LAPORAN PRAKTIKUM
Modul 6
“Double Linked List (Bagian Pertama)”

Disusun Oleh:
Dimastian Aji Wibowo (2311104058)
SE-07-02

Dosen :
Wahyu Andi Saputra, S.Pd., M.Eng.

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO

2024

1. **Tujuan**
   - Memahami konsep modul *linked list*.
   - Mengaplikasikan konsep *double linked list* dengan menggunakan *pointer* dengan menggunakan bahasa C++

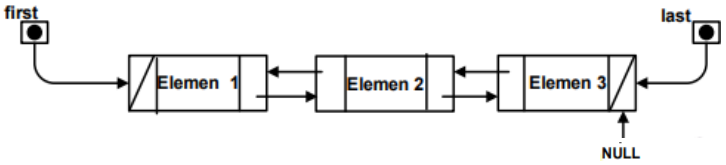
2. **Landasan Teori**

Double Linked List

Double Linked list adalah *linked list* yang masing – masing elemen nya memiliki 2 *successor*, yaitu *successor* yang menunjuk pada elemen sebelumnya (*prev*) dan *successor* yang menunjuk pada elemen sesudahnya (*next*).

Gambar berikut menunjukan bentuk *Double Linked list* dengan elemen kosong:

Gambar berikut menunjukan bentuk *Double Linked list* dengan 3 elemen:



Double linked list juga menggunakan dua buah successor utama yang terdapat pada *list*, yaitu first (successor yang menunjuk elemen pertama) dan last (susesor yang menunjuk elemen terakhir *list*).

Komponen – komponen dalam *double linked list*:

1. First : pointer pada *list* yang menunjuk pada elemen pertama *list*.
2. Last : pointer pada *list* yang menunjuk pada elemen terakhir *list*.
3. Next : pointer pada elemen sebagai successor yang menunjuk pada

elemen didepannya.

4. Prev : pointer pada elemen sebagai successor yang menunjuk pada elemen dibelakangnya.

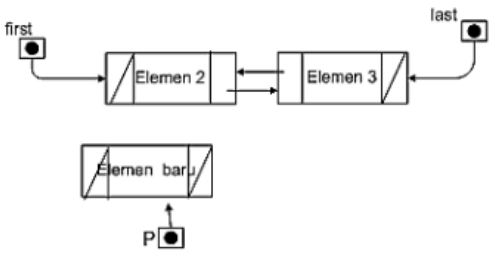
Contoh pendeklarasian struktur data untuk *double linked list*:

```
#ifndef doublelist_H
#define doublelist_H
#include "boolean.h"
#define Nil NULL
#define info(P) (P)->info
#define next(P) (P)->next
#define prev(P) (P)->prev
#define first(L) ((L).first)
#define last(L) ((L).last)
/*deklarasi record dan struktur data double linked list*/
typedef int infotype;
typedef struct elmlist *address;
struct elmlist {
infotype info;
address next;
address prev;
};
/* definisi list: */
/* list kosong jika First(L)=Nil */
struct list{
address first;
address last;
};
#endif
```

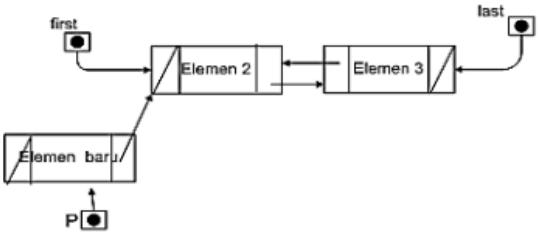
A. Insert

1. Insert First

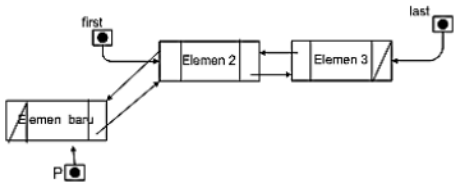
Langkah – langkah dalam proses insert first:



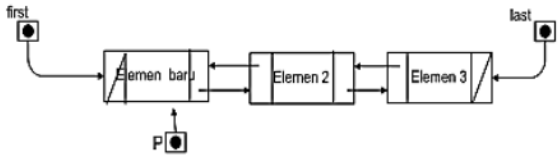

```
P = alokasi(X);
next(P) = Nil;
prev(P) = Nil;
```




```
next(P) = first(L);
```




```
prev(first(L)) = P;
```

