

**LAPORAN PRAKTIKUM
ALGORITMA & STRUKTUR DATA
MODUL 7**



TREE (POHON)

Oleh:

Noor Khalisa NIM. 2410817220012

**PROGRAM STUDI TEKNOLOGI INFORMASI
FAKULTAS TEKNIK
UNIVERSITAS LAMBUNG MANGKURAT
JUNI 2025**

LEMBAR PENGESAHAN
LAPORAN PRAKTIKUM ALGORITMA & STRUKTUR DATA
MODUL 7

Laporan Praktikum Algoritma & Struktur Data Modul 7: Tree (Pohon) ini disusun sebagai syarat lulus mata kuliah Praktikum Algoritma & Struktur Data. Laporan Praktikum ini dikerjakan oleh:

Nama Praktikan : Noor Khalisa
NIM : 24101817220012

Menyetujui,
Asisten Praktikum

Mengetahui,
Dosen Penanggung Jawab Praktikum

Muhammad Fauzan Ahsani
NIM. 2310817310009

Muti'a Maulida, S.Kom., M.TI.
NIP. 198810272019032013

DAFTAR ISI

LEMBAR PENGESAHAN	2
DAFTAR ISI.....	3
DAFTAR GAMBAR	4
DAFTAR TABEL	5
SOAL	6
A. Source Code	8
B. Output Program.....	11
C. Pembahasan.....	14

DAFTAR GAMBAR

Gambar 1 Tampilan PreOrder Ketika Pohon Kosong	12
Gambar 2 Tampilan InOrder Ketika Pohon Kosong.....	12
Gambar 3 Tampilan PostOrder Ketika Pohon Kosong	12
Gambar 4 Tampilan Ketika Pilihan Tidak Valid	12
Gambar 5 Input Data Pertama.....	12
Gambar 6 Input Data Kedua	12
Gambar 7 Input Data Ketiga	13
Gambar 8 Input Data Keempat	13
Gambar 9 Input Data Kelima	13
Gambar 10 Input Data Keenam.....	13
Gambar 11 Input Data Ketujuh Tetapi Data Sudah Ada.....	13
Gambar 12 Input Data Kedelapan.....	13
Gambar 13 PreOrder Traversal	14
Gambar 14 InOrder Traversal	14
Gambar 15 PostOrder Traversal.....	14
Gambar 16 Exit	14

DAFTAR TABEL

Tabel 1 Source Code Soal.....	8
-------------------------------	---

SOAL

Cobalah program berikut, perbaiki output, lengkapi fungsi inOrder dan postOrder pada coding, running, simpan program!

```
1  #include <stdio.h>
2  #include <conio.h>
3  #include <stdlib.h>
4  #include <iostream>
5
6  using namespace std;
7  struct Node
8  {
9      int data;
10     Node *kiri;
11     Node *kanan;
12 };
13
14 void tambah(Node **root, int databaru)
15 {
16     if (*root == NULL)
17     {
18         Node *baru;
19         baru = new Node;
20         baru->data = databaru;
21         baru->kiri = NULL;
22         baru->kanan = NULL;
23         (*root) = baru;
24         (*root)->kiri = NULL;
25         (*root)->kanan = NULL;
26         cout << "Data bertambah";
27     }
28     else if (databaru < (*root)->data)
29         tambah(&(*root)->kiri, databaru);
30     else if (databaru > (*root)->data)
31         tambah(&(*root)->kanan, databaru);
32     else if (databaru == (*root)->data)
33         cout << "Data sudah ada";
34 }
35
36 void preOrder(Node *root)
37 {
38     if (root != NULL)
39     {
40         cout << root->data;
41         preOrder(root->kiri);
42         preOrder(root->kanan);
43     }
44 }
45
46 void inOrder(Node *root)
47 {
48     if (root != NULL)
```

```

49
50
51
52
53
54 }
55
56 void postOrder(Node *root)
57 {
58
59
60
61
62
63
64 }
65
66 int main()
67 {
68     int pil, data;
69     Node *pohon;
70     pohon = NULL;
71     do
72     {
73         system("cls");
74         cout << "1. Tambah\n";
75         cout << "2. PreOrder\n";
76         cout << "3. inOrder\n";
77         cout << "4. PostOrder\n";
78         cout << "5. Exit\n";
79         cout << "\nPilihan : ";
80         cin >> pil;
81         switch (pil)
82         {
83             case 1:
84                 cout << "\n INPUT : ";
85                 cout << "\n -----";
86                 cout << "\n Data baru : ";
87                 cin >> data;
88                 tambah(&pohon, data);
89                 break;
90             case 2:
91                 cout << "PreOrder";
92                 cout << "\n-----\n";
93                 if (pohon != NULL)
94                 {
95                     preOrder(pohon);
96

