

MODUL 11

TREE

DASAR TEORI

A. Tree (Binary Tree)

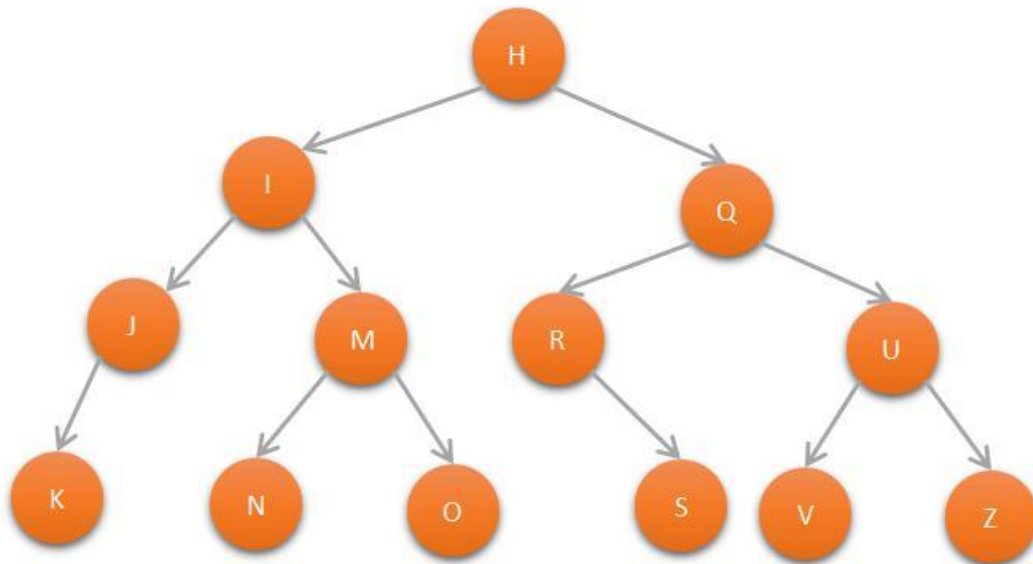
Struktur data tree merupakan kumpulan node yang saling terhubung satu sama lain dalam suatu kesatuan yang membentuk layaknya struktur sebuah pohon. Dalam penerapannya berbentuk menyerupai struktur pohon terbalik, akar (*root*) berada di atas dan daun (*leaf*) berada di bawah akar. Struktur data tree digunakan untuk menyimpan data-data hierarki seperti pohon keluarga, skema pertandingan, struktur organisasi. Terdapat beberapa terminologi/istilah dalam struktur data tree ini sebagaimana telah disajikan dan dijelaskan pada tabel 1.

Predecessor	Node yang berada di atas node tertentu
Successor	Node yang berada di bawah node tertentu
Ancestor	Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama
Descendent	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 descendent-nya
Size	Banyaknya node dalam suatu tree
Height	Banyaknya tingkatan/level dalam suatu tree
Roof	Node khusus yang tidak memiliki predecessor
Leaf	Node-node dalam tree yang tidak memiliki successor
Degree	Banyaknya child dalam suatu node

Tabel 1 Terminologi dalam Struktur Data Tree

Binary tree atau pohon biner merupakan struktur data pohon akan tetapi setiap simpul dalam pohon diprasyarkan memiliki simpul satu level di bawahnya (child) tidak

lebih dari 2 simpul, artinya jumlah child yang diperbolehkan yakni 0, 1, dan 2. Gambar 1, menunjukkan contoh dari struktur data binary tree.



