# LAPORAN PRAKTIKUM
# Modul 13
# Multi Linked List


**Disusun Oleh:**
**Zhafir Zaidan Avail (2311104059)**
**S1-SE-07-2**

**Dosen :**
**Wahyu Andi Saputra, S.Pd., M.Eng**

**PROGRAM STUDI S1 SOFTWARE ENGINEERING**
**FAKULTAS INFORMATIKA**
**TELKOM UNIVERSITY**
**PURWOKERTO**

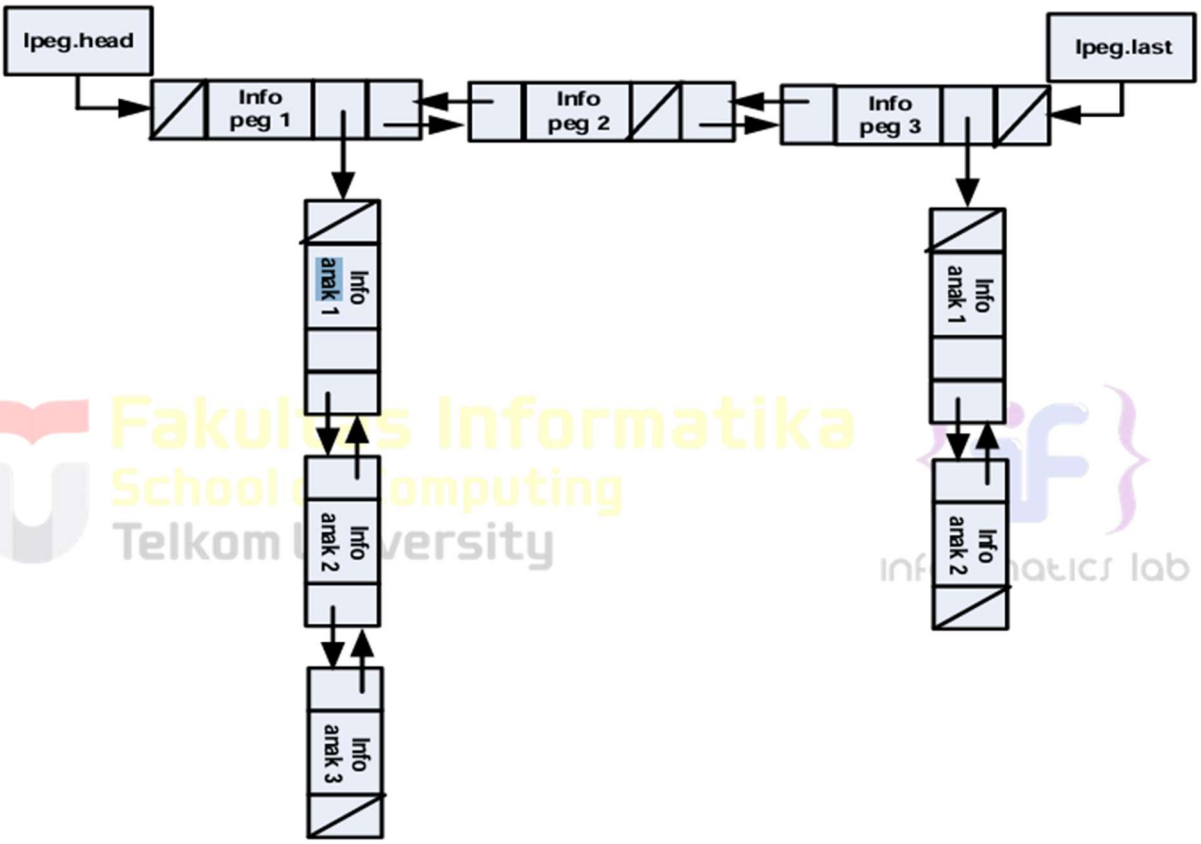
**2024**

## 1. Tujuan

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

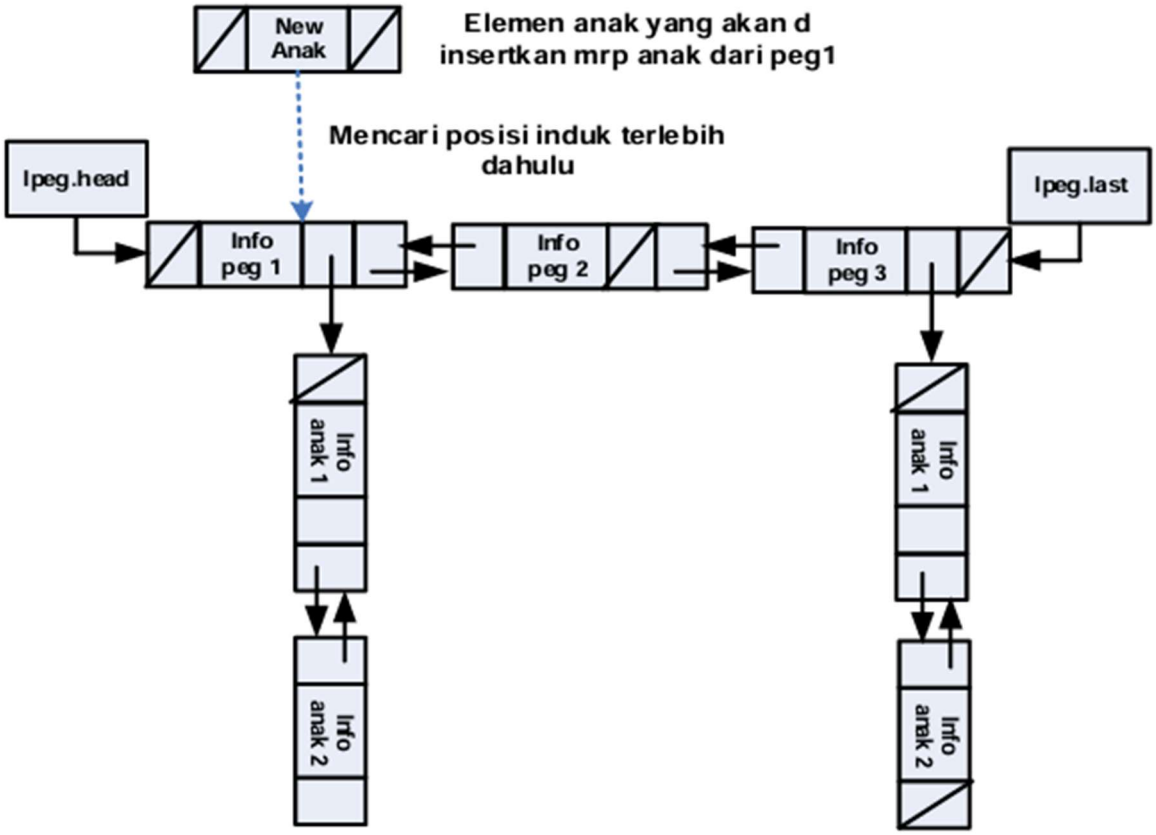
## 2. Landasan Teori

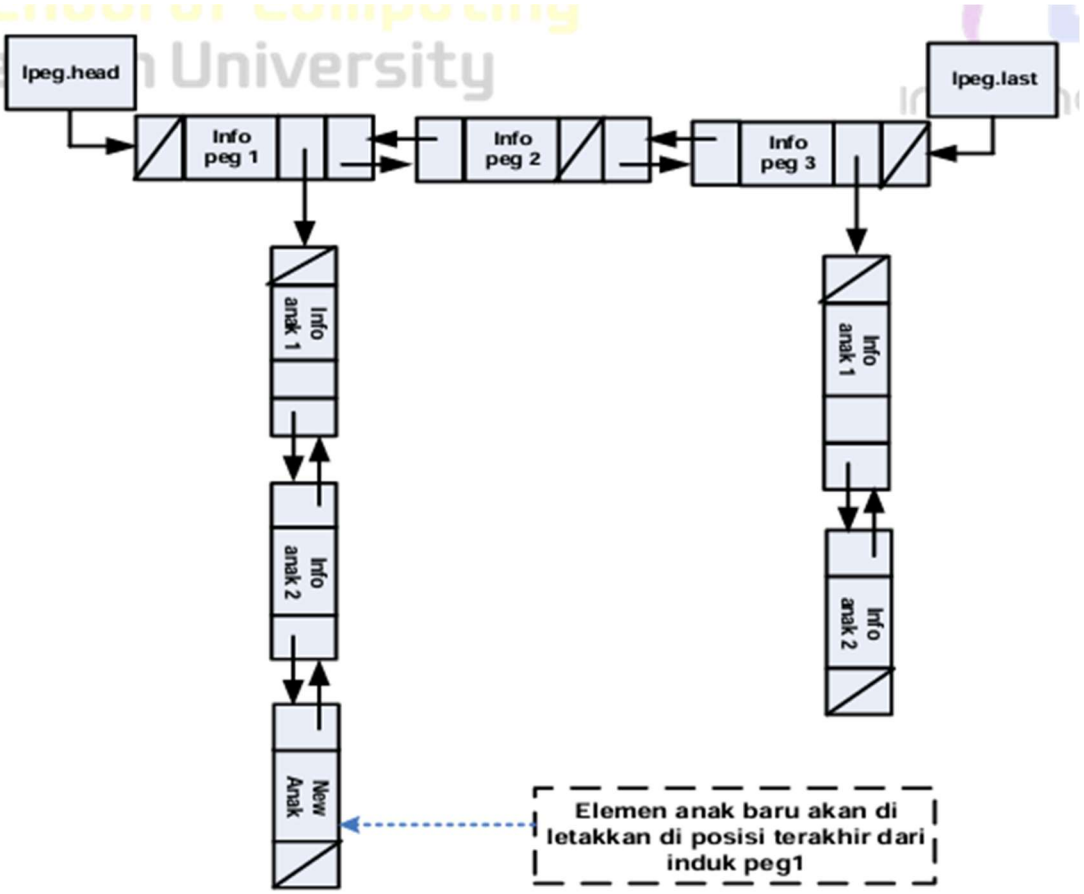
Multi List merupakan sekumpulan list yang berbeda yang memiliki suatu keterhubungan satu sama lain. Tiap elemen dalam multi link list dapat membentuk list sendiri.Biasanya ada yang bersifat sebagai list induk dan list anak.




