LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL XI
MULTI LINKED LIST

Disusun oleh :

Junadil Muqorobin (103112400281)

Dosen

Fahrudin Mukti Wibowo S.Kom., M.Eng

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Struktur data merupakan fondasi penting dalam pengembangan perangkat lunak, terutama ketika program harus mengelola data dalam jumlah besar secara efisien. Salah satu struktur data yang sering digunakan untuk menangani data yang bersifat dinamis adalah **linked list**. Linked list menyediakan fleksibilitas dalam penambahan, penghapusan, dan pengolahan elemen tanpa harus melakukan pergeseran memori seperti pada array. Mekanisme pointer yang saling terhubung memungkinkan linked list bekerja secara efisien dalam konteks data yang berubah-ubah.

Dalam praktik pengembangan sistem, kebutuhan untuk mengelompokkan data secara hierarkis sering muncul. Untuk itu, digunakan **Multilist**, yaitu struktur data yang memungkinkan satu elemen induk memiliki satu atau lebih elemen anak. Multilist digunakan ketika data harus disajikan dalam bentuk hubungan satu-ke-banyak, misalnya *kategori → item*, *departemen → pegawai*, atau *jenis kendaraan → kendaraan*. Setiap elemen induk dalam multilist memiliki pointer menuju list anak sehingga struktur data menjadi lebih teratur dan mudah dikelola. Menurut penelitian Pratama dan Kurniawan (2021), penggunaan multilist dapat menyederhanakan proses pengorganisasian data bertingkat sekaligus mempercepat akses terhadap kelompok data tertentu.

Selain multilist, bentuk struktur data lain yang banyak digunakan adalah Circular Linked List. Pada circular list, elemen terakhir tidak menunjuk ke nilai null, tetapi kembali ke elemen pertama sehingga membentuk lingkaran. Struktur seperti ini sangat berguna ketika data diproses secara berulang atau siklis,

seperti penjadwalan proses, permainan (game loop), dan antrian yang tidak pernah berhenti. Circular linked list juga memberi efisiensi lebih baik pada operasi traversal karena tidak ada kondisi berhenti akibat nilai null. Hal ini sejalan dengan penelitian Hermawan (2020) yang menunjukkan bahwa circular list mendukung operasi iteratif yang berkesinambungan tanpa overhead pengecekan kondisi akhir daftar.

Implementasi multilist dan circular list pada bahasa C++ melibatkan konsep pointer, dynamic memory (alokasi dan dealokasi), serta pemahaman mendalam mengenai hubungan antar node. Penggunaan prosedur seperti *insert*, *delete*, *find*, dan *traversal* merupakan bagian fundamental dalam manipulasi linked list. Dalam beberapa kasus, pengembangan aplikasi membutuhkan kombinasi antara multilist dan circular list agar struktur data lebih adaptif. Penelitian Wibowo dan Santoso (2022) juga mencatat bahwa struktur data pointer-based seperti linked list mampu meningkatkan fleksibilitas sistem yang menangani perubahan data secara real-time.

Dengan demikian, penguasaan multilist dan circular linked list menjadi penting bagi mahasiswa maupun pengembang perangkat lunak. Selain memperkuat pemahaman mengenai struktur data, kemampuan ini juga menjadi dasar untuk membangun logika program yang lebih kompleks dan efisien. Praktikum ini memberikan pengalaman langsung dalam mengimplementasikan kedua struktur tersebut melalui operasi dasar seperti penyisipan, penghapusan, pencarian, dan pencetakan informasi sehingga mahasiswa mendapatkan gambaran nyata penerapan struktur data dalam kasus dunia nyata.

