LAPORAN PRAKTIKUM
Modul 10
“TREE”

Disusun Oleh:
Fahmi hasan asagaf -2311104074
Kelas:SE 07 02

Dosen :
WAHYU ANDI SAPUTRA

PROGRAM STUDI S1 SOFTWARE ENGINEERING
TELKOM UNIVERSITY
PURWOKERTO
2024

Tujuan Praktikum

1.Memahami penggunaan rekursif
2.Mengimplementasikan Bentuk"Fungsi rekursif
3.Mengimplementasikan Struktur Data Tree Khususnya Binary Tree

Landasan Teori

Landasan Teori

Tree

Tree adalah struktur data hierarkis yang terdiri dari node, di mana satu node utama disebut *root*, dan node lainnya terhubung dalam hubungan parent-child. Setiap node dapat memiliki 0 atau lebih child, tetapi hanya memiliki satu parent. Struktur ini sering digunakan untuk merepresentasikan data dengan hubungan hierarkis seperti pohon keluarga, sistem file, atau organisasi.

Beberapa karakteristik tree:

1. Root: Node paling atas dalam tree.
2. Leaf: Node yang tidak memiliki child.
3. Edge: Hubungan antara dua node.
4. Depth: Jarak dari root ke node tertentu.
5. Height: Jarak terpanjang dari node ke leaf.

Jenis-jenis tree meliputi:

- Binary Tree (setiap node memiliki maksimal 2 child).
- Binary Search Tree (BST) (binary tree dengan aturan child kiri lebih kecil dan child kanan lebih besar dari parent).
- AVL Tree, B-Trees, dll.

Rekursif

Rekursif adalah teknik pemrograman di mana sebuah fungsi memanggil dirinya sendiri untuk menyelesaikan masalah yang lebih kecil dari masalah aslinya. Prinsip utama rekursif adalah adanya:

1. Base Case: Kondisi penghentian yang mencegah fungsi terus-menerus memanggil dirinya sendiri.
2. Recursive Case: Bagian di mana fungsi memanggil dirinya sendiri dengan input yang lebih kecil.

Contoh:

- Rekursif sering digunakan dalam traversing tree (preorder, inorder, postorder).
- Menyelesaikan masalah seperti faktorial, deret Fibonacci, atau pencarian pada struktur data tree.

Rekursif harus digunakan dengan hati-hati agar tidak menyebabkan *stack overflow* akibat tidak adanya base case atau jika terlalu banyak pemanggilan fungsi.

Telkom University Purwokerto

Guided

Code

main.cpp

```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;
}
```

Telkom University Purwokerto

```
// 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
}
```

Telkom University Purwokerto

```c
// 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;
}
```

```cpp
// 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

