

**LAPORAN PRAKTIKUM
STRUKTUR DATA**

**MODUL 10
TREE (BAGIAN PERTAMA)**



Disusun Oleh :

NAMA : Muhammad Azzam Satria

NIM : 103112400112

Dosen

FAHRUDIN MUKTI WIBOWO S. Kom., M. Eng

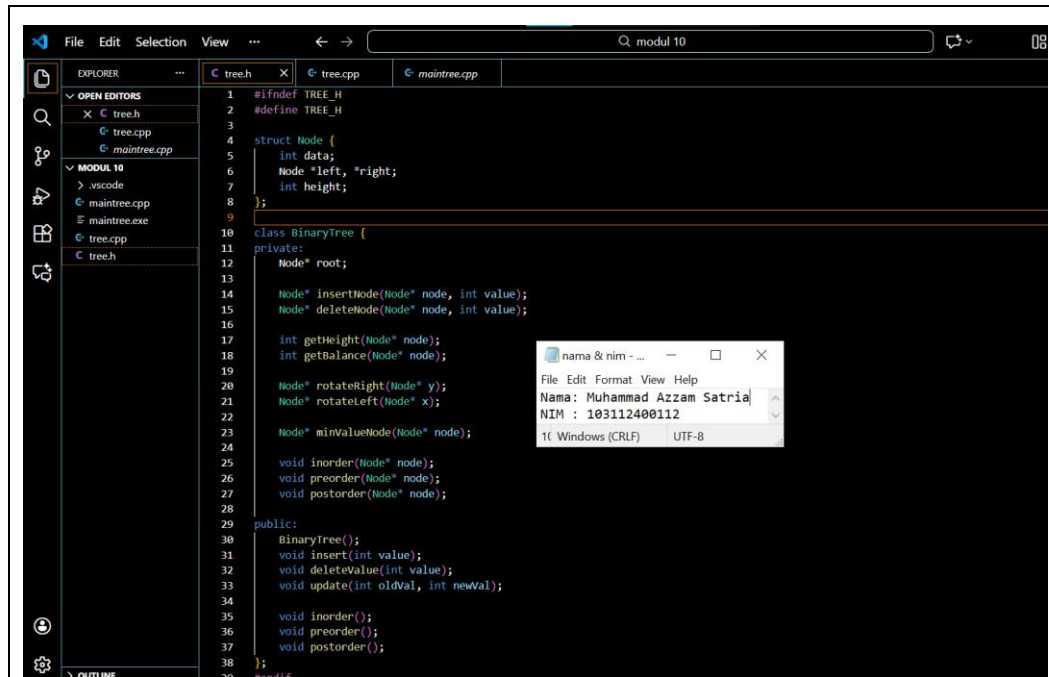
**PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025**

A. Dasar Teori

Struktur data tree adalah suatu bentuk penyimpanan data yang non-linier dan terdiri dari node yang saling terhubung dengan istilahnya yaitu parent dan child. Pada tree node pertama pada pohon akar disebut root sedangkan node yang tidak mempunyai akar disebut leaf. Setiap node tersebut bisa memiliki satu atau lebih child yang terhubung melalui cabang. Bentuk tree di modul ini ada 2 yaitu Binary Tree yang setiap nodenya maksimal hanya mempunyai dua child dan BinarySearchTree yaitu tree dengan aturan nilai yang lebih kecil akan ditempatkan di child sebelah kiri dan nilai yang lebih besar ditempatkan di child sebelah kanan.

Di dalam modul terdapat beberapa operasi dasar diantaranya yaitu insert, search dan traversal untuk membantu dalam menyimpan dan menelusuri data di dalam struktur tree. Cara kerja penambahan dan pencarian data yaitu dengan membandingkan nilai pada setiap node. Traversal seperti inoorder, preorder dan postorder memiliki fungsi untuk menampilkan isi tree sesuai yang dijalankan. Penggunaan algoritma rekursif yang terdapat pada modul membuat traversal di suatu program dapat dibuat menjadi lebih sederhana. Akan tetapi, teknik ini memiliki kelemahan yaitu jika tree terlalu dalam, maka akan terjadi pemanggilan fungsi yang berlebihan. Sehingga kondisi tersebut dapat memicu terjadinya overflow pada program.

B. Guided



```
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 class BinaryTreeNode {
11 private:
12     Node* root;
13
14     Node* insertNode(Node* node, int value);
15     Node* deleteNode(Node* node, int value);
16
17     int getHeight(Node* node);
18     int getBalance(Node* node);
19
20     Node* rotateRight(Node* y);
21     Node* rotateLeft(Node* x);
22
23     Node* minValueNode(Node* node);
24
25     void inorder(Node* node);
26     void preorder(Node* node);
27     void postorder(Node* node);
28
29 public:
30     BinaryTreeNode();
31     void insert(int value);
32     void deleteValue(int value);
33     void update(int oldVal, int newVal);
34
35     void inorder();
36     void preorder();
37     void postorder();
38 };
39 #endif
```

