Nama : Nabiilah Nur Fauziyyah

NPM : 2310631170105

Kelas : 2C – Informatika

Tugas Struktur Data

Link Github : https://github.com/fauziyyah22/NabiilahNF_SD_Tugas7

1. Buatlah laporan dari source code pada link di bawah ini :
   Mengukur Maximal Depth Pada Binary Tree CPP Program

```cpp
#include <iostream>
#include <string>

using namespace std;

// Fungsi untuk menampilkan identitas
void identitas() {
    cout << "\nProgram Mengukur Maksimal Depth pada Binary Tree CPP Program";
    cout << "\nNama      : Nabiilah Nur Fauziyyah";
    cout << "\nNPM       : 2310631170105";
    cout << "\nKelas     : 2C - Informatika" << "\n";
}

// Merepresentasikan setiap simpul dalam pohon
struct Node {
    char label;
    Node *right, *left, *parent;
};

// root adalah pointer yang menunjuk ke simpul akar dari pohon
Node *root = NULL;

// Fungsi ini memeriksa apakah pohon kosong, yaitu jika root bernilai NULL
bool emptyTree() {
    return root == NULL;
}

// Fungsi ini membuat simpul root dari pohon dengan label yang diberikan
void createTree(char label) {
    if (root) { // Jika pohon sudah ada, fungsi akan memberikan pesan bahwa pohon sudah dibuat
        cout << "Tree sudah dibuat" << endl;
    } else { // Jika pohon belum ada, simpul root akan dibuat dan diinisialisasi dengan label yang diberikan
        root = new Node();
        root->label = label;
        root->left = NULL;
        root->right = NULL;
        root->parent = NULL;
        cout << label << " Berhasil menjadi root" << endl;
    }
}

// Fungsi ini mencari simpul dengan label tertantu dalam pohon
Node* search(Node* node, char label) { // Menerima simpul saat ini sebagai argumen
    if (!node) { // Jika simpul dengan label yang dicari ditemukan, fungsi akan mengembalikan pointer ke simpul tersebut
        return NULL;
    }
    if (node->label == label) { // Jika tidak ditemukan, fungsi akan mengembalikan NULL
        return node;
    }
    // Pencarian dilakukan secara rekursif di subtree kiri dan kanan
    Node* leftSearch = search(node->left, label);
    if (leftSearch) {
        return leftSearch;
    }
    return search(node->right, label);
}
```

```cpp
// Fungsi ini digunakan untuk memasukkan simpul baru ke dalam pohon
Node* insert(char label, char parentLabel, string child) {
    if (emptyTree()) {
        cout << "Tree masih kosong, Tolong dibuat dulu!" << endl;
        return NULL;
    }

    Node* parentNode = search(root, parentLabel);
    if (!parentNode) {
        cout << "Parent node tidak ditemukan!" << endl;
        return NULL;
    }
    // Jika semua syarat terpenuhi, simpul  baru akan dibuat dan ditambahkan
    // ke pohon sebagai anak kiri atau kanan dari simpul induk
    if (child == "left") {
        if (parentNode->left) {
            cout << "Anak bagian kiri dari node " << parentNode->label << " sudah ada isinya!" << endl;
        } else {
            Node* newNode = new Node();
            newNode->label = label;
            newNode->left = NULL;
            newNode->right = NULL;
            newNode->parent = parentNode;
            parentNode->left = newNode;
            cout << "Label " << label << " berhasil dibuat di anak kiri dari node " << parentNode->label << endl;
            return newNode;
        }
    } else if (child == "right") {
        if (parentNode->right) {
            cout << "Anak bagian kanan dari node " << parentNode->label << " sudah ada isinya!" << endl;
        } else {
            Node* newNode = new Node();
            newNode->label = label;
            newNode->left = NULL;
            newNode->right = NULL;
            newNode->parent = parentNode;
            parentNode->right = newNode;
            cout << "Label " << label << " berhasil dibuat di anak kanan dari node " << parentNode->label << endl;
            return newNode;
        }
    }
    return NULL;
}

// Fungsi ini digunakan untuk memperbarui label sebuah simpul dalam pohon
void updateLabel(char label, Node* node) {
    if (emptyTree()) {
        cout << "Buat tree terlebih dahulu!" << endl;
    } else {
        if (!node) {
            cout << "Tidak ada node ini atau masih kosong!" << endl;
        } else { // Jika simpul ditemukan, labelnya akan diperbarui dengan label baru yang diberikan
            cout << "Label sebelumnya = " << node->label << endl;
            node->label = label;
            cout << "Label setelahnya = " << node->label << endl;
        }
    }
}

// Fungsi ini digunakan untuk mencetak label sebuah simpul dalam pohon
void retrieveLabel(Node* node) {
    if (!node) {
        cout << "Tidak ada Node ini atau masih kosong" << endl;
    } else {
        cout << "Label dari node ini adalah " << node->label << endl;
    }
}
```

```
126 // Fungsi ini digunakan untuk menghapus subtree yang dimulai dari sebuah simpul
127 void deleteSub(Node* node) {
128     if (node != NULL) {
129         if (node != root && node == node->parent->left) {
130             node->parent->left = NULL;
131         }
132         deleteSub(node->left);
