

**LAPORAN PRAKTIKUM STRUKTUR
DATA**

**MODUL X (10)
TREE**



Disusun Oleh :

NAMA : Gien Darrel Adli

NIM : 10312430008

Dosen

FAHRUDIN MUKTI WIBOWO

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

Algoritma merupakan fondasi dalam pembuatan program komputer. Secara sederhana, algoritma adalah serangkaian langkah logis dan sistematis yang disusun untuk menyelesaikan suatu masalah. C++, berfungsi sebagai alat untuk mengimplementasikan algoritma tersebut agar dapat dimengerti dan dieksekusi oleh komputer. C++ sering digunakan sebagai bahasa pengantar untuk mempelajari konsep pemrograman dasar karena strukturnya yang terorganisir dan kemampuannya untuk menangani operasi tingkat rendah.

Untuk membangun logika dalam program sesuai dengan algoritma yang dirancang, C++ menyediakan struktur kontrol, yang terbagi menjadi dua jenis utama:

Struktur Percabangan (Conditional): Digunakan untuk pengambilan keputusan, di mana program akan menjalankan blok kode tertentu jika suatu kondisi terpenuhi. Struktur ini mencakup if-else untuk mengevaluasi kondisi boolean dan switch-case untuk memilih blok kode berdasarkan nilai dari sebuah variabel.

Struktur Perulangan (Looping): Digunakan untuk mengeksekusi blok kode yang sama secara berulang kali selama kondisi tertentu masih terpenuhi. C++ menyediakan tiga jenis perulangan utama: for, while, dan do-while, yang masing-masing memiliki karakteristik penggunaan yang spesifik dalam implementasi algoritma.

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

Guided 1 (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 *rightRotate(Node *y);
    Node *leftRotate(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

```

Deskripsi:

File `tree.h` ini berisi deklarasi struktur `Node` dan kelas `BinaryTree` untuk mengelola sebuah binary tree. Setiap node menyimpan data, pointer ke anak kiri dan kanan, serta tinggi node. Kelas `BinaryTree` menyediakan operasi dasar seperti insert, delete, update, dan traversal inorder, preorder, serta postorder, dengan detail logika pengelolaan pohon disembunyikan di bagian private.

Guided 2 (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::rightRotate(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::leftRotate(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 rightRotate(node);

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

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

    if (balance < -1 && value < node->right->data)
    {
        node->right = rightRotate(node->right);
        return leftRotate(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 rightRotate(root);

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

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

```

```

        return leftRotate(root);

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

```

Deskripsi:

File `tree.cpp` ini berisi implementasi kelas `BinaryTree` yang merealisasikan **AVL Tree**. Kode ini mengatur pembuatan pohon, proses penambahan dan penghapusan node dengan menjaga keseimbangan menggunakan rotasi kiri dan kanan, serta menghitung tinggi dan balance factor setiap node. Selain itu, disediakan fungsi update data dengan cara menghapus lalu menyisipkan ulang nilai baru, serta fungsi traversal inorder, preorder, dan postorder untuk menampilkan isi pohon.

Guided 3 (main.cpp)

```

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

using namespace std;

int main()
{
    BinaryTree tree;

    cout << "=== INSERT DATA ===\n"
        << 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\n"
    << 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 diupdate (20 -> 25): " << endl;
cout << "Inorder: ";
tree.inOrder();

tree.update(20, 25);

cout << "sesudah diupdate (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;
}

```

Deskripsi:

File `main.cpp` ini berfungsi sebagai program utama untuk menguji kelas `BinaryTree`. Program membuat sebuah objek tree, lalu melakukan operasi insert beberapa data dan menampilkan hasil traversal inorder, preorder, dan postorder. Setelah itu dilakukan update data dengan mengganti nilai tertentu, kemudian ditampilkan kembali hasilnya. Terakhir, program menghapus sebuah node dari tree dan menampilkan kondisi tree setelah proses penghapusan.