```

```

97         else
98             cout << "Masih Kosong";
99         break;
100     case 3:
101         cout << "InOrder";
102         cout << "\n-----\n";
103         if (pohon != NULL)
104         {
105             inOrder(pohon);
106         }
107         else
108             cout << "Masih Kosong";
109         break;
110     case 4:
111         cout << "PostOrder";
112         cout << "\n-----\n";
113         if (pohon != NULL)
114         {
115             postOrder(pohon);
116         }
117         else
118             cout << "Masih Kosong";
119         break;
120     case 5:
121         return 0;
122     }
123     _getch();
124 } while (pil != 5);
125 return EXIT_FAILURE;
126 }

```

A. Source Code

Tabel 1 Source Code Soal

1	#include <iostream>
2	#include <conio.h>
3	#include <stdlib.h>
4	
5	using namespace std;
6	
7	struct Node {
8	int data;
9	Node *left;
10	Node *right;
11	};
12	
13	void insert(Node **root, int newData){
14	if (*root == nullptr){
15	Node *newNode;
16	newNode = new Node;
17	
18	newNode -> data = newData;


```

19         newNode -> left = nullptr;
20         newNode -> right = nullptr;
21
22         *root = newNode;
23
24         cout << "Data has been added\n";
25     }
26     else if (newData < (*root) -> data){
27         insert(&((*root)->left), newData);
28     }
29     else if (newData > (*root) -> data){
30         insert(&((*root)->right), newData);
31     }
32     else if (newData == (*root) -> data){
33         cout << "Data is already exist\n";
34     }
35 }
36
37 void preOrder(Node *root){
38     if (root != nullptr) {
39         cout << root->data << " ";
40         preOrder(root->left);
41         preOrder(root->right);
42     }
43 }
44
45 void inOrder(Node *root){
46     if (root != nullptr) {
47         inOrder(root->left);
48         cout << root->data << " ";
49         inOrder(root->right);
50     }
51 }
52
53 void postOrder(Node *root){
54     if (root != nullptr) {
55         postOrder(root->left);
56         postOrder(root->right);
57         cout << root->data << " ";
58     }
59 }
60
61 // side quest
62 void printTree(){
63

```

```

64 };
65
66 int main(){
67     int opt, val;
68     Node *tree;
69     tree = NULL;
70
71     do {
72         system("cls");
73
74         cout << "1. Insert\n";
75         cout << "2. PreOrder\n";
76         cout << "3. InOrder\n";
77         cout << "4. PostOrder\n";
78         cout << "5. Exit\n";
79
80         cout << "\nOption: "; cin >> opt;
81
82         switch (opt) {
83
84             case 1:
85                 cout << "\n Input:";
86                 cout << "\n -----";
87                 cout << "\n New data: ";
88                 cin >> val;
89                 insert(&tree, val);
90                 break;
91
92             case 2:
93                 cout << "PreOrder Traversal\n";
94                 cout <<
95                 "=====\n";
96                 if (tree == NULL) {
97                     cout << "Tree is empty!\n";
98                 }
99                 else {
100                     preOrder(tree);
101                 }
102                 break;
103
104             case 3:
105                 cout << "InOrder Traversal\n";
106                 cout <<
107                 "=====\n";
108                 if (tree == NULL) {

