M. Bahrul Ulum, S.kom, M.Kom

Program Studi Teknik Informatika

Fakultas Ilmu Komputer

Universitas Esa Unggul

2018



Modul Kuliah Struktur Data

TREE

**TREE**

Teori

Binary Tree (Pohon Biner) yaitu pohon yang setiap simpul/node-nya paling banyak mempunyai dua buah subpohon.

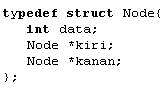
**Contoh implementasi :**

untuk membuat pohon silsilah keluarga, ungkapan aritmatika yang setiap operatornya dipasang sebagai simpul pencabangan dan operand-operandnya sebagai subpohon, dll.

Binary tree dapat diimplementasikan dalam C++ dengan menggunakan double linkedlist.

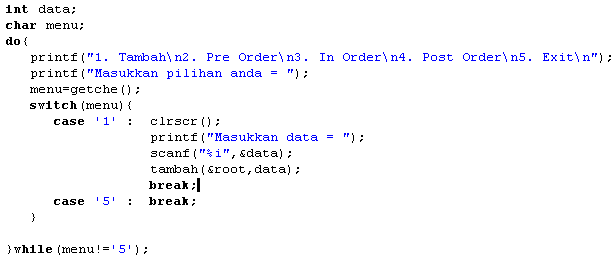
Tree merupakan struktur hirarkis dari sebuah susunan data. Dalam tree,kita mempunyai sebuah Root yang nantinya akan menjadi awal dari tree tersebut. Untuk tree, kita juga membutuhkan struct. Ada banyak jenis-jenis Tree, tapi yang akan kita pelajari saat ini adalah implementasi dari Binary Tree, yaitu Tree yang hanya memiliki 2 tangan.

Langkah awal membuat Binary Tree adalah membuat struct seperti pada inisialisasi double linked list.





Setelah kita membuat struct, kita membuat void main yang di dalamnya kita deklarasikan sebuah variable yang bertipe data Node (struct yang telah kita buat) untuk nantinya digunakan sebagai root.



Tree merupakan salah satu bentuk struktur data tidak linear yang menggambarkan hubungan yang bersifat hirarkis (hubungan one to many) antara elemen-elemen. Tree bisa didefinisikan sebagai kumpulan simpul/node dengan satu elemen khusus yang disebut Root dan node lainnya. Tree juga adalah suatu graph yang acyclic, simple, connected yang tidak mengandung loop.

Sebuah binary search tree (bst) adalah sebuah pohon biner yang boleh kosong, dan setiap nodenya harus memiliki identifier/value. Value pada semua node subpohon sebelah kiiri adalah selalu lebih kecil dari value dari root, sedangkan value subpohon di sebelah kanan adalah sama atau lebih besar dari value pada root, masing-masing subpohon tersebut (kiri dan kanan) itu sendiri adalah juga binary search tree.

Struktur data bst sangat penting dalam struktur pencarian, misalkan dalam kasus pencarian dalam sebuah list, jika list sudah dalam keadaan terurut maka proses pencarian akan semakin cepat, jika kita menggunakan list contigue dan melakukan pencarian biner,akan tetapi jika kita ingin melakukan perubahan isi list (insert atau delete), menggunakan list contigue akan sangat lambat, karena prose insert dan delete dalam list contigue butuh memindahkan linked-list, yang untuk operasi insert atau delete tinggal mengatur- atur pointer,akan tetapi pada n-linked list, kita tidak bisa melakukan pointer sembarangan setiap saat, kecuali hanya satu kali dengan kata lain hanya secara squential.

