LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL XI
MULTI LINKED LIST

Disusun Oleh :

NAMA : RISKY CAHAYU

NIM : 103112430121

Dosen

FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Multi Linked List merupakan struktur data yang terdiri dari beberapa list yang saling terhubung satu sama lain. Setiap elemen dalam Multi Linked List dapat membentuk list sendiri. Biasanya, ada dua jenis list: list induk dan list anak. List induk dapat menunjuk ke beberapa list anak, dan setiap list anak dapat berisi elemen-elemen yang saling terkait.

Struktur Multi Linked List

Struktur ini terdiri dari dua jenis list:

1. List Induk: Berisi elemen yang menunjuk ke list anak.
2. List Anak: Berisi elemen-elemen yang saling terkait dalam list anak.

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

Guided 1

Kode main.cpp

```cpp
// Risky Cahayu
// 103112430121

#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

```
PS C:\Users\ASUS\Videos\strukdat> cd "c:\Users\ASUS\Videos\strukdat\Modul 11\Guided\" ; if ($?)

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

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

Setelah Update:
Parent A -> Child A1* -> Child A2
Parent B* -> Child B1
Parent B* -> Child B1
Parent C

Setelah Delete:
Parent A -> Child A1*
Parent B* -> Child B1
PS C:\Users\ASUS\Videos\strukdat\Modul 11\Guided>
```

Deskripsi:

Program ini mengimplementasikan struktur data Multilevel Linked List (MLL) yang terdiri dari parent node dan child node dengan hubungan satu ke banyak, di mana setiap parent memiliki sebuah linked list child sendiri. Algoritma dimulai dengan pembuatan node parent dan child menggunakan alokasi dinamis, kemudian menyediakan operasi dasar seperti insert parent di akhir list, insert child pada parent tertentu, menampilkan seluruh data parent beserta child-nya, memperbarui data parent dan child, serta menghapus child tertentu atau parent beserta seluruh child-nya. Proses pencarian parent atau child dilakukan secara sekuensial (linear search), sedangkan penghapusan node memastikan keterkaitan pointer next dan prev tetap konsisten untuk menjaga integritas struktur double linked list. Pada fungsi main, algoritma mendemonstrasikan alur penggunaan struktur data mulai dari penambahan data, pembaruan, hingga penghapusan, sehingga menunjukkan cara kerja Multilevel Linked List secara lengkap dan terstruktur.

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

Unguided 1

Kode circularlist.cpp

```cpp
// Risky Cahayu
// 103112430121

#include "circularList.h"
```

```cpp
#include <iostream>
using namespace std;

void CreateList(List &L) {
    L.first = NULL;
}

address alokasi(string nama, string nim, char jenis_kelamin, float ipk) {
    address P = new ElmList;
    P->nama = nama;
    P->nim = nim;
    P->jenis_kelamin = jenis_kelamin;
    P->ipk = ipk;
    P->next = NULL;
    return P;
}

void dealokasi(address P) {
    delete P;
}

void insertFirst(List &L, address P) {
  if (L.first == NULL) {
        L.first = P;
        P->next = L.first;
    } else {
        address last = L.first;
        while (last->next != L.first) {
            last = last->next;
        }
        last->next = P;
        P->next = L.first;
        L.first = P;
    }
}

void insertAfter(List &L, address P, address Prec) {
    P->next = Prec->next;
    Prec->next = P;
}

void insertLast(List &L, address P) {
    if (L.first == NULL) {
        L.first = P;
        P->next = L.first;
    } 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 == NULL) {
        P = NULL;
    } else {
        P = L.first;
        address last = L.first;
        while (last->next != L.first) {
            last = last->next;
        }
        if (L.first == L.first->next) {
            L.first = NULL;
        } else {
            L.first = L.first->next;
            last->next = L.first;
        }
    }
}

void deleteAfter(List &L, address &P, address Prec) {
    P = Prec->next;
    Prec->next = P->next;
    P->next = NULL;
}

void deleteLast(List &L, address &P) {
    address last = L.first;
    address beforeLast;
    if (L.first != NULL) {
        while (last->next != L.first) {
            beforeLast = last;
            last = last->next;
        }
        if (L.first == L.first->next) {
            L.first = NULL;
        } else {
            beforeLast->next = L.first;
        }
    }
    P = last;
    delete P;
}
```

```
address findElm(List L, string nim) {
    address P = L.first;
    while (P != NULL && P->nim != nim) {
        P = P->next;
    }
    return P;
}

void printInfo(List L) {
    address P = L.first;
    if (P != NULL) {
        do {
            cout << "Nama: " << P->nama << ", NIM: " << P->nim << ",
Jenis Kelamin: "
            << P->jenis_kelamin << ", IPK: " << P->ipk << endl;
            P = P->next;
        } while (P != L.first);
    } else {
        cout << "List Kosong!" << endl;
    }
}
```

Kode circularlist.h

