LAPORAN PRAKTIKUM
MODUL 13
MULTI LINKED LIST

Disusun Oleh:

Satria Putra Dharma Prayudha - 21104036

SE07-02

Dosen :

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

A. Tujuan

1. Memahami penggunaan Multi Linked list.
2. Mengimplementasikan Multi Linked list dalam beberapa studi kasus.

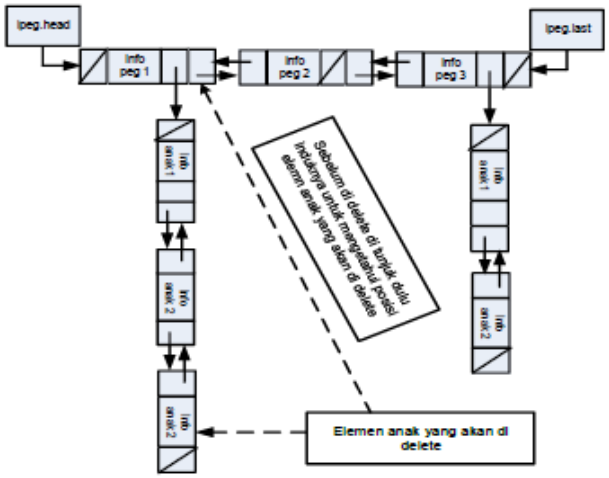
B. Landasan Teori

Landasan teori ini berdasarkan pada modul pembelajaran praktikum struktur data kali ini :

2.1 Multi Linked List

Multi Linked List adalah struktur data yang terdiri dari sekumpulan list yang berbeda, tetapi memiliki keterhubungan satu sama lain. Setiap elemen dalam Multi Linked List dapat membentuk list-nya sendiri, yang dikenal sebagai list anak, sementara list utama disebut sebagai list induk. Dalam implementasi Multi Linked List, setiap elemen induk dapat memiliki referensi ke beberapa elemen anak.

Misalnya, pada sebuah organisasi, list induk bisa berupa daftar pegawai, dan setiap pegawai bisa memiliki sublist berupa daftar anak tugas atau proyek yang mereka kelola. Dengan demikian, Multi Linked List memungkinkan representasi struktur hierarkis yang lebih kompleks.



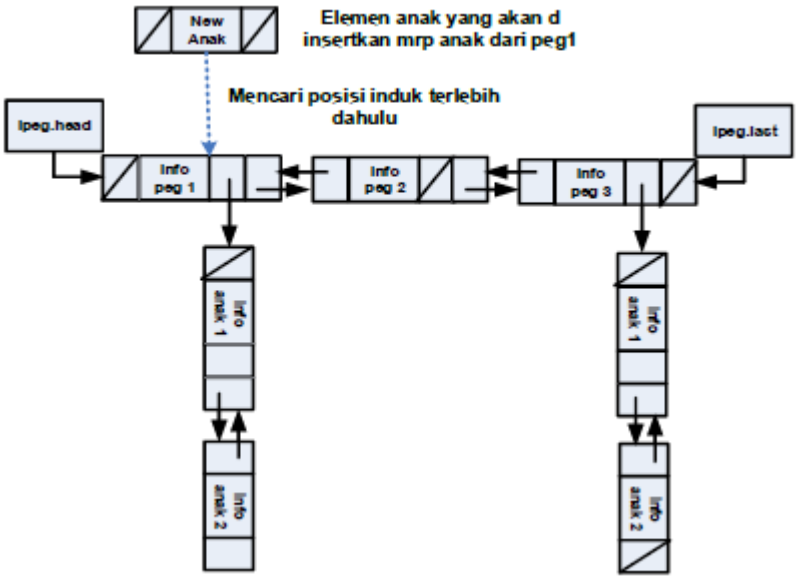
Multi Linked List

2.1.1 Insert

A. Insert Anak

Proses penambahan elemen anak harus dimulai dengan menemukan elemen induknya terlebih dahulu. Setelah induk ditemukan, elemen anak baru dapat disisipkan.

Penambahan elemen anak biasanya dilakukan pada posisi terakhir dalam list anak (insert last). Setelah posisi induk ditemukan, elemen anak disisipkan dengan menghubungkan elemen terakhir dari list anak ke elemen anak baru yang akan ditambahkan.



