LAPORAN PRAKTIKUM
Modul 13
“MULTI LINKED LIST”

Disusun Oleh:
Faishal Arif Setiawan -2311104066
Kelas -SE 07 02

Dosen :
Wahyu Andi Saputa 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 .

   Dalam multi-linked list, setiap pointer mengurutkan node berdasarkan kriteria tertentu. Setiap penunjuk dapat mengurutkan list berdasarkan kriteria yang berbeda dan membentuk list yang berbeda.

3. **Guided**

Telkom University Purwokerto

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

using namespace std;


struct Node {
    int data;
    Node* next;
    Node* child;

    Node(int val) : data(val), next(nullptr), child(nullptr) {}
};


class MultiLinkedList {
private:
    Node* head;

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


    void addParent(int data) {
        Node* newNode = new Node(data);
        newNode->next = head;
        head = newNode;
    }


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


    void display() {
        Node* current = head;
        while (current != nullptr) {
            cout << "Parent: " << current->data << " -> ";
            Node* child = current->child;
            while (child != nullptr) {
                cout << child->data << " ";
                child = child->next;
            }
            cout << endl;
            current = current->next;
        }
    }

    ~MultiLinkedList() {

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


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

int main() {
    MultiLinkedList mList;

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

    mList.addChild(1, 10);
    mList.addChild(1, 11);
    mList.addChild(2, 20);
    mList.addChild(2, 20);
    mList.addChild(3, 30);
    mList.addChild(3, 30);
    mList.display();

    return 0;
}
```

Struct Node:

```cpp
struct Node {
    int data;
    Node* next;
    Node* child;

    Node(int val) : data(val), next(nullptr), child(nullptr) {}
};
```

Data: untuk menyimpan nilai dari node
Next:pointer menuju node berikutnya.
Child:pointer menuju child last dari node tersebut.
Kelas Multi Linked List:
Node*head:pointer menuju nodem utama di daftar induk.
Fungsi addParent:

```
void addParent(int data) {
    Node* newNode = new Node(data);
    newNode->next = head;
    head = newNode;
}
```

Menambahkan node induk baru (parent) ke head list.
Node yang baru ditambahkan menjadi *head* dari daftar induk.

Fungsi addchild:

```
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;
    }
}
```

Menambahkan node anak ke dalam daftar child dari node induk yang sesuai (parentData).

Fungsi display:

```
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;
    }
}
```

Menampilkan seluruh **node induk** beserta daftar **node anak**.

Destructor Multi Linked List:

```
    ~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;
        }
    }
};
```

Destructor ini digunakan untuk membebaskan memori yang dialokasikan secara dinamis. Untuk setiap node induk, destructor membebaskan semua child node sebelum membebaskan node induk itu sendiri.

Output:

```
PS D:\struktur data pemograman\GUIDED9\GUIDED10\output> & .\'guided1.exe'
Parent: 3 -> 30 30
Parent: 2 -> 20 20
Parent: 1 -> 11 10
PS D:\struktur data pemograman\GUIDED9\GUIDED10\output>
```

2.

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

using namespace std;

struct EmployeeNode {
    string name;
    EmployeeNode* next;
    EmployeeNode* subordinate;

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


class EmployeeList {
private:
    EmployeeNode* head;

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


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


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


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

    ~EmployeeList() {

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


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

int main() {
    EmployeeList empList;

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

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

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

    empList.display();

    return 0;
}
```

EmployeeNode:

```
struct EmployeeNode {
    string name;
    EmployeeNode* next;
    EmployeeNode* subordinate;

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

name: Menyimpan nama pegawai.
next: Pointer menuju node pegawai berikutnya dalam linked list utama.
subordinate: Pointer menuju linked list bawahannya (subordinate list).
Konstruktor: Menginisialisasi nama pegawai dan menyetel next serta subordinate ke nullptr.
Kelas EmployeeList: Kelas ini mengelola operasi-operasi utama pada struktur multi linked list pegawai.

AddEmployee:

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

Menambahkan pegawai baru ke linked list utama.
Pegawai baru ditambahkan di depan list dengan memindahkan head ke node baru.

addSubordinate:

```
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;
    }
}
```

Menambahkan bawahan (subordinate) untuk seorang manager yang sudah ada.

Display:

```
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;
    }
}
```

Menampilkan seluruh pegawai dalam linked list utama beserta daftar bawahan mereka.

Employeelist:

```
    ~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;
        }
    }
};
```

Menghapus semua node pegawai dan bawahan mereka dari memori untuk mencegah kebocoran memori.

Output:

```
PS D:\struktur data pemograman\GUIDED9\GUIDED10\output> & .\'guided2.exe'
Manager: Charlie -> Brian Frans
Manager: Bob -> Frank
Manager: Alice -> Eve David
PS D:\struktur data pemograman\GUIDED9\GUIDED10\output>
```

3.

Struct Employee:

```
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) {}
};
```

1.

name: Menyimpan nama karyawan.
next: Pointer menuju karyawan berikutnya di linked list utama.
subordinate: Pointer menuju linked list yang berisi bawahan dari karyawan tersebut.

addemployee:

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

Menambahkan karyawan baru ke linked list utama.
Node karyawan baru ditambahkan di depan list (menjadi head).

Addsubordinate:

```
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
        manager->subordinate = newSubordinate; // Memperbarui subordinate
    } else {
        cout << "Manager not found!" << endl;
    }
```

Menambahkan bawahan ke karyawan tertentu (manajer).

Deleteemployee:

```
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 karyawan dari linked list utama.

Deletesubordinate:

```
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 << "." << end
        } else {
            cout << "Subordinate not found!" << endl;
        }
    } else {
        cout << "Manager not found!" << endl;
    }
}
```

Menghapus bawahan dari seorang manajer.

Display:

```
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;
    }
}
```

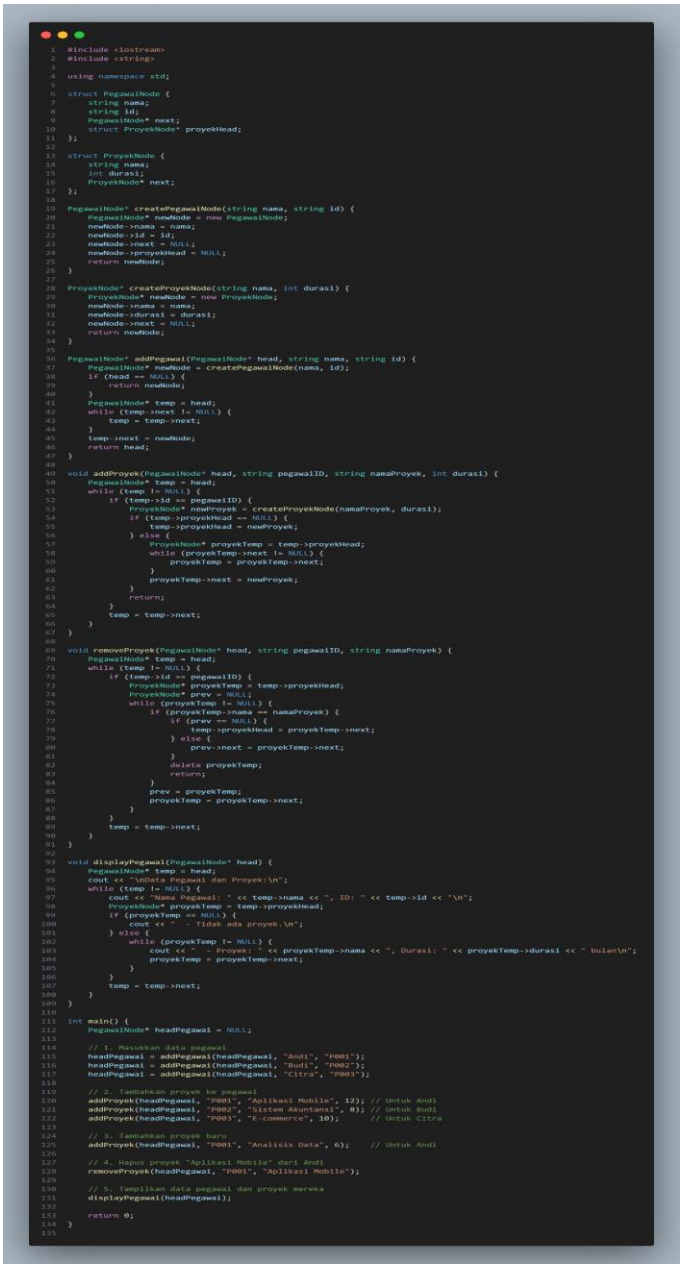
Menampilkan daftar karyawan dan subordinate mereka

Output:

```
Updated employee list:
Manager: Bob -> Frank
Manager: Alice -> Eve
PS D:\struktur data pemograman\GUIDED9\GUIDED10\output>
```

4. **Unguided**
   1.