133         if (node != root && node == node->parent->right) {
134             node->parent->right = NULL;
135         }
136         deleteSub(node->right);
137         if (node == root) {
138             root = NULL;
139             delete root;
140         } else {
141             delete node;
142         }
143     }
144 }
145
```

```
146 // Fungsi ini digunakan untuk menghapus seluruh pohon
147 void clearTree(Node* node) {
148     if (node != NULL) {
149         if (node != root && node == node->parent->left) {
150             node->parent->left = NULL;
151         }
152         clearTree(node->left);
153         if (node != root && node == node->parent->right) {
154             node->parent->right = NULL;
155         }
156         clearTree(node->right);
157         if (node == root) {
158             root = NULL;
159             delete root;
160         } else {
161             delete node;
162         }
163     }
164 }
165
```

```cpp
// Fungsi - fungsi ini digunakan untuk melakukan traversal pada pohon
// Simpul saat ini, anak kiri, anak kanan
void preOrder(Node* node) {
    if (!node) {
        return;
    } else {
        cout << node->label << " ";
        preOrder(node->left);
        preOrder(node->right);
    }
}
// Anak kiri, anak kanan, simpul saat ini
void postOrder(Node* node) {
    if (!node) {
        return;
    } else {
        postOrder(node->left);
        postOrder(node->right);
        cout << node->label << " ";
    }
}
// Anak kiri, simpul saat ini, anak kanan
void inOrder(Node* node) {
    if (!node) {
        return;
    } else {
        inOrder(node->left);
        cout << node->label << " ";
        inOrder(node->right);
    }
}

// Fungsi ini digunakan untuk menghitung kedalaman maksimal dari pohon
int maxDepth(Node* node) {
    if (node == NULL) {
        return 0;
    } else { // Rekursi untuk menghitung kedalaman dari subtree kiri dan kanan
        int leftDepth = maxDepth(node->left);
        int rightDepth = maxDepth(node->right);
        return max(leftDepth, rightDepth) + 1; // Mengembalikan kedalaman maksimal dari pohon
    }
}

// Fungsi utama program yang melakukan interaksi pengguna
int main() {
    char label, parentLabel;
    string child;
    int choice;

    cout << "Masukkan label untuk root: ";
    cin >> label;
    createTree(label);

    while (true) {
        cout << "\n1. Tambah node\n2. Pre-order traversal\n3. In-order traversal\n4. Post-order traversal\n5. Cari node\n6. Hapus subtree\n7. Hitung kedalaman maksimal\n8. Keluar\nPilih: ";
        cin >> choice;
```

```
        switch (choice) {
            case 1:
                cout << "Masukkan label node baru: ";
                cin >> label;
                cout << "Masukkan label parent node: ";
                cin >> parentLabel;
                cout << "Masukkan posisi (left/right): ";
                cin >> child;
                insert(label, parentLabel, child);
                break;
            case 2:
                cout << "Pre-order traversal: ";
                preOrder(root);
                cout << endl;
                break;
            case 3:
                cout << "In-order traversal: ";
                inOrder(root);
                cout << endl;
                break;
            case 4:
                cout << "Post-order traversal: ";
                postOrder(root);
                cout << endl;
                break;
```

```
        case 5:
            cout << "Masukkan label node yang dicari: ";
            cin >> label;
            Node* foundNode;
            foundNode = search(root, label);
            if (foundNode) {
                cout << "Node dengan label " << label << " ditemukan." << endl;
            } else {
                cout << "Node dengan label " << label << " tidak ditemukan." << endl;
            }
            break;
        case 6:
            cout << "Masukkan label node yang akan dihapus subtree-nya: ";
            cin >> label;
            Node* nodeToDelete;
            nodeToDelete = search(root, label);
            if (nodeToDelete) {
                deleteSub(nodeToDelete);
                cout << "Subtree dengan root " << label << " telah dihapus." << endl;
            } else {
                cout << "Node tidak ditemukan!" << endl;
            }
            break;
```

```
271             case 7:
272                 cout << "Kedalaman maksimal tree: " << maxDepth(root) << endl;
273                 break;
274             case 8:
275                 clearTree(root);
276                 cout << "Tree berhasil dibersihkan." << endl;
277                 identitas();
278                 return 0;
279             default:
280                 cout << "Pilihan tidak valid!" << endl;
281                 break;
282         }
283     }
284 }
285
286
```

Penjelasan :

Program ini merupakan implementasi dari struktur data Binary Tree dalam bahasa C++. Program ini memungkinkan pengguna untuk membuat, mengedit, dan menjelajahi pohon biner.

