

LAPORAN PRAKTIKUM

Modul 13

“MULTI LINKED LIST”



Disusun Oleh:

Rengganis Tantri Pramudita - 2311104065

SE0702

Dosen :

Wahyu Andi Saputra

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

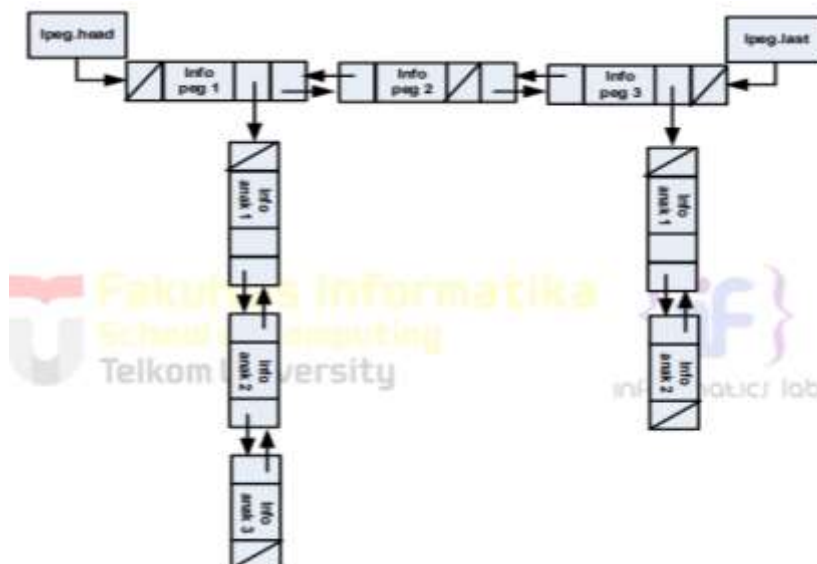
1. Tujuan

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

2. Landasan Teori

Multi Linked List adalah salah satu struktur data yang mengembangkan konsep linked list dasar, di mana setiap node dapat memiliki lebih dari satu pointer yang mengarah ke node lain, sehingga memungkinkan representasi hubungan yang lebih kompleks. Konsep ini berguna untuk memodelkan struktur data yang memerlukan hubungan ganda atau lebih, seperti graf atau tabel multidimensi. Dalam implementasinya, multi linked list sering digunakan untuk mempermudah pengelolaan data yang bersifat hierarkis atau relasional. Landasan teori dari multi linked list mencakup prinsip-prinsip dasar linked list seperti penggunaan pointer untuk menghubungkan node, efisiensi dalam traversal data, serta kemampuan untuk mengelola memori secara dinamis. Selain itu, multi linked list menawarkan fleksibilitas dalam akses data, memungkinkan operasi yang lebih kompleks dibandingkan single linked list, seperti pencarian, penambahan, dan penghapusan elemen dalam konteks relasional. Konsep ini juga banyak diterapkan pada aplikasi seperti basis data, navigasi hierarki file, dan representasi graf dalam ilmu komputer.

Contoh Multi Linked List



3. Guided

Guided 1

```
#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;
}

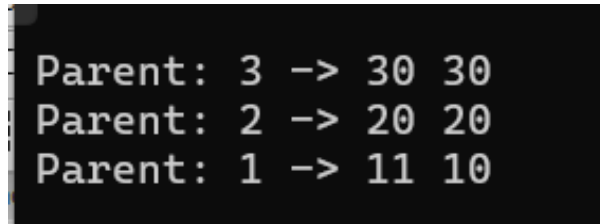
```

Penjelasan:

Kode program tersebut mengimplementasikan konsep **Multi Linked List** dalam bahasa pemrograman C++ untuk mengorganisir data yang memiliki hubungan hierarki antara **parent** dan **child**. Struktur Node digunakan untuk merepresentasikan setiap elemen, di mana setiap node memiliki atribut data (data yang disimpan), pointer next (untuk menunjuk ke node berikutnya di level yang sama), dan pointer child (untuk menunjuk ke sub-list atau anak node tersebut). Kelas MultiLinkedList menyediakan metode utama seperti addParent untuk menambahkan node parent baru di awal daftar, addChild untuk menambahkan node anak ke parent tertentu, dan display untuk menampilkan seluruh data parent beserta data anak-anaknya dalam format yang terstruktur. Selain itu, terdapat destructor untuk memastikan semua node, baik parent maupun child, dihapus dari memori sehingga tidak terjadi **memory leak**. Pada fungsi main, beberapa node parent dengan nilai tertentu

ditambahkan, kemudian anak-anak ditambahkan ke masing-masing parent. Akhirnya, fungsi display dipanggil untuk mencetak struktur parent dan child ke layar dalam bentuk hierarki, di mana setiap parent diikuti oleh daftar anak-anaknya.

Outputnya



```
Parent: 3 -> 30 30
Parent: 2 -> 20 20
Parent: 1 -> 11 10
```

Guided 2

```
#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;
}

```

Penjelasan

Kode program ini mengimplementasikan **Multi Linked List** untuk merepresentasikan hierarki karyawan dalam sebuah organisasi, di mana setiap karyawan dapat memiliki bawahan atau anak. Struktur `EmployeeNode` digunakan untuk menyimpan data nama karyawan dengan pointer `next` yang menghubungkan node karyawan satu dengan yang lain di tingkat yang sama, serta pointer `subordinate` untuk menunjuk ke daftar bawahan karyawan tersebut. Kelas `EmployeeList` menyediakan metode utama seperti `addEmployee` untuk menambahkan karyawan ke daftar utama, `addSubordinate` untuk menambahkan bawahan pada karyawan tertentu dengan mencarinya berdasarkan nama, dan `display` untuk menampilkan semua karyawan beserta bawahan mereka. Pada fungsi `main`, beberapa karyawan ditambahkan, diikuti dengan penambahan bawahan ke karyawan tertentu, seperti bawahan "David" dan "Eve" untuk "Alice" serta "Frank" untuk "Bob". Program kemudian mencetak struktur hierarki karyawan beserta bawahan mereka. Destructor memastikan semua node, termasuk node bawahan, dihapus dari memori untuk mencegah **memory leak**.

Outputnya

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

Guided 3

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

using namespace std;

// Struktur untuk node karyawan
struct EmployeeNode {
    string name; // Nama karyawan
    EmployeeNode* next; // Pointer ke karyawan berikutnya
    EmployeeNode* subordinate; // Pointer ke subordinate pertama

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

// Kelas untuk Multi-Linked List Karyawan
class EmployeeList {
private:
    EmployeeNode* head; // Pointer ke kepala list

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

```

// Menambahkan karyawan (induk)
void addEmployee(string name) {
    EmployeeNode* newEmployee = new EmployeeNode(name);
    newEmployee->next = head; // Menyambungkan ke karyawan sebelumnya
    head = newEmployee; // Memperbarui head
}

// Menambahkan subordinate ke karyawan tertentu
void addSubordinate(string managerName, string subordinateName) {
    EmployeeNode* manager = head;
    while (manager != nullptr && manager->name != managerName) {
        manager = manager->next;
    }
    if (manager != nullptr) { // Jika manajer ditemukan
        EmployeeNode* newSubordinate = new EmployeeNode(subordinateName);
        newSubordinate->next = manager->subordinate; // Menyambungkan ke subordinate sebelumnya
        manager->subordinate = newSubordinate; // Memperbarui subordinate
    } else {
        cout << "Manager not found!" << endl;
    }
}

// Menghapus karyawan (induk)
void deleteEmployee(string name) {
    EmployeeNode** current = &head;
    while (*current != nullptr && (*current)->name != name) {
        current = &((*current)->next);
    }

    if (*current != nullptr) { // Jika karyawan ditemukan
        EmployeeNode* toDelete = *current;
        *current = (*current)->next;

        // Hapus semua subordinate dari node ini
        while (toDelete->subordinate != nullptr) {
            EmployeeNode* subTemp = toDelete->subordinate;
            toDelete->subordinate = toDelete->subordinate->next;
            delete subTemp;
        }
        delete toDelete;
        cout << "Employee " << name << " deleted." << endl;
    } else {
        cout << "Employee not found!" << endl;
    }
}

// Menghapus subordinate dari karyawan tertentu
void deleteSubordinate(string managerName, string subordinateName) {
    EmployeeNode* manager = head;
    while (manager != nullptr && manager->name != managerName) {
        manager = manager->next;
    }
}

