Aturan Praktikum Struktur Data

1. **Akun GitHub:** Setiap praktikan wajib memiliki akun GitHub yang aktif dan digunakan selama praktikum berlangsung.
2. **Invite Collaborator:** Setiap praktikan diwajibkan untuk menambahkan collaborator di setiap repository
    a. Asisten Praktikum: AndiniNH
    b. Asisten Praktikum : 4ldiputra
3. **Repository Praktikum:** Setiap praktikan diwajibkan untuk membuat satu repository di GitHub yang akan digunakan untuk seluruh tugas dan laporan praktikum. Repository ini harus diatur dengan rapi dan sesuai dengan instruksi yang akan diberikan di lampiran.
4. **Penamaan Folder:** Penamaan folder dalam repository akan dibahas secara rinci di lampiran. Praktikan wajib mengikuti aturan penamaan yang telah ditentukan.

Nomor	Pertemuan	Penamaan
1	Penganalan Bahasa C++ Bagian Pertama	01_Pengenalan_CPP_Bagian_1
2	Pengenalan Bahasa C++ Bagian Kedua	02_Pengenalan_CPP_Bagian_2
3	Abstract Data Type	03_Abstract_Data_Type
4	Single Linked List Bagian Pertama	04_Single_Linked_List_Bagian_1
5	Single Linked List Bagian Kedua	05_Single_Linked_List_Bagian_2
6	Double Linked List Bagian Pertama	06_Double_Linked_List_Bagian_1
7	Stack	07_Stack
8	Queue	08_Queue
9	Assessment Bagian Pertama	09_Assessment_Bagian_1
10	Tree Bagian Pertama	10_Tree_Bagian_1
11	Tree Bagian Kedua	11_Tree_Bagian_2
12	Asistensi Tugas Besar	12_Asistensi_Tugas_Besar
13	Multi Linked List	13_Multi_Linked_List
14	Graph	14_Graph
15	Assessment Bagian Kedua	15_Assessment_Bagian_2
16	Tugas Besar	16_Tugas_Besar

5. **Jam Praktikum:**
    - Jam masuk praktikum adalah **1 jam lebih lambat** dari jadwal yang tercantum. Sebagai contoh, jika jadwal praktikum adalah pukul 06.30 - 09.30, maka aturan praktikum akan diatur sebagai berikut:
        - **06.30 - 07.30**: Waktu ini digunakan untuk **Tugas Praktikum dan Laporan Praktikum** yang dilakukan di luar laboratorium.
        - **07.30 - 0G.30**: Sesi ini mencakup **tutorial, diskusi, dan kasus problem-solving**. Kegiatan ini berlangsung di dalam laboratorium dengan alokasi waktu sebagai berikut:
            - **60 menit pertama**: Tugas terbimbing.
            - **60 menit kedua**: Tugas mandiri.
6. **Pengumpulan Tugasn Pendahuluan:** Tugas Pendahuluan (TP) wajib dikumpulkan melalui GitHub sesuai dengan format berikut:

    nama_repo/nama_pertemuan/TP_Pertemuan_Ke.md

    Sebagai contoh:

    STD_Yudha_Islalmi_Sulistya_XXXXXXXX/01_Running_Modul/TP_01.md

7. **Pengecekan Tugas Pendahuluan:** Pengumpulan laporan praktikum akan diperiksa **1 hari sebelum praktikum selanjutnya** dimulai. Pastikan tugas telah diunggah tepat waktu untuk menghindari sanksi.

8. **Struktur Laporan Praktikum**

Cover :

LAPORAN PRAKTIKUM
Modul 13
“MULTI LINKED LIST”

Disusun Oleh:
Kafka Putra Riyadi – 2311104041

Kelas:
SE 07-02

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024

Tujuan

1. Memahami penggunaan *Multi Linked list*.

2. Mengimplementasikan *Multi Linked list* dalam beberapa studi kasus.

Landasan Teori

Multi Linked List (MLL) adalah struktur data yang merupakan pengembangan dari linked list, di mana setiap node memiliki lebih dari satu pointer (link) untuk menunjuk ke beberapa node lainnya. Berbeda dengan single linked list yang hanya memiliki satu koneksi ke node berikutnya atau double linked list yang memiliki dua koneksi (sebelumnya dan berikutnya), MLL memungkinkan adanya banyak koneksi dari satu node. Struktur ini cocok digunakan untuk merepresentasikan hubungan yang kompleks, seperti graf, pohon dengan banyak anak, atau matriks spars, di mana setiap elemen dapat terhubung ke beberapa elemen lainnya secara dinamis.

