LAPORAN PRAKTIKUM STRUKTUR DATA

MODUL
MULTI LINKED LIST

Disusun Oleh :
NAMA : Ayu Setyaning Tyas
NIM : 103112430119

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Multi linked list adalah jenis daftar khusus yang berisi dua atau lebih urutan kunci logis. Sebelum memeriksa detail tentang daftar multi-linked, lihat apa yang merupakan daftar terkait. Daftar terkait adalah struktur data yang bebas dari batasan ukuran apa pun sampai memori tumpukan tidak penuh. Kami telah melihat berbagai jenis daftar terkait, seperti Singly Linked List, Daftar Terkait Edaran, dan Daftar Terhubung Doubly. Di sini kita akan melihat tentang daftar multi-linked.

Multi linked list dapat digunakan untuk mempertahankan sublist dalam daftar data yang lebih besar tanpa harus benar-benar menduplikasi catatan atau menyimpannya secara fisik dalam file yang terpisah. Variabel pointer diatur, satu untuk setiap sublist yang harus dipertahankan. Ini disimpan dalam struktur catatan itu sendiri.

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

Guided 1

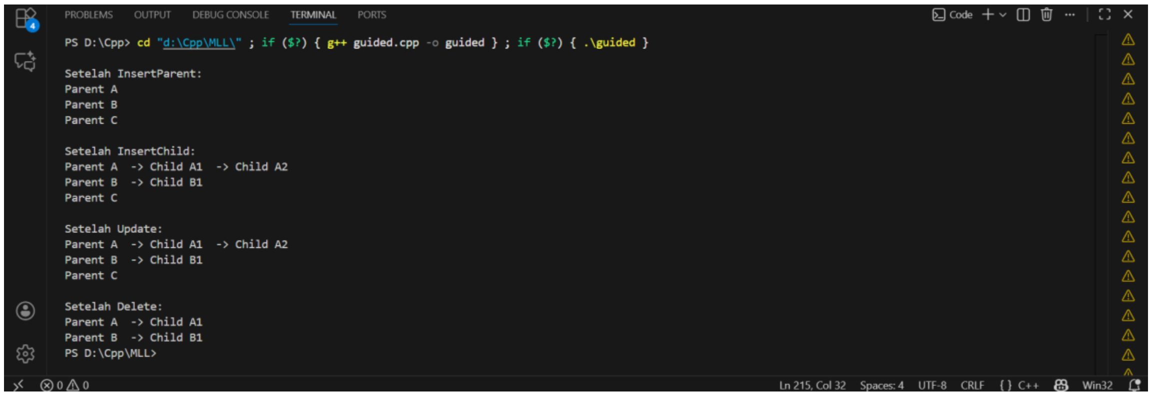
Code *guided.cpp*

```
 #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 insertClild(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);

    insertClild(list, "Parent A", "Child A1");
    insertClild(list, "Parent A", "Child A2");
    insertClild(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;
}
```

Screenshoot Output



```
PS D:\Cpp> cd "d:\Cpp\MLL\" ; if ($?) { g++ guided.cpp -o guided } ; if ($?) { .\guided }

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:\Cpp\MLL>
```

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

Unguided 1

Code *multilist.h*

```
#ifndef MULTILIST_H_INCLUDED
#define MULTILIST_H_INCLUDED

#include <iostream>
using namespace std;

#define Nil NULL

typedef bool boolean;
typedef int infotypeinduk;
typedef int infotypeanak;

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 findElm(listanak L, infotypeanak X);
void insertFirst(listinduk &L, address P);
void insertLast(listinduk &L, address P);
void insertAfter(listinduk &L, address Prec, address P);
void insertFirstAnak(listanak &L, address_anak P);
void insertLastAnak(listanak &L, address_anak P);
void insertAfterAnak(listanak &L, address_anak P, address_anak Prec);
void delFirst(listinduk &L, address &P);
void delLast(listinduk &L, address &P);
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 L);

#endif
```