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.

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


New Anak
Elemen anak yang akan d insertkan mrp anak dari peg1
Mencari posisi induk terlebih dahulu
lpeg.head
lpeg.last
Info peg 1
Info peg 2
Info peg 3
Info anak 1
Info anak 2
Info anak 1
Info anak 2



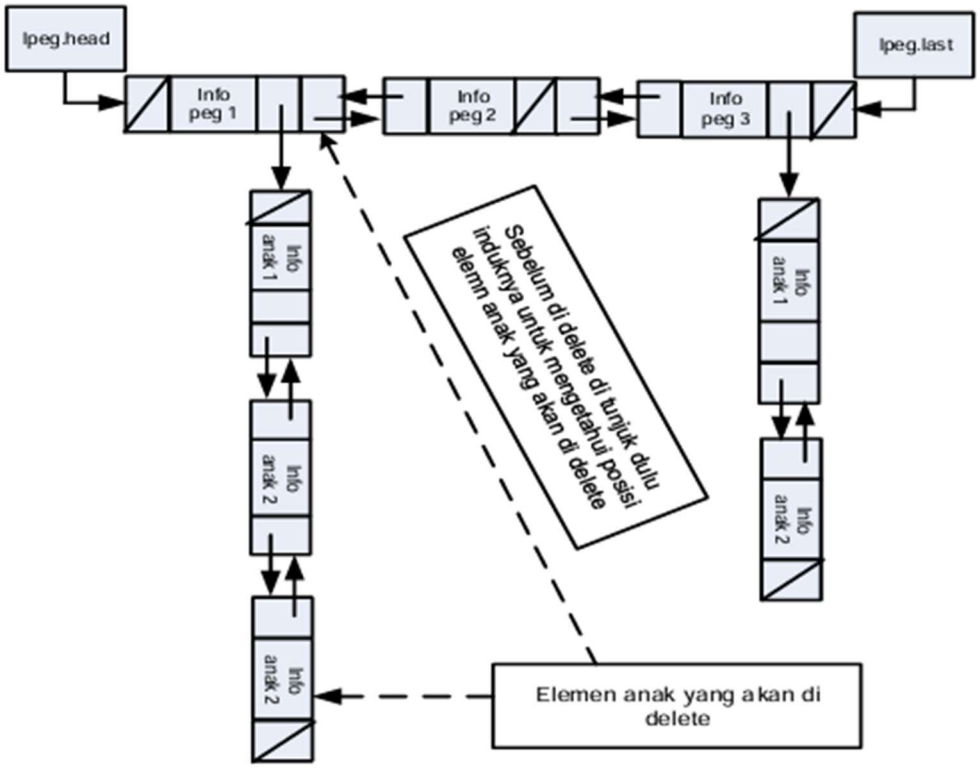

```
/* buat dahulu elemen yang akan disisipkan */ address_anak
alokasiAnak(infotypeanak X){
address_anak p = alokasi(X);
next(p) = null;
prev(p) = null;
return p;
}
/* mencari apakah ada elemen pegawai dengan info X */
address findElm(listinduk L, infotypeinduk X){
address cariInduk = head(L);
do{
if(cariInduk.info == 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;
}
```

