LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL X
PENGENALAN CODE BLOCKS

Disusun Oleh :
NAMA : Sinta Sintiani
NIM : 103112430047

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Tree adalah struktur data non-linear yang digunakan untuk merepresentasikan hubungan hierarki dan terdiri dari node yang saling terhubung melalui edge, dengan satu node utama yang disebut root. Setiap node dapat memiliki child, sedangkan node tanpa child disebut leaf. Dalam pemrograman C++, tree sering digunakan untuk proses pencarian, pengurutan, dan pengelolaan data terstruktur, terutama melalui Binary Search Tree (BST) yang menetapkan aturan data kiri lebih kecil dari parent dan data kanan lebih besar. Operasi umum pada tree meliputi insert, search, delete, dan traversal seperti preorder, inorder, dan postorder. Struktur ini memberikan efisiensi tinggi saat terorganisasi dengan baik, meskipun kinerjanya dapat menurun jika tree tidak seimbang.

Guided 1

tree.h

```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

```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 D:\SEM 3\STRUKTUR DATA\mod 10> cd "d:\SEM 3\STRUKTUR DATA\Modul 10\" ; if ($?)
}
=== 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
```

Deskripsi:

Program tersebut merupakan implementasi struktur data AVL Tree dalam bahasa C++, yaitu bentuk khusus dari Binary Search Tree yang mampu menjaga keseimbangan tinggi pohon secara otomatis melalui operasi rotasi kiri dan kanan. Program ini menyediakan fitur insert, delete, dan update untuk mengelola data pada tree, serta mendukung traversal inorder, preorder, dan postorder untuk menampilkan isi tree. Selama proses penambahan maupun penghapusan node, program secara otomatis menghitung height dan balance factor setiap node, kemudian melakukan rotasi jika diperlukan agar pohon tetap seimbang. Dengan demikian, program tersebut mampu memastikan pencarian dan pengolahan data tetap berjalan efisien meskipun data dimasukkan secara acak.

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 printInOrder(address root);

#endif
```

bstree.cpp

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

void createStack(Stack &S) {
    S.top = -1;
}

void push(Stack &S, infotype x) {
    if (S.top < MAX - 1) {
        S.top++;
        S.info[S.top] = x;
    } else {
        cout << "Stack penuh!" << endl;
    }
}

infotype pop(Stack &S) {
    if (S.top >= 0) {
        infotype x = S.info[S.top];
        S.top--;
        return x;
    } else {
        cout << "Stack kosong!" << endl;
        return -1;
    }
```

```
}

void printInfo(Stack S) {
    cout << "[TOP] ";
    for (int i = S.top; i >= 0; i--) {
        cout << S.info[i] << " ";
    }
    cout << endl;
}

void balikStack(Stack &S) {
    Stack temp;
    createStack(temp);
    while (S.top >= 0) {
        push(temp, pop(S));
    }
    S = temp;
} #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;
    } else 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"
#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);

    printInOrder(root);
    return 0;
}
```

Screenshots Output

```
PS D:\SEM 3\STRUKTUR DATA\Modul 10> cd "d:\SEM 3\STRUKTUR DATA\mod 10\" ;
Hello World!
1 2 3 4 5 6 7
```

Deskripsi:

Program di atas merupakan implementasi struktur data stack menggunakan array statis dalam bahasa C++. Stack didefinisikan dengan kapasitas maksimum (MAX = 20) dan menggunakan variabel top untuk menandai elemen paling atas. Fungsi createStack digunakan untuk menginisialisasi stack agar kosong, push menambahkan elemen ke atas stack, dan pop menghapus serta mengembalikan elemen teratas. Fungsi printInfo menampilkan seluruh isi stack dari elemen paling atas ke bawah, sedangkan balikStack berfungsi untuk membalik urutan elemen stack menggunakan stack sementara. Program utama melakukan serangkaian operasi push dan pop, menampilkan isi stack, lalu membalik dan menampilkan hasilnya kembali.

Unguided 2

bstree.h

```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 printInOrder(address root);

int hitungJumlahNode(address root);

int hitungTotalInfo(address root, int start);

int hitungKedalaman(address root, int start);

#endif
```

bstree.cpp

```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 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 0;
    return root->info +
           hitungTotalInfo(root->left, start) +
           hitungTotalInfo(root->right, start);
}

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"
#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);

    printInOrder(root);
    cout << "\n";

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

    return 0;
```

```
}
```

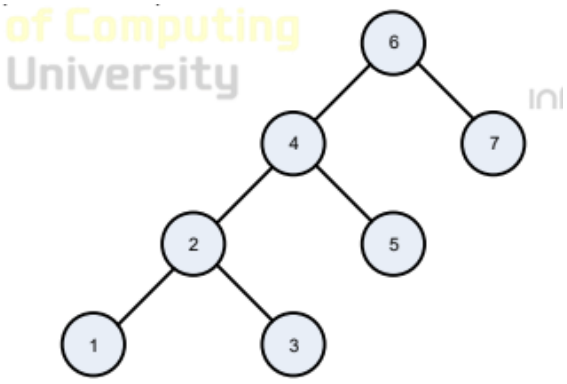
Screenshots Output

```
PS D:\SEM 3\STRUKTUR DATA\mod 10> cd "d:\SEM 3\STRUKTUR DATA\mod 10\" ;
Hello world!
1 - 2 - 3 - 4 - 5 - 6 - 7 -
kedalaman : 5
jumlah node : 7
total : 28
```

Deskripsi:

Program di atas terdiri dari file header, fungsi-fungsi BST, dan program utama. Program memungkinkan pembuatan node, penyisipan data, pencarian, serta penelusuran inorder untuk menampilkan data secara terurut. Selain itu, disediakan fungsi untuk menghitung jumlah node, total nilai info, dan kedalaman tree. Pada main.cpp, program membangun BST dari sejumlah nilai dan menampilkan hasil traversal beserta informasi kedalaman, jumlah node, dan total nilai pada tree.

No 3



Gambar 10-18 Ilustrasi *Tree*

Pre-order

6-4-2-1-3-5-7

Post order

1-3-2-5-4-7

B. Kesimpulan

Praktikum ini menunjukkan bahwa traversal pada struktur tree memiliki pola kunjungan simpul yang berbeda sesuai metode yang digunakan. Melalui contoh tree yang diberikan,

metode pre-order melakukan kunjungan dimulai dari root kemudian ke subtree kiri dan kanan, sedangkan metode post-order mengunjungi subtree terlebih dahulu sebelum kembali ke root. Dengan menerapkan kedua teknik traversal ini, mahasiswa dapat memahami cara kerja navigasi data dalam struktur tree serta bagaimana urutan simpul dipengaruhi oleh bentuk dan kedalaman tree. Praktikum ini membantu memperkuat konsep dasar dalam struktur data khususnya binary tree dan proses traversal.

C. Referensi

https://www.geeksforgeeks.org/dsa/introduction-to-tree-data-structure/

https://www.programiz.com/dsa/trees