

**LAPORAN PRAKTIKUM  
STRUKTUR DATA**

**MODUL 11  
MULTI LINKED LIST**



**Disusun Oleh :**  
NAMA : FARIS WALID AWWAL AIDI  
NIM : 103112430133

**Dosen**  
FAHRUDIN MUKTI WIBOWO

**PROGRAM STUDI STRUKTUR DATA  
FAKULTAS INFORMATIKA  
TELKOM UNIVERSITY PURWOKERTO  
2025**

## A. Dasar Teori

Multi Linked List adalah kumpulan dari beberapa list yang berbeda namun memiliki hubungan satu sama lain. Dalam struktur data ini, setiap elemen dalam multi linked list dapat membentuk list-nya sendiri, dan biasanya terdapat konsep list induk (parent list) dan list anak (child list). Sebagai contoh implementasi yang diberikan, terdapat dua buah list utama: list pegawai (induk) dan list anak. Di sini, list pegawai bertindak sebagai list induk, dan setiap elemen dalam list pegawai dapat menunjuk ke satu list anak yang terpisah. Operasi dasar pada multi linked list meliputi penambahan (insert) dan penghapusan (delete). Dalam melakukan operasi pada list anak, seperti menambah atau menghapus elemen anak, posisi elemen induk harus diketahui terlebih dahulu. Sementara itu, operasi pada list induk, seperti penambahan, mirip dengan konsep linked list biasa (Singly, Doubly, atau Circular).

Salah satu karakteristik penting saat melakukan penghapusan pada Multi Linked List adalah perlakuan terhadap data terkait. Untuk menghapus elemen anak dari suatu elemen induk, posisi elemen induk tersebut harus ditentukan terlebih dahulu. Sebaliknya, jika elemen induk dihapus, seluruh elemen anak yang terhubung dengan elemen induk tersebut juga harus dihapus secara otomatis. Modul praktikum ini bertujuan untuk membantu pemahaman dan implementasi konsep Multi Linked List dalam berbagai studi kasus. Secara implementasi, struktur data ini didefinisikan dengan menggunakan struct untuk elemen list induk (struct elemen\_list\_induk) yang mencakup informasi dan penunjuk ke list anak, serta struct untuk elemen list anak (struct elemen\_list\_anak), di mana keduanya menggunakan representasi doubly linked list (memiliki penunjuk next dan prev).

B. Guided (berisi screenshot source code & output program disertai penjelasannya)

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

struct ChildNode {
    string info;
    ChildNode *next;
    ChildNode *prev;
};

struct ParentNode {
    string info;
    ChildNode *childHead;
    ParentNode *next;
    ParentNode *prev;
};

ParentNode *createParent(string info) {
    ParentNode *newNode = new ParentNode;
    newNode->info = info;
    newNode->childHead = NULL;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}

ChildNode *createChild(string info) {
    ChildNode *newNode = new ChildNode;
    newNode->info = info;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}

void insertParent(ParentNode *&head, string info) {
    ParentNode *newNode = createParent(info);
    if (head == NULL) {
        head = newNode;
    } else {
        ParentNode *temp = head;
```

```

while (temp->next != NULL) {
    temp = temp->next;
}
temp->next = newNode;
newNode->prev = temp;
}
}

void insertChild(ParentNode *head, string parentInfo, string childInfo) {
ParentNode *p = head;
while (p != NULL && p->info != parentInfo) {
    p = p->next;
}

if (p != NULL) {
    ChildNode *newChild = createChild(childInfo);
    if (p->childHead == NULL) {
        p->childHead = newChild;
    } else {
        ChildNode *c = p->childHead;
        while (c->next != NULL) {
            c = c->next;
        }
        c->next = newChild;
        newChild->prev = c;
    }
}
}

void printAll(ParentNode *head) {
while (head != NULL) {
    cout << head->info;
    ChildNode *c = head->childHead;
    while (c != NULL) {
        cout << " -> " << c->info;
        c = c->next;
    }
    cout << endl;
    head = head->next;
}
}

```

```

void updateParent(ParentNode *head, string oldInfo, string newInfo) {
    ParentNode *p = head;
    while (p != NULL) {
        if (p->info == oldInfo) {
            p->info = newInfo;
            return;
        }
        p = p->next;
    }
}

void updateChild(ParentNode *head, string parentInfo, string oldChildInfo, string newChildInfo) {
    ParentNode *p = head;
    while (p != NULL && p->info != parentInfo) {
        p = p->next;
    }

    if (p != NULL) {
        ChildNode *c = p->childHead;
        while (c != NULL) {
            if (c->info == oldChildInfo) {
                c->info = newChildInfo;
                return;
            }
            c = c->next;
        }
    }
}

void deleteChild(ParentNode *head, string parentInfo, string childInfo) {
    ParentNode *p = head;
    while (p != NULL && p->info != parentInfo) {
        p = p->next;
    }

    if (p != NULL) {
        ChildNode *c = p->childHead;
        while (c != NULL) {
            if (c->info == childInfo) {
                if (c == p->childHead) {
                    p->childHead = c->next;
                    if (p->childHead != NULL) {
                        p->childHead->prev = p;
                    }
                }
                else {
                    c->prev->next = c->next;
                    if (c->next != NULL) {
                        c->next->prev = c->prev;
                    }
                }
                delete(c);
            }
            c = c->next;
        }
    }
}

```