B. Guided

1. Implementasi Multi Linked List

```
#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;
```

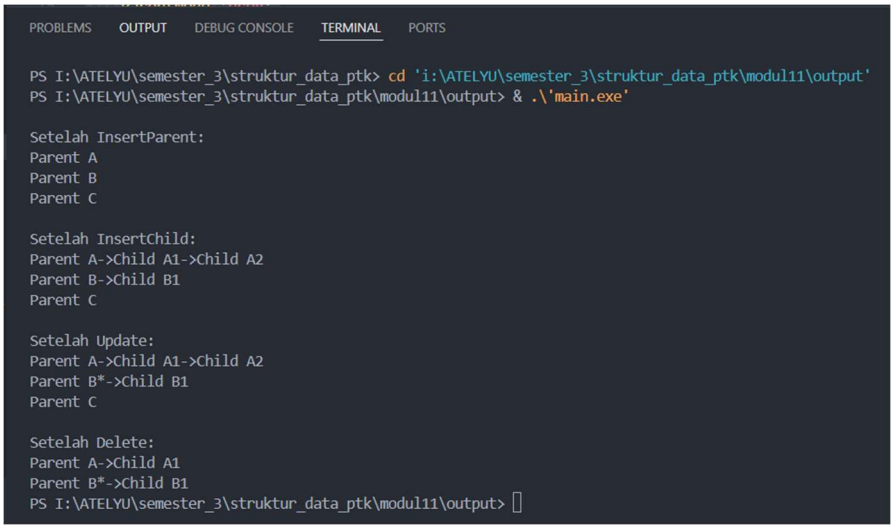
```
        }
        temp->next = newNode;
        newNode->prev = temp;
    }
}

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 output:



Deskripsi:

Program ini mengimplementasikan struktur data Multilist menggunakan konsep parent–child

berbasis *doubly linked list*. Program berjalan dengan langkah kerja sebagai berikut:

- Inisialisasi Struct
  - **ParentNode**, berisi info, pointer childHead, serta pointer next dan prev untuk membentuk *doubly linked list* parent.
  - **ChildNode**, berisi info, next, dan prev sebagai daftar anak yang berada di bawah parent tertentu.
- Menambahkan Parent dan Child
  - insertParent() menambahkan parent baru pada posisi paling akhir dalam list induk. Jika list masih kosong, parent pertama menjadi head.
  - insertChild() mencari parent berdasarkan parentInfo. Jika ditemukan, child baru ditambahkan pada bagian akhir list anak milik parent tersebut.
- Update Parent dan Child
  - updateParent() mencari parent tertentu dan mengganti nilai info-nya dengan data baru.
  - updateChild() bekerja dengan cara mencari parent terlebih dahulu, kemudian menelusuri list anak hingga menemukan child yang akan diperbarui.
- Menghapus Parent dan Child
  - deleteChild() menghapus child tertentu dari parent tertentu. Fungsi ini mengatur ulang pointer next dan prev, baik ketika child berada di awal, tengah, atau akhir.

- deleteParent() menghapus seluruh child yang dimiliki parent terlebih dahulu, kemudian menghapus node parent dari list induk.

- Menampilkan Hasil pada main

  Pada bagian main menjalankan seluruh fungsi dengan alur sebagai berikut:

  - Membuat list induk kosong.
  - Menambahkan tiga parent: *Parent A*, *Parent B*, dan *Parent C*.
  - Menambahkan child pada beberapa parent: A → A1, A2 dan B → B1.
  - Melakukan update data pada parent dan child tertentu.
  - Menghapus *Child A2* dan menghapus seluruh node milik *Parent C*.
  - Menampilkan isi list setelah setiap operasi menggunakan fungsi printAll().

C. Unguided

1. Implementasi ADT Multi Linked List

Source code multilist.h

```
#ifndef MULTILIST_H_INCLUDED
#define MULTILIST_H_INCLUDED
#define Nil NULL

typedef bool boolean;
typedef int infotypeanak;
typedef int infotypeinduk;
typedef struct elemen_list_induk *address;
typedef struct elemen_list_anak *address_anak;

/* ----- Struct Anak ----- */
struct elemen_list_anak{
    infotypeanak info;
```

```
    address_anak next;
    address_anak prev;
};

/* List anak */
struct listanak {
    address_anak first;
    address_anak last;
};

/* ----- Struct Induk ----- */
struct elemen_list_induk{
    infotypeinduk info;
    listanak lanak;       // list anak milik elemen induk
    address next;
    address prev;
};

/* List induk */
struct listinduk {
    address first;
    address last;
};

/********* pengecekan apakah list kosong **********/
boolean ListEmpty(listinduk L);
/* true jika list induk kosong */

boolean ListEmptyAnak(listanak L);
/* true jika list anak kosong */

/********* pembuatan list kosong *********/
void CreateList(listinduk &L);
void CreateListAnak(listanak &L);

/********* manajemen memori *********/
address alokasi(infotypeinduk X);
address_anak alokasiAnak(infotypeanak X);

void dealokasi(address P);
void dealokasiAnak(address_anak P);