1. Pengguna diminta untuk memasukkan label untuk root tree.
2. Fungsi createTree() kemudian dipanggil untuk membuat simpul root dengan label yang dimasukkan.
3. Program masuk ke dalam loop while (true) yang berjalan tak terbatas, sehingga terus menampilkan menu interaktif kepada pengguna.
4. Pengguna diberikan pilihan untuk melakukan berbagai operasi pada pohon biner, seperti menambahkan node baru, melakukan traversal, mencari node, menghapus subtree, menghitung kedalaman maksimal, dan keluar dari program.
5. Program membaca pilihan yang dimasukkan oleh pengguna menggunakan pernyataan switch.
6. Bergantung pada pilihan yang dipilih, program memanggil fungsi-fungsi yang sesuai untuk menjalankan operasi yang diminta.
7. Terdapat beberapa operasi yang dapat dilakukan:
8. Menambahkan node baru ke pohon dengan memanggil fungsi insert().
9. Melakukan traversal pada pohon menggunakan pre-order, in-order, dan post-order traversal.
10. Mencari node dalam pohon dengan menggunakan fungsi search().
11. Menghapus subtree dari pohon menggunakan fungsi deleteSub().
12. Menghitung kedalaman maksimal pohon dengan memanggil fungsi maxDepth().
13. Keluar dari program dengan membersihkan pohon dan menampilkan identitas pembuat program.
14. Setiap operasi yang dilakukan akan memberikan output yang sesuai kepada pengguna, seperti pesan sukses atau pesan kesalahan.
15. Pengguna dapat terus berinteraksi dengan program sampai memilih untuk keluar.
16. Setelah pengguna memilih untuk keluar dari program, program akan membersihkan pohon dan menampilkan identitas pembuat program sebelum akhirnya berakhir.

Demikianlah alur kerja dari program ini, di mana pengguna dapat berinteraksi dengan pohon biner dan melakukan berbagai operasi yang relevan sesuai dengan kebutuhan mereka.

Hasil dari program tersebut sebagai berikut :

```
Masukkan label untuk root: A
A Berhasil menjadi root

1. Tambah node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Cari node
6. Hapus subtree
7. Hitung kedalaman maksimal
8. Keluar
Pilih: 1
Masukkan label node baru: B
Masukkan label parent node: A
Masukkan posisi (left/right): left
Label B berhasil dibuat di anak kiri dari node A
```

```
1. Tambah node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Cari node
6. Hapus subtree
7. Hitung kedalaman maksimal
8. Keluar
Pilih: 1
Masukkan label node baru: C
Masukkan label parent node: A
Masukkan posisi (left/right): right
Label C berhasil dibuat di anak kanan dari node A
```

```
1. Tambah node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Cari node
6. Hapus subtree
7. Hitung kedalaman maksimal
8. Keluar
Pilih: 2
Pre-order traversal: A B C
```

```
1. Tambah node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Cari node
6. Hapus subtree
7. Hitung kedalaman maksimal
8. Keluar
Pilih: 3
In-order traversal: B A C
```

```
1. Tambah node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Cari node
6. Hapus subtree
7. Hitung kedalaman maksimal
8. Keluar
Pilih: 4
Post-order traversal: B C A
```

```
1. Tambah node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Cari node
6. Hapus subtree
7. Hitung kedalaman maksimal
8. Keluar
Pilih: 5
Masukkan label node yang dicari: A
Node dengan label A ditemukan.

1. Tambah node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Cari node
6. Hapus subtree
7. Hitung kedalaman maksimal
8. Keluar
Pilih: 5
Masukkan label node yang dicari: D
Node dengan label D tidak ditemukan.
```

```
1. Tambah node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Cari node
6. Hapus subtree
7. Hitung kedalaman maksimal
8. Keluar
Pilih: 6
Masukkan label node yang akan dihapus subtree-nya: A
Subtree dengan root A telah dihapus.

1. Tambah node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Cari node
6. Hapus subtree
7. Hitung kedalaman maksimal
8. Keluar
Pilih: 6
Masukkan label node yang akan dihapus subtree-nya: D
Node tidak ditemukan!
```

```
1. Tambah node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Cari node
6. Hapus subtree
7. Hitung kedalaman maksimal
8. Keluar
Pilih: 7
Kedalaman maksimal tree: 0

1. Tambah node
2. Pre-order traversal
3. In-order traversal
4. Post-order traversal
5. Cari node
6. Hapus subtree
7. Hitung kedalaman maksimal
8. Keluar
Pilih: 8
Tree berhasil dibersihkan.

Program Mengukur Maksimal Depth pada Binary Tree CPP Program
Nama     : Nabiilah Nur Fauziyyah
NPM      : 2310631170105
Kelas    : 2C - Informatika
```