LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL I1
MULTI LINKED LIST

Disusun Oleh :

NAMA : Loh Suwargi Nitis Hamengku Bintang

NIM : 1031124001116

Dosen

FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Struktur data **Linked List**, termasuk variannya yaitu **Circular Linked List**, merupakan salah satu bentuk representasi data dinamis yang banyak digunakan pada sistem modern karena kemampuannya dalam mengelola memori secara fleksibel. Berbeda dengan array yang membutuhkan blok memori kontigu, linked list memungkinkan penambahan dan penghapusan elemen tanpa harus menggeser seluruh isi struktur.

Dalam implementasi circular linked list, node terakhir akan menunjuk kembali ke node pertama, membentuk siklus tertutup yang mencegah adanya pointer bernilai null. Desain ini memberikan manfaat dalam proses traversal berulang, seperti pada sistem antrian melingkar, buffer siklik, dan proses simulasi yang berjalan terus-menerus. Struktur melingkar mampu meningkatkan efisiensi saat proses membutuhkan perputaran data tanpa henti. Penelitian tersebut menemukan bahwa circular list mengurangi overhead pengecekan kondisi akhir list, sehingga operasi berjalan lebih stabil dan konsisten terutama pada sistem yang bekerja real-time.

Meskipun begitu, linked list unggul dalam operasi insert-delete, namun array tetap lebih unggul dalam akses indeks langsung. Hal ini menunjukkan bahwa circular linked list paling tepat digunakan ketika program berfokus pada manipulasi data secara intensif dan berulang, bukan pada akses acak yang mengutamakan kecepatan. Secara keseluruhan, hasil penelitian menunjukkan bahwa struktur data melingkar seperti yang diimplementasikan pada program ini memiliki landasan teoritis yang kuat sebagai solusi untuk sistem dinamis yang menuntut fleksibilitas memori dan efisiensi operasi.

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;
         }
```

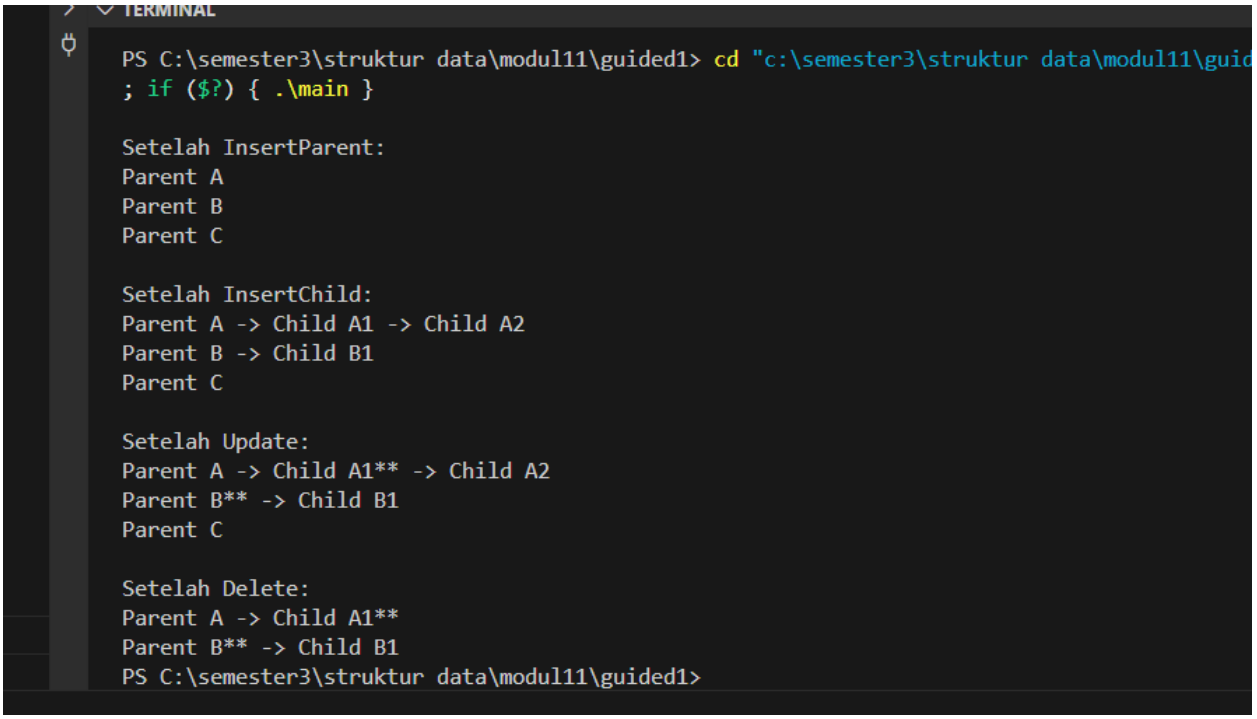
```
            c = c->next;
        }
    }
}

