

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	Pengantalan 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 - 08.30:** 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 Tugas 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.

**LAPORAN PRAKTIKUM
MODUL 13
MULTI LINKED LIST**



Disusun Oleh :

Izzaty Zahara Br Barus – 2311104052

Kelas :

SE-07-02

Dosen :

Wahyu Andi Saputra, S.pd,M.Eng

**PROGRAM STUDI SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024**

I. TUJUAN

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

II. LANDASAN TEORI

Definisi Multi Linked List adalah struktur data yang merupakan pengembangan dari linked list biasa. Dalam struktur ini, setiap node tidak hanya memiliki satu pointer (atau referensi) ke node berikutnya, tetapi juga dapat memiliki beberapa pointer yang mengarah ke node lain. Hal ini memungkinkan penyimpanan data yang lebih kompleks dan terorganisir dengan lebih baik, sehingga cocok digunakan untuk merepresentasikan hubungan yang bersifat banyak ke banyak (many-to-many).

Karakteristik Multi Linked List

1. Beberapa Pointer: Setiap node dalam Multi Linked List dapat memiliki lebih dari satu pointer. Pointer-pointer ini memungkinkan node untuk terhubung ke beberapa node lain secara simultan. Misalnya, dalam konteks manajemen pegawai dan proyek, setiap pegawai dapat memiliki pointer ke proyek yang mereka kelola, dan setiap proyek dapat memiliki pointer ke pegawai yang mengelolanya.
2. Struktur Hierarkis: Multi Linked List sering digunakan untuk merepresentasikan hubungan yang bersifat hierarkis atau relasional. Sebagai contoh, dalam sistem manajemen proyek, pegawai dapat memiliki beberapa proyek, dan setiap proyek dapat memiliki beberapa pegawai yang terlibat. Dengan demikian, Multi Linked List dapat menggambarkan hubungan secara dua arah atau lebih.
3. Fleksibilitas: Multi Linked List memberikan fleksibilitas dalam pengelolaan data yang saling terkait. Dengan menggunakan beberapa pointer, kita dapat dengan mudah menavigasi, mengakses, dan memanipulasi data tanpa harus melakukan pengolahan yang kompleks. Struktur ini sangat efektif dalam mengelola data yang saling berhubungan erat.

Keunggulan Multi Linked List

- Memungkinkan representasi data yang lebih kompleks dibandingkan linked list biasa.
- Mampu mengelola hubungan many-to-many dengan lebih efisien.
- Memberikan fleksibilitas untuk memodifikasi hubungan antar node tanpa harus memindahkan atau menghapus node secara fisik.

Penerapan Multi Linked List

1. Manajemen Proyek: Digunakan untuk merepresentasikan hubungan antara pegawai dan proyek yang mereka kelola.

2. Sistem Basis Data: Membantu merepresentasikan relasi antara tabel-tabel yang memiliki hubungan kompleks.
3. Grafik: Multi Linked List dapat digunakan untuk menyimpan grafik (graph) di mana setiap node dapat terhubung dengan beberapa node lain.

Dengan struktur ini, Multi Linked List menjadi solusi ideal untuk permasalahan yang memerlukan hubungan kompleks antara data yang saling terkait.

III. GUIDE

1. Guide

a. Syntax

```
#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);

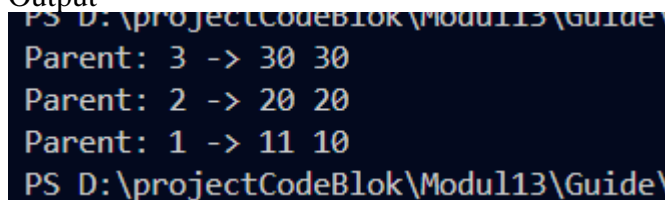
    mList.addChild(1, 10);
```

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

b. Penjelasan syntax

Kode di atas mengimplementasikan struktur data Multi Linked List menggunakan C++. Struktur ini memungkinkan setiap node untuk memiliki anak (child) selain node berikutnya (next). Kelas MultiLinkedList memiliki metode untuk menambahkan node parent (addParent), menambahkan anak pada node tertentu (addChild), dan menampilkan semua node beserta anak-anaknya (display). Pada main(), beberapa parent (1, 2, 3) ditambahkan, masing-masing dengan anak-anak tertentu (10, 11 untuk 1; 20 untuk 2; 30 untuk 3). Kode juga menyertakan destruktorkan untuk membersihkan semua node parent dan child guna mencegah kebocoran memori. Output menunjukkan hubungan parent-child yang telah dibuat.

