A. Tujuan

Setelah melakukan praktikum dalam bab ini, mahasiswa diharapkan mampu:

1. Mahasiswa mampu menjelaskan konsep dari sturktur data Tree (Binary Tree)
2. Mahasiswa mampu membuat program dengan menerapkan struktur data Tree (Binary Tree)
3. Mahasiswa mampu membuat fungsi tambah simpul dan hapus simpul pada strukutur data Tree (Binary Tree)
4. Mahasiswa mampu menjelaskan konsep dari penelusuran pohon biner dengan metode pre-order, in-order, dan post-order
5. Mahasiswa mampu membuat fungsi/methode penelusuran pre-order, in-order, dan post-order pada pohon biner

B. Dasar Teori

1. Tree (Binary Tree)

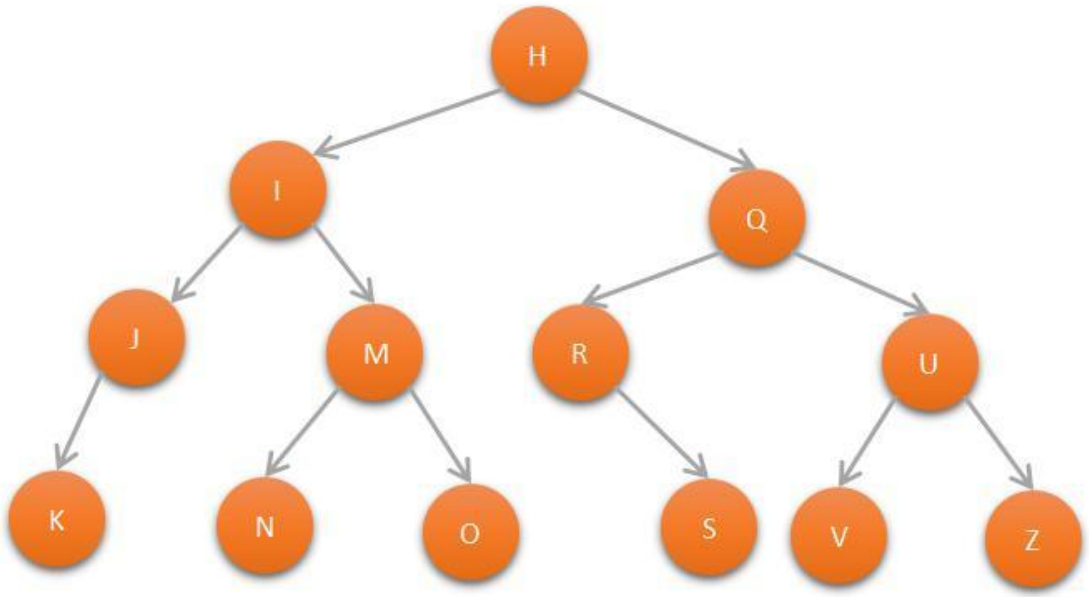
"A tree is a finite non-empty set of elements in which one element is called the root and the other elements are partitioned into m >= 0 dis joint subsets, each of which itself a tree."

Struktur data tree merupakan kumpulan node yang saling terhubung satu sama lain dalam suatu kesatuan yang membentuk layaknya struktur sebuah pohon. Dalam penerapannya berbentuk menyerupai struktur pohon terbalik, akar(root) berada diatas dan daun(leaf) berada dibawah akar. Struktur data tree digunakan untuk menyimpan data-data hirarki seperti pohon keluarga, skema pertandingan, struktur organisasi. Terdapat beberapa terminologi/istilah dalam struktur data tree ini sebagaimana telah disajikan dan dijelaskan pada tabel 01.

Predecesor	Node yang berada diatas node tertentu.
Successor	Node yang berada dibawah node tertentu.
Ancestor	Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama
Descendant	Seluruh node yang terletak setelah node tertentu dan terletak pada jalur yang sama
Parent	Predecessor satu level di atas suatu node.
Child	Successor satu level di bawah suatu node.
Sibling	Node-node yang memiliki parent yang sama
Subtree	Suatu node beserta descendantnya.
Size	Banyaknya node dalam suatu tree
Height	Banyaknya tingkatan dalam suatu tree
Root	Node khusus yang tidak memiliki predecessor.
Leaf	Node-node dalam tree yang tidak memiliki successor.
Degree	Banyaknya child dalam suatu node

Tabel 01. Terminologi dalam Struktur Data Tree

Binary tree atau pohon biner merupakan struktur data pohon akan tetapi setiap simpul dalam pohon dipersyaratkan memiliki simpul satu level dibawahnya (child) tidak lebih dari 2 simpul, artinya jumlah child yang diperbolehkan yakni 0, 1, dan 2.

