LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL X

TREE (BAGIAN PERTAMA)

Disusun Oleh :
NAMA : Besthian Guido Rafael Simbolon
NIM : 103112430258

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Queue atau antrean merupakan salah satu struktur data linier yang menerapkan prinsip FIFO (First In, First Out), artinya elemen yang pertama kali dimasukkan ke dalam antrean akan menjadi elemen pertama yang dikeluarkan. Konsep ini sangat mudah dipahami karena menyerupai antrean dalam kehidupan sehari-hari, seperti saat mengantre di loket atau kasir, di mana orang yang datang lebih dulu akan dilayani lebih dulu, sementara yang datang belakangan harus menunggu giliran.

Dalam konteks pemrograman, khususnya pada implementasi Queue menggunakan array di bahasa C++, terdapat dua penunjuk utama, yaitu head (atau front) dan tail (atau rear). Head berfungsi sebagai penunjuk posisi elemen yang akan dikeluarkan, sedangkan tail menunjukkan posisi tempat elemen baru akan dimasukkan. Proses memasukkan elemen ke dalam antrean disebut enqueue, sementara proses mengeluarkan elemen dari antrean disebut dequeue.

Penggunaan array sebagai media penyimpanan queue memiliki kelebihan dari segi kesederhanaan dan akses data yang cepat. Namun, keterbatasan utamanya adalah ukuran array yang bersifat statis, sehingga kapasitas antrean harus ditentukan sejak awal. Jika seluruh ruang array telah terisi, maka operasi enqueue tidak dapat dilakukan lagi meskipun secara logika masih terdapat ruang kosong di bagian depan akibat operasi dequeue sebelumnya. Oleh karena itu, dalam praktiknya sering dikembangkan variasi seperti circular queue untuk mengoptimalkan pemanfaatan memori.