c. Output



```
PS D:\projectCodeBlok\Modul13\Guide\  
Parent: 3 -> 30 30  
Parent: 2 -> 20 20  
Parent: 1 -> 11 10  
PS D:\projectCodeBlok\Modul13\Guide\
```

2. Guide

a. Syntax

```
#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;  
    }
```

b. Penjelasan

Kode di atas mengimplementasikan struktur data Multi Linked List untuk sistem manajemen karyawan, di mana setiap karyawan (node) dapat memiliki bawahan (subordinate). Kelas `EmployeeList` menyediakan metode untuk menambahkan karyawan baru (`addEmployee`), menambahkan bawahan pada seorang manajer tertentu (`addSubordinate`), dan menampilkan daftar manajer beserta bawahan mereka (`display`). Pada `main()`, beberapa karyawan ditambahkan (Alice, Bob, Charlie), dan setiap manajer diberikan bawahan mereka masing-masing (misalnya, Alice memiliki David dan Eve). Kode juga dilengkapi destruktur untuk membersihkan semua node dan menghindari kebocoran memori. Output menampilkan hubungan manajer-bawahan yang telah dibuat.

c. Output

```
Manager: Charlie -> Brian Frans  
Manager: Bob -> Frank  
Manager: Alice -> Eve David  
PS D:\projectCodeBlok\Modul13\Guide\output
```

3. Guide

a. Syntax

```
#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;
}
```

b. Penjelasan

Kode ini mengimplementasikan sistem manajemen karyawan menggunakan Multi Linked List, di mana setiap karyawan

(node) dapat memiliki bawahan (subordinate). Kelas `EmployeeList` memungkinkan penambahan karyawan (`addEmployee`), menambahkan bawahan ke karyawan tertentu (`addSubordinate`), menghapus karyawan beserta semua bawahannya (`deleteEmployee`), serta menghapus bawahan tertentu dari seorang manajer (`deleteSubordinate`). Dalam `main()`, beberapa karyawan dan hubungan manajer-bawahan ditambahkan (misalnya, Alice memiliki David dan Eve, Bob memiliki Frank). Kode juga mendemonstrasikan penghapusan subordinat David dari Alice dan penghapusan karyawan Charlie. Output menampilkan daftar karyawan dan bawahannya sebelum dan setelah penghapusan.