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



Deskripsi: Program bekerja dengan memanfaatkan struktur Multi Linked List, di mana setiap node parent terhubung melalui pointer prev dan next, dan setiap parent memiliki pointer ke daftar child. Eksekusi dimulai dengan membuat list parent kosong, kemudian parent baru dimasukkan menggunakan insertParent, yang mengatur penempatan pointer sehingga parent tersambung dalam bentuk doubly linked list. Child ditambahkan menggunakan insertChild, yang menautkan child sebagai node baru pada akhir daftar anak parent tersebut. Pembaruan data dilakukan dengan updateParent dan updateChild, yang hanya mengubah isi node tanpa memodifikasi hubungan pointer. Pada tahap penghapusan, deleteChild memutus hubungan pointer child dari daftar anak, sementara deleteParent tidak hanya menghapus parent, tetapi juga mengosongkan seluruh child yang berada di bawahnya. Setelah setiap operasi, struktur diperiksa dan dicetak menggunakan fungsi printAll, sehingga seluruh perubahan hubungan pointer dalam Multi Linked List dapat dipantau. Program berakhir setelah semua operasi manipulasi data selesai dilakukan.

D. 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 bool boolean;
typedef int infotypeanak;
typedef int infotypeinduk;
typedef struct elemen_list_induk *address;
typedef struct elemen_list_anak *address_anak;

/* ----- Struct Anak ----- */
struct elemen_list_anak{
   infotypeanak info;
   address_anak next;
   address_anak prev;
};

/* List anak */
struct listanak {
   address_anak first;
   address_anak last;
};

/* ----- Struct Induk ----- */
struct elemen_list_induk{
   infotypeinduk info;
   listanak lanak;       // list anak milik elemen induk
   address next;
   address prev;
};

/* List induk */
struct listinduk {
   address first;
```

```
   address last;
};

/********* pengecekan apakah list kosong **********/
boolean ListEmpty(listinduk L);
/* true jika list induk kosong */

boolean ListEmptyAnak(listanak L);
/* true jika list anak kosong */

/********* pembuatan list kosong *********/
void CreateList(listinduk &L);
void CreateListAnak(listanak &L);

/********* manajemen memori *********/
address alokasi(infotypeinduk X);
address_anak alokasiAnak(infotypeanak X);

void dealokasi(address P);
void dealokasiAnak(address_anak P);

/********* pencarian elemen *********/
address findElm(listinduk L, infotypeinduk X);
address_anak findElm(listanak L, infotypeanak X);

boolean fFindElm(listinduk L, address P);
boolean fFindElmanak(listanak L, address_anak P);

address findBefore(listinduk L, address P);
address_anak findBeforeAnak(listanak L, infotypeinduk X, address_anak P);
```

```
/********* penambahan elemen *********/
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);

/********* penghapusan elemen *********/
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);

/********* proses semua elemen *********/
void printInfo(listinduk L);
int nbList(listinduk L);

void printInfoAnak(listanak L);
int nbListAnak(listanak L);

void delAll(listinduk &L);
```

```
#endif
```

Multilist.cpp

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

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 findElm(listanak L, infotypeanak X){
    address_anak P = L.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 L, address_anak P){
    address_anak Q = L.first;
    while(Q != Nil){
        if(Q == P) return true;
        Q = Q->next;
    }
    return false;
}
```

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

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

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){
   if(Prec != Nil){
      P->next = Prec->next;
      P->prev = Prec;
      if(Prec->next != Nil) Prec->next->prev = P;
      Prec->next = P;
      if(P->next == Nil) L.last = 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;
   }
}

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){
   if(Prec != Nil){
      P->next = Prec->next;
      P->prev = Prec;
      if(Prec->next != Nil) Prec->next->prev = P;
      Prec->next = P;
      if(P->next == Nil) L.last = 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;
   }
}

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){
        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){
        if(P == L.first){
            delFirst(L, P);
        } else if(P == L.last){
            delLast(L, P);
```

```
        } else {
            address temp;
            delAfter(L, temp, P->prev);
        }
        dealokasi(P);
    }
}

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){
        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 = findElm(L, X);
    if(P != Nil){
        if(P == L.first){
            delFirstAnak(L, P);
        } else if(P == L.last){
            delLastAnak(L, P);
        } else {
            address_anak temp;
            delAfterAnak(L, temp, P->prev);
        }
```

