LAPORAN PRAKTIKUM
Modul 13
MULTI LINKED LIST

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 konsep dan implementasi Multi Linked List
2. Mampu menerapkan Multi Linked List dalam berbagai studi kasus

Ringkasan Konsep Multi Linked List

Definisi: Multi Linked List adalah struktur data yang terdiri dari beberapa list yang saling terhubung, di mana setiap elemen dapat membentuk list tersendiri. Struktur ini biasanya terdiri dari:

- List Induk (Parent List)
- List Anak (Child List)

Komponen Utama:

1. **List Induk:**
   - Menyimpan data utama
   - Memiliki pointer ke list anak
2. **List Anak:**
   - Terhubung dengan elemen induk
   - Dapat memiliki multiple elemen

Operasi Dasar:

1. **Operasi Insert:**
   - Insert Anak:
     - Memerlukan identifikasi elemen induk terlebih dahulu
     - Biasanya menggunakan konsep insert last
   - Insert Induk:
     - Mirip dengan operasi insert pada linked list biasa
2. **Operasi Delete:**
   - Delete Anak:
     - Memerlukan identifikasi induk terlebih dahulu
     - Menghapus elemen anak spesifik
   - Delete Induk:
     - Menghapus elemen induk
     - Otomatis menghapus semua anak yang terkait

Implementasi:

- Menggunakan pointer untuk menghubungkan elemen
- Memiliki struktur data terpisah untuk induk dan anak
- Memerlukan manajemen memori yang baik untuk alokasi dan dealokasi

Kegunaan:

- Manajemen data hierarkis
- Organisasi data yang memiliki relasi parent-child
- Implementasi sistem yang membutuhkan struktur data bertingkat

Contoh Aplikasi:

1. Sistem Manajemen Pegawai dan Proyek
2. Sistem Perpustakaan (Anggota dan Buku)
3. Struktur Organisasi
4. Manajemen Data Mahasiswa

Kelebihan:

- Fleksibel dalam menangani data bertingkat
- Efisien dalam operasi pencarian
- Memudahkan pengelolaan data yang saling berhubungan

Catatan Penting:

- Memerlukan pemahaman yang baik tentang pointer
- Perlu memperhatikan manajemen memori
- Penting untuk menjaga konsistensi antara list induk dan anak

