LAPORAN PRAKTIKUM STRUKTUR DATA DAN ALGORITMA

MODUL IX
GRAPH DAN TREE

Disusun Oleh :

NAMA : FAISAL KHOIRUDDIN

NIM : 2311102046

Dosen

WAHYU ANDI SAPUTRA, S.Pd., M.Eng.

PROGRAM STUDI S1 TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

INSTITUT TEKNOLOGI TELKOM PURWOKERTO

2024

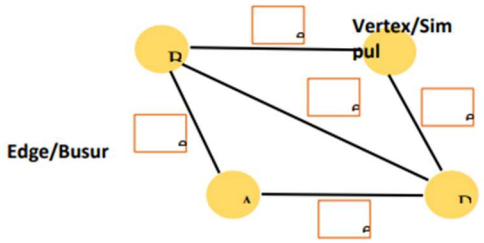
A. Dasar Teori

a. Graph

Graf atau graph adalah struktur data yang digunakan untuk merepresentasikan hubungan antara objek dalam bentuk node atau vertex dan sambungan antara node tersebut dalam bentuk sisi atau edge. Graf terdiri dari simpul dan busur yang secara matematis dinyatakan sebagai :

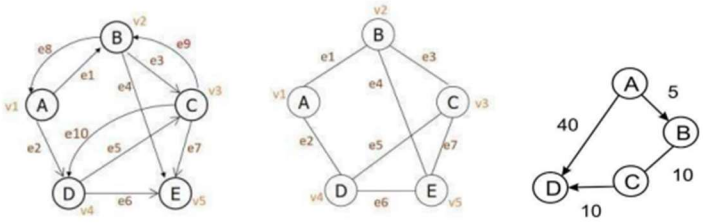
$$G = (V, E)$$

Dimana G adalah Graph, V adalah simpul atau vertex dan E sebagai sisi atau edge. Dapat digambarkan:



Graph dapat digunakan dalam berbagai aplikasi, seperti jaringan sosial, pemetaan jalan, dan pemodelan data.

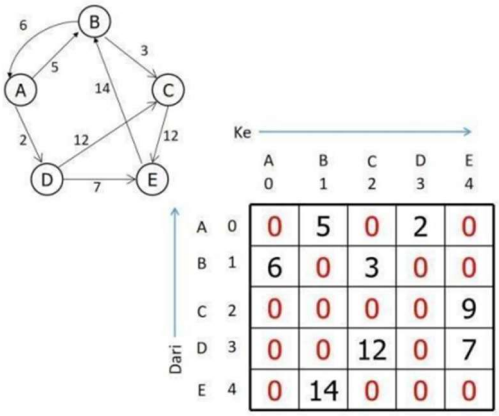
Jenis-jenis Graph



1. Graph berarah (directed graph): Urutan simpul mempunyai arti. Misal busur AB adalah e1 sedangkan busur BA adalah e8.
2. Graph tak berarah (undirected graph): Urutan simpul dalam sebuah busur tidak diperhatikan. Misal busur e1 dapat disebut busur AB atau BA.

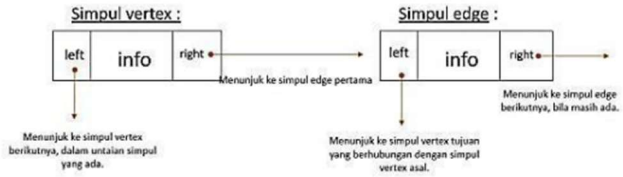
3. Weight Graph : Graph yang mempunyai nilai pada tiap edgenya.

Representasi Graph dengan Matriks

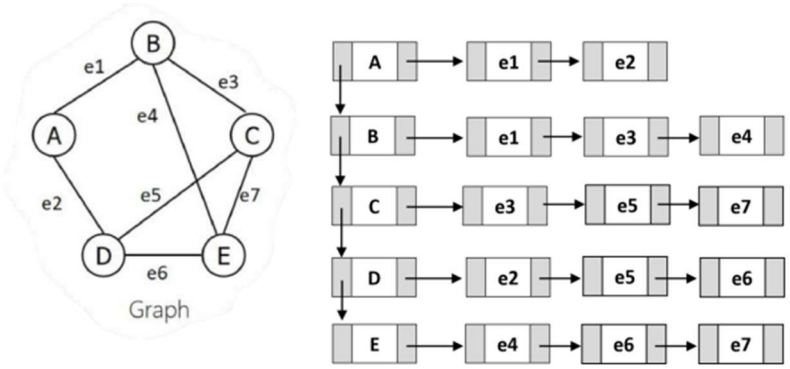


Representasi Graph dengan Matriks

Representasi dengan Linked List



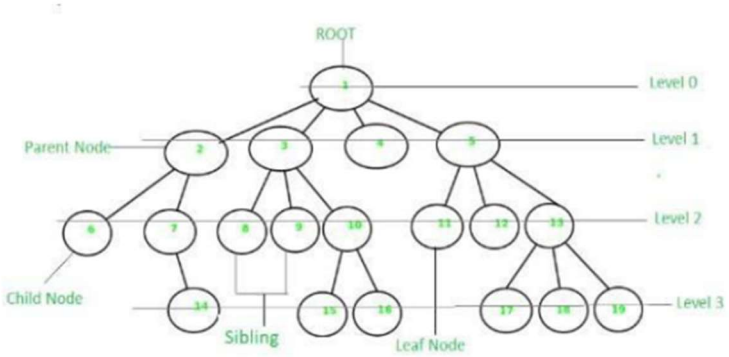
Pentingnya untuk memahami perbedaan antara simpul vertex dan simpul edge saat membuat representasi graf dalam bentuk linked list. Simpul vertex mewakili titik atau simpul dalam graf, sementara simpul edge mewakili hubungan antara simpul-simpul tersebut. Struktur keduanya bisa sama atau berbeda tergantung pada kebutuhan, namun biasanya seragam. Perbedaan antara simpul vertex dan simpul edge adalah bagaimana kita memperlakukan dan menggunakan keduanya dalam representasi graf.



Representasi Graph dengan Linked List

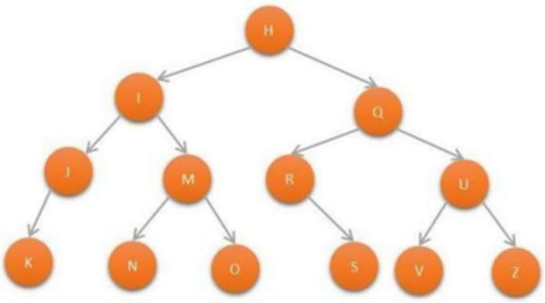
b. Tree atau Pohon

Dalam ilmu komputer, pohon/tree adalah struktur data yang sangat umum dan kuat yang menyerupai nyata pohon. Ini terdiri dari satu set node tertaut yang terurut dalam grafik yang terhubung, dimana setiap node memiliki paling banyak satu simpul induk, dan nol atau lebih simpul anak dengan urutan tertentu. Struktur data tree digunakan untuk menyimpan data-data hirarki seperti pohon keluarga, skema pertandingan, struktur organisasi. Istilah dalam struktur data tree dapat dirangkum sebagai berikut :



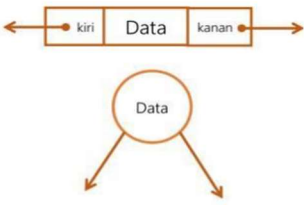
Predecessor	Node yang berada di atas node tertentu
Successor	Node yang berada di bawah node tertentu
Ancestor	Seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama
Descendent	Seluruh node yang terletak setelah node tertentu dan terletak pada jalur yang sama
Parent	Predecessor satu level di atas suatu node
Child	Successor satu level di bawah suatu node
Sibling	Node-node yang memiliki parent yang sama
Subtree	Suatu node beserta descendent-nya
Size	Banyaknya node dalam suatu tree
Height	Banyaknya tingkatan/level dalam suatu tree
Roof	Node khusus yang tidak memiliki predecessor
Leaf	Node-node dalam tree yang tidak memiliki successor
Degree	Banyaknya child dalam suatu node

Binary tree atau pohon biner merupakan struktur data pohon akan tetapi setiap simpul dalam pohon diprasyaratkan memiliki simpul satu level di bawahnya (child) tidak lebih dari 2 simpul, artinya jumlah child yang diperbolehkan yakni 0, 1, dan 2. Gambar 1, menunjukkan contoh dari struktur data binary tree.



Membuat struktur data binary tree dalam suatu program (berbahasa C++) dapat menggunakan struct yang memiliki 2 buah pointer, seperti halnya double linked list.

```
struct pohon{
    char data;
    pohon *kanan;
    pohon *kiri;
};
pohon *simpul;
```



Gambar 2 Ilustrasi Simpul 2 Pointer

Operasi pada Tree

1) Create: digunakan untuk membentuk binary tree baru yang masih
2) Clear: digunakan untuk mengosongkan binary tree yang sudah ada atau menghapus semua node pada binary tree.
3) isEmpty: digunakan untuk memeriksa apakah binary tree masih kosong atau tidak.
4) Insert: digunakan untuk memasukkan sebuah node kedalam tree.
5) Find: digunakan untuk mencari root, parent, left child, atau right child dari suatu node dengan syarat tree tidak boleh kosong.
6) Update: digunakan untuk mengubah isi dari node yang ditunjuk oleh pointer current dengan syarat tree tidak boleh kosong.
7) Retrive: digunakan untuk mengetahui isi dari node yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
8) Delete Sub: digunakan untuk menghapus sebuah subtree (node beserta seluruh descendant-nya) yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.
9) Characteristic: digunakan untuk mengetahui karakteristik dari suatu tree. Yakni size, height, serta average lenght-nya.
10) Traverse: digunakan untuk mengunjungi seluruh node-node pada tree dengan cara traversal. Terdapat 3 metode traversal yang dibahas dalam modul ini yakni Pre-Order, In-Order, dan Post-Order.

A. Pre-Order

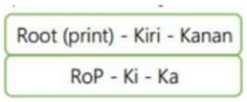
a) Penelusuran secara pre-order memiliki alur:

b) Cetak data pada simpul root

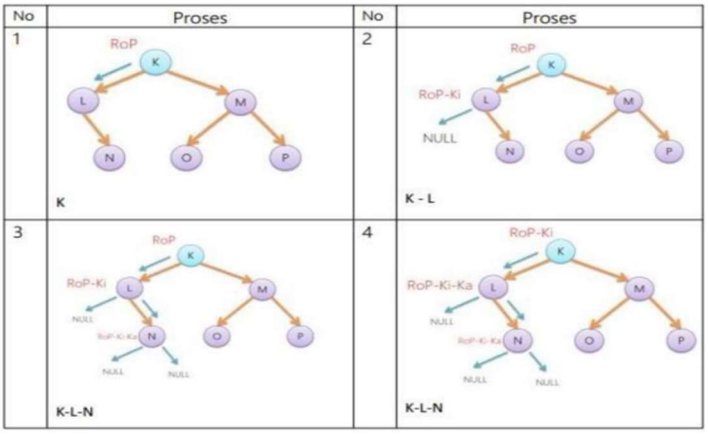
c) Secara rekursif mencetak seluruh data pada subpohon kiri

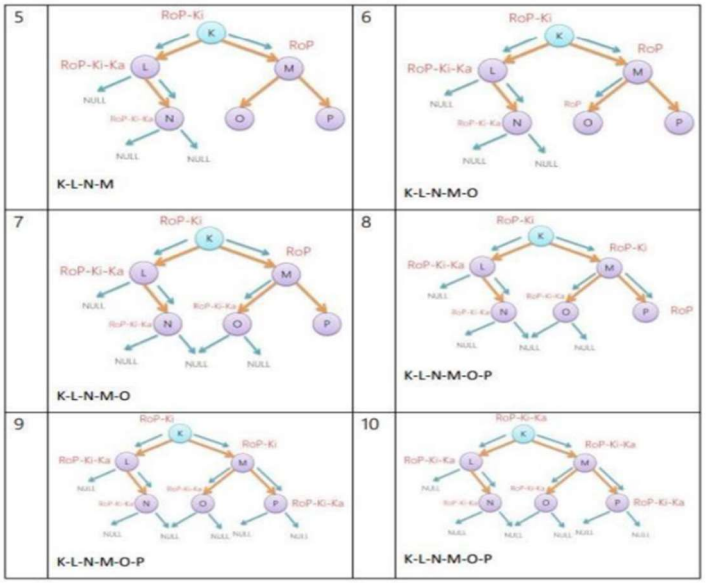
d) Secara rekursif mencetak seluruh data pada subpohon kanan

Dapat kita turunkan rumus penelusuran menjadi:



Alur pre-order





B. In-Order

Penelusuran secara in-order memiliki alur:

a) Secara rekursif mencetak seluruh data pada subpohon kiri

b) Cetak data pada root

c) Secara rekursif mencetak seluruh data pada subpohon kanan

Dapat kita turunkan rumus penelusuran menjadi:

C. Post Order

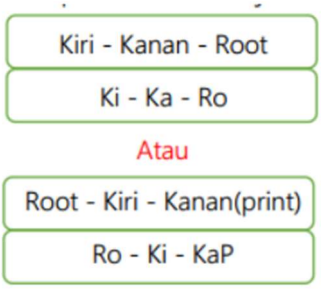
Penelusuran secara in-order memiliki alur:

a) Secara rekursif mencetak seluruh data pada subpohon kiri

b) Secara rekursif mencetak seluruh data pada subpohon kanan

c) Cetak data pada root

Dapat kita turunkan rumus penelusuran menjadi:

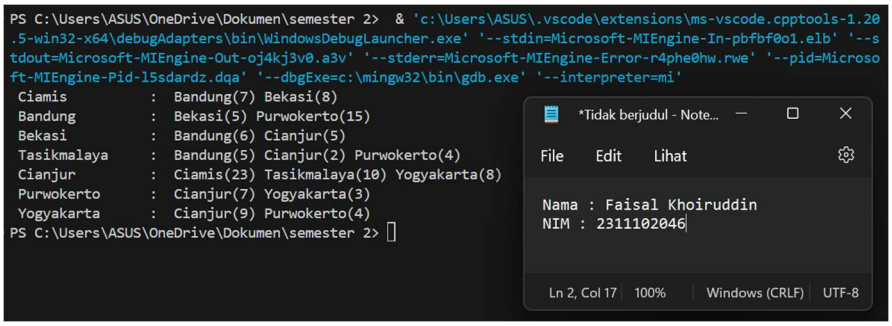


1. Guided 1

Source Code

```
#include <iostream>
#include <iomanip>
using namespace std;

string simpul[7] = {"Ciamis", "Bandung",
"Bekasi", "Tasikmalaya", "Cianjur",
"Purwokerto","Yogyakarta"};

int busur[7][7] = {
    {0, 7, 8, 0, 0, 0, 0},
    {0, 0, 5, 0, 0, 15, 0},
    {0, 6, 0, 0, 5, 0, 0,},
    {0, 5, 0, 0, 2, 4, 0,},
    {23, 0, 0, 10, 0, 0, 8},
    {0, 0, 0, 0, 7, 0, 3},
    {0, 0, 0, 0, 9, 4, 0} };
void tampilGraph()
{
    for(int baris = 0; baris < 7; baris++){
        cout << " " <<
setiosflags(ios::left) << setw(15) <<
simpul[baris] << " : ";
        for (int kolom = 0; kolom < 7;
kolom++){
            if(busur[baris][kolom] != 0){
                cout << " " <<
simpul[kolom] << "(" << busur[baris][kolom]
<< ")";
            }
        }
        cout << endl;
    }
}

int main(){
    tampilGraph();
    return 0;
}
```

Screenshots Output



Deskripsi:

Program tersebut merupakan program graph. Program tersebut menampilkan matriks dengan bobot tertentu antar simpul. Program tersebut menampilkan output .nama kota sebagai simpul dan bobot sebagai jarak antar kota

- #include <iostream> ➔ merupakan input output stream header yang digunakan sebagai standar input output operasi yang digunakan di c++
- #include <iomanip> Manipulator mengembalikan objek
- using namespace std; ➔ digunakan untuk mendeklarasikan/ memberitahukan kepada compiler bahwa kita akan menggunakan semua fungsi/class/file yang terdapat dalam namespace std
- string simpul[7] = {"Ciamis", "Bandung", "Bekasi", "Tasikmalaya", "Cianjur", "Purwokerto","Yogyakarta"}; ➔ deklarasi array simpul yang berisi nama-nama kota
- int busur[7][7] = {

  {0, 7, 8, 0, 0, 0, 0},

  {0, 0, 5, 0, 0, 15, 0},

  {0, 6, 0, 0, 5, 0, 0,},

  {0, 5, 0, 0, 2, 4, 0,},

  {23, 0, 0, 10, 0, 0, 8},

  {0, 0, 0, 0, 7, 0, 3},

  {0, 0, 0, 0, 9, 4, 0} }; ➔ deklarasi matriks busur yang berisi bobot antar simpul

- void tampilGraph() ➔ fungsi untuk menampilkan graf
- for(int baris = 0; baris < 7; baris++){ ➔ perulangan untuk setiap baris dalam matriks busur
- cout << " " << setiosflags(ios::left) << setw(15) << simpul[baris] << " : "; ➔ menampilkan simpul dan memformat output agar rata kiri dan memiliki lebar 15 karakter
- for (int kolom = 0; kolom < 7; kolom++){ ➔ perulangan untuk setiap kolom dalam matriks busur
- if(busur[baris][kolom] != 0){ ➔ jika bobot antar simpul tidak sama dengan nol
- cout << " " << simpul[kolom] << "(" << busur[baris][kolom] << ")"; ➔ menampilkan nama simpul dan bobot antar simpul
- cout << endl; ➔ menampilkan baris baru
- int main(){ ➔ merupakan fungsi utama
- tampilGraph(); ➔ memanggil fungsi untuk menampilkan graf
- return 0; ➔ program akan mengembalikan (return) nilai 0

