

LAPORAN PRAKTIKUM
Modul 13
“MULTI LINKED LIST”



Disusun Oleh:

Ahmad Al - Farizi - 2311104054

Kelas :

S1SE-07-02

Dosen :

Wahyu Andi Saputra, S.Pd, M.Eng

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024

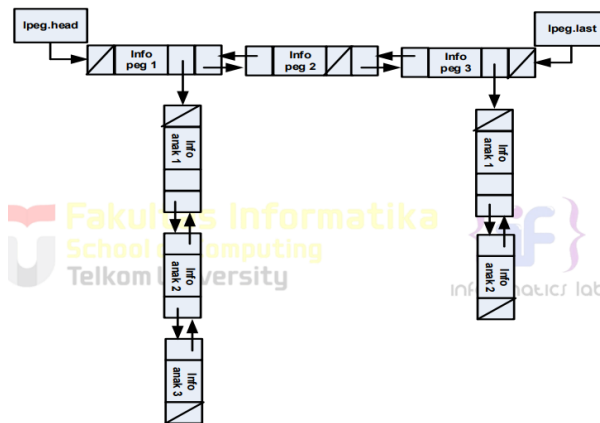
1. Tujuan

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

2. Landasan Teori

Multi List merupakan sekumpulan list yang berbeda yang memiliki suatu keterhubungan satu sama lain. Tiap elemen dalam multi link list dapat membentuk list sendiri. Biasanya ada yang bersifat sebagai list induk dan list anak.

Contoh Multi Linked list dapat dilihat pada gambar berikut:

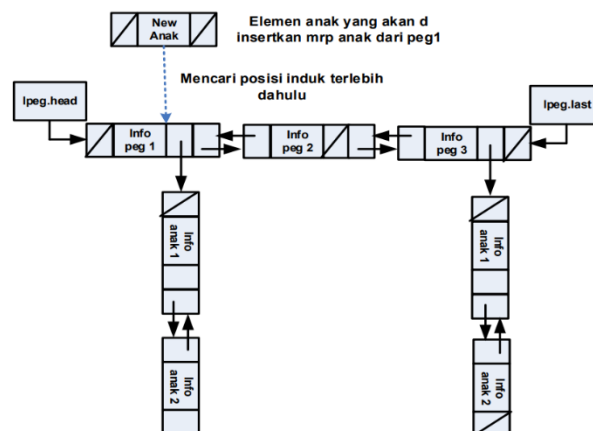


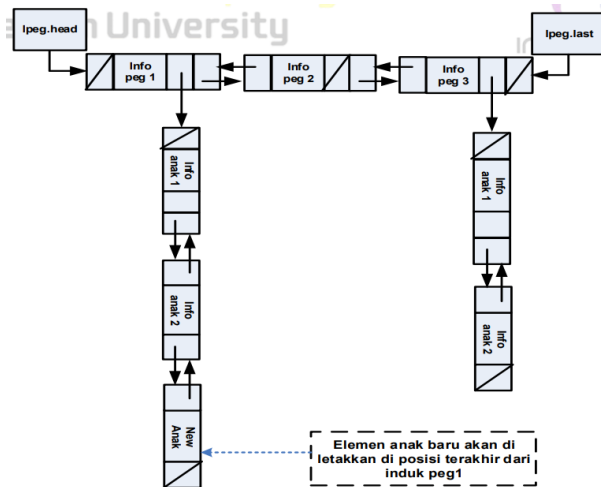
Jadi, dari implementasi di atas akan terdapat dua buah list, list pegawai dan list anak. Dimana untuk list pegawai menunjuk satu buah list anak. Disini list induknya adalah list pegawai dan list anaknya adalah list anak.

1. Insert

A. Insert Anak

Dalam penambahan elemen anak harus diketahui dulu elemen induknya. Berikut ini ilustrasi insert anak dengan konsep insert last:





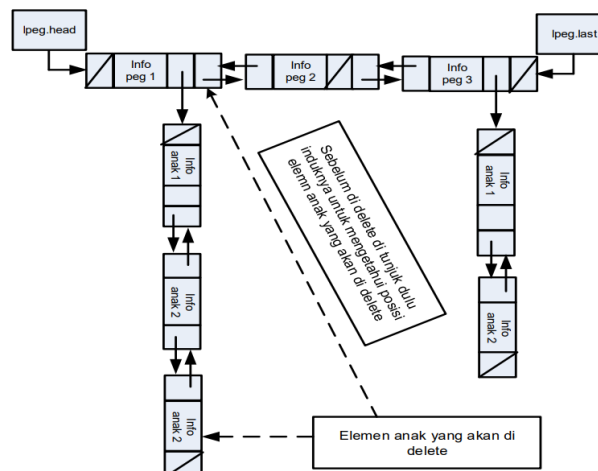
B. Insert Induk

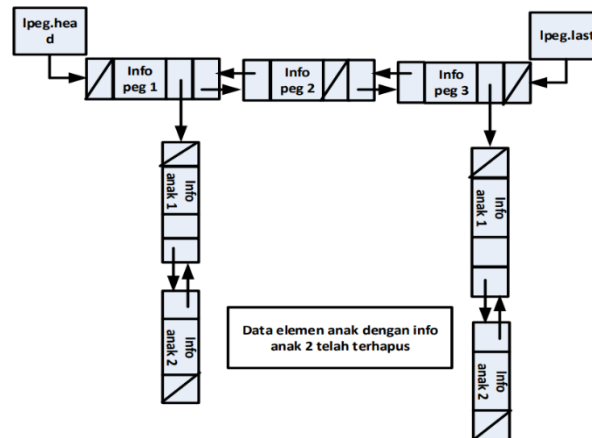
Untuk insert elemen induk sama dengan konsep insert pada single, double dan circular linked list.

2. Delete

A. Delete Anak

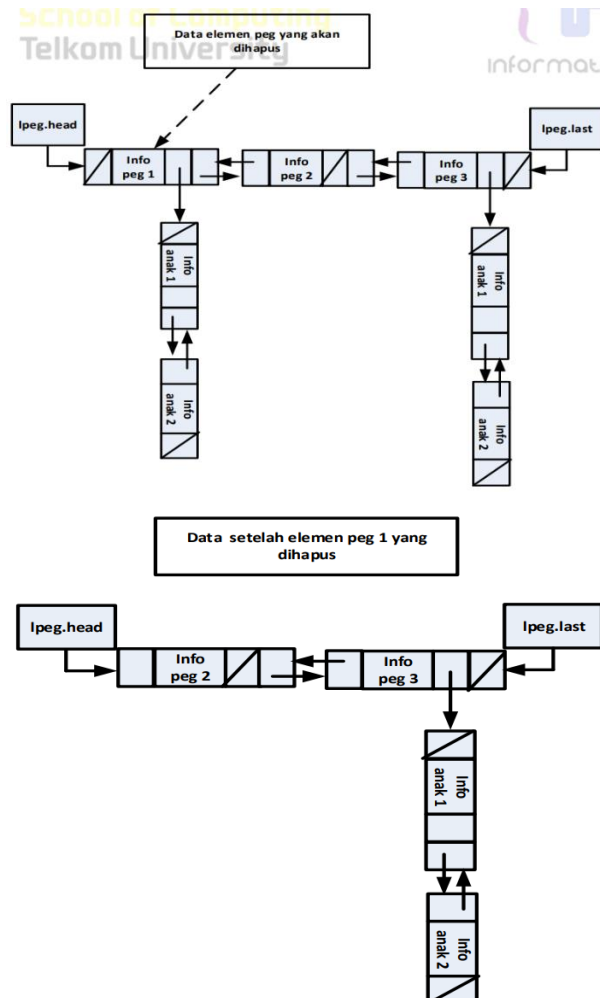
Sama dengan insert anak untuk delete anak maka harus diketahui dulu induknya. Berikut ini gambar ilustrasinya untuk delete last pada induk peg 1:





B. Delete Induk

Untuk delete elemen induk maka saat di hapus maka seluruh anak dengan induk tersebut juga harus dihapus. Berikut ini gambar ilustrasinya:



3. Guided

1. Guided 1

Program di bawah ini mengimplementasikan struktur multi-linked list menggunakan kelas MultiLinkedList, di mana setiap node parent dapat memiliki daftar anak. Fitur utama mencakup penambahan parent (addParent), penambahan anak ke parent tertentu (addChild), dan penampilan hierarki parent-child (display). Destruktor memastikan semua node dihapus untuk mencegah kebocoran memori. Pada fungsi main, program menambahkan parent, menambahkan anak ke masing-masing parent, lalu menampilkan struktur parent dan anak-anaknya.

