

**Laporan Praktikum**

**Algoritma dan Struktur Data**

Ganjil 2023/2024

Program Studi Teknik Informatika

Institut Teknologi Sumatera

|  |  |
| --- | --- |
| **Modul :** | **Binary Tree** |
| **Nama :** | **Rayhan Fadel Irwanto** |
| **NIM :** | **122140236** |
| **Kelas (Kelas Asal) :** | **ASD RA** |

Instruksi sederhana :

* Disarankan kepada **Praktikan Algoritma Struktur Data** untuk mengeditnya menggunakan Google Docs agar tidak berantakan dan rapi,
* Silahkan mengganti **Nama Modul** baik yang ada pada **Cover** dan **Header** sesuai dengan materi praktikum,
* Gunakan text styling seperti **Heading 1**, **Normal Text** yang telah terformat / Text Style lainnya yang digunakan untuk menjaga estetika laporan,
* Gunakan [**Syntax Highlighter**](https://highlight.hohli.com/index.php)untuk merapikan kode yang sudah Praktikan buat ke dalam Laporan Praktikum.

**Materi Praktikum**

Binary Tree

**Link repl.it Source Code**

<https://replit.com/@RayhanFadel/tree-TM5#main.cpp>

**Source Code**

Gunakan [**Syntax Highlighter**](https://highlight.hohli.com/index.php)untuk merapikan Source Code yang dipindahkan dari text editor anda ke dokumen ini.

1. #include <iostream>
2. using namespace std;
4. struct Tree {
5. **int info;**
7. Tree \*left;
8. Tree \*right;
9. Tree \*parent;
10. **};**
12. Tree \*Initialize(int value) {
13. Tree \*newNode = new Tree;
15. **newNode->info = value;**
16. newNode->left = nullptr;
17. newNode->right = nullptr;
18. newNode->parent = nullptr;
20. **return newNode;**
21. }
23. bool IsTreeEmpty(Tree \*P) { return P == nullptr; }
25. **void InsertLeft(Tree \*P, int value) {**
26. Tree \*newNode = Initialize(value);
28. if (P->left == nullptr) {
29. newNode->parent = P;
30. **P->left = newNode;**
31. cout << "Success insert left node with value " << value << " with "
32. << P->info << " as parrent**\n**";
33. } else {
34. cout << "Tree left was declared!";
35. **}**
36. }
38. void InsertRight(Tree \*P, int value) {
39. Tree \*newNode = Initialize(value);
40. **if (P->right == nullptr) {**
41. newNode->parent = P;
42. P->right = newNode;
43. cout << "Success insert right node with value " << value << " with "
44. << P->info << " as parrent**\n**";
45. **} else {**
46. cout << "Tree right was declared!";
47. }
48. }
50. **void PreOrder(Tree \*P) {**
51. if (IsTreeEmpty(P)) {
53. } else {
54. cout << P->info << " ";
55. **PreOrder(P->left);**
56. PreOrder(P->right);
57. }
58. }
60. **void InOrder(Tree \*P) {**
61. if (IsTreeEmpty(P)) {
63. } else {
64. InOrder(P->left);
65. **cout << P->info << " ";**
66. InOrder(P->right);
67. }
68. }
70. **void PostOrder(Tree \*P) {**
71. if (IsTreeEmpty(P)) {
73. } else {
74. PostOrder(P->left);
75. **PostOrder(P->right);**
76. cout << P->info << " ";
77. }
78. }
80. **void display(Tree \*P, int indent = 0) {**
81. cout << "<>Pre Order : ";
82. PreOrder(P);
83. cout << endl;
84. cout << "<>In Order : ";
85. **InOrder(P);**
86. cout << endl;
87. cout << "<>Post Order : ";
88. PostOrder(P);
89. cout << endl << endl;
90. **}**
92. void deleteRightLeaf(Tree \*P) {
93. if (P->right == nullptr && P->left == nullptr) {
94. if (P->parent->right != nullptr) {
95. **P->parent->right = nullptr;**
96. return;
97. }
99. if (P->parent->left != nullptr) {
100. **P->parent->left = nullptr;**
101. }
102. }
104. if (P->right != nullptr && P->left == nullptr) {
105. **deleteRightLeaf(P->right);**
106. } else if (P->right == nullptr && P->left != nullptr) {
107. deleteRightLeaf(P->left);
108. } else if (P->right != nullptr && P->left != nullptr) {
109. deleteRightLeaf(P->right);
110. **}**
111. }
113. int main() {
114. Tree \*P = Initialize(1);
115. **InsertLeft(P, 2);**
116. InsertRight(P, 3);
118. InsertLeft(P->left, 4);
119. InsertRight(P->left, 5);
121. InsertLeft(P->right, 6);
122. InsertRight(P->right, 7);
124. InsertLeft(P->left->left, 8);
126. cout << endl << "Display tree : " << endl;
127. display(P);
129. cout << "Processing delete tree: " << endl;
130. **while (!IsTreeEmpty(P)) {**
131. deleteRightLeaf(P);
132. display(P);
133. }
135. **return 0;**
136. }

**Dokumentasi Hasil Running**

|  |
| --- |
|  |
| **Gambar 1.** Output Code Binary Tree Tugas Minggu 5.  Program di atas adalah implementasi pohon biner dalam C++. Dalam struktur data `Tree`, setiap node memiliki informasi (info), pointer ke anak kiri (left), anak kanan (right), dan induk (parent). Fungsi `Initialize` digunakan untuk membuat dan menginisialisasi node baru dengan nilai tertentu. Fungsi `IsTreeEmpty` mengembalikan true jika pohon kosong. `InsertLeft` dan `InsertRight` digunakan untuk menyisipkan node baru di sebelah kiri atau kanan dari node P. Fungsi `PreOrder`, `InOrder`, dan `PostOrder` melakukan penelusuran pohon secara pre-order, in-order, dan post-order, masing-masing. Fungsi `display` menampilkan hasil dari ketiga jenis penelusuran. Selanjutnya, fungsi `deleteRightLeaf` menghapus semua node daun di sebelah kanan pohon, dimulai dari node P. Dalam fungsi `main`, sebuah pohon dibangun dengan beberapa node, kemudian dilakukan penghapusan node daun di sebelah kanan secara berulang hingga pohon kosong, dan setiap langkahnya ditampilkan. |

|  |
| --- |
|  |
|  |