LAPORAN PRAKTIKUM

STRUKTUR DATA

MODUL VII

TREE (BAGIAN PERTAMA)

Disusun Oleh :

NAMA : Muhammad Fachri Auravyano Saka

NIM : 103112430180

Dosen

FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

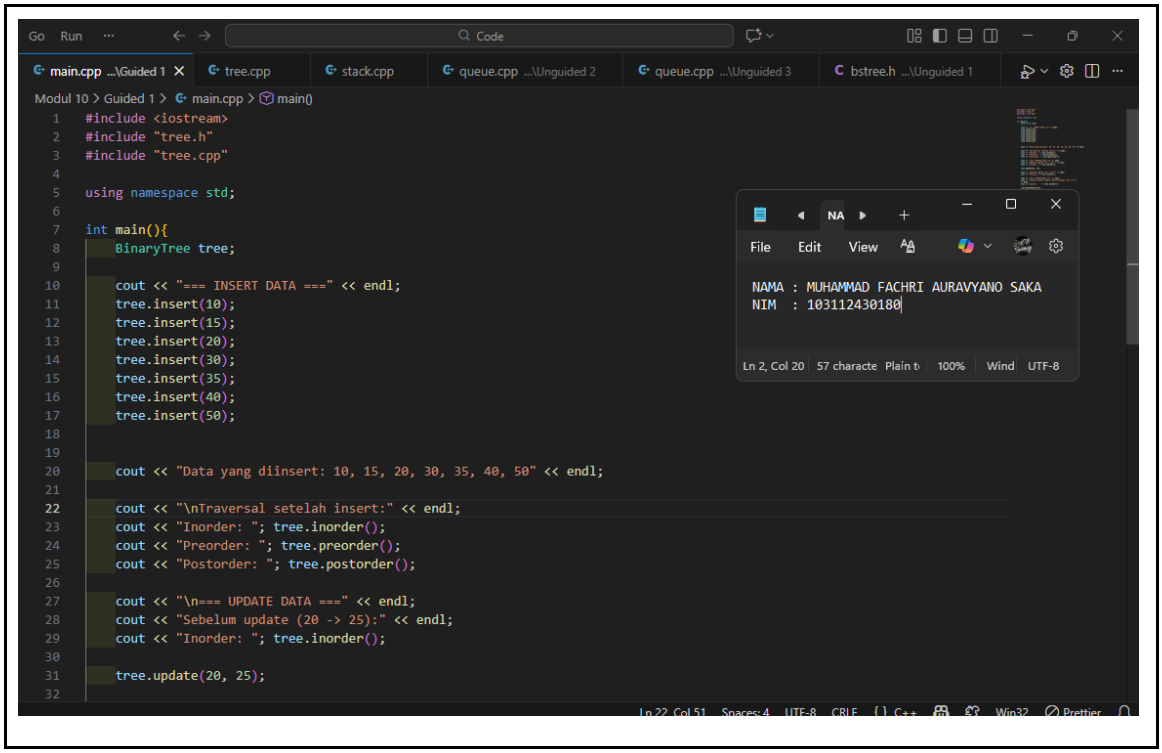
Queue atau antrean adalah struktur data yang bekerja berdasarkan prinsip FIFO (First In, First Out). Konsep ini sangat mirip dengan antrean di dunia nyata, misalnya saat mengantre membeli tiket bioskop: orang yang datang paling awal (masuk pertama) akan dilayani dan keluar paling awal juga. Sebaliknya, orang yang datang terakhir harus menunggu di posisi paling belakang.

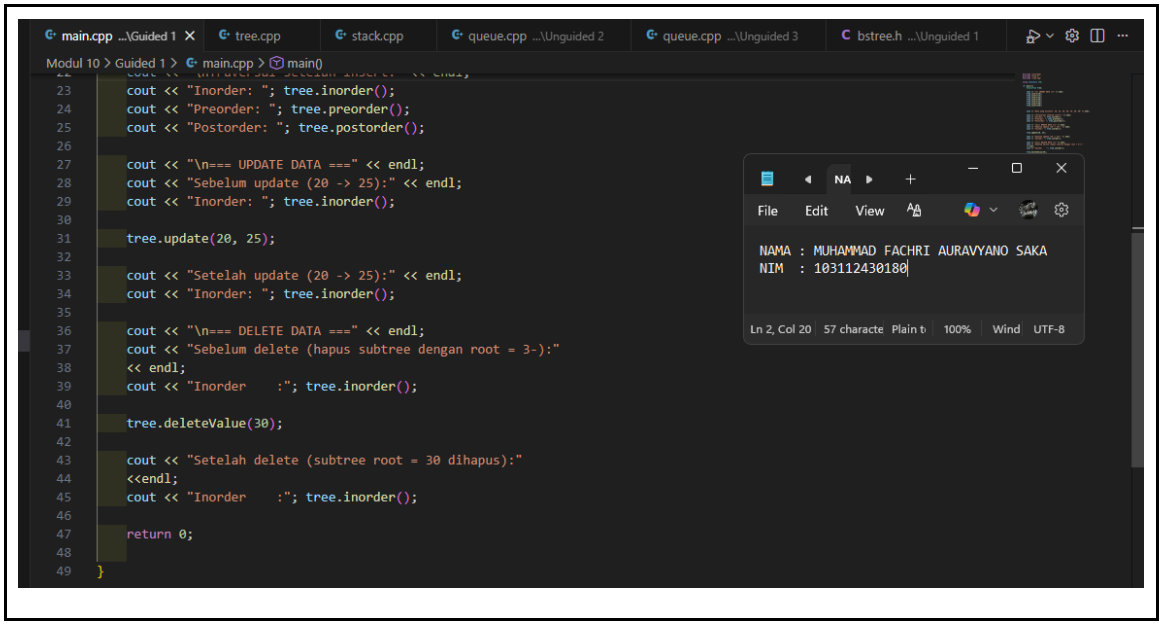
Dalam pemrograman (khususnya C++ menggunakan Array), Queue memiliki dua pintu utama: Head (depan) untuk data keluar, dan Tail (belakang) untuk data masuk. Operasi memasukkan data disebut Enqueue, sedangkan mengeluarkan data disebut Dequeue. Tantangan utama menggunakan Array untuk Queue adalah ukuran memori yang tetap.

