LAPORAN PRAKTIKUM
Modul 10&11
TREE

Disusun Oleh:
Jauhar Fajar Zuhair
2311104072
S1SE-07-2

Dosen :
Wahyu Andri Saputra, S.Pd., M.Eng.

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO

2024

Tujuan Pembelajaran

1. Memahami dan mengimplementasikan konsep fungsi rekursif
2. Mengaplikasikan dan mengimplementasikan struktur data tree, khususnya Binary Tree dalam pemrograman 3. Memahami berbagai operasi dan traversal dalam Binary Tree

Ringkasan Materi

1. Konsep Rekursif

• **Definisi**: Rekursif adalah proses pengulangan yang memanggil dirinya sendiri
• **Kriteria Rekursif**:
◦ Harus memiliki kondisi berhenti (special condition)
◦ Memiliki pemanggilan diri sendiri
• **Kelebihan**:
◦ Meningkatkan readibility (kemudahan membaca program)
◦ Meningkatkan modularity (pemecahan program menjadi modul-modul)
◦ Meningkatkan reusability (penggunaan ulang kode)
• **Kekurangan**:
◦ Membutuhkan memori lebih banyak
◦ Memerlukan waktu lebih untuk menangani activation record

2. Struktur DataTree

- **Definisi**: Tree adalah struktur data non-linear berupa graph tak berarah yang terhubung dan tidak mengandung sirkuit
- **Karakteristik**: 1. Dapat kosong (empty tree) 2. Memiliki satu node tanpa pendahulu (root) 3. Setiap node lain hanya memiliki satu pendahulu

3.TerminologiTree

1. **Child & Parent**: Hubungan antar node yang terhubung
2. **Path**: Lintasan dari satu node ke node lain
3. **Sibling**: Node-node yang memiliki parent yang sama
4. **Degree**: Jumlah anak pada suatu node
5. **Leaf**: Node yang tidak memiliki anak
6. **Internal nodes**: Node yang memiliki anak
7. **Height/Depth**: Jumlah maksimum level dalam tree

4. Jenis-JenisTree

1. **OrderedTree**: Tree dengan urutan anak yang penting
2. **BinaryTree**: Tree dengan maksimum 2 anak per node
- Complete Binary Tree
- Extended Binary Tree
- Binary Search Tree (BST) ◦ AVL Tree ◦ Heap Tree (Minimum & Maximum Heap)

5. Operasi Dasar Binary Search Tree

1. **Insert**: Penambahan node sesuai aturan BST
2. **Update**: Pembaruan nilai node dengan mempertahankan properti BST
3. **Search**: Pencarian node menggunakan algoritma binary search

4. **Delete**: Penghapusan node dengan 3 kasus:
◦ Leaf node
◦ Node dengan 1 anak
◦ Node dengan 2 anak

6.Traversal BinaryTree

1. **Pre-order**: Root → Left → Right
2. **In-order**: Left → Root → Right
3. **Post-order**: Left → Right → Root

7. Konsep Most-Left dan Most-Right

• **Most-Left**: Node dengan nilai terkecil dalam BST
• **Most-Right**: Node dengan nilai terbesar dalam BST

Struktur data tree, khususnya Binary Search Tree, sangat penting dalam pengembangan software karena efisiensinya dalam pencarian, pengurutan, dan organisasi data hierarkis. Pemahaman yang baik tentang konsep ini akan membantu dalam implementasi struktur data yang lebih kompleks dan optimisasi program.