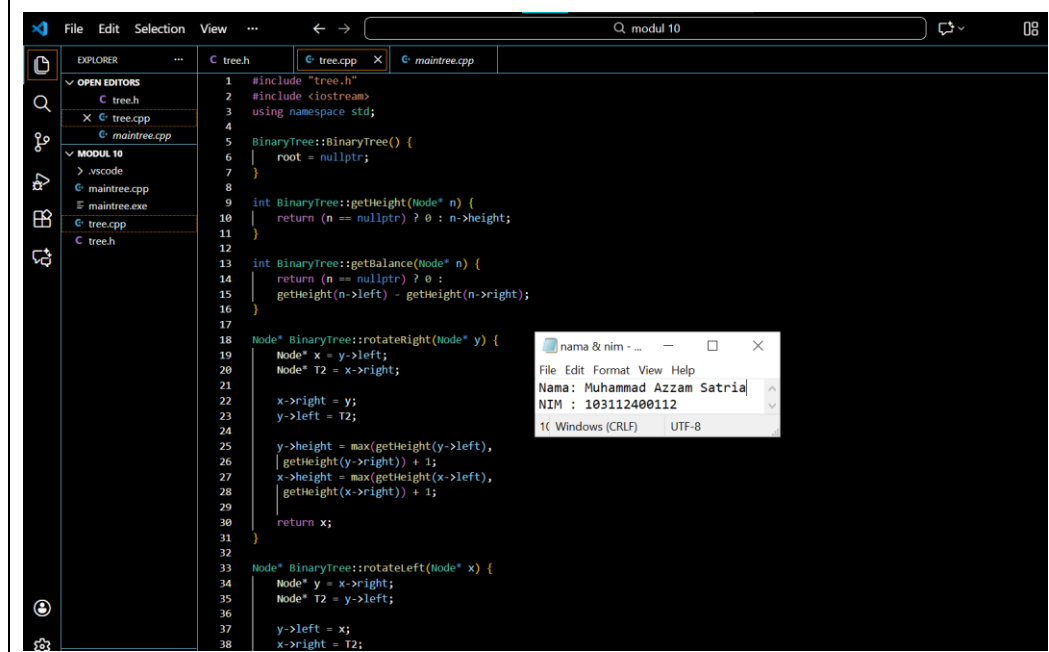
nama & nim - ...

File Edit Format View Help

Nama: Muhammad Azzam Satria

NIM : 103112400112

1(Windows (CRLF) UTF-8



```
1 #include "tree.h"
2 #include <iostream>
3 using namespace std;
4
5 BinaryTreeNode::BinaryTreeNode() {
6     root = nullptr;
7 }
8
9 int BinaryTreeNode::getHeight(Node* n) {
10     return (n == nullptr) ? 0 : n->height;
11 }
12
13 int BinaryTreeNode::getBalance(Node* n) {
14     return (n == nullptr) ? 0 :
15         getHeight(n->left) - getHeight(n->right);
16 }
17
18 Node* BinaryTreeNode::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
33 Node* BinaryTreeNode::rotateLeft(Node* x) {
34     Node* y = x->right;
35     Node* T2 = y->left;
36
37     y->left = x;
38     x->right = T2;
```

nama & nim - ...

File Edit Format View Help

Nama: Muhammad Azzam Satria

NIM : 103112400112

1(Windows (CRLF) UTF-8

```
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
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 }
95
96 Node* BinaryTree::deleteNode(Node* root, int key) {
97     if (root == nullptr)
98         return root;
99
100     if (key < root->data)
101         root->left = deleteNode(root->left, key);
102     else if (key > root->data)
103         root->right = deleteNode(root->right, key);
104     else {
105         if ((root->left == nullptr) || (root->right == nullptr)) {
106             Node* temp = root->left ? root->left : root->right;
107
108             if (temp == nullptr) {
109                 temp = root;
110                 root = nullptr;
111             } else {
112                 *root = *temp;
113             }
114             delete temp;
115         } else {
116             Node* temp = minValueNode(root->right);
117             root->data = temp->data;
```