2. Guided 2

Source Code

```
#include <iostream>
using namespace std;
/// PROGRAM BINARY TREE
// Deklarasi Pohon
struct Pohon
{
    char data;
    Pohon *left, *right, *parent;
};
Pohon *root, *baru;
// Inisialisasi
void init()
{
    root = NULL;
}
// Cek Node
int isEmpty()
```

```
{
    if (root == NULL)
        return 1; // true
    else
        return 0; // false
}
// Buat Node Baru
void buatNode(char data)
{
    if (isEmpty() == 1)
    {
        root = new Pohon();
        root->data = data;
        root->left = NULL;
        root->right = NULL;
        root->parent = NULL;
        cout << "\n Node " << data << "
berhasil dibuat menjadi root."
             << endl;
    }
    else
    {
        cout << "\n Pohon sudah dibuat" <<
endl;
    }
}
// Tambah Kiri
Pohon *insertLeft(char data, Pohon *node)
{
    if (isEmpty() == 1)
    {
        cout << "\n Buat tree terlebih
dahulu!" << endl;
        return NULL;
    }
    else
    {
        // cek apakah child kiri ada atau
tidak
        if (node->left != NULL)
        {
            // kalau ada
            cout << "\n Node " << node-
>data << " sudah ada child kiri!"
                 << endl;
            return NULL;
```

```
        }
        else
        {
            // kalau tidak ada
            baru = new Pohon();
            baru->data = data;
            baru->left = NULL;
            baru->right = NULL;
            baru->parent = node;
            node->left = baru;
            cout << "\n Node " << data << "
berhasil ditambahkan ke child kiri "
                 << baru->parent->data <<
endl;
            return baru;
        }
    }
}
// Tambah Kanan
Pohon *insertRight(char data, Pohon *node)
{
    if (root == NULL)
    {
        cout << "\n Buat tree terlebih
dahulu!" << endl;
        return NULL;
    }
    else
    {
        // cek apakah child kanan ada atau
tidak
        if (node->right != NULL)
        {
            // kalau ada
            cout << "\n Node " << node-
>data << " sudah ada child kanan!"
                 << endl;
            return NULL;
        }
        else
        {
            // kalau tidak ada
            baru = new Pohon();
            baru->data = data;
            baru->left = NULL;
            baru->right = NULL;
```

```
            baru->parent = node;
            node->right = baru;
            cout << "\n Node " << data << "
berhasil ditambahkan ke child kanan" <<
baru->parent->data << endl;
                return baru;
        }
    }
}
// Ubah Data Tree
void update(char data, Pohon *node)
{
    if (isEmpty() == 1)
    {
        cout << "\n Buat tree terlebih
dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\n Node yang ingin
diganti tidak ada!!" << endl;
        else
        {
            char temp = node->data;
            node->data = data;
            cout << "\n Node " << temp << "
berhasil diubah menjadi " << data << endl;
        }
    }
}
// Lihat Isi Data Tree
void retrieve(Pohon *node)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih
dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\n Node yang ditunjuk
tidak ada!" << endl;
        else
        {
```

```
            cout << "\n Data node : " <<
node->data << endl;
        }
    }
}
// Cari Data Tree
void find(Pohon *node)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih
dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\n Node yang ditunjuk
tidak ada!" << endl;
        else
        {
            cout << "\n Data Node : " <<
node->data << endl;
            cout << " Root : " << root-
>data << endl;
            if (!node->parent)
                cout << " Parent : (tidak
punya parent)" << endl;
            else
                cout << " Parent : " <<
node->parent->data << endl;
            if (node->parent != NULL &&
node->parent->left != node &&
                node->parent->right ==
node)
                cout << " Sibling : " <<
node->parent->left->data << endl;
            else if (node->parent != NULL
&& node->parent->right != node &&
                    node->parent->left ==
node)
                cout << " Sibling : " <<
node->parent->right->data << endl;
            else
                cout << " Sibling : (tidak
punya sibling)" << endl;
            if (!node->left)
```

```
                cout << " Child Kiri :
(tidak punya Child kiri)" << endl;
            else
                cout << " Child Kiri : " <<
node->left->data << endl;
            if (!node->right)
                cout << " Child Kanan :
(tidak punya Child kanan)" << endl;
            else
                cout << " Child Kanan : "
<< node->right->data << endl;
        }
    }
}
// Penelurusan (Traversal)
// preOrder
void preOrder(Pohon *node = root)
{
    if (!root)
        cout << "\n Buat tree terlebih
dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            cout << " " << node->data << ",
";
            preOrder(node->left);
            preOrder(node->right);
        }
    }
}
// inOrder
void inOrder(Pohon *node = root)
{
    if (!root)
        cout << "\n Buat tree terlebih
dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            inOrder(node->left);
            cout << " " << node->data << ",
";
            inOrder(node->right);
```

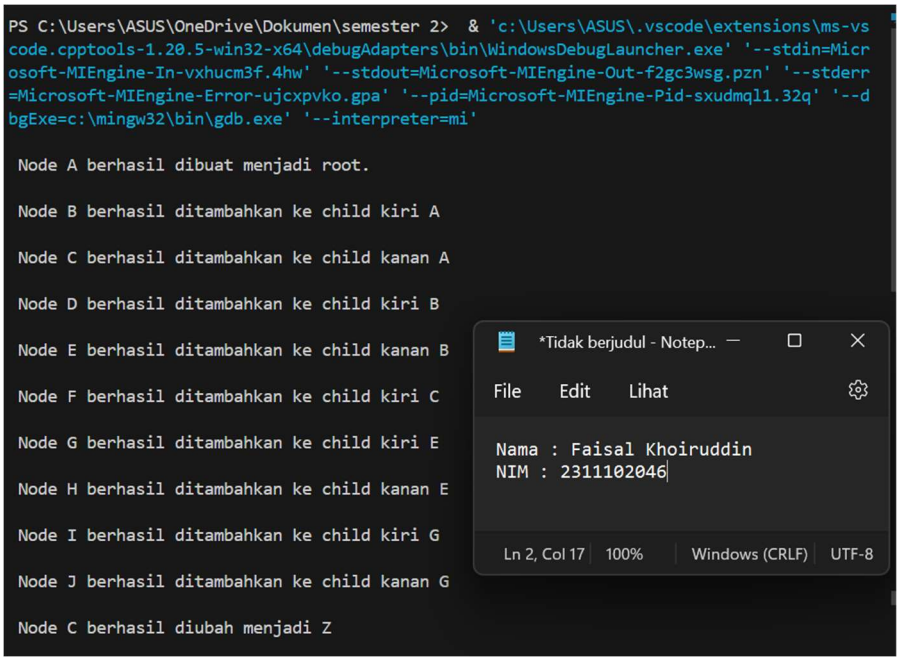
```
        }
    }
}
// postOrder
void postOrder(Pohon *node = root)
{
    if (!root)
        cout << "\n Buat tree terlebih
dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            postOrder(node->left);
            postOrder(node->right);
            cout << " " << node->data << ",
";
        }
    }
}
// Hapus Node Tree
void deleteTree(Pohon *node)
{
    if (!root)
        cout << "\n Buat tree terlebih
dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            if (node != root)
            {
                node->parent->left = NULL;
                node->parent->right = NULL;
            }
            deleteTree(node->left);
            deleteTree(node->right);
            if (node == root)
            {
                delete root;
                root = NULL;
            }
            else
            {
                delete node;
            }
```

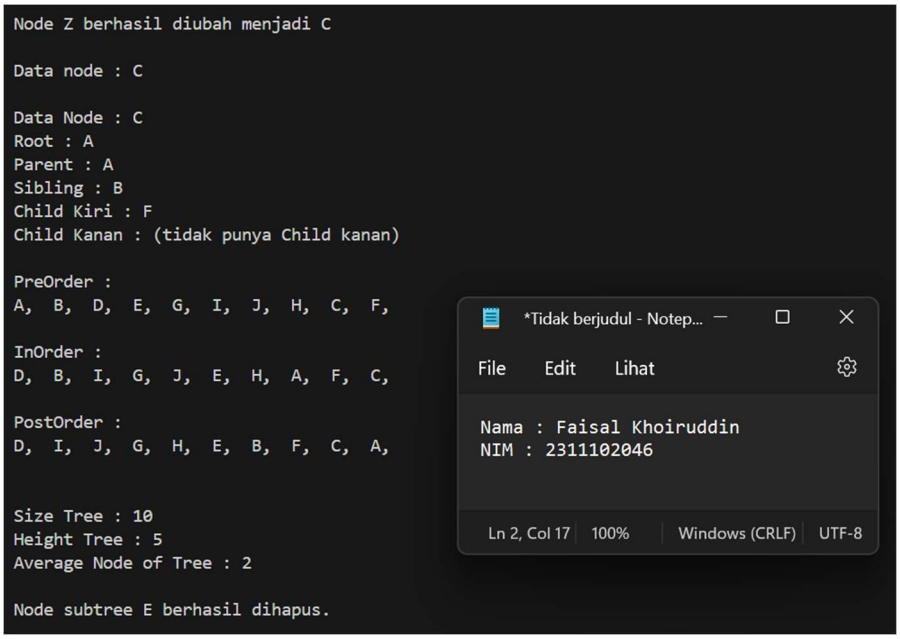
```
        }
    }
}
// Hapus SubTree
void deleteSub(Pohon *node)
{
    if (!root)
        cout << "\n Buat tree terlebih
dahulu!" << endl;
    else
    {
        deleteTree(node->left);
        deleteTree(node->right);
        cout << "\n Node subtree " << node-
>data << " berhasil dihapus." << endl;
    }
}
// Hapus Tree
void clear()
{
    if (!root)
        cout << "\n Buat tree terlebih
dahulu!!" << endl;
    else
    {
        deleteTree(root);
        cout << "\n Pohon berhasil
dihapus." << endl;
    }
}
// Cek Size Tree
int size(Pohon *node = root)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih
dahulu!!" << endl;
        return 0;
    }
    else
    {
        if (!node)
        {
            return 0;
        }
        else
```

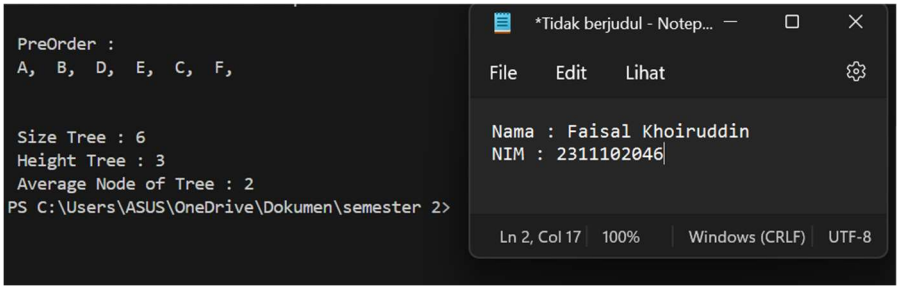
```
        {
            return 1 + size(node->left) +
size(node->right);
        }
    }
}
// Cek Height Level Tree
int height(Pohon *node = root)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih
dahulu!" << endl;
        return 0;
    }
    else
    {
        if (!node)
        {
            return 0;
        }
        else
        {
            int heightKiri = height(node-
>left);
            int heightKanan = height(node-
>right);
            if (heightKiri >= heightKanan)
            {
                return heightKiri + 1;
            }
            else
            {
                return heightKanan + 1;
            }
        }
    }
}
// Karakteristik Tree
void charateristic()
{
    cout << "\n Size Tree : " << size() <<
endl;
    cout << " Height Tree : " << height()
<< endl;
    cout << " Average Node of Tree : " <<
```

```
size() / height() << endl;
}
int main()
{
    buatNode('A');
    Pohon *nodeB, *nodeC, *nodeD, *nodeE,
*nodeF, *nodeG, *nodeH,
        *nodeI, *nodeJ;
    nodeB = insertLeft('B', root);
    nodeC = insertRight('C', root);
    nodeD = insertLeft('D', nodeB);
    nodeE = insertRight('E', nodeB);
    nodeF = insertLeft('F', nodeC);
    nodeG = insertLeft('G', nodeE);
    nodeH = insertRight('H', nodeE);
    nodeI = insertLeft('I', nodeG);
    nodeJ = insertRight('J', nodeG);
    update('Z', nodeC);
    update('C', nodeC);
    retrieve(nodeC);
    find(nodeC);
    cout << "\n PreOrder :" << endl;
    preOrder(root);
    cout << "\n"
         << endl;
    cout << " InOrder :" << endl;
    inOrder(root);
    cout << "\n"
         << endl;
    cout << " PostOrder :" << endl;
    postOrder(root);
    cout << "\n"
         << endl;
    charateristic();
    deleteSub(nodeE);
    cout << "\n PreOrder :" << endl;
    preOrder();
    cout << "\n"
         << endl;
    charateristic();
}
```

Screenshots Output


PS C:\Users\ASUS\OneDrive\Dokumen\semester 2> & 'c:\Users\ASUS\.vscode\extensions\ms-vscode.cpptools-1.20.5-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-vxhucm3f.4hw' '--stdout=Microsoft-MIEngine-Out-f2gc3wsg.pzn' '--stderr=Microsoft-MIEngine-Error-ujcxpvko.gpa' '--pid=Microsoft-MIEngine-Pid-sxudmql1.32q' '--dbgExe=c:\mingw32\bin\gdb.exe' '--interpreter=mi'
Node A berhasil dibuat menjadi root.
Node B berhasil ditambahkan ke child kiri A
Node C berhasil ditambahkan ke child kanan A
Node D berhasil ditambahkan ke child kiri B
Node E berhasil ditambahkan ke child kanan B
Node F berhasil ditambahkan ke child kiri C
Node G berhasil ditambahkan ke child kiri E
Node H berhasil ditambahkan ke child kanan E
Node I berhasil ditambahkan ke child kiri G
Node J berhasil ditambahkan ke child kanan G
Node C berhasil diubah menjadi Z
*Tidak berjudul - Notep...
File Edit Lihat
Nama : Faisal Khoiruddin
NIM : 2311102046
Ln 2, Col 17 100% Windows (CRLF) UTF-8


Node Z berhasil diubah menjadi C
Data node : C
Data Node : C
Root : A
Parent : A
Sibling : B
Child Kiri : F
Child Kanan : (tidak punya Child kanan)
PreOrder :
A, B, D, E, G, I, J, H, C, F,
InOrder :
D, B, I, G, J, E, H, A, F, C,
PostOrder :
D, I, J, G, H, E, B, F, C, A,
Size Tree : 10
Height Tree : 5
Average Node of Tree : 2
Node subtree E berhasil dihapus.
*Tidak berjudul - Notep...
File Edit Lihat
Nama : Faisal Khoiruddin
NIM : 2311102046
Ln 2, Col 17 100% Windows (CRLF) UTF-8


PreOrder :
A, B, D, E, C, F,
Size Tree : 6
Height Tree : 3
Average Node of Tree : 2
PS C:\Users\ASUS\OneDrive\Dokumen\semester 2>
*Tidak berjudul - Notep...
File Edit Lihat
Nama : Faisal Khoiruddin
NIM : 2311102046
Ln 2, Col 17 100% Windows (CRLF) UTF-8

Deskripsi:

Program tersebut merupakan program tree. Variabel pada program tersebut sudah diinputkan. Program tersebut menampilkan data berupa huruf dari penambahan setiap operasi dari tree.