/********* pencarian elemen *********/
address findElm(listinduk L, infotypeinduk X);
address_anak findElm(listanak L, infotypeanak X);
```

```
boolean fFindElm(listinduk L, address P);
boolean fFindElmanak(listanak L, address_anak P);

address findBefore(listinduk L, address P);
address_anak findBeforeAnak(listanak L, infotypeinduk X,
address_anak P);

/********* penambahan elemen *********/
void insertFirst(listinduk &L, address P);
void insertAfter(listinduk &L, address P, address Prec);
void insertLast(listinduk &L, address P);

void insertFirstAnak(listanak &L, address_anak P);
void insertAfterAnak(listanak &L, address_anak P, address_anak
Prec);
void insertLastAnak(listanak &L, address_anak P);

/********* penghapusan elemen *********/
void delFirst(listinduk &L, address &P);
void delLast(listinduk &L, address &P);
void delAfter(listinduk &L, address &P, address Prec);
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);

/********* proses semua elemen *********/
void printInfo(listinduk L);
int nbList(listinduk L);

void printInfoAnak(listanak L);
int nbListAnak(listanak L);

void delAll(listinduk &L);

#endif
```

Source code multilist.cpp

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

```
typedef bool boolean;
typedef struct listinduk list;
#include "multilist.h"

boolean ListEmpty(listinduk L){
    return (L.first == Nil);
}
boolean ListEmptyAnak(listanak L){
    return (L.first == Nil);
}

/* pembuatan list kosong */
void CreateList(listinduk &L){
    L.first = Nil;
    L.last = Nil;
}
void CreateListAnak(listanak &L){
    L.first = Nil;
    L.last = Nil;
}

address alokasi(infotypeinduk P){
    address Pnew = new elemen_list_induk;
    if(Pnew != Nil){
        Pnew->info = P;

        CreateListAnak(Pnew->lanak);
        Pnew->next = Nil;
        Pnew->prev = Nil;
    }
    return Pnew;
}
address_anak alokasiAnak(infotypeanak P){
    address_anak Pnew = new elemen_list_anak;
    if(Pnew != Nil){
        Pnew->info = P;
        Pnew->next = Nil;
        Pnew->prev = Nil;
    }
    return Pnew;
}
void dealokasi(address P){
    delete P;
}
void dealokasiAnak(address_anak P){
```

```
    delete P;
}

/* pencarian elemen */
address findElm(listinduk L, infotypeinduk X){
    if(ListEmpty(L)) return Nil;
    address P = L.first;
    while(P != Nil){
        if(P->info == X) return P;
        P = P->next;
    }
    return Nil;
}
address_anak findElm(listanak Lanak, infotypeanak X){
    if(ListEmptyAnak(Lanak)) return Nil;
    address_anak P = Lanak.first;
    while(P != Nil){
        if(P->info == X) return P;
        P = P->next;
    }
    return Nil;
}
boolean fFindElm(listinduk L, address P){
    if(ListEmpty(L)) return false;
    address Q = L.first;
    while(Q != Nil){
        if(Q == P) return true;
        Q = Q->next;
    }
    return false;
}
boolean fFindElmanak(listanak Lanak, address_anak P){
    if(ListEmptyAnak(Lanak)) return false;
    address_anak Q = Lanak.first;
    while(Q != Nil){
        if(Q == P) return true;
        Q = Q->next;
    }
    return false;
}
address findBefore(listinduk L, address P){
    if(ListEmpty(L) || P == L.first) return Nil;
    address Q = L.first;
    while(Q != Nil && Q->next != P){
        Q = Q->next;
```

```
    }
    if(Q == Nil) return Nil;
    return Q;
}
address_anak findBeforeAnak(listanak Lanak, infotypeinduk
/*X*/, address_anak P){
    if(ListEmptyAnak(Lanak) || P == Lanak.first) return Nil;
    address_anak Q = Lanak.first;
    while(Q != Nil && Q->next != P){
        Q = Q->next;
    }
    if(Q == Nil) return Nil;
    return Q;
}