B. Guided (berisi screenshot source code & output program disertai penjelasannya)

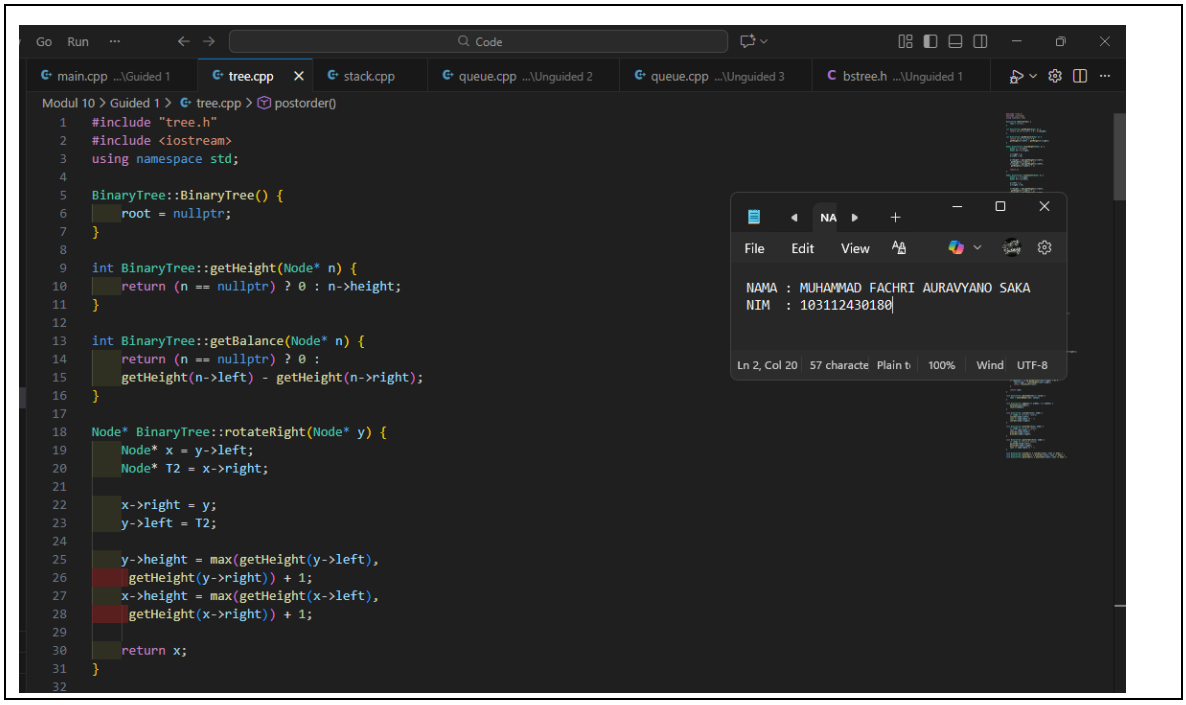
Guided 1

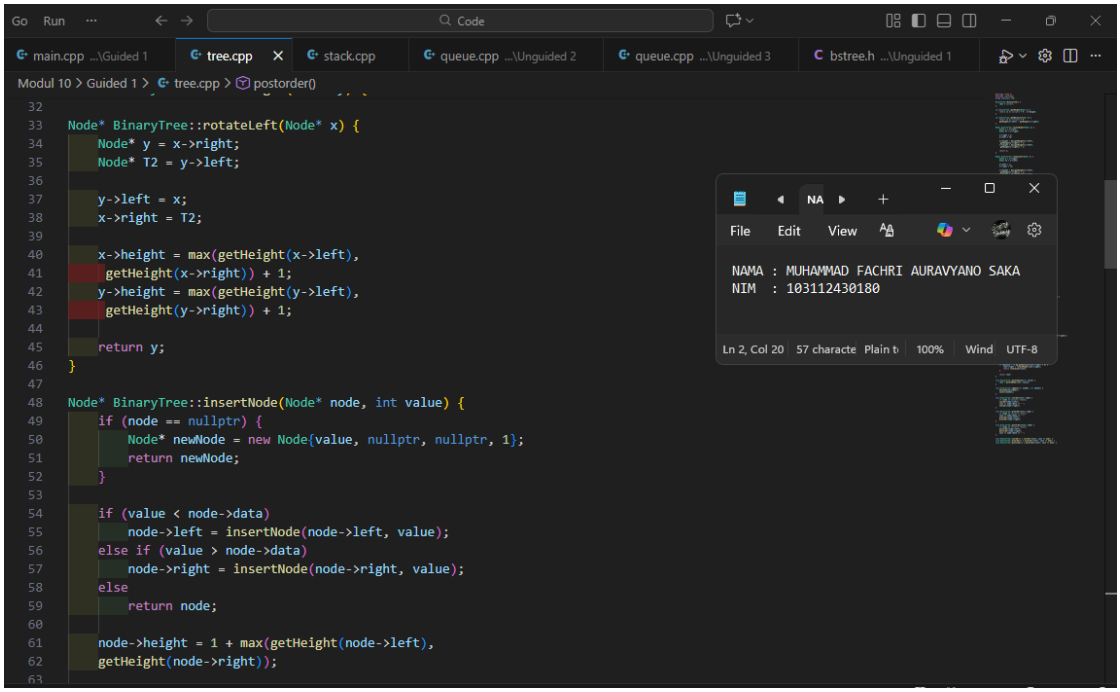
Main.cpp

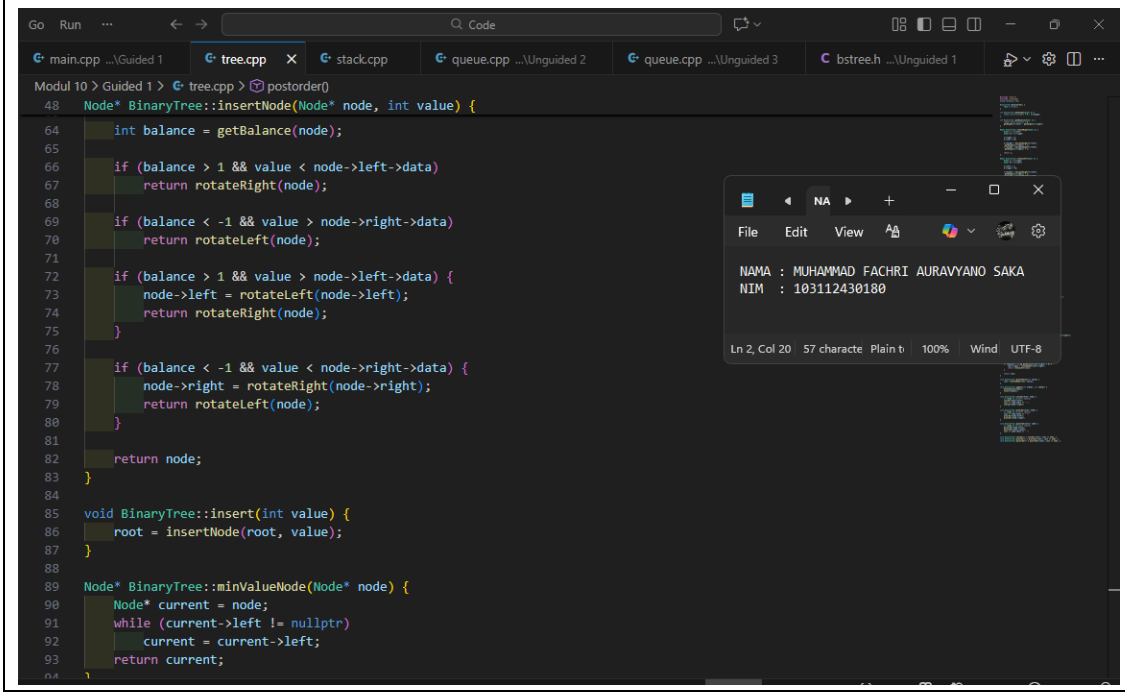


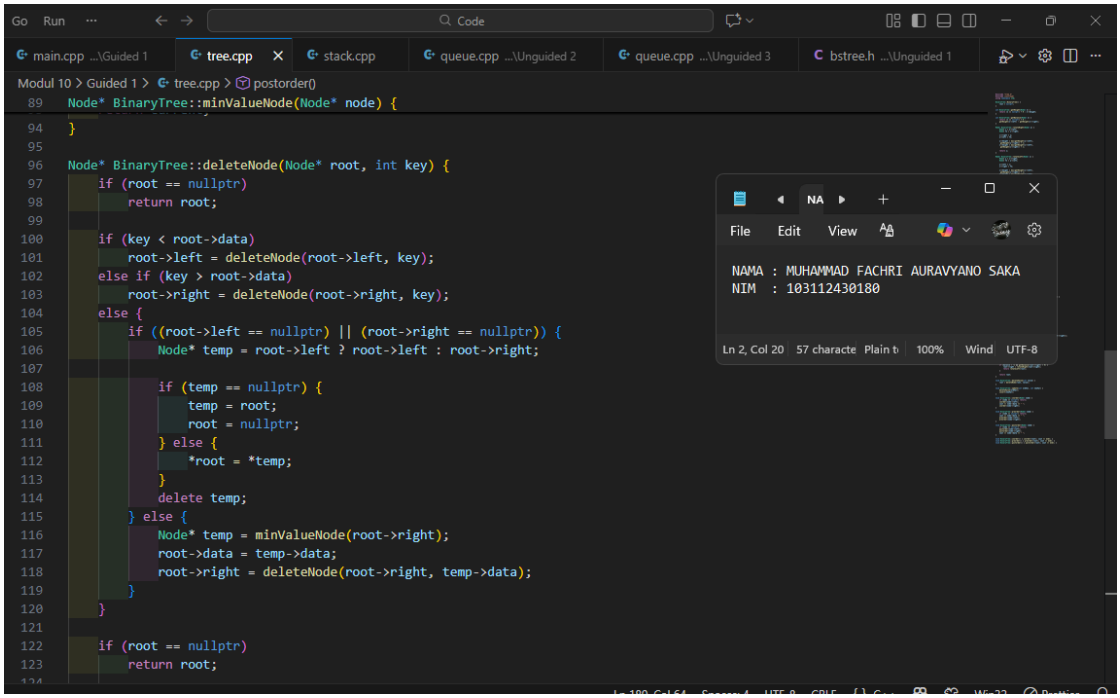

main.cpp ...\Guided 1
tree.cpp
stack.cpp
queue.cpp ...\Unguided 2
queue.cpp ...\Unguided 3
bstree.h ...\Unguided 1
Modul 10 > Guided 1 > main.cpp > main()
cout << "Inorder: "; tree.inorder();
cout << "Preorder: "; tree.preorder();
cout << "Postorder: "; tree.postorder();
cout << "\n=== UPDATE DATA ===" << endl;
cout << "Sebelum update (20 -> 25):" << endl;
cout << "Inorder: "; tree.inorder();
tree.update(20, 25);
cout << "Setelah update (20 -> 25):" << endl;
cout << "Inorder: "; tree.inorder();
cout << "\n=== DELETE DATA ===" << endl;
cout << "Sebelum delete (hapus subtree dengan root = 3-):"
<< endl;
cout << "Inorder :"; tree.inorder();
tree.deleteValue(30);
cout << "Setelah delete (subtree root = 30 dihapus):"
<<endl;
cout << "Inorder :"; tree.inorder();
return 0;
}
NA
File Edit View
NAMA : MUHAMMAD FACHRI AURAVYANO SAKA
NIM : 103112430180
Ln 2, Col 20 57 characte Plain t 100% Wind UTF-8

