

**LAPORAN PRAKTIKUM**  
**Modul XIII**  
**“MULTI LINKED LIST”**



**Disusun Oleh:**  
**Zivana Afra Yulianto -2211104039**  
**SE-07-02**

**Dosen:**  
**Wahyu Andi Saputra**

**PROGRAM STUDI S1 SOFTWARE ENGINEERING**  
**FAKULTAS INFORMATIKA**  
**TELKOM UNIVERSITY**  
**PURWOKERTO**  
**2024**

# 1. Tujuan

Tujuan dari praktikum ini adalah:

1. Memahami konsep dan implementasi *Multi Linked List* dalam pengelolaan data yang saling terhubung.
2. Menggunakan *Multi Linked List* untuk memodelkan sistem manajemen data kompleks, seperti sistem manajemen buku perpustakaan.
3. Menerapkan operasi dasar pada *Multi Linked List*, seperti penambahan data (*insert*), penghapusan data (*delete*), dan penelusuran data (*traverse*).
4. Meningkatkan kemampuan analisis dalam menyelesaikan masalah pemrograman berbasis struktur data dinamis.

# 2. Landasan Teori

Linked List adalah struktur data linier yang terdiri dari kumpulan simpul (*nodes*) yang saling terhubung melalui pointer. Setiap simpul pada *Linked List* terdiri dari dua bagian utama:

1. Data: Bagian yang menyimpan informasi.
2. Pointer: Bagian yang menunjuk ke simpul berikutnya dalam daftar.

Pada *Multi Linked List*, setiap simpul dapat memiliki lebih dari satu hubungan (pointer) yang mengarah ke daftar lain. Hal ini memungkinkan penyimpanan data yang lebih kompleks, seperti hubungan antar entitas pada sistem manajemen perpustakaan.

Ciri-ciri Multi Linked List:

1. Setiap simpul utama (*parent node*) dapat memiliki daftar yang saling terkait (*child list*).
2. Operasi dasar seperti *insert*, *delete*, dan *traverse* diterapkan secara hierarkis sesuai dengan hubungan antar data.
3. Cocok untuk representasi data dengan banyak relasi, seperti basis data atau graf berarah.

Penerapan Multi Linked List dalam sistem manajemen buku perpustakaan:

1. Simpul utama (anggota): Berisi data anggota seperti nama dan ID.
2. Simpul anak (buku): Berisi data buku yang dipinjam oleh anggota, seperti judul dan tanggal pengembalian.
3. Hubungan antar simpul mencerminkan relasi antar anggota dan buku yang mereka pinjam.

Keunggulan Multi Linked List adalah kemampuannya untuk menyimpan data dinamis dengan relasi yang kompleks tanpa memerlukan struktur data yang kaku seperti tabel matriks.

### 3. Guided

#### GUIDED I:

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

using namespace std;

// Struktur Node untuk menyimpan data, pointer ke node berikutnya, dan pointer ke child
struct Node
{
    int data;
    Node *next;
    Node *child;

    // Constructor untuk inisialisasi node
    Node(int val) : data(val), next(nullptr), child(nullptr) {}
};

// Kelas MultiLinkedList untuk mengelola multi linked list
class MultiLinkedList
{
private:
    Node *head; // Pointer ke head list

public:
    // Constructor untuk menginisialisasi list kosong
    MultiLinkedList() : head(nullptr) {}

    // Menambahkan parent baru ke awal list
    void addParent(int data)
    {
        Node *newNode = new Node(data);
        newNode->next = head;
        head = newNode;
    }

    // Menambahkan child ke parent yang sesuai
    void addChild(int parentData, int childData)
    {
        Node *parent = head;

        // Mencari parent berdasarkan data
        while (parent != nullptr && parent->data != parentData)
        {
            parent = parent->next;
        }

        // Jika parent ditemukan, tambahkan child
        if (parent != nullptr)
        {
            Node *newChild = new Node(childData);
            newChild->next = parent->child;
            parent->child = newChild;
        }
        else
        {
            cout << "Parent not found!" << endl;
        }
    }

    // Menampilkan semua parent beserta child-nya
    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;
        }
    }

    // Destructor untuk membersihkan semua node
    ~MultiLinkedList()
    {
        while (head != nullptr)
        {
            Node *temp = head;
            head = head->next;

            // Hapus semua child dari parent
            while (temp->child != nullptr)
            {
                Node *childTemp = temp->child;
                temp->child = temp->child->next;
                delete childTemp;
            }