// tambah elemen
void insertFirst(listinduk &L, address P){
    if(ListEmpty(L)){
        L.first = P;
        L.last = P;
        P->next = Nil;
        P->prev = Nil;
    } else {
        P->next = L.first;
        P->prev = Nil;
        L.first->prev = P;
        L.first = P;
    }
}
void insertAfter(listinduk &L, address P, address Prec){
    if(Prec == Nil) return;
    if(Prec == L.last){
        insertLast(L, P);
        return;
    }
    P->next = Prec->next;
    P->prev = Prec;
    Prec->next->prev = P;
    Prec->next = P;
}
void insertLast(listinduk &L, address P){
    if(ListEmpty(L)){
        insertFirst(L, P);
    } else {
        L.last->next = P;
```

```
        P->prev = L.last;
        P->next = Nil;
        L.last = P;
    }
}

/* penambahan elemen (anak) */
void insertFirstAnak(listanak &L, address_anak P){
    if(ListEmptyAnak(L)){
        L.first = P;
        L.last = P;
        P->next = Nil;
        P->prev = Nil;
    } else {
        P->next = L.first;
        P->prev = Nil;
        L.first->prev = P;
        L.first = P;
    }
}
void insertAfterAnak(listanak &L, address_anak P, address_anak
Prec){
    if(Prec == Nil) return;
    if(Prec == L.last){
        insertLastAnak(L, P);
        return;
    }
    P->next = Prec->next;
    P->prev = Prec;
    Prec->next->prev = P;
    Prec->next = P;
}
void insertLastAnak(listanak &L, address_anak P){
    if(ListEmptyAnak(L)){
        insertFirstAnak(L, P);
    } else {
        L.last->next = P;
        P->prev = L.last;
        P->next = Nil;
        L.last = P;
    }
}

/* penghapusan elemen (induk) */
void delFirst(listinduk &L, address &P){
```

```
    if(ListEmpty(L)) { P = Nil; return; }
    P = L.first;
    if(L.first == L.last){
        L.first = Nil;
        L.last = Nil;
    } else {
        L.first = P->next;
        L.first->prev = Nil;
    }
    P->next = Nil;
    P->prev = Nil;
}
void delLast(listinduk &L, address &P){
    if(ListEmpty(L)) { P = Nil; return; }
    P = L.last;
    if(L.first == L.last){
        L.first = Nil;
        L.last = Nil;
    } else {
        L.last = P->prev;
        L.last->next = Nil;
    }
    P->next = Nil;
    P->prev = Nil;
}
void delAfter(listinduk &L, address &P, address Prec){
    if(Prec == Nil || Prec->next == Nil) { P = Nil; return; }
    P = Prec->next;
    Prec->next = P->next;
    if(P->next != Nil) P->next->prev = Prec;
    else L.last = Prec;
    P->next = Nil;
    P->prev = Nil;
}
void delP (listinduk &L, infotypeinduk X){
    if(ListEmpty(L)) return;
    address P = findElm(L, X);
    if(P == Nil) return;
    // hapus semua anak pada P terlebih dahulu
    address_anak c = P->lanak.first;
    while(c != Nil){
        address_anak temp = c;
        c = c->next;
        dealokasiAnak(temp);
    }
```

```
    if(P == L.first){
        address tmp;
        delFirst(L, tmp);
        dealokasi(P);
    } else if(P == L.last){
        address tmp;
        delLast(L, tmp);
        dealokasi(P);
    } else {
        P->prev->next = P->next;
        P->next->prev = P->prev;
        dealokasi(P);
    }
}

/* penghapusan elemen (anak) */
void delFirstAnak(listanak &L, address_anak &P){
    if(ListEmptyAnak(L)) { P = Nil; return; }
    P = L.first;
    if(L.first == L.last){
        L.first = Nil;
        L.last = Nil;
    } else {
        L.first = P->next;
        L.first->prev = Nil;
    }
    P->next = Nil;
    P->prev = Nil;
}
void delLastAnak(listanak &L, address_anak &P){
    if(ListEmptyAnak(L)) { P = Nil; return; }
    P = L.last;
    if(L.first == L.last){
        L.first = Nil;
        L.last = Nil;
    } else {
        L.last = P->prev;
        L.last->next = Nil;
    }
    P->next = Nil;
    P->prev = Nil;
}
void delAfterAnak(listanak &L, address_anak &P, address_anak
Prec){
    if(Prec == Nil || Prec->next == Nil) { P = Nil; return; }
```

