

# **LAPORAN PRAKTIKUM**

## **Modul 13**

### **“MULTI LINKED LIST”**



**Disusun Oleh:**

**Rifqi Mohamad Ramdani -2311104044**

**SE-07-02**

**Dosen :**

**Wahyu Andi Saputra, S.pd,M.Eng,**

**PROGRAM STUDI S1 SOFTWARE ENGINEERING**

**FAKULTAS INFORMATIKA**

**TELKOM UNIVERSITY**

**PURWOKERTO 2024**

## 1. Tujuan

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

## 2. Landasan Teori

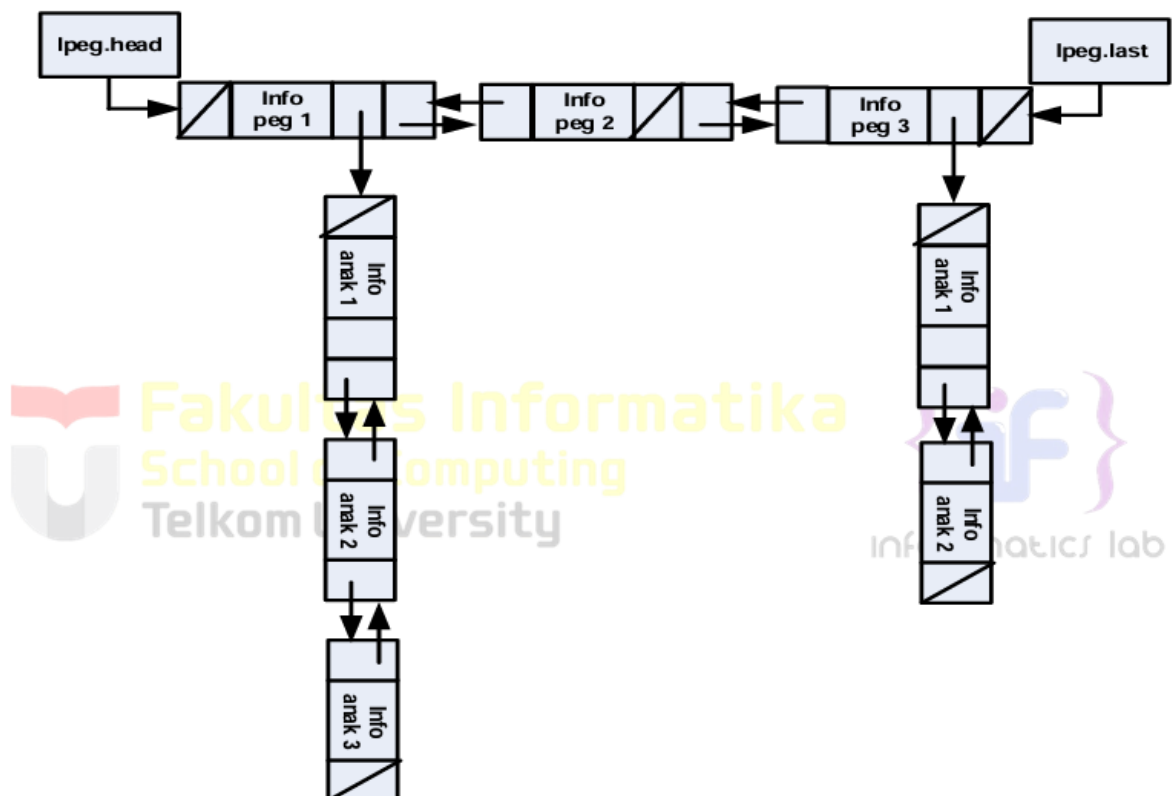
Multi Linked List: Merupakan pengembangan dari Linked List yang memungkinkan setiap node memiliki lebih dari satu pointer, masing-masing menunjuk ke struktur data lainnya. Biasanya, Multi Linked List digunakan untuk menyimpan data yang memiliki hubungan lebih kompleks, seperti data hierarkis.

## 3. Guided

Multi Linked List

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.

Contoh Multi Linked list dapat dilihat pada gambar berikut.

