LAPORAN PRAKTIKUM

PERTEMUAN 9

TREE

Nama :

Resita Istania Purwanto (2311104037)

Dosen :

Yudha Islami Sulistya

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. UNGUIDED

1. Kode binary:

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

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

Codeium: Refactor | Explain
class BinaryTree {
private:
    Pohon* root;
    Pohon* baru;

public:
    BinaryTree() : root(NULL), baru(NULL) {}

    Codeium: Refactor | Explain | Generate Function Comment | ×
    bool isEmpty() {
        return root == NULL;
    }

    Codeium: Refactor | Explain | Generate Function Comment | ×
    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;
        }
    }

    Codeium: Refactor | Explain | Generate Function Comment | ×
```

```cpp
        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;
        }
        baru = new Pohon{data, NULL, NULL, node};
        node->left = baru;
        cout << "\nNode " << data << " berhasil ditambahkan ke child kiri " << node->data << endl;
        return baru;
    }

    Codeium: Refactor | Explain | Generate Function Comment | ×
    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;
        }
        baru = new Pohon{data, NULL, NULL, node};
        node->right = baru;
        cout << "\nNode " << data << " berhasil ditambahkan ke child kanan " << node->data << endl;
        return baru;
    }

    Codeium: Refactor | Explain | Generate Function Comment | ×
    void update(char data, Pohon* node) {
```

```
61 >        if (isEmpty()) {···
65          if (!node) {
66              cout << "\nNode yang ingin diganti tidak ada!" << endl;
67              return;
68          }
69          char temp = node->data;
70          node->data = data;
71          cout << "\nNode " << temp << " berhasil diubah menjadi " << data << endl;
72      }
73
            Codeium: Refactor | Explain | Generate Function Comment | ×
74      void retrieve(Pohon* node) {
75          if (isEmpty()) {
76              cout << "\nBuat tree terlebih dahulu!" << endl;
77              return;
78          }
79          if (!node) {
80              cout << "\nNode yang ditunjuk tidak ada!" << endl;
81              return;
82          }
83          cout << "\nData node: " << node->data << endl;
84      }
85
            Codeium: Refactor | Explain | Generate Function Comment | ×
86      void showChildren(Pohon* node) {
87          if (!node) {
88              cout << "\nNode tidak ditemukan!" << endl;
89              return;
90          }
91
92          cout << "\nChild dari Node " << node->data << ": ";
```

```
93              if (node->left) cout << node->left->data << " ";
94              if (node->right) cout << node->right->data << " ";
95              cout << endl;
96
97              cout << "Descendants dari Node " << node->data << ": ";
98              if (node->left) {
99                  cout << node->left->data << " ";
100                 showChildren(node->left);
101             }
102             if (node->right) {
103                 cout << node->right->data << " ";
104                 showChildren(node->right);
105             }
106     }
107
            Codeium: Refactor | Explain | Generate Function Comment | ×
108     Pohon* getRoot() {
109         return root;
110     }
111 };
```

main.cpp

```cpp
#include <iostream>
#include "binary_tree.cpp"  // Include file binary_tree.cpp
using namespace std;

Codeium: Refactor | Explain | Generate Function Comment | X
int main() {
    BinaryTree tree;
    char data, parent;
    int pilihan;

    do {
        cout << "\nMenu:\n";
        cout << "1. Buat Pohon\n";
        cout << "2. Tambah Node Kiri\n";
        cout << "3. Tambah Node Kanan\n";
        cout << "4. Update Node\n";
        cout << "5. Tampilkan Node\n";
        cout << "6. Tampilkan Child dan Descendant\n";
        cout << "0. Keluar\n";
        cout << "Pilih menu: ";
        cin >> pilihan;

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

        case 2:
            cout << "Masukkan data untuk child kiri dan parent node: ";
            cin >> data >> parent;
            {
                Pohon* parentNode = tree.getRoot();
                if (parentNode) {
                    if (parentNode->data == parent) {
                        tree.insertLeft(data, parentNode);
                    } else {
                        cout << "Parent node tidak ditemukan.\n";
                    }
                } else {
                    cout << "Tree kosong, buat pohon terlebih dahulu.\n";
                }
            }
            break;

        case 3:
            cout << "Masukkan data untuk child kanan dan parent node: ";
            cin >> data >> parent;
            {
                Pohon* parentNode = tree.getRoot();
                if (parentNode) {
                    if (parentNode->data == parent) {
                        tree.insertRight(data, parentNode);
                    } else {
                        cout << "Parent node tidak ditemukan.\n";
                    }
                } else {
                    cout << "Tree kosong, buat pohon terlebih dahulu.\n";
                }
            }
            break;
```

