MAKALAH TREE DALAM

BAHASA C



OLEH:

DIKI CANDRA

NIM 2022903430010

D4 TEKNOLOGI REKAYASA KOMPUTER JARINGAN

POLITEKNIK NEGERI LHOKSEUMAWE

2022/2023

**DAFTAR ISI**

KATA PENGANTAR i

DAFTAR ISI ii

A. BAB I PENDAHULUAN 1

1.1 Latar Belakang 1

1.2 Tujuan Penulisan 2

1.3 Manfaat Penulisan 2

B. BAB II PEMBAHAN 3

2.1 Tree 3

2.2 Binary Tree 3

2.3 Istilah Dalam Tree 5

2.4 Istilah Pada Pohon Biner 7

2.5 Sifat Utama Pohon Berakar 9

2.6 Kunjungan Pada Pohon Biner 10

2.7 Aplikasi Pohon Biner 11

C. BAB III PENUTUP 14

3.1 Kesimpulan 14

3.2 Saran 14

**BAB I**

**PENDAHULUAN**

* 1. **Latar Belakang**

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 yangdisebut Root dan node lainnya terbagi menjadi himpunan-himpeunan yang saling tak berhubungan satu sama lainnya (disebut subtree).

Tree juga adalah suatu graph yangacyclic, 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 kiri adalah selalu lebih kecil dari value dari root, sedangkan value subpohon disebelah kanan adalah sama atau lebih besar dari value pada root, masing–masing subpohon tersebut (kiri&kanan) itu sendiri adalah juga bst.

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 sangat cepat, jika kita menggunanan list contigue dan melakukan pencarian biner. akan tetapi, jika kita ingin melakukan perubahan isi list (insert ataudelete), menggunakan list contigue akan sangat lambat, karena proses insert dan delete dalam list contigue butuh memindahkan banyak elemen setiap saat. mungkin kita bisa juga menggunakan 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 sequential.

**1.2. Tujuan Penulisan**

1. Memahami tentang Tree
2. Memahami pengertian dari Binary Tree
3. Mengetahui istilah-istilah dalam Tree
4. Mengetahui istilah pada pohon Biner
5. Mengetahui sifat utama pada pohon berakar
6. Mengetahui cara kunjungan pohon Biner
7. Mengetahui aplikasi pohon Biner

**1.3. Manfaat Penulisan**

Penulis membuat makalah ini agar dapat bermanfaat bagi pembaca, terutama bagi penulis sendiri. Manfaat tersebut antara lain seperti, menjadikan mahasiswa Indonesia menjadi mahasiswa madani yang mampu memanfaatkan potensi individu dalam mengembangkan karya tulis.

**BAB II**

**PEMBAHASAN**

* 1. **TREE**

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.

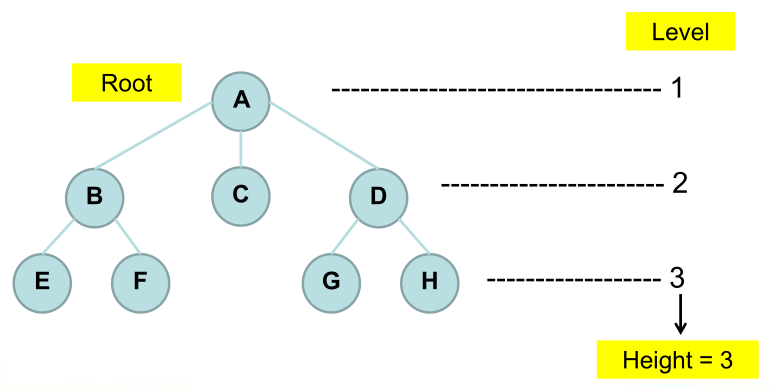
* 1. **BINARY TREE**

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.

**2.3. ISTILAH DALAM TREE**

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)

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)

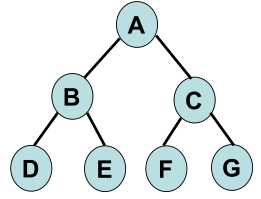
13. Degree (Derajat)

Banyaknya child yang dimiliki oleh suatu node. (contoh : Node A memiliki derajat 3, node B memiliki derajat 2)

**2.4. ISTILAH PADA POHON BINER**

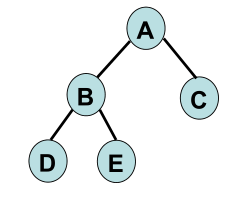
**a). Pohon Biner Penuh (Full Binary Tree)**

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



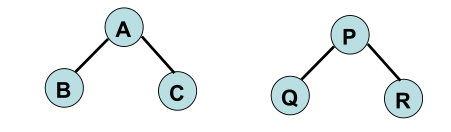
**b). Pohon Biner Lengkap (Complete Binary Tree)**

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



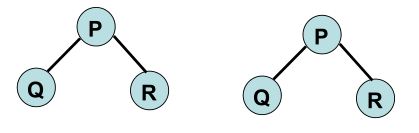
**c). Pohon Biner Similer**

Dua pohon yang memiliki struktur yang sama tetapi informasinya berbeda.



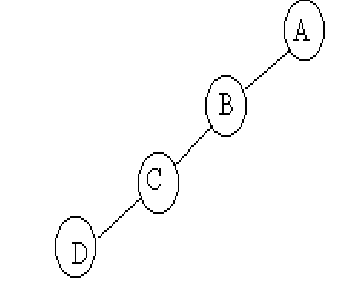
**d). Pohon Biner Ekivalent**

Dua pohon yang memiliki struktur dan informasi yangsama.



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

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



* 1. **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** |

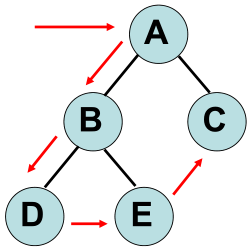
1. Banyaknya Simpul untuk setiap Level I adalah :

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

* 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 .

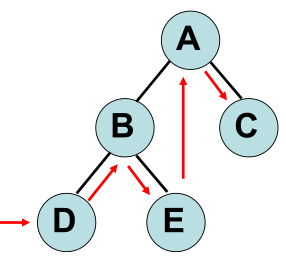




1. Kunjungan secara inorder ( symetric order), mempunyai urutan :  
   a. Kunjungi cabang kiri,

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

c. Kunjungi cabang kanan .

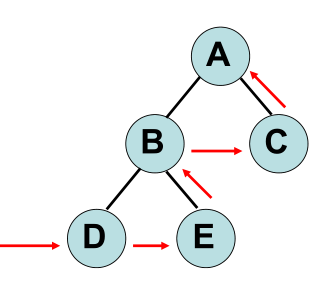




1. Kunjungan secara postorder, mempunyai urutan :  
   a. Kunjungi cabang kiri,

b. Kunjungi cabang kanan,

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



****

* 1. **APLIKASI POHON BINER**

Notasi Prefix, Infix dan Postfix

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

1. Prefix

Yaitu notasi yang terbentuk atas operator dengan operand, dimana operator berada didepan operand.

Contoh : A + B \* C (Infix)

Maka notasi prefixnya adalah + A\*BC

Pemecahannya :

A + B \* C

Diketahaui ada 3 operand yaitu : A, B, C, dan 2 operator yaitu : +, \*. Proses dimulai dengan melihat dari hirarkhi operator. Contoh diatas operator yang tertinggi adalah \* kemudian +.