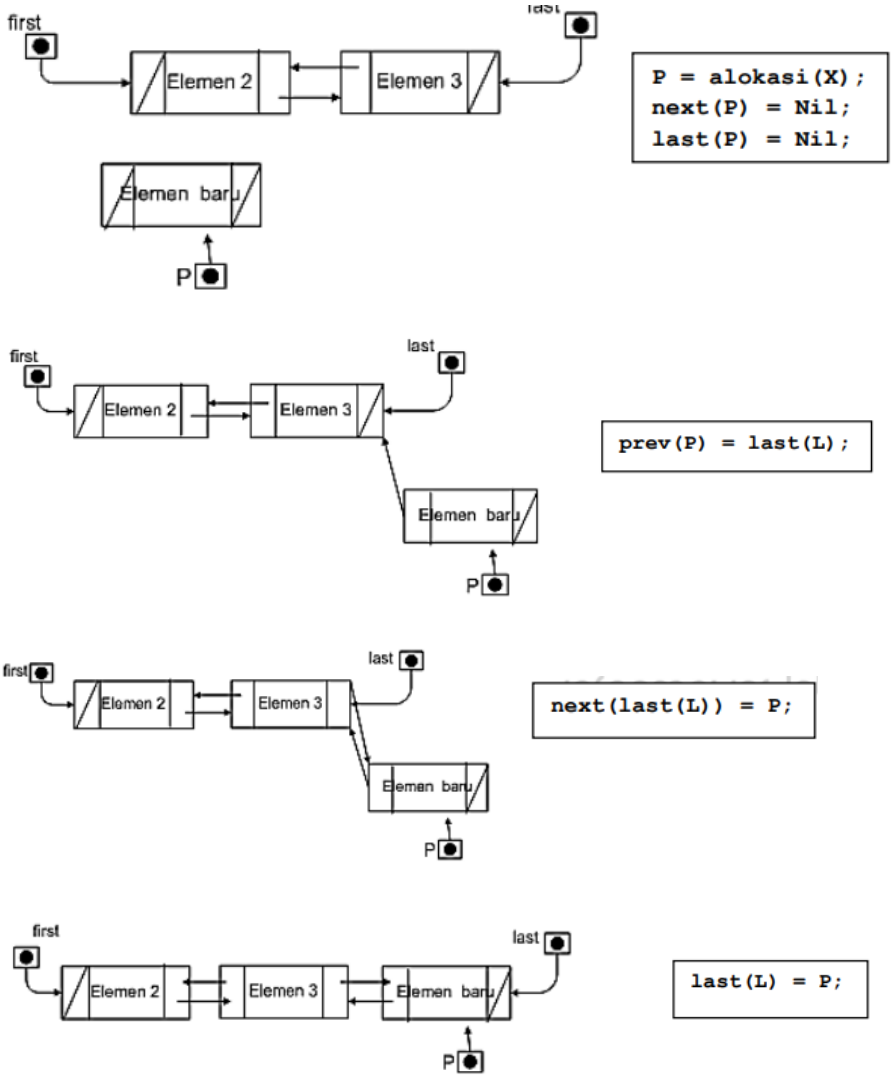



```
first(L) = P;
```

```
void insertFirst(list &L, address &P){
next(P) = first(L);
prev(first(L)) = P;
first(L) = P;
}
```

