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



**Tree (Pohon)**

|  |  |
| --- | --- |
| **Oleh:** | |
| **Rizki Adhitiya Maulana** | **NIM. 2410817110014** |

**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 : Rizki Adhitiya Maulana

NIM : 2410817110014

|  |  |
| --- | --- |
| Menyetujui,  Asisten Praktikum  Muhammad Fauzan Ahsani  NIM. 2310817310009 | Mengetahui,  Dosen Penanggung Jawab Praktikum  Muti’a Maulida, S.Kom., M.TI.  NIP. 198810272019032013 |

# DAFTAR ISI

[LEMBAR PENGESAHAN i](#_Toc200486964)

[DAFTAR ISI ii](#_Toc200486965)

[DAFTAR TABEL iii](#_Toc200486966)

[DAFTAR GAMBAR iv](#_Toc200486967)

[SOAL 1 1](#_Toc200486968)

[A Source Code 4](#_Toc200486969)

[B Output Program 10](#_Toc200486970)

[C Pembahasan 12](#_Toc200486971)

[TAUTAN GITHUB 18](#_Toc200486972)

# DAFTAR TABEL

[Tabel 1 Source Code 4](#_Toc200486999)

# DAFTAR GAMBAR

[Gambar 1 Soal 1 Modul 7 3](#_Toc200523682)

[Gambar 2 Tampilan Menu Tree 10](#_Toc200523684)

[Gambar 3 Tampilan Menu Insert Tree 10](#_Toc200523685)

[Gambar 4 Tampilan Menu PreOrder Tree 10](#_Toc200523686)

[Gambar 5 Tampilan Menu InOrder Tree 11](#_Toc200523687)

[Gambar 6 Tampilan Menu PostOrder Tree 11](#_Toc200523688)

[Gambar 7 Tampilan Menu Exit 11](#_Toc200523689)

[Gambar 8 Ilustrasi Insertion Pada Tree 14](#_Toc200523690)

[Gambar 9 Ilustrasi PreOrder Pada Tree 15](#_Toc200523691)

[Gambar 10 Ilustrasi InOrde Pada Tree 16](#_Toc200523692)

[Gambar 11 ilustrasi PostOrder Pada Tree 17](#_Toc200523693)

