LAPORAN PRAKTIKUM
PERTEMUAN 9
TREE

Nama :

Haza Zaidan Zidna Fann

(2311104056)

Dosen :

Wahyu Andi Saputra

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. TUJUAN

Memahami konsep penggunaan fungsi rekursif.

Mengimplementasikan bentuk-bentuk fungsi rekursif.

Mengaplikasikan struktur data *tree* dalam sebuah kasus pemrograman.

Mengimplementasikan struktur data *tree*, khususnya *Binary Tree*.

II. LANDASAN TEORI

Tree dalam C++ adalah struktur data non-linear yang merepresentasikan hubungan hierarkis. Tree terdiri dari node yang diatur sebagai root (puncak), parent (induk), child (anak), dan leaf (node tanpa anak). Operasi utama meliputi traversal (*preorder*, *inorder*, *postorder*), pencarian, penambahan, dan penghapusan.

Jenis-jenis tree mencakup:

Binary Tree: Setiap node memiliki maksimal dua anak.

Binary Search Tree (BST): Tree terurut berdasarkan aturan nilai kiri lebih kecil dan kanan lebih besar.

AVL Tree: Versi BST yang selalu seimbang.

General Tree: Node memiliki anak tanpa batasan jumlah.

Implementasi dalam C++: Tree dapat diimplementasikan menggunakan struct/class dan pointer untuk menghubungkan node

III. GUIDED

```cpp
#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;
}
```

```
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
```

Program ini mengimplementasikan binary tree, yaitu struktur data hierarkis dengan node yang memiliki anak kiri, anak kanan, dan induk. Berikut adalah penjelasannya:

1. Struktur Pohon:

   Node menyimpan data serta pointer ke anak kiri, kanan, dan induk. Pohon dimulai dari root, yang merupakan node pertama.

2. Fungsi Utama:

   Inisialisasi pohon kosong.

   Penambahan node baru sebagai root, anak kiri, atau anak kanan.

   Operasi traversal untuk mengunjungi node dalam urutan *pre-order* (node -> kiri -> kanan), *in-order* (kiri -> node -> kanan), dan *post-order* (kiri -> kanan -> node).

   Pencarian node tertentu dan pengubahan data pada node.

   Menghapus node dengan mengatur ulang strukturnya.

   Menemukan node paling kiri (nilai terkecil) atau kanan (nilai terbesar).

3. Hasil Operasi:

   Program membuat pohon dengan root 'F' dan beberapa node anak, lalu melakukan

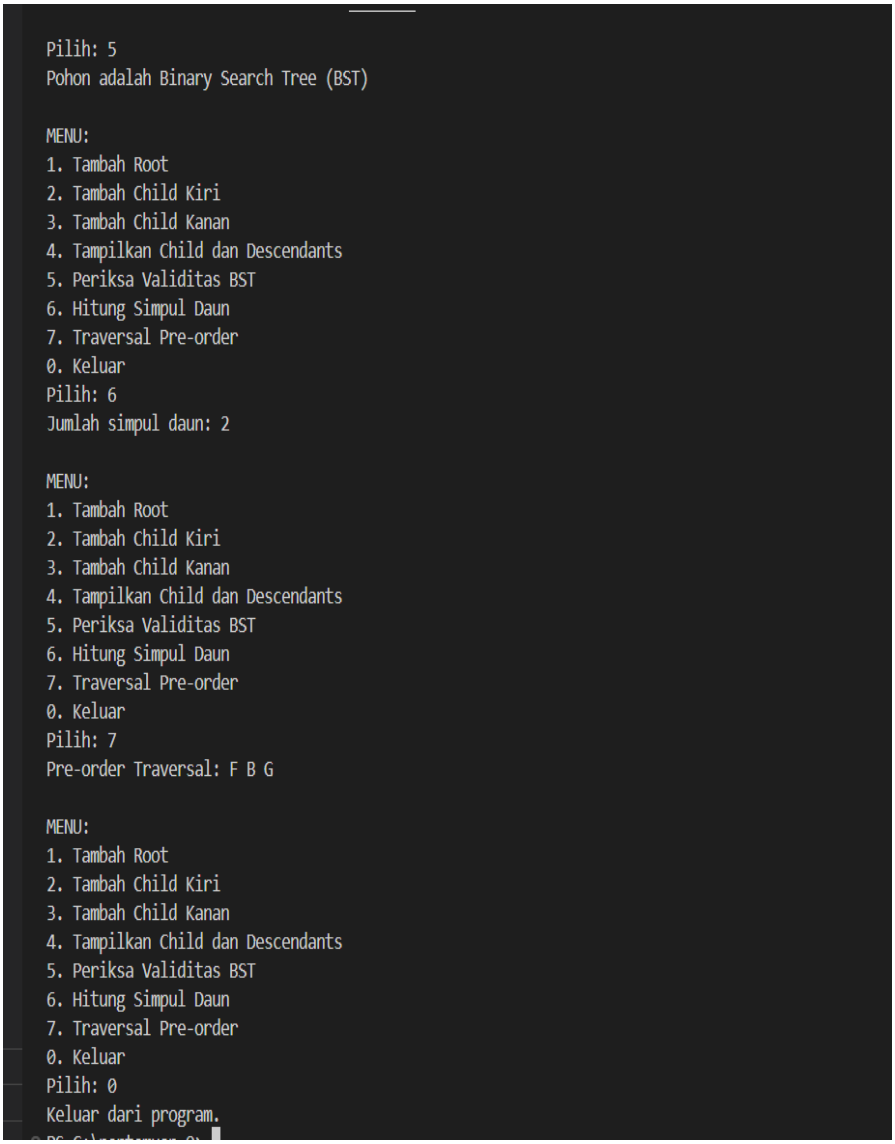
traversal dan modifikasi, seperti menghapus node 'D'. Hasil traversal memperlihatkan susunan node sebelum dan sesudah penghapusan.