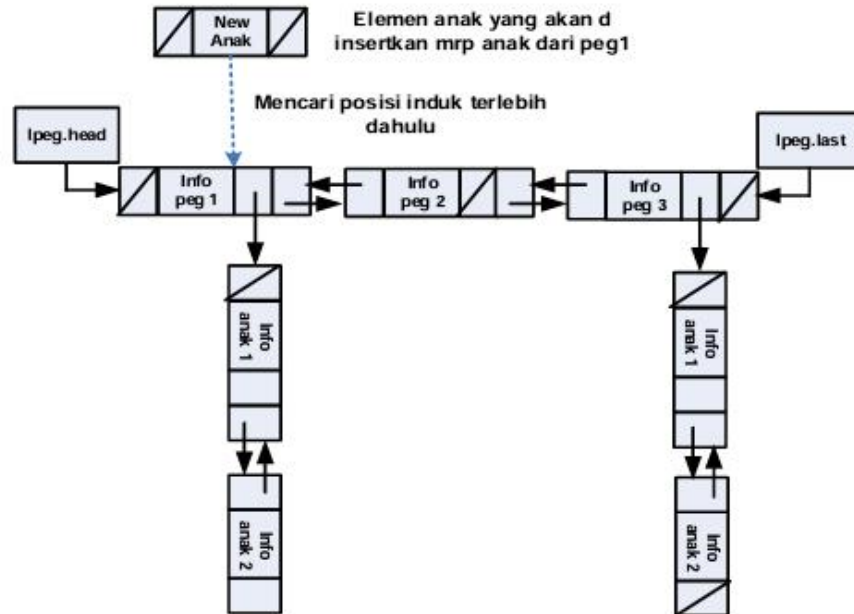


Gambar 13-1 Multi Linked list

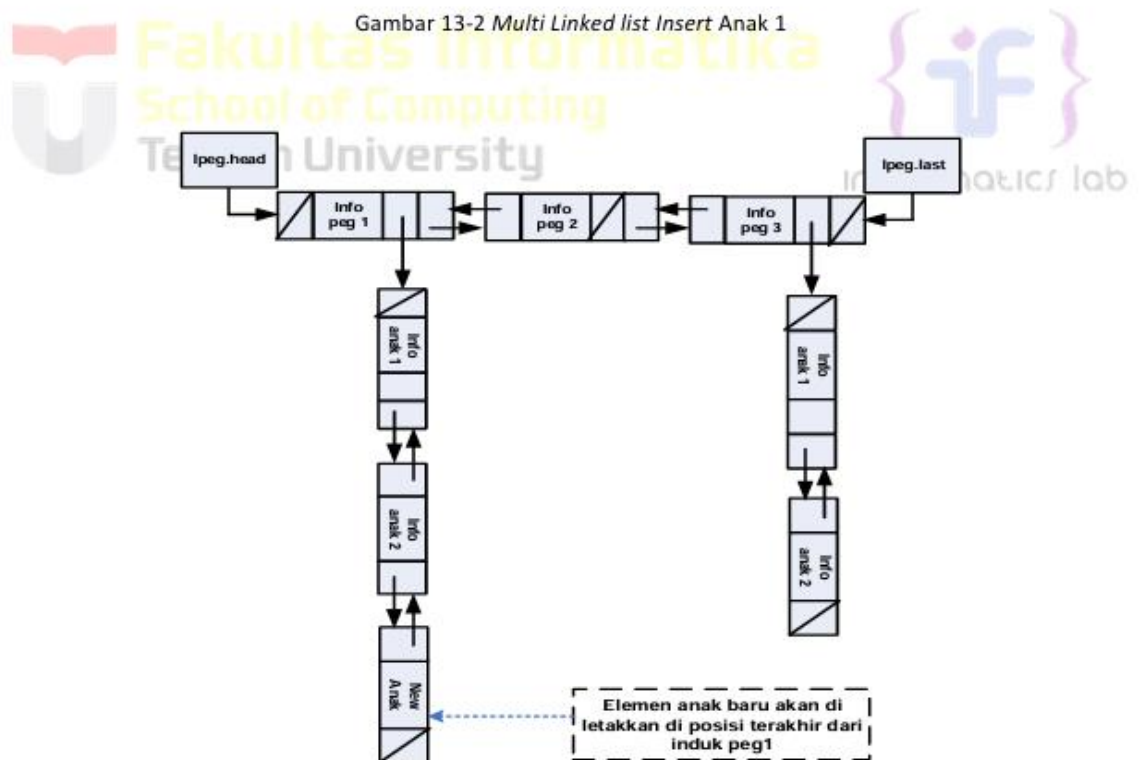
Jadi, dari implementasi di atas akan terdapat dua buah list, list pegawai dan list anak. Dimana untuk list pegawai menunjuk satu buah list anak. Disini list induknya adalah list pegawai dan list anaknya adalah list anak.

Insert A. Insert Anak Dalam penambahan elemen anak harus diketahui dulu elemen induknya.

Berikut ini ilustrasi *insert* anak dengan konsep *insert last*:



Gambar 13-2 Multi Linked list Insert Anak 1



```
/* buat dahulu elemen yang akan disisipkan */
address_anak alokasiAnak(inftypeanak X){
```

```

    address_anak p = alokasi(X);
    next(p) = null;
    prev(p) = null;
    return p;
}
/* mencari apakah ada elemen pegawai dengan info X */
address findElm(listinduk L, infotypeinduk X){
    address cariInduk = head(L);
    do{
        if(cariInduk.info == X){
            return cariInduk;
        }else{
            cariInduk = next(cariInduk);
        }
    }while(cariInduk.info!=X || cariInduk!=last(L))
}
/* menyisipkan anak pada akhir list anak */
void insertLastAnak(listanak &Lanak, address_anak P){
    address_anak ! = head(&Lanak);
    do{
        Q = next(Q);
    }while(next(&Lanak)!=NULL)
    next(Q) = P;
    prev(P) = Q;
    next(P) = NULL;
}

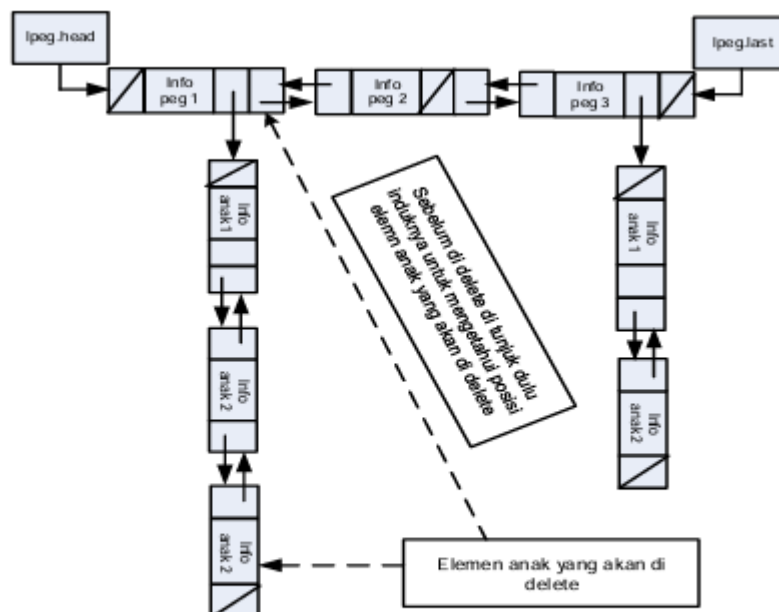
```

## Insert Induk

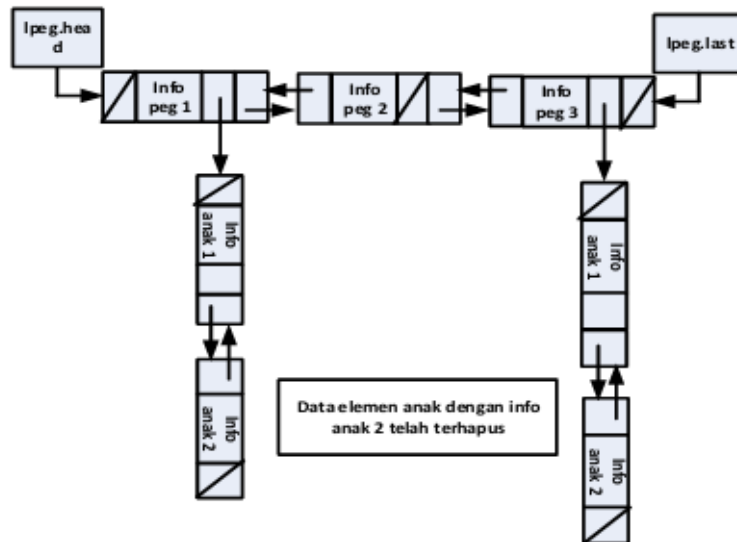
Untuk insert elemen induk sama dengan konsep insert pada single, double dan circular linked list

