

**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

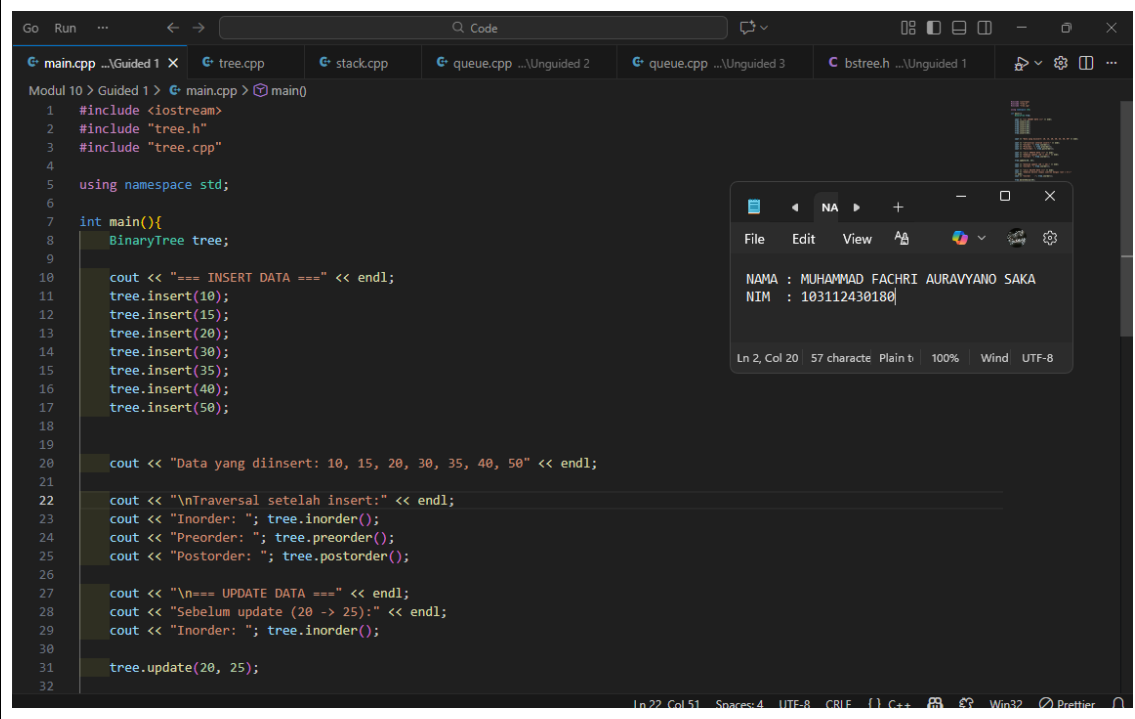
Queue atau antrean adalah struktur data yang bekerja berdasarkan prinsip FIFO (First In, First Out). Konsep ini sangat mirip dengan antrean di dunia nyata, misalnya saat mengantre membeli tiket bioskop: orang yang datang paling awal (masuk pertama) akan dilayani dan keluar paling awal juga. Sebaliknya,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



```
1  #include <iostream>
2  #include "tree.h"
3  #include "tree.cpp"
4
5  using namespace std;
6
7  int main(){
8      BinaryTree tree;
9
10     cout << "=== INSERT DATA ===" << endl;
11     tree.insert(10);
12     tree.insert(15);
13     tree.insert(20);
14     tree.insert(30);
15     tree.insert(35);
16     tree.insert(40);
17     tree.insert(50);
18
19
20     cout << "Data yang diinsert: 10, 15, 20, 30, 35, 40, 50" << endl;
21
22     cout << "\nTraversal setelah insert:" << endl;
23     cout << "Inorder: "; tree.inorder();
24     cout << "Preorder: "; tree.preorder();
25     cout << "Postorder: "; tree.postorder();
26
27     cout << "\n=== UPDATE DATA ===" << endl;
28     cout << "Sebelum update (20 -> 25):" << endl;
29     cout << "Inorder: "; tree.inorder();
30
31     tree.update(20, 25);
32
```

Output:

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

Ln 2, Col 20 57 character Plain text 100% Window UTF-8

```
main.cpp ...\Guided 1 X tree.cpp stack.cpp queue.cpp ...\Unguided 2 queue.cpp ...\Unguided 3 bstree.h ...\Unguided 1
Modul 10 > Guided 1 > main.cpp > main()
22 cout << "Inorder: "; tree.inorder();
23 cout << "Preorder: "; tree.preorder();
24 cout << "Postorder: "; tree.postorder();
25
26
27 cout << "\n=== UPDATE DATA ===" << endl;
28 cout << "Sebelum update (20 -> 25):" << endl;
29 cout << "Inorder: "; tree.inorder();
30
31 tree.update(20, 25);
32
33 cout << "Setelah update (20 -> 25):" << endl;
34 cout << "Inorder: "; tree.inorder();
35
36 cout << "\n=== DELETE DATA ===" << endl;
37 cout << "Sebelum delete (hapus subtree dengan root = 3-):"
38 << endl;
39 cout << "Inorder: "; tree.inorder();
40
41 tree.deleteValue(30);
42
43 cout << "Setelah delete (subtree root = 30 dihapus):"
44 << endl;
45 cout << "Inorder: "; tree.inorder();
46
47 return 0;
48
49 }
```

