

**LAPORAN PRAKTIKUM**  
**MODUL 13**  
**MULTI LINKED LIST**



**DISUSUN OLEH :**  
**TIURMA GRACE ANGELINA – 2311104042**  
**SE0702**

**DOSEN :**  
**WAHYU ANDI SAPUTRA**

**PROGRAM STUDI S1 SOFTWARE ENGINEERING**  
**FAKULTAS INFORMATIKA**  
**TELKOM UNIVERSITY**  
**PURWOKERTO**  
**2024**

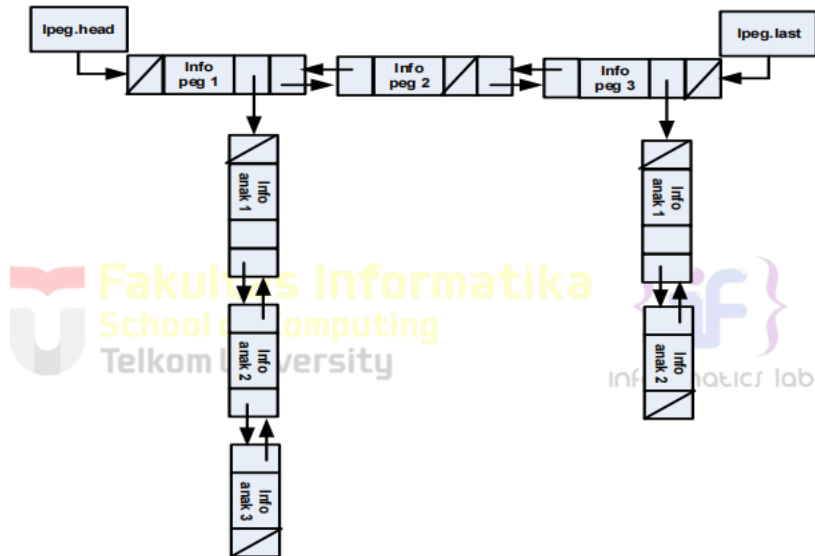
## **1. TUJUAN**

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

## **2. Landasan Teori**

Multi Linked List adalah struktur data yang terdiri dari kumpulan daftar (list) yang saling berhubungan, di mana setiap elemen dalam suatu daftar dapat membentuk daftar baru sebagai bagian dari elemen tersebut. Pada dasarnya, terdapat dua tipe daftar dalam Multi Linked List, yaitu daftar induk (parent list) dan daftar anak (child list). Daftar induk berisi elemen-elemen yang masing-masing dapat merujuk pada daftar anaknya, menciptakan hubungan hierarkis antara elemen induk dan elemen anak. Keunggulan Multi Linked List terletak pada kemampuannya untuk mengelola hubungan kompleks antar data secara terstruktur. Dalam penggunaannya, Multi Linked List sering diterapkan untuk merepresentasikan data yang memiliki relasi hirarkis, seperti sistem organisasi, data pegawai dan anak tugasnya, atau basis data dengan hubungan antar tabel. Operasi dasar pada Multi Linked List mencakup penambahan (insert) dan penghapusan (delete) elemen, baik pada daftar induk maupun daftar anak. Dalam proses penambahan elemen anak, elemen induk terkait harus ditemukan terlebih dahulu, sehingga elemen baru dapat dihubungkan dengan tepat. Begitu pula dengan penghapusan elemen, terutama pada elemen induk, penghapusan tersebut akan berdampak pada semua elemen anak yang terhubung dengan induk tersebut. Implementasi Multi Linked List biasanya memanfaatkan struktur pointer untuk merepresentasikan hubungan antara elemen-elemen dalam daftar. Struktur ini mencakup elemen daftar induk yang memiliki pointer ke daftar anaknya, serta elemen-elemen daftar anak yang saling terhubung secara linear maupun doubly linked. Hal ini memungkinkan fleksibilitas dalam traversing dan manipulasi data.

