

LAPORAN PRAKTIKUM

Modul 13

Multi Linked List



Disusun Oleh:

Alvin Bagus Firmansyah - 2311104070

Kelas

SE-07-02

Dosen :

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

1. Tujuan

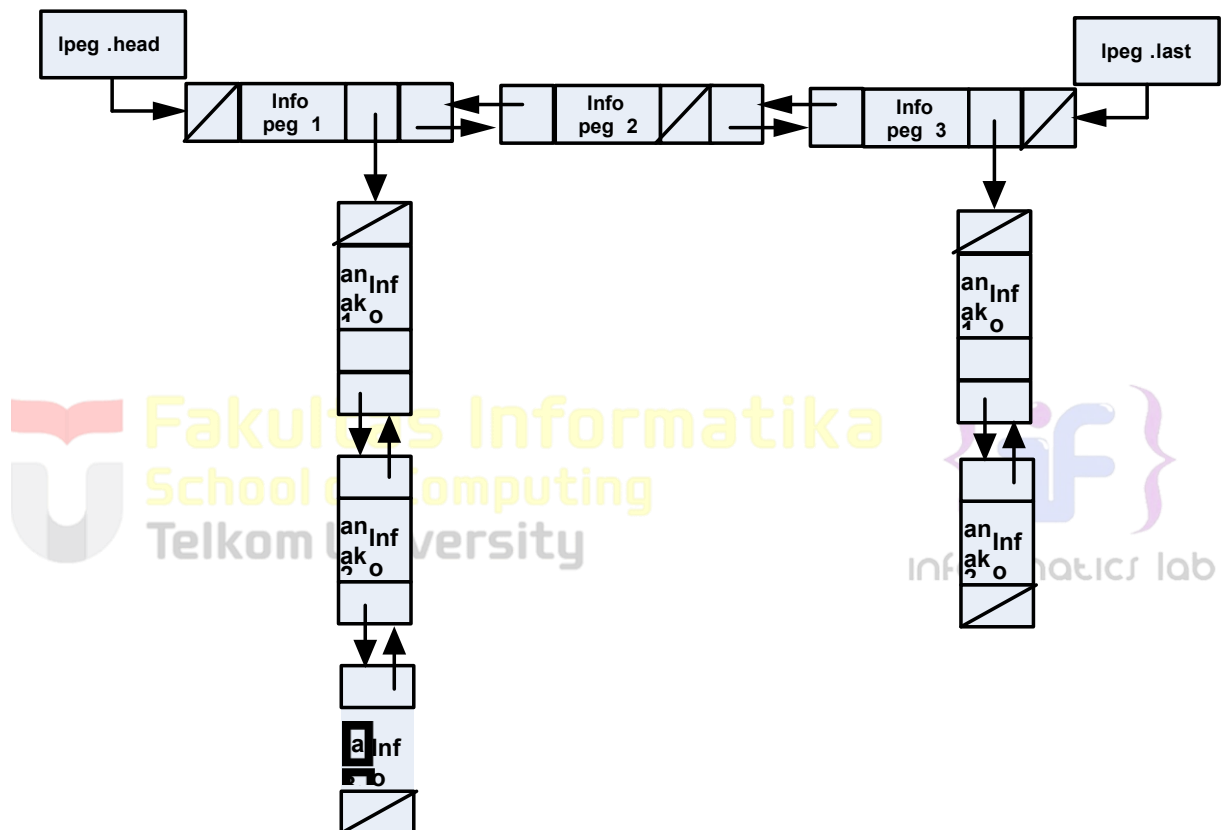
1. Memahami penggunaan *Multi Linked list*.
2. Mengimplementasikan *Multi Linked list* dalam beberapa studi kasus.

2. Landasan Teori

13.1 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.