```

    p->childHead->prev = NULL;
}
} else {
    c->prev->next = c->next;
    if (c->next != NULL) {
        c->next->prev = c->prev;
    }
}
delete c;
return;
}
c = c->next;
}
}
}

void deleteParent(ParentNode *&head, string info) {
    ParentNode *p = head;
    while (p != NULL) {
        if (p->info == info) {
            ChildNode *c = p->childHead;
            while (c != NULL) {
                ChildNode *tempC = c;
                c = c->next;
                delete tempC;
            }
            if (p == head) {
                head = p->next;
                if (head != NULL) {
                    head->prev = NULL;
                }
            } else {
                p->prev->next = p->next;
                if (p->next != NULL) {
                    p->next->prev = p->prev;
                }
            }
            delete p;
            return;
        }
        p = p->next;
    }
}

```

```
    }

}

int main() {
    ParentNode *list = NULL;

    insertParent(list, "Parent A");
    insertParent(list, "Parent B");
    insertParent(list, "Parent C");

    cout << "\nSetelah InsertParent: " << endl;
    printAll(list);

    insertChild(list, "Parent A", "Child A1");
    insertChild(list, "Parent A", "Child A2");
    insertChild(list, "Parent B", "Child B1");

    cout << "\nSetelah InsertChild: " << endl;
    printAll(list);

    updateParent(list, "Parent B", "Parent B*");
    updateChild(list, "Parent A", "Child A1", "Child A1*");

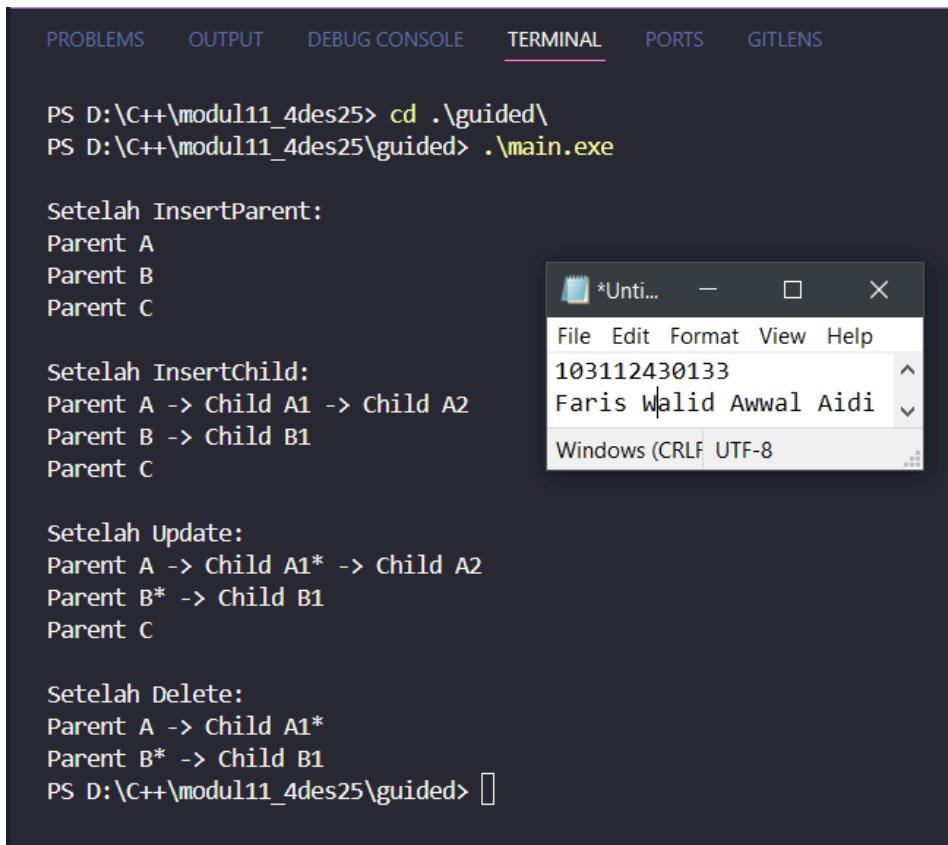
    cout << "\nSetelah Update: " << endl;
    printAll(list);

    deleteChild(list, "Parent A", "Child A2");
    deleteParent(list, "Parent C");

    cout << "\nSetelah Delete: " << endl;
    printAll(list);