## Delete

- A. Delete Anak Sama dengan insert anak untuk delete anak maka harus diketahui dulu induknya. Berikut ini Gambar ilustrasinya untuk delete last pada induk peg 1:



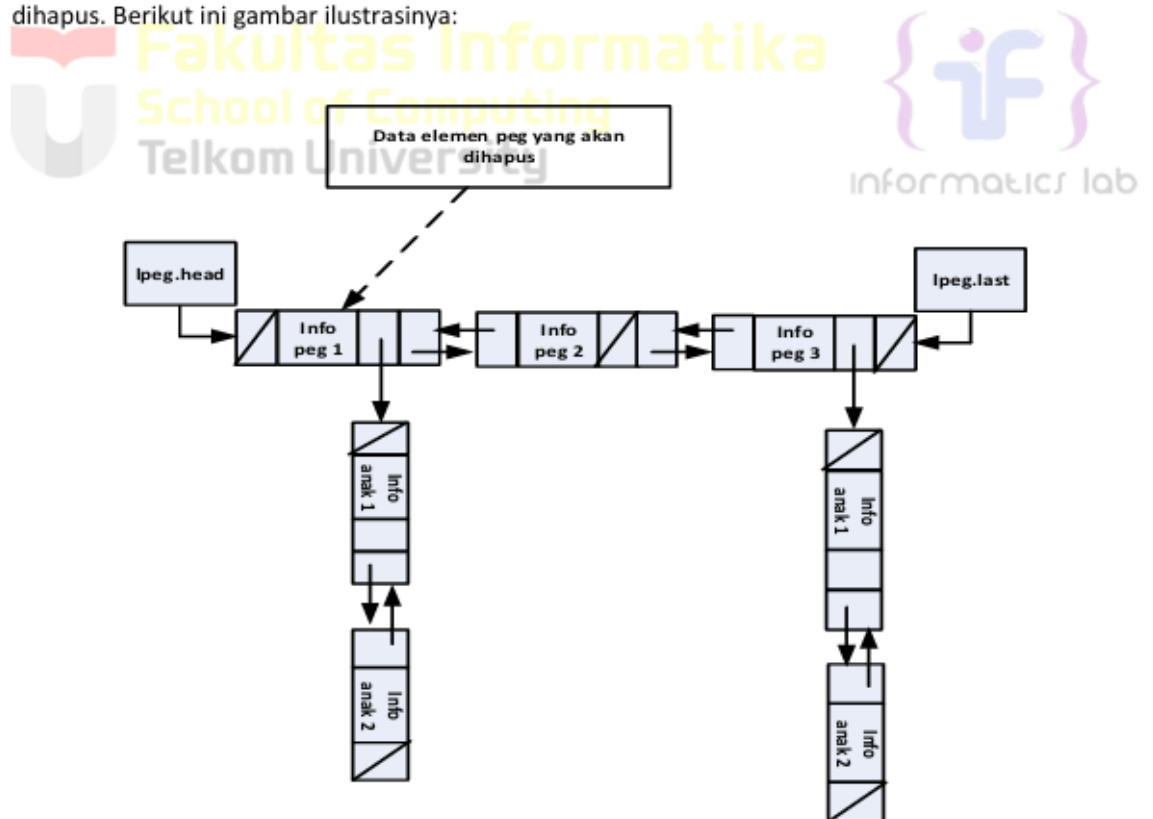
Gambar 13-4 Multi Linked list Delete Anak 1



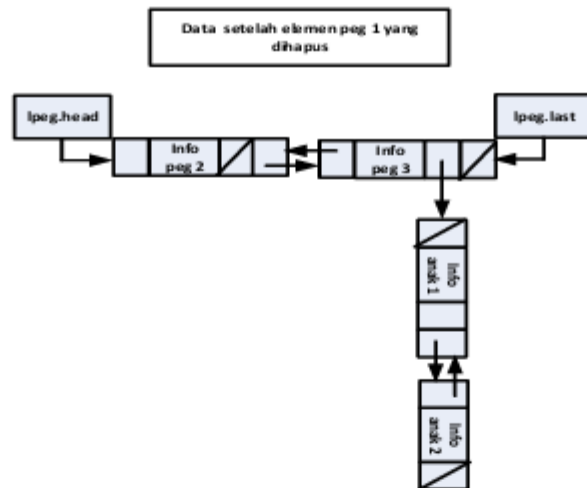
Gambar 13-5 Multi Linked list Delete Anak 2

### B. Delete Induk

Untuk *delete* elemen induk maka saat di hapus maka seluruh anak dengan induk tersebut juga harus dihapus. Berikut ini gambar ilustrasinya:



Gambar 13-6 Multi Linked list Delete Induk 1



Gambar 13-7 Multi Linked list Delete Induk 2

```

1  /*file : multilist .h*/
2  /* contoh ADT list berkait dengan representasi fisik pointer*/
3  /* representasi address dengan pointer*/
4
5  /* info tipe adalah integer */
6  #ifndef MULTILIST_H_INCLUDED
7  #define MULTILIST_H_INCLUDED
8  #include <stdio.h>
9  #define Nil NULL
10 #define info(P) (P)->info
11 #define next(P) (P)->next
12 #define first(L) ((L).first)
13 #define last(L) ((L).last)
14
15 typedef int infotypeanak;
16 typedef int infotypeinduk;
17 typedef struct elemen_list_induk *address;
18 typedef struct elemen_list_anak *address_anak;
19 /* define list : */
20
21 /* list kosong jika first(L)=Nil
22 setiap elemen address P dapat diacu info(P) atau next(P)
23 elemen terakhir list jika addressnya last, maka next(last) = Nil */
24 struct elemen_list_anak{
25     /* struct ini untuk menyimpan elemen anak dan pointer penunjuk
26     elemen tetangganya */
27     infotypeanak info;
28     address_anak next;
29     address_anak prev;
30 };
31
32 struct listanak {
33     /* struct ini digunakan untuk menyimpan list anak itu sendiri */
34     address_anak first;
35     address_anak last;
36 };
37
38 struct elemen_list_induk{
39     /* struct ini untuk menyimpan elemen induk dan pointer penunjuk
40     elemen tetangganya */
41     infotypeanak info;
42     struct listanak listanak;
43     address next;
44     address prev;

