

# LAPORAN PRAKTIKUM

## MODUL 13

### MULTI LINKED LIST



**Nama :**

Candra Dinata (2311104061)

**Kelas :**

S1SE 07 02

**Dosen :**

Wahyu Andi Saputra

**PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK**

**FAKULTAS INFORMATIKA**

**TELKOM UNIVERSITY PURWOKERTO**

**2024**

## I. TUJUAN

Tujuan dari praktikum ini adalah agar mahasiswa memahami konsep dasar Multi Linked List, termasuk perbedaannya dengan struktur linked list lainnya, serta mampu mengimplementasikan struktur data Multi Linked List dalam C++ untuk memecahkan permasalahan yang melibatkan pengelolaan data hierarkis atau relasional secara dinamis. Praktikum ini juga bertujuan melatih mahasiswa dalam merancang, menambahkan, menghapus, dan menelusuri elemen-elemen pada Multi Linked List secara efisien, sehingga dapat meningkatkan pemahaman mereka dalam penerapan struktur data yang kompleks.

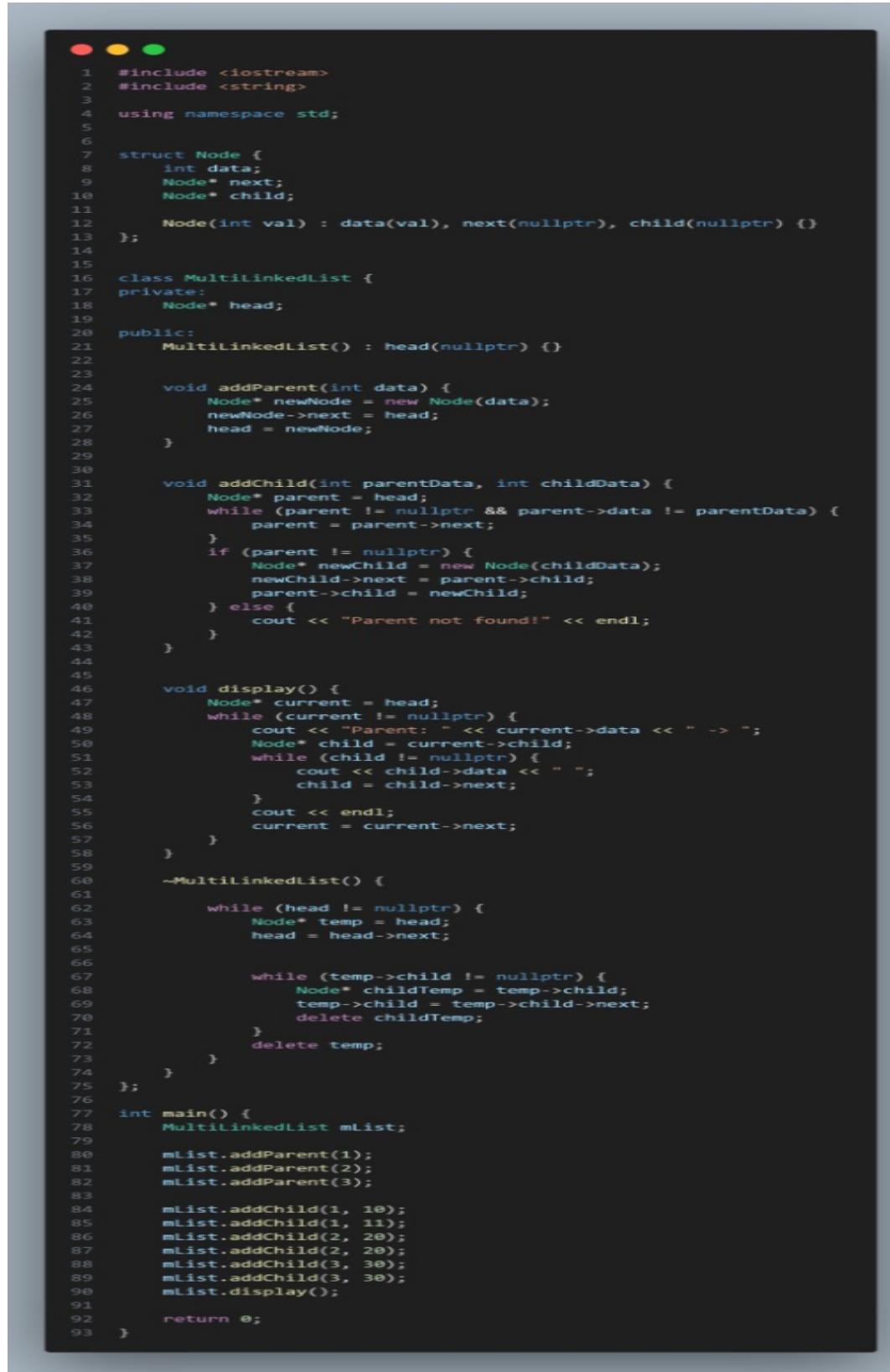
## II. DASAR TEORI

Multi Linked List adalah salah satu bentuk struktur data yang kompleks dan fleksibel, yang memungkinkan sebuah node memiliki lebih dari satu pointer untuk menghubungkan ke node lain, sehingga membentuk hubungan data yang bersifat multi-level atau multidimensi. Struktur ini sering digunakan untuk merepresentasikan hubungan hierarkis atau relasional, seperti representasi matriks, graf, atau tabel yang memiliki atribut berelasi. Tidak seperti Single Linked List atau Double Linked List, yang hanya memungkinkan traversal dalam satu atau dua arah, Multi Linked List memungkinkan akses ke beberapa cabang data dari satu node, tergantung pada jumlah pointer yang didefinisikan. Implementasi Multi Linked List memerlukan pemahaman mendalam tentang pointer, alokasi memori dinamis, serta algoritma traversal dan manipulasi node yang efisien. Dengan struktur ini, programmer dapat memodelkan hubungan data yang

kompleks, seperti relasi antar entitas pada database atau sistem manajemen data lainnya, secara lebih efektif.

