LAPORAN PRAKTIKUM

MODUL 10,11

“TREE”

Disusun Oleh:

Tiurma Grace Angelina (2311104042)

SE-07-02

Dosen :

Wahyu Andi Saputra, S.Pd., M.Eng

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

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

2. **Landasan Teori**

    Rekursif adalah teknik pemrograman di mana suatu fungsi memanggil dirinya sendiri untuk menyelesaikan suatu masalah selama kondisi tertentu terpenuhi, dengan setiap pemanggilan mendekatkan fungsi pada kondisi berhenti (base case). Teknik ini memudahkan penulisan solusi untuk masalah yang memiliki pola penyelesaian berulang, seperti perhitungan matematis atau struktur data seperti pohon biner. Rekursif meningkatkan keterbacaan program dan menyederhanakan logika, meskipun sering kali memerlukan memori dan waktu eksekusi tambahan untuk menangani aktivitas pemanggilan. Oleh karena itu, rekursif biasanya digunakan saat pendekatan iteratif sulit diterapkan atau saat efisiensi bukan prioritas utama dibandingkan dengan kejelasan program.

3. **Guided**
    - **Array**

    1. CODE :

```
2.  #include
<iostream>
3.  using namespace
std;
4.
5.  /// PROGRAM
BINARY TREE
6.
7.  // Struktur data
pohon biner untuk
menyimpan data dan
pointer ke anak kiri,
kanan, dan induk
8.  struct Pohon {
9.      char data;
// Data yang disimpan
di node (tipe char)
10.     Pohon *left,
*right;      // Pointer ke
anak kiri dan anak
kanan
11.     Pohon *parent;
// Pointer ke node induk
12. };
13.
14. // Variabel global
untuk menyimpan root
(akar) pohon dan node
baru
15. Pohon *root, *baru;
16.
17. // Inisialisasi pohon
agar kosong
18. 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 << "\nNode " << data << " berhasil dibuat menjadi root." << endl;
    } else {
        cout << "\nPohon sudah dibuat." << endl; // Root sudah ada, tidak membuat node baru
    }
```

```
35. }
36.
37. // Menambahkan node baru sebagai anak kiri dari node tertentu
38. Pohon* insertLeft(char data, Pohon *node) {
39.     if (node->left != NULL) { // Jika anak kiri sudah ada
40.         cout << "\nNode " << node->data << " sudah ada child kiri!" << endl;
41.         return NULL; // Tidak menambahkan node baru
42.     }
43.     // Membuat node baru dan menghubungkannya ke node sebagai anak kiri
44.     baru = new Pohon{data, NULL, NULL, node};
45.     node->left = baru;
46.     cout << "\nNode " << data << " berhasil ditambahkan ke child kiri " << node->data << endl;
47.     return baru; // Mengembalikan pointer ke node baru
```

```
48. }
49.
50. // Menambahkan node baru sebagai anak kanan dari node tertentu
51. Pohon* insertRight(char data, Pohon *node) {
52.     if (node->right != NULL) { // Jika anak kanan sudah ada
53.         cout << "\nNode " << node->data << " sudah ada child kanan!" << endl;
54.         return NULL; // Tidak menambahkan node baru
55.     }
56.     // Membuat node baru dan menghubungkannya ke node sebagai anak kanan
57.     baru = new Pohon{data, NULL, NULL, node};
58.     node->right = baru;
59.     cout << "\nNode " << data << " berhasil ditambahkan ke child kanan " << node->data << endl;
```

```
60.     return baru; // Mengembalikan pointer ke node baru