Guided

Guided1.cpp

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

/// PROGRAM BINARY TREE

// Struktur data pohon biner untuk menyimpan data dan pointer ke anak kiri,
kanan, dan induk
struct Pohon {
    char data;                  // Data yang disimpan di node (tipe char)
    Pohon *left, *right;        // Pointer ke anak kiri dan anak kanan
    Pohon *parent;              // Pointer ke node induk
};

// Variabel global untuk menyimpan root (akar) pohon dan node baru
Pohon *root, *baru;

// Inisialisasi pohon agar kosong
void init() {
    root = NULL; // Mengatur root sebagai NULL (pohon kosong)
}

// Mengecek apakah pohon kosong
bool isEmpty() {
    return root == NULL; // Mengembalikan true jika root adalah NULL
}

// Membuat node baru sebagai root pohon
```

```
void buatNode(char data) {
    if (isEmpty()) { // Jika pohon kosong
        root = new Pohon{data, NULL, NULL, NULL}; // Membuat node baru
sebagai root
        cout << "\nNode " << data << " berhasil dibuat menjadi root." <<
endl;
    } else {
        cout << "\nPohon sudah dibuat." << endl; // Root sudah ada, tidak
membuat node baru
    }
}

// Menambahkan node baru sebagai anak kiri dari node tertentu
Pohon* insertLeft(char data, Pohon *node) {
    if (node->left != NULL) { // Jika anak kiri sudah ada
        cout << "\nNode " << node->data << " sudah ada child kiri!" << endl;
        return NULL; // Tidak menambahkan node baru
    }
    // Membuat node baru dan menghubungkannya ke node sebagai anak kiri
    baru = new Pohon{data, NULL, NULL, node};
    node->left = baru;
    cout << "\nNode " << data << " berhasil ditambahkan ke child kiri " <<
node->data << endl;
    return baru; // Mengembalikan pointer ke node baru
}

// Menambahkan node baru sebagai anak kanan dari node tertentu
Pohon* insertRight(char data, Pohon *node) {
    if (node->right != NULL) { // Jika anak kanan sudah ada
        cout << "\nNode " << node->data << " sudah ada child kanan!" << endl;
        return NULL; // Tidak menambahkan node baru
    }
    // Membuat node baru dan menghubungkannya ke node sebagai anak kanan
    baru = new Pohon{data, NULL, NULL, node};
    node->right = baru;
    cout << "\nNode " << data << " berhasil ditambahkan ke child kanan " <<
node->data << endl;
    return baru; // Mengembalikan pointer ke node baru
}

// Mengubah data di dalam sebuah node
void update(char data, Pohon *node) {
    if (!node) { // Jika node tidak ditemukan
        cout << "\nNode yang ingin diubah tidak ditemukan!" << endl;
        return;
    }
    char temp = node->data; // Menyimpan data lama
    node->data = data;      // Mengubah data dengan nilai baru
    cout << "\nNode " << temp << " berhasil diubah menjadi " << data << endl;
}

// Mencari node dengan data tertentu
void find(char data, Pohon *node) {
    if (!node) return; // Jika node tidak ada, hentikan pencarian

    if (node->data == data) { // Jika data ditemukan
        cout << "\nNode ditemukan: " << data << endl;
```

```
        return;
    }
    // Melakukan pencarian secara rekursif ke anak kiri dan kanan
    find(data, node->left);
    find(data, node->right);
}

// Traversal Pre-order (Node -> Kiri -> Kanan)
void preOrder(Pohon *node) {
    if (!node) return; // Jika node kosong, hentikan traversal
    cout << node->data << " "; // Cetak data node saat ini
    preOrder(node->left);      // Traversal ke anak kiri
    preOrder(node->right);     // Traversal ke anak kanan
}

// Traversal In-order (Kiri -> Node -> Kanan)
void inOrder(Pohon *node) {
    if (!node) return; // Jika node kosong, hentikan traversal
    inOrder(node->left);  // Traversal ke anak kiri
    cout << node->data << " "; // Cetak data node saat ini
    inOrder(node->right); // Traversal ke anak kanan
}

// Traversal Post-order (Kiri -> Kanan -> Node)
void postOrder(Pohon *node) {
    if (!node) return; // Jika node kosong, hentikan traversal
    postOrder(node->left);  // Traversal ke anak kiri
    postOrder(node->right); // Traversal ke anak kanan
    cout << node->data << " "; // Cetak data node saat ini
}

// Menghapus node dengan data tertentu
Pohon* deleteNode(Pohon *node, char data) {
    if (!node) return NULL; // Jika node kosong, hentikan

