

LAPORAN PRAKTIKUM
Tree
Bagian Pertama



Disusun Oleh:

Ryan Gabriel Togar Simamora (2311104045)

Kelas : SE0702

Dosen :

Wahyu Andi Saputra

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2024

I. Tujuan

1. Memahami konsep penggunaan fungsi rekursif.
2. Mengimplementasikan bentuk-bentuk fungsi rekursif.
3. Mengaplikasikan struktur data tree dalam sebuah kasus pemrograman.
4. Mengimplementasikan struktur data tree, khususnya Binary Tree.

II. Landasan Teori

A. Pengertian Tree

Tree adalah struktur data non-linear yang direpresentasikan sebagai grafik tak berarah yang terhubung tanpa mengandung sirkuit. Tree terdiri dari simpul (nodes) yang saling terhubung dengan karakteristik :

- ❖ Tree kosong disebut sebagai empty tree.
- ❖ Terdapat satu simpul tanpa pendahulu, disebut akar (root).
- ❖ Semua simpul lainnya memiliki satu pendahulu.

Terminologi dalam Tree

- ❖ Root (Akar): Simpul utama dari tree.
- ❖ Parent dan Child: Simpul B, C, dan D adalah anak dari simpul A; A adalah orang tua mereka.
- ❖ Path (Lintasan): Jalur dari satu simpul ke simpul lain, misalnya dari A ke J.
- ❖ Sibling (Saudara kandung): Anak-anak yang memiliki orang tua yang sama.
- ❖ Degree (Derajat): Jumlah anak yang dimiliki oleh suatu simpul.
- ❖ Leaf (Daun): Simpul tanpa anak (derajat 0).
- ❖ Internal Nodes: Simpul yang memiliki anak.
- ❖ Height (Tinggi): Jumlah maksimum simpul dalam cabang tree.

Jenis-Jenis Tree

- ❖ Ordered Tree: Pohon di mana urutan anak-anak penting.
- ❖ Binary Tree: Setiap simpul hanya memiliki maksimum dua anak (left dan right).
- ❖ Complete Binary Tree: Semua level terisi penuh kecuali level terakhir.
- ❖ Extended Binary Tree: Semua simpul memiliki tepat 0 atau 2 anak.
- ❖ Binary Search Tree (BST): Struktur di mana nilai anak kiri lebih kecil dan anak kanan lebih besar dari simpul induk.
- ❖ AVL Tree: BST dengan perbedaan tinggi maksimum antara subtree kiri dan kanan adalah 1.
- ❖ Heap Tree: Tree dengan aturan:
 - Minimum Heap: Parent lebih kecil dari anak-anaknya.
 - Maximum Heap: Parent lebih besar dari anak-anaknya.

Operasi Dasar dalam Binary Search Tree (BST)

- ❖ Insert: Menambahkan elemen ke tree.
- ❖ Nilai lebih kecil dari root dimasukkan ke subtree kiri.
- ❖ Nilai lebih besar dari root dimasukkan ke subtree kanan.
- ❖ Update: Melakukan regenerasi jika posisi simpul melanggar aturan BST.
- ❖ Search: Mencari nilai dengan algoritma binary search secara rekursif.

Traversal pada Binary Tree

- ❖ Pre-order: Mengunjungi root, lalu subtree kiri, dan subtree kanan.
- ❖ In-order: Mengunjungi subtree kiri, root, lalu subtree kanan.
- ❖ Post-order: Mengunjungi subtree kiri, subtree kanan, lalu root.

Implementasi

Tree dapat diimplementasikan menggunakan representasi linier (array) atau linked list. Operasi seperti pembuatan node, traversal, pencarian, dan penghitungan informasi total node dilakukan menggunakan algoritma rekursif.

Contoh Implementasi

Insert Node: Fungsi rekursif untuk menambahkan elemen pada BST.

Traversal: Menampilkan elemen tree dengan metode pre-order, in-order, atau post-order.

Hitung Kedalaman: Menggunakan fungsi rekursif untuk mencari tinggi maksimum tree.

Contoh Dalam Kehidupan Nyata

Struktur Organisasi

Deskripsi: Sebuah perusahaan memiliki CEO sebagai root. Di bawahnya ada kepala divisi (anak), dan di bawah kepala divisi ada manajer hingga karyawan.

Implementasi:

Root: CEO.

