Aturan Praktikum Struktur Data

1. **Akun GitHub:** Setiap praktikan wajib memiliki akun GitHub yang aktif dan digunakan selama praktikum berlangsung.
2. **Invite Collaborator:** Setiap praktikan diwajibkan untuk menambahkan collaborator di setiap repository
    a. Asisten Praktikum: AndiniNH
    b. Asisten Praktikum : 4ldiputra
3. **Repository Praktikum:** Setiap praktikan diwajibkan untuk membuat satu repository di GitHub yang akan digunakan untuk seluruh tugas dan laporan praktikum. Repository ini harus diatur dengan rapi dan sesuai dengan instruksi yang akan diberikan di lampiran.
4. **Penamaan Folder:** Penamaan folder dalam repository akan dibahas secara rinci di lampiran. Praktikan wajib mengikuti aturan penamaan yang telah ditentukan.

Nomor	Pertemuan	Penamaan
1	Penganalan Bahasa C++ Bagian Pertama	01_Pengenalan_CPP_Bagian_1
2	Pengenalan Bahasa C++ Bagian Kedua	02_Pengenalan_CPP_Bagian_2
3	Abstract Data Type	03_Abstract_Data_Type
4	Single Linked List Bagian Pertama	04_Single_Linked_List_Bagian_1
5	Single Linked List Bagian Kedua	05_Single_Linked_List_Bagian_2
6	Double Linked List Bagian Pertama	06_Double_Linked_List_Bagian_1
7	Stack	07_Stack
8	Queue	08_Queue
9	Assessment Bagian Pertama	09_Assessment_Bagian_1
10	Tree Bagian Pertama	10_Tree_Bagian_1
11	Tree Bagian Kedua	11_Tree_Bagian_2
12	Asistensi Tugas Besar	12_Asistensi_Tugas_Besar
13	Multi Linked List	13_Multi_Linked_List
14	Graph	14_Graph
15	Assessment Bagian Kedua	15_Assessment_Bagian_2
16	Tugas Besar	16_Tugas_Besar

5. **Jam Praktikum:**
    - Jam masuk praktikum adalah **1 jam lebih lambat** dari jadwal yang tercantum. Sebagai contoh, jika jadwal praktikum adalah pukul 06.30 - 09.30, maka aturan praktikum akan diatur sebagai berikut:
        - **06.30 - 07.30**: Waktu ini digunakan untuk **Tugas Praktikum dan Laporan Praktikum** yang dilakukan di luar laboratorium.
        - **07.30 - 0G.30**: Sesi ini mencakup **tutorial, diskusi, dan kasus problem-solving**. Kegiatan ini berlangsung di dalam laboratorium dengan alokasi waktu sebagai berikut:
            - **60 menit pertama**: Tugas terbimbing.
            - **60 menit kedua**: Tugas mandiri.
6. **Pengumpulan Tugasn Pendahuluan:** Tugas Pendahuluan (TP) wajib dikumpulkan melalui GitHub sesuai dengan format berikut:

    nama_repo/nama_pertemuan/TP_Pertemuan_Ke.md

    Sebagai contoh:

    STD_Yudha_Islalmi_Sulistya_XXXXXXXX/01_Running_Modul/TP_01.md

7. **Pengecekan Tugas Pendahuluan:** Pengumpulan laporan praktikum akan diperiksa **1 hari sebelum praktikum selanjutnya** dimulai. Pastikan tugas telah diunggah tepat waktu untuk menghindari sanksi.

8. **Struktur Laporan Praktikum**

1. Cover :

LAPORAN PRAKTIKUM
Modul 10
“TREE (BAGIAN PERTAMA)”

Disusun Oleh:
KAFKA PUTRA RIYADI-2311104041

Kelas:
SE-07-02

Dosen :
Wahyu Andi Saputra, S.Pd., M.Eng,

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024

2. Tujuan

1. Memahami konsep penggunaan fungsi rekursif.
2. Mengimplementasikan bentuk-bentuk fungsi rekursif.
3. Mengaplikasikan struktur data *tree* dalam sebuah kasus pemrograman.
4. Mengimplementasikan struktur data *tree*, khususnya *Binary Tree*.