B. Guided (berisi screenshot source code & output program disertai penjelasannya)

Guided 1

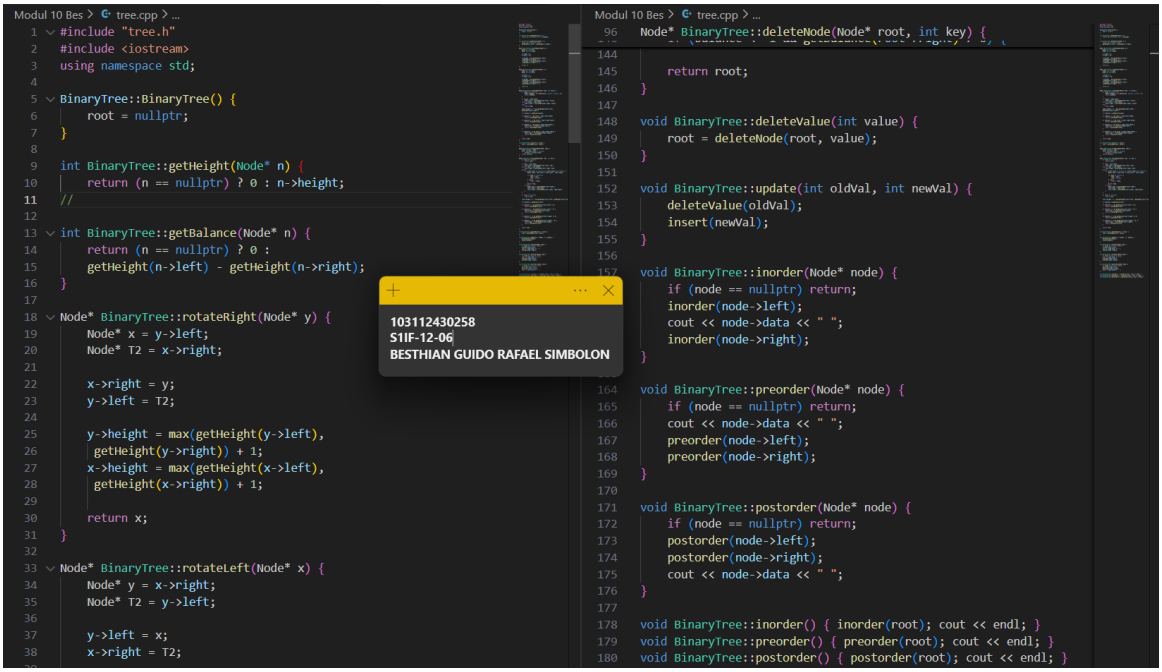
Main.cpp

Modul 10 Bes > main.cpp > main()

```cpp
#include <iostream>
#include "tree.h"
#include "tree.cpp"

using namespace std;

int main(){
    BinaryTree tree;

    cout << "=== INSERT DATA ===" << endl;
    tree.insert(10);
    tree.insert(15);
    tree.insert(20);
    tree.insert(30);
    tree.insert(35);
    tree.insert(40);
    tree.insert(50);


    cout << "Data yang diinsert: 10, 15, 20, 30, 35, 40, 50" << endl;

    cout << "\nTraversal setelah insert:" << endl;
    cout << "Inorder: "; tree.inorder();
    cout << "Preorder: "; tree.preorder();
    cout << "Postorder: "; tree.postorder();

    cout << "\n=== UPDATE DATA ===" << endl;
    cout << "Sebelum update (20 -> 25):" << endl;
    cout << "Inorder: "; tree.inorder();

    tree.update(20, 25);

    cout << "Setelah update (20 -> 25):" << endl;
    cout << "Inorder: "; tree.inorder();

    cout << "\n=== DELETE DATA ===" << endl;
    cout << "Sebelum delete (hapus subtree dengan root = 3-):"
    << endl;
    cout << "Inorder    :"; tree.inorder();

    tree.deleteValue(30);

    cout << "Setelah delete (subtree root = 30 dihapus):"
    <<endl;
    cout << "Inorder    :"; tree.inorder();

    return 0;
```

103112430258
S1IF-12-06
BESTHIAN GUIDO RAFAEL SIMBOLON

Tree.cpp

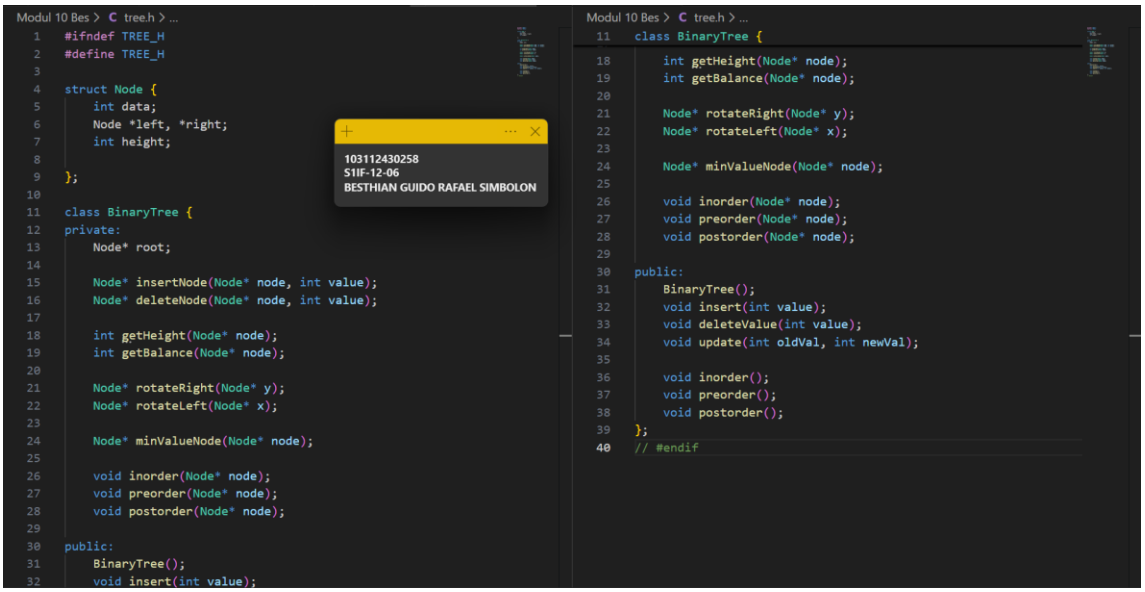


```cpp
#include "tree.h"
#include <iostream>
using namespace std;

BinaryTree::BinaryTree() {
    root = nullptr;
}

int BinaryTree::getHeight(Node* n) {
    return (n == nullptr) ? 0 : n->height;
}

int BinaryTree::getBalance(Node* n) {
    return (n == nullptr) ? 0 :
    getHeight(n->left) - getHeight(n->right);
}

Node* BinaryTree::rotateRight(Node* y) {
    Node* x = y->left;
    Node* T2 = x->right;

    x->right = y;
    y->left = T2;

    y->height = max(getHeight(y->left),
     getHeight(y->right)) + 1;
    x->height = max(getHeight(x->left),
     getHeight(x->right)) + 1;

    return x;
}

Node* BinaryTree::rotateLeft(Node* x) {
    Node* y = x->right;
    Node* T2 = y->left;

    y->left = x;
    x->right = T2;
```

