Tugas Pendahuluan Modul 9

STRUKTUR DATA - Ganjil 2024/2025

"Tree"

Ketentuan Tugas Pendahuluan

1. Tugas Pendahuluan dikerjakan secara Individu.
2. TP ini bersifat WAJIB, tidak mengerjakan = PENGURANGAN POIN JURNAL / TES ASESMEN.
3. Hanya MENGUMPULKAN tetapi TIDAK MENGERJAKAN = PENGURANGAN POIN JURNAL / TES ASESMEN.
4. Deadline pengumpulan TP Modul 2 adalah Senin, 30 September 2024 pukul 07.30 WIB.
5. TIDAK ADA TOLERANSI KETERLAMBATAN, TERLAMBAT ATAU TIDAK MENGUMPULKAN TP MAKA DIANGGAP TIDAK MENGERJAKAN.
6. DILARANG PLAGIAT (PLAGIAT = E).
7. Kerjakan TP dengan jelas agar dapat dimengerti.
8. Codingan diupload di Github dan upload Laporan di Lab menggunakan format PDF dengan ketentuan:
   TP_MOD_[XX]_NIM_NAMA.pdf

CP (WA):

- Andini (082243700965)
- Imelda (082135374187)

SELAMAT MENGERJAKAN^^

LAPORAN PRAKTIKUM

PERTEMUAN 9

STRUKTUR DATA

Nama :

Zulfa Mustafa Akhyar Iswahyudi (2311104010)

Dosen :

Yudha Islami Sulistya

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

A. Tujuan
   Untuk melatih kompetensi Mahasiswa untuk memperdalam skill pemrograman C++

B. Tools

   Codeblocks, VSCode, Github

TUGAS PENDAHULUAN - UNGUIDED

-

LATIHAN – UNGUIDED

Tiga Soal unguided ini :
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.

Kita jadikan jadi satu program yang sudah dimodifikasi:

modifikasiTree.cpp

```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);
}

// 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', '\xFF') ? "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;
}
```

2311104810_ZULFA MUSTAFA AKHYAR ISWAHYUDI
Hari Doo Rang

Kita bedah satu per satu dalam penjelasan kali ini :

1.) Struktur Awal

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

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

Pohon *root = NULL, *baru;

// Inisial Awal Tree
Qodo Gen: Options | Test this function
void init()
{
    root = NULL;
}

// Cek Kekosongan Tree
Qodo Gen: Options | Test this function
bool isEmpty()
{
    return root == NULL;
}
```

Mulai dari Struct sampai init adalah proses untuk membuat struktural mentah dari program Tree dengan deklarasi ‚data‘ dengan tipe data char dan deklarasi Tree beserta cabang-cabang seperti *Rigth, *Left, dan *Parent.

Atur juga bahwa *root sebagai NULL yang menandakan bahwa Tree masih kosong. Buat method untuk menentukan True/False untuk mengecek apakah Tree kosong atau tidak.

```
// Cari Node Berdasarkan Data
Qodo Gen: Options | Test this function
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;
}
```

Jangan lupa juga untuk menambahkan method cariNode dengan inisial Tree ‘Pohon‘ untuk mendeteksi seluruh node pada Tree untuk ditemukan nilai yang sesuai dengan inputan yang dicari User.

2.) Node

```cpp
// Buat Node
Qodo Gen: Options | Test this function
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
Qodo Gen: Options | Test this function
Pohon *insertLeft(char data, Pohon *node)
{
    if (isEmpty())
    {
        cout << "\nBuat tree dulu." << endl;
        return NULL;
    }
    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;
}