    // Rekursif mencari node yang akan dihapus
    if (data < node->data) {
        node->left = deleteNode(node->left, data); // Cari di anak kiri
    } else if (data > node->data) {
        node->right = deleteNode(node->right, data); // Cari di anak kanan
    } else {
        // Jika node ditemukan
        if (!node->left) { // Jika tidak ada anak kiri
            Pohon *temp = node->right; // Anak kanan menggantikan posisi node
            delete node;
            return temp;
        } else if (!node->right) { // Jika tidak ada anak kanan
            Pohon *temp = node->left; // Anak kiri menggantikan posisi node
            delete node;
            return temp;
        }

        // Jika node memiliki dua anak, cari node pengganti (successor)
        Pohon *successor = node->right;
        while (successor->left) successor = successor->left; // Cari node
terkecil di anak kanan
        node->data = successor->data; // Gantikan data dengan successor
```

```
        node->right = deleteNode(node->right, successor->data); // Hapus
successor
    }
    return node;
}

// Menemukan node paling kiri
Pohon* mostLeft(Pohon *node) {
    if (!node) return NULL; // Jika node kosong, hentikan
    while (node->left) node = node->left; // Iterasi ke anak kiri hingga
mentok
    return node;
}

// Menemukan node paling kanan
Pohon* mostRight(Pohon *node) {
    if (!node) return NULL; // Jika node kosong, hentikan
    while (node->right) node = node->right; // Iterasi ke anak kanan hingga
mentok
    return node;
}

// Fungsi utama
int main() {
    init(); // Inisialisasi pohon
    buatNode('F'); // Membuat root dengan data 'F'
    insertLeft('B', root); // Menambahkan 'B' ke anak kiri root
    insertRight('G', root); // Menambahkan 'G' ke anak kanan root
    insertLeft('A', root->left); // Menambahkan 'A' ke anak kiri dari 'B'
    insertRight('D', root->left); // Menambahkan 'D' ke anak kanan dari 'B'
    insertLeft('C', root->left->right); // Menambahkan 'C' ke anak kiri dari
'D'
    insertRight('E', root->left->right); // Menambahkan 'E' ke anak kanan
dari 'D'

    // Traversal pohon
    cout << "\nPre-order Traversal: ";
    preOrder(root);
    cout << "\nIn-order Traversal: ";
    inOrder(root);
    cout << "\nPost-order Traversal: ";
    postOrder(root);

    // Menampilkan node paling kiri dan kanan
    cout << "\nMost Left Node: " << mostLeft(root)->data;
    cout << "\nMost Right Node: " << mostRight(root)->data;

    // Menghapus node
    cout << "\nMenghapus node D.";
    root = deleteNode(root, 'D');
    cout << "\nIn-order Traversal setelah penghapusan: ";
    inOrder(root);

