LAPORAN PRAKTIKUM
Modul 10 & 11
TREE

Disusun Oleh:
Aulia Jasifa Br Ginting 2311104060
S1SE-07-02

Dosen :
Wahyu Andi Saputra, S.Pd., M.Eng

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO

2024

1. **Tujuan**
    1. Memahami konsep pengguna fungsi rekursif.
    2. Mengimplementasikan bentuk-bentuk fungsi rekursif.
    3. Mengplikasikan struktur data *tree*, khususnya *Binary Tree*.
2. **Landasan Teori**

    Rekursif adalah suatu proses di mana sebuah fungsi atau prosedur memanggil dirinya sendiri untuk menyelesaikan suatu masalah. Penggunaan rekursif dalam pemrograman memiliki manfaat seperti meningkatkan keterbacaan (readability), modularitas (modularity), dan penggunaan kembali (reusability) subprogram. Namun, untuk menjadi rekursif, sebuah subprogram harus memiliki kondisi khusus yang menghentikan pemanggilan dirinya dan pemanggilan diri itu sendiri jika kondisi tersebut tidak terpenuhi. Meskipun rekursif dapat menyederhanakan solusi dan membuat algoritma lebih mudah dipahami, ia juga memiliki kekurangan, seperti penggunaan memori yang lebih besar dan waktu eksekusi yang lebih lama karena penyimpanan activation record. Oleh karena itu, rekursif sebaiknya digunakan ketika penyelesaian secara iteratif sulit dilakukan, efisiensi sudah memadai, atau kejelasan logika program lebih penting daripada efisiensi.

    Tree adalah struktur data non-linier yang digambarkan sebagai graf tak berarah yang terhubung dan tidak mengandung sirkuit. Karakteristik dari tree mencakup kondisi di mana tree kosong berarti tidak ada node, terdapat satu node tanpa pendahulu yang disebut akar (root), dan semua node lainnya memiliki satu node pendahulu.

    Jenis-jenis Tree

    - Ordered Tree
    - Binary Tree
        - Complete Binary Tree
        - Extended Binary Tree
        - Binary Search Tree
        - AVL Tree
        - Heap Tree

3. **Guided**

    Code Program

```
#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;
}
```

Outputnya:

```
PS C:\Users\LENOVO\Documents\STUDYING\SEMESTER 3\Struktur

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
```

4. Unguided

```
#include <iostream>
#include <limits> // Untuk nilai batas min dan max
using namespace std;
```

Telkom University Purwokerto

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

Pohon *root, *baru;

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

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

// Membuat node baru sebagai root
void buatNode(char data) {
    if (isEmpty()) {
        root = new Pohon{data, NULL, NULL, NULL};
        cout << "Node " << data << " berhasil dibuat sebagai root.\n";
    } else {
        cout << "Pohon sudah ada!\n";
    }
}

// Menambahkan node ke kiri
Pohon* insertLeft(char data, Pohon *node) {
    if (node->left != NULL) {
        cout << "Child kiri dari " << node->data << " sudah ada!\n";
        return NULL;
    }
    baru = new Pohon{data, NULL, NULL, node};
    node->left = baru;
    cout << "Node " << data << " ditambahkan ke kiri dari " << node->data
   << "\n";
    return baru;
}

// Menambahkan node ke kanan
Pohon* insertRight(char data, Pohon *node) {
    if (node->right != NULL) {
        cout << "Child kanan dari " << node->data << " sudah ada!\n";
        return NULL;
    }
    baru = new Pohon{data, NULL, NULL, node};
    node->right = baru;
    cout << "Node " << data << " ditambahkan ke kanan dari " << node->data
   << "\n";
    return baru;
}

// 2 Modifikasi mencari node berdasarkan data secara rekrusif
Pohon* findNode(char data, Pohon *node) {
```

```
    if (!node) return NULL; // Basis rekursif: jika node NULL, kembalikan
   NULL

    if (node->data == data) // Jika data cocok, kembalikan node
        return node;