- Kelebihan Multi Linked List (MLL)

Salah satu kelebihan utama Multi Linked List adalah fleksibilitasnya yang tinggi dalam merepresentasikan struktur data non-linear yang kompleks, seperti graf atau jaringan. Selain itu, MLL efisien dalam penyimpanan data spars, misalnya untuk elemen-elemen non-nol dalam matriks besar. Dengan struktur yang dinamis, MLL memungkinkan alokasi memori sesuai kebutuhan sehingga dapat menghemat ruang dibandingkan dengan struktur statis. MLL juga ideal untuk memodelkan hubungan banyak-ke-banyak antar elemen atau node, seperti dalam database atau sistem jaringan.

- Kekurangan Multi Linked List (MLL)

Meskipun memiliki fleksibilitas tinggi, Multi Linked List memiliki beberapa kekurangan, salah satunya adalah kompleksitas implementasi yang lebih tinggi dibandingkan single atau double linked list. Penggunaan memori pada MLL juga lebih boros karena setiap node membutuhkan beberapa pointer untuk menyimpan referensi ke node lainnya. Selain itu, proses traversal atau penelusuran node menjadi lebih sulit dan memerlukan algoritma khusus seperti Depth First Search (DFS) atau Breadth First Search (BFS), yang dapat memperlambat kinerja jika jumlah node dan koneksinya sangat besar.

Guided

Code Program Guided1

```cpp
#include <iostream>
#include <string>
using namespace std;
struct Node {
    int data;
    Node* next;
    Node* child;

    Node(int val) : data(val), next(nullptr), child(nullptr) {}
};
class MultiLinkedList {
private:
    Node* head;

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

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


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

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

    mList.addChild(1, 10);
    mList.addChild(1, 11);
    mList.addChild(2, 20);
    mList.addChild(2, 20);
    mList.addChild(3, 30);
    mList.addChild(3, 30);
    mList.display();
    return 0;
}
```

Outputannya:

```
Parent: 3 -> 30 30
Parent: 2 -> 20 20
Parent: 1 -> 11 10
PS D:\STRUKTUR DATA P11\output>
```

Code Program Guided2

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

using namespace std;
struct EmployeeNode {
    string name;
    EmployeeNode* next;
    EmployeeNode* subordinate;

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


class EmployeeList {
private:
    EmployeeNode* head;

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


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


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

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

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

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

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

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

    empList.display();
    return 0;
}
```

Outputannya:

```
Manager: Charlie -> Brian Frans
Manager: Bob -> Frank
Manager: Alice -> Eve David
PS D:\STRUKTUR DATA P11\output>
```

Code program Guided3

Pada foto code program guided 3 saya bagi menjadi 2 bagian tidak blur hasilnya pada saat dijadikan pdf

Telkom University Purwokerto

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

using namespace std;
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) {}
};

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;
        }
    }
```

Telkom University Purwokerto

```cpp
    // 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;
}
```

Outputannya:

```
Initial employee list:
Manager: Charlie ->
Manager: Bob -> Frank
Manager: Alice -> Eve David
Subordinate David deleted from Alice.
Employee Charlie deleted.

Updated employee list:
Manager: Bob -> Frank
Manager: Alice -> Eve
PS D:\STRUKTUR DATA P11\output>
```

Unguided

Code Unguided1

Pada gambar codingan unguided1, saya bagi menjadi 2 gambar agar tidak blur pada saat dijadikan pdf

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

struct Proyek {
    string namaProyek;
    int durasi;
    Proyek* nextProyek;
};

struct Pegawai {
    string namaPegawai;
    string idPegawai;
    Proyek* headProyek;
    Pegawai* nextPegawai;
};

Pegawai* headPegawai = nullptr;

void tambahPegawai(string nama, string id) {
    Pegawai* pegawaiBaru = new Pegawai;
    pegawaiBaru->namaPegawai = nama;
    pegawaiBaru->idPegawai = id;
    pegawaiBaru->headProyek = nullptr;
    pegawaiBaru->nextPegawai = headPegawai;
    headPegawai = pegawaiBaru;
}

void tambahProyek(string idPegawai, string namaProyek, int durasi) {
    Pegawai* currPegawai = headPegawai;
    while (currPegawai != nullptr) {
        if (currPegawai->idPegawai == idPegawai) {
            Proyek* proyekBaru = new Proyek;
            proyekBaru->namaProyek = namaProyek;
            proyekBaru->durasi = durasi;
            proyekBaru->nextProyek = currPegawai->headProyek;
            currPegawai->headProyek = proyekBaru;
            return;
        }
        currPegawai = currPegawai->nextPegawai;
    }
    cout << "Pegawai dengan ID " << idPegawai << " tidak ditemukan!\n";
}

