

LAPORAN PRAKTIKUM

MODUL VIII

“QUEUE (ANTREAN)”



Disusun Oleh:

Alya Rabani - 2311104076

S1SE-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

Fungsi rekursif adalah proses di mana suatu fungsi memanggil dirinya sendiri secara langsung atau tidak langsung. Dengan menggunakan algoritma rekursif, masalah tertentu dapat diselesaikan dengan cukup mudah. Pada beberapa kasus fungsi rekursif bisa lebih mudah dipahami dan sederhana daripada dengan Solusi iterative. Maka dari itu penggunaan fungsi ini dapat membantu mengurangi jumlah kode yang dibutuhkan serta memudahkan untuk dibaca. Di dalam system pemrograman sendiri, rekursif memiliki dua jenis yaitu, rekursif langsung dimana fungsi memanggil dirinya sendiri secara langsung dalam proses eksekusi dan tidak langsung merupakan fungsi a memanggil fungsi b dan fungsi b memanggil fungsi a.

Salah satu bentuk rekursi langsung adalah tree recursion di mana fungsi memanggil dirinya sendiri lebih dari satu kali dalam satu langkah eksekusi. Pola ini menciptakan struktur seperti pohon (tree structure), karena setiap pemanggilan menghasilkan cabang baru yang dapat memanggil rekursi lagi.

Ciri-Ciri Tree Recursion:

- Fungsi memanggil dirinya sendiri beberapa kali dalam satu langkah eksekusi.
- Biasanya digunakan untuk masalah yang memiliki sub-masalah bercabang.
- Kompleksitas waktu cenderung meningkat secara eksponensial karena banyaknya pemanggilan ulang fungsi untuk nilai yang sama (kecuali menggunakan teknik seperti memoization).

Pada struktur data tree adalah struktur hierarkis yang terdiri dari Kumpulan simpul (nodes) yang dihubungkan oleh cabang (edges). Tree sering digunakan untuk merepresentasikan data yang memiliki hubungan hierarkis atau bersarang, seperti struktur file di komputer, silsilah keluarga, atau ekspresi matematika. Komponen dasar tree ada Root yang merupakan node utama atau simpul paling atas dari tree, Parent node yang memiliki cabang menuju node lain, child node yang terhubung dari parent, leaf node yang tidak memiliki child, edge hubungan atau garis yang menghubungkan dua node, subtree bagian dari tree yang merupakan tree kecil dengan root sendiri, level jarak suatu node dari root dalam hal jumlah edges, height panjang jalur terpanjang dari root ke leaf, degree jumlah child yang dimiliki oleh suatu node.

Tree memiliki banyak jenis yang paling familiar adalah Binary Search Tree (BST). Digunakan untuk pencarian cepat, penyisipan, dan penghapusan data yang teratur. BST memiliki keunggulan seperti operasi pencarian, penyisipan, dan penghapusan dapat dilakukan dalam waktu rata-rata $O(\log n)$.

jika tree seimbang. BST biasa digunakan untuk penggunaan basis data untuk indeks sederhana dan aplikasi pencarian sistem yang membutuhkan data terurut, seperti daftar produk. Namun, BST tidak selalu efisien jika tree menjadi tidak seimbang (skewed), sehingga perlu optimisasi seperti AVL atau Red-Black Tree.

Operasi-operasi yang digunakan pada tree seperti berikut:

- Traversal/Penelusuran Tree, adalah proses menjelajahi semua node dalam tree. Contohnya:
 - ✓ Pre-order: Root → Left → Right.
 - ✓ In-order: Left → Root → Right.
 - ✓ Post-order: Left → Right → Root.
 - ✓ Level-order: Traversal per level dari atas ke bawah.
- Insertion/penyisipan, untuk menambahkan node baru sesuai aturan tree.
- Deletion/penghapusan, menghapus node dan menjaga struktur tree tetap valid.
- Search, pencarian node tertentu berdasarkan nilai.