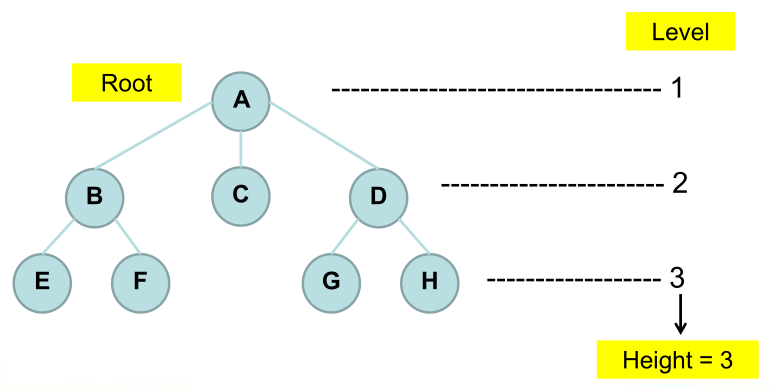
* **Pengertian Binary Tree**

Binary Tree merupakan salah satu bentuk struktur data tidak linear yang menggambarkanhubungan yang bersifat hirarkis (hubungan one to many) antara elemen-elemen. Tree bisa didefinisikan sebagai kumpulan simpul/node dengan satu elemen khusus yang disebut Root dan node lainnya ( disebut subtree).

Dalam tree terdapat jenis-jenis tree yang memiliki sifat khusus, diantaranya adalah binary tree.

Binary tree adalah suatu tree dengan syarat bahawa tiap node (simpul) hanya boleh memiliki maksimal dua subtree dan kedua subtree tersebut harus terpisah. Tiap node dalam binary treee boleh memiliki paling banyak dua child (anak simpul), secara khusus anaknya dinamakan kiri dan kanan.

Binary Tree merupakan himpunan vertex-vertex yang terdiri dari 2 subtree (dengan disjoint) yaitu subtree kiri dan subtree kanan. Setiap vertex dalam binary tree mempunyai derajat keluar max = 2.

****

Sebuah pohon biner adalah grafik asiklis yang terhubung dimana setiap tingkatan dari susut tidak lebih dari 3. Ini dapat ditunjukkan bahwa dalam pohon biner manapun, terdapat persis dua atau lebih simpul dengan tingkat satu daripada yang terdapat dengan tingkat tiga, tetapi bisa terdapat angka apa saja dari simpul dengan tingkat dua. Sebuah pohon biner berakar merupakan sebuah grafik yang mempunyai satu dari sudutnya dengan tingkat tidak lebih dari dua sebagai akar.

Dengan akar yang dipilih, setiap sudut akan memiliki ayah khusus, dan diatas dua anak bagaimanapun juga, sejauh ini terdapat keterbatasan informasi untuk membedakan antara anak kiri atau kanan. Jika kita membuang keperluan yang tak terkoneksi, membolehkan bermacam koneksi dalam komponen di grafik, kita memanggil struktur sebuah hutan.

Sebuah jalan lain untuk mendefinisikan pohon biner melalui definisi rekursif pada grafik langsung. Sebuah pohon biner dapat berarti :

* Sebuah sudut tunggal.
* Sebuah graf yang dibentuk dengan mengambil dua pohon biner, menambahkan sebuah sudut, dan menambahkan sebuah panah langsung dari sudut yang baru ke akar dai setiap pohon biner.

Pohon biner dapat dikontruksi dari bahasa pemrogaraman primitif dalam berbagai cara. Dalam bahasa yang menggunakan records dan referensi. Pohon biner secara khas dikontruksi dengan mengambil sebuah struktur simpul pohon yang memuat beberapa data dan referensi ke anak kiri dan anak kanan.

Kadang-kadang itu juga memuat sebuah referensi ke ayahnya yang khas. Jika sebuah simpul mempunyai kurang dari dua anak, beberapa penunjuk anak diaatur kedalam nilai nol khusus atau kesebuah simpul sentinel.

Pohon biner dapat juga disimpan sebagai struktur data implisit dalam array, dan jika pohon tersebut merupakan sebuah pohon biner lengkap, metode ini tidak boros tempat. Dalam penyusunan yang rapat ini, jika sebuah simpul memiliki indeks i, anaknya dapat ditemukan pada indeks ke-2i+1 dan 2i+2, meskipun ayahnya (jika ada) ditemukan pada indeks lantai ((i-1)/2) (asumsikan akarnya memiliki indeks kosong). Metode ini menguntungkan dari banyak penyimpanan yang rapat dan memiliki referensi lokal yang lebih baik, teristimewa selama sebuah preordeer traversal.

* **Istilah-istilah dalam pohon**

1. Predesesor

Node yang berada diatas node tertentu.

(contoh : B predesesor dari E dan F)

2. Succesor

Node yang berada dibawah node tertentu.