```

```

44     };
45     struct listinduk {
46         /* struct ini digunakan untuk menyimpan list induk itu sendiri */
47         address first;
48         address last;
49     };
50
51     /***** pengecekan apakah list kosong *****/
52     boolean ListEmpty(listinduk L);
53     /*mengembalikan nilai true jika list induk kosong*/
54     boolean ListEmptyAnak(listanak L);
55     /*mengembalikan nilai true jika list anak kosong*/
56
57     /***** pembuatan list kosong *****/
58     void CreateList(listinduk &L);
59     /* I.S. sembarang
60        F.S. terbentuk list induk kosong*/
61     void CreateListAnak(listanak &L);
62     /* I.S. sembarang
63        F.S. terbentuk list anak kosong*/
64
65     /***** manajemen memori *****/
66     address alokasi(infotypeinduk P);
67     /* mengirimkan address dari alokasi sebuah elemen induk
68        jika alokasi berhasil, maka nilai address tidak Nil dan jika gagal
69        nilai address Nil */
70
71     address_anak alokasiAnak(infotypeanak P);
72     /* mengirimkan address dari alokasi sebuah elemen anak
73        jika alokasi berhasil, maka nilai address tidak Nil dan jika gagal
74        nilai address_anak Nil */
75
76     void dealokasi(address P);
77     /* I.S. P terdefinisi
78        F.S. memori yang digunakan P dikembalikan ke sistem */
79
80     void dealokasiAnak(address_anak P);
81     /* I.S. P terdefinisi
82        F.S. memori yang digunakan P dikembalikan ke sistem */
83     /***** pencarian sebuah elemen list *****/
84     address findElm(listinduk L, infotypeinduk X);
85     /* mencari apakah ada elemen list dengan info(P) = X
86        jika ada, mengembalikan address elemen tab tsb, dan Nil jika sebaliknya
87     */
88     address_anak findElm(listanak Lanak, infotypeanak X);
89     /* mencari apakah ada elemen list dengan info(P) = X
90        jika ada, mengembalikan address elemen tab tsb, dan Nil jika sebaliknya
91     */
92     boolean fFindElm(listinduk L, address P);
93     /* mencari apakah ada elemen list dengan alamat P
94        mengembalikan true jika ada dan false jika tidak ada */
95     boolean fFindElmanak(listanak Lanak, address_anak P);
96     /* mencari apakah ada elemen list dengan alamat P
97        mengembalikan true jika ada dan false jika tidak ada */
98
99     address findBefore(listinduk L, address P);
100    /* mengembalikan address elemen sebelum P
101       jika P berada pada awal list, maka mengembalikan nilai Nil */
102    address_anak findBeforeAnak(listanak Lanak, infotypeinduk X, address_anak
103    P);
104    /* mengembalikan address elemen sebelum P dimana info(P) = X
105       jika P berada pada awal list, maka mengembalikan nilai Nil */
106
107    /***** penambahan elemen *****/
108    void insertFirst(listinduk &L, address P);
109    /* I.S. sembarang, P sudah dialokasikan
110       F.S. menempatkan elemen beralamat P pada awal list */

```



```

111
112 void insertAfter(listinduk &L, address P, address Prec);
113 /* I.S. sembarang, P dan Prec alamat salah satu elemen list
114    F.S. menempatkan elemen beralamat P sesudah elemen beralamat Prec */
115
116 void insertLast(listinduk &L, address P);
117 /* I.S. sembarang, P sudah dialokasikan
118    F.S. menempatkan elemen beralamat P pada akhir list */
119
120 void insertFirstAnak(listanak &L, address_anak P);
121 /* I.S. sembarang, P sudah dialokasikan
122    F.S. menempatkan elemen beralamat P pada awal list */
123
124 void insertAfterAnak(listanak &L, address_anak P, address_anak Prec);
125 /* I.S. sembarang, P dan Prec alamat salah satu elemen list
126    F.S. menempatkan elemen beralamat P sesudah elemen beralamat Prec */
127
128 void insertLastAnak(listanak &L, address_anak P);
129 /* I.S. sembarang, P sudah dialokasikan
130    F.S. menempatkan elemen beralamat P pada akhir list */
131
132 /***** penghapusan sebuah elemen *****/
133 void delFirst(listinduk &L, address &P);
134 /* I.S. list tidak kosong
135    F.S. adalah alamat dari alamat elemen pertama list
136    sebelum elemen pertama list dihapus
137    elemen pertama list hilang dan list mungkin menjadi kosong
138    first elemen yang baru adalah successor first elemen yang lama */
139 void delLast(listinduk &L, address &P);
140 /* I.S. list tidak kosong
141    F.S. adalah alamat dari alamat elemen terakhir list
142    sebelum elemen terakhir list dihapus
143    elemen terakhir list hilang dan list mungkin menjadi kosong
144    last elemen yang baru adalah successor last elemen yang lama */
145
146 void delAfter(listinduk &L, address &P, address Prec);
147 /* I.S. list tidak kosong, Prec alamat salah satu elemen list
148    F.S. P adalah alamat dari next(Prec), menghapus next(Prec) dari list */
149 void delP(listinduk &L, infotypeinduk X);
150 /* I.S. sembarang
151    F.S. jika ada elemen list dengan alamat P, dimana info(P)=X, maka P
152    dihapus
153    dan P di-dealokasi, jika tidak ada maka list tetap
154    list mungkin akan menjadi kosong karena penghapusan */
155
156 void delFirstAnak(listanak &L, address_anak &P);
157 /* I.S. list tidak kosong
158    F.S. adalah alamat dari alamat elemen pertama list
159    sebelum elemen pertama list dihapus
160    elemen pertama list hilang dan list mungkin menjadi kosong
161    first elemen yang baru adalah successor first elemen yang lama */
162 void delLastAnak(listanak &L, address_anak &P);
163 /* I.S. list tidak kosong
164    F.S. adalah alamat dari alamat elemen terakhir list
165    sebelum elemen terakhir list dihapus
166    elemen terakhir list hilang dan list mungkin menjadi kosong
167    last elemen yang baru adalah successor last elemen yang lama */
168
169 void delAfterAnak(listanak &L, address_anak &P, address_anak Prec);
170 /* I.S. list tidak kosong, Prec alamat salah satu elemen list
171    F.S. P adalah alamat dari next(Prec), menghapus next(Prec) dari list */
172 void delPAnak(listanak &L, infotypeanak X);
173 /* I.S. sembarang
174    F.S. jika ada elemen list dengan alamat P, dimana info(P)=X, maka P
175    dihapus
176    dan P di-dealokasi, jika tidak ada maka list tetap
177    list mungkin akan menjadi kosong karena penghapusan */

