LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL XI
MULTI LINKED LIST

Disusun Oleh :
NAMA : ABYAN RAHMAN AL FARIZ
NIM : 103112430021

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Multi Linked List (Multilist) adalah pengembangan dari linked list biasa, di mana setiap node (simpul) tidak hanya menunjuk ke satu node berikutnya, tetapi bisa memiliki pointer ganda untuk menunjuk ke elemen lain, seringkali untuk menangani hubungan "satu ke banyak" (1-N) atau "banyak ke banyak" (N-M), seperti menghubungkan data induk dengan data anaknya secara terpisah dalam struktur yang saling terkait

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

Guided 1 (main.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 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;
        }
    }
}
```

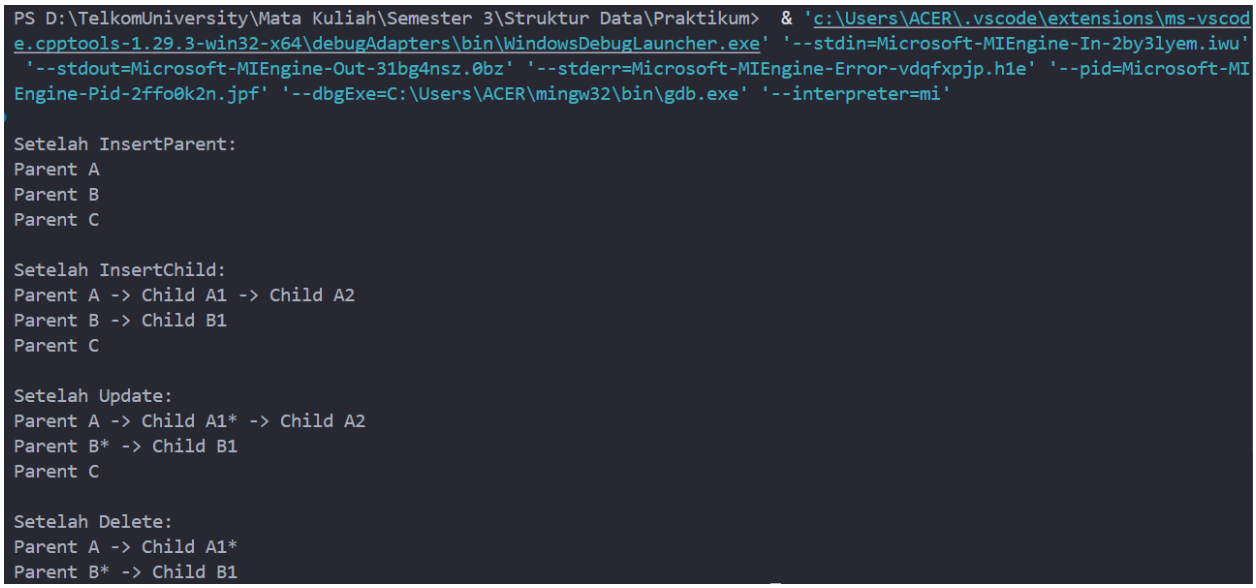
```
void printAll(ParentNode *head) {
    while (head != NULL) {
        cout << head->info;
        ChildNode *c = head->childHead;
        while (c != NULL) {
            cout << " -> " << c->info;
            c = c->next;
        }
        cout << endl;
        head = head->next;
    }
}

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 ini mengimplementasikan struktur data Multi Linked List yang terdiri dari parent node dan child node dengan konsep double linked list pada masing-masing level. Setiap parent menyimpan sebuah informasi dan memiliki pointer childHead yang menunjuk ke daftar child, sedangkan hubungan antar parent maupun antar child dihubungkan dengan pointer next dan prev. Program menyediakan operasi lengkap mulai dari pembuatan node (createParent dan createChild), penambahan data (insertParent dan insertChild), penampilan seluruh struktur data (printAll), pembaruan data parent dan child (updateParent dan updateChild), hingga penghapusan child tertentu dan parent beserta seluruh child-nya (deleteChild dan deleteParent).