```

```

107         cout << "Tree is empty!\n";
108     }
109     else {
110         inOrder(tree);
111     }
112     break;
113
114     case 4:
115         cout << "PostOrder Traversal\n";
116         cout <<
117         "===== \n";
118         if (tree == NULL) {
119             cout << "Tree is empty!\n";
120         }
121         else {
122             postOrder(tree);
123         }
124         break;
125
126     case 5:
127         return 0;
128
129     default:
130         cout << "Option is not valid! Please
131         re-enter your option";
132         break;
133     }
134     getch();
135     } while(opt != 5);
136     return 0;
137 }

```

B. Output Program

```
PROBLEMS OUTPUT DEBUG CON
○ 1. Insert
  2. PreOrder
  3. InOrder
  4. PostOrder
  5. Exit

Option: 2
PreOrder Traversal
=====
Tree is empty!
█
```

Gambar 1 Tampilan PreOrder Ketika Pohon Kosong

```
PROBLEMS OUTPUT DEBUG CO
○ 1. Insert
  2. PreOrder
  3. InOrder
  4. PostOrder
  5. Exit

Option: 3
InOrder Traversal
=====
Tree is empty!
█
```

Gambar 2 Tampilan InOrder Ketika Pohon Kosong

```
PROBLEMS OUTPUT DEBUG CON
○ 1. Insert
  2. PreOrder
  3. InOrder
  4. PostOrder
  5. Exit

Option: 4
PostOrder Traversal
=====
Tree is empty!
█
```

Gambar 3 Tampilan PostOrder Ketika Pohon Kosong

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
○ 1. Insert
  2. PreOrder
  3. InOrder
  4. PostOrder
  5. Exit

Option: 9
Option is not valid! Please re-enter your option█
```

Gambar 4 Tampilan Ketika Pilihan Tidak Valid

```
PROBLEMS OUTPUT DEBUG CONSOLE
○ 1. Insert
  2. PreOrder
  3. InOrder
  4. PostOrder
  5. Exit

Option: 1

Input:
-----
New data: 25
Data has been added
█
```

Gambar 5 Input Data Pertama

```
PROBLEMS OUTPUT DEBUG CONSOLE
○ 1. Insert
  2. PreOrder
  3. InOrder
  4. PostOrder
  5. Exit

Option: 1

Input:
-----
New data: 8
Data has been added
█
```

Gambar 6 Input Data Kedua

```
PROBLEMS OUTPUT DEBUG CONSOLE
○ 1. Insert
  2. PreOrder
  3. InOrder
  4. PostOrder
  5. Exit

Option: 1

Input:
-----
New data: 20
Data has been added
█
```

Gambar 7 Input Data Ketiga

```
PROBLEMS OUTPUT DEBUG CONSOLE
○ 1. Insert
  2. PreOrder
  3. InOrder
  4. PostOrder
  5. Exit

Option: 1

Input:
-----
New data: 24
Data has been added
█
```

Gambar 8 Input Data Keempat

```
PROBLEMS OUTPUT DEBUG CONSOLE
○ 1. Insert
  2. PreOrder
  3. InOrder
  4. PostOrder
  5. Exit

Option: 1

Input:
-----
New data: 19
Data has been added
█
```

Gambar 9 Input Data Kelima

```
PROBLEMS OUTPUT DEBUG CONSOLE
○ 1. Insert
  2. PreOrder
  3. InOrder
  4. PostOrder
  5. Exit

Option: 1

Input:
-----
New data: 2
Data has been added
█
```

Gambar 10 Input Data Keenam

```
PROBLEMS OUTPUT DEBUG CONSOLE
○ 1. Insert
  2. PreOrder
  3. InOrder
  4. PostOrder
  5. Exit

Option: 1

Input:
-----
New data: 20
Data is already exist
█
```

Gambar 11 Input Data Ketujuh Tetapi Data Sudah Ada