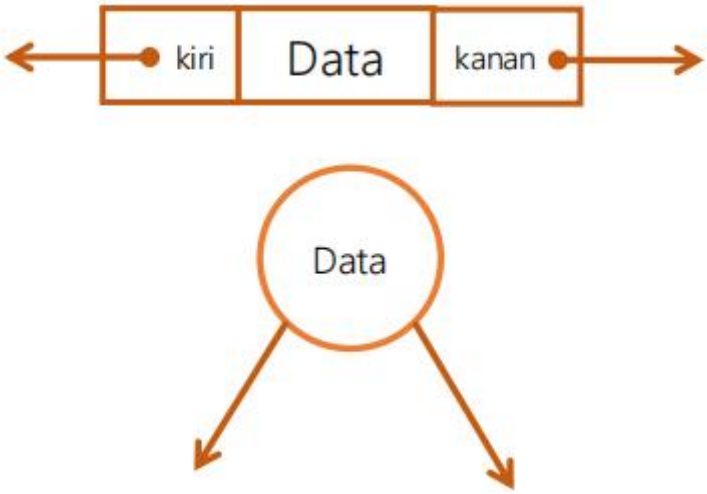


Gambar 01, menunjukkan contoh dari struktur data binary tree

2. Binary Tree dalam Program

Membuat struktur data binary tree dalam suatu program (berbahasa C++) dapat menggunakan struct yang memiliki 2 buah pointer, seperti halnyadouble linked list. Ilustrasi struct 2 buah pointer dapat dilihat pada gambar 02. Untuk membuatnya (dalam bahasa C++) adalah sebagai berikut,

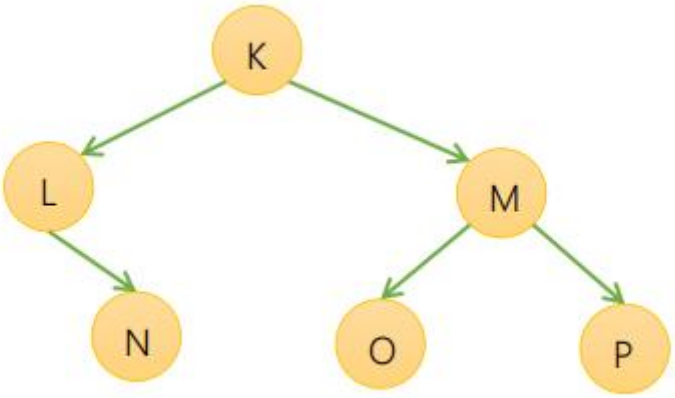
```
struct pohon{
    pohon* kanan;
    char data;
    pohon* kiri;
};
pohon* simpul;
```



Gambar 02. Ilustrasi Simpul

3. Penelusuran Pohon

Apa yang dimaksud dengan penelusuran pohon (tree traversal) ?. Penelusuran pohon mengacu pada proses mengunjungi semua simpul pada pohon tepat satu kali. Kata mengunjungi disini lebih mengacu kepada makna menampilkan data dari simpul yang dikunjungi. Terdapat 3 metode traversal yang dibahas dalam modul ini yakni Pre-Order, In-Order, dan Post-Order.