Kode Program:

```
#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;

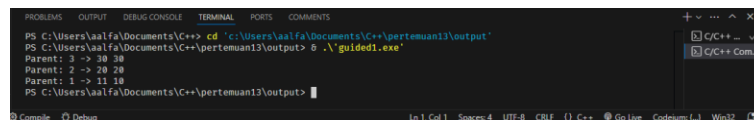
    mList.addParent(1);
    mList.addParent(2);
    mList.addParent(3);

    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 dari Kode Program:



2. Guided 2

Program di bawah ini membangun struktur hirarki karyawan menggunakan kelas `EmployeeList`, di mana setiap karyawan dapat memiliki bawahan. Fungsi utama meliputi penambahan karyawan (`addEmployee`), penambahan bawahan (`addSubordinate`), dan penampilan struktur hirarki (`display`). Pada fungsi `main`, beberapa karyawan beserta bawahannya ditambahkan, lalu struktur ditampilkan. Destruktor memastikan semua memori dibersihkan.

Kode Program:

```

#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;
    }
}

};

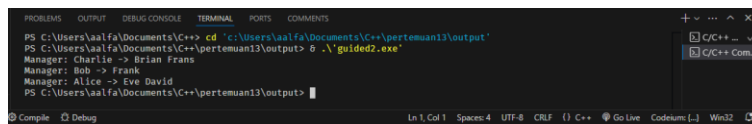
int main() {
    EmployeeList empList;

    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 dari Kode Program:



```
PS C:\Users\laalfa\Documents\C++> cd 'C:\Users\laalfa\Documents\C++\pertemuan13\output'  
PS C:\Users\laalfa\Documents\C++\pertemuan13\output> .\guided2.exe  
Manager: Charlie -> Brian Frans  
Manager: Bob -> Frank  
Manager: Alice -> Eve David  
PS C:\Users\laalfa\Documents\C++\pertemuan13\output>
```

3. Guided 3

Program ini membuat struktur hirarki karyawan dengan kelas EmployeeList menggunakan multi-linked list. Karyawan dapat ditambahkan sebagai induk (addEmployee) atau bawahan (addSubordinate), serta dihapus bersama bawahannya (deleteEmployee) atau hanya bawahan tertentu (deleteSubordinate). Fungsi display menampilkan struktur karyawan. Pada fungsi main, karyawan dan bawahan ditambahkan, struktur awal ditampilkan, lalu dilakukan penghapusan sebelum menampilkan struktur yang diperbarui. Destruktor membersihkan memori.

Kode Program:

```
#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

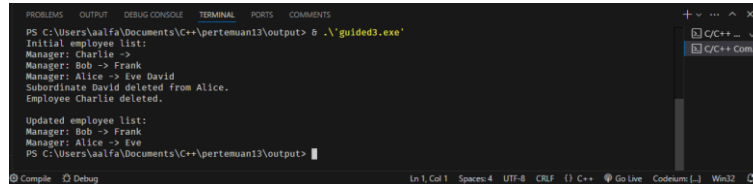
empList.deleteSubordinate("Alice", "David"); // Menghapus David dari
Alice
empList.deleteEmployee("Charlie"); // Menghapus Charlie

cout << "\nUpdated employee list:" << endl;
empList.display(); // Menampilkan isi daftar setelah penghapusan

return 0;
}

```

Output dari Kode Program:



```

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 C:\Users\aa\fa\Documents\C++\pertemuan13\output>

```

4. Unguided

1. Nomor 1

Program ini menggunakan Multi Linked List untuk mengelola data pegawai dan proyek. Setiap pegawai memiliki nama dan ID, serta setiap proyek memiliki nama dan durasi. Program ini memungkinkan penambahan pegawai dan proyek, penghapusan proyek dari pegawai, serta menampilkan data pegawai beserta proyek mereka.

Kode Program:

```

#include <iostream>

#include <string>

using namespace std;

struct Book {

```

```
string title;
string returnDate;
Book* next;
};

struct Member {
    string memberName;
    string memberID;
    Book* bookHead;
    Member* next;
};

class MultiLinkedList {
private:
    Member* head;

    Member* findMember(const string& memberID) {
        Member* current = head;
        while (current != nullptr && current->memberID != memberID) {
            current = current->next;
        }
        return current;
    }

    void deleteBooks(Book* bookHead) {
        while (bookHead != nullptr) {
            Book* temp = bookHead;
            bookHead = bookHead->next;
            delete temp;
        }
    }
}
```

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

    ~MultiLinkedList() {
        while (head != nullptr) {
            Member* temp = head;
            head = head->next;
            deleteBooks(temp->bookHead);
            delete temp;
        }
    }

    void addMember(const string& name, const string& id) {
        Member* newMember = new Member{name, id, nullptr, nullptr};
        if (head == nullptr) {
            head = newMember;
        } else {
            Member* current = head;
            while (current->next != nullptr) {
                current = current->next;
            }
            current->next = newMember;
        }
    }

    void addBook(const string& memberID, const string& title, const string&
returnDate) {
        Member* member = findMember(memberID);
        if (member == nullptr) return;