### III. GUIDED

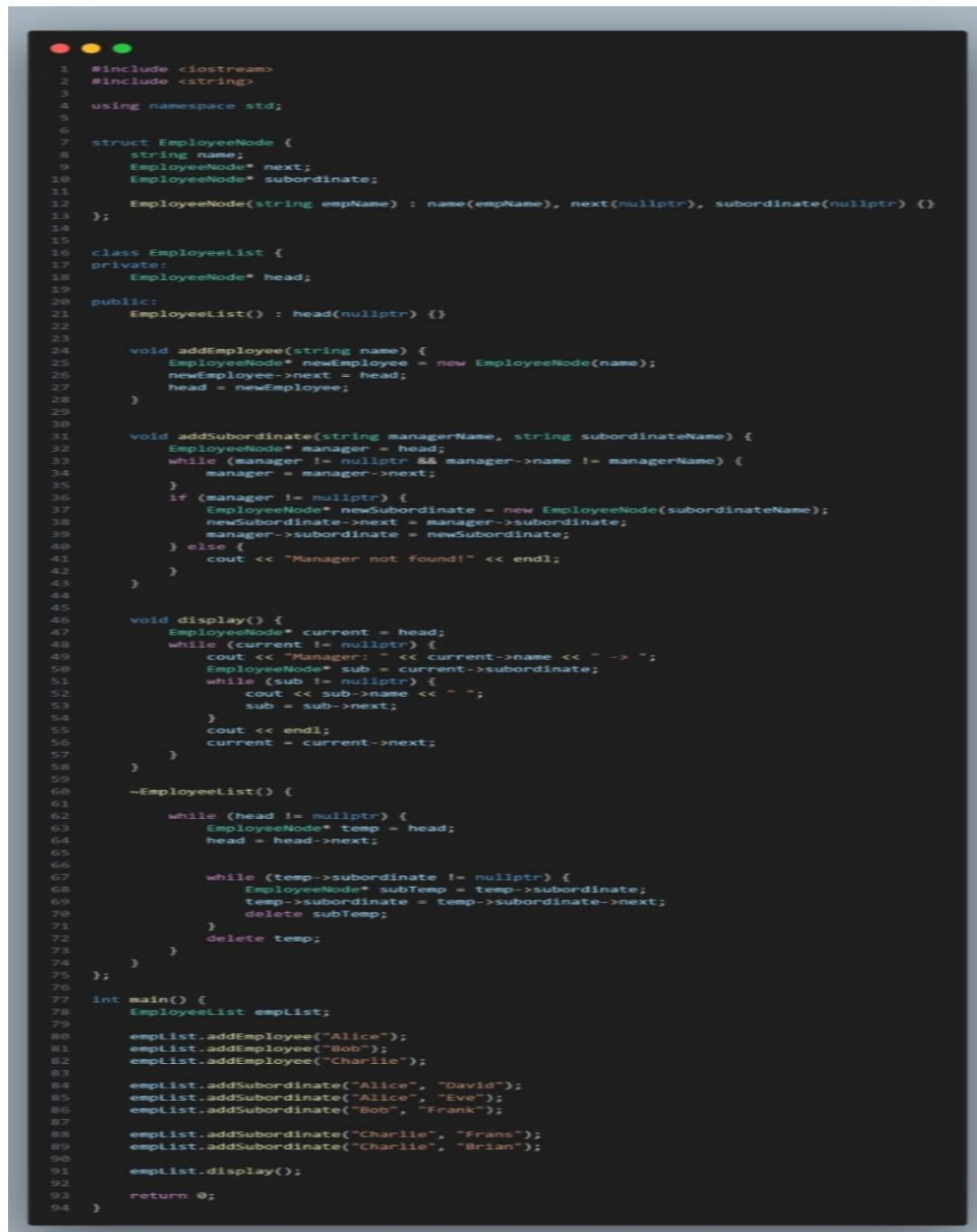
#### 1.



```
1 #include <iostream>
2 #include <string>
3
4 using namespace std;
5
6
7 struct Node {
8     int data;
9     Node* next;
10    Node* child;
11
12    Node(int val) : data(val), next(nullptr), child(nullptr) {}
13 };
14
15
16 class MultiLinkedList {
17 private:
18     Node* head;
19
20 public:
21     MultiLinkedList() : head(nullptr) {}
22
23
24     void addParent(int data) {
25         Node* newNode = new Node(data);
26         newNode->next = head;
27         head = newNode;
28     }
29
30
31     void addChild(int parentData, int childData) {
32         Node* parent = head;
33         while (parent != nullptr && parent->data != parentData) {
34             parent = parent->next;
35         }
36         if (parent != nullptr) {
37             Node* newChild = new Node(childData);
38             newChild->next = parent->child;
39             parent->child = newChild;
40         } else {
41             cout << "Parent not found!" << endl;
42         }
43     }
44
45
46     void display() {
47         Node* current = head;
48         while (current != nullptr) {
49             cout << "Parent: " << current->data << " -> ";
50             Node* child = current->child;
51             while (child != nullptr) {
52                 cout << child->data << " ";
53                 child = child->next;
54             }
55             cout << endl;
56             current = current->next;
57         }
58     }
59
60     ~MultiLinkedList() {
61
62         while (head != nullptr) {
63             Node* temp = head;