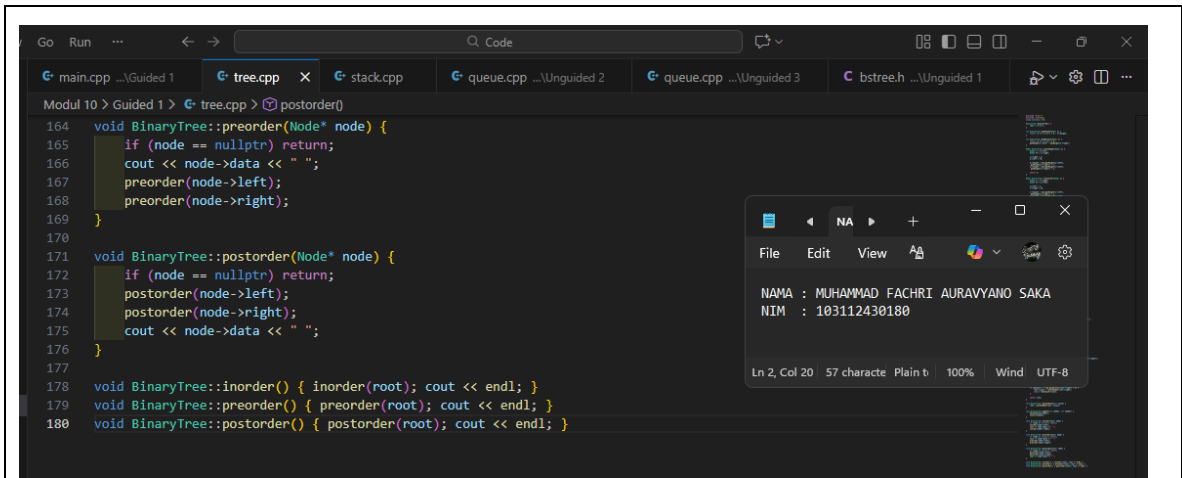
## Tree.cpp

```
Go Run ... tree.cpp X stack.cpp queue.cpp ...\Unguided 2 queue.cpp ...\Unguided 3 bstree.h ...\Unguided 1
Modul 10 > Guided 1 > tree.cpp > postorder()
1 #include "tree.h"
2 #include <iostream>
3 using namespace std;
4
5 BinaryTree::BinaryTree() {
6     root = nullptr;
7 }
8
9 int BinaryTree::getHeight(Node* n) {
10     return (n == nullptr) ? 0 : n->height;
11 }
12
13 int BinaryTree::getBalance(Node* n) {
14     return (n == nullptr) ? 0 :
15         getHeight(n->left) - getHeight(n->right);
16 }
17
18 Node* BinaryTree::rotateRight(Node* y) {
19     Node* x = y->left;
20     Node* T2 = x->right;
21
22     x->right = y;
23     y->left = T2;
24
25     y->height = max(getHeight(y->left),
26         getHeight(y->right)) + 1;
27     x->height = max(getHeight(x->left),
28         getHeight(x->right)) + 1;
29
30     return x;
31 }
32 }
```

```
Go Run ... < -> Code
main.cpp ...\Guided 1 tree.cpp x stack.cpp queue.cpp ...\Unguided 2 queue.cpp ...\Unguided 3 bstree.h ...\Unguided 1
Modul 10 > Guided 1 > tree.cpp > postorder()
32
33 Node* BinaryTree::rotateLeft(Node* x) {
34     Node* y = x->right;
35     Node* T2 = y->left;
36
37     y->left = x;
38     x->right = T2;
39
40     x->height = max(getHeight(x->left),
41                     getHeight(x->right)) + 1;
42     y->height = max(getHeight(y->left),
43                     getHeight(y->right)) + 1;
44
45     return y;
46 }
47
48 Node* BinaryTree::insertNode(Node* node, int value) {
49     if (node == nullptr) {
50         Node* newNode = new Node(value, nullptr, nullptr, 1);
51         return newNode;
52     }
53
54     if (value < node->data)
55         node->left = insertNode(node->left, value);
56     else if (value > node->data)
57         node->right = insertNode(node->right, value);
58     else
59         return node;
60
61     node->height = 1 + max(getHeight(node->left),
62                           getHeight(node->right));
63 }
```

```
Go Run ... < -> Code
main.cpp ...\Guided 1 tree.cpp x stack.cpp queue.cpp ...\Unguided 2 queue.cpp ...\Unguided 3 bstree.h ...\Unguided 1
Modul 10 > Guided 1 > tree.cpp > postorder()
48 Node* BinaryTree::insertNode(Node* node, int value) {
64     int balance = getBalance(node);
65
66     if (balance > 1 && value < node->left->data)
67         return rotateRight(node);
68
69     if (balance < -1 && value > node->right->data)
70         return rotateLeft(node);
71
72     if (balance > 1 && value > node->left->data) {
73         node->left = rotateLeft(node->left);
74         return rotateRight(node);
75     }
76
77     if (balance < -1 && value < node->right->data) {
78         node->right = rotateRight(node->right);
79         return rotateLeft(node);
80     }
81
82     return node;
83 }
84
85 void BinaryTree::insert(int value) {
86     root = insertNode(root, value);
87 }
88
89 Node* BinaryTree::minValueNode(Node* node) {
90     Node* current = node;
91     while (current->left != nullptr)
92         current = current->left;
93     return current;
94 }
```