        Book* newBook = new Book{title, returnDate, member->bookHead};
        member->bookHead = newBook;
```



```
}

void removeMember(const string& memberID) {
    Member* current = head;
    Member* previous = nullptr;

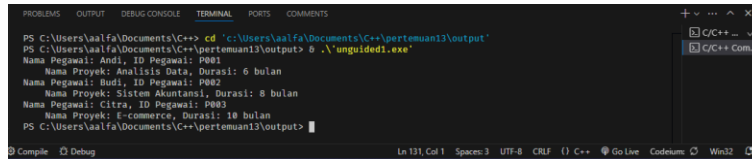
    while (current != nullptr && current->memberID != memberID) {
        previous = current;
        current = current->next;
    }

    if (current != nullptr) {
        if (previous == nullptr) {
            head = current->next;
        } else {
            previous->next = current->next;
        }
        deleteBooks(current->bookHead);
        delete current;
    }
}

void display() {
    Member* current = head;
    while (current != nullptr) {
        cout << "Nama Anggota: " << current->memberName << ", ID
Anggota: " << current->memberID << endl;
        Book* book = current->bookHead;
        while (book != nullptr) {
            cout << "    Judul Buku: " << book->title << ", Tanggal
Pengembalian: " << book->returnDate << endl;
            book = book->next;
        }
    }
}
```

```
        }  
        current = current->next;  
    }  
}  
};  
  
int main() {  
    MultiLinkedList mll;  
  
    // Masukkan data anggota  
    mll.addMember("Rani", "A001");  
    mll.addMember("Dito", "A002");  
    mll.addMember("Vina", "A003");  
  
    // Tambahkan buku yang dipinjam  
    mll.addBook("A001", "Pemrograman C++", "01/12/2024");  
    mll.addBook("A002", "Algoritma Pemrograman", "15/12/2024");  
  
    // Tambahkan buku baru  
    mll.addBook("A001", "Struktur Data", "10/12/2024");  
  
    // Hapus anggota Dito beserta buku yang dipinjam  
    mll.removeMember("A002");  
  
    // Tampilkan seluruh data anggota dan buku yang dipinjam  
    mll.display();  
  
    return 0;  
}
```

Output dari Kode Program:



2. Nomor 2

Program ini menggunakan Multi Linked List untuk mengelola data anggota perpustakaan dan buku yang dipinjam. Setiap anggota memiliki nama dan ID, dan setiap buku memiliki judul dan tanggal pengembalian. Program ini mendukung penambahan anggota dan buku, penghapusan anggota beserta buku yang dipinjam, serta menampilkan data anggota beserta buku yang mereka pinjam.

Kode Program:

```
#include <iostream>
#include <string>
#include <sstream>
#include <iomanip>
#include <ctime>

using namespace std;

struct Book {
    string title;
    string returnDate;
    Book* next;
};

struct Member {
    string memberName;
    string memberID;
    Book* bookHead;
    Member* next;
```

```
};

class MultiLinkedList {
private:
    Member* head;

    Member* findMember(const string& memberID) {
        Member* current = head;
        while (current != nullptr && current->memberID != memberID) {
            current = current->next;
        }
        return current;
    }

    void deleteBooks(Book* bookHead) {
        while (bookHead != nullptr) {
            Book* temp = bookHead;
            bookHead = bookHead->next;
            delete temp;
        }
    }

    tm stringToDate(const string& dateStr) {
        tm tmDate = {};
        stringstream ss(dateStr);
        ss >> get_time(&tmDate, "%d/%m/%Y");
        return tmDate;
    }

    bool compareDates(const string& date1, const string& date2) {
        tm tmDate1 = stringToDate(date1);
        tm tmDate2 = stringToDate(date2);
```

```
        return mktime(&tmDate1) < mktime(&tmDate2);
    }

    void sortedInsert(Book*& head, Book* newBook) {
        if (head == nullptr || compareDates(newBook->returnDate, head->returnDate)) {
            newBook->next = head;
            head = newBook;
        } else {
            Book* current = head;
            while (current->next != nullptr && !compareDates(newBook->returnDate, current->next->returnDate)) {
                current = current->next;
            }
            newBook->next = current->next;
            current->next = newBook;
        }
    }