Guided

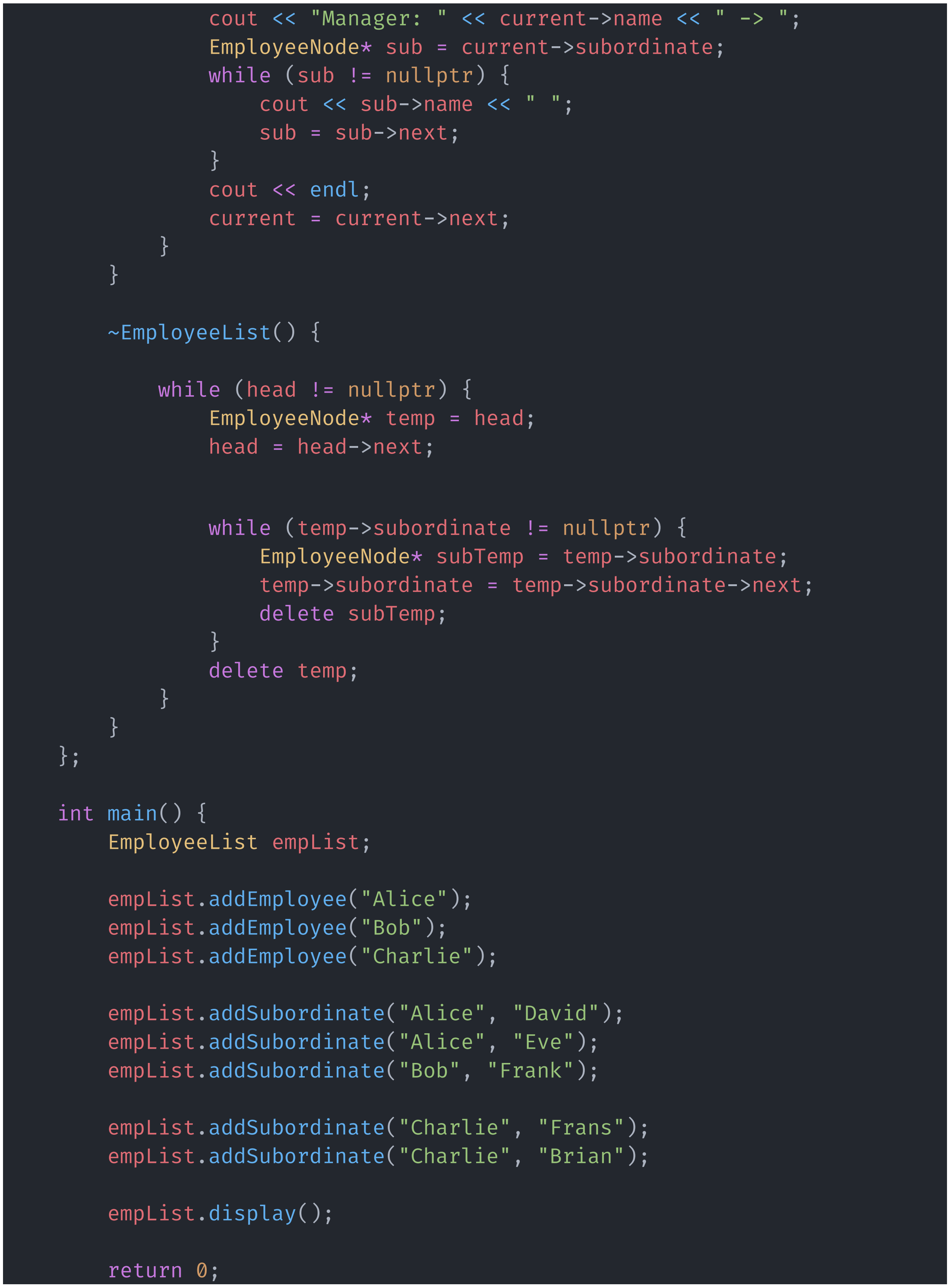
Guided1.cpp

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

using namespace std;


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

```cpp
    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;
}
```

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) {
```



```cpp
            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;
    }
```

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);
```

```cpp
            }

            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;
}
```

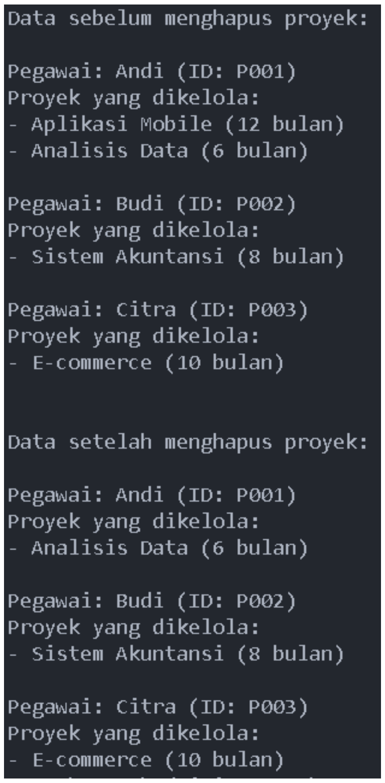
Unguided

unGuided1.cpp

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

// Struktur untuk proyek
struct Project
{
    string name;
```

```cpp
    int duration;
    Project *next;
};

// Struktur untuk pegawai
struct Employee
{
    string name;
    string id;
    Project *firstProject;
    Employee *next;
};

class EmployeeManagement
{
private:
    Employee *head;

public:
    EmployeeManagement()
    {
        head = NULL;
    }

    // Fungsi untuk menambah pegawai baru
    void addEmployee(string name, string id)
    {
        Employee *newEmployee = new Employee;
        newEmployee->name = name;
        newEmployee->id = id;
        newEmployee->firstProject = NULL;
        newEmployee->next = NULL;

        if (head == NULL)
        {
            head = newEmployee;
        }
        else
        {
            Employee *current = head;
            while (current->next != NULL)
            {
                current = current->next;
            }
            current->next = newEmployee;
        }
    }

    // Fungsi untuk menambah proyek ke pegawai
    void addProject(string employeeId, string projectName, int duration)
    {
        Employee *emp = findEmployee(employeeId);
        if (emp != NULL)
        {
            Project *newProject = new Project;
            newProject->name = projectName;
            newProject->duration = duration;
            newProject->next = NULL;

            if (emp->firstProject == NULL)
            {
                emp->firstProject = newProject;
            }
            else
            {
                Project *current = emp->firstProject;
                while (current->next != NULL)
                {
                    current = current->next;
                }
                current->next = newProject;
            }
        }
    }