```
// Risky Cahayu
// 103112430121

#ifndef CIRCULARLIST_H
#define CIRCULARLIST_H

#include <string>
using namespace std;

typedef struct ElmList *address;
typedef struct List {
    address first;
} List;

typedef struct ElmList {
    string nama;
    string nim;
    char jenis_kelamin;
    float ipk;
    address next;
} ElmList;

void CreateList(List &L);
```

```
address alokasi(string nama, string nim, char jenis_kelamin, float ipk);
void dealokasi(address P);
void insertFirst(List &L, address P);
void insertAfter(List &L, address P, address Prec);
void insertLast(List &L, address P);
void deleteFirst(List &L, address &P);
void deleteAfter(List &L, address &P, address Prec);
void deleteLast(List &L, address &P);
address findElm(List L, string nim);
void printInfo(List L);

#endif
```

Kode main.cpp

```
// Risky Cahayu
// 103112430121

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

int main() {
    List L;
    address P;

    CreateList(L);

    P = alokasi("Nanda B. P.", "11013134395", 'L', 3.3);
    insertFirst(L, P);

    P = alokasi("David H. M.", "11013130314", 'L', 3.71);
    insertLast(L, P);

    P = alokasi("Masiyah Nur Aulia", "1103130120", 'P', 4.0);
    insertFirst(L, P);

    printInfo(L);

    deleteFirst(L, P);
    cout << "Setelah Delete First:" << endl;
    printInfo(L);

    return 0;
}
```

Screenshots Output

```
PS C:\Users\ASUS\Videos\strukdat\Modul 11\Unguided\soal1> ./main
Nama: Masiyah Nur Aulia, NIM: 1103130120, Jenis Kelamin: P, IPK: 4
Nama: Nanda B. P., NIM: 11013134395, Jenis Kelamin: L, IPK: 3.3
Nama: David H. M., NIM: 11013130314, Jenis Kelamin: L, IPK: 3.71
Setelah Delete First:
Nama: Nanda B. P., NIM: 11013134395, Jenis Kelamin: L, IPK: 3.3
Nama: David H. M., NIM: 11013130314, Jenis Kelamin: L, IPK: 3.71
PS C:\Users\ASUS\Videos\strukdat\Modul 11\Unguided\soal1>
```

Deskripsi:

Pada Unguided 1, praktikum ini berfokus pada implementasi Multi Linked List dalam struktur data. Multi Linked List menggabungkan lebih dari satu list yang saling terhubung satu sama lain, dengan list induk yang menunjuk ke list anak. Praktikum ini menguji pemahaman tentang cara menambah (insert) dan menghapus (delete) elemen dalam list induk dan list anak, serta bagaimana mencari dan menampilkan elemen berdasarkan kriteria tertentu. Dalam implementasinya, fungsi-fungsi seperti insertFirst, insertLast, dan insertAfter digunakan untuk menambah elemen pada posisi tertentu, sementara findElm dan deleteFirst digunakan untuk mencari dan menghapus elemen. Tujuan dari praktikum ini adalah agar mahasiswa dapat memahami konsep dasar Multi Linked List serta dapat mengimplementasikannya dengan benar melalui kode yang diberikan.

Unguided 2

Kode circularlist.cpp

```cpp
// Risky Cahayu
// 103112430121

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

void CreateList(List &L) {
    L.first = NULL;
}

address alokasi(string nama, string nim, char jenis_kelamin, float ipk) {
    address P = new ElmList;
    P->nama = nama;
    P->nim = nim;
    P->jenis_kelamin = jenis_kelamin;
    P->ipk = ipk;
    P->next = NULL;
    P->prev = NULL;
    return P;
}
```

```
void dealokasi(address P) {
    delete P;
}

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

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

void insertLast(List &L, address P) {
    if (L.first == NULL) {
        L.first = P;
        P->next = L.first;
        P->prev = L.first;
    } else {
        address last = L.first;
        while (last->next != L.first) {
            last = last->next;
        }
        last->next = P;
        P->prev = last;
        P->next = L.first;
        L.first->prev = P;
    }
}
```

```
void deleteFirst(List &L, address &P) {
    if (L.first == NULL) {
        P = NULL;
    } else {
        P = L.first;
        address last = L.first;
        while (last->next != L.first) {
            last = last->next;
        }
        if (L.first == L.first->next) {
            L.first = NULL;
        } else {
            L.first = L.first->next;
            last->next = L.first;
            L.first->prev = last;
        }
    }
}

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

void deleteLast(List &L, address &P) {
    address last = L.first;
    if (L.first != NULL) {
        while (last->next != L.first) {
            last = last->next;
        }
        if (L.first == L.first->next) {
            L.first = NULL;
        } else {
            last->prev->next = L.first;
            L.first->prev = last->prev;
        }
    }
    P = last;
    delete P;
}

address findElm(List L, string nim) {
```

```
    address P = L.first;
    while (P != NULL && P->nim != nim) {
        P = P->next;
    }
    return P;
}

void printInfo(List L) {
    address P = L.first;
    if (P != NULL) {
        do {
            cout << "Nama : " << P->nama << endl;
            cout << "NIM  : " << P->nim << endl;
            cout << "L/P  : " << P->jenis_kelamin << endl;
            cout << "IPK  : " << P->ipk << endl;
            cout << endl;
            P = P->next;
        } while (P != L.first);
    } else {
        cout << "List Kosong!" << endl;
    }
}
```

Kode circularlist.h