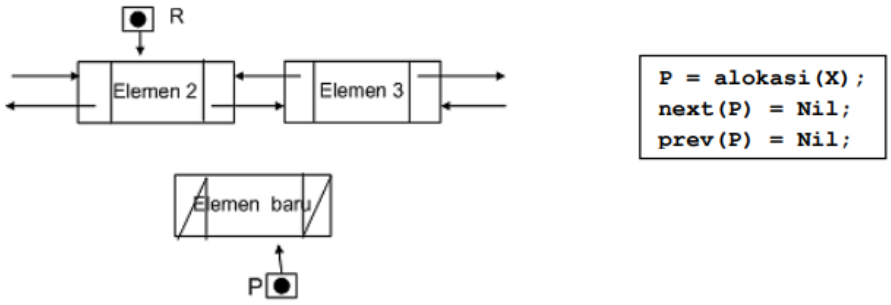
2. Insert Last

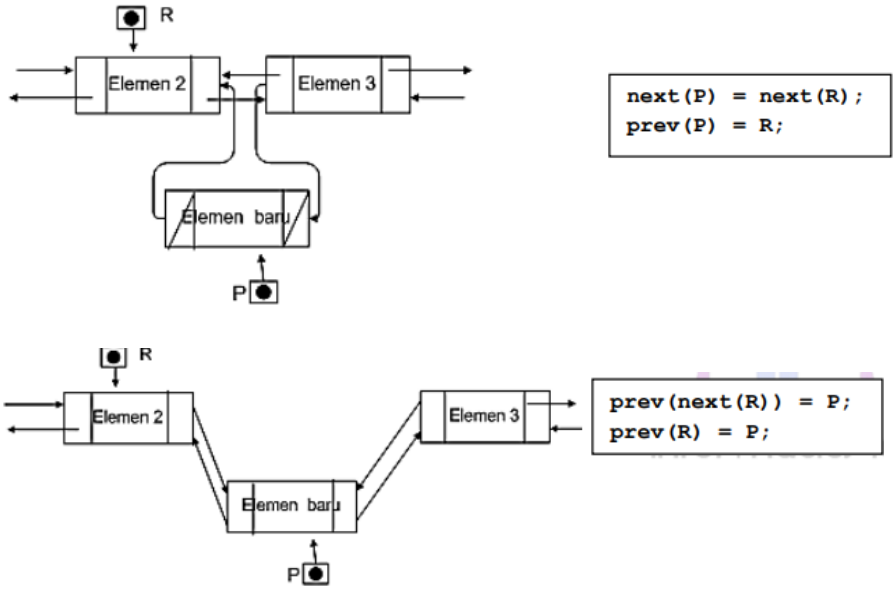
Langkah – langkah dalam proses insert last:



3. Insert After

Langkah – langkah dalam proses insert after:





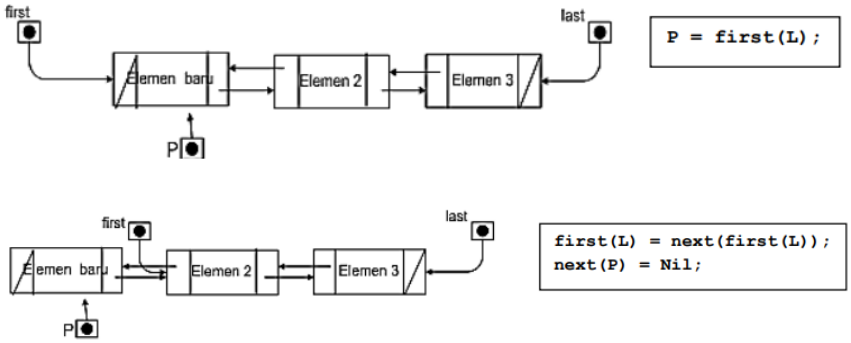
4. Insert Before

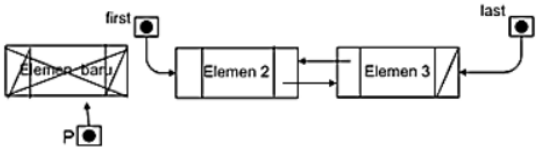
Diatas hanya dijelaskan tentang insert after. Insert before hanya kebalikan dari insert after. Perbedaan Insert After dan Insert Before terletak pada pencarian elemennya.

B. Delete

1. Delete First

Langkah – langkah dalam proses delete first:



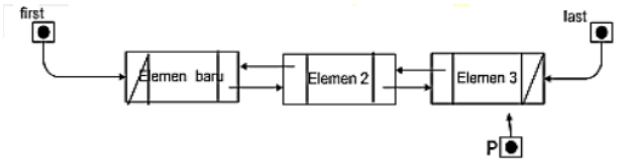



```
prev(first(L)) = Nil;
next(P) = Nil;
return P;
        atau
dealokasi(P);
```

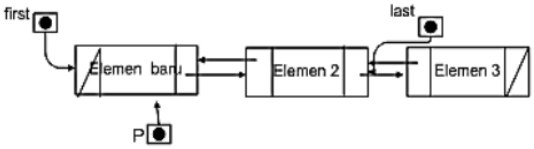
```
/* contoh sintak delete first */
void deleteFirst(list &L, address &P){
P = first(L);
first(L) = next(first(L));
prev (P) = null;
prev(first(L)) = null;
next(P) = null;
}
```