            // Hapus parent
            delete temp;
        }
    }
};

// Fungsi utama untuk menguji MultiLinkedList
int main()
{
    MultiLinkedList mList;

    // Menambahkan parent
    mList.addParent(1);
    mList.addParent(2);
    mList.addParent(3);

    // Menambahkan child ke parent yang sesuai
    mList.addChild(1, 10);
    mList.addChild(1, 11);
    mList.addChild(2, 20);
    mList.addChild(2, 21);
    mList.addChild(3, 30);
    mList.addChild(3, 31);

    // Menampilkan hasil
    mList.display();

    return 0;
}

```

## SCREENSHOOT OUTPUT :

```
Parent: 3 -> 31 30
Parent: 2 -> 21 20
Parent: 1 -> 11 10
PS C:\STD_Zivana_Afra_Yulianto>
```

## DESKRIPSI :

Program ini mengimplementasikan Multi Linked List di C++. Struktur ini memungkinkan setiap node dalam linked list utama (parent) memiliki daftar anak (child). Fitur utama program meliputi:

1. Penambahan Parent: Menambahkan node baru sebagai parent di awal linked list.
2. Penambahan Child: Menambahkan node child ke parent yang sesuai berdasarkan nilai data.
3. Menampilkan Struktur: Menampilkan semua parent beserta child yang dimilikinya.
4. Pengelolaan Memori: Membersihkan semua node (parent dan child) saat program selesai menggunakan destructor.

## GUIDED 2 :

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

using namespace std;

// Struktur EmployeeNode untuk menyimpan nama karyawan, pointer ke node
// berikutnya, dan pointer ke subordinate
struct EmployeeNode
{
    string name;
    EmployeeNode *next;
    EmployeeNode *subordinate;

    // Constructor untuk inisialisasi node karyawan
    EmployeeNode(string empName) : name(empName), next(nullptr),
    subordinate(nullptr) {}
};

// Kelas EmployeeList untuk mengelola struktur hirarki karyawan
class EmployeeList
{
private:
    EmployeeNode *head; // Pointer ke head list

public:
    // Constructor untuk menginisialisasi list kosong
    EmployeeList() : head(nullptr) {}

    // Menambahkan karyawan baru ke awal list
    void addEmployee(string name)
    {
        EmployeeNode *newEmployee = new EmployeeNode(name);
        newEmployee->next = head;
        head = newEmployee;
    }
}
```

```

// Menambahkan subordinate ke karyawan yang sesuai
void addSubordinate(string managerName, string subordinateName)
{
    EmployeeNode *manager = head;

    // Mencari manager berdasarkan nama
    while (manager != nullptr && manager->name != managerName)
    {
        manager = manager->next;
    }

    // Jika manager ditemukan, tambahkan subordinate
    if (manager != nullptr)
    {
        EmployeeNode *newSubordinate = new EmployeeNode(subordinateName);
        newSubordinate->next = manager->subordinate;
        manager->subordinate = newSubordinate;
    }
    else
    {
        cout << "Manager not found!" << endl;
    }
}

// Menampilkan semua manager beserta subordinate-nya
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;
    }
}

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

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

        // Hapus manager
        delete temp;
    }
}

// Fungsi utama untuk menguji EmployeeList
int main()
{
    EmployeeList empList;

```

```

// Menambahkan manager
empList.addEmployee("Alice");
empList.addEmployee("Bob");
empList.addEmployee("Charlie");

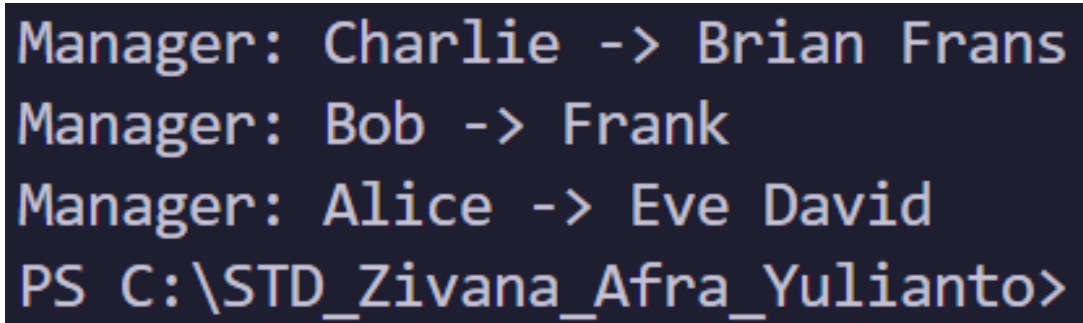
// Menambahkan subordinate ke manager yang sesuai
empList.addSubordinate("Alice", "David");
empList.addSubordinate("Alice", "Eve");
empList.addSubordinate("Bob", "Frank");
empList.addSubordinate("Charlie", "Frans");
empList.addSubordinate("Charlie", "Brian");

// Menampilkan hasil
empList.display();

return 0;
}

