LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL 13
MULTI LINKED LIST

Disusun Oleh :
NAMA : AGUNG RAMADHAN
NIM : 103112430060

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Multi Linked List merupakan pengembangan dari struktur data linked list yang memungkinkan setiap node memiliki lebih dari satu pointer. Pointer-pointer tersebut digunakan untuk menghubungkan satu node dengan beberapa node lain, sehingga struktur ini mampu merepresentasikan hubungan data yang lebih kompleks dibandingkan single linked list. Dengan adanya lebih dari satu link dalam sebuah node, multi linked list dapat digunakan untuk memodelkan relasi antar data yang bersifat dinamis dan saling berhubungan.

Secara umum, multi linked list terdiri dari node utama dan node pendukung atau node anak yang saling terhubung melalui pointer relasi. Struktur ini sering digunakan untuk menggambarkan hubungan one-to-many maupun many-to-many, di mana satu data dapat memiliki keterkaitan dengan banyak data lain. Oleh karena itu, multi linked list banyak diterapkan pada sistem yang membutuhkan pengelolaan relasi data, seperti sistem akademik, manajemen data, dan representasi hubungan antar objek secara efisien.

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

Guided 1

Kode 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;
```

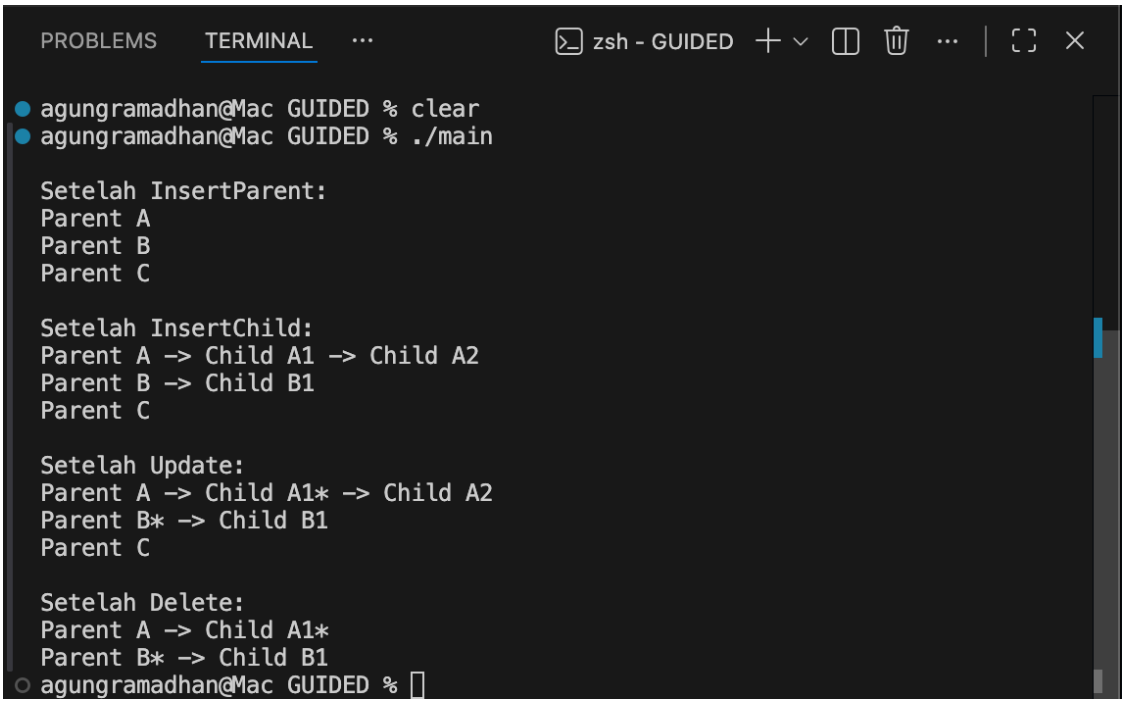
```
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}

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;
}
```

Screenshot:



Deskripsi:

Program ini mengimplementasikan struktur data multi linked list menggunakan bahasa C++, di mana setiap node parent dapat memiliki beberapa node child. Struktur parent dan child dibuat dalam bentuk double linked list, sehingga setiap node memiliki pointer `next` dan `prev`. Program menyediakan operasi dasar seperti penambahan data parent dan child, penampilan seluruh isi list, pembaruan data, serta penghapusan node parent beserta seluruh child-nya atau penghapusan child tertentu. Melalui fungsi-fungsi tersebut, program mampu mengelola hubungan one-to-many secara dinamis, sehingga cocok digunakan untuk merepresentasikan relasi data yang saling terhubung dalam sebuah sistem.

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

Unguided 1

Kode circularlist.h