- #include <iostream> ➔ merupakan input output stream header yang digunakan sebagai standar input output operasi yang digunakan di c++
- using namespace std; ➔ digunakan untuk mendeklarasikan/ memberitahukan kepada compiler bahwa kita akan menggunakan semua fungsi/class/file yang terdapat dalam namespace std
- struct Pohon ➔ Deklarasi struct Pohon
- char data; ➔ deklarasi variabel data dalam struct pohon
- Pohon *left, *right, *parent; ➔ pointer ke anak kiri, anak kanan, dan orang tua
- Pohon *root, *baru; ➔ deklarasi pointer untuk root pohon dan node baru
- void init() ➔ Inisialisasi pohon
- root = NULL; ➔ mengatur root menjadi NULL
- int isEmpty() ➔ Cek Node jika kosong
- if (root == NULL) ➔ jika root adalah kosong
- return 1; ➔ mengembalikan 1 true
- else ➔ kalua tidak
- return 0; ➔ mengembalikan 0 false
- void buatNode(char data) ➔ fungsi buat node baru dengan parameter data tipe data char
- if (isEmpty() == 1) ➔ jika pohon kosong = 1
- root = new Pohon(); ➔ membuat node baru mengisi ke root
- root->data = data; ➔ mengatur data dari node root
- root->left = NULL; ➔ mengatur anak kiri dari root menjadi NULL
- root->right = NULL; ➔ mengatur anak kanan dari root menjadi NULL

- root->parent = NULL; ➔ mengatur parent dari root menjadi NULL
- cout << "\n Node " << data << " berhasil dibuat menjadi root." << endl; ➔ menampilkan pesan berhasil
- else ➔ kalau tidak
- cout << "\n Pohon sudah dibuat" << endl; ➔ menampilkan pesan pohon sudah dibuat
- Pohon *insertLeft(char data, Pohon *node) ➔ Tambah Kiri
- if (isEmpty() == 1) ➔ jika pohon kosong
- cout << "\n Buat tree terlebih dahulu!" << endl; ➔ menampilkan pesan buat tree terlebih dahulu
- return NULL; ➔ mengembalikan NULL
- else ➔ kalau tidak
- if (node->left != NULL) ➔ cek apakah child kiri ada atau tidak
- cout << "\n Node " << node->data << " sudah ada child kiri!" << endl; ➔ menampilkan pesan sudah ada
- return NULL; ➔ mengembalikan NULL
- else ➔ kalau tidak ada
- baru = new Pohon(); ➔ membuat node baru
- baru->data = data; ➔ mengatur data dari node baru
- baru->left = NULL; ➔ mengatur anak kiri dari node baru menjadi NULL
- baru->right = NULL; ➔ mengatur anak kanan dari node baru menjadi NULL
- baru->parent = node; ➔ mengatur orang tua dari node baru menjadi node yang diberikan
- node->left = baru; ➔ mengatur anak kiri dari node yang diberikan menjadi node baru
- cout << "\n Node " << data << " berhasil ditambahkan ke child kiri " << baru->parent->data << endl; ➔ menampilkan pesan berhasil

- return baru; ➔ mengembalikan pointer ke node baru
- Pohon *insertRight(char data, Pohon *node) ➔ menambahkan node kanan
- if (root == NULL) ➔ jika root kosong
- cout << "\n Buat tree terlebih dahulu!" << endl; ➔ menampilkan statement buat tree terlebih dahulu
- return NULL; ➔ mengembalikan NULL
- else ➔ kalau tidak
- if (node->right != NULL) ➔ jika node kanan tidak NULL
- cout << "\n Node " << node->data << " sudah ada child kanan!" << endl; ➔ menampilkan statement sudah ada
- return NULL; ➔ mengembalikan NULL
- else ➔ kalau tidak
- baru = new Pohon(); ➔ membuat node baru
- baru->data = data; ➔ mengatur data dari node baru
- baru->left = NULL; ➔ mengatur anak kiri dari node baru menjadi NULL
- baru->right = NULL; ➔ mengatur anak kanan dari node baru menjadi NULL
- baru->parent = node; ➔ mengatur parent dari node yang diberikan menjadi node baru
- node->right = baru; ➔ mengatur anak kanan dari node yang diberikan menjadi node baru
- cout << "\n Node " << data << " berhasil ditambahkan ke child kanan " << baru->parent->data << endl; ➔ menampilkan statement berhasil
- return baru; ➔ mengembalikan pointer ke node baru
- void update(char data, Pohon *node) ➔ fungsi untuk mengupdate data dalam node
- if (isEmpty() == 1) ➔ jika kosong sama dengan 1

- cout << "\n Buat tree terlebih dahulu!" << endl; ➔ menampilkan statement buat tree terlebih dahulu
- else ➔ kalau tidak
- if (!node) ➔ jika node tidak ada
- cout << "\n Node yang ingin diganti tidak ada!!" << endl; ➔ menampilkan statement Node yang ingin diganti tidak ada!
- Else ➔ kalau tidak
- char temp = node->data; ➔ menyimpan data saat ini dari node
- node->data = data; ➔ memperbarui data dari node
- cout << "\n Node " << temp << " berhasil diubah menjadi " << data << endl; ➔ menampilkan statement Node berhasil diubah menjadi dilanjutkan memanggil variabel data
- void retrieve(Pohon *node) ➔ fungsi untuk mengambil data dalam node
- if (!root) ➔ jika tidak ada root
- cout << "\n Buat tree terlebih dahulu!" << endl; ➔ menampilkan statement Buat tree terlebih dahulu
- else ➔ kalau tidak
- if (!node) ➔ jika tidak ada node
- cout << "\n Node yang ditunjuk tidak ada!" << endl; ➔ menampilkan statement Node yang ditunjuk tidak ada!
- Else ➔ kalau tidak
- cout << "\n Data node : " << node->data << endl; ➔ menampilkan data dari node
- void find(Pohon *node) ➔ fungsi mencari node dalam pohon
- if (!root) ➔ jika tidak ada root
- cout << "\n Buat tree terlebih dahulu!" << endl; ➔ menampilkan statement Buat tree terlebih dahulu!
- Else ➔ kalau tidak
- if (!node) ➔ jika tidak ada node

- cout << "\n Node yang ditunjuk tidak ada!" << endl; ➔ menampilkan statement Node yang ditunjuk tidak ada!
- Else ➔ kalau tidak
- cout << "\n Data Node : " << node->data << endl; ➔ data dari node
- cout << " Root : " << root->data << endl; ➔ menampilkan data dari root
- if (!node->parent) ➔ jika tidak memiliki orang tua
- cout << " Parent : (tidak punya parent)" << endl; ➔ menampilkan statement Parent : (tidak punya parent)
- else ➔ kalau node memiliki orang tua
- cout << " Parent : " << node->parent->data << endl; ➔ menampilkan data dari orang tua
- if (node->parent != NULL && node->parent->left != node && node->parent->right == node) ➔ jika node memiliki saudara kiri
- cout << " Sibling : " << node->parent->left->data << endl; ➔ menampilkan data dari saudara
- else if (node->parent != NULL && node->parent->right != node && node->parent->left == node) ➔ jika node memiliki saudara kanan
- cout << " Sibling : " << node->parent->right->data << endl; ➔ menampilkan data dari saudara
- else ➔ jika node tidak memiliki saudara
- cout << " Sibling : (tidak punya sibling)" << endl; ➔ menampilkan statement Sibling : (tidak punya sibling)
- if (!node->left) ➔ jika node tidak memiliki anak kiri
- cout << " Child Kiri : (tidak punya Child kiri)" << endl; ➔ menampilkan pesan Child Kiri : (tidak punya Child kiri)
- else ➔ jika node memiliki anak kiri
- cout << " Child Kiri : " << node->left->data << endl; ➔ menampilkan data dari anak kiri

- if (!node->right) ➔ jika node tidak memiliki anak kanan
- cout << " Child Kanan : (tidak punya Child kanan)" << endl; ➔ menampilkan Child Kanan : (tidak punya Child kanan)
- else ➔ jika node memiliki anak kanan
- cout << " Child Kanan : " << node->right->data << endl; ➔ menampilkan data dari anak kanan
- void preOrder(Pohon *node = root) ➔ fungsi untuk melakukan penelusuran pohon dalam preorder
- if (!root) ➔ jika tidak ada root
- cout << "\n Buat tree terlebih dahulu!" << endl; ➔ menampilkan statement Buat tree terlebih dahulu!
- Else ➔ k alau tidak
- if (node != NULL)➔ jika node tidak kosong
- cout << " " << node->data << ", "; ➔ menampilkan data dari node
- preOrder(node->left); ➔ melakukan penelusuran preorder pada subtree kiri secara rekursif
- preOrder(node->right); ➔ melakukan penelusuran preorder pada subtree kanan secara rekursif
- void inOrder(Pohon *node = root) ➔ fungsi melakukan penelusuran dalam inorder
- if (!root) ➔ jika tidak ada root
- cout << "\n Buat tree terlebih dahulu!" << endl; ➔ menampilkan statement Buat tree terlebih dahulu!
- Else ➔ kalau tidak
- if (node != NULL) ➔ jika node ada
- inOrder(node->left); ➔ melakukan penelusuran secara inorder pada subtree kiri secara rekursif
- cout << " " << node->data << ", "; ➔ menampilkan data dari node
- inOrder(node->right); ➔ melakukan penelusuran secara inorder pada subtree kanan secara rekursif

- void postOrder(Pohon *node = root) ➔ fungsi melakukan penelusuran secara postorder
- if (!root) ➔ jika tidak ada root
- cout << "\n Buat tree terlebih dahulu!" << endl;
- else ➔ kalau tidak
- if (node != NULL) ➔ jika node tidak kosong
- postOrder(node->left); ➔ melakukan penelusuran postorder pada subtree kiri secara rekursif
- postOrder(node->right); ➔ melakukan penelusuran postorder pada subtree kanan secara rekursif
- cout << " " << node->data << ", "; ➔ menampilkan data dari node
- void deleteTree(Pohon *node) ➔ fungsi untuk menghapus node pohon
- if (!root) ➔ jika tidak ada root
- cout << "\n Buat tree terlebih dahulu!" << endl; ➔ menampilkan statement Buat tree terlebih dahulu!
- Else ➔ kalau tidak
- if (node != NULL) ➔ jika node tidak NULL
- if (node != root) ➔ jika node tidak sama dengan root
- node->parent->left = NULL; ➔ mengatur anak kiri dari orang tua menjadi NULL
- node->parent->right = NULL; ➔ mengatur anak kanan dari orang tua menjadi NULL
- deleteTree(node->left); ➔ menghapus subtree kiri secara rekursif
- deleteTree(node->right); ➔ menghapus subtree kanan secara rekursif
- if (node == root) ➔ jika node adalah root
- delete root; ➔ menghapus root
- root = NULL; ➔ mengatur root menjadi NULL

- else ➔ kalau tidak
- delete node; ➔ menghapus node
- void deleteSub(Pohon *node) ➔ fungsi untuk menghapus subtree
- if (!root) ➔ jika tidak ada root
- cout << "\n Buat tree terlebih dahulu!" << endl; ➔ cout menampilkan staetemnet Buat tree terlebih dahulu!
- Else ➔ kalau tidak
- deleteTree(node->left); ➔ menghapus subtree kiri
- deleteTree(node->right); ➔ menghapus subtree kanan
- cout << "\n Node subtree " << node->data << " berhasil dihapus." << endl; ➔ menampilkan statement berhasil
- void clear() ➔ fungsi untuk menghapus pohon
- if (!root) ➔ jika tidak ada root
- cout << "\n Buat tree terlebih dahulu!!" << endl; ➔ menampilkan statement Buat tree terlebih dahulu!
- Else ➔ kalau tidak
- deleteTree(root); ➔ menghapus pohon
- cout << "\n Pohon berhasil dihapus." << endl; ➔ menampikan statement Pohon berhasil dihapus
- int size(Pohon *node = root) ➔ fungsi untuk memeriksa ukuran pohon
- if (!root) ➔ jika tidak ada root
- cout << "\n Buat tree terlebih dahulu!!" << endl; ➔ menampilkan statement Buat tree terlebih dahulu!!
- return 0; ➔ mengembalikan 0
- else ➔ kalau tidak
- if (!node) ➔ jika tidak ada node
- return 0; ➔ mengembalikan 0

- else ➔ kalau tidak
- return 1 + size(node->left) + size(node->right); ➔ mengembalikan jumlah node dalam pohon
- int height(Pohon *node = root) ➔ fungsi untuk memeriksa ketinggian pohon
- if (!root) ➔ jika tidak ada root
- cout << "\n Buat tree terlebih dahulu!" << endl; ➔ menampilkan statement Buat tree terlebih dahulu!
- return 0; ➔ mengembalika 0
- else ➔ kalau tidak
- if (!node) ➔ kalai tidak ada node
- return 0; ➔ mengembalikan 0
- else ➔ kalau tidak
- int heightKiri = height(node->left); ➔ menghitung ketinggian subtree kiri
- int heightKanan = height(node->right); ➔ menghitung ketinggian subtree kanan
- if (heightKiri >= heightKanan) ➔ jika tinggi subtree kiri >= tinggi subtree kanan
- return heightKiri + 1; ➔ mengembalikan tinggi subtree kiri tambah 1
- else ➔ kalau tidak
- return heightKanan + 1; ➔ mengembalikan tinggi subtree kana tambah 1
- void charateristic() ➔ fungsi untuk menampilkan karakteristik pohon
- cout << "\n Size Tree : " << size() << endl; ➔ menampilkan ukuran pohon
- cout << " Height Tree : " << height() << endl; ➔ menampilkan tinggi pohon
- cout << " Average Node of Tree : " << size() / height() << endl;➔

menampilkan rata-rata node dalam pohon

- int main() ➔ merupakan fungsi utama
- buatNode('A'); ➔ membuat node root dengan data A
- Pohon *nodeB, *nodeC, *nodeD, *nodeE, *nodeF, *nodeG, *nodeH, *nodeI, *nodeJ; ➔ deklarasi pointer ke beberapa node
- nodeB = insertLeft('B', root); ➔ menambahkan node dengan data B sebagai anak kiri dari root
- nodeC = insertRight('C', root); ➔ menambahkan node dengan data C sebagai anak kanan dari root
- nodeD = insertLeft('D', nodeB); ➔ menambahkan node dengan data D sebagai anak kiri dari node B
- nodeE = insertRight('E', nodeB); ➔ menambahkan node dengan data E sebagai anak kanan dari node B
- nodeF = insertLeft('F', nodeC); ➔ menambahkan node dengan data F sebagai anak kiri dari node C
- nodeG = insertLeft('G', nodeE); ➔ menambahkan node dengan data G sebagai anak kiri dari node E
- nodeH = insertRight('H', nodeE); ➔ menambahkan node dengan data H sebagai anak kanan dari node E
- nodeI = insertLeft('I', nodeG); ➔ menambahkan node dengan data I sebagai anak kiri dari node G
- nodeJ = insertRight('J', nodeG); ➔ menambahkan node dengan data I sebagai anak kanan dari node G
- update('Z', nodeC); ➔ memperbarui data dari node C menjadi Z
- update('C', nodeC); ➔ memperbarui data dari node C menjadi C
- retrieve(nodeC); ➔ menampilkan data dari node C
- find(nodeC); ➔ menampikan informasi dari node C
- cout << "\n PreOrder :" << endl; ➔ menampilkan statement PreOrder
- preOrder(root); ➔ melakukan penelusuran pohondalam urutan preorder

- cout << "\n" << endl; ➔ menampikan baris baru
- cout << " InOrder :" << endl; ➔ menampilkan statement InOrder
- inOrder(root); ➔ melakukan penelusuran pohon dalam urutan ionerder
- cout << "\n" << endl; ➔ menampikan baris baru
- cout << " PostOrder :" << endl; ➔ menampilkan statement PostOrder
- postOrder(root); ➔ melakukan penelusuran pohon dalam urutan postorder
- cout << "\n" << endl; ➔ menampikan baris baru
- charateristic(); ➔ menampilkan karakteristik pohon
- deleteSub(nodeE); ➔ menghapus subtree
- cout << "\n PreOrder :" << endl; ➔ menampilkan statement PreOrder
- preOrder(); ➔ melakukan penelusuran pohon dalam urutan preorder
- cout << "\n" << endl; ➔ menampikan baris baru
- charateristic(); ➔ menampilkan karakteristik pohon

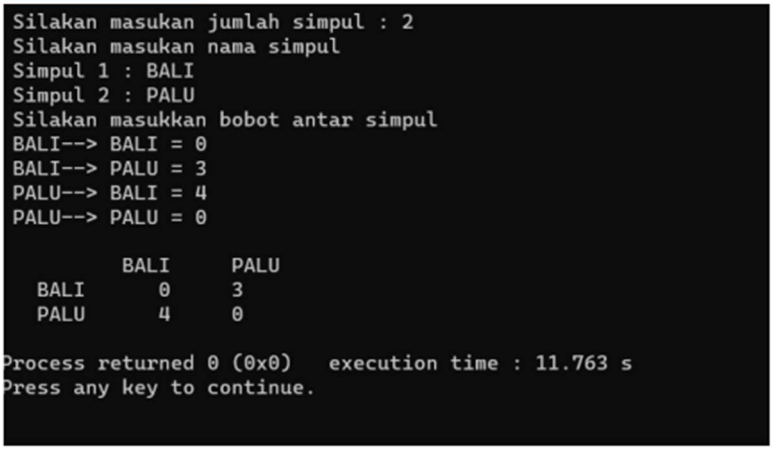
B. Unguided/Tugas

1. Unguided 1

*Cantumkan NIM pada salah satu variabel di dalam program.

Contoh : int nama_22102003;

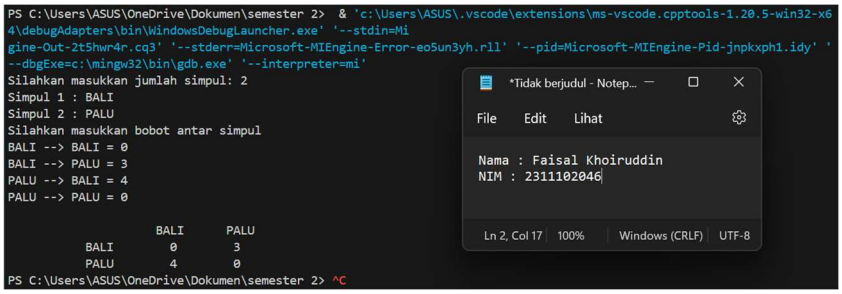
Buatlah program graph dengan menggunakan inputan user untuk menghitung jarak dari sebuah kota ke kota lainnya. Output Program




```
#include <iostream>
#include <iomanip>
#include <vector>
#include <map>

using namespace std;

int main()
{
    int jumlah_simpul;
    cout << "Silahkan masukkan jumlah
simpul: ";
    cin >> jumlah_simpul;

    map<string, map<string, int>> busur;
    vector<string> simpul(jumlah_simpul);
    for(int i = 0; i < jumlah_simpul; i++)
    {
        cout << "Simpul " << i + 1 << " :
";
        cin >> simpul[i];
    }

    cout << "Silahkan masukkan bobot antar
simpul" << endl;
    for (int i = 0; i < jumlah_simpul;
i++)
    {
```

```
        for (int j = 0; j < jumlah_simpul;
++j)
        {
            cout << simpul[i] << " --> "
<< simpul[j] << " = ";
            int
bobot_Faisal_Khoiruddin_2311102046;
            cin >>
bobot_Faisal_Khoiruddin_2311102046;
            busur[simpul[i]][simpul[j]] =
bobot_Faisal_Khoiruddin_2311102046;
        }
    }

    cout << endl << setw(15) << " ";
    for (int i = 0; i < jumlah_simpul;
++i)
    {
        cout << setw(10) << simpul[i];
    }
    cout << endl;
    for(int i = 0; i < jumlah_simpul; ++i)
    {
        cout << setw(15) << simpul[i];
        for (int j = 0; j < jumlah_simpul;
++j)
        {
            cout << setw(9) <<
busur[simpul[i]][simpul[j]];
        }
        cout << endl;
    }
    return 0;
}
```

Screenshots Output



Deskripsi

Program tersebut merupakan graph. Program tersebut pengguna diminta menginputkan jumlah simpul kemudian menginpukan simpul 1 dan 2 setelah itu menginputkan bobot antar simpul. Program tersebut menampilkan output matriks dari nama simpul dan bobotnya..