```

Screenshoot output :



```

Manager: Charlie -> Brian Frans
Manager: Bob -> Frank
Manager: Alice -> Eve David
PS C:\STD_Zivana_Afra_Yulianto>

```

Deskripsi :

Program ini mengimplementasikan struktur hirarki karyawan menggunakan Multi Linked List di C++. Struktur ini memungkinkan setiap karyawan (manager) memiliki daftar bawahan (subordinate). Fitur utama program meliputi:

1. Penambahan Karyawan (Manager): Menambahkan karyawan baru ke awal daftar.
2. Penambahan Bawahan: Menambahkan bawahan ke karyawan (manager) yang sesuai berdasarkan nama.
3. Menampilkan Hirarki: Menampilkan seluruh struktur organisasi, menunjukkan setiap manager beserta daftar bawahannya.
4. Pengelolaan Memori: Membersihkan semua node (manager dan bawahan) secara otomatis saat program selesai menggunakan destructor.

### GUIDED 3:

```
#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

    cout << endl;

    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;
}

```

SCREENSHOOT OUTPUT :

```

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 C:\STD_Zivana_Afra_Yulianto> ^C

```

## DESKRIPSI :

Program ini mengimplementasikan struktur hirarki karyawan menggunakan Multi Linked List di C++. Struktur ini memungkinkan setiap karyawan (manager) memiliki daftar bawahan (subordinate). Fitur utama program meliputi:

1. Penambahan Karyawan (Manager): Menambahkan karyawan baru ke awal daftar.
2. Penambahan Bawahan: Menambahkan bawahan ke karyawan (manager) yang sesuai berdasarkan nama.
3. Menampilkan Hirarki: Menampilkan seluruh struktur organisasi, menunjukkan setiap manager beserta daftar bawahannya.
4. Pengelolaan Memori: Membersihkan semua node (manager dan bawahan) secara otomatis saat program selesai menggunakan destructor.

## Cara Kerja:

- Program menggunakan kelas `EmployeeList` untuk mengelola struktur organisasi.
- Fungsi `addEmployee` menambahkan karyawan ke daftar utama.
- Fungsi `addSubordinate` menambahkan bawahan ke karyawan tertentu.
- Fungsi `display` mencetak seluruh hirarki organisasi.

## 4. Unguided

### UNGUIDED 1 :

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

using namespace std;

// Struktur untuk node proyek
struct ProjectNode
{
    string projectName; // Nama proyek
    int duration;        // Durasi proyek (bulan)
    ProjectNode *next;   // Pointer ke proyek berikutnya

    ProjectNode(string name, int dur) : projectName(name), duration(dur),
    next(nullptr) {}
};

// Struktur untuk node pegawai
struct EmployeeNode
{
    string employeeName; // Nama pegawai
    string employeeID;   // ID pegawai
    EmployeeNode *next;  // Pointer ke pegawai berikutnya
    ProjectNode *projects; // Pointer ke daftar proyek

    EmployeeNode(string name, string id) : employeeName(name), employeeID(id),
    next(nullptr), projects(nullptr) {}
};

// Kelas untuk mengelola Multi Linked List Pegawai dan Proyek
class EmployeeProjectList
{
private:
    EmployeeNode *head; // Pointer ke kepala daftar pegawai
```

```

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

    // Menambahkan pegawai baru
    void addEmployee(string name, string id)
    {
        EmployeeNode *newEmployee = new EmployeeNode(name, id);
        newEmployee->next = head;
        head = newEmployee;
    }

    // Menambahkan proyek ke pegawai tertentu
    void addProject(string employeeID, string projectName, int duration)
    {
        EmployeeNode *employee = head;
        while (employee != nullptr && employee->employeeID != employeeID)
        {
            employee = employee->next;
        }

        if (employee != nullptr)
        { // Jika pegawai ditemukan
            ProjectNode *newProject = new ProjectNode(projectName, duration);
            newProject->next = employee->projects;
            employee->projects = newProject;
        }
        else
        {
            cout << "Employee not found!" << endl;
        }
    }

