LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL 11

MULTI LINKED LIST

Disusun Oleh :

NAMA : Muhammad Omar Nadiv

NIM : 103112430063

Dosen

Fahrudin Mukti Wibowo, S.Kom., M.Eng.

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Multi Linked List adalah struktur data berbasis linked list yang memiliki lebih dari satu pointer hubungan antar data. Jika pada single linked list setiap node hanya memiliki satu pointer menuju node berikutnya, pada multi linked list setiap node dapat memiliki dua atau lebih pointer yang menghubungkan node ke arah atau kategori berbeda. Struktur ini biasanya digunakan untuk merepresentasikan data yang memiliki hubungan ganda, berlapis, atau bertingkat, seperti data mahasiswa dalam kelas, kategori barang dalam toko, atau hubungan antara departemen dan karyawan.

Multi linked list memungkinkan penyimpanan data dalam bentuk hirarki atau kategori yang saling terhubung, misalnya list utama berisi kelompok besar, kemudian setiap kelompok memiliki sub-list sendiri. Dengan pendekatan ini, pencarian dan pengorganisasian data menjadi lebih fleksibel dibandingkan struktur linear biasa. Operasi dasar seperti insert, delete, dan traversal dapat dilakukan dengan mengikuti lebih dari satu jalur pointer sesuai kebutuhan. Multi linked list umum digunakan dalam sistem informasi yang membutuhkan relasi banyak-ke-banyak atau data yang dikelompokkan secara dinamis.

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;
       }
```

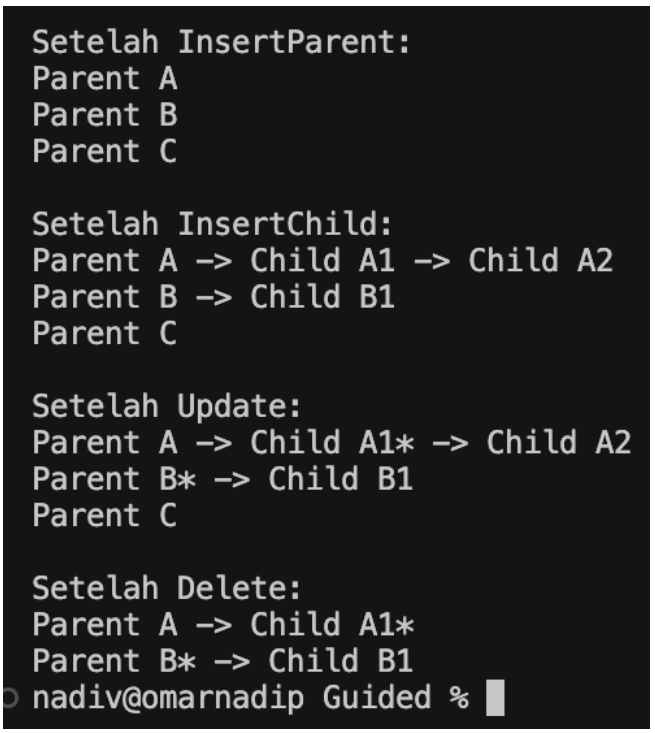
```
        cout << endl;
        head = head->next;
    }
}

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;
}d
```

Screenshots Output



Deskripsi:

Program ini mengimplementasikan struktur data Multi Linked List yang terdiri dari parent dan child, di mana setiap parent dapat memiliki daftar child sendiri. Baik parent maupun child disusun menggunakan doubly linked list, sehingga setiap node memiliki pointer ke node berikutnya dan sebelumnya. Struktur ini memungkinkan penyimpanan data bertingkat dalam bentuk hubungan satu-ke-banyak.

Program menyediakan operasi dasar seperti menambah parent dan child, memperbarui data pada parent atau child, serta menghapus child tertentu atau parent beserta seluruh child-nya. Selain itu, fungsi printAll digunakan untuk menampilkan seluruh parent beserta daftar anaknya. Melalui operasi-operasi tersebut, program memperlihatkan cara pengelolaan relasi bertingkat menggunakan multi linked list secara dinamis.

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

Unguided 1 (file main.cpp)

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

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

   cout << "=== INSERT DATA INDUK ===" << endl;
   insertLast(L, alokasi(10));
   insertLast(L, alokasi(20));
   insertLast(L, alokasi(30));
   printInfo(L);

   cout << "\n=== INSERT DATA ANAK ===" << endl;
   address P10 = findElm(L, 10);
   address P20 = findElm(L, 20);

   insertLastAnak(P10->lanak, alokasiAnak(101));
   insertLastAnak(P10->lanak, alokasiAnak(102));
   insertLastAnak(P20->lanak, alokasiAnak(201));

   printInfo(L);

   cout << "\n=== DELETE ANAK (102 dari induk 10) ===" << endl;
   delPAnak(P10->lanak, 102);
   printInfo(L);

   cout << "\n=== DELETE INDUK (30) ===" << endl;
   delP(L, 30);
   printInfo(L);

   cout << "\n=== JUMLAH DATA ===" << endl;
   cout << "Jumlah Induk: " << nbList(L) << endl;
   cout << "Jumlah Anak Induk 10: " << nbListAnak(P10->lanak) << endl;
   cout << "Jumlah Anak Induk 20: " << nbListAnak(P20->lanak) << endl;
```

```
    return 0;
}
```