3. Guided

Pada guided ini merupakan implementasi dari struktur data tree biner. Program ini dimulai dengan mendefinisikan struktur tree, yang menyimpan data karakter dan pointer ke anak kiri, anak kanan, serta induk dari setiap node. Fungsi init digunakan untuk menginisialisasi pohon agar kosong, sedangkan fungsi isEmpty memeriksa apakah pohon tersebut kosong. Fungsi buatNode membuat node baru sebagai akar pohon jika pohon masih kosong. Selanjutnya, terdapat fungsi insertLeft dan insertRight untuk menambahkan node baru sebagai anak kiri atau kanan dari node tertentu, dengan pengecekan untuk memastikan bahwa anak tersebut belum ada. Program juga menyediakan fungsi untuk memperbarui data node, mencari node berdasarkan data, serta melakukan traversal pohon dengan metode pre-order, in-order, dan post-order. Selain itu, terdapat fungsi untuk menghapus node tertentu dan menemukan node paling kiri dan kanan dalam pohon. Di dalam fungsi main, pohon diinisialisasi, beberapa node ditambahkan, dan dilakukan traversal untuk menampilkan struktur pohon. Program juga menunjukkan cara menghapus node dan menampilkan hasil traversal setelah penghapusan.

Kode program:

```

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, *right; // 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, NULL}; // Membuat node baru sebagai root
30         cout << "\nNode " << data << " berhasil dibuat menjadi root." << endl;
31     } else {
32         cout << "\nPohon sudah dibuat." << endl; // Root sudah ada, tidak membuat node baru
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 << "\nNode " << node->data << " sudah ada child kiri!" << endl;
40         return NULL; // Tidak menambahkan node baru
41     }
42     // Membuat node baru dan menghubungkannya ke node sebagai anak kiri
43     baru = new Pohon{data, NULL, NULL, node};
44     node->left = baru;
45     cout << "\nNode " << data << " berhasil ditambahkan ke child kiri " << node->data << endl;
46     return baru; // Mengembalikan pointer ke node baru
47 }
48
49 // Menambahkan node baru sebagai anak kanan dari node tertentu
50 Pohon* insertRight(char data, Pohon *node) {
51     if (node->right != NULL) { // Jika anak kanan sudah ada
52         cout << "\nNode " << node->data << " sudah ada child kanan!" << endl;
53         return NULL; // Tidak menambahkan node baru
54     }
55     // Membuat node baru dan menghubungkannya ke node sebagai anak kanan
56     baru = new Pohon{data, NULL, NULL, node};
57     node->right = baru;
58     cout << "\nNode " << data << " berhasil ditambahkan ke child kanan " << node->data << endl;
59     return baru; // Mengembalikan pointer ke node baru
60 }
61
62 // Mengubah data di dalam sebuah node
63 void update(char data, Pohon *node) {
64     if (!node) { // Jika node tidak ditemukan
65         cout << "\nNode yang ingin diubah tidak ditemukan!" << endl;
66         return;
67     }
68     char temp = node->data; // Menyimpan data lama
69     node->data = data;      // Mengubah data dengan nilai baru
70     cout << "\nNode " << temp << " berhasil diubah menjadi " << data << endl;
71 }
72
73 // Mencari node dengan data tertentu
74 void find(char data, Pohon *node) {
75     if (!node) return; // Jika node tidak ada, hentikan pencarian
76
77     if (node->data == data) { // Jika data ditemukan
78         cout << "\nNode ditemukan: " << data << endl;
79         return;
80     }
81     // Melakukan pencarian secara rekursif ke anak kiri dan kanan
82     find(data, node->left);
83     find(data, node->right);
84 }