3. Landasan Teori

Dalam struktur data, tree adalah model data hierarkis yang terdiri dari simpul (nodes) yang dihubungkan oleh sisi (edges). Berikut adalah penjelasan lebih lanjut mengenai komponen utama dalam tree:

a. Root (Akar):
   Simpul utama dalam tree yang tidak memiliki parent.
b. Node (Simpul):
   Setiap elemen dalam tree. Bisa berisi data dan memiliki relasi dengan simpul lainnya.
c. Edge (Tepi):
   Penghubung antara dua simpul yang menunjukkan hubungan parent-child.
d. Parent (Induk):
   Simpul yang memiliki satu atau lebih child yang terhubung kepadanya.
e. Child (Anak):
   Simpul yang terhubung langsung ke parent.
f. Leaf (Daun):
   Simpul yang tidak memiliki child (anak).
g. Height (Tinggi):
   Panjang jalur terpanjang dari root ke leaf.
h. Subtree:
   Bagian dari tree yang terdiri dari simpul tertentu beserta semua simpul turunannya.

Tree bersifat acyclic, artinya tidak ada siklus di dalamnya. Struktur ini digunakan dalam berbagai aplikasi seperti pencarian data (misalnya, binary search tree) dan penyusunan data (misalnya, dalam file system).

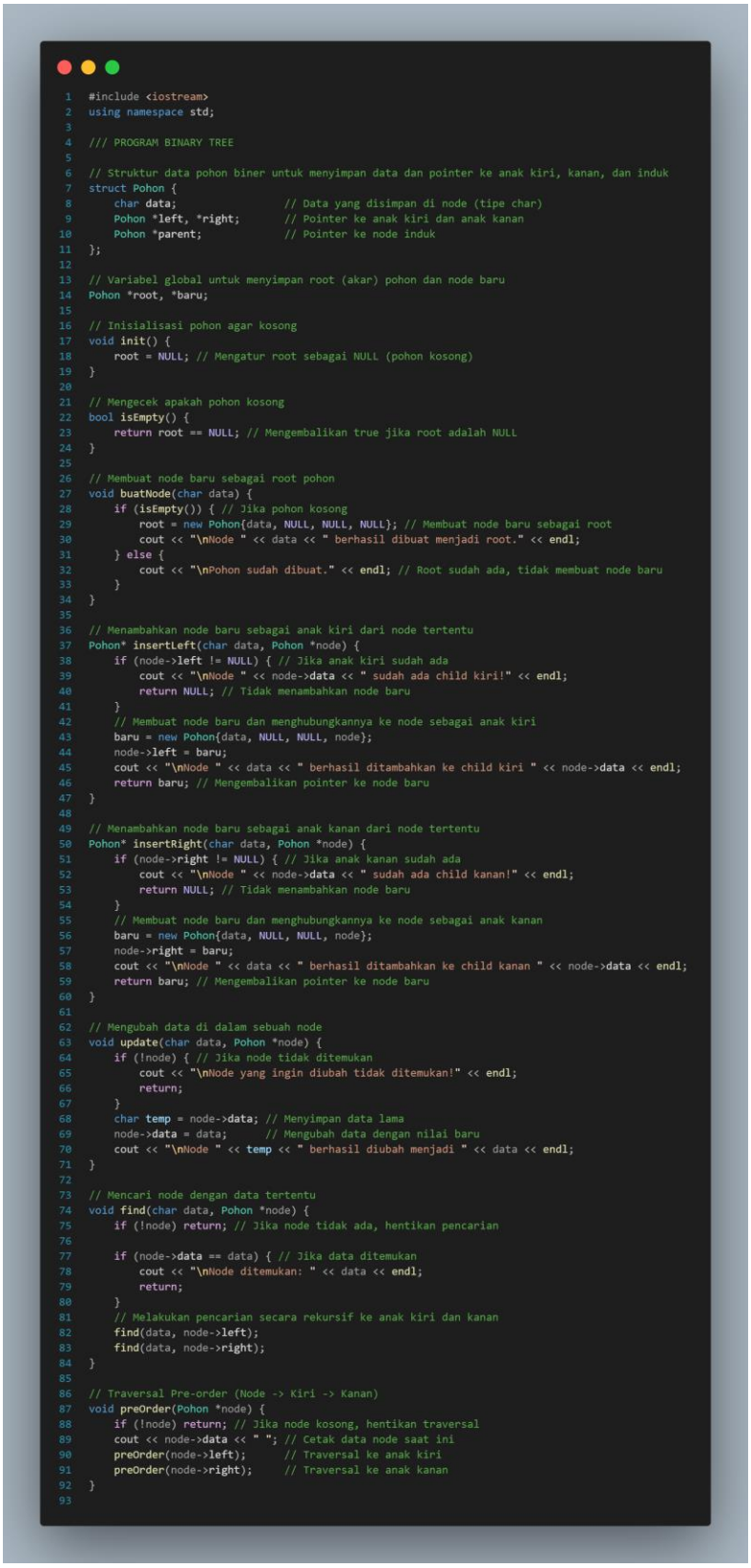
Berikut jenis-jenis *tree* secara:

1. Binary Tree: Setiap node punya maksimal 2 anak.
2. Binary Search Tree (BST): Binary Tree dengan aturan anak kiri < induk < anak kanan.
3. Balanced Tree: Pohon seimbang, contoh: AVL Tree, Red-Black Tree.
4. Complete Binary Tree: Semua level penuh kecuali terakhir, diisi dari kiri ke kanan.
5. Full Binary Tree: Setiap node non-daun punya tepat 2 anak.
6. Perfect Binary Tree: Semua level penuh, semua daun di level sama.
7. N-ary Tree: Node punya maksimal N anak.
8. Heap Tree:
   a. Max-Heap: Induk > Anak.
   b. Min-Heap: Induk < Anak.
9. B-Tree: Pohon seimbang untuk basis data, node punya banyak anak.
10. Trie: Untuk pencarian string, sering dipakai untuk *autocomplete*.
11. Segment Tree: Untuk operasi pada rentang nilai.
12. Suffix Tree: Untuk pencocokan pola string.
13. Spanning Tree: Subgraf pohon dari graf, tanpa siklus.

4. **Guided**

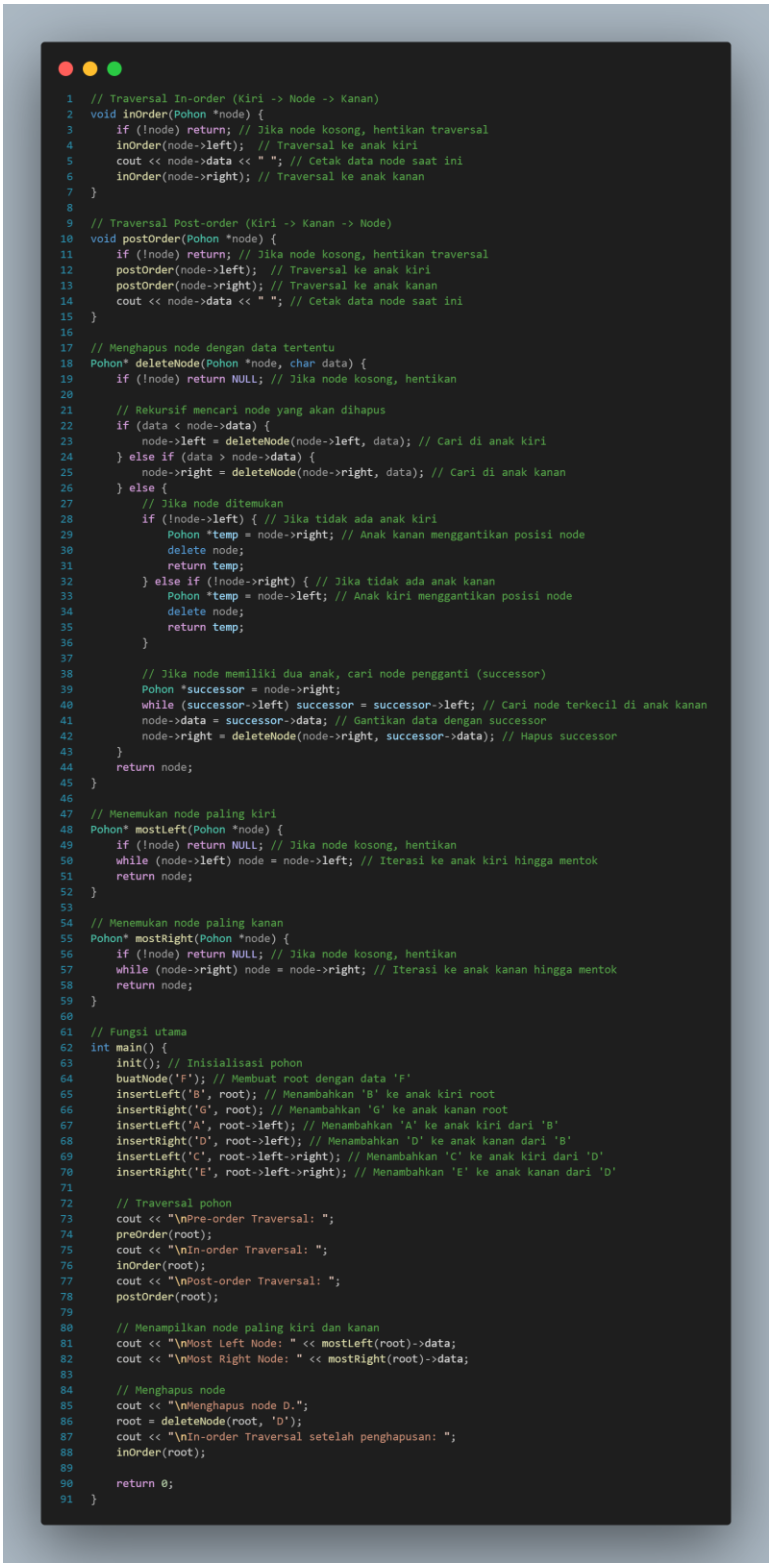
Pada guided ini saya bagi menjadi 2 agar tidak blur dan pecah foto codingannya pada saat dijadikan PDF