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

Unguided 2 (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) {
    if (P != Nil) {
        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) return;
    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;
    }

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

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

```
            L.First = 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;
        P->next = Nil;
    } else {
        address Q = L.First;
        while (Q->next->next != L.First) Q = Q->next;
        P = Q->next; // last
        Q->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.\n";
        return;
    }
    cout << "Daftar Mahasiswa:\n";
    address P = L.First;
    int idx = 1;
    do {
```

```
        cout << idx++ << ". ";
        cout << "Nama: " << P->info.nama << ", NIM: " <<
P->info.nim
             << ", JK: " << P->info.jenis_kelamin
             << ", IPK: " << P->info.ipk << '\n';
        P = P->next;
    } while (P != L.First);
}

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);
}
```

Unguided 2 (circularlist.h)

```
#ifndef CIRCULARLIST_H
#define CIRCULARLIST_H

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

#define Nil nullptr

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
```

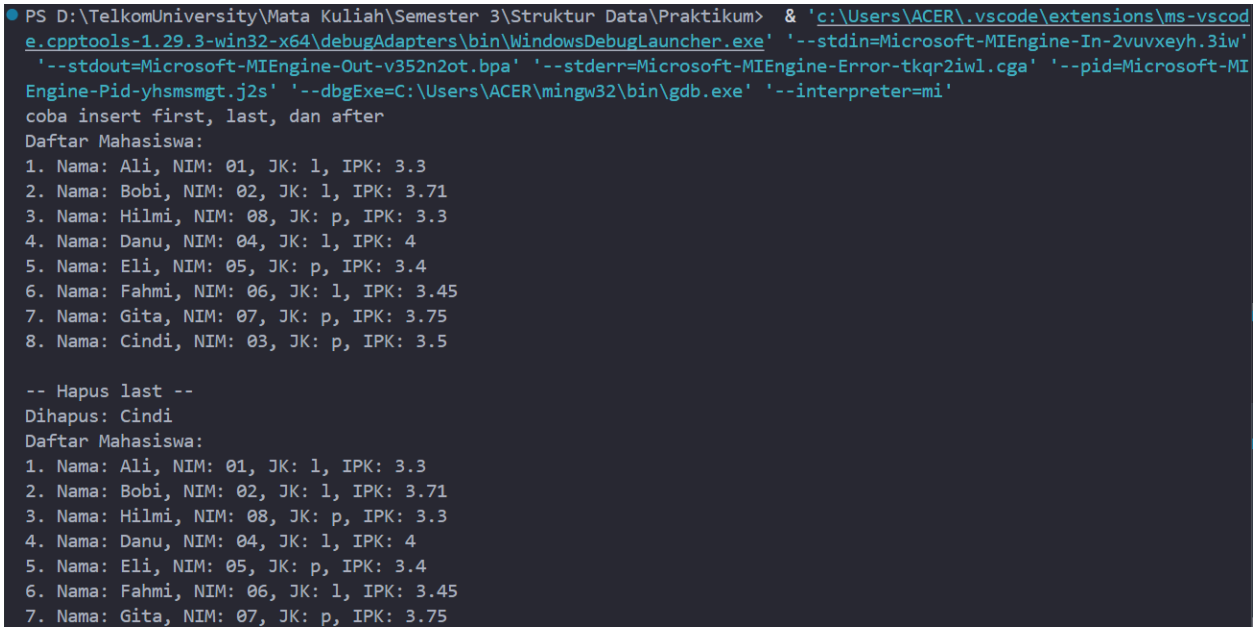
Unguided 2 (main.cpp)

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

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

    address P1, P2;
    infotype x;

    cout << "coba insert first, last, dan after\n";
```

```
    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);
    if (P1 != Nil) insertAfter(L, P1, P2);

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

    x.nim = "04";
    P1 = findElm(L,x);
    P2 = createData("Eli", "05", 'p', 3.4);
    if (P1 != Nil) insertAfter(L, P1, P2);

    printInfo(L);

    cout << "\n-- Hapus last --\n";
    address del;
    deleteLast(L, del);
    if (del != Nil) {
        cout << "Dihapus: " << del->info.nama << "\n";
        dealokasi(del);
    }
    printInfo(L);