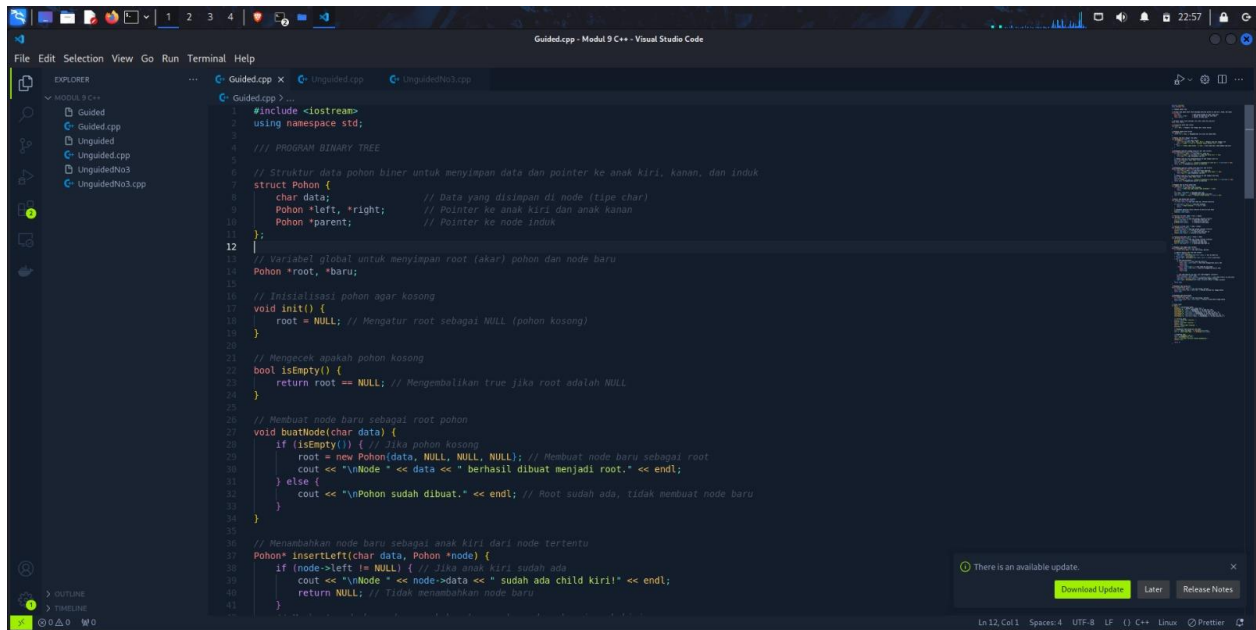
Children: Kepala Divisi.

Leaves (daun): Karyawan tanpa bawahan.

Struktur data tree digunakan dalam banyak kasus pemrograman seperti pencarian data, pengelolaan hierarki, dan representasi grafis. Pemahaman tentang karakteristik, jenis, dan operasinya sangat penting dalam implementasi algoritma berbasis tree.

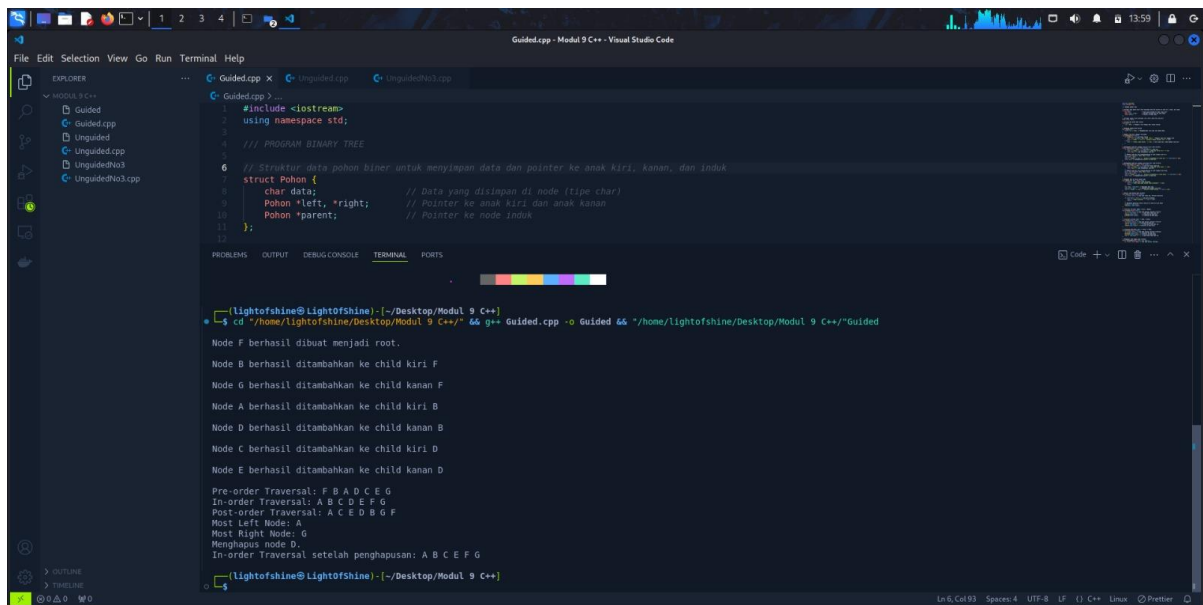
III. Guided

File Guided1.cpp



```
1 #include <iostream>
2 using namespace std;
3
4 // PROGRAM BINARY TREE
5
6 // Struktur data pohon biner untuk menyimpan data dan pointer ke anak kiri, kanan, dan induk
7 struct Pohon {
8     char data;           // Data yang disimpan di node (tipe char)
9     Pohon *left;         // Pointer ke anak kiri dan anak kanan
10    Pohon *parent;        // Pointer ke node induk
11 };
12
13 // Variabel global untuk menyimpan root (akar) pohon dan node baru
14 Pohon *root, *baru;
15
16 // Inisialisasi pohon agar kosong
17 void init() {
18     root = NULL; // Mengatur root sebagai NULL (pohon kosong)
19 }
20
21 // Mengecek apakah pohon kosong
22 bool isEmpty() {
23     return root == NULL; // Mengembalikan true jika root adalah NULL
24 }
25
26 // Membuat node baru sebagai root pohon
27 void buatNode(char data) {
28     if (isEmpty()) { // Jika pohon kosong
29         root = new Pohon(data, NULL, NULL); // Membuat node baru sebagai root
30         cout << "Node " << data << " berhasil dibuat menjadi root." << endl;
31     } else {
32         cout << "Node " << data << " sudah ada, tidak membuat node baru." << endl;
33     }
34 }
35
36 // Menambahkan node baru sebagai anak kiri dari node tertentu
37 Pohon* insertLeft(char data, Pohon *node) {
38     if (node->left == NULL) { // Jika anak kiri sudah ada
39         cout << "Node " << data << " sudah ada child kiri!" << endl;
40         return NULL; // Tidak menambahkan node baru
41     }
```

