorder(root->right);

}

// greater sum function

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

{

if(root==NULL)

{

return;

}

solve(root->right,sum);

sum+=root->data;

root->data=sum-root->data;

solve(root->left,sum);

}

void greater\_sum(bst \*& root)

{

int sum=0;

solve(root , sum);

return ;

}

};

int main()

{

// queue<int> \* q= NULL;

// for(int i=0;i<10;i++)

// {

// q->push(q,i+1);

// }

// while(!q->isempty(q))

// {

// cout<<q->top(q)<<endl;

// q->pop(q);

// }

bst \* root=NULL;

root->insert(root);

//root->inorder(root);

root->level\_order(root);

cout<<endl;

root-> greater\_sum(root);

root->level\_order(root);

}

Q5

#include <iostream>

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> \*&q)

{

return q == NULL ? true : false;

}

};

class bst

{

public:

int data;

bst \*right;

bst \*left;

bst(int d)

{

this->data = d;

this->left = NULL;

this->right = NULL;

}

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

{

if (root == NULL)

{

root = new bst(d);

return root;

}

if (d > root->data)

{

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

}

else

{

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

}

}

void insert(bst \*&root)

{

int d;

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

cin >> d;

while (d != -1)

{

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

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

cin >> d;

}

}

void level\_order(bst \*root)

{

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 << endl;

}

else

{

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

if (f->left)

{

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

}

if (f->right)

{

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

}

}

}

}

//subtree

void find\_node(bst \* root, bst \* & t, int d)

{

if(root==NULL)

{

return;

}

if(root->data==d) {t=root; return;}

find\_node(root->left,t,d);

find\_node(root->right,t,d);

}

int sum (bst \* root)

{

if(root==NULL)

{

return 0;

}

int l=sum(root->left);

int r=sum(root->right);

return root->data+l+r;

}

void solve(bst \* root)

{

cout<<"enter the value of node for the calculation of left and right subtree sum: ";

int d;

cin>>d;

bst \* target=NULL;

find\_node(root,target,d);

int l = sum(root->left);

int r = sum(root->right);

if(l==r )

{

cout<<"sum of all the nodes of right subtree of "<<d<<" is equal then the sum of all nodes left subtree .\n";

}

else if(l>r)

{

cout<<"sum of all the nodes of right subtree of "<<d<<" is less then the sum of all nodes left subtree .\n";

}

else

{

cout<<"sum of all the nodes of right subtree of "<<d<<" is greater then the sum of all nodes left subtree .\n";

}

}

};

int main()

{

bst \* root=NULL;

root->insert(root);

root->level\_order(root);

cout<<endl;

root->solve(root);

}

Lab b

Q1

#include <iostream>

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> \*&q, t d)

{

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

if (q == NULL)

{

q = n;

return;

}

else

{

queue<t> \*temp = q;

while (temp->next != NULL)

{

temp = temp->next;

}

temp->next = n;

}

}

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

{

if (q != NULL)

{

queue<t> \*n = q;

q = q->next;

delete n;

}

}

t top(queue<t> \*s)

{

if (s != NULL)

{

return s->data;

}

return NULL;

}

bool isempty(queue<t> \*&s)

{

return s == NULL ? true : false;

}

};

class node

{

public:

int data;

node \*left;

node \*right;

node(int d)

{

this->data = d;

this->left = NULL;

this->right = NULL;

}

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

{

if (root == NULL)

{

return new node(d);

}

if (root->data < d)

{

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

}

else if (root->data > d)

{

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

}

else

{

cout << "\nduplicate data .\n ";

return root;

}

// return root;

}

void insert(node \*&root)

{

int d;

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

cin >> d;

while (d != -1)

{

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

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

cin >> d;

}

}

void level\_order(node \*root)

{

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

q->push(q, root);

q->push(q, NULL);

while (!q->isempty(q))

{

node \*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 inorder(node \*root)

{

if (root == NULL)

return;

inorder(root->left);

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

inorder(root->right);

}

void height\_cal(node \*root, int h, int &ans)

{

if (root == NULL)

{

ans = max(ans, h);

return;

}

height\_cal(root->left, h + 1, ans);

height\_cal(root->right, h + 1, ans);

}

int height\_c(node \*root)

{

int ans = 0;

height\_cal(root, 0, ans);

return ans;

}

int get\_balance\_factor(node \*root)

{

if (root == NULL)

return -1;

return height\_c(root->left) - height\_c(root->right);

}

node \*left\_r(node \*x)

{

node \*y = x->left;

node \*b = y->right;

y->right = x;

x->left = b;

return y;

}

node \*right\_r(node \*x)

{

node \*y = x->right;

node \*b = y->left;

y->left = x;

x->right = b;

return y;

}

node \*convert\_to\_avl(node \*root)

{

if (root == NULL)

return NULL;

root->left = convert\_to\_avl(root->left);

root->right = convert\_to\_avl(root->right);

int bf = get\_balance\_factor(root);

if (bf > 1 && get\_balance\_factor(root->left) >= 0)

{

return left\_r(root);