Code *multilist.cpp*

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

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

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

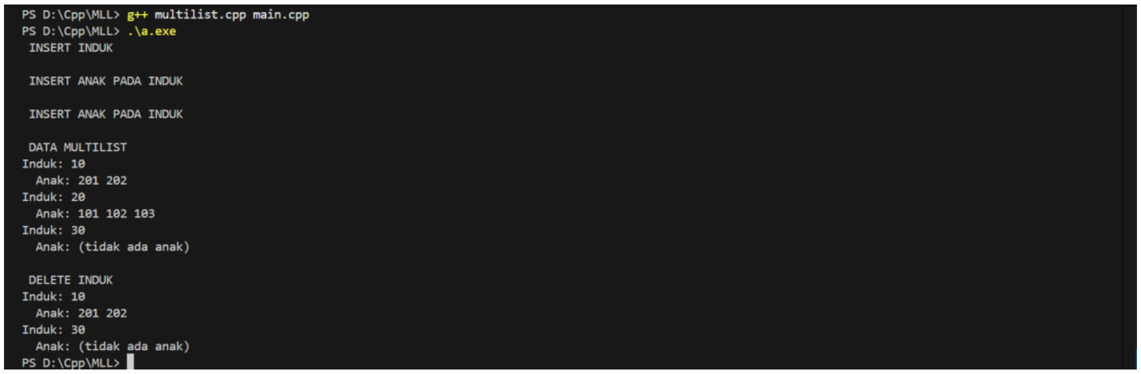
```
address_anak alokasiAnak(infotypeanak X) {
   address_anak P = new elemen_list_anak;

   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 findElm(listanak L, infotypeanak X) {
   address_anak P = L.first;

   while (P != Nil) {
      if (P->info == X)
         return P;
      P = P->next;
   }
   return Nil;
}

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

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;
    }
}

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

    if (P != Nil) {
        address_anak Q = P->lanak.first;
        while (Q != Nil) {
            address_anak temp = Q;
            Q = Q->next;
            dealokasiAnak(temp);
        }

        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;
        }
        dealokasi(P);
    }
}

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;
    }
}

void printInfo(listinduk L) {
    address P = L.first;

    while (P != Nil) {
        cout << "Induk: " << P->info << endl;
        cout << "  Anak: ";
        address_anak Q = P->lanak.first;

        if (Q == Nil)
            cout << "(tidak ada anak)";
        else {
            while (Q != Nil) {
                cout << Q->info << " ";
                Q = Q->next;
            }
        }
        cout << endl;
        P = P->next;
    }
}
```

Code *main1.cpp*

```
#include <iostream>

#include "multilist.h"

using namespace std;


int main() {

    listinduk L;

    CreateList(L);


    cout << " INSERT INDUK " << endl;

    insertLast(L, alokasi(10));

    insertLast(L, alokasi(20));
    insertLast(L, alokasi(30));

    cout << "\n INSERT ANAK PADA INDUK " << endl;
    address induk20 = findElm(L, 20);

    insertLastAnak(induk20->lanak, alokasiAnak(101));
    insertLastAnak(induk20->lanak, alokasiAnak(102));
    insertLastAnak(induk20->lanak, alokasiAnak(103));

    cout << "\n INSERT ANAK PADA INDUK " << endl;
    address induk10 = findElm(L, 10);

    insertLastAnak(induk10->lanak, alokasiAnak(201));
    insertLastAnak(induk10->lanak, alokasiAnak(202));

    cout << "\n DATA MULTILIST " << endl;
    printInfo(L);

    cout << "\n DELETE INDUK " << endl;
    delP(L, 20);

    printInfo(L);

    return 0;
}
```

Screenshots Output



```
PS D:\Cpp\MLL> g++ multilist.cpp main.cpp
PS D:\Cpp\MLL> .\a.exe
 INSERT INDUK

 INSERT ANAK PADA INDUK

 INSERT ANAK PADA INDUK

 DATA MULTILIST
Induk: 10
  Anak: 201 202
Induk: 20
  Anak: 101 102 103
Induk: 30
  Anak: (tidak ada anak)

 DELETE INDUK
Induk: 10
  Anak: 201 202
Induk: 30
  Anak: (tidak ada anak)
PS D:\Cpp\MLL>
```

Deskripsi:

Program ini membuat daftar bertingkat (induk dan anak) menggunakan doubly linked list untuk pengarahan yang mudah. Program ini punya fungsi lengkap untuk menambah, mencari, dan menghapus data, termasuk kemampuan menghapus induk beserta semua anaknya sekaligus. Jadi program ini adalah struktur untuk mengelola data yang memiliki hubungan hierarki.

Unguided 2

Code *circularlist.h*

```
#ifndef CIRCULARLIST_H_INCLUDED
#define CIRCULARLIST_H_INCLUDED
#define Nil NULL

#include <string>
using namespace std;

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

typedef struct 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
```

Code *circularlist.cpp*

```
#include <iostream>
#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 = P;
    return P;
}

void dealokasi(address P){
    delete P;
}

void insertFirst(List &L, address P){
    if(L.first == Nil){
        L.first = P;
    } else {
        address Q = L.first;

        while(Q->next != L.first){
            Q = Q->next;
        }

        Q->next = P;
        P->next = L.first;
        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){
        insertFirst(L, 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 = Nil;
    }
    else if(L.first->next == L.first){
        P = L.first;
        L.first = Nil;
    }
    else {
        address last = L.first;

        while(last->next != L.first){
            last = last->next;
        }

        P = L.first;
        L.first = L.first->next;
        last->next = L.first;
    }
}
```