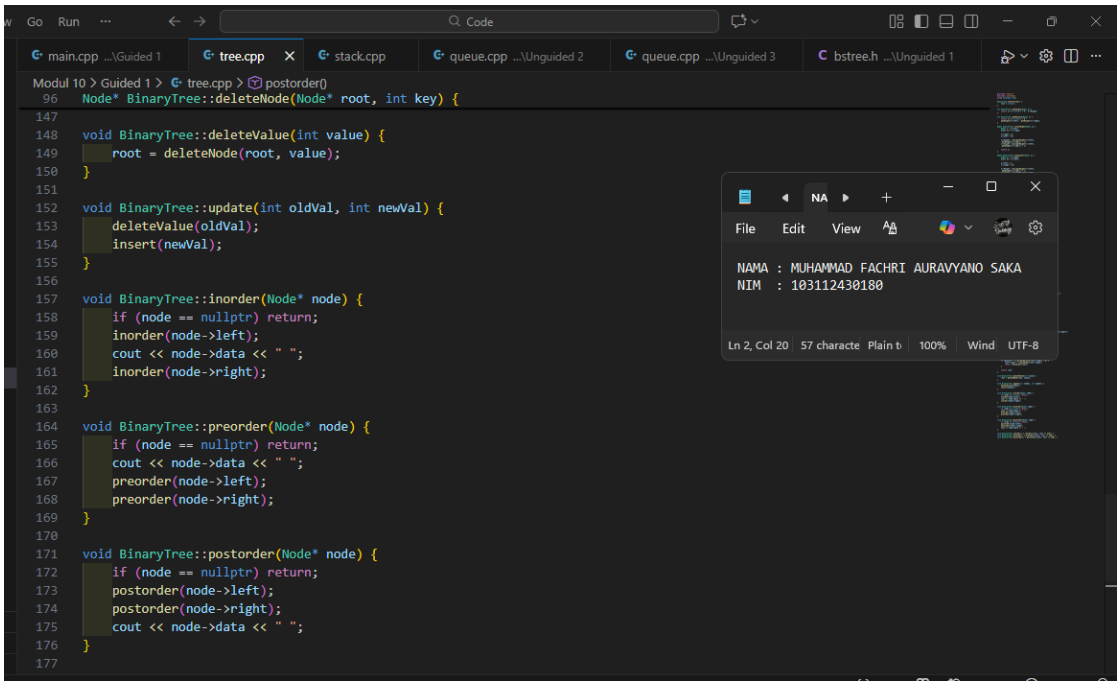
Tree.cpp

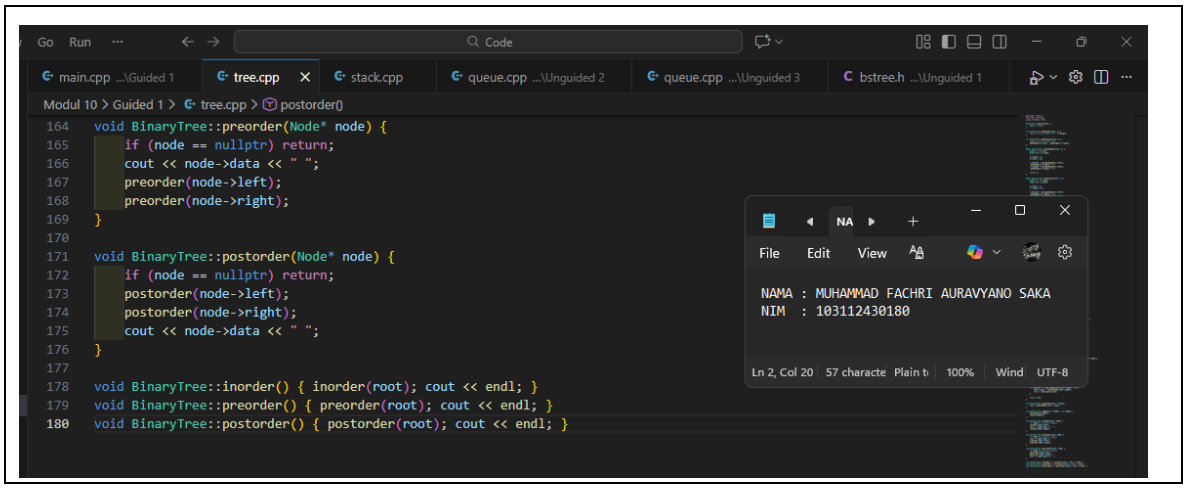

Go Run
Code
main.cpp ...\Guided 1
tree.cpp
stack.cpp
queue.cpp ...\Unguided 2
queue.cpp ...\Unguided 3
bstree.h ...\Unguided 1
Modul 10 > Guided 1 > tree.cpp > postorder()
#include "tree.h"
#include <iostream>
using namespace std;
BinaryTree::BinaryTree() {
root = nullptr;
}
int BinaryTree::getHeight(Node* n) {
return (n == nullptr) ? 0 : n->height;
}
int BinaryTree::getBalance(Node* n) {
return (n == nullptr) ? 0 :
getHeight(n->left) - getHeight(n->right);
}
Node* BinaryTree::rotateRight(Node* y) {
Node* x = y->left;
Node* T2 = x->right;
x->right = y;
y->left = T2;
y->height = max(getHeight(y->left),
getHeight(y->right)) + 1;
x->height = max(getHeight(x->left),
getHeight(x->right)) + 1;
return x;
}
NA
File Edit View
NAMA : MUHAMMAD FACHRI AURAVYANO SAKA
NIM : 103112430180
Ln 2, Col 20 57 characte Plain t 100% Wind UTF-8


Go Run
Code
main.cpp ...\Guided 1
tree.cpp
stack.cpp
queue.cpp ...\Unguided 2
queue.cpp ...\Unguided 3
bstree.h ...\Unguided 1
Modul 10 > Guided 1 > tree.cpp > postorder()
Node* BinaryTree::rotateLeft(Node* x) {
Node* y = x->right;
Node* T2 = y->left;
y->left = x;
x->right = T2;
x->height = max(getHeight(x->left),
getHeight(x->right)) + 1;
y->height = max(getHeight(y->left),
getHeight(y->right)) + 1;
return y;
}
Node* BinaryTree::insertNode(Node* node, int value) {
if (node == nullptr) {
Node* newNode = new Node{value, nullptr, nullptr, 1};
return newNode;
}
if (value < node->data)
node->left = insertNode(node->left, value);
else if (value > node->data)
node->right = insertNode(node->right, value);
else
return node;
node->height = 1 + max(getHeight(node->left),
getHeight(node->right));
NA
File Edit View
NAMA : MUHAMMAD FACHRI AURAVYANO SAKA
NIM : 103112430180
Ln 2, Col 20 57 characte Plain t 100% Wind UTF-8

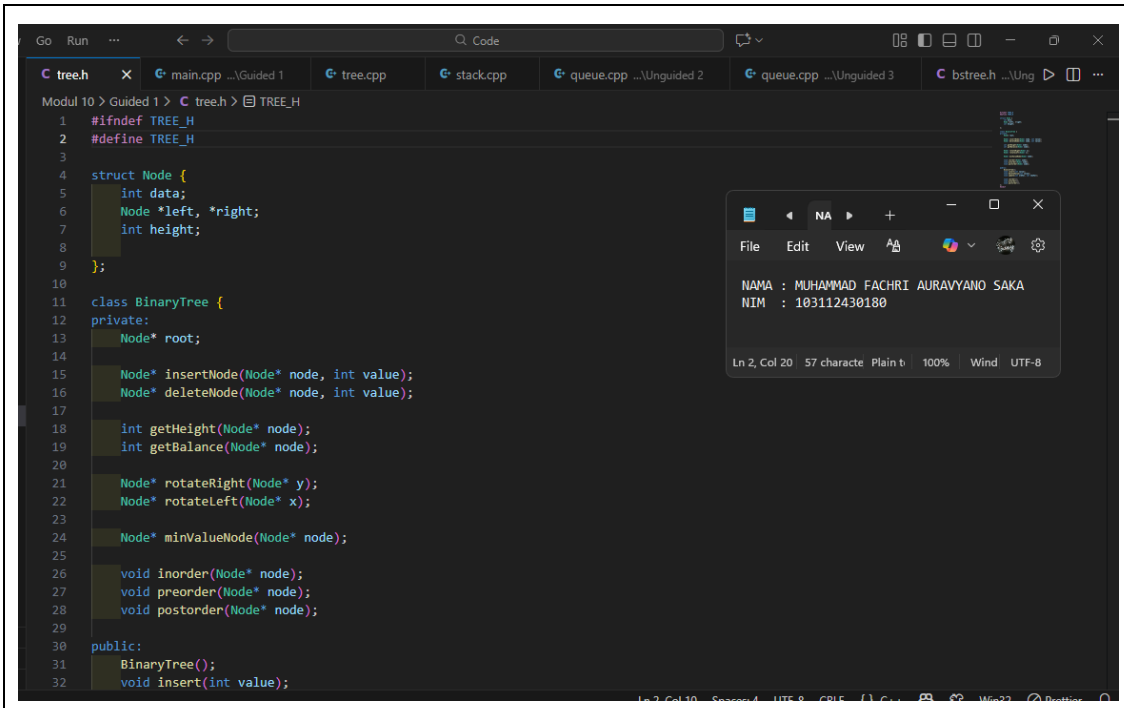

Go Run
Code
main.cpp ...\Guided 1
tree.cpp
stack.cpp
queue.cpp ...\Unguided 2
queue.cpp ...\Unguided 3
bstree.h ...\Unguided 1
Modul 10 > Guided 1 > tree.cpp > postorder()
Node* BinaryTree::insertNode(Node* node, int value) {
int balance = getBalance(node);
if (balance > 1 && value < node->left->data)
return rotateRight(node);
if (balance < -1 && value > node->right->data)
return rotateLeft(node);
if (balance > 1 && value > node->left->data) {
node->left = rotateLeft(node->left);
return rotateRight(node);
}
if (balance < -1 && value < node->right->data) {
node->right = rotateRight(node->right);
return rotateLeft(node);
}
return node;
}
void BinaryTree::insert(int value) {
root = insertNode(root, value);
}
Node* BinaryTree::minValueNode(Node* node) {
Node* current = node;
while (current->left != nullptr)
current = current->left;
return current;
NA
File Edit View
NAMA : MUHAMMAD FACHRI AURAVYANO SAKA
NIM : 103112430180
Ln 2, Col 20 57 characte Plain t 100% Wind UTF-8

