LAPORAN PRAKTIKUM

PERTEMUAN 10

TREE BAGIAN DUA

Nama :

Resita Istania Purwanto (2311104037)

Dosen :

Yudha Islami Sulistya

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. GUIDED

Kode:

guided > tree.cpp > ...

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

/// PROGRAM BINARY TREE

// Deklarasi Pohon
Codeium: Refactor | Explain
struct Pohon {
    char data;
    Pohon *left, *right, *parent;
};

Pohon *root;

// Inisialisasi
Codeium: Refactor | Explain | ×
void init() {
    root = NULL;
}

// Cek Node
Codeium: Refactor | Explain | ×
bool isEmpty() {
    return root == NULL;
}

// Buat Node Baru
Codeium: Refactor | Explain | ×
void buatNode(char data) {
    if (isEmpty()) {
        root = new Pohon{data, NULL, NULL, NULL};
        cout << "\nNode " << data << " berhasil dibuat menjadi root." << endl;
    } else {
        cout << "\nPohon sudah dibuat." << endl;
    }
}

// Tambah Kiri
Codeium: Refactor | Explain | ×
Pohon *insertLeft(char data, Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    }
    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 Kanan
Codeium: Refactor | Explain | ×
Pohon *insertRight(char data, Pohon *node) {
    if (isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    }
    if (node->right != NULL) {
        cout << "\nNode " << node->data << " sudah ada child kanan!" << endl;
        return NULL;
    }
    Pohon *baru = new Pohon{data, NULL, NULL, node};
```

```
51  Pohon *insertRight(char data, Pohon *node) {
61      node->right = baru;
62      cout << "\nNode " << data << " berhasil ditambahkan ke child kanan " << node->data << endl;
63      return baru;
64  }
65
66  // Fungsi untuk Menghapus Node
    Codeium: Refactor | Explain | ×
67  void deleteNode(Pohon *&node) {
68      if (!node) return;
69
70      // Leaf node
71      if (!node->left && !node->right) {
72          delete node;
73          node = NULL;
74      }
75      // Node dengan satu child
76      else if (node->left && !node->right) {
77          Pohon *temp = node;
78          node = node->left;
79          node->parent = temp->parent;
80          delete temp;
81      } else if (!node->left && node->right) {
82          Pohon *temp = node;
83          node = node->right;
84          node->parent = temp->parent;
85          delete temp;
86      }
87      // Node dengan dua child
88      else {
89          // Ambil most-left dari right subtree
90          Pohon *successor = node->right;
91          while (successor->left) {
92              successor = successor->left;
```

```
93              }
94              node->data = successor->data;
95              deleteNode(successor);
96          }
97  }
98
99  // Cari Most-Left Node
    Codeium: Refactor | Explain | ×
100 Pohon *mostLeft(Pohon *node) {
101     if (!node) return NULL;
102     while (node->left) {
103         node = node->left;
104     }
105     return node;
106 }
107
108 // Cari Most-Right Node
    Codeium: Refactor | Explain | ×
109 Pohon *mostRight(Pohon *node) {
110     if (!node) return NULL;
111     while (node->right) {
112         node = node->right;
113     }
114     return node;
115 }
116
117 // Fungsi main
    Codeium: Refactor | Explain | ×
118 int main() {
119     init();
120     buatNode('A');
121     Pohon *nodeB = insertLeft('B', root);
122     Pohon *nodeC = insertRight('C', root);
```

```
118  int main() {
123      insertLeft('D', nodeB);
124      insertRight('E', nodeB);
125      insertLeft('F', nodeC);
126      insertRight('G', nodeC);
127
128      cout << "\n== Most-Left Node ==";
129      Pohon *left = mostLeft(root);
130      if (left) cout << "\nMost-Left Node: " << left->data << endl;
131
132      cout << "\n== Most-Right Node ==";
133      Pohon *right = mostRight(root);
134      if (right) cout << "\nMost-Right Node: " << right->data << endl;
135
136      cout << "\n== Hapus Node ==";
137      deleteNode(nodeB);  // Contoh menghapus node B
138      cout << "Node B telah dihapus." << endl;
139
140      return 0;
141  }
```

Output:

```
Node A berhasil dibuat menjadi root.

Node B berhasil ditambahkan ke child kiri A

Node C berhasil ditambahkan ke child kanan A

Node D berhasil ditambahkan ke child kiri B

Node E berhasil ditambahkan ke child kanan B

Node F berhasil ditambahkan ke child kiri C

Node G berhasil ditambahkan ke child kanan C

== Most-Left Node ==
Most-Left Node: D

== Most-Right Node ==
Most-Right Node: G

== Hapus Node ==Node B telah dihapus.
```

Penjelasan: Program ini mengimplementasikan binary tree menggunakan struktur Pohon yang memiliki atribut data, left, right, dan parent.

1. Membuat Root: Fungsi buatNode() membuat node root pertama kali.
2. Menambah Node: Fungsi insertLeft() dan insertRight() menambahkan node di kiri dan kanan suatu node.
3. Menghapus Node: Fungsi deleteNode() menghapus node dengan menangani berbagai kondisi (leaf, satu child, dua child).
4. Mencari Node Paling Kiri/Kanan: Fungsi mostLeft() dan mostRight() mencari node paling kiri dan kanan.

Fungsi main:

1. Membuat pohon dengan root 'A' dan menambahkan beberapa node.
2. Menampilkan node paling kiri dan kanan.
3. Menghapus node B.

Output: Menampilkan node paling kiri dan kanan, serta mengonfirmasi penghapusan node B.