LAPORAN PRAKTIKUM STRUKTUR DATA

PERTEMUAN 10

MULTI LINKED LIST

Nama :

Reyner Atira Prasetyo (2311104057)

S1SE-07-02

Dosen :

Wahyu Andi Saputra, S.Pd., M.Eng.

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. TUJUAN

a. Memahami konsep penggunaan fungsi rekursif.
b. Mengimplementasikan bentuk-bentuk fungsi rekursif.
c. Mengaplikasikan struktur data tree dalam sebuah kasus pemrograman.
d. Mengimplementasikan struktur data tree, khususnya Binary Tree.
e. Mengimplementasikan struktur data tree, khususnya Binary Tree.

TOOL

1. Visual Studio Code

2. GCC

II. DASAR TEORI

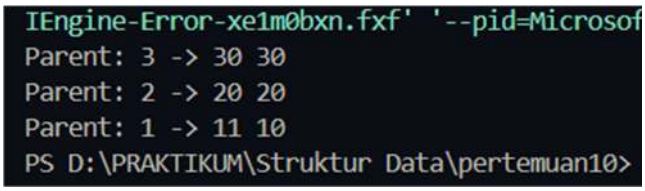
Multi List merupakan sekumpulan list yang berbeda yang memiliki suatu keterhubungan satu sama lain. Tiap elemen dalam multi link list dapat membentuk list sendiri.Biasanya ada yang bersifat sebagai list induk dan list anak**.**

III. GUIDED

1. Guided1

```cpp
#include <iostream>
#include <string>

using namespace std;


struct Node {
    int data;
    Node* next;
    Node* child;

    Node(int val) : data(val), next(nullptr), child(nullptr) {}
};


class MultiLinkedList {
private:
    Node* head;

public:
    MultiLinkedList() : head(nullptr) {}


    void addParent(int data) {
        Node* newNode = new Node(data);
        newNode->next = head;
        head = newNode;
    }


    void addChild(int parentData, int childData) {
        Node* parent = head;
        while (parent != nullptr && parent->data != parentData) {
            parent = parent->next;
        }
        if (parent != nullptr) {
            Node* newChild = new Node(childData);
            newChild->next = parent->child;
            parent->child = newChild;
        } else {
            cout << "Parent not found!" << endl;
        }
    }


    void display() {
        Node* current = head;
        while (current != nullptr) {
            cout << "Parent: " << current->data << " -> ";
            Node* child = current->child;
            while (child != nullptr) {
                cout << child->data << " ";
                child = child->next;
            }
            cout << endl;
            current = current->next;
        }
    }

    ~MultiLinkedList() {

        while (head != nullptr) {
            Node* temp = head;
            head = head->next;


            while (temp->child != nullptr) {
                Node* childTemp = temp->child;
                temp->child = temp->child->next;
                delete childTemp;
            }
            delete temp;
        }
    }
};

int main() {
    MultiLinkedList mList;

    mList.addParent(1);
    mList.addParent(2);
    mList.addParent(3);

    mList.addChild(1, 10);
    mList.addChild(1, 11);
    mList.addChild(2, 20);
    mList.addChild(2, 20);
    mList.addChild(3, 30);
    mList.addChild(3, 30);
    mList.display();

    return 0;
}

```

Hasil Run :