```



```

    }

    if (manager != nullptr) { // Jika manajer ditemukan
        EmployeeNode** currentSub = &(manager->subordinate);
        while (*currentSub != nullptr && (*currentSub)->name != subordinateName) {
            currentSub = &((*currentSub)->next);
        }

        if (*currentSub != nullptr) { // Jika subordinate ditemukan
            EmployeeNode* toDelete = *currentSub;
            *currentSub = (*currentSub)->next; // Menghapus dari list

            delete toDelete; // Menghapus node subordinate
            cout << "Subordinate " << subordinateName << " deleted from " << managerName << "." <<
endl;
        } else {
            cout << "Subordinate not found!" << endl;
        }
    } else {
        cout << "Manager not found!" << endl;
    }
}

// Menampilkan daftar karyawan dan subordinate mereka
void display() {
    EmployeeNode* current = head;
    while (current != nullptr) {
        cout << "Manager: " << current->name << " -> ";
        EmployeeNode* sub = current->subordinate;
        while (sub != nullptr) {
            cout << sub->name << " ";
            sub = sub->next;
        }
        cout << endl;
        current = current->next;
    }
}

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

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

```

```

    }
};

int main() {
    EmployeeList empList;

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

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

    cout << "Initial employee list:" << endl;
    empList.display(); // Menampilkan isi daftar karyawan

    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;
}

```

Penjelasan

Kode di atas mengimplementasikan struktur Multi-Linked List untuk merepresentasikan hierarki karyawan dan bawahan dalam organisasi. Struktur `EmployeeNode` digunakan untuk menyimpan data nama karyawan dengan dua pointer: `next` untuk menunjuk ke karyawan lain di tingkat yang sama, dan `subordinate` untuk menunjuk ke daftar bawahan dari karyawan tersebut. Kelas `EmployeeList` menyediakan fungsi-fungsi utama seperti `addEmployee` untuk menambahkan karyawan sebagai kepala daftar, `addSubordinate` untuk menambahkan bawahan ke karyawan tertentu, `deleteEmployee` untuk menghapus karyawan beserta semua bawahannya, serta `deleteSubordinate` untuk menghapus satu bawahan dari karyawan tertentu. Metode `display` digunakan untuk menampilkan daftar karyawan dan bawahan mereka dalam format yang mudah dibaca. Pada fungsi `main`, program menambahkan beberapa karyawan dan bawahan, lalu menghapus salah satu bawahan dari "Alice" serta menghapus "Charlie" sepenuhnya. Terakhir, program menampilkan kondisi daftar karyawan sebelum dan sesudah penghapusan. Destructor memastikan memori dibersihkan secara otomatis untuk mencegah memory leak.