Outputnya



```
(Lightofshine@Lightofshine):[~/Desktop/Modul 9 C++]
$ cd "/home/lightofshine/Desktop/Modul 9 C++/" && g++ Guided.cpp -o Guided && ./Guided
Node F berhasil dibuat menjadi root.
Node B berhasil ditambahkan ke child kiri F
Node G berhasil ditambahkan ke child kanan F
Node A berhasil ditambahkan ke child kiri B
Node D berhasil ditambahkan ke child kanan B
Node C berhasil ditambahkan ke child kiri D
Node E berhasil ditambahkan ke child kanan D
Pre-order Traversal: F B A D C E G
In-order Traversal: A B C D E F G
Post-order Traversal: A C E D B G F
Most Left Node: A
Most Right Node: G
Menghapus node D.
In-order Traversal setelah penghapusan: A B C E F G
(Lightofshine@Lightofshine):[~/Desktop/Modul 9 C++]
```

Untuk Source codenya lebih lengkapnya sebagai berikut :

```

#include <iostream>
using namespace std;

// PROGRAM BINARY TREE

// Struktur data pohon biner untuk menyimpan data dan pointer ke anak kiri, kanan, dan induk
struct Pohon {
    char data; // Data yang disimpan di node (type char)
    Pohon *left; // Pointer ke anak kiri dan anak kanan
    Pohon *parent; // Pointer ke node induk
};

// Variabel global untuk menyimpan root (akar) pohon dan node baru
Pohon *root, *baru;

// Inisialisasi pohon agar kosong
void init() {
    root = NULL; // Mengatur root sebagai NULL (pohon kosong)
}

// Mengecek apakah pohon kosong
bool isEmpty() {
    return root == NULL; // Mengembalikan true jika root adalah NULL
}

// Membuat node baru sebagai root pohon
void buatNode(char data) {
    if (isEmpty()) { // Jika pohon kosong
        root = new Pohon(data, NULL, NULL, NULL); // Membuat node baru sebagai root
        cout << "Node " << data << " berhasil dibuat menjadi root." << endl;
    } else {
        cout << "Pohon sudah dibuat." << endl; // Root sudah ada, tidak membuat node baru
    }
}

// Menambahkan node baru sebagai anak kiri dari node tertentu
Pohon* insertLeft(char data, Pohon *node) {
    if (node->left != NULL) { // Jika anak kiri sudah ada
        cout << "Node " << node->data << " sudah ada child kiri!" << endl;
        return NULL; // Tidak menambahkan node baru
    }
    // Membuat node baru dan menghubungkannya ke node sebagai anak kiri
    baru = new Pohon(data, NULL, NULL, node);
    node->left = baru;
    cout << "Node " << data << " berhasil ditambahkan ke child kiri " << node->data << endl;
    return baru; // Mengembalikan pointer ke node baru
}

// Menambahkan node baru sebagai anak kanan dari node tertentu
Pohon* insertRight(char data, Pohon *node) {
    if (node->right != NULL) { // Jika anak kanan sudah ada
        cout << "Node " << node->data << " sudah ada child kanan!" << endl;
        return NULL; // Tidak menambahkan node baru
    }
    // Membuat node baru dan menghubungkannya ke node sebagai anak kanan
    baru = new Pohon(data, NULL, NULL, node);
    node->right = baru;
    cout << "Node " << data << " berhasil ditambahkan ke child kanan " << node->data << endl;
    return baru; // Mengembalikan pointer ke node baru
}

// Mengubah data di dalam sebuah node
void update(char data, Pohon *node) {
    if (!node) { // Jika node tidak ditemukan
        cout << "Node yang ingin diubah tidak ditemukan!" << endl;
        return;
    }
    char temp = node->data; // Menyimpan data lama
    node->data = data; // Mengubah data dengan nilai baru
    cout << "Node " << temp << " berhasil diubah menjadi " << data << endl;
}

// Mencari node dengan data tertentu
void find(char data, Pohon *node) {
    if (!node) return; // Jika node tidak ada, hentikan pencarian
    if (node->data == data) { // Jika data ditemukan
        cout << "Node ditemukan: " << data << endl;
        return;
    }
    // Melakukan pencarian secara rekursif ke anak kiri dan kanan
    find(data, node->left);
