

**LAPORAN PRAKTIKUM
STRUKTUR DATA
MODUL 13
“ MULTI LINKED LIST”**



Disusun Oleh:
Dhiya Ulhaq Ramadhan 2211104053
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

- Memahami struktur dan implementasi Multi Linked List sebagai struktur data yang menghubungkan beberapa list berbeda
- Mengidentifikasi konsep list induk dan list anak dalam Multi Linked List
- Menguasai operasi dasar Multi Linked List seperti insert dan delete pada list induk maupun anak
- Mengimplementasikan ADT Multi Linked List dalam penyelesaian studi kasus yang melibatkan data bertingkat

2. Landasan Teori

Multi Linked List merupakan struktur data yang terdiri dari beberapa list yang saling terhubung satu sama lain, dimana setiap elemen dalam multi linked list dapat membentuk list tersendiri. Dalam implementasinya, Multi Linked List biasanya memiliki list yang bersifat sebagai induk dan list yang bersifat sebagai anak. List induk berperan sebagai list utama yang menaungi list-list anak di bawahnya. Setiap elemen dalam list induk dapat memiliki pointer yang menunjuk ke sebuah list anak.

Operasi dasar dalam Multi Linked List meliputi insert dan delete baik pada list induk maupun list anak. Untuk operasi pada list anak, perlu diketahui terlebih dahulu elemen induk yang terkait. Saat melakukan penghapusan elemen induk, seluruh elemen anak yang terhubung dengan induk tersebut juga harus dihapus untuk menjaga integritas struktur data. Implementasi Multi Linked List memerlukan manajemen memori yang baik karena melibatkan alokasi dan dealokasi memori untuk elemen-elemen dalam kedua jenis list.

3. Guided 1

Source code :

```

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

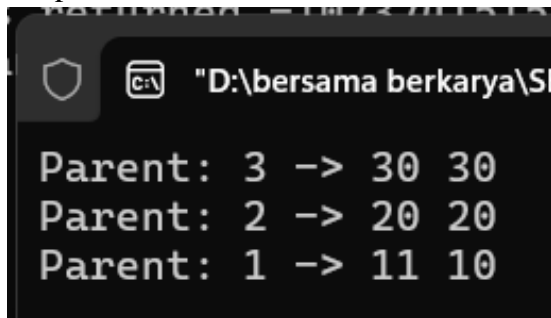
```

```

70
71 int main() {
72     MultiLinkedList mList;
73
74     mList.addParent(1);
75     mList.addParent(2);
76     mList.addParent(3);
77
78     mList.addChild(1, 10);
79     mList.addChild(1, 11);
80     mList.addChild(2, 20);
81     mList.addChild(2, 20);
82     mList.addChild(3, 30);
83     mList.addChild(3, 30);
84     mList.display();
85
86     return 0;
87 }

```

Output :



```

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

```

Guided 2

Source code

```

1  #include <iostream>
2  #include <string>
3
4  using namespace std;
5
6  struct EmployeeNode {
7      string name;
8      EmployeeNode* next;
9      EmployeeNode* subordinate;
10
11      EmployeeNode(string empName) : name(empName), next(nullptr), subordinate(nullptr) {}
12  };
13
14  class EmployeeList {
15  private:
16      EmployeeNode* head;
17
18  public:
19      EmployeeList() : head(nullptr) {}
20
21      void addEmployee(string name) {
22          EmployeeNode* newEmployee = new EmployeeNode(name);
23          newEmployee->next = head;
24          head = newEmployee;
25      }
26
27      void addSubordinate(string managerName, string subordinateName) {
28          EmployeeNode* manager = head;
29          while (manager != nullptr && manager->name != managerName) {
30              manager = manager->next;
31          }

```

```

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

```

Output

```

D:\bersama berkarya\SEMES  x  +
Manager: Charlie -> Brian Frans
Manager: Bob -> Frank
Manager: Alice -> Eve David