void hapusProyek(string idPegawai, string namaProyek) {
    Pegawai* currPegawai = headPegawai;
    while (currPegawai != nullptr) {
        if (currPegawai->idPegawai == idPegawai) {
            Proyek* currProyek = currPegawai->headProyek;
            Proyek* prevProyek = nullptr;
            while (currProyek != nullptr) {
                if (currProyek->namaProyek == namaProyek) {
                    if (prevProyek == nullptr) {
                        currPegawai->headProyek = currProyek->nextProyek;
                    } else {
                        prevProyek->nextProyek = currProyek->nextProyek;
                    }
                    delete currProyek;
                    return;
                }
                prevProyek = currProyek;
                currProyek = currProyek->nextProyek;
            }
        }
        currPegawai = currPegawai->nextPegawai;
    }
    cout << "Proyek " << namaProyek << " tidak ditemukan untuk pegawai dengan ID " << idPegawai << "!\n";
}
```

```cpp
void tampilkanData() {
    Pegawai* currPegawai = headPegawai;
    while (currPegawai != nullptr) {
        cout << "Pegawai: " << currPegawai->namaPegawai << " (ID: " << currPegawai->idPegawai << ")\n";
        Proyek* currProyek = currPegawai->headProyek;
        while (currProyek != nullptr) {
            cout << "  Proyek: " << currProyek->namaProyek << " (Durasi: " << currProyek->durasi << " bulan)\n";
            currProyek = currProyek->nextProyek;
        }
        currPegawai = currPegawai->nextPegawai;
    }
}

int main() {
    int pilihan;
    do {
        cout << "\nMenu:\n";
        cout << "1. Tambah Pegawai\n";
        cout << "2. Tambah Proyek\n";
        cout << "3. Hapus Proyek\n";
        cout << "4. Tampilkan Data\n";
        cout << "5. Keluar\n";
        cout << "Pilihan: ";
        cin >> pilihan;
        cin.ignore();

        if (pilihan == 1) {
            string nama, id;
            cout << "Masukkan Nama Pegawai: ";
            getline(cin, nama);
            cout << "Masukkan ID Pegawai: ";
            getline(cin, id);
            tambahPegawai(nama, id);
        } else if (pilihan == 2) {
            string id, namaProyek;
            int durasi;
            cout << "Masukkan ID Pegawai: ";
            getline(cin, id);
            cout << "Masukkan Nama Proyek: ";
            getline(cin, namaProyek);
            cout << "Masukkan Durasi Proyek (bulan): ";
            cin >> durasi;
            cin.ignore();
            tambahProyek(id, namaProyek, durasi);
        } else if (pilihan == 3) {
            string id, namaProyek;
            cout << "Masukkan ID Pegawai: ";
            getline(cin, id);
            cout << "Masukkan Nama Proyek yang akan dihapus: ";
            getline(cin, namaProyek);
            hapusProyek(id, namaProyek);
        } else if (pilihan == 4) {
            tampilkanData();
        } else if (pilihan != 5) {
            cout << "Pilihan tidak valid.\n";
        }
    } while (pilihan != 5);

    return 0;
}
```

Outputannya:

```
Menu:
1. Tambah Pegawai
2. Tambah Proyek
3. Hapus Proyek
4. Tampilkan Data
5. Keluar
Pilihan: 1
Masukkan Nama Pegawai: Andi
Masukkan ID Pegawai: P001
```

```
Menu:
1. Tambah Pegawai
2. Tambah Proyek
3. Hapus Proyek
4. Tampilkan Data
5. Keluar
Pilihan: 1
Masukkan Nama Pegawai: Budi
Masukkan ID Pegawai: P002
```

```
Menu:
1. Tambah Pegawai
2. Tambah Proyek
3. Hapus Proyek
4. Tampilkan Data
5. Keluar
Pilihan: 1
Masukkan Nama Pegawai: Citra
Masukkan ID Pegawai: P003
```

```
Menu:
1. Tambah Pegawai
2. Tambah Proyek
3. Hapus Proyek
4. Tampilkan Data
5. Keluar
Pilihan: 2
Masukkan ID Pegawai: P001
Masukkan Nama Proyek: Aplikasi Mobile
Masukkan Durasi Proyek (bulan): 12
```

```
Menu:
1. Tambah Pegawai
2. Tambah Proyek
3. Hapus Proyek
4. Tampilkan Data
5. Keluar
Pilihan: 2
Masukkan ID Pegawai: P002
Masukkan Nama Proyek: Sistem Akuntansi
Masukkan Durasi Proyek (bulan): 8
```