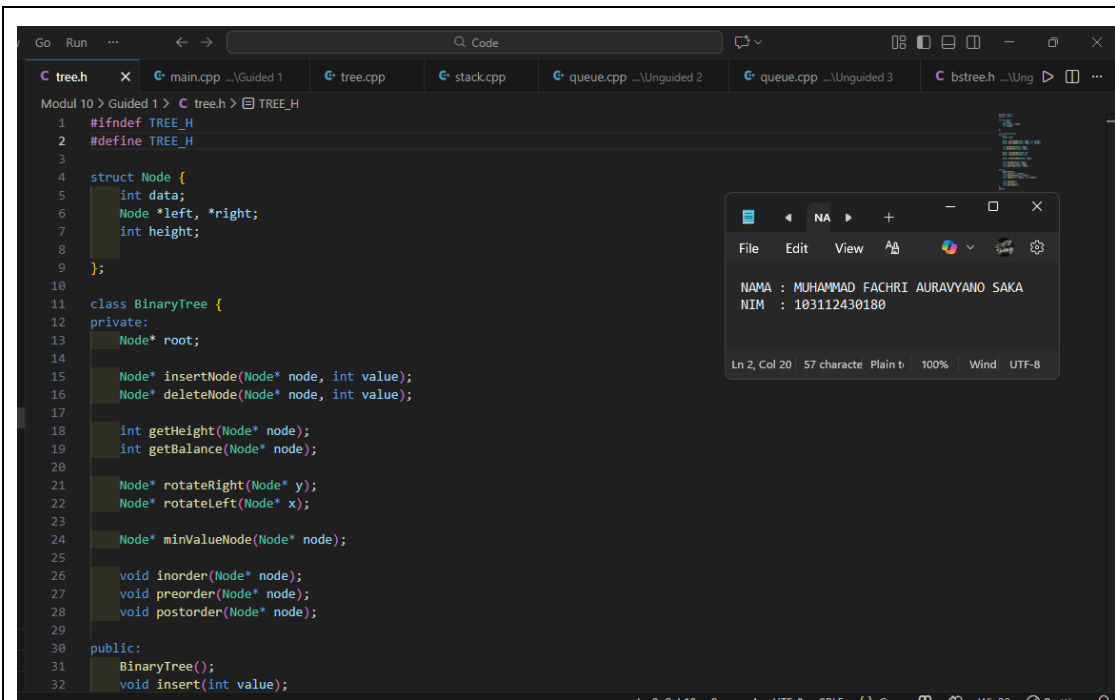
```
164 void BinaryTree::preorder(Node* node) {
165     if (node == nullptr) return;
166     cout << node->data << " ";
167     preorder(node->left);
168     preorder(node->right);
169 }
170
171 void BinaryTree::postorder(Node* node) {
172     if (node == nullptr) return;
173     postorder(node->left);
174     postorder(node->right);
175     cout << node->data << " ";
176 }
177
178 void BinaryTree::inorder() { inorder(root); cout << endl; }
179 void BinaryTree::preorder() { preorder(root); cout << endl; }
180 void BinaryTree::postorder() { postorder(root); cout << endl; }
```