```

```

86 // Traversal Pre-order (Node -> Kiri -> Kanan)
87 void preOrder(Pohon *node) {
88     if (!node) return; // Jika node kosong, hentikan traversal
89     cout << node->data << " "; // Cetak data node saat ini
90     preOrder(node->left); // Traversal ke anak kiri
91     preOrder(node->right); // Traversal ke anak kanan
92 }
93
94 // Traversal In-order (Kiri -> Node -> Kanan)
95 void inOrder(Pohon *node) {
96     if (!node) return; // Jika node kosong, hentikan traversal
97     inOrder(node->left); // Traversal ke anak kiri
98     cout << node->data << " "; // Cetak data node saat ini
99     inOrder(node->right); // Traversal ke anak kanan
100 }
101
102 // Traversal Post-order (Kiri -> Kanan -> Node)
103 void postOrder(Pohon *node) {
104     if (!node) return; // Jika node kosong, hentikan traversal
105     postOrder(node->left); // Traversal ke anak kiri
106     postOrder(node->right); // Traversal ke anak kanan
107     cout << node->data << " "; // Cetak data node saat ini
108 }
109
110 // Menghapus node dengan data tertentu
111 Pohon* deleteNode(Pohon *node, char data) {
112     if (!node) return NULL; // Jika node kosong, hentikan
113
114     // Rekursif mencari node yang akan dihapus
115     if (data < node->data) {
116         node->left = deleteNode(node->left, data); // Cari di anak kiri
117     } else if (data > node->data) {
118         node->right = deleteNode(node->right, data); // Cari di anak kanan
119     } else {
120         // Jika node ditemukan
121         if (!node->left) { // Jika tidak ada anak kiri
122             Pohon *temp = node->right; // Anak kanan menggantikan posisi node
123             delete node;
124             return temp;
125         } else if (!node->right) { // Jika tidak ada anak kanan
126             Pohon *temp = node->left; // Anak kiri menggantikan posisi node
127             delete node;
128             return temp;
129         }
130
131         // Jika node memiliki dua anak, cari node pengganti (successor)
132         Pohon *successor = node->right;
133         while (successor->left) successor = successor->left; // Cari node terkecil di anak kanan
134         node->data = successor->data; // Gantikan data dengan successor
135         node->right = deleteNode(node->right, successor->data); // Hapus successor
136     }
137     return node;
138 }
139
140 // Menemukan node paling kiri
141 Pohon* mostLeft(Pohon *node) {
142     if (!node) return NULL; // Jika node kosong, hentikan
143     while (node->left) node = node->left; // Iterasi ke anak kiri hingga mentok
144     return node;
145 }
146
147 // Menemukan node paling kanan
148 Pohon* mostRight(Pohon *node) {
149     if (!node) return NULL; // Jika node kosong, hentikan
150     while (node->right) node = node->right; // Iterasi ke anak kanan hingga mentok
151     return node;
152 }
153
154 // Fungsi utama
155 int main() {
156     init(); // Inisialisasi pohon
157     buatNode('F'); // Membuat root dengan data 'F'
158     insertLeft('B', root); // Menambahkan 'B' ke anak kiri root
159     insertRight('G', root); // Menambahkan 'G' ke anak kanan root
160     insertLeft('A', root->left); // Menambahkan 'A' ke anak kiri dari 'B'
161     insertRight('D', root->left); // Menambahkan 'D' ke anak kanan dari 'B'
162     insertLeft('C', root->left->right); // Menambahkan 'C' ke anak kiri dari 'D'
163     insertRight('E', root->left->right); // Menambahkan 'E' ke anak kanan dari 'D'
164
165     // Traversal pohon
166     cout << "\nPre-order Traversal: ";
167     preOrder(root);
168     cout << "\nIn-order Traversal: ";
169     inOrder(root);
170     cout << "\nPost-order Traversal: ";
171     postOrder(root);
172
173     // Menampilkan node paling kiri dan kanan
174     cout << "\nMost Left Node: " << mostLeft(root)->data;
175     cout << "\nMost Right Node: " << mostRight(root)->data;
176
177     // Menghapus node
178     cout << "\nMenghapus node D.";
179     root = deleteNode(root, 'D');
180     cout << "\nIn-order Traversal setelah penghapusan: ";
181     inOrder(root);
182
183     return 0;
184 }

```

Hasil output dari program:

```
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
PS D:\tugas yall\praktikum sd\pertemuan 10\output>
```

4. Unguided

Program ini menggunakan struktur data tree untuk menyimpan node yang dibuat agar user yang menginputkannya sendiri, memiliki anak kiri dan kanan serta mendukung operasi tambahan sesuai dengan ketiga soal yang diberikan, seperti mencari node, memeriksa properti Binary Search Tree (BST), dan menghitung jumlah simpul daun.

Node tree berisi data (huruf atau karakter) dan tiga pointer:

left → menunjuk ke anak kiri.

right → menunjuk ke anak kanan.

parent → menunjuk ke induk dari node.

Fungsi pada program yang awalnya sudah ada seperti mengecek apakah tree kosong, membuat node baru sebagai root, serta menambahkan node baru sebagai anak kiri atau anak kanan dari node tertentu.