Codingan 1:



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

/// PROGRAM BINARY TREE

// Struktur data pohon biner untuk menyimpan data dan pointer ke anak kiri, kanan, dan induk
struct Pohon {
    char data;              // Data yang disimpan di node (tipe char)
    Pohon *left, *right;    // Pointer ke anak kiri dan anak kanan
    Pohon *parent;          // Pointer ke node induk
};

// Variabel global untuk menyimpan root (akar) pohon dan node baru
Pohon *root, *baru;

// Inisialisasi pohon agar kosong
void init() {
    root = NULL; // Mengatur root sebagai NULL (pohon kosong)
}

// Mengecek apakah pohon kosong
bool isEmpty() {
    return root == NULL; // Mengembalikan true jika root adalah NULL
}

// Membuat node baru sebagai root pohon
void buatNode(char data) {
    if (isEmpty()) { // Jika pohon kosong
        root = new Pohon{data, NULL, NULL, NULL}; // Membuat node baru sebagai root
        cout << "\nNode " << data << " berhasil dibuat menjadi root." << endl;
    } else {
        cout << "\nPohon sudah dibuat." << endl; // Root sudah ada, tidak membuat node baru
    }
}

// Menambahkan node baru sebagai anak kiri dari node tertentu
Pohon* insertLeft(char data, Pohon *node) {
    if (node->left != NULL) { // Jika anak kiri sudah ada
        cout << "\nNode " << node->data << " sudah ada child kiri!" << endl;
        return NULL; // Tidak menambahkan node baru
    }
    // Membuat node baru dan menghubungkannya ke node sebagai anak kiri
    baru = new Pohon{data, NULL, NULL, node};
    node->left = baru;
    cout << "\nNode " << data << " berhasil ditambahkan ke child kiri " << node->data << endl;
    return baru; // Mengembalikan pointer ke node baru
}

// Menambahkan node baru sebagai anak kanan dari node tertentu
Pohon* insertRight(char data, Pohon *node) {
    if (node->right != NULL) { // Jika anak kanan sudah ada
        cout << "\nNode " << node->data << " sudah ada child kanan!" << endl;
        return NULL; // Tidak menambahkan node baru
    }
    // Membuat node baru dan menghubungkannya ke node sebagai anak kanan
    baru = new Pohon{data, NULL, NULL, node};
    node->right = baru;
    cout << "\nNode " << data << " berhasil ditambahkan ke child kanan " << node->data << endl;
    return baru; // Mengembalikan pointer ke node baru
}