// Tambah Node Bagian Kanan
Qodo Gen: Options | Test this function
Pohon *insertRight(char data, Pohon *node)
{
    if (isEmpty())
    {
        cout << "\nBuat tree dulu." << endl;
        return NULL;
    }
    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;
}
```

Selanjutnya adalah method-method dengan deklarasi ‘Pohon‘ sebagai konstanta Tree yang sudah dibuat. Satu untuk create, satu untuk membuat Node

pada bagian kiri, satu untuk membuat Node pada bagian kanan. Untuk masing-masing method diisikan fungsi isEmpty agar Tree di cek setiap saat ingin melakukan manipulasi Node pada Tree.

Pengkondisian juga dilakukan untuk menambahkan, mengurangi, atau membuat node baru agar sistem dapat memproses pembaruan Tree.

3.) Display Output

```
// Menampilkan Child Dari Node Tertentu
Qodo Gen: Options | Test this function
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
Qodo Gen: Options | Test this function
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);
    }
}
```

Selanjutnya adalah method-method untuk menampilkan data node dari Tree. Disini terbagi menjadi dua yang dimana kedua method untuk mencari child maupun descendant memiliki struktur sintaks kode yang mirip yaitu pengkondisian untuk mengecek seluruh node dari bagian kiri maupun kanan dan menampilkan data-data node yang tertangkap oleh pengkondisian.

4.) Recursive Function

```c
// Fungsi Rekursif untuk validasi jika Tree adalah BST
Qodo Gen: Options | Test this function
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
Qodo Gen: Options | Test this function
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);
}
```

Selanjutnya adalah method boolean dan method integer yang punya fungsi berbeda. Method **is_valid_bst** bertujuan untuk melakukan validasi seluruh node pada Tree. Didalamnya kita menggunakan pengkondisian dengan pipeline agar node paling kecil dan node paling besar dapat divalidasi untuk setiap node bagian kiri dan node bagian kanan.

Sedangkan untuk method **cari_simpul_daun** bertujuan untuk menjumlahkan seluruh node pada Tree. Disini kita mengecek seluruh node dari kiri maupun node dari kanan agar dapat dijumlahkan. Jangan lupa untuk membuat pengkondisian yang mengecek kekosongan node pada Tree.

5.) Interface Menu

```cpp
// Interface Pengguna
Qodo Gen: Options | Test this function
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: ";
            cin >> node->data;
            insertLeft(data, node);
            break;
        case 3:
            cout << "Masukkan data node kanan: ";
            cin >> data;
            cout << "Masukkan parent node: ";
            cin >> node->data;
            insertRight(data, node);
            break;
        case 4:
            cout << "Masukkan node yang ingin diperiksa: ";
            cin >> node->data;
            tampilChild(node);
            break;
        case 5:
            cout << "Masukkan node untuk tampil descendant: ";
            cin >> node->data;
            tampilDescendant(node);
            break;
        case 6:
            cout << (is_valid_bst(root, 'a', 'z') ? "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);
}

Qodo Gen: Options | Test this function
int main()
{
    init();
    menu();
    return 0;
}
```

Kita masuk ke Main Program. Main Program ini berisi menu output untuk melakukan manipulasi Tree beserta inputan dari User pada Terminal. Kita deklarasikan dulu agar inputan dari User berupa angka (integer) dengan variabel ‘pilihan‘, lalu data pada node adalah tipe data ‘char‘, dan deklarasi Tree ‘Pohon‘ dan *node agar di set default kosong.
Kemudian baru kita memasuki menu-menu untuk membuat dan memanipulasi Tree. Percabangannya kita pakai ‘Switch-case’. Jadi untuk setiap menu akan diberikan operasi program yang berbeda sehingga program Tree dapat terimplementasi secara sempurna.

Output :

```
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: 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: 3
Masukkan data node kanan: H
Masukkan parent node: A

Node H 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: H

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 H
```

```
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
3. Tambah Node Kanan
4. Tampilkan Child
5. Tampilkan Descendant
6. Validasi BST
7. Hitung Jumlah Simpul Daun
8. Keluar
Pilih: 8
Keluar.
```

Semoga Selalu diberi kemudahan^^