LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL XI

MULTI LINKED LIST

Disusun Oleh :

NAMA : Taufik Hafit Zakaria
NIM : 103112430093

Dosen

WAHYU ANDI SAPUTRA

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Multi Linked List merupakan pengembangan dari struktur data Linked List yang lebih kompleks, di mana struktur ini terdiri dari sekumpulan list yang berbeda namun memiliki keterhubungan satu sama lain. Berbeda dengan *Single* atau *Double Linked List* yang bersifat linear (satu dimensi), Multi Linked List memungkinkan representasi data yang bersifat hirarkis atau memiliki relasi "One-to-Many" (satu ke banyak). Dalam implementasinya, biasanya terdapat elemen yang berperan sebagai **List Induk** (*Parent List*) dan elemen yang berperan sebagai **List Anak** (*Child List*). Setiap elemen pada list induk memiliki pointer tambahan yang menunjuk ke alamat awal (*head*) dari list anak yang dimilikinya, sehingga membentuk cabang-cabang list baru dari setiap elemen induk.

Karakteristik Multi Linked List

- **Struktur Hirarkis (Parent-Child):** Memiliki struktur yang membedakan antara elemen induk dan elemen anak. Satu elemen induk dapat memiliki relasi ke banyak elemen anak, namun elemen anak spesifik hanya dimiliki oleh satu induk (dalam model standar).
- **Konektivitas Antar List:** Selain memiliki pointer next (dan prev) untuk menghubungkan sesama elemen dalam satu list, elemen induk memiliki pointer tambahan (misalnya child_ptr) yang menghubungkan ke elemen pertama dari list anak.
- **Relasi One-to-Many:** Sangat merepresentasikan hubungan data satu-ke-banyak. Contohnya: Satu Pegawai (Induk) memiliki banyak Anak/Tanggungan (Anak), atau Satu Mahasiswa (Induk) mengambil banyak Mata Kuliah (Anak).

Keuntungan Multi Linked List

- **Representasi Data Realistis:** Sangat ideal untuk memodelkan data dunia nyata yang memiliki sub-kategori atau pengelompokan, memungkinkan data terorganisir secara logis berdasarkan kepemilikannya (misalnya, data penjualan dikelompokkan berdasarkan nama sales).
- **Fleksibilitas Manipulasi Data:** Memudahkan operasi pada sub-kelompok data tertentu. Misalnya, menambah "anak" baru pada "pegawai A" tidak akan mengganggu struktur data milik "pegawai B".
- **Efisiensi Memori Dinamis:** Menggunakan alokasi memori dinamis (pointer), sehingga jumlah induk maupun jumlah anak per induk dapat bertambah dan berkurang secara fleksibel tanpa batasan ukuran statis seperti pada Array multidimensi.

Keterbatasan Multi Linked List

- **Kompleksitas Algoritma:** Implementasi operasi jauh lebih rumit dibandingkan list biasa. Operasi pada list anak (seperti *insert* atau *delete*) mengharuskan program untuk menelusuri (*traversal*) dan menemukan posisi elemen induknya terlebih dahulu.
- **Penghapusan Bersyarat (Cascade Delete):** Saat menghapus sebuah elemen induk (*Delete Parent*), seluruh elemen anak yang terhubung dengan induk tersebut wajib dihapus atau didealokasi terlebih dahulu. Jika tidak, akan terjadi *memory leak* di mana data anak masih tersimpan di memori namun tidak memiliki akses (menjadi data "yatim").
- **Overhead Memori:** Membutuhkan ruang memori tambahan untuk menyimpan pointer ekstra pada setiap node induk yang menunjuk ke list anak.