Tanda \* diapit oleh dua operand yaitu B dan C yaitu B \* C , prefixnya dengan menggabungkan operand dan memindahkan operator kedepan dari operand, sehingga fungsi B \* C, notasi prefixnya menjadi \*BC. Sehingga hasil sementara dari notasi prefix adalah

A + \*BC

Selanjutnya mencari prefix untuk operator yang berikutnya, yaitu +, cara yang dilakukan sama seperti di atas, operator +, diapit oleh 2 operand, yaitu A dan \*BC, gabungkan operand, sehingga menjadi A\*BC, lalu pindahkan operator kedepan operand, sehingga hasil akhir menjadi

+ A \* B C

Contoh yang lain :

1. A + B – C \* D

2 3 1 —–> hirarkhi level

A + B – \*CD —–> 1

+ AB – \*CD —–> 2

– +AB \*CD —–> 3

2. A \* B ^ C – D

2 1 3 —–> hirarkhi

A \* ^BC – D —–> 1

\*A^BC – D —–> 2

-\*A^BCD —–> 3

3. A + ( B – C ) \* D

3 1 2 —–> hirarkhi

A + -BC \* D —–> 1 (karena diapit tanda paranthesis atau kurung buka/tutup, ( ) )

A + \*-BCD —–> 2

+ A \*-BCD —–> 3

2. Infix

Yaitu notasi yang terbentuk atas operator dengan operand, dimana operator berada diantara operand. Notasi ini hanya dikenal oleh manusia dan selalu digunakan dalam perhitungan aritmatika.

Contoh : A + B \* C

( A + B ) \* C

A – ( B + C ) \* D ^ E

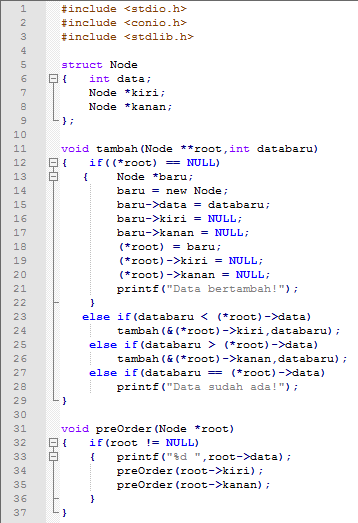
3. Postfix

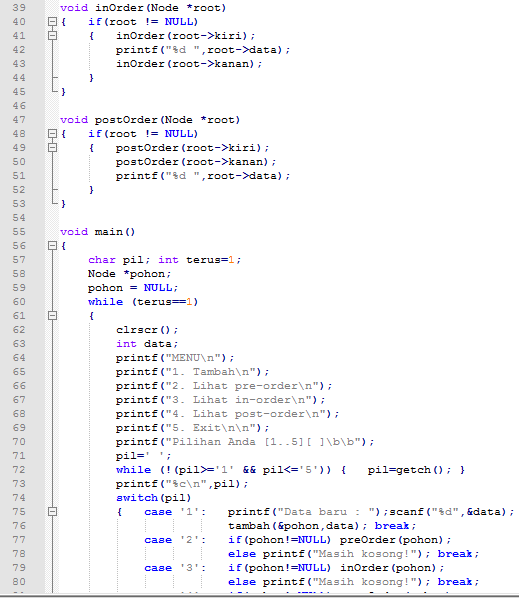
Yaitu notasi yang terbentuk atas operator dengan operand, dimana operator berada dibelakang operand. Notasi ini hanya dikenal oleh processor dan dipahami dalam ALU.

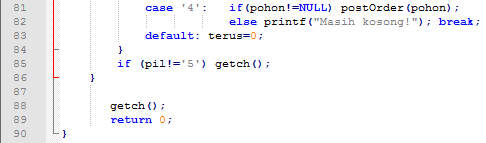
Contoh : A + B \* C (Infix). Maka notasi postfixnya adalah ABC\*+

CONTOH PROGRAM (*TREE*)

Contoh 1:







**LOGIKA PROGRAM**

Pada pemrograman kali ini menggunakan 3 header yaitu **<stdio.h>**,**<conio.h>**, dan **<stdlib.h>**. lalu terdapat **struct Node{**berfungsi sebagai pendeklarasian sebuah struct yang diberi nama Node, yang berisi pendeklarasian variabel data yang bertipe data integer. Lalu ada **Node \*kiri;** \* adalah sebuah pointer. Lalu ada juga **Node \*kanan; }**

Lalu masuk ke fungsi **void tambah** yang berparameter **(Node \*\*root, int databaru)** lalu ada percabangan if dengan kondisi **if((\*root) == NULL){** ketika root tidak diberi nilai, maka **Node \*baru;**membuat pointer baru yang diberi nama “baru”,**baru = new Node;**nilai dari baru sama dengan Node baru,**baru ->data = databaru;**ketika pointer baru pindah ke pointer data maka nilainya databaru,**baru->kiri = NULL;**jika pointer baru menuju ke pointer kiri maka nilainya tidak boleh kosong.**baru->kanan=NULL;**ketika nilai pointer baru menuju ke pointer kanan maka nilainya tidak boleh kosong. **(\*root) = baru;**niali dari pointer root sama dengan baru. **(\*root)->kiri = NULL;**jika pointer root menuju kiri maka nilai tidak boleh kosong, begitu juga dengan **(\*root)->kanan=NULL;.**Lalu perintah cetak kalimat “Data bertambah”. Kemudian ada kondisi **else if(databaru < (\*root)->data)** ketika nilai dari databaru lebih kecil dari pointer \*root menuju data, maka program akan melakukan **tambah(&(\*root)->kiri, databaru);**dan **else if(databaru > (\*root)->data);**ketika nilai dari databaru lebih besar dari pointer root menuju data, program akan melakukan **tambah(&(\*root)->kanan, databaru);**dan **else if(databaru == (\*root)->data)** ketika nilai dari databaru sama dengan nilai dari pointer \*root menuju data maka program akan mencetak “Data Sudah Ada!”

Fungsi ke-2 kali ini adalah **void preOrder(Node \*root)**pada fungsi kali ini, bertujuan untuk mengatur posisi data sesuai dengan aturan preOrder. **If(root != NULL){** ketika nilai dari root tidak sama dengan NULL, maka program akan menjalankan perintah cetak (**“%d”, root->data);**jadi pertama data akan dicetak terlebih dahulu, **preOrder(root->kiri);**lalu data akan dipindahkan ke kiri ,**preOrder(root->kanan);**yang pada akhirnya data akan dipindahkan dikanan.

Penjelasan yang sama seperti pada **preOrder** yang membedakannya ialah, susunan letak dari pointer-pointer itu sendiri.

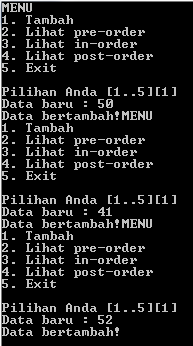
Ini juga sama seperti **preOrder** yang membedakannya ialah letak dari susunan pointer.

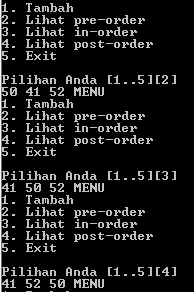
Fungsi utama yaitu **void main(){**didalamnya terdapat pendeklarasian dari variabel pil yang bertipe data char/string, dan variabel terus yang sudah diberi nilai yaitu 1 dan bertipe data integer. Lalu terdapat perulangan menggunakan while dengan kondisi **while(terus==1){** ketika nilai dari variabel terus ==1, maka lakukan. **Clrscr();** berfungsi untuk membersihkan layar ketika digunakan kembali. Lalu ada pendeklarasian variabel data yang bertipe data integer. Lalu cetak pilihan-pilihan yang terdapat pada menu ini. Lalu perintah input untuk memilih menu yang ingin dipilih. Lalu ada perulangan while kembali dengan kondisi **while(!(pil>=’1’ && pil <=’5’)) {pil=getch();}**jadi ketika pil lebih dari 1 dan pil kurang dari 5 maka lakukan **print(“%c\n”,pil);** lalu masuk ke switch case.