(contoh : E dan F merupakan succesor dari B)

3. Ancestor

Seluruh node yang terletak sebelum node tertentu dan

terletak pada jalur yang sama.

(contoh : A dan B merupakan ancestor dari F)

Istilah – istilah Dalam Pohon

4. Descendant

Seluruh node yang terletak sesudah node tertentu

dan terletak pada jalur yang sama.

(contoh : F dan B merupakan ancestor dari A)

5. Parent

Predesesor satu level diatas satu node

(contoh : B merupakan parent dari F)

6. Child

Succesor satu level dibawah satu node

(contoh : F merupakan child dari B)

7. Sibling

Node yang memiliki parent yang sama dengan satu

node (contoh : E dan F adalah sibling)

8. Subtree

Bagian dari tree yang berupa suatu node beserta

descendant-nya (contoh : Subtree B, E, F dan

Subtree D, G, H)

9. Size

Banyaknya node dalam suatu tree (contoh : gambar

tree diatas memiliki size = 8)

10. Height

Banyaknya tingkat/level dalam suatu tree (contoh :

gambar tree diatas memiliki height = 3)

11. Root (Akar)

Node khusus dalam tree yang tidak memiliki

predesesor (Contoh : A)

12. Leaf (Daun)

Node-node dalam tree yang tidak memiliki daun

(contoh : Node E,F,C,G,H)

13. Degree (Derajat)

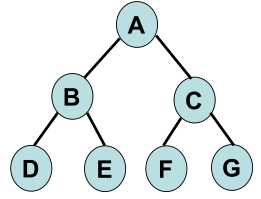
Banyaknya child yang dimiliki oleh suatu node

(contoh : Node A memiliki derajat 3, node B memiliki derajat 2)

* **Istilah pada pohon Binar**

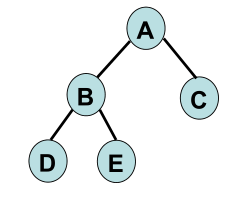
1. **Pohon Biner Penuh (Full Binary Tree)**

Semua simpul (kecuali daun) memiliki 2 anak dan tiap cabang memiliki panjang ruas yang sama.



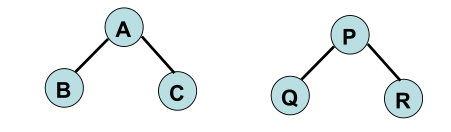
1. **Pohon Biner Lengkap (Complete Binary Tree)**

Hampir sama dengan Pohon BinerPenuh, semua simpul (kecualidaun) memiliki 2 anak tetapi tiap cabang memiliki panjang ruas berbeda.



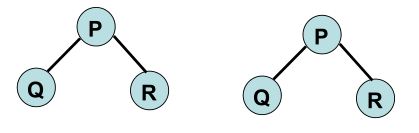
1. **Pohon Biner Similer**

Dua pohon yang memiliki struktur yang sama tetapi informasinya berbeda.



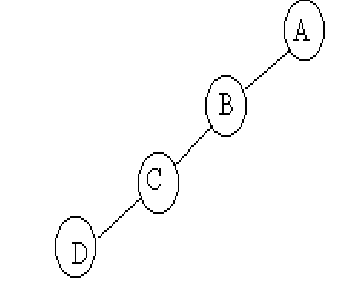
1. **Pohon Biner Ekivalent**

Dua pohon yang memiliki struktur dan informasi yangsama.



1. **Pohon Biner Miring (Skewed Tree)**

Dua pohon yang semua simpulnya mempunyai satu anak / turunan kecuali daun.



* **Sifat utama Pohon Berakar**

1. Jika Pohon mempunyai Simpul sebanyak n, maka banyaknya ruas atau edge adalah (n-1).
2. Mempunyai Simpul Khusus yang disebut Root, jika Simpul tersebut memiliki derajat keluar >= 0, dan derajat masuk = 0.
3. Mempunyai Simpul yang disebut sebagai Daun / Leaf, jika Simpul tersebut berderajat keluar = 0, dan berderajat masuk = 1.
4. Setiap Simpul mempunyai Tingkatan / Level yang dimulai dari Root yang Levelnya = 1 sampai dengan Level ke - n pada daun paling bawah. Simpul yang mempunyai Level sama disebut Bersaudara atau Brother atau Stribling.
5. Pohon mempunyai Ketinggian atau Kedalaman atau Height, yang merupakan Level tertinggi
6. Pohon mempunyai Weight atau Berat atau Bobot, yang banyaknya daun (leaf) pada Pohon.
7. Banyaknya Simpul Maksimum sampai Level N adalah :