```
PROBLEMS OUTPUT DEBUG CONSOLE
○ 1. Insert
  2. PreOrder
  3. InOrder
  4. PostOrder
  5. Exit

Option: 1

Input:
-----
New data: 6
Data has been added
█
```

Gambar 12 Input Data Kedelapan

```
PROBLEMS OUTPUT DEBUG CON
○ 1. Insert
  2. PreOrder
  3. InOrder
  4. PostOrder
  5. Exit

Option: 2
PreOrder Traversal
=====
25 8 2 6 20 19 24
```

Gambar 13 PreOrder Traversal

```
PROBLEMS OUTPUT DEBUG CONS
○ 1. Insert
  2. PreOrder
  3. InOrder
  4. PostOrder
  5. Exit

Option: 3
InOrder Traversal
=====
2 6 8 19 20 24 25
```

Gambar 14 InOrder Traversal

```
PROBLEMS OUTPUT DEBUG CO
○ 1. Insert
  2. PreOrder
  3. InOrder
  4. PostOrder
  5. Exit

Option: 4
PostOrder Traversal
=====
6 2 19 24 20 8 25
```

Gambar 15 PostOrder Traversal

```
PROBLEMS OUTPUT DEBUG CONSOLE
1. Insert
2. PreOrder
● 3. InOrder
4. PostOrder
5. Exit

