LAPORAN PRAKTIKUM

- **Identitas Praktikum**

  Nama MK : Struktur Data
  Kode MK : CCK2AAB4
  Bobot SKS : 4 SKS
  Tempat : L-Program, Gedung DC, lantai 3
  Hari, tanggal : Selasa, 26 November 2024
  Jam : 12:30-15:30 WIB
  Topik praktikum : Modul-10 TREE

- **Identitas Mahasiswa**

  Nama lengkap : Afad Fath Musyarof Halim
  NIM : 2211104030
  Program Studi : S-1 Software Engineering

- **Hasil Praktikum**

10. TREE

10.1 Recursive

Adalah fungsi yang memanggil dirinya sendiri

Contoh:

- Recursive menghitung Faktorial
  - Code

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

int factorial(int n) {
    if (n <= 1) {
        return 1;
    }
    return n * factorial(n - 1);
}

int main() {
    int num;
    cout << "Masukkan bilangan : ";
    cin >> num;

    if (num < 0) {
        cout << "Faktorial tidak terdefinisi untuk bilangan negatif." << endl;
    } else {
        cout << "Rumus Faktorial: n! = n x (n-1) x (n-2) x ... x 1" << endl;
        cout << "Faktorial dari " << num << " adalah " << factorial(num) << endl;
    }

    return 0;
}
```

- Output

```
Afadfath | output                                    main
#  & .\'Recursive_factorial.exe'
Masukkan bilangan : 0
Rumus Faktorial: n! = n x (n-1) x (n-2) x ... x 1
Faktorial dari 0 adalah 1
Afadfath | output                                    main
#  & .\'Recursive_factorial.exe'
Masukkan bilangan : -1
Faktorial tidak terdefinisi untuk bilangan negatif.
Afadfath | output                                    main
#  & .\'Recursive_factorial.exe'
Masukkan bilangan : 10
Rumus Faktorial: n! = n x (n-1) x (n-2) x ... x 1
Faktorial dari 10 adalah 3628800
Afadfath | output                                    main
#
```

- Recursive menampilkan angka berurutan
  - Code

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

void cetakAngka(int n) {
    if (n == 0) {
        return;
    }
    cetakAngka(n - 1);
    cout << n << " ";
}

int main() {
    int N;
    cout << "Masukkan angka N: ";
    cin >> N;

    cout << "Angka dari 1 hingga " << N << ": ";
    cetakAngka(N);
    cout << endl;

    return 0;
}
```

  - Output

```
Afadfath | output                                    main
#  & .\'Recursive_print.exe'
Masukkan angka N: 3
Angka dari 1 hingga 3: 1 2 3
Afadfath | output                                    main
#  & .\'Recursive_print.exe'
Masukkan angka N: 5
Angka dari 1 hingga 5: 1 2 3 4 5
Afadfath | output                                    main
#
```

10.2 Ordered Tree

Adalah metode struktur data dimana data tidak memiliki urutan tertentu namun merupakan turunan dari suatu angka untuk setiap data yang di inputkan kecuali urutan paling awal (root)

Contoh:

- Code

```cpp
#include <iostream>
#include <vector>
using namespace std;

struct Node {
    int data;
    vector<Node*> children;

    Node(int value) {
        data = value;
    }
};

void tambahAnak(Node* parent, int value) {
    Node* child = new Node(value);
    parent->children.push_back(child);
}

void cetakTree(Node* root, int depth = 0) {
    if (!root) return;

    for (int i = 0; i < depth; i++) cout << "  ";
    cout << root->data << endl;

    for (Node* child : root->children) {
        cetakTree(child, depth + 1);
    }
}

int main() {
    Node* root = new Node(1);

    tambahAnak(root, 2);
    tambahAnak(root, 3);
    tambahAnak(root, 4);

    tambahAnak(root->children[0], 5);
    tambahAnak(root->children[0], 6);

    tambahAnak(root->children[1], 7);

    cout << "Tree:" << endl;
    cetakTree(root);

    return 0;
}
```

- Output

```
Afadfath | output                main
# & .\'Tree.exe'
Tree:
1
  2
    5
    6
  3
    7
  4
Afadfath | output                main
#
```

10.3 Binary Tree

Adalah Tree yang hanya memiliki maksimal 2 turunan.

