

LAPORAN PRAKTIKUM

Modul 9

POHON



Disusun Oleh:

Adhiansyah Muhammad Pradana Farawowan - 2211104038

S1SE-07-02

Asisten Praktikum:

Aldi Putra

Andini Nur Hidayah

Dosen:

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

PROGRAM STUDI S1 REKAYASAN PERANGKAT LUNAK

FAKULTAS INFORMATIKA

UNIVERSITAS TELKOM PURWOKERTO

2024

A. Tujuan

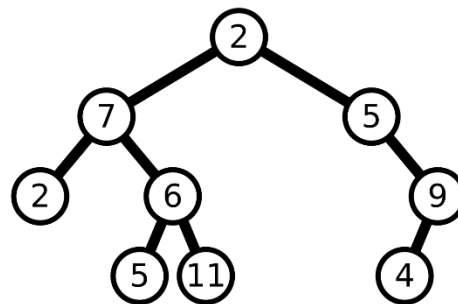
Laporan praktikum ini memiliki tujuan di bawah berikut.

1. Memperkenalkan konsep *tree*, selanjutnya akan disebut *pohon*, sebagai salah satu struktur data
2. Memahami dan mengelola cara kerja pohon
3. Mengimplementasikan antrean dalam bahasa C++

B. Landasan Teori

a. Pohon

Pohon (*tree*) adalah struktur data bertingkat berupa graf dengan tepat satu sisi di antara setiap dua simpul sehingga tidak ada sirkuit atau putaran. Ciri khasnya adalah adanya satu simpul yang tidak memiliki induk, disebut *akar*.



b. Terminologi dasar

1. Akar (*root*)
Simpul teratas.
2. Induk (*Parent*)
Simpul yang memiliki satu atau lebih simpul anak.
3. Anak (*child*)
Simpul yang merupakan keturunan dari simpul induk.
4. Daun (*leaf*)
Simpul yang tidak punya simpul anak.
5. Subpohon (*subtree*)
Pohon yang terdiri dari sebuah node beserta semua keturunannya.
6. Kedalaman (*depth*) / level
Jumlah sisi dari akar ke sebuah simpul.
7. Ketinggian (*height*)
Jumlah sisi dalam lintasan terpanjang dari daun ke akar.

c. Penelusuran

C. Bimbingan (*guided*)

Bimbingan hari ini adalah mengimplementasikan sebuah pohon, dengan modifikasi sendiri.

guided.cpp

```
#include <iostream>

struct Pohon
{
    char data;
    Pohon *kiri;
    Pohon *kanan;
    Pohon *induk;
};

Pohon *akar;
Pohon *baru;

void buat_pohon()
{
    akar = nullptr;
}

bool kosong()
{
    return akar == nullptr;
}

void buat_simpul_akar(char data)
{
    if (kosong())
    {
        akar = new Pohon;
        akar->data = data;
        akar->kanan = nullptr;
        akar->kiri = nullptr;
        akar->induk = nullptr;

        std::cout << "Simpul akar " << akar->data << " berhasil dibuat" << '\n';
    }
    else
    {
        // Kalau pohon sudah dibuat
        std::cout << "Akar sudah ada" << '\n';
    }
}

Pohon *masukkan_ke_kiri(char data, Pohon *simpul)
{
    if (simpul->kiri != nullptr)
    {
        std::cout << "Simpul " << simpul->data << " sudah ada di kiri" << '\n';
        return nullptr;
    }

    baru = new Pohon;
    baru->data = data;
    baru->kiri = nullptr;
    baru->kanan = nullptr;
    baru->induk = simpul;

    simpul->kiri = baru;
    std::cout << "Simpul " << simpul->data << " dibuat di kiri" << '\n';

    return baru;
}

Pohon *masukkan_ke_kanan(char data, Pohon *simpul)
{
    if (simpul->kanan != nullptr)
    {
        std::cout << "Simpul " << simpul->data << " sudah ada di kanan" << '\n';
        return nullptr;
    }

    baru = new Pohon;
    baru->data = data;
```

```

    baru->kiri = nullptr;
    baru->kanan = nullptr;
    baru->induk = simpul;

    simpul->kanan = baru;
    std::cout << "Simpul " << simpul->data << " dibuat di kanan" << '\n';

    return baru;
}

void perbarui_isi_simpul(char data, Pohon *simpul)
{
    if (!simpul)
    {
        std::cout << "Tidak ditemukan" << '\n';
        return;
    }

    char temp = simpul->data;
    simpul->data = data;
    std::cout << "Berhasil diubah menjadi " << data << '\n';
}

void cari_simpul(char data, Pohon *simpul)
{
    if (!simpul)
    {
        std::cout << "Simpul tidak ditemukan" << '\n';
        return;
    }

    if (simpul->data == data)
    {
        std::cout << "Ditemukan" << '\n';
        return;
    }

    cari_simpul(data, simpul->kiri);
    cari_simpul(data, simpul->kanan);
}

void pre_order(Pohon *simpul)
{
    if (!simpul)
    {
        return;
    }

    std::cout << simpul->data << " ";
    pre_order(simpul->kiri);
    pre_order(simpul->kanan);
}

void in_order(Pohon *simpul)
{
    if (!simpul)
    {
        return;
    }

    in_order(simpul->kiri);
    std::cout << simpul->data << " ";
    in_order(simpul->kanan);
}

void post_order(Pohon *simpul)
{
    if (!simpul)
    {
        return;
    }

    post_order(simpul->kiri);
    post_order(simpul->kanan);
}

