LAPORAN PRAKTIKUM

PERTEMUAN 9

Pengenalan C++ : TREE

Nama :

Ahmad Uffi Lestari Ma’ruf (2311104015)

Dosen :

YUDHA ISLAMI SULISTYA, S.Kom., M.Kom.

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

1. **Guided**

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

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

    Pohon(char d, Pohon *l = NULL, Pohon *r = NULL, Pohon *p = NULL)
        : data(d), left(l), right(r), parent(p) {}
};

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(isEmpty()) {
    cout << "\nBuat tree terlebih dahul!"<< endl;
    return NULL;
    }
    if (node->left != NULL) {
        cout << "\n Node " << node->data << " sudah ada child di kiri "<<
endl;
        return NULL;
    }
    baru = new Pohon(data, NULL, NULL, node);
    node ->left = baru;
    cout << "\nNode " << data << " berhasil ditambahkan di kiri " << node-
>data << endl;
    return baru;
}

```

```
47. Pohon *insertRight(char data, Pohon *node) {
48.     if (isEmpty()) {
49.         cout << "\nBuat tree terlebih dahulu!" << endl;
50.         return NULL;
51.     }
52.     if (node->right != NULL) {
53.         cout << "\nNode " << node->data << " Sudah ada child dikanan!" <<
    endl;
54.         return NULL;
55.     }
56.     baru = new Pohon(data, NULL, NULL, node);
57.     node->right = baru;
58.     cout << "\nNode " << data << " Berhasil ditambahkan ke child kanan " <<
    node->data << endl;
59.     return baru;
60. }
61.
62. void update(char data, Pohon *node) {
63.     if(isEmpty()) {
64.         cout << "\nBuat tree terlebih dahulu!" << endl;
65.         return;
66.     }
67.     if (!node) {
68.         cout << "\nNode yang diinginkan tidak ada!" << endl;
69.         return;
70.     }
71.     char temp = node->data;
72.     cout << "\nNode " << temp << " berhasil diubah menjadi " << data << endl;
73.
74. }
75.
76. void retrieve(Pohon *node) {
77.     if(isEmpty()) {
78.         cout << "\nBuat tree terlebih dahulu!" << endl;
79.         return;
80.     }
81.     if(!node) {
82.         cout << "\nNode yang diinginkan tidak ada!" << endl;
83.         return;
84.     }
85.     cout << "\nData node: " << node->data << endl;
86. }
87.
88. void find(Pohon *node) {
89.     if(isEmpty()) {
90.         cout << "\nBuat tree terlebih dahulu!" << endl;
91.         return;
92.     }
```

```
93.     if(!node) {
94.         cout << "\nNode yang diinginkan tidak ada!" << endl;
95.         return;
96.     }
97.     cout << "\nData node: " << node->data << endl;
98.     cout << "Root: " << root->data << endl;
99.     cout << "Parent: " << (node->parent ? node->parent->data : 'Tidak ada
    parent') << endl;
100.
101.      if(node->parent) {
102.          if(node->parent->left == node && node->parent->right){
103.              cout << "Sibling: " << node->parent->right->data << endl;
104.          } else if (node->parent->right == node && node->parent->left) {
105.              cout << "Sibling: " << node->parent->left->data << endl;
106.          } else {
107.              cout << "Sibling: Tidak ada sibling" << endl;
108.          }
109.      }
110.  }
111.
112.  int main() {
113.      init();
114.      buatNode('A');
115.      Pohon *nodeB = insertLeft('B', root);
116.      Pohon *nodeC = insertRight('C', root);
117.      insertLeft('D', nodeB);
118.      insertRight('E', nodeB);
119.      insertLeft('F', nodeC);
120.      insertLeft('G', nodeC);
121.
122.      cout << "\n== Pemanggila Retrieve ==" << endl;
123.      retrieve(root);
124.      retrieve(nodeB);
125.      retrieve(nodeC);
126.
127.      cout << "\n== Pemanggilan Find (Node B) ==" << endl;
128.      find(nodeB);
129.
130.      cout << "\n== Pemanggilan Find (Node C) ==" << endl;
131.      find(nodeC);
132.
133.      cout << "\n== Pemanggilan Update (Mengubah Node B menjadi Z)";
134.      update('Z', nodeB);
135.
136.      cout << "\n== Pemanggilan Retrieve setelah Update";
137.      retrieve(nodeB);
138.
139.      return 0;
```