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

Guided 1

```cpp
//Taufik Hafit Zakaria
//103112430093
//guided.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;
    }
}

```

```
53  void insertChild(ParentNode *head, string parentInfo, string childInfo) {
54      ParentNode *p = head;
55      while (p != NULL && p->info != parentInfo) {
56          p = p->next;
57      }
58
59      if (p != NULL) {
60          ChildNode *newChild = createChild(childInfo);
61          if (p->childHead == NULL) {
62              p->childHead = newChild;
63          } else {
64              ChildNode *c = p->childHead;
65              while (c->next != NULL) {
66                  c = c->next;
67              }
68              c->next = newChild;
69              newChild->prev = c;
70          }
71      }
72  }
73
74  void printAll(ParentNode *head) {
75      while (head != NULL) {
76          cout << head->info;
77          ChildNode *c = head->childHead;
78          while (c != NULL) {
79              cout << " -> " << c->info;
80              c = c->next;
81          }
82          cout << endl;
83          head = head->next;
84      }
85  }
86
87  void updateParent(ParentNode *head, string oldInfo, string newInfo) {
88      ParentNode *p = head;
89      while (p != NULL) {
90          if (p->info == oldInfo) {
91              p->info = newInfo;
92              return;
93          }
94          p = p->next;
95      }
96  }
97
98  void updateChild(ParentNode *head, string parentInfo, string oldChildInfo, string newChildInfo) {
99      ParentNode *p = head;
100     while (p != NULL && p->info != parentInfo) {
101         p = p->next;
102     }
103
104     if (p != NULL) {
105         ChildNode *c = p->childHead;
106         while (c != NULL) {
107             if (c->info == oldChildInfo) {
108                 c->info = newChildInfo;
109                 return;
110             }
111             c = c->next;
112         }
113     }
114 }
115
```

```
116 void deleteChild(ParentNode *head, string parentInfo, string childInfo) {
117     ParentNode *p = head;
118     while (p != NULL && p->info != parentInfo) {
119         p = p->next;
120     }
121 
122     if (p != NULL) {
123         ChildNode *c = p->childHead;
124         while (c != NULL) {
125             if (c->info == childInfo) {
126                 if (c == p->childHead) {
127                     p->childHead = c->next;
128                     if (p->childHead != NULL) {
129                         p->childHead->prev = NULL;
130                     }
131                 } else {
132                     c->prev->next = c->next;
133                     if (c->next != NULL) {
134                         c->next->prev = c->prev;
135                     }
136                 }
137                 delete c;
138                 return;
139             }
140             c = c->next;
141         }
142     }
143 }
144 
145 void deleteParent(ParentNode *&head, string info) {
146     ParentNode *p = head;
147     while (p != NULL) {
148         if (p->info == info) {
149             ChildNode *c = p->childHead;
150             while (c != NULL) {
151                 ChildNode *tempC = c;
152                 c = c->next;
153                 delete tempC;
154             }
155 
156             if (p == head) {
157                 head = p->next;
158                 if (head != NULL) {
159                     head->prev = NULL;
160                 }
161             } else {
162                 p->prev->next = p->next;
163                 if (p->next != NULL) {
164                     p->next->prev = p->prev;
165                 }
166             }
167             delete p;
168             return;
169         }
170         p = p->next;
171     }
172 }
173 
```

```cpp
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;
}
```

Screenshots Output

```
PS C:\programing> cd "c:\programing\Struktur Data\LAPRAK 11\" ; if ($?) { g++ guided.cpp -o guided } ; if ($?) { .\guided }

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
PS C:\programing\Struktur Data\LAPRAK 11>
```

Deskripsi:

Program ini adalah implementasi struktur data Multi Linked List (List of Lists) menggunakan C++ dengan pendekatan representasi pointer (doubly linked list dinamis), baik untuk struktur list induk (*Parent*) maupun list anak (*Child*). Intinya, program ini menerapkan prinsip relasi satu ke banyak (*one-to-many*), di mana setiap node induk memiliki pointer khusus yang menunjuk ke node awal (*head*) dari linked list anak, memungkinkan pembentukan struktur data hierarkis yang fleksibel. Kode program ditulis dalam satu file bernama guided.cpp yang mencakup definisi struktur ChildNode dan ParentNode, implementasi fungsi manajemen memori dan pointer, serta fungsi main yang berfungsi sebagai driver untuk mendemonstrasikan operasi list.

