LAPORAN PRAKTIKUM
Modul 13
“Multi Linked List”

Disusun Oleh:
Dimastian Aji Wibowo (2311104058)
SE-07-02

Dosen :
Wahyu Andi Saputra, S.Pd., M.Eng.

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

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.

A. Insert

a. Insert Anak

Dalam penambahan elemen anak harus diketahui dulu elemen induknya

```
/* Membuat elemen yang akan disisipkan */
address_anak alokasiAnak(infotypeanak X) {
    address_anak p = alokasi(X);
    next(p) = NULL;
    prev(p) = NULL;
    return p;
}

/* Mencari apakah ada elemen pegawai dengan info X */
address findElm(listinduk L, infotypeinduk X) {
    address cariInduk = head(L);
    do {
        if (cariInduk.info == X) {
            return cariInduk;
        } else {
            cariInduk = next(cariInduk);
        }
    } while (cariInduk.info != X || cariInduk != 
last(L));
    return NULL; // Tambahkan ini jika elemen tidak 
ditemukan
}

/* Menyisipkan anak pada akhir list anak */
void insertLastAnak(listanak &Lanak, address_anak P) {
    if (head(Lanak) == NULL) {
        head(Lanak) = P;
        prev(P) = NULL;
        next(P) = NULL;
```

```
    } else {
        address_anak Q = head(Lanak);
        while (next(Q) != NULL) {
            Q = next(Q);
        }
        next(Q) = P;
        prev(P) = Q;
        next(P) = NULL;
    }
}
```