```
// AGUNG RAMADHAN
// 103112430060

#ifndef CIRCULARLIST_H
#define CIRCULARLIST_H

#include <iostream>
using namespace std;

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
```

Kode circularlist.cpp

```
// AGUNG RAMADHAN
// 103112430060

#include "circularlist.h"

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

address alokasi(infotype x)
{
    address P = new elmList;
    P->info = x;
    P->next = NULL;
    return P;
}

void dealokasi(address &P)
{
    delete P;
    P = NULL;
}

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

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

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

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

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

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

address findElm(List L, infotype x)
{
    if (L.first == NULL)
        return NULL;

    address P = L.first;
    do
    {
        if (P->info.nim == x.nim)
        {
            return P;
        }
        P = P->next;
    } while (P != L.first);

    return NULL;
}

void printInfo(List L)
{
    if (L.first == NULL)
        return;
```

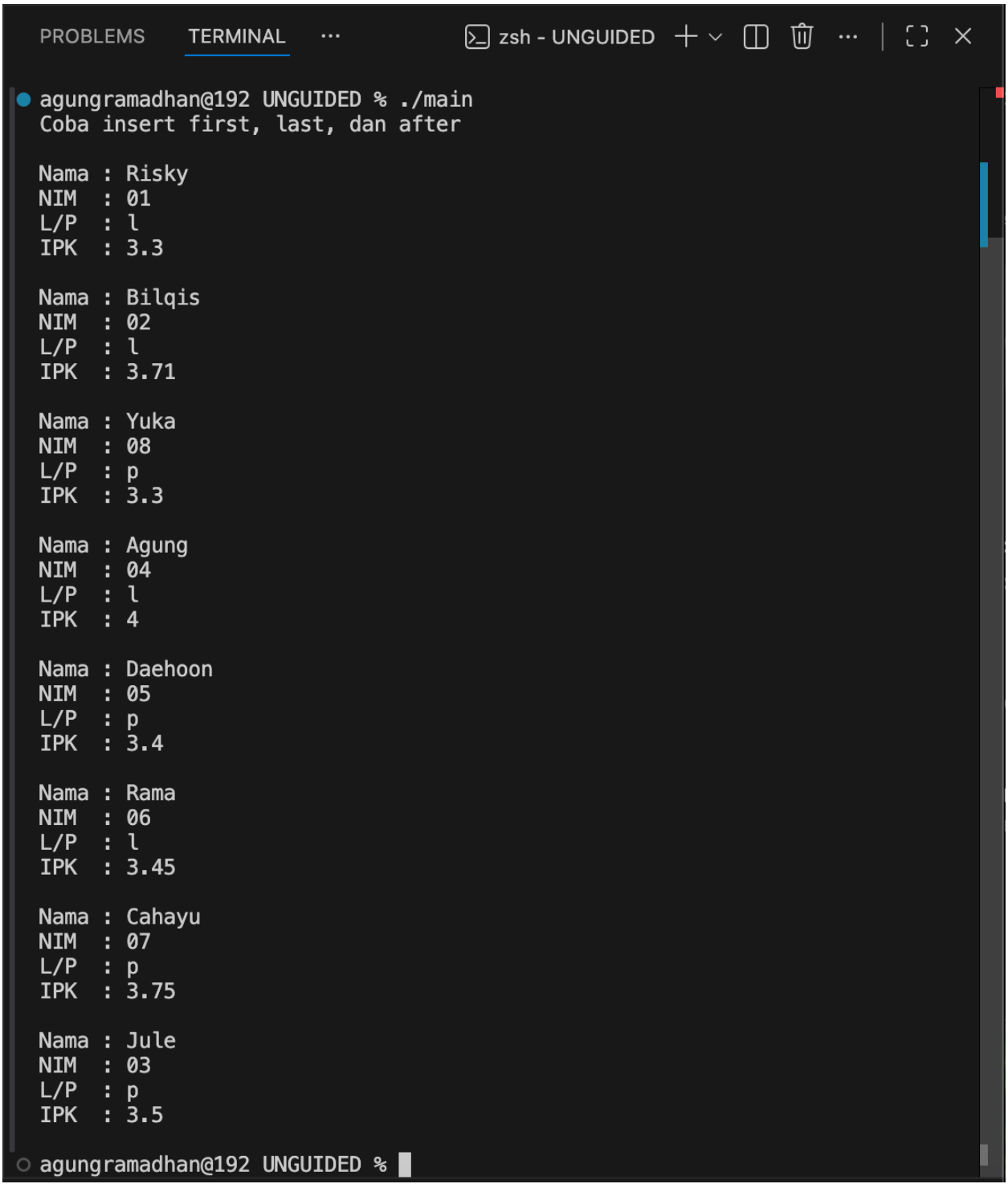
```
    address P = L.first;
    do
    {
        cout << "Nama : " << P->info.nama << endl;
        cout << "NIM  : " << P->info.nim << endl;
        cout << "L/P  : " << P->info.jenis_kelamin << endl;
        cout << "IPK  : " << P->info.ipk << endl
             << endl;
        P = P->next;
    }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