103112430258
S1IF-12-06
BESTHIAN GUIDO RAFAEL SIMBOLON

```cpp
Node* BinaryTree::deleteNode(Node* root, int key) {

    return root;
}

void BinaryTree::deleteValue(int value) {
    root = deleteNode(root, value);
}

void BinaryTree::update(int oldVal, int newVal) {
    deleteValue(oldVal);
    insert(newVal);
}

void BinaryTree::inorder(Node* node) {
    if (node == nullptr) return;
    inorder(node->left);
    cout << node->data << " ";
    inorder(node->right);
}

void BinaryTree::preorder(Node* node) {
    if (node == nullptr) return;
    cout << node->data << " ";
    preorder(node->left);
    preorder(node->right);
}

void BinaryTree::postorder(Node* node) {
    if (node == nullptr) return;
    postorder(node->left);
    postorder(node->right);
    cout << node->data << " ";
}

void BinaryTree::inorder() { inorder(root); cout << endl; }
void BinaryTree::preorder() { preorder(root); cout << endl; }
void BinaryTree::postorder() { postorder(root); cout << endl; }
```

Tree.h

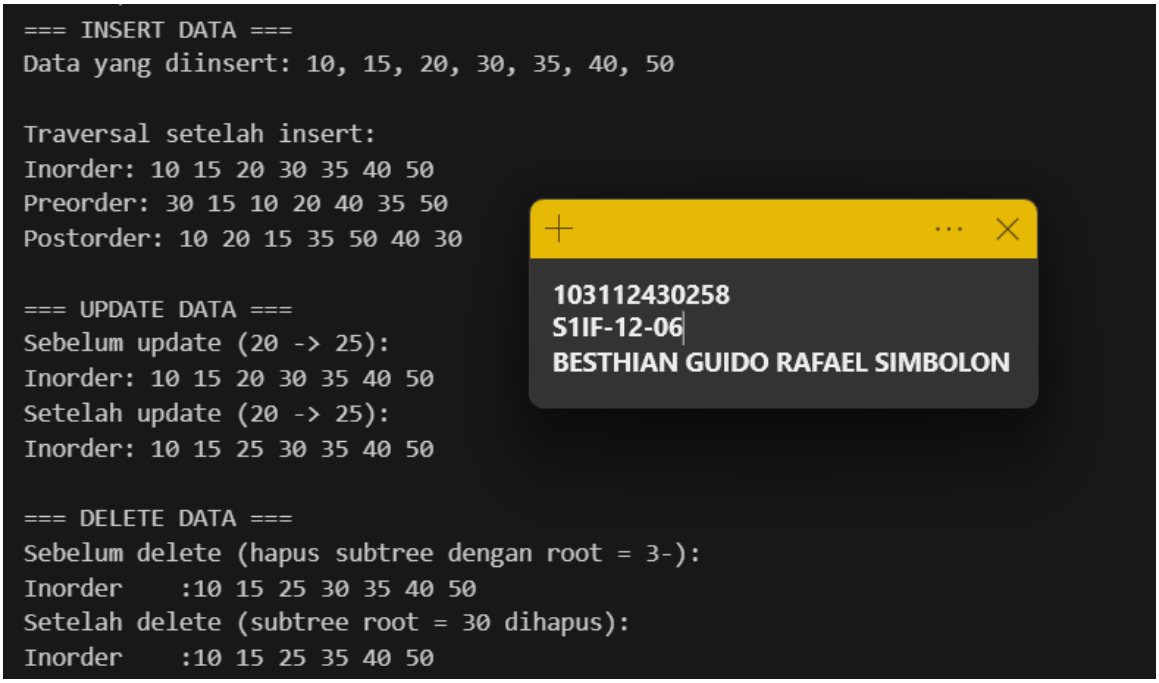


```cpp
#ifndef TREE_H
#define TREE_H

struct Node {
    int data;
    Node *left, *right;
    int height;

};

class BinaryTree {
private:
    Node* root;

    Node* insertNode(Node* node, int value);
    Node* deleteNode(Node* node, int value);

    int getHeight(Node* node);
    int getBalance(Node* node);

    Node* rotateRight(Node* y);
    Node* rotateLeft(Node* x);

    Node* minValueNode(Node* node);

    void inorder(Node* node);
    void preorder(Node* node);
    void postorder(Node* node);

public:
    BinaryTree();
    void insert(int value);
```