}

if (bf > 1 && get\_balance\_factor(root->left) < 0)

{

root->left = right\_r(root->left);

return left\_r(root);

}

if (bf < -1 && get\_balance\_factor(root->right) <= 0)

{

return right\_r(root);

}

if (bf < -1 && get\_balance\_factor(root->right) > 0)

{

root->right = left\_r(root->right);

return right\_r(root);

}

return root;

}

};

int main()

{

// queue<int> \* q=NULL;

// for(int i=0;i<10;i++)

// {

// q->push(q,i+1);

// }

// while(!q->isempty(q))

// {

// cout<<q->top(q);

// q->pop(q);

// }

node \*root = NULL;

root->insert(root);

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

root->level\_order(root);

cout << "in order :\n";

root->inorder(root);

root = root->convert\_to\_avl(root);

cout << "\ncoverted into avl : ";

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

root->level\_order(root);

cout << "in order :\n";

root->inorder(root);

}

Q3

#include <iostream>

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> \*&q, t d)

{

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

if (q == NULL)

{

q = n;

return;

}

else

{

queue<t> \*temp = q;

while (temp->next != NULL)

{

temp = temp->next;

}

temp->next = n;

}

}

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

{

if (q != NULL)

{

queue<t> \*n = q;

q = q->next;

delete n;

}

}

t top(queue<t> \*s)

{

if (s != NULL)

{

return s->data;

}

return NULL;

}

bool isempty(queue<t> \*&s)

{

return s == NULL ? true : false;

}

};

class node

{

public:

int data;

node \*left;

node \*right;

node(int d)

{

this->data = d;

this->left = NULL;

this->right = NULL;

}

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

{

if (root == NULL)

{

return new node(d);

}

if (root->data < d)

{

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

}

else if (root->data > d)

{

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

}

else

{

cout << "\nduplicate data .\n ";

return root;

}

// return root;

}

void insert(node \*&root)

{

int d;

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

cin >> d;

while (d != -1)

{

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

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

cin >> d;

}

}

void level\_order(node \*root)

{

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

q->push(q, root);

q->push(q, NULL);

while (!q->isempty(q))

{

node \*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 inorder(node \*root)

{

if (root == NULL)

return;

inorder(root->left);

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

inorder(root->right);

}

};

int main()

{

node \*root = NULL;

root->insert(root);

cout << "\nlevel order traversal :\n";

root->level\_order(root);

cout << "depth first (inorder) :\n";

root->inorder(root);

}

Q4

#include <iostream>

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> \*&q, t d)

{

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

if (q == NULL)

{

q = n;

return;

}

else

{

queue<t> \*temp = q;

while (temp->next != NULL)

{

temp = temp->next;

}

temp->next = n;

}

}

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

{

if (q != NULL)

{

queue<t> \*n = q;

q = q->next;

delete n;

}

}

t top(queue<t> \*s)

{

if (s != NULL)

{

return s->data;

}

return NULL;

}

bool isempty(queue<t> \*&s)

{

return s == NULL ? true : false;

}

};

class node

{

void height\_cal(node \*root, int h, int &ans)

{

if (root == NULL)

{

ans = max(ans, h);

return;

}

height\_cal(root->left, h + 1, ans);

height\_cal(root->right, h + 1, ans);

}

int get\_balance\_factor(node \*root)

{

if (root == NULL)

return -1;

return height\_c(root->left) - height\_c(root->right);

}

node \*left\_r(node \*x)

{

node \*y = x->left;

node \*b = y->right;

y->right = x;

x->left = b;

return y;

}

node \*right\_r(node \*x)

{

node \*y = x->right;

node \*b = y->left;

y->left = x;

x->right = b;

return y;

}

public:

int data;

node \*left;

node \*right;

node(int d)

{

this->data = d;

this->left = NULL;

this->right = NULL;

}

node \*insert\_into\_avl(node \*root, int d)

{

if (root == NULL)

{

return new node(d);

}

if (root->data < d)

{

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

}

else if (root->data > d)

{

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

}

else

{

cout << "\nduplicate data .\n ";

return root;

}

int bf = get\_balance\_factor(root);

if (bf > 1 && get\_balance\_factor(root->left) >= 0)

{

return left\_r(root);

}

if (bf > 1 && get\_balance\_factor(root->left) < 0)

{

root->left = right\_r(root->left);

return left\_r(root);

}

if (bf < -1 && get\_balance\_factor(root->right) <= 0)

{

return right\_r(root);

}

if (bf < -1 && get\_balance\_factor(root->right) > 0)

{

root->right = left\_r(root->right);

return right\_r(root);

}

return root;

}

int height\_c(node \*root)

{

int ans = 0;

height\_cal(root, 0, ans);

return ans;

}

void level\_order(node \*root)

{

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

q->push(q, root);

q->push(q, NULL);

while (!q->isempty(q))

{

node \*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 inorder(node \*root)

{

if (root == NULL)

return;

inorder(root->left);

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

inorder(root->right);

}

};

int main()