2. Delete Last

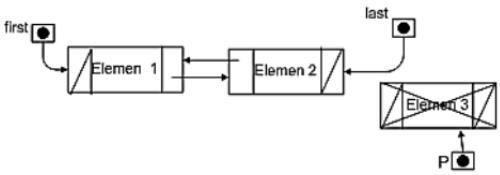
Langkah – langkah dalam proses delete last:




```
P = last(L);
```



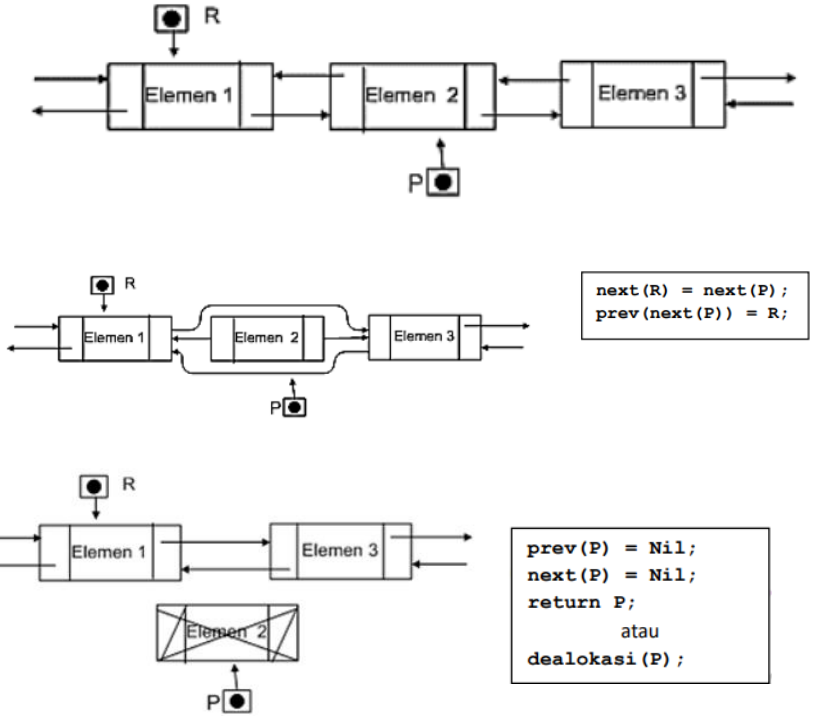

```
last(L) = prev(last(L));
```




```
prev(P) = Nil;
next(last(L)) = Nil;
return P;
        atau
dealokasi(P);
```

3. Delete After

Langkah – langkah dalam proses delete after:



4. Delete Before

Diatas hanya dijelaskan tentang delete after. Delete before hanya kebalikan dari delete after. Perbedaan Delete After dan Delete Before terletak pada pencarian elemennya.

5. Update, View, dan Searching

Proses pencarian, update data dan view data pada dasarnya sama dengan proses pada single *linked list*. Hanya saja pada *double linked list* lebih mudah dalam melakukan proses akses elemen, karena bisa melakukan iterasi maju dan mundur. Seperti halnya single *linked list*, *double linked list* juga mempunyai ADT yang pada dasarnya sama dengan ADT yang ada pada single *linked list*.

```
/*file : doublelist .h*/
/* contoh ADT list berkait dengan representasi fisik
pointer*/
```

```
/* representasi address dengan pointer*/
/* info tipe adalah integer */
#ifndef doublelist_H
#define doublelist_H
#include <stdio.h>
#define Nil NULL
#define info(P) (P)->info
#define next(P) (P)->next
#define prev(P) (P)->prev
#define first(L) ((L).first)
#define last(L) ((L).last)
typedef int infotype;
typedef struct elmlist *address;
/* pendefinisian tipe data bentukan elemen list
 dengan dua successor, yaitu next dan prev */
struct elmlist{
infotype info;
address prev;
address next;
};
/* definisi double linked list : list kosong jika
first(L)=Nil
 setiap elemen address P dapat diacu info(P) atau
next(P)
 elemen terakhir adalah last
 nama tipe list yang dipakai adalah 'list', sama dengan
pada singe list*/
struct list {
address first,last;
};
/** Deklarasi fungsi primitif lain **/
/** Sama dengan Single Linked list **/
```

3. Guided

1. Kelas node yang mendefinisikan elemen dasar dari linked list yang mana setiap node memiliki data, prev, dan next
2. Kelas DoubleLinkedList merupakan kelas yang mengelola struktur dari linked list dan menyediakan berbagai fungsi untuk manipulasi data dan terdapat head dan tail.
3. Membuat konstruktor yang menginisialisasi head dan tail dengan nullptr, menandakan bahwa list dimulai dalam keadaan kosong.