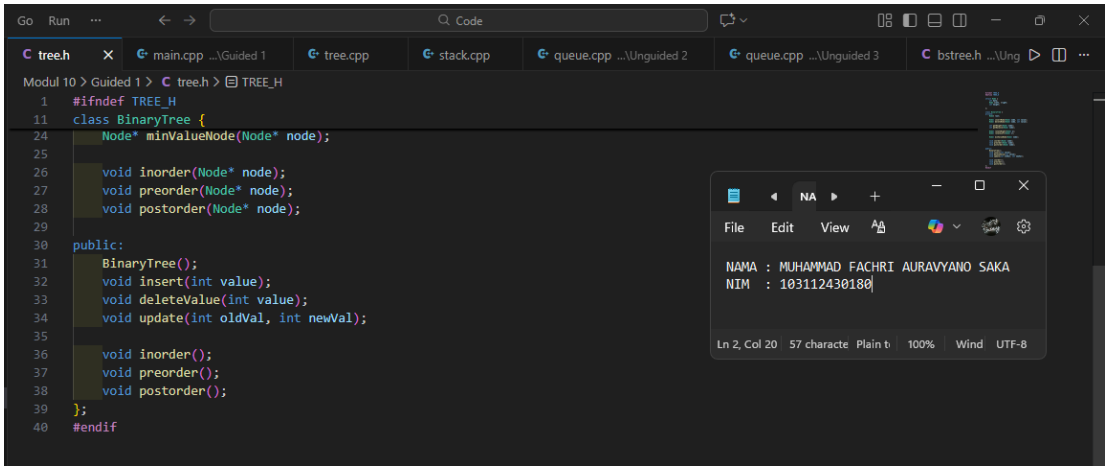

Go Run
Code
main.cpp ...\Guided 1
tree.cpp
stack.cpp
queue.cpp ...\Unguided 2
queue.cpp ...\Unguided 3
bstree.h ...\Unguided 1
Modul 10 > Guided 1 > tree.cpp > postorder()
Node* BinaryTree::minValueNode(Node* node) {
}
Node* BinaryTree::deleteNode(Node* root, int key) {
if (root == nullptr)
return root;
if (key < root->data)
root->left = deleteNode(root->left, key);
else if (key > root->data)
root->right = deleteNode(root->right, key);
else {
if ((root->left == nullptr) || (root->right == nullptr)) {
Node* temp = root->left ? root->left : root->right;
if (temp == nullptr) {
temp = root;
root = nullptr;
} else {
*root = *temp;
}
delete temp;
} else {
Node* temp = minValueNode(root->right);
root->data = temp->data;
root->right = deleteNode(root->right, temp->data);
}
}
if (root == nullptr)
return root;
NA
File Edit View
NAMA : MUHAMMAD FACHRI AURAVYANO SAKA
NIM : 103112430180
Ln 2, Col 20 57 characte Plain t 100% Wind UTF-8


Go Run
Code
main.cpp ...\Guided 1
tree.cpp
stack.cpp
queue.cpp ...\Unguided 2
queue.cpp ...\Unguided 3
bstree.h ...\Unguided 1
Modul 10 > Guided 1 > tree.cpp > postorder()
Node* BinaryTree::deleteNode(Node* root, int key) {
void BinaryTree::deleteValue(int value) {
root = deleteNode(root, value);
}
void BinaryTree::update(int oldVal, int newVal) {
deleteValue(oldVal);
insert(newVal);
}
void BinaryTree::inorder(Node* node) {
if (node == nullptr) return;
inorder(node->left);
cout << node->data << " ";
inorder(node->right);
}
void BinaryTree::preorder(Node* node) {
if (node == nullptr) return;
cout << node->data << " ";
preorder(node->left);
preorder(node->right);
}
void BinaryTree::postorder(Node* node) {
if (node == nullptr) return;
postorder(node->left);
postorder(node->right);
cout << node->data << " ";
}
NA
File Edit View
NAMA : MUHAMMAD FACHRI AURAVYANO SAKA
NIM : 103112430180
Ln 2, Col 20 57 characte Plain t 100% Wind UTF-8

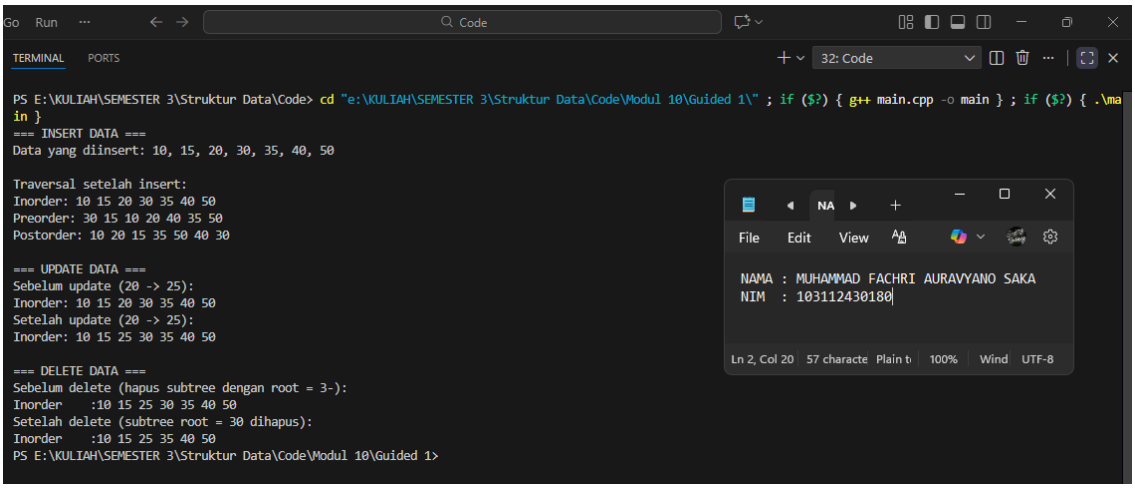


Tree.h





Screenshots Output



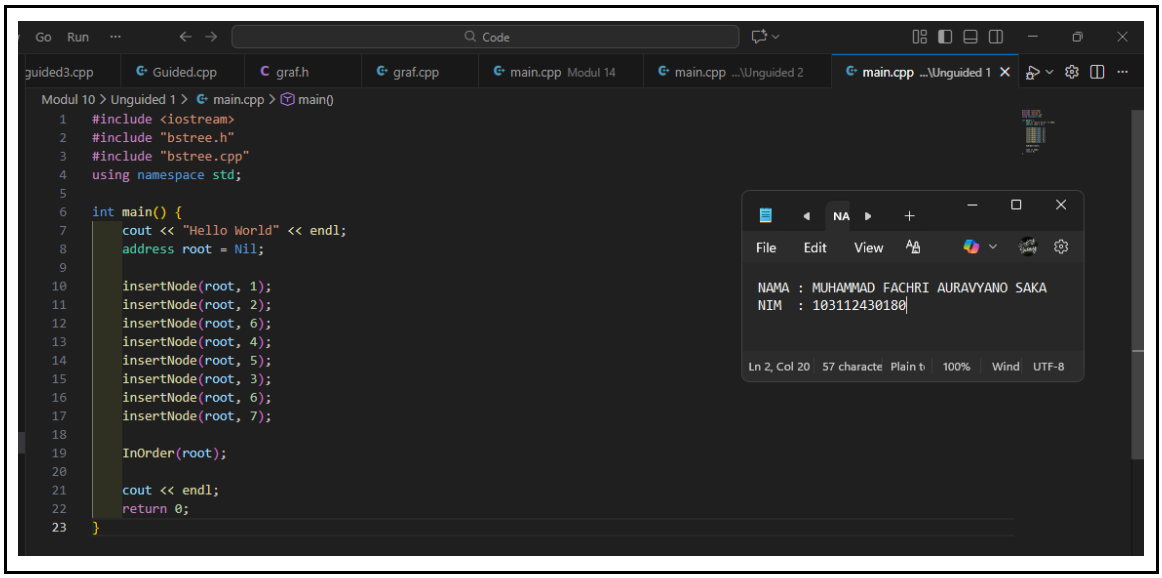
Deskripsi:

Program ini merupakan implementasi dari struktur data AVL Tree (sebuah Binary Search Tree yang dapat menyeimbangkan dirinya sendiri secara otomatis) menggunakan bahasa C++. Meskipun kelasnya dinamakan BinaryTree, kode ini menerapkan logika penyeimbangan menggunakan rotasi (rotate left dan rotate right) dan perhitungan ketinggian (height) node untuk memastikan pohon tidak miring (skewed), sehingga performa pencarian data tetap efisien. Program mencakup fitur utama untuk menambah data (insert), menghapus data (delete), dan memperbarui data (update)—di mana update dilakukan dengan menghapus nilai lama dan memasukkan nilai baru. Di bagian main, program mendemonstrasikan fungsi-fungsi tersebut dengan memasukkan sekumpulan angka, lalu menampilkan hasilnya menggunakan tiga metode penelusuran (traversal): Inorder, Preorder, dan Postorder.