{

node \*root = NULL;

cout << "enter the data to insert into avl (-1 to exit): ";

int d;

cin >> d;

while (d != -1)

{

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

cout << "enter the data to insert into avl (-1 to exit): ";

cin >> d;

}

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

root->level\_order(root);

cout << "height of the avl tree : " << root->height\_c(root);

}

// 12 23 16 49 34 42 45 -1

Q5

#include <iostream>

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> \*&q, t d)

{

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

if (q == NULL)

{

q = n;

return;

}

else

{

queue<t> \*temp = q;

while (temp->next != NULL)

{

temp = temp->next;

}

temp->next = n;

}

}

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

{

if (q != NULL)

{

queue<t> \*n = q;

q = q->next;

delete n;

}

}

t top(queue<t> \*s)

{

if (s != NULL)

{

return s->data;

}

return NULL;

}

bool isempty(queue<t> \*&s)

{

return s == NULL ? true : false;

}

};

class node

{

void height\_cal(node \*root, int h, int &ans)

{

if (root == NULL)

{

ans = max(ans, h);

return;

}

height\_cal(root->left, h + 1, ans);

height\_cal(root->right, h + 1, ans);

}

int get\_balance\_factor(node \*root)

{

if (root == NULL)

return -1;

return height\_c(root->left) - height\_c(root->right);

}

node \*left\_r(node \*x)

{

node \*y = x->left;

node \*b = y->right;

y->right = x;

x->left = b;

return y;

}

node \*right\_r(node \*x)

{

node \*y = x->right;

node \*b = y->left;

y->left = x;

x->right = b;

return y;

}

public:

int data;

node \*left;

node \*right;

node(int d)

{

this->data = d;

this->left = NULL;

this->right = NULL;

}

node \*insert\_into\_avl(node \*root, int d)

{

if (root == NULL)

{

return new node(d);

}

if (root->data < d)

{

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

}

else if (root->data > d)

{

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

}

else

{

cout << "\nduplicate data .\n ";

return root;

}

int bf = get\_balance\_factor(root);

if (bf > 1 && get\_balance\_factor(root->left) >= 0)

{

return left\_r(root);

}

if (bf > 1 && get\_balance\_factor(root->left) < 0)

{

root->left = right\_r(root->left);

return left\_r(root);

}

if (bf < -1 && get\_balance\_factor(root->right) <= 0)

{

return right\_r(root);

}

if (bf < -1 && get\_balance\_factor(root->right) > 0)

{

root->right = left\_r(root->right);

return right\_r(root);

}

return root;

}

int height\_c(node \*root)

{

int ans = 0;

height\_cal(root, 0, ans);

return ans;

}

void level\_order(node \*root)

{

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

q->push(q, root);

q->push(q, NULL);

while (!q->isempty(q))

{

node \*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 inorder(node \*root)

{

if (root == NULL)

return;

inorder(root->left);

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

inorder(root->right);

}

void searching(node \*root, node \*&ans, int key)

{

if (root == NULL)

return;

if (root->data == key)

{

ans = root;

return;

}

searching(root->left, ans, key);

searching(root->right, ans, key);

}

node \*search(node \*root, int key)

{

node \*ans = NULL;

searching(root, ans, key);

return ans;

}

};

int main()

{

node \*root = NULL;

cout << "enter the data to insert into avl (-1 to exit): ";

int d;

cin >> d;

while (d != -1)

{

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

cout << "enter the data to insert into avl (-1 to exit): ";

cin >> d;

}

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

root->level\_order(root);

int key;

cout << "enter the key for searching :";

cin >> key;

node \*target = root->search(root, key);

if (target == NULL)

{

cout << "the key is not present : ";

}

else

{

cout << "the key is present :";

cout << "height of target : " << root->height\_c(target);

}

}

Q6

#include <iostream>

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> \*&q, t d)

{

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

if (q == NULL)

{

q = n;

return;

}

else

{

queue<t> \*temp = q;

while (temp->next != NULL)

{

temp = temp->next;

}

temp->next = n;

}

}

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

{

if (q != NULL)

{

queue<t> \*n = q;

q = q->next;

delete n;

}

}

t top(queue<t> \*s)

{

if (s != NULL)

{

return s->data;

}

return NULL;

}

bool isempty(queue<t> \*&s)

{

return s == NULL ? true : false;

}

};

class node

{

void height\_cal(node \*root, int h, int &ans)

{

if (root == NULL)

{

ans = max(ans, h);

return;

}

height\_cal(root->left, h + 1, ans);

height\_cal(root->right, h + 1, ans);

}

int get\_balance\_factor(node \*root)

{

if (root == NULL)

return -1;

return height\_c(root->left) - height\_c(root->right);

}

node \*left\_r(node \*x)

{

node \*y = x->left;

node \*b = y->right;

y->right = x;

x->left = b;

return y;

}

node \*right\_r(node \*x)

{

node \*y = x->right;

node \*b = y->left;

y->left = x;

x->right = b;

return y;

}

int min\_value(node \*root)

{

node \*r = root->right;

while (r->left != NULL)

r = r->left;

return r->data;