- #include <iostream> ➔ merupakan input output stream header yang digunakan sebagai standar input output operasi yang digunakan di c++
- #include <iomanip> ➔ Manipulator mengembalikan objek
- #include <vector> ➔ Vektor menyimpan elemen jenis tertentu dalam pengaturan linier, dan memungkinkan akses acak cepat ke elemen apa pun.
- #include <map> ➔ digunakan untuk memasukkan memasukkan library map yang merupakan bagian dari Strandard Template Library
- using namespace std; ➔ digunakan untuk mendeklarasikan/ memberitahukan kepada compiler bahwa kita akan menggunakan semua fungsi/class/file yang terdapat dalam namespace std
- int main() ➔ merupakan fungsi utama
- int jumlah_simpul; ➔ deklarasi variabel jumlah simpul
- cout << "Silahkan masukkan jumlah simpul: "; ➔ menampikan statement Silahkan masukkan jumlah simpul:
- cin >> jumlah_simpul; ➔ menerima inputan dari pengguna dan menyimpan ke dalam variabel jumlah simpul
- map<string, map<string, int>> busur; ➔ deklarasi map untuk menyimpan bobot antar simpul
- vector<string> simpul(jumlah_simpul); ➔ deklarasi vector untuk menyimpan nama simpul
- for(int i = 0; i < jumlah_simpul; i++) ➔ perulangan untuk menerima input nama simpul
- cout << "Simpul " << i + 1 << " : "; ➔ menampilkan statement untuk meminta input nama simpul
- cin >> simpul[i]; ➔ menerima input nama simpul dari pengguna
- cout << "Silahkan masukkan bobot antar simpul" << endl;

➔ menampilkan statement Silahkan masukkan bobot antar simpul

- for (int i = 0; i < jumlah_simpul; i++) ➔ perulangan untuk menerima input bobot antar simpul
- for (int j = 0; j < jumlah_simpul; ++j) ➔ perulangan untuk menerima input bobot antar simpul
- cout << simpul[i] << " --> " << simpul[j] << " = "; ➔ menampilkan statement untuk meminta input bobot antar simpul
- int bobot_Faisal_Khoiruddin_2311102046; ➔ deklarasi variabel untuk menyimpan bobot antar simpul
- cin >> bobot_Faisal_Khoiruddin_2311102046; ➔ menerima input dari pengguna dan menyimpan ke dalam variabel bobot
- busur[simpul[i]][simpul[j]] = bobot_Faisal_Khoiruddin_2311102046; ➔ menyimpan bobot antar simpul ke dalam map
- cout << endl << setw(15) << " "; ➔ menampilkan spasi sebelum header kolom
- for (int i = 0; i < jumlah_simpul; ++i) ➔ perulangan untuk menampilkan header kolom
- cout << setw(10) << simpul[i]; ➔ menampilkan nama simpul sebagai header kolom
- cout << endl; ➔ menampilkan baris baru
- for(int i = 0; i < jumlah_simpul; ++i) ➔ perulangan untuk menceta baris
- cout << setw(15) << simpul[i]; ➔ menampilkan nama simpul sebagai header baris
- for (int j = 0; j < jumlah_simpul; ++j) ➔ perulangan untuk menampilkan bobot antar simpul
- cout << setw(9) << busur[simpul[i]][simpul[j]]; ➔ menampilkan bobot antar simpul
- cout << endl; ➔ menampilkan baris baru
- return 0; ➔ program akan mengembalikan (return) nilai 0

2. Unguided 2

Modifikasi guided tree diatas dengan program menu menggunakan input data tree dari user dan berikan fungsi tambahan untuk

menampilkan node child dan descendant dari node yang diinput kan!

Source Code

```
#include <iostream>
using namespace std;
/// PROGRAM BINARY TREE
// Deklarasi Pohon
struct Pohon
{
    char data;
    Pohon *left, *right, *parent;
};
Pohon *root, *baru;
// Inisialisasi
void init()
{
    root = NULL;
}
// Cek Node
int isEmpty()
{
    if (root == NULL)
        return 1; // true
    else
        return 0; // false
}
// Buat Node Baru
void buatNode(char data)
{
    if (isEmpty() == 1)
    {
        root = new Pohon();
        root->data = data;
        root->left = NULL;
        root->right = NULL;
        root->parent = NULL;
        cout << "\n Node " << data << "
berhasil dibuat menjadi root." << endl;
    }
    else
    {
        cout << "\n Pohon sudah dibuat" <<
endl;
    }
}
```

```
// Tambah Kiri
Pohon *insertLeft(char data, Pohon *node)
{
    if (isEmpty() == 1)
    {
        cout << "\n Buat tree terlebih
dahulu!" << endl;
        return NULL;
    }
    else
    {
        // cek apakah child kiri ada atau
tidak
        if (node->left != NULL)
        {
            // kalau ada
            cout << "\n Node " << node-
>data << " sudah ada child kiri!"
                 << endl;
            return NULL;
        }
        else
        {
            // kalau tidak ada
            baru = new Pohon();
            baru->data = data;
            baru->left = NULL;
            baru->right = NULL;
            baru->parent = node;
            node->left = baru;
            cout << "\n Node " << data <<
" berhasil ditambahkan ke child kiri " <<
baru->parent->data << endl;
            return baru;
        }
    }
}
// Tambah Kanan
Pohon *insertRight(char data, Pohon *node)
{
    if (root == NULL)
    {
        cout << "\n Buat tree terlebih
dahulu!" << endl;
        return NULL;
    }
```

```
    else
    {
        // cek apakah child kanan ada atau
tidak
        if (node->right != NULL)
        {
            // kalau ada
            cout << "\n Node " << node-
>data << " sudah ada child kanan!"
                 << endl;
            return NULL;
        }
        else
        {
            // kalau tidak ada
            baru = new Pohon();
            baru->data = data;
            baru->left = NULL;
            baru->right = NULL;
            baru->parent = node;
            node->right = baru;
            cout << "\n Node " << data <<
" berhasil ditambahkan ke child kanan " <<
baru->parent->data << endl;
                return baru;
        }
    }
}
// Ubah Data Tree
void update(char data, Pohon *node)
{
    if (isEmpty() == 1)
    {
        cout << "\n Buat tree terlebih
dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\n Node yang ingin
diganti tidak ada!!" << endl;
        else
        {
            char temp = node->data;
            node->data = data;
            cout << "\n Node " << temp <<
```

```
" berhasil diubah menjadi " << data <<
endl;
        }
    }
}
// Lihat Isi Data Tree
void retrieve(Pohon
*node)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih
dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\n Node yang ditunjuk
tidak ada!" << endl;
        else
        {
            cout << "\n Data node : " <<
node->data << endl;
        }
    }
}
// Cari Data Tree
void find(Pohon *node)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih
dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\n Node yang ditunjuk
tidak ada!" << endl;
        else
        {
            cout << "\n Data Node : " <<
node->data << endl;
            cout << " Root : " << root-
>data << endl;
            if (!node->parent)
```

```
                cout << " Parent : (tidak
punya parent)" << endl;
            else
                cout << " Parent : " <<
node->parent->data << endl;
            if (node->parent != NULL &&
node->parent->left != node &&
                node->parent->right ==
node)
                cout << " Sibling : " <<
node->parent->left->data << endl;
            else if (node->parent != NULL
&& node->parent->right != node &&
                    node->parent->left ==
node)
                cout << " Sibling : " <<
node->parent->right->data << endl;
            else
                cout << " Sibling : (tidak
punya sibling)" << endl;
            if (!node->left)
                cout << " Child Kiri :
(tidak punya Child kiri)" << endl;
            else
                cout << " Child Kiri : "
<< node->left->data << endl;
            if (!node->right)
                cout << " Child Kanan :
(tidak punya Child kanan)" << endl;
            else
                cout << " Child Kanan : "
<< node->right->data << endl;
        }
    }
}
// Penelurusan (Traversal)
// preorder
void preOrder(Pohon *node = root)
{
    if (!root)
        cout << "\n Buat tree terlebih
dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
```

```
            cout << " " << node->data <<
", ";
            preOrder(node->left);
            preOrder(node->right);
        }
    }
}
// inOrder
void inOrder(Pohon *node = root)
{
    if (!root)
        cout << "\n Buat tree terlebih
dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            inOrder(node->left);
            cout << " " << node->data <<
", ";
            inOrder(node->right);
        }
    }
}
// postOrder
void postOrder(Pohon *node = root)
{
    if (!root)
        cout << "\n Buat tree terlebih
dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            postOrder(node->left);
            postOrder(node->right);
            cout << " " << node->data <<
", ";
        }
    }
}
// Hapus Node Tree
void deleteTree(Pohon *node)
{
    if (!root)
        cout << "\n Buat tree terlebih
```

```
dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            if (node != root)
            {
                node->parent->left = NULL;
                node->parent->right =
NULL;
            }
            deleteTree(node->left);
            deleteTree(node->right);
            if (node == root)
            {
                delete root;
                root = NULL;
            }
            else
            {
                delete node;
            }
        }
    }
}
// Hapus SubTree
void deleteSub(Pohon *node)
{
    if (!root)
        cout << "\n Buat tree terlebih
dahulu!" << endl;
    else
    {
        deleteTree(node->left);
        deleteTree(node->right);
        cout << "\n Node subtree " <<
node->data << " berhasil dihapus." <<
endl;
    }
}
// Hapus Tree
void clear()
{
    if (!root)
        cout << "\n Buat tree terlebih
dahulu!!" << endl;
```

```
    else
    {
        deleteTree(root);
        cout << "\n Pohon berhasil
dihapus." << endl;
    }
}
// Cek Size Tree
int size(Pohon *node = root)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih
dahulu!!" << endl;
        return 0;
    }
    else
    {
        if (!node)
        {
            return 0;
        }
        else
        {
            return 1 + size(node->left) +
size(node->right);
        }
    }
}
// Cek Height Level Tree
int height(Pohon *node = root)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih
dahulu!" << endl;
        return 0;
    }
    else
    {
        if (!node)
        {
            return 0;
        }
        else
        {
```

```
            int heightKiri = height(node->left);
            int heightKanan = height(node->right);
            if (heightKiri >= heightKanan)
            {
                return heightKiri + 1;
            }
            else
            {
                return heightKanan + 1;
            }
        }
    }
}
// Karakteristik Tree
void charateristic()
{
    cout << "\n Size Tree : " << size() << endl;
    cout << " Height Tree : " << height() << endl;
    cout << " Average Node of Tree : " << size() / height() << endl;
}

void tampilkanChild (Pohon *node)
{
    if (node->left != NULL)
    {
        cout << "Child Kiri: " << node->left->data << endl;
    }
    if (node->right != NULL)
    {
        cout << "Child Kanan: " << node->right->data << endl;
    }
}

void tampilkanDescendant (Pohon *node)
{
    if (node == NULL)
    {
        return;
    }
```

```
    cout << node->data << " ";
    tampilkanDescendant(node->left);
    tampilkanDescendant(node->right);
}

Pohon* cariNode (char data, Pohon *node)
{
    if (node == NULL)
    {
        return NULL;
    }
    if (node->data == data)
    {
        cout << "Node " << data << "
ditemukan.\n";
        return node;
    }
    Pohon* foundNode = cariNode(data,
node->left);
    if (foundNode == NULL)
    {
        foundNode == cariNode(data, node-
>right);
    }
    return foundNode;
}

int main()
{
    int
pilihan_Faisal_Khoiruddin_2311102046;
    char data;
    Pohon *node;
    do {
        cout <<
"\n================================" <<
endl;
        cout <<
"|               Menu            |" <<
endl;
        cout <<
"================================" <<
endl;
        cout << "| 1  | Buat
Node                |" << endl;
        cout << "| 2  | Insert
```

```
Left                |" << endl;
        cout << "| 3  | insert
Right               |" << endl;
        cout << "| 4  | Update
Node                |" << endl;
        cout << "| 5  | Retrieve
Node              |" << endl;
        cout << "| 6  | Temukan
Node                |" << endl;
        cout << "| 7  | Preorder
Traversal         |" << endl;
        cout << "| 8  | Inorder
Traversal          |" << endl;
        cout << "| 9  | Postorder
Traversal       |" << endl;
        cout << "| 10 | Delete
Subtree             |" << endl;
        cout << "| 11 | Clear
Tree                 |" << endl;
        cout << "| 12 | Size of
Tree                |" << endl;
        cout << "| 13 | Height of
Tree             |" << endl;
        cout << "| 14 | Characteristic of
Tree   |" << endl;
        cout << "| 15 |
Keluar                      |" << endl;
        cout <<
"================================" <<
endl;
        cout << "Masukkan pilihan : ";
        cin >>
pilihan_Faisal_Khoiruddin_2311102046;
        switch
(pilihan_Faisal_Khoiruddin_2311102046)
        {
            case 1:
                cout << "Masukkan data: ";
                cin >> data;
                buatNode(data);
                break;
            case 2:
                cout << "Masukkan data
node: ";
                cin >> data;
                node = cariNode(data,
```

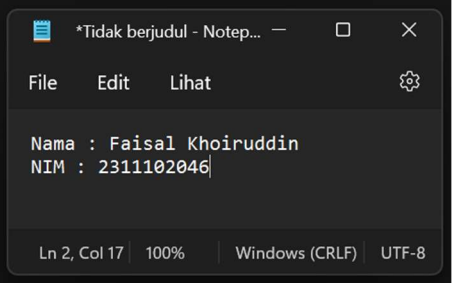
```
root);
                if (node != NULL)
                {
                    cout << "Node
ditemukan. Masukkan data child kiri: ";
                    cin >> data;
                    insertLeft(data,
node);
                } else {
                    cout << "Node tidak
ditemukan,\n";
                }
                break;
            case 3:
                cout << "Masukkan data
node: ";
                cin >> data;
                node = cariNode (data,
root);
                if (node != NULL)
                {
                    cout << "Node
ditemukan. Masukkan data child kanan: ";
                    cin >> data;
                    insertRight(data,
node);
                } else {
                    cout << "Node tidak
ditemukan.\n";
                }
                break;
            case 4:
                cout << "Masukkan data
node: ";
                cin >> data;
                node = cariNode(data,
root);
                if (node != NULL)
                {
                    cout << "Node
ditemukan. Masukkan data baru: ";
                    cin >> data;
                    update(data, node);
                } else {
                    cout << "Node tidak
ditemukan.\n";
```

```
                }
                break;
            case 5:
                cout << "Masukkan data
node: ";
                cin >> data;
                node = cariNode(data,
root);
                retrieve(node);
                break;
            case 6:
                cout << "Masukkan data
node: ";
                cin >> data;
                node = cariNode(data,
root);
                find(node);
                break;
            case 7:
                cout << "Preorder
Traversal: ";
                preOrder(root);
                cout << "\n";
                break;
            case 8:
                cout << "Inorder
Traversal: ";
                inOrder(root);
                cout << "\n";
                break;
            case 9:
                cout << "Postorder
Traversal: ";
                postOrder(root);
                cout << "\n";
                break;
            case 10:
                cout << "Masukkan data
node: ";
                cin >> data;
                node = cariNode(data,
root);
                deleteSub(node);
                break;
            case 11:
                clear();
```

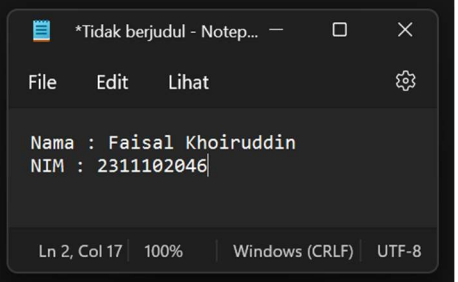
```
                break;
            case 12:
                cout << "Size if Tree: "
<< size(root) << "\n";
                break;
            case 13:
                cout << "Height of Tree: "
<< height(root) << "\n";
                break;
            case 14:
                charateristic();
                break;
            case 15:
                cout << "Keluar dari
program.\n";
                break;
            default:
                cout << "Pilihan tidak
valid. Coba lagi.\n";
        }

    } while
(pilihan_Faisal_Khoiruddin_2311102046 !=
0);
    return 0;
}
```

