LAPORAN PRAKTIKUM

MODUL 13

Disusun Oleh:

Naura Aisha Zahira (2311104078)

S1SE-07-02

Dosen :

Wahyu Andi Saputra, 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 Linked List adalah struktur data yang menghubungkan beberapa linked list, memungkinkan representasi relasi kompleks antara data, seperti hubungan parent-child. Multi Linked List adalah variasi dari linked list yang memiliki lebih dari satu pointer atau link pada setiap node. Ini memungkinkan untuk menghubungkan beberapa linked list dan merepresentasikan berbagai jenis hubungan antar data, seperti hubungan 1-n, m-n, dan 1-1. Dalam konteks ini, setiap node dapat memiliki referensi ke node lain yang berfungsi sebagai parent atau child, sehingga membentuk struktur yang lebih kompleks daripada linked list biasa.

3. Guided

1. Guided 1

Sourcecode:

```
#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
```

2. Guided 2

Sourcecode:

```
#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
```

3. Guided 3

Sourcecode:

```
#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) {
```

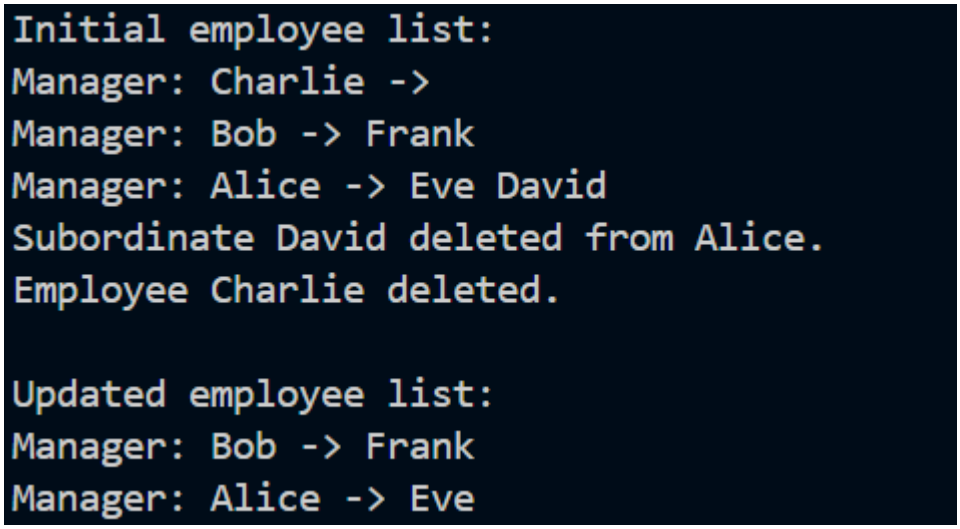
```
            manager = manager->next;
        }
        if (manager != nullptr) { // Jika manajer ditemukan
            EmployeeNode* newSubordinate = new EmployeeNode(subordinateName);
            newSubordinate->next = manager->subordinate; // Menyambungkan ke
subordinate sebelumnya
            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:



4. Unguided

1. Soal:

```
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.
```

Code:

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

struct Proyek {
    string namaProyek;
    int durasi;
    Proyek* proyekBerikutnya;

    Proyek(string nama, int dur) : namaProyek(nama), durasi(dur),
proyekBerikutnya(nullptr) {}
};

struct Pegawai {
    string namaPegawai;
    string idPegawai;
    Proyek* kepalaProyek;
    Pegawai* pegawaiBerikutnya;

    Pegawai(string nama, string id) : namaPegawai(nama), idPegawai(id),
```

```
kepalaProyek(nullptr), pegawaiBerikutnya(nullptr) {}
};

class MultiLinkedList {
private:
    Pegawai* kepalaPegawai;

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

    void tambahPegawai(string nama, string id) {
        Pegawai* pegawaiBaru = new Pegawai(nama, id);
        if (!kepalaPegawai) {
            kepalaPegawai = pegawaiBaru;
        } else {
            Pegawai* temp = kepalaPegawai;
            while (temp->pegawaiBerikutnya) temp = temp->pegawaiBerikutnya;
            temp->pegawaiBerikutnya = pegawaiBaru;
        }
    }

    void tambahProyek(string id, string namaProyek, int durasi) {
        Pegawai* pegawai = cariPegawai(id);
        if (pegawai) {
            Proyek* proyekBaru = new Proyek(namaProyek, durasi);
            if (!pegawai->kepalaProyek) {
                pegawai->kepalaProyek = proyekBaru;
            } else {
                Proyek* temp = pegawai->kepalaProyek;
                while (temp->proyekBerikutnya) temp = temp->proyekBerikutnya;
                temp->proyekBerikutnya = proyekBaru;
            }
        }
    }
```

```
    void hapusProyek(string id, string namaProyek) {
        Pegawai* pegawai = cariPegawai(id);
        if (pegawai && pegawai->kepalaProyek) {
            Proyek* temp = pegawai->kepalaProyek;
            Proyek* prev = nullptr;

            while (temp) {
                if (temp->namaProyek == namaProyek) {
                    if (prev) {
                        prev->proyekBerikutnya = temp->proyekBerikutnya;
                    } else {
                        pegawai->kepalaProyek = temp->proyekBerikutnya;
                    }
                    delete temp;
                    return;
                }
                prev = temp;
                temp = temp->proyekBerikutnya;
            }
        }
    }

    void tampilkanData() {
        Pegawai* temp = kepalaPegawai;
        while (temp) {
            cout << "Pegawai: " << temp->namaPegawai << " (ID: " << temp->idPegawai
<< ")\n";
            Proyek* proyekTemp = temp->kepalaProyek;
            while (proyekTemp) {
                cout << "  -> Proyek: " << proyekTemp->namaProyek << ", Durasi: " <<
proyekTemp->durasi << " bulan\n";
                proyekTemp = proyekTemp->proyekBerikutnya;
            }
            temp = temp->pegawaiBerikutnya;
            cout << endl;
```

```
        }
    }

private:
    Pegawai* cariPegawai(string id) {
        Pegawai* temp = kepalaPegawai;
        while (temp) {
            if (temp->idPegawai == id) return temp;
            temp = temp->pegawaiBerikutnya;
        }
        return nullptr;
    }
};

int main() {
    MultiLinkedList daftar;

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

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

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

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

    cout << "Data Pegawai dan Proyek:\n";
    daftar.tampilkanData();

    return 0;
}
```

Output:

```
Data Pegawai dan Proyek:
Pegawai: Andi (ID: P001)
  -> Proyek: Analisis Data, Durasi: 6 bulan

Pegawai: Budi (ID: P002)
  -> Proyek: Sistem Akuntansi, Durasi: 8 bulan

Pegawai: Citra (ID: P003)
  -> Proyek: E-commerce, Durasi: 10 bulan
```

2. Soal:

```
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.
```

Code:

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

struct Buku {
    string judulBuku;
    string tanggalPengembalian;
    Buku* bukuBerikutnya;

    Buku(string judul, string tanggal) : judulBuku(judul),
tanggalPengembalian(tanggal), bukuBerikutnya(nullptr) {}
```

```
};

struct Anggota {
    string namaAnggota;
    string idAnggota;
    Buku* kepalaBuku;
    Anggota* anggotaBerikutnya;

    Anggota(string nama, string id) : namaAnggota(nama), idAnggota(id),
kepalaBuku(nullptr), anggotaBerikutnya(nullptr) {}
};

class MultiLinkedList {
private:
    Anggota* kepalaAnggota;