// Mengubah data di dalam sebuah node
void update(char data, Pohon *node) {
    if (!node) { // Jika node tidak ditemukan
        cout << "\nNode yang ingin diubah tidak ditemukan!" << endl;
        return;
    }
    char temp = node->data; // Menyimpan data lama
    node->data = data;      // Mengubah data dengan nilai baru
    cout << "\nNode " << temp << " berhasil diubah menjadi " << data << endl;
}

// Mencari node dengan data tertentu
void find(char data, Pohon *node) {
    if (!node) return; // Jika node tidak ada, hentikan pencarian

    if (node->data == data) { // Jika data ditemukan
        cout << "\nNode ditemukan: " << data << endl;
        return;
    }
    // Melakukan pencarian secara rekursif ke anak kiri dan kanan
    find(data, node->left);
    find(data, node->right);
}

// Traversal Pre-order (Node -> Kiri -> Kanan)
void preOrder(Pohon *node) {
    if (!node) return; // Jika node kosong, hentikan traversal
    cout << node->data << " "; // Cetak data node saat ini
    preOrder(node->left);      // Traversal ke anak kiri
    preOrder(node->right);     // Traversal ke anak kanan
}

```

Code Program Guided 2 (lanjutan dari codingan diatas) :



```cpp
// Traversal In-order (Kiri -> Node -> Kanan)
void inOrder(Pohon *node) {
    if (!node) return; // Jika node kosong, hentikan traversal
    inOrder(node->left);  // Traversal ke anak kiri
    cout << node->data << " "; // Cetak data node saat ini
    inOrder(node->right); // Traversal ke anak kanan
}

// Traversal Post-order (Kiri -> Kanan -> Node)
void postOrder(Pohon *node) {
    if (!node) return; // Jika node kosong, hentikan traversal
    postOrder(node->left);  // Traversal ke anak kiri
    postOrder(node->right); // Traversal ke anak kanan
    cout << node->data << " "; // Cetak data node saat ini
}

// Menghapus node dengan data tertentu
Pohon* deleteNode(Pohon *node, char data) {
    if (!node) return NULL; // Jika node kosong, hentikan

    // Rekursif mencari node yang akan dihapus
    if (data < node->data) {
        node->left = deleteNode(node->left, data); // Cari di anak kiri
    } else if (data > node->data) {
        node->right = deleteNode(node->right, data); // Cari di anak kanan
    } else {
        // Jika node ditemukan
        if (!node->left) { // Jika tidak ada anak kiri
            Pohon *temp = node->right; // Anak kanan menggantikan posisi node
            delete node;
            return temp;
        } else if (!node->right) { // Jika tidak ada anak kanan
            Pohon *temp = node->left; // Anak kiri menggantikan posisi node
            delete node;
            return temp;
        }

        // Jika node memiliki dua anak, cari node pengganti (successor)
        Pohon *successor = node->right;
        while (successor->left) successor = successor->left; // Cari node terkecil di anak kanan
        node->data = successor->data; // Gantikan data dengan successor
        node->right = deleteNode(node->right, successor->data); // Hapus successor
    }
    return node;
}

// Menemukan node paling kiri
Pohon* mostLeft(Pohon *node) {
    if (!node) return NULL; // Jika node kosong, hentikan
    while (node->left) node = node->left; // Iterasi ke anak kiri hingga mentok
    return node;
}

// Menemukan node paling kanan
Pohon* mostRight(Pohon *node) {
    if (!node) return NULL; // Jika node kosong, hentikan
    while (node->right) node = node->right; // Iterasi ke anak kanan hingga mentok
    return node;
}

// Fungsi utama
int main() {
    init(); // Inisialisasi pohon
    buatNode('F'); // Membuat root dengan data 'F'
    insertLeft('B', root); // Menambahkan 'B' ke anak kiri root
    insertRight('G', root); // Menambahkan 'G' ke anak kanan root
    insertLeft('A', root->left); // Menambahkan 'A' ke anak kiri dari 'B'
    insertRight('D', root->left); // Menambahkan 'D' ke anak kanan dari 'B'
    insertLeft('C', root->left->right); // Menambahkan 'C' ke anak kiri dari 'D'
    insertRight('E', root->left->right); // Menambahkan 'E' ke anak kanan dari 'D'

