LAPORAN PRAKTIKUM

MODUL 9

TREE (BAGIAN PERTAMA)

Nama :

Candra Dinata (2311104061)

Kelas :

S1SE 07 02

Dosen :

Wahyu Andi Saputra

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. TUJUAN

Tujuan pembelajaran dari praktikum pemrograman struktur data tree dalam C++ adalah untuk memahami konsep dasar, implementasi, dan aplikasi dari struktur data tree. Mahasiswa diharapkan mampu mengenali jenis-jenis tree seperti binary tree, binary search tree (BST), dan tree lainnya, serta memahami cara kerja operasi dasar seperti traversal (in-order, pre-order, post-order), pencarian, penambahan, dan penghapusan node pada tree. Selain itu, praktikum ini bertujuan untuk melatih keterampilan mahasiswa dalam mengimplementasikan tree menggunakan pointer dan rekursi, sehingga dapat diaplikasikan untuk memecahkan berbagai permasalahan komputasi seperti pengelolaan data hierarkis, pencarian yang efisien, dan optimasi algoritma. Dengan memahami materi ini, mahasiswa diharapkan dapat mengembangkan kemampuan logika pemrograman yang lebih kompleks dan mengintegrasikan struktur data tree ke dalam program berbasis C++.

II. DASAR TEORI

Struktur data tree adalah salah satu struktur data hierarkis yang digunakan untuk merepresentasikan hubungan antar elemen dalam bentuk node yang dihubungkan oleh edge, menyerupai struktur pohon dengan akar (root) sebagai titik awal. Tree terdiri dari node yang memiliki satu parent (kecuali root) dan dapat memiliki sejumlah child. Salah satu jenis tree yang umum adalah binary tree, di mana setiap node maksimal memiliki dua child. Binary Search Tree (BST) adalah variasi binary tree yang

menerapkan aturan khusus: nilai pada node di sebelah kiri lebih kecil dari node induk, sedangkan nilai di sebelah kanan lebih besar. Operasi pada tree, seperti traversal (pre-order, in-order, dan post-order), pencarian, penambahan, dan penghapusan, sering dilakukan menggunakan pendekatan rekursif. Tree digunakan dalam berbagai aplikasi seperti struktur file, representasi ekspresi matematika, pengambilan keputusan, dan algoritma pencarian cepat karena efisiensinya dalam mengorganisasi data secara hierarkis dan terstruktur.