|  |
| --- |
| **2 (N) - 1** |

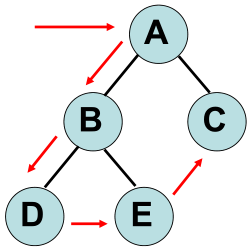
1. Banyaknya Simpul untuk setiap Level I adalah :

|  |
| --- |
| **N**  ∑ **2 ( I – 1)**  **I = 1** |

* **Kunjungan pada pohon Biner**

Kunjungan pohon biner terbagi menjadi 3 bentuk binary tree :

1. Kunjungan secara preorder ( Depth First Order), mempunyai urutan :
2. Cetak isi simpul yang dikunjungi ( simpul akar ),
3. Kunjungi cabang kiri,
4. Kunjungi cabang kanan .

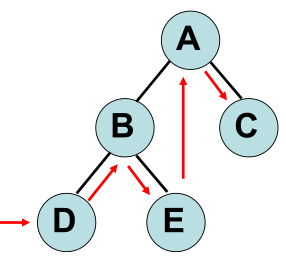




1. Kunjungan secara inorder ( symetric order), mempunyai urutan :  
   a. Kunjungi cabang kiri,

b. Cetak isi simpul yang dikunjungi (simpul akar),

c. Kunjungi cabang kanan .





1. Kunjungan secara postorder, mempunyai urutan :  
   a. Kunjungi cabang kiri,

b. Kunjungi cabang kanan,

c. Cetak isi simpul yang dikunjungi ( simpul akar ).



****

* **Aplikasi pohon Biner**

**Notasi Prefix, Infix dan Postfix**

Pada bagian ini akan dibahas tentang bagaimana menyusun sebuah Pohon Binar yang apabila dikunjungisecara PreOrder akan menghasilkan Notasi Prefix,kunjungan secara InOrder menghasilkan Notasi Infix, dankunjungan PostOrder menghasilkan Notasi Postfix.

* 1. **Contoh kasus dan Program**

**Contoh Program Tree dalam C++**

#include <iostream.h>

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

struct Node{

int data;

Node \*kiri;

Node \*kanan;

};

int count;

void tambah(Node \*\*root, int databaru)

{

if((\*root) == NULL){

Node \*baru;

baru = new Node;

baru->data = databaru;

baru->kiri = NULL;

baru->kanan = NULL;

(\*root) = baru;

(\*root)->kiri = NULL;

(\*root)->kanan = NULL;

printf("Data telah Dimasukkan");

}

else if(databaru < (\*root)->data)

tambah(&(\*root)->kiri,databaru);

else if(databaru > (\*root)->data)

tambah(&(\*root)->kanan,databaru);

else if(databaru == (\*root)->data)

printf("Data sudah ada!!");

}

void preOrder(Node \*root){

if(root != NULL){

printf("%d " ,root->data);

preOrder(root->kiri);

preOrder(root->kanan);

}

}

void inOrder(Node \*root){

if(root != NULL){

inOrder(root->kiri);

printf("%d ",root->data);

inOrder(root->kanan);

}

}

void postOrder(Node \*root){

if(root != NULL){

postOrder(root->kiri);

postOrder(root->kanan);

printf("%d ",root->data);

}

}

void search(Node \*\*root, int cari)

{

if((\*root) == NULL){

printf("Maaf,Data tidak ditemukan!");

}

else if(cari < (\*root)->data)

search(&(\*root)->kiri,cari);

else if(cari > (\*root)->data)

search(&(\*root)->kanan,cari);

else if(cari == (\*root)->data)

printf("Data ditemukan!!!");

}

void hapus(Node \*\*root, int del)

{

if((\*root) == NULL){

printf("Data tidak ada!!");

}

else if(del < (\*root)->data)

hapus(&(\*root)->kiri,del);

else if(del > (\*root)->data)

hapus(&(\*root)->kanan,del);

else if(del == (\*root)->data)

