LAPORAN PRAKTIKUM
Modul 13
“MULTI LINKED LIST”

Disusun Oleh:
Aji Prasetyo Nugroho - 2211104049
S1SE-07-2

Assisten Praktikum :
Aldi Putra
Andini Nur Hidayah

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024

A. Tujuan

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

B. Landasan Teori

1. Definisi Multi Linked List

   Multi Linked List adalah struktur data yang merupakan pengembangan dari konsep linked list biasa. Dalam multi linked list, setiap node dalam linked list dapat memiliki lebih dari satu pointer yang menunjuk ke node lainnya, sehingga memungkinkan penyimpanan data yang lebih kompleks, seperti relasi antar data yang memiliki sifat hierarkis atau terhubung ke beberapa elemen lainnya.

   Struktur ini sering digunakan untuk merepresentasikan hubungan many-to-many antar data. Misalnya, pada sistem perpustakaan, data anggota dan buku yang dipinjam dapat direpresentasikan dengan multi linked list, di mana setiap anggota memiliki daftar buku yang dipinjam.

2. Komponen Multi Linked List

   1) Node Utama (Parent Node): Node utama berisi informasi utama (misalnya data anggota) dan pointer ke node lainnya.
   2) Node Anak (Child Node): Node ini merepresentasikan entitas yang terkait dengan node utama (misalnya data buku yang dipinjam oleh anggota).
   3) Pointer: Pointer adalah bagian dari node yang menunjuk ke node berikutnya. Dalam multi linked list, sebuah node dapat memiliki beberapa pointer:
      - Pointer ke node anak.
      - Pointer ke node berikutnya dalam daftar utama.

3. Kelebihan Multi Linked List

   1) Representasi Hubungan Kompleks: Multi linked list mampu merepresentasikan hubungan antar data yang kompleks, seperti hubungan many-to-many.
   2) Fleksibilitas: Memungkinkan penyimpanan data dalam struktur yang fleksibel tanpa perlu ukuran tetap (dynamic size).
   3) Efisiensi dalam Pengelolaan Data Terkait: Memudahkan akses dan manipulasi data yang saling terhubung (seperti menambah, menghapus, atau menampilkan data terkait).

4. Kekurangan Multi Linked List
    1) Kompleksitas Implementasi: Dibandingkan dengan single linked list, implementasi multi linked list lebih rumit karena melibatkan banyak pointer.
    2) Overhead Memori: Karena setiap node membutuhkan lebih dari satu pointer, penggunaan memori cenderung lebih tinggi.
    3) Kesalahan Pointer: Pengelolaan pointer yang tidak tepat dapat menyebabkan kesalahan seperti dangling pointer atau memory leaks.
5. Contoh Kasus Penggunaan Multi Linked List
    1) Sistem Perpustakaan:
        - Node utama menyimpan data anggota perpustakaan.
        - Node anak menyimpan data buku yang dipinjam oleh anggota tersebut.
    2) Sistem Manajemen Proyek:
        - Node utama menyimpan data proyek.
        - Node anak menyimpan daftar tugas atau anggota tim yang terlibat dalam proyek tersebut.
    3) Sistem Akademik:
        - Node utama menyimpan data mahasiswa.
        - Node anak menyimpan daftar mata kuliah yang diambil oleh mahasiswa tersebut.

C. Guided

- Guided 1

  Source 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 :

```
Parent: 3 -> 30 30
Parent: 2 -> 20 20
Parent: 1 -> 11 10
PS D:\Praktikum STD_2211104049>
```

- Guided 2

Source 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 :

```
Manager: Charlie -> Brian Frans
Manager: Bob -> Frank
Manager: Alice -> Eve David
PS D:\Praktikum STD_2211104049>
```

- Guided 3

Source 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 :

```
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:\Praktikum STD_2211104049>
```