64             head = head->next;
65
66             while (temp->child != nullptr) {
67                 Node* childTemp = temp->child;
68                 temp->child = temp->child->next;
69                 delete childTemp;
70             }
71             delete temp;
72         }
73     }
74 };
75
76
77 int main() {
78     MultiLinkedList mList;
79
80     mList.addParent(1);
81     mList.addParent(2);
82     mList.addParent(3);
83
84     mList.addChild(1, 10);
85     mList.addChild(1, 11);
86     mList.addChild(2, 20);
87     mList.addChild(2, 20);
88     mList.addChild(3, 30);
89     mList.addChild(3, 30);
90
91     mList.display();
92
93 }
```

Kode ini mengimplementasikan struktur data Multi Linked List di C++, yang memungkinkan hubungan hierarkis antara elemen-elemen menggunakan dua pointer: satu untuk elemen "next" yang menghubungkan node-node dalam satu level, dan satu lagi untuk pointer "child" yang menghubungkan anak-anak dari setiap node. Kelas MultiLinkedList memiliki metode untuk menambahkan node induk (addParent), menambahkan anak ke node induk tertentu (addChild), serta menampilkan seluruh struktur data (display). Setiap node dalam daftar memiliki data integer, pointer ke node berikutnya (next), dan pointer ke anak pertama (child). Program ini membuat beberapa node induk, menambahkan anak ke masing-masing induk, dan menampilkan seluruh struktur data dengan mencetak node induk beserta anak-anaknya. Destruktor kelas mengelola alokasi memori dengan menghapus semua node dan anak-anaknya ketika objek MultiLinkedList dihancurkan, untuk mencegah kebocoran memori.