Contoh multi linked list :



### 3. Guided

#### 1. Guided 1

Code :

```
#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 :

```

C:\Users\USER\Music\13\guided1\bin\Debug\guided1.exe
Parent: 3 -> 30 30
Parent: 2 -> 20 20
Parent: 1 -> 11 10

Process returned 0 (0x0)   execution time : 0.177 s
Press any key to continue.

```

### Penjelasan :

Program ini membuat Multi Linked List, di mana setiap node memiliki child dan hubungan next. Berikut penjelasannya:

1. Node: Setiap node menyimpan data, pointer ke node berikutnya (next), dan pointer ke anak (child).
2. MultiLinkedList:
  - o addParent: Menambahkan parent di awal list.
  - o addChild: Menambahkan anak ke parent tertentu.
  - o display: Menampilkan seluruh struktur (parent beserta anak-anaknya).

- Destruktor: Membersihkan semua node untuk mencegah kebocoran memori.
3. main():
- Menambahkan parent (1, 2, 3).
  - Menambahkan anak untuk tiap parent (contoh: parent 1 punya anak 10, 11).
  - Mencetak list.

## 2. Guided 2

Code :

```
#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 :

```

C:\Users\USER\Music\13\guided2\bin\Debug\guided2.exe
Manager: Charlie -> Brian Frans
Manager: Bob -> Frank
Manager: Alice -> Eve David

Process returned 0 (0x0)   execution time : 0.076 s
Press any key to continue.

```

#### Penjelasan :

Program ini mengimplementasikan Multi-Linked List untuk mengelola karyawan dan subordinate-nya.

Komponen Utama:

1. EmployeeNode: Representasi karyawan dengan atribut:
  - name: Nama karyawan.
  - next: Karyawan berikutnya.
  - subordinate: Subordinate pertama.
2. EmployeeList:
  - addEmployee: Menambahkan karyawan (manajer) ke awal list.
  - addSubordinate: Menambahkan subordinate ke karyawan tertentu.
  - display: Menampilkan daftar karyawan dan subordinate mereka.
  - Destructor: Menghapus semua karyawan dan subordinate dari memori.
3. Fungsi main():
  - Menambahkan karyawan: Alice, Bob, Charlie.
  - Menambahkan subordinate:
    - Alice: David, Eve.
    - Bob: Frank.
    - Charlie: Frans, Brian.



- Menampilkan seluruh hierarki karyawan.

### 3. Guided 3

Code :

```
#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

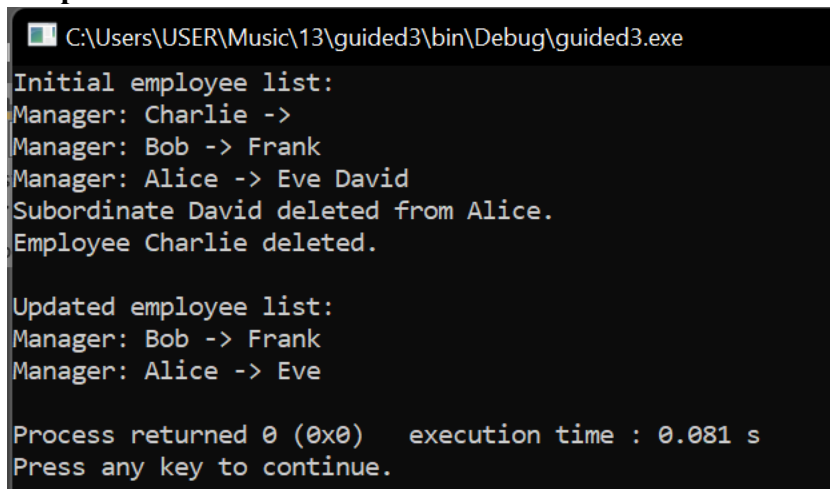
    empList.deleteSubordinate("Alice", "David"); // Menghapus David
    dari Alice
    empList.deleteEmployee("Charlie"); // Menghapus Charlie

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

    return 0;
}

```

**Output :**



```

C:\Users\USER\Music\13\guided3\bin\Debug\guided3.exe
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

Process returned 0 (0x0)   execution time : 0.081 s
Press any key to continue.

