

LAPORAN PRAKTIKUM
Modul 10
“TREE (BAGIAN PERTAMA-kEDUA)”



Disusun Oleh:

Nama : Ganes Gemi Putra

NIM : 2311104075

Kelas : SE-07-02

Dosen : WAHYU ANDI SAPUTRA

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024

1. 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.

2. Landasan Teori

Konsep Rekursif

Rekursif adalah teknik pemrograman di mana sebuah fungsi memanggil dirinya

sendiri untuk menyelesaikan masalah yang dapat dibagi menjadi submasalah yang lebih kecil. Hal ini sering digunakan dalam algoritma yang memiliki pola penyelesaian yang berulang. Ciri utama fungsi rekursif adalah:

- Adanya kondisi penghentian (base case).
- Pemanggilan fungsi itu sendiri selama kondisi penghentian belum tercapai.

Rekursif relevan dalam pengimplementasian pohon biner (binary tree), terutama pada operasi traversal (pre-order, in-order, post-order), pencarian, atau perhitungan simpul daun.

Struktur Data Pohon (Tree)

Pohon adalah struktur data non-linear yang terdiri dari simpul (nodes) dengan hubungan hierarkis:

- Akar (Root): Simpul utama yang tidak memiliki orang tua.
- Cabang (Child): Simpul yang memiliki orang tua.
- Daun (Leaf): Simpul tanpa anak.
- Tinggi (Height): Jumlah maksimum simpul dari akar hingga daun.

Pohon biner (Binary Tree) adalah pohon di mana setiap simpul hanya memiliki maksimal dua anak (kiri dan kanan). Dalam Binary Search Tree (BST), aturan tambahan adalah:

- Anak kiri harus lebih kecil dari orang tua.
- Anak kanan harus lebih besar dari orang tua.

Traversal pada Pohon

Traversal adalah proses mengunjungi semua simpul dalam pohon:

- Pre-order: Kunjungi simpul saat ini, lalu anak kiri, kemudian anak kanan.
- In-order: Kunjungi anak kiri, lalu simpul saat ini, kemudian anak kanan.
- Post-order: Kunjungi anak kiri, lalu anak kanan, kemudian simpul saat ini.

Traversal ini sering diimplementasikan secara rekursif untuk memanfaatkan sifat hierarki pohon.

Implementasi Operasi pada Pohon Biner

Operasi-operasi penting yang umum diterapkan pada pohon biner adalah:

- Insert: Menambahkan simpul ke posisi yang sesuai berdasarkan aturan BST.
- Search: Mencari simpul tertentu menggunakan algoritma pencarian rekursif.
- Delete: Menghapus simpul dan merekonstruksi ulang pohon agar tetap valid.
- Traversal: Menelusuri semua simpul pohon sesuai urutan tertentu.

Perhitungan Simpul Daun

Dalam konteks program Anda, simpul daun adalah simpul yang tidak memiliki anak kiri maupun kanan. Fungsi rekursif dapat digunakan untuk menghitung jumlah simpul daun dengan cara memeriksa setiap simpul, apakah ia merupakan daun, lalu menambahkan perhitungannya.

Validasi Binary Search Tree (BST)

Untuk memastikan bahwa suatu pohon memenuhi properti BST, diperlukan fungsi validasi rekursif. Fungsi ini memeriksa:

- Apakah nilai anak kiri lebih kecil dari orang tua.

- Apakah nilai anak kanan lebih besar dari orang tua.
- Properti tersebut harus berlaku pada seluruh simpul pohon.

3. Guided

The screenshot shows a C++ IDE with a file named `main.cpp` and a terminal window displaying the program's output.