Multi Linked List Insert Anak

B. Insert Induk

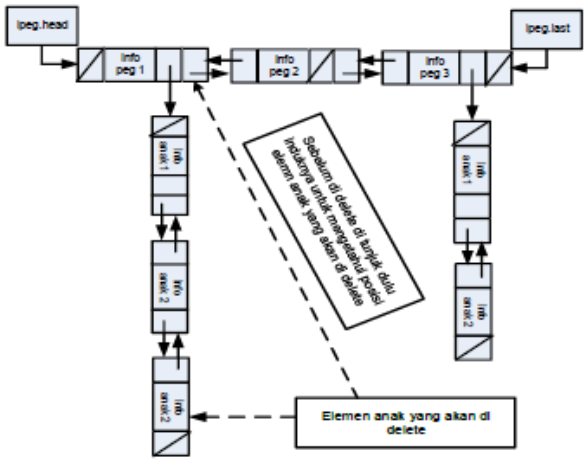
Proses penambahan elemen induk dalam Multi Linked List serupa dengan proses insert pada Single Linked List, Double Linked List, atau Circular Linked List. Elemen induk baru dapat ditambahkan di awal, setelah elemen tertentu, atau di akhir list induk.

2.1.2 Delete

A. Delete Anak

Untuk menghapus elemen anak, elemen induknya harus ditemukan

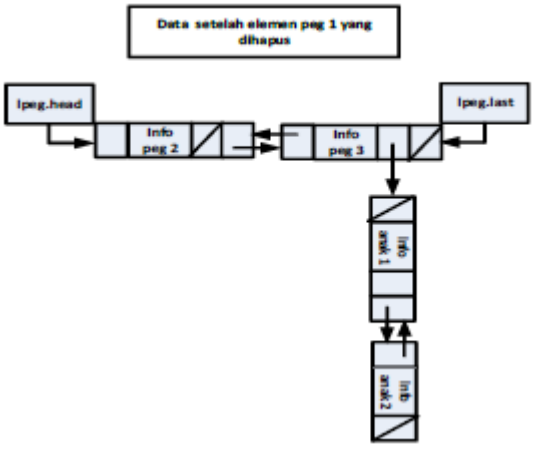
terlebih dahulu. Setelah itu, elemen anak yang ingin dihapus dapat diidentifikasi dan dihapus dari list anak tersebut. Jika elemen anak yang dihapus adalah elemen terakhir, pointer dari elemen sebelumnya perlu diperbarui untuk menandakan akhir baru dari list anak.



Multi Linked List Delete Anak

B. Delete Induk

Penghapusan elemen induk berarti seluruh list anak yang terhubung dengan elemen induk tersebut juga harus dihapus. Saat elemen induk dihapus, semua elemen anak yang terkait dengan induk tersebut juga harus dihapus untuk menjaga konsistensi struktur Multi Linked List.



Multi Linked List Delete Induk

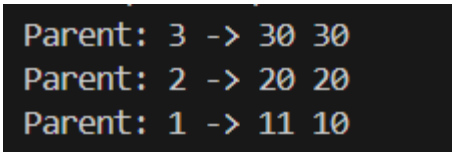
C. Guided

a. Guided1

Code :

```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;
}
```

Output :



Penjelasan : Program ini mengimplementasikan struktur data Multi-Linked List untuk menyimpan data integer. Program ini mendefinisikan struktur Node yang memiliki data integer, pointer ke node berikutnya (next), dan pointer ke node anak (child). Kelas MultiLinkedList memiliki metode untuk menambahkan node induk (addParent), menambahkan node anak ke node induk tertentu (addChild), dan menampilkan seluruh node beserta anak-anaknya (display). Program ini juga memiliki destructor untuk membersihkan memori yang digunakan dengan menghapus semua node dan anak-anaknya ketika objek MultiLinkedList dihancurkan. Pada fungsi main, beberapa node induk dan anak ditambahkan ke dalam list, dan kemudian list tersebut ditampilkan.

b. Guided2

Code :

```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;
}
```

Output :

```
Manager: Charlie -> Brian Frans
Manager: Bob -> Frank
Manager: Alice -> Eve David
```

Penjelasan : Program ini mengimplementasikan struktur data Multi-Linked List untuk mengelola data karyawan. Struktur EmployeeNode menyimpan nama karyawan, pointer ke karyawan berikutnya (next), dan pointer ke subordinate pertama (subordinate). Kelas EmployeeList memiliki fungsi untuk menambahkan karyawan (addEmployee), menambahkan subordinate ke karyawan tertentu (addSubordinate), dan menampilkan daftar karyawan beserta subordinate mereka (display). Program ini juga memiliki destructor untuk membersihkan memori yang digunakan dengan menghapus semua karyawan dan subordinate ketika objek EmployeeList dihancurkan. Pada fungsi main, dilakukan perubahan pada karyawan dan subordinate dengan ditambahkan ke dalam list, yang kemudian list tersebut ditampilkan.