4. Membuat fungsi Insert, deleteNode, update, dan display.
5. Membuat fungsi main dan menampilkan beberapa opsi yang dapat dipilih pengguna dan untuk setiap pilihan program meminta input pengguna dan memanggil fungsi yang sesuai pada objek list.

4. Unguided

A. Unguided 1

1. Membuat file header diikuti membuat *header guard* yang digunakan untuk mencegah multiple inclusions dari file header yang sama dan membuat pustaka iostream.
2. Membuat struktur untuk mendefinisikan struktur data untuk kendaraan yang memiliki tiga atribut yaitu string nopol, string warna, dan int thnBuat.
3. Memberi nama alias infotype untuk struct kendaraan dan memberi nama alias address untuk pointer ke Elm*List*, yang akan digunakan untuk menunjuk ke elemen dalam *linked list*.
4. Membuat struktur untuk mendefinisikan elemen dari *double linked list* yang memiliki tiga atribut yaitu info untuk menyimpan informasi kendaraan dari tipe infotype, next yang bertipe address merupakan pointer yang menunjuk ke elemen berikutnya dalam *list*, dan prev yang bertipe address merupakan pointer yang menunjuk ke elemen sebelumnya dalam *list*.
5. Mendefinisikan struktur *list* yang menyimpan dua pointer First untuk menunjuk ke elemen pertama dalam *list* dan Last untuk menunjuk ke elemen terakhir dalam *list*.
6. Mendeklarasikan fungsi – fungsi yang akan diimplementasikan di file .cpp, mulai dari Create*List*(*List* &L);, address alokasi(infotype x);, void dealokasi(address &P);, void printInfo(const *List* &L);, dan void insertLast(*List* &L, address P);.

```
#ifndef DOUBLELIST_H
#define DOUBLELIST_H
#include <string>
using namespace std;

struct kendaraan {
    string nopol;
    string warna;
    int thnBuat;
};
typedef kendaraan infotype;
typedef struct ElmList *address;
struct ElmList {
    infotype info;
    address next;
    address prev;
};
struct List {
    address First;
    address Last;
};
void CreateList(List &L);
address alokasi(infotype x);
void dealokasi(address &P);
void printInfo(const List &L);
void insertLast(List &L, address P);
#endif
```

7. Membuat file *doublelist*.cpp untuk mengimplementasikan file header dan menjadi sumber dari file main.
8. Membuat fungsi Create*List* dengan parameter L yaitu referensi ke objek *List* yang akan diinisialisasi yang bertujuan untuk menginisialisasi *list* kosong dengan mengatur pointer First dan Last ke nullptr.
9. Membuat fungsi alokasi untuk mengalokasikan memori untuk elemen baru dalam *list* dan menginisialisasi informasi yang akan disimpan dengan proses menggunakan new untuk membuat elemen baru Elm*List*, mengisi atribut info dengan x, dan mengatur pointer next dan prev ke nullptr, dan mengembalikan pointer P ke elemen baru yang sudah dialokasikan.
10. Membuat fungsi dealokasi untuk menghapus elemen dari memori dan mengatur pointer yang menunjuk ke elemen tersebut menjadi nullptr dengan menggunakan delete untuk membebaskan memori yang

digunakan oleh elemen P dan mengatur P menjadi nullptr untuk menghindari pointer yang menggantung.

11. Membuat fungsi printInfo untuk menampilkan informasi semua elemen dalam *list*.

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

void CreateList(List &L) {
    L.First = nullptr;
    L.Last = nullptr;
}
address alokasi(infotype x) {
    address P = new ElmList;
    P->info = x;
    P->next = nullptr;
    P->prev = nullptr;
    return P;
}
void dealokasi(address &P) {
    delete P;
    P = nullptr;
}
void printInfo(const List &L) {
    address P = L.First;
    int i = 1;
    while (P != nullptr) {
        cout << "Data List " << i++ << endl;
        cout << "no polisi : " << P->info.nopol << endl;
        cout << "warna     : " << P->info.warna << endl;
        cout << "tahun     : " << P->info.thnBuat << endl;
        P = P->next;
    }
}
```

12. Membuat fungsi insertLast dengan proses jika *list* kosong (L.First == nullptr), elemen baru P menjadi elemen pertama dan terakhir (L.Last) dan jika *list* tidak kosong, elemen baru P ditambahkan di akhir dengan memperbarui pointer next dari elemen terakhir dan mengatur prev dari elemen baru menjadi elemen terakhir yang ada. Pointer Last kemudian diperbarui untuk menunjuk ke elemen baru.

