LAPORAN

MODUL 13

MULTI LINKED LIST

Disusun Oleh :

NAMA :

Devi Nurliana

NIM :

103112400144

Dosen :

FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

A. DASAR TEORI

Multi Linked List adalah struktur data yang memungkinkan satu elemen terhubung ke lebih dari satu elemen lain melalui beberapa pointer. Struktur ini biasanya digunakan untuk merepresentasikan hubungan yang bersifat hierarkis atau relasi satu ke banyak, seperti dalam data orang tua dan anak atau data mahasiswa yang memiliki atribut yang saling berkaitan. Setiap node dalam Multi Linked List bisa memiliki pointer ke node berikutnya serta pointer tambahan untuk menghubungkan ke node lain, sehingga data bisa dikelola dengan lebih fleksibel dibandingkan struktur linked list tunggal. Operasi dasar yang bisa dilakukan meliputi pembuatan list, menambah data, menghapus data, mencari elemen, dan menampilkan seluruh isi list sesuai dengan hubungan yang telah dibentuk.

B. GUIDED

a. Source Code

```
#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");
```

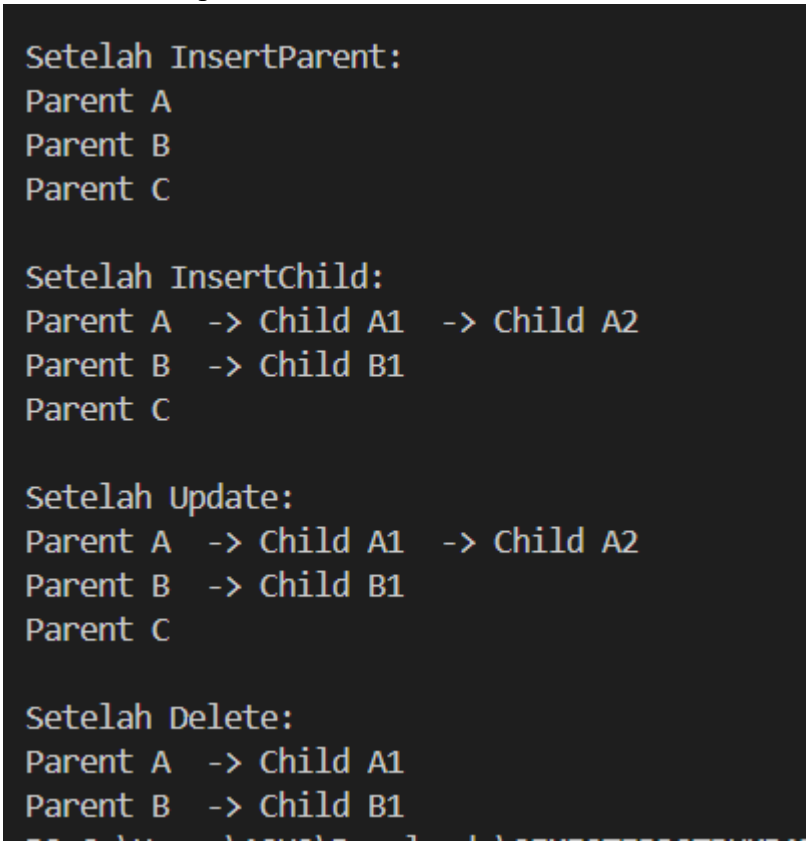
```
    cout << "\nSetelah Update:" << endl;
    printAll(list);

    deleteChild(list, "Parent A", "Child A2");
    deleteParent(list, "Parent C");

    cout << "\nSetelah Delete:" << endl;
    printAll(list);

    return 0;
}
```

b. Screenshot Output



c. Deskripsi

Program ini bertujuan untuk mengimplementasikan struktur data multilist yang terdiri dari node *parent* dan node *child* menggunakan pointer ganda (*prev* dan *next*).