Fungsi tambahan program seperti soal pertama yang menyuruh untuk menampilkan anak dan descendant dari node tertentu. Kemudian pada soal kedua dibuat pencarian node berdasarkan data secara rekursif dan juga dilakukan uji fungsi BST pada tree. Soal ketiga membuat fungsi untuk menghitung jumlah simpul daun.

Pada fungsi utama dibuat menu program, untuk membuat menu itu berjalan digunakan switch case yang memberikan menu interaktif kepada pengguna agar mereka dapat memilih aksi tertentu. Setiap case mewakili pilihan menu yang akan dijalankan.

- Case 1: Buat Root

Membuat node pertama (root) dari pohon. Jika root sudah ada, program akan memberi peringatan bahwa pohon sudah dibuat.

- Case 2: Tambah Child Kiri
Menambahkan node baru sebagai anak kiri dari node yang ditentukan oleh pengguna. Menggunakan fungsi findNode untuk mencari node induk.
- Case 3: Tambah Child Kanan
Menambahkan node baru sebagai anak kanan dari node yang ditentukan oleh pengguna. Sama seperti Case 2, menggunakan findNode untuk mencari node induk.
- Case 4: Tampilkan Child Node
Menampilkan anak kiri dan anak kanan dari node tertentu. Jika node tidak memiliki anak, program akan memberi keterangan bahwa anak tersebut kosong.
- Case 5: Tampilkan Descendant Node
Menampilkan semua descendant (keturunan) dari node yang diinputkan, termasuk anak, cucu, cicit, dan seterusnya. Descendant dicetak menggunakan traversal Pre-order mulai dari anak kiri ke kanan.
- Case 6: Periksa Apakah Pohon Adalah BST
Memeriksa apakah pohon memenuhi properti Binary Search Tree (BST). Terdapat syarat BST untuk setiap node yaitu, semua node di anak kiri memiliki nilai lebih kecil dan semua node di anak kanan memiliki nilai lebih besar.
- Case 7: Hitung Jumlah Simpul Daun
Menghitung jumlah simpul daun dalam pohon biner. Simpul daun adalah node yang tidak memiliki anak kiri maupun kanan.
- Case 0: Keluar
Program akan keluar dari menu.

```

1  #include <iostream>
2  #include <limits> // Untuk nilai batas min dan max
3  using namespace std;
4
5  // Struktur data pohon biner
6  struct Pohon {
7      char data;
8      Pohon *left, *right;
9      Pohon *parent;
10 };
11
12 Pohon *root, *baru;
13
14 // Inisialisasi pohon
15 void init() {
16     root = NULL;
17 }
18
19 // Mengecek apakah pohon kosong
20 bool isEmpty() {
21     return root == NULL;
22 }
23
24 // Membuat node baru sebagai root
25 void buatNode(char data) {
26     if (isEmpty()) {
27         root = new Pohon{data, NULL, NULL, NULL};
28         cout << "Node " << data << " berhasil dibuat sebagai root.\n";
29     } else {
30         cout << "Pohon sudah ada!\n";
31     }
32 }
33
34 // Menambahkan node ke kiri
35 Pohon* insertLeft(char data, Pohon *node) {
36     if (node->left != NULL) {
37         cout << "Child kiri dari " << node->data << " sudah ada!\n";
38         return NULL;
39     }
40     baru = new Pohon{data, NULL, NULL, node};
41     node->left = baru;
42     cout << "Node " << data << " ditambahkan ke kiri dari " << node->data << "\n";
43     return baru;
44 }
45
46 // Menambahkan node ke kanan
47 Pohon* insertRight(char data, Pohon *node) {
48     if (node->right != NULL) {
49         cout << "Child kanan dari " << node->data << " sudah ada!\n";
50         return NULL;
51     }
52     baru = new Pohon{data, NULL, NULL, node};
53     node->right = baru;
54     cout << "Node " << data << " ditambahkan ke kanan dari " << node->data << "\n";
55     return baru;
56 }
57
58 // 2 Modifikasi mencari node berdasarkan data secara rekursif
59 Pohon* findNode(char data, Pohon *node) {
60     if (!node) return NULL; // Basis rekursif: jika node NULL, kembalikan NULL
61
62     if (node->data == data) // Jika data cocok, kembalikan node
63         return node;
64
65     // Cari ke anak kiri dan kanan secara rekursif
66     Pohon *leftResult = findNode(data, node->left);
67     if (leftResult) return leftResult; // Jika ditemukan di anak kiri, kembalikan hasil
68
69     Pohon *rightResult = findNode(data, node->right);
70     return rightResult; // Jika ditemukan di anak kanan, kembalikan hasil
71 }
72