103112430258
S1IF-12-06
BESTHIAN GUIDO RAFAEL SIMBOLON

```cpp
class BinaryTree {
    int getHeight(Node* node);
    int getBalance(Node* node);

    Node* rotateRight(Node* y);
    Node* rotateLeft(Node* x);

    Node* minValueNode(Node* node);

    void inorder(Node* node);
    void preorder(Node* node);
    void postorder(Node* node);

public:
    BinaryTree();
    void insert(int value);
    void deleteValue(int value);
    void update(int oldVal, int newVal);

    void inorder();
    void preorder();
    void postorder();
};
// #endif
```

Screenshots Output



Deskripsi:

Program ini merupakan contoh penerapan struktur data AVL Tree, yaitu pengembangan dari Binary Search Tree yang memiliki kemampuan menyeimbangkan tinggi pohonnya secara otomatis. Walaupun kelas yang digunakan diberi nama BinaryTree, di dalam implementasinya telah diterapkan mekanisme khas AVL Tree, seperti perhitungan tinggi setiap node serta operasi rotasi ke kiri dan ke kanan. Mekanisme ini bertujuan untuk mencegah pohon menjadi tidak seimbang, sehingga operasi pencarian, penyisipan, dan penghapusan data tetap berjalan secara efisien.

Fungsi utama yang disediakan dalam program meliputi insert untuk menambahkan data, delete untuk menghapus data, serta update yang dilakukan dengan cara menghapus nilai lama kemudian menyisipkan nilai baru. Pada fungsi main, program menampilkan contoh penggunaan seluruh fitur tersebut dengan memasukkan sejumlah data ke dalam pohon dan menampilkan strukturnya melalui tiga metode traversal, yaitu Inorder untuk menampilkan data terurut, Preorder untuk melihat struktur pohon, dan Postorder untuk menelusuri node setelah anak-anaknya.

C. Unguided/Tugas (berisi screenshot source code & output program disertai penjelasannya)

Unguided 1

main.cpp

Modul 10 Bes > ungu > main.cpp > main()

```cpp
#include <iostream>
#include "bstree.h"
#include "bstree.cpp"
using namespace std;

int main() {
    cout << "Hello World" << endl;
    address root = Nil;

    insertNode(root, 1);
    insertNode(root, 2);
    insertNode(root, 6);
    insertNode(root, 4);
    insertNode(root, 5);
    insertNode(root, 3);
    insertNode(root, 6);
    insertNode(root, 7);

    InOrder(root);

    cout << endl;
    return 0;
}
```

```
103112430258
S1IF-12-06
BESTHIAN GUIDO RAFAEL SIMBOLON
```

bstree.cpp

Modul 10 Bes > ungu > bstree.cpp > findNode(infotype, address)

```cpp
#include "bstree.h"
address alokasi(infotype x) {
    address P = new Node;
    P->info = x;
    P->left = Nil;
    P->right = Nil;
    return P;
}

void insertNode(address &root, infotype x) {

    if (root == Nil) {
        root = alokasi(x);
    }
    else {
        if (x < root->info) {
            insertNode(root->left, x);
        }
        else if (x > root->info) {
            insertNode(root->right, x);
        }
    }
}

void InOrder(address root) {
    if (root != Nil) {
        InOrder(root->left);
        cout << root->info << " - ";
        InOrder(root->right);
    }
}

address findNode(infotype x, address root) {
    if (root == Nil) return Nil;
    if (root->info == x) return root;
    if (x < root->info) {
        return findNode(x, root->left);
    } else {
        return findNode(x, root->right);
```

