LAPORAN PRAKTIKUM
STRUKTUR DATA DAN ALGORITME

MODUL XI

TREE

Disusun Oleh :

Nama : Fatkhurrohman Purnomo

NIM : 21102125

Dosen Pengampu

Ipam Fuaddina Adam, S.T., M.Kom.

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

INSTITUT TEKNOLOGI TELKOM PURWOKERTO

PURWOKERTO

2022

A. Dasar Teori

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.

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

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.

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.

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

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

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

Ref:

Modul XI: TREE

Tree pada C++ (Tree Awal) - nblognlife

B. 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();
}
```

Deskripsi:

- **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.

Output:

```
Node C berhasil diubah menjadi Z

Node Z berhasil diubah menjadi C

Data node : C

Data Node : A
Root : A
Parent : (tidak punya parent)
Sibling : (tidak punya sibling)
Child Kiri : B
Child Kanan : C

PreOrder :
E,  G,  I,  J,  H,

InOrder :
I,  G,  J,  E,  H,

PostOrder :
I,  J,  G,  H,  E,


Size Tree : 10
Height Tree : 5
Average Node of Tree : 2

Node subtree E berhasil dihapus.

PreOrder :
A,  B,  D,  E,  C,  F,
```

C. Tugas (Unguided)

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

```
// NAMA : FATKHURROHMAN PURNOMO
// NIM : 21102125
// MODIF MENU DAN INPUTAN DARI USER

#include <iostream>
```

```
using namespace std;

/// PROGRAM MENU 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 << "Node " << data << " berhasil dibuat!" <<
endl;
    }
    else
    {
```

```
        cout << "\n Pohon sudah dibuat" << endl;
    }
}

// Tambah Kiri
Pohon *insertLeft(char data, Pohon *node)
{
    if (isEmpty() == 1)
    {
        cout << "\n Buat tree 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 tree 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 (!root)
    {
        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 (!root)
    {
        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 (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\n Node yang ditunjuk tidak ada!" <<
endl;
        else
        {
            cout << "\n Data Node\t: " << node->data <<
endl;
            cout << " Root\t\t: " << root->data << endl;

            if (!node->parent)
```

```
                cout << " Parent\t\t: (tidak punya parent)"
<< endl;
            else
                cout << " Parent\t\t: " << node->parent-
>data << endl;

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

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

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

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

            else

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

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

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

// Penelurusan (Traversal)
// preOrder
void preOrder(Pohon *node = root)
{
```

```
    if (!root)
        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 (!root)
        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 (!root)
        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 (!root)
        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 (!root)
        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 (!root)
        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 (!root)
    {
        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 (!root)
    {
        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\t\t: " << size() << endl;
    cout << " Height Tree\t\t: " << height() << endl;
    cout << " Average Node of Tree\t: " << size() / height()
<< endl;
}

// main
void menu()
{
    int pil, i = 0;   // Counter array var. data
    int j = 0, k = 0; // Counter array var. node
    char data[100], par, baru;
    Pohon *node[100];

    init();

    do
    {
        system("cls");
```

```
        cout <<
"========================================================="
<< endl;
        cout << "|            Binary Tree Mod Menu &
Inputan             |" << endl;
        cout <<
"========================================================="
<< endl;
        cout << " 1. Buat
Tree                                          " << endl;
        cout << " 2. Tambah Tree
Kiri                                    " << endl;
        cout << " 3. Tambah Tree
Kanan                                   " << endl;
        cout << " 4.
Ubah                                              " <<
endl;
        cout << " 5.
Isi                                               " <<
endl;
        cout << " 6.
cari                                              " <<
endl;
        cout << " 7.
PreOrder                                          " <<
endl;
        cout << " 8.
InOrder                                           " <<
endl;
        cout << " 9.
PostOrder                                         " <<
endl;
        cout << " 10. Hapus
Tree                                        " << endl;
        cout << " 11. Hapus
SubTree                                     " << endl;
        cout << " 12. Karakteristik
Tree                                  " << endl;
        cout << " 0.
Keluar                                            " <<
endl;
        cout <<
"========================================================="
<< endl;
        cout << " Masukkan pilihan anda : ";
```

```
        cin >> pil;
        cout << endl;

        switch (pil)
        {
        case 1:
            cout << "Membuat Tree" << endl;
            cout << "Masukkan Node: ";
            cin >> data[i]; // Masukkan data

            buatNode(data[i]); // Buat node

            i++; // melanjutkan counter array var. data
            break;
        case 2:
            cout << "Tambah Tree Kiri" << endl;
            cout << " Masukkan Node\t\t: ";
            cin >> data[i]; // Masukkan data

            if (root->left == NULL) // Jika node kiri kosong
            {
                node[j] = insertLeft(data[i], root); //
Tambah node kiri
            }
            else
            {
                cout << " Masukkan Node Parent\t: ";
                cin >> par;                    // Masukkan
data parent
                while (par != node[k]->data) // Cari node
parent
                {
                    k++; // melanjutkan counter array var.
node
                }
                node[j] = insertLeft(data[i], node[k]); //
Tambah node kiri
            }

            i++; // melanjutkan counter array var. data
            j++; // melanjutkan counter array var. node
            break;
        case 3:
            cout << "Tambah Tree Kanan" << endl;
```

