

**Laporan Praktikum**

**Algoritma dan Struktur Data**

Ganjil 2023/2024

Program Studi Teknik Informatika

Institut Teknologi Sumatera

|  |  |
| --- | --- |
| **Modul :** | **Binary Tree** |
| **Nama :** | **Muhammad Yusuf** |
| **NIM :** | **122140193** |
| **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/@muhammad1221401/Muhammad-Yusuf-Prak-ASD-RA122140193#minggu5/tugas.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. #include <iomanip>
3. using namespace std;
5. **struct Tree{**
6. int info;
8. Tree \*left;
9. Tree \*right;
10. **Tree \*parent;**
11. };
13. Tree \*Initialize(int value){
14. Tree \*newNode = new Tree;
16. newNode->info = value;
17. newNode->left = nullptr;
18. newNode->right = nullptr;
19. newNode->parent = nullptr;
21. return newNode;
22. }
24. bool IsTreeEmpty(Tree \*P) { return P == nullptr; }
26. void InsertLeft(Tree \*P, int value){
27. Tree \*newNode = Initialize(value);
29. if (P->left == nullptr){
30. **newNode->parent = P;**
31. P->left = newNode;
32. cout << "Success insert left node with value " << value << " with " << 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 " << P->info << " as parrent**\n**";
44. }
45. **else{**
46. cout << "Tree right was declared!";
47. }
48. }
50. **void PreOrder(Tree \*P){**
51. if (IsTreeEmpty(P)){
52. }else{
53. cout << P->info << " ";
54. PreOrder(P->left);
55. **PreOrder(P->right);**
56. }
57. }
59. void InOrder(Tree \*P){
60. **if (IsTreeEmpty(P)){**
61. }else{
62. InOrder(P->left);
63. cout << P->info << " ";
64. InOrder(P->right);
65. **}**
66. }
68. void PostOrder(Tree \*P){
69. if (IsTreeEmpty(P)){
70. **}else{**
71. PostOrder(P->left);
72. PostOrder(P->right);
73. cout << P->info << " ";
74. }
75. **}**
77. void display(Tree \*P){
78. cout << "<>Pre Order : ";
79. PreOrder(P);
80. **cout << endl;**
81. cout << "<>In Order : ";
82. InOrder(P);
83. cout << endl;
84. cout << "<>Post Order : ";
85. **PostOrder(P);**
86. cout << endl << endl;
87. }
89. void displayIndent(Tree \*P, int level=0){
90. **if (P != nullptr){**
91. displayIndent(P->right, level + 1);
92. cout << setw(level \* 3) << " " << P->info << endl;
93. displayIndent(P->left, level + 1);
94. }
95. **}**
97. void deleteRightLeaf(Tree \*P){
98. if (P->right == nullptr && P->left == nullptr){
99. if (P->parent->right != nullptr){
100. **P->parent->right = nullptr;**
101. return;
102. }
104. if (P->parent->left != nullptr){
105. **P->parent->left = nullptr;**
106. }
107. }
109. if (P->right != nullptr && P->left == nullptr){
110. **deleteRightLeaf(P->right);**
111. }
112. else if (P->right == nullptr && P->left != nullptr){
113. deleteRightLeaf(P->left);
114. }
115. **else if (P->right != nullptr && P->left != nullptr){**
116. deleteRightLeaf(P->right);
117. }
118. }
120. **int main(){**
121. Tree \*P = Initialize(1);
122. InsertLeft(P, 2);
123. InsertRight(P, 3);
125. **InsertLeft(P->left, 4);**
126. InsertRight(P->left, 5);
128. InsertLeft(P->right, 6);
129. InsertRight(P->right, 7);
131. InsertLeft(P->left->left, 8);
133. cout << endl << "Display tree : " << endl;
134. display(P);
136. cout << "Display tree with indent : " << endl;
137. displayIndent(P);
139. cout << "Processing delete tree...." << endl;
140. **while (!IsTreeEmpty(P)){**
141. deleteRightLeaf(P);
142. displayIndent(P);
143. display(P);
144. }
146. return 0;
147. }

**Dokumentasi Hasil Running**

|  |
| --- |
|  |
| **Gambar 1.** Output Code Binary Tree Tugas Minggu 5.  Program di atas adalah implementasi pohon biner dalam C++. Setiap node dalam pohon memiliki informasi (info), pointer ke anak kiri (left), anak kanan (right), dan pointer ke induk (parent). Fungsi `Initialize` digunakan untuk membuat dan menginisialisasi node baru dengan nilai tertentu. Fungsi `IsTreeEmpty` mengembalikan true jika pohon kosong. Fungsi `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 penelusuran pre-order, in-order, dan post-order. Fungsi `displayIndent` menampilkan pohon dengan indentasi sesuai tingkatan node. Fungsi `deleteRightLeaf` menghapus semua node daun di sebelah kanan pohon, dimulai dari node P. Dalam fungsi `main`, sebuah pohon dibangun dengan beberapa node, dan hasil penelusuran serta tampilan dengan indentasi ditampilkan. Selanjutnya, program melakukan penghapusan node daun di sebelah kanan secara berulang hingga pohon kosong, dan setiap langkahnya ditampilkan. |

|  |
| --- |
|  |
|  |