{

(\*root)=NULL;

printf("Data telah Terhapus");

}

}

int main(){

int pil,cari,del;

Node \*pohon;

pohon = NULL;

do{

int data;

system("cls");

printf(" PROGRAM TREE LANJUTAN \n");

printf("================================\n");

printf(" 1. Masukkan Data \n");

printf(" 2. Transverse \n");

printf(" 3. Cari \n");

printf(" 4. Hapus \n");

printf(" 5. Clear Data \n");

printf(" 6. Keluar \n");

printf("================================\n");

printf("Masukkan Pilihan Anda : ");

scanf("%d",&pil);

switch(pil){

case 1:

printf("Masukkan data baru : ");

scanf("%d", &data);

tambah(&pohon,data);

break;

case 2:

printf("\nPreOrder : ");

if(pohon!=NULL) preOrder(pohon);

else printf("Data masih kosong");

printf("\ninOrder : ");

if(pohon!=NULL) inOrder(pohon);

else printf("Data masih kosong");

printf("\npostOrder : ");

if(pohon!=NULL) postOrder(pohon);

else printf("Data masih kosong");

break;

case 3:

printf("Cari data : ");

scanf("%d", &cari);

search(&pohon,cari);

break;

case 4:

printf("Hapus data : ");

scanf("%d", &del);

hapus(&pohon,del);

break;

case 5:

pohon = NULL;

printf("Semua data telah terhapus");

break;

case 6:

return 0;

default:

printf("Maaf, pilihan Anda Salah");

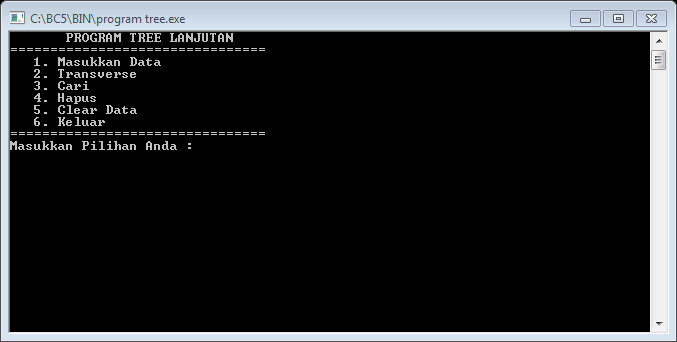
}

getch();

}while(pil!=7);

}

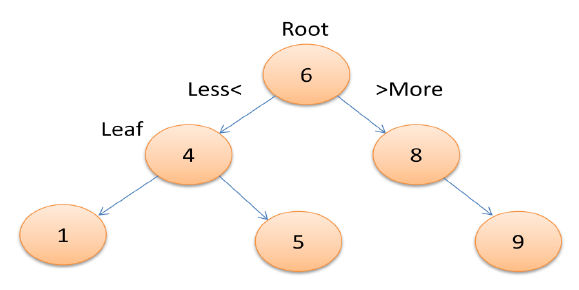
Tampilan program saat dijalankan :



Tampilan Transverse setelah diinput data 1,3,7 dan 5.



Ilustrasi Binary Tree



Pembentukan Tree

**Langkah-langkah Pembentukan Binary Tree**

1. Siapkan node baru
   1. alokasikan memory-nya
   2. masukkan info-nya
   3. set pointer kiri & kanan = NULL
2. Sisipkan pada posisi yang tepat

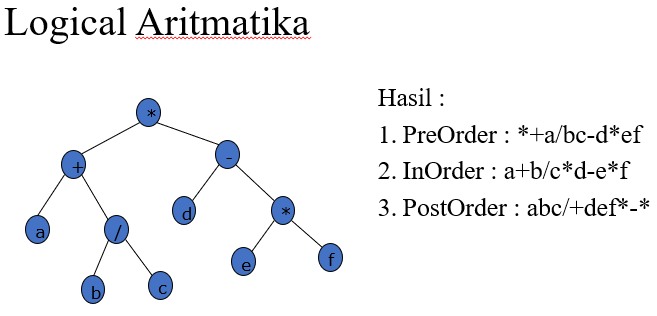
* **penelusuran** 🡪 utk menentukan posisi yang tepat; info yang nilainya lebih besar dari parent akan ditelusuri di sebelah kanan, yang lebih kecil dari parent akan ditelusuri di sebelah kiri
* **penempatan** 🡪 info yang nilainya lebih dari parent akan ditempatkan di sebelah kanan, yang lebih kecil di sebelah kiri