## 2.



```
1 #include <iostream>
2 #include <string>
3
4 using namespace std;
5
6
7 struct EmployeeNode {
8     string name;
9     EmployeeNode* next;
10    EmployeeNode* subordinate;
11 };
12 EmployeeNode(string empName) : name(empName), next(nullptr), subordinate(nullptr) {}
13 };
14
15
16 class EmployeeList {
17 private:
18     EmployeeNode* head;
19 public:
20     EmployeeList() : head(nullptr) {}
21
22
23     void addEmployee(string name) {
24         EmployeeNode* newEmployee = new EmployeeNode(name);
25         newEmployee->next = head;
26         head = newEmployee;
27     }
28
29
30     void addSubordinate(string managerName, string subordinateName) {
31         EmployeeNode* manager = head;
32         while (manager != nullptr && manager->name != managerName) {
33             manager = manager->next;
34         }
35         if (manager != nullptr) {
36             EmployeeNode* newSubordinate = new EmployeeNode(subordinateName);
37             newSubordinate->next = manager->subordinate;
38             manager->subordinate = newSubordinate;
39         } else {
40             cout << "Manager not found!" << endl;
41         }
42     }
43
44
45     void display() {
46         EmployeeNode* current = head;
47         while (current != nullptr) {
48             cout << "Manager: " << current->name << " -> ";
49             EmployeeNode* sub = current->subordinate;
50             while (sub != nullptr) {
51                 cout << sub->name << " - ";
52                 sub = sub->next;
53             }
54             cout << endl;
55             current = current->next;
56         }
57     }
58
59     ~EmployeeList() {
60
61         while (head != nullptr) {
62             EmployeeNode* temp = head;
63             head = head->next;
64
65
66             while (temp->subordinate != nullptr) {
67                 EmployeeNode* subTemp = temp->subordinate;
68                 temp->subordinate = temp->subordinate->next;
69                 delete subTemp;
70             }
71             delete temp;
72         }
73     }
74 };
75
76 int main() {
77     EmployeeList empList;
78
79     empList.addEmployee("Alice");
80     empList.addEmployee("Bob");
81     empList.addEmployee("Charlie");
82
83     empList.addSubordinate("Alice", "David");
84     empList.addSubordinate("Alice", "Eve");
85     empList.addSubordinate("Bob", "Frank");
86
87     empList.addSubordinate("Charlie", "Frans");
88     empList.addSubordinate("Charlie", "Brian");
89
90     empList.display();
91
92     return 0;
93 }
94 }
```

Kode ini mengimplementasikan struktur data Multi Linked List untuk mengelola daftar karyawan dalam sebuah perusahaan, dengan setiap karyawan dapat memiliki bawahan. Kelas EmployeeList memiliki metode untuk menambahkan karyawan ke dalam daftar (addEmployee), menambahkan bawahan pada karyawan yang ditunjuk sebagai manajer (addSubordinate), serta menampilkan seluruh daftar karyawan beserta bawahan mereka (display). Setiap node dalam daftar karyawan (EmployeeNode) menyimpan nama karyawan, pointer ke karyawan berikutnya (next), dan pointer ke bawahan pertama (subordinate). Program ini membuat beberapa karyawan, menambahkan bawahan ke masing-masing manajer, dan kemudian menampilkan struktur organisasi yang menunjukkan hubungan manajer dengan bawahan mereka. Destruktor kelas menangani penghapusan semua node dan bawahan mereka untuk mencegah kebocoran memori.