    find(data, node->right);
}

// Traversal Pre-order (Node -> Kiri -> Kanan)
void preOrder(Pohon *node) {
    if (!node) return; // Jika node kosong, hentikan traversal
    cout << node->data << " "; // Cetak data node saat ini
    preOrder(node->left); // Traversal ke anak kiri
    preOrder(node->right); // Traversal ke anak kanan
}

// Traversal In-order (Kiri -> Node -> Kanan)
void inOrder(Pohon *node) {
    if (!node) return; // Jika node kosong, hentikan traversal
    inOrder(node->left); // Traversal ke anak kiri
    cout << node->data << " "; // Cetak data node saat ini
    inOrder(node->right); // Traversal ke anak kanan
}

// Traversal Post-order (Kiri -> Kanan -> Node)
void postOrder(Pohon *node) {
    if (!node) return; // Jika node kosong, hentikan traversal
    postOrder(node->left); // Traversal ke anak kiri
    postOrder(node->right); // Traversal ke anak kanan
    cout << node->data << " "; // Cetak data node saat ini
}

// Menghapus node dengan data tertentu
Pohon* deleteNode(Pohon *node, char data) {
    if (!node) return NULL; // Jika node kosong, hentikan
    // Rekursif mencari node yang akan dihapus
    if (data == node->data) {
        node->left = deleteNode(node->left, data); // Cari di anak kiri
    } else if (data < node->data) {
        node->right = deleteNode(node->right, data); // Cari di anak kanan
    } else {
        // Jika node ditemukan
        if (!node->left) { // Jika tidak ada anak kiri
            Pohon *temp = node->right; // Anak kanan menggantikan posisi node
            delete node;
            return temp;
        } else if (!node->right) { // Jika tidak ada anak kanan
            Pohon *temp = node->left; // Anak kiri menggantikan posisi node
            delete node;
            return temp;
        }
        // Jika node memiliki dua anak, cari node pengganti (successor)
        Pohon *successor = node->right;
        while (successor->left) successor = successor->left; // Cari node terkecil di anak kanan
        node->data = successor->data; // Gantikan data dengan successor
        node->right = deleteNode(node->right, successor->data); // Hapus successor
    }
    return node;
}

// Menemukan node paling kiri
Pohon* mostLeft(Pohon *node) {
    if (!node) return NULL; // Jika node kosong, hentikan
    while (node->left) node = node->left; // Iterasi ke anak kiri hingga mentak
    return node;
}

// Menemukan node paling kanan
Pohon* mostRight(Pohon *node) {
    if (!node) return NULL; // Jika node kosong, hentikan
    while (node->right) node = node->right; // Iterasi ke anak kanan hingga mentak
    return node;
}

// Fungsi utama
int main() {
    init(); // Inisialisasi pohon
    buatNode('A'); // Membuat root dengan data 'A'
    insertLeft('B', root); // Menambahkan 'B' ke anak kiri root
    insertRight('C', root); // Menambahkan 'C' ke anak kanan root
    insertLeft('A', root->left); // Menambahkan 'A' ke anak kiri dari 'B'
    insertRight('D', root->left); // Menambahkan 'D' ke anak kanan dari 'B'
    insertLeft('C', root->left->right); // Menambahkan 'C' ke anak kiri dari 'D'
    insertRight('E', root->left->right); // Menambahkan 'E' ke anak kanan dari 'D'

    // Traversal pohon
    cout << "Pre-order Traversal: ";
    preOrder(root);
    cout << "In-order Traversal: ";
    inOrder(root);
    cout << "Post-order Traversal: ";
    postOrder(root);

    // Menampilkan node paling kiri dan kanan
    cout << "Most Left Node: " << mostLeft(root)->data;
    cout << "Most Right Node: " << mostRight(root)->data;

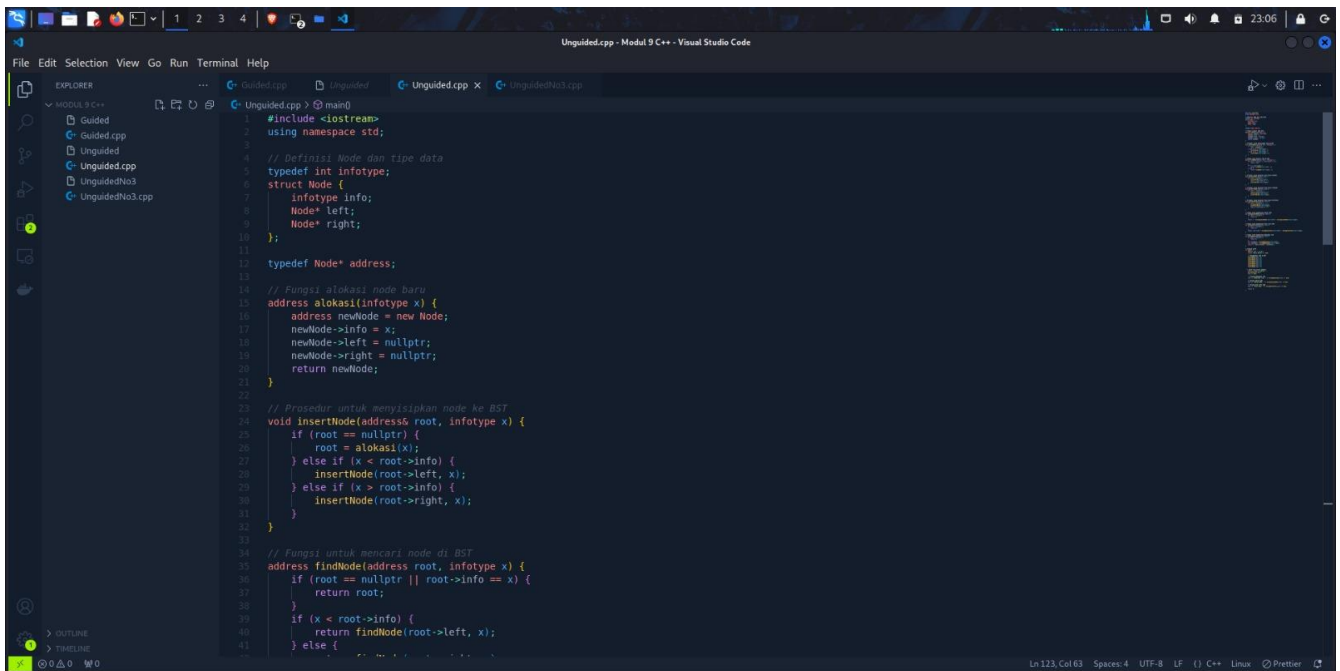
    // Menghapus node
    cout << "Menghapus node D:";
    root = deleteNode(root, 'D');
    cout << "In-order Traversal setelah penghapusan: ";
    inOrder(root);

    return 0;
}

