LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL 10
TREE

Disusun Oleh :
NAMA : HERDIAN ABDILLAH PURNOMO
NIM : 103112430048

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Binary Search Tree (BST) adalah salah satu struktur data berbentuk pohon di mana setiap node hanya memiliki maksimal dua anak yaitu anak kiri dan anak kanan. BST memiliki aturan utama: nilai yang lebih kecil dari node ditempatkan di sisi kiri, sedangkan nilai yang lebih besar ditempatkan di sisi kanan. Dengan aturan ini pencarian data bisa dilakukan lebih cepat karena kita tidak perlu memeriksa semua data satu per satu, cukup mengikuti arah kiri atau kanan seperti menelusuri jalur. Selain pencarian BST juga memungkinkan proses seperti penambahan data, penghapusan data, dan penelusuran data dilakukan dengan lebih teratur. Tiga jenis traversal umum adalah in-order, pre-order, dan post-order. Dalam penggunaannya, BST juga dapat dihitung kedalamannya, jumlah node yang dimiliki, serta total nilai data yang ada di dalamnya.

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);
}
```

```cpp
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

```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.inorder();
    cout << "Posorder : "; tree.inorder();

    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

```
\MODUL 10\GUIDED\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile
=== INSERT DATA ====
Data yang diinsert: 10, 15, 20, 30, 35, 40, 50

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

=== 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
PS C:\Users\Lenovo\Documents\PRAKTIKUM STRUKDAT\FILE\MODUL 10\GUIDED>
```

Deskripsi:

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

SOAL 1

bstree.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);
```

```
#endif
```

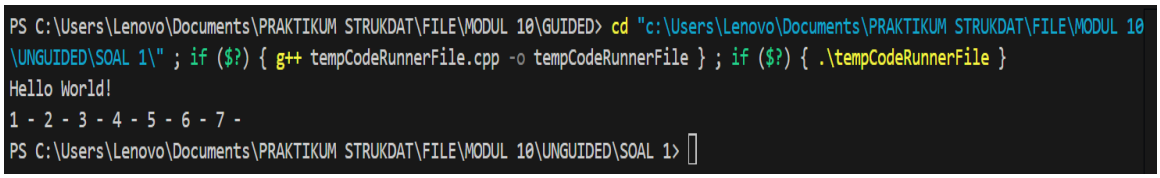
bs*tree.cpp*

```
#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

```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



SOAL 2

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);

int hitungJumlahNode(address root);

int hitungTotalInfo(address root);

int hitungKedalaman(address root);

#endif
```

b*stree.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;
    } else {
        return 1 + hitungJumlahNode(root->left) +
hitungJumlahNode(root->right);
    }
}

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

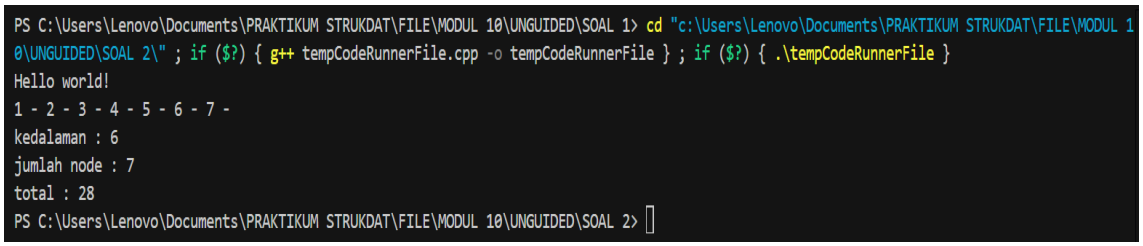
```
hitungTotalInfo(root->right);
    }
}

int hitungKedalaman(address root) {
    if (root == Nil) {
        return -1;
    } else {

        int leftDepth = hitungKedalaman(root->left);
        int rightDepth = hitungKedalaman(root->right);

        return 1 + max(leftDepth, rightDepth);
    }
}
```

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);
    cout << endl;

    int kedalaman = hitungKedalaman(root) + 2;
    cout << "kedalaman : " << kedalaman << endl;
```

```
    int jumlahNode = hitungJumlahNode(root);
    cout << "jumlah node : " << jumlahNode << endl;

    int totalInfo = hitungTotalInfo(root);
    cout << "total : " << totalInfo << endl;

    return 0;
}
```

Screenshots Output



SOAL 3

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);

int hitungJumlahNode(address root);

int hitungTotalInfo(address root);

int hitungKedalaman(address root);

void PreOrder(address root);

void PostOrder(address root);

#endif
```

bs*tree.cpp*

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

using namespace std;
```