```

```

        std::cout << simpul->data << " ";
    }

    Pohon *hapus_simpul(Pohon *simpul, char data)
    {
        if (!simpul)
        {
            return nullptr;
        }

        if (data < simpul->data)
        {
            simpul->kiri = hapus_simpul(simpul->kiri, data);
        }
        else if (data > simpul->data)
        {
            simpul->kanan = hapus_simpul(simpul->kanan, data);
        }
        else
        {
            if (!simpul->kiri)
            {
                Pohon *temp = simpul->kanan;
                delete simpul;
                return temp;
            }
            else if (!simpul->kanan)
            {
                Pohon *temp = simpul->kiri;
                delete simpul;
                return temp;
            }

            Pohon *penerus = simpul->kanan;
            while (penerus->kiri)
            {
                penerus = penerus->kiri;
            }

            simpul->data = penerus->data;
            simpul->kanan = hapus_simpul(simpul->kanan, penerus->data);
        }

        return simpul;
    }

    Pohon *simpul_paling_kiri(Pohon *simpul) {
        if (!simpul) {
            return nullptr;
        }

        while (simpul->kiri)
        {
            simpul = simpul->kiri;
        }

        return simpul;
    }

    Pohon *simpul_paling_kanan(Pohon *simpul) {
        if (!simpul) {
            return nullptr;
        }

        while (simpul->kanan)
        {
            simpul = simpul->kanan;
        }

        return simpul;
    }

    int main() {
        buat_pohon();
    }

```

```

    buat_simpul_akar('F');

    masukkan_ke_kiri('B', akar);
    masukkan_ke_kanan('G', akar);

    masukkan_ke_kiri('A', akar->kiri);
    masukkan_ke_kanan('D', akar->kiri);

    masukkan_ke_kiri('C', akar->kiri->kanan);
    masukkan_ke_kanan('E', akar->kiri->kanan);

    // Penelusuran pohon
    std::cout << "Preorder: ";
    pre_order(akar);
    std::cout << '\n';

    std::cout << "Inorder: ";
    in_order(akar);
    std::cout << '\n';

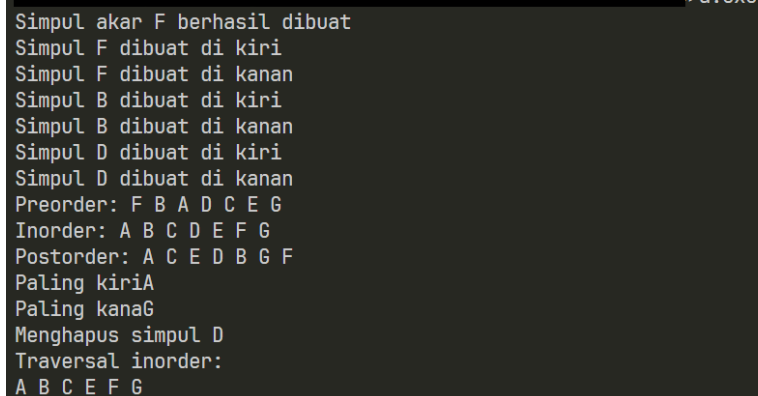
    std::cout << "Postorder: ";
    post_order(akar);
    std::cout << '\n';

    // Menampilkan simpul paling kiri dan paling kanan
    std::cout << "Paling kiri" << simpul_paling_kiri(akar)->data << '\n';
    std::cout << "Paling kanan" << simpul_paling_kanan(akar)->data << '\n';

    // Menghapus simpul
    std::cout << "Menghapus simpul D" << '\n';
    akar = hapus_simpul(akar, 'D');
    std::cout << "Traversal inorder: " << '\n';
    in_order(akar);
}