Modul 10 Bes > ungu > bstree.cpp > findNode(infotype, address)

```cpp
void InOrder(address root) {
    if (root != Nil) {
        InOrder(root->left);
        cout << root->info << " - ";
        InOrder(root->right);
    }
}

address findNode(infotype x, address root) {
    if (root == Nil) return Nil;
    if (root->info == x) return root;
    if (x < root->info) {
        return findNode(x, root->left);
    } else {
        return findNode(x, root->right);
    }
}
```

```
103112430258
S1IF-12-06
BESTHIAN GUIDO RAFAEL SIMBOLON
```

bstree.h

```
Modul 10 Bes > ungu > C bstree.h > ...
1   #ifndef BSTREE_H
2   #define BSTREE_H
3   #include <iostream>
4   #define Nil NULL
5   using namespace std;
6
7   typedef int infotype;
8   typedef struct Node *address;
9
10  struct Node {
11      infotype info;
12      address left;
13      address right;
14  };
15
16  address alokasi(infotype x);
17  void insertNode(address &root, infotype x);
18  void InOrder(address root);
19
20  address findNode(infotype x, address root);
21
22  #endif
```

103112430258
S1IF-12-06
BESTHIAN GUIDO RAFAEL SIMBOLON

Screenshot Output

```
[Running] cd "d:\KULIAH\SEMESTER 3\STRUKTUR
SD\LAPRAKSD\Modul 10 Bes\ungu\"main
Hello World
1 - 2 - 3 - 4 - 5 - 6 - 7 -

[Done] exited with code=0 in 0.995 seconds
```

103112430258
S1IF-12-06
BESTHIAN GUIDO RAFAEL SIMBOLON

Deskripsi:

Program ini merupakan implementasi struktur data Binary Search Tree (BST) sederhana yang tidak menggunakan mekanisme penyeimbangan otomatis seperti rotasi. Data disusun berdasarkan aturan BST, yaitu setiap nilai yang lebih kecil dari node induk ditempatkan pada subtree kiri, sedangkan nilai yang lebih besar ditempatkan pada subtree kanan. Program ini juga dirancang untuk menolak data duplikat, sehingga jika suatu nilai dimasukkan lebih dari satu kali, nilai tersebut tidak akan ditambahkan kembali ke dalam pohon.

Selain menyediakan fungsi dasar untuk menambahkan data (insert) dan menampilkan data secara terurut menggunakan traversal InOrder, program ini dilengkapi dengan beberapa fungsi rekursif untuk keperluan analisis. Fungsi-fungsi tersebut digunakan untuk menghitung jumlah seluruh node dalam pohon, menjumlahkan semua nilai yang tersimpan, serta menentukan kedalaman atau tinggi maksimum dari pohon BST.