    // Cari ke anak kiri dan kanan secara rekursif
    Pohon *leftResult = findNode(data, node->left);
    if (leftResult) return leftResult; // Jika ditemukan di anak kiri,
   kembalikan hasil

    Pohon *rightResult = findNode(data, node->right);
    return rightResult; // Jika ditemukan di anak kanan, kembalikan hasil
}

// 1 Modifikasi menampilkan child dari node tertentu
void tampilChild(Pohon *node) {
    if (!node) {
        cout << "Node tidak ditemukan.\n";
        return;
    }
    cout << "Node: " << node->data << "\n";
    if (node->left) cout << "Child Kiri: " << node->left->data << "\n";
    else cout << "Child Kiri: NULL\n";
    if (node->right) cout << "Child Kanan: " << node->right->data << "\n";
    else cout << "Child Kanan: NULL\n";
}

// 1 Modifikasi menampilkan descendant dari node tertentu (rekursif)
void tampilDescendant(Pohon *node) {
    if (!node) return;
    if (node->left || node->right)
        cout << node->data << " memiliki descendant: ";
    if (node->left) cout << node->left->data << " ";
    if (node->right) cout << node->right->data << " ";
    cout << "\n";
    tampilDescendant(node->left);
    tampilDescendant(node->right);
}

// 2 Modifikasi menambahkan fungsi validasi apakah pohon adalah BST
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);
}

// 3 Modifikasi menghitung jumlah simpul daun
int cari_simpul_daun(Pohon *node) {
    if (!node) return 0; // Basis rekursi
    if (!node->left && !node->right) return 1; // Node daun
    return cari_simpul_daun(node->left) + cari_simpul_daun(node->right);
}

// Fungsi utama dengan menu
int main()
```

```
{
    int pilihan;
    char data, parent;
    Pohon *temp = NULL;
    init();

    // 1 Modifikasi membuat menu program agar user dapat menginputkan
   sendiri
    do {
        cout << "\n=== MENU POHON BINARY TREE ===\n";
        cout << "1. Buat Root\n";
        cout << "2. Tambah Child Kiri\n";
        cout << "3. Tambah Child Kanan\n";
        cout << "4. Tampilkan Child Node\n";
        cout << "5. Tampilkan Descendant Node\n";
        cout << "6. Cek Apakah Pohon adalah BST\n";
        cout << "7. Hitung Jumlah Simpul Daun\n";
        cout << "0. Keluar\n";
        cout << "Pilihan: ";
        cin >> pilihan;

        switch (pilihan) {
            case 1:
                cout << "Masukkan data root: ";
                cin >> data;
                buatNode(data);
                break;
            case 2:
                cout << "Masukkan data parent: ";
                cin >> parent;
                cout << "Masukkan data anak kiri: ";
                cin >> data;
                temp = findNode(parent, root); // Cari node berdasarkan
   data
                if (temp) {
                    insertLeft(data, temp);
                    } else {
                        cout << "Node dengan data " << parent << " tidak
   ditemukan!\n";
                    }
                break;
            case 3:
                cout << "Masukkan data parent: ";
                cin >> parent;
                cout << "Masukkan data anak kanan: ";
                cin >> data;
                temp = findNode(parent, root); // Cari node berdasarkan
   data
                if (temp) {
                    insertRight(data, temp);
                } else {
                    cout << "Node dengan data " << parent << " tidak
   ditemukan!\n";
                }
                break;
            case 4:
                cout << "Masukkan node yang ingin ditampilkan child-nya:
```