File Edit View A A 100% Wind UTF-8

NAMA : MUHAMMAD FACHRI AURAVYANO SAKA  
NIM : 103112430180

Ln 2, Col 20 57 character Plain t 100% Wind UTF-8

## Tree.h

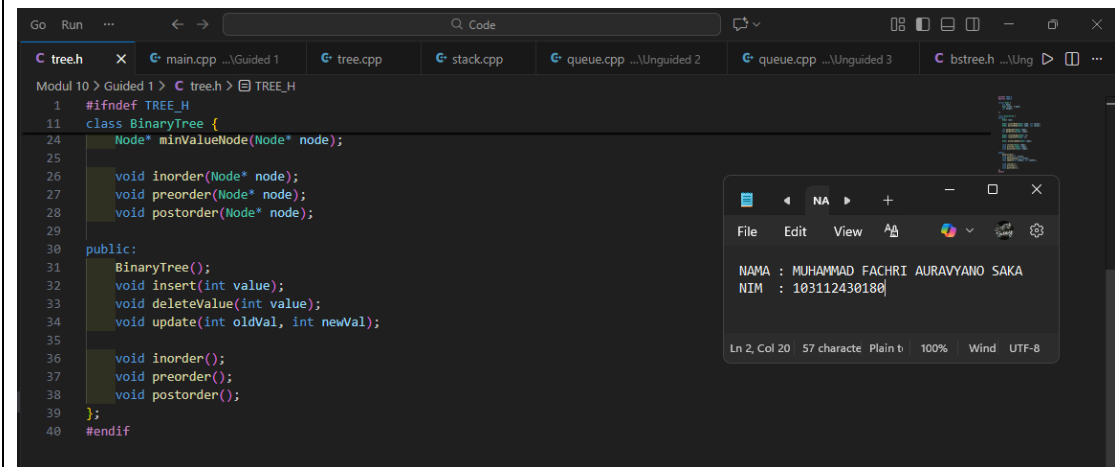


```
1 #ifndef TREE_H
2 #define TREE_H
3
4 struct Node {
5     int data;
6     Node *left, *right;
7     int height;
8 };
9
10
11 class BinaryTree {
12 private:
13     Node* root;
14
15     Node* insertNode(Node* node, int value);
16     Node* deleteNode(Node* node, int value);
17
18     int getHeight(Node* node);
19     int getBalance(Node* node);
20
21     Node* rotateRight(Node* y);
22     Node* rotateLeft(Node* x);
23
24     Node* minValueNode(Node* node);
25
26     void inorder(Node* node);
27     void preorder(Node* node);
28     void postorder(Node* node);
29
30 public:
31     BinaryTree();
32     void insert(int value);
```

File Edit View A A 100% Wind UTF-8

NAMA : MUHAMMAD FACHRI AURAVYANO SAKA  
NIM : 103112430180

Ln 2, Col 20 57 character Plain t 100% Wind UTF-8



```
24 Node* minValueNode(Node* node);
25
26 void inorder(Node* node);
27 void preorder(Node* node);
28 void postorder(Node* node);
29
30 public:
31     BinaryTree();
32     void insert(int value);
33     void deleteValue(int value);
34     void update(int oldVal, int newVal);
35
36     void inorder();
37     void preorder();
38     void postorder();
39 };
40 #endif
```

File Edit View A A 100% Wind UTF-8

NAMA : MUHAMMAD FACHRI AURAVYANO SAKA  
NIM : 103112430180

Ln 2, Col 20 57 character Plain t 100% Wind UTF-8

```
PS E:\KULIAH\SEMESTER 3\Struktur Data\Code> cd "e:\KULIAH\SEMESTER 3\Struktur Data\Code\Modul 10\Guided 1\"; if ($?) { g++ main.cpp -o main }; if ($?) { .\ma
in }
=== INSERT DATA ===
Data yang diinsert: 10, 15, 20, 30, 35, 40, 50

Traversial setelah insert:
Inorder: 10 15 20 30 35 40 50
Preorder: 30 15 10 20 40 35 50
Postorder: 10 20 15 35 50 40 30

=== UPDATE DATA ===
Sebelum update (20 -> 25):
Inorder: 10 15 20 30 35 40 50
Setelah update (20 -> 25):
Inorder: 10 15 25 30 35 40 50

=== DELETE DATA ===
Sebelum delete (hapus subtree dengan root = 3-):
Inorder :10 15 25 30 35 40 50
Setelah delete (subtree root = 30 dihapus):
Inorder :10 15 25 35 40 50
PS E:\KULIAH\SEMESTER 3\Struktur Data\Code\Modul 10\Guided 1>
```

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

```
Modul 10 > Unguided 1 > main.cpp > main()
1 #include <iostream>
2 #include "bstree.h"
3 #include "bstree.cpp"
4 using namespace std;
5
6 int main() {
7     cout << "Hello World" << endl;
8     address root = Nil;
9
10    insertNode(root, 1);
11    insertNode(root, 2);
12    insertNode(root, 6);
13    insertNode(root, 4);
14    insertNode(root, 5);
15    insertNode(root, 3);
16    insertNode(root, 6);
17    insertNode(root, 7);
18
19    InOrder(root);
20
21    cout << endl;
22    return 0;
23 }
```

bstree.cpp

```
Go Run ... < -> Code
bstree.cpp ...Unguided 1 X bstree.h ...Unguided 2 bstree.cpp ...Unguided 2 unguided3.cpp Guided.cpp graf.h
Modul 10 > Unguided 1 > bstree.cpp > ...
1 #include "bstree.h"
2 address alokasi(intofype x) {
3     address P = new Node;
4     P->info = x;
5     P->left = Nil;
6     P->right = Nil;
7     return P;
8 }
9
10 void insertNode(address &root, intofype x) {
11
12     if (root == Nil) {
13         root = alokasi(x);
14     }
15     else {
16         if (x < root->info) {
17             insertNode(root->left, x);
18         }
19         else if (x > root->info) {
20             insertNode(root->right, x);
21         }
22     }
23 }
24
25 void InOrder(address root) {
26     if (root != Nil) {
27         InOrder(root->left);
28         cout << root->info << " - ";
29         InOrder(root->right);
30     }
31 }
32
```