c. Output

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

Updated employee list:
Manager: Bob -> Frank
Manager: Alice -> Eve
PS D:\projectCodeBlok\Modul13\Guide\output>
```

IV. UNGUIDED

1. Unguided 1

a. Syntax :

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

using namespace std;

class Proyek {
public:
    string nama;
    int durasi;
    Proyek* next;

    Proyek(string n, int d) : nama(n), durasi(d), next(nullptr) {}
};
```

```
class Pegawai {
public:
    string nama;
    string id;
    Proyek* proyek_head;
    Pegawai* next;

    Pegawai(string n, string i) : nama(n), id(i),
    proyek_head(nullptr), next(nullptr) {}
};

class ManajemenPegawai {
private:
    Pegawai* pegawai_head;

public:
    ManajemenPegawai() : pegawai_head(nullptr) {}

    void tambahPegawai(string nama, string id) {
        Pegawai* pegawai = new Pegawai(nama, id);
        if (!pegawai_head) {
            pegawai_head = pegawai;
        } else {
            Pegawai* current = pegawai_head;
            while (current->next) {
                current = current->next;
            }
            current->next = pegawai;
        }
    }

    void tambahProyek(string id_pegawai, string nama_proyek, int
    durasi) {
        Pegawai* current = pegawai_head;
        while (current) {
            if (current->id == id_pegawai) {
                Proyek* proyek = new Proyek(nama_proyek, durasi);
                proyek->next = current->proyek_head;
                current->proyek_head = proyek;
                return;
            }
            current = current->next;
        }
    }

    void hapusProyek(string id_pegawai, string nama_proyek) {
        Pegawai* current = pegawai_head;
        while (current) {
            if (current->id == id_pegawai) {
                Proyek* proyek = current->proyek_head;
```



```
        Proyek* prev = nullptr;
        while (proyek) {
            if (proyek->nama == nama_proyek) {
                if (prev) {
                    prev->next = proyek->next;
                } else {
                    current->proyek_head = proyek->next;
                }
                delete proyek;
                return;
            }
            prev = proyek;
            proyek = proyek->next;
        }
        current = current->next;
    }

    void tampilkanData() {
        Pegawai* current = pegawai_head;
        while (current) {
            cout << "Pegawai: " << current->nama << ", ID: " <<
current->id << endl;
            Proyek* proyek = current->proyek_head;
            while (proyek) {
                cout << "  Proyek: " << proyek->nama << ", Durasi: "
<< proyek->durasi << " bulan" << endl;
                proyek = proyek->next;
            }
            current = current->next;
        }
    }
};

int main() {
    ManajemenPegawai manajemen;
    manajemen.tambahPegawai("Andi", "P001");
    manajemen.tambahPegawai("Budi", "P002");
    manajemen.tambahPegawai("Citra", "P003");

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

    manajemen.tambahProyek("P001", "Analisis Data", 6);
    manajemen.hapusProyek("P001", "Aplikasi Mobile");

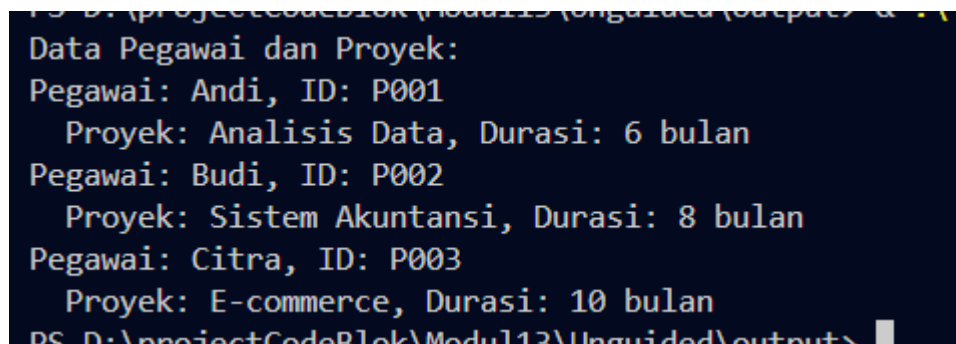
    cout << "Data Pegawai dan Proyek:" << endl;
    manajemen.tampilkanData();
}
```

```
        return 0;  
    }
```

b. Penjelasan Syntax

Kode ini mengimplementasikan sistem manajemen pegawai dan proyek menggunakan Multi Linked List. Setiap pegawai dapat memiliki beberapa proyek yang terhubung melalui pointer. Kelas ManajemenPegawai menyediakan fungsi untuk menambahkan pegawai (tambahPegawai), menambahkan proyek ke pegawai tertentu (tambahProyek), menghapus proyek tertentu dari pegawai (hapusProyek), dan menampilkan semua data pegawai serta proyek mereka (tampilkanData). Dalam main(), beberapa pegawai (Andi, Budi, Citra) dan proyek mereka ditambahkan, lalu satu proyek milik Andi ("Aplikasi Mobile") dihapus. Output menunjukkan data pegawai beserta proyek-proyek mereka yang tersisa.

c. Output



```
PS D:\projectCodeBlok\Modul13\Unguided> .\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  
PS D:\projectCodeBlok\Modul13\Unguided>
```