2. Guided2

```cpp
#include <iostream>
#include <string>

using namespace std;


struct EmployeeNode {
    string name;
    EmployeeNode* next;
    EmployeeNode* subordinate;

    EmployeeNode(string empName) : name(empName), next(nullptr), subordinate(nullptr) {}
};


class EmployeeList {
private:
    EmployeeNode* head;

public:
    EmployeeList() : head(nullptr) {}


    void addEmployee(string name) {
        EmployeeNode* newEmployee = new EmployeeNode(name);
        newEmployee->next = head;
        head = newEmployee;
    }


    void addSubordinate(string managerName, string subordinateName) {
        EmployeeNode* manager = head;
        while (manager != nullptr && manager->name != managerName) {
            manager = manager->next;
        }
        if (manager != nullptr) {
            EmployeeNode* newSubordinate = new EmployeeNode(subordinateName);
            newSubordinate->next = manager->subordinate;
            manager->subordinate = newSubordinate;
        } else {
            cout << "Manager not found!" << endl;
        }
    }


    void display() {
        EmployeeNode* current = head;
        while (current != nullptr) {
            cout << "Manager: " << current->name << " -> ";
            EmployeeNode* sub = current->subordinate;
            while (sub != nullptr) {
                cout << sub->name << " ";
                sub = sub->next;
            }
            cout << endl;
            current = current->next;
        }
    }

    ~EmployeeList() {

        while (head != nullptr) {
            EmployeeNode* temp = head;
            head = head->next;


            while (temp->subordinate != nullptr) {
                EmployeeNode* subTemp = temp->subordinate;
                temp->subordinate = temp->subordinate->next;
                delete subTemp;
            }
            delete temp;
        }
    }
};

int main() {
    EmployeeList empList;

    empList.addEmployee("Alice");
    empList.addEmployee("Bob");
    empList.addEmployee("Charlie");

    empList.addSubordinate("Alice", "David");
    empList.addSubordinate("Alice", "Eve");
    empList.addSubordinate("Bob", "Frank");

    empList.addSubordinate("Charlie", "Frans");
    empList.addSubordinate("Charlie", "Brian");

    empList.display();

    return 0;
}

```

Hasil Run :

3. Guided3

```cpp
#include <iostream>
#include <string>

using namespace std;

// Struktur untuk node karyawan
struct EmployeeNode {
    string name; // Nama karyawan
    EmployeeNode* next; // Pointer ke karyawan berikutnya
    EmployeeNode* subordinate; // Pointer ke subordinate pertama

    EmployeeNode(string empName) : name(empName), next(nullptr), subordinate(nullptr) {}
};

// Kelas untuk Multi-linked List Karyawan
class EmployeeList {
private:
    EmployeeNode* head; // Pointer ke kepala list

public:
    EmployeeList() : head(nullptr) {}

    // Menambahkan karyawan (induk)
    void addEmployee(string name) {
        EmployeeNode* newEmployee = new EmployeeNode(name);
        newEmployee->next = head; // Menyambungkan ke karyawan sebelumnya
        head = newEmployee; // Memperbarui head
    }

    // Menambahkan subordinate ke karyawan tertentu
    void addSubordinate(string managerName, string subordinateName) {
        EmployeeNode* manager = head;
        while (manager != nullptr && manager->name != managerName) {
            manager = manager->next;
        }
        if (manager != nullptr) { // Jika manajer ditemukan
            EmployeeNode* newSubordinate = new EmployeeNode(subordinateName);
            newSubordinate->next = manager->subordinate; // Menyambungkan ke subordinate sebelumnya
            manager->subordinate = newSubordinate; // Memperbarui subordinate
        } else {
            cout << "Manager not found!" << endl;
        }
    }

    // Menghapus karyawan (induk)
    void deleteEmployee(string name) {
        EmployeeNode** current = &head;
        while (*current != nullptr && (*current)->name != name) {
            current = &((*current)->next);
        }

        if (*current != nullptr) { // Jika karyawan ditemukan
            EmployeeNode* toDelete = *current;
            *current = (*current)->next;

            // Hapus semua subordinate dari node ini
            while (toDelete->subordinate != nullptr) {
                EmployeeNode* subTemp = toDelete->subordinate;
                toDelete->subordinate = toDelete->subordinate->next;
                delete subTemp;
            }
            delete toDelete;
            cout << "Employee " << name << " deleted." << endl;
        } else {
            cout << "Employee not found!" << endl;
        }
    }

    // Menghapus subordinate dari karyawan tertentu
    void deleteSubordinate(string managerName, string subordinateName) {
        EmployeeNode* manager = head;
        while (manager != nullptr && manager->name != managerName) {
            manager = manager->next;
        }

        if (manager != nullptr) { // Jika manajer ditemukan
            EmployeeNode** currentSub = &(manager->subordinate);
            while (*currentSub != nullptr && (*currentSub)->name != subordinateName) {
                currentSub = &((*currentSub)->next);
            }

            if (*currentSub != nullptr) { // Jika subordinate ditemukan
                EmployeeNode* toDelete = *currentSub;
                *currentSub = (*currentSub)->next; // Menghapus dari list

                delete toDelete; // Menghapus node subordinate
                cout << "Subordinate " << subordinateName << " deleted from " << managerName << "." << endl;
            } else {
                cout << "Subordinate not found!" << endl;
            }
        } else {
            cout << "Manager not found!" << endl;
        }
    }

    // Menampilkan daftar karyawan dan subordinate mereka
    void display() {
        EmployeeNode* current = head;
        while (current != nullptr) {
            cout << "Manager: " << current->name << " -> ";
            EmployeeNode* sub = current->subordinate;
            while (sub != nullptr) {
                cout << sub->name << " ";
                sub = sub->next;
            }
            cout << endl;
            current = current->next;
        }
    }

    ~EmployeeList() {
        // Destructor untuk membersihkan memori
        while (head != nullptr) {
            EmployeeNode* temp = head;
            head = head->next;

            // Hapus semua subordinate dari node ini
            while (temp->subordinate != nullptr) {
                EmployeeNode* subTemp = temp->subordinate;
                temp->subordinate = temp->subordinate->next;
                delete subTemp;
            }
            delete temp;
        }
    }
};

int main() {
    EmployeeList empList;

    empList.addEmployee("Alice");
    empList.addEmployee("Bob");
    empList.addEmployee("Charlie");

    empList.addSubordinate("Alice", "David");
    empList.addSubordinate("Alice", "Eve");
    empList.addSubordinate("Bob", "Frank");

    cout << "Initial employee list:" << endl;
    empList.display(); // Menampilkan isi daftar karyawan

    empList.deleteSubordinate("Alice", "David"); // Menghapus David dari Alice
    empList.deleteEmployee("Charlie"); // Menghapus Charlie

    cout << "\nUpdated employee list:" << endl;
    empList.display(); // Menampilkan isi daftar setelah penghapusan

    return 0;
}
```