```

Guided 3

```

1  #include <iostream>
2  #include <string>
3
4  using namespace std;
5
6  // Struktur untuk node karyawan
7  struct EmployeeNode {
8      string name; // Nama karyawan
9      EmployeeNode* next; // Pointer ke karyawan berikutnya
10     EmployeeNode* subordinate; // Pointer ke subordinate pertama
11
12     EmployeeNode(string empName) : name(empName), next(nullptr), subordinate(nullptr) {}
13 };
14
15 // Kelas untuk Multi-Linked List Karyawan
16 class EmployeeList {
17 private:
18     EmployeeNode* head; // Pointer ke kepala list
19
20 public:
21     EmployeeList() : head(nullptr) {}
22
23     // Menambahkan karyawan (induk)
24     void addEmployee(string name) {
25         EmployeeNode* newEmployee = new EmployeeNode(name);
26         newEmployee->next = head; // Menyambungkan ke karyawan sebelumnya
27         head = newEmployee; // Memperbarui head
28     }
29
30     // Menambahkan subordinate ke karyawan tertentu
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) { // Jika manager ditemukan
37             EmployeeNode* newSubordinate = new EmployeeNode(subordinateName);
38             newSubordinate->next = manager->subordinate; // Menyambungkan ke subordinate sebelumnya
39             manager->subordinate = newSubordinate; // Memperbarui subordinate
40         } else {
41             cout << "Manager not found!" << endl;
42         }
43     }
44
45     // Menghapus karyawan (induk)
46     void deleteEmployee(string name) {
47         EmployeeNode** current = &head;
48         while (*current != nullptr && (*current)->name != name) {
49             current = &((*current)->next);
50         }
51         if (*current != nullptr) { // Jika karyawan ditemukan
52             EmployeeNode* toDelete = *current;
53             *current = (*current)->next;
54
55             // Hapus semua subordinate dari node ini
56             while (toDelete->subordinate != nullptr) {
57                 EmployeeNode* subTemp = toDelete->subordinate;
58                 toDelete->subordinate = toDelete->subordinate->next;
59                 delete subTemp;
60             }
61             delete toDelete;
62         }

```

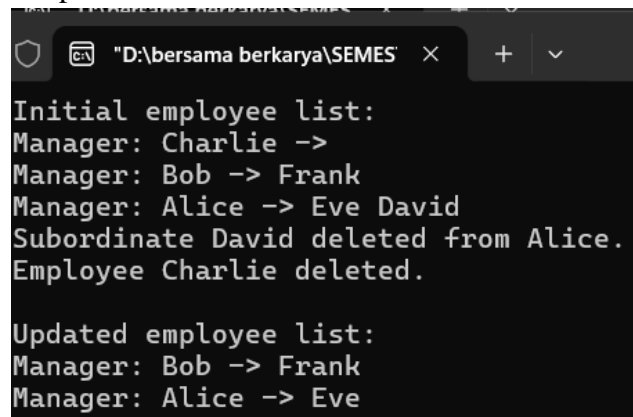
```

63         cout << "Employee " << name << " deleted." << endl;
64     } else {
65         cout << "Employee not found!" << endl;
66     }
67 }
68
69 // Menghapus subordinate dari karyawan tertentu
70 void deleteSubordinate(string managerName, string subordinateName) {
71     EmployeeNode* manager = head;
72     while (manager != nullptr && manager->name != managerName) {
73         manager = manager->next;
74     }
75
76     if (manager != nullptr) { // Jika manager ditemukan
77         EmployeeNode** currentSub = &(manager->subordinate);
78         while (*currentSub != nullptr && (*currentSub)->name != subordinateName) {
79             currentSub = &((*currentSub)->next);
80         }
81
82         if (*currentSub != nullptr) { // Jika subordinate ditemukan
83             EmployeeNode* toDelete = *currentSub;
84             *currentSub = (*currentSub)->next; // Menghapus dari list
85
86             delete toDelete; // Menghapus node subordinate
87             cout << "Subordinate " << subordinateName << " deleted from " << managerName << "." << endl;
88         } else {
89             cout << "Subordinate not found!" << endl;
90         }
91     } else {
92         cout << "Manager not found!" << endl;
93     }
94 }
95
96 // Menampilkan daftar karyawan dan subordinate mereka
97 void display() {
98     EmployeeNode* current = head;
99     while (current != nullptr) {
100         cout << "Manager: " << current->name << " -> ";
101         EmployeeNode* sub = current->subordinate;
102         while (sub != nullptr) {
103             cout << sub->name << " ";
104             sub = sub->next;
105         }
106         cout << endl;
107         current = current->next;
108     }
109 }
110
111 ~EmployeeList() {
112     // Destructor untuk membersihkan memori
113     while (head != nullptr) {
114         EmployeeNode* temp = head;
115         head = head->next;
116
117         // Hapus semua subordinate dari node ini
118         while (temp->subordinate != nullptr) {
119             EmployeeNode* subTemp = temp->subordinate;
120             temp->subordinate = temp->subordinate->next;
121             delete subTemp;
122         }
123         delete temp;
124     }