2. Unguided 2

a. Syntax

```
#include <iostream>  
#include <string>  
  
using namespace std;  
  
class Buku {  
public:  
    string judul;  
    string tanggal_pengembalian;  
    Buku* next;  
  
    Buku(string j, string t) : judul(j), tanggal_pengembalian(t),  
    next(nullptr) {}  
};  
  
class Anggota {
```

```
public:
    string nama;
    string id;
    Buku* buku_head;
    Anggota* next;

    Anggota(string n, string i) : nama(n), id(i),
    buku_head(nullptr), next(nullptr) {}
};

class ManajemenPerpustakaan {
private:
    Anggota* anggota_head;

public:
    ManajemenPerpustakaan() : anggota_head(nullptr) {}

    void tambahAnggota(string nama, string id) {
        Anggota* anggota = new Anggota(nama, id);
        if (!anggota_head) {
            anggota_head = anggota;
        } else {
            Anggota* current = anggota_head;
            while (current->next) {
                current = current->next;
            }
            current->next = anggota;
        }
    }

    void tambahBuku(string id_anggota, string judul, string
    tanggal_pengembalian) {
        Anggota* current = anggota_head;
        while (current) {
            if (current->id == id_anggota) {
                Buku* buku = new Buku(judul,
                tanggal_pengembalian);
                buku->next = current->buku_head;
                current->buku_head = buku;
                return;
            }
            current = current->next;
        }
    }

    void hapusAnggota(string id_anggota) {
        Anggota* current = anggota_head;
        Anggota* prev = nullptr;
        while (current) {
            if (current->id == id_anggota) {
```

```
        if (prev) {
            prev->next = current->next;
        } else {
            anggota_head = current->next;
        }
        Buku* buku = current->buku_head;
        while (buku) {
            Buku* temp = buku;
            buku = buku->next;
            delete temp;
        }
        delete current;
        return;
    }
    prev = current;
    current = current->next;
}

void tampilkanData() {
    Anggota* current = anggota_head;
    while (current) {
        cout << "Anggota: " << current->nama << ", ID: " <<
current->id << endl;
        Buku* buku = current->buku_head;
        while (buku) {
            cout << " Buku: " << buku->judul << ", Tanggal
Pengembalian: " << buku->tanggal_pengembalian << endl;
            buku = buku->next;
        }
        current = current->next;
    }
}

};

int main() {
    ManajemenPerpustakaan manajemen;
    manajemen.tambahAnggota("Rani", "A001");
    manajemen.tambahAnggota("Dito", "A002");
    manajemen.tambahAnggota("Vina", "A003");

    manajemen.tambahBuku("A001", "Pemrograman C++",
"01/12/2024");
    manajemen.tambahBuku("A002", "Algoritma
Pemrograman", "15/12/2024");
    manajemen.tambahBuku("A001", "Struktur Data",
"10/12/2024");

    manajemen.hapusAnggota("A002");
}
```

```
cout << "Data Anggota dan Buku yang Dipinjam:" << endl;  
manajemen.tampilkanData();  
  
return 0;  
}
```

b. Penjelasan

Kode ini adalah sistem manajemen perpustakaan berbasis Multi Linked List. Setiap anggota perpustakaan memiliki daftar buku yang dipinjam, yang terhubung melalui pointer. Kelas ManajemenPerpustakaan menyediakan fungsi untuk menambahkan anggota (tambahAnggota), menambahkan buku yang dipinjam oleh anggota (tambahBuku), menghapus anggota bersama buku-bukunya (hapusAnggota), dan menampilkan data anggota serta buku yang dipinjam (tampilkanData). Pada bagian main(), beberapa anggota ditambahkan (Rani, Dito, Vina), dua buku dipinjam oleh Rani, dan satu oleh Dito. Kemudian, Dito dihapus dari sistem, sehingga hanya data Rani dan Vina yang ditampilkan.

c. Output

```
Data Anggota dan Buku yang Dipinjam:  
Anggota: Rani, ID: A001  
    Buku: Struktur Data, Tanggal Pengembalian: 10/12/2024  
    Buku: Pemrograman C++, Tanggal Pengembalian: 01/12/2024  
Anggota: Vina, ID: A003  
PS D:\projectCodeBlok\Modul13\Unguided\output> |
```

V. KESIMPULAN

Multi Linked List adalah jenis struktur data yang digunakan untuk mengelola hubungan antar data yang lebih rumit, seperti hubungan banyak-ke-banyak atau yang memiliki hierarki. Struktur ini berbeda dari Linked List biasa karena setiap node di dalamnya memiliki lebih dari satu pointer, yang memungkinkan kita menghubungkan satu node dengan beberapa node lainnya.

Ini sangat berguna untuk merepresentasikan berbagai hubungan yang kompleks. Misalnya, dalam sebuah perusahaan, seorang pegawai bisa terhubung dengan beberapa proyek, atau seorang manajer bisa memiliki banyak bawahan. Begitu juga, dalam sebuah perpustakaan, seorang anggota bisa meminjam banyak buku.

Keunggulan dari Multi Linked List adalah kemampuannya untuk menyusun data yang saling terhubung dalam berbagai cara yang fleksibel. Kita bisa menambah, menghapus, atau mencari data dengan mudah, sesuai dengan hubungan antar node. Struktur data ini sering digunakan di berbagai aplikasi, mulai dari manajemen proyek, pengelolaan data karyawan, sistem perpustakaan, hingga basis data yang membutuhkan relasi antar tabel.

Dengan kemampuannya mengelola data yang saling terkait, Multi Linked List

adalah alat yang sangat berguna untuk menyelesaikan berbagai masalah yang melibatkan hubungan kompleks antar entitas dalam dunia nyata.