    return 0;
}
```

## Screenshots Output



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

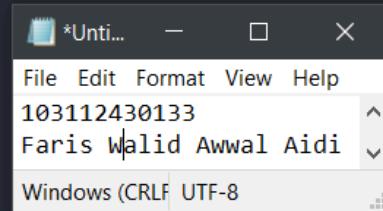
PS D:\C++\modul11_4des25> cd .\guided\
PS D:\C++\modul11_4des25\guided> .\main.exe

Setelah InsertParent:
Parent A
Parent B
Parent C

Setelah InsertChild:
Parent A -> Child A1 -> Child A2
Parent B -> Child B1
Parent C

Setelah update:
Parent A -> Child A1* -> Child A2
Parent B* -> Child B1
Parent C

Setelah Delete:
Parent A -> Child A1*
Parent B* -> Child B1
PS D:\C++\modul11_4des25\guided> 
```



## Deskripsi:

Program C++ ini mengimplementasikan konsep Multi Linked List, di mana terdapat list induk (ParentNode) dan list anak (ChildNode). List induk diimplementasikan sebagai Doubly Linked List (memiliki penunjuk next dan prev) yang menghubungkan elemen-elemen induk secara horizontal, seperti data Pegawai. Setiap elemen induk memiliki penunjuk ke head dari list anaknya (childHead), yang juga diimplementasikan sebagai Doubly Linked List secara vertikal, seperti data Anak. Program ini menyediakan fungsi-fungsi dasar untuk mengelola struktur data tersebut, meliputi insertParent (menambah elemen induk di akhir), insertChild (menambah elemen anak di akhir list anak pada induk tertentu), updateParent dan updateChild (mengubah info pada elemen induk/anak), deleteChild (menghapus elemen anak tertentu), dan deleteParent (menghapus elemen induk beserta semua elemen anaknya). Secara keseluruhan, program ini mendemonstrasikan bagaimana dua list yang berbeda dihubungkan untuk membentuk Multi Linked List.

C. Unguided/Tugas (berisi screenshot source code & output program disertai penjelasannya)

1. Multilist

a. multilist.h

```
#ifndef MULTILIST_H_INCLUDED
#define MULTILIST_H_INCLUDED

#include <stddef.h>
#define Nil NULL

typedef int infotypeanak;
typedef int infotypeinduk;
typedef int boolean;
#define true 1
#define false 0

struct elemen_list_anak;
struct elemen_list_induk;

typedef struct elemen_list_induk *address;
typedef struct elemen_list_anak *address_anak;

struct elemen_list_anak {
    infotypeanak info;
    address_anak next;
    address_anak prev;
};

struct listanak {
    address_anak first;
    address_anak last;
};

struct elemen_list_induk {
    infotypeinduk info;
    listanak lanak;
    address next;
    address prev;
};

struct listinduk {
```

```

address first;
address last;
};

boolean ListEmpty (listinduk L);
boolean ListEmptyAnak (listanak L);

void CreateList (listinduk &L);
void CreateListAnak (listanak &L);

address alokasi (infotypeinduk X);
address_anak alokasiAnak (infotypeanak X);
void dealokasi (address P);
void dealokasiAnak (address_anak P);

address findElm (listinduk L, infotypeinduk X);
address_anak findElmAnak (listanak Lanak, infotypeanak X);
boolean fFindElm (listinduk L, address P);
boolean fFindElmAnak (listanak Lanak, address_anak P);
address findBefore (listinduk L, address P);
address_anak findBeforeAnak (listanak Lanak, infotypeinduk X, address_anak P);

void insertFirst (listinduk &L, address P);
void insertAfter (listinduk &L, address P, address Prec);
void insertLast (listinduk &L, address P);
void insertFirstAnak (listanak &L, address_anak P);
void insertAfterAnak (listanak &L, address_anak P, address_anak Prec);
void insertLastAnak (listanak &L, address_anak P);

void delFirst (listinduk &L, address &P);
void delLast (listinduk &L, address &P);
void delAfter (listinduk &L, address &P, address Prec);
void delP (listinduk &L, infotypeinduk X);

void delFirstAnak (listanak &L, address_anak &P);
void delLastAnak (listanak &L, address_anak &P);
void delAfterAnak (listanak &L, address_anak &P, address_anak Prec);
void delPAnak (listanak &L, infotypeanak X);

void printInfo (listinduk L);
void printInfoAnak (listanak Lanak);
int nbList (listinduk L);

```

```

int nbListAnak (listanak Lanak);

void delAll (listinduk &L);

#endif

```

b. multilist.cpp

```

#include "multilist.h"
#include <iostream>

using namespace std;

boolean ListEmpty (listinduk L) {
    return L.first == Nil;
}

boolean ListEmptyAnak (listanak L) {
    return L.first == Nil;
}

void CreateList (listinduk &L) {
    L.first = Nil;
    L.last = Nil;
}

void CreateListAnak (listanak &L) {
    L.first = Nil;
    L.last = Nil;
}

address alokasi (infotypeinduk X) {
    address P = new elemen_list_induk;
    if (P != Nil) {
        P->info = X;
        CreateListAnak(P->lanak);
        P->next = Nil;
        P->prev = Nil;
    }
    return P;
}

