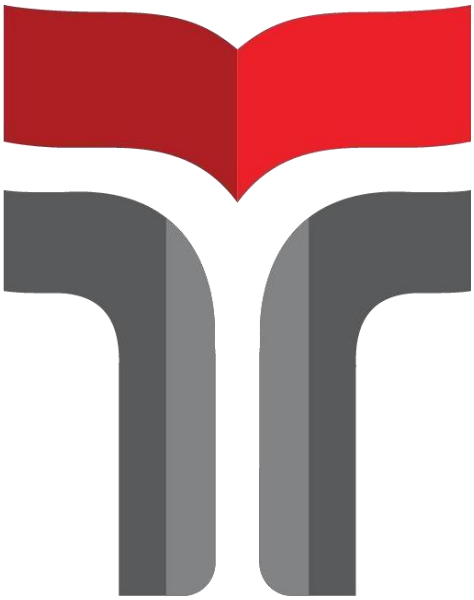
LAPORAN PRAKTIKUM
STRUKTUR DATA 13
"MULTI LINKED LIST"



Oleh:

NAMA: Ammar Dzaki Nandana

NIM: 2311104071

KELAS: SE 07 02

DOSEN: Wahyu Andi Saputra

PRODI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

INSTITUT TEKNOLOGI TELKOM PURWOKERTO

2023/2024

I. TUJUAN

· Memahami konsep dasar dan prinsip kerja *Multi Linked List* sebagai salah satu struktur data lanjutan.
· Mempelajari perbedaan antara *Multi Linked List* dengan struktur data lain, seperti *Single Linked List*, *Double Linked List*, dan *Circular Linked List*.
· Mengimplementasikan *Multi Linked List* dalam program menggunakan bahasa pemrograman, seperti C++, Python, atau Java.
· Memahami cara kerja setiap operasi dasar pada *Multi Linked List*, seperti penambahan, penghapusan, dan penelusuran node.
· Mengetahui cara merepresentasikan hubungan antar data yang kompleks menggunakan *Multi Linked List*.
· Mengembangkan keterampilan untuk memetakan masalah dunia nyata ke dalam struktur *Multi Linked List*.
· Mengidentifikasi kelebihan dan kekurangan *Multi Linked List* dalam aplikasi praktis dibandingkan dengan struktur data lainnya.
· Memahami konsep pointer dan penggunaannya dalam membangun relasi kompleks antar node dalam *Multi Linked List*.
· Menganalisis efisiensi waktu dan memori dalam implementasi dan penggunaan *Multi Linked List*.
· Mempraktikkan desain struktur data yang efisien dan modular dalam pembuatan program berbasis *Multi Linked List*.
· Mempelajari cara debugging untuk memastikan *Multi Linked List* bekerja sesuai dengan logika yang diharapkan.
· Mengetahui aplikasi praktis *Multi Linked List*, seperti untuk graf, sistem rekomendasi, atau representasi matriks.
· Melatih kemampuan kerja sama tim dalam memahami dan mengimplementasikan konsep struktur data lanjutan.
· Menyusun dokumentasi kode yang baik untuk mempermudah pengembangan lebih lanjut.
· Mengembangkan pemahaman yang mendalam tentang bagaimana struktur data memengaruhi performa algoritma pada suatu program.

II. DASAR TEORI

1. Pengertian Multi Linked List

Multi Linked List adalah jenis struktur data berbasis *linked list* yang memungkinkan setiap node memiliki lebih dari satu pointer yang mengarah ke node lain. Hal ini berbeda dengan *Single Linked List* yang hanya memiliki satu pointer untuk menghubungkan elemen berikutnya (*next*), atau *Double Linked List* yang memiliki dua pointer untuk menghubungkan elemen berikutnya (*next*) dan sebelumnya (*prev*).

Pada *Multi Linked List*, setiap node dapat memiliki beberapa pointer, yang biasanya digunakan untuk mewakili relasi antar data yang lebih kompleks. Contoh penggunaan *Multi Linked List* adalah untuk merepresentasikan graf, hubungan antar elemen matriks, atau sistem hierarki.

2. Struktur Dasar Multi Linked List

Node pada *Multi Linked List* biasanya memiliki komponen berikut:

- **Data:** Informasi atau nilai yang disimpan dalam node.
- **Pointer:** Satu atau lebih pointer yang digunakan untuk menghubungkan node tersebut ke node lain.

Contoh struktur node dalam C++:

```
struct Node {
    int data;
    Node* next1; // Pointer ke node berikutnya dalam relasi pertama
    Node* next2; // Pointer ke node berikutnya dalam relasi kedua
};
```

3. Implementasi dan Operasi Dasar

Operasi dasar yang dapat dilakukan pada *Multi Linked List* meliputi:

1. **Inisialisasi:** Membuat struktur awal dari *Multi Linked List*.
2. **Penambahan Node:** Menambahkan elemen baru ke dalam struktur sesuai dengan relasi yang ditentukan.
3. **Penghapusan Node:** Menghapus elemen tertentu dan menyesuaikan relasi pointer.
4. **Traversal:** Menelusuri elemen-elemen dalam *Multi Linked List* berdasarkan jalur relasi tertentu.

4. Kelebihan dan Kekurangan Multi Linked List

Kelebihan:

- Mampu merepresentasikan relasi kompleks antar elemen.
- Fleksibel untuk berbagai kebutuhan, seperti graf, sistem rekomendasi, atau hubungan matriks.

Kekurangan:

- Kompleksitas implementasi lebih tinggi dibandingkan *Single Linked List* atau *Double Linked List*.
- Memerlukan lebih banyak memori untuk menyimpan banyak pointer.