Telkom University Purwokerto



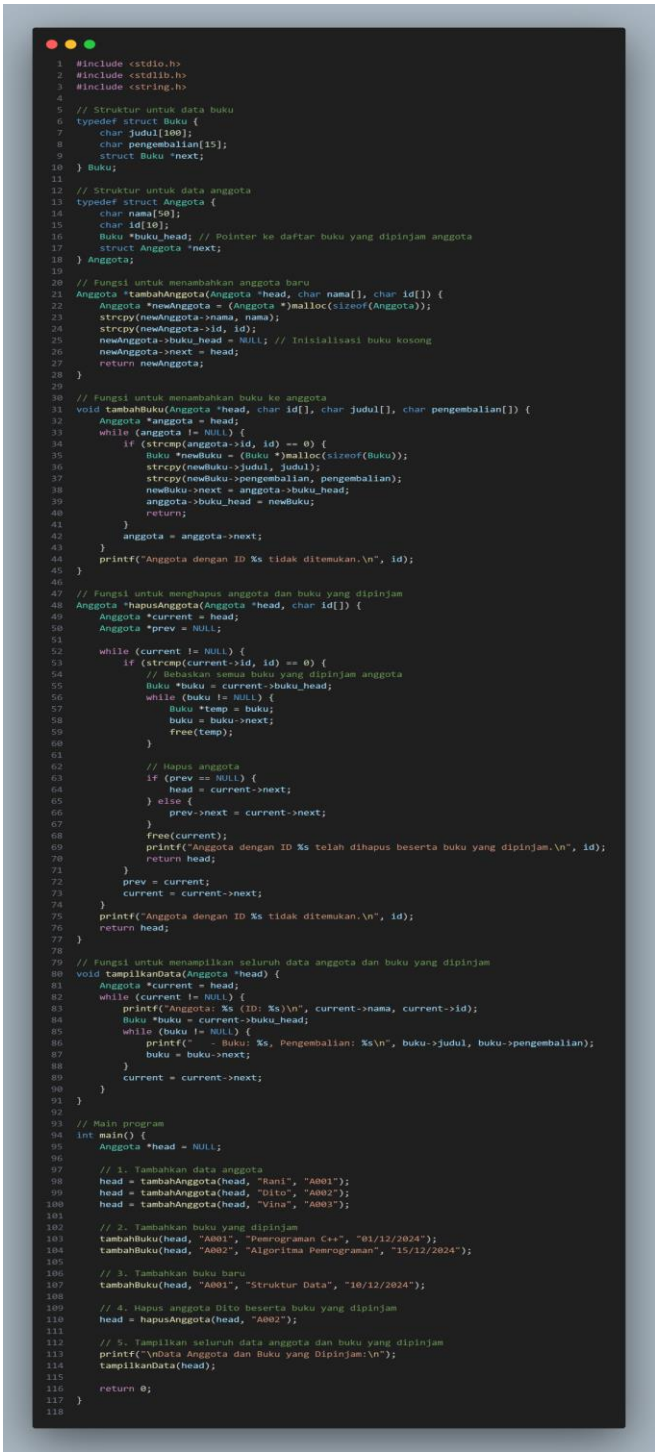
```cpp
#include <iostream>
#include <string>

using namespace std;

struct PegawaiNode {
    string nama;
    string id;
    PegawaiNode* next;
    struct ProyekNode* proyekHead;
};

struct ProyekNode {
    string nama;
    int durasi;
    ProyekNode* next;
};

PegawaiNode* createPegawaiNode(string nama, string id) {
    PegawaiNode* newNode = new PegawaiNode;
    newNode->nama = nama;
    newNode->id = id;
    newNode->next = NULL;
    newNode->proyekHead = NULL;
    return newNode;
}

ProyekNode* createProyekNode(string nama, int durasi) {
    ProyekNode* newNode = new ProyekNode;
    newNode->nama = nama;
    newNode->durasi = durasi;
    newNode->next = NULL;
    return newNode;
}

PegawaiNode* addPegawai(PegawaiNode* head, string nama, string id) {
    PegawaiNode* newNode = createPegawaiNode(nama, id);
    if (head == NULL) {
        return newNode;
    }
    PegawaiNode* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
    return head;
}

void addProyek(PegawaiNode* head, string pegawaiID, string namaProyek, int durasi) {
    PegawaiNode* temp = head;
    while (temp != NULL) {
        if (temp->id == pegawaiID) {
            ProyekNode* newProyek = createProyekNode(namaProyek, durasi);
            if (temp->proyekHead == NULL) {
                temp->proyekHead = newProyek;
            } else {
                ProyekNode* proyekTemp = temp->proyekHead;
                while (proyekTemp->next != NULL) {
                    proyekTemp = proyekTemp->next;
                }
                proyekTemp->next = newProyek;
            }
            return;
        }
        temp = temp->next;
    }
}

void removeProyek(PegawaiNode* head, string pegawaiID, string namaProyek) {
    PegawaiNode* temp = head;
    while (temp != NULL) {
        if (temp->id == pegawaiID) {
            ProyekNode* proyekTemp = temp->proyekHead;
            ProyekNode* prev = NULL;