Alur program di fungsi main secara langsung mendemonstrasikan simulasi operasi manipulasi data pada struktur Multi Linked List. Awalnya, list induk diinisialisasi dan diisi dengan tiga elemen ("Parent A", "Parent B", dan "Parent C") menggunakan fungsi insertParent. Program kemudian menyisipkan elemen anak pada induk tertentu menggunakan insertChild, lalu menampilkan struktur yang terbentuk. Setelah itu, dilakukan operasi pembaruan (*update*) pada data induk dan anak, diikuti dengan operasi penghapusan (*delete*) untuk menghilangkan node anak spesifik ("Child A2") serta menghapus node induk ("Parent C") beserta seluruh anak yang terhubung dengannya (*cascade delete*). Hasil akhir ditampilkan kembali menggunakan fungsi printAll, yang membuktikan bahwa integritas relasi antar list tetap terjaga meskipun telah terjadi penyisipan, modifikasi, dan penghapusan data.

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

Unguided 1

```
//Taufik Hafit Zakaria
//103112430093
//circularlist.h

#ifndef CIRCULARLIST_H_INCLUDED
#define CIRCULARLIST_H_INCLUDED

#include <string>
using namespace std;

#define Nil NULL

struct infotype {
    string nama;
    string nim;
    char jenis_kelamin;
    float ipk;
};

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);

#endif
```

```cpp
//Taufik Hafit Zakaria
//103112430093
//circularlist.cpp

#include "circularlist.h"
#include <iostream>
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;
}

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