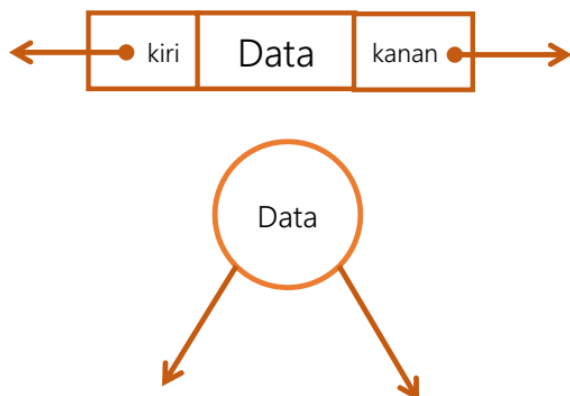
Gambar 1 Struktur Data Binary Tree

B. Binary Tree dalam Program

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

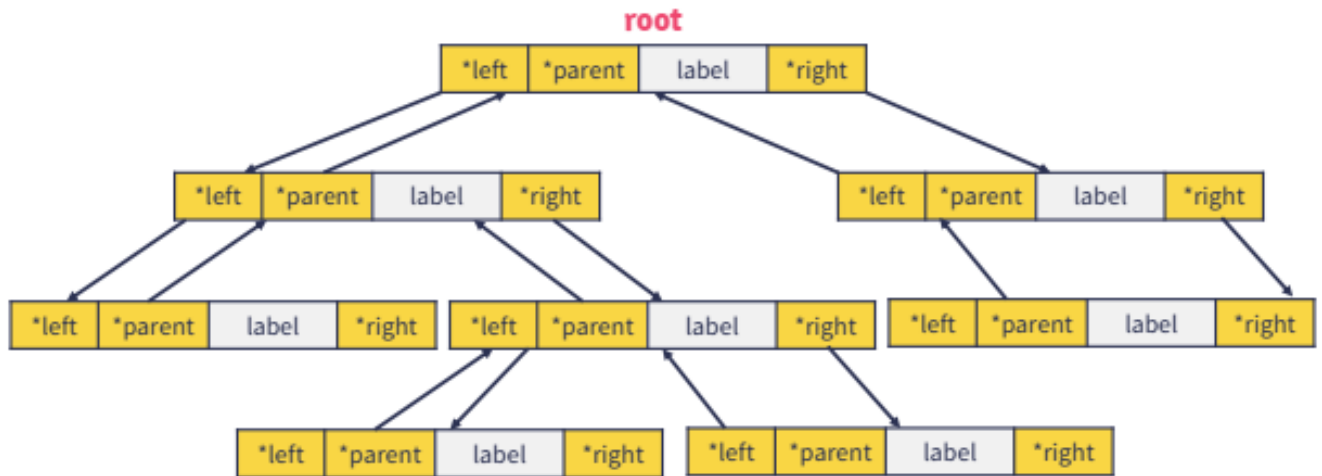
```

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



Gambar 2 Ilustrasi Simpul 2 Pointer

Selain menggunakan dua buah pointer, implementasi tree dapat menambahkan pointer tambahan agar program lebih dinamis dan fleksibel yaitu pointer parent. Berikut ilustrasi implementasi binart tree menggunakan 3 buah pointer.



Gambar 3 Ilustrasi Implementasi Binary Tree Menggunakan 3 Pointer

C. Operasi pada Tree

Create: digunakan untuk membentuk binary tree baru yang masih kosong.

Clear: digunakan untuk mengosongkan binary tree yang sudah ada atau menghapus semua node pada binary tree.

isEmpty: digunakan untuk memeriksa apakah binary tree masih kosong atau tidak.

Insert: digunakan untuk memasukkan sebuah node kedalam tree.

Find: digunakan untuk mencari root, parent, left child, atau right child dari suatu node dengan syarat tree tidak boleh kosong.