Pada **case 1:**program akan memberi inputan dengan kalimat “Data Baru: “, lalu program akan memanggil **tambah(&pohon.data); break;** pada **case 2:**ada percabangan if yang berkondisi **if(pohon!=NULL) preOrder(pohon);** ketika nilai dari pohon tidak sama dengan NULL, maka program akan memanggil fungsi **preOrder(pohon);** dan **else** cetak “Masih kosong” menandakan bahwa data belum ada isinya.**Case 3: if(pohon!=NULL) inOrder(pohon);**ketika nilai dari pohon tidak sama dengan NULL atau tidak kosong, maka program akan memanggil fungsi **inOrder(pohon);** **else**program akan mencetak “Masih Kosong”. **Case 4: if(pohon!=NULL) postOrder(pohon);** nilai dari pohon tidak sama dengan NULL atau tidak kosong, maka program akan memanggil **postOrder(pohon);** **else** cetak “Masih Kosong”. Kondisi defaultnya adalah terus==0, agar program berhenti.

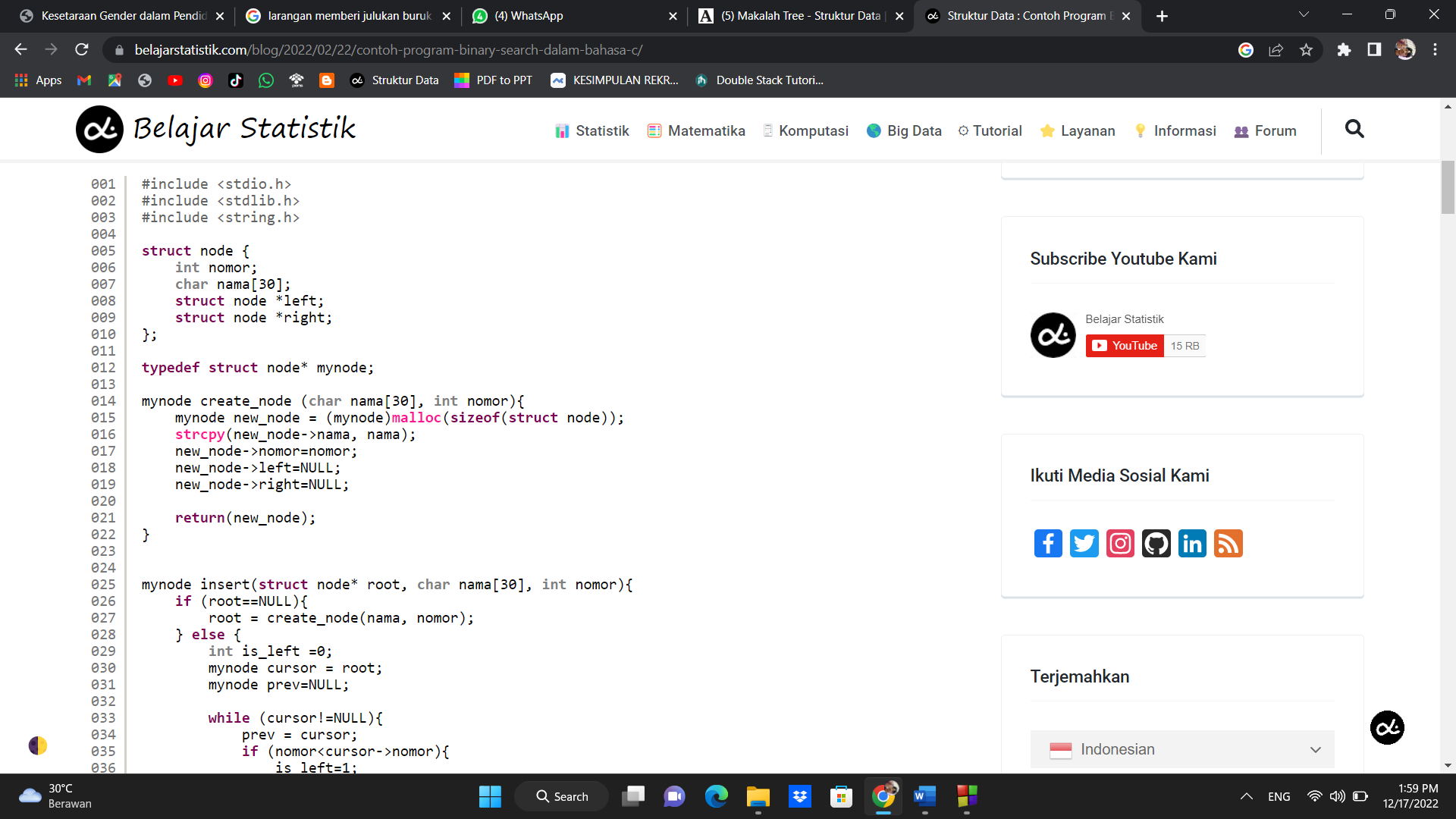
Kemudian ada percabangan if dengan kondisi **if(pil!=’5’) getch();** jadi ketika nilai dari pil tidak sama dengan 5 maka, program akan menjalankan perintah **getch();**dan yang terakhir **getch(); return 0;**