Hasil Run :

```
IEngine-Error-5peacqoh.aeb' '--pid=Microsof
Initial employee list:
Manager: Charlie ->
Manager: Bob -> Frank
Manager: Alice -> Eve David
Subordinate David deleted from Alice.
Employee Charlie deleted.

Updated employee list:
Manager: Bob -> Frank
Manager: Alice -> Eve
PS D:\PRAKTIKUM\Struktur Data\pertemuan10>
```

IV. UNGUIDED

1. Unguided1.cpp

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

struct Project {
    string projectName;
    int duration;
    Project* nextProject;
};

struct Employee {
    string employeeName;
    string employeeID;
    Project* projectHead;
    Employee* nextEmployee;
};

// Head of the Employee list
Employee* employeeHead = nullptr;

// Function to create a new employee node
Employee* createEmployee(string name, string id) {
    Employee* newEmployee = new Employee;
    newEmployee->employeeName = name;
    newEmployee->employeeID = id;
    newEmployee->projectHead = nullptr;
    newEmployee->nextEmployee = nullptr;
    return newEmployee;
}

// Function to create a new project node
Project* createProject(string name, int duration) {
    Project* newProject = new Project;
    newProject->projectName = name;
    newProject->duration = duration;
    newProject->nextProject = nullptr;
    return newProject;
}

// Function to add an employee to the list
void addEmployee(string name, string id) {
    Employee* newEmployee = createEmployee(name, id);
    if (employeeHead == nullptr) {
        employeeHead = newEmployee;
    } else {
        Employee* temp = employeeHead;
        while (temp->nextEmployee != nullptr) {
            temp = temp->nextEmployee;
        }
        temp->nextEmployee = newEmployee;
    }
    cout << "Employee : " << name << " with ID : " << id << " added.\n";
}

// Function to add a project to an employee
void addProjectToEmployee(string employeeID, string projectName, int duration) {
    Employee* temp = employeeHead;
    while (temp != nullptr && temp->employeeID != employeeID) {
        temp = temp->nextEmployee;
    }

    if (temp == nullptr) {
        cout << "Employee with ID : " << employeeID << " not found.\n";
        return;
    }

    Project* newProject = createProject(projectName, duration);
    if (temp->projectHead == nullptr) {
        temp->projectHead = newProject;
    } else {
        Project* ptemp = temp->projectHead;
        while (ptemp->nextProject != nullptr) {
            ptemp = ptemp->nextProject;
        }
        ptemp->nextProject = newProject;
    }
    cout << "Project : " << projectName << " with duration : " << duration << " months added to employee : " << temp->employeeName << ".\n";
}