```

Output dari guided 1.cpp



```

>a.exe
Simpul akar F berhasil dibuat
Simpul F dibuat di kiri
Simpul F dibuat di kanan
Simpul B dibuat di kiri
Simpul B dibuat di kanan
Simpul D dibuat di kiri
Simpul D dibuat di kanan
Preorder: F B A D C E G
Inorder: A B C D E F G
Postorder: A C E D B G F
Paling kiriA
Paling kananG
Menghapus simpul D
Traversal inorder:
A B C E F G

```

D. Tugas mandiri (*unguided*)

1. Modifikasi guided tree diatas dengan program menu menggunakan input data tree dari user dan berikan fungsi tambahan untuk menampilkan node child dan descendant dari node yang diinputkan!
2. Buatlah fungsi rekursif `is_valid_bst(node, min_val, max_val)` untuk memeriksa apakah suatu pohon memenuhi properti Binary Search Tree. Uji fungsi ini pada berbagai pohon, baik yang valid maupun tidak valid sebagai BST.
3. Buatlah fungsi rekursif `cari_simpul_daun(node)` untuk menghitung jumlah simpul daun dalam Binary Tree.

Kode

Unguided_1.cpp

```
#include <iostream>
#include <vector>

struct Pohon
{
    char data;
    Pohon *kiri;
    Pohon *kanan;
    Pohon *induk;
};

Pohon *akar;
Pohon *baru;

void buat_pohon()
{
    akar = nullptr;
}

bool kosong()
{
    return akar == nullptr;
}

void buat_simpul_akar(char data)
{
    if (kosong())
    {
        akar = new Pohon;
        akar->data = data;
        akar->kanan = nullptr;
        akar->kiri = nullptr;
        akar->induk = nullptr;

        std::cout << "Simpul akar " << akar->data << " berhasil dibuat" << '\n';
    }
    else
    {
        // Kalau pohon sudah dibuat
        std::cout << "Akar sudah ada" << '\n';
    }
}

Pohon *masukkan_ke_kiri(char data, Pohon *simpul)
{
    if (simpul->kiri != nullptr)
    {
        std::cout << "Simpul " << simpul->data << " sudah ada di kiri" << '\n';
        return nullptr;
    }

    baru = new Pohon;
    baru->data = data;
    baru->kiri = nullptr;
    baru->kanan = nullptr;
    baru->induk = simpul;

    simpul->kiri = baru;
    std::cout << "Simpul " << data << " dibuat di kiri" << '\n';

    return baru;
}

Pohon *masukkan_ke_kanan(char data, Pohon *simpul)
{
    if (simpul->kanan != nullptr)
    {
        std::cout << "Simpul " << simpul->data << " sudah ada di kanan" << '\n';
        return nullptr;
    }
}
```

```

    }

    baru = new Pohon;
    baru->data = data;
    baru->kiri = nullptr;
    baru->kanan = nullptr;
    baru->induk = simpul;

    simpul->kanan = baru;
    std::cout << "Simpul " << data << " dibuat di kanan" << '\n';

    return baru;
}

void perbarui_isi_simpul(char data, Pohon *simpul)
{
    if (!simpul)
    {
        std::cout << "Tidak ditemukan" << '\n';
        return;
    }

    char temp = simpul->data;
    simpul->data = data;
    std::cout << "Berhasil diubah menjadi " << data << '\n';
}

void cari_simpul(char data, Pohon *simpul)
{
    if (!simpul)
    {
        std::cout << "Simpul tidak ditemukan" << '\n';
        return;
    }

    if (simpul->data == data)
    {
        std::cout << "Ditemukan" << '\n';
        return;
    }

    cari_simpul(data, simpul->kiri);
    cari_simpul(data, simpul->kanan);
}

void pre_order(Pohon *simpul)
{
    if (!simpul)
    {
        return;
    }

    std::cout << simpul->data << " ";
    pre_order(simpul->kiri);
    pre_order(simpul->kanan);
}

void in_order(Pohon *simpul)
{
    if (!simpul)
    {
        return;
    }

    in_order(simpul->kiri);
    std::cout << simpul->data << " ";
    in_order(simpul->kanan);
}

void post_order(Pohon *simpul)
{
    if (!simpul)
    {
        return;
    }

