# LAPORAN PRAKTIKUM

**Modul 9**

**Tree**


**Disusun Oleh :**

**Satria Ariq Adelard**

**Dompas/2211104033SE 07 2**

**Asisten Praktikum :**

**Aldi Putra**

**Andini Nur Hidayah**

**Dosen Pengampu :**

**Wahyu Andi Saputra**

**PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK**

**FAKULTAS INFORMATIKA**

**TELKOM UNIVERSITY PURWOKERTO**


**2024**

## 1. Tujuan

a. Mahasiswa dapat mampu menjelaskan definisi dan konsep dari tree.
b. Mahasiswa dapat mampu menerapkan operasi tambah, menghapus, pada tree.
c. Mahasiswa dapat mampu menerapkan operasi tampil data pada tree.

## 2. Landasan Teori

a. Tree

Struktur data pohon (tree) adalah struktur hierarkis yang terdiri dari kumpulan simpul (nodes) yang saling terhubung melalui sisi (edges). Dalam struktur ini, terdapat satu simpul utama yang disebut akar (root) yang menjadi titik awal, dengan cabang-cabang yang tumbuh darinya. Setiap simpul dalam pohon dapat memiliki satu atau lebih anak (child), yang kemudian membentuk cabang-cabang tambahan di dalam struktur tersebut.

**Elemen utama Tree, antara lain:**

- **Akar (Root):** Simpul pertama dalam pohon yang menjadi titik awal dari seluruh struktur. Akar memiliki satu jalur utama yang mengakses seluruh elemen pohon.
- **Simpul (Node):** Setiap elemen pada pohon yang menyimpan data dan dapat memiliki satu atau lebih anak.
- **Anak (Child):** Simpul yang secara langsung terhubung ke simpul lain di atasnya, yang disebut induk.
- **Induk (Parent):** Simpul yang terhubung dengan satu atau lebih simpul di bawahnya, yang disebut anak.
- **Daun (Leaf):** Simpul yang tidak memiliki anak dan berada di ujung cabang pohon.
- **Level:** Menunjukkan posisi atau kedalaman sebuah simpul dalam pohon, dihitung mulai dari akar.
- **Kedalaman (Depth):** Jarak antara simpul tertentu dengan akar pohon.
- **Tinggi (Height):** Jalur terpanjang dari akar ke simpul daun terjauh dalam pohon.

**Jenis jenis Tree, di antaranya:**

- **Pohon Biner (Binary Tree)**: Struktur data berbentuk pohon, di mana setiap simpul hanya dapat memiliki maksimal dua cabang, yaitu cabang kiri dan cabang kanan.

- **Pohon Biner Pencarian (Binary Search Tree - BST)**: Jenis pohon biner yang tersusun sedemikian rupa sehingga nilai pada cabang kiri selalu lebih kecil dari nilai simpul induk, sedangkan nilai pada cabang kanan selalu lebih besar.
- **Heap**:

  **Heap Maksimal**: Pohon biner di mana nilai pada simpul induk selalu lebih besar dibandingkan dengan nilai pada anak-anaknya.

  **Heap Minimal**: Pohon biner di mana nilai pada simpul induk selalu lebih kecil dibandingkan dengan nilai pada anak-anaknya.

  .
- **Pohon AVL**: Sebuah pohon biner pencarian yang selalu dalam keadaan seimbang, sehingga perbedaan tinggi antara subpohon kiri dan kanan pada setiap simpul tidak pernah lebih dari satu.
- **Pohon Trie**: Struktur pohon yang digunakan untuk pencarian string, di mana setiap simpul merepresentasikan satu karakter dalam string, sehingga mempermudah dan mempercepat proses pencarian kata.

