LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL 10
TREE

Disusun Oleh :

NAMA : Loh Suwargi Nitis Hamengku Bintang

NIM : 103112400116

Dosen

FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Binary tree adalah struktur data hierarkis di mana setiap simpul (node) memiliki maksimal dua anak (child), biasanya disebut kiri (left) dan kanan (right). Struktur ini memudahkan pemodelan hubungan bertingkat dan menyediakan kerangka dasar untuk berbagai operasi seperti pencarian, penyisipan, dan penghapusan yang dapat diimplementasikan secara rekursif. Konsep dasar dan mekanisme manipulasi node (penelusuran, penambahan, penghapusan) tetap menjadi titik awal penelitian dan pengembangan algorithmic karena sifatnya yang sederhana namun sangat fleksibel untuk berbagai variasi dan optimasi.

Salah satu varian paling penting adalah Binary Search Tree (BST), yang menambah aturan ordering: semua nilai di subtree kiri lebih kecil dari nilai root, dan semua nilai di subtree kanan lebih besar. Aturan ini memungkinkan operasi pencarian rata-rata berjalan dalam O(log n), tetapi pada kasus terburuk (tree sangat tidak seimbang) kompleksitas dapat turun menjadi O(n). Untuk mengatasi masalah ketidakseimbangan, dikembangkan pohon seimbang seperti AVL tree yang melakukan rotasi otomatis untuk menjaga perbedaan tinggi antar subtree kecil, sehingga menjamin kompleksitas operasi tetap logarithmik. Beberapa studi terbaru membandingkan performa AVL dengan variasi lain (mis. red-black) dan membahas trade-off desain untuk aplikasi nyata.

Binary tree tidak hanya digunakan untuk pengelolaan data numerik, tetapi juga memiliki peran dalam pemrosesan ekspresi matematika, penyimpanan struktur file, perancangan compiler, serta representasi struktur keputusan. Misalnya, expression tree digunakan untuk menyimpan ekspresi aritmatika yang dapat di-*evaluate* melalui traversal tertentu seperti inorder atau postorder.

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* 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);
}
```

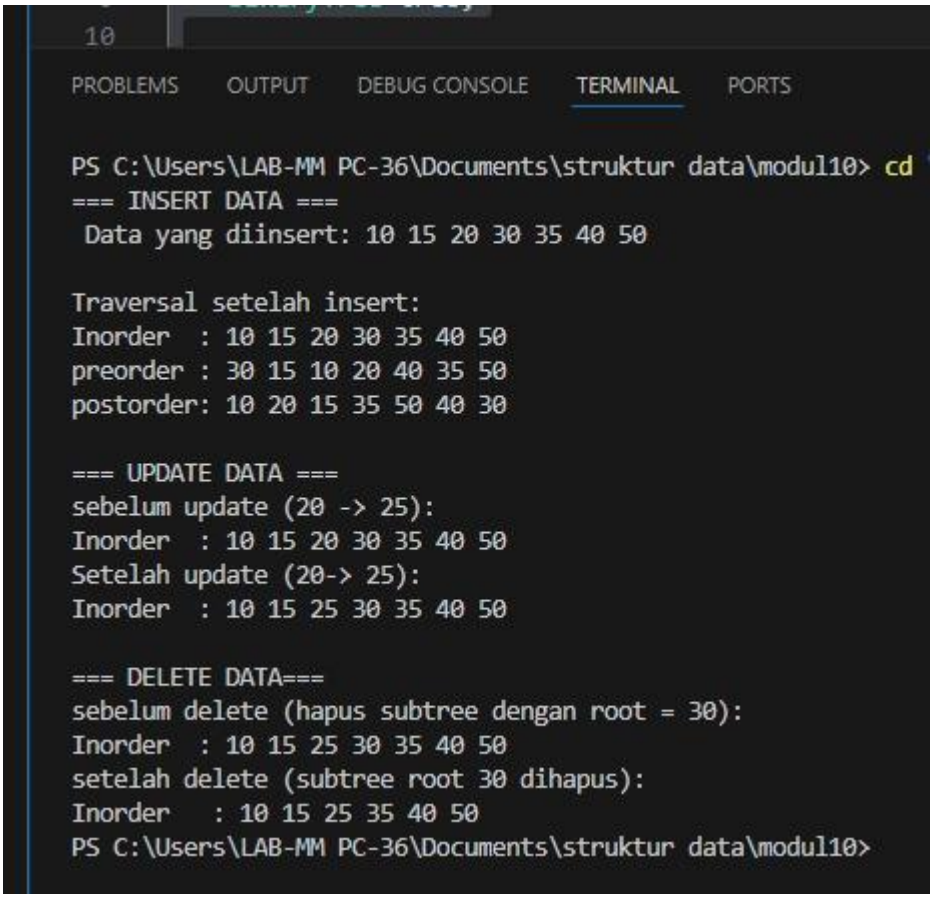
```
void BinaryTree::update(int oldVal, int newVal) {
    deleteValue(oldVal);
    insert(newVal);
}