Kode main.cpp

```
// AGUNG RAMADHAN
// 103112430060

#include "circularlist.h"

int main()
{
    List L;
    address P1, P2;
    infotype x;

    createList(L);

    cout << "Coba insert first, last, dan after\n"
         << endl;

    P1 = createData("Agung", "04", 'l', 4.0);
    insertFirst(L, P1);

    P1 = createData("Rama", "06", 'l', 3.45);
    insertLast(L, P1);

    P1 = createData("Bilqis", "02", 'l', 3.71);
    insertFirst(L, P1);
```

```
    P1 = createData("Risky", "01", 'l', 3.3);
    insertFirst(L, P1);

    P1 = createData("Cahayu", "07", 'p', 3.75);
    insertLast(L, P1);

    x.nim = "07";
    P1 = findElm(L, x);
    P2 = createData("Jule", "03", 'p', 3.5);
    insertAfter(L, P1, P2);

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

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

    printInfo(L);

    return 0;
}
```

Screenshots Output



Deskripsi:

Program ini merupakan implementasi Abstract Data Type (ADT) Circular Linked List menggunakan bahasa C++, yang digunakan untuk menyimpan dan mengelola data mahasiswa. Struktur circular linked list ditandai dengan node terakhir yang selalu menunjuk kembali ke node pertama, sehingga membentuk hubungan melingkar. Program memisahkan definisi struktur data dan operasi ke dalam file header (`circularlist.h`) dan file implementasi (`circularlist.cpp`), sementara `main.cpp` berfungsi sebagai pengujian

program. Operasi dasar yang disediakan meliputi pembuatan list, penambahan data di awal, akhir, dan setelah node tertentu, pencarian data berdasarkan NIM, penghapusan node, serta penampilan seluruh isi list. Dengan struktur ini, pengelolaan data mahasiswa dapat dilakukan secara dinamis dan efisien menggunakan konsep pointer.

D. Kesimpulan

Kesimpulannya, implementasi ADT Circular Linked List pada program ini berhasil digunakan untuk mengelola data mahasiswa secara dinamis dengan memanfaatkan konsep keterhubungan melingkar antar node. Struktur ini memungkinkan proses penambahan, pencarian, penghapusan, dan penampilan data dilakukan secara efisien tanpa bergantung pada ukuran tetap seperti pada array. Dengan pemisahan antara definisi struktur, implementasi fungsi, dan program utama, kode menjadi lebih terstruktur, mudah dipahami, serta mendukung prinsip modularitas dalam pemrograman.

E. Referensi

Sundell, H., & Tsigas, P. (2008). *Lock-Free Deques and Doubly Linked Lists*. *Journal of Parallel and Distributed Computing*, 68(7), 1008-1020.

Ng, D. T. H., & Oommen, B. J. (1992). *A Short Note on Doubly-Linked List Reorganizing Heuristics*. *The Computer Journal*, 35(5), 533-535.

Khan, M. S., Ware, A., Habib, U., Khalid, M. J., & Bahoo, N. (2022). *Skeleton Based Human Action Recognition Using Doubly Linked List*. *International Journal of Computer Trends and Technology*, 70(2), 18-21.

Lohmann, S., & Tutsch, D. (2024). *The Doubly Linked Tree of Singly Linked Rings: Providing Hard Real-Time Database Operations on an FPGA*. *Computers*, 13(1), 8.