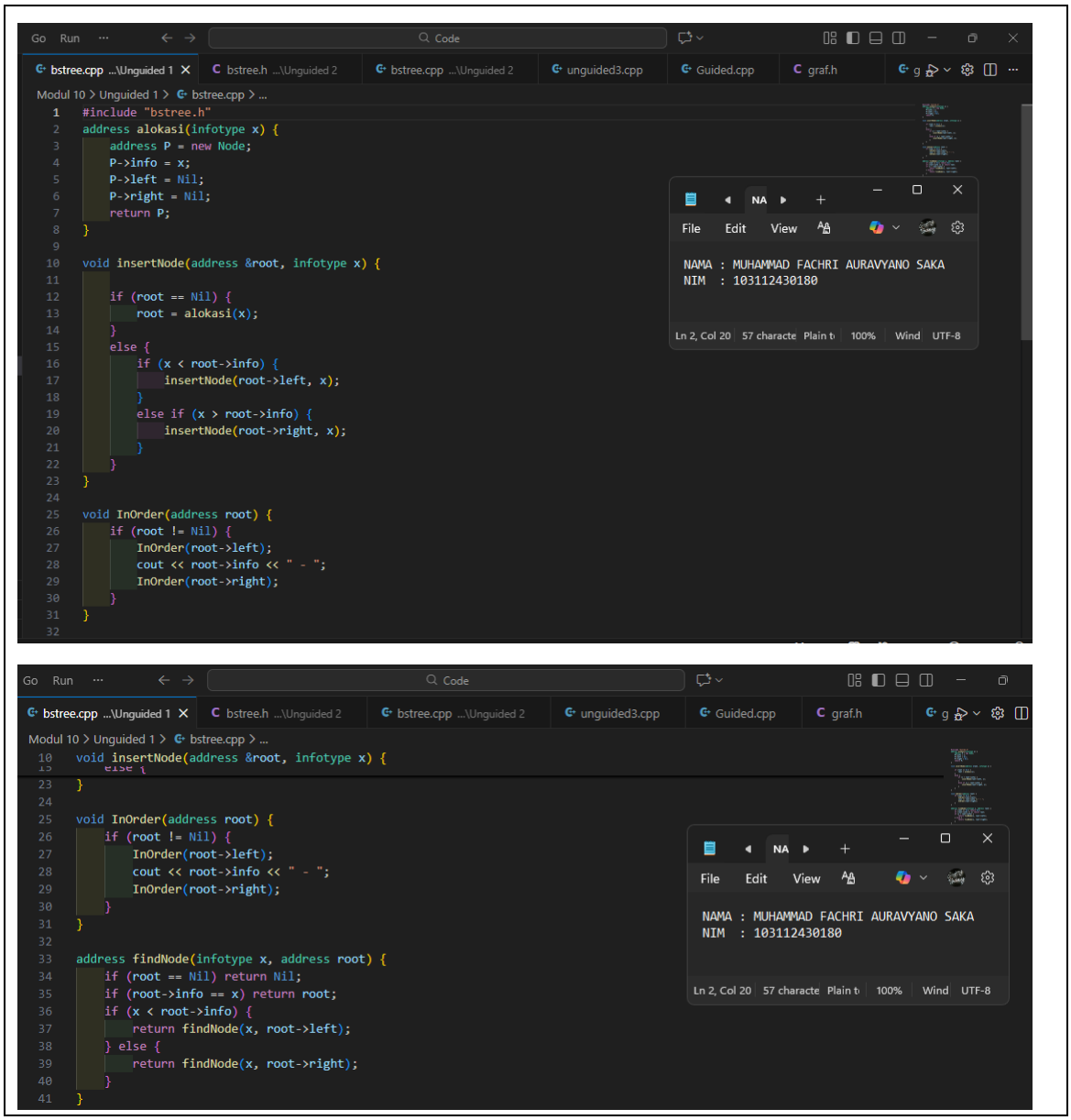
C. Unguided/Tugas (berisi screenshot source code & output program disertai penjelasannya)

Unguided 1

Main.cpp



bstree.cpp



```cpp
#include "bstree.h"
address alokasi(infotype x) {
    address P = new Node;
    P->info = x;
    P->left = Nil;
    P->right = Nil;
    return P;
}

void insertNode(address &root, infotype x) {

    if (root == Nil) {
        root = alokasi(x);
    }
    else {
        if (x < root->info) {
            insertNode(root->left, x);
        }
        else if (x > root->info) {
            insertNode(root->right, x);
        }
    }
}

void InOrder(address root) {
    if (root != Nil) {
        InOrder(root->left);
        cout << root->info << " - ";
        InOrder(root->right);
    }
}
```

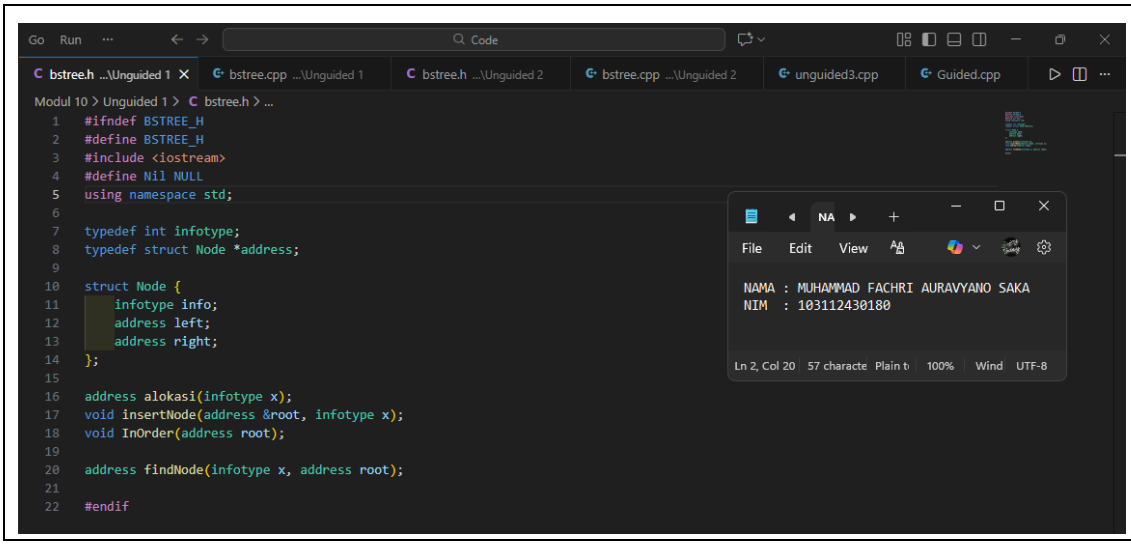
```
NAMA : MUHAMMAD FACHRI AURAVYANO SAKA
NIM  : 103112430180
```



```cpp
void insertNode(address &root, infotype x) {
}

void InOrder(address root) {
    if (root != Nil) {
        InOrder(root->left);
        cout << root->info << " - ";
        InOrder(root->right);
    }
}

address findNode(infotype x, address root) {
    if (root == Nil) return Nil;
    if (root->info == x) return root;
    if (x < root->info) {
        return findNode(x, root->left);
    } else {
        return findNode(x, root->right);
    }
}
```

```
NAMA : MUHAMMAD FACHRI AURAVYANO SAKA
NIM  : 103112430180
```

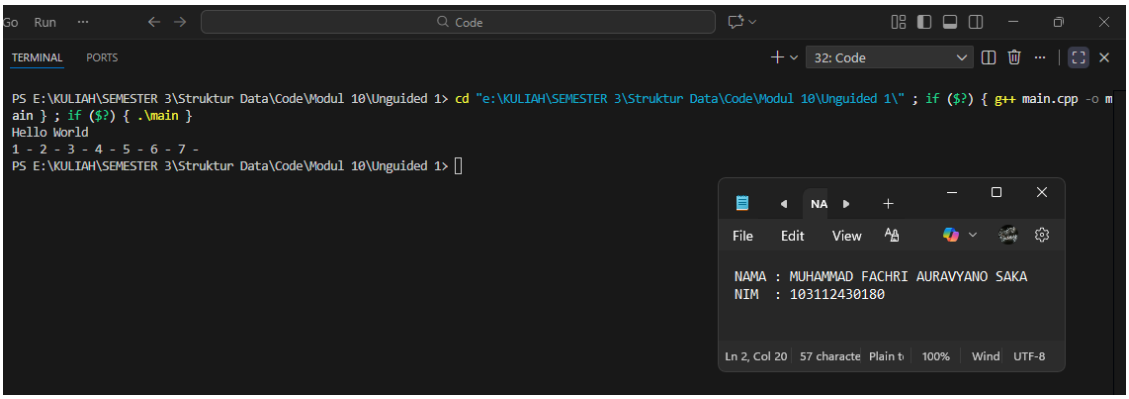
bstree.h



```cpp
#ifndef BSTREE_H
#define BSTREE_H
#include <iostream>
#define Nil NULL
using namespace std;

typedef int infotype;
typedef struct Node *address;

struct Node {
    infotype info;
    address left;
    address right;
};

address alokasi(infotype x);
void insertNode(address &root, infotype x);
void InOrder(address root);

address findNode(infotype x, address root);

#endif
```

```
NAMA : MUHAMMAD FACHRI AURAVYANO SAKA
NIM  : 103112430180
```