(file multilist.cpp)

```
#include <iostream>
#include "multilist.h"
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;
    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;
    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 Q = L.first;
    while (Q != Nil) {
        if (Q == P) return true;
        Q = Q->next;
    }
    return false;
}

boolean 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;
    while (Q != Nil && Q->next != P) {
        Q = Q->next;
    }
    return Q;
}

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

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

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

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

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

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

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

void insertLastAnak(listanak &L, address_anak P) {
    if (ListEmptyAnak(L)) {
        L.first = P;
        L.last = P;
```

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

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

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

void delAfter(listinduk &L, address &P, address Prec) {
    P = Prec->next;
    if (P != Nil) {
        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) {
        address_anak C = P->lanak.first;
        while (C != Nil) {
            address_anak H = C;
            C = C->next;
            delete H;
        }
        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;
        }
        delete P;
    }
}

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

void delLastAnak(listanak &L, address_anak &P) {
    P = L.last;
    if (L.first == L.last) {
        L.first = Nil;
```

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

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

void printInfo(listinduk L) {
    address P = L.first;
    while (P != Nil) {
        cout << P->info << ": ";
        address_anak C = P->lanak.first;
        while (C != Nil) {
            cout << C->info << " ";
```

```
            C = C->next;
        }
        cout << endl;
        P = P->next;
    }
}

void printInfoAnak(listanak L) {
    address_anak P = L.first;
    while (P != Nil) {
        cout << P->info << " ";
        P = P->next;
    }
}

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

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

void delAll(listinduk &L) {
    address P = L.first;
    while (P != Nil) {
        address Q = P;
        P = P->next;
        address_anak C = Q->lanak.first;
```

```
        while (C != Nil) {
            address_anak H = C;
            C = C->next;
            delete H;
        }
        delete Q;
    }
    L.first = Nil;
    L.last = Nil;
}
```

(file 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
```

Screenshots Output

```
nadiv@omarnadip Soal 1 % ./run
=== INSERT DATA INDUK ===
10:
20:
30:

=== INSERT DATA ANAK ===
10: 101 102
20: 201
30:

=== DELETE ANAK (102 dari induk 10) ===
10: 101
20: 201
30:

=== DELETE INDUK (30) ===
10: 101
20: 201

=== JUMLAH DATA ===
Jumlah Induk: 2
Jumlah Anak Induk 10: 1
Jumlah Anak Induk 20: 1
nadiv@omarnadip Soal 1 %
```

Deskripsi:

Program ini mengimplementasikan struktur data multilist yang terdiri dari dua level, yaitu list induk dan list anak. Setiap elemen induk dapat memiliki daftar anak sendiri yang disimpan menggunakan doubly linked list. Struktur ini digunakan untuk merepresentasikan hubungan satu-ke-banyak secara dinamis, di mana setiap induk bisa memiliki nol atau lebih elemen anak.

Pada program ini, list induk diisi dengan beberapa data menggunakan operasi insertLast. Setelah itu, elemen-elemen anak ditambahkan ke induk tertentu dengan menggunakan insertLastAnak. Program juga mendukung operasi penghapusan, seperti menghapus anak tertentu melalui delPAnak dan menghapus induk beserta seluruh daftar anaknya melalui delP. Fungsi printInfo digunakan untuk menampilkan seluruh induk beserta anak-anaknya, sedangkan nbList dan nbListAnak digunakan untuk menghitung jumlah data. Melalui proses tersebut, program memperlihatkan cara mengelola data bertingkat dengan menggunakan multilist secara terstruktur dan dinamis.

Unguided 2 (file main.cpp)

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

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;
   return alokasi(x);
}

int main() {
   List L;
   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", 'p', 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;
}
```

