LAPORAN PRAKTIKUM

STRUKTUR DATA

TREE

Nama :

Muhammad Mahrus Ali (2311104006)

Dosen :

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

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

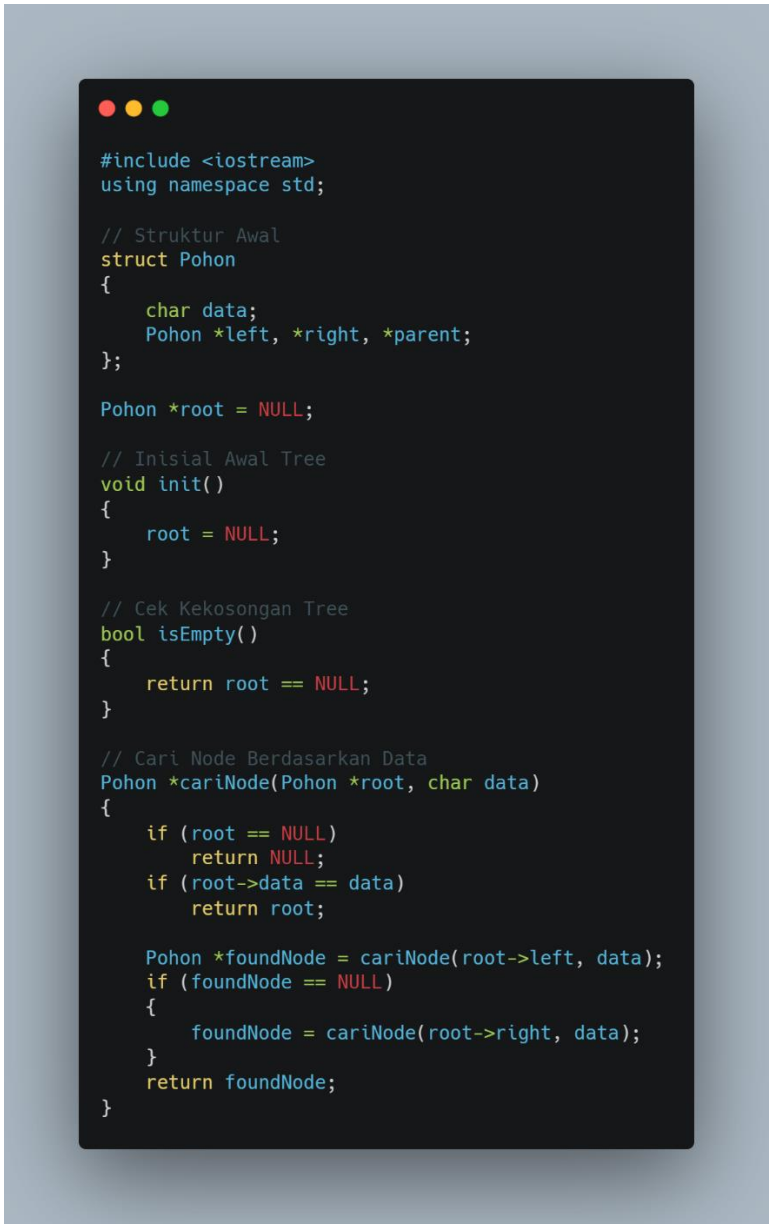
2024

A. 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. Simpul daun adalah node yang tidak memiliki anak kiri maupun kanan.

Jawab :

Inputan :



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

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

Pohon *root = NULL;

// Inisial Awal Tree
void init()
{
    root = NULL;
}

// Cek Kekosongan Tree
bool isEmpty()
{
    return root == NULL;
}

// Cari Node Berdasarkan Data
Pohon *cariNode(Pohon *root, char data)
{
    if (root == NULL)
        return NULL;
    if (root->data == data)
        return root;

    Pohon *foundNode = cariNode(root->left, data);
    if (foundNode == NULL)
    {
        foundNode = cariNode(root->right, data);
    }
    return foundNode;
}
```

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

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

Pohon *root = NULL;

// Inisial Awal Tree
void init()
{
    root = NULL;
}

// Cek Kekosongan Tree
bool isEmpty()
{
    return root == NULL;
}

// Cari Node Berdasarkan Data
Pohon *cariNode(Pohon *root, char data)
{
    if (root == NULL)
        return NULL;
    if (root->data == data)
        return root;

    Pohon *foundNode = cariNode(root->left, data);
    if (foundNode == NULL)
    {
        foundNode = cariNode(root->right, data);
    }
    return foundNode;
}

// Buat Node
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;
    }
}

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

// Tambah Node Bagian Kanan
Pohon *insertRight(char data, Pohon *node)
{
    if (node->right != NULL)
    {
        cout << "\nNode " << node->data << " sudah ada child kanan." << endl;
        return NULL;
    }
    Pohon *baru = new Pohon{data, NULL, NULL, node};
    node->right = baru;
    cout << "\nNode " << data << " berhasil ditambahkan ke child kanan " << node->data << endl;
    return baru;
}

// Menampilkan Child Dari Node Tertentu
void tampilChild(Pohon *node)
{
    if (node == NULL)
    {
        cout << "Node tidak ditemukan." << endl;
        return;
    }
    cout << "Child dari node " << node->data << ": ";
    if (node->left)
        cout << "Kiri: " << node->left->data << " ";
    if (node->right)
        cout << "Kanan: " << node->right->data;
    cout << endl;
}

// Fungsi Rekursif Untuk Menampilkan Semua Descendant
void tampilDescendant(Pohon *node)
{
    if (node == NULL)
        return;
    if (node->left)
    {
        cout << node->left->data << " ";
        tampilDescendant(node->left);
    }
    if (node->right)
    {
        cout << node->right->data << " ";
        tampilDescendant(node->right);
    }
}

// Fungsi Rekursif untuk validasi jika Tree 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 Rekursif Buat Hitung Node Daun
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);
}
```