    // Traversal pohon
    cout << "\nPre-order Traversal: ";
    preOrder(root);
    cout << "\nIn-order Traversal: ";
    inOrder(root);
    cout << "\nPost-order Traversal: ";
    postOrder(root);

    // Menampilkan node paling kiri dan kanan
    cout << "\nMost Left Node: " << mostLeft(root)->data;
    cout << "\nMost Right Node: " << mostRight(root)->data;

    // Menghapus node
    cout << "\nMenghapus node D.";
    root = deleteNode(root, 'D');
    cout << "\nIn-order Traversal setelah penghapusan: ";
    inOrder(root);

    return 0;
}
```

Tujuan dari codingan yang telah dibahas pada kelas adalah untuk memberikan implementasi dan simulasi operasi dasar pada struktur data *Binary Tree*. Operasi ini mencakup:

1. Pembuatan dan Inisialisasi Pohon:
   -Membuat pohon biner dari awal, termasuk menambahkan elemen sebagai node root, anak kiri, dan anak kanan.
2. Manipulasi Data dalam Pohon:
   -Menambahkan node ke posisi tertentu (anak kiri/kanan).
   -Mengubah nilai data node yang sudah ada.
3. Pencarian Node:
   -Mencari node berdasarkan data tertentu menggunakan traversal.
4. Traversal Pohon:
   -Menjelajahi pohon dan menampilkan data node dalam urutan Pre-order, In-order, dan Post-order.
5. Penghapusan Node:
   -Menghapus node tertentu dari pohon dengan penanganan kasus node memiliki nol, satu, atau dua anak.
6. Ekstraksi Node Ekstrem:
   -Menemukan node dengan nilai terkecil (paling kiri) dan terbesar (paling kanan).

Outputannya:

```
PS D:\Struktur Data P10\output> & .\'Guided.exe'

Node F berhasil dibuat menjadi root.

Node B berhasil ditambahkan ke child kiri F

Node G berhasil ditambahkan ke child kanan F

Node A berhasil ditambahkan ke child kiri B

Node D berhasil ditambahkan ke child kanan B

Node C berhasil ditambahkan ke child kiri D

Node E berhasil ditambahkan ke child kanan D

Pre-order Traversal: F B A D C E G
In-order Traversal: A B C D E F G
Post-order Traversal: A C E D B G F
Most Left Node: A
Most Right Node: G
Menghapus node D.
In-order Traversal setelah penghapusan: A B C E F G
PS D:\Struktur Data P10\output>
```

5. Unguided

Pada unguided, saya jadikan 1 codingannya. Ttpi fotonya bagi menjadi 2 agar tidak terlalu kecil atau blur pada saat di jadikan PDF

1. UNGUIDED 1,2 dan 3 include dalam 1 file

```cpp
#include <iostream>
#include <climits> // Untuk INT_MIN dan INT_MAX
using namespace std;

struct Pohon {
    char data;
    Pohon *left, *right;
    Pohon *parent;
};
Pohon *root, *baru;
void init() {
    root = NULL;
}
bool isEmpty() {
    return root == NULL;
}
void buatNode(char data) {
    if (isEmpty()) {
        root = new Pohon{data, NULL, NULL, NULL};
        cout << "\nNode " << data << " berhasil dibuat menjadi root." << endl;
    } else {
        cout << "\nPohon sudah dibuat." << endl;
    }
}

Pohon* insertLeft(char data, Pohon *node) {
    if (node->left != NULL) {
        cout << "\nNode " << node->data << " sudah ada child kiri!" << endl;
        return NULL;
    }
    baru = new Pohon{data, NULL, NULL, node};
    node->left = baru;
    cout << "\nNode " << data << " berhasil ditambahkan ke child kiri " << node->data << endl;
    return baru;
}


