LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL XI
MULTI LINKED LIST

Disusun Oleh :
NAMA : ZAKI MAULA DHIA
NIM : 103112400127

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

MULTI LINKED LIST adalah struktur data yang setiap node memiliki lebih dari satu node, mirip dengan struktur data TREE yang memiliki node parent dan node child MULTI LINKED LIST memiliki sebuah node parent dan child untuk membantu mengsortir data yang masuk. MULTI LINKED LIST dibuat untuk memaksimalkan potensi dari single atau double linked list dibeberapa kondisi.

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 oldChild-
Info, 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
                {
                    if (c->prev != NULL)
                    {
                        c->prev->next = c->next;
```

```
                    }
                    if (c->next != NULL)
                    {
                        c->next->prev = c->prev;
                    }
                }
                delete c;
                return;
            }
            c = c->next;
        }
    }
}

void deleteParent(ParentNode *&head, string info)
{
    ParentNode *p = head;
    while (p != NULL)
    {
        if (p->info == info)
        {
            ChildNode *c = p->childHead;
            while (c != NULL)
            {
                ChildNode *tempC = c;
                c = c->next;
                delete tempC;
            }
            if (p == head)
            {
                head = p->next;
                if (head != NULL)
                {
                    head->prev = NULL;
                }
            }
            else
            {
                p->prev->next = p->next;
                if (p->next != NULL)
                {
                    p->next->prev = p->prev;
                }
            }
```

```
            delete p;
            return;
        }
        p = p->next;
    }
}

int main()
{
    ParentNode *list = NULL;

    insertParent(list, "Parent A");
    insertParent(list, "Parent B");
    insertParent(list, "Parent C");

    cout << "\nSetelah InsertParent:" << endl;
    printAll(list);

    insertChild(list, "Parent A", "Child A1");
    insertChild(list, "Parent A", "Child A2");
    insertChild(list, "Parent B", "Child B1");

    cout << "\nSetelah InsertChild:" << endl;
    printAll(list);

    updateParent(list, "Parent B", "Parent B*");
    updateChild(list, "Parent A", "Child A1", "Child A1*");

    cout << "\nSetelah Update:" << endl;
    printAll(list);

    deleteChild(list, "Parent A", "Child A2");
    deleteParent(list, "Parent C");

    cout << "\nSetelah Delete:" << endl;
    printAll(list);

    return 0;
}
```

Screenshots Output

```
PS C:\Users\MyBook Hype AMD\Documents\strukturdata>  &
dowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-c
e.lhe' '--pid=Microsoft-MIEngine-Pid-zbpxybz5.xts' '--d

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 C:\Users\MyBook Hype AMD\Documents\strukturdata>
```

Deskripsi:

Program ini adalah implementasi dari struktur data MULTI LINKED LIST. Dalam program tersebut akan membuat daftar parent yang masing-masing memiliki child, mirip seperti sruktur data TREE yang ada dimodul 10. Program akan membuat parent A,B,C menggunakan function InsertParent(), kemudian program menambahkan child ke parent A, dan B menggunakan functon InsertChild(), lalu program melakukan update pada parent B menggunakan function updateParent() dan update child menggunakan function updateChild(), dan function deleteChild() dan deleteParent() untuk menghapus node child atau parent, dan terakhir function printALL() untuk menampilkan semua node yang dimiliki program.

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

Unguided 1

```cpp
// main.cpp
#include <iostream>
#include "multilist.h"
#include "multilist.cpp"

using namespace std;

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

    cout << "1. Membuat 3 elemen induk (10, 20, 30)...\n";
    address P1 = alokasi(10);
    address P2 = alokasi(20);
    address P3 = alokasi(30);

    insertLast(L, P1);
    insertLast(L, P2);
    insertLast(L, P3);

    cout << "2. Menambahkan anak ke induk 10...\n";
    address_anak A1 = alokasiAnak(11);
    address_anak A2 = alokasiAnak(12);
    address_anak A3 = alokasiAnak(13);

    insertLastAnak(P1->lanak, A1);
    insertLastAnak(P1->lanak, A2);
    insertLastAnak(P1->lanak, A3);

