LAPORAN PRAKTIKUM
Modul 13
MULTI LINKED LIST

Disusun Oleh:

Muhammad Shafiq Rasuna - 2311104043

Kelas :

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. Dasar Teori

Multi Linked List adalah struktur data yang merupakan perkembangan dari Linked List. Pada Multi Linked List, setiap node memiliki lebih dari satu pointer yang menghubungkan ke node lainnya, memungkinkan node memiliki beberapa hubungan dengan node lain. Struktur ini terdiri dari komponen seperti node, pointer, head dan tail, serta memiliki beberapa jenis seperti Doubly Linked List dan Circularly Linked List. Multi Linked List memungkinkan operasi dasar seperti penambahan, penghapusan, pencarian dan pengolahan data yang lebih efisien. Meskipun memiliki kelebihan seperti fleksibilitas dan efisiensi, struktur ini juga memiliki kekurangan seperti kompleksitas dan memori yang lebih besar. Multi Linked List banyak digunakan dalam aplikasi seperti manajemen jaringan sosial, sistem rekomendasi dan pengolahan data kompleks.

3. Guided

1. Kode programnya:

```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;
}
```

Hasil run :

```
Parent: 3 -> 30 30
Parent: 2 -> 20 20
Parent: 1 -> 11 10

c:\Users\ASUS\OneDrive\Dokumen\tugas smt  3\Pemograman Struktur Data 3\pertemuan13>
```

2. **Kode programnya:**

```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;
}
```

Hasil run :

```
Manager: Charlie -> Brian Frans
Manager: Bob -> Frank
Manager: Alice -> Eve David

c:\Users\ASUS\OneDrive\Dokumen\tugas smt  3\Pemograman Struktur Data 3\pertemuan13>
```

3. **Kode programnya:**

```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;
}
```

Hasil run:

```
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

c:\Users\ASUS\OneDrive\Dokumen\tugas smt  3\Pemograman Struktur Data 3\pertemuan13>
```

4. Unguided

1. Manajemen Data Pegawai dan Proyek

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

// Node untuk proyek
struct ProyekNode {
    string namaProyek;
    int durasi;
    ProyekNode* next;

    ProyekNode(string nama, int dur) : namaProyek(nama), durasi(dur), next(nullptr) {}
};

// Node untuk pegawai
struct PegawaiNode {
    string nama;
    string idPegawai;
    PegawaiNode* next;
    ProyekNode* proyekHead;

    PegawaiNode(string namaPeg, string id) : nama(namaPeg), idPegawai(id), next(nullptr), proyekHead(nullptr) {}
};

class ManajemenPegawaiProyek {
private:
    PegawaiNode* head;

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

    // Menambah pegawai baru
    void tambahPegawai(string nama, string idPegawai) {
        PegawaiNode* newPegawai = new PegawaiNode(nama, idPegawai);
        if (!head) {
            head = newPegawai;
        } else {
            PegawaiNode* current = head;
            while (current->next) {
                current = current->next;
            }
            current->next = newPegawai;
        }
    }
```

```
// Menambah proyek ke pegawai berdasarkan ID
void tambahProyek(string idPegawai, string namaProyek, int durasi) {
    PegawaiNode* current = head;
    while (current) {
        if (current->idPegawai == idPegawai) {
            ProyekNode* newProyek = new ProyekNode(namaProyek, durasi);
            if (!current->proyekHead) {
                current->proyekHead = newProyek;
            } else {
                ProyekNode* proyekCurrent = current->proyekHead;
                while (proyekCurrent->next) {
                    proyekCurrent = proyekCurrent->next;
                }
                proyekCurrent->next = newProyek;
            }
            return;
        }
        current = current->next;
    }
    cout << "Pegawai dengan ID " << idPegawai << " tidak ditemukan." << endl;
}