c. Guided3

Code :

```
#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;
}
```

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
```

Penjelasan : Program ini mengimplementasikan struktur data Multi-Linked List untuk mengelola data karyawan. Struktur EmployeeNode menyimpan nama karyawan, pointer ke karyawan berikutnya (next), dan pointer ke subordinate pertama (subordinate). Kelas EmployeeList memiliki metode untuk menambahkan karyawan (addEmployee), menambahkan subordinate ke karyawan tertentu (addSubordinate), dan menampilkan daftar karyawan beserta subordinate nya masing masing (display). Selain itu, program ini juga memiliki metode tambahan untuk menghapus karyawan (deleteEmployee) serta menghapus subordinate dari karyawan tertentu (deleteSubordinate). Program ini juga memiliki destructor untuk membersihkan memori yang digunakan dengan menghapus semua karyawan dan subordinate mereka ketika objek EmployeeList dihancurkan. Pada fungsi yang ada pada main, beberapa karyawan dan subordinate ditambahkan ke dalam list, beberapa dilakukan penghapusan, dan kemudian list menampilkan sebelum dan sesuai perubahan yang dilakukan.

D. Unguided

a. Manajemen Data Pegawai dan Proyek

Code:

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

using namespace std;

struct ProjectNode {
    string projectName;
    int duration; // in months
    ProjectNode* next;

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

struct EmployeeNode {
    string employeeName;
    string employeeID;
    EmployeeNode* next;
    ProjectNode* projectHead;

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

class EmployeeProjectList {
private:
    EmployeeNode* head;

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

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

    void addProject(string employeeID, string projectName, int duration) {
        EmployeeNode* employee = head;
        while (employee != nullptr && employee->employeeID != employeeID) {
            employee = employee->next;
        }
        if (employee != nullptr) {
            ProjectNode* newProject = new ProjectNode(projectName, duration);
            newProject->next = employee->projectHead;
            employee->projectHead = newProject;
        } else {
            cout << "Employee not found!" << endl;
        }
    }

    void deleteProject(string employeeID, string projectName) {
        EmployeeNode* employee = head;
        while (employee != nullptr && employee->employeeID != employeeID) {
            employee = employee->next;
        }
        if (employee != nullptr) {
            ProjectNode** current = &(employee->projectHead);
            while (*current != nullptr && (*current)->projectName != projectName) {
                current = &((*current)->next);
            }
            if (*current != nullptr) {
                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;
        }
    }

    void display() {
        EmployeeNode* current = head;
        while (current != nullptr) {
            cout << "Employee: " << current->employeeName << " (ID: " << current->employeeID << ") -> ";
            ProjectNode* project = current->projectHead;
            while (project != nullptr) {
                cout << "[" << project->projectName << ", " << project->duration << " months] ";
                project = project->next;
            }
            cout << endl;
            current = current->next;
        }
    }

    ~EmployeeProjectList() {
        while (head != nullptr) {
            EmployeeNode* temp = head;
            head = head->next;
            while (temp->projectHead != nullptr) {
                ProjectNode* projectTemp = temp->projectHead;
                temp->projectHead = temp->projectHead->next;
                delete projectTemp;
            }
            delete temp;
        }
    }
};

int main() {
    EmployeeProjectList empList;

    empList.addEmployee("Andi", "P001");
    empList.addEmployee("Budi", "P002");
    empList.addEmployee("Citra", "P003");

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

    cout << "Before adding new project:" << endl;
    empList.display();

    empList.addProject("P001", "Analisis Data", 6);

    cout << "\nAfter adding new project:" << endl;
    empList.display();

    empList.deleteProject("P001", "Aplikasi Mobile");

    cout << "\nAfter deleting a project:" << endl;
    empList.display();

    return 0;
}
```

Output:

```
PS D:\Kuliah\Struktur Data\Github\13_Multi_Linked_List\21104036_Satria Putra Dharma Prayudha\Unguided\output> & .\'DataPegawaidanProyek.exe'
Before adding new project:
Employee: Citra (ID: P003) -> [E-commerce, 10 months]
Employee: Budi (ID: P002) -> [Sistem Akuntansi, 8 months]
Employee: Andi (ID: P001) -> [Aplikasi Mobile, 12 months]

After adding new project:
Employee: Citra (ID: P003) -> [E-commerce, 10 months]
Employee: Budi (ID: P002) -> [Sistem Akuntansi, 8 months]
Employee: Andi (ID: P001) -> [Analisis Data, 6 months] [Aplikasi Mobile, 12 months]
Project Aplikasi Mobile deleted from Andi.

After deleting a project:
Employee: Citra (ID: P003) -> [E-commerce, 10 months]
Employee: Budi (ID: P002) -> [Sistem Akuntansi, 8 months]
Employee: Andi (ID: P001) -> [Analisis Data, 6 months]
PS D:\Kuliah\Struktur Data\Github\13_Multi_Linked_List\21104036_Satria Putra Dharma Prayudha\Unguided\output>
```

