LAPORAN PRAKTIKUM
Modul 13
MULTI LINKED LIST

Disusun Oleh:
Aulia Jasifa Br Ginting 2311104060
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

Multi Linked List adalah struktur data yang merupakan pengembangan dari linked list biasa. Dalam struktur ini, setiap node tidak hanya memiliki satu pointer (atau referensi) ke node berikutnya, tetapi juga dapat memiliki beberapa pointer yang mengarah ke node lain. Ini memungkinkan untuk menyimpan data yang lebih kompleks dan terorganisir dengan lebih baik.

Karakteristik Multi Linked List:

1. Beberapa Pointer: Setiap node dapat memiliki lebih dari satu pointer. Misalnya, dalam konteks manajemen pegawai dan proyek, setiap pegawai dapat memiliki pointer ke proyek yang mereka kelola, dan setiap proyek dapat memiliki pointer ke pegawai yang mengelolanya.

2. Struktur Hierarkis: Multi Linked List sering digunakan untuk merepresentasikan hubungan hierarkis atau relasional. Misalnya, dalam sistem manajemen, pegawai dapat memiliki beberapa proyek, dan setiap proyek dapat memiliki beberapa pegawai yang terlibat.

3. Fleksibilitas: Dengan menggunakan beberapa pointer, kita dapat dengan mudah menavigasi dan mengelola data yang saling terkait. Ini memberikan fleksibilitas dalam mengakses dan memanipulasi data.

3. **Guided**

Code programnya:

```
#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;
}
```

Outputnya:

```
PS C:\Users\LENOVO\Documents\STU
Parent: 3 -> 30 30
Parent: 2 -> 20 20
Parent: 1 -> 11 10
```

Code programnya:

```
#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;
}
```

Outputnya:

```
PS C:\Users\LENOVO\Documents\STUDYING\
Manager: Charlie -> Brian Frans
Manager: Bob -> Frank
Manager: Alice -> Eve David
```

Code programnya:

```
#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) {}
```

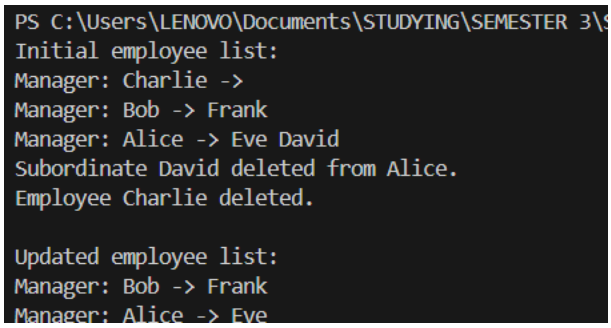
```
// Menambahkan karyawan (induk)
void addEmployee(string name) {
    EmployeeNode* newEmployee = new EmployeeNode(name);
    newEmployee->next = head; // Menyambungkan ke karyawan sebelumnya
    head = newEmployee; // Memperbarui head
}

// 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;
}
```

Outputnya:



4. Unguided

1. Manajemen Data Pegawai dan Proyek

Buatla 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 programnya:

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

```
// Struktur untuk Proyek
struct Project {
    string nama;
    int durasi;
    Project* nextProject;
    Project(string n, int d) : nama(n), durasi(d), nextProject(nullptr) {}
};

// Struktur untuk Pegawai
struct Employee {
    string nama;
    string id;
    Project* projects;
    Employee* nextEmployee;

    Employee(string n, string i) : nama(n), id(i), projects(nullptr),
nextEmployee(nullptr) {}
};

// Kelas Manajemen Pegawai dan Proyek
class EmployeeProjectManagement {
private:
    Employee* head;

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

    // Tambah Pegawai Baru
    void tambahPegawai(string nama, string id) {
        Employee* newEmployee = new Employee(nama, id);
        if (!head) {
            head = newEmployee;
        } else {
            Employee* temp = head;
            while (temp->nextEmployee) {
                temp = temp->nextEmployee;
            }
            temp->nextEmployee = newEmployee;
        }
    }

    // Tambah Proyek ke Pegawai
```

```
void tambahProyek(string namaEmployee, string namaProyek, int durasi) {
    Employee* employee = cariPegawai(namaEmployee);
    if (employee) {
        Project* newProject = new Project(namaProyek, durasi);
        if (!employee->projects) {
            employee->projects = newProject;
        } else {
            Project* temp = employee->projects;
            while (temp->nextProject) {
                temp = temp->nextProject;
            }
            temp->nextProject = newProject;
        }
    }
}

// Cari Pegawai berdasarkan Nama
Employee* cariPegawai(string nama) {
    Employee* temp = head;
    while (temp) {
        if (temp->nama == nama) {
            return temp;
        }
        temp = temp->nextEmployee;
    }
    return nullptr;
}

// Hapus Proyek dari Pegawai
void hapusProyek(string namaEmployee, string namaProyek) {
    Employee* employee = cariPegawai(namaEmployee);
    if (employee) {
        Project* current = employee->projects;
        Project* prev = nullptr;

        while (current) {
            if (current->nama == namaProyek) {
                if (prev) {
                    prev->nextProject = current->nextProject;
                } else {
                    employee->projects = current->nextProject;
                }
```

```
                delete current;
                return;
            }
            prev = current;
            current = current->nextProject;
        }
    }
}

// Tampilkan Data Pegawai dan Proyek
void tampilkanData() {
    Employee* employeeTemp = head;
    while (employeeTemp) {
        cout << "Pegawai: " << employeeTemp->nama << " (ID: " <<
employeeTemp->id << ")" << endl;

        Project* projectTemp = employeeTemp->projects;
        if (!projectTemp) {
            cout << "  Tidak memiliki proyek" << endl;
        } else {
            cout << "  Proyek:" << endl;
            while (projectTemp) {
                cout << "  - " << projectTemp->nama << " (Durasi: " << projectTemp-
>durasi << " bulan)" << endl;
                projectTemp = projectTemp->nextProject;
            }
        }
        cout << endl;
        employeeTemp = employeeTemp->nextEmployee;
    }
}

// Destruktor untuk membebaskan memori
~EmployeeProjectManagement() {
    while (head) {
        Employee* empTemp = head;
        head = head->nextEmployee;

        // Hapus semua proyek pegawai
        while (empTemp->projects) {
            Project* projTemp = empTemp->projects;
            empTemp->projects = empTemp->projects->nextProject;
```

```
            delete projTemp;
        }

        delete empTemp;
    }
  }
};

int main() {
  EmployeeProjectManagement manajemen;

  // 1. Masukkan data pegawai
  manajemen.tambahPegawai("Andi", "P001");
  manajemen.tambahPegawai("Budi", "P002");
  manajemen.tambahPegawai("Citra", "P003");

  // 2. Tambahkan proyek ke pegawai
  manajemen.tambahProyek("Andi", "Aplikasi Mobile", 12);
  manajemen.tambahProyek("Budi", "Sistem Akuntansi", 8);
  manajemen.tambahProyek("Citra", "E-commerce", 10);

  // 3. Tambahkan proyek baru untuk Andi
  manajemen.tambahProyek("Andi", "Analisis Data", 6);

  // 4. Hapus proyek "Aplikasi Mobile" dari Andi
  manajemen.hapusProyek("Andi", "Aplikasi Mobile");

  // 5. Tampilkan data pegawai dan proyek
  manajemen.tampilkanData();

  return 0;
}
```

