|  |
| --- |
| **Data Structures and Algorithms**  ***Section*: BSCE2021 Assignment # 9 *Total marks*: 100**  ***Name*** : ***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Roll number* : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |

***Submission:***

• *Email instructor or TA if there are any questions. You cannot look at others’ solutions or use others’ solutions, however, you can discuss it with each other. Plagiarism will be dealt with according to the course policy.*

*• Submission after due time will not be accepted.*

**There should be a Report explaining your code and highlighting the results. Follow this naming convention for your report RollNumber\_Assignment#.pdf e.g BSCE21000\_Assignment7.pdf.**

**TASK:**

Implemented the Binary trees with the linked list and implement the following functionality to this tree.

* Insert
* Delete
* Display
* Find the height of a tree
* Find the size of a tree
* The degree of the tree
* Attaching an entirely new tree as a sub-tree,
* is\_root
* is\_leaf

A tree whose elements have at most 2 children is called a binary tree. Since each element in a binary tree can have only 2 children, we typically name them the left and right child.

Program should be menu driven. the program should continue to run until the exit option is selected. Handle all corner cases.

|  |
| --- |
| #include <iostream>  using namespace std;  class node { public:  int data;  node \*left; //declaring  node \*right;   node(int d) {  data = d;  left = nullptr; //initializing  right = nullptr;  }   static node \*newNode(int val) {  cout << "ENTER VALUE TO ENTER OF CHILD\nENTER -1 FOR NO VALUE." << endl; //taking value  cin >> val;  if (val < 0 && val != -1) { //checking that if the value is less than zero and is not equal to -1 then the value is negative  cout << "VALUE ENTERED IS NEGATIVE." << endl;  }  if (val == -1) {  return nullptr; //if value is negative then return null  }  node \*temp = new node(val);  temp->data = val; //making a new pointer and then storing the val in the data  temp->right = nullptr; //pointing right and left to null  temp->left = nullptr;  return temp;  }   node \*insert(node \*root, int val) { //in case of inserting  if (root == nullptr) {  node \*temp1 = newNode(val); //if the roll is null then make a new temp and store the value in it  return temp1; //then return temp  }  if (root->left == nullptr) { //if the left side is null   root->left = insert(root->left, data); //call the insert function on left side of tree  }  if (root->right == nullptr) {  root->left = insert(root->left, data); //call the insert function on left side of tree  }  return root;  }   void deleteTree(node \*root) {  if (root == nullptr) //if root is null  return;   deleteTree(root->left); //calling the left side of the tree  deleteTree(root->right); //calling the left side of the tree   cout << "\n Deleting node: " <<root->data; //displaying  delete root; //deleting  }   int height(node \*root) {  if (root == nullptr) //checking that if the root is null  return 0;  else {  int left\_side;  int right\_side; //declaring  left\_side = height(root->left); //calling left and right side of trees and storing then in the variables  right\_side = height(root->right);  if (left\_side >= right\_side) { //if left is equal or greater than display the left by adding 1 to it  return left\_side + 1;  } else {  return right\_side + 1; //if right is greater than display the right by adding 1 to it  }  }  }   int size(node \*root) {  if (root == nullptr) //checking if the root is null  return 0;  else {  int left\_side; //declaring  int right\_side;  left\_side = size(root->left);  right\_side = size(root->right); //calling left and right sides and displaying  return (left\_side + 1 + right\_side); //returning by adding left and right as well as 1 to it  }   }   int helper(node \*root, int val, int level) {  if (root == nullptr)  return 0; //checking if the root is null   if (root->data == val) //if data is equal to val  