b. Insert Induk

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

B. Delete

a. Delete Anak

Sama dengan insert anak untuk delete anak maka harus diketahui dulu induknya

b. Delete Induk

Untuk delete elemen induk maka saat di hapus maka seluruh anak dengan induk tersebut juga harus dihapus.

```
/* file : multilist .h */
/* contoh ADT list berkait dengan representasi fisik
pointer */
/* representasi address dengan pointer */
/* info tipe adalah integer */
#ifndef MULTILIST_H_INCLUDED
#define MULTILIST_H_INCLUDED
#include <stdio.h>
#define Nil NULL
#define info(P) (P)->info
#define next(P) (P)->next
#define first(L) ((L).first)
#define last(L) ((L).last)
typedef int infotypeanak;
typedef int infotypeinduk;
typedef struct elemen_list_induk *address;
typedef struct elemen_list_anak *address_anak;

/* define list : */
/* list kosong jika first(L)=Nil
   setiap elemen address P dapat diacu info(P) atau
next(P)
```

```
   elemen terakhir list jika addressnya last, maka
next(last) = Nil */
struct elemen_list_anak {
    /* struct ini untuk menyimpan elemen anak dan
pointer penunjuk
       elemen tetangganya */
    infotypeanak info;
    address_anak next;
    address_anak prev;
};

struct listanak {
    /* struct ini digunakan untuk menyimpan list anak
itu sendiri */
    address_anak first;
    address_anak last;
};

struct elemen_list_induk {
    /* struct ini untuk menyimpan elemen induk dan
pointer penunjuk
       elemen tetangganya */
    infotypeanak info;
    listanak lanak;
    address next;
    address prev;
};

struct listinduk {
    /* struct ini digunakan untuk menyimpan list induk
itu sendiri */
    address first;
    address last;
};

/********* pengecekan apakah list kosong **********/
boolean ListEmpty(listinduk L);
/* mengembalikan nilai true jika list induk kosong */

boolean ListEmptyAnak(listanak L);
/* mengembalikan nilai true jika list anak kosong */

/********* pembuatan list kosong *********/
void CreateList(listinduk &L);
/* I.S. sembarang
   F.S. terbentuk list induk kosong */

void CreateListAnak(listanak &L);
```

```
/* I.S. sembarang
   F.S. terbentuk list anak kosong */

/********* manajemen memori *********/
address alokasi(infotypeinduk P);
/* mengirimkan address dari alokasi sebuah elemen
induk
   jika alokasi berhasil, maka nilai address tidak Nil
dan jika gagal
   nilai address Nil */

address_anak alokasiAnak(infotypeanak P);
/* mengirimkan address dari alokasi sebuah elemen anak
   jika alokasi berhasil, maka nilai address tidak Nil
dan jika gagal
   nilai address_anak Nil */

void dealokasi(address P);
/* I.S. P terdefinisi
   F.S. memori yang digunakan P dikembalikan ke sistem
*/

void dealokasiAnak(address_anak P);
/* I.S. P terdefinisi
   F.S. memori yang digunakan P dikembalikan ke sistem
*/

/********* pencarian sebuah elemen list *********/
address findElm(listinduk L, infotypeinduk X);
/* mencari apakah ada elemen list dengan info(P) = X
   jika ada, mengembalikan address elemen tab tsb, dan
Nil jika sebaliknya */

address_anak findElm(listanak Lanak, infotypeanak X);
/* mencari apakah ada elemen list dengan info(P) = X
   jika ada, mengembalikan address elemen tab tsb, dan
Nil jika sebaliknya */

boolean fFindElm(listinduk L, address P);
/* mencari apakah ada elemen list dengan alamat P
   mengembalikan true jika ada dan false jika tidak
ada */

boolean fFindElmanak(listanak Lanak, address_anak P);
/* mencari apakah ada elemen list dengan alamat P
   mengembalikan true jika ada dan false jika tidak
ada */
```

```
address findBefore(listinduk L, address P);
/* mengembalikan address elemen sebelum P
   jika P berada pada awal list, maka mengembalikan
nilai Nil */

address_anak findBeforeAnak(listanak Lanak,
infotypeinduk X, address_anak P);
/* mengembalikan address elemen sebelum P dimana
info(P) = X
   jika P berada pada awal list, maka mengembalikan
nilai Nil */

/********* penambahan elemen **********/
void insertFirst(listinduk &L, address P);
/* I.S. sembarang, P sudah dialokasikan
   F.S. menempatkan elemen beralamat P pada awal list
*/

#endif // MULTILIST_H_INCLUDED
```

3. Guided

A. Guided 1

1. Membuat struct Node digunakan untuk merepresentasikan setiap elemen dalam multi-linked list dan memiliki int data untuk menyimpan nilai data pada node, Node* next sebagai pointer menuju node berikutnya dalam daftar induk, dan Node* child sebagai pointer menuju daftar anak yang terkait dengan node induk.
2. Membuat class MultiLinkedList dengan variabel private Node* head yang digunakan untuk menyimpan pointer ke node pertama dalam daftar induk.
3. Membuat konstruktor untuk menginisialisasi head dengan nullptr untuk menandakan list kosong.
4. Fungsi addParent(int data) untuk menambahkan node induk baru di awal list dengan membuat node baru dan mengatur next dari node baru ke node lama yang sebelumnya menjadi head, dan mengatur head ke node baru.

```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;
    }
```

5. Fungsi addChild(int parentData, int childData) digunakan untuk menambahkan node anak ke node induk tertentu dengan proses mencari node induk dengan data parentData, jika ditemukan maka buat node anak baru, lalu mengatur next dari node anak ke node anak sebelumnya(jika ada), dan menghubungkan node anak baru sebagai child dari node induk.
6. Fungsi display() digunakan untuk menampilkan isi multi linked list menggunakan perulangan.

```cpp
    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;
        }
    }
};
```