```

IV. Unguided

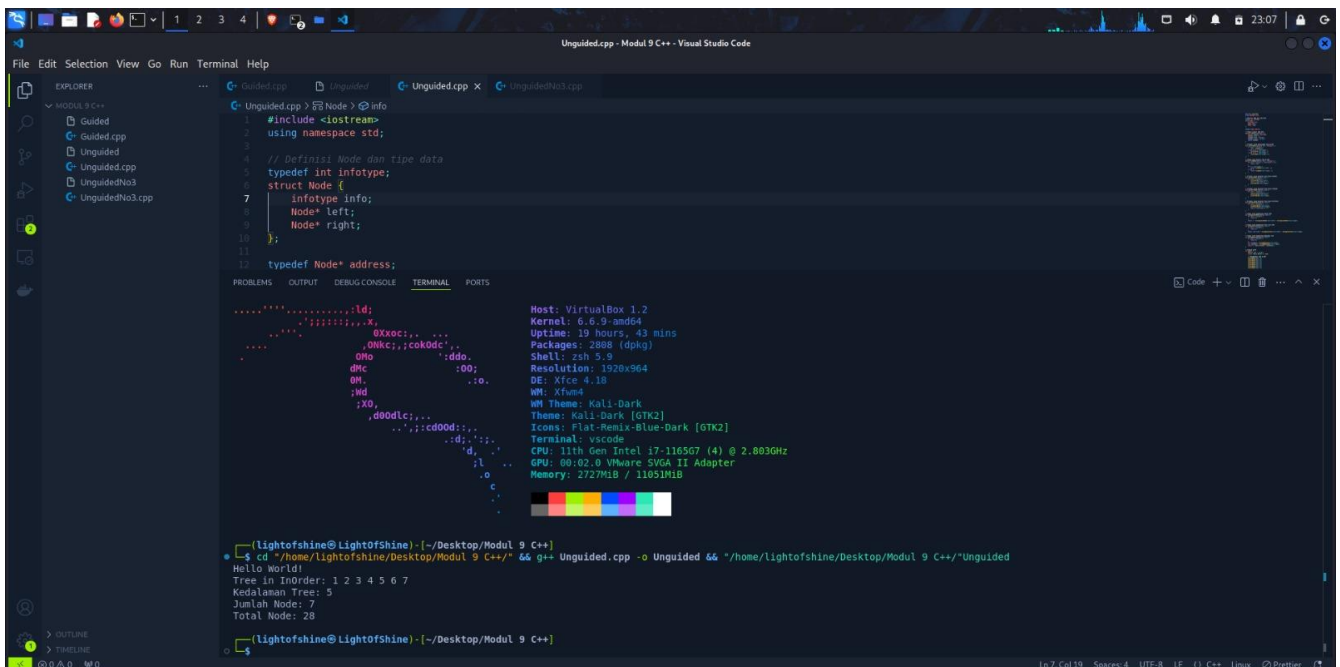
File Unguided No 1 dan 2



The screenshot shows the Visual Studio Code editor with the file 'Unguided.cpp' open. The code is a C++ program for a Binary Search Tree (BST). It includes a main function that calls 'insertNode' and 'findNode'. The 'insertNode' function recursively inserts a new node into the tree based on its value relative to the current node. The 'findNode' function recursively searches for a node with a specific value. The code is well-commented in Indonesian.

```
1 #include <iostream>
2 using namespace std;
3
4 // Definisi Node dan tipe data
5 typedef int infotype;
6 struct Node {
7     infotype info;
8     Node* left;
9     Node* right;
10 };
11
12 typedef Node* address;
13
14 // Fungsi alokasi node baru
15 address alokasi(infotype x) {
16     address newNode = new Node;
17     newNode->info = x;
18     newNode->left = nullptr;
19     newNode->right = nullptr;
20     return newNode;
21 }
22
23 // Prosedur untuk menyisipkan node ke BST
24 void insertNode(address& root, infotype x) {
25     if (root == nullptr) {
26         root = alokasi(x);
27     } else if (x < root->info) {
28         insertNode(root->left, x);
29     } else if (x > root->info) {
30         insertNode(root->right, x);
31     }
32 }
33
34 // Fungsi untuk mencari node di BST
35 address findNode(address root, infotype x) {
36     if (root == nullptr || root->info == x) {
37         return root;
38     }
39     if (x < root->info) {
40         return findNode(root->left, x);
41     } else {
42         return findNode(root->right, x);
43     }
44 }
```