            while (proyekTemp != NULL) {
                if (proyekTemp->nama == namaProyek) {
                    if (prev == NULL) {
                        temp->proyekHead = proyekTemp->next;
                    } else {
                        prev->next = proyekTemp->next;
                    }
                    delete proyekTemp;
                    return;
                }
                prev = proyekTemp;
                proyekTemp = proyekTemp->next;
            }
        }
        temp = temp->next;
    }
}

void displayPegawai(PegawaiNode* head) {
    PegawaiNode* temp = head;
    cout << "\nData Pegawai dan Proyek:\n";
    while (temp != NULL) {
        cout << "Nama Pegawai: " << temp->nama << ", ID: " << temp->id << "\n";
        ProyekNode* proyekTemp = temp->proyekHead;
        if (proyekTemp == NULL) {
            cout << "  - Tidak ada proyek.\n";
        } else {
            while (proyekTemp != NULL) {
                cout << "  - Proyek: " << proyekTemp->nama << ", Durasi: " << proyekTemp->durasi << " bulan\n";
                proyekTemp = proyekTemp->next;
            }
        }
        temp = temp->next;
    }
}

int main() {
    PegawaiNode* headPegawai = NULL;

    // 1. Masukkan data pegawai
    headPegawai = addPegawai(headPegawai, "Andi", "P001");
    headPegawai = addPegawai(headPegawai, "Budi", "P002");
    headPegawai = addPegawai(headPegawai, "Citra", "P003");

    // 2. Tambahkan proyek ke pegawai
    addProyek(headPegawai, "P001", "Aplikasi Mobile", 12); // Untuk Andi
    addProyek(headPegawai, "P002", "Sistem Akuntansi", 8); // Untuk Budi
    addProyek(headPegawai, "P003", "E-commerce", 10);      // Untuk Citra

    // 3. Tambahkan proyek baru
    addProyek(headPegawai, "P001", "Analisis Data", 6);    // Untuk Andi

    // 4. Hapus proyek "Aplikasi Mobile" dari Andi
    removeProyek(headPegawai, "P001", "Aplikasi Mobile");

    // 5. Tampilkan data pegawai dan proyek mereka
    displayPegawai(headPegawai);

    return 0;
}
```