7. Membuat fungsi main() diikuti dengan membuat objek MultiLinkedList bernama mList dan memanggil fungsi addParent(), addChild(), dan display().

```
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. Guided 2

1. Membuat struct EmployeeNode yang digunakan untuk merepresentasikan setiap node dalam daftar pegawai diikuti dengan string name untuk menyimpan nama pegawai, EmployeeNode* next sebagai pointer menuju node pegawai berikutnya, dan EmployeeNode* subordinate sebagai pointer menuju daftar bawahan.
2. Membuat konstruktor yang akan menginisialisasi name, next, dan subordinate ketika objek baru dibuat.
3. Membuat class EmployeeList dengan variabel private EmployeeNode* head digunakan untuk menunjuk ke node pertama dalam daftar pegawai, membuat konstruktor untuk class EmployeeList untuk mengatur head menjadi nullptr untuk menunjukan bahwa daftar pegawai masih kosong.
4. Fungsi addEmployee(string name) digunakan untuk menambahkan pegawai baru ke daftar induk dengan proses membuat node pegawai baru dan node pegawai baru akan menjadi head dari daftar dan menunjuk ke node pegawai sebelumnya.
5. Fungsi addSubordinate(string managerName, string subordinateName) digunakan untuk menambahkan bawahan (subordinate) ke pegawai tertentu dengan melakukan perulangan pada daftar utama untuk mencari pegawai dengan nama managerName, jika ditemukan maka node baru dibuat untuk bawahan dan ditambahkan di awal daftar bawahan, dan jika pegawai dengan nama tersebut tidak ditemukan maka menampilkan pesan manager tidak ditemukan.

```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;
        }
    }
```

6. Fungsi display() digunakan untuk menampilkan seluruh daftar pegawai beserta bawahannya dengan melakukan perulangan pada daftar induk dengan proses untuk setiap pegawai maka dilakukan perulangan pada daftar bawahannya menggunakan pointer subordinate, dan menampilkan nama manager beserta nama bawahan.

```cpp
    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;
        }
    }
};
```

7. Membuat fungsi main() dengan membuat objek EmployeeList bernama emplist dan memanggil fungsi addEmployee(), addSubordinate, dan display().

```cpp
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;
}
```

C. Guided 3

1. Sama seperti kode sebelumnya tapi dengan penambahan fungsi deleteEmployee(string name) yang digunakan untuk menghapus seorang karyawan beserta subordinatenya dari daftar karyawan dengan proses menggunakan perulangan untuk mencari karyawan yang ingin dihapus, jika karyawan ditemukan maka node karyawan akan dihapus dan menghapus semua subordinate dari node karyawwan tersebut, jika karyawan tidak ditemukan maka menampilkan pesan karyawan tidak ditemukan.
2. Fungsi deleteSubordinate(string managerName, string subordinateName) untuk menghapus subordinate dari seorang manajer tertentu dengan menggunakan perulangan untuk mencari manager jika ditemukan maka pointer manager akan menunjuk ke node manager tersebut, jika manager ditemukan maka cari subordinate yang ingin dihapus menggunakan perulangan, jika subordinate ditemukan maka hapus node subordinate dari daftar, jika subordinate tidak ditemukan maka menampilkan pesan subordinate tidak ditemukan, dan jika manager tidak ditemukan maka menampilkan pesan manager tidak ditemukan.