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 << "\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



Deskripsi: Program ini, AVL Tree mendukung beberapa operasi utama: insert, delete, update, serta tiga jenis traversal yaitu inorder, preorder, dan postorder. Proses penyisipan dan penghapusan node dilakukan secara rekursif, kemudian diikuti oleh pengecekan faktor keseimbangan untuk memastikan tree tetap stabil. Ketika terjadi ketidakseimbangan, digunakan teknik rotasi seperti rotateLeft, rotateRight, serta kombinasi rotasi ganda untuk mengembalikan struktur tree menjadi seimbang. Selain itu, program juga menyediakan fungsi update, yaitu menghapus nilai lama dan menyisipkan nilai baru ke dalam tree.

Pada bagian main.cpp, program membuat sebuah objek BinaryTree dan melakukan serangkaian operasi seperti memasukkan data (10, 15, 20, 30, 35, 40, 50), menampilkan hasil traversal setelah insert, mengubah nilai 20 menjadi 25, serta menghapus nilai 30 beserta subtreenya. Setiap operasi menampilkan hasil traversal inorder, preorder, dan postorder sehingga perubahan struktur tree dapat terlihat secara jelas. Program ini bertujuan membantu mahasiswa memahami konsep AVL Tree, rotasi penyeimbang, serta penerapan traversal pada struktur data tree.

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

Unguided 1

Bstree.h

```
#ifndef BSTREE_H
```

```
#define BSTREE_H

typedef int infotype;

typedef struct Node *address;

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

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

#endif
```

Bstree.cpp

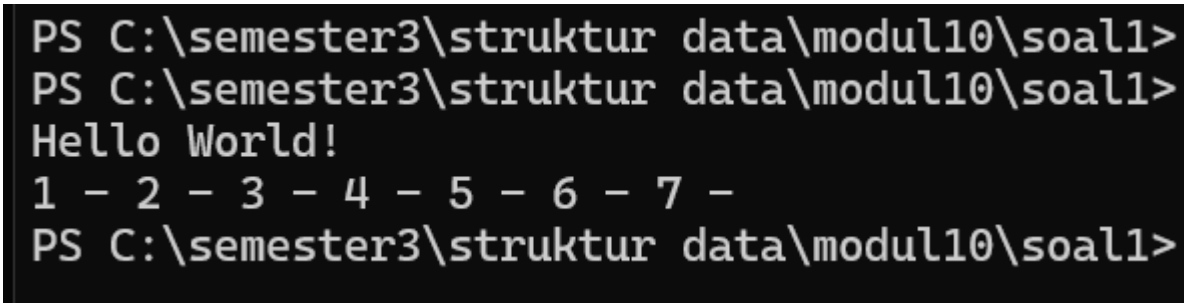
```
#include <iostream>
#include "bstree.h"
using namespace std;

address alokasi(infotype x) {
    address p = new Node;
    p->info = x;
    p->left = 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 printInOrder(address root) {
    if (root != NULL) {
        printInOrder(root->left);
        cout << root->info << " - ";
        printInOrder(root->right);
    }
}
```

Main.cpp

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

    printInOrder(root);

    return 0;
}
```

Screenshots Output



Deskripsi: Sebuah ADT Binary Search Tree yang diimplementasikan menggunakan pointer dan struktur Node. ADT ini dipisahkan ke dalam tiga file, yakni bstree.h, bstree.cpp, dan main.cpp, agar struktur program lebih rapi dan sesuai standar pemrograman modular. Fungsi dasar seperti alokasi node, penyisipan data sesuai aturan BST, pencarian nilai tertentu, dan pencetakan data secara inorder menjadi komponen utama yang harus dibuat. Setelah ADT selesai, program utama digunakan untuk menguji tree dengan memasukkan beberapa nilai dan menampilkan hasil traversal inorder yang menunjukkan struktur BST yang benar.

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

Unguided 2

Bstree.h