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

    void tambahAnggota(string nama, string id) {
        Anggota* anggotaBaru = new Anggota(nama, id);
        if (!kepalaAnggota) {
            kepalaAnggota = anggotaBaru;
        } else {
            Anggota* temp = kepalaAnggota;
            while (temp->anggotaBerikutnya) temp = temp->anggotaBerikutnya;
            temp->anggotaBerikutnya = anggotaBaru;
        }
    }

    void tambahBuku(string id, string judulBuku, string tanggalPengembalian) {
        Anggota* anggota = cariAnggota(id);
        if (anggota) {
            Buku* bukuBaru = new Buku(judulBuku, tanggalPengembalian);
            if (!anggota->kepalaBuku) {
```

```
            anggota->kepalaBuku = bukuBaru;
        } else {
            Buku* temp = anggota->kepalaBuku;
            while (temp->bukuBerikutnya) temp = temp->bukuBerikutnya;
            temp->bukuBerikutnya = bukuBaru;
        }
    }
}

void hapusAnggota(string id) {
    Anggota* temp = kepalaAnggota;
    Anggota* prev = nullptr;

    while (temp) {
        if (temp->idAnggota == id) {
            Buku* bukuTemp = temp->kepalaBuku;
            while (bukuTemp) {
                Buku* hapus = bukuTemp;
                bukuTemp = bukuTemp->bukuBerikutnya;
                delete hapus;
            }

            if (prev) {
                prev->anggotaBerikutnya = temp->anggotaBerikutnya;
            } else {
                kepalaAnggota = temp->anggotaBerikutnya;
            }
            delete temp;
            return;
        }
        prev = temp;
        temp = temp->anggotaBerikutnya;
    }
}
```

```
    void tampilkanData() {
        Anggota* temp = kepalaAnggota;
        while (temp) {
            cout << "Anggota: " << temp->namaAnggota << " (ID: " << temp->idAnggota
<< ")\n";
            Buku* bukuTemp = temp->kepalaBuku;
            while (bukuTemp) {
                cout << "  -> Buku: " << bukuTemp->judulBuku << ", Pengembalian: " <<
bukuTemp->tanggalPengembalian << "\n";
                bukuTemp = bukuTemp->bukuBerikutnya;
            }
            temp = temp->anggotaBerikutnya;
            cout << endl;
        }
    }

private:
    Anggota* cariAnggota(string id) {
        Anggota* temp = kepalaAnggota;
        while (temp) {
            if (temp->idAnggota == id) return temp;
            temp = temp->anggotaBerikutnya;
        }
        return nullptr;
    }
};

int main() {
    MultiLinkedList perpustakaan;

    // 1. Menambahkan anggota
    perpustakaan.tambahAnggota("Rani", "A001");
    perpustakaan.tambahAnggota("Dito", "A002");
    perpustakaan.tambahAnggota("Vina", "A003");
```

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

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

    // 4. Menghapus anggota Dito
    perpustakaan.hapusAnggota("A002");

    // 5. Menampilkan data anggota dan buku yang dipinjam
    cout << "Data Anggota dan Buku yang Dipinjam:\n";
    perpustakaan.tampilkanData();

    return 0;
}
```

Output:

```
Data Anggota dan Buku yang Dipinjam:
Anggota: Rani (ID: A001)
  -> Buku: Pemrograman C++, Pengembalian: 01/12/2024
  -> Buku: Struktur Data, Pengembalian: 10/12/2024

Anggota: Vina (ID: A003)
```

3. Kesimpulan

Pada praktikum Modul 13 ini, telah dipelajari dan diimplementasikan konsep Multi Linked List yang memungkinkan pengelolaan data dengan hubungan hierarkis seperti parent-child. Multi Linked List merupakan pengembangan dari struktur data linked list biasa, di mana setiap node dapat memiliki lebih dari satu pointer. Implementasi ini digunakan dalam berbagai studi kasus, seperti hubungan antara manajer dan bawahan, pegawai dan proyek, serta anggota perpustakaan dengan buku yang dipinjam. Melalui program yang dikembangkan, fitur-fitur seperti penambahan node, penghapusan node, dan penampilan data berhasil diimplementasikan. Dengan adanya praktikum ini, pemahaman tentang struktur data kompleks semakin mendalam, khususnya dalam menangani hubungan antar data yang tidak linear.