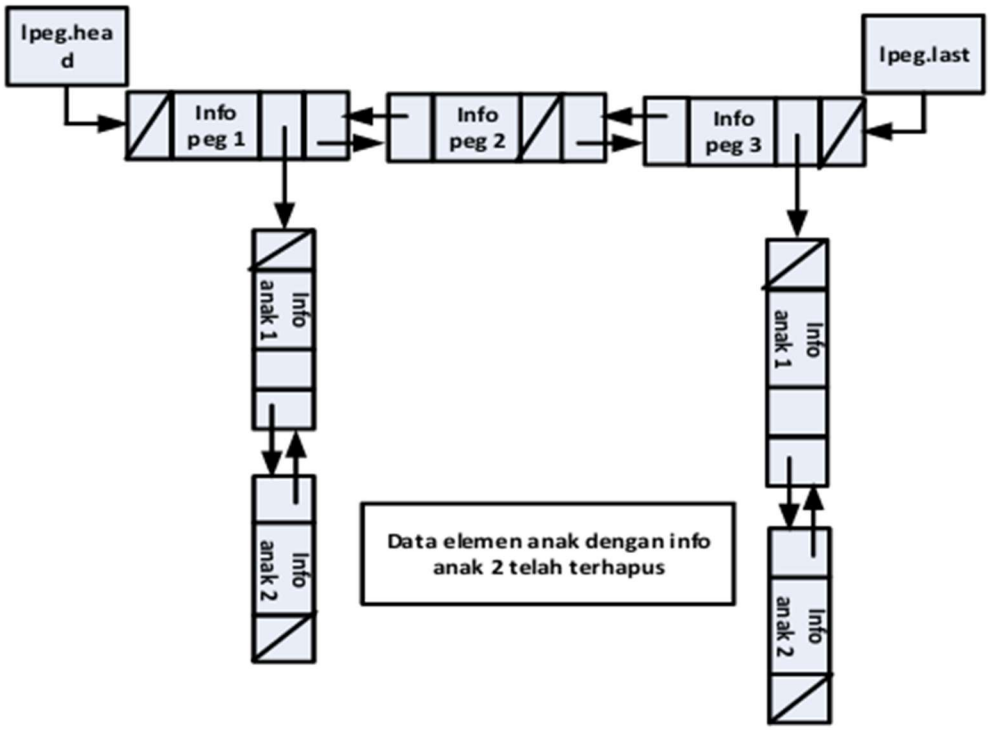
B. Insert Induk

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

linked list.

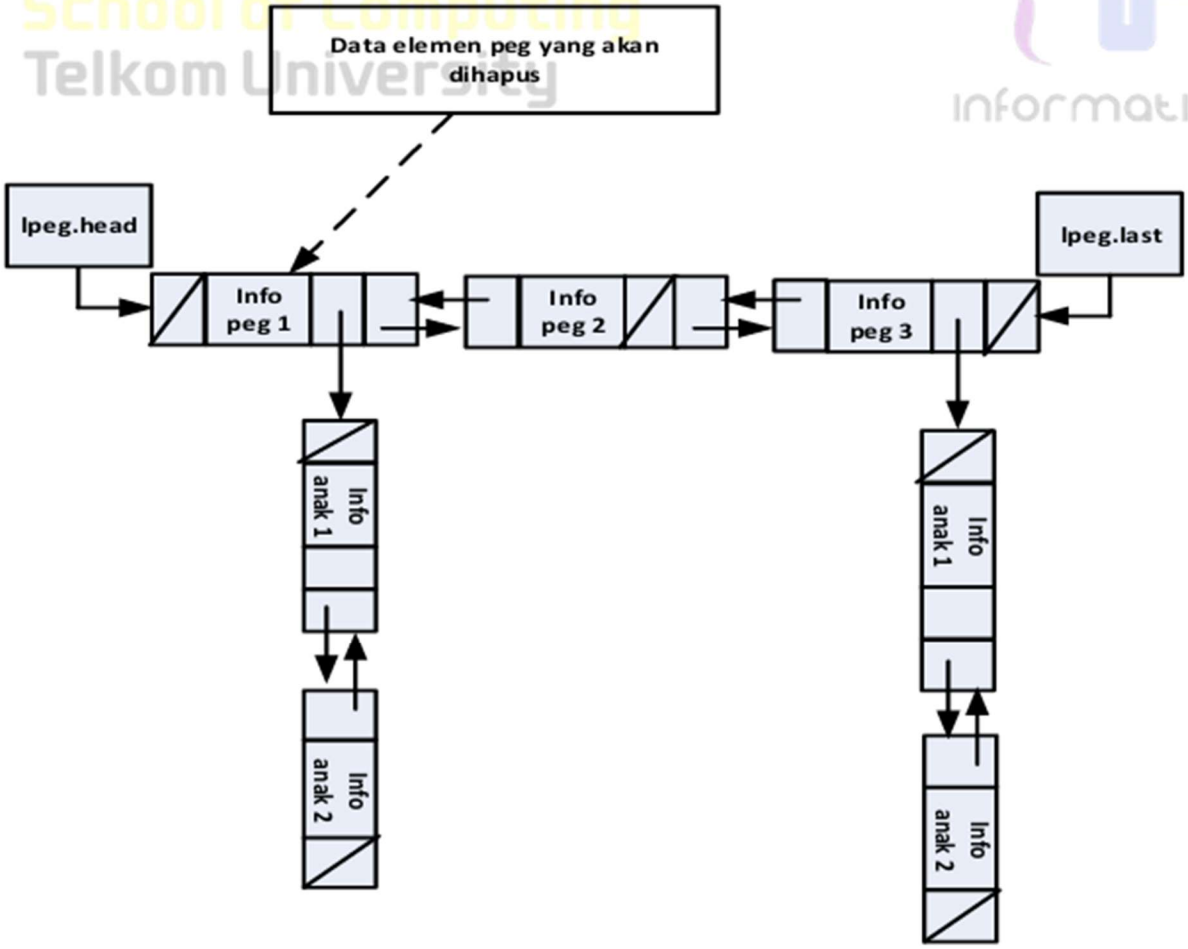
2. Delete
   A. Delete Anak
      Sama dengan insert anak untuk delete anak maka harus diketahui dulu induknya.Berikut ini Gambar ilustrasinya untuk delete last pada induk peg 1:

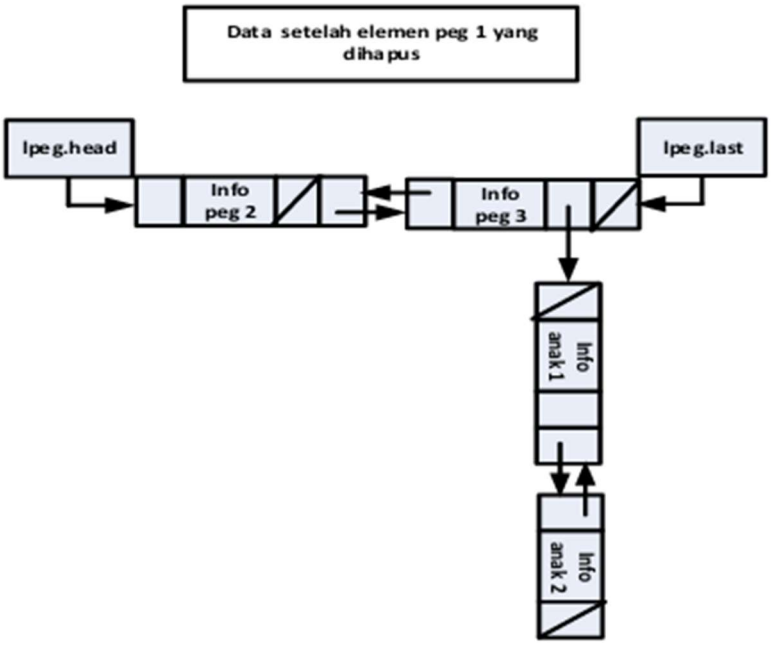





B. Delete Induk

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




```
/*file : multilist .h*/
```

```
/* contoh ADT list berkait dengan representasi fisik pointer*/
/* representasi address dengan pointer*/

/* info tipe adalah integer */
#ifndef MULTILIST_H_INCLUDED
#define MULTILIST_H_INCLUDED
#include <stdio.h>
#define Nil NULL
#define info(P) (P)->info
#define next(P) (P)->next
#define first(L) ((L).first)
#define last(L) ((L).last)

typedef int infotypeanak;
typedef int infotypeinduk;
typedef struct elemen_list_induk *address;
typedef struct elemen_list_anak *address_anak;
/* define list : */

/* list kosong jika first(L)=Nil
setiap elemen address P dapat diacu info(P) atau next(P)
elemen terakhir list jika addressnya last, maka next(last) = Nil */
struct elemen_list_anak{
/* struct ini untuk menyimpan elemen anak dan pointer penunjuk
   elemen tetangganya */
 infotypeanak info;
 address_anak next;
 address_anak prev;
};

struct listanak {
/* struct ini digunakan untuk menyimpan list anak itu sendiri */
 address_anak first;
 address_anak last;
};

struct elemen_list_induk{
/* struct ini untuk menyimpan elemen induk dan pointer penunjuk
   elemen tetangganya */
 infotypeanak info;
 listanak lanak;
 address next;
 address prev;
};
struct listinduk {
/* struct ini digunakan untuk menyimpan list induk itu sendiri */
 address first;
 address last;
};

/********* pengecekan apakah list kosong ***********/
boolean ListEmpty(listinduk L);
/*mengembalikan nilai true jika list induk kosong*/
boolean ListEmptyAnak(listanak L);
/*mengembalikan nilai true jika list anak kosong*/

/********* pembuatan list kosong *********/
void CreateList(listinduk &L);
/* I.S. sembarang
   F.S. terbentuk list induk kosong*/
void CreateListAnak(listanak &L);
/* I.S. sembarang
   F.S. terbentuk list anak kosong*/

/********* manajemen memori *********/
address alokasi(infotypeinduk P);
```

```
/* mengirimkan address dari alokasi sebuah elemen induk
   jika alokasi berhasil, maka nilai address tidak Nil dan jika gagal
   nilai address Nil */

address_anak alokasiAnak(infotypeanak P);
/* mengirimkan address dari alokasi sebuah elemen anak
   jika alokasi berhasil, maka nilai address tidak Nil dan jika gagal
   nilai address_anak Nil */

void dealokasi(address P);
/* I.S. P terdefinisi
   F.S. memori yang digunakan P dikembalikan ke sistem */

void dealokasiAnak(address_anak P);
/* I.S. P terdefinisi
   F.S. memori yang digunakan P dikembalikan ke sistem */
/********* pencarian sebuah elemen list *********/
address findElm(listinduk L, infotypeinduk X);
/* mencari apakah ada elemen list dengan info(P) = X
   jika ada, mengembalikan address elemen tab tsb, dan Nil jika sebaliknya
*/
address_anak findElm(listanak Lanak, infotypeanak X);
/* mencari apakah ada elemen list dengan info(P) = X
   jika ada, mengembalikan address elemen tab tsb, dan Nil jika sebaliknya
*/
boolean fFindElm(listinduk L, address P);
/* mencari apakah ada elemen list dengan alamat P
   mengembalikan true jika ada dan false jika tidak ada */
boolean fFindElmanak(listanak Lanak, address_anak P);
/* mencari apakah ada elemen list dengan alamat P
   mengembalikan true jika ada dan false jika tidak ada */

address findBefore(listinduk L, address P);
/* mengembalikan address elemen sebelum P
   jika P berada pada awal list, maka mengembalikan nilai Nil */
address_anak findBeforeAnak(listanak Lanak, infotypeinduk X, address_anak
P);
/* mengembalikan address elemen sebelum P dimana info(P) = X
   jika P berada pada awal list, maka mengembalikan nilai Nil */

/********* penambahan elemen **********/
void insertFirst(listinduk &L, address P);
/* I.S. sembarang, P sudah dialokasikan
   F.S. menempatkan elemen beralamat P pada awal list */

void insertAfter(listinduk &L, address P, address Prec);
/* I.S. sembarang, P dan Prec alamt salah satu elemen list
   F.S. menempatkan elemen beralamat P sesudah elemen beralamat Prec */

void insertLast(listinduk &L, address P);
/* I.S. sembarang, P sudah dialokasikan
   F.S. menempatkan elemen beralamat P pada akhir list */

void insertFirstAnak(listanak &L, address_anak P);
/* I.S. sembarang, P sudah dialokasikan
   F.S. menempatkan elemen beralamat P pada awal list */

void insertAfterAnak(listanak &L, address_anak P, address_anak Prec);