    // Menghapus proyek dari pegawai tertentu
    void deleteProject(string employeeID, string projectName)
    {
        EmployeeNode *employee = head;
        while (employee != nullptr && employee->employeeID != employeeID)
        {
            employee = employee->next;
        }

        if (employee != nullptr)
        { // Jika pegawai ditemukan
            ProjectNode **current = &(employee->projects);
            while (*current != nullptr && (*current)->projectName != projectName)
            {
                current = &((*current)->next);
            }

            if (*current != nullptr)
            { // Jika proyek ditemukan
                ProjectNode *toDelete = *current;
                *current = (*current)->next;
                delete toDelete;
                cout << "Project " << projectName << " deleted from " << employee->employeeName << "." << endl;
            }
            else
            {
                cout << "Project not found!" << endl;
            }
        }
        else
        {
            cout << "Employee not found!" << endl;
        }
    }
}

```

```

// Menampilkan daftar pegawai dan proyek mereka
void display()
{
    EmployeeNode *current = head;
    while (current != nullptr)
    {
        cout << "Employee: " << current->employeeName << " (ID: " << current->employeeID << ")\nProjects: ";
        ProjectNode *project = current->projects;
        if (project == nullptr)
        {
            cout << "No projects assigned.";
        }
        else
        {
            while (project != nullptr)
            {
                cout << project->projectName << " (" << project->duration << " months) ";
                project = project->next;
            }
            cout << "\n";
            current = current->next;
        }
    }
}

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

        while (temp->projects != nullptr)
        {
            ProjectNode *projectTemp = temp->projects;
            temp->projects = temp->projects->next;
            delete projectTemp;
        }
        delete temp;
    }
}

};

int main()
{
    EmployeeProjectList empList;

    // Menambahkan data pegawai
    empList.addEmployee("Andi", "P001");
    empList.addEmployee("Budi", "P002");
    empList.addEmployee("Citra", "P003");

    // Menambahkan proyek ke pegawai
    empList.addProject("P001", "Aplikasi Mobile", 12);
    empList.addProject("P002", "Sistem Akuntansi", 8);
    empList.addProject("P003", "E-commerce", 10);

    // Menambahkan proyek baru
    empList.addProject("P001", "Analisis Data", 6);

    // Menghapus proyek dari pegawai
    empList.deleteProject("P001", "Aplikasi Mobile");

    // Menampilkan data pegawai dan proyek mereka
    cout << "\nData Pegawai dan Proyek:\n";
    empList.display();

    return 0;
}

```

## SCREENSHOOT OUTPUT :

```
Project Aplikasi Mobile deleted from Andi.

Data Pegawai dan Proyek:
Employee: Citra (ID: P003)
Projects: E-commerce (10 months)
Employee: Budi (ID: P002)
Projects: Sistem Akuntansi (8 months)
Employee: Andi (ID: P001)
Projects: Analisis Data (6 months)
PS C:\STD_Zivana_Afra_Yulianto>
```

## DESKRIPSI :

Program di atas menggunakan Multi Linked List untuk mengelola data pegawai dan proyek. Berikut adalah fungsionalitas utama:

1. Tambah Pegawai: Data pegawai berupa nama dan ID ditambahkan ke daftar.
2. Tambah Proyek: Proyek baru dengan nama dan durasi ditambahkan ke pegawai tertentu berdasarkan ID.
3. Hapus Proyek: Proyek tertentu dapat dihapus dari pegawai.
4. Tampilkan Data: Menampilkan semua pegawai beserta proyek yang mereka kerjakan.

## UNDUIDED 2 :

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

// Struktur untuk data buku
struct Book
{
    string title;
    string returnDate;
    Book *nextBook;

    Book(string t, string r) : title(t), returnDate(r), nextBook(nullptr) {}
};

// Struktur untuk data anggota
struct Member
{
    string name;
    string id;
    Book *borrowedBooks;
    Member *nextMember;

    Member(string n, string i) : name(n), id(i), borrowedBooks(nullptr),
    nextMember(nullptr) {}
};
```

```

// Kelas untuk mengelola sistem perpustakaan
class Library
{
private:
    Member *head;

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

    // Menambahkan anggota baru
    void addMember(string name, string id)
    {
        Member *newMember = new Member(name, id);
        newMember->nextMember = head;
        head = newMember;
    }

    // Menambahkan buku ke anggota tertentu
    void addBook(string memberId, string title, string returnDate)
    {
        Member *member = findMember(memberId);
        if (member)
        {
            Book *newBook = new Book(title, returnDate);
            newBook->nextBook = member->borrowedBooks;
            member->borrowedBooks = newBook;
        }
        else
        {
            cout << "Anggota dengan ID " << memberId << " tidak ditemukan!\n";
        }
    }

    // Menghapus anggota beserta buku yang dipinjam
    void deleteMember(string memberId)
    {
        Member *current = head;
        Member *previous = nullptr;

        while (current && current->id != memberId)
        {
            previous = current;
            current = current->nextMember;
        }

        if (!current)
        {
            cout << "Anggota dengan ID " << memberId << " tidak ditemukan!\n";
            return;
        }

        // Hapus semua buku yang dipinjam oleh anggota
        Book *book = current->borrowedBooks;
        while (book)
        {
            Book *temp = book;
            book = book->nextBook;
            delete temp;
        }

        // Hapus anggota dari daftar
        if (previous)
        {
            previous->nextMember = current->nextMember;
        }
        else
        {
            head = current->nextMember;
        }

        delete current;
        cout << "Anggota dengan ID " << memberId << " berhasil dihapus.\n";
    }
}