Penjelasan: Program ini berisikan pengelolaan data pegawai serta proyek yang dikelola oleh setiap pegawai menggunakan struktur data Multi-Linked List. Setiap pegawai memiliki data berupa nama dan ID, serta dapat memiliki beberapa proyek yang dikelola. Proyek memiliki data berupa nama proyek dan durasi (dalam bulan). Struktur data EmployeeNode digunakan untuk menyimpan informasi pegawai dan pointer ke proyek pertama yang dikelola, sedangkan ProjectNode digunakan untuk menyimpan informasi proyek dan pointer ke proyek berikutnya. Kelas EmployeeProjectList menyediakan fungsi untuk menambahkan pegawai (addEmployee), menambahkan proyek ke pegawai tertentu (addProject), menghapus proyek dari pegawai tertentu (deleteProject), dan menampilkan daftar pegawai beserta proyek mereka (display).

Pada bagian main, program menambahkan beberapa pegawai beserta dengan proyeknya, kemudian program menampilkan daftar pegawai dan proyek mereka masing masing sebelum dan sesudah penambahan proyek baru, dan serta setelah penghapusan proyek tertentu. Pada program ditambahkan pegawai yaitu Andi, Budi, dan Citra, dengan proyek "Aplikasi Mobile", "Sistem Akuntansi", dan "E-commerce". Pada prrogram ini juga ditambahkan proyek baru "Analisis Data" untuk Andi serta menghapus proyek "Aplikasi Mobile" dari Andi. Diakhir program menampilkan secara keseluruhan hasil dari program yang telah dijalankan.

b. Sistem Manajemen Buku Perpustakaan

Code:

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

using namespace std;

struct BookNode {
    string bookTitle;
    string returnDate;
    BookNode* next;

    BookNode(string title, string date) : bookTitle(title), returnDate(date), next(nullptr) {}
};

struct MemberNode {
    string memberName;
    string memberID;
    MemberNode* next;
    BookNode* bookHead;

    MemberNode(string name, string id) : memberName(name), memberID(id), next(nullptr),
bookHead(nullptr) {}
};

class LibraryManagement {
private:
    MemberNode* head;

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

    void addMember(string name, string id) {
        MemberNode* newMember = new MemberNode(name, id);
        newMember->next = head;
        head = newMember;
    }

    void addBook(string memberID, string bookTitle, string returnDate) {
        MemberNode* member = head;
        while (member != nullptr && member->memberID != memberID) {
            member = member->next;
        }
        if (member != nullptr) {
            BookNode* newBook = new BookNode(bookTitle, returnDate);
            newBook->next = member->bookHead;
            member->bookHead = newBook;
        } else {
            cout << "Member not found!" << endl;
        }
    }

    void deleteMember(string memberID) {
        MemberNode** current = &head;
        while (*current != nullptr && (*current)->memberID != memberID) {
            current = &((*current)->next);
        }
        if (*current != nullptr) {
            MemberNode* toDelete = *current;
            *current = (*current)->next;
            while (toDelete->bookHead != nullptr) {
                BookNode* bookTemp = toDelete->bookHead;
                toDelete->bookHead = toDelete->bookHead->next;
                delete bookTemp;
            }
            delete toDelete;
            cout << "Member " << memberID << " deleted." << endl;
        } else {
            cout << "Member not found!" << endl;
        }
    }

    void display() {
        MemberNode* current = head;
        while (current != nullptr) {
            cout << "Member: " << current->memberName << " (ID: " << current->memberID << ") -> ";
            BookNode* book = current->bookHead;
            while (book != nullptr) {
                cout << "[" << book->bookTitle << ", Return: " << book->returnDate << "] ";
                book = book->next;
            }
            cout << endl;
            current = current->next;
        }
    }