```



```

72
73 // 1 Modifikasi menampilkan child dari node tertentu
74 void tampilChild(Pohon *node) {
75     if (!node) {
76         cout << "Node tidak ditemukan.\n";
77         return;
78     }
79     cout << "Node: " << node->data << "\n";
80     if (node->left) cout << "Child Kiri: " << node->left->data << "\n";
81     else cout << "Child Kiri: NULL\n";
82     if (node->right) cout << "Child Kanan: " << node->right->data << "\n";
83     else cout << "Child Kanan: NULL\n";
84 }
85
86 // 1 Modifikasi menampilkan descendant dari node tertentu (rekursif)
87 void tampilDescendant(Pohon *node) {
88     if (!node) return;
89     if (node->left || node->right)
90         cout << node->data << " memiliki descendant: ";
91     if (node->left) cout << node->left->data << " ";
92     if (node->right) cout << node->right->data << " ";
93     cout << "\n";
94     tampilDescendant(node->left);
95     tampilDescendant(node->right);
96 }
97
98 // 2 Modifikasi menambahkan fungsi validasi apakah pohon adalah BST
99 bool is_valid_bst(Pohon *node, char min_val, char max_val) {
100     if (!node) return true;
101     if (node->data <= min_val || node->data >= max_val) return false;
102     return is_valid_bst(node->left, min_val, node->data) &&
103         is_valid_bst(node->right, node->data, max_val);
104 }
105
106 // 3 Modifikasi menghitung jumlah simpul daun
107 int cari_simpul_daun(Pohon *node) {
108     if (!node) return 0; // Basis rekursi
109     if (!node->left && !node->right) return 1; // Node daun
110     return cari_simpul_daun(node->left) + cari_simpul_daun(node->right);
111 }
112
113 // Fungsi utama dengan menu
114 int main()
115 {
116     int pilihan;
117     char data, parent;
118     Pohon *temp = NULL;
119     init();
120
121     // 1 Modifikasi membuat menu program agar user dapat menginputkan sendiri
122     do {
123         cout << "\n=== MENU POHON BINARY TREE ===\n";
124         cout << "1. Buat Root\n";
125         cout << "2. Tambah Child Kiri\n";
126         cout << "3. Tambah Child Kanan\n";
127         cout << "4. Tampilkan Child Node\n";
128         cout << "5. Tampilkan Descendant Node\n";
129         cout << "6. Cek Apakah Pohon adalah BST\n";
130         cout << "7. Hitung Jumlah Simpul Daun\n";
131         cout << "0. Keluar\n";
132         cout << "Pilihan: ";
133         cin >> pilihan;

```

```

135     switch (pilihan) {
136     case 1:
137         cout << "Masukkan data root: ";
138         cin >> data;
139         buatNode(data);
140         break;
141     case 2:
142         cout << "Masukkan data parent: ";
143         cin >> parent;
144         cout << "Masukkan data anak kiri: ";
145         cin >> data;
146         temp = findNode(parent, root); // Cari node berdasarkan data
147         if (temp) {
148             insertLeft(data, temp);
149         } else {
150             cout << "Node dengan data " << parent << " tidak ditemukan!\n";
151         }
152         break;
153     case 3:
154         cout << "Masukkan data parent: ";
155         cin >> parent;
156         cout << "Masukkan data anak kanan: ";
157         cin >> data;
158         temp = findNode(parent, root); // Cari node berdasarkan data
159         if (temp) {
160             insertRight(data, temp);
161         } else {
162             cout << "Node dengan data " << parent << " tidak ditemukan!\n";
163         }
164         break;
165     case 4:
166         cout << "Masukkan node yang ingin ditampilkan child-nya: ";
167         cin >> data;
168         temp = findNode(data, root); // Cari node berdasarkan data
169         if (temp) {
170             tampilChild(temp);
171         } else {
172             cout << "Node dengan data " << data << " tidak ditemukan!\n";
173         }
174         break;
175     case 5:
176         cout << "Masukkan node yang ingin ditampilkan descendant-nya: ";
177         cin >> data;
178         tampilDescendant(findNode(data, root));
179         break;
180     case 6:
181         if (is_valid_bst(root, numeric_limits<char>::min(), numeric_limits<char>::max()))
182             cout << "Pohon adalah BST yang valid.\n";
183         else
184             cout << "Pohon bukan BST yang valid.\n";
185         break;
186     case 7:
187         cout << "Jumlah simpul daun: " << cari_simpul_daun(root) << "\n";
188         break;
189     case 0:
190         cout << "Keluar...\n";
191         break;
192     default:
193         cout << "Pilihan tidak valid!\n";
194     }
195 } while (pilihan != 0);
196
197 return 0;
198 }
199