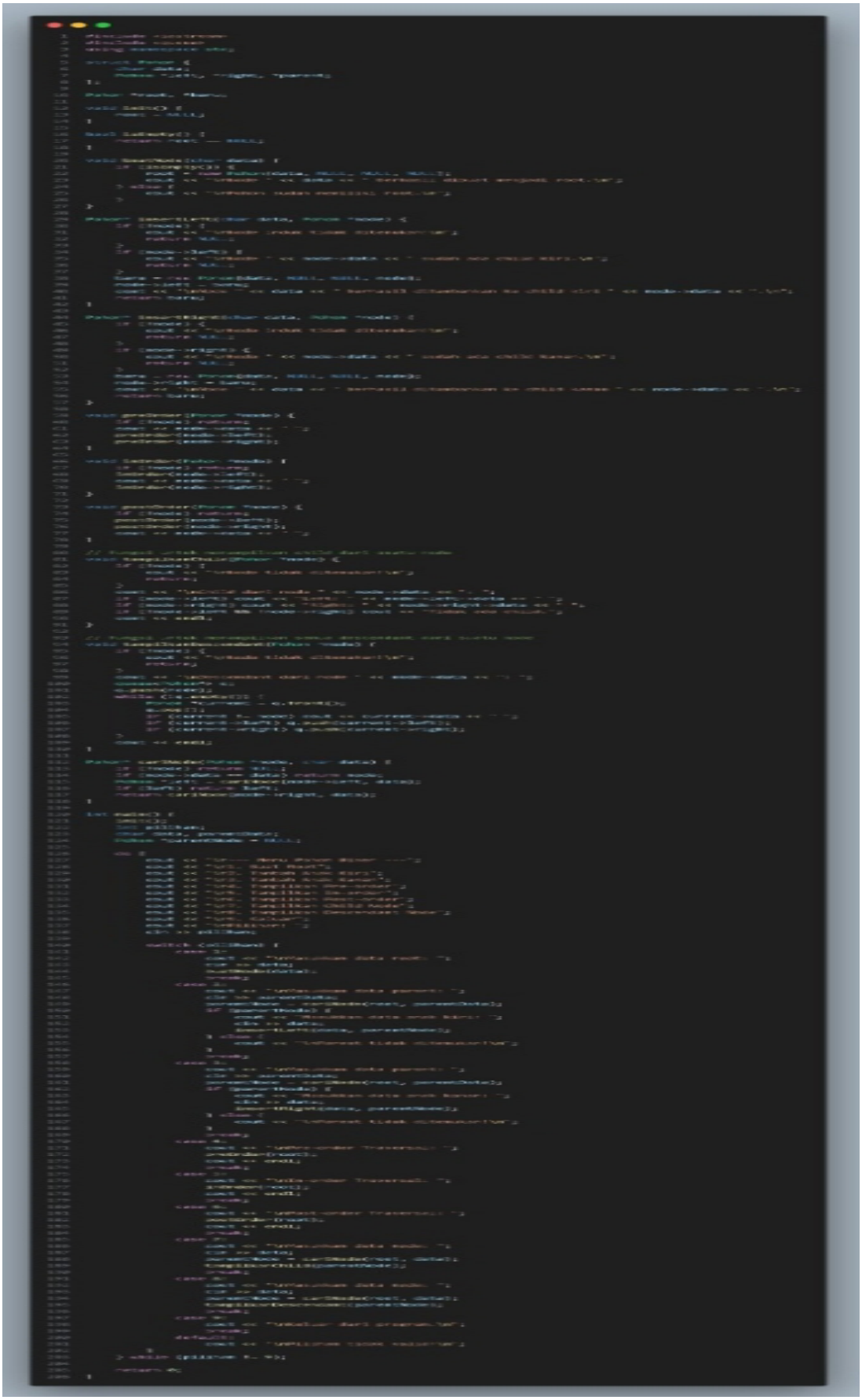
III. GUIDED

1.

Kode ini adalah implementasi dari pohon biner dalam bahasa C++, di mana setiap node pohon menyimpan data bertipe char dan memiliki pointer ke anak kiri, anak kanan, serta induk. Program ini dimulai dengan mendeklarasikan struktur data Pohon yang memiliki atribut data, pointer ke anak kiri dan kanan, serta pointer ke induk. Fungsi init() digunakan untuk menginisialisasi pohon agar kosong, sementara isEmpty() memeriksa apakah pohon kosong. Program memungkinkan pembuatan pohon dengan fungsi buatNode(), serta menambahkan anak kiri dan kanan menggunakan insertLeft() dan insertRight(). Fungsi update() memungkinkan perubahan data pada node tertentu, dan find() digunakan untuk mencari node dengan data tertentu. Traversal pohon dapat dilakukan menggunakan metode pre-order, in-order, dan post-order yang diimplementasikan dalam fungsi preOrder(), inOrder(), dan postOrder(). Selain itu, program ini juga menyediakan fungsi untuk menghapus node (deleteNode()), serta mencari node paling kiri dan kanan menggunakan fungsi mostLeft() dan mostRight(). Program ini mengimplementasikan berbagai operasi dasar pada pohon biner dan mendemonstrasikan penggunaannya dalam fungsi main(), yang mencakup pembuatan pohon, penambahan node, traversal, dan penghapusan node.

IV. UNGUIDED

1.



Kode ini merupakan implementasi pohon biner dalam bahasa C++ yang memungkinkan pengguna untuk melakukan berbagai operasi dasar pada pohon, seperti pembuatan node, penambahan anak kiri dan kanan, serta traversal pohon dengan metode pre-order, in-order, dan post-order. Struktur data Pohon memiliki empat atribut: data untuk menyimpan informasi karakter, serta pointer left, right, dan parent yang menghubungkan node dengan anak kiri, anak kanan, dan induknya. Program dimulai dengan fungsi init() yang menginisialisasi pohon sebagai kosong dan menyediakan fungsi isEmpty() untuk memeriksa apakah pohon kosong. Fungsi buatNode() digunakan untuk membuat root pohon, sementara fungsi insertLeft() dan insertRight() memungkinkan penambahan anak kiri dan kanan pada node tertentu. Program juga menyediakan fungsi untuk menampilkan traversal pohon (preOrder(), inOrder(), dan postOrder()) serta fungsi tambahan untuk menampilkan child dan descendant dari suatu node. Fungsi cariNode() digunakan untuk mencari node berdasarkan data tertentu.