D. Unguided

```
1. Manajemen Data Pegawai dan Proyek
Buatlah program menggunakan Multi Linked List untuk menyimpan data pegawai dan
proyek yang dikelola setiap pegawai.
- Setiap pegawai memiliki data: Nama Pegawai dan ID Pegawai.
- Setiap proyek memiliki data: Nama Proyek** dan **Durasi (bulan).

Instruksi:
1. Masukkan data pegawai berikut:
   - Pegawai 1: Nama = "Andi", ID = "P001".
   - Pegawai 2: Nama = "Budi", ID = "P002".
   - Pegawai 3: Nama = "Citra", ID = "P003".
2. Tambahkan proyek ke pegawai:
   - Proyek 1: Nama = "Aplikasi Mobile", Durasi = 12 bulan (Untuk Andi).
   - Proyek 2: Nama = "Sistem Akuntansi", Durasi = 8 bulan (Untuk Budi).
   - Proyek 3: Nama = "E-commerce", Durasi = 10 bulan (Untuk Citra).
3. Tambahkan proyek baru:
   - Proyek 4: Nama = "Analisis Data", Durasi = 6 bulan (Untuk Andi).
4. Hapus proyek "Aplikasi Mobile" dari Andi.
5. Tampilkan data pegawai dan proyek mereka.


2. Sistem Manajemen Buku Perpustakaan
Gunakan Multi Linked List untuk menyimpan data anggota perpustakaan dan daftar buku
yang dipinjam.
- Setiap anggota memiliki data: Nama Anggota dan ID Anggota.
- Setiap buku memiliki data: Judul Buku dan Tanggal Pengembalian.

Instruksi:
1. Masukkan data anggota berikut:
   - Anggota 1: Nama = "Rani", ID = "A001".
   - Anggota 2: Nama = "Dito", ID = "A002".
   - Anggota 3: Nama = "Vina", ID = "A003".
2. Tambahkan buku yang dipinjam:
   - Buku 1: Judul = "Pemrograman C++", Pengembalian = "01/12/2024" (Untuk Rani).
   - Buku 2: Judul = "Algoritma Pemrograman", Pengembalian = "15/12/2024" (Untuk
Dito).
3. Tambahkan buku baru:
   - Buku 3: Judul = "Struktur Data", Pengembalian = "10/12/2024" (Untuk Rani).
4. Hapus anggota Dito beserta buku yang dipinjam.
5. Tampilkan seluruh data anggota dan buku yang dipinjam.
```

- **Unguided 1**

  Source Code :

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

struct Proyek {
    string namaProyek;
    int durasi;
    Proyek* next;
};
```

```
struct Pegawai {
    string namaPegawai;
    string idPegawai;
    Proyek* proyekHead;
    Pegawai* next;
};

class MultiLinkedList {
private:
    Pegawai* head;

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

    void tambahPegawai(const string& nama, const string& id) {
        Pegawai* pegawaiBaru = new Pegawai{nama, id, nullptr, head};
        head = pegawaiBaru;
    }

    void tambahProyek(const string& idPegawai, const string& namaProyek, int
durasi) {
        Pegawai* pegawai = cariPegawai(idPegawai);
        if (pegawai) {
            Proyek* proyekBaru = new Proyek{namaProyek, durasi, pegawai-
>proyekHead};
            pegawai->proyekHead = proyekBaru;
        } else {
            cout << "Pegawai dengan ID " << idPegawai << " tidak
ditemukan.\n";
        }
    }

    void hapusProyek(const string& idPegawai, const string& namaProyek) {
        Pegawai* pegawai = cariPegawai(idPegawai);
        if (pegawai) {
            Proyek* current = pegawai->proyekHead;
            Proyek* prev = nullptr;
            while (current && current->namaProyek != namaProyek) {
                prev = current;
                current = current->next;
            }
            if (current) {
                if (prev) {
                    prev->next = current->next;
                } else {
                    pegawai->proyekHead = current->next;
                }
                delete current;
```