    // Fungsi untuk mencari pegawai berdasarkan ID
    Employee *findEmployee(string id)
    {
        Employee *current = head;
        while (current != NULL)
        {
            if (current->id == id)
            {
                return current;
            }
            current = current->next;
        }
        return NULL;
    }
    // Fungsi untuk menghapus proyek
    void deleteProject(string employeeId, string projectName)
    {
        Employee *emp = findEmployee(employeeId);
        if (emp != NULL && emp->firstProject != NULL)
        {
            Project *current = emp->firstProject;
            Project *prev = NULL;

            // Jika proyek yang akan dihapus ada di awal
            if (current->name == projectName)
            {
                emp->firstProject = current->next;
                delete current;
                return;
            }

            // Mencari proyek yang akan dihapus
            while (current != NULL && current->name != projectName)
            {
                prev = current;
                current = current->next;
            }

            // Jika proyek ditemukan
            if (current != NULL)
            {
                prev->next = current->next;
                delete current;
            }
        }
    }

    // Fungsi untuk menampilkan semua data
    void displayAll()
    {
        Employee *currentEmp = head;
        while (currentEmp != NULL)
        {
            cout << "\nPegawai: " << currentEmp->name << " (ID: " <<
currentEmp->id << ")" << endl;
            cout << "Proyek yang dikelola:" << endl;

            Project *currentProj = currentEmp->firstProject;
            if (currentProj == NULL)
            {
                cout << "- Tidak ada proyek" << endl;
            }
            while (currentProj != NULL)
            {
                cout << "- " << currentProj->name << " (" << currentProj-
>duration << " bulan)" << endl;
                currentProj = currentProj->next;
            }
            currentEmp = currentEmp->next;
        }
    }
};

int main()
{
    EmployeeManagement em;

    // 1. Menambahkan pegawai
    em.addEmployee("Andi", "P001");
    em.addEmployee("Budi", "P002");
    em.addEmployee("Citra", "P003");

    // 2. Menambahkan proyek ke pegawai
    em.addProject("P001", "Aplikasi Mobile", 12);
    em.addProject("P002", "Sistem Akuntansi", 8);
    em.addProject("P003", "E-commerce", 10);

    // 3. Menambahkan proyek baru untuk Andi
    em.addProject("P001", "Analisis Data", 6);

    cout << "Data sebelum menghapus proyek:" << endl;
    em.displayAll();

    // 4. Menghapus proyek "Aplikasi Mobile" dari Andi
    em.deleteProject("P001", "Aplikasi Mobile");

    cout << "\n\nData setelah menghapus proyek:" << endl;
    em.displayAll();