// Menghapus proyek dari pegawai berdasarkan ID pegawai dan nama proyek
void hapusProyek(string idPegawai, string namaProyek) {
    PegawaiNode* current = head;
    while (current) {
        if (current->idPegawai == idPegawai) {
            ProyekNode* proyekCurrent = current->proyekHead;
            ProyekNode* prev = nullptr;
            while (proyekCurrent) {
                if (proyekCurrent->namaProyek == namaProyek) {
                    if (prev) {
                        prev->next = proyekCurrent->next;
                    } else {
                        current->proyekHead = proyekCurrent->next;
                    }
                    delete proyekCurrent;
                    cout << "Proyek '" << namaProyek << "' berhasil dihapus dari " << current->nama << "." << endl;
                    return;
                }
                prev = proyekCurrent;
                proyekCurrent = proyekCurrent->next;
            }
            cout << "Proyek '" << namaProyek << "' tidak ditemukan untuk pegawai " << current->nama << "." << endl;
            return;
        }
        current = current->next;
    }
    cout << "Pegawai dengan ID " << idPegawai << " tidak ditemukan." << endl;
}
```

```
    // Menampilkan data pegawai dan proyek mereka
    void tampilkanData() {
        PegawaiNode* current = head;
        while (current) {
            cout << "Pegawai: " << current->nama << " (ID: " << current->idPegawai << ")" << endl;
            ProyekNode* proyekCurrent = current->proyekHead;
            if (proyekCurrent) {
                while (proyekCurrent) {
                    cout << "  - Proyek: " << proyekCurrent->namaProyek << ", Durasi: " << proyekCurrent->durasi << " bulan" << endl;
                    proyekCurrent = proyekCurrent->next;
                }
            } else {
                cout << "  Tidak ada proyek." << endl;
            }
            current = current->next;
            cout << "-" << endl;
        }
    }
};

int main() {
    ManajemenPegawaiProyek manajemen;

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

    // 2. Menambahkan proyek ke pegawai
    manajemen.tambahProyek("P001", "Aplikasi Mobile", 12);
    manajemen.tambahProyek("P002", "Sistem Akuntansi", 8);
    manajemen.tambahProyek("P003", "E-commerce", 10);

    // 3. Menambahkan proyek baru untuk Andi
    manajemen.tambahProyek("P001", "Analisis Data", 6);

    // 4. Menghapus proyek "Aplikasi Mobile" dari Andi
    manajemen.hapusProyek("P001", "Aplikasi Mobile");

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

    return 0;
}
```

Hasil run:

```
Proyek 'Aplikasi Mobile' berhasil dihapus dari Andi.
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
-