Pada bagian utama program, terdapat menu yang memungkinkan pengguna untuk memilih berbagai operasi yang ingin dilakukan pada pohon biner. Pengguna dapat membuat root, menambahkan anak kiri dan kanan dengan memilih parent node, menampilkan traversal pohon, serta menampilkan child atau descendant dari suatu node. Fungsi tampilkanChild() menampilkan anak kiri dan kanan dari node tertentu, sedangkan tampilkanDescendant() menampilkan semua keturunan (descendants) dari node yang dipilih dengan menggunakan traversal breadth-first (queue). Program ini mengimplementasikan fitur pencarian node (cariNode()) untuk memastikan bahwa operasi penambahan anak atau tampilan informasi hanya dilakukan pada node yang valid. Secara keseluruhan, program ini memberikan antarmuka interaktif untuk mengelola pohon biner, termasuk menampilkan hasil traversal dan informasi terkait struktur pohon.

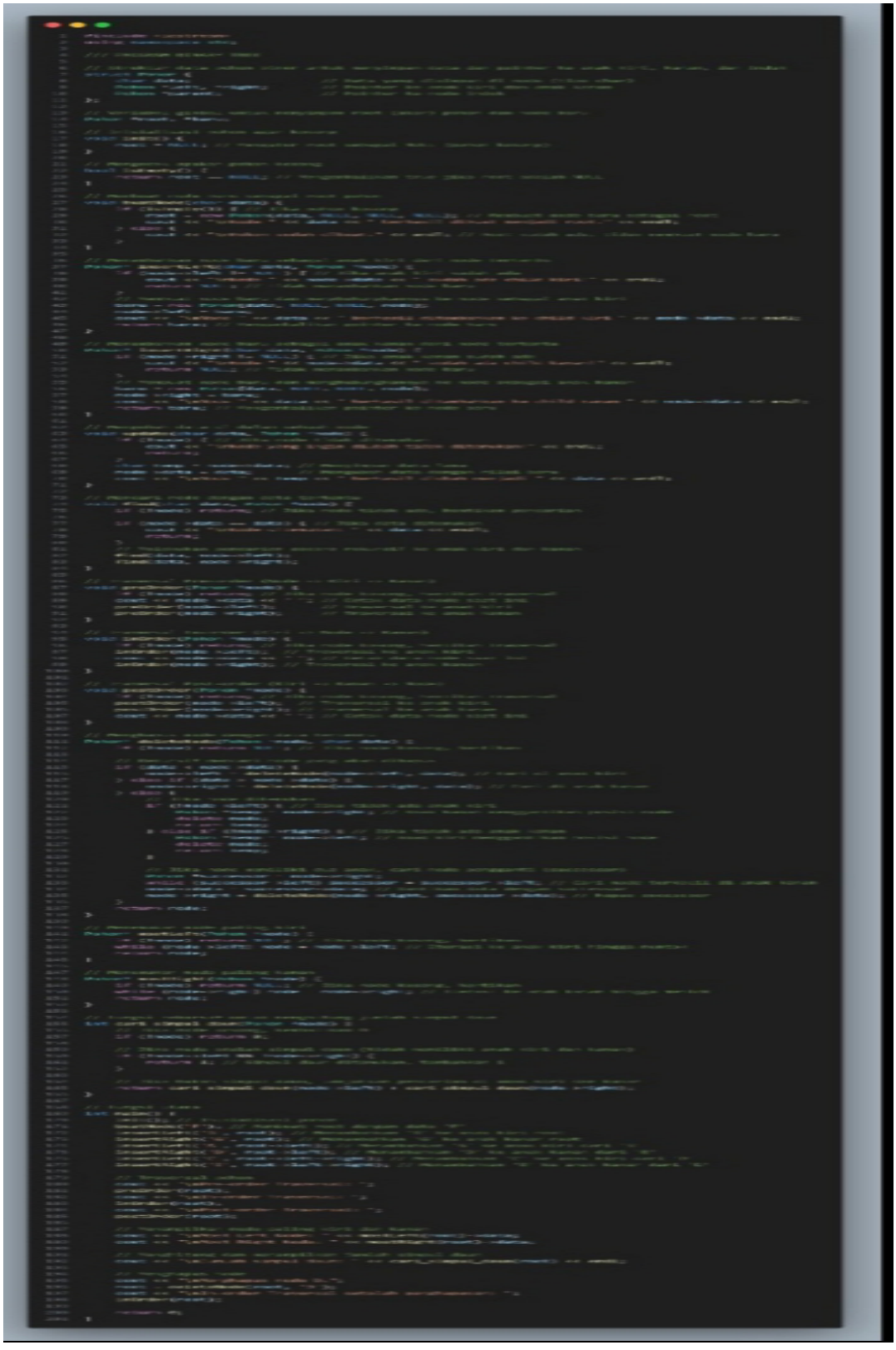
2.

```cpp
#include <iostream>
#include <climits>
using namespace std;

/// Struktur pohon biner
struct Pohon {
    int data;         // Data node
    Pohon *left;      // Anak kiri
    Pohon *right;     // Anak kanan
};

// Deklarasi root pohon
Pohon *root = NULL;

// Fungsi untuk membuat node baru
Pohon* buatNode(int data) {
    Pohon* nodeBaru = new Pohon();
    nodeBaru->data = data;
    nodeBaru->left = NULL;
    nodeBaru->right = NULL;
    return nodeBaru;
}

// Menambahkan node baru ke kiri
Pohon* insertLeft(Pohon *node, int data) {
    if (!node) return NULL; // Jika node kosong, hentikan
    if (!node->left) {
        node->left = buatNode(data); // Tambahkan anak kiri
        cout << "Node " << data << " berhasil ditambahkan ke kiri " << node->data << endl;