}

public:

int data;

node \*left;

node \*right;

node(int d)

{

this->data = d;

this->left = NULL;

this->right = NULL;

}

int height\_c(node \*root)

{

int ans = 0;

height\_cal(root, 0, ans);

return ans;

}

node \*insert\_into\_avl(node \*&root, int d)

{

if (root == NULL)

{

return new node(d);

}

if (root->data < d)

{

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

}

else if (root->data > d)

{

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

}

else

{

cout << "\nduplicate data .\n ";

return root;

}

int bf = get\_balance\_factor(root);

if (bf > 1 && get\_balance\_factor(root->left) >= 0)

{

return left\_r(root);

}

if (bf > 1 && get\_balance\_factor(root->left) < 0)

{

root->left = right\_r(root->left);

return left\_r(root);

}

if (bf < -1 && get\_balance\_factor(root->right) <= 0)

{

return right\_r(root);

}

if (bf < -1 && get\_balance\_factor(root->right) > 0)

{

root->right = left\_r(root->right);

return right\_r(root);

}

return root;

}

node \*delete\_form\_avl(node \*&root, int key)

{

if (root == NULL)

return root;

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

if (root->data < key)

{

root->right = delete\_form\_avl(root->right, key);

}

else if (root->data > key)

{

root->left = delete\_form\_avl(root->left, key);

}

else

{

if (root->left == NULL)

{

node \*a = root->right;

delete root;

return a;

}

else if (root->right == NULL)

{

node \*b = root->left;

delete root;

return b;

}

else if (root->left != NULL && root->right != NULL)

{

int miniv = min\_value(root);

root->data = miniv;

root->right = delete\_form\_avl(root->right, miniv);

}

}

if (root == NULL)

return root;

int bf = get\_balance\_factor(root);

if (bf > 1 && get\_balance\_factor(root->left) >= 0)

{

return left\_r(root);

}

if (bf > 1 && get\_balance\_factor(root->left) < 0)

{

root->left = right\_r(root->left);

return left\_r(root);

}

if (bf < -1 && get\_balance\_factor(root->right) <= 0)

{

return right\_r(root);

}

if (bf < -1 && get\_balance\_factor(root->right) > 0)

{

root->right = left\_r(root->right);

return right\_r(root);

}

return root;

}

void level\_order(node \*&root)

{

if (root == NULL)

{

cout << "NULL";

return;

}

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

q->push(q, root);

q->push(q, NULL);

while (!q->isempty(q))

{

node \*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 inorder(node \*&root)

{

if (root == NULL)

return;

inorder(root->left);

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

inorder(root->right);

}

};

int main()

{

node \*root = NULL;

cout << "enter the data to insert into avl (-1 to exit): ";

int d;

cin >> d;

while (d != -1)

{

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

cout << "enter the data to insert into avl (-1 to exit): ";

cin >> d;

}

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

root->level\_order(root);

int key;

cout << "enter the key for deletion (-1 to exit):";

cin >> key;

while (key != -1)

{

root = root->delete\_form\_avl(root, key);

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

root->level\_order(root);

cout << "enter the key for deletion (-1 to exit):";

cin >> key;

}

}

Q7

#include <iostream>

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> \*&q, t d)

{

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

if (q == NULL)

{

q = n;

return;

}

else

{

queue<t> \*temp = q;

while (temp->next != NULL)

{

temp = temp->next;

}

temp->next = n;

}

}

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

{

if (q != NULL)

{

queue<t> \*n = q;

q = q->next;

delete n;

}

}

t top(queue<t> \*s)

{

if (s != NULL)

{

return s->data;

}

return NULL;

}

bool isempty(queue<t> \*&s)

{

return s == NULL ? true : false;

}

};

class node

{

void height\_cal(node \*root, int h, int &ans)

{

if (root == NULL)

{

ans = max(ans, h);

return;

}

height\_cal(root->left, h + 1, ans);

height\_cal(root->right, h + 1, ans);

}

int get\_balance\_factor(node \*root)

{

if (root == NULL)

return -1;

return height\_c(root->left) - height\_c(root->right);

}

node \*left\_r(node \*x)

{

node \*y = x->left;

node \*b = y->right;

y->right = x;

x->left = b;

return y;

}

node \*right\_r(node \*x)

{

node \*y = x->right;

node \*b = y->left;

y->left = x;

x->right = b;

return y;

}

int min\_value(node \*root)

{

node \*r = root->right;

while (r->left != NULL)

r = r->left;

return r->data;

}

public:

int data;

node \*left;

node \*right;

node(int d)

{

this->data = d;

this->left = NULL;

this->right = NULL;

}

int height\_c(node \*root)

{

int ans = 0;

height\_cal(root, 0, ans);

return ans;

}

node \*insert\_into\_avl(node \*&root, int d)

{

if (root == NULL)

{

return new node(d);

}

if (root->data < d)

{

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

}

else if (root->data > d)

{

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

}

else

{

cout << "\nduplicate data .\n ";

return root;