Unguided 2

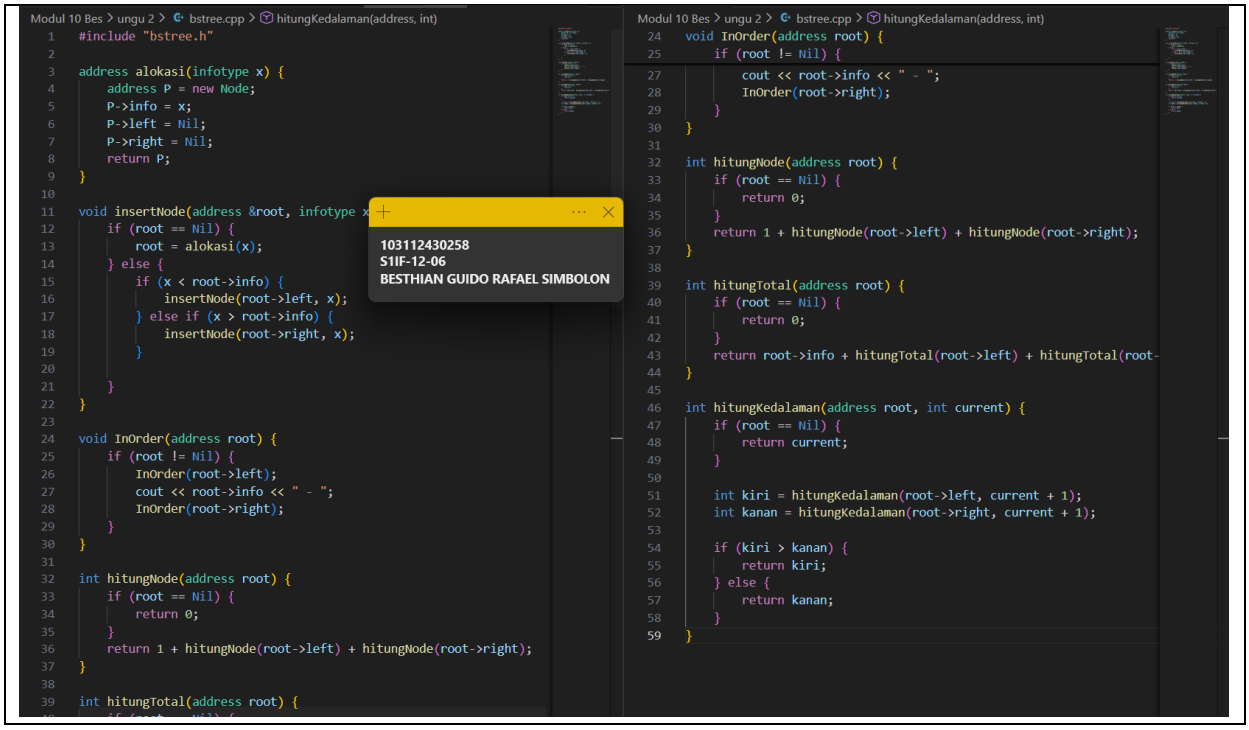
main.cpp

Modul 10 Bes > ungu 2 > main.cpp > main()