    }
    return node->left;
}

// Menambahkan node baru ke kanan
Pohon* insertRight(Pohon *node, int data) {
    if (!node) return NULL; // Jika node kosong, hentikan
    if (!node->right) {
        node->right = buatNode(data); // Tambahkan anak kanan
        cout << "Node " << data << " berhasil ditambahkan ke kanan " << node->data << endl;
    }
    return node->right;
}

// Fungsi rekursif untuk memeriksa apakah pohon adalah BST
bool is_valid_bst(Pohon *node, int min_val, int max_val) {
    if (!node) return true; // Pohon kosong dianggap valid
    // Jika data node melanggar batas, kembalikan false
    if (node->data <= min_val || node->data >= max_val) return false;
    // Periksa anak kiri dan kanan dengan batas nilai diperbarui
    return is_valid_bst(node->left, min_val, node->data) &&
           is_valid_bst(node->right, node->data, max_val);
}

// Fungsi utama
int main() {
    // Membuat pohon valid sebagai BST
    root = buatNode(10);
    Pohon *node5 = insertLeft(root, 5);
    Pohon *node20 = insertRight(root, 20);
    insertLeft(node5, 1);
    insertRight(node5, 8);
    insertLeft(node20, 15);
    insertRight(node20, 25);

    // Uji pohon valid sebagai BST
    cout << "\nApakah pohon valid sebagai BST? "
         << (is_valid_bst(root, INT_MIN, INT_MAX) ? "Ya" : "Tidak") << endl;

    // Membuat pohon tidak valid sebagai BST
    root = buatNode(10);
    insertLeft(root, 20); // Nilai salah untuk BST
    insertRight(root, 5);

    // Uji pohon tidak valid sebagai BST
    cout << "\nApakah pohon valid sebagai BST? "
         << (is_valid_bst(root, INT_MIN, INT_MAX) ? "Ya" : "Tidak") << endl;

    return 0;
}
```