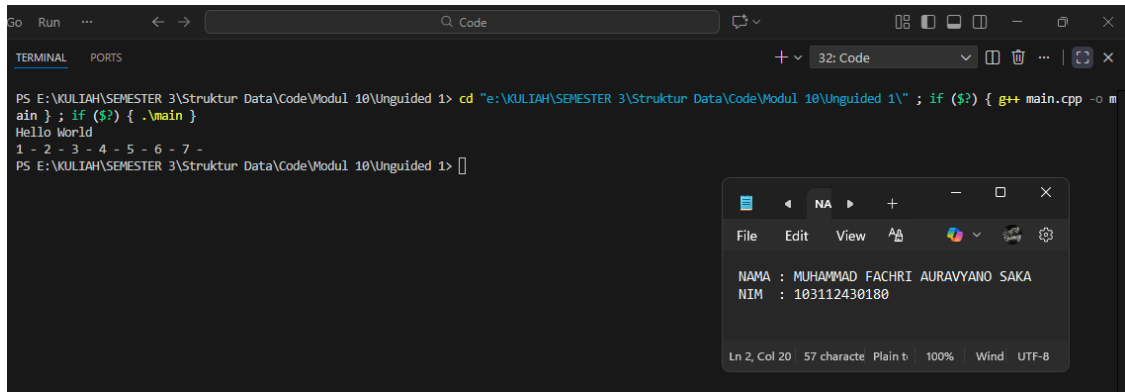
```
Go Run ... < -> Code
bstree.cpp ...Unguided 1 X bstree.h ...Unguided 2 bstree.cpp ...Unguided 2 unguided3.cpp Guided.cpp graf.h
Modul 10 > Unguided 1 > bstree.cpp > ...
10 void insertNode(address &root, intofype x) {
11     else {
23 }
24
25 void InOrder(address root) {
26     if (root != Nil) {
27         InOrder(root->left);
28         cout << root->info << " - ";
29         InOrder(root->right);
30     }
31 }
32
33 address findNode(intofype x, address root) {
34     if (root == Nil) return Nil;
35     if (root->info == x) return root;
36     if (x < root->info) {
37         return findNode(x, root->left);
38     } else {
39         return findNode(x, root->right);
40     }
41 }
```

## bstree.h

```
Go Run ... < -> Code
C bstree.h ...Unguided 1 X bstree.cpp ...Unguided 1 bstree.h ...Unguided 2 bstree.cpp ...Unguided 2 unguided3.cpp Guided.cpp
Modul 10 > Unguided 1 > C bstree.h > ...
1 #ifndef BSTREE_H
2 #define BSTREE_H
3 #include <iostream>
4 #define Nil NULL
5 using namespace std;
6
7 typedef int intofype;
8 typedef struct Node *address;
9
10 struct Node {
11     intofype info;
12     address left;
13     address right;
14 };
15
16 address alokasi(intofype x);
17 void insertNode(address &root, intofype x);
18 void InOrder(address root);
19
20 address findNode(intofype x, address root);
21
22 #endif
```



## Screenshot Output



```
PS E:\KULIAH\SEMESTER 3\Struktur Data\Code\Modul 10\Unguided 1> cd "e:\KULIAH\SEMESTER 3\Struktur Data\Code\Modul 10\Unguided 1\" ; if ($?) { g++ main.cpp -o m
ain } ; if ($?) { .\main }
Hello World
1 - 2 - 3 - 4 - 5 - 6 - 7 -
PS E:\KULIAH\SEMESTER 3\Struktur Data\Code\Modul 10\Unguided 1> []
```