### 3.

```

1 //include <iostream>
2 //include <string>
3
4 using namespace std;
5
6 // Strukur data node karyawan
7 struct EmployeeNode {
8     string name; // Nama karyawan
9     EmployeeNode* next; // Pointer ke karyawan berikutnya
10    EmployeeNode* subordinate; // Pointer ke bawahan pertama
11
12    EmployeeNode(string empName, EmployeeNode* next, EmployeeNode* subordinate) {
13        name = empName;
14        this->next = next;
15        this->subordinate = subordinate;
16    }
17
18    ~EmployeeNode() {
19        delete next;
20        delete subordinate;
21    }
22
23    void addSubordinate(EmployeeNode* sub) {
24        subordinate = sub;
25    }
26
27    void display() {
28        cout << name << endl;
29        if (subordinate != NULL) {
30            subordinate->display();
31        }
32    }
33
34    void addEmployee(string name) {
35        EmployeeNode* manager = head;
36        while (manager != NULL && manager->name != name) {
37            manager = manager->next;
38        }
39
40        if (manager == NULL) { // jika manajer ditemu
41            EmployeeNode* newEmployee = new EmployeeNode(name);
42            newEmployee->next = head; // Memasangkan list karyawan setelahnya
43            head = newEmployee; // Mengubah head
44        }
45
46        // Menambahkan bawahan ke bawahan sebelumnya
47        void addSubordinate(string managerName, string subordinateName) {
48            EmployeeNode* manager = head;
49            while (manager != NULL && manager->name != managerName) {
50                manager = manager->next;
51            }
52
53            if (manager == NULL) { // jika manajer ditemu
54                EmployeeNode* newSubordinate = new EmployeeNode(subordinateName);
55                newSubordinate->next = manager->subordinate;
56                manager->subordinate = newSubordinate; // Mengubah bawahan sebelumnya
57            } else {
58                cout << "Manager not found!" << endl;
59            }
60        }
61
62        // Menghapus bawahan
63        void deleteEmployee(string name) {
64            EmployeeNode* current = head;
65            while (current != NULL && current->name != name) {
66                current = current->next;
67            }
68
69            if (current == NULL) { // jika karyawan ditemu
70                cout << "Employee not found!" << endl;
71            } else {
72                EmployeeNode* temp = current->next;
73                current->next = temp->next;
74                delete temp; // Menghapus node bawahan
75                cout << "Employee " << name << " deleted." << endl;
76            }
77        }
78
79        // Menghapus bawahan dari karyawan tertentu
80        void deleteSubordinate(string managerName, string subordinateName) {
81            EmployeeNode* manager = head;
82            while (manager != NULL && manager->name != managerName) {
83                manager = manager->next;
84            }
85
86            if (manager == NULL) { // jika manajer ditemu
87                EmployeeNode* currentSub = manager->subordinate;
88                while (currentSub != NULL && currentSub->name != subordinateName) {
89                    currentSub = currentSub->next;
90                }
91
92                if (currentSub == NULL) { // jika bawahan ditemu
93                    cout << "Employee " << managerName << " has no subordinates." << endl;
94                } else {
95                    cout << "Subordinate " << subordinateName << " deleted from " << managerName << endl;
96                }
97            } else {
98                cout << "Manager not found!" << endl;
99            }
100
101        // Menghapus daftar karyawan dan bawahan mereka
102        void employeeList() {
103            EmployeeNode* current = head;
104            while (current != NULL) {
105                EmployeeNode* sub = current->subordinate;
106                while (sub != NULL) {
107                    sub = sub->next;
108                }
109                current->subordinate = sub;
110                delete sub; // Menghapus node bawahan
111                cout << "Employee " << current->name << " deleted from " << current->subordinate->name << endl;
112            }
113            cout << endl;
114        }
115
116        ~EmployeeList() {
117            EmployeeNode* head = head;
118            while (head != NULL) {
119                EmployeeNode* temp = head;
120                head = head->next;
121                delete temp; // Menghapus node
122            }
123        }
124
125        void main() {
126            EmployeeList empList;
127
128            empList.addEmployee("Alice");
129            empList.addEmployee("Bob");
130            empList.addEmployee("Charlie");
131
132            empList.addSubordinate("Alice", "David");
133            empList.addSubordinate("Alice", "Eve");
134            empList.addSubordinate("Bob", "Frank");
135
136            cout << "Initial employee list:" << endl;
137            empList.display(); // Menampilkan list awal karyawan
138
139            empList.deleteSubordinate("Alice", "David"); // Menghapus David dari Alice
140            empList.deleteEmployee("Charlie"); // Menghapus Charlie
141
142            cout << "Updated employee list:" << endl;
143            empList.display(); // Menampilkan list daftar setelah penghapusan
144
145        return 0;
146    }
147
148}

