LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL X

TREE

Disusun Oleh :
NAMA : DEVI YULIANA
NIM : 103112400151

Dosen
FAHRUDIN MUKTI WIBOWO
STRUKTUR DATA

PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Struktur data tree—terutama Binary Search Tree—adalah cara yang rapi dan efisien untuk menyimpan data yang punya hubungan terurut. Dengan memahami cara kerja insert, search, delete, dan berbagai jenis traversal, kita bisa melihat bagaimana tree berubah bentuk sesuai operasi yang dilakukan. Rekursi juga terbukti sangat membantu karena banyak proses di tree bisa diselesaikan dengan pola pemanggilan fungsi berulang.

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

tree.h

```
#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* root, int key);

    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
```

tree.cpp

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

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

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

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

inline void updateHeight(Node* n) {
    n->height = 1 + max(n->left ? n->left->height : 0,
                        n->right ? n->right->height : 0);
}

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

    x->right = y;
```

```
    y->left = T2;

    updateHeight(y);
    updateHeight(x);

    return x;
}

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

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

    updateHeight(x);
    updateHeight(y);

    return y;
}

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

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

    updateHeight(node);
    int balance = getBalance(node);

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

```
    }

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

    return node;
}

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

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

Node* BinaryTree::deleteNode(Node* root, int key) {
    if (!root) 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 || !root->right) {
            Node* temp = root->left ? root->left : root->right;
            if (!temp) {
                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) return root;

    updateHeight(root);
    int balance = getBalance(root);

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

    if (balance < -1) {
        if (getBalance(root->right) <= 0)
            return rotateLeft(root);
        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) return;
    inorder(node->left);
    cout << node->data << " ";
    inorder(node->right);
}

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

void BinaryTree::postorder(Node* node) {
    if (!node) 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; }
```

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

    tree.deleteValue(30);

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


    return 0;
}
```

Screenshots Output

```
PS C:\Users\musli\Downloads\SEMESTER 3\SEMESTER3\MODUL 10> cd "c:\Users\musli\Downloads\SEMESTER 3\SEMESTER3\MODUL 10\" ; if ($?) { g++ main.cpp -o main } ; if ($?)
 { .\main }
=== INSERT DATA ===
Data yang diinsert: 10, 15, 20, 30, 35, 40, 50

Traversal setelah insert:
Inorder   : 10 15 20 30 35 40 50
Preorder  : 30 15 10 20 40 35 50
Postorder : 10 20 15 35 50 40 30

=== UPDATE DATA ===
Sebelum update (20 -> 25):
Inorder   : 10 15 20 30 35 40 50
Setelah update (20 -> 25):
Inorder   : 10 15 25 30 35 40 50

=== DELETE DATA ===
Sebelum delete (hapus subtree dengan root = 30):
Inorder   : 10 15 25 30 35 40 50
Setelah delete (subtree root = 30 dihapus):
Inorder   : 10 15 25 35 40 50
```

NAMA : DEVI YULIANA
NIM : 103112400151
KELAS : S1IF-12-06

Deskripsi :

Program ini mengelola *Binary Search Tree (BST)* dengan menyediakan operasi insert, update, delete, dan traversal. Data awal dimasukkan ke dalam BST, kemudian hasil traversal (inorder, preorder, postorder) ditampilkan untuk menunjukkan struktur tree. Program juga melakukan update nilai pada node tertentu dan menampilkan kembali hasil traversal untuk melihat perubahan struktur. Selanjutnya, program menghapus sebuah subtree berdasarkan root tertentu dan menampilkan traversal inorder setelah penghapusan.

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

Unguided 1

bstree.h

```h
#ifndef BSTREE_H
#define BSTREE_H

#define Nil NULL

typedef int infotype;

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

typedef Node* address;

void alokasi(address &P, infotype X);
```

```
void insertNode(address &root, infotype X);
address findNode(address root, infotype X);
void printInOrder(address root);
void printPreOrder(address root);
void printPostOrder(address root);

int hitungNode(address root);
int hitungTotal(address root);
int hitungKedalaman(address root, int start);

#endif
```

bstree.cpp

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

void alokasi(address &P, infotype X){
    P = new Node;
    P->info = X;
    P->left = Nil;
    P->right = Nil;
}

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

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

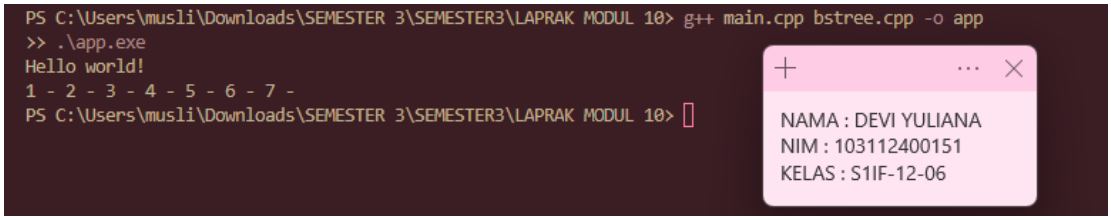
```
        return findNode(root->left, X);
    } else {
        return findNode(root->right, X);
    }
}

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

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

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

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

int hitungTotal(address root){
    if(root == Nil){
```

```
        return 0;
    } else {
        return root->info + hitungTotal(root->left) +
hitungTotal(root->right);
    }
}

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

main.cpp

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

    printInOrder(root);
    cout << endl;

    return 0;
}
```

Screenshots Output



Deskripsi :

Program ini membuat dan menampilkan isi Binary Search Tree (BST). Setelah dijalankan, program menampilkan "Hello world!", lalu memasukkan beberapa angka ke dalam tree. Hasil traversal inorder menunjukkan angka-angka tersusun rapi dari 1 sampai 7, yang menandakan proses insert berhasil dan tree bekerja sesuai aturan BST.

D. Kesimpulan

Modul ini membantu memahami bahwa struktur data tree—terutama Binary Search Tree—adalah cara yang rapi dan efisien untuk menyimpan data yang punya hubungan terurut. Dengan memahami cara kerja insert, search, delete, dan berbagai jenis traversal, kita bisa melihat bagaimana tree berubah bentuk sesuai operasi yang dilakukan. Rekursi juga terbukti sangat membantu karena banyak proses di tree bisa diselesaikan dengan pola pemanggilan fungsi berulang.

E. Referensi

Anita Sindar, R. M. S. (2019). *Struktur Data Dan Algoritma Dengan C++ (Vol. 1). CV. AA. RIZKY.*