# SOAL 1

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

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

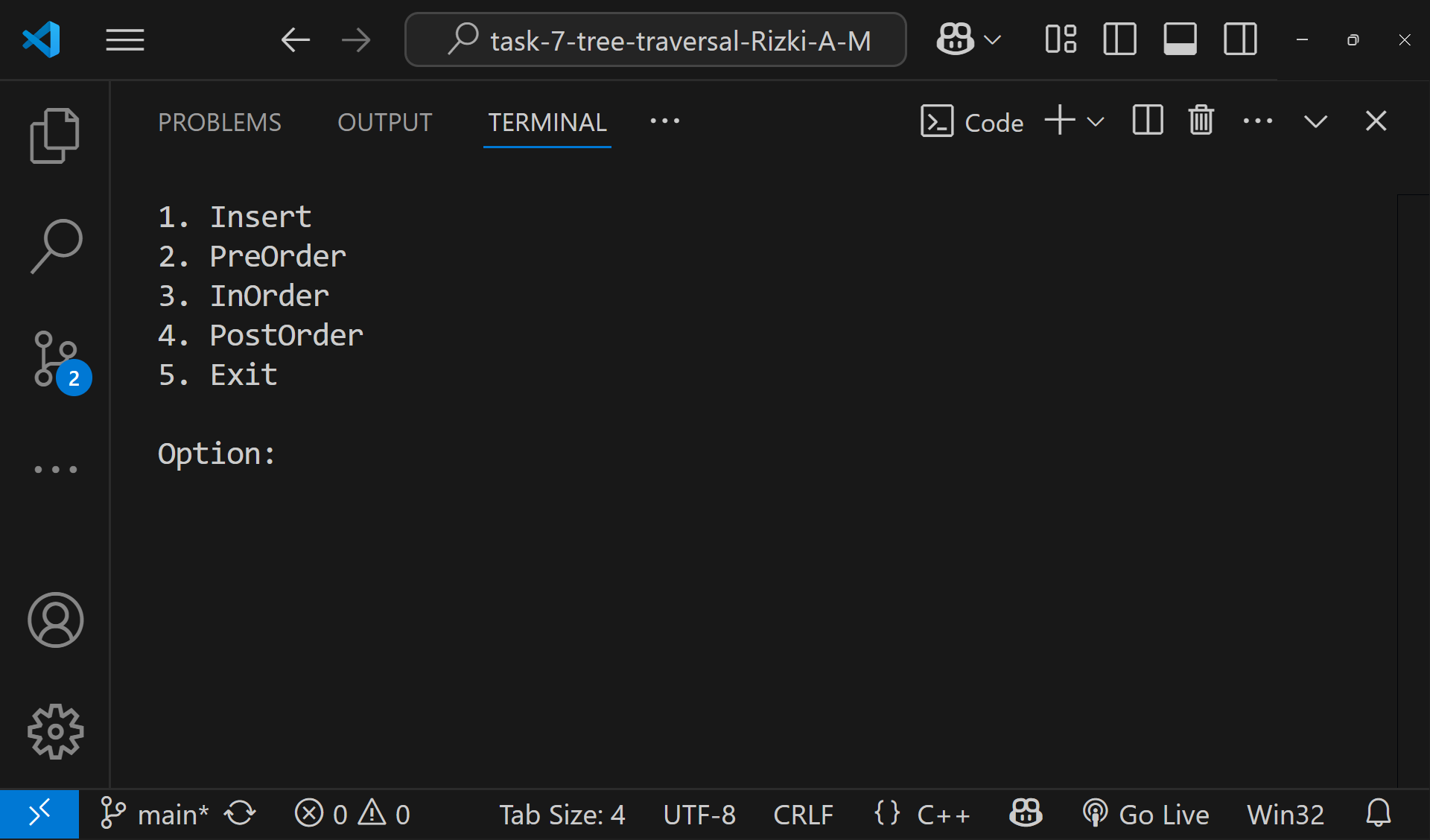
Gambar Soal 1 Modul 7

## Source Code

Tabel Source Code

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161 | #include <iostream>  #include <conio.h>  #include <stdlib.h>  using namespace std;  struct Node  {      int data;      Node \*left;      Node \*right;  };  void insert(Node \*\*root, int newData)  {      if (\*root == nullptr)      {          Node \*newNode;          newNode = new Node;          newNode -> data = newData;          newNode -> left = nullptr;          newNode -> right = nullptr;          \*root = newNode;          cout << " Data has been added";      }      else if (newData < (\*root) -> data)      {          insert(&((\*root)->left), newData);      }      else if (newData > (\*root) -> data)      {          insert(&((\*root)->right), newData);      }      else if (newData == (\*root) -> data)      {          cout << " Data is already exist";      }  }  void preOrder(Node \*root)  {      if (root == nullptr) return;      cout << root->data << " ";      preOrder(root->left);      preOrder(root->right);  }  void inOrder(Node \*root)  {      if (root == nullptr) return;      inOrder(root->left);      cout << root->data << " ";      inOrder(root->right);  }  void postOrder(Node \*root)  {      if (root == nullptr) return;      postOrder(root->left);      postOrder(root->right);      cout << root->data << " ";  }  // side quest  void printTree() {  }  void freeTree(Node \*root)  {      if (root == nullptr) return;      freeTree(root->left);      freeTree(root->right);      delete root;  }  int main()  {      int opt, val;      Node \*tree;      tree = nullptr;      do      {          system("cls");          cout << "1. Insert\n";          cout << "2. PreOrder\n";          cout << "3. InOrder\n";          cout << "4. PostOrder\n";          cout << "5. Exit\n";          cout << "\nOption: "; cin >> opt;          switch (opt)          {              case 1:                  cout << "\n Input:";                  cout << "\n ------";                  cout << "\n New data: ";                  cin >> val;                  insert(&tree, val);                  break;              case 2:                  cout << "PreOrder Traversal\n";                  cout << "==========================\n";                  if (tree == nullptr)                  {                      cout << "Tree is empty!\n";                  }                  else                  {                      preOrder(tree);                  }                  break;              case 3:                  cout << "InOrder Traversal\n";                  cout << "==========================\n";                  if (tree == nullptr)                  {                      cout << "Tree is empty!\n";                  }                  else                  {                      inOrder(tree);                  }                  break;              case 4:                  cout << "PostOrder Traversal\n";                  cout << "==========================\n";                  if (tree == nullptr)                  {                      cout << "Tree is empty!\n";                  }                  else                  {                      postOrder(tree);                  }                  break;              case 5:                  freeTree(tree);                  return 0;                default:                  cout << "Option is not valid! Please re-enter your option";                  break;          }          getch();      }      while(opt != 5);      return 0;  } |