Output:

```
Data Pegawai dan Proyek:
Nama Pegawai: Andi, ID: P001
  - Proyek: Analisis Data, Durasi: 6 bulan
Nama Pegawai: Budi, ID: P002
  - Proyek: Sistem Akuntansi, Durasi: 8 bulan
Nama Pegawai: Citra, ID: P003
  - Proyek: E-commerce, Durasi: 10 bulan
PS D:\struktur data pemograman\GUIDED9\GUIDED10\output>
```

Telkom University Purwokerto

2.



```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Struktur untuk data buku
typedef struct Buku {
    char judul[100];
    char pengembalian[15];
    struct Buku *next;
} Buku;

// Struktur untuk data anggota
typedef struct Anggota {
    char nama[50];
    char id[10];
    Buku *buku_head; // Pointer ke daftar buku yang dipinjam anggota
    struct Anggota *next;
} Anggota;

// Fungsi untuk menambahkan anggota baru
Anggota *tambahAnggota(Anggota *head, char nama[], char id[]) {
    Anggota *newAnggota = (Anggota *)malloc(sizeof(Anggota));
    strcpy(newAnggota->nama, nama);
    strcpy(newAnggota->id, id);
    newAnggota->buku_head = NULL; // Inisialisasi buku kosong
    newAnggota->next = head;
    return newAnggota;
}

// Fungsi untuk menambahkan buku ke anggota
void tambahBuku(Anggota *head, char id[], char judul[], char pengembalian[]) {
    Anggota *anggota = head;
    while (anggota != NULL) {
        if (strcmp(anggota->id, id) == 0) {
            Buku *newBuku = (Buku *)malloc(sizeof(Buku));
            strcpy(newBuku->judul, judul);
            strcpy(newBuku->pengembalian, pengembalian);
            newBuku->next = anggota->buku_head;
            anggota->buku_head = newBuku;
            return;
        }
        anggota = anggota->next;
    }
    printf("Anggota dengan ID %s tidak ditemukan.\n", id);
}

// Fungsi untuk menghapus anggota dan buku yang dipinjam
Anggota *hapusAnggota(Anggota *head, char id[]) {
    Anggota *current = head;
    Anggota *prev = NULL;

    while (current != NULL) {
        if (strcmp(current->id, id) == 0) {
            // Bebaskan semua buku yang dipinjam anggota
            Buku *buku = current->buku_head;
            while (buku != NULL) {
                Buku *temp = buku;
                buku = buku->next;
                free(temp);
            }

            // Hapus anggota
            if (prev == NULL) {
                head = current->next;
            } else {
                prev->next = current->next;
            }
            free(current);
            printf("Anggota dengan ID %s telah dihapus beserta buku yang dipinjam.\n", id);
            return head;
        }
        prev = current;
        current = current->next;
    }
    printf("Anggota dengan ID %s tidak ditemukan.\n", id);
    return head;
}

// Fungsi untuk menampilkan seluruh data anggota dan buku yang dipinjam
void tampilkanData(Anggota *head) {
    Anggota *current = head;
    while (current != NULL) {
        printf("Anggota: %s (ID: %s)\n", current->nama, current->id);
        Buku *buku = current->buku_head;
        while (buku != NULL) {
            printf("  - Buku: %s, Pengembalian: %s\n", buku->judul, buku->pengembalian);
            buku = buku->next;
        }
        current = current->next;
    }
}

// Main program
int main() {
    Anggota *head = NULL;

    // 1. Tambahkan data anggota
    head = tambahAnggota(head, "Rani", "A001");
    head = tambahAnggota(head, "Dito", "A002");
    head = tambahAnggota(head, "Vina", "A003");

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

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

    // 4. Hapus anggota Dito beserta buku yang dipinjam
    head = hapusAnggota(head, "A002");

    // 5. Tampilkan seluruh data anggota dan buku yang dipinjam
    printf("\nData Anggota dan Buku yang Dipinjam:\n");
    tampilkanData(head);

    return 0;
}
```

Output:

```
Data Anggota dan Buku yang Dipinjam:
Anggota: Vina (ID: A003)
Anggota: Rani (ID: A001)
   - Buku: Struktur Data, Pengembalian: 10/12/2024
   - Buku: Pemrograman C++, Pengembalian: 01/12/2024
PS D:\struktur data pemograman\GUIDED9\GUIDED10\output>
```

5. **Kesimpulan**

praktikum ini tentang Multi Linked List memberikan pemahaman mendalam tentang bagaimana struktur data ini digunakan untuk merepresentasikan hubungan hierarki antara node induk (parent) dan node anak (child). Melalui implementasi fungsi seperti addEmployee, addSubordinate, deleteEmployee, dan deleteSubordinate, kita dapat mengelola data secara efisien, baik untuk menambah, menampilkan, maupun menghapus node pada list. Struktur ini sangat berguna untuk menyelesaikan permasalahan yang melibatkan relasi hierarkis, seperti manajemen data karyawan dan bawahan, serta penerapannya dapat dikembangkan lebih lanjut dalam berbagai studi kasus lain yang membutuhkan keterhubungan antar elemen secara fleksibel dan terorganisir.