## Lab 09

**Implementation of Binary Trees**

**ADT of Binary Tree**

##### ADT BinaryTree

{

##### Objects:

A finite set of nodes either empty or consisting of a root node, left BinaryTree and right BinaryTree.

##### Operations:

**BinaryTree** ();

// Creates an empty BinaryTree

**BinaryTree** (ElementType info);

// Creates a single node BinaryTree with information info

**BinaryTree** (BinaryTree lbt, ElementType info, BinaryTree rbt);

// Creates a Binarytree whose left subtree is lbt, whose right subtree is rbt,

// and whose root node contain info.

Boolean **IsEmpty**();

// If number of elements in the BinaryTree is 0 return TRUE,

// otherwise return FALSE.

BinaryTree **LChild**();

// If IsEmpty(), return error, else return left subtree of \*this

ElementType **Data**();

// If IsEmpty(), return error, else return data in the root node of \*this

BinaryTree **RChild**();

// If IsEmpty(), return error, else return right subtree of \*this

};

## Implicit Static Array Representaion

The n nodes of an almost complete binary tree can be numbered from 1 to n, so that the number assigned to a left son is twice the number assigned to its father, and number assigned to a right son is one more than twice the number assigned to its father.

In C++, arrays start from 0; therefore we'll number the nodes from 0 to n-1. The left son of node at position p is at position 2p+1 and right son is at 2p+2. The root of the tree is at position 0.

#### Node Representation

# define NUMNODES 500

struct TreeNode

{

int info;

int left, right, father;

};

TreeNode BT[NUMNODES];

## Dynamic Representation

struct Tree Node

{

int info;

TeeNode \*left, \*right, \*father;

};

typedef TreeNode \*NODEPTR;

If a tree is always traversed from top to bottom, father field is unnecessary.

The maketree() function, which allocates a node and sets it as the root of a single-node binary tree may be written as:

##### NODEPTR maketree (int x)

{

*NODEPTR p =new TreeNode; p  info = x;*

*p  left = NULL; p  right = NULL; return (p);*

}

The function setleft (p,x) sets a node with contents x as the left son of node(p).

##### void setleft (NODEPTR p, int x)

{

*if (p == NULL)*

*cout << “Void Insertion”; else if (p  left != NULL) cout << “Invalid Insertion”; else*

*p  left = maketree (x);*

}

**Binary Tree Traversals in C++**

Three C++ routines pretrav, intrav, and posttrav are given below. The parameter to each routine is the pointer to the root node of a binary tree.

##### void pretrav (NODEPTR tree)

{

*if (tree != NULL)*

*{*

*cout << tree  info; /\* visit the root \*/ pretrav (tree  left);*

*pretrav (tree  right);*

*}*

}

##### void intrav (NODEPTR tree)

{

*if (tree != NULL)*

*{*

*intrav (tree  left);*

*cout << treeinfo;*

*intrav (tree  right);*

*}*

}

##### void posttrav (NODEPTR tree)

{

if (tree != NULL)

{

posttrav (tree left);

posttrav (tree  right);

cout << tree  info;

*}*

}

**Task:**

1. Perform the pre-order in-order and post-order traversal on Binary Tree