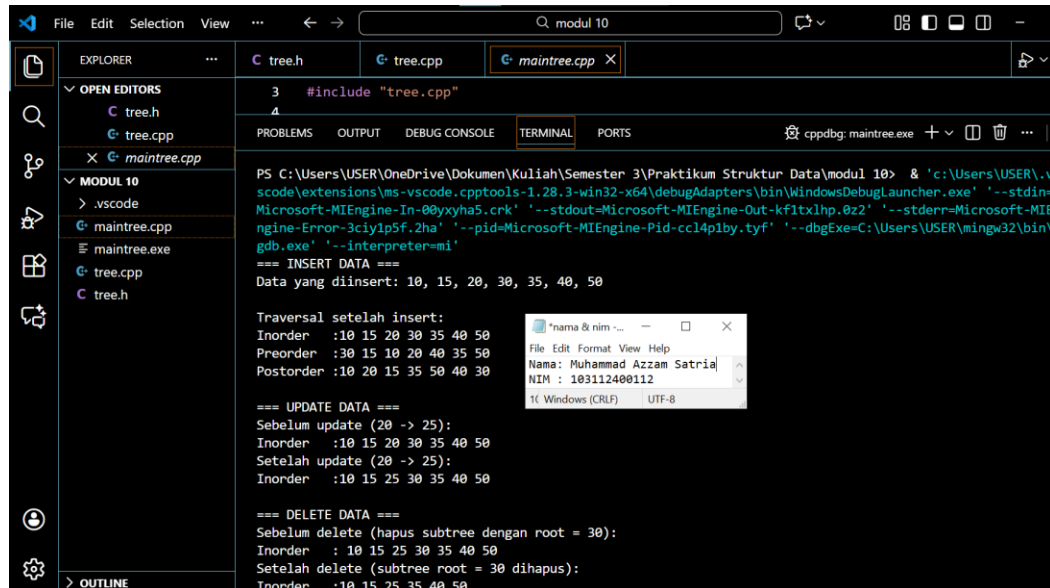
```
118         root->right = deleteNode(root->right, temp->data);
119     }
120 }
121
122 if (root == nullptr)
123     return root;
124
125 root->height = 1 + max(getHeight(root->left), getHeight(root->right));
126
127 int balance = getBalance(root);
128
129 if (balance > 1 && getBalance(root->left) >= 0)
130     return rotateRight(root);
131
132 if (balance > 1 && getBalance(root->left) < 0) {
133     root->left = rotateLeft(root->left);
134     return rotateRight(root);
135 }
136
137 if (balance < -1 && getBalance(root->right) <= 0)
138     return rotateLeft(root);
139
140 if (balance < -1 && getBalance(root->right) > 0) {
141     root->right = rotateRight(root->right);
142     return rotateLeft(root);
143 }
144
145 return root;
146
147 void BinaryTree::deleteValue(int value) {
148     root = deleteNode(root, value);
149 }
150
151
152 void BinaryTree::update(int oldVal, int newVal) {
153     deleteValue(oldVal);
154     insert(newVal);
155 }
```

```
157 void BinaryTree::inorder(Node* node) {
158     if (node == nullptr) return;
159     inorder(node->left);
160     cout << node->data << " ";
161     inorder(node->right);
162 }
163
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; }
181 }
```

```
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    cout << "Data yang diinsert: 10, 15, 20, 30, 35, 40, 50" << endl;
20
21    cout << "\nTraversal setelah insert:" << endl;
22    cout << "Inorder   :"; tree.inorder();
23    cout << "Preorder  :"; tree.preorder();
24    cout << "Postorder :"; tree.postorder();
25
26    cout << "\n=== UPDATE DATA ===" << endl;
```

```
27    cout << "Sebelum update (20 -> 25):" << endl;
28    cout << "Inorder   :"; tree.inorder();
29
30    tree.update(20, 25);
31
32    cout << "Setelah update (20 -> 25):" << endl;
33    cout << "Inorder   :"; tree.inorder();
34
35    cout << "\n=== DELETE DATA ===" << endl;
36    cout << "Sebelum delete (hapus subtree dengan root = 30):" << endl;
37    cout << "Inorder   :"; tree.inorder();
38
39    tree.deleteValue(30);
40
41    cout << "Setelah delete (subtree root = 30 dihapus):" << endl;
42    cout << "Inorder   :"; tree.inorder();
43
44    return 0;
45 }
```

Screenshots Output



```
3 #include "tree.h"
4
5 int main()
6 {
7     // Insert Data
8     cout << "=== INSERT DATA ===" << endl;
9     Data yang diinsert: 10, 15, 20, 30, 35, 40, 50
10
11     // Traversal setelah insert
12     cout << "Traversal setelah insert:" << endl;
13     Inorder :10 15 20 30 35 40 50
14     Preorder :30 15 10 20 40 35 50
15     Postorder :10 20 15 35 50 40 30
16
17     // Update Data
18     cout << "=== UPDATE DATA ===" << endl;
19     Sebelum update (20 -> 25):
20     Inorder :10 15 20 30 35 40 50
21     Setelah update (20 -> 25):
22     Inorder :10 15 25 30 35 40 50
23
24     // Delete Data
25     cout << "=== DELETE DATA ===" << endl;
26     Sebelum delete (hapus subtree dengan root = 30):
27     Inorder :10 15 25 30 35 40 50
28     Setelah delete (subtree root = 30 dihapus):
29     Inorder :10 15 25 35 40 50
30 }
```

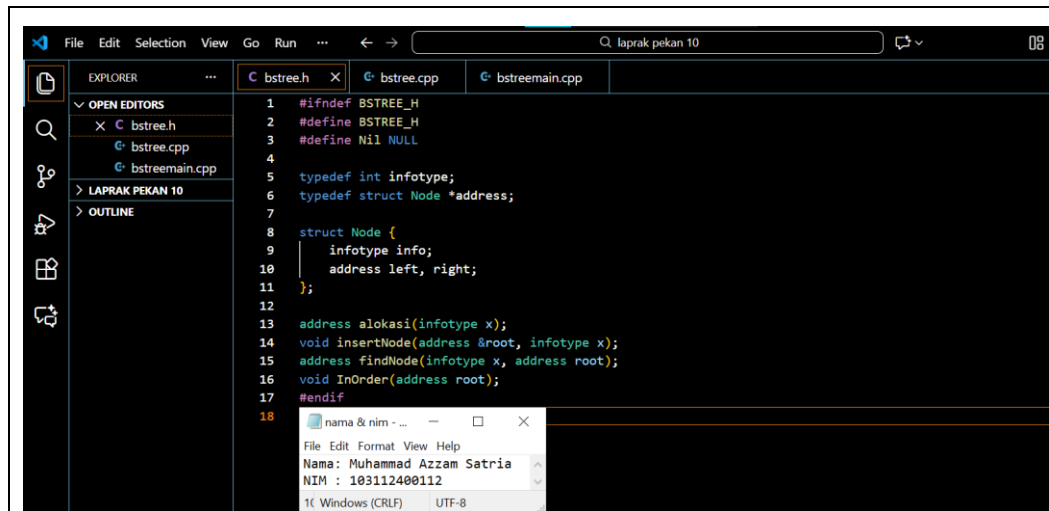
Deskripsi:

Program ini mengimplementasikan struktur data tree dengan menggunakan konsep ADT (Abstract Data Type). Setiap node pada tree mempunyai tiga bagian yaitu data bertipe integer, pointer ke anak sebelah kiri dan pointer ke anak sebelah kanan. Tinggi setiap node disimpan untuk membantu melakukan pengecekan keseimbangan tree. Fungsi insert() yaitu untuk menambahkan data baru ke dalam tree yang kemudian akan dilakukan pengecekan agar tree tetap seimbang. Jika struktur tree belum seimbang, maka akan dilakukan rotasi kiri, kanan atau kombinasi dari keduanya. Proses delete() cara kerjanya sama seperti fungsi insert, namun proses ini diawali dengan pencarian node pengganti seperti nilai terkecil di subtree kanan. Pada program juga mempunyai fungsi inorder(), preorder() dan postorder() untuk menampilkan isi tree berdasarkan jenis traversal yang dipilih.

Selain operasi dasar insert dan delete, program ini mempunyai fungsi update() yang cara kerjanya menghapus nilai lama dan menambahkan nilai baru. Kemudian program mempunyai fungsi tambahan yaitu getHeight() dan getBalance() untuk memudahkan dalam melakukan pengecekan kondisi tree setelah menjalankan fungsi update(). Pada file maintree.cpp program menjalankan berbagai operasi seperti menambah data, mengubah nilai data, menghapus data dan menampilkan isi tree. Output program ini mencetak kondisi

tree yaitu berupa data yang diinsert, traversal setelah insert, update data dan delete data.

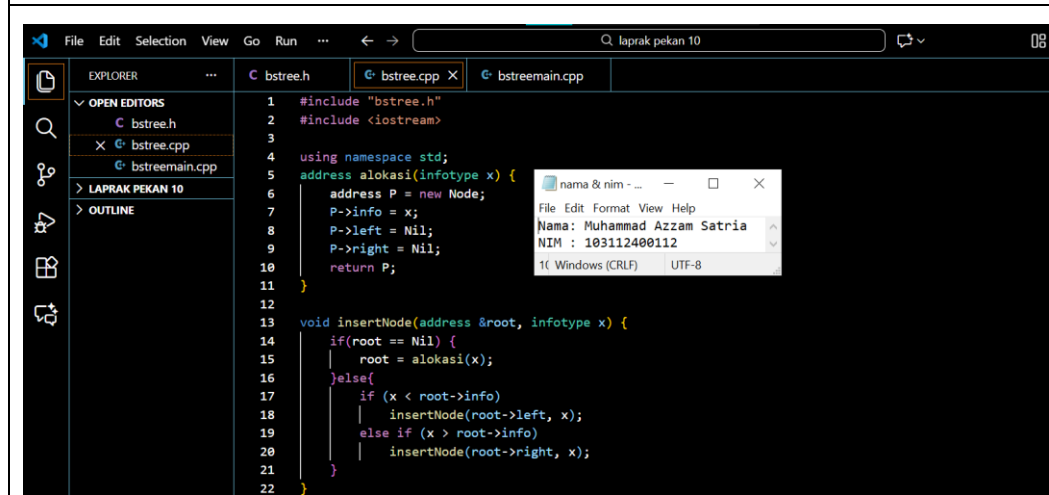
C. Unguied 1



This screenshot shows the Visual Studio Code interface with the `bstree.h` file open. The Explorer sidebar on the left shows the project structure with files `bstree.h`, `bstree.cpp`, and `bstreemain.cpp`. The main editor displays the following C++ code:

```
1 #ifndef BSTREE_H
2 #define BSTREE_H
3 #define Nil NULL
4
5 typedef int infotype;
6 typedef struct Node *address;
7
8 struct Node {
9     infotype info;
10    address left, right;
11 };
12
13 address alokasi(infotype x);
14 void insertNode(address &root, infotype x);
15 address findNode(infotype x, address root);
16 void InOrder(address root);
17 #endif
```