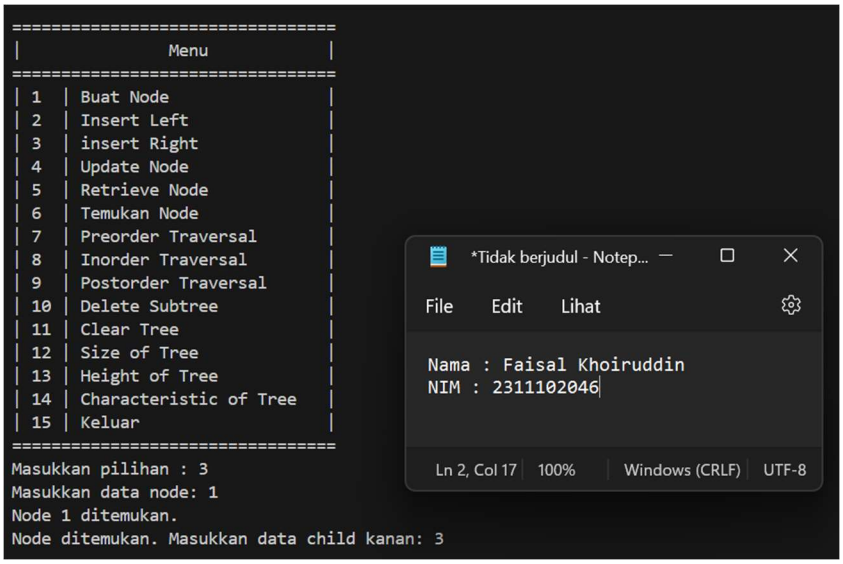
Screenshots Output

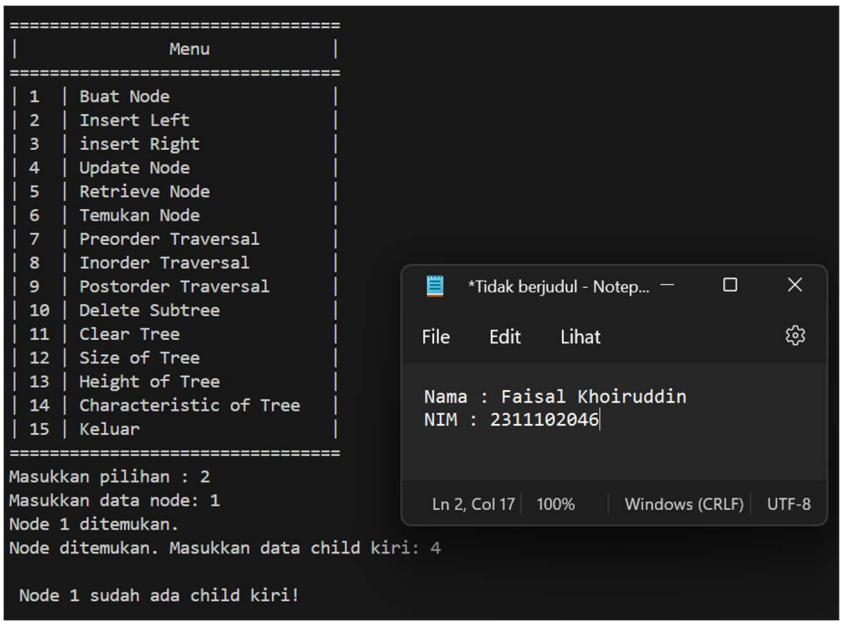
```
PS C:\Users\ASUS\OneDrive\Dokumen\semester 2>  & 'c:\Users\ASUS\.vscode\extensions
\ms-vscode.cpptools-1.20.5-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '
--stdin=Microsoft-MIEngine-In-euonbkje.q1x' '--stdout=Microsoft-MIEngine-Out-ewhgn
dne.akj' '--stderr=Microsoft-MIEngine-Error-1uarubcs.oui' '--pid=Microsoft-MIEngin
e-Pid-nlfgxkbe.tbl' '--dbgExe=c:\mingw32\bin\gdb.exe' '--interpreter=mi'

================================
|              Menu            |
================================
| 1  | Buat Node               |
| 2  | Insert Left             |
| 3  | insert Right            |
| 4  | Update Node             |
| 5  | Retrieve Node           |
| 6  | Temukan Node            |
| 7  | Preorder Traversal      |
| 8  | Inorder Traversal       |
| 9  | Postorder Traversal     |
| 10 | Delete Subtree          |
| 11 | Clear Tree              |
| 12 | Size of Tree            |
| 13 | Height of Tree          |
| 14 | Characteristic of Tree  |
| 15 | Keluar                  |
================================
Masukkan pilihan : 1
Masukkan data: 1

 Node 1 berhasil dibuat menjadi root.
```

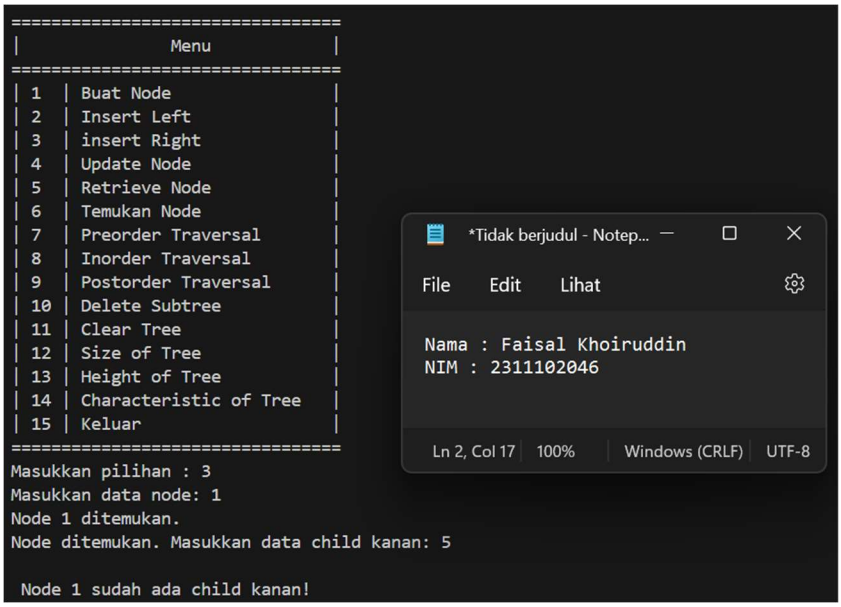


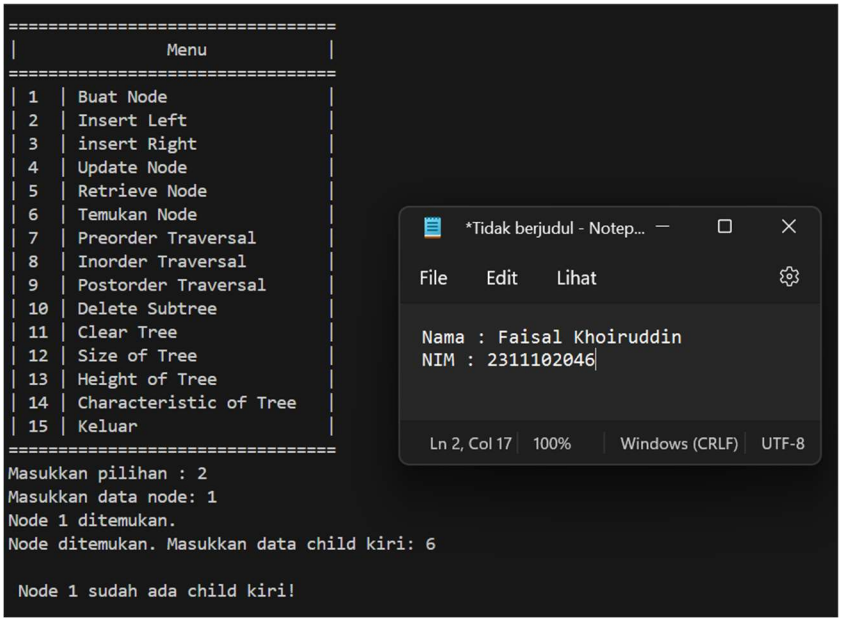

```
================================
|              Menu            |
================================
| 1  | Buat Node               |
| 2  | Insert Left             |
| 3  | insert Right            |
| 4  | Update Node             |
| 5  | Retrieve Node           |
| 6  | Temukan Node            |
| 7  | Preorder Traversal      |
| 8  | Inorder Traversal       |
| 9  | Postorder Traversal     |
| 10 | Delete Subtree          |
| 11 | Clear Tree              |
| 12 | Size of Tree            |
| 13 | Height of Tree          |
| 14 | Characteristic of Tree  |
| 15 | Keluar                  |
================================
Masukkan pilihan : 2
Masukkan data node: 1
Node 1 ditemukan.
Node ditemukan. Masukkan data child kiri: 2

 Node 2 berhasil ditambahkan ke child kiri 1
```

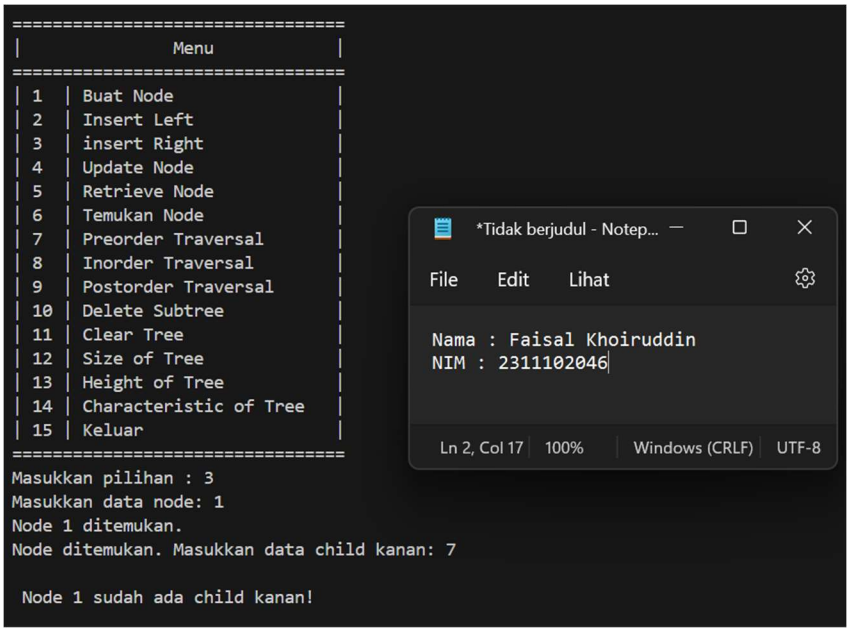


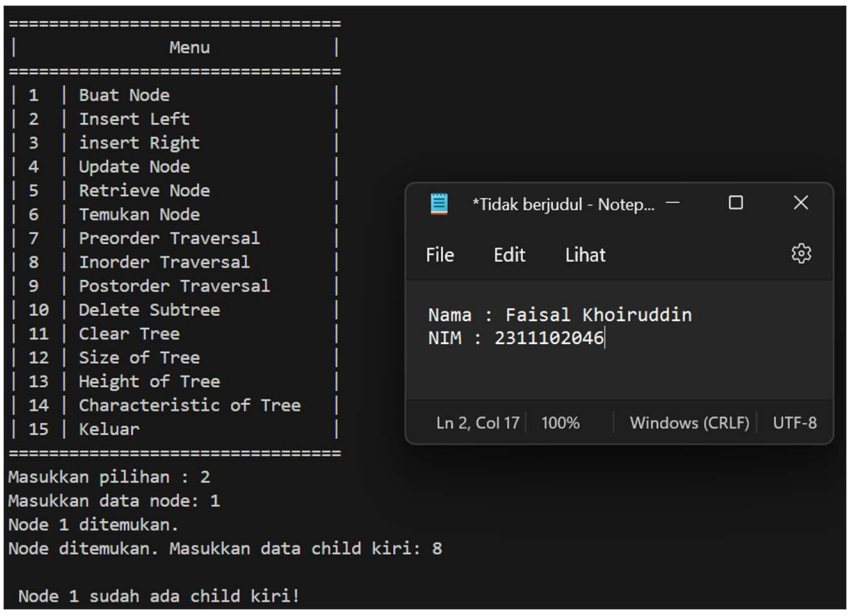

==========
| Menu |
==========
1	Buat Node
2	Insert Left
3	insert Right
4	Update Node
5	Retrieve Node
6	Temukan Node
7	Preorder Traversal
8	Inorder Traversal
9	Postorder Traversal
10	Delete Subtree
11	Clear Tree
12	Size of Tree
13	Height of Tree
14	Characteristic of Tree
15	Keluar
==========
Masukkan pilihan : 3
Masukkan data node: 1
Node 1 ditemukan.
Node ditemukan. Masukkan data child kanan: 3
*Tidak berjudul - Notep...
File Edit Lihat
Nama : Faisal Khoiruddin
NIM : 2311102046
Ln 2, Col 17 100% Windows (CRLF) UTF-8

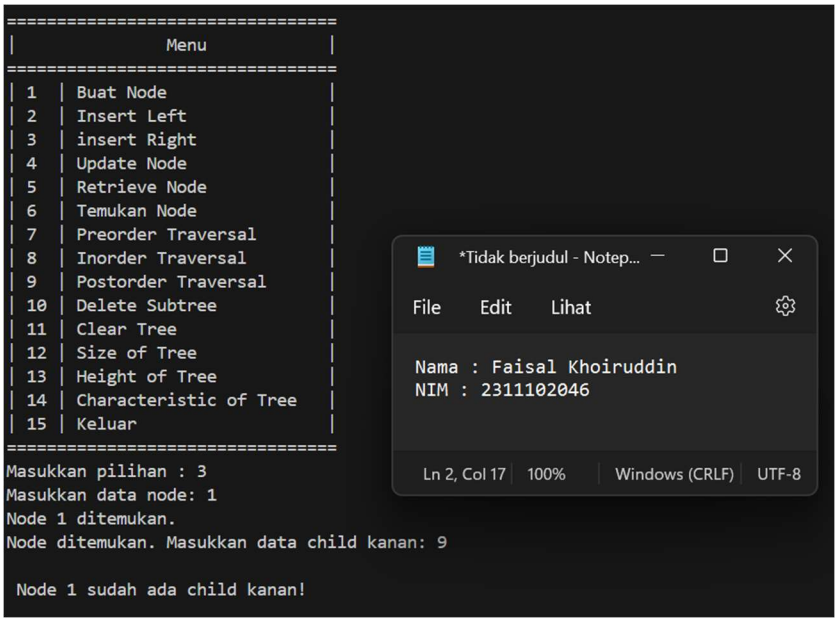

==========
| Menu |
==========
1	Buat Node
2	Insert Left
3	insert Right
4	Update Node
5	Retrieve Node
6	Temukan Node
7	Preorder Traversal
8	Inorder Traversal
9	Postorder Traversal
10	Delete Subtree
11	Clear Tree
12	Size of Tree
13	Height of Tree
14	Characteristic of Tree
15	Keluar
==========
Masukkan pilihan : 2
Masukkan data node: 1
Node 1 ditemukan.
Node ditemukan. Masukkan data child kiri: 4
Node 1 sudah ada child kiri!
*Tidak berjudul - Notep...
File Edit Lihat
Nama : Faisal Khoiruddin
NIM : 2311102046
Ln 2, Col 17 100% Windows (CRLF) UTF-8

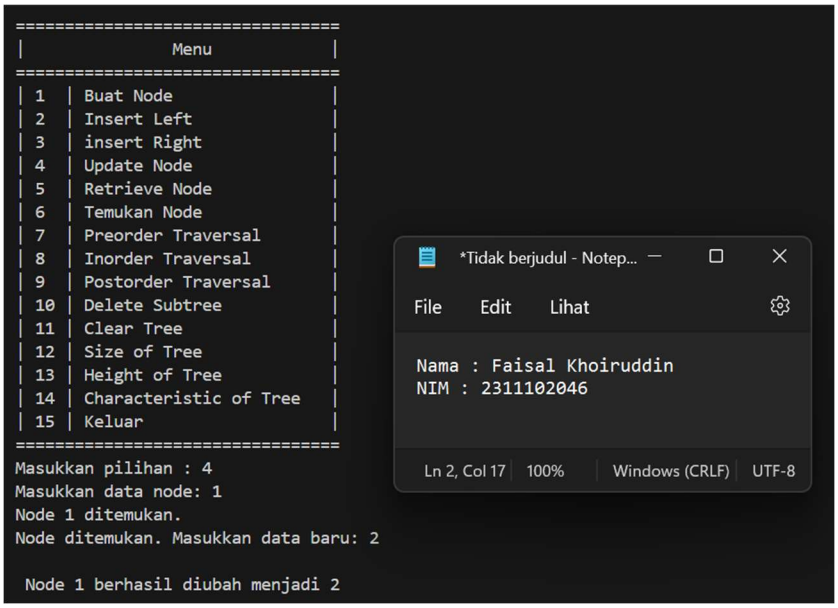

==========
| Menu |
==========
1	Buat Node
2	Insert Left
3	insert Right
4	Update Node
5	Retrieve Node
6	Temukan Node
7	Preorder Traversal
8	Inorder Traversal
9	Postorder Traversal
10	Delete Subtree
11	Clear Tree
12	Size of Tree
13	Height of Tree
14	Characteristic of Tree
15	Keluar
==========
Masukkan pilihan : 3
Masukkan data node: 1
Node 1 ditemukan.
Node ditemukan. Masukkan data child kanan: 5
Node 1 sudah ada child kanan!
*Tidak berjudul - Notep...
File Edit Lihat
Nama : Faisal Khoiruddin
NIM : 2311102046
Ln 2, Col 17 100% Windows (CRLF) UTF-8

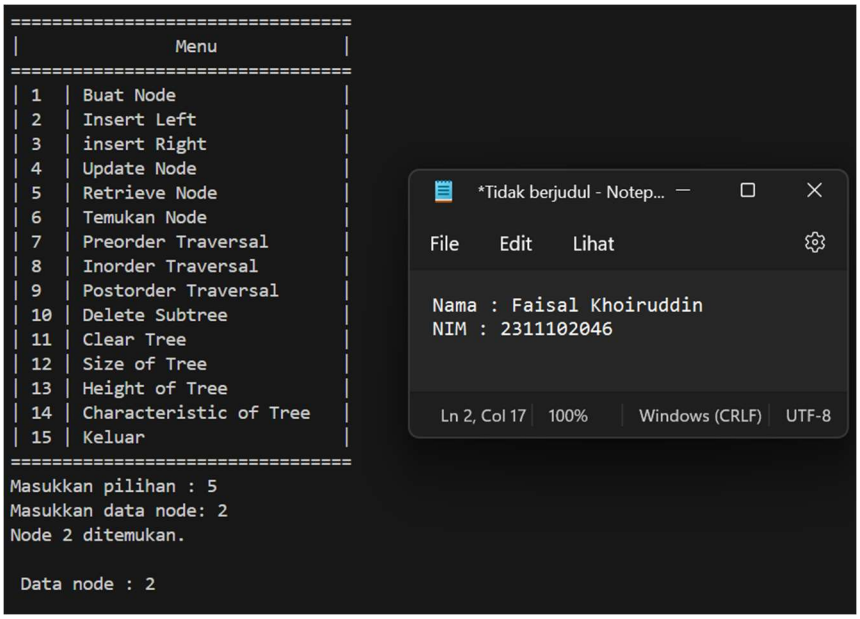

==========
| Menu |
==========
1	Buat Node
2	Insert Left
3	insert Right
4	Update Node
5	Retrieve Node
6	Temukan Node
7	Preorder Traversal
8	Inorder Traversal
9	Postorder Traversal
10	Delete Subtree
11	Clear Tree
12	Size of Tree
13	Height of Tree
14	Characteristic of Tree
15	Keluar
==========
Masukkan pilihan : 2
Masukkan data node: 1
Node 1 ditemukan.
Node ditemukan. Masukkan data child kiri: 6
Node 1 sudah ada child kiri!
*Tidak berjudul - Notep...
File Edit Lihat
Nama : Faisal Khoiruddin
NIM : 2311102046
Ln 2, Col 17 100% Windows (CRLF) UTF-8

