LAPORAN PRAKTIKUM
PERTEMUAN 13
MULTI LINGKED LIST

Nama :

Haza Zaidan Zidna Fann

(2311104056)

Dosen :

Wahyu Andi Saputra

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. TUJUAN

Memahami penggunaan *Multi Linked list*.
Mengimplementasikan *Multi Linked list* dalam beberapa studi kasus.

II. LANDASAN TEORI

Multi-linked list adalah struktur data yang menghubungkan elemen-elemen dalam bentuk daftar, di mana setiap elemen dapat memiliki lebih dari satu pointer yang mengarah ke elemen lainnya. Setiap node dalam multi-linked list dapat menyimpan referensi ke beberapa node, memungkinkan hubungan yang lebih kompleks antara data. Struktur ini berguna untuk mengorganisir data dengan hubungan yang lebih dinamis dan fleksibel dibandingkan dengan daftar biasa.

III. GUIDED

```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;
}
```

Struktur Data Node:

Menyimpan data induk/anak.

Pointer next menghubungkan elemen dalam daftar induk.

Pointer child menghubungkan elemen anak dari setiap induk.

Kelas MultiLinkedList:

addParent: Menambahkan elemen induk di awal daftar.

addChild: Menambahkan elemen anak ke induk yang sudah ada.

display: Menampilkan semua induk beserta daftar anaknya.

Destruktor: Membersihkan semua memori yang dialokasikan.

Main Function:

Menambahkan tiga induk (1, 2, 3).

Menambahkan beberapa anak ke setiap induk.

Menampilkan hasil berupa daftar induk dan anak.

```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;
}

```

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

Struktur Data EmployeeNode:

Menyimpan nama pegawai (name).

Pointer next menghubungkan pegawai dalam daftar utama.

Pointer subordinate menghubungkan daftar bawahan dari masing-masing pegawai.

Kelas EmployeeList:

addEmployee: Menambahkan pegawai baru di awal daftar.

addSubordinate: Menambahkan bawahan untuk pegawai tertentu (induk).

display: Menampilkan semua pegawai beserta daftar bawahan mereka.

Destruktor: Menghapus semua pegawai dan bawahan untuk membebaskan memori.

Fungsi Utama (main):

Menambahkan tiga pegawai utama: *Alice*, *Bob*, *Charlie*.

Menambahkan beberapa bawahan untuk masing-masing pegawai.

Menampilkan daftar pegawai beserta bawahan mereka.

```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;
}
```

```
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
```

EmployeeNode:

Menyimpan nama karyawan (name), pointer ke karyawan berikutnya (next), dan pointer ke daftar bawahan (subordinate).

Fitur Utama:

addEmployee:

Menambahkan karyawan baru sebagai induk di awal daftar.

addSubordinate:

Menambahkan bawahan ke karyawan tertentu (induk).

deleteEmployee:

Menghapus karyawan dari daftar, beserta seluruh bawahannya.

deleteSubordinate:

Menghapus bawahan dari karyawan tertentu tanpa menghapus karyawan itu sendiri.

display:

Menampilkan semua karyawan dan daftar bawahannya.

Destruktor:

Menghapus seluruh karyawan dan bawahan untuk membebaskan memori.

Main Function:

Menambahkan tiga karyawan: *Alice*, *Bob*, dan *Charlie*.

Menambahkan bawahan ke masing-masing karyawan.

Menampilkan daftar awal.