```

```

address_anak alokasiAnak (infotypeanak X) {
    address_anak P = new elemen_list_anak;
    if (P != Nil) {
        P->info = X;
        P->next = Nil;
        P->prev = Nil;
    }
    return P;
}

void dealokasi (address P) {
    delete P;
}

void dealokasiAnak (address_anak P) {
    delete P;
}

address findElm (listinduk L, infotypeinduk X) {
    address P = L.first;
    while (P != Nil) {
        if (P->info == X) {
            return P;
        }
        P = P->next;
    }
    return Nil;
}

address_anak findElmAnak (listanak Lanak, infotypeanak X) {
    address_anak P = Lanak.first;
    while (P != Nil) {
        if (P->info == X) {
            return P;
        }
        P = P->next;
    }
    return Nil;
}

boolean fFindElm (listinduk L, address P) {
    address current = L.first;

```

```

while (current != Nil) {
    if (current == P) return true;
    current = current->next;
}
return false;
}

boolean fFindElmAnak (listanak Lanak, address_anak P) {
    address_anak current = Lanak.first;
    while (current != Nil) {
        if (current == P) return true;
        current = current->next;
    }
    return false;
}

address findBefore (listinduk L, address P) {
    if (P == L.first || P == Nil) return Nil;
    return P->prev;
}

address_anak findBeforeAnak (listanak Lanak, infotypeinduk X, address_anak P) {
    if (P == Lanak.first || P == Nil) return Nil;
    return P->prev;
}

void insertFirst (listinduk &L, address P) {
    P->next = L.first;
    if (!ListEmpty(L)) {
        L.first->prev = P;
    } else {
        L.last = P;
    }
    L.first = P;
    P->prev = Nil;
}

void insertAfter (listinduk &L, address P, address Prec) {
    if (Prec == L.last) {
        insertLast(L, P);
        return;
    }
}

```

```

 $P->next = Prec->next;$ 
 $P->prev = Prec;$ 
 $Prec->next->prev = P;$ 
 $Prec->next = P;$ 
}

void insertLast (listinduk &L, address P) {
    if (ListEmpty(L)) {
        insertFirst(L, P);
    } else {
        L.last->next = P;
        P->prev = L.last;
        L.last = P;
        P->next = Nil;
    }
}

void insertFirstAnak (listanak &L, address_anak P) {
    P->next = L.first;
    if (!ListEmptyAnak(L)) {
        L.first->prev = P;
    } else {
        L.last = P;
    }
    L.first = P;
    P->prev = Nil;
}

void insertAfterAnak (listanak &L, address_anak P, address_anak Prec) {
    if (Prec == L.last) {
        insertLastAnak(L, P);
        return;
    }
    P->next = Prec->next;
    P->prev = Prec;
    Prec->next->prev = P;
    Prec->next = P;
}

void insertLastAnak (listanak &L, address_anak P) {
    if (ListEmptyAnak(L)) {
        insertFirstAnak(L, P);
    }
}

```

```

} else {
    L.last->next = P;
    P->prev = L.last;
    L.last = P;
    P->next = Nil;
}
}

void delFirst (listinduk &L, address &P) {
    P = L.first;
    if (!ListEmpty(L)) {
        if (L.first == L.last) {
            CreateList(L);
        } else {
            L.first = P->next;
            L.first->prev = Nil;
            P->next = Nil;
            P->prev = Nil;
        }
    }
}

void delLast (listinduk &L, address &P) {
    P = L.last;
    if (!ListEmpty(L)) {
        if (L.first == L.last) {
            CreateList(L);
        } else {
            L.last = P->prev;
            L.last->next = Nil;
            P->prev = Nil;
            P->next = Nil;
        }
    }
}

void delAfter (listinduk &L, address &P, address Prec) {
    if (Prec != Nil && Prec->next != Nil) {
        P = Prec->next;
        if (P == L.last) {
            delLast(L, P);
        } else {

```

```

Prec->next = P->next;
P->next->prev = Prec;
P->next = Nil;
P->prev = Nil;
}
} else {
    P = Nil;
}
}

void delP (listinduk &L, infotypeinduk X) {
    address P = findElm(L, X);
    if (P != Nil) {
        address_anak c = P->lanak.first;
        while (c != Nil) {
            address_anak tempC = c;
            c = c->next;
            dealokasiAnak(tempC);
        }

        if (P == L.first) {
            delFirst(L, P);
        } else if (P == L.last) {
            delLast(L, P);
        } else {
            P->prev->next = P->next;
            P->next->prev = P->prev;
            P->next = Nil;
            P->prev = Nil;
        }
        dealokasi(P);
    }
}

void delFirstAnak (listanak &L, address_anak &P) {
    P = L.first;
    if (!ListEmptyAnak(L)) {
        if (L.first == L.last) {
            CreateListAnak(L);
        } else {
            L.first = P->next;
            L.first->prev = Nil;
        }
    }
}

