LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL I0

Disusun Oleh :
NAMA : IBTIDA ZADA UTOMO
NIM : 103112430037

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Binary Search Tree (BST) merupakan salah satu struktur data pohon yang paling umum digunakan untuk proses penyimpanan dan pencarian data secara efisien. Dalam BST, setiap node memiliki maksimal dua anak, yaitu left child dan right child. Nilai pada subtree kiri selalu lebih kecil dari nilai root, sedangkan nilai pada subtree kanan selalu lebih besar, sehingga mempermudah proses pencarian karena bentuk datanya terurut. Pada modul ini juga diperkenalkan AVL Tree, yaitu bentuk BST yang mampu menyeimbangkan diri secara otomatis melalui perhitungan tinggi (height) dan selisih keseimbangan (balance factor). Ketika terjadi ketidakseimbangan, pohon akan melakukan rotasi seperti left rotation, right rotation, maupun kombinasi keduanya agar struktur tetap stabil. Selain itu, dipelajari pula berbagai traversal seperti pre-order, in-order, dan post-order, yang masing-masing memiliki tujuan berbeda: in-order menghasilkan data terurut, pre-order digunakan untuk menampilkan struktur pohon, dan post-order untuk proses penghapusan. Operasi insert, delete, update, serta analisis jumlah node, total nilai node, dan kedalaman pohon menjadi bagian penting dalam memahami bagaimana BST dan AVL bekerja secara dinamis.

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

Guided 1

```
#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);
        void deleteValue(int value);
        void update(int oldVal, int newVal);

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

```
#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;

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

    return y;
}

Node* BinaryTree::insertNode(Node* node, int value) {
    if (node == nullptr) {
        Node* newNode = new Node{value, nullptr, nullptr, 1};
        return newNode;
    }

    if (value < node->data)
        node->left = insertNode(node->left, value);
    else if (value > node->data)
        node->right = insertNode(node->right, value);
    else
        return node;

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

    int balance = getBalance(node);
```

```
    if (balance > 1 && value < node->left->data)
        return rotateRight(node);

    if (balance < -1 && value > node->right->data)
        return rotateLeft(node);

    if (balance > 1 && value > node->left->data) {
        node->left = rotateLeft(node->left);
        return rotateRight(node);
    }

    if (balance < -1 && value < node->right->data) {
        node->right = rotateRight(node->right);
        return rotateLeft(node);
    }

    return node;
}

void BinaryTree::insert(int value) {
    root = insertNode(root, value);
}

Node* BinaryTree::minValueNode(Node* node) {
    Node* current = node;
    while (current->left != nullptr)
        current = current->left;
    return current;
}

Node* BinaryTree::deleteNode(Node* root, int key) {
    if (root == nullptr)
        return root;

    if (key < root->data)
        root->left = deleteNode(root->left, key);
    else if (key > root->data)
        root->right = deleteNode(root->right, key);
    else {
        if ((root->left == nullptr) || (root->right == nullptr)) {
            Node* temp = root->left ? root->left : root->right;