Outputnya



The screenshot shows the Visual Studio Code editor with the 'Unguided.cpp' file open. The 'TERMINAL' tab is active, displaying the output of the program. The output shows the tree structure in InOrder, the number of nodes, and the total number of nodes. The program also prints 'Hello World!' and a color bar.

```
Host: VirtualBox 1.2
Kernel: 6.6.9-amd64
Uptime: 19 hours, 43 mins
Packages: 2888 (dpkg)
Shell: zsh 5.9
Resolution: 1920x964
DE: Xfce 4.18
WM: Xfce4
WM Theme: Kali-Dark
Theme: Kali-Dark [GTK2]
Icons: Flat-Remix-Blue-Dark [GTK2]
Terminal: vscode
CPU: 11th Gen Intel i7-1165G7 (4) @ 2.803GHz
GPU: 00:02.0 VMware SVGA II Adapter
Memory: 2727MiB / 11051MiB

(Lightofshine@LightOfShine) ~/Desktop/Modul 9 C++
$ cd ~/home/Lightofshine/Desktop/Modul 9 C++/ && g++ Unguided.cpp -o Unguided && ./Unguided
Hello World!
Tree in InOrder: 1 2 3 4 5 6 7
Kedalaman Trees: 5
Jumlah Node: 7
Total Node: 28

(Lightofshine@LightOfShine) ~/Desktop/Modul 9 C++
$
```