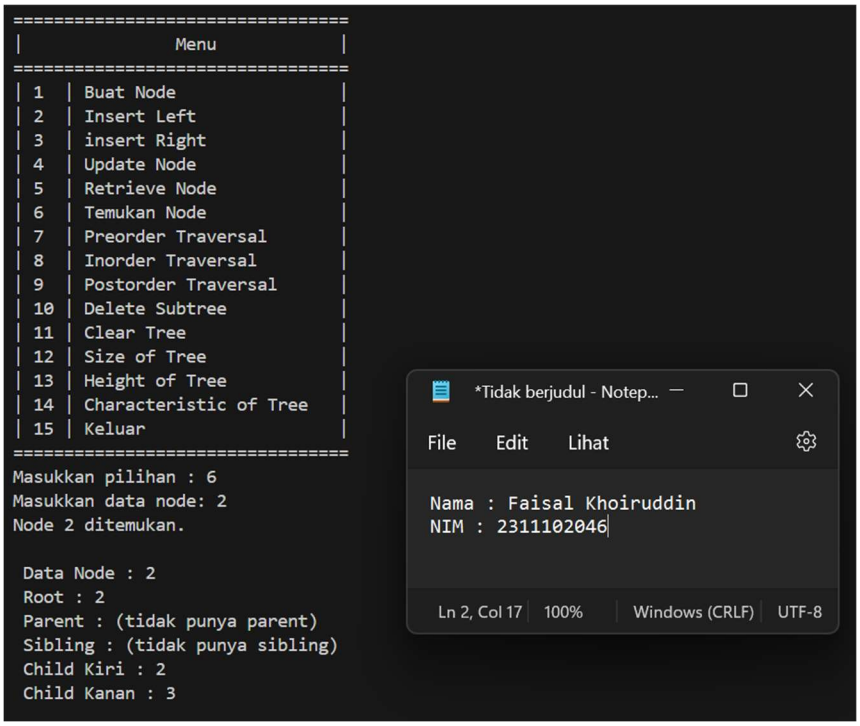

==========
| Menu |
==========
1	Buat Node
2	Insert Left
3	insert Right
4	Update Node
5	Retrieve Node
6	Temukan Node
7	Preorder Traversal
8	Inorder Traversal
9	Postorder Traversal
10	Delete Subtree
11	Clear Tree
12	Size of Tree
13	Height of Tree
14	Characteristic of Tree
15	Keluar
==========
Masukkan pilihan : 3
Masukkan data node: 1
Node 1 ditemukan.
Node ditemukan. Masukkan data child kanan: 7
Node 1 sudah ada child kanan!
*Tidak berjudul - Notep...
File Edit Lihat
Nama : Faisal Khoiruddin
NIM : 2311102046
Ln 2, Col 17 100% Windows (CRLF) UTF-8

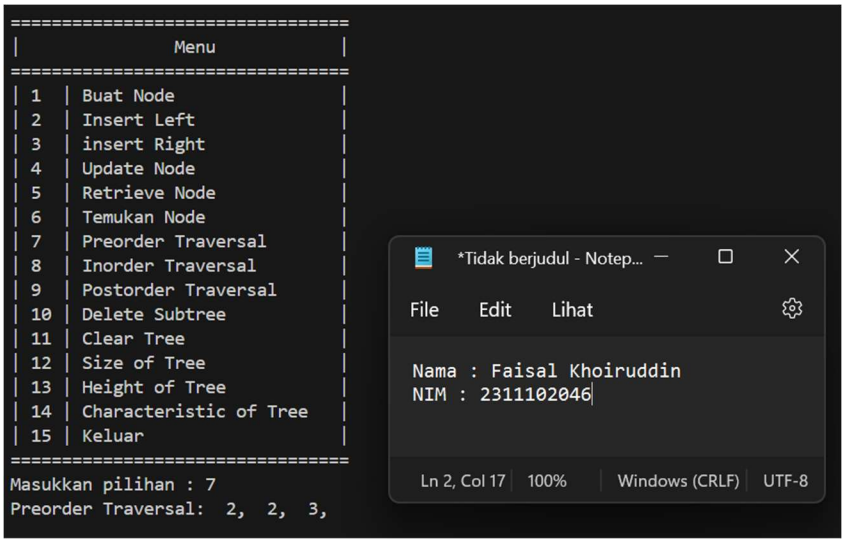

==========
| Menu |
==========
1	Buat Node
2	Insert Left
3	insert Right
4	Update Node
5	Retrieve Node
6	Temukan Node
7	Preorder Traversal
8	Inorder Traversal
9	Postorder Traversal
10	Delete Subtree
11	Clear Tree
12	Size of Tree
13	Height of Tree
14	Characteristic of Tree
15	Keluar
==========
Masukkan pilihan : 2
Masukkan data node: 1
Node 1 ditemukan.
Node ditemukan. Masukkan data child kiri: 8
Node 1 sudah ada child kiri!
*Tidak berjudul - Notep...
File Edit Lihat
Nama : Faisal Khoiruddin
NIM : 2311102046
Ln 2, Col 17 100% Windows (CRLF) UTF-8

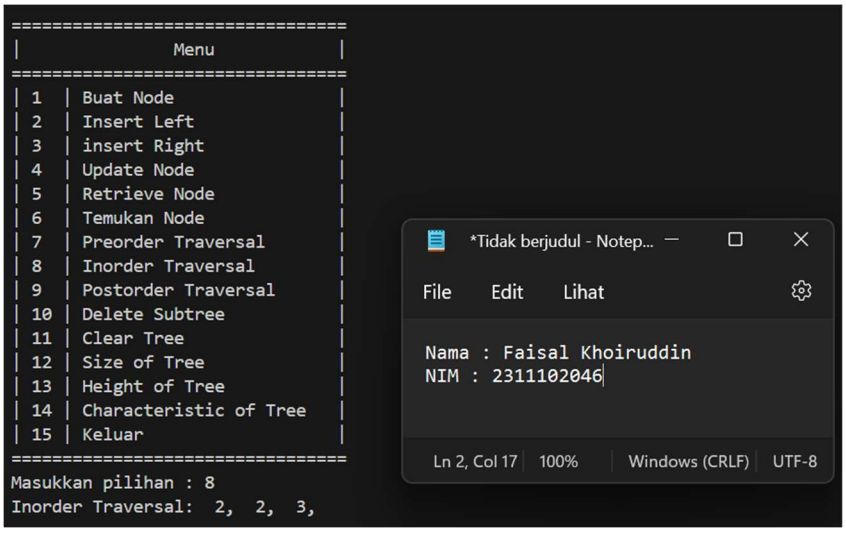

==========
| Menu |
==========
1	Buat Node
2	Insert Left
3	insert Right
4	Update Node
5	Retrieve Node
6	Temukan Node
7	Preorder Traversal
8	Inorder Traversal
9	Postorder Traversal
10	Delete Subtree
11	Clear Tree
12	Size of Tree
13	Height of Tree
14	Characteristic of Tree
15	Keluar
==========
Masukkan pilihan : 3
Masukkan data node: 1
Node 1 ditemukan.
Node ditemukan. Masukkan data child kanan: 9
Node 1 sudah ada child kanan!
*Tidak berjudul - Notep...
File Edit Lihat
Nama : Faisal Khoiruddin
NIM : 2311102046
Ln 2, Col 17 100% Windows (CRLF) UTF-8

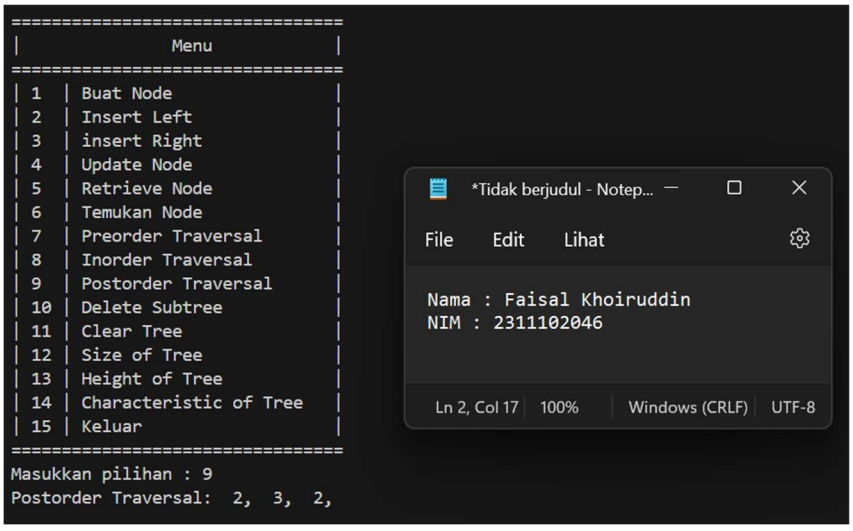

==========
| Menu |
==========
1	Buat Node
2	Insert Left
3	insert Right
4	Update Node
5	Retrieve Node
6	Temukan Node
7	Preorder Traversal
8	Inorder Traversal
9	Postorder Traversal
10	Delete Subtree
11	Clear Tree
12	Size of Tree
13	Height of Tree
14	Characteristic of Tree
15	Keluar
==========
Masukkan pilihan : 4
Masukkan data node: 1
Node 1 ditemukan.
Node ditemukan. Masukkan data baru: 2
Node 1 berhasil diubah menjadi 2
*Tidak berjudul - Notep...
File Edit Lihat
Nama : Faisal Khoiruddin
NIM : 2311102046
Ln 2, Col 17 100% Windows (CRLF) UTF-8

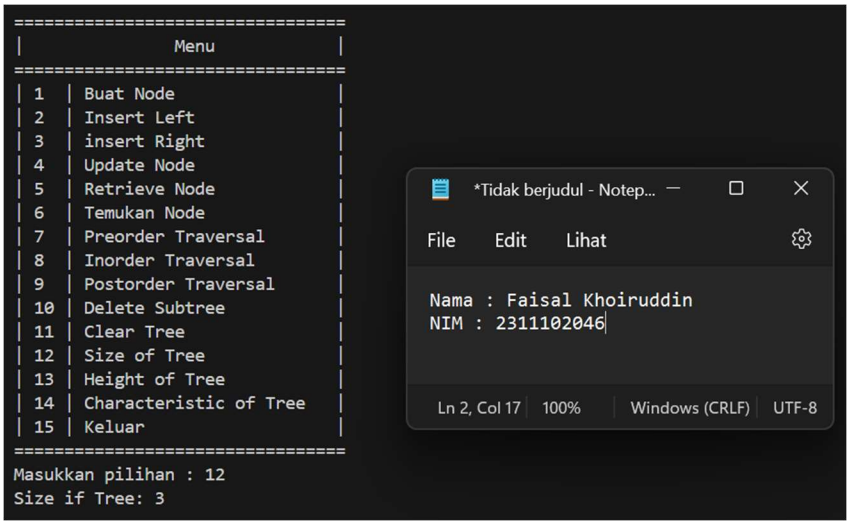

==========
| Menu |
==========
1	Buat Node
2	Insert Left
3	insert Right
4	Update Node
5	Retrieve Node
6	Temukan Node
7	Preorder Traversal
8	Inorder Traversal
9	Postorder Traversal
10	Delete Subtree
11	Clear Tree
12	Size of Tree
13	Height of Tree
14	Characteristic of Tree
15	Keluar
==========
Masukkan pilihan : 5
Masukkan data node: 2
Node 2 ditemukan.
Data node : 2
*Tidak berjudul - Notep...
File Edit Lihat
Nama : Faisal Khoiruddin
NIM : 2311102046
Ln 2, Col 17 100% Windows (CRLF) UTF-8

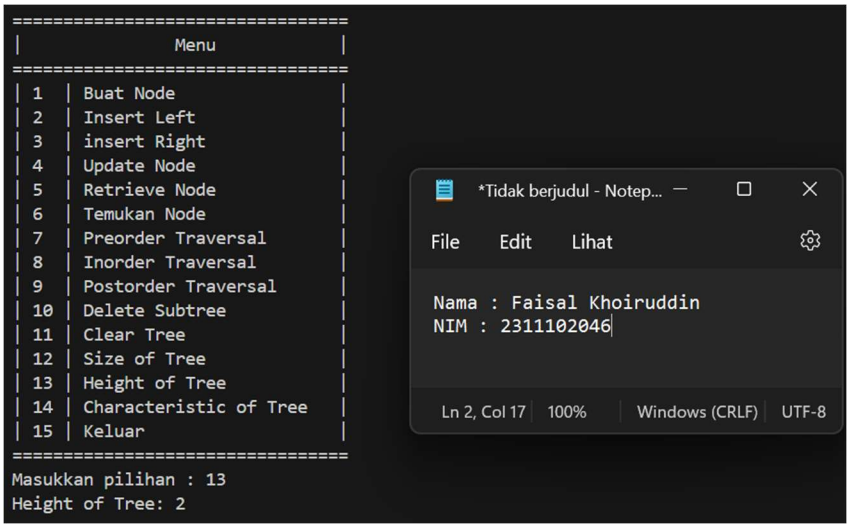

==========
| Menu |
==========
1	Buat Node
2	Insert Left
3	insert Right
4	Update Node
5	Retrieve Node
6	Temukan Node
7	Preorder Traversal
8	Inorder Traversal
9	Postorder Traversal
10	Delete Subtree
11	Clear Tree
12	Size of Tree
13	Height of Tree
14	Characteristic of Tree
15	Keluar
==========
Masukkan pilihan : 6
Masukkan data node: 2
Node 2 ditemukan.
Data Node : 2
Root : 2
Parent : (tidak punya parent)
Sibling : (tidak punya sibling)
Child Kiri : 2
Child Kanan : 3
*Tidak berjudul - Notep...
File Edit Lihat
Nama : Faisal Khoiruddin
NIM : 2311102046
Ln 2, Col 17 100% Windows (CRLF) UTF-8

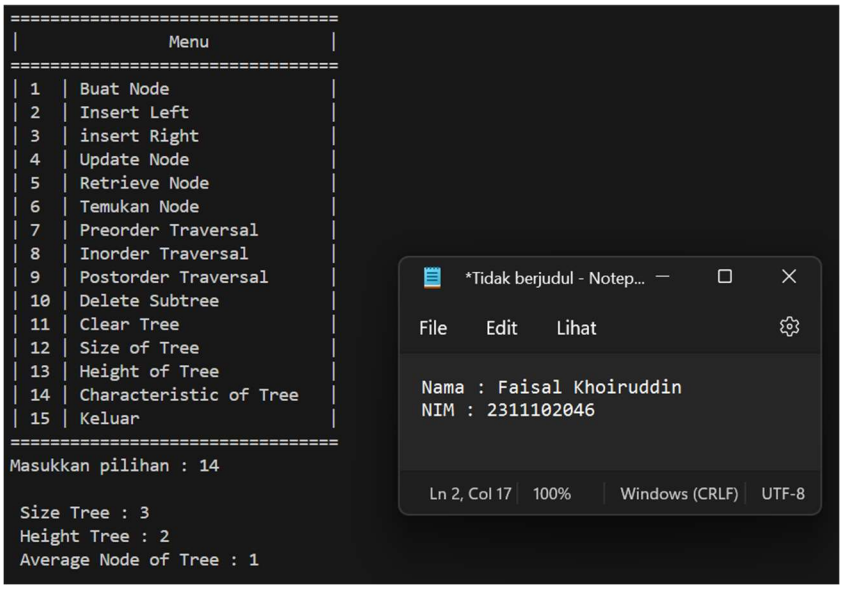

==========
| Menu |
==========
1	Buat Node
2	Insert Left
3	insert Right
4	Update Node
5	Retrieve Node
6	Temukan Node
7	Preorder Traversal
8	Inorder Traversal
9	Postorder Traversal
10	Delete Subtree
11	Clear Tree
12	Size of Tree
13	Height of Tree
14	Characteristic of Tree
15	Keluar
==========
Masukkan pilihan : 7
Preorder Traversal: 2, 2, 3,
*Tidak berjudul - Notep...
File Edit Lihat
Nama : Faisal Khoiruddin
NIM : 2311102046
Ln 2, Col 17 100% Windows (CRLF) UTF-8