```
#ifndef BSTREE_H
#define BSTREE_H

typedef int infotype;
typedef struct Node *address;

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

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);
```

```
#endif
```

Bstree.cpp

```
#include <iostream>
#include "bstree.h"
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 printInOrder(address root) {
    if (root != NULL) {
        printInOrder(root->left);
        cout << root->info << " - ";
        printInOrder(root->right);
    }
}

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

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

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

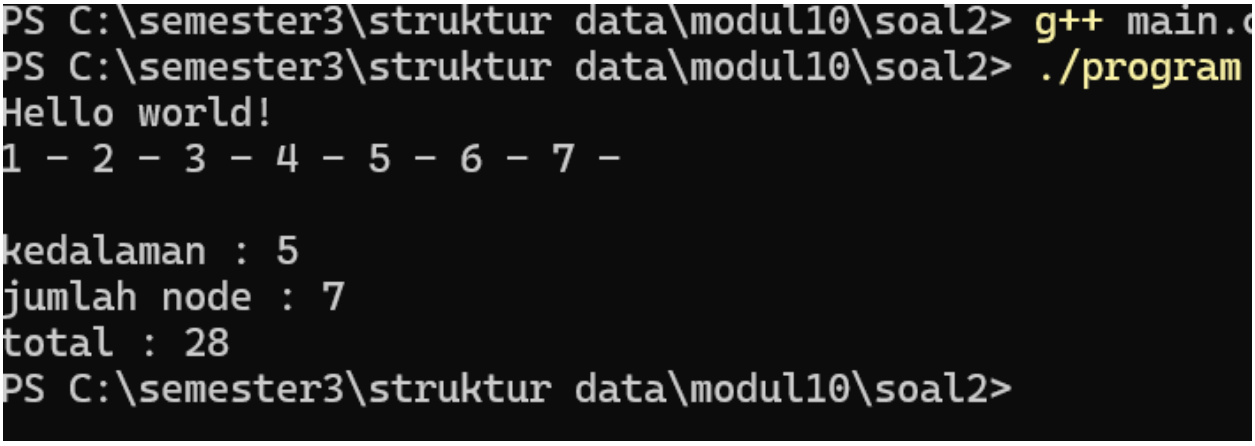
```
    int leftDepth = hitungKedalaman(root->left, start + 1);
    int rightDepth = hitungKedalaman(root->right, start + 1);

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

Main.cpp

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

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);

    printInOrder(root);
    cout << endl << endl;

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

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

    return 0;
}
```

Screenshots Output



Deskripsi: Melanjutkan pengembangan ADT dengan menambahkan fungsi-fungsi yang dapat menghitung berbagai aspek dalam sebuah Binary Search Tree. Tiga fungsi tersebut adalah perhitungan jumlah total node, penjumlahan nilai info dari semua node, dan pengukuran kedalaman tree. Ketiga proses ini dilakukan secara rekursif agar traversal terhadap seluruh node dapat dilakukan dengan benar. Bagian ini dirancang untuk melatih mahasiswa dalam melakukan analisis struktur tree dan memahami bagaimana setiap node terhubung satu sama lain dalam sebuah hierarki.

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

Unguided 3

Bstree.h

```
#ifndef BSTREE_H
#define BSTREE_H


typedef int infotype;
typedef struct Node *address;
```

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

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

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

#endif
```

Bstree.cpp

```
#include <iostream>
#include "bstree.h"
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 printInOrder(address root) {
    if (root != NULL) {
        printInOrder(root->left);
        cout << root->info << " - ";
        printInOrder(root->right);
    }
}

void printPreOrder(address root) {
```

```
    if (root != NULL) {
        cout << root->info << " - ";
        printPreOrder(root->left);
        printPreOrder(root->right);
    }
}