5. Contoh Penerapan Multi Linked List

Multi Linked List banyak digunakan dalam:

- **Graf:** Merepresentasikan simpul-simpul dan hubungan antar simpul.
- **Sistem hierarki:** Untuk merepresentasikan hubungan antar data dalam struktur organisasi.
- **Sparse Matrix:** Mewakili matriks dengan banyak elemen nol secara efisien.

III. GUIDED

```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;
}

```

```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;
}

```

```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;
}
```

IV. UNGUIDED

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

struct Proyek {
    string namaProyek;
    int durasi; // dalam bulan
    Proyek* next;
};

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

class MultiLinkedList {
private:
    Pegawai* head;

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

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

    void tambahProyek(string idPegawai, 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(string idPegawai, 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.\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;
        }
    }

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

int main() {
    MultiLinkedList mll;

    // Tambah Pegawai
    mll.tambahPegawai("Andi", "P001");
    mll.tambahPegawai("Budi", "P002");
    mll.tambahPegawai("Citra", "P003");

    // Tambah Proyek
    mll.tambahProyek("P001", "Aplikasi Mobile", 12);
    mll.tambahProyek("P002", "Sistem Akuntansi", 8);
    mll.tambahProyek("P003", "E-commerce", 10);
    mll.tambahProyek("P001", "Analisis Data", 6);

    // Hapus Proyek
    mll.hapusProyek("P001", "Aplikasi Mobile");

    // Tampilkan Data
    mll.tampilkanData();

    return 0;
}
```

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

struct Buku {
    string judulBuku;
    string tanggalPengembalian;
    Buku* next;
};

struct Anggota {
    string namaAnggota;
    string idAnggota;
    Buku* bukuHead;
    Anggota* next;
};

class MultiLinkedListPerpustakaan {
private:
    Anggota* head;

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

    void tambahAnggota(string nama, string id) {
        Anggota* anggotaBaru = new Anggota{nama, id, nullptr, head};
        head = anggotaBaru;
    }

    void tambahBuku(string idAnggota, string judul, string tanggal) {
        Anggota* anggota = cariAnggota(idAnggota);
        if (anggota) {
            Buku* bukuBaru = new Buku{judul, tanggal, anggota->bukuHead};
            anggota->bukuHead = bukuBaru;
        } else {
            cout << "Anggota dengan ID " << idAnggota << " tidak ditemukan.\n";
        }
    }

    void hapusAnggota(string idAnggota) {
        Anggota* current = head;
        Anggota* prev = nullptr;
        while (current && current->idAnggota != idAnggota) {
            prev = current;
            current = current->next;
        }
        if (current) {
            if (prev) {
                prev->next = current->next;
            } else {
                head = current->next;
            }

            Buku* bukuCurrent = current->bukuHead;
            while (bukuCurrent) {
                Buku* temp = bukuCurrent;
                bukuCurrent = bukuCurrent->next;
                delete temp;
            }
            delete current;
            cout << "Anggota dengan ID " << idAnggota << " berhasil dihapus.\n";
        } else {
            cout << "Anggota dengan ID " << idAnggota << " tidak ditemukan.\n";
        }
    }

    void tampilkanData() {
        Anggota* currentAnggota = head;
        while (currentAnggota) {
            cout << "Anggota: " << currentAnggota->namaAnggota
                 << " (ID: " << currentAnggota->idAnggota << ")\n";
            Buku* currentBuku = currentAnggota->bukuHead;
            while (currentBuku) {
                cout << "  - Buku: " << currentBuku->judulBuku
                     << ", Pengembalian: " << currentBuku->tanggalPengembalian << "\n";
                currentBuku = currentBuku->next;
            }
            currentAnggota = currentAnggota->next;
        }
    }

private:
    Anggota* cariAnggota(string id) {
        Anggota* current = head;
        while (current) {
            if (current->idAnggota == id) return current;
            current = current->next;
        }
        return nullptr;
    }
};

int main() {
    MultiLinkedListPerpustakaan perpustakaan;

    // Tambah Anggota
    perpustakaan.tambahAnggota("Rani", "A001");
    perpustakaan.tambahAnggota("Dito", "A002");
    perpustakaan.tambahAnggota("Vina", "A003");

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

    // Hapus Anggota
    perpustakaan.hapusAnggota("A002");

    // Tampilkan Data
    perpustakaan.tampilkanData();

    return 0;
}
```

V. KESIMPULAN

Berdasarkan hasil praktikum yang telah dilakukan, dapat disimpulkan bahwa *Multi Linked List* merupakan salah satu struktur data yang fleksibel dan efisien untuk merepresentasikan hubungan antar data yang kompleks. Struktur ini memungkinkan setiap node memiliki lebih dari satu pointer, sehingga dapat digunakan untuk berbagai kebutuhan, seperti representasi graf, sistem hierarki, dan matriks jarang (*sparse matrix*). Melalui implementasi dan pengujian operasi dasar, seperti penambahan, penghapusan, dan penelusuran node, dapat diketahui bahwa *Multi Linked List* memiliki kelebihan dalam fleksibilitas relasi antar elemen, meskipun memerlukan lebih banyak memori dibandingkan dengan struktur data sederhana seperti *Single Linked List*. Praktikum ini juga menunjukkan pentingnya pemahaman konsep pointer untuk memastikan hubungan antar node dalam *Multi Linked List* dapat terkelola dengan baik. Dengan demikian, *Multi Linked List* menjadi solusi yang tepat untuk berbagai permasalahan yang melibatkan hubungan data yang kompleks dan dinamis.