## Output Program



Gambar Tampilan Menu Tree

Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Konten yang dihasilkan AI mungkin salah.

Gambar Tampilan Menu Insert Tree

Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Konten yang dihasilkan AI mungkin salah.

Gambar Tampilan Menu PreOrder Tree

Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Konten yang dihasilkan AI mungkin salah.

Gambar Tampilan Menu InOrder Tree

Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Konten yang dihasilkan AI mungkin salah.

Gambar Tampilan Menu PostOrder Tree

*Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Konten yang dihasilkan AI mungkin salah.*

Gambar Tampilan Menu Exit

## Pembahasan

* **Alur Program**

Program ini dimulai dengan mendefinisikan ***struktur*** *Node*, yang menjadi dasar pembentuk pohon biner. Setiap *Node* terdiri dari sebuah *variabel integer* yang akan menyimpan nilai, serta dua *pointer left dan right* yang masing-masing menunjuk ke anak kiri dan anak kanan dari *node* tersebut. Pada awal program, *pointer tree* akan diinisialisasi sebagai *nullptr* yang artinya pohon biner dalam keadaan kosong.

Selanjutnya, program akan menampilkan **menu utama** kepada pengguna dalam sebuah ***loop*** *do while*. Menu yang ada akan menyediakan lima opsi, yaitu Insert (untuk menambahkan data baru), PreOrder (untuk menampilkan data dengan urutan pre-order traversal), InOrder (untuk menampilkan data dengan urutan in-order traversal), PostOrder (untuk menampilkan data dengan urutan post-order traversal), dan Exit (untuk keluar dari program). Setiap selesai menjalankan fungsi yang ada di setiap pilihan maka tampilan layar program akan dibersihkan dengan *system ("cls")* sehingga tidak adanya penumpukkan hasil output pada layar. Loop akan terus berulang selama nilai variabel pilihan tidak sama dengan 5, dan setelah selesai menjalankan program yang ada (kecuali keluar), program akan menunggu input tombol apa pun karena adanya *getch()* sebelum akhirnya akan dibersihkan oleh *system ("cls")*.

Pemilihan opsi menu diatur menggunakan switch case. Apabila pengguna memilih **opsi 1 yaitu (**Insert**)**, program akan meminta input data baru yang kemudian disimpan dalam variabel val. Data ini lalu disisipkan ke dalam pohon melalui pemanggilan *fungsi insert()*. *Fungsi insert()* bekerja secara rekursif, yang mana apabila *root* saat ini adalah *nullptr*, *node* baru akan dibuat dan menjadi *root* di posisi tersebut. Jika data baru lebih kecil dari data pada *node* *root* saat ini, penyisipan akan dilanjutkan secara rekursif ke sub-pohon kiri. Sebaliknya, jika data baru lebih besar, penyisipan akan dilanjutkan ke sub-pohon kanan. Apabila data yang dimasukkan sudah ada, program akan menampilkan pesan "Data is already exist".