```

```

    P->next = Nil;
    P->prev = Nil;
}
}

void delLastAnak (listanak &L, address_anak &P) {
    P = L.last;
    if (!ListEmptyAnak(L)) {
        if (L.first == L.last) {
            CreateListAnak(L);
        } else {
            L.last = P->prev;
            L.last->next = Nil;
            P->prev = Nil;
            P->next = Nil;
        }
    }
}

void delAfterAnak (listanak &L, address_anak &P, address_anak Prec) {
    if (Prec != Nil && Prec->next != Nil) {
        P = Prec->next;
        if (P == L.last) {
            delLastAnak(L, P);
        } else {
            Prec->next = P->next;
            P->next->prev = Prec;
            P->next = Nil;
            P->prev = Nil;
        }
    } else {
        P = Nil;
    }
}

void delPAnak (listanak &L, infotypeanak X) {
    address_anak P = findElmAnak(L, X);
    if (P != Nil) {
        if (P == L.first) {
            address_anak temp;
            delFirstAnak(L, temp);
        }
    }
}

```

```

    dealokasiAnak(temp);
} else if (P == L.last) {
    address_anak temp;
    delLastAnak(L, temp);
    dealokasiAnak(temp);
} else {
    P->prev->next = P->next;
    if (P->next != Nil) {
        P->next->prev = P->prev;
    }
    dealokasiAnak(P);
}
}

void printInfoAnak (listanak Lanak) {
    address_anak P = Lanak.first;
    if (ListEmptyAnak(Lanak)) {
        cout << "Kosong";
        return;
    }
    while (P != Nil) {
        cout << P->info;
        if (P->next != Nil) {
            cout << ", ";
        }
        P = P->next;
    }
}

void printInfo (listinduk L) {
    address P = L.first;
    if (ListEmpty(L)) {
        cout << "List Induk Kosong." << endl;
        return;
    }
    cout << "--- List Pegawai (Induk) ---" << endl;
    while (P != Nil) {
        cout << "Pegawai " << P->info << ":" ;
        printInfoAnak(P->lanak);
        cout << endl;
        P = P->next;
    }
}

```

```

}

cout << "-----" << endl;
}

int nbList (listinduk L) {
    int count = 0;
    address P = L.first;
    while (P != Nil) {
        count++;
        P = P->next;
    }
    return count;
}

int nbListAnak (listanak Lanak) {
    int count = 0;
    address_anak P = Lanak.first;
    while (P != Nil) {
        count++;
        P = P->next;
    }
    return count;
}

void delAll (listinduk &L) {
    address P;
    while (!ListEmpty(L)) {
        delFirst(L, P);

        address_anak c = P->lanak.first;
        while (c != Nil) {
            address_anak tempC = c;
            c = c->next;
            dealokasiAnak(tempC);
        }
        dealokasi(P);
    }
    L.last = Nil;
}

```

c. main.cpp

```
#include "multilist.h"
#include <iostream>

using namespace std;

int main() {
    listinduk PegawaiList;
    CreateList(PegawaiList);

    cout << "### Pengujian Multi Linked List Pegawai-Anak ###" << endl;

    address p1 = alokasi(10);
    address p2 = alokasi(20);
    address p3 = alokasi(30);

    insertFirst(PegawaiList, p1);
    insertLast(PegawaiList, p3);
    insertAfter(PegawaiList, p2, p1);

    cout << "\n[Step 1: Insert Induk]" << endl;
    printInfo(PegawaiList);
    cout << "Jumlah Total Pegawai: " << nbList(PegawaiList) << endl;

    address_anak a1 = alokasiAnak(101);
    address_anak a2 = alokasiAnak(102);
    address_anak a3 = alokasiAnak(103);

    insertFirstAnak(p1->lanak, a1);
    insertAfterAnak(p1->lanak, a2, a1);
    insertLastAnak(p1->lanak, a3);

    address_anak c1 = alokasiAnak(301);
    insertLastAnak(p3->lanak, c1);

    cout << "\n[Step 2: Insert Anak]" << endl;
    printInfo(PegawaiList);
    cout << "Jumlah Anak Pegawai 10: " << nbListAnak(p1->lanak) << endl;

    cout << "\n[Step 3: Delete Anak 102 dari Pegawai 10]" << endl;
    delPAnak(p1->lanak, 102);
    printInfo(PegawaiList);
```

```
cout << "\n[Step 4: Delete Pegawai 20]" << endl;
delP(PegawaiList, 20);
printInfo(PegawaiList);

address foundP = findElm(PegawaiList, 30);
cout << "\n[Step 5: Pencarian]" << endl;
if (foundP != Nil) {
    cout << "Pegawai 30 ditemukan." << endl;
}
address_anak foundA = findElmAnak(foundP->lanak, 301);
if (foundA != Nil) {
    cout << "Anak 301 dari Pegawai 30 ditemukan." << endl;
}

cout << "\n[Step 6: Delete All]" << endl;
delAll(PegawaiList);
printInfo(PegawaiList);
cout << "Jumlah Total Pegawai: " << nbList(PegawaiList) << endl;

return 0;
}
```