// Function to delete a project from an employee
void deleteProjectFromEmployee(string employeeID, string projectName) {
    Employee* temp = employeeHead;
    while (temp != nullptr && temp->employeeID != employeeID) {
        temp = temp->nextEmployee;
    }

    if (temp == nullptr) {
        cout << "Employee with ID : " << employeeID << " not found.\n";
        return;
    }

    Project* pTemp = temp->projectHead;
    Project* prev = nullptr;

    while (pTemp != nullptr && pTemp->projectName != projectName) {
        prev = pTemp;
        pTemp = pTemp->nextProject;
    }

    if (pTemp == nullptr) {
        cout << "Project : " << projectName << " not found for employee : " << temp->employeeName << ".\n";
        return;
    }

    if (prev == nullptr) {
        temp->projectHead = pTemp->nextProject;
    } else {
        prev->nextProject = pTemp->nextProject;
    }

    delete pTemp;
    cout << "Project : " << projectName << " removed from employee : " << temp->employeeName << ".\n";
}

// Function to display all employees and their projects
void displayEmployees() {
    Employee* temp = employeeHead;
    while (temp != nullptr) {
        cout << "Employee Name : " << temp->employeeName << ", ID : " << temp->employeeID << "\n";
        Project* pTemp = temp->projectHead;
        while (pTemp != nullptr) {
            cout << "Project : " << pTemp->projectName << ", Duration : " << pTemp->duration << " months\n";
            pTemp = pTemp->nextProject;
        }
        temp = temp->nextEmployee;
        cout << "\n";
    }
}

int main() {
    // Adding employees
    addEmployee("Andi", "P001");
    addEmployee("Budi", "P002");
    addEmployee("Citra", "P003");
    cout << "\n";
    displayEmployees();
    cout << "-----------------------------------------------------------------------------------------------\n";

    // Adding projects to employees
    addProjectToEmployee("P001", "Aplikasi Mobile", 12);
    addProjectToEmployee("P002", "Sistem Akuntansi", 8);
    addProjectToEmployee("P003", "E-commerce", 10);
    cout << "\n";
    displayEmployees();
    cout << "-----------------------------------------------------------------------------------------------\n";

    // Adding a new project to Andi
    addProjectToEmployee("P001", "Analisis Data", 6);
    cout << "\n";
    displayEmployees();
    cout << "-----------------------------------------------------------------------------------------------\n";

    // Deleting a project from Andi
    deleteProjectFromEmployee("P001", "Aplikasi Mobile");
    cout << "\n";
    displayEmployees();

    return 0;
}
```

Penjelasan :

Struktur Data

- Project: Representasi proyek dengan atribut:
    - projectName: Nama proyek.
    - duration: Durasi proyek (dalam bulan).
    - nextProject: Pointer ke proyek berikutnya.
- Employee: Representasi karyawan dengan atribut:
    - employeeName: Nama karyawan.
    - employeeID: ID karyawan.
    - projectHead: Pointer ke daftar proyek milik karyawan.
    - nextEmployee: Pointer ke karyawan berikutnya.

Fungsi Utama:

- createEmployee(): Membuat node karyawan baru.
- createProject(): Membuat node proyek baru.
- addEmployee(): Menambahkan karyawan baru ke daftar.
- addProjectToEmployee(): Menambahkan proyek baru ke karyawan tertentu berdasarkan ID.
- deleteProjectFromEmployee(): Menghapus proyek tertentu dari karyawan berdasarkan ID.
- displayEmployees(): Menampilkan daftar karyawan beserta semua proyek mereka.

Fungsi main():

- Menambahkan karyawan: Andi, Budi, Citra.
- Menambahkan proyek ke masing-masing karyawan.
- Menampilkan daftar karyawan dan proyek.
- Menambahkan proyek tambahan untuk Andi dan menghapus proyek

"Aplikasi Mobile" dari Andi.

Hasil Run :

```
Employee : Andi with ID : P001 added.
Employee : Budi with ID : P002 added.
Employee : Citra with ID : P003 added.