```
    P = Prec->next;
    Prec->next = P->next;
    if(P->next != Nil) P->next->prev = Prec;
    else L.last = Prec;
    P->next = Nil;
    P->prev = Nil;
}
void delPAnak (listanak &L, infotypeanak X){
    if(ListEmptyAnak(L)) return;
    address_anak P = findElm(L, X);
    if(P == Nil) return;
    if(P == L.first){
        address_anak tmp;
        delFirstAnak(L, tmp);
        dealokasiAnak(P);
    } else if(P == L.last){
        address_anak tmp;
        delLastAnak(L, tmp);
        dealokasiAnak(P);
    } else {
        P->prev->next = P->next;
        P->next->prev = P->prev;
        dealokasiAnak(P);
    }
}

/* proses semua elemen list */
void printInfo(list L){
    if(ListEmpty(*((listinduk*)&L))) return;
    address P = L.first;
    while(P != Nil){
        cout << P->info;
        // print anak
        address_anak C = P->lanak.first;
        if(C != Nil) cout << " -> ";
        while(C != Nil){
            cout << C->info;
            if(C->next != Nil) cout << ", ";
            C = C->next;
        }
        cout << endl;
        P = P->next;
    }
}
int nbList(list L){
```

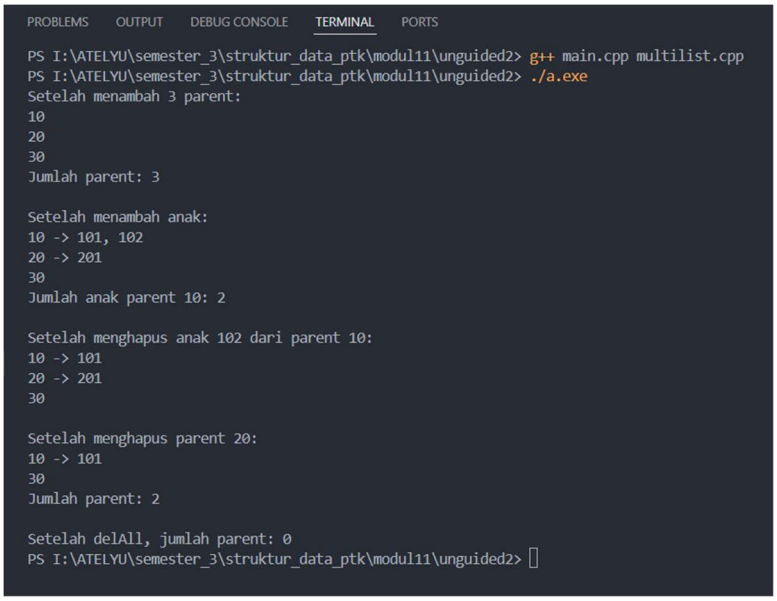
```
    int cnt = 0;
    address P = L.first;
    while(P != Nil){
        cnt++;
        P = P->next;
    }
    return cnt;
}
void printInfoAnak(listanak Lanak){
    address_anak P = Lanak.first;
    while(P != Nil){
        cout << P->info;
        if(P->next != Nil) cout << ", ";
        P = P->next;
    }
    cout << endl;
}
int nbListAnak(listanak Lanak){
    int cnt = 0;
    address_anak P = Lanak.first;
    while(P != Nil){
        cnt++;
        P = P->next;
    }
    return cnt;
}

/* proses terhadap list */
void delAll(listinduk &L){
    while(!ListEmpty(L)){
        address P;
        delFirst(L, P);
        // hapus semua anak
        address_anak c = P->lanak.first;
        while(c != Nil){
            address_anak tmp = c;
            c = c->next;
            dealokasiAnak(tmp);
        }
        dealokasi(P);
    }
}
```

Source code main.cpp

```cpp
#include <iostream>
using namespace std;
typedef bool boolean;
typedef struct listinduk list;
#include "multilist.h"

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

    address p1 = alokasi(10);
    address p2 = alokasi(20);
    address p3 = alokasi(30);

    insertLast(L, p1);
    insertLast(L, p2);
    insertLast(L, p3);

    cout << "Setelah menambah 3 parent:" << endl;
    printInfo(L);
    cout << "Jumlah parent: " << nbList(L) << endl << endl;

    address_anak a1 = alokasiAnak(101);
    address_anak a2 = alokasiAnak(102);
    address_anak a3 = alokasiAnak(201);

    address found = findElm(L, 10);
    if(found != Nil){
        insertLastAnak(found->lanak, a1);
        insertLastAnak(found->lanak, a2);
    }

    found = findElm(L, 20);
    if(found != Nil){
        insertLastAnak(found->lanak, a3);
    }

    cout << "Setelah menambah anak:" << endl;
    printInfo(L);
    cout << "Jumlah anak parent 10: " <<
nbListAnak(findElm(L,10)->lanak) << endl << endl;