**Operasi dasar pada Tree, antara lain:**

- Traversal: Proses mengunjungi setiap simpul dalam pohon. Traversal pada pohon dapat dilakukan dengan tiga cara utama:
- Pre-order: Mengunjungi akar terlebih dahulu, kemudian anak kiri, dan anak kanan.
- In-order: Mengunjungi anak kiri, kemudian akar, dan anak kanan. Pada pohon biner pencarian, traversal in-order menghasilkan urutan data yang terurut.
- Post-order: Mengunjungi anak kiri, anak kanan, dan terakhir akar.
- Level-order: Mengunjungi simpul-simpul pada setiap level secara berurutan.
- Pencarian (Search): Mencari elemen tertentu dalam pohon dengan cara traversal atau menggunakan algoritma khusus pada pohon biner pencarian.
- Penyisipan (Insertion): Menambahkan simpul baru ke dalam pohon.
- Penghapusan (Deletion): Menghapus simpul dari pohon, dengan memperhatikan struktur pohon setelah penghapusan.

## 3. Guided

a. Guided 1

Source Code

```
#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
}

// 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;
```

Output

```
OUTPUT  DEBUG CONSOLE  PROBLEMS  TERMINAL  PORTS  COMMENTS

PS D:\Praktikum Struktur Data\Pertemuan9> cd 'd:\Praktikum Struktur Data\Pertemuan9\GUIDED\output'
PS D:\Praktikum Struktur Data\Pertemuan9\GUIDED\output> & .\'Tree.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:\Praktikum Struktur Data\Pertemuan9\GUIDED\output>
```

Deskripsi

Program ini mengimplementasikan struktur data pohon biner dengan berbagai operasi. Struktur pohon didefinisikan dalam struct Pohon, yang memiliki atribut data untuk menyimpan karakter, serta pointer left, right, dan parent untuk menghubungkan node dengan anak kiri, anak kanan, dan induknya.

Variabel global root digunakan untuk menyimpan akar pohon, sementara baru digunakan untuk membuat node baru. Fungsi init menginisialisasi pohon menjadi kosong, dan isEmpty memeriksa apakah pohon kosong. Fungsi buatNode membuat node baru sebagai akar jika pohon kosong, sementara insertLeft dan insertRight menambahkan node sebagai anak kiri atau kanan dari node tertentu. Program juga menyediakan fungsi update untuk memperbarui data node, find untuk mencari node tertentu, serta traversal preorder, inorder, dan postorder untuk menampilkan elemen-elemen dalam urutan tertentu. Selain itu, fungsi deleteNode memungkinkan penghapusan node dengan mempertimbangkan berbagai kasus, seperti node dengan nol, satu, atau dua anak. Program ini juga memiliki fungsi untuk menemukan node paling kiri (mostLeft) dan paling kanan (mostRight). Dalam fungsi main, pohon dibangun secara bertahap dengan data tertentu, dan hasil traversal serta penghapusan node ditampilkan.

## 4. Unguided

a. Soal 1

Source Code

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

struct Pohon
{
    char data;
    Pohon *left, *right, *parent;
};

// Root pohon
Pohon *root = NULL, *baru;

// Fungsi untuk menginisialisasi pohon
void init()
{
    root = NULL;
}

// Fungsi untuk memeriksa apakah pohon kosong
bool isEmpty()
{
    return root == NULL;
}

// Fungsi untuk membuat node root
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;
    }
}

// Fungsi untuk menambahkan anak kiri
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;
}

// Fungsi untuk menambahkan anak kanan
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;
}

// Fungsi untuk menampilkan anak dari node
void displayChildren(Pohon *node)
{
    if (node == NULL)
    {
        cout << "Node tidak ditemukan." << endl;
        return;
    }
```

```
    cout << "Children dari node " << node->data << ": ";

    // Cek jika anak kiri ada
    if (node->left != NULL)
    {
        cout << node->left->data << " ";
    }

    // Cek jika anak kanan ada
    if (node->right != NULL)
    {
        cout << node->right->data << " ";
    }

    // Jika tidak ada anak kiri dan kanan
    if (node->left == NULL && node->right == NULL)
    {
        cout << "Tidak ada anak." << endl;
    }
    else
    {
        cout << endl;
    }
}