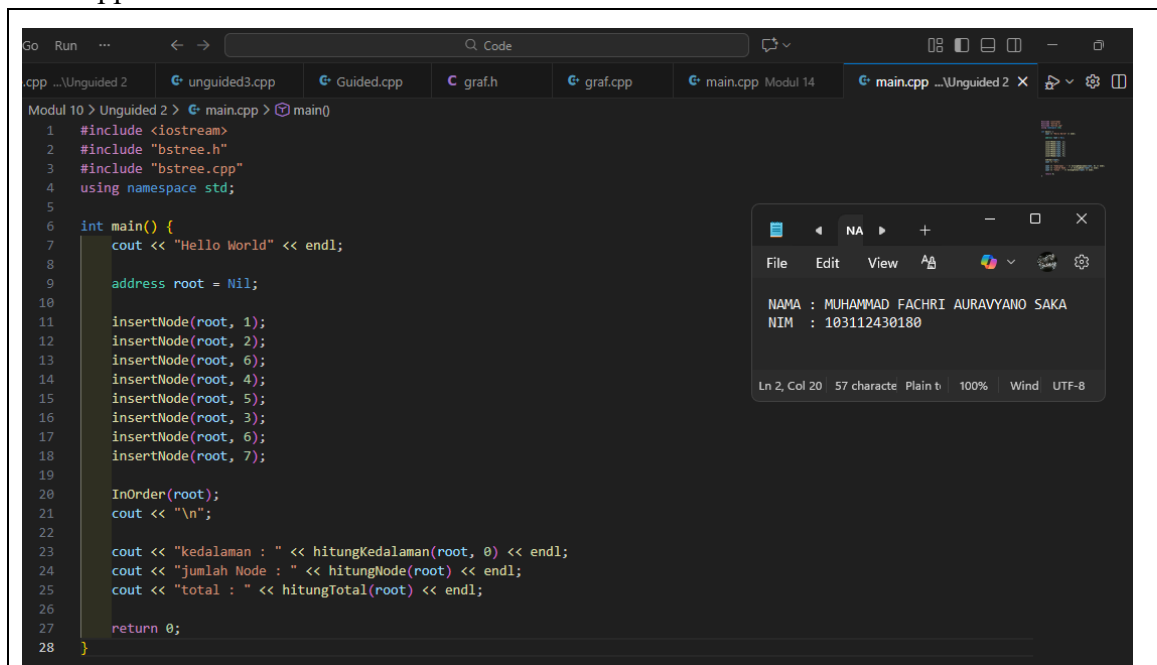
NAMA : MUHAMMAD FACHRI AURAVYANO SAKA  
NIM : 103112430180

## 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 secaraurut (menaik) menggunakan penelusuran `InOrder`.

## Unguided 2

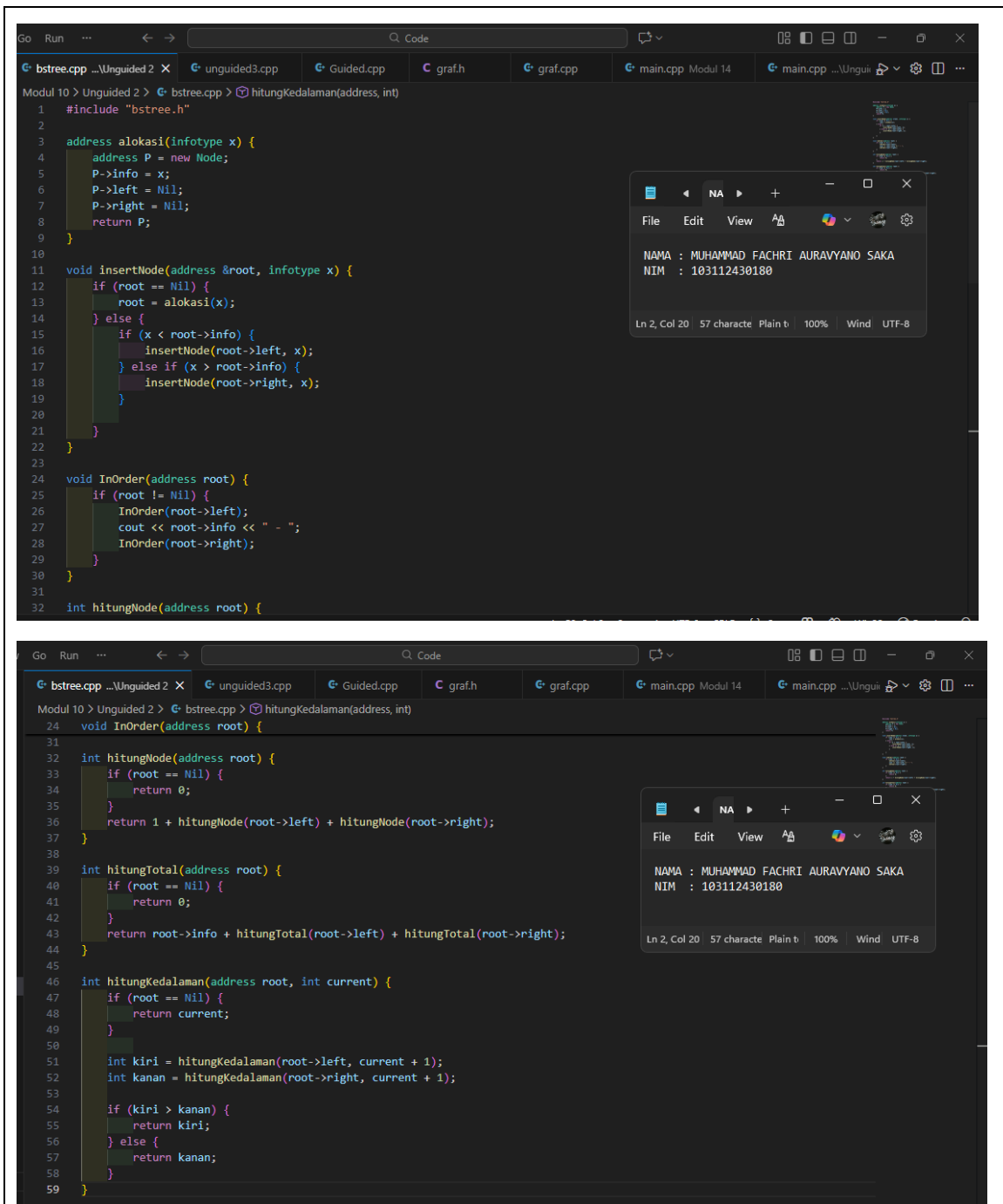
### Main.cpp



```
Modul 10 > Unguided 2 > main.cpp > main()
1 #include <iostream>
2 #include "bstree.h"
3 #include "bstree.cpp"
4 using namespace std;
5
6 int main() {
7     cout << "Hello World" << endl;
8
9     address root = Nil;
10
11     insertNode(root, 1);
12     insertNode(root, 2);
13     insertNode(root, 6);
14     insertNode(root, 4);
15     insertNode(root, 5);
16     insertNode(root, 3);
17     insertNode(root, 6);
18     insertNode(root, 7);
19
20     InOrder(root);
21     cout << "\n";
22
23     cout << "kedalaman : " << hitungKedalaman(root, 0) << endl;
24     cout << "jumlah Node : " << hitungNode(root) << endl;
25     cout << "total : " << hitungTotal(root) << endl;
26
27     return 0;
28 }
```

