

**LAPORAN PRAKTIKUM  
STRUKTUR DATA**

**MODUL X  
TREE**



**Disusun Oleh :**

NAMA : Ridha Akifah

NIM : 103112400132

**Dosen**

FAHRUDIN MUKTI WIBOWO

**PROGRAM STUDI STRUKTUR DATA  
FAKULTAS INFORMATIKA  
TELKOM UNIVERSITY PURWOKERTO  
2025**

## A. Dasar Teori

Rekursif merupakan mekanisme pemanggilan fungsi oleh dirinya sendiri untuk menyelesaikan masalah yang berpola, dengan syarat memiliki *base case* dan pemanggilan ulang selama kondisi khusus belum terpenuhi, namun penggunaannya sering kurang efisien karena membutuhkan memori dan waktu lebih banyak. Pada struktur data, tree merupakan struktur non-linear yang terdiri dari node yang saling terhubung tanpa sirkuit, memiliki root sebagai simpul awal, serta elemen lain yang tersusun dalam hubungan parent-child, dengan istilah penting seperti leaf, internal node, sibling, degree, dan height. Salah satu bentuk tree yang banyak digunakan adalah Binary Tree, khususnya Binary Search Tree (BST), yang memiliki aturan bahwa nilai di subtree kiri selalu lebih kecil dari parent, dan nilai di subtree kanan lebih besar, sehingga memungkinkan operasi penting seperti insert, search, update, delete, serta traversal (pre-order, in-order, dan post-order) untuk pengolahan data secara terstruktur dan efisien.

## 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* 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

```

tree.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;
```

```
    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; }

```

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 << "\nTreversal setelah insert" << endl;

cout << "Inorder      : "; tree.inorder();
cout << "Preeorder     : "; tree.inorder();
cout << "Postorder      : "; tree.inorder();


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


tree.update(20, 25);


cout << "stelah upd  dte (20->25):" <<endl;
cout << "Inorder : "; tree.inorder();


cout << "\n=== Delete DATA===" << endl;
    cout << "Sebelum delete (haps subtree dengan root = 30) : "
<< endl;
    cout << "Inorder : "; tree.inorder();


tree.deleteValue(30);


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


return 0;
}

```

Screenshoot Output

```

PS D:\C++\Assesment Struktur Data\Guided Modul 10> cd "d:\C++\Assesment Struktur Data\Guided Modul 10\" ; if ($?) { g++ main.cpp
-o main } ; if ($?) { .\main }
===INSERT DATA===
Data yang diinsert: 10, 15, 20, 30, 35, 40, 50

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

=== UPDATE DATA===
Sebelum update (20->25):
Inorder : 10 15 20 30 35 40 50
stelah upd dte (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
Stlh delete (subtree dengan root = 30) :
Inorder : 10 15 25 35 40 50
PS D:\C++\Assesment Struktur Data\Guided Modul 10>

```

Deskripsi:

Program ini menampilkan cara kerja Binary Search Tree (BST) dengan melakukan operasi insert, update, dan delete pada data. Setelah tiap operasi, program menampilkan hasil traversal inorder, preorder, dan postorder untuk menunjukkan perubahan struktur tree. Program ini menunjukkan bagaimana BST mengelola data secara terurut dan dinamis.

- C. 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;

```

```
address alokasi(infotype x);  
  
void insertNode(address &root, infotype x);  
  
address findNode(infotype x, address root);  
  
void inOrder(address root);  
  
#endif
```

bstree.cpp

```
#include "bstree.h"  
  
#include <iostream>  
  
using namespace std;  
  
address alokasi(infotype x) {  
    address p = new Node;  
    p->info = x;  
    p->left = NULL;  
    p->right = NULL;  
    return p;  
}  
  
void insertNode(address &root, infotype x) {  
    if (root == NULL) {  
        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 == NULL) return NULL;
    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 != NULL) {
        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 = NULL;

    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);

    return 0;

}

```

### Screenshoot Output

```

PS D:\C++\Unguided Modul 10> cd "d:\C++\Unguided Modul 10\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
Hello World!
1 - 2 - 3 - 4 - 5 - 6 - 7 -
PS D:\C++\Unguided Modul 10> 

```

### Deskripsi:

Program ini mengimplementasikan Binary Search Tree (BST) menggunakan linked list. Setiap node menyimpan data dan dua pointer (left dan right). Fungsi utama yang dibuat meliputi pembuatan node baru, menyisipkan data ke dalam BST sesuai aturan kiri-lebih kecil dan kanan-lebih besar, mencari node, serta menampilkan isi tree dengan traversal in-order.

