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->right=NULL;

this->left=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);

}

return root;

}

void insert(node \* & root)

{

int d;

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

cin>>d;

while(d!=-1)

{

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

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

cin>>d;

}

}

void preorder(node \* root)

{

if(root==NULL) return ;

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

preorder(root->left);

preorder(root->right);

}

int min\_value(node \* root)

{

node \* r=root->right;

while(r->left!=NULL)

{

r=r->left;

}

return r->data;

}

node \* deletion(node \* root, int d)

{

if(root==NULL) return NULL;

if(root->data < d)

{

root->right=deletion(root->right,d);

}

else if(root->data > d)

{

root->left=deletion(root->left,d);

}

else

{

if(root->left==NULL)

{

node \* a =root->right;

delete root;

return a;

}

else if(root->right==NULL)

{

node \* a =root->left;

delete root;

return a;

}

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

{

int x=min\_value(root);

//cout<<"\n\*"<<x<<"\n";

root->data=x;

root->right=deletion(root->right,x);

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);

}

}

}

}

};

int main()

{

node \* root =NULL;

root->insert(root);

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

root->level\_order(root);

cout<<"\n";

int d;

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

cin>>d;

while(d!=-1)

{

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

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

root->level\_order(root);

cout<<"\n";

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

cin>>d;

}

}

// 10 2030, 25, 15, 5, 35, 45, 55, 50, 45, 40

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> \*&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;

}

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

{

if(root==NULL){ans=max(h,ans); 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 \* 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);

}

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->left)<=0)

{

return right\_r(root);

}

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

{

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

return right\_r(root);

}

return root;

}

void insert(node \* & root)

{

int d;

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

cin>>d;

while(d!=-1)

{

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

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

cin>>d;

}

}

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);

}

}

}

}

};

int main()

{

node \* root=NULL;

root->insert(root);

// root->level\_order(root);

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

}

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

{

if(root==NULL){ans=max(h,ans); 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 \* 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);

}

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->left)<=0)

{

return right\_r(root);

}

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

{

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

return right\_r(root);

}

return root;

}

// void insert(node \* & root,int d)

// {

// root=insert\_into\_avl(root,d);

// }

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);

}

}

}

}

};

int main()

{

node \* root=NULL;

int n;

cout<<"enter the length of the sorted array : ";

cin>>n;

int arr[n];

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

{

cout<<"enter the "<<i+1<<" th1 element : ";

cin>>arr[i];

}

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

{

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

}

root->level\_order(root);

cout<<"\nthe heigh of the avl tree is : "<<root->height\_c(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

{

public:

int data;

node \* left;

node \* right;

node(int d)

{

this->data=d;

this->right=NULL;

this->left=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);

}

return root;

}

void insert(node \* & root)

{

int d;

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

cin>>d;

while(d!=-1)

{

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

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

cin>>d;

}

}

void preorder(node \* root)

{

if(root==NULL) return ;

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

preorder(root->left);

preorder(root->right);

}

int min\_value(node \* root)

{

node \* r=root->right;

while(r->left!=NULL)

{

r=r->left;

}

return r->data;

}

node \* deletion(node \* root, int d)

{

if(root==NULL) return NULL;

if(root->data < d)

{

root->right=deletion(root->right,d);

}

else if(root->data > d)

{

root->left=deletion(root->left,d);

}

else

{

if(root->left==NULL)

{

node \* a =root->right;

delete root;

return a;

}

else if(root->right==NULL)

{

node \* a =root->left;

delete root;

return a;

}

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

{

int x=min\_value(root);

//cout<<"\n\*"<<x<<"\n";

root->data=x;

root->right=deletion(root->right,x);

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

{

if(root==NULL){ans=max(h,ans); 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);

}

void check(node \* root, bool & c)

{

if(root==NULL) return;

if(get\_balance\_factor(root)<-1 || get\_balance\_factor(root)>1 ){c=false; return;}

if(root->left!=NULL && root->data<root->left->data){c=false; return;};

if(root->right!=NULL && root->data>root->right->data){c=true; return;};

check(root->right,c);

check(root->left,c);

}

bool check\_avl(node \* root)

{

bool c=true;

check(root,c);

return c;

}

};

int main()

{

node \* root =NULL;

root->insert(root);

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

root->level\_order(root);

cout<<"\n";

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

{

cout<<"\nthis bst is avl";

}

else

{

cout<<"\nthis bst is not avl";

}

}

// 10 2030, 25, 15, 5, 35, 45, 55, 50, 45, 40

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

{

public:

int data;

node \* left;

node \* right;

node(int d)

{

this->data=d;

this->left=NULL;

this->right=NULL;

}

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

{

if(root==NULL){ans=max(h,ans); 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 \* 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);

}

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->left)<=0)

{

return right\_r(root);

}

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

{

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

return right\_r(root);

}

return root;

}

int min\_value(node \* root)

{

node \* r=root->right;

while(r->left!=NULL)

{

r=r->left;

}

return r->data;

}

node \* deletion(node \* root,int d)

{

if(root==NULL) return NULL;

if(root->data < d)

{

root->right=deletion(root->right,d);

}

else if(root->data > d)

{

root->left=deletion(root->left,d);

}

else

{

if(root->left==NULL)

{

node \* a =root->right;

delete root;

return a;

}

else if(root->right==NULL)

{

node \* a =root->left;

delete root;

return a;

}

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

{

int x=min\_value(root);

//cout<<"\n\*"<<x<<"\n";

root->data=x;

root->right=deletion(root->right,x);

}

}

if(root==NULL) return NULL;

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->left)<=0)

{

return right\_r(root);

}

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

{

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

return right\_r(root);

}

return root;

}

void insert(node \* & root)

{

int d;

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

cin>>d;

while(d!=-1)

{

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

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

cin>>d;

}

}

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);

}

}

}

}

};

int main()

{

node \* root=NULL;

root->insert(root);

root->level\_order(root);

int d;

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

cin>>d;

while(d!=-1)

{

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

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

root->level\_order(root);

cout<<"\n";

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

cin>>d;

}

}