```



```

    }

    post_order(simpul->kiri);
    post_order(simpul->kanan);
    std::cout << simpul->data << " ";
}

Pohon *hapus_simpul(Pohon *simpul, char data)
{
    if (!simpul)
    {
        return nullptr;
    }

    if (data < simpul->data)
    {
        simpul->kiri = hapus_simpul(simpul->kiri, data);
    }
    else if (data > simpul->data)
    {
        simpul->kanan = hapus_simpul(simpul->kanan, data);
    }
    else
    {
        if (!simpul->kiri)
        {
            Pohon *temp = simpul->kanan;
            delete simpul;
            return temp;
        }
        else if (!simpul->kanan)
        {
            Pohon *temp = simpul->kiri;
            delete simpul;
            return temp;
        }

        Pohon *penerus = simpul->kanan;
        while (penerus->kiri)
        {
            penerus = penerus->kiri;
        }

        simpul->data = penerus->data;
        simpul->kanan = hapus_simpul(simpul->kanan, penerus->data);
    }

    return simpul;
}

Pohon *simpul_paling_kiri(Pohon *simpul)
{
    if (!simpul)
    {
        return nullptr;
    }

    while (simpul->kiri)
    {
        simpul = simpul->kiri;
    }

    return simpul;
}

Pohon *simpul_paling_kanan(Pohon *simpul)
{
    if (!simpul)
    {
        return nullptr;
    }

    while (simpul->kanan)
    {

```

```

        simpul = simpul->kanan;
    }

    return simpul;
}

Pohon *dapatkan_simpul(char data, Pohon *simpul)
{
    if (!simpul)
    {
        return nullptr;
    }

    if (simpul->data == data)
    {
        return simpul;
    }

    Pohon *result = dapatkan_simpul(data, simpul->kiri);
    if (result)
    {
        return result;
    }
    return dapatkan_simpul(data, simpul->kanan);
}

void tampilkan_anak(Pohon *simpul)
{
    if (!simpul)
    {
        return;
    }

    std::cout << "Simpul " << simpul->data << "\n";
    std::cout << "Kiri:" << " " << "Kanan:" << "\n";
    if (simpul->kiri)
    {
        std::cout << simpul->kiri->data << " ";
    }

    if (simpul->kanan)
    {
        std::cout << "\t\t\t";
        std::cout << simpul->kanan->data << " ";
    }

    std::cout << '\n';
}

void tampilkan_keturunan(Pohon *simpul)
{
    if (!simpul)
    {
        return;
    }

    if (simpul->kiri != nullptr)
    {
        std::cout << simpul->kiri->data << " ";
        tampilkan_keturunan(simpul->kiri);
    }
    if (simpul->kanan != nullptr)
    {
        std::cout << simpul->kanan->data << " ";
        tampilkan_keturunan(simpul->kanan);
    }

    std::cout << '\n';
}

// Edisi spesial BST
// Pohon adalah BST jika traversal dengan inorder menghasilkan deretan yang berurutan
// Waktunya pakai std::vector!
void deretan_dari_inorder(Pohon *simpul, std::vector<char> &deretan_char)