NAMA : MUHAMMAD FACHRI AURAVYANO SAKA  
NIM : 103112430180

### Bstree.cpp



Bstree.h

```
1 #ifndef BSTREE_H
2 #define BSTREE_H
3 #include <iostream>
4 #define Nil NULL
5 using namespace std;
6
7 typedef int infotype;
8 typedef struct Node *address;
9
10 struct Node {
11     infotype info;
12     address left;
13     address right;
14 };
15
16 address alokasi(infotype x);
17 void insertNode(address &root, infotype x);
18 void InOrder(address root);
19
20 int hitungNode(address root);
21 int hitungTotal(address root);
22 int hitungKedalaman(address root, int current);
23
24 #endif
```

## Screenshot Output

```
PS E:\KULIAH\SEMESTER 3\Struktur Data\Code\Modul 10\Unguided 1> cd "e:\KULIAH\SEMESTER 3\Struktur Data\Code\Modul 10\Unguided 2\" ; if ($?) { g++ main.cpp -o main ; if ($?) { .\main }
Hello World
1 - 2 - 3 - 4 - 5 - 6 - 7 -
kedalaman : 5
jumlah Node : 7
total : 28
PS E:\KULIAH\SEMESTER 3\Struktur Data\Code\Modul 10\Unguided 2>
```

## 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

The screenshot shows a C++ IDE with a file explorer at the top displaying a project structure with folders like 'src' and 'include', and files like 'main.cpp', 'graf.h', and 'graf.cpp'. The main editor window shows the code for 'unguided3.cpp' at line 10, which includes a function call to 'deleteTree(Node\*)'. The code defines a 'Node' struct with 'data', 'left', and 'right' pointers, and implements 'createNode', 'preOrder', and 'postOrder' functions. A terminal window on the right shows the output of a program, displaying the name 'MUHAMMAD FACHRI AURAVYANO SAKA' and the NIM '103112430180'.

```

1  #include <iostream>
2  using namespace std;
3
4  struct Node {
5      int data;
6      Node* left;
7      Node* right;
8  };
9
10 Node* createNode(int data) {
11     Node* newNode = new Node();
12     newNode->data = data;
13     newNode->left = NULL;
14     newNode->right = NULL;
15     return newNode;
16 }
17
18 void preOrder(Node* root) {
19     if (root == NULL) {
20         return;
21     }
22     cout << root->data << " ";
23     preOrder(root->left);
24     preOrder(root->right);
25 }
26
27 void postOrder(Node* root) {
28     if (root == NULL) {
29         return;
30     }
31     postOrder(root->left);
32     postOrder(root->right);