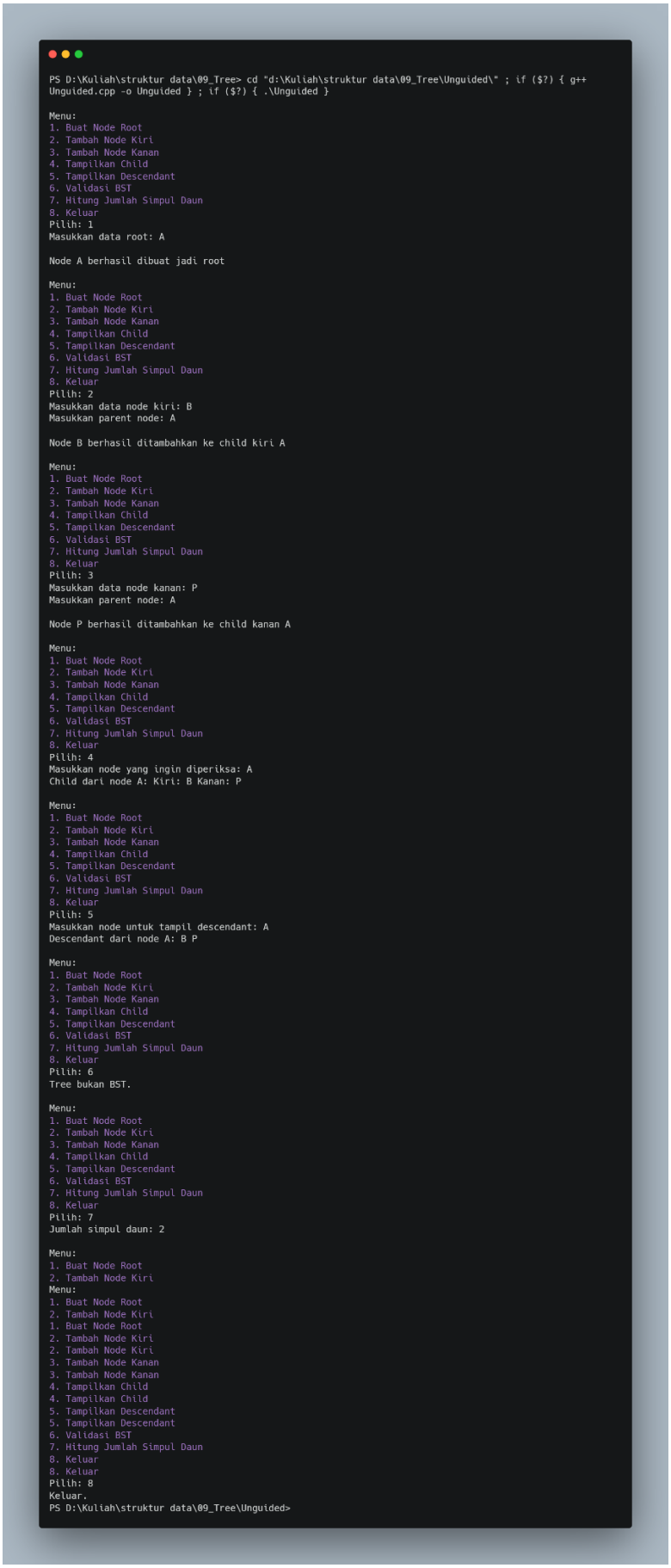
```cpp
// Interface Pengguna
void menu()
{
    int pilihan;
    char data;
    Pohon *node = NULL;

    do
    {
        cout << "\nMenu:" << endl;
        cout << "1. Buat Node Root" << endl;
        cout << "2. Tambah Node Kiri" << endl;
        cout << "3. Tambah Node Kanan" << endl;
        cout << "4. Tampilkan Child" << endl;
        cout << "5. Tampilkan Descendant" << endl;
        cout << "6. Validasi BST" << endl;
        cout << "7. Hitung Jumlah Simpul Daun" << endl;
        cout << "8. Keluar" << endl;
        cout << "Pilih: ";
        cin >> pilihan;

        switch (pilihan)
        {
        case 1:
            cout << "Masukkan data root: ";
            cin >> data;
            buatNode(data);
            break;
        case 2:
        {
            cout << "Masukkan data node kiri: ";
            cin >> data;
            cout << "Masukkan parent node: ";
            char parentData;
            cin >> parentData;
            node = cariNode(root, parentData);
            if (node != NULL)
                insertLeft(data, node);
            else
                cout << "Parent node tidak ditemukan." << endl;
            break;
        }
        case 3:
        {
            cout << "Masukkan data node kanan: ";
            cin >> data;
            cout << "Masukkan parent node: ";
            char parentData;
            cin >> parentData;
            node = cariNode(root, parentData);
            if (node != NULL)
                insertRight(data, node);
            else
                cout << "Parent node tidak ditemukan." << endl;
            break;
        }
        case 4:
        {
            cout << "Masukkan node yang ingin diperiksa: ";
            char parentData;
            cin >> parentData;
            node = cariNode(root, parentData);
            tampilChild(node);
            break;
        }
        case 5:
        {
            cout << "Masukkan node untuk tampil descendant: ";
            char parentData;
            cin >> parentData;
            node = cariNode(root, parentData);
            if (node != NULL)
            {
                cout << "Descendant dari node " << node->data << ": ";
                tampilDescendant(node);
                cout << endl;
            }
            else
                cout << "Node tidak ditemukan." << endl;
            break;
        }
        case 6:
            cout << (is_valid_bst(root, '\0', '\x7F') ? "Tree adalah BST." : "Tree bukan BST.") <<
endl;
            break;
        case 7:
            cout << "Jumlah simpul daun: " << cari_simpul_daun(root) << endl;
            break;
        case 8:
            cout << "Keluar." << endl;
            break;
        default:
            cout << "Pilihan tidak valid." << endl;
        }
    } while (pilihan != 8);
}

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

Maka Outputnya sebagai berikut :