```

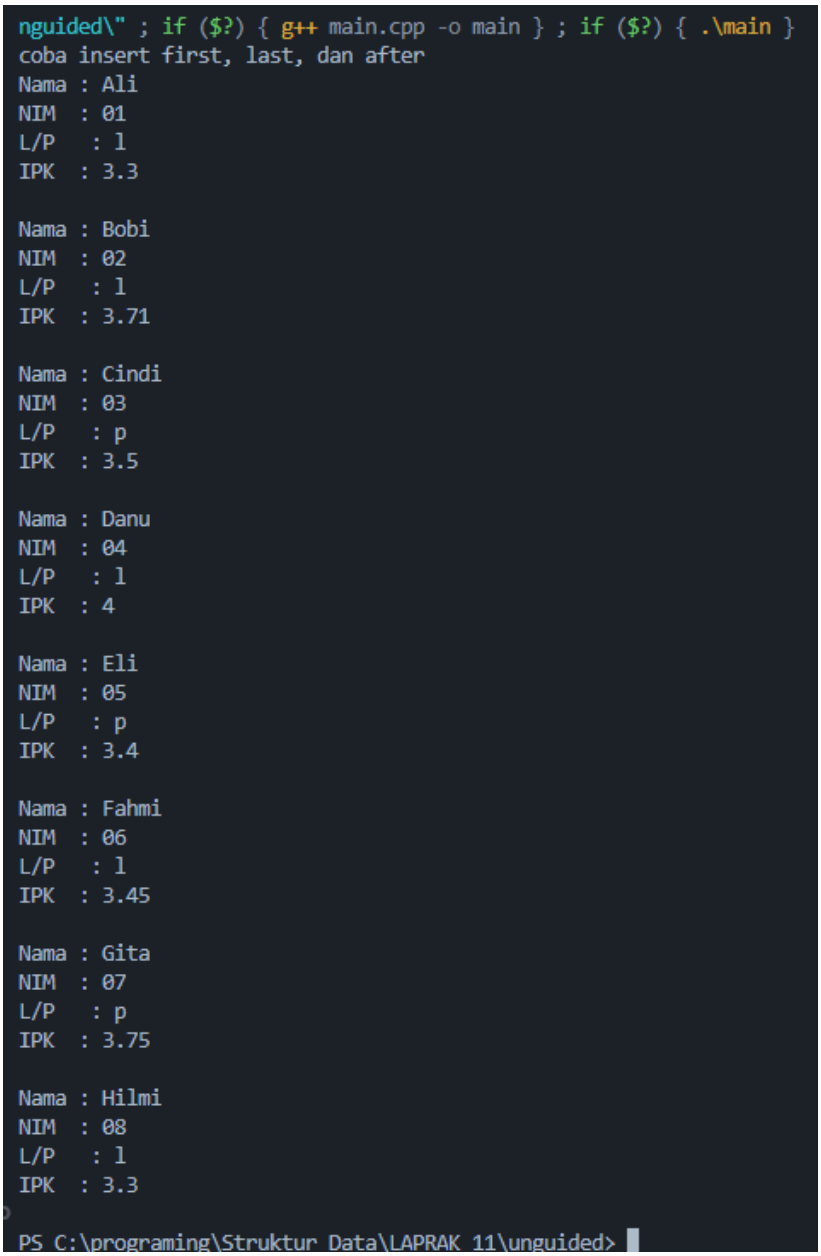
```
39  void insertAfter(List &L, address Prec, address P) {
40      P->next = Prec->next;
41      Prec->next = P;
42  }
43
44  void insertLast(List &L, address P) {
45      if (L.first == Nil) {
46          P->next = P;
47          L.first = P;
48      } else {
49          address Q = L.first;
50          while (Q->next != L.first) {
51              Q = Q->next;
52          }
53          P->next = L.first;
54          Q->next = P;
55      }
56  }
57
58  void deleteFirst(List &L, address &P) {
59      if (L.first != Nil) {
60          P = L.first;
61          if (P->next == L.first) {
62              L.first = Nil;
63          } else {
64              address Q = L.first;
65              while (Q->next != L.first) {
66                  Q = Q->next;
67              }
68              L.first = P->next;
69              Q->next = L.first;
70          }
71          P->next = Nil;
72      }
73  }
74
```

```cpp
void deleteAfter(List &L, address Prec, address &P) {
    if (Prec != Nil) {
        P = Prec->next;
        if (P == L.first && P->next == L.first) {
            L.first = Nil;
        } else {
            Prec->next = P->next;
            if (P == L.first) {
                L.first = P->next;
            }
        }
        P->next = Nil;
    }
}

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

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

```cpp
//Taufik Hafit Zakaria
//103112430093
//main.cpp

#include "circularlist.h"
#include "circularlist.cpp"
#include <iostream>
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);
}

void sortList(List &L) {
    if (L.first == Nil) {
        return;
    }

    address P = L.first;
    address Q;
    infotype temp;

    do {
        Q = P->next;
        while (Q != L.first) {
            if (P->info.nim > Q->info.nim) {
                temp = P->info;
                P->info = Q->info;
                Q->info = temp;
            }
            Q = Q->next;
        }
        P = P->next;
    } while (P->next != L.first);
}

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

    createList(L);

```

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

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

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

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

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

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

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

    x.nim = "02";
    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);

    sortList(L);

    printInfo(L);

    return 0;
}
```

Screenshots Output



Deskripsi:

Program ini adalah implementasi struktur data *Circular Singly Linked List* menggunakan C++ dengan pendekatan representasi pointer (linked list dinamis). Intinya, program ini menerapkan prinsip senarai berantai melingkar, di mana pointer next pada elemen terakhir tidak menunjuk ke NULL, melainkan kembali menunjuk ke elemen pertama (*head*), sehingga memungkinkan penelusuran data secara kontinu tanpa ujung. Kode program diorganisir secara modular ke dalam tiga file: circularlist.h sebagai header yang berisi definisi tipe data infotype untuk data mahasiswa dan prototipe fungsi, circularlist.cpp yang berisi implementasi logika dasar operasi list seperti penyisipan dan penghapusan, serta main.cpp yang berfungsi sebagai driver untuk mendemonstrasikan operasi list.

