

**LAPORAN PRAKTIKUM
STRUKTUR DATA**

**MODUL XI
MULTI LINKED LIST**



Disusun Oleh :

Nama: Dina Nadhyfa
NIM : 103112430052

Dosen

Fahrudin Mukti Wibowo

**PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025**

A. Dasar Teori

Multi Linked List merupakan pengembangan lanjutan dari struktur data Linked List biasa, di mana setiap node memiliki lebih dari satu pointer untuk menghubungkan ke node lain, seperti pointer ke parent dan child, sehingga membentuk hierarki atau multiple rantai akses. Berbeda dengan single Linked List yang hanya memiliki satu pointer ke node berikutnya, Multi Linked List memungkinkan navigasi multidimensi, misalnya satu pointer menunjuk ke node sibling dan yang lain ke node anak, yang sangat berguna untuk representasi data kompleks seperti file directory atau graf pohon. Menurut GeeksforGeeks, struktur ini meningkatkan efisiensi akses data non-linear dengan mengurangi traversal panjang, meskipun menambah overhead memori karena multiple pointer per node.

Dalam konteks pemrograman C/C++, implementasi Multi Linked List biasanya didefinisikan dengan struct node yang berisi data, pointer next (untuk sibling), dan pointer down (untuk child), memfasilitasi operasi insert, delete, dan traverse secara rekursif. Keunggulan utamanya adalah fleksibilitas dalam menyimpan relasi banyak-ke-banyak antar data, sebagaimana dijelaskan dalam modul akademik, di mana satu node parent dapat memiliki multiple child list independen. GeeksforGeeks menekankan bahwa varian ini ideal untuk aplikasi seperti sparse matrix atau inverted index pencarian, dengan kompleksitas waktu O(1) untuk akses langsung via pointer spesifik.

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

Guided 1

```
#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 = 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

```

PS D:\COOLYEAH\SEMESTER 3\Strukdat\Modul 12> cd "d:\COOLYEAH\SEMESTER 3\Strukdat\Modul 12\Guided\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { ./main }

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:\COOLYEAH\SEMESTER 3\Strukdat\Modul 12\Guided>

```

Deskripsi:

Program ini mengimplementasikan struktur data Multi-Linked List dengan relasi parent-child menggunakan C++. Struktur terdiri dari dua jenis node: ParentNode (linked list utama) dan ChildNode (linked list untuk setiap parent). Keduanya menggunakan doubly

linked list dengan pointer next dan prev. Program menyediakan operasi CRUD lengkap: insert parent/child, update info parent/child, delete child/parent, dan print seluruh struktur. Saat parent dihapus, semua child-nya juga otomatis dihapus untuk menghindari memory leak. Fungsi main() mendemonstrasikan semua operasi dengan membuat 3 parent (A, B, C), menambah beberapa child, melakukan update, dan menghapus data. Output ditampilkan dalam format "Parent -> Child1 -> Child2".

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

Unguided 1

Multilist.h

```
#ifndef MULTILIST_H_INCLUDED
#define MULTILIST_H_INCLUDED
#define Nil NULL

typedef int infotypeanak;
typedef int infotypeinduk;
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;
};

// pengecekan apakah list kosong
bool ListEmpty(listinduk L);
bool ListEmptyAnak(listanak L);

// pembuatan list kosong
```

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

// manajemen memori
address alokasi(infotypeinduk P);
address_anak alokasiAnak(infotypeanak P);
void dealokasi(address P);
void dealokasiAnak(address_anak P);

// pencarian sebuah elemen list
address findElm(listinduk L, infotypeinduk X);
address_anak findElmAnak(listanak Lanak, infotypeanak X);
bool fFindElm(listinduk L, address P);
bool fFindElmanak(listanak Lanak, address_anak P);
address findBefore(listinduk L, address P);
address_anak findBeforeAnak(listanak Lanak, address_anak P);

// penambahan elemen - INDUK
void insertFirst(listinduk &L, address P);
void insertAfter(listinduk &L, address P, address Prec);
void insertLast(listinduk &L, address P);

// penambahan elemen - ANAK
void insertFirstAnak(listanak &L, address_anak P);
void insertAfterAnak(listanak &L, address_anak P, address_anak Prec);
void insertLastAnak(listanak &L, address_anak P);

// penghapusan sebuah elemen - INDUK
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);

// penghapusan sebuah elemen - ANAK
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);

// proses semua elemen list
void printInfo(listinduk L);
int nbList(listinduk L);
void printInfoAnak(listanak Lanak);
int nbListAnak(listanak Lanak);

// proses terhadap list
void delAll(listinduk &L);
void delAllAnak(listanak &L);
```

```
| #endif |
```

