DHA Suffa University
Department of Computer Science
CS 2001L – Data Structures and Algorithms Lab
Spring 2021

Lab 09 – Trees

Objective:

To learn about

- General Tree
- Binary Tree

General Tree:

A general tree is a tree in which each node can have an unlimited out degree. Each node may have as many children as is necessary to satisfy its requirements.

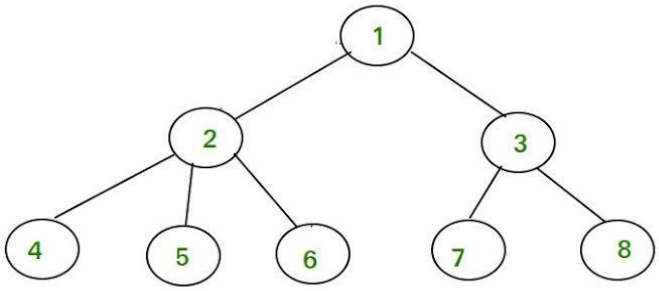


Figure 9.1 – General Tree

Implementation of General Tree:

Node.h

```
#include<iostream>
#include <vector>
struct Node
{
  int data;
  std::vector<Node *> child;
  Node(int data);
  ~Node();
};
```

Node.cpp

```
#include"Node.h"
Node::Node(int data):data(data){}
Node::~Node(){}
```

GeneralTree.h

```
#include"Node.h"
#include<queue>
class GeneralTree{

    public:
        GeneralTree();
        ~GeneralTree();
        Node* insertNode(int data);
        void levelOrderTraversal(Node *root);
};
```

GeneralTree.cpp

```
#include "GeneralTree.h"
GeneralTree:: GeneralTree(){}
GeneralTree:: ~GeneralTree(){}
Node* GeneralTree:: insertNode(int data)
{
    Node *temp = new Node(data);
    return temp;

}

   void GeneralTree:: levelOrderTraversal(Node *root){
       if (root==NULL)
           return;

       std::queue<Node *> q;
       q.push(root);
       while (!q.empty())
       {
           int n = q.size();
           while (n > 0)
           {
               Node * p = q.front();
               q.pop();
               std::cout << p->data << " ";
```

```
                for (int i=0; i<p->child.size(); i++)
                {
                     q.push(p->child[i]);
                }
                n--;
            }

            std::cout << std::endl;
    }
}
```

Driver.cpp

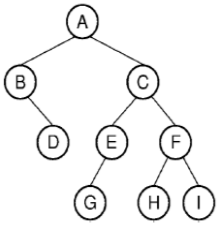
```
#include"GeneralTree.h"
int main()
{
    GeneralTree b;
    Node *root = b.insertNode(7);
    (root->child).push_back(b.insertNode(21));
    (root->child).push_back(b.insertNode(23));
    (root->child).push_back(b.insertNode(25));
    (root->child).push_back(b.insertNode(27));
    (root->child[0]->child).push_back(b.insertNode(33));
    (root->child[0]->child).push_back(b.insertNode(44));
    (root->child[0]->child).push_back(b.insertNode(55));
    (root->child[2]->child).push_back(b.insertNode(66));
    (root->child[2]->child[0]->child).push_back(b.insertNode(3));
    (root->child[3]->child).push_back(b.insertNode(7));
    (root->child[3]->child).push_back(b.insertNode(8));
    (root->child[3]->child).push_back(b.insertNode(9));

    b.levelOrderTraversal(root);
    std::cout<<std::endl;
}
```

Binary Tree

A binary tree **T** is defined as a finite set of elements, called nodes, such that:

a) T is empty (called the null tree or empty tree), or

b) T contains a distinguished node R, called the root of T, and the remaining nodes of T form an ordered pair of disjoint binary trees T_1 and T_2.



Implementation of binary tree:

Node.h

```
#include<iostream>
struct Node
{
  int data;
  Node *left;
  Node *right;
  Node(int data);
  ~Node();
};
```

Node.cpp

```
#include"Node.h"
Node::Node(int data):data(data), left(nullptr), right(nullptr){}
Node::~Node(){}
```

BinaryTree.h

```
#include"Node.h"
class BinaryTree{

    public:
        BinaryTree();
        ~BinaryTree();
        Node* insertNode(int data);
        void inorderTraversal(Node *root);
        void preorderTraversal(Node *root);
        void postorderTraversal(Node *root);

};
```

BinaryTree.cpp

```
#include"BinaryTree.h"
BinaryTree:: BinaryTree(){}
BinaryTree:: ~BinaryTree(){}
Node* BinaryTree:: insertNode(int data)
{
    Node *temp = new Node(data);
    return temp;

}
void BinaryTree:: inorderTraversal(Node *root)
{
    if(root == NULL)
        return;
    inorderTraversal(root->left);
    std::cout<<root->data<<" ";
    inorderTraversal(root->right);

}


 void BinaryTree:: preorderTraversal(Node *root)
 {
     if(root == NULL)
         return;
     std::cout<<root->data<<" ";
     preorderTraversal(root->left);
     preorderTraversal(root->right);

 }
```

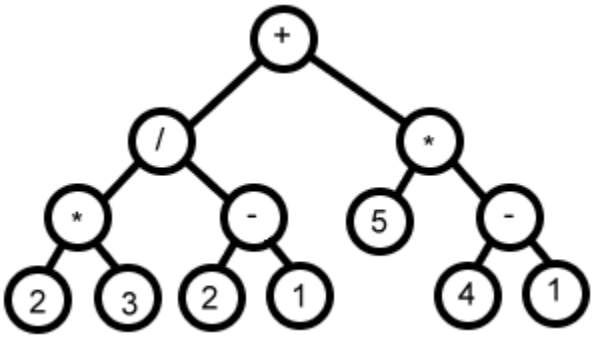
```
 void BinaryTree:: postorderTraversal(Node *root)
 {
     if(root == NULL)
         return;
     postorderTraversal(root->left);
     postorderTraversal(root->right);
     std::cout<<root->data<<" ";

 }
```

Driver.cpp

```
#include"BinaryTree.h"
int main()
{
    BinaryTree b;
    Node *root = b.insertNode(7);
    root->left  = b.insertNode(9);
    root->right =b.insertNode(10);
    root->left->left  = b.insertNode(4);
/*          7
          /       \
        9           10
      /   \        /  \
    4    NULL  NULL  NULL
   /  \
NULL NULL
*/
    b.inorderTraversal(root);
    std::cout<<std::endl;
    b.preorderTraversal(root);
    std::cout<<std::endl;
    b.postorderTraversal(root);

}
```

LAB ASSIGNMENT

1. Evaluate the following expression tree consisting of basic binary operators i.e., + , – ,* and /. Your program should print the resultant value after evaluation as output.



SUBMISSION GUIDELINES

- Take a screenshot of each task (code and its output), labeled properly.
- Place all the screenshots and the code files (properly labeled) in a single folder labeled with Roll No and Lab No. e.g. **'cs191xxx_Lab01'**.
- Submit the folder at LMS
- **-100%** policies for plagiarism.