- Koding

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

struct Node {
    int data;
    Node *left;
    Node *right;

    Node(int value) {
        data = value;
        left = right = nullptr;
    }
};

Node *insert(Node *root, int value) {
    if (root == nullptr) {
        return new Node(value);
    }

    if (value < root->data) {
        root->left = insert(root->left, value);
    }
    else if (value > root->data) {
        root->right = insert(root->right, value);
    }

    return root;
}

bool search(Node *root, int value){
    if (root == nullptr) return false;

    if (root->data == value) return true;
    else if (value < root->data) return search(root->left, value);
    else return search(root->right, value);
}

int main() {
    Node *root = nullptr;

    root = insert(root, 50);
    root = insert(root, 30);
    root = insert(root, 70);
    root = insert(root, 20);
    root = insert(root, 40);
    root = insert(root, 60);
    root = insert(root, 80);

    cout << endl;

    int key = 40;
    if (search(root, key))
    {
        cout << "Element " << key << " found in BST." << endl;
    }
    else
    {
        cout << "Element " << key << " not found in BST." << endl;
    }

    return 0;
}
```

- Output

```
Afadfath | output                    main
# & .\'Tree_Binary.exe'

Element 40 found in BST.
Afadfath | output                    main
#
```

10.4 Traversal Binary Tree

Traversal adalah metode untuk mencetak nilai dari setiap elemen yang ada pada tree

Contoh apabila menggunakan kodingan binary tree di atas (No 10.3)

- Kodingan

```
void inOrderTraversal(Node *root) {
    if (root != nullptr) {
        inOrderTraversal(root->left);
        cout << root->data << " ";
        inOrderTraversal(root->right);
    }
}
```

- Output

```
Afadfath | output                                    main
# & .\'Tree_Binary.exe'
In-order Traversal: 20 30 40 50 60 70 80
Afadfath | output                                    main
#
```

10.5 Soal

1. Membuat ADT Binary Tree
   - Coding
     - Bstree.h

```h
#ifndef BSTREE_H
#define BSTREE_H

typedef int infotype;

struct Node {
    infotype info;
    Node* left;
    Node* right;
};

typedef Node* address;

address alokasi(infotype x);

void insertNode(address &root, infotype x);

address findNode(infotype x, address root);

void printInorder(address root);

#endif // BSTREE_H
```

- Bstree.cpp

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

using namespace std;

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

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

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

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

- Main.cpp

```cpp
#include "bstree.cpp"
#include <iostream>

using namespace std;

int main() {
    cout << "Hello World" << endl;
    address root = NULL;

    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);

    return 0;
}
```

- Output

```
Afadfath | output                                  main
# & .\'main.exe'
Hello World
1 - 2 - 3 - 4 - 5 - 6 - 6 - 7 -
Afadfath | output                                  main
#
```

- Penjelasan
    - *Alokasi* untuk membuat Node baru sebelum dimasukan ke tree
    - *Insert Node* untuk memasukan Node yang telah dibuat ke posisi berdasarkan prinsip binary tree yang akan di tempatkan sebagai turunan (child) dari node yang sesuai, jika lebih kecil dari node akan di sebelah kiri, jika lebih besar akan di sebelah kanan. Jika nilai node sudah ada pada tree maka tidak akan dimasukan
    - *Find node* untuk mencari node berdasarkan inputan lalu di iterasikan posisinya seperti saat insert namun daripada menambahkan node, fungsi akan mengembalikan nilai dari inputan apabila inputan ditemukan pada tree
    - *Print inorder* untuk menampilkan seluruh elemen pada tree dari yang paling kiri(kecil) ke paling kanan (besar)

2. Buat Fungsi tambahan
   - fungsi hitungJumlahNode
   - fungsi hitungTotalInfo
   - fungsi hitungKedalaman

   - tambahan Coding
     - bstree.h

```
int hitungJumlahNode(address root);

int hitungTotalInfo(address root);

int hitungKedalaman(address root, int start);
```

     - bstree.cpp

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

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

int hitungKedalaman(address root, int start) {
    if (root == nullptr) {
        return start;
    } else {
        int Kiri = hitungKedalaman(root->left, start + 1);
        int Kanan = hitungKedalaman(root->right, start + 1);
        return max(Kiri, Kanan);
    }
}
```

     - main.cpp

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

   - Output

```
Afadfath | output                                  main
# & .\'main.exe'
Hello World
1 - 2 - 3 - 4 - 5 - 6 - 7 -
kedalaman : 5
jumlah Node : 7
total : 28
Afadfath | output                                  main
#
```

- Penjelasan
  - *hitungJumlahNode* berfungsi dengan menambahkan nilai 1 untuk setiap node yang ada, jika node sudah tidak memiliki turunan maka fungsi akan menghentikan perhitungan
  - *hitungTotalInfo* berfungsi dengan menjumlahkan setiap nilai dari masing-masing node
  - *hitungKedalaman* berfungsi dengan menelusuri turunan dari root bagian kiri dan kanan lalu menambahkan nilai turunan 1 untuk setiap 1 kali turun, lalu dari turunan kiri dan kanan dibandingkan dan mengembalikan nilai dengan nilai turunan terbanyak