// Fungsi untuk menampilkan keturunan dari node
void displayDescendants(Pohon *node)
{
    if (node == NULL)
    {
        cout << "Node tidak ditemukan." << endl;
        return;
    }

    cout << "Descendants dari node " << node->data << ": ";
    if (node->left != NULL)
    {
        cout << node->left->data << " ";
        displayDescendants(node->left);
    }
    if (node->right != NULL)
    {
        cout << node->right->data << " ";
        displayDescendants(node->right);
    }

    cout << endl;
}

// Fungsi rekursif untuk mencari node
Pohon *findNode(char data, Pohon *node)
{
```

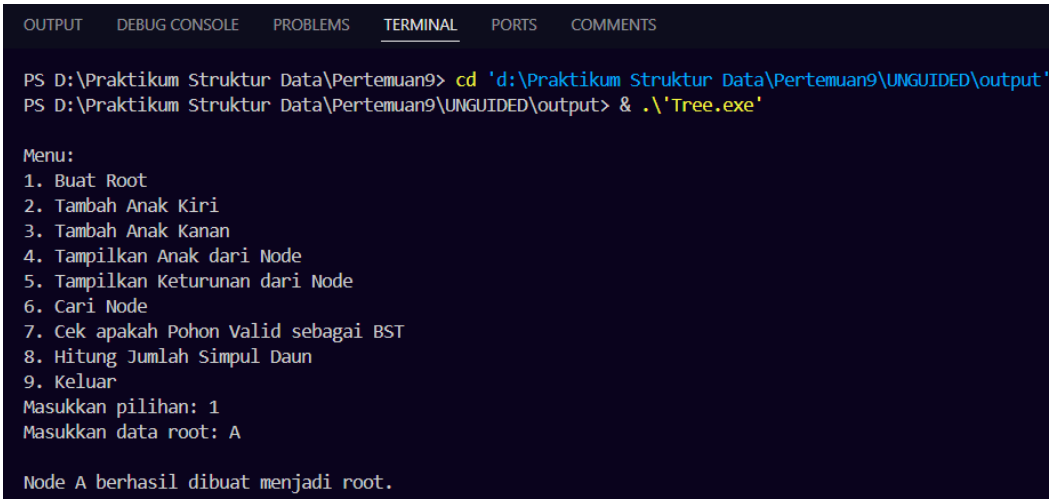
```
    if (node == NULL)
        return NULL;
    if (node->data == data)
        return node;
    Pohon *leftSearch = findNode(data, node->left);
    if (leftSearch)
        return leftSearch;
    return findNode(data, node->right);
}

// Fungsi untuk memeriksa apakah pohon adalah BST
bool is_valid_bst(Pohon *node, char min_val, char max_val)
{
    if (node == NULL)
        return true;

    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);
}

// Fungsi untuk menghitung jumlah simpul daun
int countLeafNodes(Pohon *node)
{
    if (node == NULL)
        return 0;
    if (node->left == NULL && node->right == NULL)
        return 1;
    return countLeafNodes(node->left) + countLeafNodes(node->right);
}

int main()
{
    init();

    // Deklarasi variabel yang dipakai untuk menyimpan node
    Pohon *parentLeft;
    Pohon *parentRight;
    Pohon *node;
    Pohon *nodeDescendants;
    Pohon *foundNode;

    // Menu untuk memasukkan data pohon dan memilih operasi
    int choice;
    char data, parentData;

    while (true)
    {
        cout << "\nMenu:\n";
        cout << "1. Buat Root\n";
        cout << "2. Tambah Anak Kiri\n";
```