```cpp
#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;
}
```

3. Membuat fungsi main() dengan membuat objek EmployeeList bernama emplist dan memanggil fungsi addEmployee(), addSubordinate, display(), deleteSubordinate(), dan deleteManager().

4. Unguided

A. Unguided 1

1. Mendefinisikan struct NodeProyek yang digunakan untuk menyimpan data proyek dengan atribut string namaProyek untuk menyimpan nama proyek, int durasi untuk menyimpan durasi proyek dalam bulan, dan pointer next yang menunjuk ke proyek berikutnya.
2. Mendefinisikan struct namaPegawai untuk menyimpan data pegawai dengan atribut string namaPegawai untuk menyimpan nama pegawai, string id pegawai, pointer headProyek yang menunjuk ke proyek pertama pegawai tersebut, dan pointer next yang menunjuk ke pegawai berikutnya.
3. Membuat class MultiLinkedList untuk mengelola daftar pegawai dan proyek yang terkait dengan setiap pegawai dan kelas ini terdapat pointer headPegawai sebagai pointer utama yang menunjuk ke daftar pegawai.
4. Fungsi tambahPegawai() untuk menambahkan pegawai baru ke dalam daftar dengan membuat NodePegawai baru, menyambungkan objek NodePegawai baru ke dalam daftar pegawai dengan cara menjadikannya sebagai headPegawai dan sebelumnya menunjuk ke pegawai yang sudah ada di daftar (jika ada).
5. Fungsi tambahProyek() untuk menambahkan proyek baru ke dalam daftar proyek pegawai berdasarkan ID pegawai dengan proses memanggil fungsi cariPegawai() untuk mencari pegawai yang memiliki ID yang sesuai dengan parameter idPegawai, jika pegawai ditemukan maka sebuah objek NodeProyek baru dibuat untuk proyek baru dengan nama proyek dan durasi yang diberikan, lalu proyek baru ditambahkan ke dalam daftar proyek pegawai dengan cara menyambungkan objek NodeProyek baru ini ke proyek yang sudah ada, jika proyek baru menjadi proyek pertama jika daftar proyek pegawai masih kosong, dan Jika pegawai dengan ID tersebut tidak ditemukan maka pesan ID pegawai tidak ditemukan.

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

struct NodeProyek {
    string namaProyek;
    int durasi;
    NodeProyek* next;

    NodeProyek(string nama, int dur) : namaProyek(nama), durasi(dur), next(nullptr) {}
};
struct NodePegawai {
    string namaPegawai;
    string idPegawai;
    NodeProyek* headProyek;
    NodePegawai* next;

    NodePegawai(string nama, string id) : namaPegawai(nama), idPegawai(id), headProyek(nullptr), next(nullptr) {}
};
class MultiLinkedList {
private:
    NodePegawai* headPegawai;

public:
    MultiLinkedList() : headPegawai(nullptr) {}
    void tambahPegawai(string nama, string id) {
        NodePegawai* pegawaiBaru = new NodePegawai(nama, id);
        pegawaiBaru->next = headPegawai;
        headPegawai = pegawaiBaru;
    }
    void tambahProyek(string idPegawai, string namaProyek, int durasi) {
        NodePegawai* pegawai = cariPegawai(idPegawai);
        if (pegawai) {
            NodeProyek* proyekBaru = new NodeProyek(namaProyek, durasi);
            proyekBaru->next = pegawai->headProyek;
            pegawai->headProyek = proyekBaru;
        } else {
            cout << "Pegawai dengan ID " << idPegawai << " tidak ditemukan.\n";
        }
    }
```