}

int bf = get\_balance\_factor(root);

if (bf > 1 && get\_balance\_factor(root->left) >= 0)

{

return left\_r(root);

}

if (bf > 1 && get\_balance\_factor(root->left) < 0)

{

root->left = right\_r(root->left);

return left\_r(root);

}

if (bf < -1 && get\_balance\_factor(root->right) <= 0)

{

return right\_r(root);

}

if (bf < -1 && get\_balance\_factor(root->right) > 0)

{

root->right = left\_r(root->right);

return right\_r(root);

}

return root;

}

node \*delete\_subtree\_form\_avl(node \*&root, int key)

{

if (root == NULL)

return root;

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

if (root->data < key)

{

root->right = delete\_subtree\_form\_avl(root->right, key);

}

else if (root->data > key)

{

root->left = delete\_subtree\_form\_avl(root->left, key);

}

else

{

return NULL;

}

if (root == NULL)

return root;

int bf = get\_balance\_factor(root);

if (bf > 1 && get\_balance\_factor(root->left) >= 0)

{

return left\_r(root);

}

if (bf > 1 && get\_balance\_factor(root->left) < 0)

{

root->left = right\_r(root->left);

return left\_r(root);

}

if (bf < -1 && get\_balance\_factor(root->right) <= 0)

{

return right\_r(root);

}

if (bf < -1 && get\_balance\_factor(root->right) > 0)

{

root->right = left\_r(root->right);

return right\_r(root);

}

return root;

}

void level\_order(node \*&root)

{

if (root == NULL)

{

cout << "NULL\n";

return;

}

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

q->push(q, root);

q->push(q, NULL);

while (!q->isempty(q))

{

node \*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 inorder(node \*&root)

{

if (root == NULL)

return;

inorder(root->left);

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

inorder(root->right);

}

};

int main()

{

node \*root = NULL;

cout << "enter the data to insert into avl (-1 to exit): ";

int d;

cin >> d;

while (d != -1)

{

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

cout << "enter the data to insert into avl (-1 to exit): ";

cin >> d;

}

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

root->level\_order(root);

int key;

cout << "enter the data of first node of sutree for deletion (-1 to exit):";

cin >> key;

while (key != -1)

{

root = root->delete\_subtree\_form\_avl(root, key);

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

root->level\_order(root);

cout << "enter the data of first node of sutree for deletion (-1 to exit):";

cin >> key;

}

}

Q8

#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> \*&q, t d)

{

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

if (q == NULL)

{

q = n;

return;

}

else

{

queue<t> \*temp = q;

while (temp->next != NULL)

{

temp = temp->next;

}

temp->next = n;

}

}

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

{

if (q != NULL)

{

queue<t> \*n = q;

q = q->next;

delete n;

}

}

t top(queue<t> \*s)

{

if (s != NULL)

{

return s->data;

}

return NULL;

}

bool isempty(queue<t> \*&s)

{

return s == NULL ? true : false;

}

};

class node

{

void height\_cal(node \*root, int h, int &ans)

{

if (root == NULL)

{

ans = max(ans, h);

return;

}

height\_cal(root->left, h + 1, ans);

height\_cal(root->right, h + 1, ans);

}

int get\_balance\_factor(node \*root)

{

if (root == NULL)

return -1;

return height\_c(root->left) - height\_c(root->right);

}

node \*left\_r(node \*x)

{

node \*y = x->left;

node \*b = y->right;

y->right = x;

x->left = b;

return y;

}

node \*right\_r(node \*x)

{

node \*y = x->right;

node \*b = y->left;

y->left = x;

x->right = b;

return y;

}

int min\_value(node \*root)

{

node \*r = root->right;

while (r->left != NULL)

r = r->left;

return r->data;

}

public:

int data;

node \*left;

node \*right;

node(int d)

{

this->data = d;

this->left = NULL;

this->right = NULL;

}

int height\_c(node \*root)

{

int ans = 0;

height\_cal(root, 0, ans);

return ans;

}

node \*insert\_into\_avl(node \*&root, int d)

{

if (root == NULL)

{

return new node(d);

}

if (root->data < d)

{

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

}

else if (root->data > d)

{

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

}

else

{

cout << "\nduplicate data .\n ";

return root;

}

int bf = get\_balance\_factor(root);

if (bf > 1 && get\_balance\_factor(root->left) >= 0)

{

return left\_r(root);

}

if (bf > 1 && get\_balance\_factor(root->left) < 0)

{

root->left = right\_r(root->left);

return left\_r(root);

}

if (bf < -1 && get\_balance\_factor(root->right) <= 0)

{

return right\_r(root);

}

if (bf < -1 && get\_balance\_factor(root->right) > 0)

{

root->right = left\_r(root->right);

return right\_r(root);

}

return root;

}

node \*delete\_form\_avl(node \*&root, int key)

{

if (root == NULL)

return root;

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

if (root->data < key)

{

root->right = delete\_form\_avl(root->right, key);

}

else if (root->data > key)

{

root->left = delete\_form\_avl(root->left, key);

}

else

{

if (root->left == NULL)