```
140.  }
```

Hasil output :

```
Node A berhasil dibuat menjadi root

Node B berhasil ditambahkan di kiri A

Node C Berhasil ditambahkan ke child kanan A

Node D berhasil ditambahkan di kiri B

Node E Berhasil ditambahkan ke child kanan B

Node F berhasil ditambahkan di kiri C

 Node C sudah ada child di kiri

== Pemanggila Retrieve ==

Data node: A

Data node: B

Data node: C

== Pemanggilan Find (Node B) ==

Data node: B
Root: A
Parent: 65
Sibling: C

== Pemanggilan Find (Node C) ==

Data node: C
Root: A
Parent: 65
Sibling: B

== Pemanggilan Update (Mengubah Node B menjadi Z)
Node B berhasil diubah menjadi Z

== Pemanggilan Retrieve setelah Update
Data node: B
```

2. Unguided

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

struct Pohon {
    int data;  // Changed to int for BST validation
    Pohon *left, *right, *parent;

    Pohon(int d, Pohon *l = NULL, Pohon *r = NULL, Pohon *p = NULL)
        : data(d), left(l), right(r), parent(p) {}
};

Pohon *root, *baru;

void init() {
    root = NULL;
}

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

void buatNode(int 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(int data, Pohon *node) {
    if(isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    }
    if (node->left != NULL) {
        cout << "\nNode " << node->data << " sudah ada child di kiri" << endl;
        return NULL;
    }
    baru = new Pohon(data, NULL, NULL, node);
    node->left = baru;
    cout << "\nNode " << data << " berhasil ditambahkan di kiri " << node->data <<
endl;
    return baru;
}
```

```
Pohon *insertRight(int data, Pohon *node) {
    if(isEmpty()) {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    }
    if (node->right != NULL) {
        cout << "\nNode " << node->data << " sudah ada child di kanan" << endl;
        return NULL;
    }
    baru = new Pohon(data, NULL, NULL, node);
    node->right = baru;
    cout << "\nNode " << data << " berhasil ditambahkan di kanan " << node->data
<< endl;
    return baru;
}

// New function to display child nodes
void tampilkanChild(Pohon *node) {
    if (!node) {
        cout << "Node tidak ditemukan!" << endl;
        return;
    }

    cout << "Node " << node->data << " memiliki:" << endl;
    if (node->left)
        cout << "- Child kiri: " << node->left->data << endl;
    else
        cout << "- Tidak ada child kiri" << endl;

    if (node->right)
        cout << "- Child kanan: " << node->right->data << endl;
    else
        cout << "- Tidak ada child kanan" << endl;
}

// New function to display descendants recursively
void tampilkanDescendant(Pohon *node, int level = 0) {
    if (!node) return;

    if (level > 0) {
        for (int i = 0; i < level; i++) cout << "  ";
        cout << "└ " << node->data << endl;
    }

    tampilkanDescendant(node->left, level + 1);
    tampilkanDescendant(node->right, level + 1);
}
```

```cpp
// New function to validate BST
bool is_valid_bst(Pohon *node, int min_val = INT_MIN, int max_val = INT_MAX) {
    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);
}

// New function to count leaf nodes
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);
}

// Function to find node by value
Pohon* findNode(Pohon* node, int value) {
    if (!node) return NULL;
    if (node->data == value) return node;

    Pohon* leftResult = findNode(node->left, value);
    if (leftResult) return leftResult;

    return findNode(node->right, value);
}

int main() {
    init();
    int choice, data, parentValue;
    Pohon* selectedNode;

    while (true) {
        cout << "\n=== MENU PROGRAM BINARY TREE ===" << endl;
        cout << "1. Buat Root" << endl;
        cout << "2. Tambah Node Kiri" << endl;
        cout << "3. Tambah Node Kanan" << endl;
        cout << "4. Tampilkan Child dari Node" << endl;