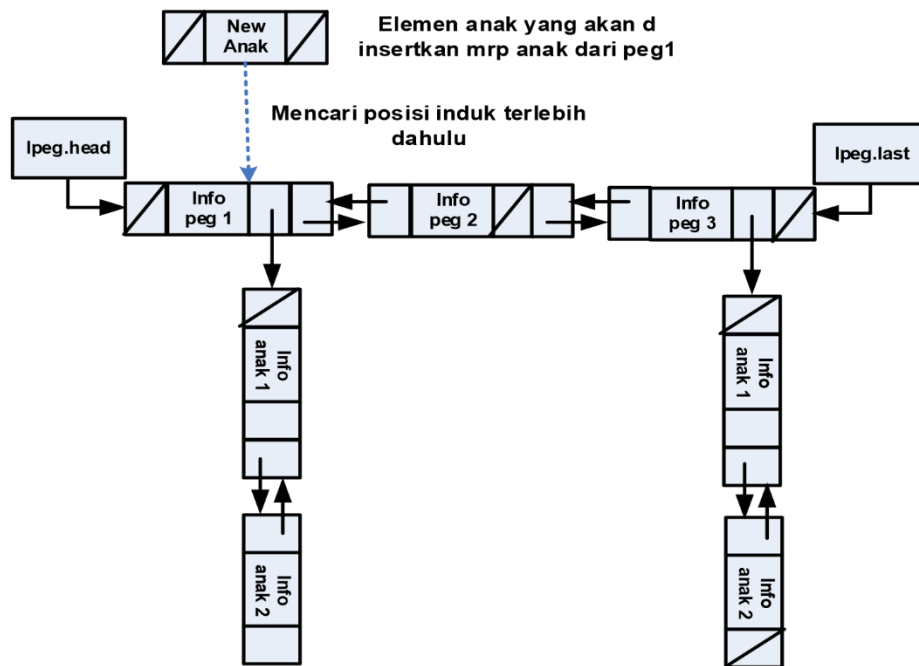
Jadi, dari implementasi di atas akan terdapat dua buah *list*, *list* pegawai dan *list* anak. Dimana untuk *list* pegawai menunjuk satu buah *list* anak. Disini *list* induknya adalah *list* pegawai dan *list* anaknya adalah *list* anak.

13.1.1 Insert

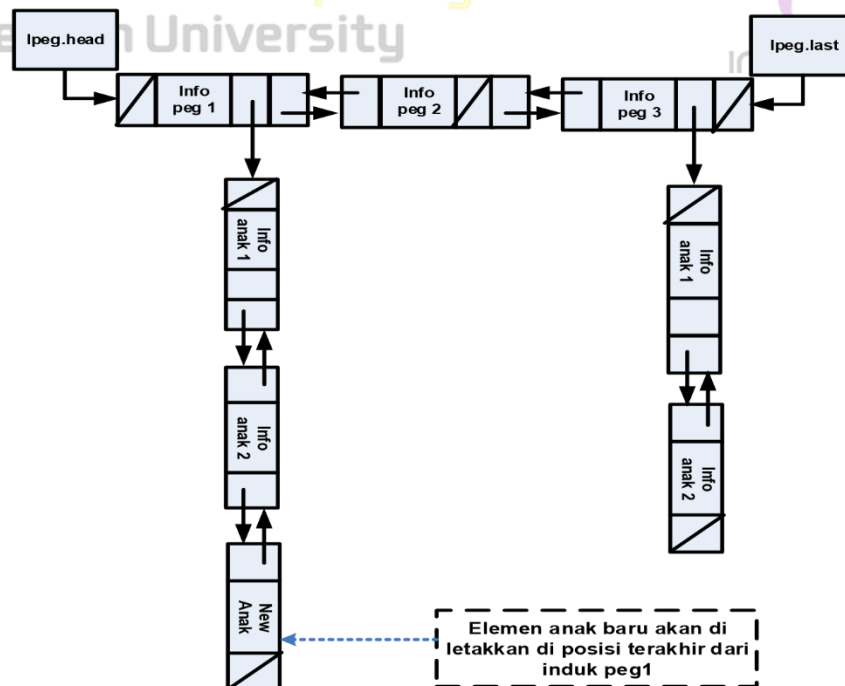
A. Insert Anak

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



Elemen anak baru akan di letakkan di posisi terakhir dari induk peg1

```
/* buat dahulu elemen yang akan disisipkan */
address_anak alokasiAnak(infotypeanak 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;
}