IV. UNGUIDED

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

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

// Deklarasi root
Pohon *root = NULL;

// inisialisasi pohon
void init() {
    root = NULL;
}

// Fungsi membuat node baru
Pohon* buatNode(char data, Pohon *parent = NULL) {
    return new Pohon{data, NULL, NULL, parent};
}

// Menambahkan node ke pohon
Pohon* insertLeft(char data, Pohon *node) {
    if (node->left != NULL) {
        cout << "Node " << node->data << " sudah ada child kiri!" << endl;
        return NULL;
    }
    node->left = buatNode(data, node);
    cout << "Node " << data << " berhasil ditambahkan ke child kiri " << node->data << endl;
    return node->left;
}

Pohon* insertRight(char data, Pohon *node) {
    if (node->right != NULL) {
        cout << "Node " << node->data << " sudah ada child kanan!" << endl;
        return NULL;
    }
    node->right = buatNode(data, node);
    cout << "Node " << data << " berhasil ditambahkan ke child kanan " << node->data << endl;
    return node->right;
}

// Fungsi traversal pre-order
void preOrder(Pohon *node) {
    if (!node) return;
    cout << node->data << " ";
    preOrder(node->left);
    preOrder(node->right);
}

// Fungsi untuk menampilkan child dan descendant dari node tertentu
void tampilkanChild(Pohon *node) {
    if (!node) {
        cout << "Node tidak ditemukan!" << endl;
        return;
    }
    cout << "Node: " << node->data << endl;
    if (node->left)
        cout << "Child Kiri: " << node->left->data << endl;
    else
        cout << "Child Kiri: NULL" << endl;

    if (node->right)
        cout << "Child Kanan: " << node->right->data << endl;
    else
        cout << "Child Kanan: NULL" << endl;

    cout << "Descendants (Pre-order): ";
    preOrder(node);
    cout << endl;
}

// Fungsi memeriksa apakah pohon adalah Binary Search Tree
bool is_valid_bst(Pohon *node, char min_val, char max_val) {
    if (!node) 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 menghitung jumlah simpul daun
int cari_simpul_daun(Pohon *node) {
    if (!node) return 0;
    if (!node->left && !node->right) return 1;
    return cari_simpul_daun(node->left) + cari_simpul_daun(node->right);
}

// Menu utama
void menu() {
    char data, parentData;
    int pilihan;
    Pohon *parentNode = NULL;

    do {
        cout << "\nMENU:\n"
             << "1. Tambah Root\n"
             << "2. Tambah Child Kiri\n"
             << "3. Tambah Child Kanan\n"
             << "4. Tampilkan Child dan Descendants\n"
             << "5. Periksa Validitas BST\n"
             << "6. Hitung Simpul Daun\n"
             << "7. Traversal Pre-order\n"
             << "0. Keluar\n"
             << "Pilih: ";
        cin >> pilihan;

        switch (pilihan) {
            case 1:
                if (!root) {
                    cout << "Masukkan data root: ";
                    cin >> data;
                    root = buatNode(data);
                    cout << "Root berhasil dibuat dengan data: " << data << endl;
                } else {
                    cout << "Root sudah ada!" << endl;
                }
                break;