(file circularlist.cpp)

```
#include "circularlist.h"
using namespace std;

void createList(List &L)
{
    L.first = Nil;
}

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

void dealokasi(address P)
{
    delete P;
}
```

```
void insertFirst(List &L, address P)
{
   if (L.first == Nil)
   {
       L.first = P;
       P->next = P;
   }
   else
   {
       address Q = L.first;
       while (Q->next != L.first)
       {
           Q = Q->next;
       }
       P->next = L.first;
       Q->next = P;
       L.first = P;
   }
}

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

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

```
        {
            Q = Q->next;
        }
        Q->next = P;
        P->next = L.first;
    }
}

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

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

```
    }
}

void deleteLast(List &L, address &P)
{
    if (L.first != Nil)
    {
        address Q = L.first;
        if (Q->next == Q)
        {
            P = Q;
            L.first = Nil;
        }
        else
        {
            address R = Nil;
            while (Q->next != L.first)
            {
                R = Q;
                Q = Q->next;
            }
            P = Q;
            R->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);
```

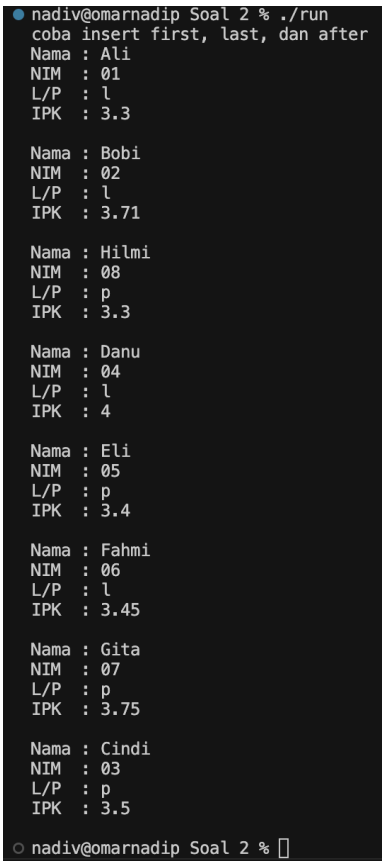
```
    return Nil;
}

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

    address P = L.first;
    do {
        cout << "Nama : " << P->info.nama << endl;
        cout << "NIM  : " << P->info.nim << endl;
        cout << "L/P  : " << P->info.jenis_kelamin << endl;
        cout << "IPK  : " << P->info.ipk << endl;
        cout << endl;

        P = P->next;
    } while (P != L.first);
}
```

(file circularlist.h)

```
#ifndef CIRCULARLIST_H_INCLUDED
#define CIRCULARLIST_H_INCLUDED

#include <iostream>
#include <string>
#define Nil NULL
using namespace std;

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

struct ElmList;
```

```
typedef ElmList* address;

struct ElmList {
   infotype info;
   address next;
};

struct List {
   address first;
};

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
```

Screenshots Output



Deskripsi:

Program ini mengimplementasikan circular linked list untuk menyimpan data mahasiswa yang mencakup nama, NIM, jenis kelamin, dan IPK. Setiap elemen terhubung secara melingkar sehingga node terakhir selalu menunjuk kembali ke node pertama. Program menyediakan operasi dasar seperti membuat list, mengalokasikan elemen, menyisipkan data di awal, akhir, atau setelah node tertentu, mencari elemen berdasarkan NIM, menghapus elemen, serta menampilkan seluruh isi list. Melalui fungsi createData dan serangkaian pengujian pada main.cpp, program menunjukkan cara kerja circular list dalam menyimpan dan menata data secara dinamis serta berulang tanpa menggunakan penanda akhir seperti NULL.

D. Kesimpulan

Pada modul ini, saya mempelajari konsep Multi Linked List, yaitu struktur data yang menghubungkan elemen induk dengan elemen-anak secara terpisah namun saling terkait. Setiap node induk memiliki list anak tersendiri, sehingga struktur ini cocok digunakan untuk data bertingkat seperti kategori–sub kategori atau dosen–mahasiswa. Melalui implementasi operasi dasar seperti insert, update, delete, pencarian, dan penampilan data, saya memahami bagaimana mengelola hubungan satu-ke-banyak dalam bentuk linked list. Modul ini membantu saya memahami struktur data kompleks yang lebih fleksibel dibandingkan linked list biasa, serta bagaimana mengimplementasikannya dalam program C++ secara sistematis.

E. Referensi

https://geeksforgeeks.org

https://www.duniailkom.com

https://en.wikipedia.org/wiki/Linked_list

https://www.tutorialspoint.com/data_structures_algorithms/linked_list_algorithms.htm

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