```
Menu:
1. Tambah Pegawai
2. Tambah Proyek
3. Hapus Proyek
4. Tampilkan Data
5. Keluar
Pilihan: 2
Masukkan ID Pegawai: P003
Masukkan Nama Proyek: E-Commerce
Masukkan Durasi Proyek (bulan): 10
```

```
Menu:
1. Tambah Pegawai
2. Tambah Proyek
3. Hapus Proyek
4. Tampilkan Data
5. Keluar
Pilihan: 2
Masukkan ID Pegawai: P001
Masukkan Nama Proyek: Analisis Data
Masukkan Durasi Proyek (bulan): 6
```

```
Menu:
1. Tambah Pegawai
2. Tambah Proyek
3. Hapus Proyek
4. Tampilkan Data
5. Keluar
Pilihan: 3
Masukkan ID Pegawai: P001
Masukkan Nama Proyek yang akan dihapus: Aplikasi Mobile
```

```
Menu:
1. Tambah Pegawai
2. Tambah Proyek
3. Hapus Proyek
4. Tampilkan Data
5. Keluar
Pilihan: 4
Pegawai: Citra (ID: P003)
  Proyek: E-Commerce (Durasi: 10 bulan)
Pegawai: Budi (ID: P002)
  Proyek: Sistem Akuntansi (Durasi: 8 bulan)
Pegawai: Andi  (ID: P001)
  Proyek: Analisis Data (Durasi: 6 bulan)
```

Unguided2

Gambar codingan unguided2 saya bagi juga menjadi 2 bagian

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

struct Buku {
    string judulBuku;
    string tanggalPengembalian;
    Buku* nextBuku;
};

struct Anggota {
    string namaAnggota;
    string idAnggota;
    Buku* headBuku;
    Anggota* nextAnggota;
};

Anggota* headAnggota = nullptr;

void tambahAnggota(string nama, string id) {
    Anggota* anggotaBaru = new Anggota;
    anggotaBaru->namaAnggota = nama;
    anggotaBaru->idAnggota = id;
    anggotaBaru->headBuku = nullptr;
    anggotaBaru->nextAnggota = headAnggota;
    headAnggota = anggotaBaru;
}

void tambahBuku(string idAnggota, string judulBuku, string tanggalPengembalian) {
    Anggota* currAnggota = headAnggota;
    while (currAnggota != nullptr) {
        if (currAnggota->idAnggota == idAnggota) {
            Buku* bukuBaru = new Buku;
            bukuBaru->judulBuku = judulBuku;
            bukuBaru->tanggalPengembalian = tanggalPengembalian;
            bukuBaru->nextBuku = currAnggota->headBuku;
            currAnggota->headBuku = bukuBaru;
            return;
        }
        currAnggota = currAnggota->nextAnggota;
    }
    cout << "Anggota dengan ID " << idAnggota << " tidak ditemukan!\n";
}

void hapusAnggota(string idAnggota) {
    Anggota* currAnggota = headAnggota;
    Anggota* prevAnggota = nullptr;
    while (currAnggota != nullptr) {
        if (currAnggota->idAnggota == idAnggota) {
            if (prevAnggota == nullptr) {
                headAnggota = currAnggota->nextAnggota;
            } else {
                prevAnggota->nextAnggota = currAnggota->nextAnggota;
            }
            // Hapus semua buku yang dipinjam anggota
            Buku* currBuku = currAnggota->headBuku;
            while (currBuku != nullptr) {
                Buku* temp = currBuku;
                currBuku = currBuku->nextBuku;
                delete temp;
            }
            delete currAnggota;
            return;
        }
        prevAnggota = currAnggota;
        currAnggota = currAnggota->nextAnggota;
    }
    cout << "Anggota dengan ID " << idAnggota << " tidak ditemukan!\n";
}
```

```cpp
void tampilkanData() {
    Anggota* currAnggota = headAnggota;
    while (currAnggota != nullptr) {
        cout << "Anggota: " << currAnggota->namaAnggota << " (ID: " << currAnggota->idAnggota << ")\n";
        Buku* currBuku = currAnggota->headBuku;
        while (currBuku != nullptr) {
            cout << "  Buku: " << currBuku->judulBuku << " (Pengembalian: " << currBuku->tanggalPengembalian << ")\n";
            currBuku = currBuku->nextBuku;
        }
        currAnggota = currAnggota->nextAnggota;
    }
}

int main() {
    int pilihan;
    do {
        cout << "\nMenu:\n";
        cout << "1. Tambah Anggota\n";
        cout << "2. Tambah Buku\n";
        cout << "3. Hapus Anggota\n";
        cout << "4. Tampilkan Data\n";
        cout << "5. Keluar\n";
        cout << "Pilihan: ";
        cin >> pilihan;
        cin.ignore();

        if (pilihan == 1) {
            string nama, id;
            cout << "Masukkan Nama Anggota: ";
            getline(cin, nama);
            cout << "Masukkan ID Anggota: ";
            getline(cin, id);
            tambahAnggota(nama, id);
        } else if (pilihan == 2) {
            string id, judulBuku, tanggalPengembalian;
            cout << "Masukkan ID Anggota: ";
            getline(cin, id);
            cout << "Masukkan Judul Buku: ";
            getline(cin, judulBuku);
            cout << "Masukkan Tanggal Pengembalian: ";
            getline(cin, tanggalPengembalian);
            tambahBuku(id, judulBuku, tanggalPengembalian);
        } else if (pilihan == 3) {
            string id;
            cout << "Masukkan ID Anggota yang akan dihapus: ";
            getline(cin, id);
            hapusAnggota(id);
        } else if (pilihan == 4) {
            tampilkanData();
        } else if (pilihan != 5) {
            cout << "Pilihan tidak valid.\n";
        }
    } while (pilihan != 5);

    return 0;
}
```