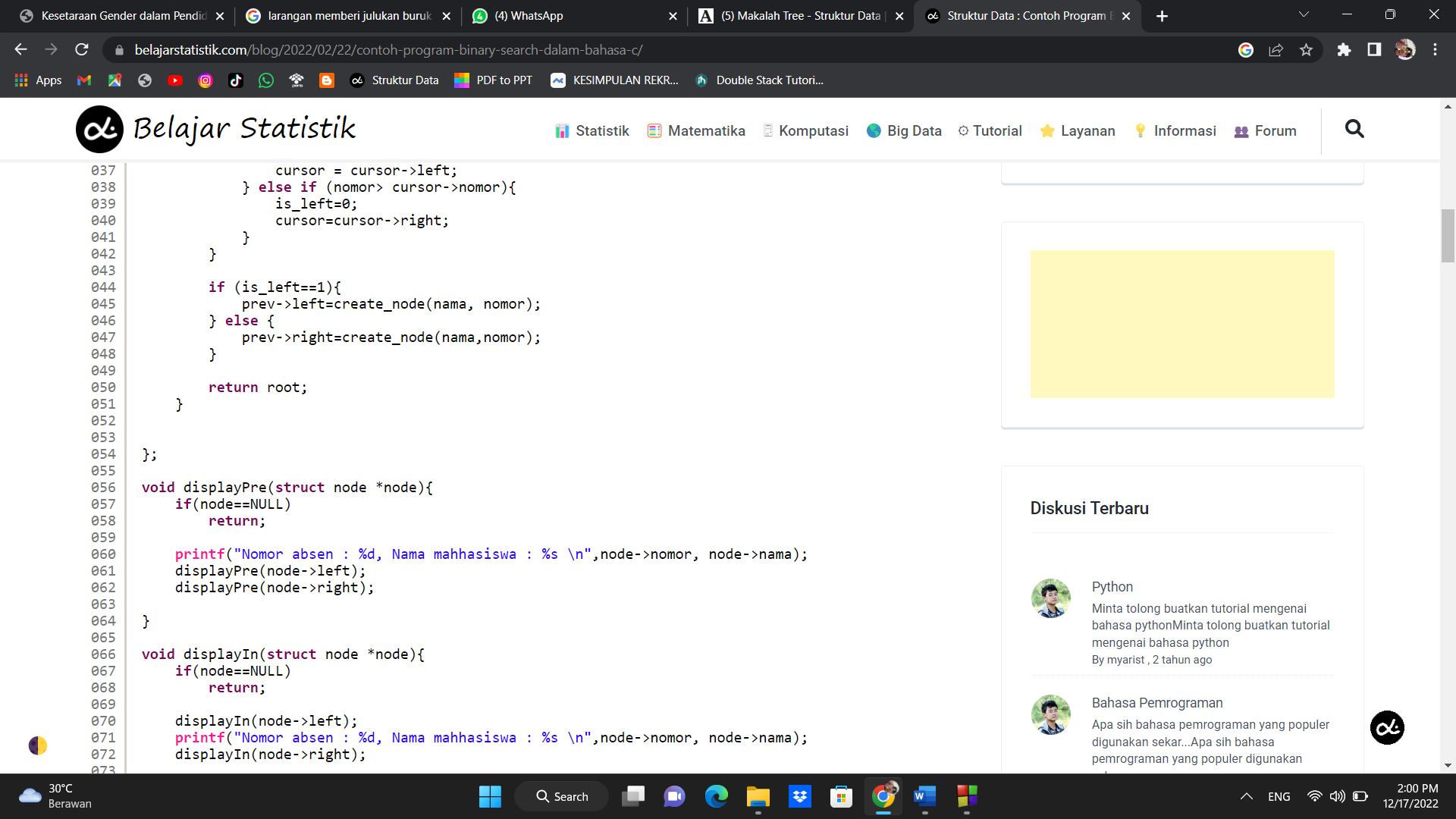
OUTPUT

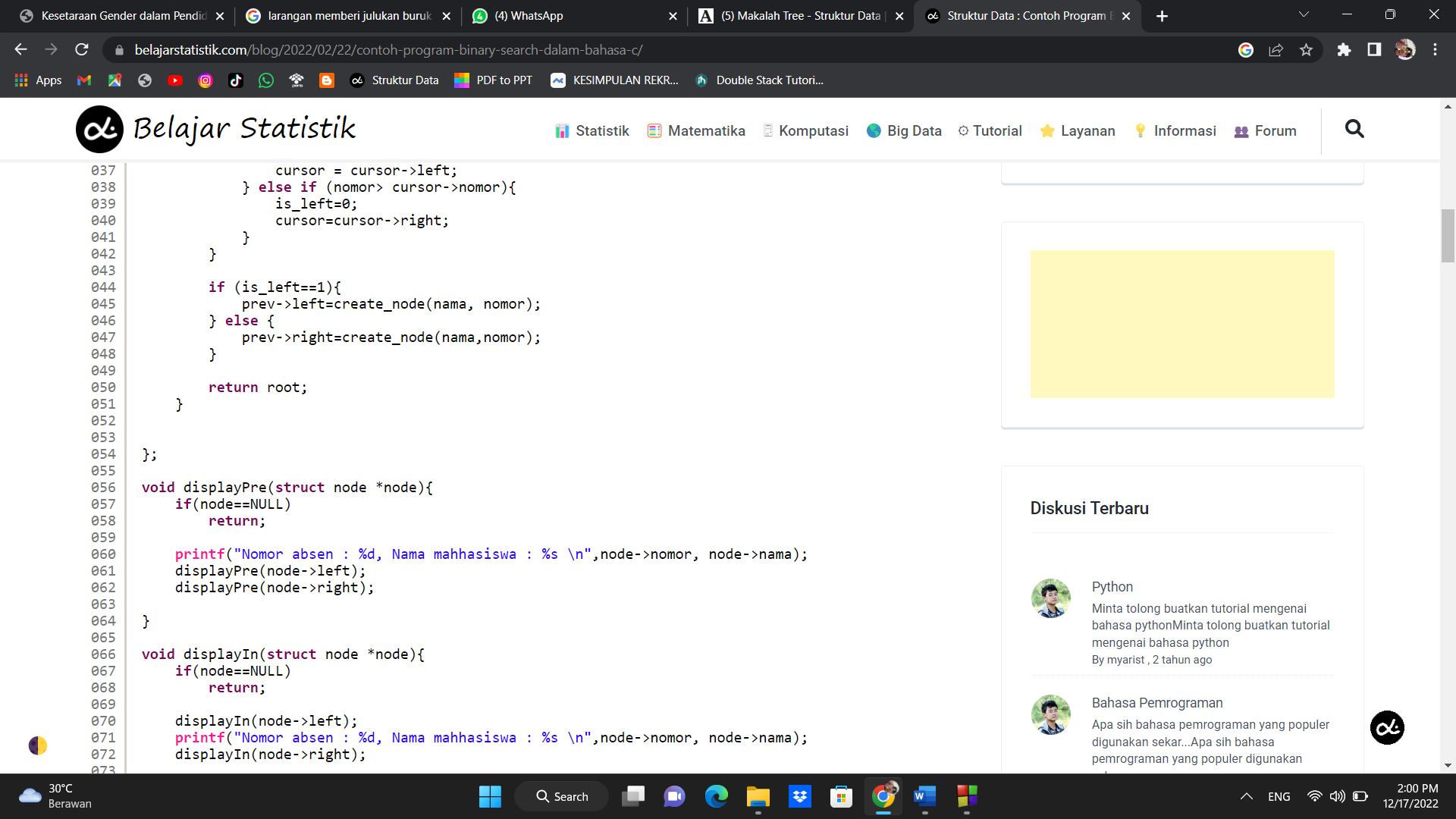