            if (temp == nullptr) {
```

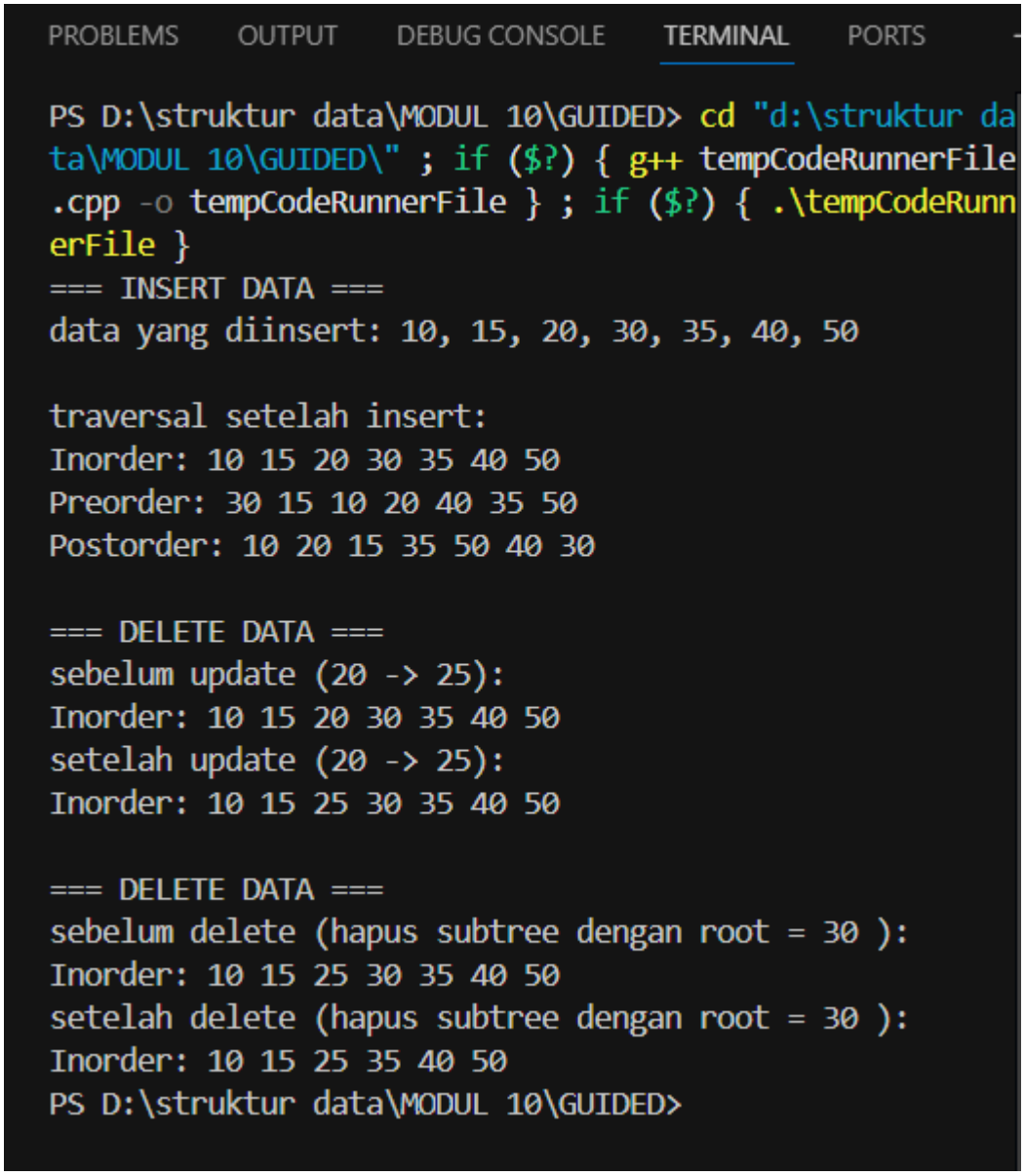
```
                temp = root;
                root = nullptr;
            } else {
                *root = *temp;
            }
            delete temp;
        } else {
            Node* temp = minValueNode(root->right);
            root->data = temp->data;
            root->right = deleteNode(root->right, temp->data);
        }
    }

    if (root == nullptr)
        return root;

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

    int balance = getBalance(root);

    if (balance > 1 && getBalance(root->left) >= 0)
        return rotateRight(root);

    if (balance > 1 && getBalance(root->left) < 0) {
        root->left = rotateLeft(root->left);
        return rotateRight(root);
    }

    if (balance < -1 && getBalance(root->right) <= 0)
        return rotateLeft(root);

    if (balance < -1 && getBalance(root->right) > 0) {
        root->right = rotateRight(root->right);
        return rotateLeft(root);
    }