{

node \*a = root->right;

delete root;

return a;

}

else if (root->right == NULL)

{

node \*b = root->left;

delete root;

return b;

}

else if (root->left != NULL && root->right != NULL)

{

int miniv = min\_value(root);

root->data = miniv;

root->right = delete\_form\_avl(root->right, miniv);

}

}

if (root == NULL)

return root;

int bf = get\_balance\_factor(root);

if (bf > 1 && get\_balance\_factor(root->left) >= 0)

{

return left\_r(root);

}

if (bf > 1 && get\_balance\_factor(root->left) < 0)

{

root->left = right\_r(root->left);

return left\_r(root);

}

if (bf < -1 && get\_balance\_factor(root->right) <= 0)

{

return right\_r(root);

}

if (bf < -1 && get\_balance\_factor(root->right) > 0)

{

root->right = left\_r(root->right);

return right\_r(root);

}

return root;

}

void level\_order(node \*&root)

{

if (root == NULL)

{

cout << "NULL";

return;

}

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

q->push(q, root);

q->push(q, NULL);

while (!q->isempty(q))

{

node \*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 deletion\_at\_k(node \*&root, int k)

{

if (root == NULL)

{

cout << "NULL\n";

return;

}

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

q->push(q, root);

q->push(q, NULL);

int c = 0;

vector<int> a;

while (!q->isempty(q))

{

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

q->pop(q);

if (f == NULL)

{

if (!q->isempty(q))

{

q->push(q, NULL);

}

if (c == k)

break;

c++;

}

else

{

if (k == c)

{

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 = root->delete\_form\_avl(root, a[i]);

}

}

void inorder(node \*&root)

{

if (root == NULL)

return;

inorder(root->left);

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

inorder(root->right);

}

};

int main()

{

node \*root = NULL;

cout << "enter the data to insert into avl (-1 to exit): ";

int d;

cin >> d;

while (d != -1)

{

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

cout << "enter the data to insert into avl (-1 to exit): ";

cin >> d;

}

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

root->level\_order(root);

int key;

cout << "enter the level k for deletion (-1 to exit):";

cin >> key;

while (key != -1)

{

root->deletion\_at\_k(root, key);

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

root->level\_order(root);

cout << "enter the level k (-1 to exit):";

cin >> key;

}

}

// 12 23 16 49 34 42 45 -1

Q9

#include <iostream>

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> \*&q, t d)

{

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

if (q == NULL)

{

q = n;

return;

}

else

{

queue<t> \*temp = q;

while (temp->next != NULL)

{

temp = temp->next;

}

temp->next = n;

}

}

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

{

if (q != NULL)

{

queue<t> \*n = q;

q = q->next;

delete n;

}

}

t top(queue<t> \*s)

{

if (s != NULL)

{

return s->data;

}

return NULL;

}

bool isempty(queue<t> \*&s)

{

return s == NULL ? true : false;

}

};

class node

{

void height\_cal(node \*root, int h, int &ans)

{

if (root == NULL)

{

ans = max(ans, h);

return;

}

height\_cal(root->left, h + 1, ans);

height\_cal(root->right, h + 1, ans);

}

int get\_balance\_factor(node \*root)

{

if (root == NULL)

return -1;

return height\_c(root->left) - height\_c(root->right);

}

node \*left\_r(node \*x)

{

node \*y = x->left;

node \*b = y->right;

y->right = x;

x->left = b;

return y;

}

node \*right\_r(node \*x)

{

node \*y = x->right;

node \*b = y->left;

y->left = x;

x->right = b;

return y;

}

public:

int data;

node \*left;

node \*right;

node(int d)

{

this->data = d;

this->left = NULL;

this->right = NULL;

}

node \*insert\_into\_avl(node \*root, int d)

{

if (root == NULL)

{

return new node(d);

}

if (root->data < d)

{

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

}

else if (root->data > d)

{

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

}

else

{

cout << "\nduplicate data .\n ";

return root;

}

int bf = get\_balance\_factor(root);

if (bf > 1 && get\_balance\_factor(root->left) >= 0)

{

return left\_r(root);

}

if (bf > 1 && get\_balance\_factor(root->left) < 0)

{

root->left = right\_r(root->left);

return left\_r(root);

}

if (bf < -1 && get\_balance\_factor(root->right) <= 0)

{

return right\_r(root);

}

if (bf < -1 && get\_balance\_factor(root->right) > 0)

{

root->right = left\_r(root->right);

return right\_r(root);

}

return root;

}

int height\_c(node \*root)

{

int ans = 0;

height\_cal(root, 0, ans);

return ans;

}

void level\_order(node \*root)

{

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

q->push(q, root);

q->push(q, NULL);

while (!q->isempty(q))

{

node \*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 inorder(node \*root)

{

if (root == NULL)

return;

inorder(root->left);

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

inorder(root->right);

}

void searching(node \*root, node \*&ans, int key)

{

if (root == NULL)

return;

if (root->data == key)

{

ans = root;

return;

}

searching(root->left, ans, key);

searching(root->right, ans, key);