```
PS D:\Kuliah\struktur data\09_Tree> cd "d:\Kuliah\struktur data\09_Tree\Unguided\" ; if ($?) { g++
Unguided.cpp -o Unguided } ; if ($?) { .\Unguided }

Menu:
1. Buat Node Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Child
5. Tampilkan Descendant
6. Validasi BST
7. Hitung Jumlah Simpul Daun
8. Keluar
Pilih: 1
Masukkan data root: A

Node A berhasil dibuat jadi root

Menu:
1. Buat Node Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Child
5. Tampilkan Descendant
6. Validasi BST
7. Hitung Jumlah Simpul Daun
8. Keluar
Pilih: 2
Masukkan data node kiri: B
Masukkan parent node: A

Node B berhasil ditambahkan ke child kiri A

Menu:
1. Buat Node Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Child
5. Tampilkan Descendant
6. Validasi BST
7. Hitung Jumlah Simpul Daun
8. Keluar
Pilih: 3
Masukkan data node kanan: P
Masukkan parent node: A

Node P berhasil ditambahkan ke child kanan A

Menu:
1. Buat Node Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Child
5. Tampilkan Descendant
6. Validasi BST
7. Hitung Jumlah Simpul Daun
8. Keluar
Pilih: 4
Masukkan node yang ingin diperiksa: A
Child dari node A: Kiri: B Kanan: P

Menu:
1. Buat Node Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Child
5. Tampilkan Descendant
6. Validasi BST
7. Hitung Jumlah Simpul Daun
8. Keluar
Pilih: 5
Masukkan node untuk tampil descendant: A
Descendant dari node A: B P

Menu:
1. Buat Node Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Child
5. Tampilkan Descendant
6. Validasi BST
7. Hitung Jumlah Simpul Daun
8. Keluar
Pilih: 6
Tree bukan BST.

Menu:
1. Buat Node Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Child
5. Tampilkan Descendant
6. Validasi BST
7. Hitung Jumlah Simpul Daun
8. Keluar
Pilih: 7
Jumlah simpul daun: 2

Menu:
1. Buat Node Root
2. Tambah Node Kiri
Menu:
1. Buat Node Root
2. Tambah Node Kiri
1. Buat Node Root
2. Tambah Node Kiri
2. Tambah Node Kiri
3. Tambah Node Kanan
3. Tambah Node Kanan
4. Tampilkan Child
4. Tampilkan Child
5. Tampilkan Descendant
5. Tampilkan Descendant
6. Validasi BST
7. Hitung Jumlah Simpul Daun
8. Keluar
8. Keluar
Pilih: 8
Keluar.
PS D:\Kuliah\struktur data\09_Tree\Unguided>
```