LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL X
TREE

Disusun Oleh :
NAMA : Muhamad Naufal Ammar
NIM : 103112430036

Dosen
WAHYU ANDI SAPUTRA

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Struktur data Binary Search Tree (BST) tetap menjadi fondasi penting dalam komputasi modern karena fleksibilitas dan efisiensinya dalam pengelolaan data terurut, memungkinkan operasi penyisipan, pencarian, dan traversing dengan kompleksitas rata-rata yang baik selama tree relatif seimbang dan tetap relevan di berbagai domain termasuk pemrosesan data besar, sistem paralel, dan aplikasi real-time. Misalnya, penelitian tentang konstruksi paralel menunjukkan bahwa dengan pendekatan spekulatif, BST dapat dibangun secara simultan dalam lingkungan multiprosesor sambil mempertahankan struktur yang sama dengan eksekusi sekuensial, sehingga menjamin konsistensi dan reproducibility struktur data. Dalam context big data, pendekatan "norm-based" BST telah terbukti membantu mempercepat algoritma K-Nearest Neighbors, menunjukkan bahwa BST bukan hanya cocok untuk operasi tradisional (insert/search), tetapi juga berguna untuk klasifikasi dan data mining. Di sisi sistem, adaptasi BST untuk lingkungan multi-thread atau hardware khusus (seperti FPGA) memperlihatkan bahwa BST dapat dioptimalkan untuk throughput dan kinerja tinggi, baik dalam konteks concurrency maupun akselerasi hardware. Oleh karena itu, BST tidak sekedar struktur data dasar, melainkan fondasi adaptif yang terus relevan seiring berkembangnya kebutuhan komputasi dari aplikasi sederhana hingga komputasi paralel, big data, dan sistem kinerja tinggi.

B. Guided (berisi screenshot source code & output program disertai penjelasannya)

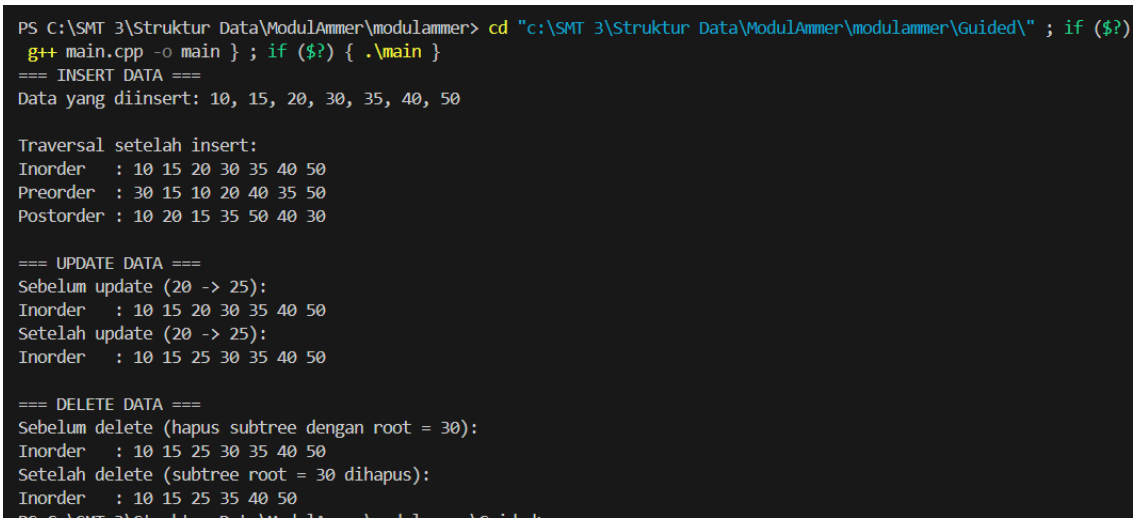
Guided 1

```cpp
#include "bstree.h"
#include <iostream>
using namespace std;
address alokasi(infotype x) {
    address P = new node;
    if (P != nullptr) {
        P->info = x;
        P->left = Nil;
        P->right = Nil;
    }
    return P;
}

void insertNode(address &root, infotype x) {
    if (root == Nil) {
        root = alokasi(x);
    } else {
        if (x < root->info) {
            insertNode(root->left, x);
        } else if (x > root->info) {
            insertNode(root->right, x);
        }
    }
}

address searchNode(infotype x, address root) {
    if (root == Nil) {
        return Nil;
    } else if (x == root->info) {
        return root;
    } else if (x < root->info) {
        return searchNode(x, root->left);
    } else {
        return searchNode(x, root->right);
    }
}

void InOrder(address root) {
    if (root != Nil) {
        InOrder(root->left);
        cout << root->info << " ";
        InOrder(root->right);
    }
}
```

```cpp
int hitungNode(address root) {
    if (root == Nil) return 0;
    return 1 + hitungNode(root->left) + hitungNode(root->right);
}

int hitungTotal(address root) {
    if (root == Nil) return 0;
    return root->info + hitungTotal(root->left) + hitungTotal(root->right);
}

int hitungKedalaman(address root, int start) {
    if (root == Nil) return start;
    int leftDepth = hitungKedalaman(root->left, start + 1);
    int rightDepth = hitungKedalaman(root->right, start + 1);
    return max(leftDepth, rightDepth);
}

void PreOrder(address root) {
    if (root != Nil) {
        cout << root->info << " ";
        PreOrder(root->left);
        PreOrder(root->right);
    }
}

void PostOrder(address root) {
    if (root != Nil) {
        PostOrder(root->left);
        PostOrder(root->right);
        cout << root->info << " ";
    }
}
```

```cpp
#include <iostream>
#include "tree.h"
#include "tree.cpp"

using namespace std;

int main() {
    BinaryTree tree;

    cout << "=== INSERT DATA ===" << endl;
    tree.insert(10);
    tree.insert(15);
    tree.insert(20);
    tree.insert(30);
    tree.insert(35);
    tree.insert(40);
    tree.insert(50);


    cout << "Data yang diinsert: 10, 15, 20, 30, 35, 40, 50"
    << endl;

    cout << "\nTraversal setelah insert:" << endl;
    cout << "Inorder   : "; tree.inorder();
    cout << "Preorder  : "; tree.preorder();
    cout << "Postorder : "; tree.postorder();

    cout << "\n=== UPDATE DATA ===" << endl;
    cout << "Sebelum update (20 -> 25):" << endl;
    cout << "Inorder   : "; tree.inorder();

    tree.update(20, 25);

    cout << "Setelah update (20 -> 25):" << endl;
    cout << "Inorder   : "; tree.inorder();

    cout << "\n=== DELETE DATA ===" << endl;
    cout << "Sebelum delete (hapus subtree dengan root = 30):"
    << endl;
    cout << "Inorder   : "; tree.inorder();

    tree.deleteValue(30);

    cout << "Setelah delete (subtree root = 30 dihapus):"
    << endl;
    cout << "Inorder   : "; tree.inorder();
    return 0;
}
```