Pohon* insertRight(char data, Pohon *node) {
    if (node->right != NULL) {
        cout << "\nNode " << node->data << " sudah ada child kanan!" << endl;
        return NULL;
    }
    baru = new Pohon{data, NULL, NULL, node};
    node->right = baru;
    cout << "\nNode " << data << " berhasil ditambahkan ke child kanan " << node->data << endl;
    return baru;
}
void tampilkanChild(Pohon *node) {
    if (!node) return;
    cout << "Child dari node " << node->data << ": ";
    if (node->left) cout << node->left->data << " (kiri) ";
    if (node->right) cout << node->right->data << " (kanan) ";
    if (!node->left && !node->right) cout << "Tidak ada child.";
    cout << endl;
}

void tampilkanDescendant(Pohon *node) {
    if (!node) return;
    if (node->left || node->right) {
        if (node->left) {
            cout << node->left->data << " ";
            tampilkanDescendant(node->left);
        }
        if (node->right) {
            cout << node->right->data << " ";
            tampilkanDescendant(node->right);
        }
    }
}
bool is_valid_bst(Pohon *node, int min_val, int max_val) {
    if (!node) return true; // Pohon kosong adalah BST
    if (node->data <= min_val || node->data >= max_val) return false;
    return is_valid_bst(node->left, min_val, node->data) &&
           is_valid_bst(node->right, node->data, max_val);
}
int cari_simpul_daun(Pohon *node) {
    if (!node) return 0;
    if (!node->left && !node->right) return 1;
    return cari_simpul_daun(node->left) + cari_simpul_daun(node->right);
}
void preOrder(Pohon *node) {
    if (!node) return;
    cout << node->data << " ";
    preOrder(node->left);
    preOrder(node->right);
}
```

Telkom University Purwokerto

2.

```cpp
void menuInput() {
    char pilihan, data, parentData;
    Pohon *parentNode = NULL;
    while (true) {
        cout << "\nMenu Pohon Biner:\n";
        cout << "1. Tambah Node Root\n";
        cout << "2. Tambah Child Kiri\n";
        cout << "3. Tambah Child Kanan\n";
        cout << "4. Tampilkan Child\n";
        cout << "5. Tampilkan Descendant\n";
        cout << "6. Periksa BST\n";
        cout << "7. Hitung Simpul Daun\n";
        cout << "8. Traversal Pre-order\n";
        cout << "9. Keluar\n";
        cout << "Pilihan: ";
        cin >> pilihan;
        switch (pilihan) {
            case '1':
                cout << "Masukkan data root: ";
                cin >> data;
                buatNode(data);
                break;
            case '2':
                cout << "Masukkan parent: ";
                cin >> parentData;
                cout << "Cari parent...\n";
                parentNode = root; // Untuk mencari node parent
                if (parentNode) {
                    cout << "Masukkan data untuk child kiri: ";
                    cin >> data;
                    insertLeft(data, parentNode);
                } else cout << "Parent tidak ditemukan!\n";
                break;
            case '3':
                cout << "Masukkan parent: ";
                cin >> parentData;
                parentNode = root;
                if (parentNode) {
                    cout << "Masukkan data untuk child kanan: ";
                    cin >> data;
                    insertRight(data, parentNode);
                } else cout << "Parent tidak ditemukan!\n";
                break;
            case '4':
                cout << "Masukkan node: ";
                cin >> data;
                tampilkanChild(root);
                break;
            case '5':
                cout << "Masukkan node: ";
                            cin >> data;
                parentNode = root;
                if (parentNode) {
                    cout << "Descendant dari " << data << ": ";
                    tampilkanDescendant(parentNode);
                    cout << endl;
                } else {
                    cout << "Node tidak ditemukan!\n";
                }
                break;
            case '6':
                cout << "Memeriksa apakah pohon adalah BST... ";
                cout << (is_valid_bst(root, INT_MIN, INT_MAX) ? "Ya, pohon adalah BST.\n" : "Tidak, pohon bukan BST.\n");
                break;
            case '7':
                cout << "Jumlah simpul daun: " << cari_simpul_daun(root) << endl;
                break;
            case '8':
                cout << "Pre-order Traversal: ";
                preOrder(root);
                cout << endl;
                break;
            case '9':
                cout << "Keluar dari program.\n";
                return;
            default:
                cout << "Pilihan tidak valid! Silakan coba lagi.\n";
        }
    }
}
int main() {
    init(); // Inisialisasi pohon
    menuInput(); // Menjalankan menu interaktif
    return 0;
}
```

Outputannya:

Apabila kita menginput angka 1 maka tampilan outputannya seperti ini, yaitu untuk membuat akar pohonnya atau induknya:

```
Menu Pohon Biner:
1. Tambah Node Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child
5. Tampilkan Descendant
6. Periksa BST
7. Hitung Simpul Daun
8. Traversal Pre-order
9. Keluar
Pilihan: 1
Masukkan data root: A

Pohon sudah dibuat.
```

Apabila kita menginput angka 2 maka tampilan outputannya seperti ini:

```
Menu Pohon Biner:
1. Tambah Node Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child
5. Tampilkan Descendant
6. Periksa BST
7. Hitung Simpul Daun
8. Traversal Pre-order
9. Keluar
Pilihan: 2
Masukkan parent: A
Cari parent...
Masukkan data untuk child kiri: B

Node 1 sudah ada child kiri!
```

Fungsi dari menginput 2 yaitu untuk menambahkan child atau anak dari induk A dari sisi kiri.

Apabila kita menginput angka 3 maka tampilan outputannya seperti ini:

```
Menu Pohon Biner:
1. Tambah Node Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child
5. Tampilkan Descendant
6. Periksa BST
7. Hitung Simpul Daun
8. Traversal Pre-order
9. Keluar
Pilihan: 3
Masukkan parent: A
Masukkan data untuk child kanan: C

Node C berhasil ditambahkan ke child kanan 1
```

Fungsi dari menginput 3 yaitu untuk menambahkan child atau anak dari induk A dari sisi kanan.

Apabila kita menginput angka 4 maka tampilan outputannya seperti ini:

```
Menu Pohon Biner:
1. Tambah Node Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child
5. Tampilkan Descendant
6. Periksa BST
7. Hitung Simpul Daun
8. Traversal Pre-order
9. Keluar
Pilihan: 4
Masukkan node: A
Child dari node A: B (kiri) C (kanan)
```

Input angka 5 untuk menampilkan Descendant nya:

```
Menu Pohon Biner:
1. Tambah Node Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child
5. Tampilkan Descendant
6. Periksa BST
7. Hitung Simpul Daun
8. Traversal Pre-order
9. Keluar
Pilihan: 5
Masukkan node: A
Descendant dari A: B C
```

Output dari Inputan 6

```
Menu Pohon Biner:
1. Tambah Node Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child
5. Tampilkan Descendant
6. Periksa BST
7. Hitung Simpul Daun
8. Traversal Pre-order
9. Keluar
Pilihan: 6
Memeriksa apakah pohon adalah BST... Tidak, pohon bukan BST.
```

Output dari Inputan 7

```
Menu Pohon Biner:
1. Tambah Node Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child
5. Tampilkan Descendant
6. Periksa BST
7. Hitung Simpul Daun
8. Traversal Pre-order
9. Keluar
Pilihan: 7
Jumlah simpul daun: 2
```

Output dari Inputan 8

```
Menu Pohon Biner:
1. Tambah Node Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child
5. Tampilkan Descendant
6. Periksa BST
7. Hitung Simpul Daun
8. Traversal Pre-order
9. Keluar
Pilihan: 8
Pre-order Traversal: A B C
```

Output dari Inputan 9 yaitu keluar dari programnya

6. Kesimpulan

Jadi kesimpulannya Tree adalah model data hierarkis yang terdiri dari simpul-simpul (nodes) yang dihubungkan oleh sisi (edges) dan bersifat acyclic (tidak memiliki siklus). Komponen utama tree meliputi root (simpul utama tanpa parent), parent (simpul induk), child (simpul anak), leaf (simpul tanpa anak), subtree (bagian pohon dari simpul tertentu), serta height (tinggi pohon). Jenis-jenis tree meliputi Binary Tree (maksimal 2 anak per simpul), Binary Search Tree (BST) dengan aturan anak kiri < induk < anak kanan, Balanced Tree yang seimbang, serta variasi seperti Complete, Full, dan Perfect Binary Tree. Tree lainnya termasuk Trie untuk pencarian string, Heap untuk struktur prioritas, B-Tree untuk basis data, Segment Tree untuk rentang nilai, dan Suffix Tree untuk pencocokan pola. Tree digunakan dalam berbagai aplikasi, seperti pencarian data, pengelolaan file system, basis data, dan algoritma string.