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

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

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

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

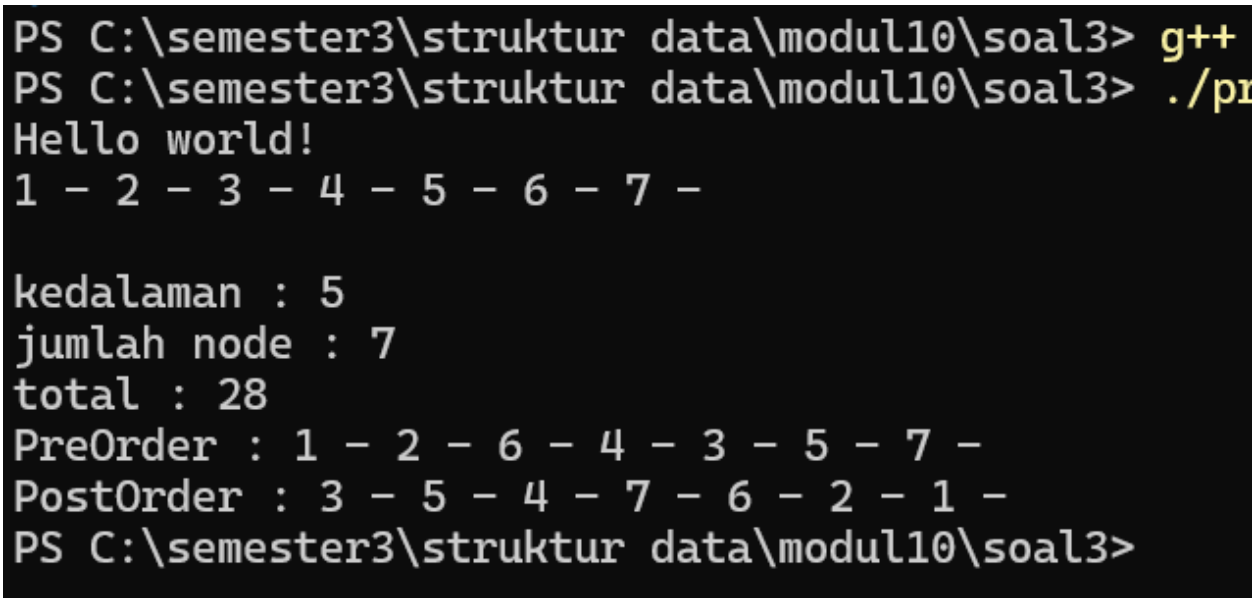
```
    if (root == NULL)
        return start;

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

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

Main.cpp

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

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);

    printInOrder(root);
```

```
    cout << endl << endl;

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

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

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

    return 0;
}
```

Screenshots Output



Deskripsi: Mengimplementasikan traversal preorder dan postorder sebagai tambahan dari traversal inorder yang telah dibuat sebelumnya. Traversal preorder dimulai dari root kemudian bergerak ke subtree kiri dan kanan, sedangkan postorder menelusuri subtree kiri dan kanan terlebih dahulu sebelum mencetak node root. Implementasi kedua traversal ini dilakukan secara rekursif karena tree bersifat bercabang dan membutuhkan pemrosesan yang sistematis. Melalui tugas ini, mahasiswa dapat memahami bagaimana perbedaan pola traversal menghasilkan output yang berbeda dari tree yang sama.

F. Kesimpulan

Secara keseluruhan, binary tree dan turunannya terbukti tidak hanya penting secara konseptual tetapi juga aplikatif dalam berbagai bidang, mulai dari pemrosesan ekspresi, pengindeksan basis data, pengolahan citra, hingga optimasi sistem komputasi modern. Konsistensi penggunaan dan pengembangan struktur ini dalam publikasi ilmiah terkini menunjukkan bahwa binary tree tetap menjadi elemen kunci dalam rekayasa perangkat lunak, ilmu komputer, dan algoritma masa kini. Jika dikembangkan dan diimplementasikan dengan tepat, struktur data ini mampu memberikan efisiensi tinggi sekaligus kestabilan dalam pengelolaan data berskala besar.

G. Referensi

Lorenzen, A., Leijen, D., Swierstra, W., & Lindley, S. (2024). The functional essence of imperative binary search trees. ***Proceedings of the ACM on Programming Languages*****,** ***8*****(PLDI), 518-542.**

Njoum, M., Sulaiman, R., Shukur, Z., & Qamar, F. (2024). High-Secured Image LSB Steganography Using AVL-Tree with Random RGB Channel Substitution. ***Computers, Materials & Continua*****,** ***81*****(1).**

Bhaduri, D., Toth, D., & Holan, S. H. (2025). A Review of Tree-Based Methods for Analyzing Survey Data. ***Wiley Interdisciplinary Reviews: Computational Statistics*****,** ***17*****(1), e70010.**