LAPORAN PRAKTIKUM

STRUKTUR DATA

MODUL XIII

MULTI LINKED LIST

Disusun Oleh :

Nama : Izzah Minkhotun Fannisa

NIM : 103112400198

Dosen

FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Struktur data adalah metode untuk menyusun dan menyimpan informasi di dalam sistem komputer agar bisa diambil dan dikelola dengan efektif. Memilih struktur data yang sesuai sangat memengaruhi performa aplikasi, khususnya terkait konsumsi memori dan waktu pemrosesan informasi.

Linked list adalah salah satu struktur data dinamis yang terbentuk dari kumpulan node, di mana masing-masing node menyimpan data serta pointer yang mengarah ke node berikutnya. Berbeda dari array, *linked list* tidak membutuhkan alokasi memori berurutan, sehingga lebih adaptif dalam hal penambahan atau penghapusan elemen.

Circular linked list merupakan varian dari *linked list* biasa, di mana pointer *next* pada node akhir tidak kosong, melainkan kembali ke node awal. Akibatnya, struktur data ini membentuk sebuah siklus. Karakteristik utama *circular linked list* adalah absennya batasan akhir yang pasti, sehingga traversal bisa dilakukan berulang dari titik mana saja.

Pada *circular singly linked list*, setiap node hanya dilengkapi satu pointer, yakni *next*, yang merujuk ke node selanjutnya. Struktur ini sering diterapkan dalam skenario yang mengharuskan proses berulang terus-menerus, seperti pengelolaan antrean, penjadwalan tugas, dan mekanisme rotasi informasi.

Dalam penerapannya, pointer memainkan peran krusial untuk menyambung antar node, sementara alokasi memori dinamis (menggunakan *new* dan *delete)* memungkinkan ukuran struktur data menyesuaikan keperluan aplikasi. Gabungan elemen-elemen ini membuat *circular linked list* menjadi struktur data yang efektif dan lentur untuk mengatur data secara sistematis.

B. Guide

1. Guide 1

a. Source Code

guide.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;
}
```

b. Screenshot 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
```

c. Deskripsi Program

Program ini adalah implementasi dari Multilevel Linked List dalam bahasa pemrograman C++, di mana setiap ParentNode bisa memiliki beberapa ChildNode. Struktur datanya menggunakan double linked list baik untuk level parent maupun child, sehingga setiap node memiliki pointer ke node sebelum dan sesudahnya.

Program ini terdiri dari beberapa bagian utama, yaitu:

a) Library dan Namespace

Program menggunakan library <iostream> untuk input–output dan <string> untuk pengolahan data bertipe string, serta using namespace std.

b) Struktur Data

- ChildNode
  Digunakan untuk menyimpan data child beserta pointer ke node sebelumnya (prev) dan sesudahnya (next).
- ParentNode
  Digunakan untuk menyimpan data parent, pointer ke child pertama (childHead), serta pointer ke parent sebelumnya dan sesudahnya.

c) Fungsi Pembuatan Node

- createParent() untuk membuat node parent baru.
- createChild() untuk membuat node child baru.

d) Fungsi Penambahan Data

- insertParent() untuk menambahkan parent ke dalam linked list.
- insertClild() untuk menambahkan child ke parent tertentu.

e) Fungsi Penampilan Data

- printAll() untuk menampilkan seluruh parent beserta child-nya.

f) Fungsi Update Data
   - updateParent() untuk mengubah data pada parent.
   - updateChild() untuk mengubah data pada child.

g) Fungsi Penghapusan Data
   - deleteChild() untuk menghapus child dari parent tertentu.
   - deleteParent() untuk menghapus parent beserta seluruh child yang dimilikinya.

h) Fungsi main()

   Berisi pengujian program dengan melakukan proses insert, update, delete, dan menampilkan hasil setiap tahap.