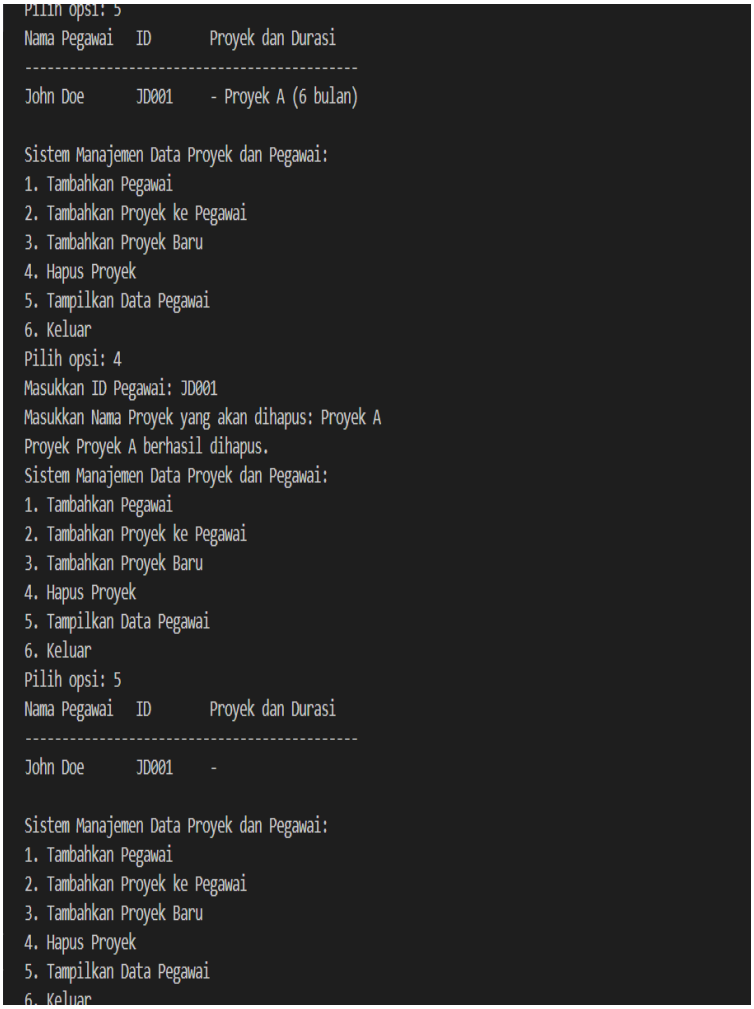
Menghapus bawahan *David* dari *Alice* dan menghapus karyawan *Charlie*.

Menampilkan daftar karyawan yang diperbarui.

IV. UNGUIDED

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

// Struktur untuk Proyek
struct Project {
    string projectName;
    int duration;
    Project* nextProject;
};

// Struktur untuk Pegawai
struct Employee {
    string employeeName;
    string employeeID;
    Project* projectHead;
    Employee* nextEmployee;
};

// Fungsi untuk membuat pegawai baru
Employee* createEmployee(string name, string id) {
    Employee* newEmployee = new Employee;
    newEmployee->employeeName = name;
    newEmployee->employeeID = id;
    newEmployee->projectHead = nullptr;
    newEmployee->nextEmployee = nullptr;
    return newEmployee;
}

// Fungsi untuk membuat proyek baru
Project* createProject(string name, int duration) {
    Project* newProject = new Project;
    newProject->projectName = name;
    newProject->duration = duration;
    newProject->nextProject = nullptr;
    return newProject;
}

// Menambahkan proyek ke pegawai
void addProject(Employee* employee, string projectName, int duration) {
    Project* newProject = createProject(projectName, duration);
    if (employee->projectHead == nullptr) {
        employee->projectHead = newProject;
    } else {
        Project* temp = employee->projectHead;
        while (temp->nextProject != nullptr) {
            temp = temp->nextProject;
        }
        temp->nextProject = newProject;
    }
}

// Menghapus proyek berdasarkan nama
void removeProject(Employee* employee, string projectName) {
    Project* temp = employee->projectHead;
    Project* prev = nullptr;

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

    if (temp == nullptr) {
        cout << "Proyek tidak ditemukan.\n";
        return;
    }

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

    delete temp;
    cout << "Proyek " << projectName << " berhasil dihapus.\n";
}

// Menampilkan data pegawai dan proyek mereka dalam bentuk tabel
void displayEmployees(Employee* head) {
    cout << left << setw(15) << "Nama Pegawai" << setw(10) << "ID" << "Proyek dan Durasi\n";
    cout << "-----------------------------------------------------\n";
    Employee* tempEmployee = head;
    while (tempEmployee != nullptr) {
        cout << left << setw(15) << tempEmployee->employeeName << setw(10) << tempEmployee->employeeID;
        Project* tempProject = tempEmployee->projectHead;
        if (tempProject == nullptr) {
            cout << "-\n";
        } else {
            cout << "- " << tempProject->projectName << " (" << tempProject->duration << " bulan)\n";
            tempProject = tempProject->nextProject;
            while (tempProject != nullptr) {
                cout << setw(25) << "" << "- " << tempProject->projectName << " (" << tempProject->duration << " bulan)\n";
                tempProject = tempProject->nextProject;
            }
        }
        tempEmployee = tempEmployee->nextEmployee;
    }
    cout << endl;
}