    return 0;
}
```

Unguided

unGuided1.cpp

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

struct Pohon
{
  int data; // Changed to int for easier BST validation
  Pohon *left, *right;
  Pohon *parent;
};

Pohon *root, *baru;

void init()
{
  root = NULL;
}

bool isEmpty()
{
  return root == NULL;
}

// Function to display children of a node
void displayChildren(Pohon *node)
{
  if (!node)
  {
    cout << "Node tidak ditemukan!" << endl;
    return;
  }

  cout << "Node " << node->data << ":" << endl;
  if (node->left)
    cout << "Child kiri: " << node->left->data << endl;
  else
    cout << "Tidak ada child kiri" << endl;

  if (node->right)
    cout << "Child kanan: " << node->right->data << endl;
  else
    cout << "Tidak ada child kanan" << endl;
}

// Function to display descendants of a node
void displayDescendants(Pohon *node)
{
  if (!node)
    return;

  if (node->left)
  {
    cout << node->left->data << " ";
```

```
    displayDescendants(node->left);
  }
  if (node->right)
  {
    cout << node->right->data << " ";
    displayDescendants(node->right);
  }
}

// Function to check if tree is BST
bool is_valid_bst(Pohon *node, int min_val, int max_val)
{
  if (!node)
    return true;

  if (node->data <= min_val || node->data >= max_val)
    return false;

  return is_valid_bst(node->left, min_val, node->data) &&
         is_valid_bst(node->right, node->data, max_val);
}

// Function to count leaf nodes
int cari_simpul_daun(Pohon *node)
{
  if (!node)
    return 0;
  if (!node->left && !node->right)
    return 1;
  return cari_simpul_daun(node->left) + cari_simpul_daun(node->right);
}

// Modified insert functions to work with integers
void buatNode(int data)
{
  if (isEmpty())
  {
    root = new Pohon{data, NULL, NULL, NULL};
    cout << "\nNode " << data << " berhasil dibuat menjadi root." << endl;
  }
  else
  {
    cout << "\nPohon sudah dibuat." << endl;
  }
}

Pohon *insertLeft(int data, Pohon *node)
{
  if (!node)
  {
    cout << "\nNode parent tidak ditemukan!" << endl;
    return NULL;
  }
  if (node->left)
  {
    cout << "\nNode " << node->data << " sudah ada child kiri!" << endl;
    return NULL;
```

```
  }
  baru = new Pohon{data, NULL, NULL, node};
  node->left = baru;
  cout << "\nNode " << data << " berhasil ditambahkan ke child kiri " <<
node->data << endl;
  return baru;
}

Pohon *insertRight(int data, Pohon *node)
{
  if (!node)
  {
    cout << "\nNode parent tidak ditemukan!" << endl;
    return NULL;
  }
  if (node->right)
  {
    cout << "\nNode " << node->data << " sudah ada child kanan!" << endl;
    return NULL;
  }
  baru = new Pohon{data, NULL, NULL, node};
  node->right = baru;
  cout << "\nNode " << data << " berhasil ditambahkan ke child kanan " <<
node->data << endl;
  return baru;
}

// Function to find a node with specific value
Pohon *findNode(int data, Pohon *node)
{
  if (!node)
    return NULL;
  if (node->data == data)
    return node;

  Pohon *leftResult = findNode(data, node->left);
  if (leftResult)
    return leftResult;

  return findNode(data, node->right);
}

void displayMenu()
{
  cout << "\n=== MENU BINARY TREE ===" << endl;
  cout << "1. Buat root" << endl;
  cout << "2. Insert node kiri" << endl;
  cout << "3. Insert node kanan" << endl;
  cout << "4. Tampilkan children" << endl;
  cout << "5. Tampilkan descendants" << endl;
  cout << "6. Cek BST" << endl;
  cout << "7. Hitung simpul daun" << endl;
  cout << "8. Keluar" << endl;
  cout << "Pilihan: ";
}

int main()
```

```
{
  init();
  int choice, data, parentData;
  Pohon *temp;

  do
  {
    displayMenu();
    cin >> choice;

    switch (choice)
    {
    case 1:
      cout << "Masukkan nilai root: ";
      cin >> data;
      buatNode(data);
      break;

    case 2:
      cout << "Masukkan nilai parent: ";
      cin >> parentData;
      cout << "Masukkan nilai node: ";
      cin >> data;
      temp = findNode(parentData, root);
      if (temp)
        insertLeft(data, temp);
      else
        cout << "Parent tidak ditemukan!" << endl;
      break;

    case 3:
      cout << "Masukkan nilai parent: ";
      cin >> parentData;
      cout << "Masukkan nilai node: ";
      cin >> data;
      temp = findNode(parentData, root);
      if (temp)
        insertRight(data, temp);
      else
        cout << "Parent tidak ditemukan!" << endl;
      break;

    case 4:
      cout << "Masukkan nilai node: ";
      cin >> data;
      temp = findNode(data, root);
      displayChildren(temp);
      break;

    case 5:
      cout << "Masukkan nilai node: ";
      cin >> data;
      temp = findNode(data, root);
      cout << "Descendants: ";
      displayDescendants(temp);
      cout << endl;
      break;
```

```
    case 6:
      if (is_valid_bst(root, INT_MIN, INT_MAX))
        cout << "Pohon adalah BST valid" << endl;
      else
        cout << "Pohon bukan BST valid" << endl;
      break;

    case 7:
      cout << "Jumlah simpul daun: " << cari_simpul_daun(root) << endl;
      break;

    case 8:
      cout << "Program selesai" << endl;
      break;

    default:
      cout << "Pilihan tidak valid!" << endl;
    }
  } while (choice != 8);

  return 0;
}
```

