LAPORAN PRAKTIKUM

PERTEMUAN 9

10&11_TREE

Nama :

Ilham Lii Assidaq (2311104068)

Dosen :

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

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. TUJUAN

1. Memahami konsep penggunaan fungsi rekursif.
2. Mengimplementasikan bentuk-bentuk fungsi rekursif.
3. Mengaplikasikan struktur data tree dalam sebuah kasus pemrograman.
4. Mengimplementasikan struktur data tree, khususnya Binary Tree.
5. Mengimplementasikan struktur data tree, khususnya Binary Tree.

II. TOOL

Dev C++

III. DASAR TEORI

Binary Search Tree (BST) adalah struktur data berbentuk hierarki yang terdiri dari node-node yang saling terhubung, di mana setiap node memiliki maksimal dua cabang (left dan right). BST memiliki karakteristik khusus, yaitu nilai pada subtree kiri selalu lebih kecil dibandingkan nilai node induk, sementara nilai pada subtree kanan selalu lebih besar. Sifat ini memungkinkan operasi seperti pencarian, penyisipan, dan penghapusan data dilakukan secara efisien, dengan kompleksitas waktu rata-rata O(log n) pada tree yang seimbang.

Rekursi adalah teknik pemrograman yang memungkinkan suatu fungsi untuk memanggil dirinya sendiri dalam menyelesaikan sub-masalah yang lebih kecil. Pada BST, rekursi sering digunakan untuk menjalankan operasi seperti penyisipan (insert), pencarian (search), dan traversal (pre-order, in-order, post-order). Agar fungsi rekursif berjalan dengan benar, diperlukan kondisi penghentian (base case) untuk mencegah loop tak terbatas, serta parameter yang dimodifikasi untuk mendekati kondisi penghentian tersebut.

Walaupun rekursi dapat menyederhanakan kode dan meningkatkan keterbacaan, pendekatan ini juga memiliki kekurangan, seperti konsumsi memori yang lebih tinggi dan potensi waktu eksekusi yang lebih lama dibandingkan metode iteratif. Traversal pada BST memungkinkan akses setiap node dalam urutan tertentu, yang berguna untuk berbagai keperluan, seperti menghitung jumlah node, menjumlahkan nilai, atau menentukan kedalaman tree. Dengan memahami prinsip-prinsip dasar ini, kita dapat lebih efektif dalam mengimplementasikan dan memanfaatkan struktur data tree serta teknik rekursi dalam pengembangan program.

IV. GUIDED

1. GUIDED 1

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

struct Pohon
{
    char data;
    Pohon *left, *right;
    Pohon *parent;
};

Pohon *root, *baru;

void init() {
    root = NULL;
}

bool isEmpty() {
    return root == NULL;
}

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;
    }
}

Pohon* insertLeft(char data, Pohon *node) {
    if (node->left != NULL) {
        cout << "\nNode " << node->data << "sudah ada child kiri" << endl;
        return NULL;
    }
    baru = new Pohon{data, NULL, NULL, node};
    node->left = baru;
    cout << "\nNode " << data << " berhasil ditambahkan ke child kiri" << node->data << endl;
    return baru;
}

Pohon* insertRight(char data, Pohon*node) {
    if (node->right != NULL) {
```

```
44.         cout << "\nNode" << node->data << " sudah ada child kanan!"
    << endl;
45.         return NULL;
46.     }
47.     baru = new Pohon{data, NULL, NULL, node};
48.     node->right = baru;
49.     cout << "\nNode " << data << " berhasil ditambahkan ke child
    kanan " << node->data << endl;
50.     return baru;
51. }
52.
53. void update(char data, Pohon *node) {
54.     if (!node) {
55.         cout << "\nNode yang ingin diubah tidak ditemukan!" <<
    endl;
56.         return;
57.     }
58.     char temp = node->data;
59.     node->data = data;
60.     cout << "\nNode " << temp << " berhasil diubah menjadi " <<
    data << endl;
61. }
62.
63. void find(char data, Pohon *node) {
64.     if (!node) return;
65.
66.     if(node->data == data) {
67.         cout << "\nNode ditemukan: " << data << endl;
68.         return;
69.     }
70.     find(data, node->left);
71.     find(data, node->right);
72. }
73.
74. void preOrder(Pohon *node) {
75.     if (node == NULL) return;
76.     cout << node->data << " ";
77.     preOrder(node->left);
78.     preOrder(node->right);
79. }
80.
81. void inOrder(Pohon *node) {
82.     if (!node) return;
83.     inOrder(node->left);
84.     cout << node->data << " ";
85.     inOrder(node->right);
86. }
87.
```