```

```

NAMA : MUHAMMAD FACHRI AURAVYANO SAKA
NIM : 103112430180

```

Ln 2, Col 20 57 character Plain text 100% Windows UTF-8

The screenshot shows a C++ IDE with a file explorer at the top displaying a project structure with folders 'src' and 'build', and files 'main.cpp', 'graf.h', 'graf.cpp', 'unguided3.cpp', and 'guided.cpp'. The main editor window shows the code for 'unguided3.cpp' with the following content:

```

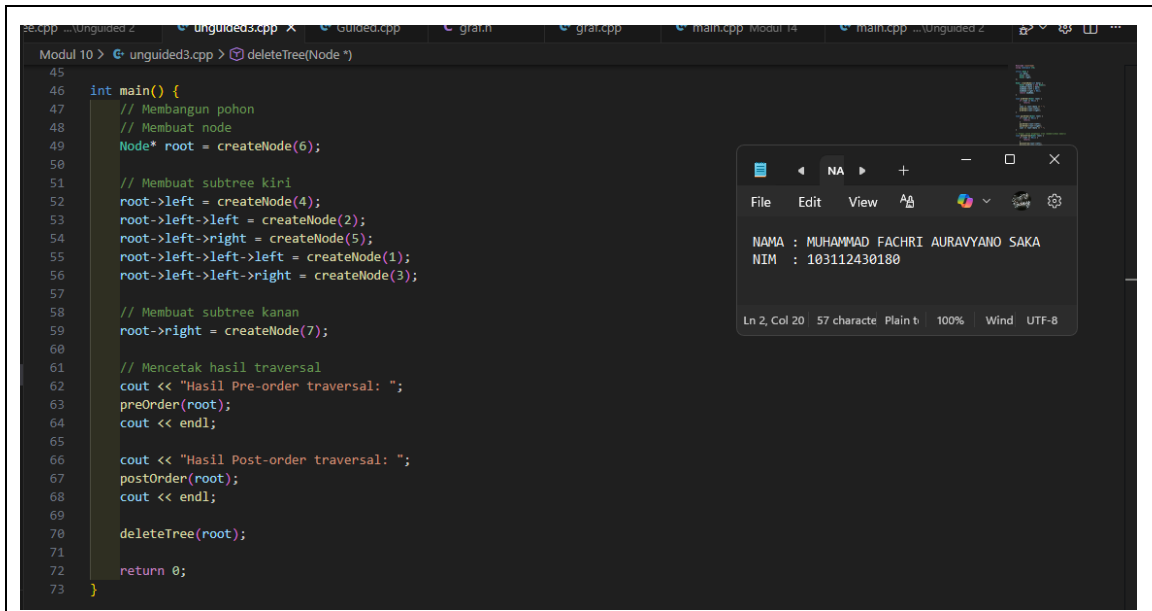
26
27 void postOrder(Node* root) {
28     if (root == NULL) {
29         return;
30     }
31     postOrder(root->left);
32     postOrder(root->right);
33     cout << root->data << " ";
34 }
35
36 // Fungsi untuk menghapus tree (membersihkan memori)
37 void deleteTree(Node* root) {
38     if (root == NULL) {
39         return;
40     }
41     deleteTree(root->left);
42     deleteTree(root->right);
43     delete root;
44 }
45
46 int main() {
47     // Membangun pohon
48     // Membuat node
49     Node* root = createNode(6);
50
51     // Membuat subtree kiri
52     root->left = createNode(4);
53     root->left->left = createNode(2);
54     root->left->right = createNode(5);
55     root->left->left->left = createNode(1);
56     root->left->left->right = createNode(3);
57

```

A student information popup window is overlaid on the right side of the editor. It contains the following text:

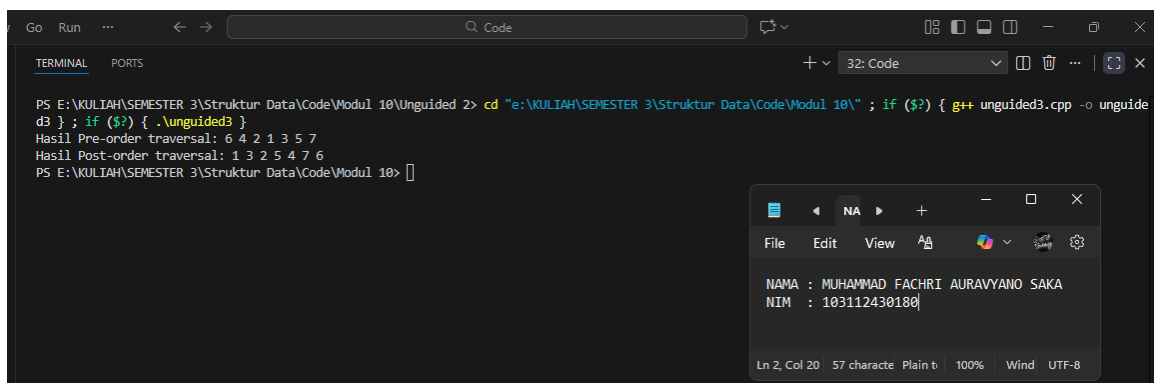
NAMA : MUHAMMAD FACHRI AURAVYANO SAKA  
 NIM : 103112430180

The popup window has a title bar with standard window controls and a menu bar with options: File, Edit, View, and a settings icon. The status bar at the bottom of the popup indicates: Ln 2, Col 20, 57 character, Plain text, 100%, Windows, UTF-8.



```
Modul 10 > uguided3.cpp > deleteTree(Node *)
45
46 int main() {
47     // Membangun pohon
48     // Membuat node
49     Node* root = createNode(6);
50
51     // Membuat subtree kiri
52     root->left = createNode(4);
53     root->left->left = createNode(2);
54     root->left->right = createNode(5);
55     root->left->left->left = createNode(1);
56     root->left->left->right = createNode(3);
57
58     // Membuat subtree kanan
59     root->right = createNode(7);
60
61     // Mencetak hasil traversal
62     cout << "Hasil Pre-order traversal: ";
63     preOrder(root);
64     cout << endl;
65
66     cout << "Hasil Post-order traversal: ";
67     postOrder(root);
68     cout << endl;
69
70     deleteTree(root);
71
72     return 0;
73 }
```

## Screenshot Output



```
Go Run ... < -> Code
TERMINAL PORTS
PS E:\KULIAH\SEMESTER 3\Struktur Data\Code\Modul 10\Unguided 2> cd "e:\KULIAH\SEMESTER 3\Struktur Data\Code\Modul 10\" ; if ($?) { g++ uguided3.cpp -o uguide
d3 } ; if ($?) { ./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> 
```

## 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 uguided3.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](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.