```
30  void insertLast(List &L, address P) {
31      if (L.First == nullptr) {
32          L.First = P;
33          L.Last = P;
34      } else {
35          L.Last->next = P;
36          P->prev = L.Last;
37          L.Last = P;
38      }
39  }
```

13. Membuat file main.cpp untuk menguji fungsi – fungsi pada file sumber.
14. Mendeklarasikan variabel L bertipe *List*, yang akan digunakan untuk menyimpan data kendaraan dan memanggil fungsi Create*List* untuk menginisialisasi *list* L.
15. Mendeklarasikan variabel data untuk menyimpan informasi kendaraan yang akan dimasukkan oleh pengguna.
16. Menggunakan perulangan untuk proses input data.
17. Memanggil fungsi printInfo(L) untuk mencetak informasi semua kendaraan yang telah dimasukkan ke dalam *list*.
18. Mendeklarasikan pointer P dan mengatur nilainya ke elemen pertama dalam *list* (L.First) dan menggunakan perulangan untuk mengulangi proses sampai P menjadi nullptr.
19. Mengembalikan nilai 0, menandakan bahwa program telah selesai dijalankan tanpa error.

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

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

    infotype data;
    string input;

    for (int i = 0; i < 4; i++) {
        cout << "Masukkan nomor polisi: ";
        cin >> data.nopol;

        cout << "Masukkan warna kendaraan: ";
        cin >> data.warna;

        cout << "Masukkan tahun kendaraan: ";
        cin >> data.thnBuat;

        address P = alokasi(data);
        insertLast(L, P);
        cout << endl;
    }

    printInfo(L);

    address P = L.First;
    while (P != nullptr) {
        address next = P->next;
        dealokasi(P);
        P = next;
    }

    return 0;
}
```

20. Berikut merrupakan output dari program tersebut.

```
"D:\College\Semester 3\Prakt
Masukkan nomor polisi: 2199
Masukkan warna kendaraan: Putih
Masukkan tahun kendaraan: 2023

Masukkan nomor polisi: 5006
Masukkan warna kendaraan: Hitam
Masukkan tahun kendaraan: 2019

Masukkan nomor polisi: 1444
Masukkan warna kendaraan: Merah
Masukkan tahun kendaraan: 2025

Masukkan nomor polisi: 2525
Masukkan warna kendaraan: Biru
Masukkan tahun kendaraan: 2025

Data List 1
no polisi : 2199
warna     : Putih
tahun     : 2023
Data List 2
no polisi : 5006
warna     : Hitam
tahun     : 2019
Data List 3
no polisi : 1444
warna     : Merah
tahun     : 2025
Data List 4
no polisi : 2525
warna     : Biru
tahun     : 2025
```

B. Unguided 2

1. Mendeklarasikan fungsi findElm pada file header.

```cpp
address findElm(const List &L, const infotype &x);
```

2. Membuat fungsi findElm pada file *doublelist*.cpp dengan parameter const *List* &L dan const infotype &x diikuti dengan menginisialisasi pointer address P = L.First;, menggunakan perulangan untuk berjalan terus selama P tidak null, yang berarti selama masih ada elemen yang tersisa untuk diperiksa dalam *list*.
3. Didalam perulangan, fungsi memeriksa apakah nomor polisi dari elemen saat ini (P->info.nopol) sama dengan nomor polisi yang dicari (x.nopol) dan jika elemen saat ini tidak cocok, pointer P diperbarui untuk menunjuk ke elemen berikutnya dalam *list*.
4. Jika loop selesai dan tidak ada elemen yang cocok ditemukan, fungsi akan mengembalikan nullptr, menandakan bahwa elemen yang dicari tidak ada dalam *list*.

```
address findElm(const List &L, const infotype &x) {
    address P = L.First;
    while (P != nullptr) {
        if (P->info.nopol == x.nopol) {
            return P;
        }
        P = P->next;
    }
    return nullptr;
}
```