/* I.S. sembarang, P dan Prec alamt salah satu elemen list
   F.S. menempatkan elemen beralamat P sesudah elemen beralamat Prec */

void insertLastAnak(listanak &L, address_anak P);
/* I.S. sembarang, P sudah dialokasikan
   F.S. menempatkan elemen beralamat P pada akhir list */

/********* penghapusan sebuah elemen *********/
```

```
void delFirst(listinduk &L, address &P);
/* I.S. list tidak kosong
   F.S. adalah alamat dari alamat elemen pertama list
   sebelum elemen pertama list dihapus
   elemen pertama list hilang dan list mungkin menjadi kosong
   first elemen yang baru adalah successor first elemen yang lama */
void delLast(listinduk &L, address &P);
/* I.S. list tidak kosong
   F.S. adalah alamat dari alamat elemen terakhir list
   sebelum elemen terakhir list dihapus
   elemen terakhir list hilang dan list mungkin menjadi kosong
   last elemen yang baru adalah successor last elemen yang lama */

void delAfter(listinduk &L, address &P, address Prec);
/* I.S. list tidak kosng, Prec alamat salah satu elemen list
   F.S. P adalah alamatdari next(Prec), menghapus next(Prec) dari list */
void delP (listinduk &L, infotypeinduk X);
/* I.S. sembarang
   F.S. jika ada elemen list dengan alamat P, dimana info(P)=X, maka P
dihapus
   dan P di-dealokasi, jika tidak ada maka list tetap
   list mungkin akan menjadi kosong karena penghapusan */

void delFirstAnak(listanak &L, address_anak &P);
/* I.S. list tidak kosong
   F.S. adalah alamat dari alamat elemen pertama list
   sebelum elemen pertama list dihapus
   elemen pertama list hilang dan list mungkin menjadi kosong
   first elemen yang baru adalah successor first elemen yang lama */
void delLastAnak(listanak &L, address_anak &P);
/* I.S. list tidak kosong
   F.S. adalah alamat dari alamat elemen terakhir list
   sebelum elemen terakhir list dihapus
   elemen terakhir list hilang dan list mungkin menjadi kosong
   last elemen yang baru adalah successor last elemen yang lama */

void delAfterAnak(listanak &L, address_anak &P, address_anak Prec);
/* I.S. list tidak kosng, Prec alamat salah satu elemen list
   F.S. P adalah alamatdari next(Prec), menghapus next(Prec) dari list */
void delPAnak (listanak &L, infotypeanak X);
/* I.S. sembarang
   F.S. jika ada elemen list dengan alamat P, dimana info(P)=X, maka P
dihapus
   dan P di-dealokasi, jika tidak ada maka list tetap
   list mungkin akan menjadi kosong karena penghapusan */
/********** proses semau elemen list *********/
void printInfo(list L);
/* I.S. list mungkin kosong
   F.S. jika list tidak kosong menampilkan semua info yang ada pada list
*/

int nbList(list L);
/* mengembalikan jumlah elemen pada list */

void printInfoAnak(listanak Lanak);
/* I.S. list mungkin kosong
   F.S. jika list tidak kosong menampilkan semua info yang ada pada list
*/

int nbListAnak(listanak Lanak);
/* mengembalikan jumlah elemen pada list anak */

/********** proses terhadap list **********/
void delAll(listinduk &L);
/* menghapus semua elemen list dan semua elemen di-dealokasi */
#endif
```

## 3. Guided

### 1. Guided 1

```
#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;
}
```

## 2. Guided 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;
}
```

## 3. Guided 3

```
#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 deleteEmployee(string name) {
        EmployeeNode** current = &head;
        while (*current != nullptr && (*current)->name != name) {
            current = &((*current)->next);
        }

        if (*current != nullptr) {
            EmployeeNode* toDelete = *current;
            *current = (*current)->next;

            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;
        }
    }

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

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

            if (*currentSub != nullptr) {
                EmployeeNode* toDelete = *currentSub;
                *currentSub = (*currentSub)->next;

                delete toDelete;
                cout << "Subordinate " << subordinateName << " deleted from " <<
managerName << "." << endl;
            } else {
                cout << "Subordinate not found!" << endl;
            }
        } 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");

    cout << "Initial employee list:" << endl;
    empList.display();