```



```

178 /***** proses semau elemen list *****/
179 void printInfo(list L);
180 /* I.S. list mungkin kosong
181    F.S. jika list tidak kosong menampilkan semua info yang ada pada list
182 */
183
184 int nbList(list L);
185 /* mengembalikan jumlah elemen pada list */
186
187 void printInfoAnak(listanak Lanak);
188 /* I.S. list mungkin kosong
189    F.S. jika list tidak kosong menampilkan semua info yang ada pada list
190 */
191
192 int nbListAnak(listanak Lanak);
193 /* mengembalikan jumlah elemen pada list anak */
194
195 /***** proses terhadap list *****/
196 void delAll(listinduk &L);
197 /* menghapus semua elemen list dan semua elemen di-dealokasi */
198
199 #endif

```

#### GUIDED

```

#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;
}

```

Maka Outputnya

```

D:\TUGAS SEMESTER 3\SAM
Parent: 3 -> 30 30
Parent: 2 -> 20 20
Parent: 1 -> 11 10

Process returned 0 (0x0)   execution time : 0.282 s
Press any key to continue.

```

2.

```

#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");

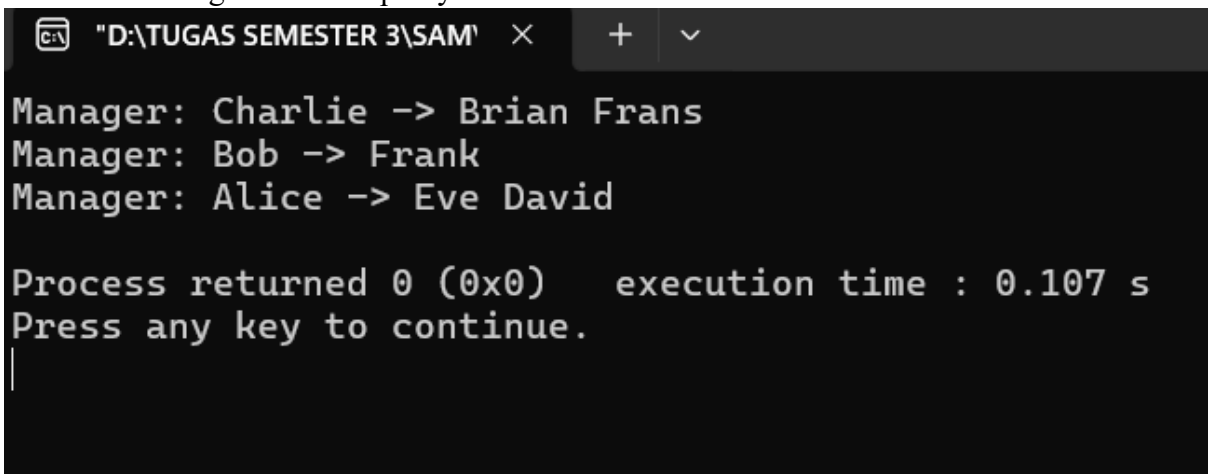
    empList.addSubordinate("Charlie", "Frans");
    empList.addSubordinate("Charlie", "Brian");

    empList.display();

    return 0;
}

```

Maka akan menghasilkan outputnya



```

D:\TUGAS SEMESTER 3\SAM >
Manager: Charlie -> Brian Frans
Manager: Bob -> Frank
Manager: Alice -> Eve David

Process returned 0 (0x0)   execution time : 0.107 s
Press any key to continue.
|

```

```

#include <iostream>
#include <string>

using namespace std;

// Struktur untuk node karyawan
struct EmployeeNode {
    string name; // Nama karyawan
    EmployeeNode* next; // Pointer ke karyawan berikutnya
    EmployeeNode* subordinate; // Pointer ke subordinate pertama

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

// Kelas untuk Multi-Linked List Karyawan
class EmployeeList {
private:
    EmployeeNode* head; // Pointer ke kepala list

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

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

    // Menambahkan subordinate ke karyawan tertentu
    void addSubordinate(string managerName, string subordinateName)
    {
        EmployeeNode* manager = head;
        while (manager != nullptr && manager->name != managerName)
        {
            manager = manager->next;
        }
        if (manager != nullptr) { // Jika manajer ditemukan
            EmployeeNode* newSubordinate = new
EmployeeNode(subordinateName);
            newSubordinate->next = manager->subordinate; //
Menyambungkan ke subordinate sebelumnya
            manager->subordinate = newSubordinate; // Memperbarui
subordinate
        } else {
            cout << "Manager not found!" << endl;
        }
    }
}

```

```

// Menghapus karyawan (induk)
void deleteEmployee(string name) {
    EmployeeNode** current = &head;
    while (*current != nullptr && (*current)->name != name) {
        current = &((*current)->next);
    }

    if (*current != nullptr) { // Jika karyawan ditemukan
        EmployeeNode* toDelete = *current;
        *current = (*current)->next;

        // Hapus semua subordinate dari node ini
        while (toDelete->subordinate != nullptr) {
            EmployeeNode* subTemp = toDelete->subordinate;
            toDelete->subordinate = toDelete->subordinate->next;
            delete subTemp;
        }
        delete toDelete;
        cout << "Employee " << name << " deleted." << endl;
    } else {
        cout << "Employee not found!" << endl;
    }
}

// Menghapus subordinate dari karyawan tertentu
void deleteSubordinate(string managerName, string
subordinateName) {
    EmployeeNode* manager = head;
    while (manager != nullptr && manager->name != managerName)
    {
        manager = manager->next;
    }

    if (manager != nullptr) { // Jika manajer ditemukan
        EmployeeNode** currentSub = &(manager->subordinate);
        while (*currentSub != nullptr && (*currentSub)->name !=
subordinateName) {
            currentSub = &((*currentSub)->next);
        }

        if (*currentSub != nullptr) { // Jika subordinate ditemukan
            EmployeeNode* toDelete = *currentSub;
            *currentSub = (*currentSub)->next; // Menghapus dari list

            delete toDelete; // Menghapus node subordinate
            cout << "Subordinate " << subordinateName << " deleted
from " << managerName << "." << endl;
        } else {
            cout << "Subordinate not found!" << endl;
        }
    }
}