    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; }
```

```
#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=== DELETE 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 = 30 ): "
<< endl;
    cout << "Inorder: "; tree.inorder();

    tree.deleteValue(30);

    cout << "setelah delete (hapus subtree dengan root = 30 ): "
<< endl;
    cout << "Inorder: "; tree.inorder();

    return 0;


}
```

Screenshots Output



Deskripsi: program diatas adalah program untuk menjelaskan struktur data pohon pencarian biner yang seimbang dan otomatis menggunakan bahasa c++ jadi program diatas ada insert data lalu transversal setelah insert kemudian delete data

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

Unguided 1

bstree.h

```
#ifndef BSTREE_H
#define BSTREE_H

typedef int infotype;

struct Node {
    infotype info;
    Node *left;
    Node *right;
};

typedef Node* address;
#define Nil NULL

address alokasi(infotype x);

void insertNode(address &root, infotype x);

address findNode(infotype x, address root);

void InOrder(address root);

#endif
```

bstree.cpp

```
#include <iostream>
#include "bstree.h"

using namespace std;

address alokasi(infotype x) {
    address P = new Node;
    P->info = x;
    P->left = Nil;
    P->right = Nil;
    return P;
}
```

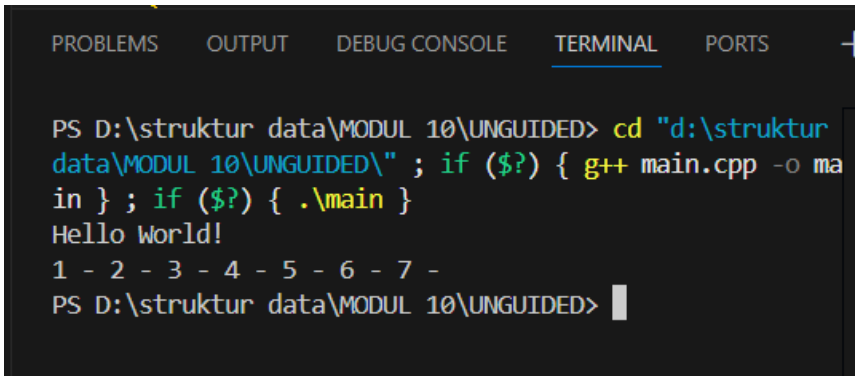
```
void insertNode(address &root, infotype x) {
    if (root == Nil) {
        root = alokasi(x);
    } else if (x < root->info) {
        insertNode(root->left, x);
    } else if (x > root->info) {
        insertNode(root->right, x);
    }
}

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

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

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, 7);

    InOrder(root);

    return 0;
}
```

Screenshots Output



Deskripsi: program diatas adalah program untuk meng implementasikan struktur data bst atau binary search tree yaitu pencarian menggunakan pohon biner jadi program nya akan otomatis mengurutkan data kita

Unguided 2

bstreee.h

```h
#ifndef BSTREE_H
#define BSTREE_H

typedef int infotype;

struct Node {
    infotype info;
    Node *left;
    Node *right;
};

typedef Node* address;
#define Nil NULL

address alokasi(infotype x);

void insertNode(address &root, infotype x);
address findNode(infotype x, address root);
void InOrder(address root);

int hitungJumlahNode(address root);
int hitungTotalInfo(address root);
int hitungKedalaman(address root, int start);

#endif
```

bstreee.cpp

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

using namespace std;

address alokasi(infotype x) {
    address P = new Node;
    P->info = x;
    P->left = Nil;
    P->right = Nil;
    return P;
}
```