61. }
62.
63. // Mengubah data di dalam sebuah node
64. void update(char data, Pohon *node) {
65.     if (!node) { // Jika node tidak ditemukan
66.         cout << "\nNode yang ingin diubah tidak ditemukan!" << endl;
67.         return;
68.     }
69.     char temp = node->data; // Menyimpan data lama
70.     node->data = data;      // Mengubah data dengan nilai baru
71.     cout << "\nNode " << temp << " berhasil diubah menjadi " << data << endl;
72. }
73.
74. // Mencari node dengan data tertentu
75. void find(char data, Pohon *node) {
76.     if (!node) return; // Jika node tidak ada, hentikan pencarian
```

```
77.
78.     if (node->data ==
data) { // Jika data
ditemukan
79.         cout <<
"\nNode ditemukan: "
<< data << endl;
80.         return;
81.     }
82.     // Melakukan
pencarian secara
rekursif ke anak kiri
dan kanan
83.     find(data, node-
>left);
84.     find(data, node-
>right);
85. }
86.
87. // Traversal Pre-
order (Node -> Kiri ->
Kanan)
88. void
preOrder(Pohon *node)
{
89.     if (!node) return;
// Jika node kosong,
hentikan traversal
90.     cout << node-
>data << " "; // Cetak
data node saat ini
91.     preOrder(node-
>left);      // Traversal
ke anak kiri
```

```
92.     preOrder(node->right);     // Traversal ke anak kanan
93. }
94.
95. // Traversal In-order (Kiri -> Node -> Kanan)
96. void inOrder(Pohon *node) {
97.     if (!node) return; // Jika node kosong, hentikan traversal
98.     inOrder(node->left);  // Traversal ke anak kiri
99.     cout << node->data << " "; // Cetak data node saat ini
100.    inOrder(node->right); // Traversal ke anak kanan
101.        }
102.
103.        // Traversal Post-order (Kiri -> Kanan -> Node)
104.        void postOrder(Pohon *node) {
105.            if (!node) return; // Jika node kosong, hentikan traversal
```

```
106.
postOrder(node->left);
// Traversal ke anak kiri
107.
postOrder(node->right);
// Traversal ke anak
kanan
108.            cout <<
node->data << " "; //
Cetak data node saat ini
109.        }
110.
111.        //
Menghapus node
dengan data tertentu
112.        Pohon*
deleteNode(Pohon
*node, char data) {
113.            if (!node)
return NULL; // Jika
node kosong, hentikan
114.
115.            //
Rekursif mencari node
yang akan dihapus
116.            if (data <
node->data) {
117.                node-
>left =
deleteNode(node->left,
data); // Cari di anak
kiri
118.            } else if
(data > node->data) {
119.                node-
>right =
```

```
deleteNode(node->right, data); // Cari di anak kanan
120.            } else {
121.                // Jika node ditemukan
122.                if (!node->left) { // Jika tidak ada anak kiri
123.                Pohon *temp = node->right; // Anak kanan menggantikan posisi node
124.                delete node;
125.                return temp;
126.                } else if (!node->right) { // Jika tidak ada anak kanan
127.                Pohon *temp = node->left; // Anak kiri menggantikan posisi node
128.                delete node;
129.                return temp;
130.                }
131.
132.                // Jika node memiliki dua anak, cari node pengganti (successor)
```