```

Kode ini mengimplementasikan struktur data Multi-Linked List untuk mengelola karyawan dan bawahannya dalam sebuah perusahaan. Setiap karyawan (induk) dapat memiliki beberapa bawahan (subordinate), yang diwakili oleh node dalam struktur EmployeeNode. Kelas EmployeeList menyediakan metode untuk menambah karyawan (addEmployee), menambahkan bawahan ke karyawan tertentu (addSubordinate), menghapus karyawan (deleteEmployee), serta menghapus bawahan dari karyawan tertentu (deleteSubordinate). Program ini menampilkan daftar karyawan beserta bawahannya menggunakan metode display, dan menangani penghapusan baik karyawan maupun bawahannya. Destruktor kelas memastikan bahwa memori yang digunakan oleh node karyawan dan bawahan dibersihkan dengan benar setelah program selesai dijalankan. Program utama mengilustrasikan penambahan, penghapusan, dan tampilan data karyawan dan bawahannya.

## IV. UNGUIDED

1.

```
PS C:\Users\Candra Dinata\pertemuan1> cd 'c:\Users\Candra Dinata\pertemuan1\modddd13\output' & .\Un  
Sistem Manajemen Data Proyek dan Pegawai:  
1. Tambahkan Pegawai  
2. Tambahkan Proyek ke Pegawai  
3. Tambahkan Proyek Baru  
4. Hapus Proyek  
5. Tampilkan Data Pegawai  
6. Keluar  
Pilih opsi: ■
```

Kode di atas merupakan implementasi sistem manajemen data pegawai dan proyek menggunakan struktur data Multi Linked List dalam C++. Terdapat dua struktur data utama, yaitu Employee dan Project. Struktur Employee menyimpan informasi tentang pegawai, seperti nama, ID, serta pointer ke proyek-proyek yang terkait dengan pegawai tersebut, sedangkan struktur Project menyimpan informasi proyek seperti nama proyek, durasi, dan pointer ke proyek berikutnya. Fungsionalitas utama meliputi pembuatan pegawai dan proyek baru (createEmployee dan createProject), menambahkan proyek ke pegawai tertentu (addProject), menghapus proyek dari daftar proyek pegawai (removeProject), menambahkan pegawai ke daftar pegawai (addEmployee), serta menampilkan seluruh data pegawai beserta proyek-proyeknya (displayEmployees).

Dalam fungsi main, terdapat menu interaktif untuk mengelola data, seperti menambahkan pegawai, menambahkan proyek ke pegawai tertentu, menghapus proyek, dan menampilkan data pegawai. Data pegawai dihubungkan melalui linked list menggunakan pointer nextEmployee, sementara proyek untuk setiap pegawai dihubungkan melalui linked list menggunakan pointer nextProject. Dengan cara ini, setiap pegawai memiliki daftar proyek yang dapat bertambah atau berkurang secara dinamis. Program ini dirancang agar user dapat mengelola data pegawai dan proyek secara efisien, mencerminkan konsep Multi Linked List yang menghubungkan dua tingkat data dalam satu sistem.