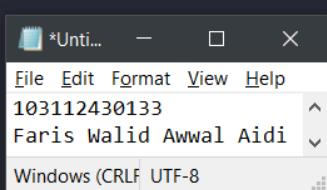
## Screenshot Output

```
Pegawai 30: Kosong
-----
Jumlah Total Pegawai: 3

[Step 2: Insert Anak]
--- List Pegawai (Induk) ---
Pegawai 10: 101, 102, 103
Pegawai 20: Kosong
Pegawai 30: 301
-----
Jumlah Anak Pegawai 10: 3

[Step 3: Delete Anak 102 dari Pegawai 10]
--- List Pegawai (Induk) ---
Pegawai 10: 101, 103
Pegawai 20: Kosong
Pegawai 30: 301
-----
[Step 4: Delete Pegawai 20]
--- List Pegawai (Induk) ---
Pegawai 10: 101, 103
Pegawai 30: 301
-----
[Step 5: Pencarian]
Pegawai 30 ditemukan.
Anak 301 dari Pegawai 30 ditemukan.

[Step 6: Delete All]
List Induk Kosong.
Jumlah Total Pegawai: 0
PS D:\C++\modul11_4des25\unguided> []
```



Deskripsi:

Program C++ di atas mengimplementasikan struktur data Multi Linked List menggunakan konsep Doubly Linked List untuk elemen induk (ParentNode/listinduk) dan elemen anak (ChildNode/listanak). Program ini menyediakan serangkaian fungsi dasar (ADT) untuk mengelola data, termasuk inisialisasi list, alokasi dan dealokasi memori, operasi pencarian (findElm), operasi penambahan (insertFirst, insertLast, insertAfter untuk induk maupun anak), operasi penghapusan (delFirst, delLast, delAfter, delP), dan fungsi pelengkap seperti menghitung jumlah elemen (nbList) serta menampilkan semua elemen (printInfo). Peran sentral program ini adalah fungsi penghapusan elemen induk (delP), yang memastikan seluruh elemen anak yang terhubung dengan induk tersebut juga dihapus dan memorinya dibebaskan (dealokasi) sebelum elemen induk itu sendiri dihapus, sesuai dengan prinsip Multi Linked List.

## 2. Circularlist

### a. circularlist.h

```
#ifndef CIRCULARLIST_H_INCLUDED
#define CIRCULARLIST_H_INCLUDED

#include <string>
#include <stddef.h>
#define Nil NULL

using namespace std;

typedef struct {
    string nama;
    string nim;
    char jenis_kelamin;
    float ipk;
} infotype;

typedef struct ElmList *address;

typedef struct ElmList {
    infotype info;
    address next;
    address prev;
} ElmList;

typedef struct {
    address First;
    address Last;
} List;

void CreateList(List &L);
address alokasi(infotype x);
void dealokasi(address P);
void insertFirst(List &L, address P);
void insertAfter(List &L, address Prec, address P);
void insertLast(List &L, address P);
void deleteFirst(List &L, address &P);
void deleteAfter(List &L, address Prec, address &P);
void deleteLast(List &L, address &P);
address findElm(List L, infotype x);
void printInfo(List L);
```

```

address createData (string nama, string nim, char jenis_kelamin, float ipk);

#endif

```

### b. circularlist.cpp

```

#include "circularlist.h"
#include <iostream>

void CreateList(List &L) {
    L.First = Nil;
    L.Last = Nil;
}

address alokasi(infotype x) {
    address P = new ElmList;
    if (P != Nil) {
        P->info = x;
        P->next = Nil;
        P->prev = Nil;
    }
    return P;
}

void dealokasi(address P) {
    delete P;
}

void insertFirst(List &L, address P) {
    P->next = L.First;
    if (L.First != Nil) {
        L.First->prev = P;
    } else {
        L.Last = P;
    }
    L.First = P;
}

void insertAfter(List &L, address Prec, address P) {
    if (Prec == Nil) {
        insertFirst(L, P);
    } else {
        P->prev = Prec;
        Prec->next = P;
    }
}

```

```

        return;
    }

    if (Prec == L.Last) {
        insertLast(L, P);
        return;
    }

    P->next = Prec->next;
    P->prev = Prec;
    Prec->next->prev = P;
    Prec->next = P;
}

void insertLast(List &L, address P) {
    if (L.First == Nil) {
        insertFirst(L, P);
    } else {
        L.Last->next = P;
        P->prev = L.Last;
        L.Last = P;
    }
}

void deleteFirst(List &L, address &P) {
    P = L.First;
    if (L.First != Nil) {
        if (L.First == L.Last) {
            CreateList(L);
        } else {
            L.First = P->next;
            L.First->prev = Nil;
            P->next = Nil;
        }
    }
}

void deleteAfter(List &L, address Prec, address &P) {
    if (Prec == Nil || Prec->next == Nil) {
        P = Nil;
        return;
    }
    P = Prec->next;
}