main.cpp Code:

```

1 #include <iostream>
2 using namespace std;
3
4 // PROGRAM BINARY TREE
5
6 // Struktur data pohon biner untuk menyimpan data dan pointer ke anak kiri, kanan, dan induk
7 struct Node {
8     char data; // Data yang disimpan di node (tipe char)
9     Node* left; // Pointer ke anak kiri dan anak kanan
10    Node* right; // Pointer ke anak kanan
11    Node* parent; // Pointer ke node induk
12}
13
14 // Variabel global untuk menyimpan root (akar) pohon dan node baru
15 Node* root = nullptr;
16
17 // Inisialisasi pohon agar kosong
18 void init() {
19     root = nullptr; // Mengatur root sebagai NULL (pohon kosong)
20 }
21
22 // Mengecek apakah pohon kosong
23 bool isEmpty() {
24     return root == nullptr; // Mengembalikan true jika root adalah NULL
25 }
26
27 // Membuat node baru sebagai root pohon
28 void makeRoot(char data) {
29     if (isEmpty()) { // Jika pohon kosong
30         Node* newNode = new Node; // Membuat node baru sebagai root
31         newNode->data = data;
32         root = newNode;
33         cout << "Node " << data << " berhasil dibuat menjadi root." << endl;
34     } else {
35         cout << "Pohon sudah dibuat." << endl; // Root sudah ada, tidak membuat node baru
36     }
37 }
38
39 // Menambahkan node baru sebagai anak kiri dari node tertentu
40 void addLeftNode(char data, Node* parent) {
41     Node* newNode = new Node;
42     newNode->data = data;
43     newNode->parent = parent;
44     parent->left = newNode;
45 }
46
47 // Menambahkan node baru sebagai anak kanan dari node tertentu
48 void addRightNode(char data, Node* parent) {
49     Node* newNode = new Node;
50     newNode->data = data;
51     newNode->parent = parent;
52     parent->right = newNode;
53 }
54
55 // Menghapus node dari pohon
56 void deleteNode(char data) {
57     // Implementasi logika penghapusan
58 }
59
60 // Pre-order Traversal
61 void preOrder(Node* node) {
62     if (node != nullptr) {
63         cout << node->data << " ";
64         preOrder(node->left);
65         preOrder(node->right);
66     }
67 }
68
69 // In-order Traversal
70 void inOrder(Node* node) {
71     if (node != nullptr) {
72         inOrder(node->left);
73         cout << node->data << " ";
74         inOrder(node->right);
75     }
76 }
77
78 // Post-order Traversal
79 void postOrder(Node* node) {
80     if (node != nullptr) {
81         postOrder(node->left);
82         postOrder(node->right);
83         cout << node->data << " ";
84     }
85 }
86
87 // Menampilkan informasi tentang node
88 void showNodeInfo(Node* node) {
89     if (node != nullptr) {
90         cout << "Node: " << node->data << ", Parent: ";
91         if (node->parent != nullptr) {
92             cout << node->parent->data << " ";
93         } else {
94             cout << "None" << " ";
95         }
96         cout << endl;
97     }
98 }
99
100 // Menu utama
101 int main() {
102     init();
103     char choice;
104     do {
105         cout << "Menu: 1. Buat Root, 2. Tambah Anak Kiri, 3. Tambah Anak Kanan, 4. Hapus Node, 5. Pre-order, 6. In-order, 7. Post-order, 8. Tampilkan Info, 9. Keluar" << endl;
106         choice = getch();
107         switch (choice) {
108             case '1': makeRoot('A'); break;
109             case '2': addLeftNode('B', root); break;
110             case '3': addRightNode('C', root); break;
111             case '4': deleteNode('D'); break;
112             case '5': preOrder(root); break;
113             case '6': inOrder(root); break;
114             case '7': postOrder(root); break;
115             case '8': showNodeInfo(root); break;
116             case '9': return 0;
117         }
118     } while (choice != '9');
119     return 0;
120 }

```

Terminal Output:

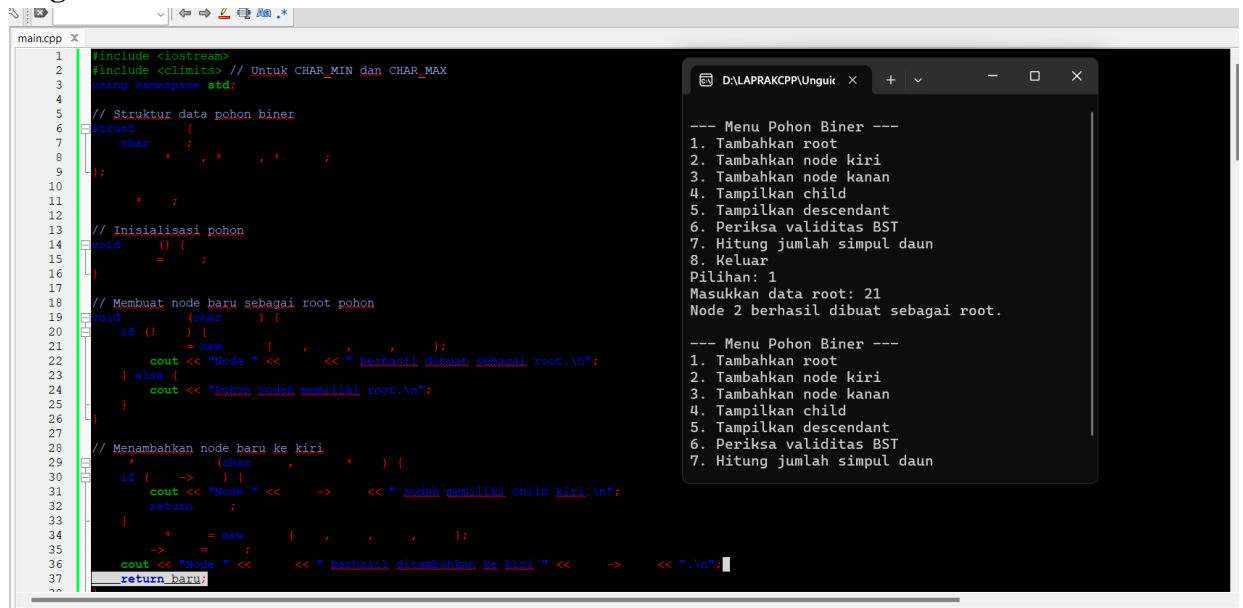
```

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 : 0.074 s
Press any key to continue.

```

4. Unguided



The screenshot displays a C++ IDE with a file named `main.cpp`. The code implements a Binary Search Tree (BST) with the following components:

- Includes:** `<iostream>` and `<limits>` for character range checks.
- Namespace:** `using namespace std;`
- Structure:** A `struct` for a binary node containing a `char` value and pointers to left and right children.
- Initialization:** A `void` function to initialize the root node.
- Node Creation:** A `void` function to create a new node and set it as the root.
- Node Insertion:** A `void` function to insert a new node as the left child of an existing node.

The terminal window on the right shows the program's execution:

```
--- Menu Pohon Biner ---
1. Tambahkan root
2. Tambahkan node kiri
3. Tambahkan node kanan
4. Tampilkan child
5. Tampilkan descendant
6. Periksa validitas BST
7. Hitung jumlah simpul daun
8. Keluar
Pilihan: 1
Masukkan data root: 21
Node 2 berhasil dibuat sebagai root.

--- Menu Pohon Biner ---
1. Tambahkan root
2. Tambahkan node kiri
3. Tambahkan node kanan
4. Tampilkan child
5. Tampilkan descendant
6. Periksa validitas BST
7. Hitung jumlah simpul daun
```

5. Kesimpulan

- **Pemahaman Rekursif:** Teknik rekursif membantu memecahkan masalah besar dengan membaginya menjadi bagian-bagian kecil yang lebih sederhana. Ini penting dalam pemrograman, terutama dalam bekerja dengan struktur data seperti pohon.
- **Traversal pada Pohon:** Traversal (penjelajahan) pohon seperti *pre-order*, *in-order*, dan *post-order* digunakan untuk mengunjungi setiap simpul pohon dalam urutan tertentu, tergantung kebutuhan.
- **Penerapan Binary Tree:** Laporan ini menunjukkan bagaimana struktur data Binary Tree dibuat dan dioperasikan, termasuk menambahkan, mencari, menghapus simpul, serta memastikan struktur Binary Search Tree (BST) valid.
- **Validasi BST:** Pohon biner memiliki aturan, yaitu semua nilai anak kiri harus lebih kecil dari induknya, dan semua nilai anak kanan harus lebih besar. Aturan ini membantu menjaga data tetap teratur.