c:\Users\ASUS\OneDrive\Dokumen\tugas smt  3\Pemograman Struktur Data 3\pertemuan13>
```

2. Sistem Manajemen Buku Perpustakaan

Kode programnya:

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

// Struktur untuk Node Buku
struct Buku {
    string judulBuku;
    string tanggalPengembalian;
    Buku* next; // Pointer ke buku berikutnya
};

// Struktur untuk Node Anggota
struct Anggota {
    string namaAnggota;
    string idAnggota;
    Buku* headBuku; // Pointer ke buku pertama
    Anggota* next;  // Pointer ke anggota berikutnya
};

// Fungsi untuk membuat node anggota baru
Anggota* buatAnggota(string nama, string id) {
    Anggota* anggotaBaru = new Anggota;
    anggotaBaru->namaAnggota = nama;
    anggotaBaru->idAnggota = id;
    anggotaBaru->headBuku = NULL;
    anggotaBaru->next = NULL;
    return anggotaBaru;
}

// Fungsi untuk membuat node buku baru
Buku* buatBuku(string judul, string tanggal) {
    Buku* bukuBaru = new Buku;
    bukuBaru->judulBuku = judul;
    bukuBaru->tanggalPengembalian = tanggal;
    bukuBaru->next = NULL;
    return bukuBaru;
}

// Fungsi untuk menambahkan buku ke anggota tertentu
void tambahBuku(Anggota* anggota, string judul, string tanggal) {
    Buku* bukuBaru = buatBuku(judul, tanggal);
    if (anggota->headBuku == NULL) {
        anggota->headBuku = bukuBaru;
    } else {
        Buku* temp = anggota->headBuku;
        while (temp->next != NULL) {
            temp = temp->next; // Cari buku terakhir
        }
        temp->next = bukuBaru;
    }
}
```

```cpp
// Fungsi untuk menghapus anggota beserta semua buku yang dipinjam
void hapusAnggota(Anggota*& head, string id) {
    Anggota* temp = head;
    Anggota* prev = NULL;

    while (temp != NULL && temp->idAnggota != id) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) {
        cout << "Anggota dengan ID \"" << id << "\" tidak ditemukan.\n";
        return;
    }

    // Hapus semua buku yang dipinjam oleh anggota
    Buku* currentBuku = temp->headBuku;
    while (currentBuku != NULL) {
        Buku* toDelete = currentBuku;
        currentBuku = currentBuku->next;
        delete toDelete;
    }

    // Hapus anggota dari daftar
    if (prev == NULL) {
        head = temp->next; // Anggota pertama dihapus
    } else {
        prev->next = temp->next;
    }

    delete temp;
    cout << "Anggota dengan ID \"" << id << "\" berhasil dihapus beserta bukunya.\n";
}

// Fungsi untuk menampilkan seluruh data anggota dan buku yang dipinjam
void tampilkanData(Anggota* head) {
    Anggota* tempAnggota = head;
    while (tempAnggota != NULL) {
        cout << "Anggota: " << tempAnggota->namaAnggota
             << " (ID: " << tempAnggota->idAnggota << ")\n";
        Buku* tempBuku = tempAnggota->headBuku;
        if (tempBuku == NULL) {
            cout << "  Tidak ada buku yang dipinjam.\n";
        } else {
            while (tempBuku != NULL) {
                cout << "  - Buku: " << tempBuku->judulBuku
                     << ", Pengembalian: " << tempBuku->tanggalPengembalian << endl;
                tempBuku = tempBuku->next;
            }
        }
        tempAnggota = tempAnggota->next;
        cout << endl;
    }
}
```

```cpp
int main() {
    // Membuat daftar anggota perpustakaan
    Anggota* headAnggota = buatAnggota("Rani", "A001");
    headAnggota->next = buatAnggota("Dito", "A002");
    headAnggota->next->next = buatAnggota("Vina", "A003");

    // Menambahkan buku yang dipinjam
    tambahBuku(headAnggota, "Pemrograman C++", "01/12/2024"); // Untuk Rani
    tambahBuku(headAnggota->next, "Algoritma Pemrograman", "15/12/2024"); // Untuk Dito

    // Menambahkan buku baru untuk Rani
    tambahBuku(headAnggota, "Struktur Data", "10/12/2024");

    // Menghapus anggota Dito beserta bukunya
    hapusAnggota(headAnggota, "A002");

    // Menampilkan seluruh data anggota dan buku
    cout << "\nData Anggota Perpustakaan dan Buku yang Dipinjam:\n";
    tampilkanData(headAnggota);

    return 0;
}
```

Output nya :

```
Data Anggota Perpustakaan 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)
  Tidak ada buku yang dipinjam.


c:\Users\ASUS\OneDrive\Dokumen\tugas smt  3\Pemograman Struktur Data 3\pertemuan13>
```

5. Kesimpulan

Berdasarkan laporan praktikum ini, dapat disimpulkan bahwa Multi Linked List merupakan pengembangan dari struktur data Linked List yang memungkinkan setiap node memiliki lebih dari satu pointer untuk menghubungkan ke node lainnya. Dengan karakteristik ini, Multi Linked List menawarkan fleksibilitas yang lebih besar dalam membangun relasi antar node, sehingga cocok untuk kasus yang memerlukan hubungan kompleks antar data.

Implementasi Multi Linked List dalam studi kasus menunjukkan bahwa struktur ini mampu mendukung operasi dasar seperti penambahan, penghapusan, pencarian, dan pengolahan data dengan efisien. Meskipun memberikan banyak kelebihan seperti efisiensi dan fleksibilitas dalam manajemen data, struktur ini juga memiliki kekurangan seperti kompleksitas algoritma dan penggunaan memori yang lebih besar.

Secara keseluruhan, Multi Linked List terbukti bermanfaat untuk aplikasi-aplikasi yang membutuhkan manajemen data kompleks, seperti sistem rekomendasi, jaringan sosial, dan pengolahan data yang membutuhkan relasi multidimensional antar elemen.