```
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
Process returned 0 (0x0)   execution time : 0.244 s
Press any key to continue.
```

Penjelasan code

Struktur Pohon

Struktur data Pohon digunakan untuk merepresentasikan setiap node dalam binary tree, dengan atribut:

- data: menyimpan nilai node.
- left dan right: pointer ke anak kiri dan kanan.
- parent: pointer ke induk node.

Fungsi Utama

- init: Menginisialisasi pohon agar kosong dengan root = NULL.
- buatNode: Membuat root node jika pohon kosong.
- insertLeft & insertRight: Menambahkan anak kiri atau kanan ke node tertentu.
- update: Mengubah data pada node tertentu.
- find: Mencari node dengan data tertentu menggunakan traversal rekursif.
- preOrder, inOrder, postOrder: Traversal tree dalam urutan Pre-order, In-order, dan Post-order.
- mostLeft & mostRight: Mencari node paling kiri dan kanan dalam tree.
- deleteNode: Menghapus node dengan beberapa kasus:
    Tidak memiliki anak.
    Memiliki satu anak (anak menggantikan posisi node yang dihapus).
    Memiliki dua anak (menggunakan *successor* untuk menggantikan data node).

Fungsi main

- Membuat pohon dengan root 'F' dan menambahkan beberapa node lainnya.
- Melakukan traversal untuk menampilkan isi pohon.
- Menampilkan node paling kiri (mostLeft) dan paling kanan (mostRight).
- Menghapus node 'D' dan menampilkan hasil traversal setelah penghapusan.

Unguided
SOAL 1

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

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

Pohon *root = nullptr;

// Inisialisasi pohon kosong
void init() {
    root = nullptr;
}

// Mengecek apakah pohon kosong
bool isEmpty() {
    return root == nullptr;
}

// Membuat node baru
Pohon* buatNode(char data, Pohon* parent = nullptr) {
    Pohon* baru = new Pohon;
    baru->data = data;
    baru->left = baru->right = nullptr;
    baru->parent = parent;
    return baru;
}

// Menambahkan node sebagai anak kiri
void insertLeft(char data, Pohon* node) {
    if (node->left != nullptr) {
        cout << "Node " << node->data << " sudah memiliki child kiri!" << endl;
    } else {
        node->left = buatNode(data, node);
        cout << "Node " << data << " berhasil ditambahkan ke child kiri " << node->data
<< endl;
```

```
    }
}

// Menambahkan node sebagai anak kanan
void insertRight(char data, Pohon* node) {
    if (node->right != nullptr) {
        cout << "Node " << node->data << " sudah memiliki child kanan!" << endl;
    } else {
        node->right = buatNode(data, node);
        cout << "Node " << data << " berhasil ditambahkan ke child kanan " << node->data
<< endl;
    }
}

// Menampilkan child dari node tertentu
void displayChild(Pohon* node) {
    if (node == nullptr) {
        cout << "Node tidak ditemukan." << endl;
        return;
    }
    cout << "Node: " << node->data << endl;
    cout << "Left Child: " << (node->left ? node->left->data : '-') << endl;
    cout << "Right Child: " << (node->right ? node->right->data : '-') << endl;
}

// Menampilkan semua descendant dari node tertentu
void displayDescendants(Pohon* node) {
    if (node == nullptr) return;
    cout << node->data << " ";
    displayDescendants(node->left);
    displayDescendants(node->right);
}

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

// Fungsi traversal in-order
void inOrder(Pohon* node) {
    if (node == nullptr) return;
    inOrder(node->left);
    cout << node->data << " ";
    inOrder(node->right);
}

// Menu interaktif
void menu() {
    int choice;
    char data, parentData;
    Pohon* parentNode;

    do {
        cout << "\n--- MENU BINARY TREE ---\n";
        cout << "1. Buat Root\n";
        cout << "2. Tambah Anak Kiri\n";
        cout << "3. Tambah Anak Kanan\n";
        cout << "4. Tampilkan Child\n";
        cout << "5. Tampilkan Descendants\n";
        cout << "6. Periksa Valid BST\n";
        cout << "7. Hitung Simpul Daun\n";
        cout << "8. Traversal In-order\n";
        cout << "0. Keluar\n";
        cout << "Pilih: ";
        cin >> choice;

        switch (choice) {
            case 1:
                if (!isEmpty()) {
                    cout << "Root sudah dibuat!" << endl;
                } else {
                    cout << "Masukkan data root: ";
```

```
>> data;
            root = buatNode(data);
            cout << "Root " << data << " berhasil dibuat." << endl;
        }
        break;

    case 2:
        if (isEmpty()) {
            cout << "Pohon belum dibuat. Buat root terlebih dahulu!" << endl;
        } else {
            cout << "Masukkan parent: ";
            cin >> parentData;
            cout << "Masukkan data anak kiri: ";
            cin >> data;
            parentNode = root;
            insertLeft(data, parentNode); // Asumsi parent selalu root untuk simplifikasi
        }
        break;

    case 3:
        if (isEmpty()) {
            cout << "Pohon belum dibuat. Buat root terlebih dahulu!" << endl;
        } else {
            cout << "Masukkan parent: ";
            cin >> parentData;
            cout << "Masukkan data anak kanan: ";
            cin >> data;
            parentNode = root;
            insertRight(data, parentNode); // Asumsi parent selalu root untuk simplifikasi
        }
        break;

    case 4:
        if (isEmpty()) {
            cout << "Pohon kosong." << endl;
        } else {
            cout << "Masukkan node untuk melihat child: ";
            cin >> data;
            displayChild(root); // Modifikasi pencarian child sesuai struktur
```

```
            }
            break;

        case 5:
            if (isEmpty()) {
                cout << "Pohon kosong." << endl;
            } else {
                cout << "Masukkan node untuk melihat descendants: ";
                cin >> data;
                displayDescendants(root); // Modifikasi pencarian descendants sesuai
struktur
                cout << endl;
            }
            break;

        case 6:
            cout << (is_valid_bst(root, CHAR_MIN, CHAR_MAX) ? "Pohon adalah
BST" : "Pohon bukan BST") << endl;
            break;

        case 7:
            cout << "Jumlah simpul daun: " << cari_simpul_daun(root) << endl;
            break;

        case 8:
            cout << "Traversal In-order: ";
            inOrder(root);
            cout << endl;
            break;

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

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

// Fungsi utama
```

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

Output

```
--- MENU BINARY TREE ---
1. Buat Root
2. Tambah Anak Kiri
3. Tambah Anak Kanan
4. Tampilkan Child
5. Tampilkan Descendants
6. Periksa Valid BST
7. Hitung Simpul Daun
8. Traversal In-order
0. Keluar
Pilih: 1
Masukkan data root: g
Root g berhasil dibuat.

--- MENU BINARY TREE ---
1. Buat Root
2. Tambah Anak Kiri
3. Tambah Anak Kanan
4. Tampilkan Child
5. Tampilkan Descendants
6. Periksa Valid BST
7. Hitung Simpul Daun
8. Traversal In-order
0. Keluar
Pilih:
```

Kesimpulan :

binary tree merupakan struktur data hierarkis yang efisien untuk menyimpan dan mengelola data. Dengan menggunakan berbagai operasi seperti pembuatan, penambahan, penghapusan, dan traversal, binary tree dapat dioptimalkan untuk berbagai kebutuhan, seperti pencarian data yang terstruktur. Implementasi ini menunjukkan pentingnya pemahaman algoritma rekursif dalam mengelola data pada binary tree.