```

```

if (P == L.Last) {
    deleteLast(L, P);
    return;
}

Prec->next = P->next;
P->next->prev = Prec;
P->next = Nil;
P->prev = Nil;
}

void deleteLast(List &L, address &P) {
    P = L.Last;
    if (L.Last != Nil) {
        if (L.First == L.Last) {
            CreateList(L);
        } else {
            L.Last = P->prev;
            L.Last->next = Nil;
            P->prev = Nil;
        }
    }
}

address findElm(List L, infotype x) {
    address P = L.First;
    while (P != Nil) {
        if (P->info.nim == x.nim) {
            return P;
        }
        P = P->next;
    }
    return Nil;
}

void printInfo(List L) {
    address P = L.First;
    if (P == Nil) {
        cout << "List Mahasiswa Kosong." << endl;
        return;
    }
    cout << "--- Data Mahasiswa (Doubly Linked List) ---" << endl;
}

```

```

while (P != Nil) {
    cout << "Nama: " << P->info.nama
        << ", NIM: " << P->info.nim
        << ", Kelamin: " << P->info.jenis_kelamin
        << ", IPK: " << P->info.ipk << endl;
    P = P->next;
}
cout << "-----" << endl;
}

address createData (string nama, string nim, char jenis_kelamin, float ipk) {
    infotype x;
    x.nama = nama;
    x.nim = nim;
    x.jenis_kelamin = jenis_kelamin;
    x.ipk = ipk;
    address P = alokasi(x);
    return P;
}

```

### c. main.cpp

```

#include "circularlist.h"
#include <iostream>

using namespace std;

int main() {
    List L;
    address P1 = Nil;
    address P2 = Nil;
    address P_del = Nil;
    infotype x;

    CreateList(L);
    cout << "### Pengujian Doubly Linked List Mahasiswa ###" << endl;
    cout << "--- Coba Insert First, Last, dan After ---" << endl;

    P1 = createData("Danu", "04", 'L', 4.0);
    insertFirst(L, P1);

    P1 = createData("Fahmi", "06", 'L', 3.45);

```

```

insertLast(L, P1);

P1 = createData("Bobi", "02", 'L', 3.71);
x.nim = "04";
P2 = findElm(L, x);
if(P2 != Nil) {
    insertAfter(L, P2, P1);
}

P1 = createData("Ali", "01", 'L', 3.3);
insertFirst(L, P1);

P1 = createData("Gita", "07", 'P', 3.75);
insertLast(L, P1);
x.nim = "07";
P2 = findElm(L, x);
P1 = createData("Cindi", "03", 'P', 3.5);
if(P2 != Nil) {
    insertAfter(L, P2, P1);
}

x.nim = "02";
P2 = findElm(L, x);
P1 = createData("Hilmi", "08", 'L', 3.3);
if(P2 != Nil) {
    insertAfter(L, P2, P1);
}

x.nim = "04";
P2 = findElm(L, x);
P1 = createData("Eli", "05", 'P', 3.4);
if(P2 != Nil) {
    insertAfter(L, P2, P1);
}

cout << "\n[Setelah Semua Insert]" << endl;
printInfo(L);

deleteFirst(L, P_del);
if(P_del != Nil) dealokasi(P_del);
cout << "\n[Setelah Delete First (Ali)]" << endl;
printInfo(L);

```

```

    deleteLast(L, P_del);
    if (P_del != Nil) dealokasi(P_del);
    cout << "\n[Setelah Delete Last (Cindi)]" << endl;
    printInfo(L);

    x.nim = "04";
    P2 = findElm(L, x);
    if (P2 != Nil) {
        deleteAfter(L, P2, P_del);
        if (P_del != Nil) dealokasi(P_del);
    }
    cout << "\n[Setelah Delete After (Eli)]" << endl;
    printInfo(L);

    return 0;
}