## 2.

```

1 #include <iostream>
2 #include <iomanip>
3 #include <string>
4 using namespace std;
5
6 struct Book {
7     string title;
8     string returnDate;
9     Book* nextBook;
10 };
11 struct Member {
12     string memberName;
13     string memberID;
14     Book* bookHead;
15     Member* nextMember;
16 };
17 Member* createMember(string name, string id) {
18     Member* newMember = new Member;
19     newMember->memberName = name;
20     newMember->memberID = id;
21     newMember->bookHead = nullptr;
22     newMember->nextMember = nullptr;
23     return newMember;
24 }
25 Book* createBook(string title, string returnDate) {
26     Book* newBook = new Book;
27     newBook->title = title;
28     newBook->returnDate = returnDate;
29     newBook->nextBook = nullptr;
30     return newBook;
31 }
32 void addBook(Member* member, string title, string returnDate) {
33     Book* newBook = createBook(title, returnDate);
34     if (member->bookHead == nullptr) {
35         member->bookHead = newBook;
36     } else {
37         Book* temp = member->bookHead;
38         while (temp->nextBook != nullptr) {
39             temp = temp->nextBook;
40         }
41         temp->nextBook = newBook;
42     }
43 }
44 void removeMember(Member*& head, string id) {
45     Member* temp = head;
46     Member* prev = nullptr;
47     while (temp != nullptr && temp->memberID != id) {
48         prev = temp;
49         temp = temp->nextMember;
50     }
51     if (temp == nullptr) {
52         cout << "Anggota dengan ID " << id << " tidak ditemukan.\n";
53         return;
54     }
55     if (prev == nullptr) {
56         head = temp->nextMember;
57     } else {
58         prev->nextMember = temp->nextMember;
59     }
60     Book* bookTemp = temp->bookHead;
61     while (bookTemp != nullptr) {
62         Book* toDelete = bookTemp;
63         bookTemp = bookTemp->nextBook;
64         delete toDelete;
65     }
66     delete temp;
67     cout << "Anggota dengan ID " << id << " Beserta buku yang dipinjam berhasil dihapus.\n";
68 }
69 void displayMembers(Member* head) {
70     Member* tempMember = head;
71     int memberNumber = 1;
72     while (tempMember != nullptr) {
73         cout << "Anggota " << memberNumber << ": Nama = " << tempMember->memberName << ", ID = " << tempMember->memberID << "\n";
74         Book* tempBook = tempMember->bookHead;
75         int bookNumber = 1;
76         while (tempBook != nullptr) {
77             cout << " - Buku " << bookNumber << ": Judul = " << tempBook->title << ", Pengembalian = " << tempBook->returnDate << "\n";
78             tempBook = tempBook->nextBook;
79             bookNumber++;
80         }
81         tempMember = tempMember->nextMember;
82         memberNumber++;
83     }
84 }
85
86 int main() {
87     Member* head = nullptr;
88
89     head = createMember("Rani", "A001");
90     head->nextMember = createMember("Dito", "A002");
91     head->nextMember->nextMember = createMember("Vina", "A003");
92
93     cout << "Data Awal Anggota Perpustakaan:\n\n";
94     cout << "Anggota 1: Nama = " << Rani << ", ID = " << A001 << ".\n";
95     cout << "Anggota 2: Nama = " << Dito << ", ID = " << A002 << ".\n";
96     cout << "Anggota 3: Nama = " << Vina << ", ID = " << A003 << ".\n";
97
98     cout << "Setelah menambahkan buku yang dipinjam:\n\n";
99     addBook(head, "Pemrograman C++", "01/12/2024");
100    addBook(head->nextMember, "Algoritma Pemrograman", "15/12/2024");
101    displayMembers(head);
102    cout << "\nSetelah menambahkan buku baru:\n\n";
103    addBook(head, "Struktur Data", "10/12/2024");
104    displayMembers(head);
105
106    cout << "Setelah menghapus anggota Dito:\n\n";
107    removeMember(head, "A002");
108    displayMembers(head);
109    return 0;
110 }

