LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL XI
MULTI LINKED LIST

Disusun Oleh :
NAMA : Sinta Sintiani
NIM : 103112430047

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Multilinked list (multilist) adalah struktur data berbasis linked list yang digunakan untuk merepresentasikan data yang memiliki hubungan kompleks atau bersifat many-to-many, di mana setiap node dapat memiliki lebih dari satu pointer untuk menghubungkan diri dengan list lain. Dalam multilist, biasanya terdapat node induk, node anak, dan node relasi yang menghubungkan keduanya sehingga data dapat disimpan secara terstruktur tanpa duplikasi. Struktur ini memungkinkan penyimpanan data hierarki atau jaringan, seperti hubungan mahasiswa–mata kuliah, kategori–produk, atau dosen–kelas, dengan operasi dasar seperti penambahan node, penghapusan node, dan traversal data yang lebih fleksibel namun lebih kompleks dibanding linked list biasa.

Guided 1

```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;
}
```

Screenshots Output

```
PS D:\SEM 3\STRUKTUR DATA\mod 11> cd "d:\SEM 3\STRUKTUR DATA\Modul 11\" ; if ($?) { g++ tempCode
}

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
```

Deskripsi:

Program di atas merupakan implementasi sederhana dengan parent dan child, di mana setiap parent memiliki daftar child berbentuk doubly linked list. Program menyediakan fungsi untuk membuat node parent dan child, menambahkan parent ke list utama, menambahkan child ke parent tertentu, menampilkan seluruh data, memperbarui informasi parent maupun child, serta menghapus child atau parent beserta seluruh child-nya. Melalui fungsi-fungsi tersebut, program dapat mengelola hubungan hierarki antara

parent dan child secara dinamis menggunakan pointer.

Unguided 1

multilist.h

```
#ifndef MULTILIST_H_INCLUDED
#define MULTILIST_H_INCLUDED

#include <iostream>
using namespace std;

#define Nil NULL

typedef bool boolean;
typedef int infotypeinduk;
typedef int infotypeanak;

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

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 X);
address_anak alokasiAnak(infotypeanak X);
void dealokasi(address P);
void dealokasiAnak(address_anak P);
address findElm(listinduk L, infotypeinduk X);
address_anak findElm(listanak L, infotypeanak X);
void insertFirst(listinduk &L, address P);
void insertLast(listinduk &L, address P);
void insertAfter(listinduk &L, address Prec, address P);
void insertFirstAnak(listanak &L, address_anak P);
void insertLastAnak(listanak &L, address_anak P);
void insertAfterAnak(listanak &L, address_anak P, address_anak Prec);
void delFirst(listinduk &L, address &P);
void delLast(listinduk &L, address &P);
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);
void printInfoAnak(listanak L);

#endif
```

multilist.cpp

```
#include <iostream>
#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 X) {
    address P = new elemen_list_induk;

    P->info = X;
    CreateListAnak(P->lanak);
    P->next = Nil;
    P->prev = Nil;
    return P;
}

address_anak alokasiAnak(infotypeanak X) {
    address_anak P = new elemen_list_anak;

    P->info = X;
    P->next = Nil;
    P->prev = Nil;
    return P;
}

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) {
        if (P->info == X)
            return P;
        P = P->next;
    }
    return Nil;
}

address_anak findElm(listanak L, infotypeanak X) {
    address_anak P = L.first;

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

```
    return Nil;
}

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

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

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

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

void insertLastAnak(listanak &L, address_anak P) {
```

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

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

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

void delP(listinduk &L, infotypeinduk X) {
    address P = findElm(L, X);

    if (P != Nil) {
        address_anak Q = P->lanak.first;
        while (Q != Nil) {
            address_anak temp = Q;
            Q = Q->next;
            dealokasiAnak(temp);
        }

        if (P == L.first) {
            delFirst(L, P);
        } else if (P == L.last) {
            delLast(L, P);
        } else {
```

```
            P->prev->next = P->next;
            P->next->prev = P->prev;
        }
        dealokasi(P);
    }
}

void delLastAnak(listanak &L, address_anak &P) {
    P = L.last;

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

void printInfo(listinduk L) {
    address P = L.first;

    while (P != Nil) {
        cout << "Induk: " << P->info << endl;
        cout << "  Anak: ";
        address_anak Q = P->lanak.first;

        if (Q == Nil)
            cout << "(tidak ada anak)";
        else {
            while (Q != Nil) {
                cout << Q->info << " ";
                Q = Q->next;
            }
        }
        cout << endl;
        P = P->next;
    }
}
```