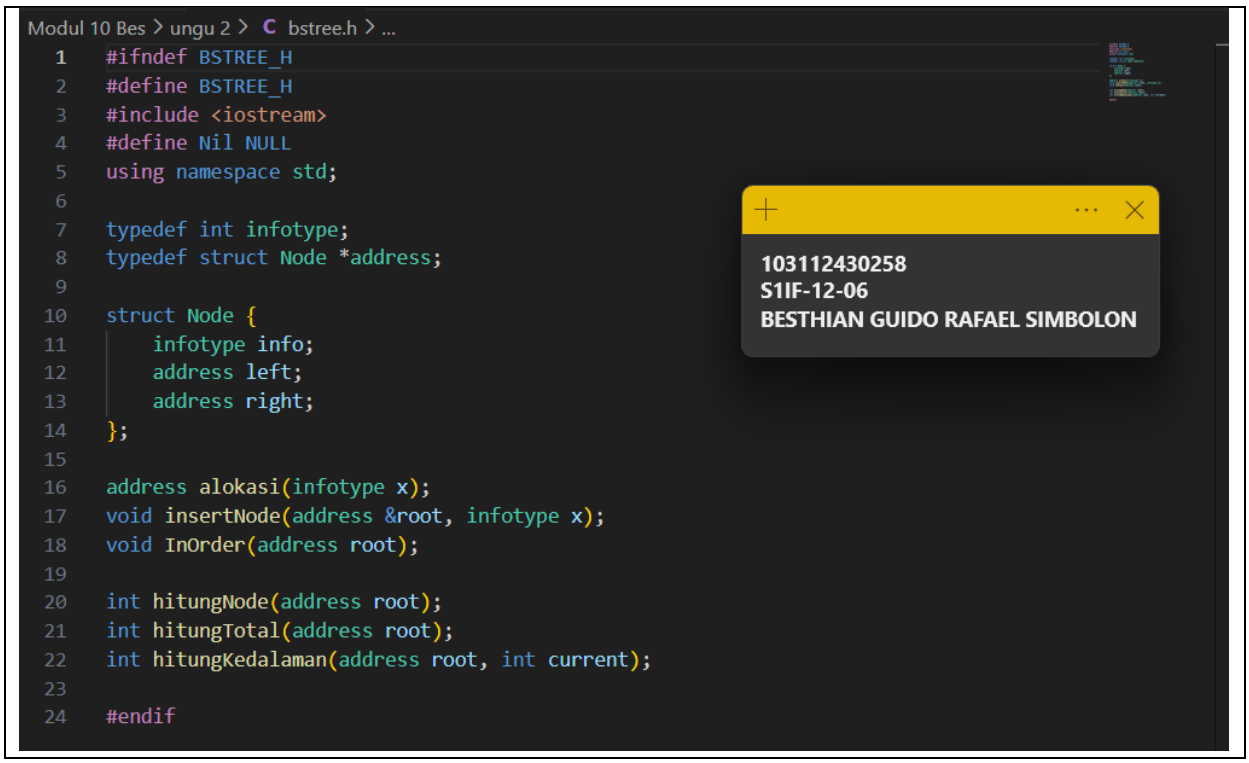
```cpp
#include <iostream>
#include "bstree.h"
#include "bstree.cpp"
using namespace std;

int main() {
    cout << "Hello World" << endl;

    address root = Nil;

    insertNode(root, 1);
    insertNode(root, 2);
    insertNode(root, 6);
    insertNode(root, 4);
    insertNode(root, 5);
    insertNode(root, 3);
    insertNode(root, 6);
    insertNode(root, 7);

    InOrder(root);
    cout << "\n";

    cout << "kedalaman : " << hitungKedalaman(root, 0) << endl;
    cout << "jumlah Node : " << hitungNode(root) << endl;
    cout << "total : " << hitungTotal(root) << endl;

    return 0;
}
```

103112430258
S1IF-12-06
BESTHIAN GUIDO RAFAEL SIMBOLON

bstree.cpp



bstree.h



Screenshot Output

```
[Running] cd "d:\KULIAH\SEMESTER 3\STRUKTUR DATA\CPP SD\LAPRAKSD\Modul 10 Bes\ungu 2\" &&
SD\LAPRAKSD\Modul 10 Bes\ungu 2\"main
Hello World
1 - 2 - 3 - 4 - 5 - 6 - 7 -
kedalaman : 5
jumlah Node : 7
total : 28
```

103112430258
S1IF-12-06
BESTHIAN GUIDO RAFAEL SIMBOLON

Deskripsi

Program ini merupakan penerapan struktur data Binary Search Tree (BST) dasar yang tidak dilengkapi dengan proses penyeimbangan otomatis melalui rotasi. Data angka disusun mengikuti aturan BST, yaitu nilai yang lebih kecil dari node induk diletakkan pada bagian kiri pohon, sedangkan nilai yang lebih besar berada di bagian kanan. Program ini juga mengatur agar tidak terjadi duplikasi data, sehingga ketika nilai yang sama dimasukkan lebih dari satu kali, nilai tersebut akan diabaikan dan tidak disimpan kembali.

Selain menyediakan fungsi utama untuk menambahkan data ke dalam pohon serta menampilkan data secara berurutan melalui traversal InOrder, program ini juga memiliki beberapa fungsi rekursif untuk keperluan perhitungan. Fungsi-fungsi tersebut digunakan untuk mengetahui jumlah keseluruhan node, menghitung total nilai data yang tersimpan, serta menentukan kedalaman atau tinggi maksimum dari pohon BST.