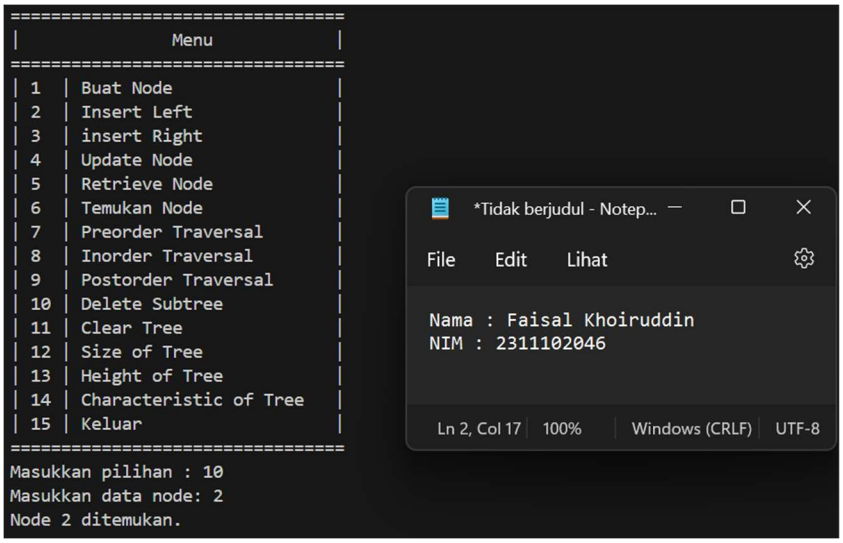

==========
| Menu |
==========
1	Buat Node
2	Insert Left
3	insert Right
4	Update Node
5	Retrieve Node
6	Temukan Node
7	Preorder Traversal
8	Inorder Traversal
9	Postorder Traversal
10	Delete Subtree
11	Clear Tree
12	Size of Tree
13	Height of Tree
14	Characteristic of Tree
15	Keluar
==========
Masukkan pilihan : 8
Inorder Traversal: 2, 2, 3,
*Tidak berjudul - Notep...
File Edit Lihat
Nama : Faisal Khoiruddin
NIM : 2311102046
Ln 2, Col 17 100% Windows (CRLF) UTF-8

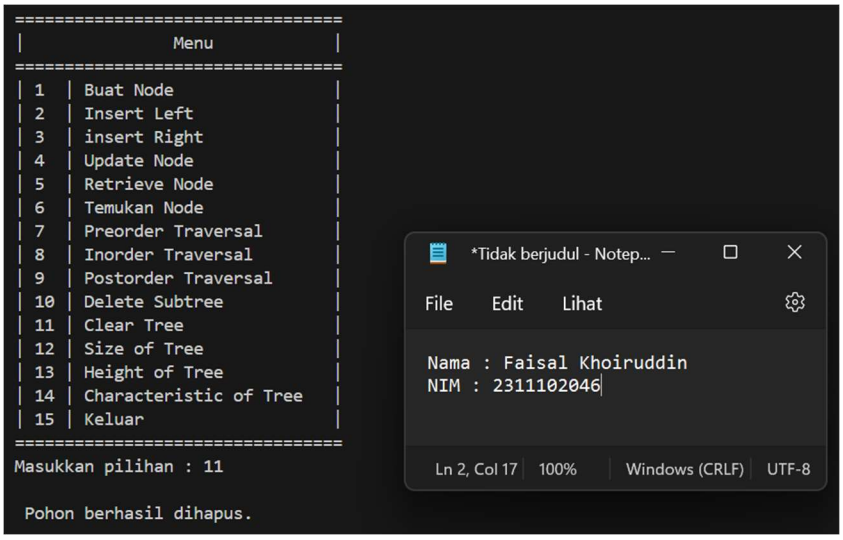

==========
| Menu |
==========
1	Buat Node
2	Insert Left
3	insert Right
4	Update Node
5	Retrieve Node
6	Temukan Node
7	Preorder Traversal
8	Inorder Traversal
9	Postorder Traversal
10	Delete Subtree
11	Clear Tree
12	Size of Tree
13	Height of Tree
14	Characteristic of Tree
15	Keluar
==========
Masukkan pilihan : 9
Postorder Traversal: 2, 3, 2,
*Tidak berjudul - Notep...
File Edit Lihat
Nama : Faisal Khoiruddin
NIM : 2311102046
Ln 2, Col 17 100% Windows (CRLF) UTF-8

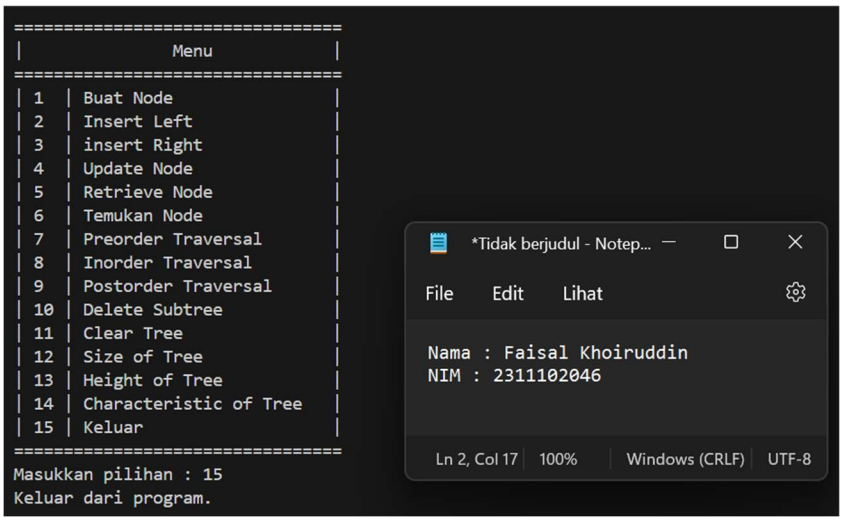

==========
| Menu |
==========
1	Buat Node
2	Insert Left
3	insert Right
4	Update Node
5	Retrieve Node
6	Temukan Node
7	Preorder Traversal
8	Inorder Traversal
9	Postorder Traversal
10	Delete Subtree
11	Clear Tree
12	Size of Tree
13	Height of Tree
14	Characteristic of Tree
15	Keluar
==========
Masukkan pilihan : 12
Size if Tree: 3
*Tidak berjudul - Notep...
File Edit Lihat
Nama : Faisal Khoiruddin
NIM : 2311102046
Ln 2, Col 17 100% Windows (CRLF) UTF-8


==========
| Menu |
==========
1	Buat Node
2	Insert Left
3	insert Right
4	Update Node
5	Retrieve Node
6	Temukan Node
7	Preorder Traversal
8	Inorder Traversal
9	Postorder Traversal
10	Delete Subtree
11	Clear Tree
12	Size of Tree
13	Height of Tree
14	Characteristic of Tree
15	Keluar
==========
Masukkan pilihan : 13
Height of Tree: 2
*Tidak berjudul - Notep...
File Edit Lihat
Nama : Faisal Khoiruddin
NIM : 2311102046
Ln 2, Col 17 100% Windows (CRLF) UTF-8


==========
| Menu |
==========
1	Buat Node
2	Insert Left
3	insert Right
4	Update Node
5	Retrieve Node
6	Temukan Node
7	Preorder Traversal
8	Inorder Traversal
9	Postorder Traversal
10	Delete Subtree
11	Clear Tree
12	Size of Tree
13	Height of Tree
14	Characteristic of Tree
15	Keluar
==========
Masukkan pilihan : 14
Size Tree : 3
Height Tree : 2
Average Node of Tree : 1
*Tidak berjudul - Notep...
File Edit Lihat
Nama : Faisal Khoiruddin
NIM : 2311102046
Ln 2, Col 17 100% Windows (CRLF) UTF-8


==========
| Menu |
==========
1	Buat Node
2	Insert Left
3	insert Right
4	Update Node
5	Retrieve Node
6	Temukan Node
7	Preorder Traversal
8	Inorder Traversal
9	Postorder Traversal
10	Delete Subtree
11	Clear Tree
12	Size of Tree
13	Height of Tree
14	Characteristic of Tree
15	Keluar
==========
Masukkan pilihan : 10
Masukkan data node: 2
Node 2 ditemukan.
*Tidak berjudul - Notep...
File Edit Lihat
Nama : Faisal Khoiruddin
NIM : 2311102046
Ln 2, Col 17 100% Windows (CRLF) UTF-8


==========
| Menu |
==========
1	Buat Node
2	Insert Left
3	insert Right
4	Update Node
5	Retrieve Node
6	Temukan Node
7	Preorder Traversal
8	Inorder Traversal
9	Postorder Traversal
10	Delete Subtree
11	Clear Tree
12	Size of Tree
13	Height of Tree
14	Characteristic of Tree
15	Keluar
==========
Masukkan pilihan : 11
Pohon berhasil dihapus.
*Tidak berjudul - Notep...
File Edit Lihat
Nama : Faisal Khoiruddin
NIM : 2311102046
Ln 2, Col 17 100% Windows (CRLF) UTF-8


==========
| Menu |
==========
1	Buat Node
2	Insert Left
3	insert Right
4	Update Node
5	Retrieve Node
6	Temukan Node
7	Preorder Traversal
8	Inorder Traversal
9	Postorder Traversal
10	Delete Subtree
11	Clear Tree
12	Size of Tree
13	Height of Tree
14	Characteristic of Tree
15	Keluar
==========
Masukkan pilihan : 15
Keluar dari program.
*Tidak berjudul - Notep...
File Edit Lihat
Nama : Faisal Khoiruddin
NIM : 2311102046
Ln 2, Col 17 100% Windows (CRLF) UTF-8

Deskripsi:

Program tersebut merupakan program tree. Program tersebut yaitu membuat tree dengan fungsi tambahan untuk menampilkan node child dan descendant. Pengguna menginputkan pilihan dari kemudian menginputkan angka yang merupakan child. Kemudian muncul output dari opsi dari yang dipilih oleh pengguna.

- #include <iostream> ➔ merupakan input output stream header yang digunakan sebagai standar input output operasi yang digunakan di c++
- using namespace std; ➔ digunakan untuk mendeklarasikan/ memberitahukan kepada compiler bahwa kita akan menggunakan semua fungsi/class/file yang terdapat dalam namespace std
- struct Pohon ➔ Deklarasi struct Pohon
- char data; ➔ deklarasi variabel data dalam struct pohon
- Pohon *left, *right, *parent; ➔ pointer ke anak kiri, anak kanan, dan orang tua
- Pohon *root, *baru; ➔ deklarasi pointer untuk root pohon dan node baru
- void init() ➔ Inisialisasi pohon
- root = NULL; ➔ mengatur root menjadi NULL
- int isEmpty() ➔ Cek Node jika kosong
- if (root == NULL) ➔ jika root adalah kosong
- return 1; ➔ mengembalikan 1 true
- else ➔ kalua tidak
- return 0; ➔ mengembalikan 0 false
- void buatNode(char data) ➔ fungsi buat node baru dengan parameter data tipe data char
- if (isEmpty() == 1) ➔ jika pohon kosong = 1

- root = new Pohon(); ➔ membuat node baru mengisi ke root
- root->data = data; ➔ mengatur data dari node root
- root->left = NULL; ➔ mengatur anak kiri dari root menjadi NULL
- root->right = NULL; ➔ mengatur anak kanan dari root menjadi NULL
- root->parent = NULL; ➔ mengatur parent dari root menjadi NULL
- cout << "\n Node " << data << " berhasil dibuat menjadi root." << endl; ➔ menampilkan pesan berhasil
- else ➔ kalau tidak
- cout << "\n Pohon sudah dibuat" << endl; ➔ menampilkan pesan pohon sudah dibuat
- Pohon *insertLeft(char data, Pohon *node) ➔ Tambah Kiri
- if (isEmpty() == 1) ➔ jika pohon kosong
- cout << "\n Buat tree terlebih dahulu!" << endl; ➔ menampilkan pesan buat tree terlebih dahulu
- return NULL; ➔ mengembalikan NULL
- else ➔ kalau tidak
- if (node->left != NULL) ➔ cek apakah child kiri ada atau tidak
- cout << "\n Node " << node->data << " sudah ada child kiri!" << endl; ➔ menampilkan pesan sudah ada
- return NULL; ➔ mengembalikan NULL
- else ➔ kalau tidak ada
- baru = new Pohon(); ➔ membuat node baru
- baru->data = data; ➔ mengatur data dari node baru
- baru->left = NULL; ➔ mengatur anak kiri dari node baru menjadi NULL
- baru->right = NULL; ➔ mengatur anak kanan dari node baru menjadi NULL

- baru->parent = node; ➔ mengatur orang tua dari node baru menjadi node yang diberikan
- node->left = baru; ➔ mengatur anak kiri dari node yang diberikan menjadi node baru
- cout << "\n Node " << data << " berhasil ditambahkan ke child kiri " << baru->parent->data << endl; ➔ menampilkan pesan berhasil
- return baru; ➔ mengembalikan pointer ke node baru
- Pohon *insertRight(char data, Pohon *node) ➔ menambahkan node kanan
- if (root == NULL) ➔ jika root kosong
- cout << "\n Buat tree terlebih dahulu!" << endl; ➔ menampilkan statement buat tree terlebih dahulu
- return NULL; ➔ mengembalikan NULL
- else ➔ kalau tidak
- if (node->right != NULL) ➔ jika node kanan tidak NULL
- cout << "\n Node " << node->data << " sudah ada child kanan!" << endl; ➔ menampilkan statement sudah ada
- return NULL; ➔ mengembalikan NULL
- else ➔ kalau tidak
- baru = new Pohon(); ➔ membuat node baru
- baru->data = data; ➔ mengatur data dari node baru
- baru->left = NULL; ➔ mengatur anak kiri dari node baru menjadi NULL
- baru->right = NULL; ➔ mengatur anak kanan dari node baru menjadi NULL
- baru->parent = node; ➔ mengatur parent dari node yang diberikan menjadi node baru
- node->right = baru; ➔ mengatur anak kanan dari node yang diberikan menjadi node baru
- cout << "\n Node " << data << " berhasil ditambahkan ke child

kanan " << baru->parent->data << endl; ➔ menampilkan statement berhasil

- return baru; ➔ mengembalikan pointer ke node baru
- void update(char data, Pohon *node) ➔ fungsi untuk mengupdate data dalam node
- if (isEmpty() == 1) ➔ jika kosong sama dengan 1
- cout << "\n Buat tree terlebih dahulu!" << endl; ➔ menampilkan statement buat tree terlebih dahulu
- else ➔ kalau tidak
- if (!node) ➔ jika node tidak ada
- cout << "\n Node yang ingin diganti tidak ada!!" << endl; ➔ menampilkan statement Node yang ingin diganti tidak ada!
- Else ➔ kalau tidak
- char temp = node->data; ➔ menyimpan data saat ini dari node
- node->data = data; ➔ memperbarui data dari node
- cout << "\n Node " << temp << " berhasil diubah menjadi " << data << endl; ➔ menampilkan statement Node berhasil diubah menjadi dilanjutkan memanggil variabel data
- void retrieve(Pohon *node) ➔ fungsi untuk mengambil data dalam node
- if (!root) ➔ jika tidak ada root
- cout << "\n Buat tree terlebih dahulu!" << endl; ➔ menampilkan statement Buat tree terlebih dahulu
- else ➔ kalau tidak
- if (!node) ➔ jika tidak ada node
- cout << "\n Node yang ditunjuk tidak ada!" << endl; ➔ menampilkan statement Node yang ditunjuk tidak ada!
- Else ➔ kalau tidak
- cout << "\n Data node : " << node->data << endl; ➔ menampilkan data dari node

- void find(Pohon *node) ➔ fungsi mencari node dalam pohon
- if (!root) ➔ jika tidak ada root
- cout << "\n Buat tree terlebih dahulu!" << endl; ➔ menampilkan statement Buat tree terlebih dahulu!
- Else ➔ kalau tidak
- if (!node) ➔ jika tidak ada node
- cout << "\n Node yang ditunjuk tidak ada!" << endl; ➔ menampilkan statement Node yang ditunjuk tidak ada!
- Else ➔ kalau tidak
- cout << "\n Data Node : " << node->data << endl; ➔ data dari node
- cout << " Root : " << root->data << endl; ➔ menampilkan data dari root
- if (!node->parent) ➔ jika tidak memiliki orang tua
- cout << " Parent : (tidak punya parent)" << endl; ➔ menampilkan statement Parent : (tidak punya parent)
- else ➔ kalau node memiliki orang tua
- cout << " Parent : " << node->parent->data << endl; ➔ menampilkan data dari orang tua
- if (node->parent != NULL && node->parent->left != node && node->parent->right == node) ➔ jika node memiliki saudara kiri
- cout << " Sibling : " << node->parent->left->data << endl; ➔ menampilkan data dari saudara
- else if (node->parent != NULL && node->parent->right != node && node->parent->left == node) ➔ jika node memiliki saudara kanan
- cout << " Sibling : " << node->parent->right->data << endl; ➔ menampilkan data dari saudara
- else ➔ jika node tidak memiliki saudara
- cout << " Sibling : (tidak punya sibling)" << endl; ➔

menampilkan statement Sibling : (tidak punya sibling)

- if (!node->left) ➔ jika node tidak memiliki anak kiri
- cout << " Child Kiri : (tidak punya Child kiri)" << endl; ➔ menampilkan pesan Child Kiri : (tidak punya Child kiri)
- else ➔ jika node memiliki anak kiri
- cout << " Child Kiri : " << node->left->data << endl; ➔ menampilkan data dari anak kiri
- if (!node->right) ➔ jika node tidak memiliki anak kanan
- cout << " Child Kanan : (tidak punya Child kanan)" << endl; ➔ menampilkan Child Kanan : (tidak punya Child kanan)
- else ➔ jika node memiliki anak kanan
- cout << " Child Kanan : " << node->right->data << endl; ➔ menampilkan data dari anak kanan
- void preOrder(Pohon *node = root) ➔ fungsi untuk melakukan penelusuran pohon dalam preorder
- if (!root) ➔ jika tidak ada root
- cout << "\n Buat tree terlebih dahulu!" << endl; ➔ menampilkan statement Buat tree terlebih dahulu!
- Else ➔ k alau tidak
- if (node != NULL)➔ jika node tidak kosong
- cout << " " << node->data << ", "; ➔ menampilkan data dari node
- preOrder(node->left); ➔ melakukan penelusuran preorder pada subtree kiri secara rekursif
- preOrder(node->right); ➔ melakukan penelusuran preorder pada subtree kanan secara rekursif
- void inOrder(Pohon *node = root) ➔ fungsi melakukan penelusuran dalam inorder
- if (!root) ➔ jika tidak ada root
- cout << "\n Buat tree terlebih dahulu!" << endl; ➔ menampilkan statement Buat tree terlebih dahulu!