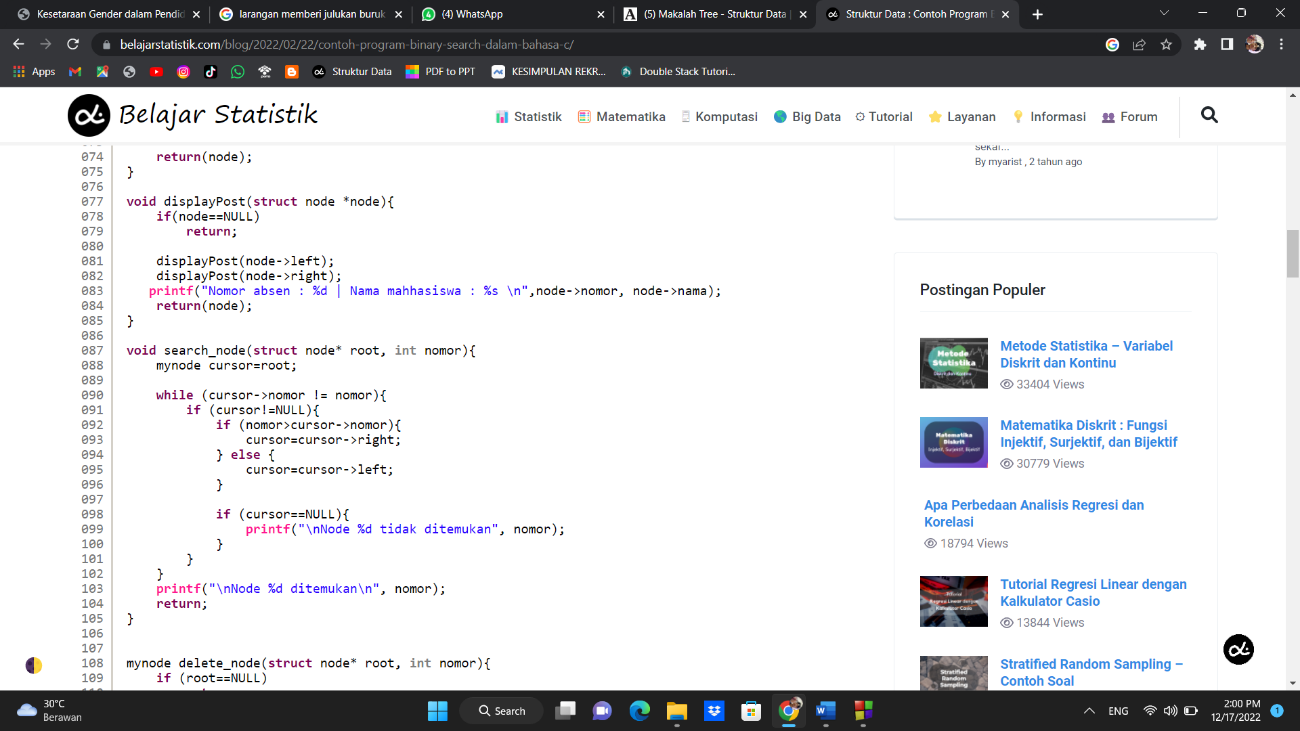


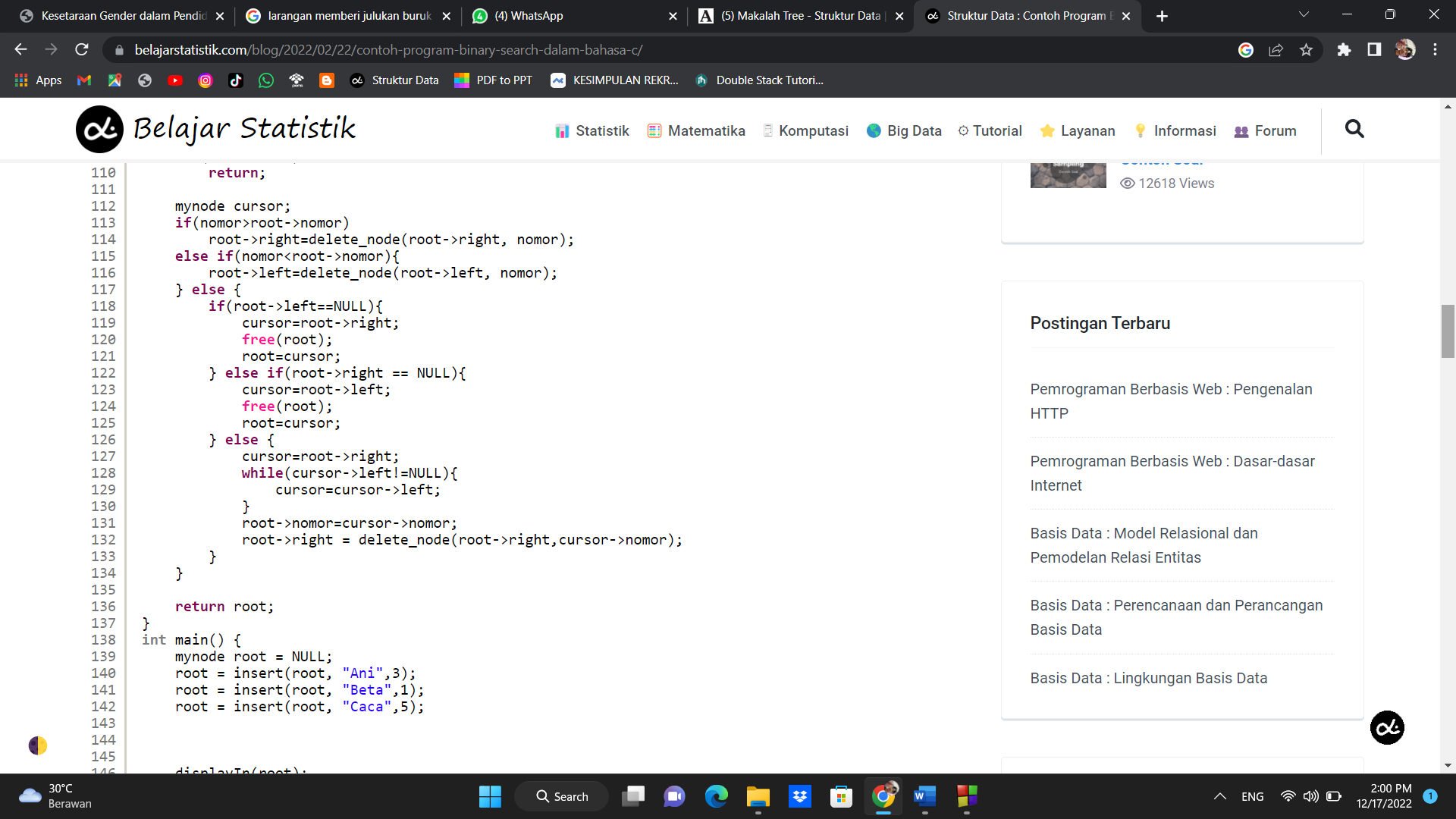
CONTOH 2:

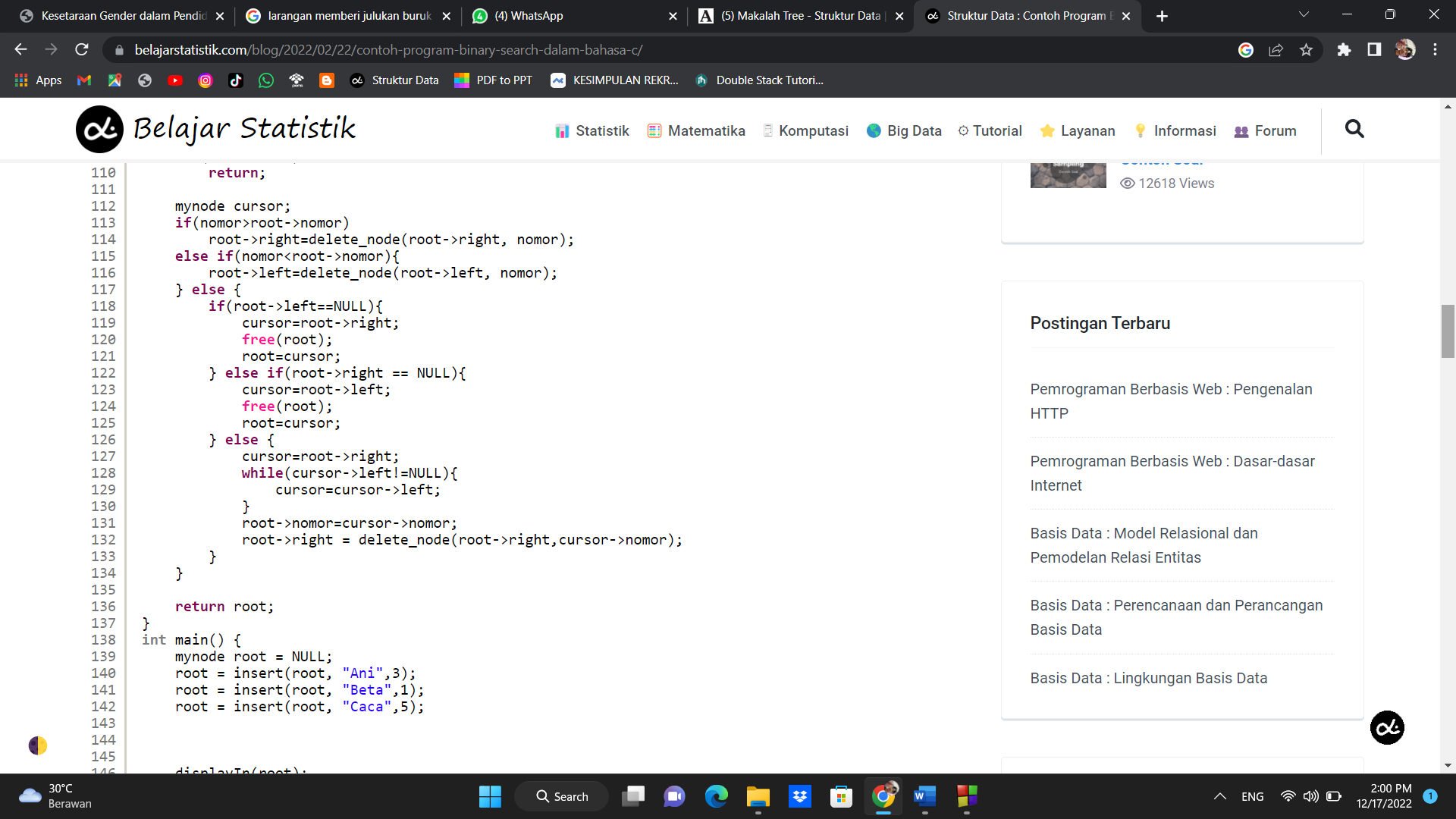


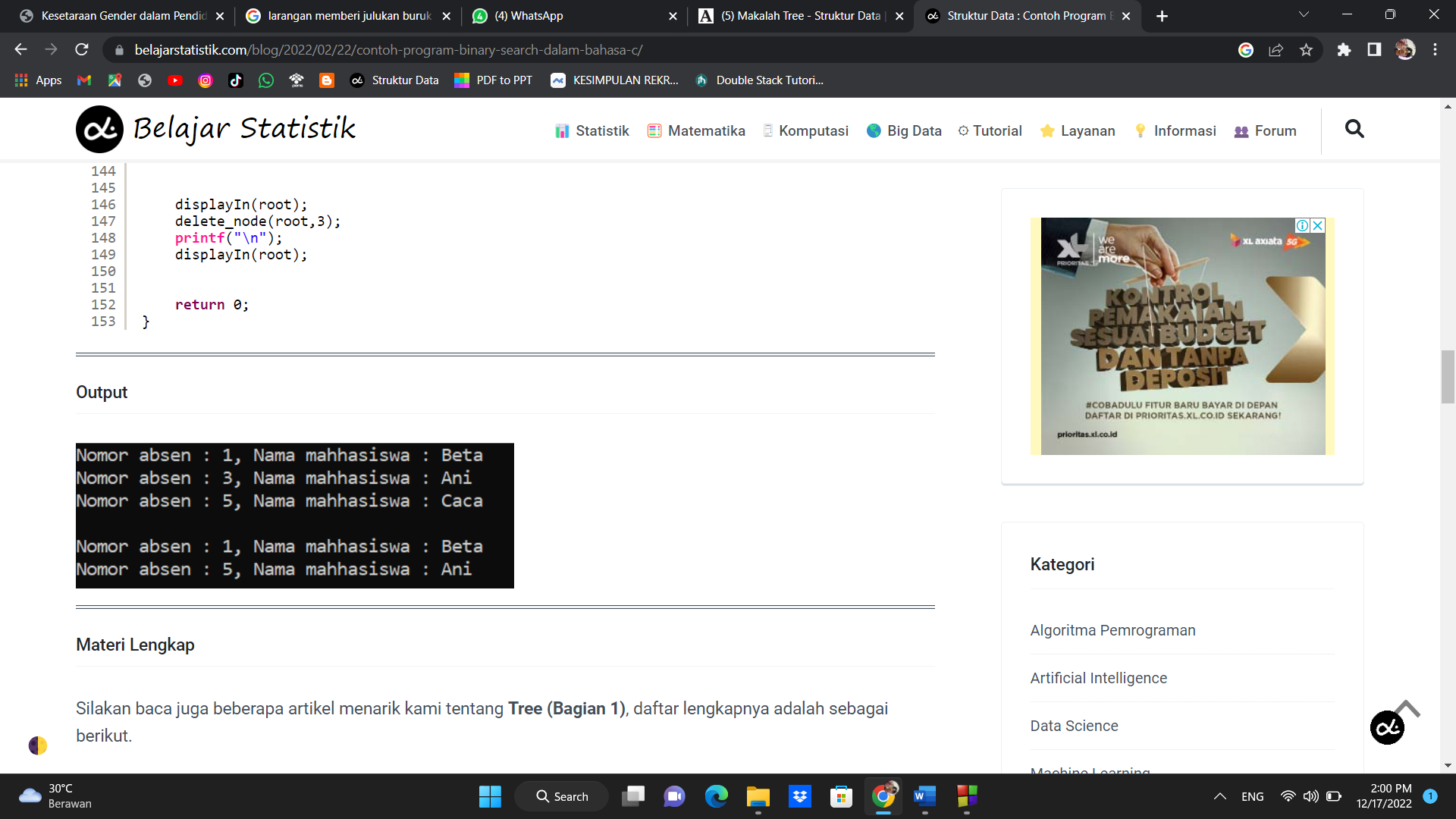




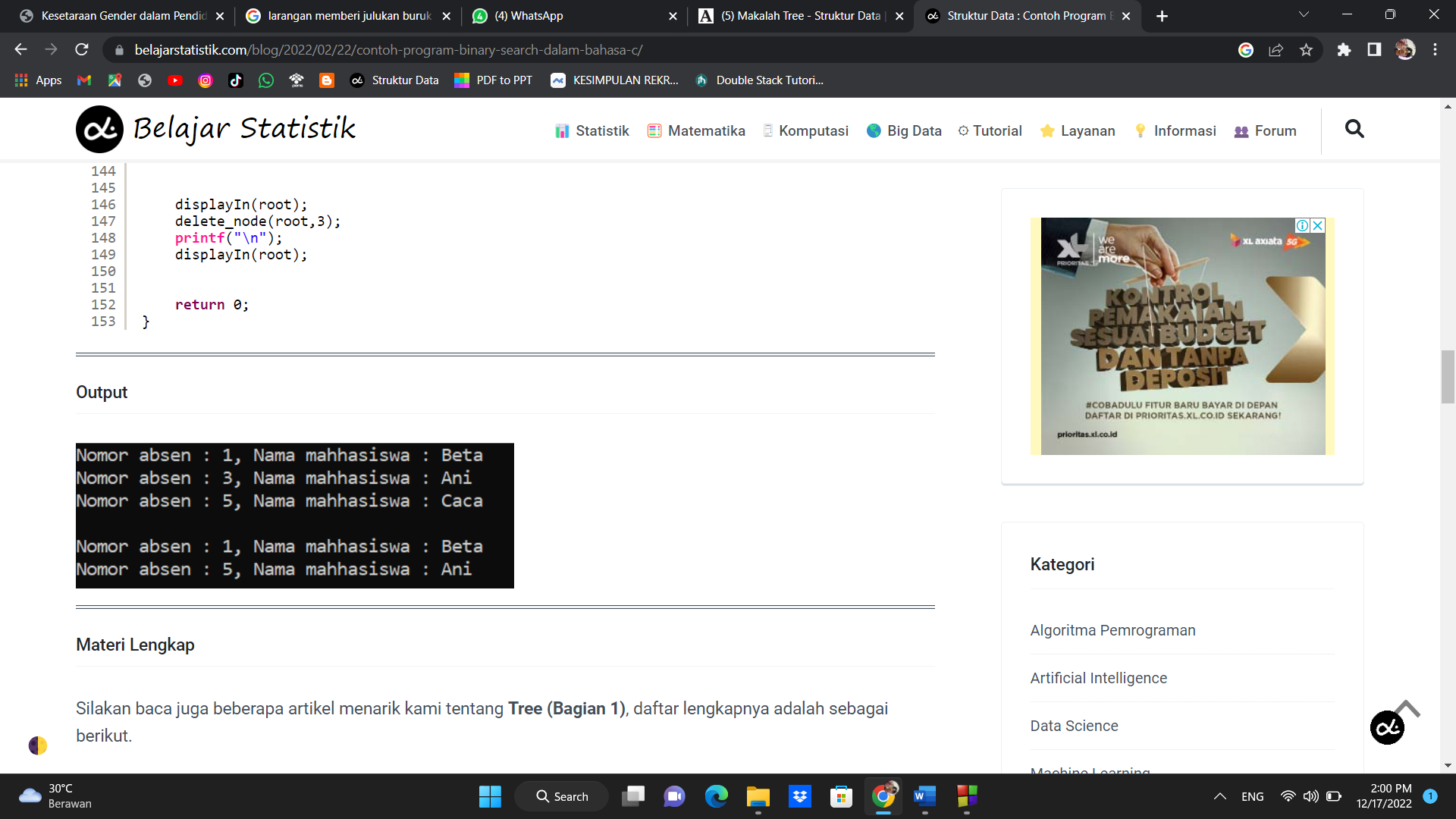








OUTPUT :



CONTOH 3:

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

#include <iostream>

using namespace std;