```

```
125     }  
126   };  
127  
128   int main() {  
129       EmployeeList empList;  
130  
131       empList.addEmployee("Alice");  
132       empList.addEmployee("Bob");  
133       empList.addEmployee("Charlie");  
134  
135       empList.addSubordinate("Alice", "David");  
136       empList.addSubordinate("Alice", "Eve");  
137       empList.addSubordinate("Bob", "Frank");  
138  
139       cout << "Initial employee list:" << endl;  
140       empList.display(); // Menampilkan isi daftar karyawan  
141  
142       empList.deleteSubordinate("Alice", "David"); // Menghapus David dari Alice  
143       empList.deleteEmployee("Charlie"); // Menghapus Charlie  
144  
145       cout << "\nUpdated employee list:" << endl;  
146       empList.display(); // Menampilkan isi daftar setelah penghapusan  
147  
148       return 0;  
149   }
```

Output



```
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
```


UNGUIDED

1. Manajemen Data Pegawai dan Proyek

Buatlah program menggunakan Multi Linked List untuk menyimpan data pegawai dan proyek yang dikelola setiap pegawai.

- Setiap pegawai memiliki data: Nama Pegawai dan ID Pegawai.
- Setiap proyek memiliki data: Nama Proyek** dan **Durasi (bulan).

Instruksi:

- Masukkan data pegawai berikut:
 - Pegawai 1: Nama = "Andi", ID = "P001".
 - Pegawai 2: Nama = "Budi", ID = "P002".
 - Pegawai 3: Nama = "Citra", ID = "P003".
- Tambahkan proyek ke pegawai:
 - Proyek 1: Nama = "Aplikasi Mobile", Durasi = 12 bulan (Untuk Andi).
 - Proyek 2: Nama = "Sistem Akuntansi", Durasi = 8 bulan (Untuk Budi).
 - Proyek 3: Nama = "E-commerce", Durasi = 10 bulan (Untuk Citra).
- Tambahkan proyek baru:
 - Proyek 4: Nama = "Analisis Data", Durasi = 6 bulan (Untuk Andi).
- Hapus proyek "Aplikasi Mobile" dari Andi.
- Tampilkan data pegawai dan proyek mereka.

Jawaban :