```



```

    }
    } else {
        cout << "Manager not found!" << endl;
    }
}

// Menampilkan daftar karyawan dan subordinate mereka
void display() {
    EmployeeNode* current = head;
    while (current != nullptr) {
        cout << "Manager: " << current->name << " -> ";
        EmployeeNode* sub = current->subordinate;
        while (sub != nullptr) {
            cout << sub->name << " ";
            sub = sub->next;
        }
        cout << endl;
        current = current->next;
    }
}

~EmployeeList() {
    // Destructor untuk membersihkan memori
    while (head != nullptr) {
        EmployeeNode* temp = head;
        head = head->next;

        // Hapus semua subordinate dari node ini
        while (temp->subordinate != nullptr) {
            EmployeeNode* subTemp = temp->subordinate;
            temp->subordinate = temp->subordinate->next;
            delete subTemp;
        }
        delete temp;
    }
}

};

int main() {
    EmployeeList empList;

    empList.addEmployee("Alice");
    empList.addEmployee("Bob");
    empList.addEmployee("Charlie");

    empList.addSubordinate("Alice", "David");
    empList.addSubordinate("Alice", "Eve");
    empList.addSubordinate("Bob", "Frank");

    cout << "Initial employee list." << endl;
}

```

```

empList.display(); // Menampilkan isi daftar karyawan

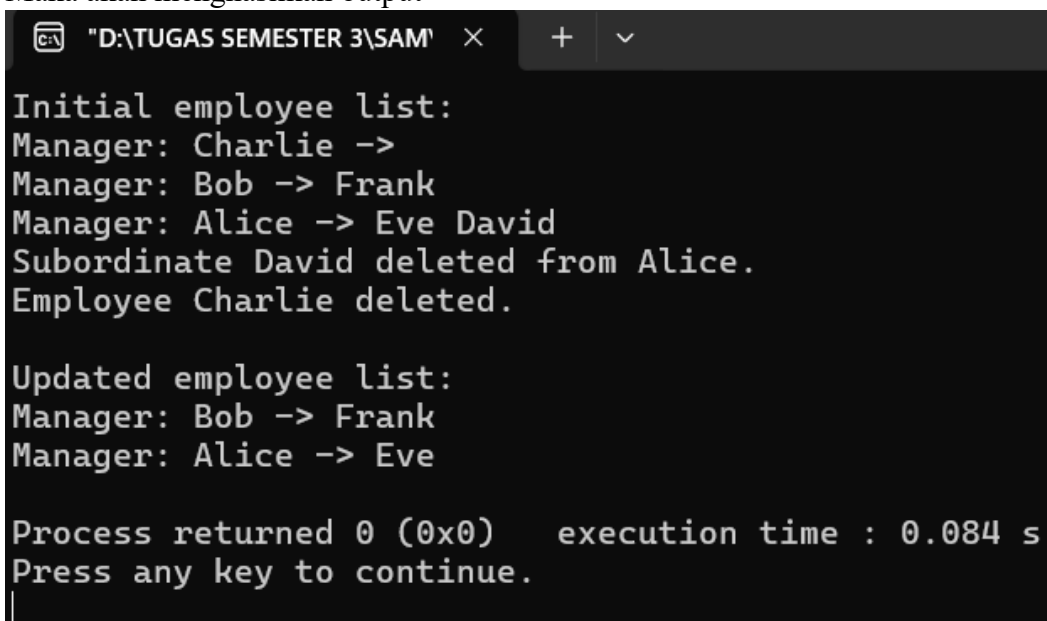
empList.deleteSubordinate("Alice", "David"); // Menghapus David
dari Alice
empList.deleteEmployee("Charlie"); // Menghapus Charlie

cout << "\nUpdated employee list:" << endl;
empList.display(); // Menampilkan isi daftar setelah penghapusan

return 0;
}

```

Maka akan menghasilkan 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

Process returned 0 (0x0)   execution time : 0.084 s
Press any key to continue.
|

```

#### 4. 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:

1. Masukkan data pegawai berikut: - Pegawai 1: Nama = "Andi", ID = "P001". - Pegawai 2: Nama = "Budi", ID = "P002". - Pegawai 3: Nama = "Citra", ID = "P003".

2. 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).
3. Tambahkan proyek baru: - Proyek 4: Nama = "Analisis Data", Durasi = 6 bulan (Untuk Andi).
4. Hapus proyek "Aplikasi Mobile" dari Andi.
5. Tampilkan data pegawai dan proyek mereka.

```
#include <iostream>
#include <string>

using namespace std;

struct PegawaiNode {
    string nama;
    string id;
    PegawaiNode* next;
    struct ProyekNode* proyekHead;
};

struct ProyekNode {
    string nama;
    int durasi;
    ProyekNode* next;
};

PegawaiNode* createPegawaiNode(string nama, string id) {
    PegawaiNode* newNode = new PegawaiNode;
    newNode->nama = nama;
    newNode->id = id;
    newNode->next = NULL;
    newNode->proyekHead = NULL;
    return newNode;
}

ProyekNode* createProyekNode(string nama, int durasi) {
    ProyekNode* newNode = new ProyekNode;
    newNode->nama = nama;
    newNode->durasi = durasi;
    newNode->next = NULL;
    return newNode;
}

PegawaiNode* addPegawai(PegawaiNode* head, string nama, string id) {
    PegawaiNode* newNode = createPegawaiNode(nama, id);
```

```

    if (head == NULL) {
        return newNode;
    }
    PegawaiNode* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
    return head;
}

```

```

void addProyek(PegawaiNode* head, string pegawaiID, string namaProyek, int
durasi) {
    PegawaiNode* temp = head;
    while (temp != NULL) {
        if (temp->id == pegawaiID) {
            ProyekNode* newProyek = createProyekNode(namaProyek, durasi);
            if (temp->proyekHead == NULL) {
                temp->proyekHead = newProyek;
            } else {
                ProyekNode* proyekTemp = temp->proyekHead;
                while (proyekTemp->next != NULL) {
                    proyekTemp = proyekTemp->next;
                }
                proyekTemp->next = newProyek;
            }
            return;
        }
        temp = temp->next;
    }
}

```