```

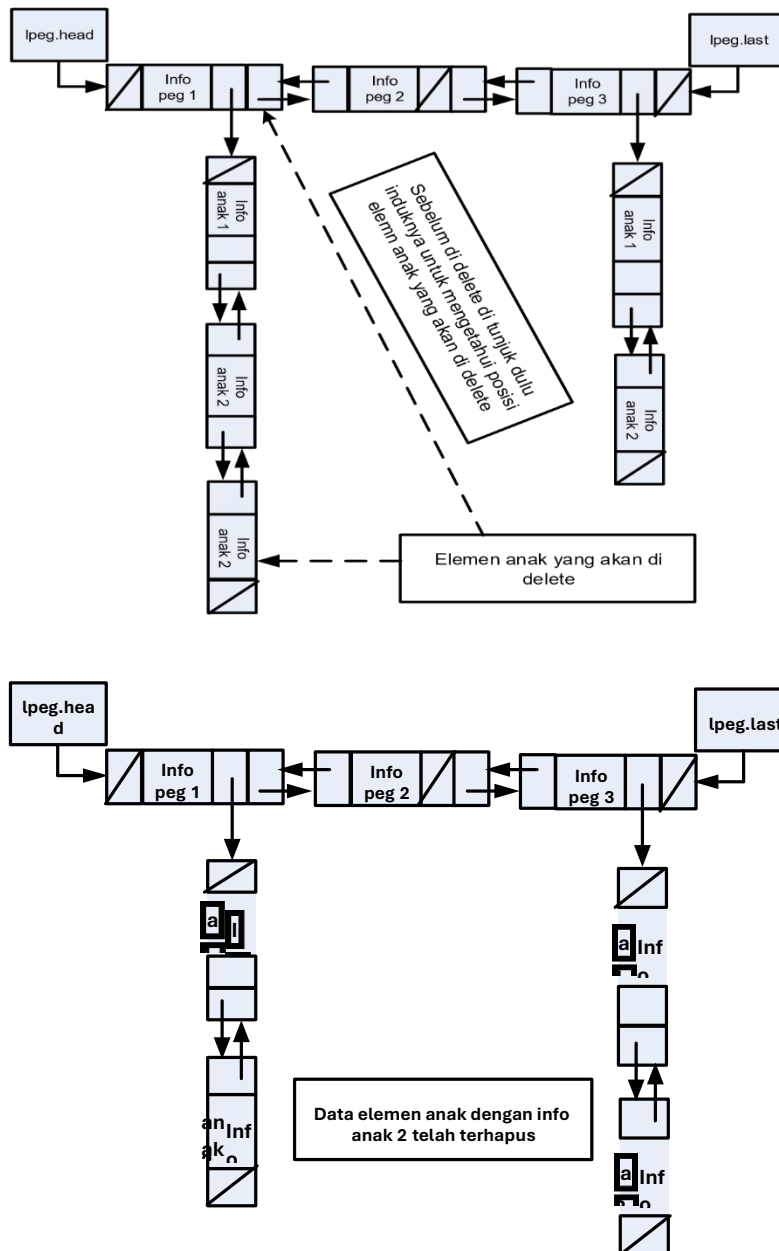
B. *Insert Induk*

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

13.1.2 Delete

A. Delete Anak

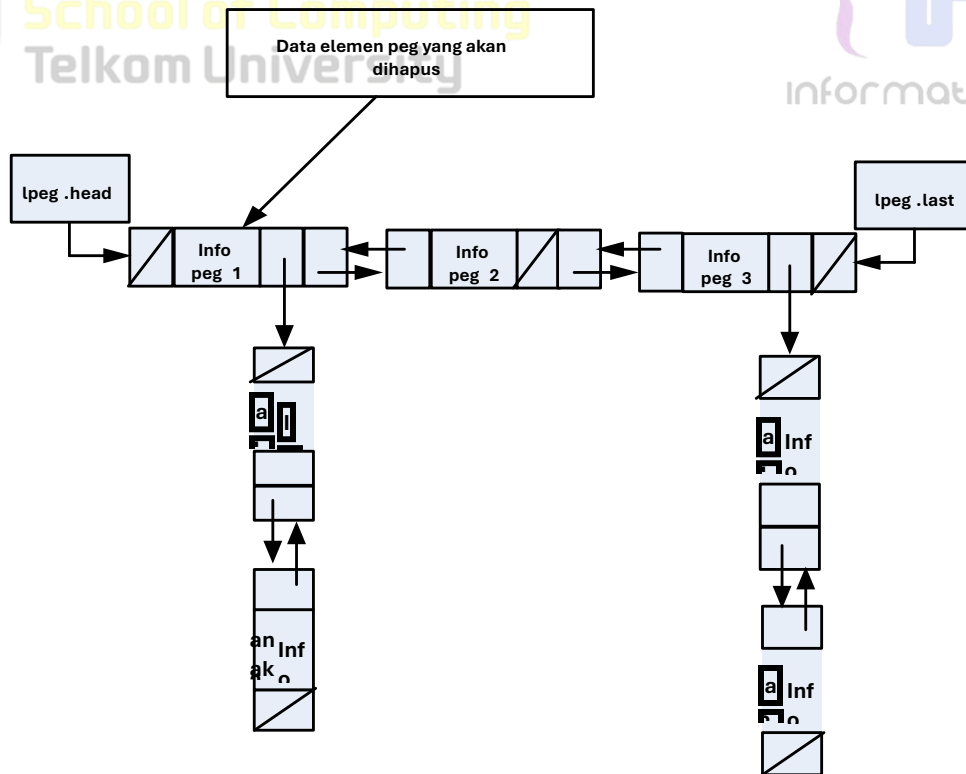
Sama dengan *insert* anak untuk *delete* anak maka harus diketahui dulu induknya. Berikut ini Gambar ilustrasinya untuk *delete last* pada induk peg 1:



B. Delete Induk

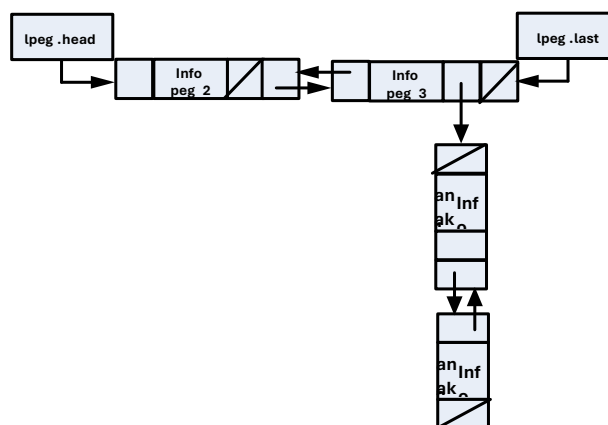
Untuk *delete* elemen induk maka saat di hapus maka seluruh anak dengan induk tersebut juga harus

dihapus. Berikut ini gambar ilustrasinya:



Data setelah elemen peg 1 yang

dihapus



```

1  /*file : multilist .h*/
2  /* contoh ADT list berkaitan 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
18 typedef struct elemen_list_induk *address; typedef
19 struct elemen_list_anak *address_anak;
20 /* define list : */
21
22 /* list kosong jika first(L)=Nil
23 setiap elemen address P dapat diacu info(P) atau next(P)
24 elemen terakhir list jika addressnya last, maka next(last) = Nil */ struct
25 elemen_list_anak{
26 /* struct ini untuk menyimpan elemen anak dan pointer
27 penunjuk elemen tetangganya */ infotypeanak info;
28 address_anak next; address_anak prev;
29 };
30
31 struct listanak {
32 /* struct ini digunakan untuk menyimpan list anak itu sendiri */
33 address_anak first; address_anak last;
34 };
35
36 struct elemen_list_induk{
37 /* struct ini untuk menyimpan elemen induk dan pointer
38 penunjuk elemen tetangganya */ infotypeanak info;
39 listanak lanak; address next; address prev;
40 };
41
42
43

```



```

44 };
45 struct listinduk {
46 /* struct ini digunakan untuk menyimpan list induk itu sendiri */      address
47 first;
48     address last;
49 };
50
51 /****** pengecekan apakah list kosong *****/
52 boolean ListEmpty(listinduk L);
53 /*mengembalikan nilai true jika list induk kosong*/ boolean
54 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*/ void
61 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     nilai
69     address Nil */
70
71 address_anak alokasiAnak(infotypeanak P);
72 /* mengirimkan address dari alokasi sebuah elemen anak     jika alokasi
73     berhasil, maka nilai address tidak Nil dan jika gagal
74     nilai address_anak Nil */
75 void dealokasi(address P);
76 /* I.S. P terdefinisi
77     F.S. memori yang digunakan P dikembalikan ke sistem */
78 void dealokasiAnak(address_anak P);
79 /* I.S. P terdefinisi
80     F.S. memori yang digunakan P dikembalikan ke sistem */
81 /****** pencarian sebuah elemen list *****/
82 address findElm(listinduk L, infotypeinduk X); /* mencari
83     apakah ada elemen list dengan info(P) = X
84     jika ada, mengembalikan address elemen tab tsb, dan Nil jika sebaliknya
85     */
86 address_anak findElm(listanak Lanak, infotypeanak X); /*
87     mencari apakah ada elemen list dengan info(P) = X
88     jika ada, mengembalikan address elemen tab tsb, dan Nil jika sebaliknya
89     */
90 boolean fFindElm(listinduk L, address P);