Screenshots Output :

```
=== UPDATE DATA ===
sebelum diupdate (20 -> 25):
Inorder: 10 15 20 30 35 40 50
sesudah diupdate (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
PS D:\darrel\kuliah\Semester3\Struktur Data\laprak>
```

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

bstree.h

```
#ifndef BSTREE_H
#define BSTREE_H

#include <iostream>
using namespace std;

typedef int infotype;
typedef struct Node *address;

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

#define Nil NULL
```

```

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

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

void printPreorder(address root);
void printPostorder(address root);

#endif

```

Deskripsi:

File bstree.h ini berisi deklarasi struktur dan fungsi untuk mengelola **Binary Search Tree (BST)**. Di dalamnya didefinisikan struktur Node yang menyimpan data serta pointer ke anak kiri dan kanan, dan disediakan fungsi untuk alokasi node, penyisipan dan pencarian data, traversal inorder, preorder, dan postorder, serta perhitungan jumlah node, total nilai data, dan kedalaman pohon.

bstree.cpp

```

#include "bstree.h"

address alokasi(infotype x)
{
    address p = new Node;
    if (p != Nil)
    {
        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);
    }
}

```

```

    {
        insertNode(root->right, x);
    }
}

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

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

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

```

```

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;
    }
}

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 << " - ";
    }
}

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

```

Deskripsi:

File bstree.cpp ini berisi implementasi fungsi-fungsi **Binary Search Tree (BST)**. Kode ini mengatur proses alokasi node, penyisipan data sesuai aturan BST, pencarian node

secara rekursif, penampilan data dengan traversal inorder, preorder, dan postorder, serta perhitungan jumlah node, total nilai seluruh data, dan kedalaman maksimum pohon.

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);

    cout << "Inorder Traversal: ";
    printInorder(root);
    cout << endl;

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

    cout << "\nPre-order : ";
```

```

printPreorder(root);

cout << endl;

cout << "Post-order : ";
printPostorder(root);
cout << endl;

return 0;
}

```

Deskripsi:

File main.cpp ini berfungsi sebagai program utama untuk menjalankan dan menguji Binary Search Tree. Program membuat sebuah tree kosong, lalu menyisipkan beberapa nilai secara manual. Setelah itu, program menampilkan traversal inorder, menghitung kedalaman tree, jumlah node, dan total nilai data, kemudian menampilkan hasil traversal preorder dan postorder.

Screenshots Output Unguided:

```

● Hello World
Inorder Traversal: 1 - 2 - 3 - 4 - 5 - 6 - 7 -
kedalaman : 5
jumlah Node : 7
total : 28

Pre-order : 1 - 2 - 6 - 4 - 3 - 5 - 7 -
Post-order : 3 - 5 - 4 - 7 - 6 - 2 - 1 -
PS D:\darrel\kuliah\Semester3\Struktur Data\laprak>

```

D. Kesimpulan

Kesimpulan modul 10 ini adalah mempelajari dan mengimplementasikan struktur data **Binary Tree dan Binary Search Tree** menggunakan bahasa C++. Pada modul ini dibuat file header untuk mendefinisikan struktur node dan deklarasi fungsi, file implementasi untuk merealisasikan operasi-operasi pada tree seperti insert, delete, update, pencarian, traversal inorder, preorder, dan postorder, serta perhitungan jumlah node, total nilai, dan kedalaman tree. Seluruh fungsi tersebut kemudian diuji melalui program utama sehingga konsep kerja tree, khususnya pengurutan data pada BST dan pengelolaan struktur pohon secara rekursif, dapat dipahami dan berjalan sesuai teori.

E. Referensi

Kaswar, A. B., & Zain, S. G. (2021). Mudah Belajar Pemrograman Dasar C++. Syiah Kuala University Press.

Hanief, S., Jepriana, I. W., & Kom, S. (2020). Konsep Algoritme dan Aplikasinya dalam Bahasa Pemrograman C++. Penerbit Andi.

Imamuddin, A., & Sobarnas, M. A. (2021). PEMBELAJARAN JARAK JAUH PEMROGRAMAN DASAR MENGGUNAKAN BAHASA C++ UNTUK UMUM: SEBUAH PROGRAM PENGABDIAN KEPADA MASYARAKAT. BEMAS: Jurnal Bermasyarakat, 1(2), 59-67.