Update: digunakan untuk mengubah isi dari node yang ditunjuk oleh pointer current dengan syarat tree tidak boleh kosong.

Retrive: digunakan untuk mengetahui isi dari node yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.

Delete Sub: digunakan untuk menghapus sebuah subtree (node beserta seluruh descendant-nya) yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.

Characteristic: digunakan untuk mengetahui karakteristik dari suatu tree. Yakni size, height, serta average length-nya.

Traverse: digunakan untuk mengunjungi seluruh node-node pada tree dengan cara traversal.

D. Penelusuran Pohon (Traversal)

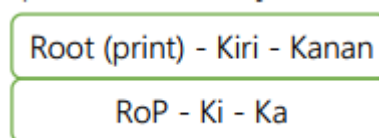
Penelusuran pohon mengacu pada proses mengunjungi semua simpul pada pohon tepat satu kali. Kata mengunjungi disini lebih mengacu kepada makna menampilkan data dari simpul yang dikunjungi. Terdapat 3 metode traversal yang dibahas dalam modul ini yakni Pre-Order, In-Order, dan Post-Order.

Pre-Order

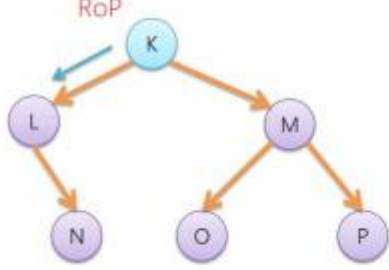