6. Fungsi hapusProyek() untuk menghapus proyek tertentu dari daftar proyek pegawai berdasarkan nama proyek dengan proses memanggil fungsi cariPegawai() untuk mencari pegawai berdasarkan ID pegawai, jika pegawai ditemukan maka program akan menelusuri daftar proyek pegawai untuk menemukan proyek dengan nama yang sesuai, jika proyek ditemukan maka proyek tersebut akan dihapus dengan mengubah pointer next dari proyek sebelumnya untuk melewatkan proyek yang ingin dihapus, jika proyek yang dihapus adalah proyek pertama maka headProyek pegawai diubah untuk menunjuk ke proyek berikutnya dan jika tidak ditemukan maka menampilkan pesan proyek tidak ditemukan.
7. Fungsi tampilkanData() untuk menampilkan semua data pegawai beserta proyek-proyek yang dimiliki oleh masing-masing pegawai dengan proses menelusuri daftar pegawai satu per satu mulai dari headPegawai menggunakan perulangan, dan untuk setiap pegawai menampilkan nama pegawai dan ID pegawai, kemudian daftar proyek pegawai ditelusuri dan untuk setiap proyek, nama proyek dan durasinya ditampilkan. Jika pegawai tidak memiliki proyek, maka pesan "Tidak ada proyek" akan ditampilkan.

```
    void hapusProyek(string idPegawai, string namaProyek) {
        NodePegawai* pegawai = cariPegawai(idPegawai);
        if (pegawai) {
            NodeProyek* prev = nullptr;
            NodeProyek* curr = pegawai->headProyek;
            while (curr) {
                if (curr->namaProyek == namaProyek) {
                    if (prev) {
                        prev->next = curr->next;
                    } else {
                        pegawai->headProyek = curr->next;
                    }
                    delete curr;
                    cout << "Proyek " << namaProyek << " berhasil dihapus dari pegawai " << pegawai->namaPegawai << ".\n";
                    return;
                }
                prev = curr;
                curr = curr->next;
            }
            cout << "Proyek " << namaProyek << " tidak ditemukan pada pegawai " << pegawai->namaPegawai << ".\n";
        } else {
            cout << "Pegawai dengan ID " << idPegawai << " tidak ditemukan.\n";
        }
    }
    void tampilkanData() {
        NodePegawai* pegawai = headPegawai;
        while (pegawai) {
            cout << "Pegawai: " << pegawai->namaPegawai << " (ID: " << pegawai->idPegawai << ")\n";
            NodeProyek* proyek = pegawai->headProyek;
            if (!proyek) {
                cout << "  Tidak ada proyek.\n";
            }
            while (proyek) {
                cout << "  - Proyek: " << proyek->namaProyek << ", Durasi: " << proyek->durasi << " bulan\n";
                proyek = proyek->next;
            }
            pegawai = pegawai->next;
        }
    }
```

8. Mendefinisikan metode cariPegawai() yang digunakan untuk mencari pegawai berdasarkan ID pegawai dengan mengembalikan pointer ke pegawai yang ditemukan atau nullptr jika pegawai dengan ID tersebut tidak ada.
9. Fungsi main() yang menjalankan program dengan memanggil metode tambahPegawai() untuk menambahkan beberapa pegawai, tambahProyek() untuk menambahkan proyek-proyek ke pegawai yang sesuai, hapusProyek() untuk menghapus proyek tertentu dari pegawai, dan tampilkanData() untuk menampilkan seluruh data pegawai beserta proyek-proyek yang dimilikinya.

```
private:
    NodePegawai* cariPegawai(string id) {
        NodePegawai* curr = headPegawai;
        while (curr) {
            if (curr->idPegawai == id) {
                return curr;
            }
            curr = curr->next;
        }
        return nullptr;
    }
};
int main() {
    MultiLinkedList daftarPegawai;

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

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

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

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

    cout << "\nData Pegawai dan Proyek:\n";
    daftarPegawai.tampilkanData();

    return 0;
}
```

10. Program ini memanfaatkan struktur data linked list ganda, di mana setiap pegawai memiliki daftar proyek yang dapat ditambahkan atau dihapus. Dengan menggunakan struktur ini, program memungkinkan penanganan data pegawai dan proyek yang efisien dan fleksibel.

11. Berikut merupakan output dari program tersebut.

```
Data Pegawai dan Proyek:
Pegawai: Citra (ID: P003)
  - Proyek: E-commerce, Durasi: 10 bulan
Pegawai: Budi (ID: P002)
  - Proyek: Sistem Akuntansi, Durasi: 8 bulan
Pegawai: Andi (ID: P001)
  - Proyek: Analisis Data, Durasi: 6 bulan
```