Screenshot Output

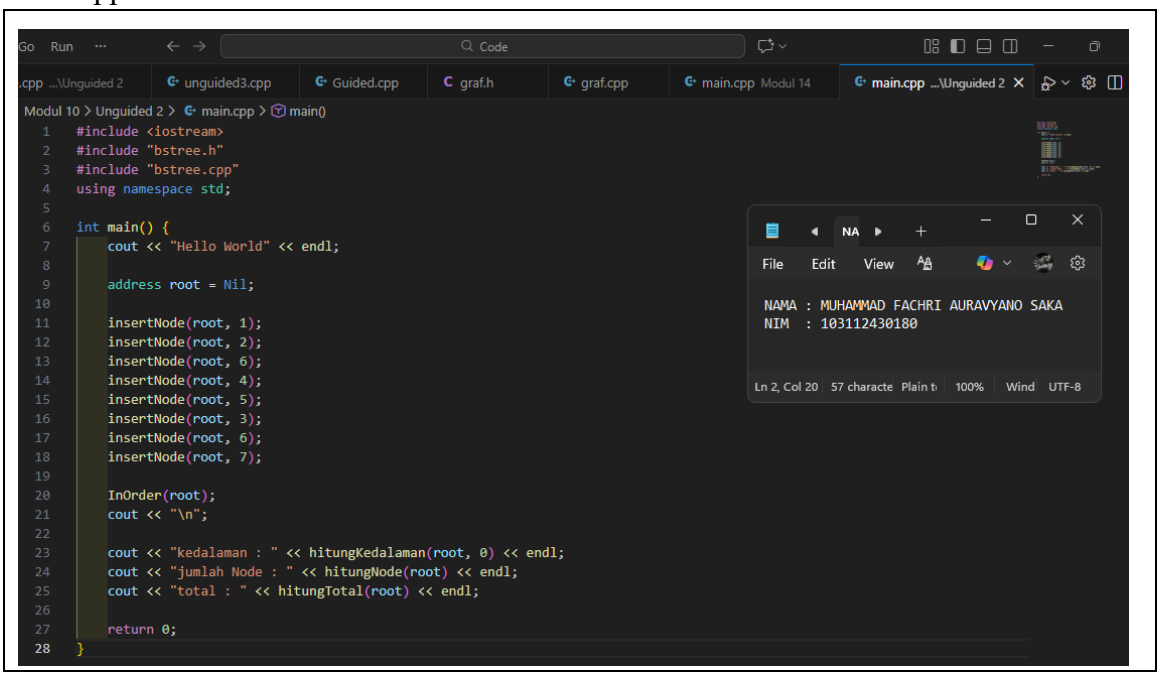


Deskripsi:

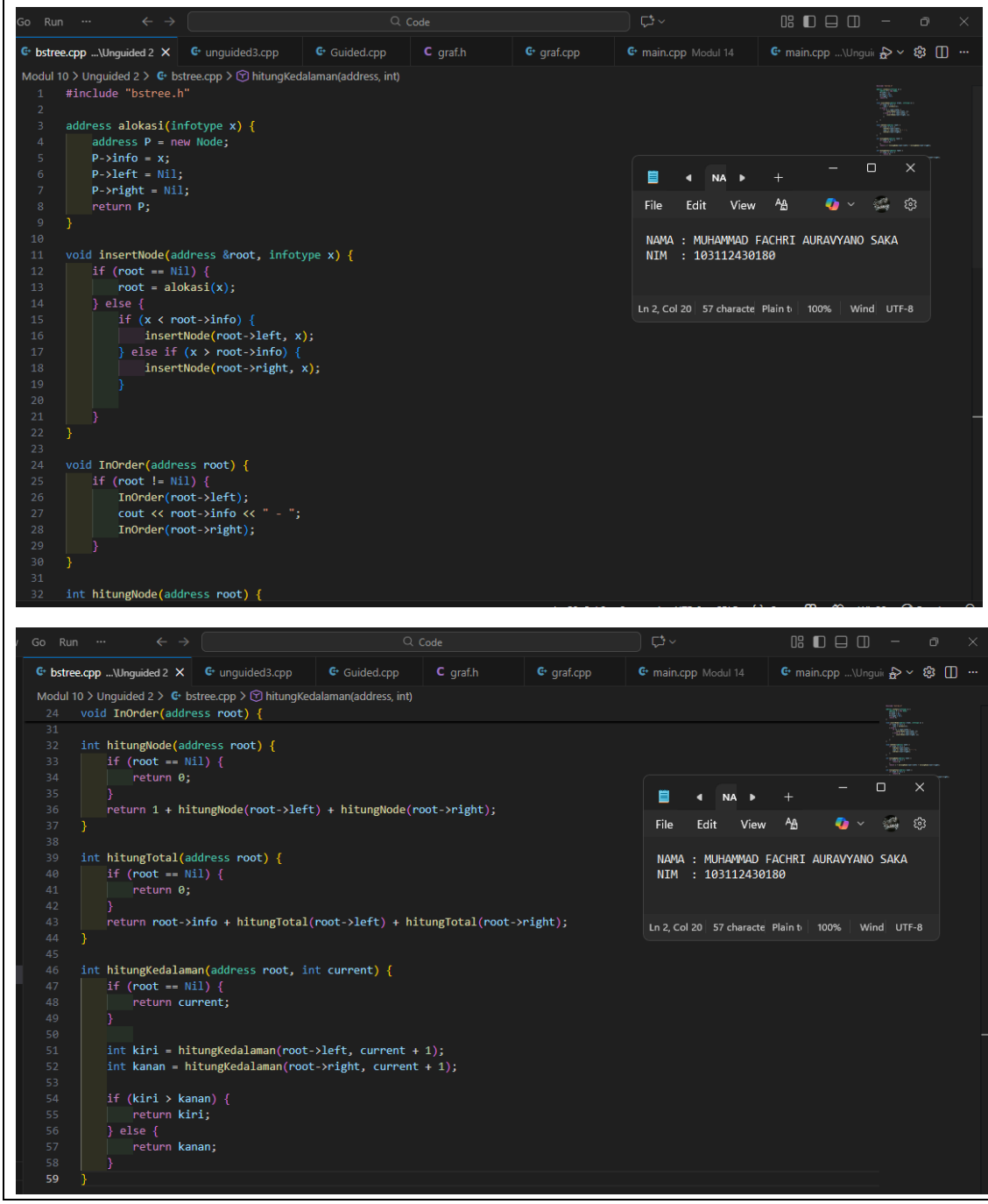
Program ini merupakan implementasi dasar dari Binary Search Tree (BST) standar dalam bahasa C++. Berbeda dengan kode sebelumnya (AVL Tree), program ini tidak memiliki mekanisme penyeimbangan otomatis (rotasi), sehingga struktur pohon murni terbentuk berdasarkan urutan data yang masuk. Logika utamanya ada pada fungsi insertNode, yang secara rekursif menempatkan angka lebih kecil ke anak kiri dan angka lebih besar ke anak kanan. Kode ini juga menangani duplikasi dengan cara mengabaikan nilai yang sama (seperti angka 6 yang dimasukkan dua kali di main, yang kedua tidak akan ditambahkan). Program diakhiri dengan menampilkan seluruh data yang telah tersusun secara urut (menaik) menggunakan penelusuran InOrder.

Unguided 2

Main.cpp



Bstree.cpp



```cpp
#include "bstree.h"

address alokasi(infotype x) {
    address P = new Node;
    P->info = x;
    P->left = Nil;
    P->right = Nil;
    return P;
}

void insertNode(address &root, infotype x) {
    if (root == Nil) {
        root = alokasi(x);
    } else {
        if (x < root->info) {
            insertNode(root->left, x);
        } else if (x > root->info) {
            insertNode(root->right, x);
        }

    }
}

void InOrder(address root) {
    if (root != Nil) {
        InOrder(root->left);
        cout << root->info << " - ";
        InOrder(root->right);
    }
}

int hitungNode(address root) {
```

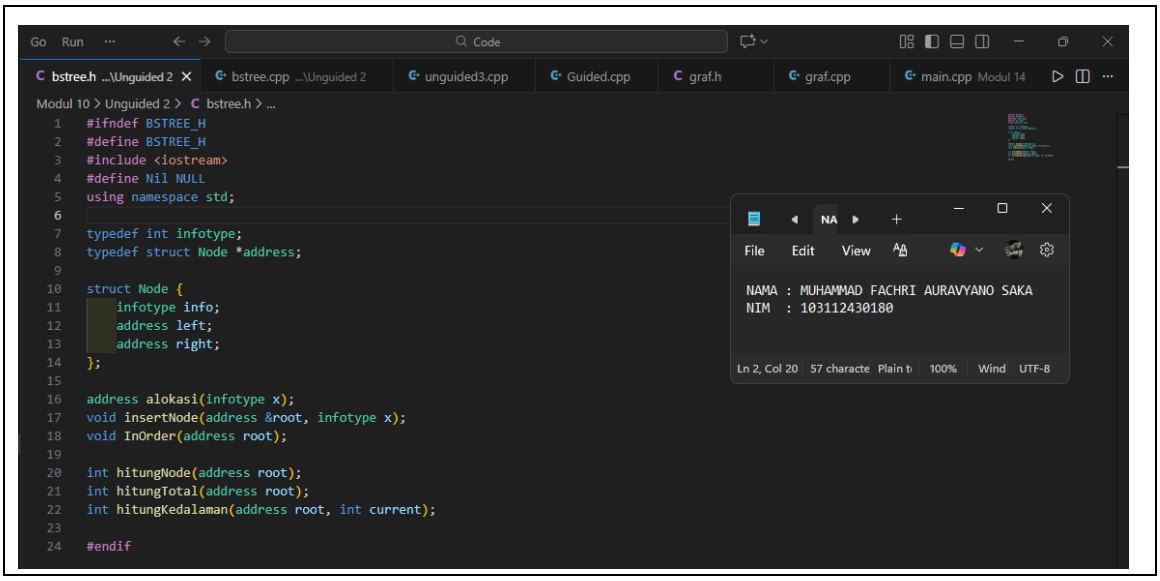
```
NAMA : MUHAMMAD FACHRI AURAVYANO SAKA
NIM  : 103112430180
```