```
            cout << " Masukkan Node\t\t: ";
            cin >> data[i]; // Masukkan data

            if (root->right == NULL) // Jika node kanan
kosong

            {
                node[j] = insertRight(data[i], root); //
Tambah node kanan
            }
            else
            {
                cout << " Masukkan Node Parent\t: "; //
Masukkan data parent
                cin >> par;
                while (par != node[k]->data) // Cari node
parent
                {
                    k++;
                }
                node[j] = insertRight(data[i], node[k]);
            }

            i++; // melanjutkan counter array var. data
            j++; // melanjutkan counter array var. data
            break;
        case 4:
            cout << "Ubah" << endl;
            cout << " Masukkan Node baru: ";
            cin >> baru;
            cout << " Masukkan Node yang ingin diubah: ";
            cin >> par;

            if (par == root->data) // Jika node yang diubah
adalah root
            {
                update(baru, root); // Ubah node root
            }
            else
            {
                while (par != node[k]->data) // Cari node
root
                {
                    k++; // melanjutkan counter array var.
node
```

```
                }
                update(baru, node[k]); // Ubah node root
            }

            break;
        case 5:
            cout << "Isi" << endl;
            cout << " Masukkan Node yang ingin dilihat
datanya: ";
            cin >> par;

            if (par == root->data) // Jika node yang dilihat
adalah root
            {
                retrieve(root); // Lihat data root
            }
            else
            {
                while (par != node[k]->data) // Cari node
root
                {
                    k++;
                }
                retrieve(node[k]); // Lihat data root
            }

            break;
        case 6:
            cout << "Cari" << endl;
            cout << " Masukkan Node yang ingin dicari: ";
            cin >> par;

            if (par == root->data)
            {
                find(root); // Cari node root
            }
            else
            {
                while (par != node[k]->data)
                {
                    k++;
                }
                find(node[k]);
            }
```

```
            break;
        case 7:
            cout << "PreOrder" << endl;
            cout << " Dari Node apa yang ingin ditelusuri:
";
            cin >> par;

            if (par == root->data)
            {
                preOrder(root);
            }
            else
            {
                while (par != node[k]->data)
                {
                    k++;
                }

                preOrder(node[k]);
            }

            break;
        case 8:
            cout << "InOrder" << endl;
            cout << " Dari Node apa yang ingin ditelusuri:
";
            cin >> par;

            if (par == root->data)
            {
                inOrder(root);
            }
            else
            {
                while (par != node[k]->data)
                {
                    k++;
                }

                inOrder(node[k]);
            }

            break;
        case 9:
            cout << "PostOrder" << endl;
```

```
            cout << " Dari Node apa yang ingin ditelusuri:
";
            cin >> par;

            if (par == root->data)
            {
                postOrder(root);
            }
            else
            {
                while (par != node[k]->data)
                {
                    k++;
                }

                postOrder(node[k]);
            }

            break;
        case 10:
            cout << "Hapus Tree" << endl;
            deleteTree(root); // Hapus tree
            break;
        case 11:
            cout << "Hapus SubTree" << endl;
            cout << " Masukkan Sub Tree yang ingin dihapus:
";
            cin >> par;

            if (par == root->data)
            {
                deleteSub(root);
            }
            else
            {
                while (par != node[k]->data)
                {
                    k++;
                }

                deleteSub(node[k]);
            }

            break;
        case 12:
```

```
            cout << "Karakteristik Tree" << endl;
            charateristic(); // Cari karakteristik tree
            break;

        case 0:
            cout << "Keluar" << endl;
            break;
        default:
            cout << " Pilihan tidak ada" << endl;
        }
        cout << endl;

        k = 0; // Reset counter array var. node

        system("pause");
    } while (pil != 0); // Selama pilihan tidak 0
}

int main()
{
    mainProgram();
    return 0;
}
```

Deskripsi:

- **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.
- **Menu**: untuk memanggil fungsi-fungsi yang ada dalam program, menggunakan input user supaya lebih fleksibel sesuai keinginan.

Output:

```
============================================================
|            Binary Tree Mod Menu & Inputan                |
============================================================
 1. Buat Tree
 2. Tambah Tree Kiri
 3. Tambah Tree Kanan
 4. Ubah
 5. Isi
 6. cari
 7. PreOrder
 8. InOrder
 9. PostOrder
 10. Hapus Tree
 11. Hapus SubTree
 12. Karakteristik Tree
 0. Keluar
============================================================
```

```
 Masukkan pilihan anda : 1

Membuat Tree
Masukkan Node: A
Node A berhasil dibuat!
```

```
 Masukkan pilihan anda : 2

Tambah Tree Kiri
 Masukkan Node           : B

 Node B berhasil ditambahkan ke child kiri A
```

```
 Masukkan pilihan anda : 3

Tambah Tree Kanan
 Masukkan Node           : C

 Node C berhasil ditambahkan ke child kanan A
```

```
 Masukkan pilihan anda : 4

Ubah
 Masukkan Node baru: Z
 Masukkan Node yang ingin diubah: C

 Node C berhasil diubah menjadi Z
```

```
PreOrder
 Dari Node apa yang ingin ditelusuri: A
 A,   B,   D,   E,   G,   I,   J,   H,   C,   F,
sh: 1: pause: not found
sh: 1: cls: not found
```

```
InOrder
 Dari Node apa yang ingin ditelusuri: A
 D,   B,   I,   G,   J,   E,   H,   A,   F,   C,
```

```
PostOrder
 Dari Node apa yang ingin ditelusuri: A
 D,   I,   J,   G,   H,   E,   B,   F,   C,   A,
```

```
 Masukkan pilihan anda : 10

Hapus Tree
```

```
 Masukkan pilihan anda : 11

Hapus SubTree
 Masukkan Sub Tree yang ingin dihapus: G

 Node subtree G berhasil dihapus.
```

D. Kesimpulan

1. Bisa membuat program Tree
2. Belajar lebih dalam pengaplikasian graph dengan lebih baik

3. Lebih mahir dalam menggunakan bahasa C++
4. Bisa melakukan problem solving bagi program yang error
5. Lebih paham dalam membuat program
6. Melatih daya pikir, imajinasi, dan langkah-langkah dalam membuat program