Employee Name : Andi, ID : P001

Employee Name : Budi, ID : P002

Employee Name : Citra, ID : P003

-----------------------------------------------------------------------------------------------
Project : Aplikasi Mobile with duration : 12 months added to employee : Andi.
Project : Sistem Akuntansi with duration : 8 months added to employee : Budi.
Project : E-commerce with duration : 10 months added to employee : Citra.

Employee Name : Andi, ID : P001
Project : Aplikasi Mobile, Duration : 12 months

Employee Name : Budi, ID : P002
Project : Sistem Akuntansi, Duration : 8 months

Employee Name : Citra, ID : P003
Project : E-commerce, Duration : 10 months

-----------------------------------------------------------------------------------------------
Project : Analisis Data with duration : 6 months added to employee : Andi.

Employee Name : Andi, ID : P001
Project : Aplikasi Mobile, Duration : 12 months
Project : Analisis Data, Duration : 6 months

Employee Name : Budi, ID : P002
Project : Sistem Akuntansi, Duration : 8 months

Employee Name : Citra, ID : P003
Project : E-commerce, Duration : 10 months

-----------------------------------------------------------------------------------------------
```

```
-----------------------------------------------------------------------------------------------
Project : Aplikasi Mobile removed from employee : Andi.

Employee Name : Andi, ID : P001
Project : Analisis Data, Duration : 6 months

Employee Name : Budi, ID : P002
Project : Sistem Akuntansi, Duration : 8 months

Employee Name : Citra, ID : P003
Project : E-commerce, Duration : 10 months
```

2. Unguided2.cpp

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

struct Book {
    string title;
    string returnDate;
    Book* nextBook;
};

struct Member {
    string name;
    string id;
    Book* bookHead;
    Member* nextMember;
};

// Head of the Member list
Member* memberHead = nullptr;

// Function to create a new member node
Member* createMember(string name, string id) {
    Member* newMember = new Member;
    newMember->name = name;
    newMember->id = id;
    newMember->bookHead = nullptr;
    newMember->nextMember = nullptr;
    return newMember;
}

// Function to create a new book node
Book* createBook(string title, string returnDate) {
    Book* newBook = new Book;
    newBook->title = title;
    newBook->returnDate = returnDate;
    newBook->nextBook = nullptr;
    return newBook;
}

// Function to add a member to the list
void addMember(string name, string id) {
    Member* newMember = createMember(name, id);
    if (memberHead == nullptr) {
        memberHead = newMember;
    } else {
        Member* temp = memberHead;
        while (temp->nextMember != nullptr) {
            temp = temp->nextMember;
        }
        temp->nextMember = newMember;
    }
    cout << "Member : " << name << " with ID : " << id << " added.\n";
}

// Function to add a book to a member
void addBookToMember(string memberId, string title, string returnDate) {
    Member* temp = memberHead;
    while (temp != nullptr && temp->id != memberId) {
        temp = temp->nextMember;
    }

    if (temp == nullptr) {
        cout << "Member with ID " << memberId << " not found.\n";
        return;
    }

    Book* newBook = createBook(title, returnDate);
    if (temp->bookHead == nullptr) {
        temp->bookHead = newBook;
    } else {
        Book* btemp = temp->bookHead;
        while (btemp->nextBook != nullptr) {
            btemp = btemp->nextBook;
        }
        btemp->nextBook = newBook;
    }
    cout << "Book : " << title << " with return date : " << returnDate << " added to member : " << temp->name << ".\n";
}

// Function to delete a member and all their books
void deleteMember(string memberId) {
    Member* temp = memberHead;
    Member* prev = nullptr;

    while (temp != nullptr && temp->id != memberId) {
        prev = temp;
        temp = temp->nextMember;
    }

    if (temp == nullptr) {
        cout << "Member with ID " << memberId << " not found.\n";
        return;
    }

    if (prev == nullptr) {
        memberHead = temp->nextMember;
    } else {
        prev->nextMember = temp->nextMember;
    }

    // Delete all books of the member
    Book* btemp = temp->bookHead;
    while (btemp != nullptr) {
        Book* toDelete = btemp;
        btemp = btemp->nextBook;
        delete toDelete;
    }

    delete temp;
    cout << "Member with ID : " << memberId << " deleted.\n";
}

// Function to display all members and their books
void displayMembers() {
    Member* temp = memberHead;
    while (temp != nullptr) {
        cout << "Member Name: " << temp->name << ", ID: " << temp->id << "\n";
        Book* btemp = temp->bookHead;
        while (btemp != nullptr) {
            cout << "  Book Title: " << btemp->title << ", Return Date: " << btemp->returnDate << "\n";
            btemp = btemp->nextBook;
        }
        temp = temp->nextMember;
        cout << "\n";
    }
}

int main() {
    // Adding members
    addMember("Rani", "A001");
    addMember("Jilo", "A002");
    addMember("Vina", "A003");
    cout << "\n";
    displayMembers();
    cout << "-----------------------------------------------------------------------------------------------\n";

    // Adding books to members
    addBookToMember("A001", "Pemrograman C++", "01/12/2024");
    addBookToMember("A003", "Algoritma Pemrograman", "15/12/2024");
    cout << "\n";
    displayMembers();
    cout << "-----------------------------------------------------------------------------------------------\n";

    // Adding a new book to Rani
    addBookToMember("A001", "Struktur Data", "10/12/2024");
    cout << "\n";
    displayMembers();
    cout << "-----------------------------------------------------------------------------------------------\n";

    // Deleting member Jito
    deleteMember("A002");
    cout << "\n";
    displayMembers();

    return 0;
}
```