```

```

// Menampilkan data anggota dan buku yang dipinjam
void displayData()
{
    Member *current = head;
    while (current)
    {
        cout << "Nama: " << current->name << ", ID: " << current->id << "\n";
        Book *book = current->borrowedBooks;
        while (book)
        {
            cout << " - Judul Buku: " << book->title << ", Tanggal Pengembalian: " << book->returnDate << "\n";
            book = book->nextBook;
        }
        current = current->nextMember;
    }
}

private:
// Mencari anggota berdasarkan ID
Member *findMember(string memberId)
{
    Member *current = head;
    while (current)
    {
        if (current->id == memberId)
        {
            return current;
        }
        current = current->nextMember;
    }
    return nullptr;
};

int main()
{
    Library library;

    // Menambahkan data anggota
    library.addMember("Rani", "A001");
    library.addMember("Dito", "A002");
    library.addMember("Vina", "A003");

    // Menambahkan buku yang dipinjam
    library.addBook("A001", "Pemrograman C++", "01/12/2024");
    library.addBook("A002", "Algoritma Pemrograman", "15/12/2024");

    // Menambahkan buku baru untuk Rani
    library.addBook("A001", "Struktur Data", "10/12/2024");

    // Menghapus anggota Dito beserta buku yang dipinjam
    library.deleteMember("A002");

    // Menampilkan seluruh data anggota dan buku yang dipinjam
    cout << "\nData Anggota dan Buku yang Dipinjam:\n";
    library.displayData();

    return 0;
}

```



Screenshoot output :

```
Anggota dengan ID A002 berhasil dihapus.

Data Anggota dan Buku yang Dipinjam:
Nama: Vina, ID: A003
Nama: Rani, ID: A001
  - Judul Buku: Struktur Data, Tanggal Pengembalian: 10/12/2024
  - Judul Buku: Pemrograman C++, Tanggal Pengembalian: 01/12/2024
PS C:\STD_Zivana Afra Yulianto>
```

Deskripsi :

Kode di atas mengimplementasikan sistem manajemen perpustakaan dengan Multi Linked List. Fungsi utama mencakup:

1. Menambah anggota: Menambahkan anggota baru ke dalam daftar.
2. Menambah buku: Menambahkan buku yang dipinjam oleh anggota tertentu.
3. Menghapus anggota: Menghapus anggota beserta semua buku yang dipinjam.
4. Menampilkan data: Menampilkan semua anggota dan buku yang mereka pinjam.

## 5. Kesimpulan

Dari praktikum ini, dapat disimpulkan bahwa:

1. *Multi Linked List* adalah struktur data yang efektif untuk menyelesaikan masalah pengelolaan data dengan relasi hierarkis, seperti sistem manajemen perpustakaan.
2. Implementasi *Multi Linked List* memungkinkan pengelolaan data yang fleksibel, termasuk penambahan dan penghapusan data pada level yang berbeda.
3. Dengan *Multi Linked List*, relasi antara data anggota perpustakaan dan buku yang dipinjam dapat dimodelkan dengan cara yang terstruktur dan efisien.
4. Pemahaman konsep dasar dan operasi pada *Multi Linked List* merupakan langkah penting dalam mempelajari struktur data tingkat lanjut dan penerapannya pada kasus dunia nyata.