// Fungsi untuk menambahkan pegawai ke daftar
void addEmployee(Employee*& head, string name, string id) {
    Employee* newEmployee = createEmployee(name, id);
    if (head == nullptr) {
        head = newEmployee;
    } else {
        Employee* temp = head;
        while (temp->nextEmployee != nullptr) {
            temp = temp->nextEmployee;
        }
        temp->nextEmployee = newEmployee;
    }
}

int main() {
    Employee* head = nullptr;

    int choice;
    string name, id, projectName;
    int duration;

    do {
        cout << "Sistem Manajemen Data Proyek dan Pegawai:\n";
        cout << "1. Tambahkan Pegawai\n";
        cout << "2. Tambahkan Proyek ke Pegawai\n";
        cout << "3. Tambahkan Proyek Baru\n";
        cout << "4. Hapus Proyek\n";
        cout << "5. Tampilkan Data Pegawai\n";
        cout << "6. Keluar\n";
        cout << "Pilih opsi: ";
        cin >> choice;

        switch (choice) {
            case 1:
                int numEmployees;
                cout << "Masukkan jumlah pegawai yang ingin ditambahkan: ";
                cin >> numEmployees;
                for (int i = 0; i < numEmployees; i++) {
                    cout << "Masukkan Nama Pegawai: ";
                    cin.ignore();
                    getline(cin, name);
                    cout << "Masukkan ID Pegawai: ";
                    cin >> id;
                    addEmployee(head, name, id);
                }
                break;

            case 2:
                cout << "Masukkan ID Pegawai: ";
                cin >> id;
                cout << "Masukkan Nama Proyek: ";
                cin.ignore();
                getline(cin, projectName);
                cout << "Masukkan Durasi Proyek (bulan): ";
                cin >> duration;
                {
                    Employee* temp = head;
                    while (temp != nullptr && temp->employeeID != id) {
                        temp = temp->nextEmployee;
                    }
                    if (temp != nullptr) {
                        addProject(temp, projectName, duration);
                    } else {
                        cout << "Pegawai dengan ID " << id << " tidak ditemukan.\n";
                    }
                }
                break;

            case 3:
                cout << "Masukkan Nama Pegawai: ";
                cin.ignore();
                getline(cin, name);
                cout << "Masukkan Nama Proyek: ";
                getline(cin, projectName);
                cout << "Masukkan Durasi Proyek (bulan): ";
                cin >> duration;
                {
                    Employee* temp = head;
                    while (temp != nullptr && temp->employeeName != name) {
                        temp = temp->nextEmployee;
                    }
                    if (temp != nullptr) {
                        addProject(temp, projectName, duration);
                        cout << "Proyek " << projectName << " berhasil ditambahkan ke pegawai dengan nama " << name << ".\n";
                    } else {
                        cout << "Pegawai dengan nama " << name << " tidak ditemukan.\n";
                    }
                }
                break;

            case 4:
                cout << "Masukkan ID Pegawai: ";
                cin >> id;
                cout << "Masukkan Nama Proyek yang akan dihapus: ";
                cin.ignore();
                getline(cin, projectName);
                {
                    Employee* temp = head;
                    while (temp != nullptr && temp->employeeID != id) {
                        temp = temp->nextEmployee;
                    }
                    if (temp != nullptr) {
                        removeProject(temp, projectName);
                    } else {
                        cout << "Pegawai dengan ID " << id << " tidak ditemukan.\n";
                    }
                }
                break;

            case 5:
                displayEmployees(head);
                break;

            case 6:
                cout << "Keluar dari program.\n";
                break;

            default:
                cout << "Pilihan tidak valid.\n";
        }
    } while (choice != 6);

    return 0;
}
```

```
Sistem Manajemen Data Proyek dan Pegawai:
1. Tambahkan Pegawai
2. Tambahkan Proyek ke Pegawai
3. Tambahkan Proyek Baru
4. Hapus Proyek
5. Tampilkan Data Pegawai
6. Keluar
Pilih opsi: 1
Masukkan jumlah pegawai yang ingin ditambahkan: 1
Masukkan Nama Pegawai: John Doe
Masukkan ID Pegawai: JD001
Sistem Manajemen Data Proyek dan Pegawai:
1. Tambahkan Pegawai
2. Tambahkan Proyek ke Pegawai
3. Tambahkan Proyek Baru
4. Hapus Proyek
5. Tampilkan Data Pegawai
6. Keluar
Pilih opsi: 2
Masukkan ID Pegawai: JD001
Masukkan Nama Proyek: Proyek A
Masukkan Durasi Proyek (bulan): 6
Sistem Manajemen Data Proyek dan Pegawai:
1. Tambahkan Pegawai
2. Tambahkan Proyek ke Pegawai
3. Tambahkan Proyek Baru
4. Hapus Proyek
5. Tampilkan Data Pegawai
6. Keluar
Pilih opsi: 5
Nama Pegawai   ID       Proyek dan Durasi
-------------------------------------------
John Doe       JD001     - Proyek A (6 bulan)