```


91	/* mencari apakah ada elemen list dengan alamat P
92	mengembalikan true jika ada dan false jika tidak ada */
93	boolean fFindElmanak(listanak Lanak, address_anak P); /*
94	mencari apakah ada elemen list dengan alamat P
95	mengembalikan true jika ada dan false jika tidak ada */
96	
97	address findBefore(listinduk L, address P); /*
98	mengembalikan address elemen sebelum P
99	jika P berada pada awal list, maka mengembalikan nilai Nil */
100	address_anak findBeforeAnak(listanak Lanak, infotypeinduk X, address_anak
101	P);
102	/* mengembalikan address elemen sebelum P dimana info(P) = X
103	jika P berada pada awal list, maka mengembalikan nilai Nil */
104	/****** penambahan elemen *****/
105	void insertFirst(listinduk &L, address P);
106	/* I.S. sembarang, P sudah dialokasikan
107	F.S. menempatkan elemen beralamat P pada awal list */
108	
109	
110	


```

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

```

158	F.S. adalah alamat dari alamat elemen terakhir list sebelum
159	elemen terakhir list dihapus
160	elemen terakhir list hilang dan list mungkin menjadi kosong last
161	elemen yang baru adalah successor last elemen yang lama */
162	
163	void delAfterAnak(listanak &L, address_anak &P, address_anak Prec);
164	/* I.S. list tidak kosng, Prec alamat salah satu elemen list
165	F.S. P adalah alamatdari next(Prec), menghapus next(Prec) dari list */ void
166	delPAnak (listanak &L, infotypeanak X);
167	/* I.S. sembarang
168	F.S. jika ada elemen list dengan alamat P, dimana info(P)=X, maka P dihapus
169	dan P di-dealokasi, jika tidak ada maka list tetap list
170	mungkin akan menjadi kosong karena penghapusan */
171	
172	
173	
174	
175	
176	
177	

```

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

```

3. Guided

1.

Kode Program:

```

#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 temp;
        }
    }

```

```

        delete childTemp;
    }
    delete temp;
}
};

int main() {
    MultiLinkedList mList;

    mList.addParent(1);
    mList.addParent(2);
    mList.addParent(3);

    mList.addChild(1, 10);
    mList.addChild(1, 11);
    mList.addChild(2, 20);
    mList.addChild(2, 20);
    mList.addChild(3, 30);
    mList.addChild(3, 30);
    mList.display();

    return 0;
}

```

Hasil Outputnya:

```

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 : 0.898 s
Press any key to continue.

```

2.

Kode Progam:

```
#include <iostream>
```

```

#include <string>

using namespace std;

struct EmployeeNode {
    string name;
    EmployeeNode* next;
    EmployeeNode* subordinate;

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

class EmployeeList {
private:
    EmployeeNode* head;

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

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

    void addSubordinate(string managerName, string subordinateName) {
        EmployeeNode* manager = head;
        while (manager != nullptr && manager->name != managerName) {
            manager = manager->next;
        }
        if (manager != nullptr) {
            EmployeeNode* newSubordinate = new EmployeeNode(subordinateName);
            newSubordinate->next = manager->subordinate;
            manager->subordinate = newSubordinate;
        } else {
            cout << "Manager not found!" << endl;
        }
    }

    void display() {
        EmployeeNode* current = head;
        while (current != nullptr) {
            cout << "Manager: " << current->name << " -> ";
            EmployeeNode* sub = current->subordinate;

```



```

        while (sub != nullptr) {
            cout << sub->name << " ";
            sub = sub->next;
        }
        cout << endl;
        current = current->next;
    }
}

~EmployeeList() {

    while (head != nullptr) {
        EmployeeNode* temp = head;
        head = head->next;

        while (temp->subordinate != nullptr) {
            EmployeeNode* subTemp = temp->subordinate;
            temp->subordinate = temp->subordinate->next;
            delete subTemp;
        }
        delete temp;
    }
}

};

int main() {
    EmployeeList empList;

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

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

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

    empList.display();

    return 0;
}

```

Hasil Outputnya:

```
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 : 0.079 s
Press any key to continue.
```

3.