1. Pendefinisian struktur data

Program mendefinisikan dua struktur, yaitu ParentNode untuk menyimpan data parent dan *ChildNode* untuk menyimpan data *child*. Setiap *parent* dapat memiliki banyak *child* yang dihubungkan melalui pointer *childHead.*

2. Proses penambahan data

Data *parent* ditambahkan menggunakan prosedur *insertParent,* sedangkan data

child ditambahkan ke *parent* tertentu menggunakan prosedur *insertChild*. Setiap data baru disisipkan di bagian akhir list.

3. Proses update dan delete data
   Program menyediakan fungsi *updateParent* dan *updateChild* untuk mengubah data *parent* atau *child*. Selain itu, tersedia *deleteChild* untuk menghapus *child* tertentu dan *deleteParent* untuk menghapus *parent* beserta seluruh *child*-nya.

4. Penampilan data
   Prosedur *printAll* digunakan untuk menampilkan seluruh data *parent* beserta *child* yang dimilikinya, sehingga hubungan antara *parent* dan *child* dapat terlihat dengan jelas.

C. UNGUIDED

1. Circularlist
   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"

      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));

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

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

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

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

b. Screenshot

```
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  : 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 ini bertujuan untuk mengimplementasikan ADT Circular Linked List dalam menyimpan dan mengelola data mahasiswa yang terdiri dari nama, NIM, jenis kelamin, dan IPK.

1. Pendefinisian struktur

   Struktur data mahasiswa (*infotype*), node (*ElmList*), dan list (*List*) didefinisikan pada file *circularlist.h* sebagai dasar pembentukan circular linked list.

2. Penyisipan dan pencarian data
   Alokasi dan penyisipan data dilakukan menggunakan fungsi *alokasi*, *insertFirst*, *insertLast*, dan *insertAfter* yang diimplementasikan pada file *circularlist.cpp*, dengan bantuan fungsi *findElm* untuk menentukan posisi penyisipan berdasarkan NIM.

3. Penampilan hasil
   Prosedur *printInfo* digunakan untuk menampilkan seluruh data mahasiswa secara berurutan. Seluruh proses dijalankan melalui file *main.cpp* dan program diakhiri setelah hasil ditampilkan.

D. KESIMPULAN

Berdasarkan penerapan yang telah dilakukan, dapat disimpulkan bahwa *Multi Linked List* adalah struktur data yang baik untuk mewakili hubungan satu ke banyak, seperti hubungan antara parent dan child atau data mahasiswa yang saling terkait. Dengan menggunakan pointer, data bisa dikelola secara fleksibel, sehingga operasi menambahkan, mencari, mengubah, dan menghapus elemen bisa dilakukan dengan mudah.

Program yang dibuat berhasil menerapkan konsep *Multi Linked List* melalui pendefinisiannya node *parent* dan *child,* serta ADT *Circular Linked List* untuk menyimpan data mahasiswa.Prosedur seperti insertFirst, insertLast, dan insertAfter memungkinkan pengaturan letak data sesuai kebutuhan, sedangkan fungsi findElm memudahkan pencarian elemen berdasarkan NIM.

Selain itu, proses menampilkan data menggunakan prosedur *printAll* dan *printInfo* mampu menunjukkan hubungan antar data secara jelas dan terorganisir.
Dengan demikian, implementasi ini membuktikan bahwa *Multi Linked List* dan *Circular Linked List* bisa digunakan secara efektif untuk mengelola data yang saling berhubungan dalam sebuah program.

E. REFERENSI

- Modul 13 Struktur Data – *Multi Linked List*, Program Studi Teknik Informatika.
- A. S. Tanenbaum & T. H. Cormen, *Introduction to Data Structures*, Pearson Education.
- Ellis Horowitz, Sartaj Sahni, dan Susan Anderson-Freed, *Fundamentals of Data Structures in C++*, W. H. Freeman and Company.
- Abdul Kadir, *Algoritma dan Struktur Data*, Andi Publisher.