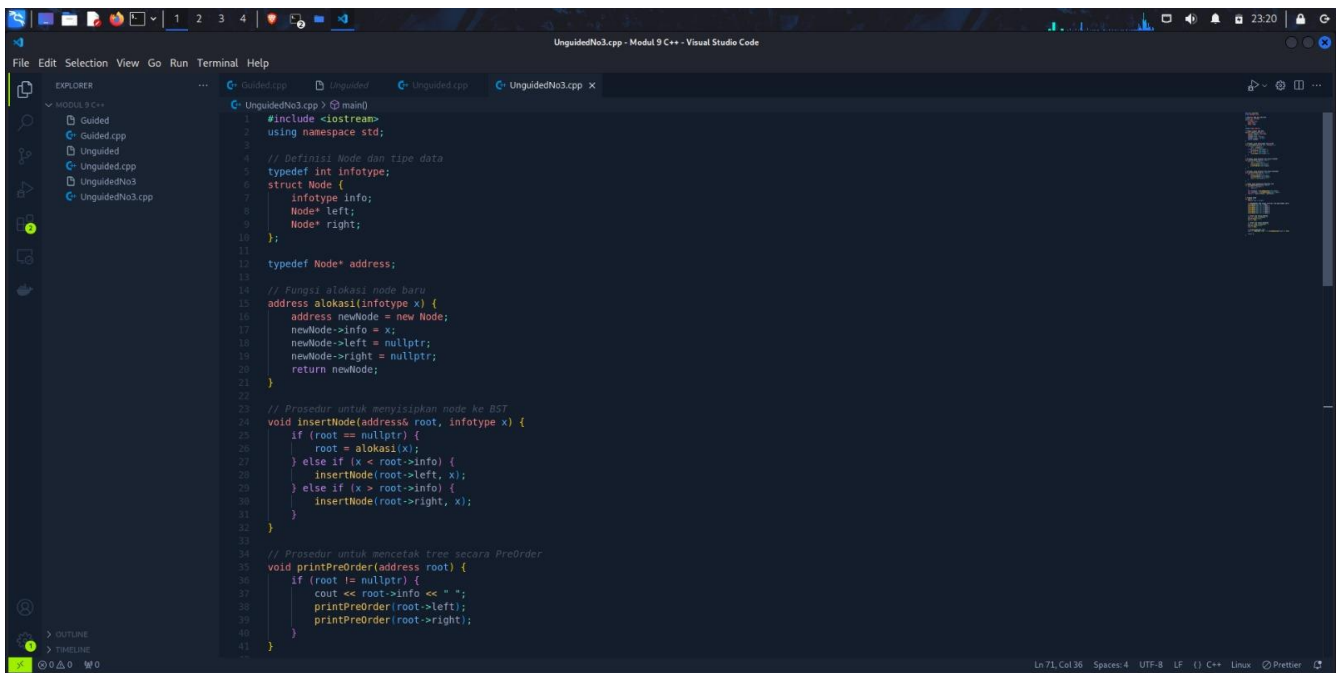
Untuk Source codenya lebih lengkapnya sebagai berikut :

```

1 #include <iostream>
2 using namespace std;
3
4 // Definisi Node dan tipe data
5 typedef int infotype;
6 struct Node {
7     infotype info;
8     Node* left;
9     Node* right;
10 };
11
12 typedef Node* address;
13
14 // Fungsi alokasi node baru
15 address alokasi(infotype x) {
16     address newNode = new Node;
17     newNode->info = x;
18     newNode->left = nullptr;
19     newNode->right = nullptr;
20     return newNode;
21 }
22
23 // Prosedur untuk menyisipkan node ke BST
24 void insertNode(address& root, infotype x) {
25     if (root == nullptr) {
26         root = alokasi(x);
27     } else if (x < root->info) {
28         insertNode(root->left, x);
29     } else if (x > root->info) {
30         insertNode(root->right, x);
31     }
32 }
33
34 // Fungsi untuk mencari node di BST
35 address findNode(address root, infotype x) {
36     if (root == nullptr || root->info == x) {
37         return root;
38     }
39     if (x < root->info) {
40         return findNode(root->left, x);
41     } else {
42         return findNode(root->right, x);
43     }
44 }
45
46 // Prosedur untuk mencetak tree secara InOrder
47 void printInOrder(address root) {
48     if (root != nullptr) {
49         printInOrder(root->left);
50         cout << root->info << " ";
51         printInOrder(root->right);
52     }
53 }
54
55 // Prosedur untuk mencetak tree secara PreOrder
56 void printPreOrder(address root) {
57     if (root != nullptr) {
58         cout << root->info << " ";
59         printPreOrder(root->left);
60         printPreOrder(root->right);
61     }
62 }
63
64 // Prosedur untuk mencetak tree secara PostOrder
65 void printPostOrder(address root) {
66     if (root != nullptr) {
67         printPostOrder(root->left);
68         printPostOrder(root->right);
69         cout << root->info << " ";
70     }
71 }
72
73 // Fungsi untuk menghitung jumlah node
74 int hitungJumlahNode(address root) {
75     if (root == nullptr) {
76         return 0;
77     }
78     return 1 + hitungJumlahNode(root->left) + hitungJumlahNode(root->right);
79 }
80
81 // Fungsi untuk menghitung total info node
82 int hitungTotalInfo(address root) {
83     if (root == nullptr) {
84         return 0;
85     }
86     return root->info + hitungTotalInfo(root->left) + hitungTotalInfo(root->right);
87 }
88
89 // Fungsi untuk menghitung kedalaman tree
90 int hitungKedalaman(address root) {
91     if (root == nullptr) {
92         return 0;
93     }
94     int leftDepth = hitungKedalaman(root->left);
95     int rightDepth = hitungKedalaman(root->right);
96     return 1 + max(leftDepth, rightDepth);
97 }
98
99 // Program utama
100 int main() {
101     address root = nullptr;
102     cout << "Hello World!" << endl;
103
104     // Menambahkan node ke BST
105     insertNode(root, 1);
106     insertNode(root, 2);
107     insertNode(root, 6);
108     insertNode(root, 4);
109     insertNode(root, 5);
110     insertNode(root, 3);
111     insertNode(root, 6);
112     insertNode(root, 7);
113
114     // Cetak tree secara InOrder
115     cout << "Tree in InOrder: ";
116     printInOrder(root);
117     cout << endl;
118
119     // Hitung kedalaman tree
120     cout << "Kedalaman Tree: " << hitungKedalaman(root) << endl;
121
122     // Hitung jumlah node
123     cout << "Jumlah Node: " << hitungJumlahNode(root) << endl;
124
125     // Hitung total info node
126     cout << "Total Node: " << hitungTotalInfo(root) << endl;
127
128     return 0;
129 }

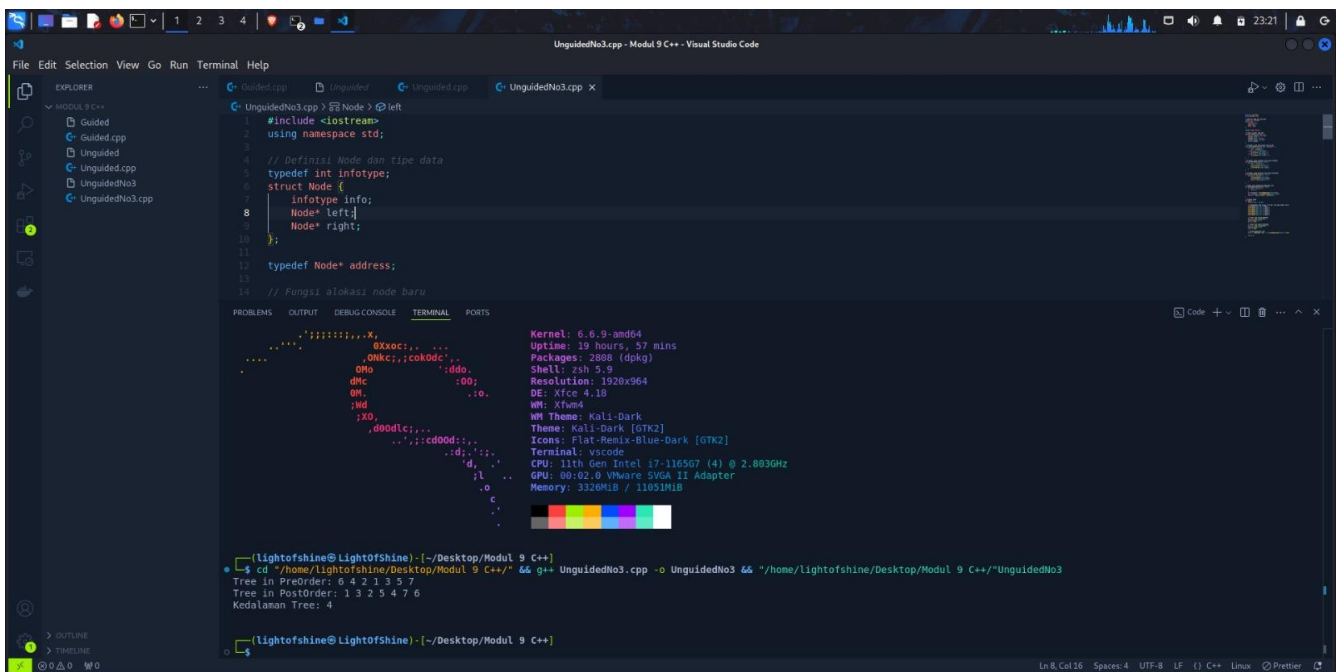
```