Penjelasan :

Struktur Data:

- Book:
  - title: Nama buku.
  - returnDate: Tanggal pengembalian.
  - nextBook: Pointer ke buku berikutnya dalam daftar buku anggota.
- Member:
  - name: Nama anggota.
  - id: ID anggota.
  - bookHead: Pointer ke daftar buku yang dipinjam oleh anggota.
  - nextMember: Pointer ke anggota berikutnya dalam daftar anggota.

Fungsi Utama:

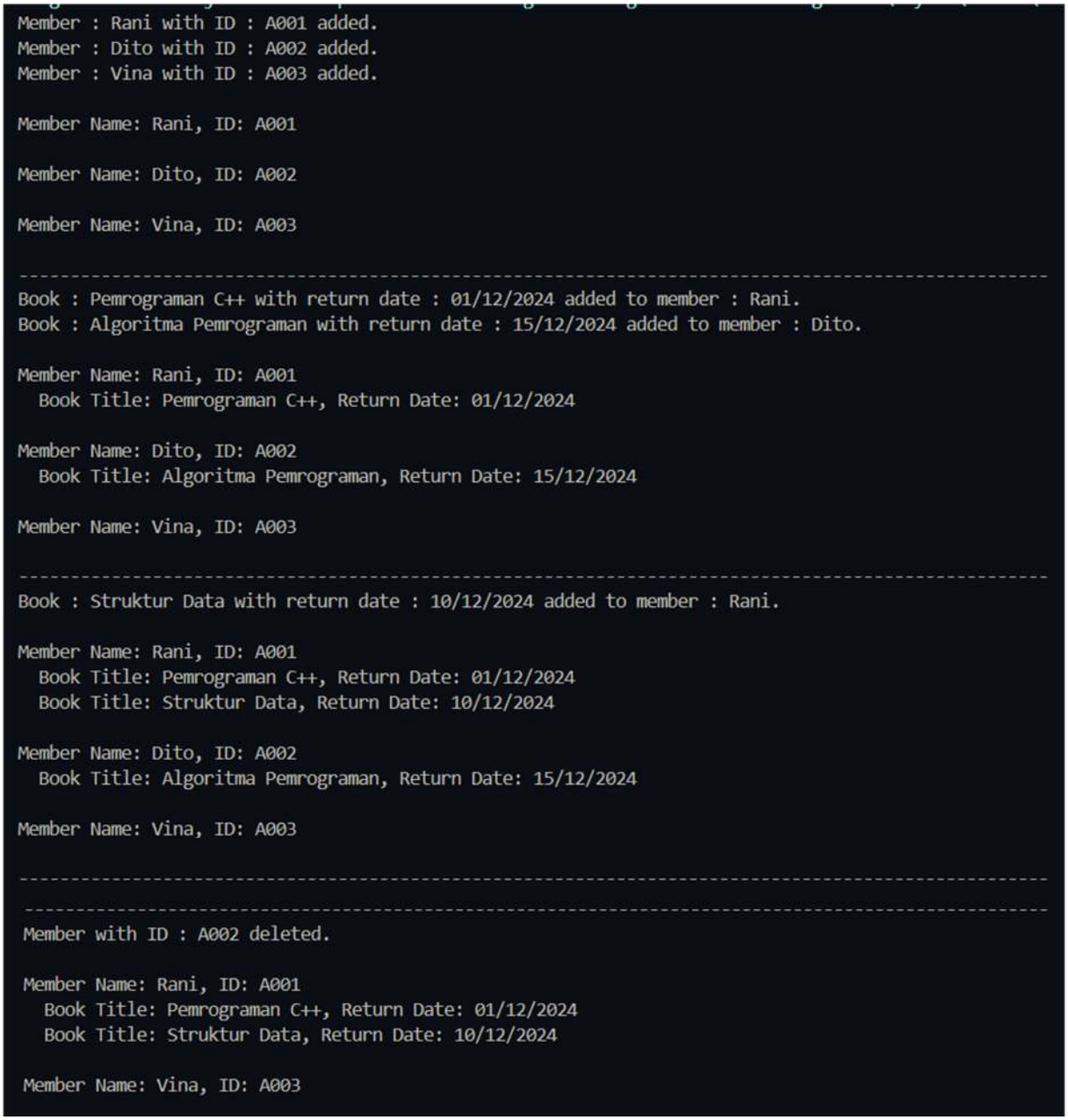
- createMember(): Membuat node anggota baru.
- createBook(): Membuat node buku baru.
- addMember(): Menambahkan anggota ke daftar.
- addBookToMember(): Menambahkan buku ke anggota tertentu berdasarkan ID.
- deleteMember(): Menghapus anggota beserta semua buku yang dipinjamnya.
- displayMembers(): Menampilkan daftar anggota dan buku yang dipinjam masing-masing anggota.