```

**Penjelasan :**

Program ini menggunakan Multi-Linked List untuk mengelola daftar karyawan dan subordinate mereka. Berikut komponen utamanya:

1. EmployeeNode: Representasi karyawan dengan atribut:
  - o name: Nama karyawan.
  - o next: Karyawan berikutnya.
  - o subordinate: Subordinate pertama.
2. EmployeeList:
  - o addEmployee: Menambah karyawan (manajer) di awal list.
  - o addSubordinate: Menambah subordinate ke manajer tertentu.
  - o deleteEmployee: Menghapus karyawan dan semua subordinate-nya.
  - o deleteSubordinate: Menghapus subordinate tertentu dari manajer.
  - o display: Menampilkan hierarki karyawan dan subordinate.
  - o Destructor: Membersihkan semua node dari memori.
3. Fungsi main():
  - o Menambah karyawan: Alice, Bob, Charlie.
  - o Menambah subordinate: David, Eve untuk Alice; Frank untuk Bob.
  - o Menghapus subordinate (David) dan karyawan (Charlie).
  - o Menampilkan daftar awal dan setelah penghapusan.

#### 4. Unguided

##### 1. Unguided 1

Code :

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

// Struktur data untuk proyek
struct Proyek {
    string namaProyek;
    int durasi;
    Proyek* nextProyek;
};

// Struktur data untuk pegawai
struct Pegawai {
    string namaPegawai;
    string idPegawai;
    Proyek* daftarProyek;
    Pegawai* nextPegawai;
};

// Fungsi untuk menambahkan pegawai baru
void tambahPegawai(Pegawai*& head, string nama, string id) {
    Pegawai* pegawaiBaru = new Pegawai{nama, id, nullptr, head};
    head = pegawaiBaru;
```

```

}

// Fungsi untuk menambahkan proyek ke pegawai
void tambahProyek(Pegawai* head, string idPegawai, string namaProyek,
int durasi) {
    while (head && head->idPegawai != idPegawai) {
        head = head->nextPegawai;
    }
    if (head) {
        Proyek* proyekBaru = new Proyek{namaProyek, durasi, head-
>daftarProyek};
        head->daftarProyek = proyekBaru;
    }
}

// Fungsi untuk menghapus proyek dari pegawai
void hapusProyek(Pegawai* head, string idPegawai, string namaProyek) {
    while (head && head->idPegawai != idPegawai) {
        head = head->nextPegawai;
    }
    if (head) {
        Proyek* prev = nullptr;
        Proyek* curr = head->daftarProyek;
        while (curr && curr->namaProyek != namaProyek) {
            prev = curr;
            curr = curr->nextProyek;
        }
        if (curr) {
            if (prev) {
                prev->nextProyek = curr->nextProyek;
            } else {
                head->daftarProyek = curr->nextProyek;
            }
            delete curr;
        }
    }
}

// Fungsi untuk menampilkan data pegawai dan proyek mereka
void tampilkanData(Pegawai* head) {
    while (head) {
        cout << "Nama Pegawai: " << head->namaPegawai << ", ID: " <<
head->idPegawai << endl;
        Proyek* proyek = head->daftarProyek;
        while (proyek) {

```

```

        cout << " - Nama Proyek: " << proyek->namaProyek << ", Durasi:
" << proyek->durasi << " bulan" << endl;
        proyek = proyek->nextProyek;
    }
    head = head->nextPegawai;
}
}

int main() {
    Pegawai* daftarPegawai = nullptr;

    // Tambahkan pegawai
    tambahPegawai(daftarPegawai, "Andi", "P001");
    tambahPegawai(daftarPegawai, "Budi", "P002");
    tambahPegawai(daftarPegawai, "Citra", "P003");

    // Tambahkan proyek ke pegawai
    tambahProyek(daftarPegawai, "P001", "Aplikasi Mobile", 12);
    tambahProyek(daftarPegawai, "P002", "Sistem Akuntansi", 8);
    tambahProyek(daftarPegawai, "P003", "E-commerce", 10);

    // Tambahkan proyek baru ke Andi
    tambahProyek(daftarPegawai, "P001", "Analisis Data", 6);