Kode Program:

```
#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

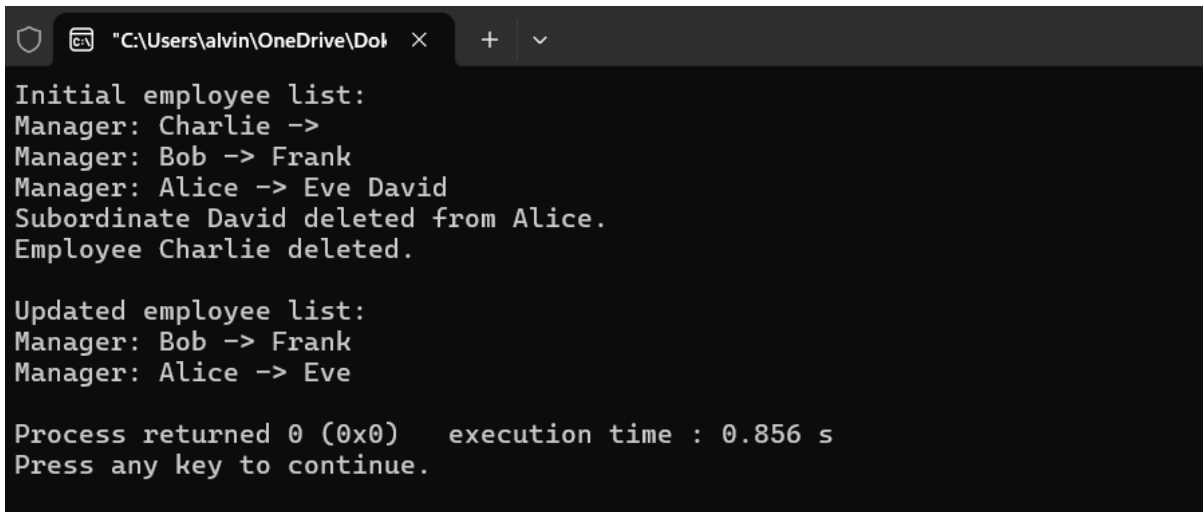
    empList.deleteSubordinate("Alice", "David"); // Menghapus David dari Alice
    empList.deleteEmployee("Charlie"); // Menghapus Charlie

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

    return 0;
}

```

Hasil Outputnya:



```

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 : 0.856 s
Press any key to continue.

```

4. Unguided

1.

Kode Program:

```
#include <iostream>
```

```

#include <string>

using namespace std;

// Struktur untuk menyimpan data proyek
struct Proyek {
    string namaProyek;
    int durasi; // durasi dalam bulan
    Proyek* next; // pointer ke proyek berikutnya
};

// Struktur untuk menyimpan data pegawai
struct Pegawai {
    string namaPegawai;
    string idPegawai;
    Proyek* listProyek; // list anak untuk proyek
    Pegawai* next; // pointer ke pegawai berikutnya
};

// Fungsi untuk menambahkan proyek ke pegawai
void tambahProyek(Pegawai* pegawai, string namaProyek, int durasi) {
    Proyek* proyekBaru = new Proyek;
    proyekBaru->namaProyek = namaProyek;
    proyekBaru->durasi = durasi;
    proyekBaru->next = nullptr;

    if (pegawai->listProyek == nullptr) {
        pegawai->listProyek = proyekBaru;
    } else {
        Proyek* temp = pegawai->listProyek;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = proyekBaru;
    }
}

// Fungsi untuk menghapus proyek berdasarkan nama proyek
void hapusProyek(Pegawai* pegawai, string namaProyek) {
    Proyek* current = pegawai->listProyek;
    Proyek* previous = nullptr;

    while (current != nullptr && current->namaProyek != namaProyek) {
        previous = current;
        current = current->next;
    }

    if (current != nullptr) {
        if (previous == nullptr) {
            pegawai->listProyek = current->next;