File Unguided No.3



```
1 #include <iostream>
2 using namespace std;
3
4 // Definisi Node dan tipe data
5 typedef int infotype;
6 struct Node {
7     infotype info;
8     Node* left;
9     Node* right;
10 };
11
12 typedef Node* address;
13
14 // Fungsi alokasi node baru
15 address alokasi(infotype x) {
16     address newNode = new Node;
17     newNode->info = x;
18     newNode->left = nullptr;
19     newNode->right = nullptr;
20     return newNode;
21 }
22
23 // Prosedur untuk menyisipkan node ke BST
24 void insertNode(address& root, infotype x) {
25     if (root == nullptr) {
26         root = alokasi(x);
27     } else if (x < root->info) {
28         insertNode(root->left, x);
29     } else if (x > root->info) {
30         insertNode(root->right, x);
31     }
32 }
33
34 // Prosedur untuk mencetak tree secara PreOrder
35 void printPreOrder(address root) {
36     if (root != nullptr) {
37         cout << root->info << " ";
38         printPreOrder(root->left);
39         printPreOrder(root->right);
40     }
41 }
```

Outputnya



```
Kernel: 6.6.9-amd64
Uptime: 19 hours, 57 mins
Packages: 2888 (dpkg)
Shell: zsh 5.9
Resolution: 1920x964
DE: Xfce 4.16
WM: Xfce4
WM Theme: Kali-Dark
Theme: Kali-Dark [GTK2]
Icons: Flat-Remix-Blue-Dark [GTK2]
Terminal: vscode
CPU: 11th Gen Intel i7-116507 (4) @ 2.803GHz
GPU: 00:02.0 VMware SVGA II Adapter
Memory: 332MiB / 1105MiB

[Lightofshine@Lightofshine] ~/Desktop/Modul 9 C++
$ cd ~/home/lightofshine/Desktop/Modul 9 C++/ && g++ UnguidedNo3.cpp -o UnguidedNo3 && ./UnguidedNo3
Tree in PreOrder: 6 4 2 1 3 5 7
Tree in PostOrder: 1 3 2 5 4 7 6
Kedalaman Tree: 4

[Lightofshine@Lightofshine] ~/Desktop/Modul 9 C++
```

Untuk Source codenya lebih lengkapnya sebagai berikut :


```

1  #include <iostream>
2  using namespace std;
3
4  // Definisi Node dan tipe data
5  typedef int infotype;
6  struct Node {
7      infotype info;
8      Node* left;
9      Node* right;
10 };
11
12 typedef Node* address;
13
14 // Fungsi alokasi node baru
15 address alokasi(infotype x) {
16     address newNode = new Node;
17     newNode->info = x;
18     newNode->left = nullptr;
19     newNode->right = nullptr;
20     return newNode;
21 }
22
23 // Prosedur untuk menyisipkan node ke BST
24 void insertNode(address& root, infotype x) {
25     if (root == nullptr) {
26         root = alokasi(x);
27     } else if (x < root->info) {
28         insertNode(root->left, x);
29     } else if (x > root->info) {
30         insertNode(root->right, x);
31     }
32 }
33
34 // Prosedur untuk mencetak tree secara PreOrder
35 void printPreOrder(address root) {
36     if (root != nullptr) {
37         cout << root->info << " ";
38         printPreOrder(root->left);
39         printPreOrder(root->right);
40     }
41 }
42
43 // Prosedur untuk mencetak tree secara PostOrder
44 void printPostOrder(address root) {
45     if (root != nullptr) {
46         printPostOrder(root->left);
47         printPostOrder(root->right);
48         cout << root->info << " ";
49     }
50 }
51
52 // Fungsi untuk menghitung kedalaman tree
53 int hitungKedalaman(address root) {
54     if (root == nullptr) {
55         return 0;
56     }
57     int leftDepth = hitungKedalaman(root->left);
58     int rightDepth = hitungKedalaman(root->right);
59     return 1 + max(leftDepth, rightDepth);
60 }
61
62 // Program utama
63 int main() {
64     address root = nullptr;
65
66     // Menambahkan node sesuai ilustrasi tree pada Gambar 10-14
67     insertNode(root, 6); // Root
68     insertNode(root, 4); // Level 2
69     insertNode(root, 7); // Level 2
70     insertNode(root, 2); // Level 3
71     insertNode(root, 5); // Level 3
72     insertNode(root, 1); // Level 4
73     insertNode(root, 3); // Level 4
74
75     // Cetak tree secara PreOrder
76     cout << "Tree in PreOrder: ";
77     printPreOrder(root);
78     cout << endl;
79
80     // Cetak tree secara PostOrder
81     cout << "Tree in PostOrder: ";
82     printPostOrder(root);
83     cout << endl;
84
85     // Hitung kedalaman tree
86     cout << "Kedalaman Tree: " << hitungKedalaman(root) << endl;
87
88     return 0;
89 }

```

V. Kesimpulan

Tree adalah struktur data non-linear yang merepresentasikan hubungan hierarkis antara simpul-simpul (nodes). Tree memiliki berbagai jenis seperti Binary Tree, Binary Search Tree (BST), dan AVL Tree, yang masing-masing memiliki aturan khusus. Operasi dasar seperti insert, search, dan traversal dilakukan untuk mengelola data secara efisien. Tree digunakan dalam berbagai aplikasi, seperti struktur organisasi dan pengelolaan data hierarkis, menjadikannya elemen penting dalam pengembangan algoritma dan pemrograman.