    cout << "3. Menambahkan anak ke induk 20...\n";
    address_anak A4 = alokasiAnak(21);
    address_anak A5 = alokasiAnak(22);

    insertLastAnak(P2->lanak, A4);
    insertLastAnak(P2->lanak, A5);

    cout << "\n=== MENAMPILKAN DATA ===\n";
    printInfo(L);

    cout << "\n=== INFORMASI ===\n";
```

```
    cout << "Jumlah induk: " << nbList(L) << endl;
    cout << "Jumlah anak induk 10: " << nbListAnak(P1->lanak) << endl;
    cout << "Jumlah anak induk 20: " << nbListAnak(P2->lanak) << endl;

    return 0;
}
```

```
// multilist.h
#ifndef MULTILIST_H_INCLUDED
#define MULTILIST_H_INCLUDED
#define Nil NULL

typedef int infotypeanak;
typedef int infotypeinduk;

typedef struct elemen_list_induk *address;
typedef struct elemen_list_anak *address_anak;

typedef bool boolean;

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 P);
address_anak alokasiAnak(infotypeanak P);

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

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);

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);
int nbList(listinduk L);

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

```
void delAll(listinduk &L);

#endif
```

```
// multilist.cpp
#include <iostream>
#include <cstdlib>
#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 P) {
    address newElm = new elemen_list_induk;
    if (newElm != Nil) {
        newElm->info = P;
        CreateListAnak(newElm->lanak);
        newElm->next = Nil;
        newElm->prev = Nil;
    }
    return newElm;
}

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

```
    if (newElm != Nil) {
        newElm->info = P;
        newElm->next = Nil;
        newElm->prev = Nil;
    }
    return newElm;
}

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 && P->info != X) {
        P = P->next;
    }
    return P;
}

address_anak findElm(listanak L, infotypeanak X) {
    address_anak P = L.first;
    while (P != Nil && P->info != X) {
        P = P->next;
    }
    return P;
}

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

boolean fFindElmanak(listanak L, address_anak P) {
```

```
    address_anak temp = L.first;
    while (temp != Nil) {
        if (temp == P) {
            return true;
        }
        temp = temp->next;
    }
    return false;
}

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

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

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

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

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

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

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

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

        if (Prec->next != Nil) {
            Prec->next->prev = P;
```

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

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

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

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

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

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

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

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

void delP(listinduk &L, infotypeinduk X) {
    address P = findElm(L, X);
    if (P != Nil) {
        if (P == L.first) {
            address temp;
            delFirst(L, temp);
            dealokasi(temp);
        } else if (P == L.last) {
            address temp;
            delLast(L, temp);
            dealokasi(temp);
        } else {
            address Prec = P->prev;
            address temp;
            delAfter(L, temp, Prec);
            dealokasi(temp);
        }
    }
}

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

```
        if (L.first == L.last) {
            L.first = Nil;
            L.last = Nil;
        } else {
            L.first = L.first->next;
            L.first->prev = Nil;
            P->next = Nil;
        }
    }
}

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

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

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

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

void delPAnak(listanak &L, infotypeanak X) {
    address_anak P = findElm(L, X);
    if (P != Nil) {
```

```
        if (P == L.first) {
            address_anak temp;
            delFirstAnak(L, temp);
            dealokasiAnak(temp);
        } else if (P == L.last) {
            address_anak temp;
            delLastAnak(L, temp);
            dealokasiAnak(temp);
        } else {
            address_anak Prec = P->prev;
            address_anak temp;
            delAfterAnak(L, temp, Prec);
            dealokasiAnak(temp);
        }
    }
}

void printInfo(listinduk L) {
    if (ListEmpty(L)) {
        cout << "List Induk Kosong" << endl;
    } else {
        address P = L.first;
        int no = 1;
        while (P != Nil) {
            cout << "Induk " << no << ": " << P->info << endl;
            if (!ListEmptyAnak(P->lanak)) {
                cout << "  Anak: ";
                printInfoAnak(P->lanak);
            } else {
                cout << "  Anak: (kosong)" << endl;
            }
            P = P->next;
            no++;
        }
    }
}

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

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

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

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)) {
        address_anak PAnak;
        while (!ListEmptyAnak(L.first->lanak)) {
            delFirstAnak(L.first->lanak, PAnak);
            dealokasiAnak(PAnak);
        }
        delFirst(L, P);
        dealokasi(P);
    }
}
```

Screenshots Output

```
PS C:\Users\MyBook Hype AMD\Documents\strukturdata>  &
dowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-f
k.xoq' '--pid=Microsoft-MIEngine-Pid-uplp5scc.xbd' '--d
1. Membuat 3 elemen induk (10, 20, 30)...
2. Menambahkan anak ke induk 10...
3. Menambahkan anak ke induk 20...

