

**LAPORAN PRAKTIKUM**  
**Modul 13**  
**“MULTI LINKED LIST”**



**Disusun Oleh:**  
**Faishal Arif Setiawan -2311104066**  
**Kelas -SE 07 02**

**Dosen :**  
**Wahyu Andi Saputa 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 List* merupakan sekumpulan *list* yang berbeda yang memiliki suatu keterhubungan satu sama lain. Tiap elemen dalam *multi link list* dapat membentuk *list* sendiri. Biasanya ada yang bersifat sebagai *list* induk dan *list* anak .

Dalam multi-linked list, setiap pointer mengurutkan node berdasarkan kriteria tertentu. Setiap penunjuk dapat mengurutkan list berdasarkan kriteria yang berbeda dan membentuk list yang berbeda.

### **3. Guided**

```

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     mList.display();
91
92     return 0;
93 }
94

```

Struct Node:

```

struct Node {
    int data;
    Node* next;
    Node* child;

    Node(int val) : data(val), next(nullptr), child(nullptr) {}
};

```

Data: untuk menyimpan nilai dari node

Next:pointer menuju node berikutnya.

Child:pointer menuju child last dari node tersebut.

Kelas Multi Linked List:

Node\*head:pointer menuju node utama di daftar induk.

Fungsi addParent:

```
void addParent(int data) {  
    Node* newNode = new Node(data);  
    newNode->next = head;  
    head = newNode;  
}
```

Menambahkan node induk baru (parent) ke head list.

Node yang baru ditambahkan menjadi *head* dari daftar induk.

Fungsi addchild:

```
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;  
    }  
}
```

Menambahkan node anak ke dalam daftar child dari node induk yang sesuai (parentData).

Fungsi display:

```
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;  
    }  
}
```

Menampilkan seluruh **node induk** beserta daftar **node anak**.

Destructor Multi Linked List:

```
~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;  
    }  
};
```

Destructor ini digunakan untuk membebaskan memori yang dialokasikan secara dinamis. Untuk setiap node induk, destructor membebaskan semua child node sebelum membebaskan node induk itu sendiri.

Output:

```
PS D:\struktur data pemograman\GUIDED9\GUIDED10\output> & .\'guided1.exe'
Parent: 3 -> 30 30
Parent: 2 -> 20 20
Parent: 1 -> 11 10
PS D:\struktur data pemograman\GUIDED9\GUIDED10\output>
```

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
20 public:
21     EmployeeList() : head(nullptr) {}
22
23
24     void addEmployee(string name) {
25         EmployeeNode* newEmployee = new EmployeeNode(name);
26         newEmployee->next = head;
27         head = newEmployee;
28     }
29
30
31     void addSubordinate(string managerName, string subordinateName) {
32         EmployeeNode* manager = head;
33         while (manager != nullptr && manager->name != managerName) {
34             manager = manager->next;
35         }
36         if (manager != nullptr) {
37             EmployeeNode* newSubordinate = new EmployeeNode(subordinateName);
38             newSubordinate->next = manager->subordinate;
39             manager->subordinate = newSubordinate;
40         } else {
41             cout << "Manager not found!" << endl;
42         }
43     }
44
45
46     void display() {
47         EmployeeNode* current = head;
48         while (current != nullptr) {
49             cout << "Manager: " << current->name << " -> ";
50             EmployeeNode* sub = current->subordinate;
51             while (sub != nullptr) {
52                 cout << sub->name << " ";
53                 sub = sub->next;
54             }
55             cout << endl;
56             current = current->next;
57         }
58     }
59
60     ~EmployeeList() {
61         while (head != nullptr) {
62             EmployeeNode* temp = head;
63             head = head->next;
64
65             while (temp->subordinate != nullptr) {
66                 EmployeeNode* subTemp = temp->subordinate;
67                 temp->subordinate = temp->subordinate->next;
68                 delete subTemp;
69             }
70             delete temp;
71         }
72     }
73
74 };
75
76
77 int main() {
78     EmployeeList emplist;
79
80     emplist.addEmployee("Alice");
81     emplist.addEmployee("Bob");
82     emplist.addEmployee("Charlie");
83
84     emplist.addSubordinate("Alice", "David");
85     emplist.addSubordinate("Alice", "Eve");
86     emplist.addSubordinate("Bob", "Frank");
87
88     emplist.addSubordinate("Charlie", "Frans");
89     emplist.addSubordinate("Charlie", "Brian");
90
91     emplist.display();
92
93     return 0;
94 }
95
```

EmployeeNode:

```
struct EmployeeNode {  
    string name;  
    EmployeeNode* next;  
    EmployeeNode* subordinate;  
  
    EmployeeNode(string empName) : name(empName), next(nullptr), subordinate(nullptr) {}  
};
```

name: Menyimpan nama pegawai.

next: Pointer menuju node pegawai berikutnya dalam linked list utama.

subordinate: Pointer menuju linked list bawahannya (subordinate list).

Konstruktor: Menginisialisasi nama pegawai dan menyetel next serta subordinate ke nullptr.

Kelas EmployeeList: Kelas ini mengelola operasi-operasi utama pada struktur multi linked list pegawai.

AddEmployee:

```
void addEmployee(string name) {  
    EmployeeNode* newEmployee = new EmployeeNode(name);  
    newEmployee->next = head;  
    head = newEmployee;  
}
```

Menambahkan pegawai baru ke linked list utama.

Pegawai baru ditambahkan di depan list dengan memindahkan head ke node baru.

addSubordinate:

```
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;  
    }  
}
```

Menambahkan bawahan (subordinate) untuk seorang manager yang sudah ada.

Display:

```
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;  
    }  
}
```

Menampilkan seluruh pegawai dalam linked list utama beserta daftar bawahan mereka.

EmployeeList:

```
~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;  
    }  
};
```

Menghapus semua node pegawai dan bawahan mereka dari memori untuk mencegah kebocoran memori.



Output:

```
PS D:\struktur data pemograman\GUIDED9\GUIDED10\output> & .\'guided2.exe'
Manager: Charlie -> Brian Frans
Manager: Bob -> Frank
Manager: Alice -> Eve David
PS D:\struktur data pemograman\GUIDED9\GUIDED10\output>
```

3.

```
1 #include <iostream>
2 #include <istring>
3 using namespace std;
4
5 // Struktur untuk node karyawan
6 struct EmployeeNode {
7     string name; // Nama karyawan
8     EmployeeNode* next; // pointer ke karyawan berikutnya
9     EmployeeNode* subordinate; // pointer ke subordinate pertama
10 };
11
12 EmployeeNode* createEmployee(string name, EmployeeNode* next, EmployeeNode* subordinate) {
13     EmployeeNode* newNode = new EmployeeNode();
14     newNode->name = name;
15     newNode->next = next;
16     newNode->subordinate = subordinate;
17     return newNode;
18 }
19
20 // Fungsi untuk menambahkan karyawan ke linked list
21 void addEmployee(string name) {
22     EmployeeNode* newNode = createEmployee(name, nullptr, nullptr);
23     EmployeeNode* head = nullptr;
24     if (head == nullptr) {
25         head = newNode;
26     } else {
27         EmployeeNode* current = head;
28         while (current->next != nullptr) {
29             current = current->next;
30         }
31         current->next = newNode;
32     }
33 }
34
35 // Fungsi untuk mencari karyawan berdasarkan nama
36 EmployeeNode* findEmployee(string name) {
37     EmployeeNode* current = head;
38     while (current != nullptr) {
39         if (current->name == name) {
40             return current;
41         }
42         current = current->next;
43     }
44     return nullptr;
45 }
46
47 // Fungsi untuk menghapus karyawan berdasarkan nama