```

```

{
    if (simpul == nullptr)
    {
        return;
    }

    deretan_dari_inorder(simpul->kiri, deretan_char);
    deretan_char.push_back(simpul->data);
    deretan_dari_inorder(simpul->kanan, deretan_char);
}

bool is_valid_bst()
{
    std::vector<char> deretan_char;
    deretan_dari_inorder(akar, deretan_char);

    for (int i = 1; i < deretan_char.size(); i = i + 1)
    {
        if (deretan_char[i] <= deretan_char[i - 1])
        {
            return false;
        }
    }
    return true;
}

// Edisi spesial BST

int jumlah_daun(Pohon *simpul)
{
    if (!simpul)
    {
        return 0;
    }

    if (!simpul->kiri && !simpul->kanan)
    {
        return 1;
    };

    return jumlah_daun(simpul->kiri) + jumlah_daun(simpul->kanan);
}

void menu()
{
    std::cout << "Menu manipulasi pohon" << '\n';
    std::cout << "[1] Buat akar" << '\n';
    std::cout << "[2] Tambah simpul anak di kiri" << '\n';
    std::cout << "[3] Tambah simpul anak di kanan" << '\n';
    std::cout << "[4] Tampilkan anak" << '\n';
    std::cout << "[5] Tampilkan keturunan" << '\n';
    std::cout << "[6] Telusur secara preorder" << '\n';
    std::cout << "[7] Telusur secara inorder" << '\n';
    std::cout << "[8] Telusur secara postorder" << '\n';
    std::cout << "[9] Validasi apakah pohon adalah BST" << '\n';
    std::cout << "[10] Hitung jumlah daun" << '\n';
    std::cout << "[99] Keluar" << '\n';
    std::cout << "\n";

    std::cout << "Pilihan: ";
}

int main()
{
    int pilihan;

    std::cout << "Pohon telah dibuat" << '\n';

    while (true)
    {
        menu();
        std::cin >> pilihan;

        if (std::cin.fail())
        {

```

```

        std::cin.clear();
        std::cin.ignore(10000, '\n');
        continue;
    }

    if (pilihan == 1)
    {
        if (!akar)
        {
            char _data;

            std::cout << "Masukkan data akar: ";
            std::cin >> _data;

            std::cout << "\n";

            buat_simpul_akar(_data);
            std::cout << "Akar berhasil dibuat." << '\n';
        }
        else
        {
            std::cout << "Akar sudah ada." << '\n';
        }
    }
    else if (pilihan == 2)
    {
        char _cari;
        char _data;

        std::cout << "Masukkan induk: ";
        std::cin >> _cari;

        std::cout << "Masukkan data: ";
        std::cin >> _data;

        Pohon *dicari = dapatkan_simpul(_cari, akar);
        if (dicari != nullptr)
        {
            masukkan_ke_kiri(_data, dicari);
            std::cout << "Berhasil ditambahkan ke kiri" << '\n';
        }
        else
        {
            std::cout << "Induk tidak ditemukan" << '\n';
        }
    }
    else if (pilihan == 3)
    {
        char _cari;
        char _data;

        std::cout << "Masukkan induk: ";
        std::cin >> _cari;

        std::cout << "Masukkan data: ";
        std::cin >> _data;

        Pohon *dicari = dapatkan_simpul(_cari, akar);
        if (dicari != nullptr)
        {
            masukkan_ke_kanan(_data, dicari);
            std::cout << "Berhasil ditambahkan ke kanan" << '\n';
        }
        else
        {
            std::cout << "Induk tidak ditemukan" << '\n';
        }
    }
    else if (pilihan == 4)
    {
        char _data;

        std::cout << "Masukkan simpul: ";
        std::cin >> _data;
    }
}

```

```

        Pohon *dicari = dapatkan_simpul(_data, akar);

        tampilkan_anak(dicari);
    }
    else if (pilihan == 5)
    {
        char _data;

        std::cout << "Masukkan simpul: ";
        std::cin >> _data;

        Pohon *dicari = dapatkan_simpul(_data, akar);

        tampilkan_keturunan(dicari);
    }
    else if (pilihan == 6)
    {
        std::cout << "Penelurusan dengan preorder: ";
        pre_order(akar);
        std::cout << '\n';
    }
    else if (pilihan == 7)
    {
        std::cout << "Penelurusan dengan inorder: ";
        in_order(akar);
        std::cout << '\n';
    }
    else if (pilihan == 8)
    {
        std::cout << "Penelurusan dengan postorder: ";
        post_order(akar);
        std::cout << '\n';
    }
    else if (pilihan == 9)
    {
        if (is_valid_bst())
        {
            std::cout << "Pohon merupakan pohon pencarian biner." << '\n';
        }
        else
        {
            std::cout << "Pohon bukan merupakan pohon pencarian biner." << '\n';
        }
    }
    else if (pilihan == 10)
    {
        std::cout << "Jumlah simpul daun: " << jumlah_daun(akar) << '\n';
    }
    else if (pilihan == 99)
    {
        std::cout << "Keluar dari program." << '\n';
        break;
    }
    else
    {
        std::cout << "Pilihan tidak valid." << '\n';
    }
}
return 0;
}