C. Unguide

1. Unguide 1

a. Source Code

circularlist.h

```
#ifndef CIRCULARLIST_H
#define CIRCULARLIST_H

#include <string>
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;
};

struct List {
    address First;
};

void createList(List &L);
address alokasi(infotype x);
void dealokasi(address &P);
```

```
void insertFirst(List &L, address P);
void insertLast(List &L, address P);
void insertAfter(List &L, address Prec, 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 = Nil;
    return P;
}

void dealokasi(address &P) {
    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 insertLast(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;
        last->next = P;
        P->next = L.First;
    }
}

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

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 << "L/P  : " << P->info.jenis_kelamin << endl;
        cout << "IPK  : " << P->info.ipk << endl << endl;
        P = P->next;
    } while (P != L.First);
}
```

main.cpp

```
#include <iostream>
#include "circularlist.h"
#include "circularlist.cpp"
```

```cpp
using namespace std;

address createData(string nama, string nim, char jk, float ipk)
{
    infotype x;
    x.nama = nama;
    x.nim = nim;
    x.jenis_kelamin = jk;
    x.ipk = ipk;
    return alokasi(x);
}

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

    createList(L);

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

    insertFirst(L, createData("Danu", "04", 'l', 4.0));
    insertLast(L, createData("Fahmi", "06", 'l', 3.45));
    insertFirst(L, createData("Bobi", "02", 'l', 3.71));
    insertFirst(L, createData("Ali", "01", 'l', 3.3));
    insertLast(L, createData("Gita", "07", 'p', 3.75));

    x.nim = "02";
    P1 = findElm(L, x);
    P2 = createData("Cindi", "03", 'p', 3.5);
    insertAfter(L, P1, P2);

    x.nim = "07";
    P1 = findElm(L, x);
    P2 = createData("Hilmi", "08", 'l', 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;
}
```

b. Screenshot Output

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

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

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

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

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

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

Nama : Eli
NIM  : 05
```

```
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  : l
IPK  : 3.3
```

c. Deskripsi Program

Program ini adalah implementasi dari Circular Singly Linked List dalam bahasa pemrograman C++, dengan fokus pada studi kasus data mahasiswa. Dalam struktur circular linked list, node akhir akan mengarah kembali ke node awal, sehingga menciptakan bentuk lingkaran tanpa adanya nilai NULL pada pointer next.

Fitur-fitur yang disediakan dalam program ini mencakup:

a) createList untuk memulai list kosong.
b) alokasi dan dealokasi untuk menangani pembuatan dan penghapusan memori node.
c) insertFirst untuk menambah data mahasiswa di bagian depan list.
d) insertLast untuk menambah data mahasiswa di bagian belakang list.
e) insertAfter untuk menambah data mahasiswa setelah node spesifik.
f) findElm untuk mencari data mahasiswa berdasarkan NIM.
g) printInfo untuk menampilkan semua data mahasiswa dalam list.

Pada bagian main(), program melakukan uji coba dengan memasukkan beberapa data mahasiswa melalui berbagai cara penyisipan, lalu menunjukkan seluruh isi circular linked list di layar. Program ini dirancang untuk memperkuat pemahaman tentang konsep pointer, alokasi memori dinamis, serta aplikasi circular linked list dalam pengaturan data terorganisir.

D. Kesimpulan

Dari hasil penerapan dan uji coba program, kita dapat menyimpulkan bahwa *circular singly linked list* efektif dalam menangani data mahasiswa secara fleksibel. Program ini berhasil mengimplementasikan berbagai fungsi penambahan data, seperti

memasukkan elemen di bagian depan, belakang, atau setelah simpul spesifik, plus kemampuan untuk menampilkan semua isi daftar dalam bentuk siklus.

Dengan program semacam ini, wawasan tentang konsep pointer, pengalokasian memori yang adaptif, serta cara kerja daftar tertaut melingkar bisa semakin mendalam. Struktur data ini terbukti berguna untuk memproses informasi yang berulang-ulang tanpa akhir yang pasti. Akibatnya*, circular linked list* menjadi pilihan ideal bagi pengembangan perangkat lunak yang memerlukan penanganan data yang berkelanjutan dan sistematis.

E. Referensi

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms (3rd ed.)*. MIT Press.

GeeksforGeeks. (n.d.). *Circular linked list*. GeeksforGeeks. https://www.geeksforgeeks.org/circular-linked-list/

Sahni, S. (2005). *Data structures, algorithms, and applications in C++*. McGraw-Hill.

Weiss, M. A. (2014). *Data structures and algorithm analysis in C++ (4th ed.).* Pearson Education.