Gambar 01. Binary Tree

3.1 Pre-Order

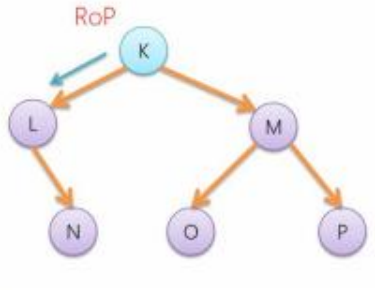
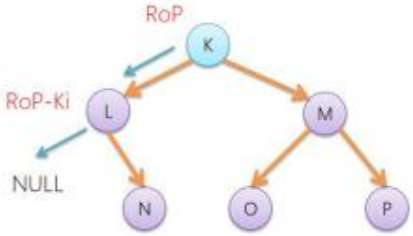
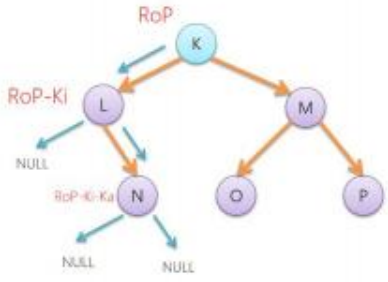
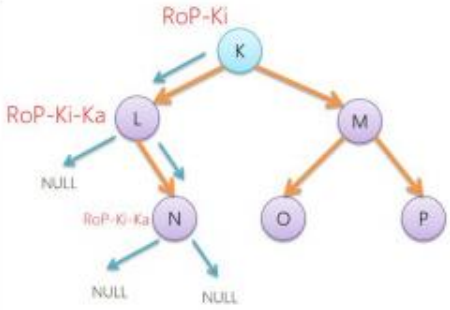
Penelusuran secara pre-order memiliki alur :

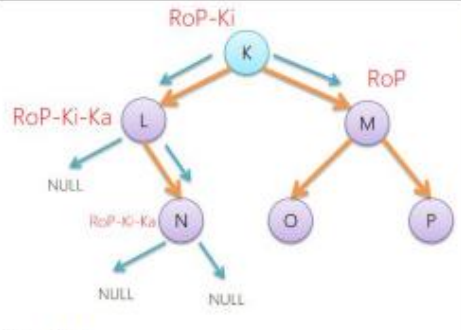
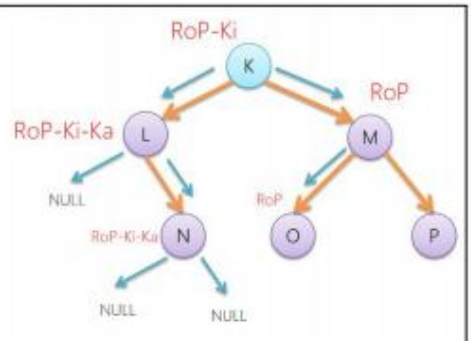
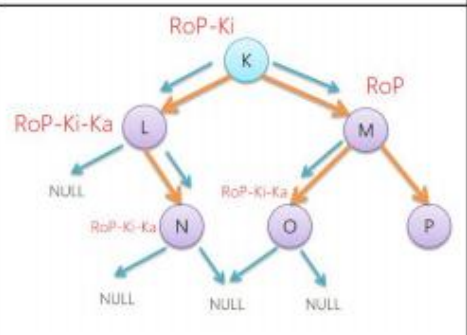
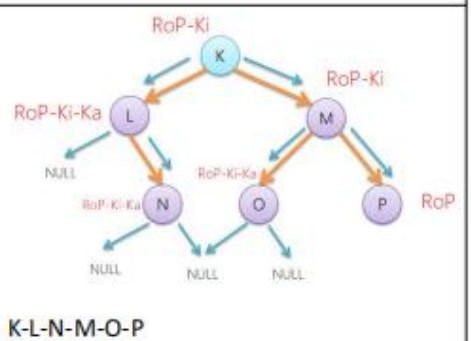
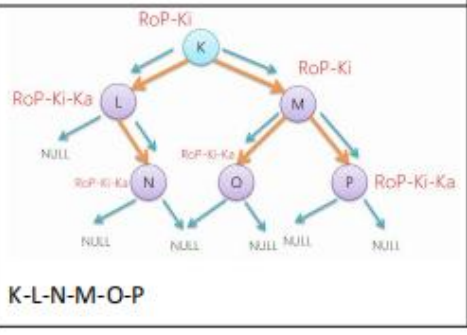
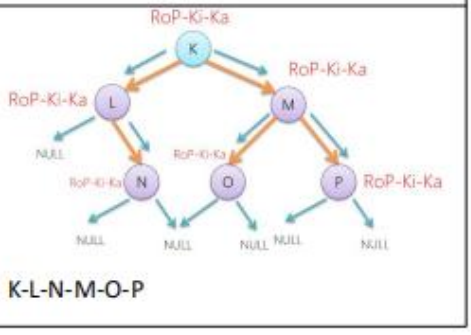
1. Cetak data pada simpul root
2. Secara rekursif mencetak seluruh data pada subpohon kiri
3. Secara rekursif mencetak seluruh data pada subpohon kanan