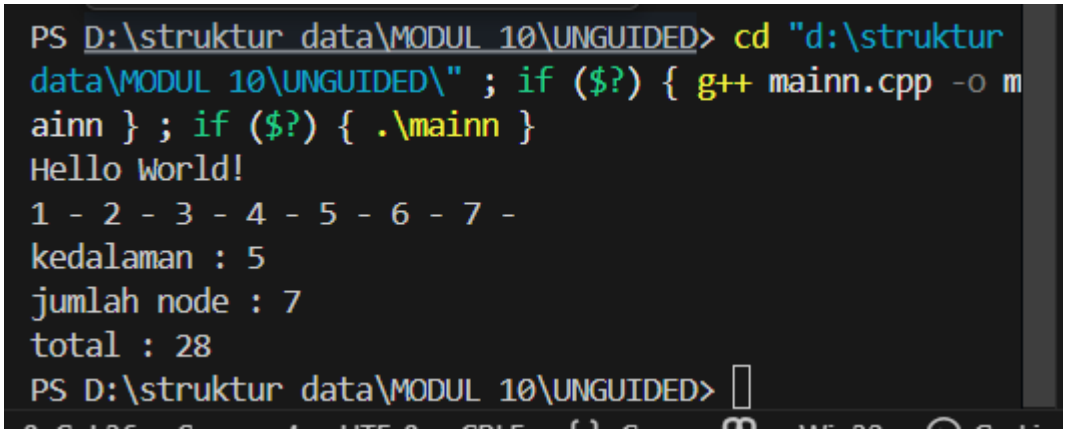
```
void insertNode(address &root, infotype x) {
    if (root == Nil) {
        root = alokasi(x);
    } else if (x < root->info) {
        insertNode(root->left, x);
    } else if (x > root->info) {
        insertNode(root->right, x);
    }
}

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

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

int hitungJumlahNode(address root) {
    if (root == Nil) return 0;
    return 1 + hitungJumlahNode(root->left) +
hitungJumlahNode(root->right);
}

int hitungTotalInfo(address root) {
    if (root == Nil) return 0;
    return root->info + hitungTotalInfo(root->left) +
hitungTotalInfo(root->right);
}

int hitungKedalaman(address root, int start) {
    if (root == Nil) return start;
    int leftDepth  = hitungKedalaman(root->left, start + 1);
    int rightDepth = hitungKedalaman(root->right, start + 1);

    return max(leftDepth, rightDepth);
```

```
}
```

mainn.cpp

```
#include <iostream>
#include "bstree.h"
#include "bstreee.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, 7);

    InOrder(root);

    cout << endl;
    cout << "kedalaman : " << hitungKedalaman(root, 0) << endl;
    cout << "jumlah node : " << hitungJumlahNode(root) << endl;
    cout << "total : " << hitungTotalInfo(root) << endl;

    return 0;
}
```

Screenshots Output



Deskripsi: program diatas adalah program untuk mengimplementasikan bst jadi nanti programnya akan otomatis mengurutkan data seperti yang ada di output dari angka 1 sampai 7 lalu jumlah node nya adalah 7

Unguided 3

bstreeee.h

```h
#ifndef BSTREE_H
#define BSTREE_H

typedef int infotype;

struct Node {
    infotype info;
    Node *left;
    Node *right;
};

typedef Node* address;
#define Nil NULL

address alokasi(infotype x);
void insertNode(address &root, infotype x);
address findNode(infotype x, address root);

void InOrder(address root);
void PreOrder(address root);
void PostOrder(address root);
```

```
int hitungJumlahNode(address root);
int hitungTotalInfo(address root);
int hitungKedalaman(address root, int start);

#endif
```