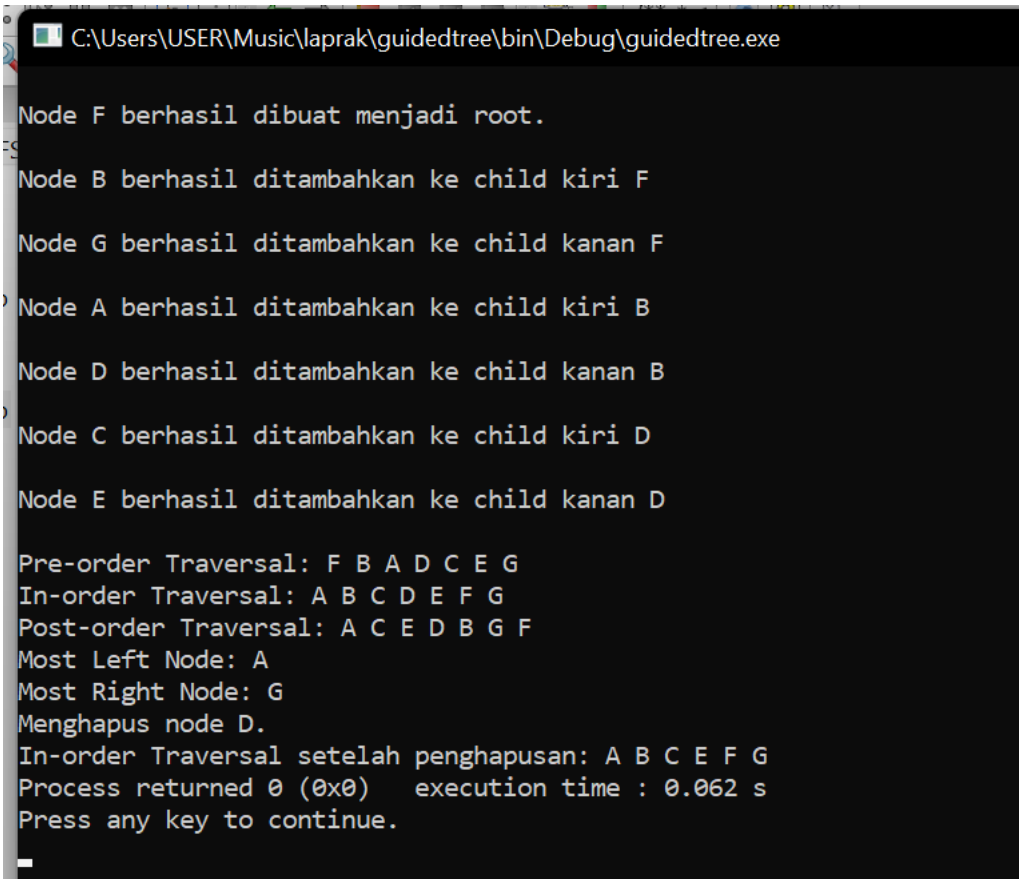
```
133.             Pohon
*successor = node-
>right;
134.             while
(successor->left)
successor = successor-
>left; // Cari node
terkecil di anak kanan
135.             node-
>data = successor-
>data; // Gantikan data
dengan successor
136.             node-
>right =
deleteNode(node-
>right, successor-
>data); // Hapus
successor
137.         }
138.         return
node;
139.     }
140.
141.     //
Menemukan node
paling kiri
142.     Pohon*
mostLeft(Pohon *node)
{
143.         if (!node)
return NULL; // Jika
node kosong, hentikan
144.         while
(node->left) node =
node->left; // Iterasi ke
```

```
anak kiri hingga
mentok
145.            return
node;
146.        }
147.
148.        //
Menemukan node
paling kanan
149.        Pohon*
mostRight(Pohon
*node) {
150.            if (!node)
return NULL; // Jika
node kosong, hentikan
151.            while
(node->right) node =
node->right; // Iterasi
ke anak kanan hingga
mentok
152.            return
node;
153.        }
154.
155.        // Fungsi
utama
156.        int main() {
157.            init(); //
Inisialisasi pohon
158.
buatNode('F'); //
Membuat root dengan
data 'F'
159.
insertLeft('B', root); //
```

```
Menambahkan 'B' ke
anak kiri root
160.
insertRight('G', root); //
Menambahkan 'G' ke
anak kanan root
161.
insertLeft('A', root-
>left); // Menambahkan
'A' ke anak kiri dari 'B'
162.
insertRight('D', root-
>left); // Menambahkan
'D' ke anak kanan dari
'B'
163.
insertLeft('C', root-
>left->right); //
Menambahkan 'C' ke
anak kiri dari 'D'
164.
insertRight('E', root-
>left->right); //
Menambahkan 'E' ke
anak kanan dari 'D'
165.
166.            //
Traversal pohon
167.            cout <<
"\nPre-order Traversal:
";
168.
preOrder(root);
169.            cout <<
"\nIn-order Traversal: ";
```

```
170.
inOrder(root);
171.            cout <<
"\nPost-order Traversal:
";
172.
postOrder(root);
173.
174.            //
Menampilkan node
paling kiri dan kanan
175.            cout <<
"\nMost Left Node: "
<< mostLeft(root)-
>data;
176.            cout <<
"\nMost Right Node: "
<< mostRight(root)-
>data;
177.
178.            //
Menghapus node
179.            cout <<
"\nMenghapus node
D.";
180.            root =
deleteNode(root, 'D');
181.            cout <<
"\nIn-order Traversal
setelah penghapusan: ";
182.
inOrder(root);
183.
184.            return 0;
185.        }
```

