

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



TREE

Oleh:

Indra Suryadilaga NIM. 2410817310014

**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 6: Tree ini disusun sebagai syarat lulus mata kuliah Praktikum Algoritma & Struktur Data. Laporan Praktikum ini dikerjakan oleh:

Nama Praktikan : Indra Suryadilaga
NIM : 2410817310014

Menyetujui,
Asisten Praktikum

Mengetahui,
Dosen Penanggung Jawab Praktikum

Harry Pratama Yunus
NIM. 2310817210010

Andreyan Rizky Baskara, S.Kom., M.Kom
NIP. 199307032019031011

DAFTAR ISI

LEMBAR PENGESAHAN	2
DAFTAR ISI.....	3
DAFTAR GAMBAR	4
DAFTAR TABEL.....	5
SOAL 1	6
A. Source Code	8
B. Output Program.....	12
C. Pembahasan.....	13
GITHUB.....	17

DAFTAR GAMBAR

Gambar 1. Soal No. 1 Source Code Tree	7
Gambar 2. Output Opsi 1 Insert	12
Gambar 3. Output Opsi 2 PreOrder.....	12
Gambar 4. Output Opsi 3 InOrder.....	13
Gambar 5. Output Opsi 4 PostOrder	13

DAFTAR TABEL

Table 1. Source Code Tree.....	8
--------------------------------	---

SOAL 1

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 }

```

Gambar 1. Soal No. 1 Source Code Tree

A. Source Code

Table 1. Source Code Tree

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 != NULL){
39         cout << root->data << " ";
40         preOrder(root->left);
41         preOrder(root->right);
42     }
43 }
44
45 void inOrder(Node *root){
46     if (root != NULL){
47         inOrder(root->left);
48         cout << root->data << " ";
49         inOrder(root->right);
50     }
51 }
52
53 void postOrder(Node *root){
54     if (root != NULL){
55         postOrder(root->left);
56         postOrder(root->right);
57         cout << root->data << " ";
58     }
59 }
60
61 void destroyTree(Node *root) {
62     if (root != nullptr) {
63         destroyTree(root->left);
64         destroyTree(root->right);
65         delete root;
66     }
67 }
68
69 int main(){
70     int opt, val;

```

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

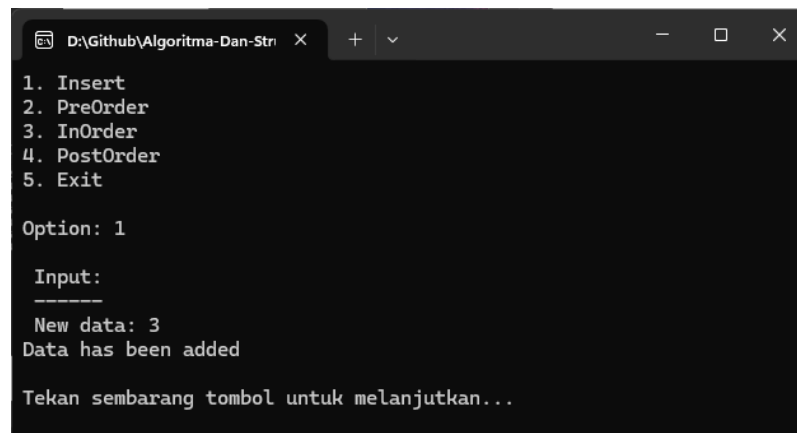
```

107         cout << "InOrder Traversal\n";
108         cout << "=====\n";
109         if (tree == NULL) {
110             cout << "Tree is empty!\n";
111         }
112         else {
113             inOrder(tree);
114         }
115         break;
116
117     case 4:
118         cout << "PostOrder Traversal\n";
119         cout << "=====\n";
120         if (tree == NULL) {
121             cout << "Tree is empty!\n";
122         }
123         else {
124             postOrder(tree);
125         }
126         break;
127
128     case 5:
129         cout << "\nExiting program and cleaning up
memory...\n";
130         destroyTree(tree);
131         tree = nullptr;
132         cout << "Memory cleaned up. Goodbye!\n";
133         break;
134
135     default:
136         cout << "Option is not valid! Please re-enter
your option";
137         break;
138     }
139
140     if (opt != 5) {

```

141	cout << "\nTekan sembarang tombol untuk melanjutkan...";
142	getch();
143	}
144	
145	} while(opt != 5);
146	return 0;
147	}