    // hapus child 102 dari parent 10
```

```
    delPAnak(findElm(L,10)->lanak, 102);
    cout << "Setelah menghapus anak 102 dari parent 10:" <<
endl;
    printInfo(L);
    cout << endl;

    // hapus parent 20 beserta anaknya
    delP(L, 20);
    cout << "Setelah menghapus parent 20:" << endl;
    printInfo(L);
    cout << "Jumlah parent: " << nbList(L) << endl << endl;

    delAll(L);
    cout << "Setelah delAll, jumlah parent: " << nbList(L) <<
endl;

    return 0;
}
```

Screenshot Output:



Deskripsi:

Program multilist yang dibuat pada praktikum ini merupakan implementasi dari struktur data *Multi Linked List*, yaitu sebuah bentuk pengorganisasian data yang menghubungkan elemen induk dan elemen anak dalam dua level list yang saling terhubung. Setiap elemen induk memiliki daftar

anaknya sendiri, sehingga struktur ini sangat cocok digunakan untuk merepresentasikan data yang bersifat hierarkis. File **multilist.h** berfungsi sebagai tempat deklarasi seluruh struktur, tipe data, dan fungsi yang akan digunakan, mulai dari pembuatan list, manajemen memori, proses pencarian, penambahan elemen, penghapusan elemen, hingga proses menampilkan isi list. Implementasi lengkap dari fungsi-fungsi tersebut terdapat pada **multilist.cpp**, yang mengatur bagaimana node dialokasikan, dihubungkan, dicari, dan dihapus dengan benar baik pada level induk maupun anak. Sementara itu, file **main.cpp** digunakan sebagai bagian pengujian program, di mana berbagai operasi seperti insert, delete, dan pencarian dijalankan untuk memastikan bahwa struktur data multilist dapat bekerja dengan baik. Melalui implementasi ini, mahasiswa dapat memahami bagaimana cara mengelola hubungan data yang lebih kompleks menggunakan pointer dan linked list bertingkat.

2. Implementasi ADT Multi Linked List

Source code circularlist.h

```h
#ifndef CIRCULARLIST_H_INCLUDED
#define CIRCULARLIST_H_INCLUDED

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

struct Mahasiswa {
    string nama;
    string nim;
    char jk;
    float ipk;
};

typedef Mahasiswa infotype;

struct Node {
```

```
    infotype info;
    Node *next;
};

typedef Node* address;

struct CircularList {
    address first;
};

void createList(CircularList &L);

address createData(infotype x);
void dealokasi(address P);

void insertFirst(CircularList &L, address P);
void insertLast(CircularList &L, address P);
void insertAfter(CircularList &L, address Prec, address P);

void deleteFirst(CircularList &L, address &P);
void deleteLast(CircularList &L, address &P);
void deleteAfter(CircularList &L, address Prec, address &P);
void deleteP(CircularList &L, string namaKey);

address findElm(CircularList L, string namaKey);

void printList(CircularList L);
int nbElm(CircularList L);

#endif
```

Source code circularlist.cpp

```
#include "bstree.h"

address alokasi(infotype x) {
address P = new Node;
P->info = x;
P->left = Nil;
P->right = Nil;
return P;
}

void insertNode(address &root, infotype x) {
if (root == Nil) {
```

```
root = alokasi(x);
}
else if (x < root->info) {
insertNode(root->left, x);
}
else if (x > root->info) {
insertNode(root->right, x);
}
else {
}
}

address findNode(infotype x, address root) {
if (root == Nil) return Nil;
if (x == root->info) return root;
if (x < root->info) return findNode(x, root->left);
return findNode(x, root->right);
}

void InOrder(address root) {
if (root != Nil) {
InOrder(root->left);
cout << root->info << " - ";
InOrder(root->right);
}
}

int hitungNode(address root) {
if (root == Nil) return 0;
return 1 + hitungNode(root->left) + hitungNode(root->right);
}

int hitungTotal(address root) {
if (root == Nil) return 0;
return root->info + hitungTotal(root->left) + hitungTotal(root-
>right);
}

int hitungKedalaman(address root) {
if (root == Nil) return 0;
int kiri = hitungKedalaman(root->left);
int kanan = hitungKedalaman(root->right);
return 1 + max(kiri, kanan);
}
```