B. Unguided 2

1. Mendefinisikan struktur NodeBuku yang memiliki tiga atribut, yaitu judulBuku untuk menyimpan judul buku, tanggalPengembalian untuk menyimpan tanggal pengembalian buku, dan next yang berfungsi sebagai pointer untuk menunjuk ke buku berikutnya dalam daftar buku yang dipinjam oleh anggota. Konstruktor NodeBuku digunakan untuk menginisialisasi judul buku, tanggal pengembalian, dan mengatur next menjadi nullptr.
2. Mendefinisikan struktur NodeAnggota yang memiliki empat atribut, yaitu namaAnggota untuk menyimpan nama anggota, idAnggota untuk menyimpan ID anggota, headBuku yang berfungsi sebagai pointer untuk menunjuk ke buku pertama yang dipinjam oleh anggota, dan next yang berfungsi sebagai pointer untuk menunjuk ke anggota berikutnya dalam daftar anggota. Konstruktor NodeAnggota digunakan untuk menginisialisasi nama anggota, ID anggota, dan mengatur headBuku serta next menjadi nullptr.
3. Mendefinisikan kelas MultiLinkedList yang memiliki atribut headAnggota sebagai pointer yang menunjuk ke anggota pertama dalam daftar anggota.
4. Fungsi tambahAnggota() untuk menambahkan anggota baru ke dalam daftar anggota.
5. Fungsi tambahBuku() untuk menambahkan buku baru yang dipinjam oleh anggota berdasarkan ID anggota.

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

struct NodeBuku {
    string judulBuku;
    string tanggalPengembalian;
    NodeBuku* next;

    NodeBuku(string judul, string tanggal) : judulBuku(judul), tanggalPengembalian(tanggal), next(nullptr) {}
};
struct NodeAnggota {
    string namaAnggota;
    string idAnggota;
    NodeBuku* headBuku;
    NodeAnggota* next;

    NodeAnggota(string nama, string id) : namaAnggota(nama), idAnggota(id), headBuku(nullptr), next(nullptr) {}
};

class MultiLinkedList {
private:
    NodeAnggota* headAnggota;
public:
    MultiLinkedList() : headAnggota(nullptr) {}
    void tambahAnggota(string nama, string id) {
        NodeAnggota* anggotaBaru = new NodeAnggota(nama, id);
        anggotaBaru->next = headAnggota;
        headAnggota = anggotaBaru;
    }
    void tambahBuku(string idAnggota, string judulBuku, string tanggalPengembalian) {
        NodeAnggota* anggota = cariAnggota(idAnggota);
        if (anggota) {
            NodeBuku* bukuBaru = new NodeBuku(judulBuku, tanggalPengembalian);
            bukuBaru->next = anggota->headBuku;
            anggota->headBuku = bukuBaru;
        } else {
            cout << "Anggota dengan ID " << idAnggota << " tidak ditemukan.\n";
        }
    }
```

6. Fungsi hapusAnggota() untuk menghapus anggota tertentu dari daftar anggota, beserta buku yang dipinjam oleh anggota tersebut.
7. Fungsi tampilkanData() untuk menampilkan seluruh data anggota beserta buku-buku yang dipinjam oleh masing-masing anggota.