Dapat kita turunkan rumus penelusuran menjadi :

Root (print) - Kiri - Kanan
RoP - Ki - Ka

Proses penelusuran secara pre-order pada pohon biner gambar 01 ditunjukkan pada tabel dibawah,

No	Proses	No	Proses
1	 K	2	 K - L
3	 K-L-N	4	 K-L-N

5	 K-L-N-M	6	 K-L-N-M-O
7	 K-L-N-M-O	8	 K-L-N-M-O-P
9	 K-L-N-M-O-P	10	 K-L-N-M-O-P

3.2 In-Order

Penelusuran secara pre-order memiliki alur :

1. Secara rekursif mencetak seluruh data pada subpohon kiri
2. Cetak data pada root
3. Secara rekursif mencetak seluruh data pada subpohon kanan

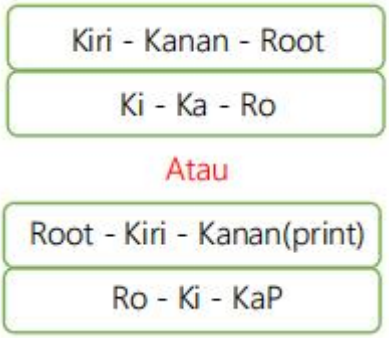
Dapat kita turunkan rumus penelusuran menjadi :

3.3 Post-Order

Penelusuran secara pre-order memiliki alur :

1. Secara rekursif mencetak seluruh data pada subpohon kiri
2. Secara rekursif mencetak seluruh data pada subpohon kanan
3. Cetak data pada root

Dapat kita turunkan rumus penelusuran menjadi :



C. GUIDED

Silahkan dijalankan, amati, dan lakukan analisis.

```
#include <iostream>

using namespace std;

struct TNode {
    int data;
    TNode *left;
    TNode *right;

    // constructor
    TNode(int value) {
```

```
        data = value;
        left = NULL;
        right = NULL;
    }
};

void preOrder(TNode *node) {
    if (node != NULL) {
        cout << node->data << " ";
        preOrder(node->left);
        preOrder(node->right);
    }
}

void inOrder(TNode *node) {
    if (node != NULL) {
        inOrder(node->left);
        cout << node->data << " ";
        inOrder(node->right);
    }
}

void postOrder(TNode *node) {
    if (node != NULL) {
        postOrder(node->left);
        postOrder(node->right);
        cout << node->data << " ";
    }
}

int main() {
    TNode *zero = new TNode(0);
    //                  0
    //                 /\
```

```
//                  NULL NULL
TNode *one = new TNode(1);
TNode *five = new TNode(5);
TNode *seven = new TNode(7);
TNode *eight = new TNode(8);
TNode *nine = new TNode(9);

seven->left = one;
//                      7
//                     /\
//                    1 NULL

seven->right = nine;
//                      7
//                     /\
//                    1    9

one->left = zero;
//                      7
//                     /\
//                    1    9
//                   / \
//                  0    NULL

one->right = five;
//                      7
//                     /\
//                    1    9
//                   / \
//                  0     5

nine->left = eight;
//                      7
//                     /\
```

```
    //                  1       9
    //               / \     / \
    //              0     5 8    NULL

    preOrder(seven);
    cout << endl;
    // 7 1 0 5 9 8

    inOrder(seven);
    cout << endl;
    // 0 1 5 7 8 9

    postOrder(seven);
    cout << endl;
}

// void inOrder(TNode *node) {
//         if (node != NULL) {
//                 inOrder(node->left);
//                 cout << node->data << " ";
//                 inOrder(node->right);
//         }
// }
```

D. UNGUIDED

1. Membuat ilustrasi rekursif pada saat pemanggilan inOrder(7) dan postOrder(7).