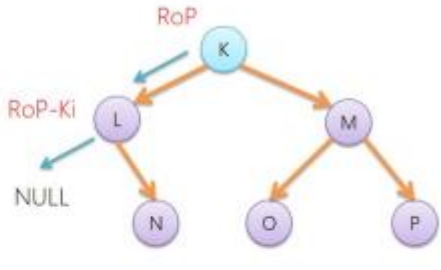
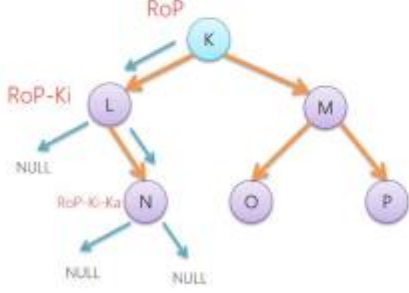
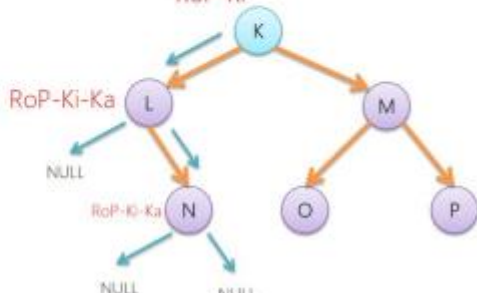
Penelusuran secara pre-order memiliki alur:

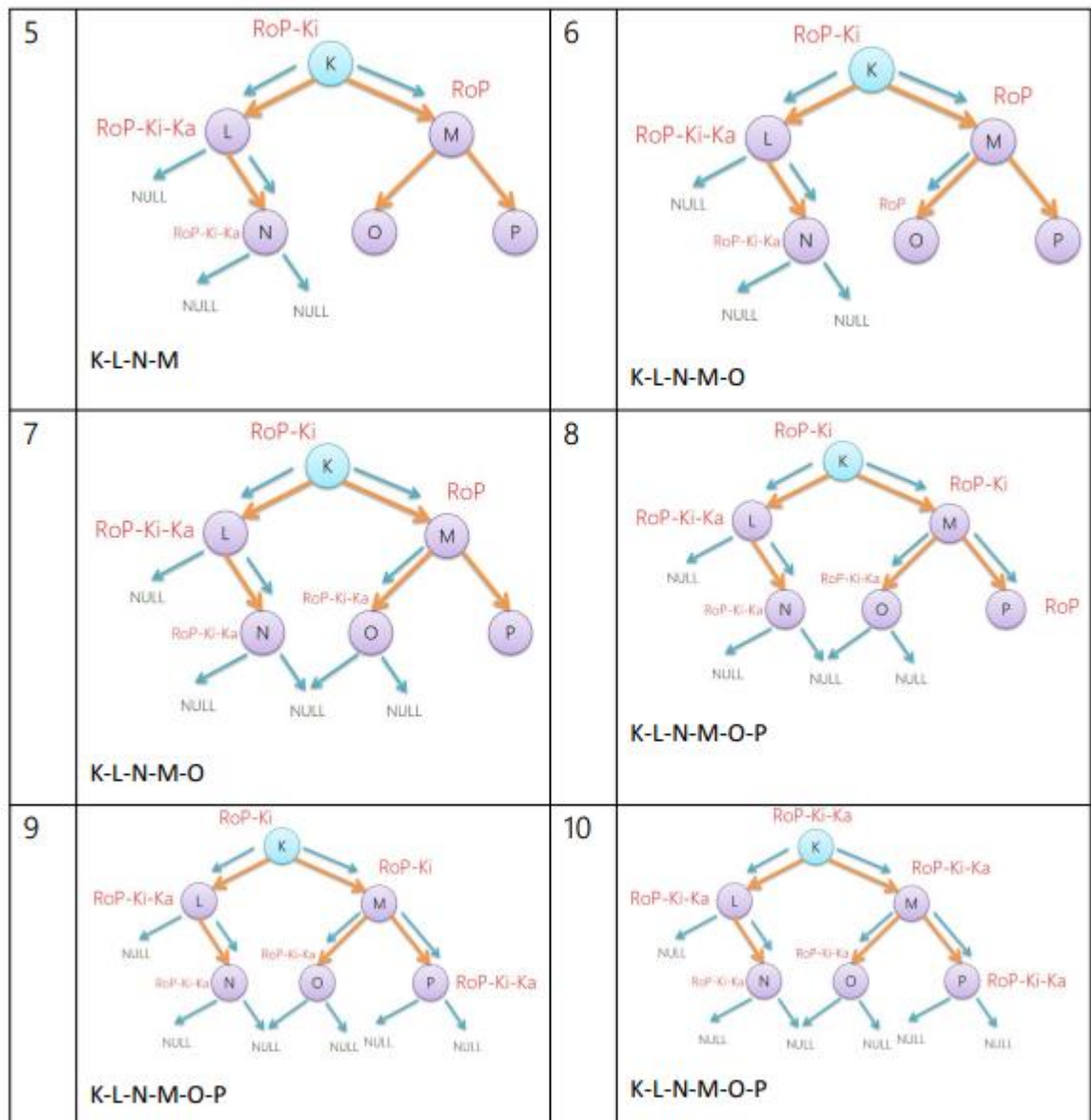
1. Cetak data pada simpul root
2. Secara rekursif mencetak seluruh data pada subpohon kiri
3. Secara rekursif mencetak seluruh data pada subpohon kanan

Dapat kita turunkan rumus penelusuran menjadi:



Proses penelusuran secara pre-order pada pohon biner gambar 01 ditunjukkan pada tabel di bawah.

No	Proses	No	Proses
1	 <p>K</p>	2	 <p>K - L</p>
3	 <p>K-L-N</p>	4	 <p>K-L-N</p>



In-Order

Penelusuran secara in-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:

Kiri - Root - Kanan

Ki - Ro - Ka

Atau

Root - Kiri(print) - Kanan

Ro - KiP - Ka

Post-Order

Penelusuran secara in-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:

Kiri - Kanan - Root

Ki - Ka - Ro

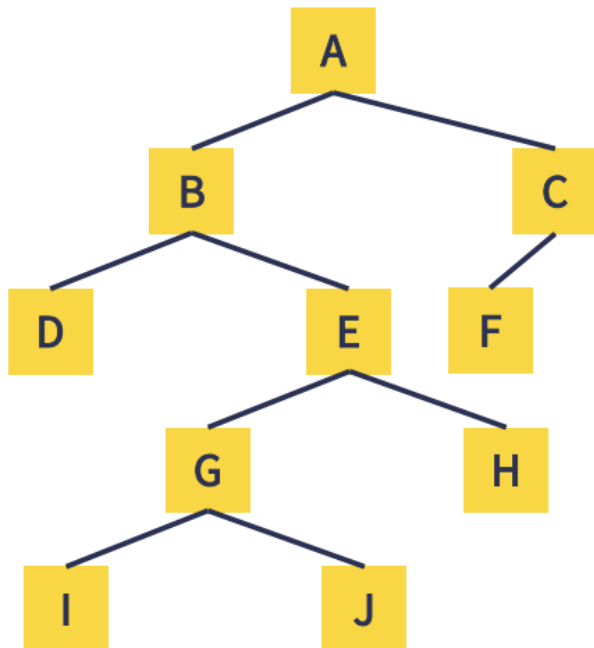
Atau

Root - Kiri - Kanan(print)

Ro - Ki - KaP

GUIDED

Program Binary Tree