### Unguided 2

bstree.h

```
int hitungJumlahNode(address root);

int hitungTotalInfo(address root);

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

bstree.cpp

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

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

int hitungKedalaman(address root, int start) {
    if (root == NULL) return start;

    int leftDepth = hitungKedalaman(root->left, start + 1);
    int rightDepth = hitungKedalaman(root->right, start + 1);

    return (leftDepth > rightDepth ? leftDepth : rightDepth);
}
```

Code main.cpp

*tidak ada perubahan pada code hanya ada penambahan*

```
cout<<"\n";

cout<<"kedalaman : "<< hitungKedalaman(root,0) << endl;

cout<<"jumlah Node : "<< hitungNode(root) << endl;

cout<<"total : "<< hitungTotal(root, 0) << endl;F
```

## Screenshots Output

```
hello World
1 - 2 - 3 - 4 - 5 - 6 - 7 -
kedalaman : 5
jumlah Node : 7
total : 28
```

## Deskripsi:

Program ini hampir sama dengan program sebelumnya hanya saja ada tambahan beberapa fungsi dalam code yang berguna untuk menghitung jumlah node, menjumlahkan total info pada node, dan menghitung kelaman maksimumnya.

## Unguided 3

bstree.h

```
void printPreOrder(address root);
void printPostOrder(address root);
```

bstree.cpp

```
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 << " - ";
    }
}
```

## Code main.cpp

*tidak ada perubahan pada code hanya ada penambahan*

```
cout << "\nPreOrder : ";  
  
printPreOrder(root);  
  
cout << "\nPostOrder: ";  
  
printPostOrder(root);  
  
return 0;
```

## Screenshoot Output

```
PS D:\C++\Unguided Modul 10> cd "d:\C++\Unguided Modul 10\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }  
hello World  
1 - 2 - 3 - 4 - 5 - 6 - 7 -  
kedalaman : 5  
jumlah Node : 7  
total : 28  
  
PreOrder : 1 - 2 - 6 - 4 - 3 - 5 - 7 -  
PostOrder: 3 - 5 - 4 - 7 - 6 - 2 - 1 -
```

## Deskripsi

Program ini serupa dengan sebelumnya, tetapi menambahkan fungsi traversal preorder dan postorder. Pada preorder, penelusuran dimulai dari root, kemudian ke kiri dan kanan, sedangkan postorder dimulai dari kiri, lalu kanan, dan terakhir root. Program ini membantu memahami berbagai cara traversal serta bagaimana struktur tree memengaruhi urutan data yang dihasilkan.

## D. Kesimpulan

Modul 10 menjelaskan konsep dasar rekursif serta penerapannya pada struktur data **Tree**, khususnya **Binary Tree** dan **Binary Search Tree (BST)**. Rekursif digunakan untuk mempermudah penyelesaian masalah yang berpola, termasuk dalam operasi-operasi pada tree seperti insert, search, delete, dan traversal. Tree sebagai struktur data non-linear memiliki berbagai terminologi penting seperti root, child, leaf, internal node, dan height. BST memungkinkan pengolahan data yang terurut melalui aturan bahwa subtree kiri selalu bernilai lebih kecil dan subtree kanan lebih besar. Melalui praktikum ini, mahasiswa memahami konsep, implementasi, dan penggunaan

operasi dasar pada BST menggunakan linked list, termasuk traversal preorder, inorder, dan postorder untuk menampilkan isi tree secara sistematis.

#### E. Referensi

[https://en.wikipedia.org/wiki/Operators\\_in\\_C\\_and\\_C%2B%2B](https://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B)

<https://learn.microsoft.com/id-id/cpp/cpp/references-cpp?view=msvc-170>

<https://medium.com/@mohamedeissabay/understanding-c-a-deep-dive-into-core-concepts-48a560679cdd>