}

node \*search(node \*root, int key)

{

node \*ans = NULL;

searching(root, ans, key);

return ans;

}

};

int main()

{

node \*root = NULL;

cout << "enter the data to insert into avl (-1 to exit): ";

int d;

cin >> d;

while (d != -1)

{

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

cout << "enter the data to insert into avl (-1 to exit): ";

cin >> d;

}

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

root->level\_order(root);

int key;

cout << "enter the key for searching :";

cin >> key;

node \*target = root->search(root, key);

if (target == NULL)

{

cout << "the key is not present : ";

}

else

{

cout << "the key is present :";

cout << "\nlevel order (subtree):\n";

target->level\_order(target);

}

}

// 12 23 16 49 34 42 45 -1

Q10

#include <iostream>

#include <vector>

#include <algorithm>

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> \*&q, t d)

{

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

if (q == NULL)

{

q = n;

return;

}

else

{

queue<t> \*temp = q;

while (temp->next != NULL)

{

temp = temp->next;

}

temp->next = n;

}

}

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

{

if (q != NULL)

{

queue<t> \*n = q;

q = q->next;

delete n;

}

}

t top(queue<t> \*s)

{

if (s != NULL)

{

return s->data;

}

return NULL;

}

bool isempty(queue<t> \*&s)

{

return s == NULL ? true : false;

}

};

class node

{

void height\_cal(node \*root, int h, int &ans)

{

if (root == NULL)

{

ans = max(ans, h);

return;

}

height\_cal(root->left, h + 1, ans);

height\_cal(root->right, h + 1, ans);

}

int get\_balance\_factor(node \*root)

{

if (root == NULL)

return -1;

return height\_c(root->left) - height\_c(root->right);

}

node \*left\_r(node \*x)

{

node \*y = x->left;

node \*b = y->right;

y->right = x;

x->left = b;

return y;

}

node \*right\_r(node \*x)

{

node \*y = x->right;

node \*b = y->left;

y->left = x;

x->right = b;

return y;

}

public:

int data;

node \*left;

node \*right;

node(int d)

{

this->data = d;

this->left = NULL;

this->right = NULL;

}

node \*insert\_into\_avl(node \*root, int d)

{

if (root == NULL)

{

return new node(d);

}

if (root->data < d)

{

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

}

else if (root->data > d)

{

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

}

else

{

// cout<<"\nduplicate data .\n ";

return root;

}

int bf = get\_balance\_factor(root);

if (bf > 1 && get\_balance\_factor(root->left) >= 0)

{

return left\_r(root);

}

if (bf > 1 && get\_balance\_factor(root->left) < 0)

{

root->left = right\_r(root->left);

return left\_r(root);

}

if (bf < -1 && get\_balance\_factor(root->right) <= 0)

{

return right\_r(root);

}

if (bf < -1 && get\_balance\_factor(root->right) > 0)

{

root->right = left\_r(root->right);

return right\_r(root);

}

return root;

}

int height\_c(node \*root)

{

int ans = 0;

height\_cal(root, 0, ans);

return ans;

}

void level\_order(node \*root)

{

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

q->push(q, root);

q->push(q, NULL);

while (!q->isempty(q))

{

node \*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 inorder(node \*root)

{

if (root == NULL)

return;

inorder(root->left);

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

inorder(root->right);

}

node \*convert\_to\_avl(node \*root)

{

if (root == NULL)

return NULL;

root->left = convert\_to\_avl(root->left);

root->right = convert\_to\_avl(root->right);

int bf = get\_balance\_factor(root);

if (bf > 1 && get\_balance\_factor(root->left) >= 0)

{

return left\_r(root);

}

if (bf > 1 && get\_balance\_factor(root->left) < 0)

{

root->left = right\_r(root->left);

return left\_r(root);

}

if (bf < -1 && get\_balance\_factor(root->right) <= 0)

{

return right\_r(root);

}

if (bf < -1 && get\_balance\_factor(root->right) > 0)

{

root->right = left\_r(root->right);

return right\_r(root);

}

return root;

}

void collect\_data(node \*root, vector<int> &a)

{

if (root == NULL)

{

return;

}

collect\_data(root->left, a);

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

collect\_data(root->right, a);

}

};

int main()

{

node \*root1 = NULL;

cout << "enter the data to insert into avl1(-1 to exit): ";

int d;

cin >> d;

while (d != -1)

{

root1 = root1->insert\_into\_avl(root1, d);

cout << "enter the data to insert into avl1(-1 to exit): ";

cin >> d;

}

cout << "\nlevel order avl 1 :\n";

root1->level\_order(root1);

node \*root2 = NULL;

cout << "enter the data to insert into avl2(-1 to exit): ";

cin >> d;

while (d != -1)

{

root2 = root2->insert\_into\_avl(root2, d);

cout << "enter the data to insert into avl2(-1 to exit): ";

cin >> d;

}

cout << "\nlevel order avl 2 :\n";

root2->level\_order(root2);

vector<int> a;

root1->collect\_data(root1, a);

root2->collect\_data(root2, a);

node \*root3 = NULL;

sort(a.begin(), a.end());