```
        case 4:
            cout << "Masukkan data baru dan node yang ingin diubah: ";
            cin >> data >> parent;
            {
                Pohon* parentNode = tree.getRoot();
                if (parentNode) {
                    if (parentNode->data == parent) {
                        tree.update(data, parentNode);
                    } else {
                        cout << "Node tidak ditemukan.\n";
                    }
                } else {
                    cout << "Tree kosong, buat pohon terlebih dahulu.\n";
                }
            }
            break;

        case 5:
            cout << "Masukkan node yang ingin dilihat: ";
            cin >> parent;
            {
                Pohon* parentNode = tree.getRoot();
                if (parentNode) {
                    if (parentNode->data == parent) {
                        tree.retrieve(parentNode);
                    } else {
                        cout << "Node tidak ditemukan.\n";
                    }
                } else {
                    cout << "Tree kosong, buat pohon terlebih dahulu.\n";
```

```
                    }
                }
                break;

            case 6:
                cout << "Masukkan node yang ingin ditampilkan child dan descendant-nya: ";
                cin >> parent;
                {
                    Pohon* parentNode = tree.getRoot();
                    if (parentNode) {
                        if (parentNode->data == parent) {
                            tree.showChildren(parentNode);
                        } else {
                            cout << "Node tidak ditemukan.\n";
                        }
                    } else {
                        cout << "Tree kosong, buat pohon terlebih dahulu.\n";
                    }
                }
                break;

            case 0:
                cout << "Keluar...\n";
                break;

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

```
        } while (pilihan != 0);

        return 0;
    }
```

output:

```
Menu:
1. Buat Pohon
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Update Node
5. Tampilkan Node
6. Tampilkan Child dan Descendant
0. Keluar
Pilih menu: 1
Masukkan data untuk root: A

Node A berhasil dibuat menjadi root.

Menu:
1. Buat Pohon
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Update Node
5. Tampilkan Node
6. Tampilkan Child dan Descendant
0. Keluar
Pilih menu: 2
Masukkan data untuk child kiri dan parent node: B A

Node B berhasil ditambahkan ke child kiri A

Menu:
1. Buat Pohon
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Update Node
5. Tampilkan Node
6. Tampilkan Child dan Descendant
```

```
0. Keluar
Pilih menu: 3
Masukkan data untuk child kanan dan parent node: C A

Node C berhasil ditambahkan ke child kanan A

Menu:
1. Buat Pohon
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Update Node
5. Tampilkan Node
6. Tampilkan Child dan Descendant
0. Keluar
Pilih menu: 4
Masukkan data baru dan node yang ingin diubah: R A

Node A berhasil diubah menjadi R

Menu:
1. Buat Pohon
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Update Node
5. Tampilkan Node
6. Tampilkan Child dan Descendant
0. Keluar
Pilih menu: 5
Masukkan node yang ingin dilihat: R

Data node: R

Menu:
1. Buat Pohon
```

```
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Update Node
5. Tampilkan Node
6. Tampilkan Child dan Descendant
0. Keluar
Pilih menu: 6
Masukkan node yang ingin ditampilkan child dan descendant-nya: R

Child dari Node R: B C
Descendants dari Node R: B
Child dari Node B:
Descendants dari Node B: C
Child dari Node C:
Descendants dari Node C:
Menu:
1. Buat Pohon
2. Tambah Node Kiri
3. Tambah Node Kanan
4. Update Node
5. Tampilkan Node
6. Tampilkan Child dan Descendant
0. Keluar
Pilih menu: 0
Keluar...
```