```
void PreOrder(address root) {
if (root != Nil) {
cout << root->info << " ";
PreOrder(root->left);
PreOrder(root->right);
}
}

void PostOrder(address root) {
if (root != Nil) {
PostOrder(root->left);
PostOrder(root->right);
cout << root->info << " ";
}
}
```

3. Source code main.cpp

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

using namespace std;

int main() {
    CircularList L;
    createList(L);

    infotype mhs1 = {"Ali", "23111234", 'L', 3.5};
    infotype mhs2 = {"Budi", "23111235", 'L', 3.8};
    infotype mhs3 = {"Cici", "23111236", 'P', 3.9};

    address P1 = createData(mhs1);
    address P2 = createData(mhs2);
    address P3 = createData(mhs3);

    cout << "Insert First Ali\n";
    insertFirst(L, P1);

    cout << "Insert Last Budi\n";
    insertLast(L, P2);

    cout << "Insert After Ali → Cici\n";
    insertAfter(L, P1, P3);

    cout << "\n=== Isi List ===\n";
```

```
    printList(L);

    cout << "\nJumlah Elemen: " << nbElm(L) << endl;

    cout << "\nDelete Budi\n";
    deleteP(L, "Budi");

    cout << "\n=== Isi List Setelah Delete ===\n";
    printList(L);

    return 0;
}
```

Screenshot Output:

Deskripsi:

Program circular linked list digunakan untuk mengimplementasikan struktur data *linked list* yang tersusun secara melingkar, di mana elemen terakhir selalu menunjuk kembali ke elemen pertama. Konsep ini memungkinkan proses traversal dilakukan tanpa titik akhir, sehingga cocok digunakan untuk sistem yang membutuhkan perputaran data secara berulang. File **circularlist.h** berisi deklarasi tipe data mahasiswa, node list, serta seluruh fungsi dasar seperti pembuatan list, alokasi node, operasi insert, delete, pencarian, dan fungsi

untuk menampilkan isi list. Implementasi dari setiap fungsi tersebut dituangkan pada **circularlist.cpp**, mulai dari cara menambahkan node di awal, akhir, atau setelah elemen tertentu, hingga proses menghapus node tanpa memutus struktur melingkar yang sudah terbentuk. Selain itu, fungsi pencarian dan perhitungan jumlah elemen juga disediakan untuk mendukung manipulasi data. File **main.cpp** digunakan sebagai media pengujian, di mana beberapa data mahasiswa dimasukkan ke dalam list, kemudian dilakukan uji operasi penyisipan, penghapusan, dan pencarian untuk memastikan bahwa circular linked list berjalan dengan benar. Melalui implementasi ini, pengguna dapat memahami mekanisme kerja struktur data circular list dan bagaimana menjaga keterhubungan node secara konsisten selama operasi berlangsung.

D. Kesimpulan

Kesimpulannya, praktikum ini membantu memahami bagaimana struktur data multilist dan circular linked list bekerja dalam mengelola data yang saling terhubung maupun bersifat siklis. Melalui proses menambah, menghapus, dan menelusuri elemen, terlihat bahwa penggunaan pointer memberikan fleksibilitas yang tidak dimiliki struktur data statis. Penerapan kedua struktur ini membuat pengelolaan data menjadi lebih terstruktur, efisien, dan mudah disesuaikan dengan kebutuhan program yang dinamis. Praktikum ini sekaligus memperkuat

pemahaman konsep dasar linked list serta cara mengimplementasikannya secara nyata dalam bahasa C++.

E. Referensi

Pratama, R., & Kurniawan, D. (2021). *Penerapan Struktur Data Multilist untuk Pengelompokan Data Bertingkat pada Sistem Informasi*. Jurnal Teknologi dan Sistem Komputer, 9(2), 112–119.

Hermawan, A. (2020). *Analisis Kinerja Circular Linked List pada Sistem Penjadwalan Proses*. Jurnal Informatika dan Sistem Cerdas, 4(1), 45–52.

Wibowo, R., & Santoso, H. (2022). *Efisiensi Struktur Data Pointer-Based dalam Pengolahan Data Dinamis*. Jurnal Ilmu Komputer Terapan, 8(3), 198–207.