Option: 5
○ PS C:\Users\ACER\Kuliah TI\Semes
```

Gambar 16 Exit

C. Pembahasan

Struktur Kode Umum

Header Files (`#include`) adalah pustaka yang berisi kode-kode siap pakai.

- `<iostream>` digunakan untuk input dan output standar (seperti `cout` untuk mencetak ke konsol dan `cin` untuk membaca input dari konsol).
- `<conio.h>` adalah library yang menyediakan fungsi-fungsi konsol input/output, seperti `getch()`, yang digunakan untuk mendapatkan karakter dari keyboard tanpa perlu menekan enter.
- `<stdlib.h>` adalah library standar yang menyediakan fungsi-fungsi umum, termasuk `system()` yang digunakan di sini untuk `system("cls")`.

`using namespace std;` adalah pernyataan yang memungkinkan agar bisa menggunakan elemen-elemen dari namespace `std` (standard) secara langsung, seperti `cout`, `cin`, `vector`, dan `sort`, tanpa perlu menambahkan `std::` di depannya.

Struktur Node

Kode ini mendefinisikan sebuah struktur (struct) bernama `Node`. Setiap `Node` adalah unit dasar dari Binary Search Tree (BST) dan memiliki tiga bagian:

- `int data` adalah tempat untuk menyimpan nilai sebenarnya dari node tersebut. Dalam kasus ini, nilainya adalah bilangan bulat.
- `Node *left` adalah pointer ke node anak di sebelah kiri. Jika tidak ada anak kiri, nilainya akan `nullptr`.
- `Node *right` adalah pointer ke node anak di sebelah kanan. Jika tidak ada anak kanan, nilainya akan `nullptr`.

Fungsi insert()

Fungsi `insert()` berfungsi untuk menambahkan data baru ke dalam Binary Search Tree (BST) secara berurutan, menjaga property BST (nilai lebih kecil di kiri, nilai lebih besar di kanan).

Parameter:

- `Node **root` adalah pointer ke pointer ke node root saat ini. Menggunakan pointer ke pointer memungkinkan fungsi untuk mengubah pointer root yang sebenarnya di fungsi `main()` jika pohon masih kosong, atau untuk mengubah pointer `left` atau `right` dari node induk ketika menambahkan node baru.
- `int newData` adalah nilai bilangan bulat yang akan ditambahkan ke dalam pohon.

Cara Kerja:

1. Kasus Dasar (Pohon Kosong atau Posisi Ditemukan)

- `if (*root == nullptr)` artinya jika pointer root yang sedang dilihat saat ini menunjuk ke `nullptr`, berarti tempat kosong di pohon untuk menyisipkan data baru telah ditemukan.
- `Node *newNode; newNode = new Node;` artinya sebuah node baru dibuat di memori (heap) menggunakan `new Node`;
- `newNode -> data = newData;` artinya nilai `newData` disalin ke anggota data dari node baru.
- `newNode -> left = nullptr; newNode -> right = nullptr;` karena ini adalah node yang baru ditambahkan di bagian paling bawah pohon, anak kiri dan kanannya diatur ke `nullptr`.
- `*root = newNode` artinya pointer saat ini (yang sebelumnya `nullptr`) sekarang diarahkan ke `newNode` yang baru saja dibuat. Ini menghubungkan node baru ke struktur pohon.
- `cout << "Data has been added\n"` merupakan pesan konfirmasi yang akan ditampilkan.

2. Menjelajah ke Kiri

- `else if (newData < (*root) -> data)` artinya jika `newData` lebih kecil dari data di node root saat ini, berarti node baru harus ditempatkan di sub-pohon kiri.
- `insert(&((*root)->left), newData)` artinya fungsi `insert()` dipanggil secara rekursif dengan meneruskan alamat dari pointer anak kiri node saat ini (`&((*root)->left)`). Ini akan terus mencari posisi yang tepat di sisi kiri pohon.

3. Menjelajah ke Kanan

`else if (newData > (*root) -> data)` artinya jika `newData` lebih besar dari data di node root saat ini, berarti node baru harus ditempatkan di sub-pohon kanan.

`insert(&((*root)->right), newData)` artinya fungsi `insert()` dipanggil secara rekursif dengan meneruskan alamat dari pointer anak kanan node saat ini (`&((*root)->right)`). Ini akan terus mencari posisi yang tepat di sisi kanan pohon.

4. Data Sudah Ada

else if (newData == (*root) -> data artinya jika newData sama dengan data di node root saat ini, berarti data tersebut sudah ada di dalam pohon. Dalam implementasi BST ini, duplikasi tidak diizinkan.

Fungsi preOrder()

Fungsi preOrder() melakukan traversal Pre-Order. Urutannya adalah Root -> Kiri -> Kanan.

Parameter: Node *root adalah pointer ke node root (atau sub-root) saat ini yang sedang dikunjungi.

Cara Kerja:

1. if (root != nullptr) artinya fungsi akan bekerja hanya jika node saat ini tidak kosong.
2. cout << root->data << “ “; maka data dari node saat ini akan dicetak terlebih dahulu.
3. preOrder(root->left), kemudian fungsi memanggil dirinya sendiri secara rekursif untuk mengunjungi seluruh sub-pohon kiri.