```
#include <iostream>

using namespace std;

///PROGRAM BINARY TREE

//Deklarasi Pohon
struct Pohon{
    char data;
    Pohon *left, *right, *parent;
};

Pohon *root, *baru;

//Inisialisasi
void init()
{
    root = NULL;
}

//Cek Node
int isEmpty()
{
    if (root == NULL)
        return 1; //true
    else
```



```

        return 0;    //false
    }

    //Buat Node Baru
    void buatNode(char data )
    {
        if(isEmpty() == 1){
            root = new Pohon();
            root->data = data;
            root->left = NULL;
            root->right = NULL;
            root->parent = NULL;
            cout << "\n Node " << data << " berhasil dibuat menjadi root." <<
endl;
        }
        else{
            cout << "\n Pohon sudah dibuat" << endl;
        }
    }

    //Tambah Kiri
    Pohon *insertLeft(char data, Pohon *node )
    {
        if(isEmpty() == 1){
            cout << "\n Buat root terlebih dahulu!" << endl;
            return NULL;
        }else{
            // cek apakah child kiri ada atau tidak
            if( node->left != NULL ){
                // kalau ada
                cout << "\n Node "<< node->data << " sudah ada child kiri!" <<
endl;
                return NULL;
            }else{
                // kalau tidak ada
                baru = new Pohon();
                baru->data = data;
                baru->left = NULL;
                baru->right = NULL;
                baru->parent = node;
                node->left = baru;
                cout << "\n Node " << data << " berhasil ditambahkan ke child
kiri " << baru->parent->data << endl;
                return baru;
            }
        }
    }
}

```

//Tambah Kanan

```

Pohon *insertRight(char data, Pohon *node )
{
    if( root == NULL ){
        cout << "\n Buat root terlebih dahulu!" << endl;
        return NULL;
    }else{
        // cek apakah child kanan ada atau tidak
        if( node->right != NULL ){
            // kalau ada
            cout << "\n Node " << node->data << " sudah ada child kanan!" << endl;
            return NULL;
        }else{
            // kalau tidak ada
            baru = new Pohon();
            baru->data = data;
            baru->left = NULL;
            baru->right = NULL;
            baru->parent = node;
            node->right = baru;
            cout << "\n Node " << data << " berhasil ditambahkan ke child kanan " << baru->parent->data << endl;
            return baru;
        }
    }
}

```

// Ubah Data Tree

```

void update(char data, Pohon *node)
{
    if(isEmpty() == 1){
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }else{
        if( !node )
            cout << "\n Node yang ingin diganti tidak ada!!" << endl;
        else{
            char temp = node->data;
            node->data = data;
            cout << "\n Node " << temp << " berhasil diubah menjadi " << data << endl;
        }
    }
}

```

// Lihat Isi Data Tree

```

void retrieve( Pohon *node )
{
    if( isEmpty() == 1 ){
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }else{
        if( !node )
            cout << "\n Node yang ditunjuk tidak ada!" << endl;
        else{
            cout << "\n Data node : " << node->data << endl;
        }
    }
}

// Cari Data Tree
void find(Pohon *node)
{
    if(isEmpty() == 1){
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }else{
        if( !node )
            cout << "\n Node yang ditunjuk tidak ada!" << endl;
        else{
            cout << "\n Data Node : " << node->data << endl;
            cout << " Root : " << root->data << endl;

            if( !node->parent )
                cout << " Parent : (tidak punya parent)" << endl;
            else
                cout << " Parent : " << node->parent->data << endl;

            if( node->parent != NULL && node->parent->left != node && node->parent->right == node )

                cout << " Sibling : " << node->parent->left->data << endl;

            else if( node->parent != NULL && node->parent->right != node && node->parent->left == node )

                cout << " Sibling : " << node->parent->right->data << endl;

            else

                cout << " Sibling : (tidak punya sibling)" << endl;

            if( !node->left )

```

```

        cout << " Child Kiri : (tidak punya Child kiri)" << endl;
    else
        cout << " Child Kiri : " << node->left->data << endl;

    if( !node->right )
        cout << " Child Kanan : (tidak punya Child kanan)" << endl;
    else
        cout << " Child Kanan : " << node->right->data << endl;
    }
}
}