```
                cout << "Proyek " << namaProyek << " berhasil dihapus.\n";
            } else {
                cout << "Proyek " << namaProyek << " tidak ditemukan pada
pegawai " << pegawai->namaPegawai << ".\n";
            }
        } else {
            cout << "Pegawai dengan ID " << idPegawai << " tidak
ditemukan.\n";
        }
    }

    void tampilkanData() {
        Pegawai* currentPegawai = head;
        while (currentPegawai) {
            cout << "Pegawai: " << currentPegawai->namaPegawai << " (ID: " <<
currentPegawai->idPegawai << ")\n";
            Proyek* currentProyek = currentPegawai->proyekHead;
            while (currentProyek) {
                cout << "  - Proyek: " << currentProyek->namaProyek << ",
Durasi: " << currentProyek->durasi << " bulan\n";
                currentProyek = currentProyek->next;
            }
            currentPegawai = currentPegawai->next;
            cout << endl;
        }
    }

private:
    Pegawai* cariPegawai(const string& idPegawai) {
        Pegawai* current = head;
        while (current) {
            if (current->idPegawai == idPegawai) {
                return current;
            }
            current = current->next;
        }
        return nullptr;
    }
};

int main() {
    MultiLinkedList mll;

    // Menambahkan data pegawai
    mll.tambahPegawai("Andi", "P001");
    mll.tambahPegawai("Budi", "P002");
    mll.tambahPegawai("Citra", "P003");
```

```
    // Menambahkan proyek ke pegawai
    mll.tambahProyek("P001", "Aplikasi Mobile", 12);
    mll.tambahProyek("P002", "Sistem Akuntansi", 8);
    mll.tambahProyek("P003", "E-commerce", 10);

    // Menambahkan proyek baru
    mll.tambahProyek("P001", "Analisis Data", 6);

    // Menghapus proyek
    mll.hapusProyek("P001", "Aplikasi Mobile");

    // Menampilkan data pegawai dan proyek
    mll.tampilkanData();

    return 0;
}
```

Output :

```
Proyek Aplikasi Mobile berhasil dihapus.
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

PS D:\Praktikum STD_2211104049>
```

- **Unguided 2**

  Source code :

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

// Struktur data untuk Buku
struct Book {
    string title;
    string returnDate;
};

// Struktur data untuk Anggota
struct Member {
```

```
    string name;
    string id;
    list<Book> borrowedBooks;
};

// Fungsi untuk menampilkan data anggota dan buku yang dipinjam
void displayMembers(const list<Member>& members) {
    for (const auto& member : members) {
        cout << "Nama Anggota: " << member.name << ", ID: " << member.id <<
endl;
        if (!member.borrowedBooks.empty()) {
            cout << "  Buku yang dipinjam:" << endl;
            for (const auto& book : member.borrowedBooks) {
                cout << "    Judul: " << book.title << ", Tanggal
Pengembalian: " << book.returnDate << endl;
            }
        } else {
            cout << "  Tidak ada buku yang dipinjam." << endl;
        }
        cout << endl;
    }
}

// Fungsi untuk menghapus anggota beserta buku yang dipinjam
void removeMember(list<Member>& members, const string& memberId) {
    members.remove_if([&](const Member& member) {
        return member.id == memberId;
    });
}

int main() {
    list<Member> members;

    // 1. Tambahkan data anggota
    members.push_back({"Rani", "A001"});
    members.push_back({"Dito", "A002"});
    members.push_back({"Vina", "A003"});

    // 2. Tambahkan buku yang dipinjam
    for (auto& member : members) {
        if (member.id == "A001") {
            member.borrowedBooks.push_back({"Pemrograman C++", "01/12/2024"});
        } else if (member.id == "A002") {
            member.borrowedBooks.push_back({"Algoritma Pemrograman",
"15/12/2024"});
        }
    }
```

```
    // 3. Tambahkan buku baru untuk Rani
    for (auto& member : members) {
        if (member.id == "A001") {
            member.borrowedBooks.push_back({"Struktur Data", "10/12/2024"});
        }
    }

    // 4. Hapus anggota Dito beserta buku yang dipinjam
    removeMember(members, "A002");

    // 5. Tampilkan seluruh data anggota dan buku yang dipinjam
    displayMembers(members);

    return 0;
}
```

Output :

```
Nama Anggota: Rani, ID: A001
  Buku yang dipinjam:
    Judul: Pemrograman C++, Tanggal Pengembalian: 01/12/2024
    Judul: Struktur Data, Tanggal Pengembalian: 10/12/2024

Nama Anggota: Vina, ID: A003
  Tidak ada buku yang dipinjam.

PS D:\Praktikum STD_2211104049>
```