Multilist.cpp

```
#include "multilist.h"
#include <iostream>
using namespace std;

// pengecekan apakah list kosong
bool ListEmpty(listinduk L) {
    return L.first == Nil;
}

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

// pembuatan list kosong
void CreateList(listinduk &L) {
    L.first = Nil;
    L.last = Nil;
}

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

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

// pencarian sebuah elemen list
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;
}

bool fFindElm(listinduk L, address P) {
    address Q = L.first;
    while (Q != Nil) {
        if (Q == P) {
            return true;
        }
        Q = Q->next;
    }
    return false;
}

bool fFindElmanak(listanak Lanak, address_anak P) {
    address_anak Q = Lanak.first;
    while (Q != Nil) {
        if (Q == P) {
            return true;
        }
        Q = Q->next;
    }
}
```

```

        }
        return false;
    }

address findBefore(listinduk L, address P) {
    address Q = L.first;
    if (Q == P) {
        return Nil;
    }
    while (Q != Nil && Q->next != P) {
        Q = Q->next;
    }
    return Q;
}

address_anak findBeforeAnak(listanak Lanak, address_anak P) {
    address_anak Q = Lanak.first;
    if (Q == P) {
        return Nil;
    }
    while (Q != Nil && Q->next != P) {
        Q = Q->next;
    }
    return Q;
}

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

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

```

```

        }
        Prec->next = P;
    }
}

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

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

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

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

```

```

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

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

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

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

```

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

void delP(listinduk &L, infotypeinduk X) {
    address P = findElm(L, X);
    if (P != Nil) {

        delAllAnak(P->lanak);

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

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

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

```

        }
    }

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

void delPanak(listanak &L, infotypeanak X) {
    address_anak P = findElmAnak(L, X);
    if (P != Nil) {
        if (P == L.first) {
            delFirstAnak(L, P);
        } else if (P == L.last) {
            delLastAnak(L, P);
        } else {
            address_anak Prec = P->prev;
            delAfterAnak(L, P, Prec);
        }
        dealokasiAnak(P);
    }
}

// proses semua elemen list
void printInfo(listinduk L) {
    address P = L.first;
    cout << "==== Data Induk dan Anak ===" << endl;
    while (P != Nil) {
        cout << "Induk: " << P->info << endl;
        cout << " Anak: ";
        printInfoAnak(P->lanak);
        cout << endl;
        P = P->next;
    }
}

void printInfoAnak(listanak Lanak) {
    address_anak P = Lanak.first;
    if (ListEmptyAnak(Lanak)) {

```

```
        cout << "(kosong)";
    } else {
        while (P != Nil) {
            cout << P->info;
            if (P->next != Nil) {
                cout << ", ";
            }
            P = P->next;
        }
    }
}

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;
}

// proses terhadap list
void delAll(listinduk &L) {
    address P;
    while (!ListEmpty(L)) {
        delFirst(L, P);
        // Hapus semua anak
        delAllAnak(P->lanak);
        dealokasi(P);
    }
}

void delAllAnak(listanak &L) {
    address_anak PA;
    while (!ListEmptyAnak(L)) {
        delFirstAnak(L, PA);
        dealokasiAnak(PA);
    }
}
```

Main.cpp

```
#include "multilist.h"
#include <iostream>
using namespace std;

int main() {
    listinduk L;
    address P_induk;
    address_anak P_anak;

    CreateList(L);

    cout << "Menambah data Parent" << endl;
    P_induk = alokasi(1);
    insertFirst(L, P_induk);

    P_induk = alokasi(2);
    insertLast(L, P_induk);

    P_induk = alokasi(3);
    insertLast(L, P_induk);

    // Menambah anak untuk parent 1
    cout << "Menambah anak untuk parent 1" << endl;
    P_induk = findElm(L, 1);
    if (P_induk != Nil) {
        P_anak = alokasiAnak(101);
        insertFirstAnak(P_induk->lanak, P_anak);

        P_anak = alokasiAnak(102);
        insertLastAnak(P_induk->lanak, P_anak);

        P_anak = alokasiAnak(103);
        insertLastAnak(P_induk->lanak, P_anak);
    }

    // Menambah anak untuk parent 2
    cout << "Menambah anak untuk parent 2" << endl;
    P_induk = findElm(L, 2);
    if (P_induk != Nil) {
        P_anak = alokasiAnak(201);
        insertFirstAnak(P_induk->lanak, P_anak);

        P_anak = alokasiAnak(202);
        insertLastAnak(P_induk->lanak, P_anak);
    }
}
```

```
// Menambah anak untuk parent 3
cout << "Menambah anak untuk parent 3" << endl;
P_induk = findElm(L, 3);
if (P_induk != Nil) {
    P_anak = alokasiAnak(301);
    insertFirstAnak(P_induk->lanak, P_anak);
}

// Menampilkan semua data
cout << endl;
printInfo(L);

// Testing delete anak
cout << endl << "Menghapus anak 102 dari parent 1..." << endl;
P_induk = findElm(L, 1);
if (P_induk != Nil) {
    delPAnak(P_induk->lanak, 102);
}

printInfo(L);

// Testing delete induk (akan menghapus semua anaknya juga)
cout << endl << "Menghapus parent 2 beserta semua anaknya..." <<
endl;
delP(L, 2);

printInfo(L);

// Informasi jumlah
cout << endl << "Jumlah parent: " << nbList(L) << endl;

// Cleanup
delAll(L);

return 0;
}
```