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

    cout << "\nUpdated employee list:" << endl;
    empList.display();

    return 0;
}
```

## 4. Unguided

### 1. Unguided 1

```
#include <iostream>
#include <string>
using namespace std;

struct Pustaka {
    string judulPustaka;
    string jadwalPengembalian;
    Pustaka* pustakaBerikutnya;
};

struct Pemustaka {
    string namaPemustaka;
```

```
    string idPemustaka;
    Pustaka* kepalaPustaka;
    Pemustaka* pemustakaBerikutnya;
};

Pemustaka* kepalaPemustaka = nullptr;

void tambahPemustaka(string nama, string id) {
    Pemustaka* pemustakaBaru = new Pemustaka;
    pemustakaBaru->namaPemustaka = nama;
    pemustakaBaru->idPemustaka = id;
    pemustakaBaru->kepalaPustaka = nullptr;
    pemustakaBaru->pemustakaBerikutnya = kepalaPemustaka;
    kepalaPemustaka = pemustakaBaru;
}

void tambahPustaka(string idPemustaka, string judulPustaka, string
jadwalPengembalian) {
    Pemustaka* pemustakaSaatIni = kepalaPemustaka;
    while (pemustakaSaatIni != nullptr) {
        if (pemustakaSaatIni->idPemustaka == idPemustaka) {
            Pustaka* pustakaBaru = new Pustaka;
            pustakaBaru->judulPustaka = judulPustaka;
            pustakaBaru->jadwalPengembalian = jadwalPengembalian;
            pustakaBaru->pustakaBerikutnya = pemustakaSaatIni->kepalaPustaka;
            pemustakaSaatIni->kepalaPustaka = pustakaBaru;
            return;
        }
        pemustakaSaatIni = pemustakaSaatIni->pemustakaBerikutnya;
    }
    cout << "Pemustaka dengan ID : " << idPemustaka << " tidak dapat ditemukan!\n";
}

void hapusPemustaka(string idPemustaka) {
    Pemustaka* pemustakaSaatIni = kepalaPemustaka;
    Pemustaka* pemustakaSebelumnya = nullptr;
    while (pemustakaSaatIni != nullptr) {
        if (pemustakaSaatIni->idPemustaka == idPemustaka) {
            if (pemustakaSebelumnya == nullptr) {
                kepalaPemustaka = pemustakaSaatIni->pemustakaBerikutnya;
            } else {
                pemustakaSebelumnya->pemustakaBerikutnya = pemustakaSaatIni-
>pemustakaBerikutnya;
            }
            Pustaka* pustakaSaatIni = pemustakaSaatIni->kepalaPustaka;
            while (pustakaSaatIni != nullptr) {
                Pustaka* temp = pustakaSaatIni;
                pustakaSaatIni = pustakaSaatIni->pustakaBerikutnya;
                delete temp;
            }
            delete pemustakaSaatIni;
            return;
        }
        pemustakaSebelumnya = pemustakaSaatIni;
        pemustakaSaatIni = pemustakaSaatIni->pemustakaBerikutnya;
    }
    cout << "Pemustaka dengan ID : " << idPemustaka << " tidak dapat ditemukan!\n";
}

void tampilkanDataPerpustakaan() {
    Pemustaka* pemustakaSaatIni = kepalaPemustaka;
    while (pemustakaSaatIni != nullptr) {
        cout << "Pemustaka : " << pemustakaSaatIni->namaPemustaka << " (ID : " <<
pemustakaSaatIni->idPemustaka << ")\n";
        Pustaka* pustakaSaatIni = pemustakaSaatIni->kepalaPustaka;
        while (pustakaSaatIni != nullptr) {
```

```
            cout << "  Pustaka : " << pustakaSaatIni->judulPustaka << "
(Pengembalian : " << pustakaSaatIni->jadwalPengembalian << ")\n";
            pustakaSaatIni = pustakaSaatIni->pustakaBerikutnya;
        }
        pemustakaSaatIni = pemustakaSaatIni->pemustakaBerikutnya;
    }
}

int main() {
    int menu;
    do {
        cout << "Program Sistem Manajemen Buku Perpustakaan";
        cout << "\nMenu Peminjaman Perpustakaan:\n";
        cout << "1. Tambah Nama Pemustaka\n";
        cout << "2. Tambah Nama Pustaka\n";
        cout << "3. Hapus Nama Pemustaka\n";
        cout << "4. Tampilkan Data\n";
        cout << "5. Keluar\n";
        cout << "Menu : ";
        cin >> menu;
        cin.ignore();

        if (menu == 1) {
            string nama, id;
            cout << "Masukkan Nama Pemustaka: ";
            getline(cin, nama);
            cout << "Masukkan ID Pemustaka: ";
            getline(cin, id);
            tambahPemustaka(nama, id);
        } else if (menu == 2) {
            string id, judulPustaka, jadwalPengembalian;
            cout << "Masukkan ID Pemustaka: ";
            getline(cin, id);
            cout << "Masukkan Judul Pustaka: ";
            getline(cin, judulPustaka);
            cout << "Masukkan Jadwal Pengembalian: ";
            getline(cin, jadwalPengembalian);
            tambahPustaka(id, judulPustaka, jadwalPengembalian);
        } else if (menu == 3) {
            string id;
            cout << "Masukkan ID Pemustaka yang akan dihapus: ";
            getline(cin, id);
            hapusPemustaka(id);
        } else if (menu == 4) {
            tampilkanDataPerpustakaan();
        } else if (menu != 5) {
            cout << "Pilihan tidak valid.\n";
        }
    } while (menu != 5);

    return 0;
}
```

**Output:**

```
Program Management Data Pegawai dan Proyek
Menu:
1. Tambah Nama Karyawan
2. Tambah Nama Tugas
3. Hapus Nama Tugas
4. Tampilkan Data Karyawan & Tugas
5. Keluar
Menu: 1
Masukkan Nama Karyawan : Zhafir Zaidan Avail
Masukkan ID Karyawan : 2311104059
Program Management Data Pegawai dan Proyek
Menu:
```