Output

```
"C:\Users\Administrator\Documents\dak kuliah\semester 3\Stuktur Data\11_Pengenalan_CPP_Bagian_11\Unguided\cmake-build-debug\Unguided.exe"

=== MENU BINARY TREE ===
1. Buat root
2. Insert node kiri
3. Insert node kanan
4. Tampilkan children
5. Tampilkan descendants
6. Cek BST
7. Hitung simpul daun
8. Keluar
Pilihan:1
 Masukkan nilai root:5

Node 5 berhasil dibuat menjadi root.

=== MENU BINARY TREE ===
1. Buat root
2. Insert node kiri
3. Insert node kanan
4. Tampilkan children
5. Tampilkan descendants
6. Cek BST
7. Hitung simpul daun
8. Keluar
Pilihan:2
 Masukkan nilai parent:5
 Masukkan nilai node:4

Node 4 berhasil ditambahkan ke child kiri 5

=== MENU BINARY TREE ===
1. Buat root
2. Insert node kiri
3. Insert node kanan
4. Tampilkan children
5. Tampilkan descendants
6. Cek BST
7. Hitung simpul daun
8. Keluar
Pilihan:2
 Masukkan nilai parent:4
 Masukkan nilai node:3

Node 3 berhasil ditambahkan ke child kiri 4

=== MENU BINARY TREE ===
1. Buat root
2. Insert node kiri
3. Insert node kanan
4. Tampilkan children
5. Tampilkan descendants
6. Cek BST
7. Hitung simpul daun
8. Keluar
Pilihan:3
 Masukkan nilai parent:4
 Masukkan nilai node:2

Node 2 berhasil ditambahkan ke child kanan 4

=== MENU BINARY TREE ===
1. Buat root
2. Insert node kiri
3. Insert node kanan
4. Tampilkan children
5. Tampilkan descendants
6. Cek BST
7. Hitung simpul daun
8. Keluar
Pilihan:3
 Masukkan nilai parent:5
 Masukkan nilai node:1

Node 1 berhasil ditambahkan ke child kanan 5

=== MENU BINARY TREE ===
1. Buat root
2. Insert node kiri
3. Insert node kanan
4. Tampilkan children
5. Tampilkan descendants
6. Cek BST
7. Hitung simpul daun
8. Keluar
Pilihan:4
 Masukkan nilai node:5
 Node 5:
Child kiri: 4
Child kanan: 1

=== MENU BINARY TREE ===
1. Buat root
2. Insert node kiri
3. Insert node kanan
4. Tampilkan children
5. Tampilkan descendants
6. Cek BST
7. Hitung simpul daun
8. Keluar
Pilihan:5
 Masukkan nilai node:5
 Descendants: 4 3 2 1

=== MENU BINARY TREE ===
1. Buat root
2. Insert node kiri
3. Insert node kanan
4. Tampilkan children
5. Tampilkan descendants
6. Cek BST
7. Hitung simpul daun
8. Keluar
Pilihan:6
 Pohon bukan BST valid

=== MENU BINARY TREE ===
1. Buat root
2. Insert node kiri
3. Insert node kanan
4. Tampilkan children
5. Tampilkan descendants
6. Cek BST
7. Hitung simpul daun
8. Keluar
Pilihan:7
 Jumlah simpul daun: 3

=== MENU BINARY TREE ===
1. Buat root
2. Insert node kiri
3. Insert node kanan
4. Tampilkan children
5. Tampilkan descendants
6. Cek BST
7. Hitung simpul daun
8. Keluar
Pilihan:8
 Program selesai

Process finished with exit code 0
```