- Else ➔ kalau tidak
- if (node != NULL) ➔ jika node ada
- inOrder(node->left); ➔ melakukan penelusuran secara inorder pada subtree kiri secara rekursif
- cout << " " << node->data << ", "; ➔ menampilkan data dari node
- inOrder(node->right); ➔ melakukan penelusuran secara inorder pada subtree kanan secara rekursif
- void postOrder(Pohon *node = root) ➔ fungsi melakukan penelusuran secara postorder
- if (!root) ➔ jika tidak ada root
- cout << "\n Buat tree terlebih dahulu!" << endl;
- else ➔ kalau tidak
- if (node != NULL) ➔ jika node tidak kosong
- postOrder(node->left); ➔ melakukan penelusuran postorder pada subtree kiri secara rekursif
- postOrder(node->right); ➔ melakukan penelusuran postorder pada subtree kanan secara rekursif
- cout << " " << node->data << ", "; ➔ menampilkan data dari node
- void deleteTree(Pohon *node) ➔ fungsi untuk menghapus node pohon
- if (!root) ➔ jika tidak ada root
- cout << "\n Buat tree terlebih dahulu!" << endl; ➔ menampilkan statement Buat tree terlebih dahulu!
- Else ➔ kalau tidak
- if (node != NULL) ➔ jika node tidak NULL
- if (node != root) ➔ jika node tidak sama dengan root
- node->parent->left = NULL; ➔ mengatur anak kiri dari orang tua menjadi NULL

- node->parent->right = NULL; ➔ mengatur anak kanan dari orang tua menjadi NULL
- deleteTree(node->left); ➔ menghapus subtree kiri secara rekursif
- deleteTree(node->right); ➔ menghapus subtree kanan secara rekursif
- if (node == root) ➔ jika node adalah root
- delete root; ➔ menghapus root
- root = NULL; ➔ mengatur root menjadi NULL
- else ➔ kalau tidak
- delete node; ➔ menghapus node
- void deleteSub(Pohon *node) ➔ fungsi untuk menghapus subtree
- if (!root) ➔ jika tidak ada root
- cout << "\n Buat tree terlebih dahulu!" << endl; ➔ cout menampilkan staetemnet Buat tree terlebih dahulu!
- Else ➔ kalau tidak
- deleteTree(node->left); ➔ menghapus subtree kiri
- deleteTree(node->right); ➔ menghapus subtree kanan
- cout << "\n Node subtree " << node->data << " berhasil dihapus." << endl; ➔ menampilkan statement berhasil
- void clear() ➔ fungsi untuk menghapus pohon
- if (!root) ➔ jika tidak ada root
- cout << "\n Buat tree terlebih dahulu!!" << endl; ➔ menampilkan statement Buat tree terlebih dahulu!
- Else ➔ kalau tidak
- deleteTree(root); ➔ menghapus pohon
- cout << "\n Pohon berhasil dihapus." << endl; ➔ menampikan statement Pohon berhasil dihapus
- int size(Pohon *node = root) ➔ fungsi untuk memeriksa

ukuran pohon

- if (!root) ➔ jika tidak ada root
- cout << "\n Buat tree terlebih dahulu!!" << endl; ➔ menampilkan statement Buat tree terlebih dahulu!!
- return 0; ➔ mengembalikan 0
- else ➔ kalau tidak
- if (!node) ➔ jika tidak ada node
- return 0; ➔ mengembalikan 0
- else ➔ kalau tidak
- return 1 + size(node->left) + size(node->right); ➔ mengembalikan jumlah node dalam pohon
- int height(Pohon *node = root) ➔ fungsi untuk memeriksa ketinggian pohon
- if (!root) ➔ jika tidak ada root
- cout << "\n Buat tree terlebih dahulu!" << endl; ➔ menampilkan statement Buat tree terlebih dahulu!
- return 0; ➔ mengembalika 0
- else ➔ kalau tidak
- if (!node) ➔ kalai tidak ada node
- return 0; ➔ mengembalikan 0
- else ➔ kalau tidak
- int heightKiri = height(node->left); ➔ menghitung ketinggian subtree kiri
- int heightKanan = height(node->right); ➔ menghitung ketinggian subtree kanan
- if (heightKiri >= heightKanan) ➔ jika tinggi subtree kiri >= tinggi subtree kanan
- return heightKiri + 1; ➔ mengembalikan tinggi subtree kiri tambah 1
- else ➔ kalau tidak

- return heightKanan + 1; ➔ mengembalikan tinggi subtree kana tambah 1
- void charateristic() ➔ fungsi untuk menampilkan karakteristik pohon
- cout << "\n Size Tree : " << size() << endl; ➔ menampilkan ukuran pohon
- cout << " Height Tree : " << height() << endl; ➔ menampilkan tinggi pohon
- cout << " Average Node of Tree : " << size() / height() << endl;➔ menampilkan rata-rata node dalam pohon
- void tampilkanChild (Pohon *node) ➔ fungsi untuk menampilkan anak dari node
- if (node->left != NULL) ➔ jika node memiliki anak kiri
- cout << "Child Kiri: " << node->left->data << endl; ➔ menampilkan data dari anak kiri
- if (node->right != NULL) ➔ jika node memiliki anak kanan
- cout << "Child Kanan: " << node->right->data << endl; ➔ menampilkan data dari anak kanan
- void tampilkanDescendant (Pohon *node) ➔ fungsi untuk menampikan keturunan dari node
- if (node == NULL) ➔ jika node tidak ada
- return; ➔ mengembalikan
- cout << node->data << " "; ➔ menampilkan data dari node
- tampilkanDescendant(node->left); ➔ menampilkan keturunan dari anak kiri secara rekursif
- tampilkanDescendant(node->right); ➔ menampilkan keturunan dari anak kanan secara rekursif
- Pohon* cariNode (char data, Pohon *node) ➔ mencari node dalam pohon
- if (node == NULL) ➔ jika node tidak ada
- return NULL; ➔ mengembalikan NULL

- if (node->data == data) ➔ jika data dari node sama dengan yang dicari
- cout << "Node " << data << " ditemukan.\n"; ➔ menampilkan statement sukses
- return node; ➔ mengembalikan node
- Pohon* foundNode = cariNode(data, node->left); ➔ mencari node dalam subtree kiri
- if (foundNode == NULL) ➔ jika node tidak ditemukan dalam subtree kiri
- foundNode == cariNode(data, node->right); ➔ mencari node dalam subtree kanan
- return foundNode; ➔ mengembalikan node yang ditemukan
- int main() ➔ merupakan fungsi utama
- int pilihan_Faisal_Khoiruddin_2311102046; ➔ deklarasi variabel pilihan
- char data; ➔ deklarasi variabel data tipe data char
- Pohon *node; ➔ pointer ke node yang akan diproses
- do { ➔ melakukan perulangan do while
- cout << "\n================================" << endl; ➔ menampilkan garis pembatas
- cout << "| Menu |" << endl; ➔ menampilkan judul menu
- cout << "================================" << endl; ➔ menampilkan garis pembatas
- cout << "| 1 | Buat Node |" << endl; ➔ menampilkan pilihan menu buat node
- cout << "| 2 | Insert Left |" << endl; ➔ menampilkan pilihan menu Insert Left
- cout << "| 3 | insert Right |" << endl; ➔ menampilkan pilihan menu Insert Right
- cout << "| 4 | Update Node |" << endl; ➔ ➔

menampilkan pilihan menu Update node

- cout << "| 5 | Retrieve Node |" << endl; ➔ menampilkan pilihan menu retrieve node
- cout << "| 6 | Temukan Node |" << endl; ➔ menampilkan pilihan menu temukan node
- cout << "| 7 | Preorder Traversal |" << endl; ➔ menampilkan pilihan menu preorder traversal
- cout << "| 8 | Inorder Traversal |" << endl; ➔ menampilkan pilihan menu inorder traversal
- cout << "| 9 | Postorder Traversal |" << endl; ➔ menampilkan pilihan menu postorder traversal
- cout << "| 10 | Delete Subtree |" << endl; ➔ menampilkan pilihan menu delete subtree
- cout << "| 11 | Clear Tree |" << endl; ➔ menampilkan pilihan menu clear tree
- cout << "| 12 | Size of Tree |" << endl; ➔ menampilkan pilihan menu size of tree
- cout << "| 13 | Height of Tree |" << endl; ➔ menampilkan pilihan menu height of tree
- cout << "| 14 | Characteristic of Tree |" << endl; ➔ menampilkan pilihan menu Characteristic of Tree
- cout << "| 15 | Keluar |" << endl; ➔ ➔ menampilkan pilihan menu Keluar
- cout << "==========" << endl; ➔ menampilkan garis pembatas
- cout << "Masukkan pilihan : "; ➔ menampilkan pesan perintah masukkan pilihan
- cin >> pilihan_Faisal_Khoiruddin_2311102046; ➔ menerima input pengguna dan menyimpan ke dalam variabel pilihan
- switch (pilihan_Faisal_Khoiruddin_2311102046)
- case 1: ➔ jika pengguna memilih opsi 1
- cout << "Masukkan data: "; ➔ menampilkan statement

Masukkan data:

- cin >> data; ➔ menerima input pengguna dan menyimpan ke dalam variabel data
- buatNode(data);➔ membuat node baru dengan data yang diberikan pengguna
- break; ➔ mengakhiri case 1
- case 2: ➔ jika pengguna memilih case 2
- cout << "Masukkan data node: "; ➔ menampilkan statement Masukkan data node:
- cin >> data; ➔ menerima input pengguna dan menyimpan ke dalam variabel data
- node = cariNode(data, root);➔ mencari node dengan data yang dibelikan penguna
- if (node != NULL) ➔ jika node ditemukan
- cout << "Node ditemukan. Masukkan data child kiri: "; ➔ menampilkan statement Node ditemukan. Masukkan data child kiri:
- cin >> data; ➔ menerima input pengguna dan menyimpan ke dalam variabel data
- insertLeft(data, node); ➔ menambahkan node baru sebagai anak kiri dari node yang ditemukan
- } else { ➔ jika node ditemukan
- cout << "Node tidak ditemukan,\n"; ➔ menampilkan statement Node tidak ditemukan
- break; ➔ mengakhiri case 2
- case 3: ➔ jika pengguna memilih case 3
- cout << "Masukkan data node: "; ➔ menampilkan statement Masukkan data node:
- cin >> data; ➔ menerima input pengguna dan menyimpan ke dalam variabel data
- node = cariNode (data, root); ➔ mencaei node dengan data

yang diberikan pengguna

- if (node != NULL) ➔ jika node tidak NULL
- cout << "Node ditemukan. Masukkan data child kanan: "; ➔ menampilkan statement Node ditemukan. Masukkan data child kanan:
- cin >> data; ➔ menerima input pengguna dan menyimpan ke dalam variabel data
- insertRight(data, node); ➔ menambahkan node baru sebagai anak kanan dari node yang ditemukan
- } else { ➔ jika node tidak ditemukan
- cout << "Node tidak ditemukan.\n"; ➔ menampilkan statement Node tidak ditemukan
- break; ➔ mengakhiri case 3
- case 4: ➔ jika pengguna memilih case 4
- cout << "Masukkan data node: "; ➔ menampilkan statement Masukkan data node:
- cin >> data; ➔ menerima input pengguna dan menyimpan ke dalam variabel data
- node = cariNode(data, root); ➔ mencari node dengan data yang diberikan pengguna
- if (node != NULL) ➔ jika node tidak kosong
- cout << "Node ditemukan. Masukkan data baru: "; ➔ menampilkan statement Node ditemukan. Masukkan data baru:
- cin >> data; ➔ menerima input pengguna dan menyimpan ke dalam variabel data
- update(data, node); ➔ memperbarui data dari node yang ditentukan
- } else { ➔ jika node tidak ditemukan
- cout << "Node tidak ditemukan.\n"; ➔ menampilkan statement Node tidak ditemukan.
- break; ➔ mengakhiri case 4

- case 5: ➔ jika pengguna memilih case 5
- cout << "Masukkan data node: "; ➔ menampilkan statement Masukkan data node:
- cin >> data; ➔ menerima input pengguna dan menyimpan ke dalam variabel data
- node = cariNode(data, root); ➔ mencari node dengan data yang diberikan pengguna
- retrieve(node); ➔ mengambil node dari node yang ditemukan
- break; ➔ mengakhiri case 5
- case 6: ➔ jika pengguna memilih case 6
- cout << "Masukkan data node: "; ➔ menampilkan statement Masukkan data node:
- cin >> data; ➔ menerima input pengguna dan menyimpan ke dalam variabel data
- node = cariNode(data, root); ➔ mencari node dengan data yang diberikan pengguna
- find(node); ➔ menemukan node dalam pohon
- break; ➔ mengakhiri case 6
- case 7: ➔ jika pengguna memilih case 7
- cout << "Preorder Traversal: "; ➔ menampilkan statement Preorder Traversal:
- preOrder(root);➔ melakukan penelusuran pohon dalam ueuran preorder
- cout << "\n"; ➔ menampilkan baris baru
- break; ➔ mengakhiri case 7
- case 8: ➔ jika pengguna memilih case 8
- cout << "Inorder Traversal: "; ➔ menampilkan statement Preorder Traversal:
- inOrder(root); ➔ melakukan penelusuran pohon dalam urutan inorder

- cout << "\n"; ➔ menempilkan baris baru
- break; ➔ mengakhiri case 8
- case 9: ➔ jika pengguna memilih case 9
- cout << "Postorder Traversal: "; ➔ menampilkan statement Preorder Traversal:
- postOrder(root); ➔ melakukan penelusuran pohon dalam urutan postorder
- cout << "\n"; ➔ menempilkan baris baru
- break; ➔ mengakhiri case 9
- case 10: ➔ jika pengguna memilih case 10
- cout << "Masukkan data node: "; ➔ menampilkan statement Masukkan data node:
- cin >> data; ➔ menempilkan baris baru
- node = cariNode(data, root); ➔ mencari node dengan data yang diberikan
- deleteSub(node); ➔ menghapus subtree yang dimulai dari node yang ditemukan
- break; ➔ mengakhiri case 10
- case 11: ➔ jika pengguna memilih case 11
- clear(); ➔ menghapus seluruh pohon
- break; ➔ mengakhiri case 11
- case 12: ➔ jika pengguna memilih case 12
- cout << "Size if Tree: " << size(root) << "\n"; ➔ menampilkan ukuran pohon
- break; ➔ mengakhiri case 12
- case 13: ➔ jika pengguna memilih case 13
- cout << "Height of Tree: " << height(root) << "\n"; ➔ menampilkan tinggi pohon
- break; ➔ mengakhiri case 13

- case 14: ➔ jika pengguna memilih case 14
- charateristic(); ➔ menampilkan karakteristik pohon
- break; ➔ mengakhiri case 14
- case 15: ➔ jika pengguna memilih case 15
- cout << "Keluar dari program.\n"; ➔ menampilkan statement Keluar dari program
- break; ➔ mengakhiri case 15
- default: ➔ jika pengguna memilih pilihan diluar 1 hingga 15
- cout << "Pilihan tidak valid. Coba lagi.\n"; ➔ menampilkan statement Pilihan tidak valid. Coba lagi
- } while (pilihan_Faisal_Khoiruddin_2311102046 != 0); ➔ melanjutkan perulangan sampai pengguna memilih untuk keluar
- return 0; ➔ program akan mengembalikan (return) nilai 0

C. Kesimpulan

1. Graf

Graf atau graph adalah struktur data yang digunakan untuk merepresentasikan hubungan antara objek dalam bentuk node atau vertex dan sambungan antara node tersebut dalam bentuk sisi atau edge. Graf terdiri dari simpul dan busur yang secara matematis dinyatakan sebagai :

$$G = (V, E)$$

Jenis-jenis Graph

1. Graph berarah (directed graph): Urutan simpul mempunyai arti. Misal busur AB adalah e1 sedangkan busur BA adalah e8.
2. Graph tak berarah (undirected graph): Urutan simpul dalam sebuah busur tidak diperhatikan. Misal busur e1 dapat disebut busur AB atau BA.
3. Weight Graph : Graph yang mempunyai nilai pada tiap edgenya.

2. Tree

Dalam ilmu komputer, pohon/tree adalah struktur data yang

sangat umum dan kuat yang menyerupai nyata pohon.

Operasi pada Tree

1) Create: digunakan untuk membentuk binary tree baru yang masih
2) Clear: digunakan untuk mengosongkan binary tree yang sudah ada atau menghapus semua node pada binary tree.
3) isEmpty: digunakan untuk memeriksa apakah binary tree masih kosong atau tidak.

4) Insert: digunakan untuk memasukkan sebuah node kedalam tree.

5) Find: digunakan untuk mencari root, parent, left child, atau right child dari suatu node dengan syarat tree tidak boleh kosong.

6) Update: digunakan untuk mengubah isi dari node yang ditunjuk oleh pointer current dengan syarat tree tidak boleh kosong.

7) Retrive: digunakan untuk mengetahui isi dari node yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.

8) Delete Sub: digunakan untuk menghapus sebuah subtree (node beserta seluruh descendant-nya) yang ditunjuk pointer current dengan syarat tree tidak boleh kosong.

9) Characteristic: digunakan untuk mengetahui karakteristik dari suatu tree. Yakni size, height, serta average lenght-nya.

10) Traverse: digunakan untuk mengunjungi seluruh node-node pada tree dengan cara traversal. Terdapat 3 metode traversal yang dibahas dalam modul ini yakni Pre-Order, In-Order, dan Post-Order.

D. Referensi

[1] Asisten Praktikum, Struktur Data “GRAPH DAN TREE", WhatsApp, 2024.

[2] TylerMSFT. (n.d.) <iomanip>. diakses dari

https://learn.microsoft.com/id-id/cpp/standard-library/iomanip-functions?view=msvc-170

[3] TylerMSFT. (n.d.) <vector>. diakses dari

https://learn.microsoft.com/id-id/cpp/standard-library/vector-class?view=msvc-170