5. Mendeklarasikan variabel searchData pada file main.cpp yang merupakan tipe infotype.
6. Membuat pesan kepada pengguna untuk meminta input nomor polisi yang ingin dicari dan menerima input dari pengguna dan menyimpannya dalam searchData.nopol.
7. Memanggil fungsi findElm untuk mencari elemen dalam *list* L yang memiliki nomor polisi yang sama dengan searchData.nopol dan hasil dari pencarian ini disimpan dalam foundElement.
8. Memeriksa apakah foundElement tidak null. Jika tidak null, berarti kendaraan dengan nomor polisi yang dicari ditemukan dan

menampilkan pesan bahwa telat ditemukan beserta informasinya, dan jika tidak ditemukan akan menampilkan pesan bahwa nomor polisi yang dicari tidak ditemukan.

```
infotype searchData;
cout << "Masukkan nomor polisi yang ingin dicari: ";
cin >> searchData.nopol;

address foundElement = findElm(L, searchData);
if (foundElement != nullptr) {
    cout << "Kendaraan ditemukan!" << endl;
    cout << "Nomor Polisi: " << foundElement->info.nopol << endl;
    cout << "Warna: " << foundElement->info.warna << endl;
    cout << "Tahun: " << foundElement->info.thnBuat << endl;
} else {
    cout << "Kendaraan dengan nomor polisi " << searchData.nopol << " tidak ditemukan." << endl;
}
```

9. Berikut merupakan output dari program tersebut

```
Masukkan nomor polisi yang ingin dicari: 2199
Kendaraan ditemukan!
Nomor Polisi: 2199
Warna: Putih
Tahun: 2023
```

C. Unguided 3

1. Mendeklarasikan fungsi deleteFirst, deleteLast, dan deleteAfter pada file header.

```
void deleteFirst(List &L, address &P);
void deleteLast(List &L, address &P);
void deleteAfter(address Prec, address &P);
```

2. Membuat fungsi deleteFirst dengan mengecek *list* kosong atau tidak jika tidak kosong maka tidak ada elemen yang dapat dihapus, menyimpan alamat elemen pertama ke P, mengubah L.First untuk menunjuk ke elemen berikutnya, jika L.First tidak nullptr, atur L.First->prev menjadi nullptr, jika Jika L.First menjadi nullptr, set L.Last juga menjadi nullptr, atur P->next dan P->prev menjadi nullptr untuk menghindari dangling pointer dan jika *list* kosong maka cetak pesan bahwa tidak ada elemen yang dihapus.

```
void deleteFirst(List &L, address &P) {
    if (L.First != nullptr) {
        P = L.First;
        L.First = L.First->next;
        if (L.First != nullptr) {
            L.First->prev = nullptr;
        } else {
            L.Last = nullptr;
        }
        P->next = nullptr;
        P->prev = nullptr;
    } else {
        cout << "List kosong, tidak ada elemen yang dihapus." << endl;
    }
}
```

3. Membuat fungsi deleteLast dengan mengecek *list* kosong atau tidak jika tidak kosong maka tidak ada elemen yang dapat dihapus, menyimpan alamat elemen terakhir ke P, mengubah L.Last untuk menunjuk ke elemen sebelumnya, jika L.Last tidak nullptr, atur L.Last->next menjadi nullptr, jika L.Last menjadi nullptr, set L.First juga menjadi nullptr, atur P->next dan P->prev menjadi nullptr untuk menghindari dangling pointer, dan jika *list* kosong maka cetak pesan bahwa tidak ada elemen yang dihapus.

```
void deleteLast(List &L, address &P) {
    if (L.Last != nullptr) {
        P = L.Last;
        L.Last = L.Last->prev;
        if (L.Last != nullptr) {
            L.Last->next = nullptr;
        } else {
            L.First = nullptr;
        }
        P->next = nullptr;
        P->prev = nullptr;
    } else {
        cout << "List kosong, tidak ada elemen yang dihapus." << endl;
    }
}
```