```

```

    } else {
        previous->next = current->next;
    }
    delete current;
}
}

// Fungsi untuk menampilkan data pegawai dan proyek
void tampilkanData(Pegawai* pegawai) {
    while (pegawai != nullptr) {
        cout << "Nama Pegawai: " << pegawai->namaPegawai << ", ID: " <<
pegawai->idPegawai << endl;
        Proyek* proyek = pegawai->listProyek;
        while (proyek != nullptr) {
            cout << "\tProyek: " << proyek->namaProyek << ", Durasi: " <<
proyek->durasi << " bulan" << endl;
            proyek = proyek->next;
        }
        pegawai = pegawai->next;
    }
}

int main() {
    // Membuat list pegawai
    Pegawai* head = nullptr;

    // Menambahkan pegawai
    Pegawai* pegawai1 = new Pegawai{"Andi", "P001", nullptr, nullptr};
    Pegawai* pegawai2 = new Pegawai{"Budi", "P002", nullptr, nullptr};
    Pegawai* pegawai3 = new Pegawai{"Citra", "P003", nullptr, nullptr};

    head = pegawai1;
    pegawai1->next = pegawai2;
    pegawai2->next = pegawai3;

    // Menambahkan proyek ke pegawai
    tambahProyek(pegawai1, "Aplikasi Mobile", 12);
    tambahProyek(pegawai2, "Sistem Akuntansi", 8);
    tambahProyek(pegawai3, "E-commerce", 10);

    // Menambahkan proyek baru untuk Andi
    tambahProyek(pegawai1, "Analisis Data", 6);

    // Menghapus proyek "Aplikasi Mobile" dari Andi
    hapusProyek(pegawai1, "Aplikasi Mobile");

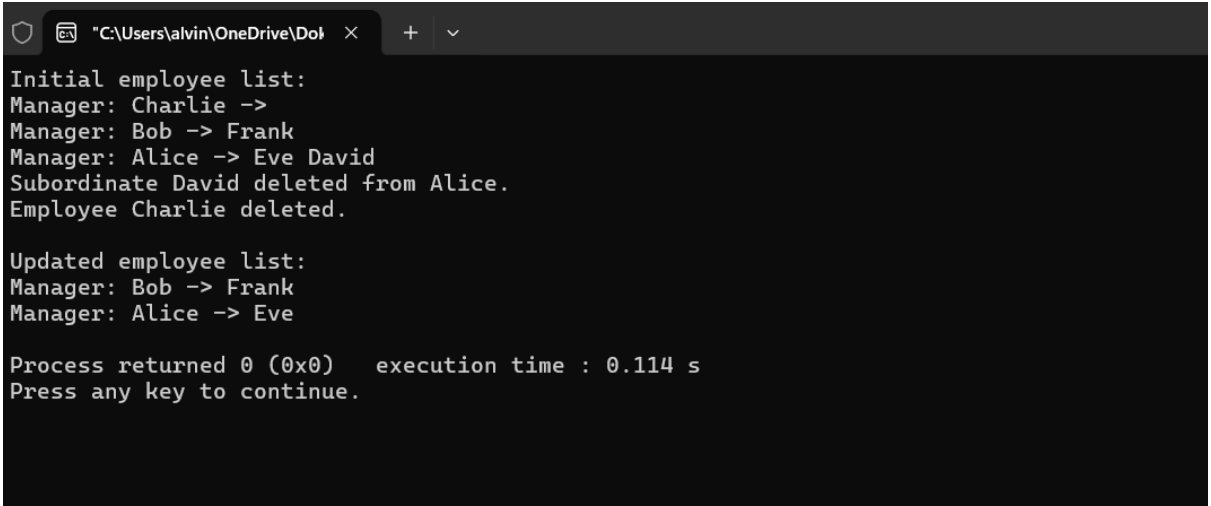
    // Menampilkan data pegawai dan proyek
    tampilkanData(head);

    return 0;
}

```

```
}
```

Hasil Outputnya:



```
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 : 0.114 s
Press any key to continue.
```

2.