Sistem Manajemen Data Proyek dan Pegawai:
1. Tambahkan Pegawai
2. Tambahkan Proyek ke Pegawai
3. Tambahkan Proyek Baru
4. Hapus Proyek
```



Struktur Data:

Project:

Menyimpan informasi proyek, seperti nama dan durasi.

Pointer nextProject menghubungkan proyek berikutnya.

Employee:

Menyimpan informasi pegawai, seperti nama dan ID.

Pointer projectHead menghubungkan daftar proyek yang dikerjakan pegawai.

Pointer nextEmployee menghubungkan pegawai berikutnya dalam daftar.

Fitur Utama:

createEmployee dan createProject:

Membuat node baru untuk pegawai atau proyek.

addEmployee:

Menambahkan pegawai baru ke dalam daftar.

addProject:

Menambahkan proyek ke pegawai tertentu.

removeProject:

Menghapus proyek dari daftar proyek pegawai tertentu berdasarkan nama proyek.

displayEmployees:

Menampilkan daftar pegawai beserta proyek-proyek mereka dalam format tabel.

Fungsi Utama (main):

Menu Interaktif:

Tambahkan Pegawai: Menambah pegawai baru.

Tambahkan Proyek ke Pegawai: Menambah proyek ke pegawai berdasarkan ID.

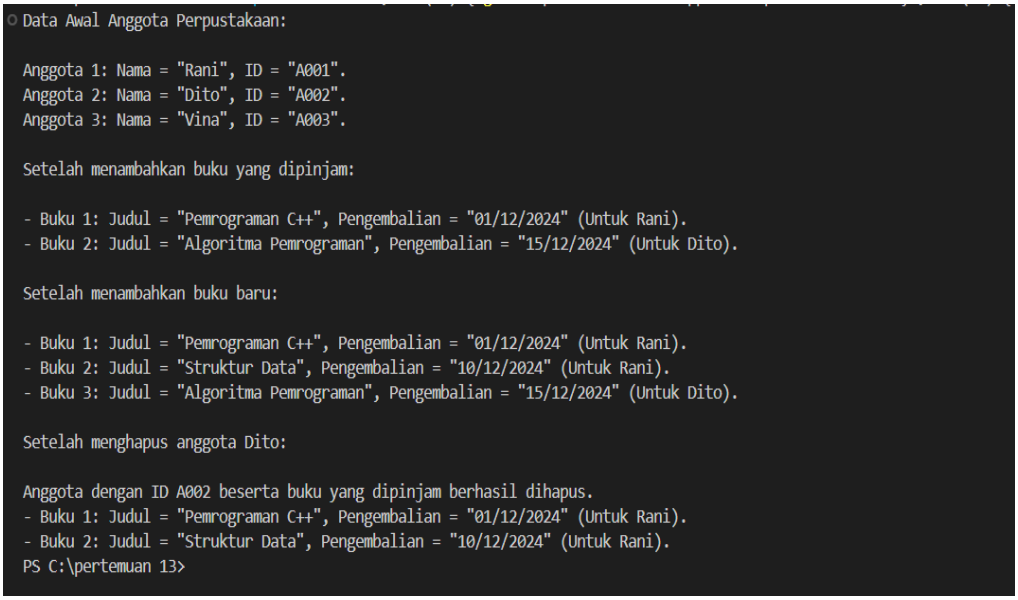
Tambahkan Proyek Baru: Menambah proyek ke pegawai berdasarkan nama.

Hapus Proyek: Menghapus proyek dari pegawai tertentu.

Tampilkan Data Pegawai: Menampilkan seluruh pegawai beserta proyek mereka.

Keluar: Mengakhiri program.

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

// Struktur untuk Buku
struct Book {
    string title;
    string returnDate;
    Book* nextBook;
};

// Struktur untuk Anggota
struct Member {
    string memberName;
    string memberID;
    Book* bookHead;
    Member* nextMember;
};

// Fungsi untuk membuat anggota baru
Member* createMember(string name, string id) {
    Member* newMember = new Member;
    newMember->memberName = name;
    newMember->memberID = id;
    newMember->bookHead = nullptr;
    newMember->nextMember = nullptr;
    return newMember;
}

// Fungsi untuk membuat buku baru
Book* createBook(string title, string returnDate) {
    Book* newBook = new Book;
    newBook->title = title;
    newBook->returnDate = returnDate;
    newBook->nextBook = nullptr;
    return newBook;
}

// Menambahkan buku ke anggota
void addBook(Member* member, string title, string returnDate) {
    Book* newBook = createBook(title, returnDate);
    if (member->bookHead == nullptr) {
        member->bookHead = newBook;
    } else {
        Book* temp = member->bookHead;
        while (temp->nextBook != nullptr) {
            temp = temp->nextBook;
        }
        temp->nextBook = newBook;
    }
}

// Menghapus anggota beserta buku yang dipinjam
void removeMember(Member*& head, string id) {
    Member* temp = head;
    Member* prev = nullptr;

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

    if (temp == nullptr) {
        cout << "Anggota dengan ID " << id << " tidak ditemukan.\n";
        return;
    }

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

    // Hapus semua buku yang dipinjam anggota
    Book* bookTemp = temp->bookHead;
    while (bookTemp != nullptr) {
        Book* toDelete = bookTemp;
        bookTemp = bookTemp->nextBook;
        delete toDelete;
    }

    delete temp;
    cout << "Anggota dengan ID " << id << " beserta buku yang dipinjam berhasil dihapus.\n";
}

// Menampilkan data anggota dan buku yang dipinjam
void displayMembers(Member* head) {
    Member* tempMember = head;
    int bookNumber = 1;
    while (tempMember != nullptr) {
        Book* tempBook = tempMember->bookHead;
        while (tempBook != nullptr) {
            cout << "- Buku " << bookNumber << ": Judul = \"" << tempBook->title << "\", Pengembalian = \"" << tempBook->returnDate << "\" (Untuk " << tempMember->memberName << ").\n";
            tempBook = tempBook->nextBook;
            bookNumber++;
        }
        tempMember = tempMember->nextMember;
    }
}

int main() {
    Member* head = nullptr;

    // Menambahkan data anggota
    head = createMember("Rani", "A001");
    head->nextMember = createMember("Dito", "A002");
    head->nextMember->nextMember = createMember("Vina", "A003");

    cout << "Data Awal Anggota Perpustakaan:\n\n";
    cout << "Anggota 1: Nama = \"Rani\", ID = \"A001\".\n";
    cout << "Anggota 2: Nama = \"Dito\", ID = \"A002\".\n";
    cout << "Anggota 3: Nama = \"Vina\", ID = \"A003\".\n\n";

    cout << "Setelah menambahkan buku yang dipinjam:\n\n";

    // Menambahkan buku yang dipinjam
    addBook(head, "Pemrograman C++", "01/12/2024");
    addBook(head->nextMember, "Algoritma Pemrograman", "15/12/2024");
    displayMembers(head);

    // Menambahkan buku baru untuk Rani
    cout << "\nSetelah menambahkan buku baru:\n\n";
    addBook(head, "Struktur Data", "10/12/2024");
    displayMembers(head);

    // Menghapus anggota Dito beserta buku yang dipinjam
    cout << "\nSetelah menghapus anggota Dito:\n\n";
    removeMember(head, "A002");
    displayMembers(head);

    return 0;
}
```