4. preOrder(root->right), fungsi memanggil dirinya sendiri secara rekursif untuk mengunjungi seluruh sub-pohon kanan.

Fungsi inOrder()

Fungsi inOrder() melakukan traversal In-Order. Urutannya adalah Kiri -> Root -> Kanan. Traversal ini sangat penting untuk BST karena akan mencetak semua elemen dalam urutan menaik.

Parameter: Node *root adalah pointer ke node root (atau sub-root) saat ini yang sedang dikunjungi.

Cara Kerja:

1. `if (root != nullptr)` artinya fungsi akan bekerja hanya jika node saat ini tidak kosong.
2. `inOrder(root->left)` artinya fungsi memanggil dirinya sendiri secara rekursif untuk mengunjungi seluruh sub-pohon kiri terlebih dahulu.
3. `cout << root->data << " ";` artinya setelah semua node di kiri selesai dikunjungi, data dari node saat ini dicetak.
4. `inOrder(root->right)` artinya fungsi memanggil dirinya sendiri secara rekursif untuk mengunjungi seluruh sub-pohon kanan.

Fungsi postOrder()

Fungsi `postOrder()` melakukan traversal Post-Order. Urutannya adalah Kiri -> Kanan -> Root.

Parameter: Node `*root` adalah pointer ke node root (atau sub-root) saat ini yang sedang dikunjungi.

Cara Kerja:

1. `if (root != nullptr)` artinya fungsi hanya akan bekerja jika node saat ini tidak kosong.
2. `postOrder(root->left)` artinya fungsi memanggil dirinya sendiri secara rekursif untuk mengunjungi seluruh sub-pohon kiri terlebih dahulu.
3. `postOrder(root->right)` artinya fungsi memanggil dirinya sendiri secara rekursif untuk mengunjungi seluruh sub-pohon kanan.
4. `cout << root->data << " ";` artinya data dari node saat ini dicetak.

Fungsi main()

Fungsi `main()` adalah titik awal eksekusi program. Fungsi ini akan bertanggung jawab untuk mengatur dan mengelola interaksi dengan user melalui menu, serta memanggil fungsi-fungsi BST yang telah didefinisikan sebelumnya.

Alur kerja dalam `main()`:

1. Inisialisasi Variabel

- `int opt, val` mendeklarasikan `opt` untuk menyimpan pilihan menu user, dan `val` untuk menyimpan nilai data yang akan dimasukkan ke dalam BST.
- `Node *tree; tree = NULL` mendeklarasikan `tree` sebagai pointer ke node akar (root) dari BST. Awalnya diinisialisasi ke `NULL`, menandakan bahwa pohon masih kosong.

2. Loop Utama Program (do-while)

`do {...} while(opt != 5)` artinya program akan terus menjalankan blok kode di dalam `do` selama pilihan user (`opt`) bukan 5 (exit). Ini memastikan menu selalu ditampilkan dan program terus berjalan hingga user memutuskan untuk keluar.

3. Tampilan Menu & Input Pilihan

- `system("cls")` digunakan untuk membersihkan terminal setiap kali loop berjalan, membuat tampilan menu selalu rapi.
- Serangkaian pernyataan `cout` menampilkan opsi-opsi menu kepada user: Insert, PreOrder, InOrder, PostOrder, dan Exit.
- `cout << "\nOption: "; cin >> opt;` meminta user untuk memasukkan pilihan dan menyimpannya ke variabel `opt`.

4. Memproses Pilihan User (switch)

- `switch (opt)` adalah struktur `switch-case` digunakan untuk menjalankan blok kode yang sesuai dengan pilihan `opt` dari user.
- `case 1 (Insert)` digunakan untuk mencetak prompt input.
 - `cin >> val` digunakan untuk membaca nilai integer yang akan dimasukkan dari user.
 - `insert(&tree, val)` memanggil fungsi `insert()` untuk menambahkan `val` ke pohon. `&tree` dilewatkan untuk memungkinkan `insert` memodifikasi pointer `tree` itu sendiri jika itu adalah node pertama, atau pointer anak dari node yang ada.
 - `break` mengakhiri eksekusi blok `switch` untuk kasus ini.

- case 2 (PreOrder), case 3 (InOrder), case 4 (PostOrder) digunakan mencetak judul traversal yang relevan.
 - if (tree == NULL) {cout << "Tree is empty!\n": } untuk memeriksa apakah pohon kosong.
 - else {preorder(tree); } (atau inOrder(tree) atau postOrder(tree)) artinya: jika pohon tidak kosong, fungsi traversal yang sesuai dipanggil dengan tree sebagai argument, yang akan mencetak data node dalam urutan yang ditentukan.
 - break mengakhiri eksekusi blok switch.
- case 5 (Exit)

return 0 ntuk segera mengakhiri program dan mengembalikan 0 sebagai kode keluar, yang menunjukkan keberhasilan eksekusi..
- default

Jika user memasukkan pilihan yang tidak valid (selain1-5), pesan kesalahan "Option is not valid" ditampilkan.

5. Jeda dan Loop Kembali

getch() setelah setiap operasi (kecuali keluar), program akan menunggu user menekan tombol apapun.

6. Mengakhiri Program

return 0 dieksekusi jika loop do-while berakhir karena opt menjadi 5.