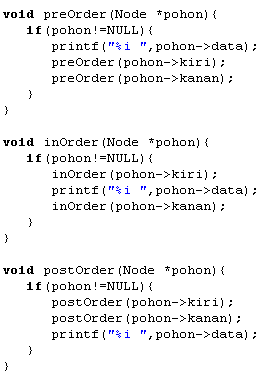
Operasi Umum Tree

Salah satu operasi yang paling umum dilakukan terhadap sebuah tree adalah kunjungan (traversing)

**Ada 3 urutan dasar yang dapat digunakan untuk mengunjungi pohon, yaitu :**

1. **PreOrder** : Cetak isi node yang dikunjungi, kunjungi Left Child, kunjungi Right Child.
2. **InOrder** : Kunjungi Left Child, cetak isi node yang dikunjungi, kunjungi Right Child.
3. **PostOrder** : Kunjungi Left Child, kunjungi Right Child cetak isi node yang dikunjungi.





Setelah itu, kita buat fungsi untuk menampilkan ketiga cara tersebut

Algoritma Pre Order:

1. Cek apakah tree kosong, jika tidak lakukan langkah selanjutnya.
2. Tampilkan isi pohon
3. Panggil fungsi diri sendiri dengan parameter pohon->kiri
4. Panggil fungsi diri sendiri dengan parameter pohon->kanan

Algoritma In Order:

1. Cek apakah tree kosong, jika tidak lakukan langkah selanjutnya.
2. Panggil fungsi diri sendiri dengan parameter pohon->kiri
3. Tampilkan isi pohon
4. Panggil fungsi diri sendiri dengan parameter pohon->kanan

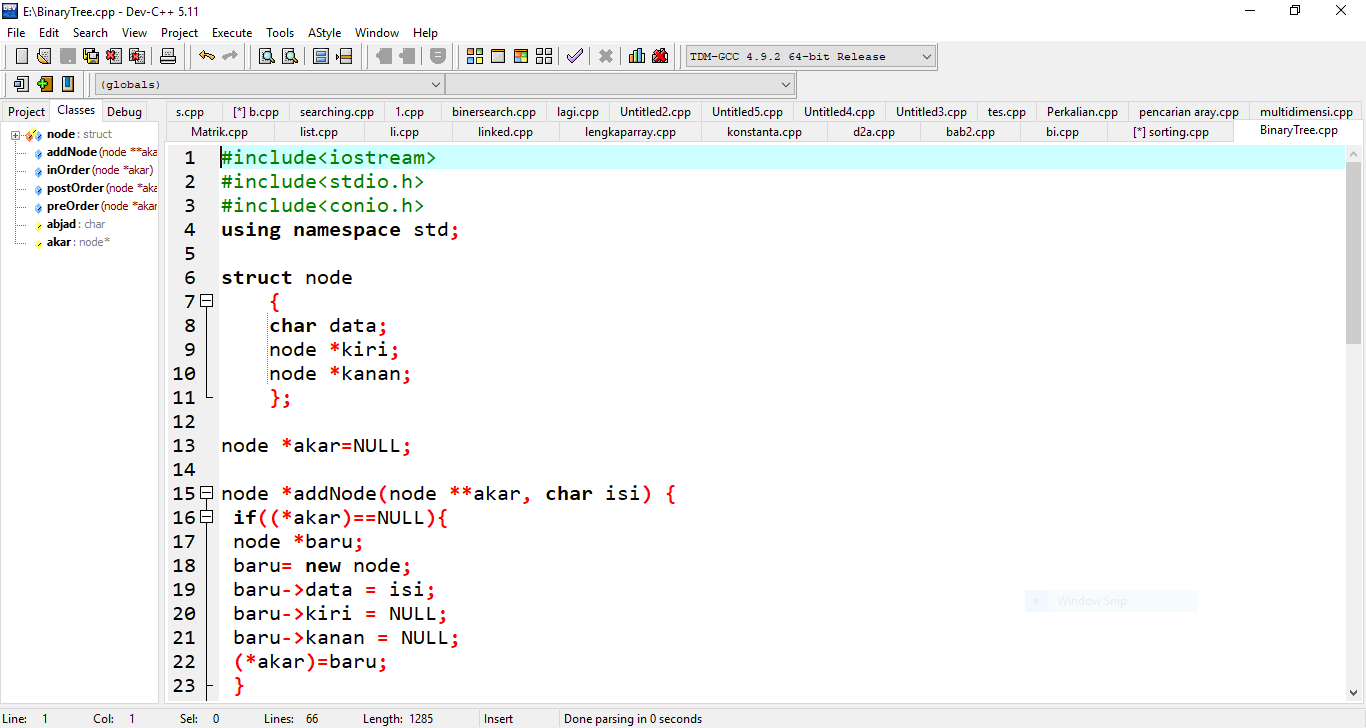
Algoritma Post Order:

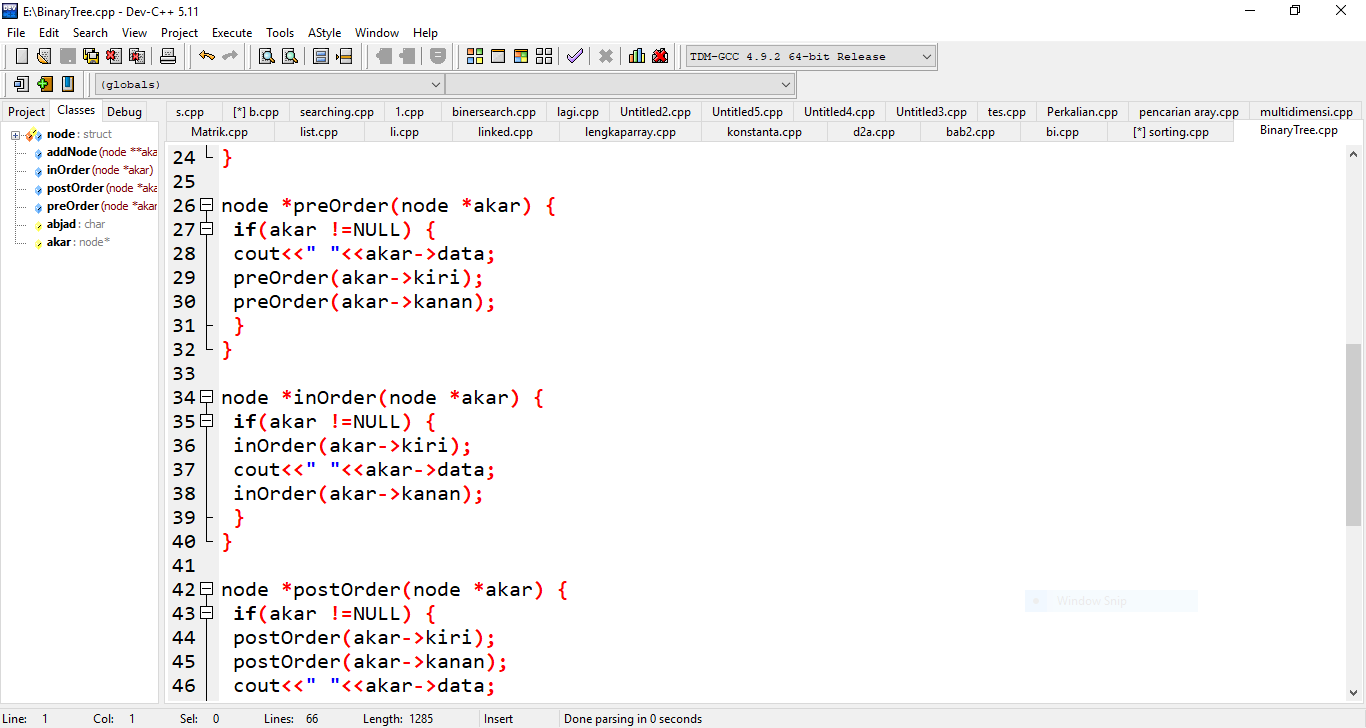
1. Cek apakah tree kosong, jika tidak lakukan langkah selanjutnya.
2. Panggil fungsi diri sendiri dengan parameter pohon->kiri
3. Panggil fungsi diri sendiri dengan parameter pohon->kanan
4. Tampilkan isi pohon