Fungsi main():

- Menambahkan anggota: Rani, Dito, dan Vina.
- Menambahkan buku ke anggota:
- Rani meminjam "Pemrograman C++".
- Dito meminjam "Algoritma Pemrograman".

- Menampilkan daftar anggota dan buku mereka.
- Menambahkan buku tambahan untuk Rani ("Struktur Data").
- Menghapus anggota Dito beserta semua buku yang dipinjamnya.

Hasil Run :



V. KESIMPULAN

Pada praktikum ini, telah berhasil dibuat dan diimplementasikan struktur data Multi Linked List. Multi Linked List merupakan salah satu variasi dari linked list yang mendukung koneksi antara elemen dalam lebih dari satu dimensi, seperti menghubungkan elemen berdasarkan kategori utama dan sub-kategori. Struktur ini sering digunakan untuk merepresentasikan data hierarkis atau relasional.

Dari hasil praktikum, dapat disimpulkan beberapa hal:

Fleksibilitas Struktur Data: Multi Linked List memungkinkan pengelolaan data yang lebih kompleks dibandingkan single atau double linked list, terutama dalam kasus di mana elemen data memiliki banyak hubungan atau relasi.

Operasi Dasar: Operasi seperti penambahan, penghapusan, dan pencarian elemen memerlukan pemahaman yang mendalam tentang struktur dan pointer, karena elemen dapat memiliki banyak hubungan antar node.

Efisiensi: Meski memberikan fleksibilitas, Multi Linked List cenderung lebih rumit untuk diimplementasikan dan memerlukan lebih banyak memori dibandingkan linked list sederhana, karena setiap node harus menyimpan lebih dari satu pointer.

Aplikasi Praktis: Struktur ini dapat digunakan dalam berbagai aplikasi nyata, seperti sistem manajemen basis data, representasi graf, atau penyimpanan data relasional dalam hierarki.

Kesulitan utama dalam implementasi Multi Linked List adalah menjaga konsistensi relasi antar node, terutama saat terjadi operasi modifikasi pada list. Oleh karena itu, penting untuk merancang algoritma yang efisien dan memastikan setiap perubahan diperiksa secara menyeluruh.

Secara keseluruhan, praktikum ini memberikan pemahaman yang baik mengenai konsep dasar dan implementasi Multi Linked List, serta pentingnya manajemen memori dan pointer dalam struktur data kompleks.