Kode ini adalah implementasi pohon biner yang menguji apakah pohon yang dibangun valid sebagai Binary Search Tree (BST) di C++. Struktur Pohon digunakan untuk menyimpan data integer dan dua pointer yang menunjuk ke anak kiri dan kanan node. Fungsi buatNode() digunakan untuk membuat node baru dengan data yang diberikan, sementara insertLeft() dan insertRight() memungkinkan penambahan node sebagai anak kiri atau kanan dari node tertentu. Fungsi is_valid_bst() adalah fungsi rekursif yang memeriksa apakah pohon memenuhi aturan BST, yaitu setiap node di kiri harus memiliki nilai lebih kecil dari node induknya, dan setiap node di kanan harus memiliki nilai lebih besar. Fungsi ini memeriksa batasan nilai menggunakan parameter min_val dan max_val yang diperbarui saat traversal ke anak kiri atau kanan. Dalam main(), program pertama-tama membangun pohon yang valid sebagai BST, kemudian menguji

kevalidannya menggunakan is_valid_bst(). Setelah itu, pohon yang tidak valid (di mana anak kiri dan kanan melanggar aturan BST) juga dibuat dan diuji untuk memverifikasi bahwa pohon tersebut tidak valid sebagai BST. Hasil validasi ditampilkan menggunakan pesan "Ya" atau "Tidak".

3.



Kode ini merupakan implementasi pohon biner di mana setiap node menyimpan data bertipe char dan memiliki tiga pointer: satu untuk anak kiri, satu untuk anak kanan, dan satu untuk induk. Pohon diinisialisasi dengan root yang bernilai NULL hingga node pertama dibuat. Fungsi-fungsi utama dalam kode ini mencakup pembuatan node, penyisipan anak kiri dan kanan, serta pencarian dan pembaruan node tertentu. Ada beberapa fungsi traversal yang digunakan untuk menelusuri pohon, yakni Pre-order, In-order, dan Post-order, yang masing-masing memproses data node dalam urutan yang berbeda (Node -> Kiri -> Kanan, Kiri -> Node -> Kanan, Kiri -> Kanan -> Node). Selain itu, terdapat fungsi deleteNode untuk menghapus node tertentu dalam pohon dan fungsi untuk mencari node paling kiri dan kanan dari pohon.

Selain fungsi dasar pohon biner, kode ini juga mencakup fitur tambahan seperti menghitung jumlah simpul daun (node tanpa anak), yang dihitung dengan rekursi pada anak kiri dan kanan dari setiap node. Di dalam main(), pohon dibangun dengan menambahkan beberapa node berturut-turut menggunakan fungsi insertLeft dan insertRight. Setelah pohon terbentuk, program menampilkan hasil traversal pohon menggunakan tiga metode yang berbeda, mencari dan menampilkan node paling kiri dan kanan, serta menghitung dan menampilkan jumlah simpul daun. Terakhir, program juga menghapus sebuah node ('D') dari pohon dan menampilkan hasil traversal pohon setelah penghapusan tersebut.

V. KESIMPULAN

Kesimpulan dari praktikum tentang pohon biner dalam C++ ini adalah bahwa struktur data pohon biner merupakan salah satu cara yang efisien untuk menyimpan data dalam bentuk hierarkis. Melalui praktikum ini, kita telah mempelajari cara mendefinisikan dan mengimplementasikan pohon biner menggunakan struktur data yang menyimpan informasi berupa node dengan pointer ke anak kiri, anak kanan, dan induk. Fungsi-fungsi dasar seperti penyisipan node, pencarian, dan traversal pohon (Pre-order, In-order, dan Post-order) sangat penting untuk memahami cara kerja pohon biner dalam berbagai aplikasi. Selain itu, praktikum ini juga menunjukkan cara-cara untuk memodifikasi pohon, seperti mengubah data node, menghapus node, dan menghitung jumlah simpul daun.

Dengan menerapkan konsep-konsep dasar tersebut, praktikum ini memberikan pemahaman yang lebih baik mengenai struktur data pohon biner serta penerapannya dalam berbagai algoritma yang melibatkan pengelolaan data berbentuk hierarkis. Implementasi pohon biner di C++ membantu memperjelas bagaimana data dapat diorganisasikan dan diakses dengan cara yang terstruktur, serta memperkenalkan kita pada operasi-operasi penting seperti pencarian, penyisipan, dan penghapusan data dalam pohon. Pemahaman ini sangat bermanfaat dalam pengembangan aplikasi yang membutuhkan struktur data hierarkis, seperti sistem manajemen basis data dan algoritma pencarian.