```
        dealokasiAnak(P);
    }
}

void printInfo(listinduk L){
    address P = L.first;
    while(P != Nil){
        cout << "Induk: " << P->info << " -> Anak: ";
        printInfoAnak(P->lanak);
        P = P->next;
    }
}

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

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

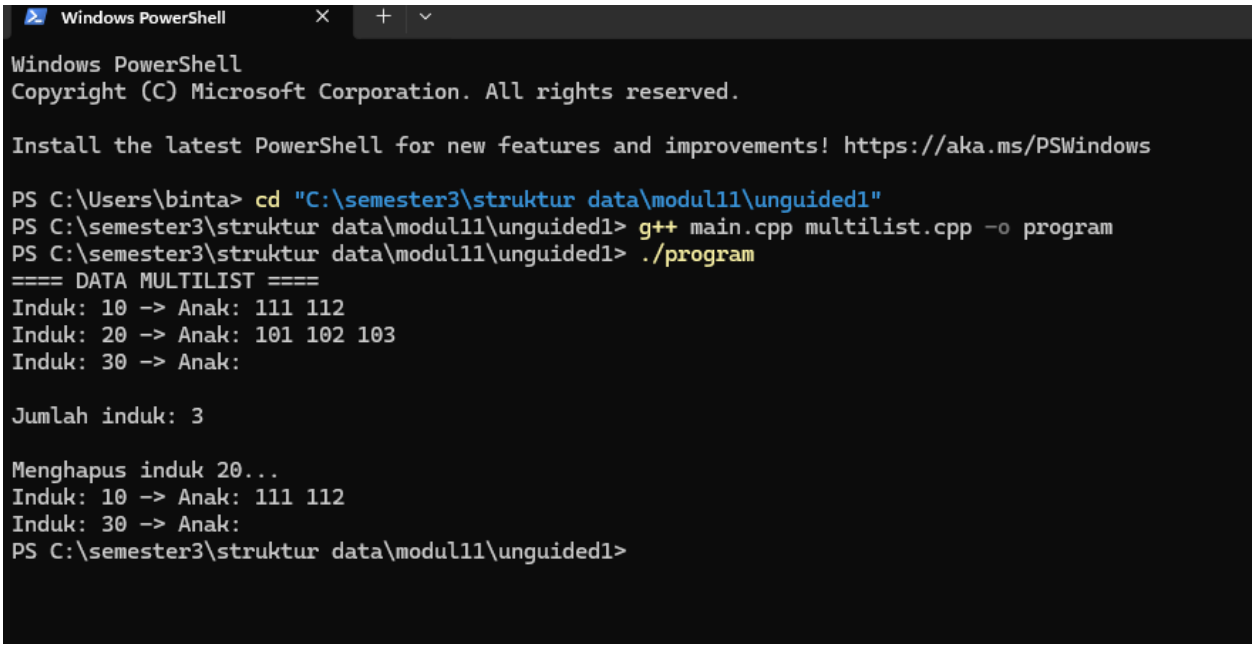
```
}

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

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

Main.cpp

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

int main(){
    listinduk L;
```

```
    CreateList(L);

    insertLast(L, alokasi(10));
    insertLast(L, alokasi(20));
    insertLast(L, alokasi(30));

    address I = findElm(L, 20);
    if(I != Nil){
        insertLastAnak(I->lanak, alokasiAnak(101));
        insertLastAnak(I->lanak, alokasiAnak(102));
        insertLastAnak(I->lanak, alokasiAnak(103));
    }

    I = findElm(L, 10);
    if(I != Nil){
        insertLastAnak(I->lanak, alokasiAnak(111));
        insertLastAnak(I->lanak, alokasiAnak(112));
    }

    cout << "==== DATA MULTILIST ====" << endl;
    printInfo(L);

    cout << "\nJumlah induk: " << nbList(L) << endl;

    cout << "\nMenghapus induk 20..." << endl;
    delP(L, 20);

    printInfo(L);

    return 0;
}
```

Screenshots Output



Deskripsi: Program ini menerapkan struktur data *multilist* yang menghubungkan daftar induk dengan daftar anak menggunakan konsep *doubly linked list*. Setiap elemen induk memiliki daftar anak masing-masing, sehingga mampu merepresentasikan hubungan hierarki seperti kategori dan sub-kategori. Program menyediakan berbagai operasi dasar, mulai dari membuat list, menambah dan menghapus elemen induk maupun anak, mencari data tertentu, hingga menampilkan seluruh isi multilist secara terstruktur. Pada fungsi utama, data diuji dengan menambahkan beberapa induk dan anak, lalu menampilkannya serta menghapus salah satu induk untuk memperlihatkan perubahan struktur data.