Struktur Data:

Book: Menyimpan informasi buku seperti judul dan tanggal pengembalian, serta pointer ke buku berikutnya.

Member: Menyimpan informasi anggota (nama, ID) dan pointer ke daftar buku yang dipinjam (linked list dari Book), serta pointer ke anggota berikutnya.

Fungsi Utama:

createMember: Membuat anggota baru dengan nama dan ID.

createBook: Membuat buku baru dengan judul dan tanggal pengembalian.

addBook: Menambahkan buku baru ke daftar buku anggota.

removeMember: Menghapus anggota beserta semua buku yang dipinjam.

displayMembers: Menampilkan semua anggota dan buku yang dipinjam.

Alur Program:

Program dimulai dengan membuat tiga anggota (Rani, Dito, dan Vina).

Buku ditambahkan ke anggota-anggota tersebut.

Buku baru ditambahkan untuk anggota Rani.

Anggota Dito dihapus bersama buku-buku yang dipinjam.

Hasil Output:

Program menampilkan status anggota dan buku yang dipinjam pada setiap tahap: setelah penambahan buku, penambahan buku baru untuk Rani, dan setelah

penghapusan anggota Dito.

V. KESIMPULAN

Multi Linked List adalah struktur data yang menghubungkan berbagai tipe data menggunakan pointer yang saling terkait, memungkinkan pengelolaan data yang lebih kompleks, seperti anggota perpustakaan dan buku yang mereka pinjam. Kelebihannya adalah fleksibilitas dalam menyimpan data, namun lebih memerlukan memori dan pengelolaan yang lebih rumit.