```
1. Tambah Nama Karyawan
2. Tambah Nama Tugas
3. Hapus Nama Tugas
4. Tampilkan Data Karyawan & Tugas
5. Keluar
Menu: 2
Masukkan ID Karyawan : 2311104059
Masukkan Nama Tugas : Pencatat Laporan Keuangan
Masukkan Durasi Tugas (bulan) : 3
Program Management Data Pegawai dan Proyek
Menu:
1. Tambah Nama Karyawan
2. Tambah Nama Tugas
3. Hapus Nama Tugas
4. Tampilkan Data Karyawan & Tugas
5. Keluar
Menu: 2
Masukkan ID Karyawan : 2311104059
Masukkan Nama Tugas : Kepala Divisi Keuangan
Masukkan Durasi Tugas (bulan) : 3
Program Management Data Pegawai dan Proyek
Menu:
1. Tambah Nama Karyawan
2. Tambah Nama Tugas
3. Hapus Nama Tugas
4. Tampilkan Data Karyawan & Tugas
5. Keluar
Menu: 3
Masukkan ID Karyawan : 2311104059
Masukkan Nama Tugas yang akan dihapus : Pencatat Laporan Keuangan
Program Management Data Pegawai dan Proyek
Menu:
1. Tambah Nama Karyawan
2. Tambah Nama Tugas
3. Hapus Nama Tugas
4. Tampilkan Data Karyawan & Tugas
5. Keluar
Menu: 4
Karyawan : Zhafir Zaidan Avail (ID: 2311104059)
  Tugas : Kepala Divisi Keuangan (Durasi : 3 bulan)
Program Management Data Pegawai dan Proyek
Menu:
1. Tambah Nama Karyawan
2. Tambah Nama Tugas
3. Hapus Nama Tugas
4. Tampilkan Data Karyawan & Tugas
5. Keluar
Menu: 5
```

## 2. Unguided 2

```
#include <iostream>
#include <string>
using namespace std;

struct Pustaka {
    string judulPustaka;
    string jadwalPengembalian;
    Pustaka* pustakaBerikutnya;
};

struct Pemustaka {
    string namaPemustaka;
    string idPemustaka;
    Pustaka* kepalaPustaka;
    Pemustaka* pemustakaBerikutnya;
};
```

```
Pemustaka* kepalaPemustaka = nullptr;

void tambahPemustaka(string nama, string id) {
    Pemustaka* pemustakaBaru = new Pemustaka;
    pemustakaBaru->namaPemustaka = nama;
    pemustakaBaru->idPemustaka = id;
    pemustakaBaru->kepalaPustaka = nullptr;
    pemustakaBaru->pemustakaBerikutnya = kepalaPemustaka;
    kepalaPemustaka = pemustakaBaru;
}

void tambahPustaka(string idPemustaka, string judulPustaka, string
jadwalPengembalian) {
    Pemustaka* pemustakaSaatIni = kepalaPemustaka;
    while (pemustakaSaatIni != nullptr) {
        if (pemustakaSaatIni->idPemustaka == idPemustaka) {
            Pustaka* pustakaBaru = new Pustaka;
            pustakaBaru->judulPustaka = judulPustaka;
            pustakaBaru->jadwalPengembalian = jadwalPengembalian;
            pustakaBaru->pustakaBerikutnya = pemustakaSaatIni->kepalaPustaka;
            pemustakaSaatIni->kepalaPustaka = pustakaBaru;
            return;
        }
        pemustakaSaatIni = pemustakaSaatIni->pemustakaBerikutnya;
    }
    cout << "Pemustaka dengan ID : " << idPemustaka << " tidak dapat ditemukan!\n";
}

void hapusPemustaka(string idPemustaka) {
    Pemustaka* pemustakaSaatIni = kepalaPemustaka;
    Pemustaka* pemustakaSebelumnya = nullptr;
    while (pemustakaSaatIni != nullptr) {
        if (pemustakaSaatIni->idPemustaka == idPemustaka) {
            if (pemustakaSebelumnya == nullptr) {
                kepalaPemustaka = pemustakaSaatIni->pemustakaBerikutnya;
            } else {
                pemustakaSebelumnya->pemustakaBerikutnya = pemustakaSaatIni-
>pemustakaBerikutnya;
            }
            Pustaka* pustakaSaatIni = pemustakaSaatIni->kepalaPustaka;
            while (pustakaSaatIni != nullptr) {
                Pustaka* temp = pustakaSaatIni;
                pustakaSaatIni = pustakaSaatIni->pustakaBerikutnya;
                delete temp;
            }
            delete pemustakaSaatIni;
            return;
        }
        pemustakaSebelumnya = pemustakaSaatIni;
        pemustakaSaatIni = pemustakaSaatIni->pemustakaBerikutnya;
    }
    cout << "Pemustaka dengan ID : " << idPemustaka << " tidak dapat ditemukan!\n";
}

void tampilkanDataPerpustakaan() {
    Pemustaka* pemustakaSaatIni = kepalaPemustaka;
    while (pemustakaSaatIni != nullptr) {
        cout << "Pemustaka : " << pemustakaSaatIni->namaPemustaka << " (ID : " <<
pemustakaSaatIni->idPemustaka << ")\n";
        Pustaka* pustakaSaatIni = pemustakaSaatIni->kepalaPustaka;
        while (pustakaSaatIni != nullptr) {
            cout << "  Pustaka : " << pustakaSaatIni->judulPustaka << "
(Pengembalian : " << pustakaSaatIni->jadwalPengembalian << ")\n";
            pustakaSaatIni = pustakaSaatIni->pustakaBerikutnya;
        }
```