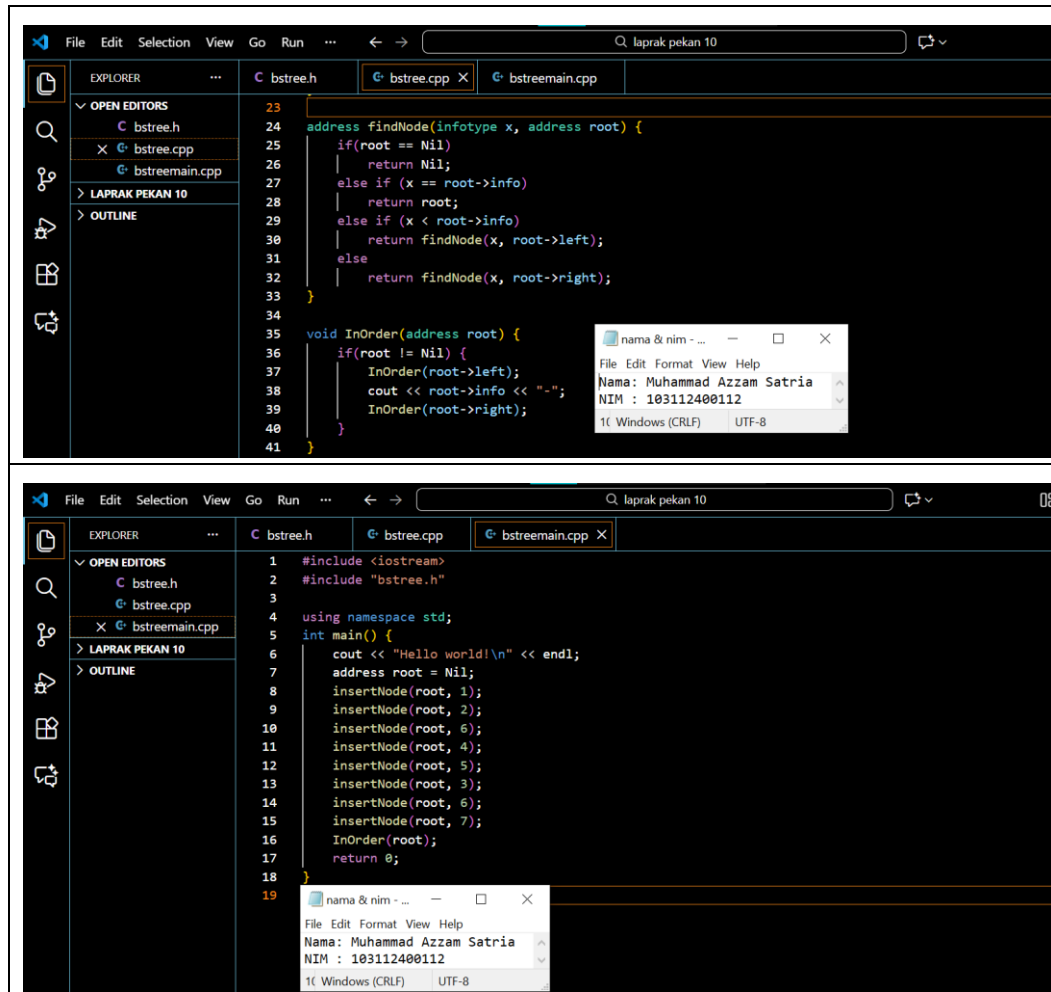
A small dialog box titled "nama & nim - ..." is overlaid on the code, containing the text: "Nama: Muhammad Azzam Satria", "NIM : 103112400112", and "1(Windows (CRLF) UTF-8".



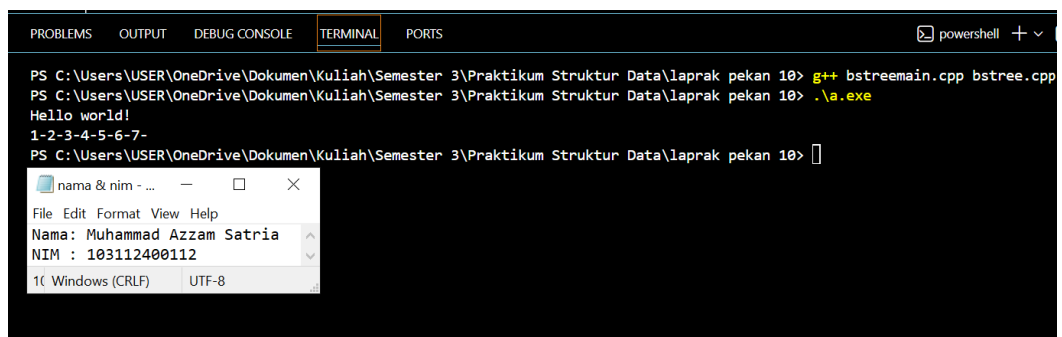
This screenshot shows the Visual Studio Code interface with the `bstree.cpp` file open. The Explorer sidebar on the left shows the project structure. The main editor displays the following C++ code:

```
1 #include "bstree.h"
2 #include <iostream>
3
4 using namespace std;
5 address alokasi(infotype x) {
6     address P = new Node;
7     P->info = x;
8     P->left = Nil;
9     P->right = Nil;
10    return P;
11 }
12
13 void insertNode(address &root, infotype x) {
14     if(root == Nil) {
15         root = alokasi(x);
16     } else {
17         if (x < root->info)
18             insertNode(root->left, x);
19         else if (x > root->info)
20             insertNode(root->right, x);
21     }
22 }
```

The same small dialog box titled "nama & nim - ..." is overlaid on the code, containing the text: "Nama: Muhammad Azzam Satria", "NIM : 103112400112", and "1(Windows (CRLF) UTF-8".