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

    ~MultiLinkedList() {
        while (head != nullptr) {
            Member* temp = head;
            head = head->next;
            deleteBooks(temp->bookHead);
            delete temp;
        }
    }

    void addMember(const string& name, const string& id) {
```

```
Member* newMember = new Member{name, id, nullptr, nullptr};
if (head == nullptr) {
    head = newMember;
} else {
    Member* current = head;
    while (current->next != nullptr) {
        current = current->next;
    }
    current->next = newMember;
}

void addBook(const string& memberID, const string& title, const string&
returnDate) {
    Member* member = findMember(memberID);
    if (member == nullptr) return;

    Book* newBook = new Book{title, returnDate, nullptr};
    sortedInsert(member->bookHead, newBook);
}

void removeMember(const string& memberID) {
    Member* current = head;
    Member* previous = nullptr;

    while (current != nullptr && current->memberID != memberID) {
        previous = current;
        current = current->next;
    }

    if (current != nullptr) {
        if (previous == nullptr) {
```

```
        head = current->next;
    } else {
        previous->next = current->next;
    }
    deleteBooks(current->bookHead);
    delete current;
}

void display() {
    Member* current = head;
    while (current != nullptr) {
        cout << "Nama Anggota: " << current->memberName << ", ID
Anggota: " << current->memberID << endl;
        Book* book = current->bookHead;
        while (book != nullptr) {
            cout << "    Judul Buku: " << book->title << ", Tanggal
Pengembalian: " << book->returnDate << endl;
            book = book->next;
        }
        current = current->next;
    }
};

int main() {
    MultiLinkedList mll;

    // Masukkan data anggota
    mll.addMember("Rani", "A001");
    mll.addMember("Dito", "A002");
    mll.addMember("Vina", "A003");
```

```
// Tambahkan buku yang dipinjam
mll.addBook("A001", "Pemrograman C++", "01/12/2024");
mll.addBook("A002", "Algoritma Pemrograman", "15/12/2024");

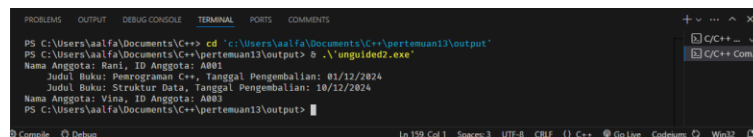
// Tambahkan buku baru
mll.addBook("A001", "Struktur Data", "10/12/2024");

// Hapus anggota Dito beserta buku yang dipinjam
mll.removeMember("A002");

// Tampilkan seluruh data anggota dan buku yang dipinjam
mll.display();

return 0;
}
```

Output Kode Program:



```
PS C:\Users\aa\Documents\C++> cd 'C:\Users\aa\Documents\C++\pertemuan13\output'
PS C:\Users\aa\Documents\C++\pertemuan13\output> .\unguided2.exe
Nama Anggota: Rani, ID Anggota: A001
Judul Buku: Pemrograman C++, Tanggal Pengembalian: 01/12/2024
Judul Buku: Struktur Data, Tanggal Pengembalian: 10/12/2024
Nama Anggota: Vina, ID Anggota: A003
PS C:\Users\aa\Documents\C++\pertemuan13\output>
```

5. Kesimpulan

Multi Linked List adalah struktur data yang terdiri dari beberapa list yang saling berhubungan, di mana tiap elemen dalam multi linked list dapat membentuk list tersendiri. Multi linked list biasanya memiliki list induk dan list anak. List induk berisi elemen-elemen yang menunjuk ke list anak, yang berisi elemen-elemen terkait. Untuk menambahkan elemen anak, terlebih dahulu harus diketahui elemen induknya, dan penambahan elemen anak dilakukan dengan menambahkan elemen di akhir list anak (insert last). Penambahan elemen induk dilakukan seperti pada single, double, atau circular linked list, yaitu dengan menambahkan elemen di posisi yang sesuai dalam list induk. Untuk menghapus elemen anak, induknya harus diketahui terlebih dahulu, dan penghapusan dilakukan dengan menghapus elemen anak terakhir (delete last) dari list anak yang terkait dengan elemen induk tertentu. Penghapusan elemen induk berarti seluruh elemen anak yang terkait dengan elemen induk tersebut juga harus dihapus, memastikan bahwa tidak ada elemen anak yang terisolasi tanpa induk. Dengan demikian, multi linked list memungkinkan hubungan yang lebih kompleks antara data,



menghubungkan elemen-elemen yang secara logis terkait satu sama lain melalui list induk dan list anak.