    ~LibraryManagement() {
        while (head != nullptr) {
            MemberNode* temp = head;
            head = head->next;
            while (temp->bookHead != nullptr) {
                BookNode* bookTemp = temp->bookHead;
                temp->bookHead = temp->bookHead->next;
                delete bookTemp;
            }
            delete temp;
        }
    }
};

int main() {
    LibraryManagement libMgmt;

    libMgmt.addMember("Rani", "A001");
    libMgmt.addMember("Dito", "A002");
    libMgmt.addMember("Vina", "A003");

    cout << "Before adding any books:" << endl;
    libMgmt.display();

    libMgmt.addBook("A001", "Pemrograman C++", "01/12/2024");
    libMgmt.addBook("A002", "Algoritma Pemrograman", "15/12/2024");

    cout << "\nBefore adding new book:" << endl;
    libMgmt.display();

    libMgmt.addBook("A001", "Struktur Data", "10/12/2024");

    cout << "\nAfter adding new book:" << endl;
    libMgmt.display();

    libMgmt.deleteMember("A002");

    cout << "\nAfter deleting a member:" << endl;
    libMgmt.display();

    return 0;
}
```

Output:

```
Before adding any books:
Member: Vina (ID: A003) ->
Member: Dito (ID: A002) ->
Member: Rani (ID: A001) ->

Before adding new book:
Member: Vina (ID: A003) ->
Member: Dito (ID: A002) -> [Algoritma Pemrograman, Return: 15/12/2024]
Member: Rani (ID: A001) -> [Pemrograman C++, Return: 01/12/2024]

After adding new book:
Member: Vina (ID: A003) ->
Member: Dito (ID: A002) -> [Algoritma Pemrograman, Return: 15/12/2024]
Member: Rani (ID: A001) -> [Struktur Data, Return: 10/12/2024] [Pemrograman C++, Return: 01/12/2024]
Member A002 deleted.

After deleting a member:
Member: Vina (ID: A003) ->
Member: Rani (ID: A001) -> [Struktur Data, Return: 10/12/2024] [Pemrograman C++, Return: 01/12/2024]
PS D:\Kuliah\Struktur Data\Github\13_Multi_Linked_List\21104036_Satria Putra Dharma Prayudha\Unguided\output>
```

Penjelasan:

Program ini berisikan tentang pengelolaan data anggota perpustakaan dan daftar buku yang dipinjam oleh setiap anggota menggunakan struktur data Multi-Linked List. Setiap anggota memiliki data berupa nama dan ID, serta dapat meminjam beberapa buku. Buku memiliki detail berupa judul buku serta tanggal pengembalian. Struktur data MemberNode digunakan untuk menyimpan informasi anggota dan pointer ke buku pertama yang dipinjam, sedangkan BookNode digunakan untuk menyimpan informasi buku dan pointer ke buku berikutnya. Kelas LibraryManagement menyediakan fungsi untuk menambahkan anggota (addMember), menambahkan buku yang dipinjam oleh anggota tertentu (addBook), menghapus anggota beserta buku yang dipinjam (deleteMember), dan menampilkan daftar anggota beserta buku yang dipinjam (display).

Pada bagian main, program menambahkan beberapa anggota dan buku, selanjutnya menampilkan daftar anggota beserta buku yang dipinjam baik sebelum maupun sesudah penambahan buku, serta setelah penghapusan anggota. Pada program dicontohkan, anggota yang ditambahkan adalah Rani, Dito, dan Vina, dengan buku yaitu "Pemrograman C++" dan "Algoritma Pemrograman". Program kemudian menambahkan buku "Struktur Data" untuk Rani, serta menghapus anggota Dito serta buku yang dipinjamnya, kemudian program menampilkan keseluruhan proses yang terjadi.

E. Kesimpulan

Pada praktikum ini, telah dipelajari bagaimana struktur data Multi Linked List digunakan untuk menangani data yang memiliki keterkaitan hierarkis, seperti hubungan antara pegawai dan tugas yang mereka kelola atau anggota perpustakaan dengan buku yang mereka pinjam. Implementasi Multi Linked List memungkinkan kita untuk merepresentasikan struktur yang lebih kompleks dengan menghubungkan node induk dan node anak, serta mengelola hubungan antar node secara efisien. Praktikum ini juga mencakup operasi dasar seperti penambahan dan penghapusan elemen pada level induk maupun anak, yang dapat diaplikasikan pada berbagai studi kasus.

Secara keseluruhan, Multi Linked List terbukti efektif dalam mengelola data yang memiliki hubungan hierarkis atau multi-level. Dengan mengimplementasikan struktur ini, kita dapat mempermudah proses pencarian, penambahan, dan penghapusan data yang saling terhubung dalam sistem. Penerapan pada studi kasus seperti manajemen data pegawai dan proyek, serta sistem perpustakaan, menunjukkan betapa fleksibelnya Multi Linked List dalam berbagai situasi manajemen data yang kompleks.