Kode Program

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

using namespace std;

// Struktur untuk menyimpan data buku
struct Buku {
    string judulBuku;
    string tanggalPengembalian;
    Buku* next; // pointer ke buku berikutnya
};

// Struktur untuk menyimpan data anggota
struct Anggota {
    string namaAnggota;
    string idAnggota;
    Buku* listBuku; // list anak untuk buku
    Anggota* next; // pointer ke anggota berikutnya
};

// Fungsi untuk menambahkan buku ke anggota
void tambahBuku(Anggota* anggota, string judulBuku, string
tanggalPengembalian) {
    Buku* bukuBaru = new Buku;
    bukuBaru->judulBuku = judulBuku;
```



```

bukuBaru->tanggalPengembalian = tanggalPengembalian;
bukuBaru->next = nullptr;

if (anggota->listBuku == nullptr) {
    anggota->listBuku = bukuBaru;
} else {
    Buku* temp = anggota->listBuku;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = bukuBaru;
}
}

// Fungsi untuk menghapus anggota beserta buku yang dipinjam
void hapusAnggota(Anggota*& head, string idAnggota) {
    Anggota* current = head;
    Anggota* previous = nullptr;

    while (current != nullptr && current->idAnggota != idAnggota) {
        previous = current;
        current = current->next;
    }

    if (current != nullptr) {
        if (previous == nullptr) {
            head = current->next;
        } else {
            previous->next = current->next;
        }

        Buku* bukuTemp = current->listBuku;
        while (bukuTemp != nullptr) {
            Buku* bukuHapus = bukuTemp;
            bukuTemp = bukuTemp->next;
            delete bukuHapus;
        }
        delete current;
    }
}

// Fungsi untuk menampilkan data anggota dan buku yang dipinjam
void tampilkanData(Anggota* anggota) {
    while (anggota != nullptr) {
        cout << "Nama Anggota: " << anggota->namaAnggota << ", ID: " <<
anggota->idAnggota << endl;
        Buku* buku = anggota->listBuku;
        while (buku != nullptr) {
            cout << "\tBuku: " << buku->judulBuku << ", Tanggal Pengembalian:
" << buku->tanggalPengembalian << endl;

```

```

        buku = buku->next;
    }
    anggota = anggota->next;
}

int main() {
    // Membuat list anggota
    Anggota* head = nullptr;

    // Menambahkan anggota
    Anggota* anggota1 = new Anggota{"Rani", "A001", nullptr, nullptr};
    Anggota* anggota2 = new Anggota{"Dito", "A002", nullptr, nullptr};
    Anggota* anggota3 = new Anggota{"Vina", "A003", nullptr, nullptr};

    head = anggota1;
    anggota1->next = anggota2;
    anggota2->next = anggota3;

    // Menambahkan buku yang dipinjam
    tambahBuku(anggota1, "Pemrograman C++", "01/12/2024");
    tambahBuku(anggota2, "Algoritma Pemrograman", "15/12/2024");
    tambahBuku(anggota3, "Struktur Data", "10/12/2024");

    // Menambahkan buku baru untuk Rani
    tambahBuku(anggota1, "Basis Data", "20/12/2024");

    // Menghapus anggota Dito beserta buku yang dipinjam
    hapusAnggota(head, "A002");

    // Menampilkan data anggota dan buku yang dipinjam
    tampilkanData(head);

    return 0;
}

```

Hasil Outputnya:

```
"C:\Users\alvin\OneDrive\Dot  x + v
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 : 0.043 s
Press any key to continue.
```

5. Kesimpulan

Kesimpulan dari program ini adalah bahwa implementasi **Multi Linked List** memungkinkan kita untuk menyimpan data yang lebih kompleks, seperti hubungan antara entitas yang memiliki keterkaitan satu sama lain, dalam hal ini antara pegawai dan proyek atau anggota dan buku. Dengan menggunakan konsep linked list ganda, di mana setiap node dapat memiliki child node, kita dapat dengan mudah menambah, menghapus, dan menampilkan data sesuai kebutuhan.

Pada program ini, setiap parent node merepresentasikan entitas utama (seperti pegawai atau anggota), sementara child node merepresentasikan entitas yang terkait (seperti proyek atau buku yang dipinjam). Program ini juga memastikan tidak ada data yang duplikat dengan memeriksa keberadaan child yang sama sebelum menambahkannya. Dengan demikian, program ini lebih efisien dalam pengelolaan data dan meminimalkan kesalahan yang bisa terjadi akibat duplikasi data.

Selain itu, program ini juga menangani alokasi dan dealokasi memori dengan baik, menggunakan destructor untuk menghapus node dan child node secara tepat sehingga menghindari kebocoran memori. Dengan penambahan fitur pengecekan duplikat dan format tampilan yang lebih jelas, program ini menjadi lebih user-friendly dan mudah dipahami.