2. kode

```cpp
#include <iostream>
#include <limits.h>  // Untuk INT_MIN dan INT_MAX
using namespace std;

// Struktur Pohon
Codeium: Refactor | Explain
struct Pohon {
    int data;
    Pohon *left, *right;
};

// Fungsi rekursif untuk memeriksa apakah pohon adalah BST
Codeium: Refactor | Explain | ×
bool is_valid_bst(Pohon* node, long min_val, long max_val) {
    if (node == nullptr) return true;  // Basis: Pohon kosong adalah BST

    // Periksa apakah data node berada dalam rentang yang valid
    if (node->data <= min_val || node->data >= max_val) {
        return false;
    }

    // Rekursi untuk subtree kiri dan kanan dengan pembaruan batas
    return is_valid_bst(node->left, min_val, node->data) &&
           is_valid_bst(node->right, node->data, max_val);
}

// Fungsi untuk membuat node baru
Codeium: Refactor | Explain | ×
Pohon* newNode(int data) {
    Pohon* node = new Pohon;
    node->data = data;
    node->left = node->right = nullptr;
    return node;
}
```

```cpp
// Fungsi untuk menguji is_valid_bst
Codeium: Refactor | Explain | ×
void test_is_valid_bst() {
    // Membuat pohon valid sebagai BST
    Pohon* root = newNode(10);
    root->left = newNode(5);
    root->right = newNode(20);
    root->left->left = newNode(3);
    root->left->right = newNode(7);
    root->right->left = newNode(15);
    root->right->right = newNode(30);

    cout << "Pohon valid sebagai BST? " << (is_valid_bst(root, LONG_MIN, LONG_MAX) ? "Ya" : "Tidak") << endl;

    // Membuat pohon tidak valid
    Pohon* invalidRoot = newNode(10);
    invalidRoot->left = newNode(5);
    invalidRoot->right = newNode(20);
    invalidRoot->left->right = newNode(25);  // Ini melanggar BST karena 25 > 10

    cout << "Pohon valid sebagai BST? " << (is_valid_bst(invalidRoot, LONG_MIN, LONG_MAX) ? "Ya" : "Tidak") << endl;
}

Codeium: Refactor | Explain | Generate Function Comment | ×
int main() {
    test_is_valid_bst();
    return 0;
}
```

output:

```
Pohon valid sebagai BST? Ya
Pohon valid sebagai BST? Tidak
```

penjelasan:

- Struktur Pohon: Setiap node menyimpan nilai data, serta pointer ke anak kiri dan kanan.
- Fungsi is_valid_bst: Memeriksa apakah pohon memenuhi aturan BST, yaitu setiap node di subtree kiri lebih kecil dari root, dan di subtree kanan lebih besar.
- Fungsi newNode: Membuat node baru.
- Fungsi test_is_valid_bst: Membuat dua pohon, satu valid dan satu tidak valid, dan memeriksa keduanya menggunakan is_valid_bst
- Output: Program menguji dua pohon dan mencetak apakah mereka valid sebagai BST atau tidak.

3. kode

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

// Struktur Pohon
Codeium: Refactor | Explain
struct Pohon {
    int data;
    Pohon *left, *right;
};

// Fungsi rekursif untuk menghitung jumlah simpul daun
Codeium: Refactor | Explain | ×
int cari_simpul_daun(Pohon* node) {
    if (node == nullptr) return 0;  // Jika node kosong, tidak ada simpul daun

    // Jika node tidak memiliki anak kiri dan kanan, maka itu adalah simpul daun
    if (node->left == nullptr && node->right == nullptr) {
        return 1;
    }

    // Rekursi untuk menghitung simpul daun di subtree kiri dan kanan
    return cari_simpul_daun(node->left) + cari_simpul_daun(node->right);
}

// Fungsi untuk membuat node baru
Codeium: Refactor | Explain | ×
Pohon* newNode(int data) {
    Pohon* node = new Pohon;
    node->data = data;
    node->left = node->right = nullptr;
    return node;
}

// Fungsi untuk menguji cari_simpul_daun
void test_cari_simpul_daun() {
    // Membuat pohon contoh
    Pohon* root = newNode(10);
    root->left = newNode(5);
    root->right = newNode(20);
    root->left->left = newNode(3);
    root->left->right = newNode(7);
    root->right->left = newNode(15);
    root->right->right = newNode(30);

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

    // Pohon hanya memiliki satu simpul daun
    Pohon* singleLeaf = newNode(5);
    singleLeaf->left = newNode(3);
    singleLeaf->right = newNode(7);
    cout << "Jumlah simpul daun (singleLeaf): " << cari_simpul_daun(singleLeaf) << endl;
}

Codeium: Refactor | Explain | Generate Function Comment | ×
int main() {
    test_cari_simpul_daun();
    return 0;
}
```

output:

```
Jumlah simpul daun: 4
Jumlah simpul daun (singleLeaf): 2
```

penjelasan:

- Struktur Pohon: Mewakili pohon dengan node yang memiliki nilai dan dua anak (kiri dan kanan).
- Fungsi cari_simpul_daun: Menghitung jumlah simpul daun dengan cara rekursif:
    - Jika node kosong, kembalikan 0.
    - Jika node adalah daun (tanpa anak), kembalikan 1.
    - Jika tidak, hitung simpul daun pada subtree kiri dan kanan.
- Fungsi newNode: Membuat node baru dengan nilai yang diberikan.
- Fungsi test_cari_simpul_daun: Menguji dua pohon dengan menghitung simpul daunnya.
- Output: Program mencetak jumlah simpul daun dari pohon yang diuji.