    return 0;
}
```

Output

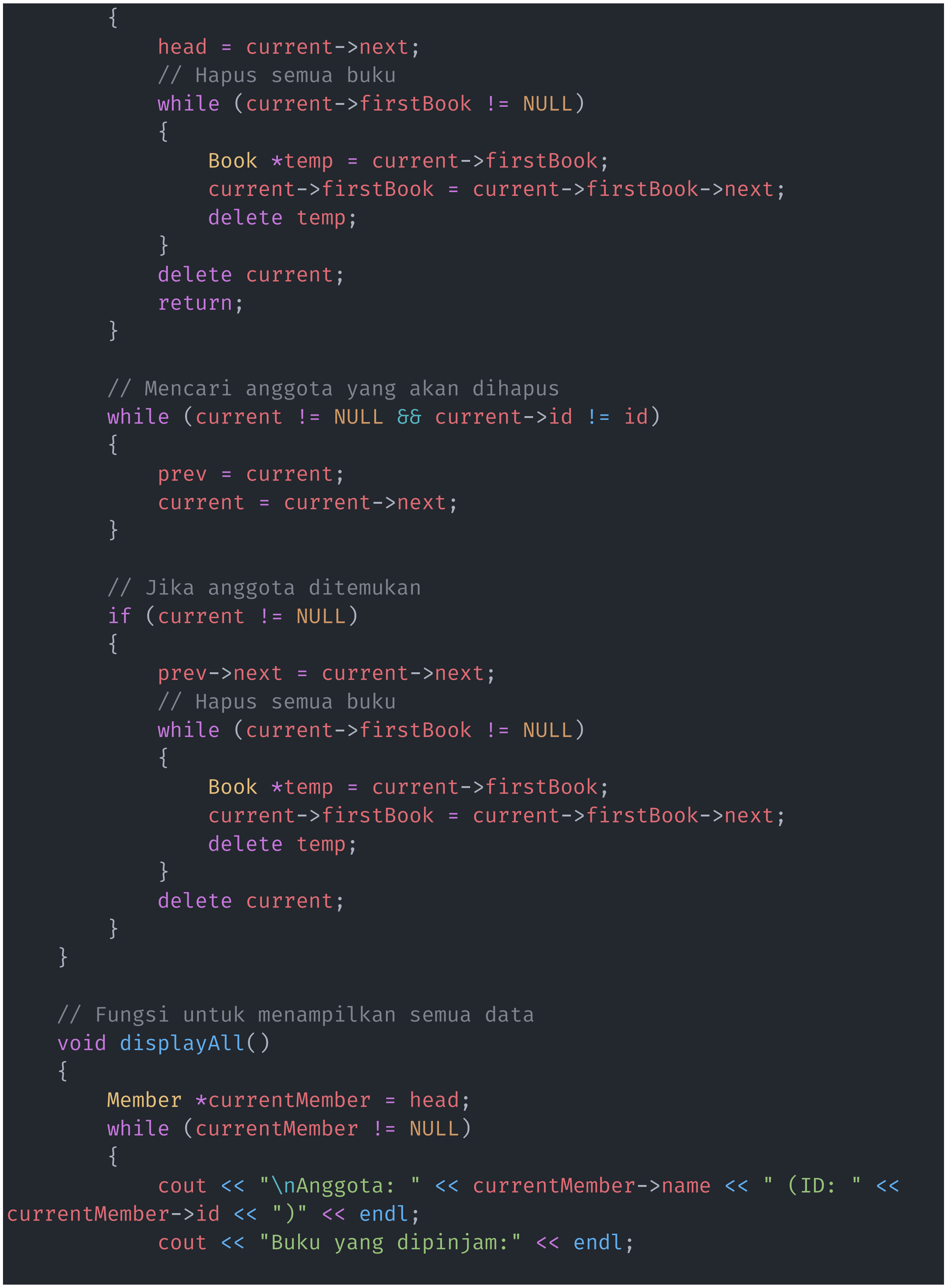


```
Data sebelum menghapus proyek:

Pegawai: Andi (ID: P001)
Proyek yang dikelola:
- Aplikasi Mobile (12 bulan)
- Analisis Data (6 bulan)

Pegawai: Budi (ID: P002)
Proyek yang dikelola:
- Sistem Akuntansi (8 bulan)

Pegawai: Citra (ID: P003)
Proyek yang dikelola:
- E-commerce (10 bulan)


Data setelah menghapus proyek:

Pegawai: Andi (ID: P001)
Proyek yang dikelola:
- Analisis Data (6 bulan)

Pegawai: Budi (ID: P002)
Proyek yang dikelola:
- Sistem Akuntansi (8 bulan)

Pegawai: Citra (ID: P003)
Proyek yang dikelola:
- E-commerce (10 bulan)
```

unGuided2.cpp

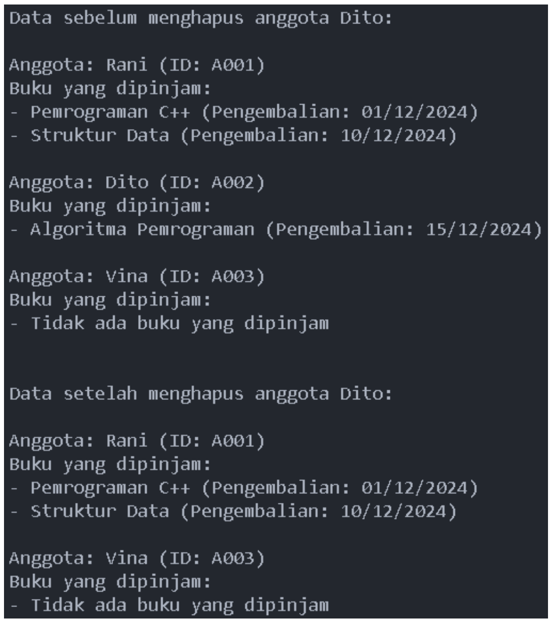
```cpp
#include <iostream>
#include <string>
using namespace std;

// Struktur untuk buku
struct Book
{
    string title;
    string returnDate;
    Book *next;
};

// Struktur untuk anggota
struct Member
{
    string name;
    string id;
    Book *firstBook;
    Member *next;
```

```
};

class LibraryManagement
{
private:
    Member *head;

public:
    LibraryManagement()
    {
        head = NULL;
    }

    // Fungsi untuk menambah anggota baru
    void addMember(string name, string id)
    {
        Member *newMember = new Member;
        newMember->name = name;
        newMember->id = id;
        newMember->firstBook = NULL;
        newMember->next = NULL;

        if (head == NULL)
        {
            head = newMember;
        }
        else
        {
            Member *current = head;
            while (current->next != NULL)
            {
                current = current->next;
            }
            current->next = newMember;
        }
    }

