LAPORAN PRAKTIKUM

MODUL 13

MULTI LINKED LIST

Disusun Oleh :

Fahmi Hasan Asagaf (2311104074)

Kelas:

SE-07-2

Dosen :

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

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. TUJUAN

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

II. LANDASAN TEORI

Penggunaan Multi Linked List merupakan salah satu teknik pengolahan data yang efektif dalam struktur data. Secara konsep, Multi Linked List adalah pengembangan dari Linked List biasa, dimana setiap node memiliki lebih dari satu pointer yang mengarah ke node lainnya. Hal ini memungkinkan data disimpan dan diakses secara efisien dalam beberapa dimensi.

Dalam implementasinya, Multi Linked List dapat digunakan dalam berbagai studi kasus seperti pengolahan data kompleks, representasi graf, dan pengembangan database. Dengan memahami konsep dan penggunaan Multi Linked List, praktikan dapat mengembangkan kemampuan dalam merancang dan mengimplementasikan algoritma yang lebih efisien dan efektif. Oleh karena itu, praktikum ini bertujuan untuk memahami penggunaan Multi Linked List dan mengimplementasikannya dalam beberapa studi kasus nyata.

III. GUIDED

1. Kode Program Guided 1:

```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:

```
d1.cpp -o guided1 } ; if ($?) { .\guided1 }
Parent: 3 -> 30 30
Parent: 2 -> 20 20
Parent: 1 -> 11 10
```

2. Kode Program Guided 2:

```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
```

Kode program guided 3

```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;
        }
    }

```

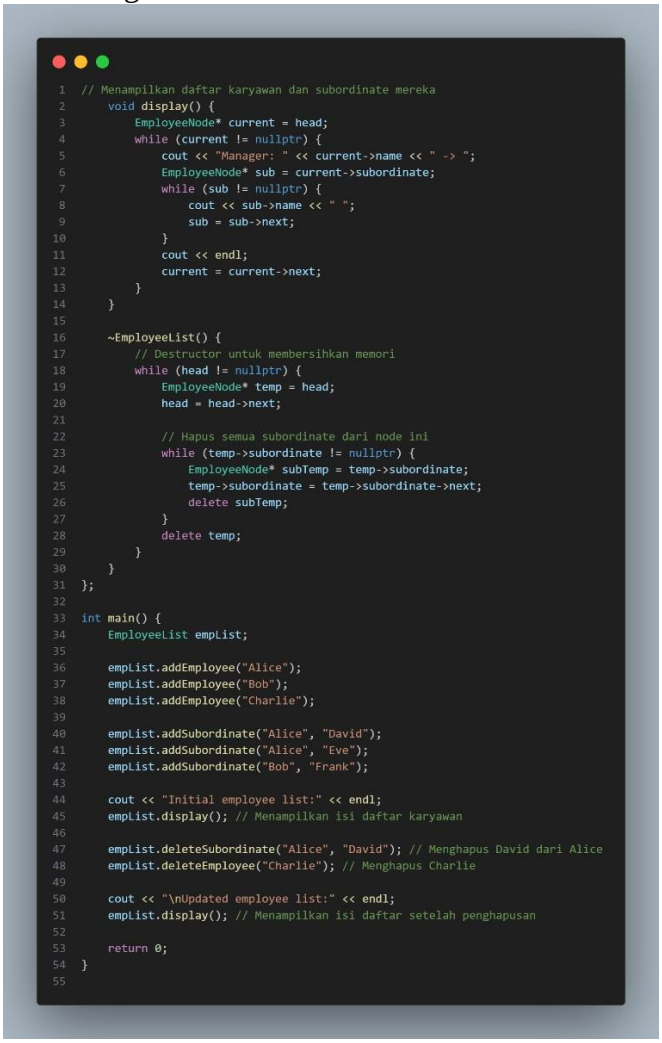
Hasil Run:

```
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
PS C:\Users\Farhan Kurniawan\Downloads\data karyawan\output>
```

IV. UNGUIDED

1. Kode Program:



```cpp
// 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;
}
```

```cpp
// Fungsi untuk menghapus proyek tertentu dari pegawai
void hapusProyek(Pegawai* pegawai, string namaProyek) {
    Proyek* temp = pegawai->headProyek;
    Proyek* prev = NULL;

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

    if (temp == NULL) {
        cout << "Proyek \"" << namaProyek << "\" tidak ditemukan.\n";
        return;
    }

    if (prev == NULL) { // proyek yang dihapus adalah proyek pertama
        pegawai->headProyek = temp->next;
    } else {
        prev->next = temp->next;
    }
    delete temp;
    cout << "Proyek \"" << namaProyek << "\" berhasil dihapus.\n";
}

// Fungsi untuk menampilkan data pegawai dan proyek mereka
void tampilkanData(Pegawai* headPegawai) {
    Pegawai* tempPegawai = headPegawai;
    while (tempPegawai != NULL) {
        cout << "Pegawai: " << tempPegawai->namaPegawai
             << " (ID: " << tempPegawai->idPegawai << ")\n";
        Proyek* tempProyek = tempPegawai->headProyek;
        if (tempProyek == NULL) {
            cout << "  Tidak ada proyek.\n";
        } else {
            while (tempProyek != NULL) {
                cout << "  - Proyek: " << tempProyek->namaProyek
                     << ", Durasi: " << tempProyek->durasi << " bulan\n";
                tempProyek = tempProyek->next;
            }
        }
        tempPegawai = tempPegawai->next;
        cout << endl;
    }
}

int main() {
    // Membuat daftar pegawai
    Pegawai* headPegawai = buatPegawai("Andi", "P001");
    headPegawai->next = buatPegawai("Budi", "P002");
    headPegawai->next->next = buatPegawai("Citra", "P003");

    // Menambahkan proyek ke pegawai
    tambahProyek(headPegawai, "Aplikasi Mobile", 12); // Proyek untuk Andi
    tambahProyek(headPegawai->next, "Sistem Akuntansi", 8); // Proyek untuk Budi
    tambahProyek(headPegawai->next->next, "E-commerce", 10); // Proyek untuk Citra

    // Menambahkan proyek baru ke Andi
    tambahProyek(headPegawai, "Analisis Data", 6);

    // Menghapus proyek "Aplikasi Mobile" dari Andi
    hapusProyek(headPegawai, "Aplikasi Mobile");

    // Menampilkan data pegawai dan proyek mereka
    cout << "\nData Pegawai dan Proyek:\n";
    tampilkanData(headPegawai);

    return 0;
}
```

Hasil Run:

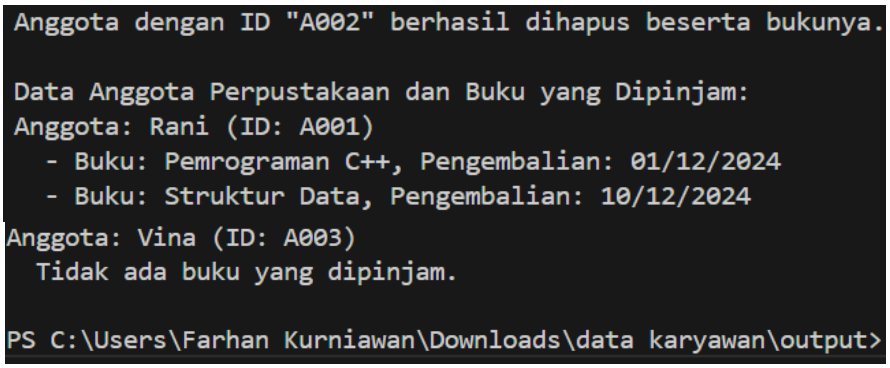
```
Proyek "Aplikasi Mobile" berhasil dihapus.

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

PS C:\Users\Farhan Kurniawan\Downloads\data karyawan\output>
```

2.kode

```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;
    }
}

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;
}
```

Hasil Run:



V. KESIMPULAN

Mempelajari Multi Linked List memberikan pemahaman mendalam tentang struktur data kompleks yang efektif. Oleh karena itu, mempelajari Multi Linked List sangat penting bagi pengembangan kemampuan dalam bidang teknologi informasi.