OUTPUT :



1. Struktur Pohon

Program ini menggunakan struktur Pohon untuk merepresentasikan pohon biner, di mana setiap node memiliki:

- data: Nilai yang disimpan (tipe char).
- left dan right: Pointer ke anak kiri dan kanan.
- parent: Pointer ke node induk (tidak selalu digunakan di semua fungsi).

2. Inisialisasi dan Pengecekan

- **init()**
  Menginisialisasi pohon agar kosong dengan mengatur root ke NULL.
- **isEmpty()**
  Mengecek apakah pohon kosong dengan memeriksa apakah root adalah NULL.

3. Membuat dan Menambahkan Node

- **buatNode(chardata)**
  Membuat node baru sebagai *root* pohon. Jika root sudah ada, program akan menolak pembuatan root baru.
- **insertLeft(char data, Pohon* node) dan insertRight(char data, Pohon*node)**
  Menambahkan node baru sebagai anak kiri atau kanan dari node tertentu. Jika posisi anak sudah diisi, program memberikan pesan peringatan.

4. Traversal Pohon

Traversal adalah cara menelusuri semua node dalam pohon. Program menyediakan tiga jenis traversal:

- **Pre-order**: Cetak node saat ini, lalu kunjungi anak kiri dan anak kanan. Fungsi: preOrder(Pohon* node)
- **In-order**: Kunjungi anak kiri, cetak node saat ini, lalu kunjungi anak kanan. Fungsi: inOrder(Pohon* node)
- **Post-order**: Kunjungi anak kiri, anak kanan, lalu cetak node saat ini. Fungsi: postOrder(Pohon* node)

5. Operasi Node

- **update(chardata,Pohon*node)**
  Mengubah nilai data di sebuah node tertentu. Jika node tidak ditemukan, program menampilkan pesan kesalahan.

- **find(chardata,Pohon*node)**
  Mencari node dengan data tertentu secara rekursif di seluruh pohon. Jika ditemukan, akan menampilkan pesan keberhasilan.

6. Penghapusan Node

- **deleteNode(Pohon*node,chardata)**
  Menghapus node dengan data tertentu:
  - Jika node tidak memiliki anak kiri atau kanan, node tersebut langsung dihapus.
  - Jika memiliki dua anak, program mencari *successor* (node terkecil di subtree kanan) untuk menggantikan node yang dihapus.

7. Node Paling Kiri dan Kanan

- **mostLeft(Pohon*node)**
  Menemukan node paling kiri dalam pohon (nilai terkecil).
- **mostRight(Pohon*node)**
  Menemukan node paling kanan dalam pohon (nilai terbesar).

4. Unguided

1. CODE :

```
#include <iostream>
#include <limits>
using namespace std;

struct Pohon {
    char data;
    Pohon *left, *right, *parent;
};

Pohon *root;

void init() {
    root = NULL;
}

bool isEmpty() {
    return root == NULL;
}

Pohon* buatNode(char data, Pohon* parent = NULL) {
    return new Pohon{data, NULL, NULL, parent};
}

Pohon* insertLeft(char data, Pohon* node) {
    if (!node) return NULL;
    if (node->left) {
        cout << "Child kiri sudah ada!\n";
        return NULL;
    }
    node->left = buatNode(data, node);
```

```
32.     return node->left;
33. }
34.
35. Pohon* insertRight(char data, Pohon*
    node) {
36.     if (!node) return NULL;
37.     if (node->right) {
38.         cout << "Child kanan sudah
    ada!\n";
39.         return NULL;
40.     }
41.     node->right = buatNode(data, node);
42.     return node->right;
43. }
44.
45. void preOrder(Pohon* node) {
46.     if (!node) return;
47.     cout << node->data << " ";
48.     preOrder(node->left);
49.     preOrder(node->right);
50. }
51.
52. void inOrder(Pohon* node) {
53.     if (!node) return;
54.     inOrder(node->left);
55.     cout << node->data << " ";
56.     inOrder(node->right);
57. }
58.
59. void postOrder(Pohon* node) {
60.     if (!node) return;
61.     postOrder(node->left);
62.     postOrder(node->right);
63.     cout << node->data << " ";
64. }
65.
```

```
66. void tampilkanChild(Pohon* node) {
67.     if (!node) return;
68.     cout << "Node " << node->data << "
    memiliki:\n";
69.     if (node->left) cout << "- Child kiri: "
    << node->left->data << endl;
70.     else cout << "- Tidak ada child
    kiri.\n";
71.     if (node->right) cout << "- Child
    kanan: " << node->right->data << endl;
72.     else cout << "- Tidak ada child
    kanan.\n";
73. }
74.
75. void tampilkanDescendant(Pohon*
    node) {
76.     if (!node) return;
77.     cout << "Descendant dari " << node-
    >data << ": ";
78.     preOrder(node);
79.     cout << endl;
80. }
81.
82. bool is_valid_bst(Pohon* node, char
    min_val, char max_val) {
83.     if (!node) return true;
84.     if (node->data <= min_val || node-
    >data >= max_val) return false;
85.     return is_valid_bst(node->left,
    min_val, node->data) &&
86.            is_valid_bst(node->right, node-
    >data, max_val);
87. }
88.
89. int cari_simpul_daun(Pohon* node) {
90.     if (!node) return 0;
```