    return 0;
```

```
}
```

Screenshots Output:



Deskripsi:

Program Circular Linked List ini digunakan untuk mengelola data mahasiswa yang berisi nama, NIM, jenis kelamin, dan IPK. Pada awal program, list dibuat dalam kondisi kosong menggunakan prosedur CreateList. Struktur circular memungkinkan node terakhir menunjuk kembali ke node pertama, sehingga list membentuk lingkaran dan tidak memiliki nilai NULL pada penunjuk akhirnya.

Selanjutnya, program mendemonstrasikan berbagai operasi dasar pada circular linked list, yaitu insertFirst, insertLast, dan insertAfter. Data mahasiswa dimasukkan ke dalam list pada posisi awal dan akhir, serta disisipkan setelah node tertentu berdasarkan pencarian NIM menggunakan fungsi findElm. Hal ini menunjukkan fleksibilitas circular linked list dalam menambahkan data di berbagai posisi tanpa bergantung pada urutan linear.

Setelah seluruh data dimasukkan, fungsi printInfo digunakan untuk menampilkan isi list secara menyeluruh. Program juga memperagakan operasi deleteLast untuk menghapus node terakhir dari list, kemudian memori node tersebut

dibebaskan menggunakan dealokasi. Setelah proses penghapusan, isi list ditampilkan kembali untuk memastikan perubahan data telah terjadi.

Unguided 1 (multilist.cpp)

```
#include "multilist.h"

void createListJurusan(ListJurusan &L) {
    L.firstJur = NULL;
}

adrJur alokasiJur(infotypeJur x) {
    adrJur P = new ElmJur;
    P->info = x;
    P->nextJur = NULL;
    P->firstMhs = NULL;
    return P;
}

adrMhs alokasiMhs(infotypeMhs x) {
    adrMhs P = new ElmMhs;
    P->info = x;
    P->nextMhs = NULL;
    return P;
}

void insertJurusan(ListJurusan &L, adrJur P) {
    if (L.firstJur == NULL) {
        L.firstJur = P;
    } else {
        adrJur Q = L.firstJur;
        while (Q->nextJur != NULL) {
            Q = Q->nextJur;
        }
        Q->nextJur = P;
    }
}

void insertMahasiswa(adrJur PJur, adrMhs PMhs) {
    if (PJur != NULL) {
```

```
        if (PJur->firstMhs == NULL) {
            PJur->firstMhs = PMhs;
        } else {
            adrMhs Q = PJur->firstMhs;
            while (Q->nextMhs != NULL)
                Q = Q->nextMhs;
            Q->nextMhs = PMhs;
        }
    }
}

adrJur findJurusan(ListJurusan L, string kode) {
    adrJur P = L.firstJur;
    while (P != NULL) {
        if (P->info.kode == kode) return P;
        P = P->nextJur;
    }
    return NULL;
}

adrMhs findMahasiswa(adrJur PJur, string nim) {
    if (PJur == NULL) return NULL;

    adrMhs P = PJur->firstMhs;
    while (P != NULL) {
        if (P->info.nim == nim) return P;
        P = P->nextMhs;
    }
    return NULL;
}

void printData(ListJurusan L) {
    adrJur PJ = L.firstJur;
    cout << "\n===== DATA JURUSAN DAN MAHASISWA =====\n";

    while (PJ != NULL) {
        cout << "Jurusan: " << PJ->info.nama << " (" <<
PJ->info.kode << ")\n";

        adrMhs PM = PJ->firstMhs;
        if (PM == NULL) {
            cout << "   Tidak ada mahasiswa.\n";
        } else {
```

```
            while (PM != NULL) {
                cout << "    - " << PM->info.nim << " | " <<
PM->info.nama << endl;
                PM = PM->nextMhs;
            }
        }
        cout << endl;

        PJ = PJ->nextJur;
    }
}
```

Unguided 1 (multilist.h)

```
#ifndef MULTILIST_H_INCLUDED
#define MULTILIST_H_INCLUDED

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

struct Mahasiswa {
    string nim;
    string nama;
};

typedef Mahasiswa infotypeMhs;

struct ElmMhs {
    infotypeMhs info;
    ElmMhs *nextMhs;
};

typedef ElmMhs* adrMhs;


struct Jurusan {
    string kode;
    string nama;
};

typedef Jurusan infotypeJur;
```

```
struct ElmJur {
    infotypeJur info;
    adrMhs firstMhs;
    ElmJur *nextJur;
};

typedef ElmJur* adrJur;

struct ListJurusan {
    adrJur firstJur;
};


void createListJurusan(ListJurusan &L);

adrJur alokasiJur(infotypeJur x);
adrMhs alokasiMhs(infotypeMhs x);

void insertJurusan(ListJurusan &L, adrJur P);
void insertMahasiswa(adrJur PJur, adrMhs PMhs);

adrJur findJurusan(ListJurusan L, string kode);
adrMhs findMahasiswa(adrJur PJur, string nim);

void printData(ListJurusan L);