// Penelurusan (Traversal)
// preOrder
void preOrder(Pohon *node = root)
{
    if(isEmpty() == 1)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else{

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

// inOrder
void inOrder(Pohon *node = root)
{
    if(isEmpty() == 1)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else{

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

// postOrder
void postOrder(Pohon *node = root)
{
    if(isEmpty() == 1)

```

```

    cout << "\n Buat tree terlebih dahulu!" << endl;
else{
    if( node != NULL ){
        postOrder(node->left);
        postOrder(node->right);
        cout << " " << node->data << ", ";
    }
}
}

// Hapus Node Tree
void deleteTree(Pohon *node)
{
    if(isEmpty() == 1)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else{
        if( node != NULL ){
            if( node != root ){
                node->parent->left = NULL;
                node->parent->right = NULL;
            }
            deleteTree(node->left);
            deleteTree(node->right);

            if( node == root ){
                delete root;
                root = NULL;
            }else{
                delete node;
            }
        }
    }
}

// Hapus SubTree
void deleteSub(Pohon *node){
    if( isEmpty() == 1 )
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else{
        deleteTree(node->left);
        deleteTree(node->right);
        cout << "\n Node subtree " << node->data << " berhasil dihapus." <<
endl;
    }
}

// Hapus Tree

```

```

void clear(){
    if( isEmpty() == 1 )
        cout << "\n Buat tree terlebih dahulu!!" << endl;
    else{
        deleteTree(root);
        cout << "\n Pohon berhasil dihapus." << endl;
    }
}

// Cek Size Tree
int size(Pohon *node = root){
    if( isEmpty() == 1 ){
        cout << "\n Buat tree terlebih dahulu!!" << endl;
        return 0;
    }else{

        if( !node ){
            return 0;
        }else{
            return 1 + size( node->left ) + size(node->right);
        }

    }
}

// Cek Height Level Tree
int height( Pohon *node = root )
{
    if( isEmpty() == 1 ){
        cout << "\n Buat tree terlebih dahulu!!" << endl;
        return 0;
    }else{
        if( !node ){
            return 0;
        }else{
            int heightKiri = height( node->left );
            int heightKanan = height( node->right );

            if( heightKiri >= heightKanan ){
                return heightKiri + 1;
            }else{
                return heightKanan + 1;
            }

        }

    }
}

```

// Karakteristik Tree

```

void charateristic()
{
    cout << "\n Size Tree : " << size() << endl;
    cout << " Height Tree : " << height() << endl;
    cout << " Average Node of Tree : " << size() / height() << endl;
}

int main()
{
    buatNode('A');

    Pohon *nodeB, *nodeC, *nodeD, *nodeE, *nodeF, *nodeG, *nodeH, *nodeI,
    *nodeJ;

    nodeB = insertLeft('B', root);
    nodeC = insertRight('C', root);
    nodeD = insertLeft('D', nodeB);
    nodeE = insertRight('E', nodeB);
    nodeF = insertLeft('F', nodeC);
    nodeG = insertLeft('G', nodeE);
    nodeH = insertRight('H', nodeE);
    nodeI = insertLeft('I', nodeG);
    nodeJ = insertRight('J', nodeG);

    update('Z', nodeC);
    update('C', nodeC);

    retrieve(nodeC);

    find(root);

    cout << "\n PreOrder :" << endl;
    preOrder(nodeE);
    cout << "\n" << endl;
    cout << " InOrder :" << endl;
    inOrder(nodeE);
    cout << "\n" << endl;
    cout << " PostOrder :" << endl;
    postOrder(nodeE);
    cout << "\n" << endl;

    charateristic();

    deleteSub(nodeE);
    cout << "\n PreOrder :" << endl;

```

```
preOrder();  
cout << "\n" << endl;  
  
charateristic();  
  
}
```


TUGAS

Modifikasi Guided menjadi program menu dengan input data tree dari user!

~ SELAMAT MENGERJAKAN 😊 ~