```
        cout << "3. Tambah Anak Kanan\n";
        cout << "4. Tampilkan Anak dari Node\n";
        cout << "5. Tampilkan Keturunan dari Node\n";
        cout << "6. Cari Node\n";
        cout << "7. Cek apakah Pohon Valid sebagai BST\n";
        cout << "8. Hitung Jumlah Simpul Daun\n";
        cout << "9. Keluar\n";
        cout << "Masukkan pilihan: ";
        cin >> choice;

        switch (choice)
        {
        case 1:
            cout << "Masukkan data root: ";
            cin >> data;
            buatNode(data);
            break;

        case 2:
            cout << "Masukkan data node yang akan diberi anak kiri: ";
            cin >> parentData;
            cout << "Masukkan data anak kiri: ";
            cin >> data;
            parentLeft = findNode(parentData, root);
            if (parentLeft != NULL)
            {
                insertLeft(data, parentLeft);
            }
            else
            {
                cout << "Node " << parentData << " tidak ditemukan.\n";
            }
            break;

        case 3:
            cout << "Masukkan data node yang akan diberi anak kanan: ";
            cin >> parentData;
            cout << "Masukkan data anak kanan: ";
            cin >> data;
            parentRight = findNode(parentData, root);
            if (parentRight != NULL)
            {
                insertRight(data, parentRight);
            }
            else
            {
                cout << "Node " << parentData << " tidak ditemukan.\n";
            }
            break;

        case 4:
            cout << "Masukkan data node untuk melihat anak-anaknya: ";
            cin >> data;
```

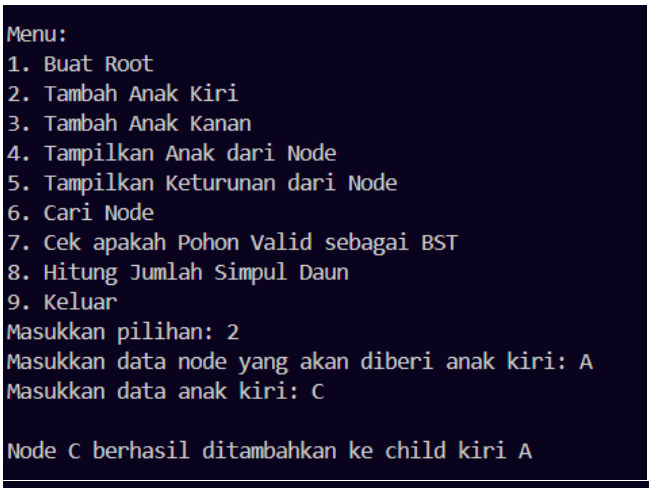
```
            node = findNode(data, root);
            displayChildren(node);
            break;

        case 5:
            cout << "Masukkan data node untuk melihat keturunannya: ";
            cin >> data;
            nodeDescendants = findNode(data, root);
            displayDescendants(nodeDescendants);
            break;

        case 6:
            cout << "Masukkan data node yang akan dicari: ";
            cin >> data;
            foundNode = findNode(data, root);
            if (foundNode != NULL)
            {
                cout << "Node " << data << " ditemukan.\n";
            }
            else
            {
                cout << "Node " << data << " tidak ditemukan.\n";
            }
            break;

        case 7:
            if (is_valid_bst(root, 'A', 'Z'))
            {
                cout << "Pohon ini adalah Binary Search Tree.\n";
            }
            else
            {
                cout << "Pohon ini bukan Binary Search Tree.\n";
            }
            break;

        case 8:
            cout << "Jumlah simpul daun: " << countLeafNodes(root) << endl;
            break;

        case 9:
            cout << "Keluar dari program.\n";
            return 0;

        default:
            cout << "Pilihan tidak valid.\n";
        }
    }

    return 0;
}
```

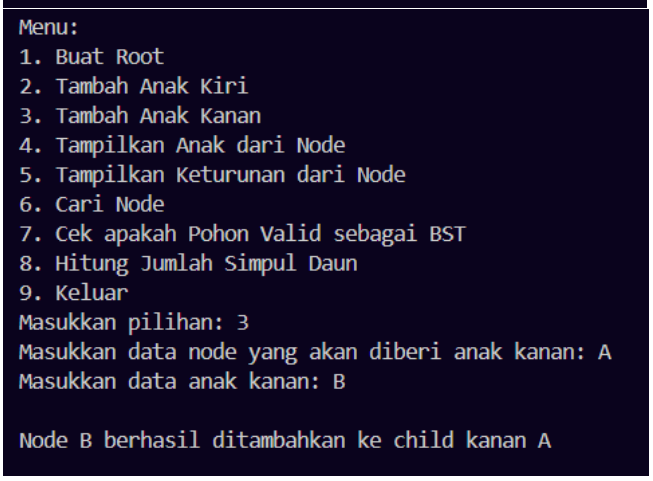
Output

- Membuat Pohon (Root)




- Menambahan Children




- Menampilkan Node