Screenshots Output



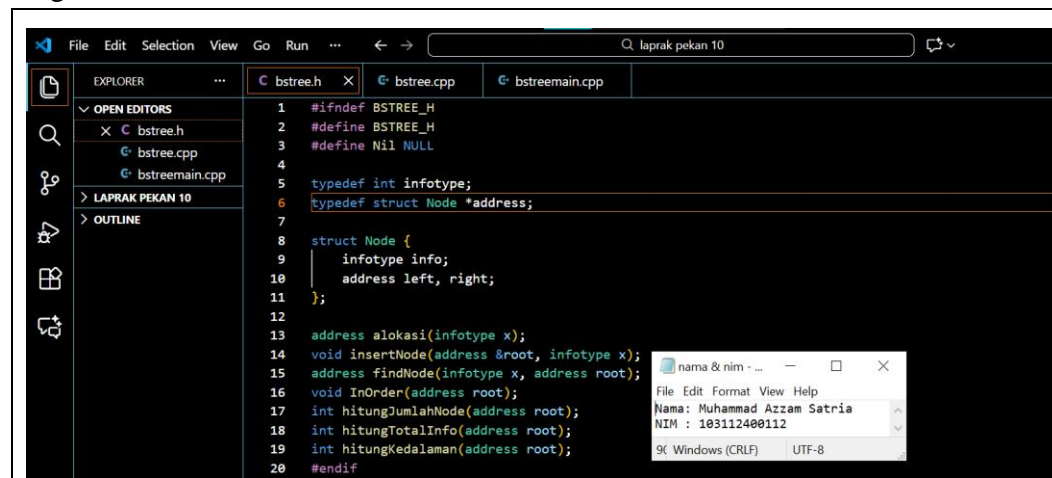
Deskripsi:

Pada soal nomor 1 di bagian header, program mendefinisikan tipe node yang menyimpan data serta pointer ke anak sebelah kiri dan kanan. Beberapa fungsi

dasar yang terdapat didalamnya yaitu fungsi alokasi node baru, insertNode, findNode dan InOrder. Mekanisme insert dilakukan dengan cara rekursif mengikuti aturan BST (Binary Search Tree), yaitu nilai yang lebih kecil akan masuk ke kiri dan nilai yang lebih besar akan masuk ke kanan.

Di bagian file implementasi (bstree.cpp) terdapat Semua fungsi yang bekerja secara rekursif. Fungsi findNode bekerja dengan cara membandingkan nilai yang sedang dicari kemudian bergerak turun ke kiri atau kanan hingga terdapat node yang kosong. Proses InOrder memiliki fungsi yaitu untuk menampilkan tree dengan urutan naik pada output. Pada bagian file bstreemain terdapat sejumlah angka yang dimasukkan sebagai inputan program. Kemudian output akan menampilkan daftar urutan angka yang tersusun dengan aturan inorder.

Unguided 2



```
1  #ifndef BSTREE_H
2  #define BSTREE_H
3  #define Nil NULL
4
5  typedef int infotype;
6  typedef struct Node *address;
7
8  struct Node {
9      infotype info;
10     address left, right;
11 };
12
13 address alokasi(infotype x);
14 void insertNode(address &root, infotype x);
15 address findNode(infotype x, address root);
16 void InOrder(address root);
17 int hitungJumlahNode(address root);
18 int hitungTotalInfo(address root);
19 int hitungKedalaman(address root);
20 #endif
```

nama & nim - ...
File Edit Format View Help
Nama: Muhammad Azzam Satria
NIM : 103112400112
9(Windows (CRLF) UTF-8

```
43 int hitungJumlahNode(address root) {
44     if(root == Nil)
45         return 0;
46     return 1 + hitungJumlahNode(root->left) + hitungJumlahNode(root->right);
47 }
48
49 int hitungTotalInfo(address root) {
50     if(root == Nil)
51         return 0;
52     return root->info + hitungTotalInfo(root->left) + hitungTotalInfo(root->right);
53 }
54
55 int hitungKedalaman(address root) {
56     if(root == Nil)
57         return 0;
58     int kiri = hitungKedalaman(root->left);
59     int kanan = hitungKedalaman(root->right);
60     return 1 + (kiri > kanan ? kiri : kanan);
61 }
```

```
1 #include <iostream>
2 #include "bstree.h"
3
4 using namespace std;
5 int main() {
6     cout << "Hello world!" << endl;
7     address root = Nil;
8     insertNode(root, 1);
9     insertNode(root, 2);
10    insertNode(root, 6);
11    insertNode(root, 4);
12    insertNode(root, 5);
13    insertNode(root, 3);
14    insertNode(root, 6);
15    insertNode(root, 7);
16    InOrder(root);
17    cout << " " << endl;
18    cout << "kedalaman : " << hitungKedalaman(root) << endl;
19    cout << "jumlah Node : " << hitungJumlahNode(root) << endl;
20    cout << "total : " << hitungTotalInfo(root) << endl;
21    return 0;
22 }
```