#endif
```

Unguided 1 (main.cpp)

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

int main() {
    ListJurusan L;
    createListJurusan(L);

    cout << "=== INPUT DATA JURUSAN ===" << endl;
    adrJur j1 = alokasiJur({"IF", "Informatika"});
```

```
    adrJur j2 = alokasiJur({"SI", "Sistem Informasi"});
    insertJurusan(L, j1);
    insertJurusan(L, j2);

    cout << "=== INPUT DATA MAHASISWA ===" << endl;
    insertMahasiswa(j1, alokasiMhs({"103112430001", "Abyan"}));
    insertMahasiswa(j1, alokasiMhs({"103112430021", "Fahmi"}));

    insertMahasiswa(j2, alokasiMhs({"103112410011", "Bintang"}));

    printData(L);

    cout << "\n=== CARI MAHASISWA ===" << endl;
    adrJur cariJur = findJurusan(L, "IF");
    adrMhs cariMhs = findMahasiswa(cariJur, "103112430001");

    if (cariMhs != NULL) {
        cout << "Mahasiswa ditemukan: " << cariMhs->info.nama <<
endl;
    } else {
        cout << "Mahasiswa tidak ditemukan!" << endl;
    }

    return 0;
}
```

Screenshots Output:

```
PS D:\TelkomUniversity\Mata Kuliah\Semester 3\Struktur Data\Praktikum>  & 'c:\Users\ACER\.vscode\extensions\ms-vscod
e.cpptools-1.29.3-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-xqy3d3s2.kc4'
 '--stdout=Microsoft-MIEngine-Out-boko2tzy.2tw' '--stderr=Microsoft-MIEngine-Error-xrajfgad.ndg' '--pid=Microsoft-MI
Engine-Pid-0ztelb20.0ii' '--dbgExe=C:\Users\ACER\mingw32\bin\gdb.exe' '--interpreter=mi'
=== INPUT DATA JURUSAN ===
=== INPUT DATA MAHASISWA ===

===== DATA JURUSAN DAN MAHASISWA =====
Jurusan: Informatika (IF)
   - 103112430001 | Abyan
   - 103112430021 | Fahmi

Jurusan: Sistem Informasi (SI)
   - 103112410011 | Bintang


=== CARI MAHASISWA ===
Mahasiswa ditemukan: Abyan
```

Deskripsi:

Program main.cpp ini berfungsi sebagai driver program untuk menguji implementasi struktur data Multi Linked List pada data jurusan dan mahasiswa. Pada

awal program, sebuah list jurusan dibuat menggunakan fungsi createListJurusan, yang akan menjadi wadah utama untuk menyimpan seluruh data jurusan beserta mahasiswa di dalamnya.

Selanjutnya, program melakukan input data jurusan dengan membuat dua jurusan, yaitu Informatika (IF) dan Sistem Informasi (SI), melalui fungsi alokasiJur, kemudian memasukkannya ke dalam list menggunakan insertJurusan. Setelah itu, program menambahkan data mahasiswa ke masing-masing jurusan menggunakan insertMahasiswa. Jurusan Informatika memiliki dua mahasiswa, sedangkan jurusan Sistem Informasi memiliki satu mahasiswa. Hal ini menunjukkan hubungan satu jurusan dapat memiliki banyak mahasiswa, yang merupakan ciri utama dari struktur multi linked list.

Setelah data dimasukkan, fungsi printData digunakan untuk menampilkan seluruh isi list, mulai dari jurusan hingga mahasiswa yang terdaftar di setiap jurusan. Program kemudian mendemonstrasikan proses pencarian data dengan mencari jurusan tertentu menggunakan findJurusan, lalu mencari mahasiswa berdasarkan NIM di dalam jurusan tersebut menggunakan findMahasiswa. Jika mahasiswa ditemukan, program akan menampilkan nama mahasiswa tersebut.

D. Kesimpulan

Modul 11 membahas struktur data linked list lanjutan, yaitu Circular Linked List dan Multi List. Circular Linked List membentuk hubungan melingkar sehingga cocok untuk proses yang berjalan berulang, sedangkan Multi List digunakan untuk merepresentasikan hubungan satu-ke-banyak seperti jurusan dan mahasiswa. Kedua struktur ini memudahkan pengelolaan data yang dinamis, efisien, dan terstruktur dalam berbagai aplikasi.

E. Referensi

https://www.geeksforgeeks.org/dsa/introduction-to-multi-linked-list/

https://www.slideshare.net/slideshow/8-multi-list-struktur-data/114941552

https://medium.com/@rk29sidhu/introduction-to-multi-linked-list-2d0ca3073883