Outputnya

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

4. Unguided

Unguided 1

```
#include <iostream>  
#include <string>  
using namespace std;  
  
struct Proyek {  
    string namaProyek;  
    int durasi;  
    Proyek* proyekBerikutnya;  
  
    Proyek(string nama, int dur) : namaProyek(nama), durasi(dur), proyekBerikutnya(nullptr) {}  
};  
  
struct Pegawai {  
    string namaPegawai;  
    string idPegawai;  
    Proyek* kepalaProyek;  
    Pegawai* pegawaiBerikutnya;  
  
    Pegawai(string nama, string id) : namaPegawai(nama), idPegawai(id), kepalaProyek(nullptr),  
pegawaiBerikutnya(nullptr) {}  
};  
  
class MultiLinkedList {  
private:  
    Pegawai* kepalaPegawai;  
  
public:  
    MultiLinkedList() : kepalaPegawai(nullptr) {}  
};
```

```

void tambahPegawai(string nama, string id) {
    Pegawai* pegawaiBaru = new Pegawai(nama, id);
    if (!kepalaPegawai) {
        kepalaPegawai = pegawaiBaru;
    } else {
        Pegawai* temp = kepalaPegawai;
        while (temp->pegawaiBerikutnya) temp = temp->pegawaiBerikutnya;
        temp->pegawaiBerikutnya = pegawaiBaru;
    }
}

```

```

void tambahProyek(string id, string namaProyek, int durasi) {
    Pegawai* pegawai = cariPegawai(id);
    if (pegawai) {
        Proyek* proyekBaru = new Proyek(namaProyek, durasi);
        if (!pegawai->kepalaProyek) {
            pegawai->kepalaProyek = proyekBaru;
        } else {
            Proyek* temp = pegawai->kepalaProyek;
            while (temp->proyekBerikutnya) temp = temp->proyekBerikutnya;
            temp->proyekBerikutnya = proyekBaru;
        }
    }
}

```

```

void hapusProyek(string id, string namaProyek) {
    Pegawai* pegawai = cariPegawai(id);
    if (pegawai && pegawai->kepalaProyek) {
        Proyek* temp = pegawai->kepalaProyek;
        Proyek* prev = nullptr;

        while (temp) {
            if (temp->namaProyek == namaProyek) {
                if (prev) {
                    prev->proyekBerikutnya = temp->proyekBerikutnya;
                } else {
                    pegawai->kepalaProyek = temp->proyekBerikutnya;
                }
                delete temp;
                return;
            }
            prev = temp;
            temp = temp->proyekBerikutnya;
        }
    }
}

```

```

    }
    prev = temp;
    temp = temp->proyekBerikutnya;
}
}

void tampilkanData() {
    Pegawai* temp = kepalaPegawai;
    while (temp) {
        cout << "Pegawai: " << temp->namaPegawai << " (ID: " << temp->idPegawai << ")\n";
        Proyek* proyekTemp = temp->kepalaProyek;
        while (proyekTemp) {
            cout << " -> Proyek: " << proyekTemp->namaProyek << ", Durasi: " << proyekTemp->durasi
<< " bulan\n";
            proyekTemp = proyekTemp->proyekBerikutnya;
        }
        temp = temp->pegawaiBerikutnya;
        cout << endl;
    }
}

private:
    Pegawai* cariPegawai(string id) {
        Pegawai* temp = kepalaPegawai;
        while (temp) {
            if (temp->idPegawai == id) return temp;
            temp = temp->pegawaiBerikutnya;
        }
        return nullptr;
    }
};

int main() {
    MultiLinkedList daftar;

    daftar.tambahPegawai("Andi", "P001");
    daftar.tambahPegawai("Budi", "P002");
    daftar.tambahPegawai("Citra", "P003");
}

```

```
daftar.tambahProyek("P001", "Aplikasi Mobile", 12);
daftar.tambahProyek("P002", "Sistem Akuntansi", 8);
daftar.tambahProyek("P003", "E-commerce", 10);

daftar.tambahProyek("P001", "Analisis Data", 6);

daftar.hapusProyek("P001", "Aplikasi Mobile");

cout << "Data Pegawai dan Proyek:\n";
daftar.tampilkanData();

return 0;
}
```

Penjelasan:

- Struktur Data:
 - Proyek: Menyimpan data nama proyek dan durasi proyek.
 - Pegawai: Menyimpan data pegawai, pointer ke proyek mereka, dan pointer ke pegawai berikutnya dalam daftar.
- Kelas MultiLinkedList:
 - tambahPegawai: Menambahkan pegawai baru ke dalam linked list.
 - tambahProyek: Menambahkan proyek ke pegawai tertentu berdasarkan ID.
 - hapusProyek: Menghapus proyek berdasarkan nama dari pegawai tertentu.
 - tampilkanData: Menampilkan data semua pegawai beserta proyek mereka.
- Fungsi main:
 - Menambahkan data pegawai dan proyek sesuai instruksi.
 - Menambahkan proyek baru.
 - Menghapus proyek "Aplikasi Mobile" dari Andi.
 - Menampilkan data pegawai dan proyek mereka.

Output

```
Data Pegawai dan Proyek:
Pegawai: Andi (ID: P001)
-> Proyek: Analisis Data, Durasi: 6 bulan

Pegawai: Budi (ID: P002)
-> Proyek: Sistem Akuntansi, Durasi: 8 bulan

Pegawai: Citra (ID: P003)
-> Proyek: E-commerce, Durasi: 10 bulan

Process returned 0 (0x0)   execution time : 0.353 s
Press any key to continue.
```

Unguided 2

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

struct Buku {
    string judulBuku;
    string tanggalPengembalian;
    Buku* bukuBerikutnya;

    Buku(string judul, string tanggal) : judulBuku(judul), tanggalPengembalian(tanggal),
    bukuBerikutnya(nullptr) {}
};

struct Anggota {
    string namaAnggota;
    string idAnggota;
    Buku* kepalaBuku;
    Anggota* anggotaBerikutnya;

    Anggota(string nama, string id) : namaAnggota(nama), idAnggota(id), kepalaBuku(nullptr),
    anggotaBerikutnya(nullptr) {}
};

class MultiLinkedList {
private:
    Anggota* kepalaAnggota;

public:
    MultiLinkedList() : kepalaAnggota(nullptr) {}

    void tambahAnggota(string nama, string id) {
        Anggota* anggotaBaru = new Anggota(nama, id);
        if (!kepalaAnggota) {
            kepalaAnggota = anggotaBaru;
        } else {
            Anggota* temp = kepalaAnggota;
            while (temp->anggotaBerikutnya) temp = temp->anggotaBerikutnya;
```

```

        temp->anggotaBerikutnya = anggotaBaru;
    }
}

void tambahBuku(string id, string judulBuku, string tanggalPengembalian) {
    Anggota* anggota = cariAnggota(id);
    if (anggota) {
        Buku* bukuBaru = new Buku(judulBuku, tanggalPengembalian);
        if (!anggota->kepalaBuku) {
            anggota->kepalaBuku = bukuBaru;
        } else {
            Buku* temp = anggota->kepalaBuku;
            while (temp->bukuBerikutnya) temp = temp->bukuBerikutnya;
            temp->bukuBerikutnya = bukuBaru;
        }
    }
}

void hapusAnggota(string id) {
    Anggota* temp = kepalaAnggota;
    Anggota* prev = nullptr;

    while (temp) {
        if (temp->idAnggota == id) {
            Buku* bukuTemp = temp->kepalaBuku;
            while (bukuTemp) {
                Buku* hapus = bukuTemp;
                bukuTemp = bukuTemp->bukuBerikutnya;
                delete hapus;
            }

            if (prev) {
                prev->anggotaBerikutnya = temp->anggotaBerikutnya;
            } else {
                kepalaAnggota = temp->anggotaBerikutnya;
            }
            delete temp;
            return;
        }
        prev = temp;
        temp = temp->anggotaBerikutnya;
    }
}

void tampilkanData() {
    Anggota* temp = kepalaAnggota;
    while (temp) {
        cout << "Anggota: " << temp->namaAnggota << " (ID: " << temp->idAnggota << ")\n";
        Buku* bukuTemp = temp->kepalaBuku;
        while (bukuTemp) {
            cout << "  -> Buku: " << bukuTemp->judulBuku << ", Pengembalian: " << bukuTemp->tanggalPengembalian << "\n";
        }
        temp = temp->anggotaBerikutnya;
    }
}

```