            case 2:
                cout << "Masukkan data parent: ";
                cin >> parentData;
                cout << "Masukkan data anak kiri: ";
                cin >> data;
                parentNode = root; // cari parent node (untuk sederhana, diasumsikan valid)
                if (parentNode)
                    insertLeft(data, parentNode);
                else
                    cout << "Parent tidak ditemukan!" << endl;
                break;

            case 3:
                cout << "Masukkan data parent: ";
                cin >> parentData;
                cout << "Masukkan data anak kanan: ";
                cin >> data;
                parentNode = root; // cari parent node (untuk sederhana, diasumsikan valid)
                if (parentNode)
                    insertRight(data, parentNode);
                else
                    cout << "Parent tidak ditemukan!" << endl;
                break;

            case 4:
                cout << "Masukkan data node: ";
                cin >> data;
                tampilkanChild(root); // langsung tampilkan semua untuk contoh sederhana
                break;

            case 5:
                if (is_valid_bst(root, numeric_limits<char>::min(), numeric_limits<char>::max()))
                    cout << "Pohon adalah Binary Search Tree (BST)" << endl;
                else
                    cout << "Pohon BUKAN Binary Search Tree (BST)" << endl;
                break;

            case 6:
                cout << "Jumlah simpul daun: " << cari_simpul_daun(root) << endl;
                break;

            case 7:
                cout << "Pre-order traversal: ";
                preOrder(root);
                cout << endl;
                break;

            case 0:
                cout << "Keluar dari program." << endl;
                break;

            default:
                cout << "Pilihan tidak valid!" << endl;
        }
    } while (pilihan != 0);
}

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

```
1. Tambah Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child dan Descendants
5. Periksa Validitas BST
6. Hitung Simpul Daun
7. Traversal Pre-order
0. Keluar
Pilih: 1
Masukkan data root: F
Root berhasil dibuat dengan data: F

MENU:
1. Tambah Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child dan Descendants
5. Periksa Validitas BST
6. Hitung Simpul Daun
7. Traversal Pre-order
0. Keluar
Pilih: 2
Masukkan data parent: F
Masukkan data anak kiri: B
Node B berhasil ditambahkan ke child kiri F

MENU:
1. Tambah Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child dan Descendants
5. Periksa Validitas BST
6. Hitung Simpul Daun
7. Traversal Pre-order
0. Keluar
Pilih: 3
Masukkan data parent: F
Masukkan data anak kanan: G
Node G berhasil ditambahkan ke child kanan F
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

1. Tambah Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child dan Descendants
5. Periksa Validitas BST
6. Hitung Simpul Daun
7. Traversal Pre-order
0. Keluar
Pilih: 4
Masukkan data node: F
Node: F
Child Kiri: B
Child Kanan: G
Descendants (Pre-order): F B G

MENU:
1. Tambah Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child dan Descendants
5. Periksa Validitas BST
6. Hitung Simpul Daun
7. Traversal Pre-order
0. Keluar
Pilih: 5
Pohon adalah Binary Search Tree (BST)

MENU:
1. Tambah Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child dan Descendants
5. Periksa Validitas BST
6. Hitung Simpul Daun
7. Traversal Pre-order
0. Keluar
Pilih: 6
Jumlah simpul daun: 2
```



Program ini mengimplementasikan pohon biner dengan berbagai fitur, seperti penambahan node, traversal, validasi, dan analisis struktur pohon. Berikut ringkasannya:

1. Struktur dan Inisialisasi:
   Pohon dimulai kosong, dengan setiap node menyimpan data, anak kiri, anak kanan, dan induknya.
2. Fungsi Utama:
   Tambah Node: Menambahkan node baru sebagai anak kiri/kanan jika belum ada.
   Traversal Pre-order: Mengunjungi node dalam urutan *node -> kiri -> kanan*.
   Validasi BST: Mengecek apakah pohon memenuhi aturan Binary Search Tree.
   Hitung Simpul Daun: Menghitung jumlah node tanpa anak.

Tampilkan Anak dan Keturunan: Menampilkan anak langsung dan semua keturunan suatu node.

3. Menu Operasi:

   Memungkinkan pengguna membuat pohon, menambah node, melakukan traversal, mengecek validitas BST, menghitung simpul daun, dan melihat keturunan suatu node.

V. KESIMPULAN

Tree adalah struktur data fleksibel yang efisien untuk merepresentasikan data kompleks, seperti ekspresi matematika, hubungan objek, atau jalur graf. Pohon biner, termasuk variasi seperti BST, Heap, AVL, dan Red-Black Tree, mendukung pencarian cepat, pengelolaan prioritas, dan keseimbangan data. Tree banyak digunakan dalam sistem database, kompresi data, dan compiler karena kemampuannya mengoptimalkan pengelolaan dan pemrosesan data terstruktur.