LAPORAN PRAKTIKUM

- **Identitas Praktikum**

  Nama MK : Struktur Data

  Kode MK : CCK2AAB4

  Bobot SKS : 4 SKS

  Tempat : L-Program, Gedung DC, lantai 3

  Hari, tanggal : Selasa, 3 Desember 2024

  Jam : 12:30-15:30 WIB

  Topik praktikum : Modul-11 TREE (bagian kedua)

- **Identitas Mahasiswa**

  Nama lengkap : Afad Fath Musyarof Halim

  NIM : 2211104030

  Program Studi : S-1 Software Engineering

- **Hasil Praktikum**

11. TREE

11.1 DELETE

Menghapus node tertentu pada binary tree

- Koding
  - Fungsi delete

```cpp
// Fungsi untuk menghapus Node dari Binary Tree
Node* deleteNode(Node* root, int key) {
    if (root == nullptr) {
        return root;
    }

    // Menavigasi ke node yang ingin dihapus
    if (key < root->data) {
        root->left = deleteNode(root->left, key);
    } else if (key > root->data) {
        root->right = deleteNode(root->right, key);
    } else {
        // Node ditemukan

        // Kasus 1: Node tanpa anak
        if (root->left == nullptr && root->right == nullptr) {
            delete root;
            return nullptr;
        }

        // Kasus 2: Node dengan satu anak
        if (root->left == nullptr) {
            Node* temp = root->right;
            delete root;
            return temp;
        }
        if (root->right == nullptr) {
            Node* temp = root->left;
            delete root;
            return temp;
        }

        // Kasus 3: Node dengan dua anak
        Node* temp = findMin(root->right); // Cari pengganti dari subtree kanan
        root->data = temp->data;          // Ganti nilai root dengan nilai pengganti
        root->right = deleteNode(root->right, temp->data); // Hapus pengganti
    }

    return root;
}
```

- Main.cpp

```cpp
int main() {
    Node* root = nullptr;

    // Masukkan nilai ke Binary Tree
    root = insert(root, 20);
    insert(root, 10);
    insert(root, 5);
    insert(root, 18);
    insert(root, 35);

    cout << "Binary Tree Inorder Awal: ";
    inorderTraversal(root);
    cout << endl;

    int choice;
    do {
        cout << "\nMenu:\n";
        cout << "1. Tampilkan Binary Tree (Inorder)\n";
        cout << "2. Hapus Node\n";
        cout << "3. Keluar\n";
        cout << "Pilihan Anda: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Binary Tree Inorder: ";
                inorderTraversal(root);
                cout << endl;
                break;

            case 2: {
                int valueToDelete;
                cout << "Masukkan nilai node yang ingin dihapus: ";
                cin >> valueToDelete;
                root = deleteNode(root, valueToDelete);
                cout << "Node " << valueToDelete << " telah dihapus.\n";
                break;
            }

            case 3:
                cout << "Keluar dari program.\n";
                break;

            default:
                cout << "Pilihan tidak valid. Coba lagi.\n";
        }
    } while (choice != 3);

    return 0;
}
```

- Output

```
# & .\'tree_a.exe'
Binary Tree Inorder Awal: 5 10 18 20 35

Menu:
1. Tampilkan Binary Tree (Inorder)
2. Hapus Node
3. Keluar
Pilihan Anda: 1
Binary Tree Inorder: 5 10 18 20 35

Menu:
1. Tampilkan Binary Tree (Inorder)
2. Hapus Node
3. Keluar
Pilihan Anda: 2
Masukkan nilai node yang ingin dihapus: 18
Node 18 telah dihapus.

Menu:
1. Tampilkan Binary Tree (Inorder)
2. Hapus Node
3. Keluar
Pilihan Anda: 1
Binary Tree Inorder: 5 10 20 35

Menu:
1. Tampilkan Binary Tree (Inorder)
2. Hapus Node
3. Keluar
Pilihan Anda: 3
Keluar dari program.
Afadfath | output                    main
#
```

- Penjelasan fungsi deleteNode
    1. Mendapatkan node yang ingin dihapus
    2. Jika node tidak memiliki turunan akan langsung dihapus
    3. Jika node memiliki 1 turunan maka turunan tersebut menggantikan posisi node
    4. Jika node memiliki 2 turunan makan turunan yang paling besar akan menggantikan posisi node

11.2 MOST LEFT

Menampilkan nilai terkecil (paling kiri) pada binary tree

- Koding
    - Fungsi

```
Node* findMostLeftNode(Node* root) {
    if (root == nullptr) {
        return nullptr; // BST kosong
    }
    Node* current = root;
    while (current->left != nullptr) {
        current = current->left;
    }
    return current;
}
```

    - Main.cpp

```
// Fungsi utama
int main() {
    Node* root = nullptr;

    // Memasukkan nilai ke dalam BST
    root = insert(root, 1);
    root = insert(root, 3);
    root = insert(root, 5);
    root = insert(root, 10);
    root = insert(root, 20);
    root = insert(root, 7);

    // Mencari most-left node
    Node* mostLeft = findMostLeftNode(root);

    if (mostLeft != nullptr) {
        cout << "Most-left tree adalah = " << mostLeft->data << endl;
    } else {
        cout << "Tree kosong." << endl;
    }

    return 0;
}
```

- Output

```
Afadfath | output                    main
# & .\'tree_b.exe'
Most-left tree adalah = 1
Afadfath | output                    main
#
```

- Penjelasan

  fungsi akan mengecek apakah node memiliki turunan ke kiri (lebih kecil dari node), jika iya posisi turun ke posisi turunan. Lalu di ulang sampai tidak menemukan node yang tidak memliki turunan kiri

11.3 MOST RIGHT

Menampilkan nilai terbesar (paling kanan) pada binary tree

- Koding
  - Fungsi

```cpp
// Fungsi untuk mencari node paling kanan (most-right)
Node* findMostRight(Node* root) {
    if (root == nullptr) {
        return nullptr;
    }
    Node* current = root;
    while (current->right != nullptr) {
        current = current->right;
    }
    return current;
}
```

  - Main.cpp

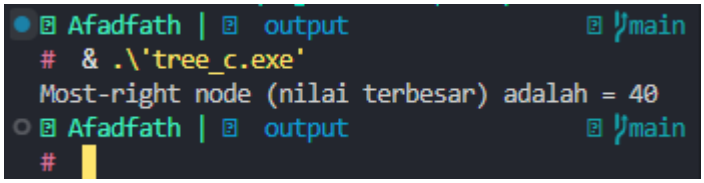
```cpp
// Fungsi utama
int main() {
    Node* root = nullptr;

    // Input data untuk BST
    root = insert(root, 5);
    root = insert(root, 10);
    root = insert(root, 20);
    root = insert(root, 15);
    root = insert(root, 30);
    root = insert(root, 40);

    // Cari most-right node
    Node* mostRight = findMostRight(root);
    if (mostRight != nullptr) {
        cout << "Most-right node (nilai terbesar) adalah = " << mostRight->data << endl;
    } else {
        cout << "Tree kosong." << endl;
    }

    return 0;
}
```

- Output



- Penjelasan

  fungsi memiliki algoritma sama dengan most left dengan perbedaan arah bukan turunan kiri namun kanan