Main.cpp

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

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

    cout << " INSERT INDUK " << endl;
    insertLast(L, alokasi(10));
    insertLast(L, alokasi(20));
    insertLast(L, alokasi(30));

    cout << "\n INSERT ANAK PADA INDUK " << endl;
    address induk20 = findElm(L, 20);

    insertLastAnak(induk20->lanak, alokasiAnak(101));
    insertLastAnak(induk20->lanak, alokasiAnak(102));
    insertLastAnak(induk20->lanak, alokasiAnak(103));

    cout << "\n INSERT ANAK PADA INDUK " << endl;
    address induk10 = findElm(L, 10);

    insertLastAnak(induk10->lanak, alokasiAnak(201));
    insertLastAnak(induk10->lanak, alokasiAnak(202));

    cout << "\n DATA MULTILIST " << endl;
    printInfo(L);

    cout << "\n DELETE INDUK " << endl;
    delP(L, 20);

    printInfo(L);

    return 0;
}
}
```

Screenshots Output

```
PS D:\SEM 3\STRUKTUR DATA\Modul 11> cd "d:\SEM 3\STRUKTUR DATA\mod 11\" ; if ($?) { g++ te
}
 INSERT INDUK

 INSERT ANAK PADA INDUK

 INSERT ANAK PADA INDUK

 DATA MULTILIST
Induk: 10
  Anak: 201 202
Induk: 20
  Anak: 101 102 103
Induk: 30
  Anak: (tidak ada anak)

 DELETE INDUK
Induk: 10
  Anak: 201 202
Induk: 30
  Anak: (tidak ada anak)
```

Deskripsi:

Program di atas adalah implementasi yang terdiri dari list induk dan list anak menggunakan doubly linked list. Setiap elemen induk memiliki list anak tersendiri, sehingga satu induk dapat menyimpan banyak elemen anak. Program menyediakan fungsi lengkap seperti membuat list, mengalokasikan node induk dan anak, menambah elemen di awal/akhir, mencari elemen, menghapus induk beserta seluruh anaknya, serta menampilkan seluruh data multilist. Pada fungsi main, program menambahkan beberapa induk, menambahkan anak pada induk tertentu, menampilkan struktur multilist, lalu menghapus salah satu induk dan menampilkan hasil akhirnya.

Unguided 2

circularlist.h

```h
#ifndef CIRCULARLIST_H
#define CIRCULARLIST_H

#include <iostream>
#include <string>
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;
    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);

#endif
```

circularlist.cpp

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

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

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

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

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

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

```
}

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

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

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

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) {
        cout << "List kosong\n";
        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;
        cout << endl;
        P = P->next;
    } while (P != L.first);
}
```

main.cpp

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

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);

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

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

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

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

    x.nim = "03";
    P1 = findElm(L, x);
    P2 = createData("Danu", "04", 'l', 4.0);
```

```
    insertAfter(L, P1, P2);

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

    x.nim = "05";
    P1 = findElm(L, x);
    P2 = createData("Fahmi", "06", 'l', 3.45);
    insertAfter(L, P1, P2);

    x.nim = "07";
    P1 = findElm(L, x);
    P2 = createData("Hilmi", "08", 'l', 3.3);
    insertAfter(L, P1, P2);

    printInfo(L);
    return 0;
}
```

Screenshots Output

```
PS D:\SEM 3\STRUKTUR DATA\mod 11> cd "d:\SEM 3\STRUKTUR DATA\mod 11\" ; if (
Nama : Bobi
NIM  : 02
JK   : l
IPK  : 3.71

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

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 : Ali
NIM  : 01
JK   : l
IPK  : 3.3

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

Nama : Hilmi
NIM  : 08
JK   : l
IPK  : 3.3
```

Deskripsi:

Program tersebut merupakan digunakan untuk mengelola data mahasiswa secara dinamis, di mana setiap elemen saling terhubung dua arah dan membentuk lingkaran. Setiap node menyimpan informasi lengkap mahasiswa seperti nama, NIM, jenis kelamin, dan IPK. Program menyediakan fungsi untuk membuat list, menambahkan node di berbagai posisi, mencari data berdasarkan NIM, menghapus elemen, serta menampilkan seluruh isi list. Pada bagian utama, beberapa data mahasiswa dibuat dan disisipkan ke dalam list menggunakan operasi insert, kemudian seluruh isi list ditampilkan sebagai output.

B. Kesimpulan

Melalui praktikum ini, mahasiswa memahami cara mengimplementasikan berbagai jenis struktur data dinamis seperti double linked list, multilist, dan circular double linked list. Praktikum ini melatih kemampuan dalam membuat node, melakukan alokasi dan dealokasi memori, serta mengelola hubungan antar-elemen melalui operasi dasar seperti insert, delete, update, find, dan traversal data. Mahasiswa juga belajar bagaimana satu struktur dapat diperluas, seperti pada multilist yang memiliki hubungan induk–anak, serta circular list yang memungkinkan traversal tanpa ujung karena node terakhir terhubung kembali ke node pertama. Secara keseluruhan, praktikum ini memberikan pemahaman mendalam tentang konsep struktur data dinamis, manajemen memori, serta logika pemrosesan data yang terstruktur dan efisien.

C. Referensi

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

https://www.w3schools.com/dsa/dsa_theory_linkedlists.php