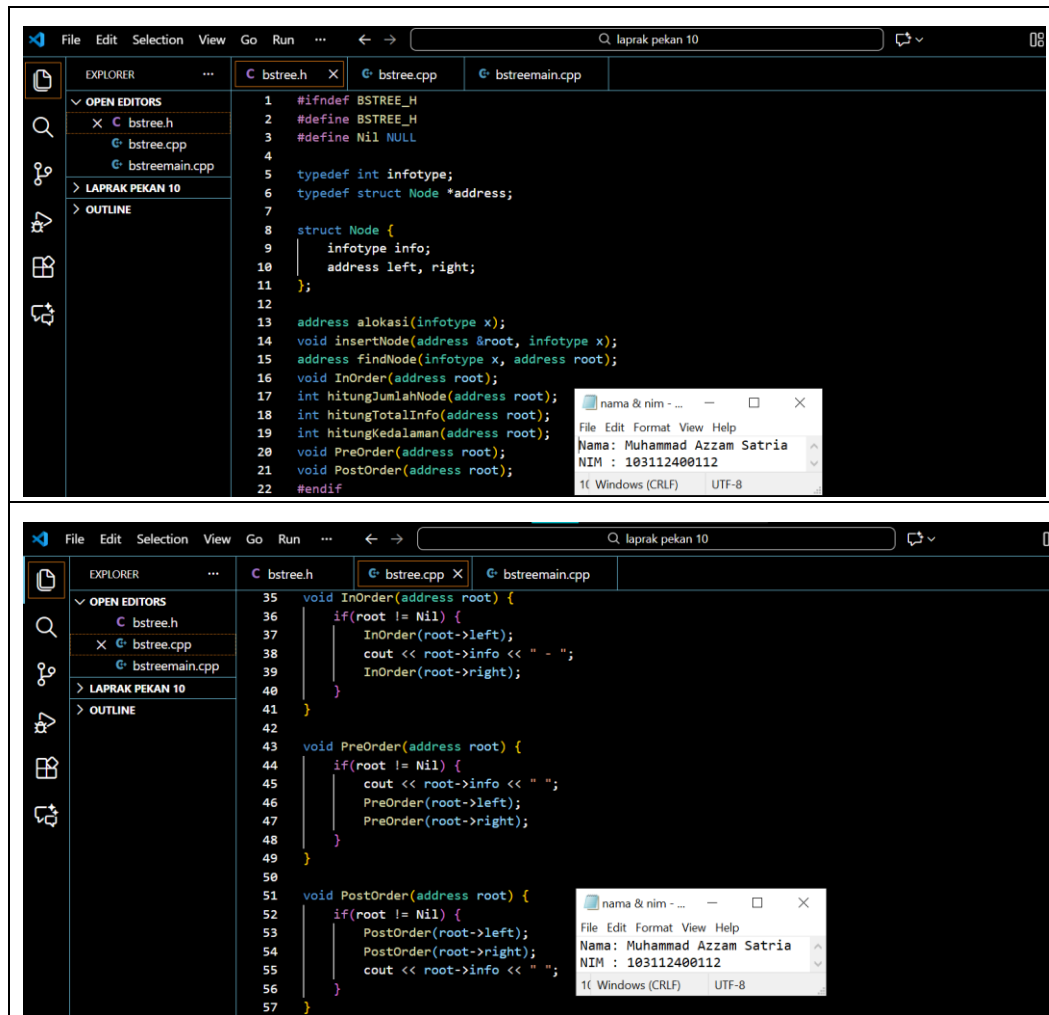
Screenshots Output

```
PS C:\Users\USER\OneDrive\Dokumen\Kuliah\Semester 3\Praktikum Struktur Data\laprak pekan 10> g++ bstreemain.cpp bstree.cpp
PS C:\Users\USER\OneDrive\Dokumen\Kuliah\Semester 3\Praktikum Struktur Data\laprak pekan 10> .\a.exe
Hello world!
1 - 2 - 3 - 4 - 5 - 6 - 7 -
kedalaman : 5
jumlah Node : 7
total : 28
PS C:\Users\USER\OneDrive\Dokumen\Kuliah\Semester 3\Praktikum Struktur Data\laprak pekan 10>
```

Deskripsi:

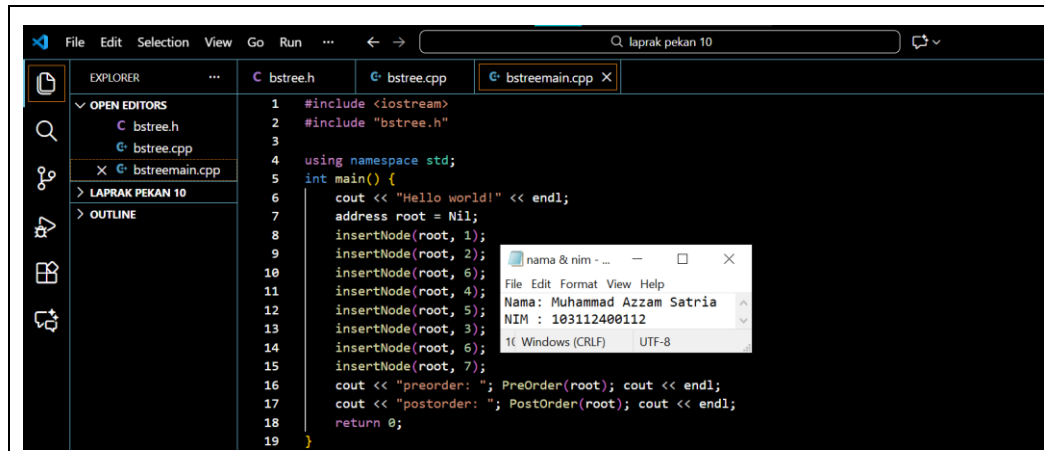
Pada soal nomor 2, program memiliki 3 fungsi tambahan yaitu fungsi `hitungJumlahNode()` untuk mengetahui berapa banyak node di dalam tree, fungsi `hitungTotalInfo()` untuk menjumlahkan semua nilai info pada node, fungsi `hitungKedalaman()` untuk mencari dalam dari sebuah tree dengan membandingkan kedalaman subtree kiri dan kanan.