```
// Risky Cahayu
// 103112430121

#ifndef CIRCULARLIST_H
#define CIRCULARLIST_H

#include <string>
using namespace std;

typedef struct ElmList *address;
typedef struct List {
    address first;
} List;

typedef struct ElmList {
    string nama;
    string nim;
    char jenis_kelamin;
    float ipk;
    address next;
    address prev;
} ElmList;
```

```
void CreateList(List &L);
address alokasi(string nama, string nim, char jenis_kelamin, float ipk);
void dealokasi(address P);
void insertFirst(List &L, address P);
void insertAfter(List &L, address P, address Prec);
void insertLast(List &L, address P);
void deleteFirst(List &L, address &P);
void deleteAfter(List &L, address &P, address Prec);
void deleteLast(List &L, address &P);
address findElm(List L, string nim);
void printInfo(List L);

#endif
```

Kode main.cpp

```
// Risky Cahayu
// 103112430121

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

int main() {
    List L;
    address P;

    CreateList(L);

    P = alokasi("Danu", "04", 'L', 4.0);
    insertFirst(L, P);
    P = alokasi("Ali", "01", 'L', 3.3);
    insertLast(L, P);
    P = alokasi("Cindi", "03", 'P', 3.5);
    insertAfter(L, P, L.first);
    P = alokasi("Bobi", "02", 'L', 3.71);
    insertAfter(L, P, L.first);
    P = alokasi("Eli", "05", 'P', 3.4);
    insertLast(L, P);
    P = alokasi("Fahmi", "06", 'L', 3.45);
    insertLast(L, P);
    P = alokasi("Gita", "07", 'P', 3.75);
    insertLast(L, P);
    P = alokasi("Hilmi", "08", 'P', 3.3);
    insertLast(L, P);

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

```
    printInfo(L);

    return 0;
}
```

Screenshots Output

```
PS C:\Users\ASUS\Videos\strukdat\Modul 11\Unguided\soal1> g++ main.cpp circularlist.cpp -o main
PS C:\Users\ASUS\Videos\strukdat\Modul 11\Unguided\soal1> ./main
coba insert first, last, dan after
Nama : Danu
NIM  : 04
L/P  : L
IPK  : 4

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

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

Nama : Ali
NIM  : 01
L/P  : L
IPK  : 3.3

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

Nama : Fahmi
NIM  : 06
L/P  : L
IPK  : 3.5

Nama : Ali
NIM  : 01
L/P  : L
IPK  : 3.3

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
```

Deskripsi:

Pada Unguided 2, praktikum ini menantang mahasiswa untuk

mengimplementasikan operasi-operasi dasar dalam Circular Doubly Linked List, di mana setiap elemen memiliki pointer ke elemen sebelumnya dan setelahnya, serta juga menunjuk kembali ke elemen pertama untuk membentuk struktur sirkular. Praktikum ini mengharuskan mahasiswa untuk memahami cara menyisipkan elemen pada awal (insertFirst), setelah elemen tertentu (insertAfter), dan di akhir list (insertLast), serta cara menghapus elemen dari posisi yang berbeda (deleteFirst, deleteLast, deleteAfter). Fungsi findElm digunakan untuk mencari elemen berdasarkan kriteria tertentu (seperti nim), dan printInfo digunakan untuk menampilkan informasi semua elemen yang ada dalam list. Tujuan dari praktikum ini adalah untuk memberikan pemahaman mendalam tentang penggunaan Circular Doubly Linked List serta implementasi fungsi-fungsi yang memanipulasi elemen dalam struktur tersebut.

E. Kesimpulan

Berdasarkan rangkaian praktikum Unguided 1 dan Unguided 2, dapat disimpulkan bahwa mahasiswa memperoleh pemahaman yang komprehensif mengenai implementasi berbagai jenis linked list, khususnya Multi Linked List, Multilevel Linked List, dan Circular Doubly Linked List. Melalui praktikum ini, mahasiswa tidak hanya mempelajari konsep teoretis tentang hubungan antar node dan penggunaan pointer next serta prev, tetapi juga mampu mengimplementasikan operasi dasar seperti penambahan, pencarian, pembaruan, penampilan, dan penghapusan elemen secara tepat. Selain itu, praktikum ini melatih ketelitian dalam menjaga konsistensi hubungan antar node agar struktur data tetap valid. Dengan demikian, praktikum Unguided ini efektif dalam meningkatkan kemampuan mahasiswa dalam memahami dan menerapkan struktur data linked list secara terstruktur dan sistematis dalam pemrograman.

F. Referensi

Modul Struktur Data – Modul 13 Multi Linked List, Telkom University.

"Data Structures and Algorithms in C++" by Adam Drozdek.

GeeksForGeeks – Multi Linked List.

TutorialsPoint – Linked List Traversal.