bstreeee.cpp

```
#include <iostream>
#include "bstree.h"

using namespace std;

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);
    }
}

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

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

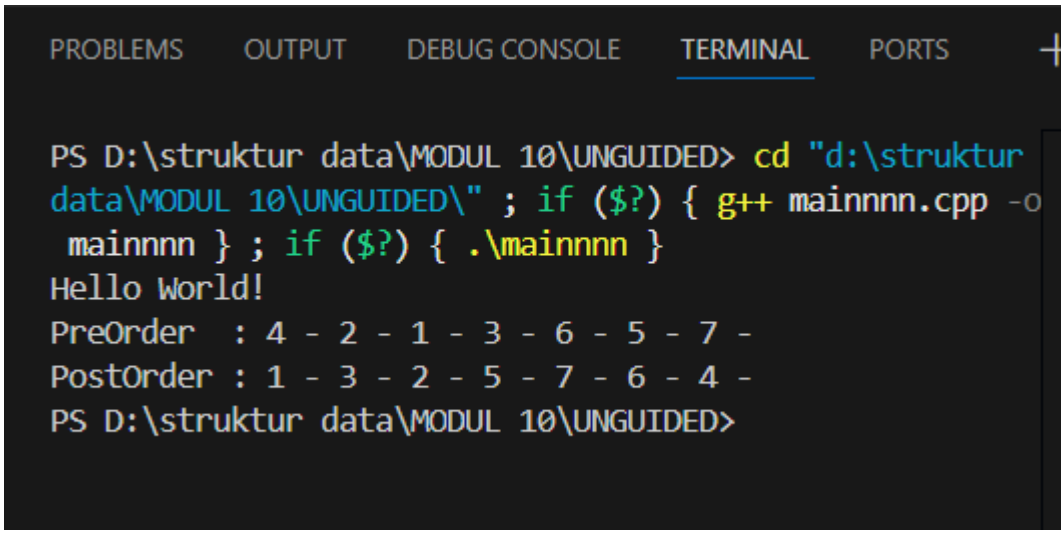
```
    }
}

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

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

int hitungJumlahNode(address root) {
    if (root == Nil) return 0;
    return 1 + hitungJumlahNode(root->left) +
hitungJumlahNode(root->right);
}

int hitungTotalInfo(address root) {
    if (root == Nil) return 0;
    return root->info + hitungTotalInfo(root->left) +
hitungTotalInfo(root->right);
}

int hitungKedalaman(address root, int start) {
    if (root == Nil) return start;
    int leftDepth  = hitungKedalaman(root->left, start + 1);
    int rightDepth = hitungKedalaman(root->right, start + 1);
    return (leftDepth > rightDepth) ? leftDepth : rightDepth;
}
```

mainnnn.cpp

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

```cpp
using namespace std;

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

    address root = Nil;

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

    cout << "PreOrder  : ";
    PreOrder(root);
    cout << endl;

    cout << "PostOrder : ";
    PostOrder(root);
    cout << endl;

    return 0;
}
```

Screenshots Output



Deskripsi:program diatas adalah program untuk print preorder dan postorder di struktur data binary search tree jadi preorder adalah mencetak dari root → kiri → kanan, kalau post order itu kiri → kanan → root

E. Kesimpulan

Dari praktikum ini dapat disimpulkan bahwa Binary Search Tree (BST) merupakan struktur data yang sangat efektif untuk mengelola data secara terurut sehingga proses pencarian, penambahan, dan penghapusan dapat dilakukan dengan cepat. Konsep keseimbangan pada AVL Tree menambah keunggulan BST dengan menjaga tinggi pohon tetap optimal, sehingga performa pencarian tetap stabil meskipun jumlah data bertambah besar. Implementasi operasi-operasi seperti insert, delete, dan update menunjukkan bagaimana perubahan data memengaruhi bentuk pohon secara keseluruhan. Selain itu, traversal in-order, pre-order, dan post-order membantu kita memahami urutan akses node sesuai kebutuhan. Melalui percobaan ini, mahasiswa dapat memahami bagaimana tree bekerja, bagaimana bentuknya berubah secara dinamis, dan bagaimana analisis jumlah node, total nilai, serta kedalaman membantu mengetahui karakteristik struktur pohon.

F. Referensi

- W3Schools Online Web Tutorials
- Belajar C++ #01: Pengenalan Bahasa C++ untuk Pemula
- Apa Itu C++? Mari Mengenal Bahasa Pemrograman Efisien Ini!
- Pengenalan Dasar Bahasa C/C++: Mulai dari Hello World hingga Struktur Dasar Program - Rumah Coding