Outputannya:

```
Menu:
1. Tambah Anggota
2. Tambah Buku
3. Hapus Anggota
4. Tampilkan Data
5. Keluar
Pilihan: 1
Masukkan Nama Anggota: Rani
Masukkan ID Anggota: A001

Menu:
1. Tambah Anggota
2. Tambah Buku
3. Hapus Anggota
4. Tampilkan Data
5. Keluar
Pilihan: 1
Masukkan Nama Anggota: Dito
Masukkan ID Anggota: A002

Menu:
1. Tambah Anggota
2. Tambah Buku
3. Hapus Anggota
4. Tampilkan Data
5. Keluar
Pilihan: 1
Masukkan Nama Anggota: Vina
Masukkan ID Anggota: A003
```

```
Menu:
1. Tambah Anggota
2. Tambah Buku
3. Hapus Anggota
4. Tampilkan Data
5. Keluar
Pilihan: 2
Masukkan ID Anggota: A001
Masukkan Judul Buku: Pemrograman C++
Masukkan Tanggal Pengembalian: 01/12/2024

Menu:
1. Tambah Anggota
2. Tambah Buku
3. Hapus Anggota
4. Tampilkan Data
5. Keluar
Pilihan: 2
Masukkan ID Anggota: A002
Masukkan Judul Buku: Algoritma Pemrograman
Masukkan Tanggal Pengembalian: 15/12/2024
```

```
Menu:
1. Tambah Anggota
2. Tambah Buku
3. Hapus Anggota
4. Tampilkan Data
5. Keluar
Pilihan: 3
Masukkan ID Anggota yang akan dihapus: A002
```

```
Menu:
1. Tambah Anggota
2. Tambah Buku
3. Hapus Anggota
4. Tampilkan Data
5. Keluar
Pilihan: 2
Masukkan ID Anggota: A001
Masukkan Judul Buku: Struktur Data
Masukkan Tanggal Pengembalian: 10/12/2024
```

```
Menu:
1. Tambah Anggota
2. Tambah Buku
3. Hapus Anggota
4. Tampilkan Data
5. Keluar
Pilihan: 4
Anggota: Vina (ID: A003)
Anggota: Rani (ID: A001)
  Buku: Struktur Data (Pengembalian: 10/12/2024)
  Buku: Pemrograman C++ (Pengembalian: 01/12/2024)
```

Kesimpulan

Multi Linked List (MLL) adalah pengembangan dari linked list yang memungkinkan setiap node memiliki lebih dari satu pointer untuk menunjuk ke beberapa node lainnya, sehingga cocok untuk merepresentasikan struktur data kompleks seperti graf, pohon dengan banyak anak, atau matriks spars. Kelebihan MLL terletak pada fleksibilitasnya dalam memodelkan hubungan non-linear, efisiensi dalam menyimpan data spars, dan alokasi memori yang dinamis sesuai kebutuhan. Namun, MLL juga memiliki kekurangan, seperti implementasinya yang lebih rumit, penggunaan memori yang lebih boros, dan proses traversal yang memerlukan algoritma khusus seperti DFS atau BFS, yang dapat memperlambat kinerja pada struktur dengan banyak node dan koneksi.

Telkom University Purwokerto