```

void removeProyek(PegawaiNode* head, string pegawaiID, string namaProyek) {
    PegawaiNode* temp = head;
    while (temp != NULL) {
        if (temp->id == pegawaiID) {
            ProyekNode* proyekTemp = temp->proyekHead;
            ProyekNode* prev = NULL;
            while (proyekTemp != NULL) {
                if (proyekTemp->nama == namaProyek) {
                    if (prev == NULL) {
                        temp->proyekHead = proyekTemp->next;
                    } else {
                        prev->next = proyekTemp->next;
                    }
                    delete proyekTemp;
                }
                prev = proyekTemp;
                proyekTemp = proyekTemp->next;
            }
            return;
        }
        temp = temp->next;
    }
}

```

```

    }
    prev = proyekTemp;
    proyekTemp = proyekTemp->next;
}
}
temp = temp->next;
}
}

void displayPegawai(PegawaiNode* head) {
    PegawaiNode* temp = head;
    cout << "\nData Pegawai dan Proyek:\n";
    while (temp != NULL) {
        cout << "Nama Pegawai: " << temp->nama << ", ID: " << temp->id << "\n";
        ProyekNode* proyekTemp = temp->proyekHead;
        if (proyekTemp == NULL) {
            cout << " - Tidak ada proyek.\n";
        } else {
            while (proyekTemp != NULL) {
                cout << " - Proyek: " << proyekTemp->nama << ", Durasi: " <<
proyekTemp->durasi << " bulan\n";
                proyekTemp = proyekTemp->next;
            }
        }
        temp = temp->next;
    }
}

int main() {
    PegawaiNode* headPegawai = NULL;

    // 1. Masukkan data pegawai
    headPegawai = addPegawai(headPegawai, "Andi", "P001");
    headPegawai = addPegawai(headPegawai, "Budi", "P002");
    headPegawai = addPegawai(headPegawai, "Citra", "P003");

    // 2. Tambahkan proyek ke pegawai
    addProyek(headPegawai, "P001", "Aplikasi Mobile", 12); // Untuk Andi
    addProyek(headPegawai, "P002", "Sistem Akuntansi", 8); // Untuk Budi
    addProyek(headPegawai, "P003", "E-commerce", 10); // Untuk Citra

    // 3. Tambahkan proyek baru
    addProyek(headPegawai, "P001", "Analisis Data", 6); // Untuk Andi

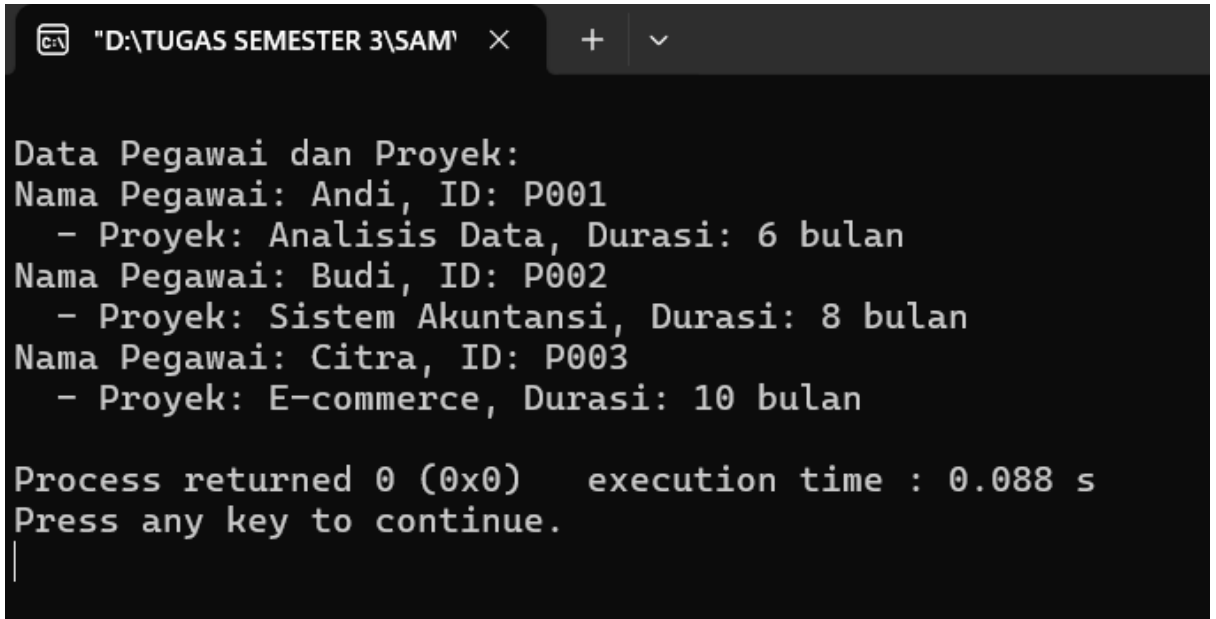
    // 4. Hapus proyek "Aplikasi Mobile" dari Andi
    removeProyek(headPegawai, "P001", "Aplikasi Mobile");
}

```

```
// 5. Tampilkan data pegawai dan proyek mereka
displayPegawai(headPegawai);