//pendeklarasian struct sebuah tree awal

typedef struct Node

{

    int data;

    Node \*kiri;

    Node \*kanan;

};

Node \*pohon = NULL;

//fungsi untuk menambahkan node baru

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

{

       //jika root masih kosong

    if ((\*root)==NULL)

       {

              //pembuatan node baru

        Node \*baru;

        //pengalokasian memori dari node yang telah dibuat

        baru = new Node;

        //inisialisasi awal node yang baru dibuat

        baru->data=databaru;

        baru->kiri=NULL;

        baru->kanan=NULL;

        (\*root) = baru;

        (\*root) -> kiri = NULL;

        (\*root) -> kanan = NULL;

        //jika menujuk menunjuk ke NULL artinya tidak mempunyai child

        printf("Data Bertambah!");

       }

    //jika data yang akan dimasukkan lebih kecil daripada elemen root, maka akan diletakkan di node sebelah kiri.

    else if (databaru<(\*root)->data)

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

    //jika data yang akan dimasukkan lebih besar daripada elemen root, maka akan diletakkan di node sebelah kanan

    else if (databaru>(\*root)->data)

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

    //jika saat dicek data yang akan dimasukkan memiliki nilai yang sama dengan data pada root

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

        printf("Data Sudah ada!");

}

//fungsi yang digunakan untuk mencetak tree secara preOrder

void preOrder(Node \*root)

{

    if(root!=NULL)

    {

        if(root->data!=NULL)

        {

            printf("%d ",root->data);

        }

        preOrder(root->kiri);

        preOrder(root->kanan);

    }

}

//fungsi yang digunakan untuk mencetak tree secara inOrder

void inOrder(Node \*root)

{

       if(root!=NULL)

       {

           inOrder(root->kiri);

           if(root->data!=NULL)

           {

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

           }

           inOrder(root->kanan);

        }

}

//fungsi yang digunakan untuk mencetak tree secara postOrder

void postOrder(Node \*root)

{

       if(root!=NULL)

       {

         postOrder(root->kiri);

         postOrder(root->kanan);

         if(root->data!=NULL)

         {

            printf("%d ",root->data);

         }

       }

}

//fungsi yang digunakan untuk melakukan pencarian data

void search(Node \*\*root, int cari)

{

    if((\*root) == NULL)

    {

        printf("Data tidak ditemukan!");

    }

    //jika data yang dicari lebih kecil dari isi root

    else if(cari < (\*root)->data)

        search(&(\*root)->kiri,cari);

    //jika data yang dicari lebih besar dari isi root

    else if(cari > (\*root)->data)

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

    //jika data yang dicari sama dengan isi dari variabel root

    else if(cari == (\*root)->data)

        printf("Data ditemukan!");

}

//fungsi untuk mengetahui jmlah node dalam tree

int count(Node \*root)

{

       if(root==NULL)

              return 0;

       else

              return count(root->kiri)+ count(root->kanan)+1;

}

//Fungsi untuk mengetahui ketinggian/kedalaman

int height(Node \*root)

{

       if(root == NULL)

              return -1;

       else{

              int u = height(root->kiri);

        int v = height(root->kanan);

        if(u > v)

          return u + 1;

        else

          return v + 1;

       }

}

//fungsi yang digunakan untuk menghapus suatu node

void hapus(Node \*\*root, int del)

{

       Node \*curr;

       Node \*parent;

       curr = (\*root);

       bool flag = false;

       while(curr != NULL)

       {

              if(curr->data == del)

              {

                     flag = true;

                     //cout<<"Data ditemukan!";

                      break;

              }

              else

              {

                  parent = curr;

                  if(del>curr->data)

                     curr = curr->kanan;

                  else

                     curr = curr->kiri;

              }

       }

    if(!flag)

    {

        cout<<"Data tidak ditemukan. Penghapusan tidak dilakukan."<<endl;

        return;

    }

       //hanya satu tingkat, dengan kata lain hanya terdapat root

       //jika hanya terdapat root, maka curr node tidak punya parent

       if(height(pohon) == 0)

       {

              if( curr->kiri== NULL && curr->kanan == NULL)

              {

                     (\*root) = NULL;

                     return;

              }

       }

       //lebih dari satu tingkat, sehingga node curr mempunyai parent

       else if(height(pohon) > 0)

       {

              //1. jika node yang dihapus tidak memiliki anak

              if( curr->kiri== NULL && curr->kanan == NULL)

              {

                     //jika node merupakan anak kiri dari parent

                     if(parent->kiri == curr)

                     {

                           //replace parent->kiri dengan NULL

                           parent->kiri = NULL;

                           delete curr;

                     }

                     else //jika node merupakan anak kanan dari parent

                     {

                           //replace parent->kanan dengan NULL

                           parent->kanan = NULL;

                           delete curr;

                     }

                     return;

              }

              //2. Jika node memiliki anak tunggal (anak kiri/anak kanan)

              if((curr->kiri == NULL && curr->kanan != NULL)|| (curr->kiri != NULL && curr->kanan == NULL))

              {

                     //jika curr memiliki anak tunggal di sebelah kanan

                     if(curr->kiri == NULL && curr->kanan != NULL)

                     {

                           //jika curr(data yang ingin dihapus) merupakan anak kiri dari parent

                           if(parent->kiri == curr)

                           {

                              //ganti isi parent->kiri dengan curr->kanan

                              parent->kiri = curr->kanan;

                              delete curr;

                         }

                         else //jika curr(data yang ingin dihapus) bukan merupakan anak kiri dari parent

                         {

                              //ganti isi parent->kanan dengan curr->kanan

                              parent->kanan = curr->kanan;

                              delete curr;

                         }

                     }

                  else //jika curr memiliki anak tunggal di sebelah kiri

                  {

                           //jika curr(data yang ingin dihapus) merupakan anak kiri dari parent

                           if(parent->kiri == curr)

                           {

                                  parent->kiri = curr->kiri; //ganti isi parent->kiri dengan curr->kiri

                                  delete curr;

                           }

                           else //jika curr(data yang ingin dihapus) bukan merupakan anak kiri dari parent

                           {

                               parent->kanan = curr->kiri; //ganti isi parent->kanan dengan curr->kiri

                               delete curr;

                            }

                   }

                   return;

              }

              //3. Node dengan dua anak

              //ganti node dengan nilai terkecil dari Sub Tree Kanan

              if (curr->kiri != NULL && curr->kanan != NULL)

              {

                     //variabel bantu ini digunakan agar posisi curr asli tidak berubah, (tetap pada posisi node yang akan dihapus)

                     //variabel bantu digunakan untuk mengarah ke suatu node

                     Node\* bantu;

                     bantu = curr->kanan;

                     //jika subtree kanan dari posisi node sekarang (curr, node yang akan dihapus) tidak memiliki anak

                     if((bantu->kiri == NULL) && (bantu->kanan == NULL))

                     {

                           //ganti node curr dengan bantu

                           // sama dengan curr = (curr->kanan)->kanan |||| semoga tidak bingung :D

                           curr = bantu;

                           delete bantu;

                           curr->kanan = NULL;

                     }

                     else //jika child/anak kanan dari node curr memiliki child

                     {

                           //jika node child kanan dari curr memiliki child kiri

                           if((curr->kanan)->kiri != NULL)

                           {

                                  //variabel bantu ini digunakan agar posisi curr asli tidak berubah, (tetap pada posisi node yang akan dihapus)

                                  //variabel bantu digunakan untuk mengarah ke suatu node

                                  Node\* bantu2;

                                  Node\* bantu3; //berlaku sebagai parent dari bantu 2

                                  bantu3 = curr->kanan;         //!perhatikan

                                  bantu2 = (curr->kanan)->kiri; //!perhatikan

                                  //mengarahkan posisi node ke node terkiri (untuk menuju ke node yang memiliki nilai terkecil)