Screenshots Output



Deskripsi:

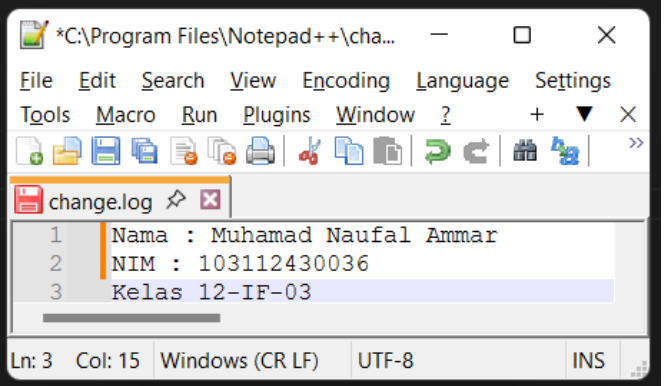
Program utama yang digunakan untuk mencoba berbagai fitur pada struktur data Binary Tree melalui sebuah objek bernama tree yang berasal dari kelas BinaryTree. Pertama, program melakukan proses penyisipan beberapa data ke dalam tree, kemudian menampilkan hasil traversal Inorder, Preorder, dan Postorder untuk menunjukkan bagaimana data tersusun di dalam pohon setelah insert dilakukan. Setelah itu, terdapat operasi update di mana nilai pada node 20 diganti menjadi 25, dan hasilnya kembali dicek menggunakan traversal inorder untuk melihat perubahan strukturnya. Terakhir, program menghapus subtree yang memiliki root dengan nilai 30, lalu ditampilkan lagi traversal inorder untuk mengetahui bentuk pohon setelah proses delete. Secara keseluruhan, program ini memperlihatkan bagaimana operasi dasar pada Binary Tree seperti insert, update, dan delete dapat mempengaruhi posisi serta struktur penyimpanan data dalam tree.