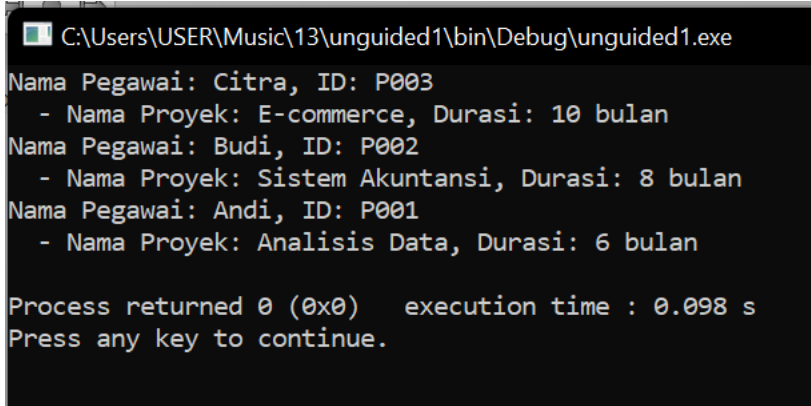
    // Hapus proyek "Aplikasi Mobile" dari Andi
    hapusProyek(daftarPegawai, "P001", "Aplikasi Mobile");

    // Tampilkan data pegawai dan proyek
    tampilkanData(daftarPegawai);

    return 0;
}

```

Output :



```

C:\Users\USER\Music\13\unguided1\bin\Debug\unguided1.exe
Nama Pegawai: Citra, ID: P003
  - Nama Proyek: E-commerce, Durasi: 10 bulan
Nama Pegawai: Budi, ID: P002
  - Nama Proyek: Sistem Akuntansi, Durasi: 8 bulan
Nama Pegawai: Andi, ID: P001
  - Nama Proyek: Analisis Data, Durasi: 6 bulan

Process returned 0 (0x0)   execution time : 0.098 s
Press any key to continue.

```

Penjelasan :

Program ini mengelola data pegawai dan proyek menggunakan Multi-Linked List.

1. Proyek: Menyimpan nama proyek, durasi, dan pointer ke proyek berikutnya.
2. Pegawai: Menyimpan nama, ID pegawai, pointer ke daftar proyek, dan pegawai berikutnya.

Fungsi Utama:

1. tambahPegawai: Menambah pegawai baru ke daftar.
2. tambahProyek: Menambah proyek ke pegawai tertentu berdasarkan ID.
3. hapusProyek: Menghapus proyek tertentu dari pegawai.
4. tampilkanData: Menampilkan seluruh pegawai dan proyek mereka.

Fungsi main():

- Menambah pegawai: Andi, Budi, Citra.
- Menambah proyek:
  - Andi: "Aplikasi Mobile" (12 bulan), "Analisis Data" (6 bulan).
  - Budi: "Sistem Akuntansi" (8 bulan).
  - Citra: "E-commerce" (10 bulan).
- Menghapus proyek "Aplikasi Mobile" dari Andi.
- Menampilkan data pegawai dan proyek.

## 2. Unguided 2

Code :

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

// Struktur data untuk buku
struct Buku {
    string judulBuku;
    string tanggalPengembalian;
    Buku* nextBuku;
};

// Struktur data untuk anggota
struct Anggota {
    string namaAnggota;
    string idAnggota;
    Buku* daftarBuku;
    Anggota* nextAnggota;
};

// Fungsi untuk menambahkan anggota baru
void tambahAnggota(Anggota*& head, string nama, string id) {
    Anggota* anggotaBaru = new Anggota{nama, id, nullptr, head};
    head = anggotaBaru;
}
```



```

// Fungsi untuk menambahkan buku ke anggota
void tambahBuku(Anggota* head, string idAnggota, string judulBuku,
string tanggalPengembalian) {
    while (head && head->idAnggota != idAnggota) {
        head = head->nextAnggota;
    }
    if (head) {
        Buku* bukuBaru = new Buku{judulBuku, tanggalPengembalian,
head->daftarBuku};
        head->daftarBuku = bukuBaru;
    }
}

// Fungsi untuk menghapus anggota beserta buku yang dipinjam
void hapusAnggota(Anggota*& head, string idAnggota) {
    Anggota* prev = nullptr;
    Anggota* curr = head;

    while (curr && curr->idAnggota != idAnggota) {
        prev = curr;
        curr = curr->nextAnggota;
    }

    if (curr) {
        if (prev) {
            prev->nextAnggota = curr->nextAnggota;
        } else {
            head = curr->nextAnggota;
        }
    }

    // Hapus semua buku yang dipinjam anggota
    Buku* buku = curr->daftarBuku;
    while (buku) {
        Buku* temp = buku;
        buku = buku->nextBuku;
        delete temp;
    }

    delete curr;
}

// Fungsi untuk menampilkan data anggota dan buku yang dipinjam
void tampilkanData(Anggota* head) {
    while (head) {