4. Membuat fungsi deleteAfter dengan memastikan prec bukan nullptr dan Prec->next juga tidak nullptr, menyimpan alamat elemen setelah Prec, mengubah pointer next dari Prec untuk melewati elemen P, jika ada elemen setelah P maka update P->next->prev untuk menunjuk ke Prec, atur P->next dan P->prev menjadi nullptr untuk menghindari dangling pointer, dan jika tidak ada elemen yang dapat dihapus

setelah Prec maka cetak pesan bahwa tidak ada elemen yang dihapus.

```
void deleteAfter(address Prec, address &P) {
    if (Prec != nullptr && Prec->next != nullptr) {
        P = Prec->next;
        Prec->next = P->next;
        if (P->next != nullptr) {
            P->next->prev = Prec;
        }
        P->next = nullptr;
        P->prev = nullptr;
    } else {
        cout << "Tidak ada elemen yang dapat dihapus setelah Prec." << endl;
    }
}
```

5. Mendeklarasikan variabel, memanggil fungsi deleteFirst, dan menampilkan informasi setelah penghapusan.

```
    address deletedFirst;
    deleteFirst(L, deletedFirst);
    cout << "Setelah menghapus elemen pertama:" << endl;
    printInfo(L);
```

6. Berikut merupakan output dari fungsi tersebut.

```
Setelah menghapus elemen pertama:
Data List 1
no polisi : 5006
warna     : Hitam
tahun     : 2017
Data List 2
no polisi : 2525
warna     : Biru
tahun     : 2025
Data List 3
no polisi : 1414
warna     : Merah
tahun     : 2023
```

7. Mendeklarasikan variabel, memanggil fungsi deleteLast, dan menampilkan informasi setelah penghapusan.

```
    address deletedLast;
    deleteLast(L, deletedLast);
    cout << "Setelah menghapus elemen terakhir:" << endl;
    printInfo(L);
```

8. Berikut merupakan output dari fungsi tersebut.

```
Setelah menghapus elemen terakhir:
Data List 1
no polisi : 2199
warna     : Putih
tahun     : 2023
Data List 2
no polisi : 5006
warna     : Hitam
tahun     : 2017
Data List 3
no polisi : 2525
warna     : Biru
tahun     : 2025
```

9. Mengecek apakah *list* kosong atau tidak, mendeklarasikan variabel, memanggil fungsi deleteAfter untuk menghapus elemen setelah elemen pertama, menampilkan informasi setelah penghapusan.

```
    if (L.First != nullptr) {
        address deletedAfter;
        deleteAfter(L.First, deletedAfter);
        cout << "Setelah menghapus elemen setelah elemen pertama:" << endl;
        printInfo(L);
    }
    address P = L.First;
    while (P != nullptr) {
        address next = P->next;
        dealokasi(P);
        P = next;
    }
```

10. Berikut merupakan output dari fungsi tersebut.

```
Setelah menghapus elemen setelah elemen pertama:
Data List 1
no polisi : 2199
warna     : Putih
tahun     : 2023
Data List 2
no polisi : 2525
warna     : Biru
tahun     : 2025
Data List 3
no polisi : 1414
warna     : Merah
tahun     : 2023
```

5. Kesimpulan

Pada praktikum ini, telah dipelajari dan diimplementasikan *Double Linked*

List menggunakan bahasa pemrograman C++. Struktur data ini memungkinkan penyimpanan elemen dalam bentuk node yang saling terhubung, di mana setiap node memiliki dua pointer yang mengarah ke node berikutnya dan sebelumnya, sehingga memungkinkan navigasi dalam dua arah. Praktikum ini berhasil mengimplementasikan fungsi untuk menambah elemen pada berbagai posisi, seperti di awal, di akhir, dan di tengah *list*, melalui fungsi insertLast dan insertAfter. Selain itu, dikembangkan pula fungsi untuk menghapus elemen dari *list*, seperti deleteFirst, deleteLast, dan deleteAfter, yang memastikan pembaruan pointer agar struktur *list* tetap terjaga. Pencarian elemen juga menjadi fokus, di mana fungsi findElm digunakan untuk mencari elemen berdasarkan kriteria tertentu, seperti nomor polisi kendaraan. Selama praktikum, dipelajari pentingnya pengelolaan memori, termasuk penggunaan fungsi dealokasi untuk membersihkan elemen yang tidak lagi diperlukan, guna menghindari kebocoran memori. Secara keseluruhan, praktikum ini memberikan pemahaman yang lebih dalam tentang manipulasi data dalam struktur data dinamis, serta menekankan fleksibilitas *Double Linked List* dibandingkan dengan array statis. Penguasaan struktur data ini merupakan keterampilan penting yang akan mendukung pemahaman konsep-konsep lebih kompleks dalam algoritma dan pemrograman di masa depan.