Untuk **opsi 2 (**PreOrder**), opsi 3 (**InOrder**)**, dan **opsi 4 (**PostOrder**)**, program terlebih dahulu memeriksa apakah tree kosong (*nullptr*). Apabila pohon yang ada kosong, pesan "Tree is empty!" akan ditampilkan. Kemudian, apabila pohon yang ada berisi data, *fungsi traversal* yang sesuai akan dipanggil untuk preOrder(), inOrder(), atau postOrder(). *Fungsi preOrder()* akan menampilkan data node saat ini, lalu mengunjungi anak kiri, dan kemudian anak kanan. *Fungsi inOrder()* akan mengunjungi anak kiri, lalu menampilkan data node saat ini, dan kemudian mengunjungi anak kanan, menghasilkan urutan data yang terurut. Sementara itu, *fungsi postOrder()* akan mengunjungi anak kiri, lalu anak kanan, dan terakhir menampilkan data node saat ini.

Terakhir, saat pengguna memilih **opsi 5 (**Exit**)**, program akan memanggil *fungsi freeTree()*. Fungsi ini bertanggung jawab atas **dealokasi memori** yang digunakan oleh pohon biner. *freeTree()* bekerja secara rekursif dengan pendekatan ***post-order traversal***, yaitu membebaskan sub-pohon kiri terlebih dahulu, lalu sub-pohon kanan, dan barulah menghapus (delete) node saat ini. Proses ini memastikan bahwa semua memori yang dialokasikan secara dinamis untuk node-node pohon dikembalikan ke sistem, mencegah memory leak. Setelah seluruh memori dibebaskan, program akan keluar dengan mengembalikan nilai 0. Apabila pengguna memilih opsi yang tidak valid, program akan menampilkan pesan kesalahan "Option is not valid! Please re-enter your option".

* **Insertion**

Fungsi penting untuk program ini adalah *insert(),* yang bertanggung jawab untuk menambahkan data baru ke dalam pohon. *insert()* akan diinisialisasikan dengan **Binary Search Tree (BST)** yaitu sebuah struktur data yang efisien untuk pencarian dan pengurutan. Fungsi ini akan mengambil dua parameter yaitu sebuah ***pointer ke pointer*** *Node* yang merepresentasikan akar (atau sub-akar) pohon saat ini, dan sebuah ***integer*** *newData* yang merupakan data yang ingin disisipkan.

Proses penyisipan dimulai dengan memeriksa apakah posisi root saat ini **kosong (null)**. Apabila iya, berarti kita telah menemukan tempat yang tepat untuk newData. Maka, sebuah newNode akan dibuat, newData disematkan di dalamnya, dan pointer left serta right dari newNode diatur ke nullptr (karena ini adalah node daun baru). newNode ini kemudian menjadi root di posisi tersebut. Kemudian, apabila root yang ada tidak kosong, fungsi akan membandingkan newData dengan data yang ada di root saat ini. Jika newData **lebih kecil**, program akan memanggil insert() secara **rekursif** untuk subtree kiri. Sebaliknya, jika newData **lebih besar**, rekursi akan terjadi pada subtree kanan. Ada hal penting disini yaitu bila newData **sama** dengan data yang sudah ada, program akan mengeluarkan pesan bahwa data duplikat, ini berguna untuk menjaga integritas BST sesuai aturan standar.

Sebuah gambar berisi lingkaran

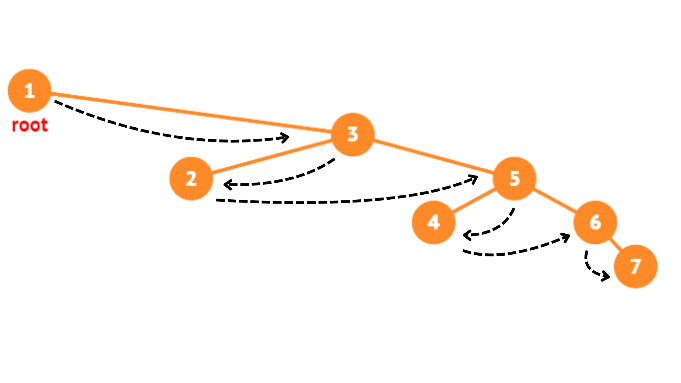
Konten yang dihasilkan AI mungkin salah.