```cpp
    void hapusAnggota(string idAnggota) {
        NodeAnggota* prev = nullptr;
        NodeAnggota* curr = headAnggota;
        while (curr) {
            if (curr->idAnggota == idAnggota) {
                hapusSemuaBuku(curr);
                if (prev) {
                    prev->next = curr->next;
                } else {
                    headAnggota = curr->next;
                }
                delete curr;
                cout << "Anggota dengan ID " << idAnggota << " berhasil dihapus beserta buku yang dipinjam.\n";
                return;
            }
            prev = curr;
            curr = curr->next;
        }
        cout << "Anggota dengan ID " << idAnggota << " tidak ditemukan.\n";
    }
    void tampilkanData() {
        NodeAnggota* anggota = headAnggota;
        while (anggota) {
            cout << "Anggota: " << anggota->namaAnggota << " (ID: " << anggota->idAnggota << ")\n";
            NodeBuku* buku = anggota->headBuku;
            if (!buku) {
                cout << "  Tidak ada buku yang dipinjam.\n";
            }
            while (buku) {
                cout << "  - Buku: " << buku->judulBuku << ", Pengembalian: " << buku->tanggalPengembalian << "\n";
                buku = buku->next;
            }
            anggota = anggota->next;
        }
    }
```

8. Mendefinisikan metode cariAnggota() yang digunakan untuk mencari anggota berdasarkan ID anggota. Metode ini mengembalikan pointer ke anggota yang ditemukan atau nullptr jika anggota dengan ID tersebut tidak ada.
9. Mendefinisikan metode hapusSemuaBuku() yang digunakan untuk menghapus seluruh buku yang dipinjam oleh anggota tertentu. Metode ini memastikan bahwa sebelum menghapus anggota, seluruh buku yang dipinjam anggota tersebut akan dihapus terlebih dahulu.
10. Fungsi main() yang menjalankan program dengan memanggil metode

tambahAnggota() untuk menambahkan beberapa anggota, tambahBuku() untuk menambahkan buku-buku yang dipinjam oleh anggota yang sesuai, hapusAnggota() untuk menghapus anggota tertentu bersama dengan buku yang dipinjamnya, dan tampilkanData() untuk menampilkan seluruh data anggota beserta buku-buku yang dipinjam.

11. Program ini memanfaatkan struktur data linked list ganda, di mana setiap anggota memiliki daftar buku yang dipinjam. Dengan menggunakan struktur ini, program memungkinkan penanganan data anggota dan buku yang efisien serta fleksibel, serta memungkinkan penghapusan anggota beserta seluruh buku yang dipinjamnya dengan aman.

```
private:
    NodeAnggota* cariAnggota(string id) {
        NodeAnggota* curr = headAnggota;
        while (curr) {
            if (curr->idAnggota == id) {
                return curr;
            }
            curr = curr->next;
        }
        return nullptr;
    }
    void hapusSemuaBuku(NodeAnggota* anggota) {
        NodeBuku* curr = anggota->headBuku;
        while (curr) {
            NodeBuku* temp = curr;
            curr = curr->next;
            delete temp;
        }
        anggota->headBuku = nullptr;
    }
};

int main() {
    MultiLinkedList daftarAnggota;

    daftarAnggota.tambahAnggota("Rani", "A001");
    daftarAnggota.tambahAnggota("Dito", "A002");
    daftarAnggota.tambahAnggota("Vina", "A003");

    daftarAnggota.tambahBuku("A001", "Pemrograman C++", "01/12/2024");
    daftarAnggota.tambahBuku("A002", "Algoritma Pemrograman", "15/12/2024");

    daftarAnggota.tambahBuku("A001", "Struktur Data", "10/12/2024");

    daftarAnggota.hapusAnggota("A002");

    cout << "\nData Anggota dan Buku yang Dipinjam:\n";
    daftarAnggota.tampilkanData();

    return 0;
}
```

12. Berikut merupakan output dari kode tersebut.

```
Data Anggota dan Buku yang Dipinjam:
Anggota: Vina (ID: A003)
  Tidak ada buku yang dipinjam.
Anggota: Rani (ID: A001)
  - Buku: Struktur Data, Pengembalian: 10/12/2024
  - Buku: Pemrograman C++, Pengembalian: 01/12/2024
```

5. Kesimpulan

Multi linked list dapat memberikan pemahaman yang lebih mendalam mengenai pengelolaan memori dinamis serta hubungan antar elemen yang saling terhubung dalam struktur data yang lebih kompleks. Dengan menggunakan konsep pointer dan alokasi memori dinamis, multi linked list

memungkinkan pembuatan struktur data yang lebih fleksibel dan efisien dalam menyimpan data dengan relasi yang lebih banyak. Praktikum ini juga mengajarkan pentingnya pemahaman terkait manipulasi pointer, pengelolaan memori, serta teknik traversal yang efektif untuk mengakses data dalam struktur linked list yang lebih kompleks.