                                  while(bantu2->kiri != NULL)

                                  {

                                         bantu3 = bantu2;

                                         bantu2 = bantu2->kiri;

                                  }

                                  //replace nilai dari node curr dengan nilai dari node bantu

                                  curr->data = bantu2->data;

                                  delete bantu2;

                                  bantu3->kiri = NULL;

                        }

                        else //jika node child kanan dari curr tidak memiliki child kiri

                        {

                              Node\* tmp;

                              tmp = curr->kanan;

                              //replace nilai dari node curr dengan nilai dari node tmp (curr->kanan)

                              curr->data = tmp->data;

                              curr->kanan = tmp->kanan;

                              delete tmp;

                        }

                     }

                     return;

              }

       }

}

//fungsi utama

int main()

{

    //deklarasikan variabel

    char pil;

    int del,cari;

    while (true)

    {

        system("cls"); //bersihkan layar

        char data;

        printf("\t#PROGRAM TREE C++#");

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

        printf("\nMENU");

        printf("\n----\n");

        printf("[1] Tambah Data\n");

        printf("[2] Lihat Pre-Order\n");

        printf("[3] Lihat In-Order\n");

        printf("[4] Lihat Post-Order\n");

        printf("[5] Delete\n");

        printf("[6] Kosongkan Data\n");

        printf("[7] Search\n");

        printf("[8] Hitung Node pada Tree\n");

        printf("[9] Kedalaman Tree\n");

        printf("[X] Keluar\n");

        printf("Pilihan Anda : ");

        scanf("%c",&pil);

        fflush(stdin); //mengosongkan buffering

        switch(pil)

        {

            //jika pil bernilai '1'

            case '1':

                     printf("\nINPUT : ");

                     printf("\n-------");

                     printf("\nMasukkan data: ");

                     scanf("%d", &data);

                     //panggil fungsi untuk menambah node yang berisi data pada tree

                     tambah(&pohon,data);

                     \_getch();

                      break;

             //jika pil bernilai '2'

              case '2':

                     printf("\nOUTPUT PRE ORDER : ");

                     printf("\n------------------\n");

                     if(pohon!=NULL)

                           //panggil fungsi untuk mencetak data secara preOrder

                           preOrder(pohon);

                     else

                           printf("Masih Kosong!!!");

                     \_getch();

                     break;

              //jika pil bernilai '3'

              case '3':

                     printf("\nOUTPUT IN ORDER : ");

                     printf("\n------------------\n");

                     if(pohon!=NULL)

                           //panggil fungsi untuk mencetak data secara inOrder

                           inOrder(pohon);

                     else

                           printf("Masih Kosong!!!");

                     \_getch();

                     break;

             //jika pil bernilai '4'

              case '4':

                     printf("\nOUTPUT POST ORDER : ");

                     printf("\n------------------\n");

                     if(pohon!=NULL)

                           //panggil fungsi untuk mencetak data secara postOrder

                           postOrder(pohon);

                     else

                           printf("Masih Kosong!!!");

                     \_getch();

                     break;

              //jika pil bernilai '5'

              case '5':

                     if(pohon!=NULL)

                     {

                           printf("\nSEBELUM NODE DIHAPUS : ");

                           printf("\n---------------------- ");

                           //panggil fungsi untuk mencetak data secara preOrder

                           printf("\nPRE ORDER  : ");

                           preOrder(pohon); //panggil fungsi

                           printf("\nIN ORDER   : ");

                           inOrder(pohon); //panggil fungsi

                           printf("\nPOST ORDER : ");

                           postOrder(pohon); //panggil fungsi

                           printf("\n\nPENGHAPUSAN DATA : ");

                           printf("\n------------------\n");

                           printf("Masukkan data yang ingin dihapus: ");

                           scanf("%d", &del);

                          //panggil fungsi yang digunakan untuk melakukan penghapusan pada suatu node

                           hapus(&pohon, del);

                           printf("\n\nSETELAH NODE DIHAPUS : ");

                           printf("\n---------------------- ");

                           printf("\nPRE ORDER  : ");

                           preOrder(pohon); //panggil fungsi

                           printf("\nIN ORDER   : ");

                           inOrder(pohon); //panggil fungsi

                           printf("\nPOST ORDER : ");

                           postOrder(pohon); //panggil fungsi

                     }

                     else

                           printf("\nMasih kosong!");

                     \_getch();

                     break;

             //jika pil bernilai '6'

              case '6':

                     pohon=NULL;

                     printf("\nPENGOSONGAN ELEMEN ");

                     printf("\n------------------");

                     printf("\nTree sudah dikosongkan!!");

                     \_getch();

                     break;

              //jika pil bernilai '7'

              case '7':

                     printf("\nOUTPUT -> Hanya untuk mengecek apakah data dimaksud terdapat dalam tree");

                     printf("\n------");

                     if(pohon!=NULL)

                     {

                           //panggil fungsi untuk mencetak data secara   preOrder

                           printf("\nPRE ORDER  : ");

                           preOrder(pohon); //panggil fungsi

                           printf("\nIN ORDER   : ");

                           inOrder(pohon); //panggil fungsi

                           printf("\nPOST ORDER : ");

                           postOrder(pohon); //panggil fungsi

                           printf("\n\nPENCARIAN DATA");

                           printf("\n--------------");

                           printf("\nMasukkan data yang ingin dicari : ");

                           scanf("%d", &cari);

                           //panggil fungsi untuk melakukan pencarian data pada tree

                           search(&pohon, cari);

                     }

                     else

                           printf("\nMasih kosong!");

                     \_getch();

                     break;

              //jika pil bernilai '8'

              case '8':

                     printf("\nJUMLAH NODE DI DALAM TREE");

                     printf("\n-------------------------");

                     printf("\nJumlah Node :  %d", count(pohon));

                     \_getch();

                     break;

              //jika pil bernilai '9'

              case '9' :

                     printf("\nKEDALAMAN TREE(LEVEL-> DIMULAI DARI 0)");

                     printf("\n----------------------------------------");

                     printf("\nKedalaman tree : %d\n", height(pohon));

                     \_getch();

                     break;

              //jika pil bernilai 'X' atau 'x'

              case 'X'|'x':

                     exit(0);

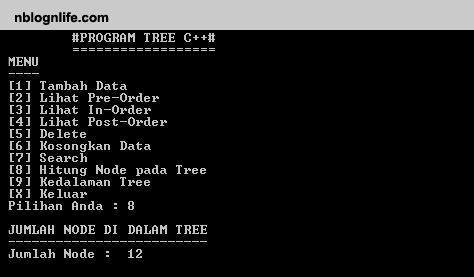
                     break;

              }

       }

}

Output :



**BAB III**

**PENUTUP**

**3.1. KESIMPULAN**

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.

**3.2. SARAN**

Makalah ini dibuat agar mahasiswa mengerti definisi dan aplikasi pohon biner dalam program C.

DAFTAR PUSTAKA

* <https://www.nblognlife.com/2014/12/binary-search-tree-bst-tree-lanjutan.html>
* <https://www.belajarstatistik.com/blog/2022/02/22/contoh-program-binary-search-dalam-bahasa-c/>
* <https://www.belajarstatistik.com/blog/2022/02/19/binary-search-tree/>
* <https://www.nblognlife.com/2014/12/binary-search-tree-bst-tree-lanjutan.html>
* <https://abdilahrf.github.io/2015/06/pengenalan-binary-search-tree/>
* <https://docplayer.info/115229333-Implementasi-binary-search-tree-pada-struktur-data-priority-queue-dalam-bahasa-c.html>
* <https://socs.binus.ac.id/2017/05/10/implementasi-insert-pada-binary-search-tree-dengan-single-dan-double-pointer/>
* <https://github.com/heryvandoro/data-structure-sample/blob/master/BinarySearchTree.cpp>
* <https://adlymuhshidieq.wordpress.com/2017/03/02/581/>