Screenshoot Output

```
PS D:\COOLYEAH\SEMESTER 3\Strukdat\Modul 12\Unguided\Soal 1> .\main
Menambah data Parent
Menambah anak untuk parent 1
Menambah anak untuk parent 2
Menambah anak untuk parent 3

==== Data Induk dan Anak ====
Induk: 1
    Anak: 101, 102, 103
Induk: 2
    Anak: 201, 202
Induk: 3
    Anak: 301

Menghapus anak 102 dari parent 1...
==== Data Induk dan Anak ====
Induk: 1
    Anak: 101, 103
Induk: 2
    Anak: 201, 202
Induk: 3
    Anak: 301

Menghapus parent 2 beserta semua anaknya...
==== Data Induk dan Anak ====
Induk: 1
    Anak: 101, 103
Induk: 3
    Anak: 301

Jumlah parent: 2
PS D:\COOLYEAH\SEMESTER 3\Strukdat\Modul 12\Unguided\Soal 1>
```

Deskripsi:

Program diatas mengimplementasikan MLL dengan struktur Induk-Anak. Terdapat dua tipe list: listinduk (list utama) dan listanak (list child di setiap induk), keduanya memiliki pointer first dan last untuk akses dari kedua arah. Struktur data menggunakan typedef untuk pointer (address & address_anak) dan infotype integer untuk data induk maupun anak. Program menyediakan operasi lengkap: insert (First/After/Last), delete (First/Last/After/byValue), search (findElm, fFindElm, findBefore), dan utility (print, count, delAll).

Fitur khusus: saat menghapus induk, semua anak otomatis terhapus untuk mencegah memory leak (cascade delete). Fungsi main() mendemonstrasikan pembuatan 3 parent (1,2,3) dengan beberapa child (101-103, 201-202, 301), lalu testing operasi delete child dan delete parent. Manajemen memori dilakukan secara manual dengan fungsi alokasi/dealokasi yang menggunakan operator new dan delete.

Unguided 2

Circularlist.h

```
#ifndef CIRCULARLIST_H_INCLUDED
#define CIRCULARLIST_H_INCLUDED
#define Nil NULL
#include <string>
using namespace std;

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

typedef struct ElmList *address;

struct ElmList {
    infotype info;
    address next;
};

struct List {
    address First;
};

// Fungsi dan prosedur
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);

#endif
```

Circularlist.cpp

```
#include "circularlist.h"
#include <iostream>
using namespace std;

void CreateList(List &L) {
```

```
L.First = Nil;
}

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

void dealokasi(address &P) {
    delete P;
    P = Nil;
}

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

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

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

```

        P->next = L.First;
        Last->next = P;
    }
}

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

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

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

address findElm(List L, infotype x) {
    if (L.First == Nil) {

```

```

        return Nil;
    }

    address P = L.First;
    do {
        if (P->info.nim == x.nim) {
            return P;
        }
        P = P->next;
    } while (P != L.First);

    return Nil;
}

void printInfo(List L) {
    if (L.First == Nil) {
        cout << "List kosong" << endl;
        return;
    }

    cout << "coba insert first, last, dan after" << endl;
    address P = L.First;
    do {
        cout << "Nama : " << P->info.nama << endl;
        cout << "NIM : " << P->info.nim << endl;
        cout << "JK : " << P->info.jenis_kelamin << endl;
        cout << "IPK : " << P->info.ipk << endl;
        cout << endl;
        P = P->next;
    } while (P != L.First);
}

```

Main.cpp

```

#include "circularlist.h"
#include <iostream>
using namespace std;

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

```

```
int main() {
    List L, A, B, L2;
    address P1 = Nil;
    address P2 = Nil;
    infotype x;