Unguided 3

main.cpp

Modul 10 Bes > ungu 3 > main.cpp > main()

```cpp
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* left;
    Node* right;
};

Node* createNode(int data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

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

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

// Fungsi untuk menghapus tree (membersihkan memori)
void deleteTree(Node* root) {
    if (root == NULL) {
        return;
```

103112430258
S1IF-12-06
BESTHIAN GUIDO RAFAEL SIMBOLON

Modul 10 Bes > ungu 3 > main.cpp > main()

```cpp

// Fungsi untuk menghapus tree (membersihkan memori)
void deleteTree(Node* root) {
    if (root == NULL) {
        return;
    }
    deleteTree(root->left);
    deleteTree(root->right);
    delete root;
}

int main() {
    // Membangun pohon
    // Membuat node
    Node* root = createNode(6);

    // Membuat subtree kiri
    root->left = createNode(4);
    root->left->left = createNode(2);
    root->left->right = createNode(5);
    root->left->left->left = createNode(1);
    root->left->left->right = createNode(3);

    // Membuat subtree kanan
    root->right = createNode(7);

    // Mencetak hasil traversal
    cout << "Hasil Pre-order traversal: ";
    preOrder(root);
    cout << endl;

    cout << "Hasil Post-order traversal: ";
    postOrder(root);
    cout << endl;

    deleteTree(root);

    return 0;
}
```

Screenshot Output

Deskripsi

Program ini merupakan contoh penerapan struktur data Binary Tree sederhana dengan proses pembentukan pohon yang dilakukan secara manual atau statis. Berbeda dengan program sebelumnya yang menyusun node secara otomatis berdasarkan perbandingan nilai, pada kode ini hubungan antar node ditentukan secara langsung dengan mengatur pointer left dan right di dalam fungsi main. Pendekatan ini bertujuan untuk memperlihatkan bentuk struktur pohon secara eksplisit.

Fungsi utama program ini adalah mendemonstrasikan proses penelusuran pohon menggunakan metode Pre-order dan Post-order. Selain itu, program juga dilengkapi dengan fungsi deleteTree yang berperan penting untuk menghapus seluruh node dari memori secara rekursif, sehingga mencegah terjadinya kebocoran memori setelah program selesai dijalankan.

D. Kesimpulan

Secara umum, ketiga program tersebut menggambarkan tahapan pembelajaran struktur data Tree secara berurutan, dimulai dari konsep paling dasar hingga penerapan algoritma yang lebih kompleks dan efisien. Tahap awal ditunjukkan melalui unguided3.cpp, yang menitikberatkan pada pemahaman struktur pohon secara mendasar. Pada program ini, pohon dibentuk secara manual dengan menghubungkan pointer antar node tanpa aturan pengurutan nilai, sehingga mahasiswa dilatih untuk memahami hubungan antar node serta proses penelusuran data.

Tahap berikutnya berlanjut ke penerapan Binary Search Tree (BST) melalui bstree.cpp. Pada program ini, pembentukan pohon sudah dilakukan secara otomatis berdasarkan perbandingan nilai, dengan aturan nilai lebih kecil ditempatkan di sisi kiri dan nilai lebih besar di sisi kanan. Selain itu, mulai diperkenalkan penggunaan rekursi yang lebih mendalam untuk melakukan perhitungan statistik seperti kedalaman pohon dan total nilai node, meskipun struktur pohon masih berpotensi menjadi tidak seimbang tergantung urutan data yang dimasukkan.

Tahap paling lanjut ditunjukkan oleh implementasi AVL Tree pada tree.cpp. Program ini menyempurnakan konsep BST dengan menambahkan sistem penyeimbangan otomatis menggunakan rotasi dan penghitungan tinggi node. Fokus pembelajaran pada tahap ini tidak hanya pada penyusunan data, tetapi juga pada upaya menjaga kinerja operasi pencarian, penyisipan, dan penghapusan agar tetap optimal dan stabil dalam berbagai kondisi.

E. Referensi

Struktur Data Tree: Konsep, Jenis, dan Aplikasinya – artikel penjelasan dasar struktur data pohon, termasuk definisi binary tree, BST, dan AVL Tree.

Struktur Data Tree (Binary Search Tree & AVL Tree) – Penjelasan Singkat – artikel pembahasan BST dan AVL Tree sebagai variasi dari tree serta fungsinya dalam teknologi.

Langkah Menguasai Binary Search Tree: Panduan Praktis – artikel berbahasa Indonesia yang mengulas definisi, properti, dan operasi dasar BST.