```

Hasil output dari program akan menjadi seperti berikut:

```
=== MENU POHON BINARY TREE ===
1. Buat Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child Node
5. Tampilkan Descendant Node
6. Cek Apakah Pohon adalah BST
7. Hitung Jumlah Simpul Daun
0. Keluar
Pilihan: 1
Masukkan data root: 7
Node 7 berhasil dibuat sebagai root.
```

```
=== MENU POHON BINARY TREE ===
1. Buat Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child Node
5. Tampilkan Descendant Node
6. Cek Apakah Pohon adalah BST
7. Hitung Jumlah Simpul Daun
0. Keluar
Pilihan: 2
Masukkan data parent: 7
Masukkan data anak kiri: 5
Node 5 ditambahkan ke kiri dari 7
```

```
=== MENU POHON BINARY TREE ===
1. Buat Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child Node
5. Tampilkan Descendant Node
6. Cek Apakah Pohon adalah BST
7. Hitung Jumlah Simpul Daun
0. Keluar
Pilihan: 3
Masukkan data parent: 7
Masukkan data anak kanan: 6
Node 6 ditambahkan ke kanan dari 7

=== MENU POHON BINARY TREE ===
1. Buat Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child Node
5. Tampilkan Descendant Node
6. Cek Apakah Pohon adalah BST
7. Hitung Jumlah Simpul Daun
0. Keluar
Pilihan: 4
Masukkan node yang ingin ditampilkan child-nya: 7
Node: 7
Child Kiri: 5
Child Kanan: 6
```

```
=== MENU POHON BINARY TREE ===
1. Buat Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child Node
5. Tampilkan Descendant Node
6. Cek Apakah Pohon adalah BST
7. Hitung Jumlah Simpul Daun
0. Keluar
Pilihan: 5
Masukkan node yang ingin ditampilkan descendant-nya: 7
7 memiliki descendant: 5 6
```

```
=== MENU POHON BINARY TREE ===
1. Buat Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child Node
5. Tampilkan Descendant Node
6. Cek Apakah Pohon adalah BST
7. Hitung Jumlah Simpul Daun
0. Keluar
Pilihan: 6
Pohon bukan BST yang valid.
```

```
=== MENU POHON BINARY TREE ===
1. Buat Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child Node
5. Tampilkan Descendant Node
6. Cek Apakah Pohon adalah BST
7. Hitung Jumlah Simpul Daun
0. Keluar
Pilihan: 7
Jumlah simpul daun: 2
```

```
=== MENU POHON BINARY TREE ===
1. Buat Root
2. Tambah Child Kiri
3. Tambah Child Kanan
4. Tampilkan Child Node
5. Tampilkan Descendant Node
6. Cek Apakah Pohon adalah BST
7. Hitung Jumlah Simpul Daun
0. Keluar
Pilihan: 0
Keluar...
```

5. Kesimpulan

Pada laporan ini menyajikan implementasi struktur data Binary Tree menggunakan konsep rekursi. Pembuatan program yang dapat melakukan berbagai operasi pada pohon biner, mulai dari membuat pohon baru, menambahkan atau menghapus simpul, hingga melakukan penelusuran pohon dengan berbagai metode (pre-order, in-order, post-order). Selain itu, pada guided dibuat modifikasi program yang dilengkapi fitur tambahan seperti menampilkan keturunan suatu simpul, memeriksa apakah pohon memenuhi syarat sebagai pohon pencarian biner (Binary Search Tree), serta menghitung jumlah daun pada pohon. Melalui praktikum ini, pemahaman mengenai penerapan rekursi dan struktur pohon biner dalam menyelesaikan masalah pemrograman yang bersifat hierarkis semakin diperdalam.