    // Fungsi untuk menambah buku ke anggota
    void addBook(string memberId, string bookTitle, string returnDate)
    {
        Member *member = findMember(memberId);
        if (member != NULL)
        {
            Book *newBook = new Book;
            newBook->title = bookTitle;
            newBook->returnDate = returnDate;
            newBook->next = NULL;

            if (member->firstBook == NULL)
            {
                member->firstBook = newBook;
            }
            else
            {
                Book *current = member->firstBook;
                while (current->next != NULL)
                {
                    current = current->next;
                }
                current->next = newBook;
            }
        }
    }

    // Fungsi untuk mencari anggota berdasarkan ID
    Member *findMember(string id)
    {
        Member *current = head;
        while (current != NULL)
        {
            if (current->id == id)
            {
                return current;
            }
            current = current->next;
        }
        return NULL;
    }

    // Fungsi untuk menghapus anggota beserta bukunya
    void deleteMember(string id)
    {
        if (head == NULL)
            return;

        Member *current = head;
        Member *prev = NULL;

        // Jika anggota yang akan dihapus ada di awal
        if (current != NULL && current->id == id)
```



```
        {
            head = current->next;
            // Hapus semua buku
            while (current->firstBook != NULL)
            {
                Book *temp = current->firstBook;
                current->firstBook = current->firstBook->next;
                delete temp;
            }
            delete current;
            return;
        }

        // Mencari anggota yang akan dihapus
        while (current != NULL && current->id != id)
        {
            prev = current;
            current = current->next;
        }

        // Jika anggota ditemukan
        if (current != NULL)
        {
            prev->next = current->next;
            // Hapus semua buku
            while (current->firstBook != NULL)
            {
                Book *temp = current->firstBook;
                current->firstBook = current->firstBook->next;
                delete temp;
            }
            delete current;
        }
    }

    // Fungsi untuk menampilkan semua data
    void displayAll()
    {
        Member *currentMember = head;
        while (currentMember != NULL)
        {
            cout << "\nAnggota: " << currentMember->name << " (ID: " <<
currentMember->id << ")" << endl;
            cout << "Buku yang dipinjam:" << endl;
            Book *currentBook = currentMember->firstBook;
            if (currentBook == NULL)
            {
                cout << "- Tidak ada buku yang dipinjam" << endl;
            }
            while (currentBook != NULL)
            {
                cout << "- " << currentBook->title << " (Pengembalian: "
<< currentBook->returnDate << ")" << endl;
                currentBook = currentBook->next;
            }
            currentMember = currentMember->next;
        }
    }
};

int main()
{
    LibraryManagement lm;

    // 1. Menambahkan anggota
    lm.addMember("Rani", "A001");
    lm.addMember("Dito", "A002");
    lm.addMember("Vina", "A003");

    // 2. Menambahkan buku yang dipinjam
    lm.addBook("A001", "Pemrograman C++", "01/12/2024");
    lm.addBook("A002", "Algoritma Pemrograman", "15/12/2024");

    // 3. Menambahkan buku baru untuk Rani
    lm.addBook("A001", "Struktur Data", "10/12/2024");

    cout << "Data sebelum menghapus anggota Dito:" << endl;
    lm.displayAll();

    // 4. Menghapus anggota Dito beserta buku yang dipinjam
    lm.deleteMember("A002");

    cout << "\n\nData setelah menghapus anggota Dito:" << endl;
    lm.displayAll();

    return 0;
}
```

Output



```
Data sebelum menghapus anggota Dito:

Anggota: Rani (ID: A001)
Buku yang dipinjam:
- Pemrograman C++ (Pengembalian: 01/12/2024)
- Struktur Data (Pengembalian: 10/12/2024)

Anggota: Dito (ID: A002)
Buku yang dipinjam:
- Algoritma Pemrograman (Pengembalian: 15/12/2024)

Anggota: Vina (ID: A003)
Buku yang dipinjam:
- Tidak ada buku yang dipinjam


Data setelah menghapus anggota Dito:

Anggota: Rani (ID: A001)
Buku yang dipinjam:
- Pemrograman C++ (Pengembalian: 01/12/2024)
- Struktur Data (Pengembalian: 10/12/2024)

Anggota: Vina (ID: A003)
Buku yang dipinjam:
- Tidak ada buku yang dipinjam
```