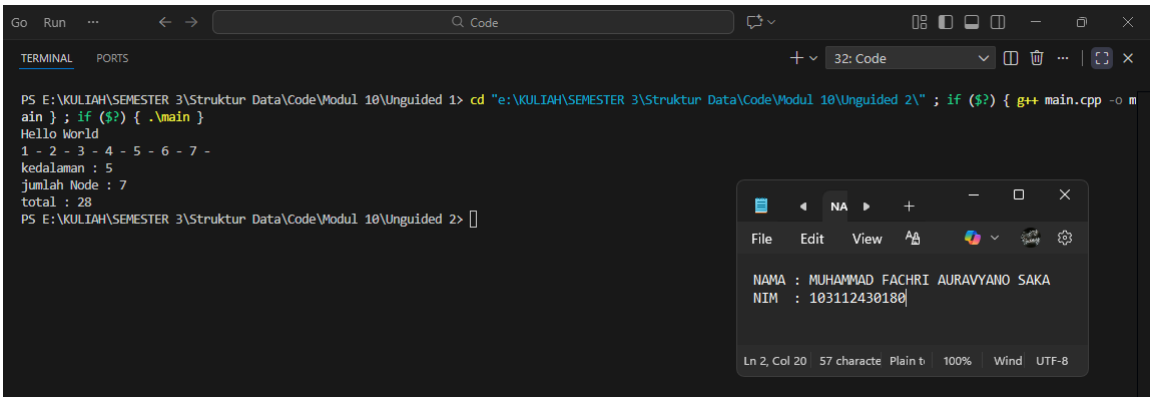
```cpp
void InOrder(address root) {

int hitungNode(address root) {
    if (root == Nil) {
        return 0;
    }
    return 1 + hitungNode(root->left) + hitungNode(root->right);
}

int hitungTotal(address root) {
    if (root == Nil) {
        return 0;
    }
    return root->info + hitungTotal(root->left) + hitungTotal(root->right);
}

int hitungKedalaman(address root, int current) {
    if (root == Nil) {
        return current;
    }

    int kiri = hitungKedalaman(root->left, current + 1);
    int kanan = hitungKedalaman(root->right, current + 1);

    if (kiri > kanan) {
        return kiri;
    } else {
        return kanan;
    }
}
```

```
NAMA : MUHAMMAD FACHRI AURAVYANO SAKA
NIM  : 103112430180
```

Bstree.h



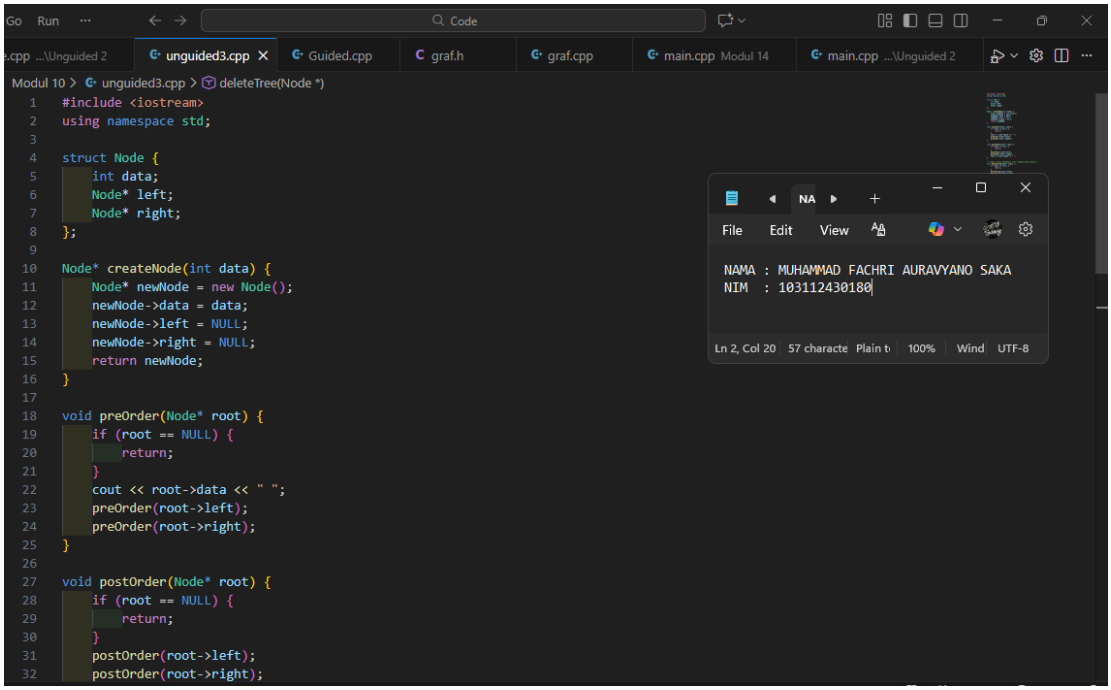
Screenshot Output

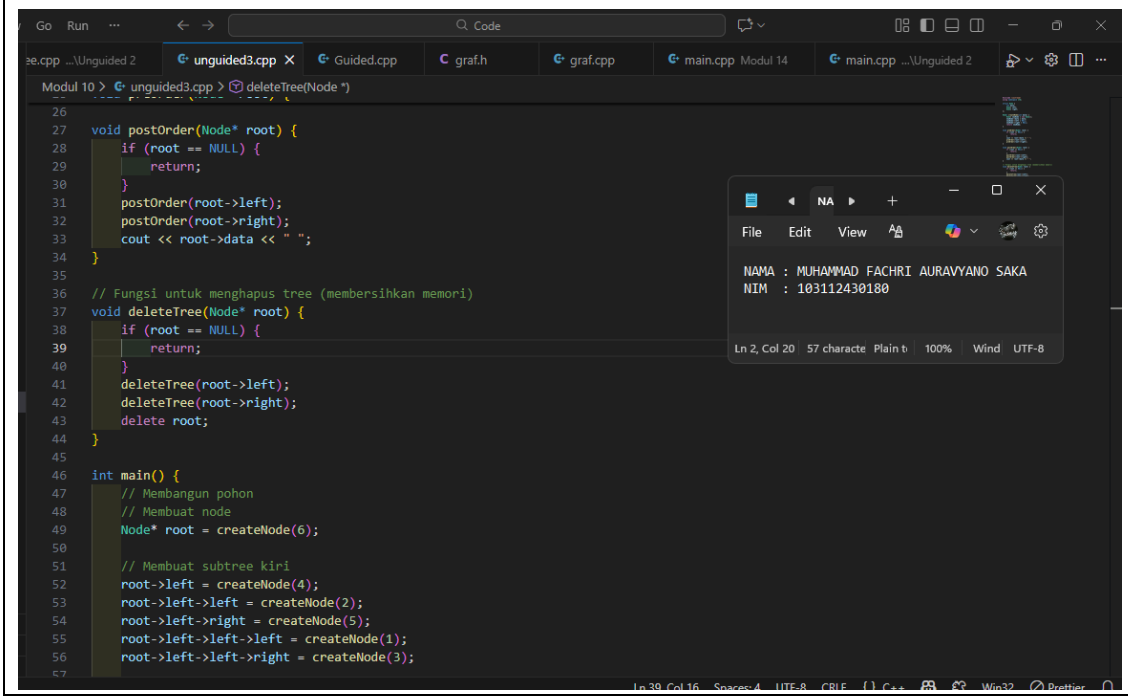


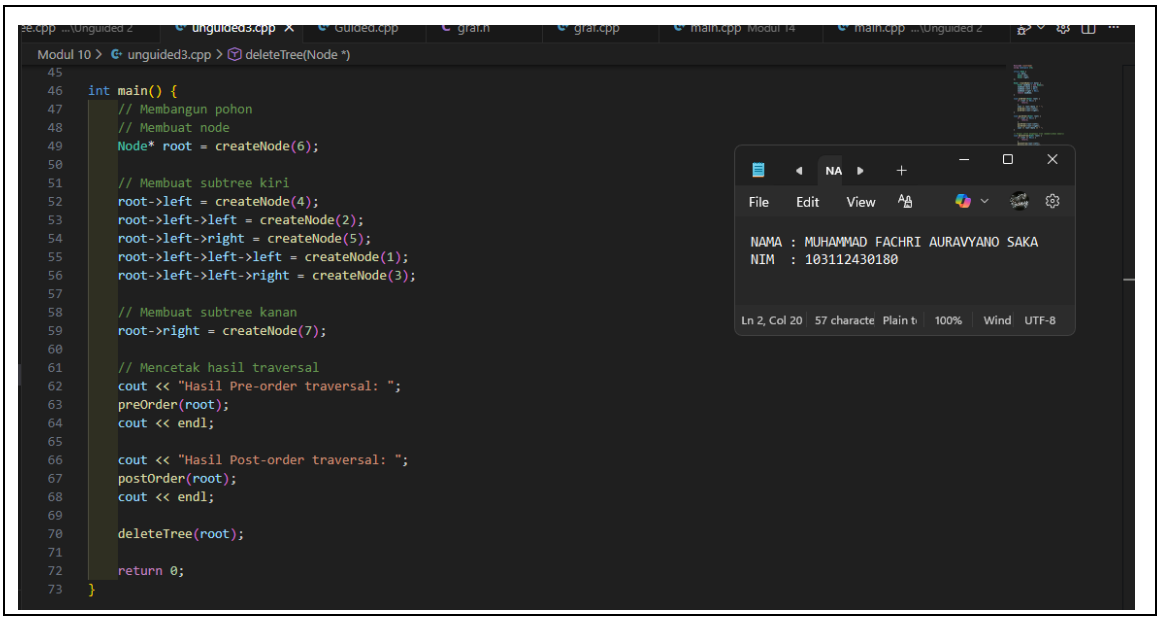
Deskripsi:

Program ini mengimplementasikan struktur data Binary Search Tree (BST) standar tanpa mekanisme penyeimbangan otomatis (rotasi). Fungsi utamanya adalah menyusun angka-angka di mana nilai yang lebih kecil ditempatkan di subtree kiri dan nilai yang lebih besar di subtree kanan. Program ini memiliki logika untuk mengabaikan duplikasi, sehingga jika angka yang sama dimasukkan dua kali (seperti angka 6 di fungsi main), angka kedua tidak akan ditambahkan ke dalam pohon. Selain fungsi dasar penyisipan (insert) dan tampilan terurut (InOrder), program ini dilengkapi dengan fungsi statistik rekursif untuk menghitung jumlah total node, penjumlahan seluruh nilai data, dan kedalaman (depth/height) maksimum pohon tersebut.

Unguided 3


Go Run
Code
unguided3.cpp
Guided.cpp
graf.h
graf.cpp
main.cpp Modul 14
main.cpp ...\Unguided 2
Modul 10 > unguided3.cpp > deleteTree(Node *)
#include <iostream>
using namespace std;
struct Node {
int data;
Node* left;
Node* right;
};
Node* createNode(int data) {
Node* newNode = new Node();
newNode->data = data;
newNode->left = NULL;
newNode->right = NULL;
return newNode;
}
void preOrder(Node* root) {
if (root == NULL) {
return;
}
cout << root->data << " ";
preOrder(root->left);
preOrder(root->right);
}
void postOrder(Node* root) {
if (root == NULL) {
return;
}
postOrder(root->left);
postOrder(root->right);
NA
File Edit View
NAMA : MUHAMMAD FACHRI AURAVYANO SAKA
NIM : 103112430180
Ln 2, Col 20 57 characte Plain t 100% Wind UTF-8

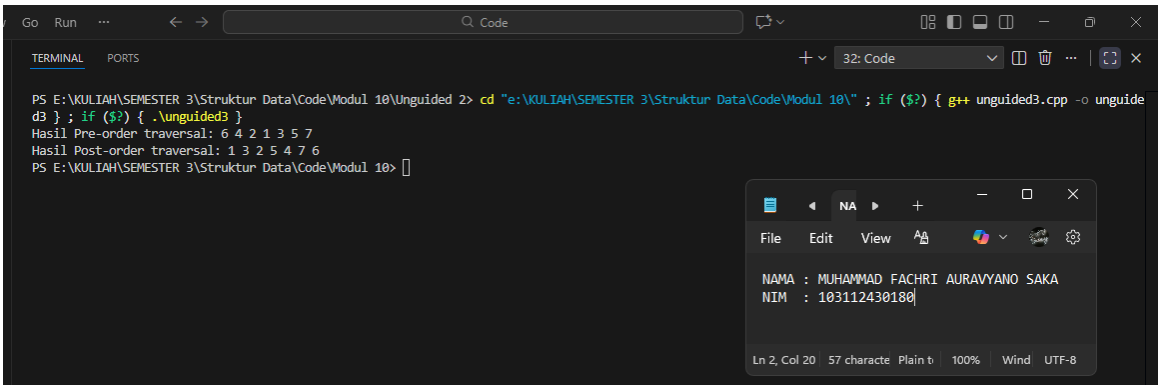

Go Run
Code
unguided3.cpp
Guided.cpp
graf.h
graf.cpp
main.cpp Modul 14
main.cpp ...\Unguided 2
Modul 10 > unguided3.cpp > deleteTree(Node *)
void postOrder(Node* root) {
if (root == NULL) {
return;
}
postOrder(root->left);
postOrder(root->right);
cout << root->data << " ";
}
// Fungsi untuk menghapus tree (membersihkan memori)
void deleteTree(Node* root) {
if (root == NULL) {
return;
}
deleteTree(root->left);
deleteTree(root->right);
delete root;
}
int main() {
// Membangun pohon
// Membuat node
Node* root = createNode(6);
// Membuat subtree kiri
root->left = createNode(4);
root->left->left = createNode(2);
root->left->right = createNode(5);
root->left->left->left = createNode(1);
root->left->left->right = createNode(3);
NA
File Edit View
NAMA : MUHAMMAD FACHRI AURAVYANO SAKA
NIM : 103112430180
Ln 2, Col 20 57 characte Plain t 100% Wind UTF-8
Ln 39, Col 16 Spaces: 4 UTF-8 CRLF C++ Win32 Prettier



```cpp
int main() {
    // Membangun pohon
    // Membuat node
    Node* root = createNode(6);

    // Membuat subtree kiri
    root->left = createNode(4);
    root->left->left = createNode(2);
    root->left->right = createNode(5);
    root->left->left->left = createNode(1);
    root->left->left->right = createNode(3);

    // Membuat subtree kanan
    root->right = createNode(7);

    // Mencetak hasil traversal
    cout << "Hasil Pre-order traversal: ";
    preOrder(root);
    cout << endl;

    cout << "Hasil Post-order traversal: ";
    postOrder(root);
    cout << endl;

    deleteTree(root);

    return 0;
}
```

```
NAMA : MUHAMMAD FACHRI AURAVYANO SAKA
NIM  : 103112430180
```

Screenshot Output



```
PS E:\KULIAH\SEMESTER 3\Struktur Data\Code\Modul 10\Unguided 2> cd "e:\KULIAH\SEMESTER 3\Struktur Data\Code\Modul 10\" ; if ($?) { g++ unguided3.cpp -o unguided3 } ; if ($?) { .\unguided3 }
Hasil Pre-order traversal: 6 4 2 1 3 5 7
Hasil Post-order traversal: 1 3 2 5 4 7 6
PS E:\KULIAH\SEMESTER 3\Struktur Data\Code\Modul 10>
```

```
NAMA : MUHAMMAD FACHRI AURAVYANO SAKA
NIM  : 103112430180
```

Deskripsi:

Program ini merupakan implementasi struktur data Binary Tree sederhana di mana konstruksi pohon dilakukan secara manual (statis). Berbeda dengan dua program sebelumnya yang menggunakan logika penyisipan otomatis (berdasarkan perbandingan nilai lebih kecil/besar), kode ini membangun pohon dengan menghubungkan pointer left dan right secara eksplisit di dalam fungsi main. Program ini berfungsi untuk mendemonstrasikan dua jenis penelusuran (traversal), yaitu Pre-order dan Post-order, serta menyertakan fungsi deleteTree yang penting untuk menghapus seluruh node dari memori (heap) setelah program selesai digunakan.

D. Kesimpulan

Secara keseluruhan, ketiga program ini merepresentasikan alur pembelajaran struktur data Tree yang bertahap, mulai dari pemahaman konstruksi dasar hingga penerapan algoritma optimasi yang kompleks. Pada tahap awal yang ditunjukkan oleh unguided3.cpp, fokus utamanya adalah pemahaman struktural di mana pohon dibangun secara manual (statis) dengan menghubungkan pointer secara eksplisit tanpa aturan pengurutan nilai, yang bertujuan untuk melatih logika dasar tentang bagaimana node saling terhubung dan bagaimana data ditelusuri.

Selanjutnya, pembelajaran berkembang ke logika Binary Search Tree (BST) melalui program bstree.cpp, di mana penyusunan pohon dilakukan secara otomatis berdasarkan nilai data—nilai kecil ke kiri dan besar ke kanan. Program ini mulai memperkenalkan penggunaan fungsi rekursif yang lebih dalam untuk operasi statistik seperti menghitung kedalaman dan total nilai node, namun strukturnya masih rentan menjadi tidak seimbang (miring) bergantung pada urutan input data.

Puncaknya adalah implementasi AVL Tree pada tree.cpp, yang menyempurnakan konsep BST dengan menambahkan mekanisme penyeimbangan otomatis melalui teknik rotasi dan pemantauan ketinggian (height) node. Kode ini menunjukkan tingkat pemahaman lanjutan di mana fokusnya bukan lagi sekadar menyusun data, melainkan menjamin efisiensi performa pencarian, penyisipan, dan penghapusan data agar tetap cepat dan stabil dalam kondisi apa pun.

E. Referensi

Muñoz, D. F. (2024, June). A C++ library for fast simulation of queues and some experimental results. In *AIP Conference Proceedings* (Vol. 3094, No. 1, p. 110002). AIP Publishing LLC.

Goponenko, A., & Carroll, S. (2019). A C++ implementation of a lock-free priority queue based on Multi-Dimensional Linked List. *Link: https://www. researchgate. net/publication/337020321_A_C_Implementation_of_a_Lock-Free _Priority_Queue_Based_on_Multi-Dimensional_Linked_List*.

Malik, D. S. (2010). *Data structures using C++*. USA.