```
        pemustakaSaatIni = pemustakaSaatIni->pemustakaBerikutnya;
    }
}

int main() {
    int menu;
    do {
        cout << "Program Sistem Manajemen Buku Perpustakaan";
        cout << "\nMenu Peminjaman Perpustakaan:\n";
        cout << "1. Tambah Nama Pemustaka\n";
        cout << "2. Tambah Nama Pustaka\n";
        cout << "3. Hapus Nama Pemustaka\n";
        cout << "4. Tampilkan Data\n";
        cout << "5. Keluar\n";
        cout << "Menu : ";
        cin >> menu;
        cin.ignore();

        if (menu == 1) {
            string nama, id;
            cout << "Masukkan Nama Pemustaka: ";
            getline(cin, nama);
            cout << "Masukkan ID Pemustaka: ";
            getline(cin, id);
            tambahPemustaka(nama, id);
        } else if (menu == 2) {
            string id, judulPustaka, jadwalPengembalian;
            cout << "Masukkan ID Pemustaka: ";
            getline(cin, id);
            cout << "Masukkan Judul Pustaka: ";
            getline(cin, judulPustaka);
            cout << "Masukkan Jadwal Pengembalian: ";
            getline(cin, jadwalPengembalian);
            tambahPustaka(id, judulPustaka, jadwalPengembalian);
        } else if (menu == 3) {
            string id;
            cout << "Masukkan ID Pemustaka yang akan dihapus: ";
            getline(cin, id);
            hapusPemustaka(id);
        } else if (menu == 4) {
            tampilkanDataPerpustakaan();
        } else if (menu != 5) {
            cout << "Pilihan tidak valid.\n";
        }
    } while (menu != 5);

    return 0;
}
```

**Output:**

```
Program Sistem Manajemen Buku Perpustakaan
Menu Peminjaman Perpustakaan:
1. Tambah Nama Pemustaka
2. Tambah Nama Pustaka
3. Hapus Nama Pemustaka
4. Tampilkan Data
5. Keluar
Menu : 1
Masukkan Nama Pemustaka: Osamu Dazai
Masukkan ID Pemustaka: 2311104059
Program Sistem Manajemen Buku Perpustakaan
Menu Peminjaman Perpustakaan:
1. Tambah Nama Pemustaka
2. Tambah Nama Pustaka
3. Hapus Nama Pemustaka
4. Tampilkan Data
```

```
5. Keluar
Menu : 2
Masukkan ID Pemustaka: 2311104059
Masukkan Judul Pustaka: No Longer Human
Masukkan Jadwal Pengembalian: 2 Februari 2025
Program Sistem Manajemen Buku Perpustakaan
Menu Peminjaman Perpustakaan:
1. Tambah Nama Pemustaka
2. Tambah Nama Pustaka
3. Hapus Nama Pemustaka
4. Tampilkan Data
5. Keluar
Menu : 1
Masukkan Nama Pemustaka: Friedrich Nietzsche
Masukkan ID Pemustaka: 2311104060
Program Sistem Manajemen Buku Perpustakaan
Menu Peminjaman Perpustakaan:
1. Tambah Nama Pemustaka
2. Tambah Nama Pustaka
3. Hapus Nama Pemustaka
4. Tampilkan Data
5. Keluar
Menu : 4
Pemustaka : Friedrich Nietzsche (ID : 2311104060)
Pemustaka : Osamu Dazai (ID : 2311104059)
  Pustaka : No Longer Human (Pengembalian : 2 Februari 2025)
Program Sistem Manajemen Buku Perpustakaan
Menu Peminjaman Perpustakaan:
1. Tambah Nama Pemustaka
2. Tambah Nama Pustaka
3. Hapus Nama Pemustaka
4. Tampilkan Data
5. Keluar
Menu : 3
Masukkan ID Pemustaka yang akan dihapus: 2311104060
Program Sistem Manajemen Buku Perpustakaan
Menu Peminjaman Perpustakaan:
1. Tambah Nama Pemustaka
2. Tambah Nama Pustaka
3. Hapus Nama Pemustaka
4. Tampilkan Data
5. Keluar
Menu : 4
Pemustaka : Osamu Dazai (ID : 2311104059)
  Pustaka : No Longer Human (Pengembalian : 2 Februari 2025)
Program Sistem Manajemen Buku Perpustakaan
Menu Peminjaman Perpustakaan:
1. Tambah Nama Pemustaka
2. Tambah Nama Pustaka
3. Hapus Nama Pemustaka
4. Tampilkan Data
5. Keluar
Menu : 5

Process returned 0 (0x0)   execution time : 136.539 s
Press any key to continue.
```

## 5. Kesimpulan

Multi List adalah struktur data yang terdiri dari sekumpulan daftar (list) yang saling terhubung, di mana setiap elemen dari daftar induk (list induk) dapat memiliki daftar turunan atau anak (list anak) yang berisi elemen terkait. Struktur ini memungkinkan representasi hierarki atau hubungan antar elemen, dengan elemen induk menunjuk ke daftar anaknya. List induk didefinisikan menggunakan elemen dengan informasi dasar dan pointer yang menunjuk ke

elemen lain dalam daftar induk atau ke daftar anak. List anak sendiri terdiri atas elemen-elemen yang terhubung dalam sebuah daftar berantai. Operasi dasar seperti menambahkan (insert) dan menghapus (delete) elemen pada list anak mensyaratkan pengetahuan mengenai elemen induknya, sementara operasi pada list induk melibatkan pengelolaan baik elemen induk maupun elemen anak yang terkait dengannya.

Struktur data Multi List mencakup beberapa fitur utama, seperti pemeriksaan apakah list kosong, pembuatan list kosong, alokasi memori, pencarian elemen, penambahan elemen (insert), dan penghapusan elemen (delete). Penambahan elemen bisa dilakukan di awal, tengah, atau akhir daftar, tergantung kebutuhan, baik untuk list induk maupun anak. Penghapusan elemen induk akan menghapus seluruh daftar anak yang terkait dengannya, sedangkan penghapusan elemen anak dilakukan hanya di dalam daftar anak tertentu. Selain itu, fungsi untuk mencetak isi dari list induk dan anak disediakan untuk menampilkan semua elemen dalam daftar beserta jumlah elemennya. Multi List sering digunakan dalam aplikasi yang memerlukan pengelolaan data dengan struktur hierarki, seperti hubungan antara pegawai dan tugasnya.