```

```

Anggota 1: Nama = "Rani", ID = "A001".
Anggota 2: Nama = "Dito", ID = "A002".
Anggota 3: Nama = "Vina", ID = "A003".

Setelah menambahkan buku yang dipinjam:

Anggota 1: Nama = "Rani", ID = "A001".
- Buku 1: Judul = "Pemrograman C++", Pengembalian = "01/12/2024".
Anggota 2: Nama = "Dito", ID = "A002".
- Buku 1: Judul = "Algoritma Pemrograman", Pengembalian = "15/12/2024".
Anggota 3: Nama = "Vina", ID = "A003".

Setelah menambahkan buku baru:

Anggota 1: Nama = "Rani", ID = "A001".
- Buku 1: Judul = "Pemrograman C++", Pengembalian = "01/12/2024".
- Buku 2: Judul = "Struktur Data", Pengembalian = "10/12/2024".
Anggota 2: Nama = "Dito", ID = "A002".
- Buku 1: Judul = "Algoritma Pemrograman", Pengembalian = "15/12/2024".
Anggota 3: Nama = "Vina", ID = "A003".

Setelah menghapus anggota Dito:

Anggota dengan ID A002 Beserta buku yang dipinjam berhasil dihapus.
Anggota 1: Nama = "Rani", ID = "A001".
- Buku 1: Judul = "Pemrograman C++", Pengembalian = "01/12/2024".
- Buku 2: Judul = "Struktur Data", Pengembalian = "10/12/2024".
Anggota 2: Nama = "Vina", ID = "A003".

```

Kode di atas adalah implementasi sistem manajemen anggota perpustakaan dengan menggunakan struktur data linked list di C++. Program ini terdiri dari dua struktur utama, yaitu Member dan Book. Struktur Member berisi informasi tentang anggota perpustakaan, seperti nama, ID, dan daftar buku yang dipinjam yang diwakili oleh pointer bookHead. Struktur Book berisi informasi tentang buku yang dipinjam, termasuk judul dan tanggal pengembalian, dengan pointer nextBook untuk menghubungkan buku-buku yang dipinjam oleh anggota. Fungsi utama dalam program ini meliputi pembuatan anggota dan buku baru, penambahan buku yang dipinjam oleh anggota, penghapusan anggota beserta buku yang dipinjam, dan menampilkan daftar anggota beserta buku yang dipinjamnya.

Dalam fungsi main, tiga anggota perpustakaan dibuat dengan ID dan nama yang telah ditentukan. Kemudian, buku-buku ditambahkan ke daftar buku yang dipinjam oleh anggota tertentu menggunakan fungsi addBook. Fungsi removeMember digunakan untuk menghapus anggota tertentu, termasuk menghapus buku-buku yang dipinjam oleh anggota tersebut. Program ini juga menampilkan data anggota dan buku yang dipinjam sebelum dan sesudah perubahan, seperti penambahan buku dan penghapusan anggota, untuk menunjukkan bagaimana data dikelola secara dinamis menggunakan linked list. Struktur linked list memungkinkan program untuk menangani data anggota dan buku yang bisa bertambah atau berkurang secara fleksibel.

## V. KESIMPULAN

Kesimpulan dari praktikum ini adalah bahwa penggunaan struktur data Multi Linked List dalam C++ memungkinkan pemrogram untuk mengelola data yang memiliki hubungan kompleks secara dinamis. Dalam praktikum ini, mahasiswa dapat memahami bagaimana setiap elemen dalam sebuah linked list dapat terhubung dengan lebih dari satu elemen lainnya, baik dalam konteks relasi satu-ke-banyak ataupun banyak-ke-banyak. Hal ini berguna untuk aplikasi yang membutuhkan struktur data fleksibel dan efisien, seperti manajemen data pegawai dan proyek, atau anggota perpustakaan dan buku yang dipinjam. Dengan menggunakan pointer untuk menghubungkan elemen-elemen data, kita dapat menambahkan, menghapus, atau mengubah data dengan lebih mudah dan efisien.

Melalui implementasi Multi Linked List, mahasiswa juga dapat mempraktikkan pemahaman mereka tentang konsep dasar pointer, alokasi memori dinamis, dan traversal data dalam struktur data lebih dari satu level. Praktikum ini memperlihatkan pentingnya keterampilan dalam mengelola data yang saling terhubung, dan memberikan wawasan tentang bagaimana teknik ini dapat diterapkan dalam berbagai sistem informasi. Selain itu, mahasiswa diajak untuk memahami cara kerja struktur linked list yang dapat berkembang secara fleksibel sesuai dengan kebutuhan aplikasi, serta bagaimana mengimplementasikan fungsi dasar seperti penambahan dan penghapusan data dalam konteks relasi antar elemen data.