```

        bukuTemp = bukuTemp->bukuBerikutnya;
    }
    temp = temp->anggotaBerikutnya;
    cout << endl;
}

private:
Anggota* cariAnggota(string id) {
    Anggota* temp = kepalaAnggota;
    while (temp) {
        if (temp->idAnggota == id) return temp;
        temp = temp->anggotaBerikutnya;
    }
    return nullptr;
}
};

int main() {
    MultiLinkedList perpustakaan;

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

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

    // 3. Menambahkan buku baru
    perpustakaan.tambahBuku("A001", "Struktur Data", "10/12/2024");

    // 4. Menghapus anggota Dito
    perpustakaan.hapusAnggota("A002");

    // 5. Menampilkan data anggota dan buku yang dipinjam
    cout << "Data Anggota dan Buku yang Dipinjam:\n";
    perpustakaan.tampilkanData();

    return 0;
}

```

Penjelasan

Program ini menggunakan **Multi Linked List** untuk mengelola data anggota perpustakaan dan daftar buku yang dipinjam. **Anggota** memiliki informasi dasar seperti nama dan ID, serta memiliki linked list buku yang merepresentasikan buku yang dipinjam oleh anggota tersebut. Setiap buku memiliki data seperti judul buku dan tanggal pengembalian.

Di dalam kelas **MultiLinkedList**, terdapat beberapa fungsi penting:

1. **tambahAnggota**: Menambahkan anggota baru ke daftar anggota.
2. **tambahBuku**: Menambahkan buku ke anggota tertentu berdasarkan ID anggota.
3. **hapusAnggota**: Menghapus anggota beserta seluruh buku yang dipinjamnya.
4. **tampilkanData**: Menampilkan semua data anggota dan buku yang dipinjam.

Dalam fungsi main, program pertama-tama menambahkan tiga anggota yaitu Rani, Dito, dan Vina. Selanjutnya, buku ditambahkan ke Rani dan Dito sesuai instruksi. Buku tambahan "Struktur Data" ditambahkan ke Rani. Setelah itu, anggota Dito beserta buku yang dipinjam dihapus. Terakhir, program menampilkan semua data anggota yang tersisa beserta daftar buku yang dipinjam.

Outputnya

```
Data Anggota dan Buku yang Dipinjam:
Anggota: Rani (ID: A001)
-> Buku: Pemrograman C++, Pengembalian: 01/12/2024
-> Buku: Struktur Data, Pengembalian: 10/12/2024

Anggota: Vina (ID: A003)
```

5. Kesimpulan

Kesimpulan dari praktikum struktur data dengan materi **Multi-Linked List** adalah bahwa struktur ini memungkinkan representasi data hierarkis yang kompleks dengan setiap node dapat memiliki pointer ke node lain (induk) dan ke node cabang (child). Melalui implementasi ini, hubungan seperti antara karyawan dan bawahannya dapat direpresentasikan dengan lebih fleksibel. Praktikum mencakup operasi dasar seperti penambahan node induk dan node bawahan, penghapusan node dari list, serta traversal untuk menampilkan seluruh data beserta relasinya. Penggunaan *pointer* menjadi kunci dalam pengelolaan struktur ini, sehingga penting untuk memastikan pembersihan memori dengan destructor agar terhindar dari *memory leaks*. Struktur Multi-Linked List ini memiliki berbagai aplikasi nyata, seperti manajemen organisasi, sistem proyek, dan representasi data bertingkat lainnya. Dengan memahami materi ini, peserta praktikum mendapatkan wawasan tentang cara mengelola struktur data yang lebih kompleks dibandingkan *linked list* biasa serta pentingnya efisiensi memori dalam pemrograman berbasis *pointer*.