Gambar Ilustrasi Insertion Pada Tree

* **Pre-Order Traversal**

Fungsi preOrder() berfungsi untuk melakukan traversal pada struktur data pohon (tree). Dalam metode ini, urutan kunjungan dimulai dari node induk (parent) terlebih dahulu, kemudian lanjut ke subtree kiri, dan yang terakhir ke subtree kanan. Traversal dilakukan secara rekursif, yang berarti fungsi akan terus memanggil dirinya sendiri selama masih ada node yang perlu dikunjungi. Langkah pertama dalam fungsi ini adalah memeriksa apakah node saat ini bernilai null. Apabila iya, maka tidak ada lagi node yang bisa dikunjungi dan fungsi akan berhenti.

Namun, apabila node yang ada tidak null, maka data atau nilai dari node tersebut langsung dicetak. Setelah itu, fungsi akan masuk ke subtree sebelah kiri dan melakukan proses yang sama secara rekursif. Setelah seluruh node di subtree kiri selesai dikunjungi, fungsi akan berpindah ke subtree kanan dan kembali menjalankan proses yang sama. Dengan pola tersebut, preOrder() selalu mengunjungi node dalam urutan: node saat ini, subtree kiri, dan subtree kanan. Pola ini berguna, misalnya, ketika ingin menyalin struktur pohon atau mengekstrak informasi dari atas ke bawah secara teratur.



Gambar Ilustrasi PreOrder Pada Tree

* **In-Order Traversal**

Fungsi inOrder() digunakan untuk melakukan penelusuran (traversal) pada struktur pohon. Pada metode ini, proses kunjungan dimulai dari **subtree kiri** terlebih dahulu, kemudian ke **node induk (parent)**, dan terakhir ke **subtree kanan** secara rekursif. Urutan ini sangat berguna karena pada tree bertipe binary search tree (BST), traversal in-order akan mencetak data dalam urutan yang terurut dari yang terkecil hingga terbesar.

Saat fungsi dijalankan, hal pertama yang dilakukan adalah memeriksa apakah node saat ini bernilai null. Apabila null, berarti tidak ada node lagi yang bisa dikunjungi, sehingga fungsi akan berhenti. Namun, apabila terdapat node, maka fungsi akan lebih dulu mengunjungi subtree kiri secara rekursif. Setelah itu, data dari node induk akan dicetak. Terakhir, fungsi akan mengunjungi subtree kanan dan mencetak datanya. Pola kunjungannya mengikuti urutan dari **kiri ke induk dan ke kanan,** dan ini berulang hingga seluruh node telah dikunjungi.

Sebuah gambar berisi lingkaran

Konten yang dihasilkan AI mungkin salah.

Gambar Ilustrasi InOrde Pada Tree

* **Post-Order Traversal**

Fungsi postOrder() digunakan untuk melakukan traversal pada struktur pohon. Dalam metode ini, urutan kunjungannya dimulai dari **subtree kiri**, lalu ke **subtree kanan**, dan terakhir ke **node induk (parent)** secara rekursif, dan cocok digunakan ketika kita ingin memproses atau menghapus node setelah semua turunannya selesai dikunjungi, seperti saat menghapus seluruh isi pohon.

Cara kerjanya dimulai dengan memeriksa apakah node saat ini bernilai null. Apabila null, maka tidak ada node lagi yang bisa dikunjungi, maka fungsi akan berhenti. Namun apabila node ada, maka program pertama-tama akan mengunjungi subtree kiri secara rekursif. Setelah itu, program lanjut ke subtree kanan dan melakukan hal yang sama. Jika kedua sisi sudah selesai dikunjungi, barulah program mencetak data dari node induk. Dengan demikian, urutan traversal-nya adalah dari **kiri ke kanan dan akhirnya ke induk**, dan pola ini berulang hingga seluruh node dalam tree selesai dikunjungi.

Sebuah gambar berisi lingkaran

Konten yang dihasilkan AI mungkin salah.

Gambar ilustrasi PostOrder Pada Tree

# TAUTAN GITHUB

https://github.com/Rizki-A-M/Rizki-A-M-PRAKTIKUM\_ALGORITMA\_DAN\_STRUKTUR\_DATA.git