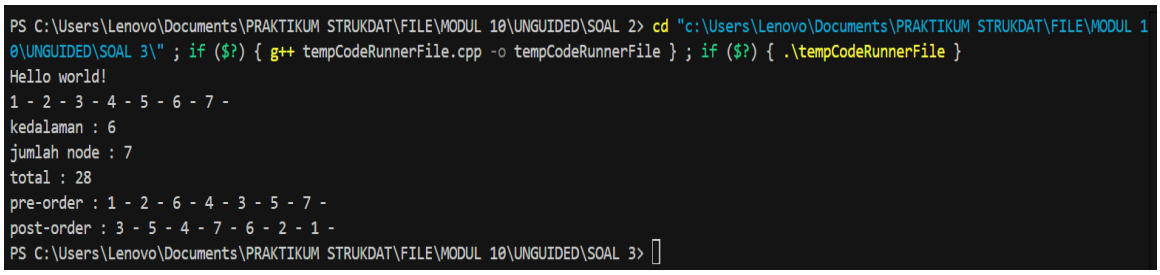
```
address alokasi(infotype x) {
    address P = new Node;
    P->info = x;
    P->left = Nil;
    P->right = Nil;
    return P;
}

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;
    } else {
        return 1 + hitungJumlahNode(root->left) +
hitungJumlahNode(root->right);
    }
}

int hitungTotalInfo(address root) {
```

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

int hitungKedalaman(address root) {
    if (root == Nil) {
        return -1;
    } else {

        int leftDepth = hitungKedalaman(root->left);
        int rightDepth = hitungKedalaman(root->right);


        return 1 + max(leftDepth, rightDepth);
    }
}

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 << " - ";
    }
}
```

main.cpp

```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);
    cout << endl;

    int kedalaman = hitungKedalaman(root) + 2;
    cout << "kedalaman : " << kedalaman << endl;

    int jumlahNode = hitungJumlahNode(root);
    cout << "jumlah node : " << jumlahNode << endl;

    int totalInfo = hitungTotalInfo(root);
    cout << "total : " << totalInfo << endl;

    cout << "pre-order : ";
    PreOrder(root);
    cout << endl;

    cout << "post-order : ";
    PostOrder(root);
    cout << endl;

    return 0;
}
```

Screenshots Output



Deskripsi Program utama ini dirancang untuk menguji fungsionalitas dasar dari struktur data Binary Search Tree (BST) yang diimplementasikan melalui Abstract Data Type (ADT). Program ini pertama-tama membangun BST dengan menyisipkan nilai-nilai 1, 2, 6, 4, 5, 3, dan 7 secara berurutan, memastikan properti BST (nilai kiri lebih kecil, nilai kanan lebih besar) tetap terjaga. Setelah struktur pohon terbentuk, program kemudian menampilkan isi pohon menggunakan tiga metode traversal utama: In-order, yang menghasilkan urutan nilai yang terurut; Pre-order, di mana root dikunjungi terlebih dahulu; dan Post-order, di mana root dikunjungi terakhir, sesuai dengan perintah Latihan 1 dan 3. Selanjutnya, program mengimplementasikan dan menguji fungsi rekursif yang diminta pada Latihan 2, yaitu hitungJumlahNode untuk mengetahui total simpul, hitungTotalInfo untuk menjumlahkan semua nilai data dalam simpul, dan hitungKedalaman untuk menentukan tinggi atau kedalaman maksimum dari BST yang telah dibuat. Keseluruhan program berfungsi sebagai validasi fungsionalitas BST dan penggunaan konsep rekursif dalam mengelola dan menganalisis struktur data non-linear.

E. Kesimpulan

Program Binary Search Tree ini menunjukkan bagaimana data dapat disimpan dan dikelola secara terstruktur menggunakan aturan penyusunan kiri-kanan berdasarkan nilai. Melalui proses penambahan data, penelusuran dengan berbagai metode, serta perhitungan kedalaman, jumlah node, dan total nilai, program ini memberikan gambaran bahwa BST mampu membantu pengolahan data menjadi lebih teratur dan mudah diakses. Dengan memahami cara kerja dasar pohon ini, kita dapat melihat bahwa BST adalah struktur data yang efisien untuk pencarian dan pengelolaan data dalam berbagai aplikasi.

F. Referensi

Raharjo, Budi. 2025. *Buku Pemrograman C++ Mudah dan Cepat Menjadi Master C.*

Wikipedia contributors. (2024, 8 Mei). C++. Wikipedia, Ensiklopedia Bebas. Diakses pada 2 Desember 2025, dari https://id.wikipedia.org/wiki/C%2B%2B