```

## Screenshot Output

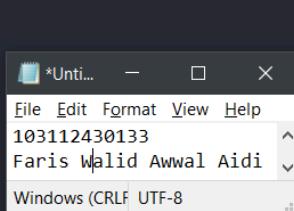
```

### Pengujian Doubly Linked List Mahasiswa ###
--- Coba Insert First, Last, dan After ---

[Setelah Semua Insert]
--- Data Mahasiswa (Doubly Linked List) ---
Nama: Ali, NIM: 01, Kelamin: L, IPK: 3.3
Nama: Danu, NIM: 04, Kelamin: L, IPK: 4
Nama: Eli, NIM: 05, Kelamin: P, IPK: 3.4
Nama: Bobi, NIM: 02, Kelamin: L, IPK: 3.71
Nama: Hilmi, NIM: 08, Kelamin: L, IPK: 3.3
Nama: Fahmi, NIM: 06, Kelamin: L, IPK: 3.45
Nama: Gita, NIM: 07, Kelamin: P, IPK: 3.75
Nama: Cindi, NIM: 03, Kelamin: P, IPK: 3.5
-----

[Setelah Delete First (Ali)]
--- Data Mahasiswa (Doubly Linked List) ---
Nama: Danu, NIM: 04, Kelamin: L, IPK: 4
Nama: Eli, NIM: 05, Kelamin: P, IPK: 3.4
Nama: Bobi, NIM: 02, Kelamin: L, IPK: 3.71
Nama: Hilmi, NIM: 08, Kelamin: L, IPK: 3.3
Nama: Fahmi, NIM: 06, Kelamin: L, IPK: 3.45
Nama: Gita, NIM: 07, Kelamin: P, IPK: 3.75
Nama: Cindi, NIM: 03, Kelamin: P, IPK: 3.5
-----

[Setelah Delete Last (Cindi)]
--- Data Mahasiswa (Doubly Linked List) ---
Nama: Danu, NIM: 04, Kelamin: L, IPK: 4
Nama: Eli, NIM: 05, Kelamin: P, IPK: 3.4
Nama: Bobi, NIM: 02, Kelamin: L, IPK: 3.71
Nama: Hilmi, NIM: 08, Kelamin: L, IPK: 3.3
Nama: Fahmi, NIM: 06, Kelamin: L, IPK: 3.45
Nama: Gita, NIM: 07, Kelamin: P, IPK: 3.75
-----
```



Nama	NIM	Kelamin	IPK
Ali	01	L	3.3
Danu	04	L	4
Eli	05	P	3.4
Bobi	02	L	3.71
Hilmi	08	L	3.3
Fahmi	06	L	3.45
Gita	07	P	3.75
Cindi	03	P	3.5

```
[Setelah Delete After (Eli)]
--- Data Mahasiswa (Doubly Linked List) ---
Nama: Danu, NIM: 04, Kelamin: L, IPK: 4
Nama: Bobi, NIM: 02, Kelamin: L, IPK: 3.71
Nama: Hilmi, NIM: 08, Kelamin: L, IPK: 3.3
Nama: Fahmi, NIM: 06, Kelamin: L, IPK: 3.45
Nama: Gita, NIM: 07, Kelamin: P, IPK: 3.75
-----
PS D:\C++\modul11_4des25\unguided\circularlist> █
```

Deskripsi:

Program C++ di atas mengimplementasikan Doubly Linked List (Daftar Berantai Ganda) untuk mengelola data mahasiswa (Nama, NIM, Jenis Kelamin, dan IPK), di mana setiap elemen memiliki penunjuk ke elemen setelahnya (next) dan elemen sebelumnya (prev). Implementasi ini mencakup 11 fungsi dasar (ADT) yang diperlukan untuk manipulasi list, seperti: membuat list kosong (CreateList), mengalokasikan memori untuk data baru (alokasi dan createData), menyisipkan elemen di awal, tengah, dan akhir list (insertFirst, insertAfter, insertLast), menghapus elemen di awal, tengah, dan akhir list (deleteFirst, deleteAfter, deleteLast), mencari elemen berdasarkan NIM (findElm), serta menampilkan seluruh isi list (printInfo). Secara keseluruhan, program ini berfungsi sebagai model lengkap untuk mengelola koleksi data mahasiswa secara dinamis dan terurut menggunakan struktur Doubly Linked List.

#### D. Kesimpulan

Kesimpulan dari kedua program yang telah dibuat, implementasi Multi Linked List menunjukkan bahwa materi ini berfokus pada variasi struktur data berantai yang kompleks. Program pertama menggambarkan Multi Linked List dengan konsep list induk dan list anak yang terhubung, di mana penghapusan elemen induk harus diikuti dengan penghapusan seluruh elemen anak yang terikat, sesuai dengan prinsip data hierarkis. Sementara itu, program kedua, meskipun merupakan implementasi Doubly Linked List murni (setiap elemen memiliki penunjuk ke depan dan belakang), ia berfungsi sebagai fondasi penting yang digunakan dalam membangun elemen-elemen list pada program pertama (seperti pada list induk dan anak). Kedua program ini secara kolektif menguji pemahaman terhadap manajemen memori dinamis (alokasi, dealokasi) dan manipulasi pointer untuk operasi dasar list di berbagai tingkat kompleksitas.

#### E. Referensi

- Raharjo, Budi. 2025. *Buku Pemrograman C++ Mudah dan Cepat Menjadi Master C*.  
*Wikipedia contributors. (2024, 8 Mei). C++. Wikipedia, Ensiklopedia Bebas. Diakses pada 2 Desember 2025, dari <https://id.wikipedia.org/wiki/C%2B%2B>*