        cout << "5. Tampilkan Descendant dari Node" << endl;
        cout << "6. Cek Validitas BST" << endl;
        cout << "7. Hitung Jumlah Daun" << endl;
        cout << "8. Keluar" << endl;
        cout << "Pilihan: ";
```

```
        cin >> choice;

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

            case 2:
                if(isEmpty()) {
                    cout << "Buat root terlebih dahulu!" << endl;
                    break;
                }
                cout << "Masukkan nilai node: ";
                cin >> data;
                cout << "Masukkan nilai parent: ";
                cin >> parentValue;
                selectedNode = findNode(root, parentValue);
                if(selectedNode)
                    insertLeft(data, selectedNode);
                else
                    cout << "Parent node tidak ditemukan!" << endl;
                break;

            case 3:
                if(isEmpty()) {
                    cout << "Buat root terlebih dahulu!" << endl;
                    break;
                }
                cout << "Masukkan nilai node: ";
                cin >> data;
                cout << "Masukkan nilai parent: ";
                cin >> parentValue;
                selectedNode = findNode(root, parentValue);
                if(selectedNode)
                    insertRight(data, selectedNode);
                else
                    cout << "Parent node tidak ditemukan!" << endl;
                break;

            case 4:
                if(isEmpty()) {
                    cout << "Pohon masih kosong!" << endl;
                    break;
                }
                cout << "Masukkan nilai node yang ingin dilihat child-nya: ";
                cin >> data;
```

```
            selectedNode = findNode(root, data);
            if(selectedNode)
                tampilkanChild(selectedNode);
            else
                cout << "Node tidak ditemukan!" << endl;
            break;

        case 5:
            if(isEmpty()) {
                cout << "Pohon masih kosong!" << endl;
                break;
            }
            cout << "Masukkan nilai node yang ingin dilihat descendant-nya: ";
            cin >> data;
            selectedNode = findNode(root, data);
            if(selectedNode) {
                cout << "Descendants dari node " << data << ":" << endl;
                tampilkanDescendant(selectedNode);
            } else
                cout << "Node tidak ditemukan!" << endl;
            break;

        case 6:
            if(isEmpty()) {
                cout << "Pohon masih kosong!" << endl;
                break;
            }
            if(is_valid_bst(root))
                cout << "Pohon adalah Binary Search Tree yang valid" << endl;
            else
                cout << "Pohon BUKAN Binary Search Tree yang valid" << endl;
            break;

        case 7:
            if(isEmpty()) {
                cout << "Pohon masih kosong!" << endl;
                break;
            }
            cout << "Jumlah simpul daun: " << cari_simpul_daun(root) << endl;
            break;

        case 8:
            cout << "Program selesai." << endl;
            return 0;

        default:
            cout << "Pilihan tidak valid!" << endl;
    }
```

```
    }

    return 0;
}
```

output yang dihasilakan :

```
=== MENU PROGRAM BINARY TREE ===
1. Buat Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Child dari Node
5. Tampilkan Descendant dari Node
6. Cek Validitas BST
7. Hitung Jumlah Daun
8. Keluar
Pilihan: 1
Masukkan nilai root: 3

Node 3 berhasil dibuat menjadi root

=== MENU PROGRAM BINARY TREE ===
1. Buat Root
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Tampilkan Child dari Node
5. Tampilkan Descendant dari Node
6. Cek Validitas BST
7. Hitung Jumlah Daun
8. Keluar
Pilihan: 8
Program selesai.
```