return 0;
}
```

Maka akan menghasilkan output



```
"D:\TUGAS SEMESTER 3\SAM" × + v

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

Process returned 0 (0x0)   execution time : 0.088 s
Press any key to continue.
|
```

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:

1. Masukkan data anggota berikut: - Anggota 1: Nama = "Rani", ID = "A001". - Anggota 2: Nama = "Dito", ID = "A002". - Anggota 3: Nama = "Vina", ID = "A003".

2. 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 Dito).

3. Tambahkan buku baru: - Buku 3: Judul = "Struktur Data", Pengembalian = "10/12/2024" (Untuk Rani).

4. Hapus anggota Dito beserta buku yang dipinjam.

5. Tampilkan seluruh data anggota dan buku yang dipinjam.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Struktur untuk data buku
typedef struct Buku {
    char judul[100];
    char pengembalian[15];
    struct Buku *next;
} Buku;

// Struktur untuk data anggota
typedef struct Anggota {
    char nama[50];
    char id[10];
    Buku *buku_head; // Pointer ke daftar buku yang dipinjam anggota
    struct Anggota *next;
} Anggota;

// Fungsi untuk menambahkan anggota baru
Anggota *tambahAnggota(Anggota *head, char nama[], char id[]) {
    Anggota *newAnggota = (Anggota *)malloc(sizeof(Anggota));
    strcpy(newAnggota->nama, nama);
    strcpy(newAnggota->id, id);
    newAnggota->buku_head = NULL; // Inisialisasi buku kosong
    newAnggota->next = head;
    return newAnggota;
}

// Fungsi untuk menambahkan buku ke anggota
void tambahBuku(Anggota *head, char id[], char judul[], char pengembalian[]) {
    Anggota *anggota = head;
    while (anggota != NULL) {
        if (strcmp(anggota->id, id) == 0) {
            Buku *newBuku = (Buku *)malloc(sizeof(Buku));
            strcpy(newBuku->judul, judul);
            strcpy(newBuku->pengembalian, pengembalian);
            newBuku->next = anggota->buku_head;
            anggota->buku_head = newBuku;
            return;
        }
        anggota = anggota->next;
    }
    printf("Anggota dengan ID %s tidak ditemukan.\n", id);
}

// Fungsi untuk menghapus anggota dan buku yang dipinjam

```



```

Anggota *hapusAnggota(Anggota *head, char id[]) {
    Anggota *current = head;
    Anggota *prev = NULL;

    while (current != NULL) {
        if (strcmp(current->id, id) == 0) {
            // Bebaskan semua buku yang dipinjam anggota
            Buku *buku = current->buku_head;
            while (buku != NULL) {
                Buku *temp = buku;
                buku = buku->next;
                free(temp);
            }

            // Hapus anggota
            if (prev == NULL) {
                head = current->next;
            } else {
                prev->next = current->next;
            }
            free(current);
            printf("Anggota dengan ID %s telah dihapus beserta buku yang dipinjam.\n", id);
            return head;
        }
        prev = current;
        current = current->next;
    }
    printf("Anggota dengan ID %s tidak ditemukan.\n", id);
    return head;
}

// Fungsi untuk menampilkan seluruh data anggota dan buku yang dipinjam
void tampilkanData(Anggota *head) {
    Anggota *current = head;
    while (current != NULL) {
        printf("Anggota: %s (ID: %s)\n", current->nama, current->id);
        Buku *buku = current->buku_head;
        while (buku != NULL) {
            printf("  - Buku: %s, Pengembalian: %s\n", buku->judul, buku->pengembalian);
            buku = buku->next;
        }
        current = current->next;
    }
}

```

```
// Main program
int main() {
    Anggota *head = NULL;

    // 1. Tambahkan data anggota
    head = tambahAnggota(head, "Rani", "A001");
    head = tambahAnggota(head, "Dito", "A002");
    head = tambahAnggota(head, "Vina", "A003");

    // 2. Tambahkan buku yang dipinjam
    tambahBuku(head, "A001", "Pemrograman C++", "01/12/2024");
    tambahBuku(head, "A002", "Algoritma Pemrograman", "15/12/2024");

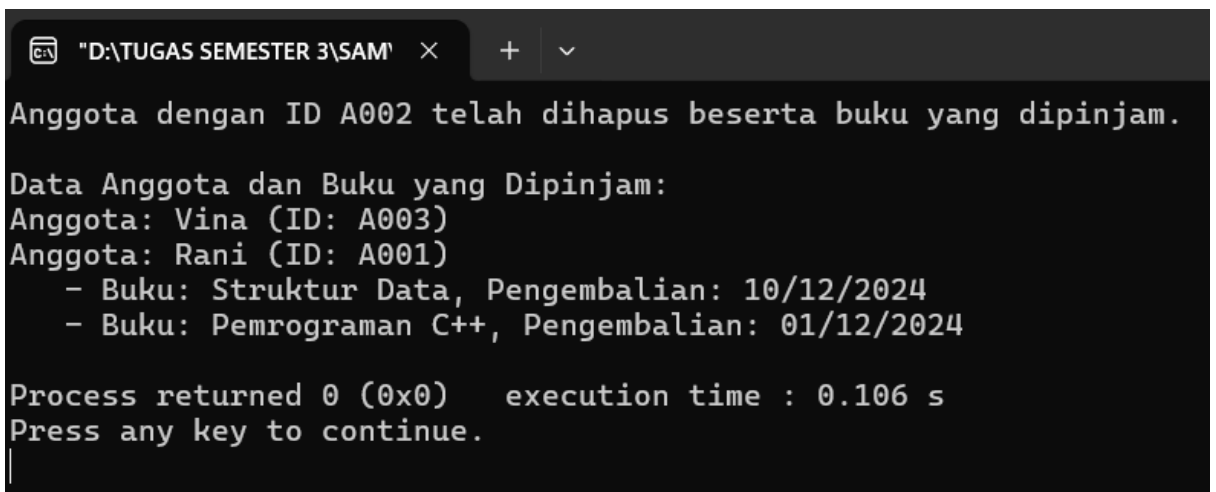
    // 3. Tambahkan buku baru
    tambahBuku(head, "A001", "Struktur Data", "10/12/2024");

    // 4. Hapus anggota Dito beserta buku yang dipinjam
    head = hapusAnggota(head, "A002");

    // 5. Tampilkan seluruh data anggota dan buku yang dipinjam
    printf("\nData Anggota dan Buku yang Dipinjam:\n");
    tampilkanData(head);

    return 0;
}
```

Maka akan menghasilkan output



```
"D:\TUGAS SEMESTER 3\SAM" x + v
Anggota dengan ID A002 telah dihapus beserta buku yang dipinjam.

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

Process returned 0 (0x0)   execution time : 0.106 s
Press any key to continue.
```

## 5. Kesimpulan

Multi Linked List adalah struktur data yang memungkinkan pengelolaan hubungan hierarkis antara elemen, seperti induk (parent) dan anak (child).

Struktur ini cocok untuk kasus data kompleks, seperti hubungan manajer-bawahan atau anggota perpustakaan-buku.

Operasi seperti penambahan, penghapusan, dan penelusuran dilakukan dengan navigasi antar node menggunakan pointer.

Keunggulan utamanya adalah fleksibilitas dalam pengorganisasian data, tetapi membutuhkan pemahaman manajemen memori yang baik.

Multi Linked List sering digunakan dalam sistem informasi manajemen, seperti pengelolaan karyawan atau proyek.