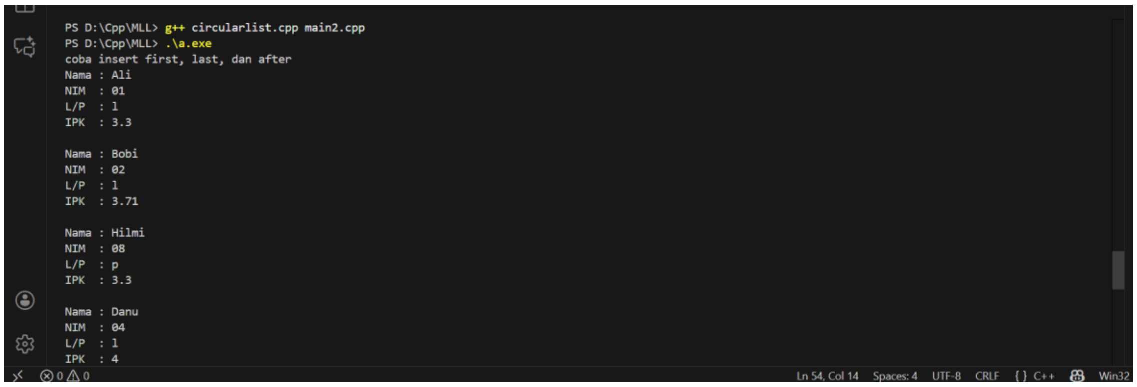
```
void deleteAfter(List &L, address Prec, address &P){
    if(Prec != Nil){
        P = Prec->next;
        Prec->next = P->next;
        P->next = Nil;
    }
}

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

        while(Q->next != L.first){
            Prev = Q;
            Q = Q->next;
        }

        P = Q;
        Prev->next = L.first;
    }
}

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);
}
```

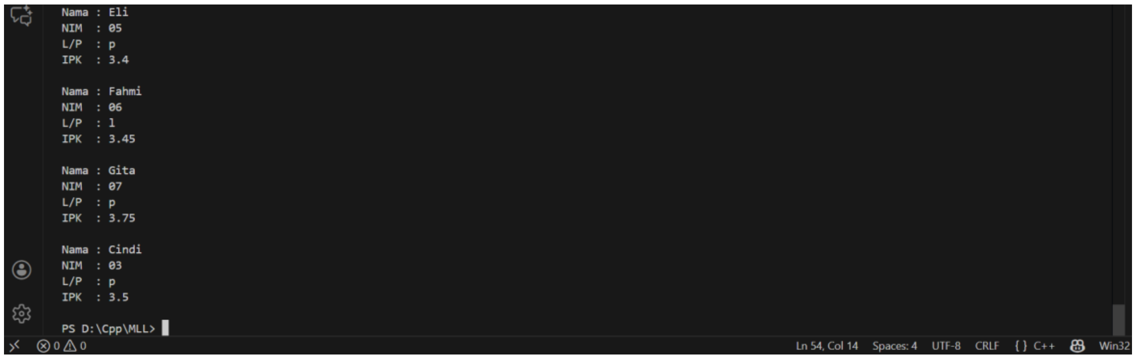
Code *main2.cpp*

```
#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;
}
```

Screenshots Output



```
PS D:\Cpp\MLL> g++ circularlist.cpp main2.cpp
PS D:\Cpp\MLL> .\a.exe
coba insert first, last, dan after
Nama : Ali
NIM  : 01
L/P  : l
IPK  : 3.3

Nama : Bobi
NIM  : 02
L/P  : l
IPK  : 3.71

Nama : Hilmi
NIM  : 08
L/P  : p
IPK  : 3.3

Nama : Danu
NIM  : 04
L/P  : l
IPK  : 4
```



```
Nama : Eli
NIM  : 05
L/P  : p
IPK  : 3.4

Nama : Fahmi
NIM  : 06
L/P  : l
IPK  : 3.45

Nama : Gita
NIM  : 07
L/P  : p
IPK  : 3.75

Nama : Cindi
NIM  : 03
L/P  : p
IPK  : 3.5

PS D:\Cpp\MLL> 
```

Deskripsi:

Program ini adalah struktur list melingkar di mana elemen terakhir menunjuk kembali ke elemen pertama. Program ini menyimpan data mahasiswa dan mendukung operasi membuat list, menambah elemen, menghapus, mencari berdasarkan NIM, serta menampilkan isi list secara melingkar.

D. Kesimpulan

Program-program ini menunjukkan dua cara pintar mengatur data. Program yang pertama, Multilist, bekerja seperti sistem folder bertingkat , pas sekali untuk data hierarkis (berjenjang) karena ia bisa membersihkan satu folder induk beserta semua isinya secara total. Sementara itu, program yang kedua adalah Daftar Melingkar , yang membuat rantai data terus berputar tanpa akhir ini sering dipakai untuk mengurus data mahasiswa dan memastikan proses penelusuran selalu berlanjut. Jadi, kita punya dua solusi unik: satu fokus pada hubungan induk-anak, dan satu lagi fokus pada alur data yang tak terputus.

E. Referensi

https://www.geeksforgeeks.org/dsa/introduction-to-multi-linked-list/
https://sites.google.com/site/proffriedmancplusplus/assignments/multilinked-list