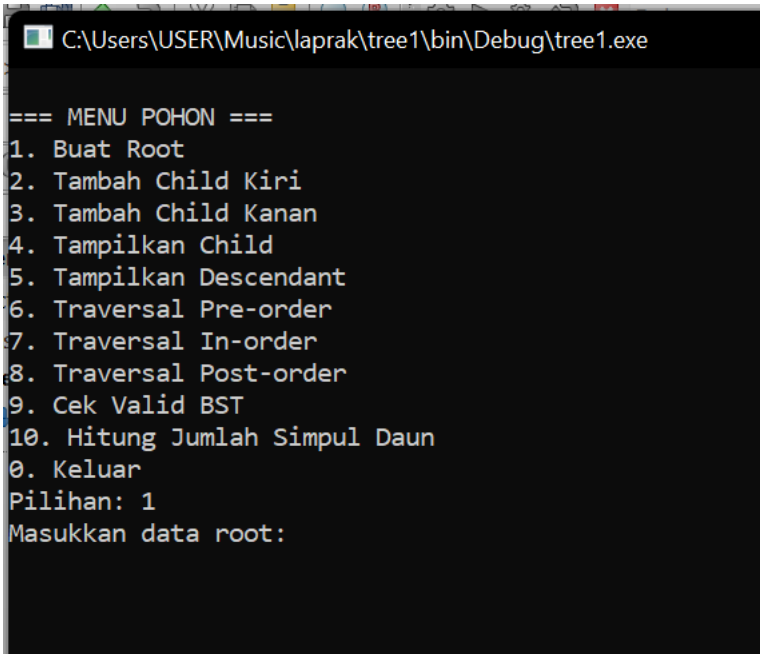
```
91.     if (!node->left && !node->right)
    return 1;
92.     return cari_simpul_daun(node->left)
    + cari_simpul_daun(node->right);
93. }
94.
95. void menu() {
96.     int pilihan;
97.     char data, parent;
98.     Pohon* currentNode = NULL;
99.
100.        do {
101.            cout << "\n=== MENU
    POHON ===\n";
102.            cout << "1. Buat Root\n";
103.            cout << "2. Tambah Child
    Kiri\n";
104.            cout << "3. Tambah Child
    Kanan\n";
105.            cout << "4. Tampilkan
    Child\n";
106.            cout << "5. Tampilkan
    Descendant\n";
107.            cout << "6. Traversal Pre-
    order\n";
108.            cout << "7. Traversal In-
    order\n";
109.            cout << "8. Traversal Post-
    order\n";
110.            cout << "9. Cek Valid BST\n";
111.            cout << "10. Hitung Jumlah
    Simpul Daun\n";
112.            cout << "0. Keluar\n";
113.            cout << "Pilihan: ";
114.            cin >> pilihan;
115.
```

```
116.            switch (pilihan) {
117.                case 1:
118.                    if (!root) {
119.                        cout << "Masukkan
    data root: ";
120.                        cin >> data;
121.                        root = buatNode(data);
122.                        cout << "Root berhasil
    dibuat.\n";
123.                    } else {
124.                        cout << "Root sudah
    ada.\n";
125.                    }
126.                    break;
127.                case 2:
128.                    cout << "Masukkan parent
    node: ";
129.                    cin >> parent;
130.                    cout << "Masukkan data:
    ";
131.                    cin >> data;
132.                    currentNode =
    insertLeft(data, root);
133.                    if (currentNode) cout <<
    "Child kiri berhasil ditambahkan.\n";
134.                    break;
135.                case 3:
136.                    cout << "Masukkan parent
    node: ";
137.                    cin >> parent;
138.                    cout << "Masukkan data:
    ";
139.                    cin >> data;
140.                    currentNode =
    insertRight(data, root);
```

```
141.                if (currentNode) cout <<
    "Child kanan berhasil ditambahkan.\n";
142.                break;
143.            case 4:
144.                cout << "Masukkan node:
    ";
145.                cin >> data;
146.                tampilkanChild(root);
147.                break;
148.            case 5:
149.                cout << "Masukkan node:
    ";
150.                cin >> data;
151.
    tampilkanDescendant(root);
152.                break;
153.            case 6:
154.                cout << "Pre-order
    Traversal: ";
155.                preOrder(root);
156.                cout << endl;
157.                break;
158.            case 7:
159.                cout << "In-order
    Traversal: ";
160.                inOrder(root);
161.                cout << endl;
162.                break;
163.            case 8:
164.                cout << "Post-order
    Traversal: ";
165.                postOrder(root);
166.                cout << endl;
167.                break;
168.            case 9:
```

```
169.                 if (is_valid_bst(root,
    numeric_limits<char>::min(),
    numeric_limits<char>::max())) {
170.                     cout << "Pohon adalah
    Binary Search Tree.\n";
171.                 } else {
172.                     cout << "Pohon bukan
    Binary Search Tree.\n";
173.                 }
174.                 break;
175.             case 10:
176.                 cout << "Jumlah simpul
    daun: " << cari_simpul_daun(root) <<
    endl;
177.                 break;
178.             case 0:
179.                 cout << "Keluar dari
    program.\n";
180.                 break;
181.             default:
182.                 cout << "Pilihan tidak
    valid.\n";
183.             }
184.         } while (pilihan != 0);
185.     }
186.
187.     int main() {
188.         init();
189.         menu();
190.         return 0;
191.     }
```