```
88. void postOrder(Pohon *node) {
89.     if (!node) return;
90.     postOrder(node->left);
91.     postOrder(node->right);
92.     cout << node->data << " ";
93. }
94. 
95. Pohon* deleteNode(Pohon *node, char data) {
96.     if (!node) return NULL;
97. 
98.     if (data < node->data) {
99.         node->left = deleteNode(node->left, data);
100.         }
101.         else if (data > node->data) {
102.             node->right = deleteNode(node->right, data);
103.         }
104.         else {
105.             if (!node->left) {
106.                 Pohon *temp = node->right;
107.                 delete node;
108.                 return temp;
109.             }
110.             else if (!node->right) {
111.                 Pohon *temp = node->left;
112.                 delete node;
113.                 return temp;
114.             }
115. 
116.             Pohon *successor = node->right;
117.             while (successor->left) successor = successor->left;
118.             node->data = successor->data;
119.             node->right = deleteNode(node->right, successor-
    >data);
120.         }
121.         return node;
122.     }
123. 
124.     Pohon* mostLeft(Pohon *node) {
125.         if (!node) return NULL;
126.         while (node->left) node = node->left;
127.         return node;
128.     }
129. 
130.     Pohon* mostRight(Pohon *node) {
131.         if (!node) return NULL;
132.         while (node->right) node = node->right;
133.         return node;
134.     }
```