Unguided 3

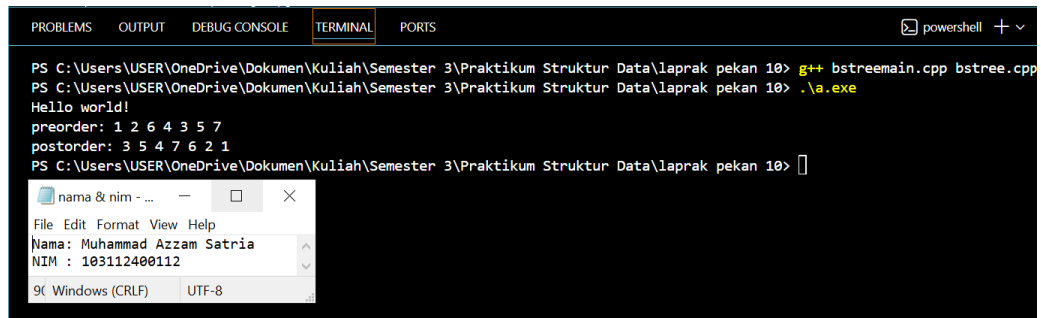


```
1  #ifndef BSTREE_H
2  #define BSTREE_H
3  #define Nil NULL
4
5  typedef int infotype;
6  typedef struct Node *address;
7
8  struct Node {
9      infotype info;
10     address left, right;
11 };
12
13 address alokasi(infotype x);
14 void insertNode(address &root, infotype x);
15 address findNode(infotype x, address root);
16 void InOrder(address root);
17 int hitungJumlahNode(address root);
18 int hitungTotalInfo(address root);
19 int hitungKedalaman(address root);
20 void PreOrder(address root);
21 void PostOrder(address root);
22 #endif
```

```
35 void InOrder(address root) {
36     if(root != Nil) {
37         InOrder(root->left);
38         cout << root->info << " - ";
39         InOrder(root->right);
40     }
41 }
42
43 void PreOrder(address root) {
44     if(root != Nil) {
45         cout << root->info << " - ";
46         PreOrder(root->left);
47         PreOrder(root->right);
48     }
49 }
50
51 void PostOrder(address root) {
52     if(root != Nil) {
53         PostOrder(root->left);
54         PostOrder(root->right);
55         cout << root->info << " - ";
56     }
57 }
```



Screenshots Output



Deskripsi:

Pada soal nomor 3, program memiliki 2 fungsi tambahan traversal yaitu PreOrder dan PostOrder. PreOrder fungsinya untuk menampilkan data dengan urutan root → anak sebelah kiri → anak sebelah kanan, sedangkan PostOrder fungsinya untuk menampilkan data dengan urutan anak sebelah kiri → anak sebelah kanan → root.

D. Kesimpulan

Struktur data tree merupakan salah satu bentuk cara penyimpanan data yang cara penyusunannya memiliki aturan bahwa nilai yang lebih kecil ditempatkan di bagian anak sebelah kiri, sedangkan nilai yang lebih besar ditempatkan di bagian anak sebelah kanan. Melalui contoh dan latihan pada modul saya belajar secara langsung bagaimana membangun struktur data tree dengan konsep ADT (Abstract Data Type). Pada soal latihan, saya mempelajari cara Membuat node baru, menambah data ke dalam tree, mencari suatu nilai dalam tree dan

menampilkan isi tree dengan menggunakan traversal InOrder.

Pada soal latihan selanjutnya, terdapat operasi tambahan untuk menganalisis isi tree seperti menghitung jumlah node, total nilai dan menentukan kedalaman suatu tree. Lalu di soal latihan terakhir terdapat penambahan fungsi PreOrder dan PostOrder agar output tree bisa menampilkan pola urutan yang berbeda. Dengan adanya laporan praktikum modul 10 ini, memberikan gambaran kepada saya mengenai pengelolaan dan analisis pada suatu struktur data tree.

E. Referensi

Ginting, S. H. N., Effendi, H., Kumar, S., Marsisno, W., Sitanggang, Y. R. U., Anwar, K., ... & Smrti, N. N. E. (2024). *Pengantar struktur data*. Penerbit Mifandi Mandiri Digital.

Febriansyah, M. F., Rhamadani, M., & Sutabri, T. (2025). Perbandingan pemanfaatan algoritma rekursif dan iteratif dalam penyelesaian struktur data pohon. *Jurnal Manajemen Informatika & Teknologi*, 5(1), 46–56.