Source code

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  // Struktur untuk node Proyek
6  struct Proyek {
7      string nama;
8      int durasi;
9      Proyek* next;
10 };
11
12 // Struktur untuk node Pegawai
13 struct Pegawai {
14     string nama;
15     string id;
16     Proyek* proyek;
17     Pegawai* next;
18 };
19
20 // Fungsi untuk membuat pegawai baru
21 Pegawai* buatPegawai(string nama, string id) {
22     Pegawai* pegawaiBaru = new Pegawai;
23     pegawaiBaru->nama = nama;
24     pegawaiBaru->id = id;
25     pegawaiBaru->proyek = NULL;
26     pegawaiBaru->next = NULL;
27     return pegawaiBaru;
28 }
29
30 // Fungsi untuk membuat proyek baru
31 Proyek* buatProyek(string nama, int durasi) {
```

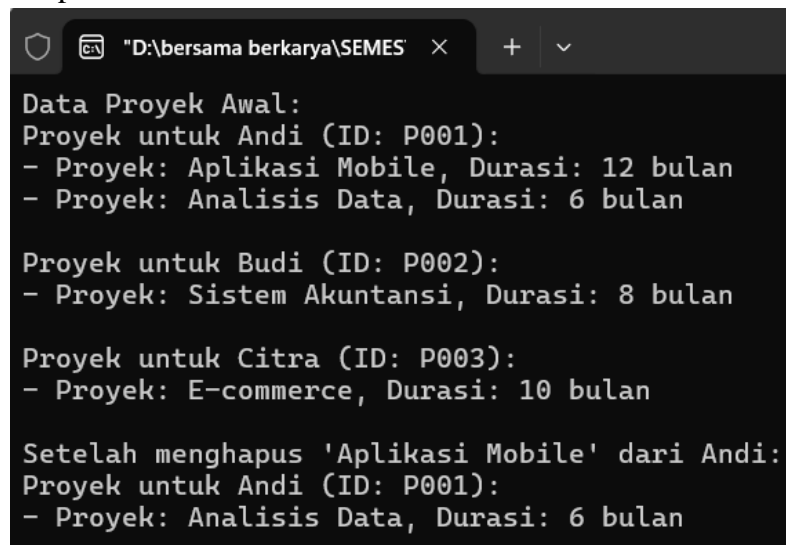
```

32     Proyek* proyekBaru = new Proyek;
33     proyekBaru->nama = nama;
34     proyekBaru->durasi = durasi;
35     proyekBaru->next = NULL;
36     return proyekBaru;
37 }
38
39 // Fungsi untuk menambahkan proyek ke pegawai
40 void tambahProyek(Pegawai* peg, string namaProyek, int durasi) {
41     Proyek* proyekBaru = buatProyek(namaProyek, durasi);
42
43     if (peg->proyek == NULL) {
44         peg->proyek = proyekBaru;
45     } else {
46         Proyek* current = peg->proyek;
47         while (current->next != NULL) {
48             current = current->next;
49         }
50         current->next = proyekBaru;
51     }
52 }
53
54 // Fungsi untuk menghapus proyek dari pegawai
55 void hapusProyek(Pegawai* peg, string namaProyek) {
56     if (peg->proyek == NULL) return;
57
58     if (peg->proyek->nama == namaProyek) {
59         Proyek* temp = peg->proyek;
60         peg->proyek = peg->proyek->next;
61         delete temp;
62         return;
63     }
64
65     Proyek* current = peg->proyek;
66     while (current->next != NULL && current->next->nama != namaProyek) {
67         current = current->next;
68     }
69
70     if (current->next != NULL) {
71         Proyek* temp = current->next;
72         current->next = current->next->next;
73         delete temp;
74     }
75 }
76
77 // Fungsi untuk menampilkan proyek-proyek seorang pegawai
78 void tampilkanProyek(Pegawai* peg) {
79     cout << "Proyek untuk " << peg->nama << " (ID: " << peg->id << "):" << endl;
80     Proyek* current = peg->proyek;
81     while (current != NULL) {
82         cout << "- Proyek: " << current->nama << ", Durasi: " << current->durasi << " bulan" << endl;
83         current = current->next;
84     }
85     cout << endl;
86 }
87
88 int main() {
89     // Membuat daftar pegawai
90     Pegawai* kepala = buatPegawai("Andi", "P001");
91     kepala->next = buatPegawai("Budi", "P002");
92     kepala->next->next = buatPegawai("Citra", "P003");

```