    CreateList(L);

    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);
    insertFirst(L, P1);

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

    P1 = createData("Gita", "07", 'p', 3.75);
    insertLast(L, P1);

    x.nim = "07";
    P1 = findElm(L, x);
    P2 = createData("Cindi", "03", 'p', 3.5);
    insertAfter(L, P1, P2);

    x.nim = "02";
    P1 = findElm(L, x);
    P2 = createData("Hilmi", "08", 'l', 3.3);
    insertAfter(L, P1, P2);

    x.nim = "04";
    P1 = findElm(L, x);
    P2 = createData("Eli", "05", 'p', 3.4);
    insertAfter(L, P1, P2);

    printInfo(L);

    return 0;
}
```

Screenshot Output

```
PS D:\COOLYEAH\SEMESTER 3\Strukdat\Modul 12\Unguided\Soal 2> .\main

coba insert first, last, dan after
coba insert first, last, dan after
Nama : Ali
NIM : 01
JK : l
IPK : 3.3

Nama : Bobi
NIM : 02
JK : l
IPK : 3.71

Nama : Hilmi
NIM : 08
JK : l
IPK : 3.3

Nama : Danu
NIM : 04
JK : l
IPK : 4

Nama : Eli
NIM : 05
JK : p
IPK : 3.4

Nama : Fahmi
NIM : 06
JK : l
IPK : 3.45

Nama : Gita
NIM : 07
JK : p
IPK : 3.75

Nama : Cindi
NIM : 03
JK : p
IPK : 3.5

PS D:\COOLYEAH\SEMESTER 3\Strukdat\Modul 12\Unguided\Soal 2> █
```

Deskripsi:

Program ini mengimplementasikan Circular Singly Linked List untuk manajemen data mahasiswa dengan C++. Struktur data menggunakan infotype berupa struct yang menyimpan nama, NIM, jenis kelamin, dan IPK mahasiswa, dengan address sebagai pointer ke ElmList. Karakteristik circular list: node terakhir menunjuk kembali ke First (tidak ke NULL), sehingga membentuk lingkaran tanpa akhir.

Program menyediakan operasi CRUD lengkap: insertFirst, insertLast, insertAfter untuk penambahan, deleteFirst, deleteLast, deleteAfter untuk penghapusan, serta findElm untuk pencarian berdasarkan NIM. Fungsi printInfo menggunakan do-while loop untuk traversal circular list, dengan kondisi berhenti ketika pointer kembali ke First. Program main() mendemonstrasikan kombinasi ketiga jenis insert: pertama insert Ali-Bobi-Danu dengan insertFirst, lalu Fahmi-Gita dengan insertLast, kemudian Hilmi-Eli-Cindi menggunakan insertAfter setelah node tertentu yang dicari dengan findElm. Output menampilkan 8 data mahasiswa terurut sesuai kombinasi operasi insert yang dilakukan, membuktikan implementasi circular linked list berfungsi dengan baik.

D. Kesimpulan

Praktikum Modul XI berhasil mengimplementasikan Multi Linked List dan Circular Linked List menggunakan C++ dengan struktur data yang matang dan operasi CRUD lengkap. Pada Multi Linked List (Guided dan Unguided), program mendemonstrasikan hierarki parent-child berbasis doubly linked list, di mana setiap parent memiliki list anak independen dengan manajemen memori manual (alokasi/dealokasi), insert/delete first/last/after/byValue, pencarian elemen, dan cascade delete untuk mencegah memory leak saat menghapus parent beserta seluruh anaknya. Sementara Circular Linked List (Unguided 2) menerapkan singly linked list siklik untuk data mahasiswa (nama, NIM, JK, IPK), memanfaatkan pointer last->next kembali ke First untuk traversal tanpa akhir, dengan operasi insertFirst/Last/After dan printInfo menggunakan do-while loop yang efisien. Secara keseluruhan, kedua implementasi membuktikan fleksibilitas linked list varian untuk aplikasi hierarkis dan siklik, dengan penanganan edge case seperti list kosong dan navigasi bidirectional/circular yang optimal.

E. Rerferensi

<https://fikti.umsu.ac.id/pengertian-linked-list-struktur-data-dalam-pemrograman/>

<https://www.geeksforgeeks.org/dsa/introduction-to-multi-linked-list/#:~:text=A%20multi%2Dlinked%20list%20is,heap%20memory%20is%20not%20full.>

<https://www.geeksforgeeks.org/dsa/circular-linked-list/>

<https://repository.unikom.ac.id/38716/1/Bab%20VII%20-%20Circular%20Linked%20List.pdf>