C. Unguided (berisi screenshot source code & output program disertai penjelasannya)

Unguided 1

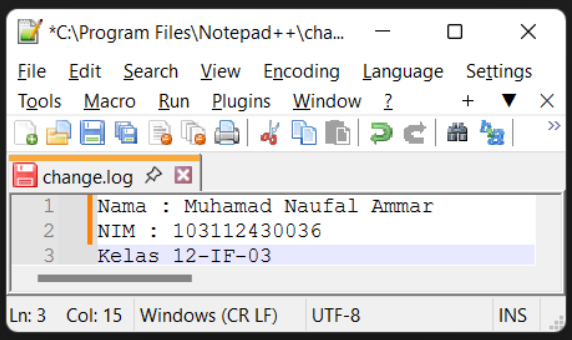
bstree.h

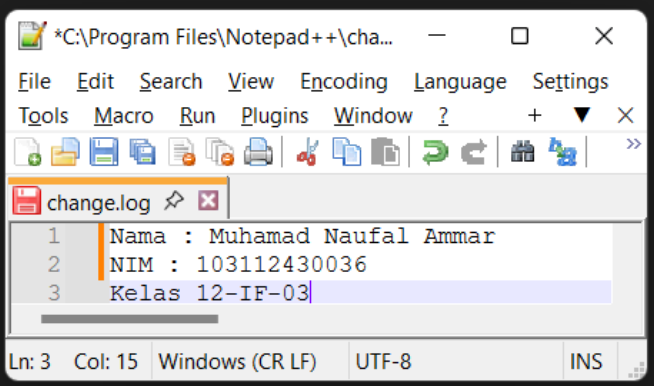
```h
#ifndef BSTREE_H
#define BSTREE_H

#include <iostream>
using namespace std;

#define Nil NULL

typedef int infotype;
typedef struct Node* address;

struct Node {
    infotype info;
    address left, right;
};

address alokasi(infotype x);
void insertNode(address &root, infotype x);
address findNode(infotype x, address root);
void printInorder(address root);

int hitungJumlahNode(address root);
int hitungTotalInfo(address root);
int hitungKedalaman(address root, int start);

void printPreorder(address root);
void printPostorder(address root);

#endif
```



bstree.cpp

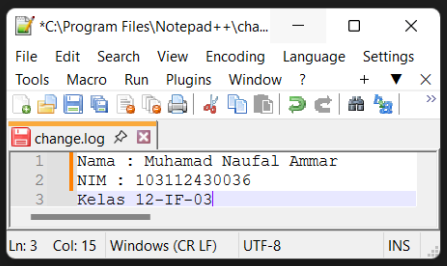
```cpp
#include "bstree.h"

address alokasi(infotype x) {
    address P = new Node;
    P->info = x;
    P->left = P->right = Nil;
    return P;
}

void insertNode(address &root, infotype x) {
    if (root == Nil) {
        root = alokasi(x);
    } else {
        if (x < root->info) {
            insertNode(root->left, x);
        } else if (x > root->info) {
            insertNode(root->right, x);
        }
    }
}

address findNode(infotype x, address root) {
    if (root == Nil) return Nil;
    if (x == root->info) return root;
    if (x < root->info) return findNode(x, root->left);
    else return findNode(x, root->right);
}

void printInorder(address root) {
    if (root != Nil) {
        printInorder(root->left);
        cout << root->info << " ";
        printInorder(root->right);
    }
}

int hitungJumlahNode(address root) {
    if (root == Nil) return 0;
    return 1 + hitungJumlahNode(root->left) + hitungJumlahNode(root->right);
}

int hitungTotalInfo(address root) {
    if (root == Nil) return 0;
    return root->info + hitungTotalInfo(root->left) + hitungTotalInfo(root->right);
}

int hitungKedalaman(address root, int start) {
    if (root == Nil) return start;
    int leftDepth = hitungKedalaman(root->left, start + 1);
    int rightDepth = hitungKedalaman(root->right, start + 1);
    return max(leftDepth, rightDepth);
}

void printPreorder(address root) {
    if (root != Nil) {
        cout << root->info << " ";
        printPreorder(root->left);
        printPreorder(root->right);
    }
}

void printPostorder(address root) {
    if (root != Nil) {
        printPostorder(root->left);
        printPostorder(root->right);
        cout << root->info << " ";
    }
}
```