```
93
94 // Menambahkan proyek ke Andi
95 tambahProyek(kepala, "Aplikasi Mobile", 12);
96 tambahProyek(kepala, "Analisis Data", 6);
97
98 // Menambahkan proyek ke Budi
99 tambahProyek(kepala->next, "Sistem Akuntansi", 8);
100
101 // Menambahkan proyek ke Citra
102 tambahProyek(kepala->next->next, "E-commerce", 10);
103
104 // Menampilkan data proyek awal
105 cout << "Data Proyek Awal:" << endl;
106 tampilkanProyek(kepala);
107 tampilkanProyek(kepala->next);
108 tampilkanProyek(kepala->next->next);
109
110 // Menghapus proyek "Aplikasi Mobile" dari Andi
111 cout << "Setelah menghapus 'Aplikasi Mobile' dari Andi:" << endl;
112 hapusProyek(kepala, "Aplikasi Mobile");
113 tampilkanProyek(kepala);
114
115 return 0;
116 }
```

Output :



```
"D:\bersama berkarya\SEMES" x + v
Data Proyek Awal:
Proyek untuk Andi (ID: P001):
- Proyek: Aplikasi Mobile, Durasi: 12 bulan
- Proyek: Analisis Data, Durasi: 6 bulan

Proyek untuk Budi (ID: P002):
- Proyek: Sistem Akuntansi, Durasi: 8 bulan

Proyek untuk Citra (ID: P003):
- Proyek: E-commerce, Durasi: 10 bulan

Setelah menghapus 'Aplikasi Mobile' dari Andi:
Proyek untuk Andi (ID: P001):
- Proyek: Analisis Data, Durasi: 6 bulan
```

Penjelasan Program

Program dimulai dengan mendefinisikan dua struktur utama: Proyek dan Pegawai. Struktur Proyek menyimpan informasi proyek termasuk nama proyek dan durasi, sementara struktur Pegawai berisi detail pegawai seperti nama dan ID. Kedua struktur menggunakan pointer untuk membuat linked list, memungkinkan pengelolaan dinamis dari beberapa proyek per pegawai dan beberapa pegawai dalam sistem. Ketika program mulai berjalan dalam fungsi main, pertama-tama program membuat tiga pegawai: Andi (ID: P001), Budi (ID: P002), dan Citra (ID: P003). Para pegawai ini terhubung bersama dalam satu linked list dengan Andi sebagai node kepala. Saat menampilkan data proyek awal, program mengeluarkan informasi setiap pegawai beserta proyek yang ditugaskan kepada mereka. Fungsi tampilkanProyek menelusuri

melalui daftar proyek setiap pegawai, menampilkan nama proyek dan durasi untuk setiap proyek yang ditugaskan kepada pegawai tersebut.

Setelah menunjukkan keadaan awal, program mendemonstrasikan fungsi penghapusan proyek dengan menghapus proyek "Aplikasi Mobile" dari daftar proyek Andi. Fungsi hapusProyek menangani ini dengan memperbarui koneksi linked list secara tepat dan membebaskan memori dari node proyek yang dihapus.

2. Sistem Manajemen Buku Perpustakaan

Gunakan Multi Linked List untuk menyimpan data anggota perpustakaan dan daftar buku

yang dipinjam.

- Setiap anggota memiliki data: Nama Anggota dan ID Anggota.
- Setiap buku memiliki data: Judul Buku dan Tanggal Pengembalian.

Instruksi:

- a. Masukkan data anggota berikut:
 - Anggota 1: Nama = "Rani", ID = "A001".
 - Anggota 2: Nama = "Dito", ID = "A002".
 - Anggota 3: Nama = "Vina", ID = "A003".
- b. Tambahkan buku yang dipinjam:
 - Buku 1: Judul = "Pemrograman C++", Pengembalian = "01/12/2024" (Untuk Rani).
 - Buku 2: Judul = "Algoritma Pemrograman", Pengembalian = "15/12/2024" (Untuk
- c. Dito).
- d. Tambahkan buku baru:
 - Buku 3: Judul = "Struktur Data", Pengembalian = "10/12/2024" (Untuk Rani).
- e. Hapus anggota Dito beserta buku yang dipinjam.
- f. Tampilkan seluruh data anggota dan buku yang dipinjam.

Jawaban :

Source code

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  // Struktur untuk node Buku
6  struct Buku {
7      string judul;
8      string tanggalPengembalian;
9      Buku* next;
10 };
11
12 // Struktur untuk node Anggota
13 struct Anggota {
14     string nama;
15     string id;
16     Buku* daftarBuku;
17     Anggota* next;
18 };
19
20 // Fungsi untuk membuat anggota baru
21 Anggota* buatAnggota(string nama, string id) {
22     Anggota* anggotaBaru = new Anggota;
23     anggotaBaru->nama = nama;
24     anggotaBaru->id = id;
25     anggotaBaru->daftarBuku = NULL;
26     anggotaBaru->next = NULL;
27     return anggotaBaru;
28 }
29
30 // Fungsi untuk membuat buku baru
31 Buku* buatBuku(string judul, string tanggalPengembalian) {
32     Buku* bukuBaru = new Buku;
33     bukuBaru->judul = judul;
34     bukuBaru->tanggalPengembalian = tanggalPengembalian;
35     bukuBaru->next = NULL;
36     return bukuBaru;
37 }
38
39 // Fungsi untuk menambahkan buku ke anggota
40 void tambahBuku(Anggota* anggota, string judul, string tanggalPengembalian) {
41     Buku* bukuBaru = buatBuku(judul, tanggalPengembalian);
42
43     if (anggota->daftarBuku == NULL) {
44         anggota->daftarBuku = bukuBaru;
45     } else {
46         Buku* current = anggota->daftarBuku;
47         while (current->next != NULL) {
48             current = current->next;
49         }
50         current->next = bukuBaru;
51     }
52 }
53
54 // Fungsi untuk menghapus anggota dan buku-bukunya
55 void hapusAnggota(Anggota*& kepala, string id) {
56     if (kepala == NULL) return;
57
58     if (kepala->id == id) {
59         Anggota* temp = kepala;
60         kepala = kepala->next;
61
62         // Hapus semua buku anggota
```

```

63     Buku* currentBuku = temp->daftarBuku;
64     while (currentBuku != NULL) {
65         Buku* tempBuku = currentBuku;
66         currentBuku = currentBuku->next;
67         delete tempBuku;
68     }
69
70     delete temp;
71     return;
72 }
73
74 Anggota* current = kepala;
75 while (current->next != NULL && current->next->id != id) {
76     current = current->next;
77 }
78
79 if (current->next != NULL) {
80     Anggota* temp = current->next;
81     current->next = current->next->next;
82
83     // Hapus semua buku anggota
84     Buku* currentBuku = temp->daftarBuku;
85     while (currentBuku != NULL) {
86         Buku* tempBuku = currentBuku;
87         currentBuku = currentBuku->next;
88         delete tempBuku;
89     }
90
91     delete temp;
92 }
93 }