Alur program di fungsi main secara langsung mendemonstrasikan operasi manipulasi data mahasiswa pada list. Awalnya, sebuah list kosong dibuat dan diisi dengan beberapa data mahasiswa (seperti Danu, Fahmi, Bobi, Ali, dan Gita) menggunakan variasi fungsi insertFirst dan insertLast. Selanjutnya, dilakukan penyisipan elemen di tengah list menggunakan insertAfter (untuk data Cindi, Hilmi, dan Eli) yang didahului dengan pencarian posisi elemen referensi menggunakan findElm. Sebelum data ditampilkan, program menjalankan prosedur sortList untuk mengurutkan seluruh data mahasiswa secara menaik (*ascending*) berdasarkan NIM. Hasil akhir ditampilkan menggunakan prosedur printInfo, yang membuktikan bahwa list mampu menyimpan, menyisipkan, dan mengurutkan data dengan benar dalam struktur melingkar.

Disini saya menambahkan satu buah fungsi lagi untuk sorting dikarenakan output yang diminta pada soal diharuskan nim terurut dari 1 sampai 8

D. Kesimpulan

Dari hasil praktikum yang telah dilakukan, dapat disimpulkan bahwa struktur data **Multi Linked List** merupakan solusi yang efektif untuk merepresentasikan data yang memiliki hubungan hierarkis atau relasi *one-to-many* (satu ke banyak). Melalui implementasi program ini, konsep "List di dalam List" dapat dipahami dengan baik, di mana setiap elemen pada list induk (*Parent*) berfungsi sebagai kepala bagi list anak (*Child*) yang terhubung dengannya. Praktikum ini secara khusus mendemonstrasikan bahwa manipulasi data pada Multi Linked List menuntut pemahaman logika pointer yang lebih dalam dibandingkan Single atau Double Linked List biasa, terutama pada operasi seperti penyisipan anak yang memerlukan pencarian alamat induk terlebih dahulu, serta operasi penghapusan induk (*delete parent*) yang mewajibkan penghapusan seluruh elemen anak terkait (*cascade delete*) guna mencegah terjadinya kebocoran memori (*memory leak*).

Selain itu, program ini membuktikan bahwa penggunaan Multi Linked List memberikan fleksibilitas yang jauh lebih tinggi dalam pengelolaan data berkelompok dibandingkan dengan penggunaan array multidimensi yang bersifat statis. Struktur ini sangat relevan dan berguna untuk memodelkan kasus-kasus nyata yang melibatkan kategorisasi, seperti data mahasiswa beserta mata kuliah yang diambilnya, struktur departemen dalam perusahaan, atau sistem kategori produk dalam *e-commerce*. Dengan memahami prinsip relasi antar-list dan manajemen alokasi memori dinamis pada Multi Linked List, mahasiswa memiliki fondasi yang kuat untuk mempelajari struktur data lanjut yang lebih kompleks, seperti representasi *Graph* (khususnya *Adjacency List*) dan struktur data N-ary Tree.

E. Referensi

Weiss, M. A. (2014). Data Structures and Algorithm Analysis in C++ (4th ed.). Pearson.

GeeksforGeeks. (2024). Circular Linked List Data Structure. Diakses pada tahun 2025 dari https://www.geeksforgeeks.org/circular-linked-list/

GeeksforGeeks. (2024). Doubly Linked List. Diakses pada tahun 2025 dari https://www.geeksforgeeks.org/doubly-linked-list/

Programiz. (2024). Circular Linked List. Diakses pada tahun 2025 dari https://www.programiz.com/dsa/circular-linked-list