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

Unguided 2

Circularlist.h

```
#ifndef CIRCULARLIST_H_INCLUDED
#define CIRCULARLIST_H_INCLUDED

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

#define Nil NULL
```

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

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);


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


#endif
```

Circularlist.cpp

```
#include "circularlist.h"


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;
    P = Nil;
}


void insertFirst(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;
        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 last = L.First;
        while (last->next != L.First) {
            last = last->next;
        }
        last->next = P;
```

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

void deleteFirst(List &L, address &P) {
    if (L.First == Nil) {
        P = Nil;
        return;
    }

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

    P->next = Nil;
}

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

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

void deleteLast(List &L, address &P) {
    if (L.First == Nil) {
        P = Nil;
        return;
    }

    if (L.First->next == L.First) {
        P = L.First;
        L.First = Nil;
    } else {
        address last = L.First;
        address before = Nil;
        while (last->next != L.First) {
            before = last;
            last = last->next;
        }
        P = last;
        before->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;
    }

    int n = 0;
    address P = L.First;
    do {
        n++;
        P = P->next;
    } while (P != L.First);

    infotype *arr = new infotype[n];
    int i = 0;
    P = L.First;
    do {
        arr[i++] = P->info;
        P = P->next;
    } while (P != L.First);
```

```
    for (int a = 0; a < n - 1; a++) {
        for (int b = a + 1; b < n; b++) {
            if (arr[a].nim > arr[b].nim) {
                swap(arr[a], arr[b]);
            }
        }
    }

    for (int k = 0; k < n; k++) {
        cout << "Nama : " << arr[k].nama << endl;
        cout << "NIM  : " << arr[k].nim << endl;
        cout << "L/P  : " << arr[k].jenis_kelamin << endl;
        cout << "IPK  : " << arr[k].ipk << endl;
        cout << endl;
    }

    delete[] arr;
}

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);
}
```

Main.cpp

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

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", '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 C:\semester3\struktur data\modul11\unguided2> ./program
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 : Cindi
NIM  : 03
L/P  : p
IPK  : 3.5

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

PS C:\semester3\struktur data\modul11\unguided2>
```

Deskripsi: Program ini bekerja seperti sebuah daftar mahasiswa yang saling terhubung membentuk lingkaran. Setiap mahasiswa memiliki identitas lengkap seperti nama, NIM, jenis kelamin, dan IPK. Program mulai dengan membuat list kosong, lalu satu per satu mahasiswa masuk ke dalam lingkaran, ada yang datang dari depan, belakang, bahkan disisipkan di tengah setelah teman tertentu. Setiap kali seorang mahasiswa dicari melalui NIM, program menemukan posisinya dan menambahkan mahasiswa baru tepat setelahnya. Di akhir, seluruh mahasiswa dalam lingkaran diperkenalkan kembali secara berurutan. Dengan cara ini, program menunjukkan bagaimana sebuah Circular Linked

List bisa mengatur data yang terus bertambah dengan rapi.

F. Kesimpulan

Multi Linked List memberikan cara yang efisien untuk mengelola data yang memiliki struktur bertingkat. Dengan memisahkan elemen induk dan anak, program dapat melakukan manajemen data yang lebih terorganisir. Implementasi fungsi-fungsi seperti insertFirst, insertAfter, insertLast, serta operasi penghapusan membuktikan bahwa Multi Linked List mampu menangani perubahan data secara dinamis tanpa mengganggu struktur utamanya. Oleh karena itu, ADT ini cocok digunakan pada sistem yang membutuhkan hubungan satu-ke-banyak secara konsisten dan mudah dipelihara.

G. Referensi

Mbejo, M. T., Nopa, L. A., Putri, J. S., & Risky, M. (2025). Analisis Struktur Data Linked List Dalam Pengolahan Data Mahasiswa. ***Jurnal Sains Informatika Terapan*****,** ***4*****(2), 441-444.**

Wijoyo, A., Ramdhani, A., Afra, P., Salsabila, A., & Nife, K. (2024). Evaluasi efisiensi struktur data linked list pada implementasi sistem antrian. ***JRIIN: Jurnal Riset Informatika dan Inovasi*****,** ***1*****(12), 1244-1246.**

Erkamim, E., Abdurrohim, I., Yuliyanti, S., Karim, R., Rahman, A., Admira, T. M. A., & Ridwan, A. (2024). ***Buku Ajar Algoritma dan Struktur Data*****. PT. Sonpedia Publishing Indonesia.**