main.cpp

```cpp
#include "bstree.h"
#include "bstree.cpp"

int main() {
    cout << "Hello World" << endl;

    address root = Nil;
    insertNode(root,1);
    insertNode(root,2);
    insertNode(root,6);
    insertNode(root,4);
    insertNode(root,5);
    insertNode(root,3);
    insertNode(root,6);
    insertNode(root,7);

    printInorder(root);
    cout << "\n";

    cout << "kedalaman : " << hitungKedalaman(root, 0) << endl;
    cout << "jumlah Node : " << hitungJumlahNode(root) << endl;
    cout << "total : " << hitungTotalInfo(root) << endl;

    cout << "PreOrder : ";
    printPreorder(root);
    cout << endl;

    cout << "PostOrder: ";
    printPostorder(root);
    cout << endl;


    return 0;
}
```



Screenshots Output

```
PS C:\SMT 3\Struktur Data\ModulAmmer\modulammer\Guided> cd "c:\SMT 3\Struktur Data\ModulAmmer\modulammer\" ; if ($?) {
 g++ main.cpp -o main } ; if ($?) { .\main }
Hello World
1 2 3 4 5 6 7
kedalaman : 5
jumlah Node : 7
total : 28
PreOrder : 1 2 6 4 3 5 7
PostOrder: 3 5 4 7 6 2 1
```

Deskripsi:

Kode program tersebut mengimplementasikan Binary Search Tree (BST) menggunakan bahasa C++, dimana setiap elemen disimpan dalam node yang memiliki informasi data bertipe integer serta pointer ke anak kiri dan kanan. Fungsi alokasi digunakan untuk membuat node baru, sementara insertNode menyisipkan data secara rekursif dengan aturan BST bahwa nilai yang lebih kecil ditempatkan di kiri dan lebih besar di kanan. Selain itu, tersedia fungsi pencarian findNode untuk menemukan node tertentu. Untuk penelusuran data, disediakan tiga metode traversal, yaitu printInorder yang mengunjungi node secara urutan ascending (left-root-right), printPreorder yang menampilkan root terlebih dahulu kemudian subtree kiri dan kanan, serta printPostorder yang menampilkan data subtree terlebih dahulu lalu root. Program juga memiliki fungsi rekursif hitungJumlahNode untuk menghitung total node, hitungTotalInfo untuk menjumlahkan semua nilai yang tersimpan, dan hitungKedalaman untuk menentukan kedalaman maksimum pohon. Pada main.cpp, beberapa nilai disisipkan sehingga membentuk struktur BST, kemudian dilakukan proses traversal serta perhitungan sifat-sifat tree untuk ditampilkan ke layar.

D. Kesimpulan

Secara keseluruhan, kode program Binary Search Tree (BST) ini mengimplementasikan struktur data pohon biner yang mampu menyimpan data integer secara terurut dengan efisien. Program ini menyediakan berbagai operasi dasar pada BST seperti penyisipan data (insert), pencarian data (find), dan penelusuran atau traversal pohon dalam tiga bentuk yaitu inorder, preorder, dan postorder untuk menampilkan isi tree berdasarkan kebutuhan algoritma. Selain itu, program ini juga dilengkapi fungsi analisis struktur tree seperti menghitung jumlah node, total nilai seluruh node, dan kedalaman maksimal tree melalui pendekatan rekursif. Dengan kombinasi fitur tersebut, program dapat digunakan sebagai contoh penerapan konsep fundamental dalam struktur data tree untuk pengelolaan dan pengolahan data secara hirarkis serta efisien dalam pencarian.

E. Referensi

Hirata, H., & Nunome, A. (2023). *Performance Evaluation on Parallel Speculation-Based Construction of a Binary Search Tree*. *International Journal of Networked and Distributed Computing, 11*, 88–111

Hassanat, A. B. A. (2018). *Norm-Based Binary Search Trees for Speeding Up KNN Big Data Classification*. *Computers, 7*(4), 54.

Aksenov, V., Gramoli, V., Kuznetsov, P., & Malova, A. (2017). *A Concurrency-Optimal Binary Search Tree*.

Melikoglu, O., Ergin, O., Salami, B., Pavon, J., Unsal, O., & Cristal, A. (2019). *A Novel FPGA-Based High Throughput Accelerator For Binary Search Trees*.

Brown, T., Sigouin, W., & Alistarh, D. (2022). *PathCAS: An Efficient Middle Ground for Concurrent Search Data Structures*.