```
Menu:
1. Buat Root
2. Tambah Anak Kiri
3. Tambah Anak Kanan
4. Tampilkan Anak dari Node
5. Tampilkan Keturunan dari Node
6. Cari Node
7. Cek apakah Pohon Valid sebagai BST
8. Hitung Jumlah Simpul Daun
9. Keluar
Masukkan pilihan: 4
Masukkan data node untuk melihat anak-anaknya: A
Children dari node A: C B
```

- Menampilkan Keturunan

```
Menu:
1. Buat Root
2. Tambah Anak Kiri
3. Tambah Anak Kanan
4. Tampilkan Anak dari Node
5. Tampilkan Keturunan dari Node
6. Cari Node
7. Cek apakah Pohon Valid sebagai BST
8. Hitung Jumlah Simpul Daun
9. Keluar
Masukkan pilihan: 5
Masukkan data node untuk melihat keturunannya: A
Descendants dari node A: C Descendants dari node C:
B Descendants dari node B:
```

- Cari Node

```
Menu:
1. Buat Root
2. Tambah Anak Kiri
3. Tambah Anak Kanan
4. Tampilkan Anak dari Node
5. Tampilkan Keturunan dari Node
6. Cari Node
7. Cek apakah Pohon Valid sebagai BST
8. Hitung Jumlah Simpul Daun
9. Keluar
Masukkan pilihan: 6
Masukkan data node yang akan dicari: B
Node B ditemukan.
```

- Cek Jenis Pohon

```
Menu:
1. Buat Root
2. Tambah Anak Kiri
3. Tambah Anak Kanan
4. Tampilkan Anak dari Node
5. Tampilkan Keturunan dari Node
6. Cari Node
7. Cek apakah Pohon Valid sebagai BST
8. Hitung Jumlah Simpul Daun
9. Keluar
Masukkan pilihan: 7
Pohon ini bukan Binary Search Tree.
```

- Hitung Jumlah Leaf

```
Menu:
1. Buat Root
2. Tambah Anak Kiri
3. Tambah Anak Kanan
4. Tampilkan Anak dari Node
5. Tampilkan Keturunan dari Node
6. Cari Node
7. Cek apakah Pohon Valid sebagai BST
8. Hitung Jumlah Simpul Daun
9. Keluar
Masukkan pilihan: 8
Jumlah simpul daun: 2
```

Deskripsi

Program di atas adalah implementasi pohon biner dengan berbagai operasi dasar dan lanjutan, serta menu interaktif. Pohon didefinisikan dalam struktur Pohon, yang memiliki atribut data untuk menyimpan nilai, serta pointer left, right, dan parent untuk menghubungkan simpul. Program ini memungkinkan pengguna membuat simpul root, menambahkan anak kiri dan kanan pada simpul tertentu, serta menampilkan anak atau keturunan dari simpul tertentu. Selain itu, terdapat fungsi untuk mencari simpul berdasarkan nilai data, menghitung jumlah simpul daun, dan memeriksa apakah pohon merupakan Binary Search Tree (BST). Program menyediakan menu interaktif yang memudahkan pengguna untuk melakukan operasi sesuai kebutuhan, seperti menampilkan struktur keturunan, validasi pohon sebagai BST, atau menghitung jumlah simpul daun. Melalui switch-case, setiap pilihan dalam menu memanggil fungsi-fungsi yang relevan, memberikan pengalaman interaktif untuk memanipulasi struktur pohon secara dinamis.