```
    ";
                cin >> data;
                temp = findNode(data, root); // Cari node berdasarkan data
                if (temp) {
                    tampilChild(temp);
                } else {
                    cout << "Node dengan data " << data << " tidak
    ditemukan!\n";
                }
                break;
            case 5:
                cout << "Masukkan node yang ingin ditampilkan descendant-
    nya: ";
                cin >> data;
                tampilDescendant(findNode(data, root));
                break;
            case 6:
                if (is_valid_bst(root, numeric_limits<char>::min(),
    numeric_limits<char>::max()))
                    cout << "Pohon adalah BST yang valid.\n";
                else
                    cout << "Pohon bukan BST yang valid.\n";
                break;
            case 7:
                cout << "Jumlah simpul daun: " << cari_simpul_daun(root)
    << "\n";
                break;
            case 0:
                cout << "Keluar...\n";
                break;
            default:
                cout << "Pilihan tidak valid!\n";
        }
    } while (pilihan != 0);

    return 0;
}
```

Outputnya:

Telkom University Purwokerto

```
=== MENU POHON BINARY TREE ===
1. Buat Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child Node
5. Tampilkan Descendant Node
6. Cek Apakah Pohon adalah BST
7. Hitung Jumlah Simpul Daun
0. Keluar
Pilihan: 1
Masukkan data root: F
Node F berhasil dibuat sebagai root.

=== MENU POHON BINARY TREE ===
1. Buat Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child Node
5. Tampilkan Descendant Node
6. Cek Apakah Pohon adalah BST
7. Hitung Jumlah Simpul Daun
0. Keluar
Pilihan: 2
Masukkan data parent: F
Masukkan data anak kiri: B
Node B ditambahkan ke kiri dari F
```

```
=== MENU POHON BINARY TREE ===
1. Buat Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child Node
5. Tampilkan Descendant Node
6. Cek Apakah Pohon adalah BST
7. Hitung Jumlah Simpul Daun
0. Keluar
Pilihan: 3
Masukkan data parent: F
Masukkan data anak kanan: G
Node G ditambahkan ke kanan dari F

=== MENU POHON BINARY TREE ===
1. Buat Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child Node
5. Tampilkan Descendant Node
6. Cek Apakah Pohon adalah BST
7. Hitung Jumlah Simpul Daun
0. Keluar
Pilihan: 4
Masukkan node yang ingin ditampilkan child-nya: F
Node: F
Child Kiri: B
Child Kanan: G
```

Telkom University Purwokerto

```
=== MENU POHON BINARY TREE ===
1. Buat Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child Node
5. Tampilkan Descendant Node
6. Cek Apakah Pohon adalah BST
7. Hitung Jumlah Simpul Daun
0. Keluar
Pilihan: 5
Masukkan node yang ingin ditampilkan descendant-nya: F
F memiliki descendant: B G



=== MENU POHON BINARY TREE ===
1. Buat Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child Node
5. Tampilkan Descendant Node
6. Cek Apakah Pohon adalah BST
7. Hitung Jumlah Simpul Daun
0. Keluar
Pilihan: 6
Pohon adalah BST yang valid.
```

```
=== MENU POHON BINARY TREE ===
1. Buat Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child Node
5. Tampilkan Descendant Node
6. Cek Apakah Pohon adalah BST
7. Hitung Jumlah Simpul Daun
0. Keluar
Pilihan: 7
Jumlah simpul daun: 2

=== MENU POHON BINARY TREE ===
1. Buat Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child Node
5. Tampilkan Descendant Node
6. Cek Apakah Pohon adalah BST
7. Hitung Jumlah Simpul Daun
0. Keluar
Pilihan: 9
Pilihan tidak valid!
```

5. **Kesimpulan**

   Pada praktikum ini, mempelajari konsep rekursif dan struktur data tree. Rekursif adalah metode pemrograman di mana fungsi memanggil dirinya sendiri, yang dapat meningkatkan keterbacaan dan modularitas kode, meskipun memiliki kekurangan dalam penggunaan memori dan waktu eksekusi. Struktur data tree, sebagai struktur non-linier, terdiri dari node yang terhubung tanpa sirkuit, dengan satu node sebagai akar (root) dan node lainnya memiliki satu pendahulu. Pemahaman tentang rekursif dan tree sangat penting dalam pengembangan algoritma dan struktur data yang efisien, serta dalam menyelesaikan berbagai permasalahan pemrograman.