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

{

root3 = root3->insert\_into\_avl(root3, a[i]);

}

cout << "\nresultant avl : ";

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

root3->level\_order(root3);

}

// 13 10 15 5 11 14 16 4 6 -1

// 33 13 53 11 21 61 8 9 -1

Q11

#include <iostream>

#include <vector>

#include <algorithm>

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> \*&q, t d)

{

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

if (q == NULL)

{

q = n;

return;

}

else

{

queue<t> \*temp = q;

while (temp->next != NULL)

{

temp = temp->next;

}

temp->next = n;

}

}

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

{

if (q != NULL)

{

queue<t> \*n = q;

q = q->next;

delete n;

}

}

t top(queue<t> \*s)

{

if (s != NULL)

{

return s->data;

}

return NULL;

}

bool isempty(queue<t> \*&s)

{

return s == NULL ? true : false;

}

};

class node

{

void height\_cal(node \*root, int h, int &ans)

{

if (root == NULL)

{

ans = max(ans, h);

return;

}

height\_cal(root->left, h + 1, ans);

height\_cal(root->right, h + 1, ans);

}

int get\_balance\_factor(node \*root)

{

if (root == NULL)

return -1;

return height\_c(root->left) - height\_c(root->right);

}

node \*left\_r(node \*x)

{

node \*y = x->left;

node \*b = y->right;

y->right = x;

x->left = b;

return y;

}

node \*right\_r(node \*x)

{

node \*y = x->right;

node \*b = y->left;

y->left = x;

x->right = b;

return y;

}

public:

int data;

node \*left;

node \*right;

node(int d)

{

this->data = d;

this->left = NULL;

this->right = NULL;

}

node \*insert\_into\_avl(node \*root, int d)

{

if (root == NULL)

{

return new node(d);

}

if (root->data < d)

{

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

}

else if (root->data > d)

{

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

}

else

{

// cout<<"\nduplicate data .\n ";

return root;

}

int bf = get\_balance\_factor(root);

if (bf > 1 && get\_balance\_factor(root->left) >= 0)

{

return left\_r(root);

}

if (bf > 1 && get\_balance\_factor(root->left) < 0)

{

root->left = right\_r(root->left);

return left\_r(root);

}

if (bf < -1 && get\_balance\_factor(root->right) <= 0)

{

return right\_r(root);

}

if (bf < -1 && get\_balance\_factor(root->right) > 0)

{

root->right = left\_r(root->right);

return right\_r(root);

}

return root;

}

int height\_c(node \*root)

{

int ans = 0;

height\_cal(root, 0, ans);

return ans;

}

void level\_order(node \*root)

{

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

q->push(q, root);

q->push(q, NULL);

while (!q->isempty(q))

{

node \*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 inorder(node \*root)

{

if (root == NULL)

return;

inorder(root->left);

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

inorder(root->right);

}

node \*convert\_to\_avl(node \*root)

{

if (root == NULL)

return NULL;

root->left = convert\_to\_avl(root->left);

root->right = convert\_to\_avl(root->right);

int bf = get\_balance\_factor(root);

if (bf > 1 && get\_balance\_factor(root->left) >= 0)

{

return left\_r(root);

}

if (bf > 1 && get\_balance\_factor(root->left) < 0)

{

root->left = right\_r(root->left);

return left\_r(root);

}

if (bf < -1 && get\_balance\_factor(root->right) <= 0)

{

return right\_r(root);

}

if (bf < -1 && get\_balance\_factor(root->right) > 0)

{

root->right = left\_r(root->right);

return right\_r(root);

}

return root;

}

void collect\_data(node \*root, vector<int> &a)

{

if (root == NULL)

{

return;

}

collect\_data(root->left, a);

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

collect\_data(root->right, a);

}

void change\_value(node \*&root, int key, int value, bool &c)

{

if (root == NULL)

return;

if (root->data == key)

{

root->data = value;

c = true;

return;

}

change\_value(root->left, key, value, c);

change\_value(root->right, key, value, c);

}

};

int main()

{

node \*root1 = NULL;

cout << "enter the data to insert into avl(-1 to exit): ";

int d;

cin >> d;

while (d != -1)

{

root1 = root1->insert\_into\_avl(root1, d);

cout << "enter the data to insert into avl(-1 to exit): ";

cin >> d;

}

cout << "\nlevel order avl :\n";

root1->level\_order(root1);

int key;

int new\_value;

cout << "enter the key to change : ";

cin >> key;

cout << "enter the new value : ";

cin >> new\_value;

bool c = false;

root1->change\_value(root1, key, new\_value, c);

if (c == false)

{

cout << "the key is not present : ";

return 0;

}

vector<int> a;

root1->collect\_data(root1, a);

node \*root = NULL;

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

root = root->insert\_into\_avl(root, a[i]);

cout << "\nlevel order of updated avl :\n";

root->level\_order(root);

}

// 13 10 15 5 11 14 16 4 6 -1

// 33 13 53 11 21 61 8 9 -1