LAPORAN PRAKTIKUM
MODUL 9
TREE

Nama :

Muhammad Rifqi Al Baqi Ananta (2311104005)

Dosen :

Yudha Islami Sulistya, S.Kom., M.Cs.

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSTY PURWOKERTO

2024

I. TP

1. UNGUIDED

CODE:

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

struct Pohon
{
    char data;
    Pohon *left, *right, *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 jadi 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 showChild(Pohon *node)
{
    if (node == NULL)
    {
        cout << "\nNode tidak ditemukan!" << endl;
        return;
    }

    cout << "\nNode: " << node->data << endl;
    cout << "Child kiri: " << (node->left ? node->left->data : '-') << endl;
    cout << "Child kanan: " << (node->right ? node->right->data : '-') << endl;
}

void showDescendants(Pohon *node)
{
    if (node == NULL)
    {
        cout << "\nNode tidak ditemukan!" << endl;
        return;
    }

    queue<Pohon *> q;
    q.push(node);

    cout << "\nDescendants of node " << node->data << ": ";
    bool hasDescendants = false;

    while (!q.empty())
    {
        Pohon *current = q.front();
        q.pop();

        if (current != node)
        {
            hasDescendants = true;
            cout << current->data << " ";
        }

        if (current->left)
            q.push(current->left);
        if (current->right)
            q.push(current->right);
    }

    if (!hasDescendants)
        cout << "(Tidak ada descendant)";
    cout << endl;
}
```

```cpp
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);
}

void test_is_valid_bst()
{
    cout << "\n=== Testing is_valid_bst ===" << endl;

    // Contoh pohon BST
    init();
    buatNode('D');
    insertLeft('B', root);
    insertRight('F', root);
    insertLeft('A', root->left);
    insertRight('C', root->left);
    insertLeft('E', root->right);
    insertRight('G', root->right);

    cout << "Pohon adalah BST: " << (is_valid_bst(root, CHAR_MIN, CHAR_MAX) ? "Ya" : "Tidak") << endl;

    // Contoh pohon yang bukan BST
    root->left->right->data = 'Z'; // Mengubah node 'C' menjadi 'Z', sehingga melanggar aturan BST
    cout << "Pohon adalah BST: " << (is_valid_bst(root, CHAR_MIN, CHAR_MAX) ? "Ya" : "Tidak") << endl;
}

```

```cpp
int cari_simpul_daun(Pohon *node)
{
    if (node == NULL)
        return 0;

    if (node->left == NULL && node->right == NULL)
        return 1;

    return cari_simpul_daun(node->left) + cari_simpul_daun(node->right);
}

void test_cari_simpul_daun()
{
    cout << "\n=== Testing cari_simpul_daun ===" << endl;

    init();
    buatNode('A');
    insertLeft('B', root);
    insertRight('C', root);
    insertLeft('D', root->left);
    insertRight('E', root->left);

    cout << "Jumlah simpul daun: " << cari_simpul_daun(root) << endl;

    insertLeft('F', root->right);
    cout << "Jumlah simpul daun setelah menambah simpul: " << cari_simpul_daun(root) << endl;
}
```

```cpp
void menu()
{
    int choice;
    char data, parentData;
    Pohon *parent;

    do
    {
        cout << "\nMenu:\n";
        cout << "1. Buat root\n";
        cout << "2. Tambah child kiri\n";
        cout << "3. Tambah child kanan\n";
        cout << "4. Tampilkan child\n";
        cout << "5. Tampilkan descendants\n";
        cout << "6. Keluar\n";
        cout << "Pilihan: ";
        cin >> choice;

        switch (choice)
        {
        case 1:
            cout << "Masukkan data root: ";
            cin >> data;
            buatNode(data);
            break;
        case 2:
            cout << "Masukkan data parent: ";
            cin >> parentData;
            cout << "Masukkan data child kiri: ";
            cin >> data;
            parent = root; // Assign root dulu (harus cari jika tree besar)
            insertLeft(data, parent);
            break;
        case 3:
            cout << "Masukkan data parent: ";
            cin >> parentData;
            cout << "Masukkan data child kanan: ";
            cin >> data;
            parent = root; // Assign root dulu (harus cari jika tree besar)
            insertRight(data, parent);
            break;
        case 4:
            cout << "Masukkan data node: ";
            cin >> parentData;
            showChild(root); // Assign root untuk sementara
            break;
        case 5:
            cout << "Masukkan data node: ";
            cin >> parentData;
            showDescendants(root); // Assign root
            break;
        case 6:
            cout << "Keluar dari program.\n";
            break;
        default:
            cout << "Pilihan tidak valid.\n";
        }
    } while (choice != 6);
}

int main()
{
    init();
    menu();
    return 0;
}
```

OUTPUT:

```
Menu:
1. Buat root
2. Tambah child kiri
3. Tambah child kanan
4. Tampilkan child
5. Tampilkan descendants
6. Periksa apakah BST
7. Hitung jumlah simpul daun
8. Keluar
Pilihan: 1
Masukkan data root: A

Node A berhasil dibuat jadi root.

Menu:
1. Buat root
2. Tambah child kiri
3. Tambah child kanan
4. Tampilkan child
5. Tampilkan descendants
6. Periksa apakah BST
7. Hitung jumlah simpul daun
8. Keluar
Pilihan: 2
Masukkan data parent: A
Masukkan data child kiri: B

Node B berhasil ditambahkan ke child kiri A
```

```
Menu:
1. Buat root
2. Tambah child kiri
3. Tambah child kanan
4. Tampilkan child
5. Tampilkan descendants
6. Periksa apakah BST
7. Hitung jumlah simpul daun
8. Keluar
Pilihan: 3
Masukkan data parent: A
Masukkan data child kanan: C

Node C berhasil ditambahkan ke child kanan A

Menu:
1. Buat root
2. Tambah child kiri
3. Tambah child kanan
4. Tampilkan child
5. Tampilkan descendants
6. Periksa apakah BST
7. Hitung jumlah simpul daun
8. Keluar
Pilihan: 4
Masukkan data node: A

Node: A
Child kiri: B
Child kanan: C
```

```
Menu:
1. Buat root
2. Tambah child kiri
3. Tambah child kanan
4. Tampilkan child
5. Tampilkan descendants
6. Periksa apakah BST
7. Hitung jumlah simpul daun
8. Keluar
Pilihan: 5
Masukkan data node: A

Descendants of node A: B C

Menu:
1. Buat root
2. Tambah child kiri
3. Tambah child kanan
4. Tampilkan child
5. Tampilkan descendants
6. Periksa apakah BST
7. Hitung jumlah simpul daun
8. Keluar
Pilihan: 6
Pohon adalah BST: Tidak
```

```
Menu:
1. Buat root
2. Tambah child kiri
3. Tambah child kanan
4. Tampilkan child
5. Tampilkan descendants
6. Periksa apakah BST
7. Hitung jumlah simpul daun
8. Keluar
Pilihan: 7
Jumlah simpul daun: 2

Menu:
1. Buat root
2. Tambah child kiri
3. Tambah child kanan
4. Tampilkan child
5. Tampilkan descendants
6. Periksa apakah BST
7. Hitung jumlah simpul daun
8. Keluar
Pilihan: 8
Keluar dari program.
PS D:\TUGAS ITTP\SEMESTER 3\Praktikum Struktur Data\Pertemuan
```

II. PENJELASAN

Program ini dirancang untuk mengimplementasikan struktur pohon biner dengan berbagai fitur interaktif. Dalam program, pengguna dapat membuat pohon dengan menentukan root, menambahkan node sebagai anak kiri atau kanan dari node tertentu, serta menampilkan informasi tentang node, termasuk anak (child) dan keturunan (descendant). Selain itu, program dilengkapi dengan fitur untuk memverifikasi apakah pohon yang dibuat memenuhi sifat Binary Search Tree (BST), yang memerlukan setiap node memiliki nilai di subtree kiri lebih kecil dan nilai di subtree kanan lebih besar. Program juga menyediakan fungsi untuk menghitung jumlah simpul daun, yaitu node yang tidak memiliki anak kiri maupun kanan. Dengan menu interaktif, pengguna dapat dengan mudah mengelola struktur pohon dan mengeksplorasi berbagai operasi dasar yang biasa digunakan dalam pengolahan pohon biner. Program ini menggabungkan konsep struktur data dinamis, rekursi, dan validasi logika, sehingga sangat berguna untuk memahami implementasi pohon biner dalam pemrograman.