=== MENAMPILKAN DATA ===
Induk 1: 10
  Anak: 11 <-> 12 <-> 13
Induk 2: 20
  Anak: 21 <-> 22
Induk 3: 30
  Anak: (kosong)

=== INFORMASI ===
Jumlah induk: 3
Jumlah anak induk 10: 3
Jumlah anak induk 20: 2
PS C:\Users\MyBook Hype AMD\Documents\strukturdata>
```

Deskripsi:

Program tersebut dibuat untuk mengimplementasikan struktur data MULTI LINKED LIST, karena struktur data tersebut memiliki kecocokan dengan data hierarkis atau bertingkat, maka program tersebut cocok dengan struktur data MULTi LINKED LIST. Program menggunakan banyak function yang dapat digunakan oleh struktur data MULTI LINKED LIST sehingga program tersebut sudah hampir lengkap, user hanya perlu mengganti main.cpp saja.

Unguided 2

```cpp
// main.cpp
#include "circularlist.h"
#include "circularlist.cpp"


int main() {
```

```
    List L;
    address P1, P2;
    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;
}
```

```
// circularlist.h
#ifndef CIRCULARLIST_H_INCLUDED
#define CIRCULARLIST_H_INCLUDED
```

```cpp
#include <iostream>
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;
    address prev;
};

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 = P->prev = Nil;
    return P;
}

void dealokasi(address P) {
    delete P;
}

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);
}

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

```
}

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

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

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

    P = L.first;

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

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

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

    address last = L.first->prev;

    if (last == L.first) {
        deleteFirst(L, P);
    } else {
        P = last;
        address before_last = last->prev;

        before_last->next = L.first;
        L.first->prev = before_last;

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

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

        Prec->next = Q;
        Q->prev = Prec;

        P->next = P->prev = 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) return;

    address P = L.first;

    do {
        cout << "Nama : " << P->info.nama << endl;
        cout << "NIM  : " << P->info.nim << endl;
        cout << "JK   : " << P->info.jenis_kelamin << endl;
        cout << "IPK  : " << P->info.ipk << endl << endl;
        P = P->next;
    } while (P != L.first);
}
```

Screenshots Output

```
PS C:\Users\MyBook Hype AMD\Documents\strukturdata>
dowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-I
m.klz' '--pid=Microsoft-MIEngine-Pid-o2pmuech.uzm' '
coba insert first, last, dan after
Nama : Ali
NIM  : 01
JK   : l
IPK  : 3.3

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

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

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

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

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

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

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

Deskripsi:

Program tersebut merupakan implementasi dari Circular Doubly Linked List, struktur data yang terdiri dari node-node yang saling terhubung dua ara. Program ini menyimpan data mahasiswa dengan menggunakan struct, semua data akan disimpan dalam node (ElmList) yang saling terhubung.

D. Kesimpulan

Materi MULTI LINKED LIST memiliki efisiensi yang lebih baik dibandingkan single maupun doubly linked list, MULTI LINKED LIST yang memiliki parent dan child sehingga struktur data tersebut sangat cocok dengan data bertingkat seperti mata kuliah – mahasiswa – nilai, algoritma search juga akan menjadi lebih efisien karena data di

kelompokan berdasarkan parent-child.

E. Referensi

- Enea, C., Saveluc, V., & Sighireanu, M. (2013, March). Compositional invariant checking for overlaid and nested linked lists. In European Symposium on Programming (pp. 129-148). Berlin, Heidelberg: Springer Berlin Heidelberg.
- Materi Praktikum Modul 11, MULTI LINKED LIST.
- https://www.geeksforgeeks.org/dsa/introduction-to-multi-linked-list/