```

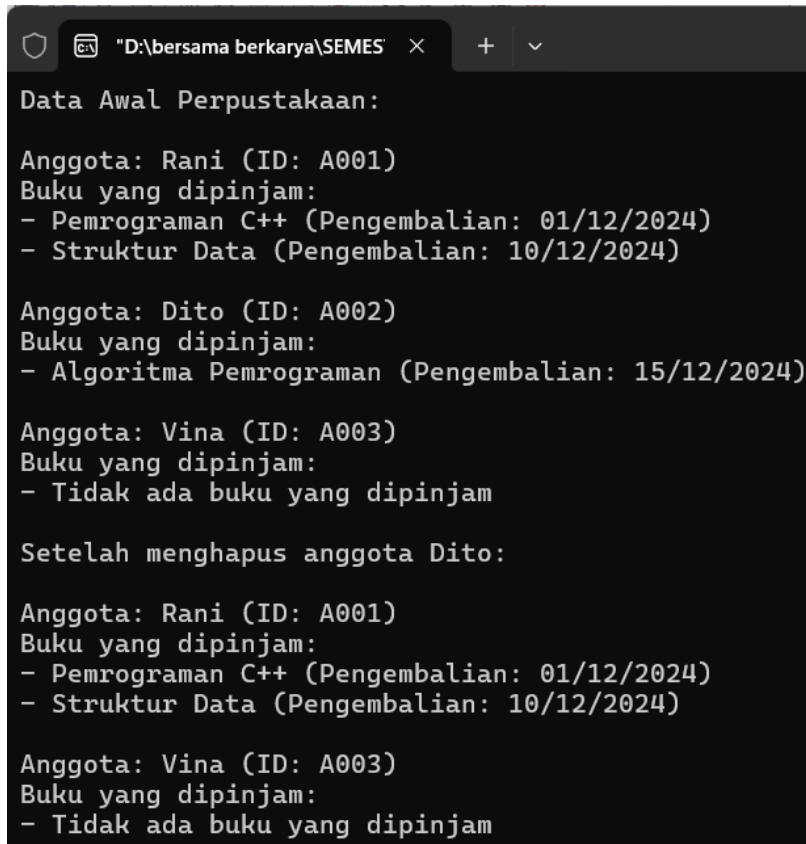
```

95 // Fungsi untuk menampilkan data anggota dan buku
96 void tampilkanData(Anggota* kepala) {
97     Anggota* currentAnggota = kepala;
98     while (currentAnggota != NULL) {
99         cout << "\nAnggota: " << currentAnggota->nama << " (ID: " << currentAnggota->id << ")" << endl;
100         cout << "Buku yang dipinjam: " << endl;
101
102         Buku* currentBuku = currentAnggota->daftarBuku;
103         if (currentBuku == NULL) {
104             cout << "- Tidak ada buku yang dipinjam" << endl;
105         }
106         while (currentBuku != NULL) {
107             cout << "- " << currentBuku->judul << " (Pengembalian: " << currentBuku->tanggalPengembalian << ")" << endl;
108             currentBuku = currentBuku->next;
109         }
110         currentAnggota = currentAnggota->next;
111     }
112 }
113
114 int main() {
115     // Membuat daftar anggota
116     Anggota* kepala = buatAnggota("Rani", "A001");
117     kepala->next = buatAnggota("Dito", "A002");
118     kepala->next->next = buatAnggota("Vina", "A003");
119
120     // Menambahkan buku untuk Rani
121     tambahBuku(kepala, "Pemrograman C++", "01/12/2024");
122     tambahBuku(kepala, "Struktur Data", "10/12/2024");
123
124     // Menambahkan buku untuk Dito
125     tambahBuku(kepala->next, "Algoritma Pemrograman", "15/12/2024");

```

```
126
127     cout << "Data Awal Perpustakaan:" << endl;
128     tampilkanData(kepala);
129
130     cout << "\nSetelah menghapus anggota Dito:" << endl;
131     hapusAnggota(kepala, "A002");
132     tampilkanData(kepala);
133
134     return 0;
135 }
```

Output :



```
Data Awal Perpustakaan:

Anggota: Rani (ID: A001)
Buku yang dipinjam:
- Pemrograman C++ (Pengembalian: 01/12/2024)
- Struktur Data (Pengembalian: 10/12/2024)

Anggota: Dito (ID: A002)
Buku yang dipinjam:
- Algoritma Pemrograman (Pengembalian: 15/12/2024)

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

Setelah menghapus anggota Dito:

Anggota: Rani (ID: A001)
Buku yang dipinjam:
- Pemrograman C++ (Pengembalian: 01/12/2024)
- Struktur Data (Pengembalian: 10/12/2024)

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

Penjelasan Program

Program ini menggunakan manajemen memori yang baik dengan mengalokasikan memori secara dinamis menggunakan operator 'new' saat membuat node baru dan menghapusnya dengan benar menggunakan 'delete' saat menghapus data. Struktur linked list yang digunakan memungkinkan penambahan dan penghapusan data secara fleksibel, cocok untuk sistem perpustakaan yang dinamis.

Kesimpulan

Kesimpulan yang saya dapatkan dari mengerjakan Guided serta Unguided yang diberikan, Multi Linked List merupakan struktur data yang kompleks namun powerful untuk merepresentasikan data yang memiliki hierarki atau hubungan induk-anak.