Outputnya:

```
PS C:\Users\LENOVO\Documents\STUDYING\SEMESTER 3\
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. 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 programnya:

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

// Struktur untuk Buku
struct Book {
    string judul;
    string tanggalPengembalian;
    Book* nextBook;
    Book(string j, string t) : judul(j), tanggalPengembalian(t), nextBook(nullptr) {}
};

// Struktur untuk Anggota
struct Member {
    string nama;
    string id;
    Book* buku;
    Member* nextMember;

    Member(string n, string i) : nama(n), id(i), buku(nullptr), nextMember(nullptr) {}
```

```
};

// Kelas Sistem Manajemen Buku Perpustakaan
class LibraryManagementSystem {
private:
    Member* head;

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

    // Tambah Anggota Baru
    void tambahAnggota(string nama, string id) {
        Member* newMember = new Member(nama, id);
        if (!head) {
            head = newMember;
        } else {
            Member* temp = head;
            while (temp->nextMember) {
                temp = temp->nextMember;
            }
            temp->nextMember = newMember;
        }
    }

    // Tambah Buku ke Anggota
    void tambahBuku(string namaAnggota, string judulBuku, string
tanggalPengembalian) {
        Member* member = cariAnggota(namaAnggota);
        if (member) {
            Book* newBook = new Book(judulBuku, tanggalPengembalian);
            if (!member->buku) {
                member->buku = newBook;
            } else {
                Book* temp = member->buku;
                while (temp->nextBook) {
                    temp = temp->nextBook;
                }
                temp->nextBook = newBook;
            }
        }
    }
```

```
// Cari Anggota berdasarkan Nama
Member* cariAnggota(string nama) {
    Member* temp = head;
    while (temp) {
        if (temp->nama == nama) {
            return temp;
        }
        temp = temp->nextMember;
    }
    return nullptr;
}

// Hapus Anggota dan Buku yang Dipinjam
void hapusAnggota(string nama) {
    Member* prev = nullptr;
    Member* current = head;

    while (current) {
        if (current->nama == nama) {
            // Hapus semua buku yang dipinjam
            while (current->buku) {
                Book* bookTemp = current->buku;
                current->buku = current->buku->nextBook;
                delete bookTemp;
            }

            if (prev) {
                prev->nextMember = current->nextMember;
            } else {
                head = current->nextMember;
            }
            delete current;
            return;
        }
        prev = current;
        current = current->nextMember;
    }
}

// Tampilkan Data Anggota dan Buku
void tampilkanData() {
    Member* memberTemp = head;
```

```
        while (memberTemp) {
            cout << "Anggota: " << memberTemp->nama << " (ID: " << memberTemp->id << ")" << endl;

            Book* bookTemp = memberTemp->buku;
            if (!bookTemp) {
                cout << "  Tidak ada buku yang dipinjam" << endl;
            } else {
                cout << "  Buku yang dipinjam:" << endl;
                while (bookTemp) {
                    cout << "  - " << bookTemp->judul << " (Tanggal Pengembalian: " << bookTemp->tanggalPengembalian << ")" << endl;
                    bookTemp = bookTemp->nextBook;
                }
            }
            cout << endl;
            memberTemp = memberTemp->nextMember;
        }
    }

    // Destruktor untuk membebaskan memori
    ~LibraryManagementSystem() {
        while (head) {
            Member* memTemp = head;
            head = head->nextMember;

            // Hapus semua buku yang dipinjam anggota
            while (memTemp->buku) {
                Book* bookTemp = memTemp->buku;
                memTemp->buku = memTemp->buku->nextBook;
                delete bookTemp;
            }

            delete memTemp;
        }
    }
};

int main() {
    LibraryManagementSystem perpustakaan;

    // 1. Masukkan data anggota
```

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

    // 2. Tambahkan buku yang dipinjam
    perpustakaan.tambahBuku("Rani", "Pemrograman C++", "01/12/2024");
    perpustakaan.tambahBuku("Dito", "Algoritma Pemrograman", "15/12/2024");

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

    // 4. Hapus anggota Dito beserta buku yang dipinjam
    perpustakaan.hapusAnggota("Dito");

    // 5. Tampilkan seluruh data anggota dan buku yang dipinjam
    perpustakaan.tampilkanData();

    return 0;
}
```

Outputnya:

```
PS C:\Users\LENOVO\Documents\STUDYING\SEMESTER 3\Struktur
Anggota: Rani (ID: A001)
  Buku yang dipinjam:
  - Pemrograman C++ (Tanggal Pengembalian: 01/12/2024)
  - Struktur Data (Tanggal Pengembalian: 10/12/2024)

Anggota: Vina (ID: A003)
  Tidak ada buku yang dipinjam
```

5. **Kesimpulan**

Pada praktikum ini mempelajari Multi Linked List, struktur data yang sangat berguna untuk menyimpan dan mengelola data yang memiliki hubungan kompleks. Dengan kemampuannya untuk memiliki beberapa pointer, struktur ini memungkinkan pengorganisasian data yang lebih baik dan akses yang lebih fleksibel.