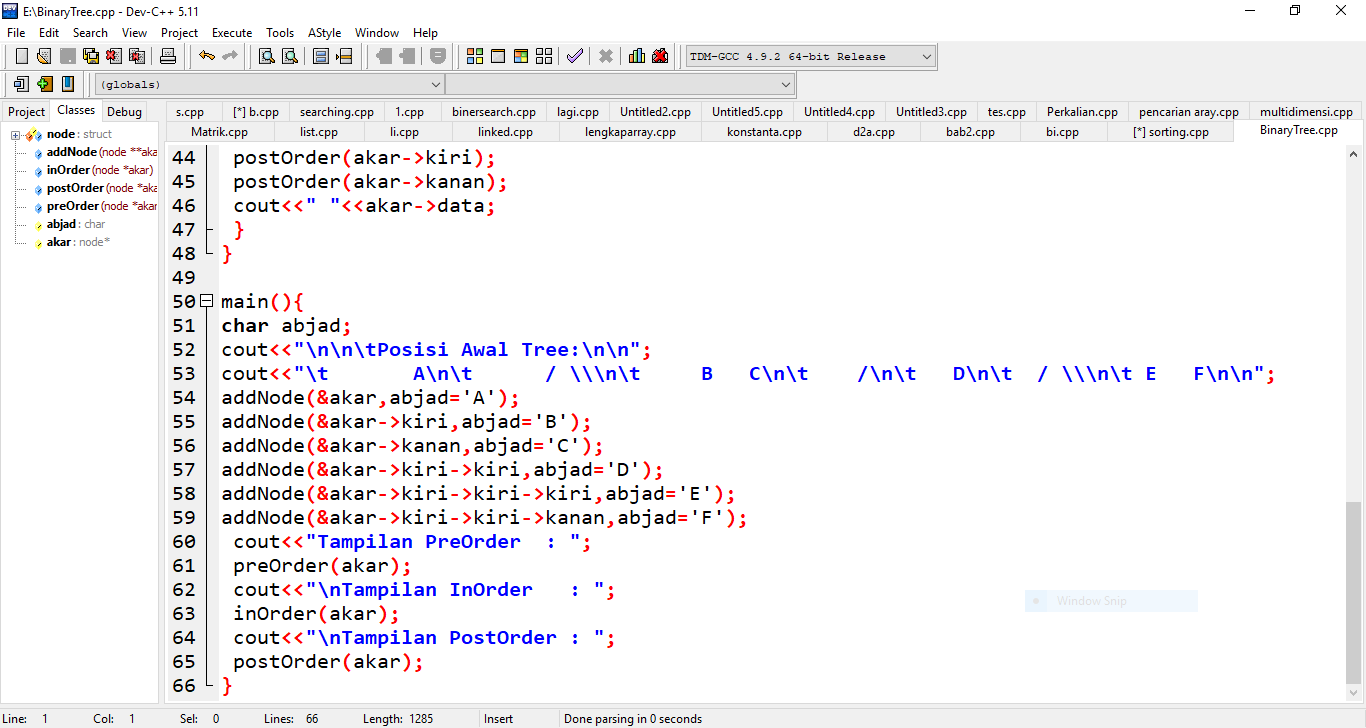
* **Pemahaman :** Jika dalam suatu fungsi memanggil fungsi, bukan berarti fungsi yang memanggil akan berhenti ketika memanggil fungsi. Tetapi fungsi pemanggil akan berjalan lagi setelah fungsi yang di panggil selesai dijalankan. Misalnya saja pada InOrder, walaupun pada langkah ke 2 fungsi InOrder memanggil dirinya sendiri tetapi langkah ke 3 dan ke 4 tetap di jalankan setelah fungsi yang di panggil pada langkah ke 2 selesai di jalankan.

Pelaksanaan Praktikum

**Program**







Latihan

* + 1. Buatlah Program C++ Binary Tree dengan 3 Operasi dasar (PreOrder, InOrder dan PostOrder) sesuai gambar dibawah ini :