```

```

        cout << "Nama Anggota: " << head->namaAnggota << ", ID: " <<
head->idAnggota << endl;
        Buku* buku = head->daftarBuku;
        while (buku) {
            cout << " - Judul Buku: " << buku->judulBuku << ", Pengembalian:
" << buku->tanggalPengembalian << endl;
            buku = buku->nextBuku;
        }
        head = head->nextAnggota;
    }
}

int main() {
    Anggota* daftarAnggota = nullptr;

    // Tambahkan anggota
    tambahAnggota(daftarAnggota, "Rani", "A001");
    tambahAnggota(daftarAnggota, "Dito", "A002");
    tambahAnggota(daftarAnggota, "Vina", "A003");

    // Tambahkan buku yang dipinjam
    tambahBuku(daftarAnggota, "A001", "Pemrograman C++",
"01/12/2024");
    tambahBuku(daftarAnggota, "A002", "Algoritma Pemrograman",
"15/12/2024");

    // Tambahkan buku baru ke Rani
    tambahBuku(daftarAnggota, "A001", "Struktur Data", "10/12/2024");

    // Hapus anggota Dito beserta buku yang dipinjam
    hapusAnggota(daftarAnggota, "A002");

    // Tampilkan data anggota dan buku
    tampilkanData(daftarAnggota);

    return 0;
}

```

**Output :**

```
C:\Users\USER\Music\13\unguided2\bin\Debug\unguided2.exe
Nama Anggota: Vina, ID: A003
Nama Anggota: Rani, ID: A001
- Judul Buku: Struktur Data, Pengembalian: 10/12/2024
- Judul Buku: Pemrograman C++, Pengembalian: 01/12/2024

Process returned 0 (0x0)   execution time : 0.090 s
Press any key to continue.
```

### Penjelasan :

Program ini mengelola data anggota perpustakaan dan buku yang dipinjam menggunakan Multi-Linked List.

1. Buku: Menyimpan judul buku, tanggal pengembalian, dan pointer ke buku berikutnya.
2. Anggota: Menyimpan nama, ID anggota, pointer ke daftar buku, dan anggota berikutnya.

Fungsi Utama:

1. tambahAnggota: Menambahkan anggota baru ke daftar.
  2. tambahBuku: Menambahkan buku yang dipinjam ke anggota tertentu berdasarkan ID.
  3. hapusAnggota: Menghapus anggota beserta buku yang dipinjam.
- tampilkanData: Menampilkan daftar anggota dan buku yang dipinjam.

Fungsi main():

- Tambahkan anggota: Rani, Dito, Vina.
- Tambahkan buku:
  - Rani: "Pemrograman C++" (01/12/2024), "Struktur Data" (10/12/2024).
  - Dito: "Algoritma Pemrograman" (15/12/2024).
- Hapus Dito beserta bukunya.
- Tampilkan data anggota dan buku yang dipinjam.

## 5. Kesimpulan

Laporan praktikum ini membahas implementasi Multi Linked List sebagai salah satu struktur data untuk mengelola hubungan hirarkis antara data induk dan data anak. Dengan menggunakan bahasa pemrograman C++, berbagai studi kasus dipraktikkan, seperti pengelolaan daftar karyawan dan subordinat, pegawai dengan proyeknya, serta anggota perpustakaan dengan daftar buku yang dipinjam. Operasi dasar seperti penambahan, penghapusan, dan penelusuran elemen dalam Multi Linked List diimplementasikan melalui fungsi-fungsi spesifik yang menunjukkan fleksibilitas struktur ini dalam memetakan relasi kompleks. Hasil dari implementasi menunjukkan bahwa Multi Linked List efektif dalam mengorganisasi data yang saling berhubungan secara hierarkis, sekaligus memastikan manajemen memori yang baik melalui penghapusan elemen-elemen terkait. Struktur ini relevan untuk aplikasi nyata, seperti sistem organisasi dan manajemen data berbasis hirarki.