B. Output Program



```

D:\Github\Algoritma-Dan-Stri x + v - □ ×
1. Insert
2. PreOrder
3. InOrder
4. PostOrder
5. Exit

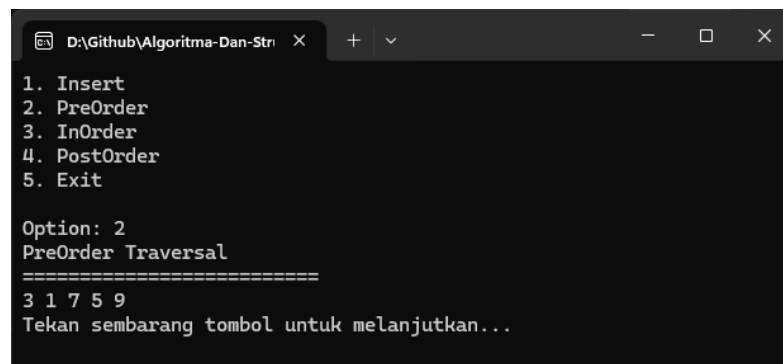
Option: 1

Input:
-----
New data: 3
Data has been added

Tekan sembarang tombol untuk melanjutkan...

```

Gambar 2. Output Opsi 1 Insert



```

D:\Github\Algoritma-Dan-Stri x + v - □ ×
1. Insert
2. PreOrder
3. InOrder
4. PostOrder
5. Exit

Option: 2
PreOrder Traversal
=====
3 1 7 5 9
Tekan sembarang tombol untuk melanjutkan...

```

Gambar 3. Output Opsi 2 PreOrder

```
D:\Github\Algoritma-Dan-Stri x + v - □ x
1. Insert
2. PreOrder
3. InOrder
4. PostOrder
5. Exit

Option: 3
InOrder Traversal
=====
1 3 5 7 9
Tekan sembarang tombol untuk melanjutkan...
```

Gambar 4. Output Opsi 3 InOrder

```
D:\Github\Algoritma-Dan-Stri x + v - □ x
1. Insert
2. PreOrder
3. InOrder
4. PostOrder
5. Exit

Option: 4
PostOrder Traversal
=====
1 5 9 7 3
Tekan sembarang tombol untuk melanjutkan...
```

Gambar 5. Output Opsi 4 PostOrder

C. Pembahasan

Alur Program Binary Search Tree

Ketika program dijalankan, sistem akan menampilkan menu utama dengan 5 pilihan yang tersedia menggunakan do-while loop yang terus berjalan hingga user memilih opsi 5 (Exit). Setiap kali user memilih opsi 1-4, program akan melakukan proses sesuai dengan pilihan yang dipilih, kemudian layar akan dibersihkan menggunakan `system("cls")` dan menu utama akan ditampilkan kembali.

Untuk opsi 1 (Insert), program akan meminta user untuk memasukkan data baru yang ingin ditambahkan ke dalam tree. Program menggunakan fungsi `insert()` yang bekerja secara rekursif untuk menemukan posisi yang tepat sesuai dengan aturan BST (Binary Search Tree). Data yang lebih kecil akan ditempatkan di subtree kiri, sedangkan data yang lebih besar akan ditempatkan di subtree kanan. Jika data yang

dimasukkan sudah ada dalam tree, program akan menampilkan pesan "Data is already exist".

Untuk opsi 2, 3, dan 4 (PreOrder, InOrder, PostOrder), program akan melakukan traversal atau penelusuran tree menggunakan tiga metode yang berbeda. Sebelum melakukan traversal, program akan memeriksa apakah tree kosong. Jika kosong, program akan menampilkan pesan "Tree is empty!". Jika tidak kosong, program akan menampilkan hasil traversal sesuai dengan metode yang dipilih.

Untuk opsi 5, program akan melakukan cleanup memory dengan memanggil fungsi `destroyTree()` untuk menghapus semua node yang telah dialokasikan secara dinamis, kemudian mengatur pointer tree menjadi `nullptr` sebelum mengakhiri program. Hal ini dilakukan untuk mencegah memory leak.

Pada akhir setiap operasi (kecuali exit), program akan menampilkan pesan "Tekan sembarang tombol untuk melanjutkan..." dan menunggu input menggunakan `getch()`. Setelah user menekan tombol apapun, layar akan dibersihkan dan menu utama akan ditampilkan kembali. Loop ini akan terus berlanjut hingga user memilih opsi 5 untuk keluar dari program.

Analisis Struktur Data Binary Search Tree

Binary Search Tree (BST) adalah struktur data tree berbasis node di mana setiap node memiliki maksimal dua child (anak), yaitu left child dan right child. Struktur Node dalam implementasi ini terdiri dari tiga komponen: data bertipe integer yang menyimpan nilai, dan dua pointer (left dan right) yang menunjuk ke child nodes.

