LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL XI
MULTI LINKED LIST

Disusun Oleh :
NAMA : Ridha Akifah
NIM : 103112400132

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Multi Linked List adalah struktur data yang terdiri atas beberapa linked list yang saling berhubungan, di mana satu elemen pada list induk memiliki list anak yang menyimpan data terkait. Setiap elemen pada list induk dapat menjadi “parent” bagi satu atau lebih elemen anak, sehingga hubungan hierarkis dapat terbentuk secara dinamis. Konsep ini memungkinkan pengelompokan data yang lebih kompleks, misalnya list pegawai yang masing-masing memiliki list anak. Operasi dasar seperti insert dan delete dilakukan baik pada level induk maupun anak, namun manipulasi data anak harus melalui induknya untuk menjaga konsistensi hubungan. Multi Linked List mempermudah representasi data bertingkat dan sering digunakan pada sistem yang membutuhkan relasi satu-ke-banyak dalam struktur yang fleksibel.

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

guided.cpp

```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

```
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
```

Deskripsi:

Program ini membuat struktur induk–anak menggunakan doubly linked list. Setiap node induk dapat memiliki daftar anak, dan program menyediakan fitur untuk menambah induk maupun anak, memperbarui data, menampilkan seluruh isi list, serta menghapus anak tertentu atau menghapus induk beserta semua anaknya. Program ini memudahkan pengelolaan data bertingkat secara dinamis.

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

Unguided 1

multilist.h

```
#ifndef MULTILIST_H_INCLUDED

#define MULTILIST_H_INCLUDED
```

```cpp
#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
```

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;
    }
}
```

main.cpp

```
#include <iostream>
#include "multilist.h"
#include "multilist.cpp"
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;
}
```

Screenshoot Output

```
 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)
```

Deskripsi:

Program ini membuat struktur induk–anak dengan doubly linked list dan menyediakan fitur tambah, cari, serta hapus, termasuk menghapus induk beserta semua anaknya. Singkatnya, program ini mengelola data hierarki secara sederhana dan efisien.

Unguided 2

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
```

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);
}
```

main.cpp

```
#include <iostream>

#include "circularlist.h"

#include "circularlist.cpp"

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("Dani", "04", 'l', 4.0);

    insertFirst(L,P1);


    P1 = createData("FBagus", "06", 'l',3.45);

    insertLast(L,P1);


    P1 = createData("Dina", "02", 'l',3.71);

    insertFirst(L,P1);


    P1 = createData("Alif", "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("Indi", "03", 'p', 3.5);
    insertAfter(L, P1, P2);


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


    x.nim = "04";
    P1 = findElm(L,x);
    P2 = createData("Ela", "05", 'p', 3.4);
    insertAfter(L, P1, P2);
    printInfo(L);
    return 0;
}
```

Screenshots Output

```
coba insert first, last, dan after
Nama : Alif
NIM  : 01
L/P  : l
IPK  : 3.3

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

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

Nama : Dani
NIM  : 04
L/P  : l
IPK  : 4

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

Deskripsi:

Program ini menggunakan list melingkar yang menghubungkan elemen terakhir ke elemen pertama. Data mahasiswa disimpan dan dapat ditambahkan, dihapus, dicari lewat NIM, serta ditampilkan berputar mengikuti struktur circular-nya. Singkatnya, program ini mempermudah pengelolaan data dalam bentuk list yang terus berulang.

D. Kesimpulan

Multi Linked List merupakan struktur data yang efektif untuk merepresentasikan hubungan satu-ke-banyak secara dinamis, karena setiap elemen induk dapat memiliki daftar anak yang terhubung langsung. Melalui operasi insert dan delete pada level induk maupun anak, struktur ini memungkinkan pengelolaan data bertingkat dengan lebih fleksibel dibanding linked list biasa. Implementasinya membantu mahasiswa memahami cara membangun dan memanipulasi hubungan hierarkis dalam suatu sistem data, serta memperkuat pemahaman mengenai pointer, manajemen memori, dan konsep list terhubung yang lebih kompleks.

E. Referensi

https://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B

https://learn.microsoft.com/id-id/cpp/cpp/references-cpp?view=msvc-170

https://medium.com/@mohamedeissabay/understanding-c-a-deep-dive-into-core-concepts-48a560679cdd