```
135.
136.      int main() {
137.          init();
138.          buatNode('F');
139.          insertLeft('B', root);
140.          insertRight('G', root);
141.          insertLeft('A', root->left);
142.          insertRight('D', root->left);
143.          insertLeft('C', root->left->right);
144.          insertRight('E', root->left->right);
145.
146.          cout << "\nPre-order Traversal: ";
147.          preOrder(root);
148.          cout << "\nIn-order Traversal: ";
149.          inOrder(root);
150.          cout << "\nPost-order Traversal: ";
151.          postOrder(root);
152.
153.          cout << "\nMost Left Node: " << mostLeft(root)->data;
154.          cout << "\nMost Right Node: " << mostRight(root)->data;
155.
156.          cout << "\nMenghapus node D.";
157.          root = deleteNode(root, 'D');
158.          cout << "\nIn-order Traversal setelah penghapusan: ";
159.          inOrder(root);
160.
161.          return 0;
162.      }
```

V. UNGUIDED

1. UNGUIDED

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

struct Pohon {
    char data;
    int intData;
    Pohon *left, *right, *parent;
};

Pohon *root = NULL;

void init() {
    root = NULL;
}

bool isEmpty() {
    return root == NULL;
}

void buatNode(char data) {
    if (isEmpty()) {
        root = new Pohon{data, 0, NULL, NULL, NULL};
        cout << "Node " << data << " berhasil dibuat sebagai root.\n";
    } else {
        cout << "Root sudah ada.\n";
    }
}

Pohon* findNode(Pohon *node, char data) {
    if (!node) return NULL;
    if (node->data == data) return node;
    Pohon *leftResult = findNode(node->left, data);
    if (leftResult) return leftResult;
    return findNode(node->right, data);
}

Pohon* insertLeft(char data, Pohon *node) {
    if (node->left != NULL) {
        cout << "Node " << node->data << " sudah memiliki anak kiri.\n";
        return NULL;
    }
    Pohon *baru = new Pohon{data, 0, NULL, NULL, node};
    node->left = baru;
    cout << "Node " << data << " berhasil ditambahkan ke anak kiri " << node->data << ".\n";
```

```
    return baru;
}

Pohon* insertRight(char data, Pohon *node) {
    if (node->right != NULL) {
        cout << "Node " << node->data << " sudah memiliki anak kanan.\n";
        return NULL;
    }
    Pohon *baru = new Pohon{data, 0, NULL, NULL, node};
    node->right = baru;
    cout << "Node " << data << " berhasil ditambahkan ke anak kanan " << node->data << ".\n";
    return baru;
}

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

void tampilkanDescendants(Pohon *node) {
    if (!node) return;
    if (node->left || node->right) cout << node->data << ": ";
    if (node->left) cout << node->left->data << " ";
    if (node->right) cout << node->right->data << " ";
    cout << "\n";
    tampilkanDescendants(node->left);
    tampilkanDescendants(node->right);
}

bool is_valid_bst(Pohon* node, int min_val, int max_val) {
    if (!node) return true;
    if (node->intData <= min_val || node->intData >= max_val)
        return false;
    return is_valid_bst(node->left, min_val, node->intData) &&
           is_valid_bst(node->right, node->intData, max_val);
}

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);
```

```
}

void menu() {
    int pilihan;
    char data, parent_data;
      Pohon *parent_node;

     do {
         cout << "\nMenu Binary Tree:";
         cout << "\n1. Add Root";
         cout << "\n2. Tambahkan Anak Kiri";
         cout << "\n3. Tambahkan Anak Kanan";
         cout << "\n4. Tampilkan Child";
         cout << "\n5. Tampilkan Descendants";
         cout << "\n6. Cek Validitas BST";
         cout << "\n7. Hitung Jumlah Simpul Daun";
         cout << "\n8. Keluar";
         cout << "\nPilih: ";
         cin >> pilihan;

         switch (pilihan) {
             case 1:
                 cout << "Masukkan data root: ";
                 cin >> data;
                 buatNode(data);
                 break;

             case 2:
                 cout << "Masukkan data parent: ";
                 cin >> parent_data;
                 cout << "Masukkan data anak kiri: ";
                 cin >> data;
                 parent_node = findNode(root, parent_data);
                 if (parent_node) insertLeft(data, parent_node);
                 else cout << "Node " << parent_data << " tidak
   ditemukan.\n";
                 break;

             case 3:
                 cout << "Masukkan data parent: ";
                 cin >> parent_data;
                 cout << "Masukkan data anak kanan: ";
                 cin >> data;
                 parent_node = findNode(root, parent_data);
                 if (parent_node) insertRight(data, parent_node);
                 else cout << "Node " << parent_data << " tidak
   ditemukan.\n";
                 break;

             case 4:
                 cout << "Masukkan data node: ";
```

```
                cin >> data;
                parent_node = findNode(root, data);
                tampilkanChild(parent_node);
                break;

            case 5:
                cout << "Masukkan data node: ";
                cin >> data;
                parent_node = findNode(root, data);
                tampilkanDescendants(parent_node);
                break;

            case 6:
                cout << "Masukkan data node: ";
                cin >> data;
                parent_node = findNode(root, data);
                cout << "Pohon valid BST: " <<
   (is_valid_bst(parent_node, INT_MIN, INT_MAX) ? "Ya" : "Tidak") <<
   endl;
                break;

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

            case 8:
                cout << "Keluar.\n";
                break;

            default:
                cout << "Pilihan tidak valid.\n";
        }
    } while (pilihan != 8);
}

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

VI. KESIMPULAN

Praktik ini bertujuan untuk memahami dan menerapkan konsep dasar pohon, terutama Binary Search Tree (BST), dengan pendekatan rekursif. Saya juga mencakup operasi fundamental seperti penambahan, penghapusan, dan traversal (pre-order, in-order, dan post-order), serta membahas konsep lanjutan seperti node paling kiri, node paling kanan, dan rekonstruksi

struktur pohon.

Saya juga dilatih untuk membuat Abstract Data Type (ADT) untuk BST menggunakan linked list. Praktik ini menekankan pentingnya efisiensi logika rekursif dalam menyelesaikan masalah non-linear. Selain itu, kegiatan ini mendorong penerapan teori ke dalam konteks nyata dalam pemrograman.

4o mini