OUTPUT :



- **Struktur Data**
  Program menggunakan struktur data Pohon untuk merepresentasikan node dalam pohon biner. Setiap node menyimpan data, pointer ke anak kiri, kanan, dan induknya.
- **Inisialisasi dan Kondisi Kosong**
  Fungsi init mengatur pohon sebagai kosong (root = NULL), sementara isEmpty mengecek apakah pohon kosong.
- **Membuat Node Baru**
  Fungsi buatNode digunakan untuk membuat node baru dengan data tertentu. Jika root belum ada, root dapat dibuat terlebih dahulu.
- **Menambahkan Anak (Child)**
  Fungsi insertLeft dan insertRight menambahkan node baru sebagai anak kiri atau kanan dari node tertentu, dengan

mengecek apakah posisi anak tersebut masih kosong.

- **Traversal Pohon**
  Terdapat tiga jenis traversal:
  - **Pre-order**: Cetak node saat ini, lalu anak kiri, dan anak kanan.
  - **In-order**: Anak kiri, cetak node saat ini, dan anak kanan.
  - **Post-order**: Anak kiri, anak kanan, lalu cetak node saat ini.

- **Menampilkan Child dan Descendant**
  - tampilkanChild: Menampilkan data anak kiri dan kanan dari node tertentu.
  - tampilkanDescendant: Menampilkan semua turunan (descendant) dari node menggunakan traversal pre-order.
- **Validasi Binary Search Tree (BST)**
  Fungsi is_valid_bst memastikan bahwa pohon memenuhi aturan BST, yaitu:
  - Nilai semua node di subtree kiri lebih kecil dari node saat ini.
  - Nilai semua node di subtree kanan lebih besar dari node saat ini.
- **Menghitung Simpul Daun**
  Fungsi cari_simpul_daun menghitung jumlah node yang tidak memiliki anak kiri maupun kanan (simpul daun).
- **Menu Interaktif**
  Program memiliki menu untuk menjalankan operasi pada pohon secara interaktif, seperti membuat root, menambah anak, traversal, validasi BST, dan menghitung simpul daun.

- **Fitur Tambahan**

    - Validasi BST cocok untuk memeriksa apakah pohon memenuhi struktur yang benar.

    - Perhitungan simpul daun memberikan informasi tentang struktur akhir pohon.

5. Kesimpulan

Praktikum ini berfokus pada pemahaman dan penerapan konsep pohon biner, termasuk proses pembuatan node, penambahan elemen, serta traversal pohon menggunakan metode *in-order*, *pre-order*, dan *post-order*. Selain itu, praktikum juga menekankan pentingnya penggunaan fungsi rekursif untuk mempermudah proses manipulasi data dalam struktur pohon. Dengan pemahaman ini, peserta dapat menguasai konsep dasar dari struktur data pohon yang banyak diaplikasikan dalam berbagai bidang, seperti pengolahan data, pencarian, dan pengambilan keputusan.