```

Output dari guided_1.cpp

```
Pohon telah dibuat
Menu manipulasi pohon
[1] Buat akar
[2] Tambah simpul anak di kiri
[3] Tambah simpul anak di kanan
[4] Tampilkan anak
[5] Tampilkan keturunan
[6] Telusur secara preorder
[7] Telusur secara inorder
[8] Telusur secara postorder
[9] Validasi apakah pohon adalah BST
[10] Hitung jumlah daun
[99] Keluar
```

```
Pilihan: 1
Masukkan data akar: D

Simpul akar D berhasil dibuat
```

```
Simpul akar D berhasil dibuat
Akar berhasil dibuat.
Menu manipulasi pohon
[1] Buat akar
[2] Tambah simpul anak di kiri
[3] Tambah simpul anak di kanan
[4] Tampilkan anak
[5] Tampilkan keturunan
[6] Telusur secara preorder
[7] Telusur secara inorder
[8] Telusur secara postorder
[9] Validasi apakah pohon adalah BST
[10] Hitung jumlah daun
[99] Keluar
```

```
Pilihan: 2
Masukkan induk: D
Masukkan data: B
Simpul B dibuat di kiri
Berhasil ditambahkan ke kiri
```

```
Menu manipulasi pohon
[1] Buat akar
[2] Tambah simpul anak di kiri
[3] Tambah simpul anak di kanan
[4] Tampilkan anak
[5] Tampilkan keturunan
[6] Telusur secara preorder
[7] Telusur secara inorder
[8] Telusur secara postorder
[9] Validasi apakah pohon adalah BST
[10] Hitung jumlah daun
[99] Keluar
```

```
Pilihan: 3
Masukkan induk: D
Masukkan data: E
Simpul E dibuat di kanan
Berhasil ditambahkan ke kanan
```

```
Menu manipulasi pohon
[1] Buat akar
[2] Tambah simpul anak di kiri
[3] Tambah simpul anak di kanan
[4] Tampilkan anak
[5] Tampilkan keturunan
[6] Telusur secara preorder
[7] Telusur secara inorder
[8] Telusur secara postorder
[9] Validasi apakah pohon adalah BST
[10] Hitung jumlah daun
[99] Keluar
```

```
Pilihan: 2
Masukkan induk: B
Masukkan data: A
Simpul A dibuat di kiri
Berhasil ditambahkan ke kiri
```

```
Menu manipulasi pohon
[1] Buat akar
[2] Tambah simpul anak di kiri
[3] Tambah simpul anak di kanan
[4] Tampilkan anak
[5] Tampilkan keturunan
[6] Telusur secara preorder
[7] Telusur secara inorder
[8] Telusur secara postorder
[9] Validasi apakah pohon adalah BST
[10] Hitung jumlah daun
[99] Keluar
```

```
Pilihan: 3
Masukkan induk: B
Masukkan data: C
Simpul C dibuat di kanan
Berhasil ditambahkan ke kanan
```

```
Menu manipulasi pohon
[1] Buat akar
[2] Tambah simpul anak di kiri
[3] Tambah simpul anak di kanan
[4] Tampilkan anak
[5] Tampilkan keturunan
[6] Telusur secara preorder
[7] Telusur secara inorder
[8] Telusur secara postorder
[9] Validasi apakah pohon adalah BST
[10] Hitung jumlah daun
[99] Keluar
```

Pilihan: 9
Pohon merupakan pohon pencarian biner.

```
Menu manipulasi pohon
[1] Buat akar
[2] Tambah simpul anak di kiri
[3] Tambah simpul anak di kanan
[4] Tampilkan anak
[5] Tampilkan keturunan
[6] Telusur secara preorder
[7] Telusur secara inorder
[8] Telusur secara postorder
[9] Validasi apakah pohon adalah BST
[10] Hitung jumlah daun
[99] Keluar
```

Pilihan: 10
Jumlah simpul daun: 3

```
Menu manipulasi pohon
[1] Buat akar
[2] Tambah simpul anak di kiri
[3] Tambah simpul anak di kanan
[4] Tampilkan anak
[5] Tampilkan keturunan
[6] Telusur secara preorder
[7] Telusur secara inorder
[8] Telusur secara postorder
[9] Validasi apakah pohon adalah BST
[10] Hitung jumlah daun
[99] Keluar
```

Pilihan: 7
Penelusuran dengan inorder: A B C D E