Aturan fundamental BST adalah bahwa untuk setiap node, semua nilai di subtree kiri harus lebih kecil dari nilai node tersebut, dan semua nilai di subtree kanan harus lebih besar dari nilai node tersebut. Aturan ini memungkinkan operasi pencarian, penyisipan, dan penghapusan dapat dilakukan dengan efisien.

Program ini menggunakan implementasi pointer-to-pointer (double pointer) pada fungsi insert() untuk memungkinkan modifikasi langsung terhadap pointer root. Hal ini memungkinkan fungsi untuk mengubah nilai pointer yang diteruskan dari fungsi pemanggil, yang sangat penting ketika menambahkan node pertama ke tree yang kosong.

Analisis Algoritma Insert

Algoritma insert() bekerja secara rekursif dengan menggunakan prinsip divide and conquer. Proses dimulai dengan memeriksa apakah root tree kosong (nullptr). Jika kosong, program akan membuat node baru menggunakan dynamic memory allocation dengan operator new, mengisi data node tersebut, dan mengatur kedua pointer child menjadi nullptr.

Jika tree tidak kosong, program akan membandingkan data baru dengan data pada node saat ini. Jika data baru lebih kecil, fungsi akan memanggil dirinya sendiri secara rekursif untuk subtree kiri. Jika data baru lebih besar, fungsi akan memanggil dirinya sendiri secara rekursif untuk subtree kanan. Jika data baru sama dengan data yang sudah ada, program akan menampilkan pesan bahwa data sudah exist dan tidak melakukan penyisipan.

Kelebihan dari algoritma insert ini adalah dapat mempertahankan properti BST secara otomatis dan memiliki implementasi yang elegant menggunakan rekursi. Time complexity untuk operasi insert pada average case adalah $O(\log n)$, di mana n adalah jumlah node dalam tree. Namun pada worst case (ketika tree menjadi skewed atau tidak seimbang), time complexity bisa mencapai $O(n)$.

Space complexity untuk operasi insert adalah $O(h)$ di mana h adalah height dari tree, karena menggunakan recursive call stack. Pada balanced tree, space complexity adalah $O(\log n)$, namun pada skewed tree bisa mencapai $O(n)$.

Analisis Algoritma Traversal

Program implementasi ini menyediakan tiga metode traversal yang berbeda: PreOrder, InOrder, dan PostOrder. Ketiga algoritma ini menggunakan pendekatan rekursif untuk mengunjungi setiap node dalam tree dengan urutan yang berbeda.

PreOrder Traversal (Root-Left-Right)

Algoritma PreOrder mengunjungi node dalam urutan: root, kemudian subtree kiri, lalu subtree kanan. Proses dimulai dengan mencetak data node saat ini, kemudian memanggil `preOrder()` secara rekursif untuk left child, dan terakhir memanggil `preOrder()` secara rekursif untuk right child. Traversal ini berguna untuk membuat copy dari tree atau untuk mengevaluasi expression tree.

InOrder Traversal (Left-Root-Right)

Algoritma InOrder mengunjungi node dalam urutan: subtree kiri, root, kemudian subtree kanan. Proses dimulai dengan memanggil `inOrder()` secara rekursif untuk left child, kemudian mencetak data node saat ini, dan terakhir memanggil `inOrder()` secara rekursif untuk right child. Keunggulan utama InOrder traversal pada BST adalah menghasilkan output yang terurut secara ascending, sehingga sangat berguna untuk mendapatkan data dalam urutan yang sorted.

PostOrder Traversal (Left-Right-Root)

Algoritma PostOrder mengunjungi node dalam urutan: subtree kiri, subtree kanan, kemudian root. Proses dimulai dengan memanggil `postOrder()` secara rekursif untuk left child, kemudian memanggil `postOrder()` secara rekursif untuk right child, dan terakhir mencetak data node saat ini. Traversal ini sangat berguna untuk operasi penghapusan tree karena memproses children sebelum parent, sehingga aman untuk menghapus node tanpa kehilangan referensi ke children.

Time complexity untuk semua metode traversal adalah $O(n)$ karena setiap node harus dikunjungi tepat satu kali. Space complexity adalah $O(h)$ dimana h adalah height tree, karena menggunakan recursive call stack yang dalam worst case bisa mencapai height maksimum tree.

GITHUB

<https://github.com/IndraSuryadilaga/Algoritma-Dan-Struktur-Data>