**POST – LAB**

#include<iostream>

#include <unordered\_map>

#include <queue>

using namespace std;

class BNode

{

public:

BNode()

{data=0;}

void setLeftChild(BNode\* n)

{leftchild=n;}

BNode\* getLeftChild()

{return leftchild;}

void setRightChild(BNode\* n)

{ rightchild=n;}

BNode\* getRightChild()

{return rightchild;}

//void setData(int pdate)

//{data=pdata;}

int getData()

{return data;}

public:

int data,pdate;

BNode\* leftchild;

BNode\* rightchild;

};

class BinaryTree

{

public:

//part1: constructor

BinaryTree \*newNode(int key)

{BinaryTree \*temp=new BinaryTree();

temp->key = key;

temp->leftchild = temp->rightchild = nullptr;

return (temp);}

// Function to build binary tree from given parent array

void BuildTree(int \*Arr, int n)

{// create an empty map

unordered\_map<int, BinaryTree\*> map;

// create n new tree nodes each having value from 0 to n-1

// and store them in a map

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

map[i] = newNode(i);

// represents root node of binary tree

BinaryTree \*root = nullptr;

// traverse the parent array and build the tree

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

{// if parent is -1, set root to current node having

// value i (stored in map[i])

if (Arr[i] == -1)

root = map[i];

else

{// get parent node for current node

BinaryTree \*ptr = map[Arr[i]];

// if parent's left child is filled,

// map the node to its right child

if (ptr->leftchild)

ptr->rightchild = map[i];

// if parent's left child is empty, map the node to it

else

ptr->leftchild = map[i];}}

// return root of the constructed tree

cout<<root;}

int calculateHeightItr (Node \*root)

{// empty tree has height 0

if (root == nullptr)

return 0;

// create an empty queue and enqueue root node

list<Node\*> queue;

queue.push\_back(root);

Node\* front = nullptr;

int height = 0;

// do till queue is empty

while (!queue.empty())

{// calculate number of nodes in current level

int size = queue.size();

// process each node of current level and enqueue their

// non-empty left and right child to queue

while (size--)

{front = queue.front();

queue.pop\_front();

if (front->left)

queue.push\_back(front->left);

if (front->right)

queue.push\_back(front->right);}

// increment height by 1 for each level

height++;}

return height;}

private:

// you may add any other private members which might be needed by recursive functions

BNode\* root;

BinaryTree \*leftchild,\*rightchild;

int key;

int Arr[15];

};

int main()

{BinaryTree<int> \*BT; //creating an object of binary tree

BT=new BinaryTree<int>();

//array to pass,0 means no node exists

int numbers[15]={0,1,2,3,4,5,6,7,8,9,10,0,12,13,14};

BT->BuildTree(numbers,15); //building the tree from the array

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"Height of the Binary Tree is: "<<endl;

BT-> calculateHeightItr(root);

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

return 0;}

**OUTPUT**



**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***