return level;   int downlevel = helper(root->left, val, level + 1); //calling it by level+1 bcz the level increases and storing it in a variable  if (downlevel != 0)  return downlevel; //if the variable is not zero then display that variable int   downlevel = helper(root->right, val, level + 1); //then increase the level and call the function and store the value in a variable  return downlevel; //then returning it  }   int degree(node \*root, int val) {  return helper(root, val, 1); //returning the value coming from the helper function  }   bool isLeaf(node \*root, int val) {  if (root != nullptr) { //checking if the root is null or not  if (root->left == nullptr && root->right == nullptr) { //checking that both the left and right are null or not  if (root->data == val) { //if null then checking that the value match with the data stored in root if yes then return tre else false  return true;  } else {  return false;  }  } else if (root->left) { //if the above answer is false then call for the left side and is a recursive call  isLeaf(root->left, val);  } else if (root->right) { //if the above answer is false then call for the left side and is a recursive call  isLeaf(root->right, val);  }  }  else{  return false;  }  }   bool isRoot(node \*root, int val) {  if (root == nullptr) { //checking that the root is null or not  return false;  } else {  if (root->data == val) { //if not then checking if the value match with teh data stored in root then return true else false  return true;  } else {  return false;  }  }  }   void display(node \*root) { //a function to display  if (root == nullptr) {  return;  }  cout << root->data << " "; //if root is not null then display  display(root->left); //calling the left side  display(root->right); //calling for the right side of the tree  } };  **MAIN.CPP:**  #include <iostream> #include "Functions.h"  using namespace std;  // int main() {   node n(0); //making ab object  node \*root; //making a pointer  int val; //declaring  int opt;  do {  cout << "\nCHOOSE OPTIONS." << endl;  cout << "1.INSERT." << endl;  cout << "2.ATTACH A SUBTREE." << endl;  cout << "3.HEIGHT OF THE TREE." << endl;  cout << "4.SIZE OF THE TREE." << endl; //taking option  cout << "5.DEGREE OF THE TREE." << endl;  cout << "6.DISPLAY." << endl;  cout << "7.IS ROOT." << endl;  cout << "8.IS LEAF." << endl;  cout << "9.DELETE." << endl;  cout << "10.EXIT." << endl;  cin >> opt;  if (opt == 1) {  root = n.insert(root, val);  cout << "THE TREE IS = "; //callling  n.display(root);  cout << endl;  }  if (opt == 2) {  root = n.insert(root, val);  n.display(root);  cout<<endl; //calling  }  if (opt == 3) {  cout << "THE HEIGHT OF TREE = ";  int height1 = n.height(root); //calling  cout << height1 << endl;  }  if (opt == 4) {  cout << "THE SIZE OF TREE = ";  int size = n.size(root); //calling  cout << size << endl;  }  if (opt == 5) {  cout << "ENTER THE NODE FOR WHICH YOU WANT FIND THE DEGREE = ";  cin >> val;  int degreeOfNode = n.degree(root, val); //calling  cout << "DEGREE OF TREE = ";  cout << degreeOfNode << endl;  }  if (opt == 6) {  cout<<"THE TREE IS = ";  n.display(root);  cout<<endl;  }  if (opt == 7) {  cout << "ENTER THE NODE FOR WHICH YOU WANNA CHECK IS ROOT NODE, OR NOT = ";  cin >> val;  if (n.isRoot(root, val)) { //calling  cout << "IS ROOT NODE." << endl;  } else {  cout << "IS NOT ROOT NODE." << endl;  }  }  if (opt == 8) {  cout << "ENTER THE NODE FOR WHICH YOU WANNA CHECK IS LEAF NODE, OR NOT = ";  cin >> val; //calling  if (n.isLeaf(root, val)) {  cout << "IS LEAF NODE." << endl;  } else {  cout << "IS NOT LEAF NODE." << endl;  }  }  if (opt == 9) {  n.deleteTree(root); //calling  root = nullptr;  n.display(root);  cout<<endl;  }  if (opt == 10) {  cout << "YOU CHOOSE TO EXIT." << endl;  exit(3);  }  if(opt!=1 && opt!=2 && opt!=3 && opt!=4 && opt!=5 && opt!=6 && opt!=7 && opt!=8 && opt!=9 && opt!=10){  cout<<"YOU HAVE ENTERED AN INVALID ARGUMENT."<<endl;  break;  }  } while (opt >= 1 && opt <= 10);  return 0; } |