48 void deleteEmployee(string name) {
49     EmployeeNode* current = head;
50     EmployeeNode* prev = nullptr;
51     while (current != nullptr) {
52         if (current->name == name) {
53             if (prev == nullptr) {
54                 head = current->next;
55             } else {
56                 prev->next = current->next;
57             }
58             delete current;
59             return;
60         }
61         prev = current;
62         current = current->next;
63     }
64 }
65
66 // Fungsi untuk menampilkan linked list
67 void displayList() {
68     EmployeeNode* current = head;
69     while (current != nullptr) {
70         cout << "Employee: " << current->name << endl;
71         current = current->next;
72     }
73 }
74
75 // Fungsi untuk menampilkan subordinate
76 void displaySubordinate(string name) {
77     EmployeeNode* current = findEmployee(name);
78     if (current == nullptr) {
79         cout << "Employee not found" << endl;
80         return;
81     }
82     EmployeeNode* sub = current->subordinate;
83     while (sub != nullptr) {
84         cout << "Subordinate: " << sub->name << endl;
85         sub = sub->next;
86     }
87 }
88
89 // Fungsi untuk menghapus subordinate
90 void deleteSubordinate(string name, string subordinateName) {
91     EmployeeNode* current = findEmployee(name);
92     if (current == nullptr) {
93         cout << "Employee not found" << endl;
94         return;
95     }
96     EmployeeNode* sub = current->subordinate;
97     EmployeeNode* prev = nullptr;
98     while (sub != nullptr) {
99         if (sub->name == subordinateName) {
100             if (prev == nullptr) {
101                 current->subordinate = sub->next;
102             } else {
103                 prev->subordinate = sub->next;
104             }
105             delete sub;
106             return;
107         }
108         prev = sub;
109         sub = sub->next;
110     }
111 }
112
113 // Fungsi untuk menampilkan subordinate
114 void displaySubordinateList() {
115     EmployeeNode* current = head;
116     while (current != nullptr) {
117         cout << "Manager: " << current->name << endl;
118         displaySubordinate(current->name);
119         current = current->next;
120     }
121 }
122
123 // Fungsi untuk menghapus subordinate
124 void deleteSubordinateList() {
125     EmployeeNode* current = head;
126     while (current != nullptr) {
127         deleteSubordinate(current->name, current->subordinate->name);
128         current = current->next;
129     }
130 }
131
132 // Fungsi untuk menampilkan linked list
133 void displayList() {
134     displayList();
135 }
136
137 // Fungsi untuk menghapus linked list
138 void deleteList() {
139     deleteList();
140 }
141
142 // Fungsi untuk menampilkan subordinate
143 void displaySubordinateList() {
144     displaySubordinateList();
145 }
146
147 // Fungsi untuk menghapus subordinate
148 void deleteSubordinateList() {
149     deleteSubordinateList();
150 }
151
152 // Fungsi untuk menampilkan linked list
153 void displayList() {
154     displayList();
155 }
```

Struct Employee:

```
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) {}  
};
```

1.

name: Menyimpan nama karyawan.

next: Pointer menuju karyawan berikutnya di linked list utama.

subordinate: Pointer menuju linked list yang berisi bawahan dari karyawan tersebut.

addemployee:

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

Menambahkan karyawan baru ke linked list utama.

Node karyawan baru ditambahkan di depan list (menjadi head).

Addsubordinate:

```
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  
        manager->subordinate = newSubordinate; // Memperbarui subordinate  
    } else {  
        cout << "Manager not found!" << endl;  
    }  
}
```

Menambahkan bawahan ke karyawan tertentu (manajer).

Deleteemployee:

```
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 karyawan dari linked list utama.

Deletesubordinate:

```
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;
    }
}
```

Menghapus bawahan dari seorang manajer.

Display:

```
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;  
    }  
}
```

Menampilkan daftar karyawan dan subordinate mereka

Output:

```
Updated employee list:  
Manager: Bob -> Frank  
Manager: Alice -> Eve  
PS D:\struktur data pemograman\GUIDED9\GUIDED10\output> █
```

#### 4. Unguided

##### 1.

```

1 #include <iostream>
2 #include <string>
3
4 using namespace std;
5
6 struct PegawaiNode {
7     string nama;
8     string ID;
9     PegawaiNode* next;
10    struct ProyekNode* proyekHead;
11 };
12
13 struct ProyekNode {
14     string nama;
15     int durasi;
16     ProyekNode* next;
17 };
18
19 PegawaiNode* createPegawaiNode(string nama, string ID) {
20     PegawaiNode* newnode = new PegawaiNode;
21     newnode->nama = nama;
22     newnode->ID = ID;
23     newnode->next = NULL;
24     newnode->proyekHead = NULL;
25     return newnode;
26 }
27
28 ProyekNode* createProyekNode(string nama, int durasi) {
29     ProyekNode* newnode = new ProyekNode;
30     newnode->nama = nama;
31     newnode->durasi = durasi;
32     newnode->next = NULL;
33     return newnode;
34 }
35
36 PegawaiNode* addPegawai(PegawaiNode* head, string nama, string ID) {
37     PegawaiNode* newnode = createPegawaiNode(nama, ID);
38     if (head == NULL) {
39         return newnode;
40     }
41     PegawaiNode* temp = head;
42     while (temp->next != NULL) {
43         temp = temp->next;
44     }
45     temp->next = newnode;
46     return head;
47 }
48
49 void addProyek(PegawaiNode* head, string pegawaiID, string namaProyek, int durasi) {
50     PegawaiNode* temp = head;
51     while (temp != NULL) {
52         if (temp->ID == pegawaiID) {
53             ProyekNode* newproyek = createProyekNode(namaProyek, durasi);
54             if (temp->proyekHead == NULL) {
55                 temp->proyekHead = newproyek;
56             } else {
57                 ProyekNode* proyekTemp = temp->proyekHead;
58                 while (proyekTemp->next != NULL) {
59                     proyekTemp = proyekTemp->next;
60                 }
61                 proyekTemp->next = newproyek;
62             }
63             return;
64         }
65         temp = temp->next;
66     }
67 }
68
69 void removeProyek(PegawaiNode* head, string pegawaiID, string namaProyek) {
70     PegawaiNode* temp = head;
71     while (temp != NULL) {
72         if (temp->ID == pegawaiID) {
73             ProyekNode* proyekTemp = temp->proyekHead;
74             ProyekNode* prev = NULL;
75             while (proyekTemp != NULL) {
76                 if (proyekTemp->nama == namaProyek) {
77                     if (prev == NULL) {
78                         temp->proyekHead = proyekTemp->next;
79                     } else {
80                         prev->next = proyekTemp->next;
81                     }
82                     delete proyekTemp;
83                     return;
84                 }
85                 prev = proyekTemp;
86                 proyekTemp = proyekTemp->next;
87             }
88             temp = temp->next;
89         }
90     }
91 }
92
93 void displayPegawai(PegawaiNode* head) {
94     PegawaiNode* temp = head;
95     cout << "Daftar Pegawai dan Proyek:\n";
96     while (temp != NULL) {
97         cout << "Nama Pegawai: " << temp->nama << ", ID: " << temp->ID << "\n";
98         ProyekNode* proyekTemp = temp->proyekHead;
99         if (proyekTemp == NULL) {
100             cout << " - Tidak ada proyek.\n";
101         } else {
102             while (proyekTemp != NULL) {
103                 cout << "Proyek: " << proyekTemp->nama << ", Durasi: " << proyekTemp->durasi << " bulan\n";
104                 proyekTemp = proyekTemp->next;
105             }
106             temp = temp->next;
107         }
108     }
109 }
110
111 int main() {
112     PegawaiNode* headPegawai = NULL;
113
114     // 1. Masukkan data pegawai
115     headPegawai = addPegawai(headPegawai, "Andi", "P001");
116     headPegawai = addPegawai(headPegawai, "Budi", "P002");
117     headPegawai = addPegawai(headPegawai, "Citra", "P003");
118
119     // 2. Tambahkan proyek ke pegawai
120     addProyek(headPegawai, "P001", "Aplikasi Mobile", 12); // untuk Andi
121     addProyek(headPegawai, "P002", "Sistem Akuntansi", 8); // untuk Budi
122     addProyek(headPegawai, "P003", "E-commerce", 10); // untuk Citra
123
124     // 3. Tambahkan proyek baru
125     addProyek(headPegawai, "P001", "Analisis Data", 6); // untuk Andi
126
127     // 4. Hapus proyek "Aplikasi Mobile" dari Andi
128     removeProyek(headPegawai, "P001", "Aplikasi Mobile");
129
130     // 5. Tampilkan data pegawai dan proyek mereka
131     displayPegawai(headPegawai);
132
133     return 0;
134 }
135

```

## Output:

Data Pegawai dan Proyek:

Nama Pegawai: Andi, ID: P001

- Proyek: Analisis Data, Durasi: 6 bulan

Nama Pegawai: Budi, ID: P002

- Proyek: Sistem Akuntansi, Durasi: 8 bulan

Nama Pegawai: Citra, ID: P003

- Proyek: E-commerce, Durasi: 10 bulan

PS D:\struktur data pemograman\GUIDED9\GUIDED10\output> █

## 2.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 // Struktur untuk data buku
6 typedef struct Buku {
7     char judul[100];
8     char pengembalian[15];
9     struct Buku *next;
10 } Buku;
11
12 // Struktur untuk data anggota
13 typedef struct Anggota {
14     char nama[50];
15     char id[10];
16     Buku *buku_head; // Pointer ke daftar buku yang dipinjam anggota
17     struct Anggota *next;
18 } Anggota;
19
20 // Fungsi untuk menambahkan anggota baru
21 Anggota *tambahAnggota(Anggota *head, char nama[], char id[]) {
22     Anggota *newAnggota = (Anggota *)malloc(sizeof(Anggota));
23     strcpy(newAnggota->nama, nama);
24     strcpy(newAnggota->id, id);
25     newAnggota->buku_head = NULL; // Inisialisasi buku kosong
26     newAnggota->next = head;
27     return newAnggota;
28 }
29
30 // Fungsi untuk menambahkan buku ke anggota
31 void tambahBuku(Anggota *head, char id[], char judul[], char pengembalian[]) {
32     Anggota *anggota = head;
33     while (anggota != NULL) {
34         if (strcmp(anggota->id, id) == 0) {
35             Buku *newBuku = (Buku *)malloc(sizeof(Buku));
36             strcpy(newBuku->judul, judul);
37             strcpy(newBuku->pengembalian, pengembalian);
38             newBuku->next = anggota->buku_head;
39             anggota->buku_head = newBuku;
40             return;
41         }
42         anggota = anggota->next;
43     }
44     printf("Anggota dengan ID %s tidak ditemukan.\n", id);
45 }
46
47 // Fungsi untuk menghapus anggota dan buku yang dipinjam
48 Anggota *hapusAnggota(Anggota *head, char id[]) {
49     Anggota *current = head;
50     Anggota *prev = NULL;
51
52     while (current != NULL) {
53         if (strcmp(current->id, id) == 0) {
54             // Hapus semua buku yang dipinjam anggota
55             Buku *buku = current->buku_head;
56             while (buku != NULL) {
57                 Buku *temp = buku;
58                 buku = buku->next;
59                 free(temp);
60             }
61
62             // Hapus anggota
63             if (prev == NULL) {
64                 head = current->next;
65             } else {
66                 prev->next = current->next;
67             }
68             free(current);
69             printf("Anggota dengan ID %s telah dihapus beserta buku yang dipinjam.\n", id);
70             return head;
71         }
72         prev = current;
73         current = current->next;
74     }
75     printf("Anggota dengan ID %s tidak ditemukan.\n", id);
76     return head;
77 }
78
79 // Fungsi untuk menampilkan seluruh data anggota dan buku yang dipinjam
80 void tampilkanData(Anggota *head) {
81     Anggota *current = head;
82     while (current != NULL) {
83         printf("Anggota: %s (ID: %s)\n", current->nama, current->id);
84         Buku *buku = current->buku_head;
85         while (buku != NULL) {
86             printf("    Buku: %s, Pengembalian: %s\n", buku->judul, buku->pengembalian);
87             buku = buku->next;
88         }
89         current = current->next;
90     }
91 }
92
93 // Main program
94 int main() {
95     Anggota *head = NULL;
96
97     // 1. Tambahkan data anggota
98     head = tambahAnggota(head, "Rani", "A001");
99     head = tambahAnggota(head, "Dito", "A002");
100     head = tambahAnggota(head, "Vina", "A003");
101
102     // 2. Tambahkan buku yang dipinjam
103     tambahBuku(head, "A001", "Pemrograman C++", "01/12/2024");
104     tambahBuku(head, "A002", "Algoritma Pemrograman", "15/12/2024");
105
106     // 3. Tambahkan buku baru
107     tambahBuku(head, "A001", "Struktur Data", "10/12/2024");
108
109     // 4. Hapus anggota Dito beserta buku yang dipinjam
110     head = hapusAnggota(head, "A002");
111
112     // 5. Tampilkan seluruh data anggota dan buku yang dipinjam
113     printf("\nData Anggota dan Buku yang Dipinjam:\n");
114     tampilkanData(head);
115
116     return 0;
117 }
118

```

**Output:**

```
Data Anggota dan Buku yang Dipinjam:
```

```
Anggota: Vina (ID: A003)
```

```
Anggota: Rani (ID: A001)
```

```
- Buku: Struktur Data, Pengembalian: 10/12/2024
```

```
- Buku: Pemrograman C++, Pengembalian: 01/12/2024
```

```
PS D:\struktur data pemograman\GUIDED9\GUIDED10\output> █
```

## 5. Kesimpulan

praktikum ini tentang Multi Linked List memberikan pemahaman mendalam tentang bagaimana struktur data ini digunakan untuk merepresentasikan hubungan hierarki antara node induk (parent) dan node anak (child). Melalui implementasi fungsi seperti `addEmployee`, `addSubordinate`, `deleteEmployee`, dan `deleteSubordinate`, kita dapat mengelola data secara efisien, baik untuk menambah, menampilkan, maupun menghapus node pada list. Struktur ini sangat berguna untuk menyelesaikan permasalahan yang melibatkan relasi hierarkis, seperti manajemen data karyawan dan bawahan, serta penerapannya dapat dikembangkan lebih lanjut dalam berbagai studi kasus lain yang membutuhkan keterhubungan antar elemen secara fleksibel dan terorganisir.