LAPORAN PRAKTIKUM

Modul 06

“DOUBLE LINKED LIST”

Disusun Oleh:

Faishal Arif Setiawan 231104066

Kelas:

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

2.LANDASAN TEORI

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*).

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.

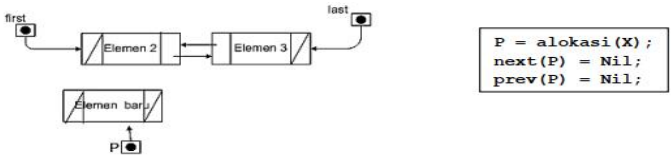
4. *Prev* : *pointer* pada elemen sebagai *successor* yang menunjuk pada elemen dibelakangnya.

2.1 INSERT

INSER FIRST

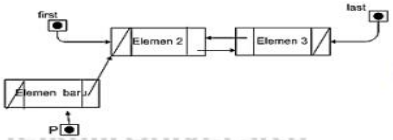
proses menambahkan elemen baru di awal (*head*) dari list.

Langkah Langkah insert fisrt:



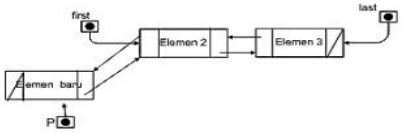

```
P = alokasi(X);
next(P) = Nil;
prev(P) = Nil;
```

Gambar 6-3 *Double Linked list Insert First* 1



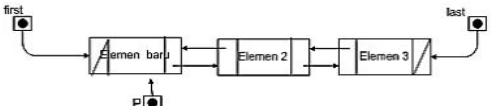

```
next(P) = first(L);
```

Gambar 6-4 *Double Linked list Insert First* 2




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

Gambar 6-5 *Double Linked list Insert First* 3

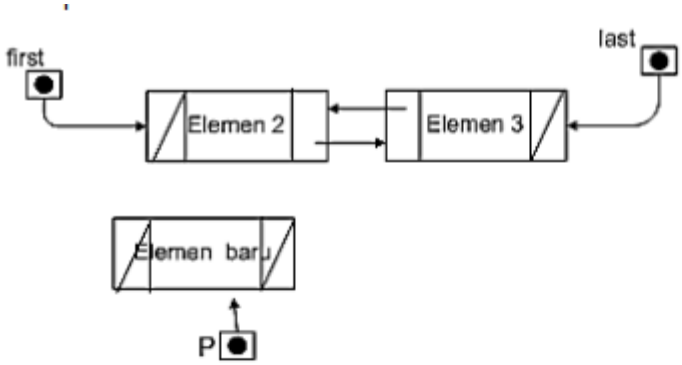



```
first(L) = P;
```

inser last

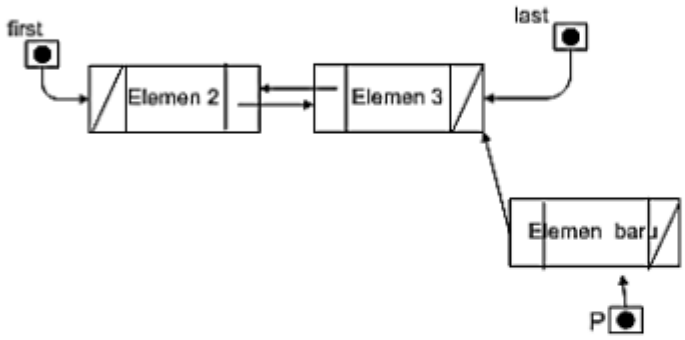
proses menambahkan elemen baru di akhir (tail) dari list

Langkah Langkah insert last:



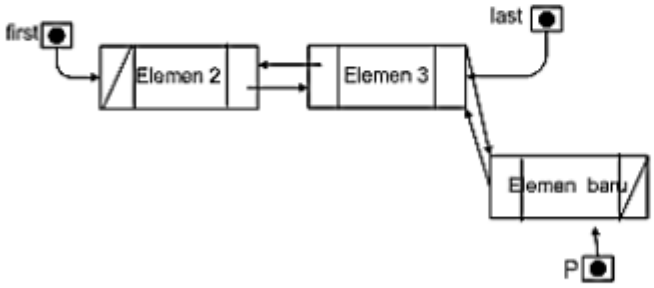

```
P = alokasi(X);
next(P) = Nil;
last(P) = Nil;
```

Gambar 6-7 *Double Linked list Insert Last* 1



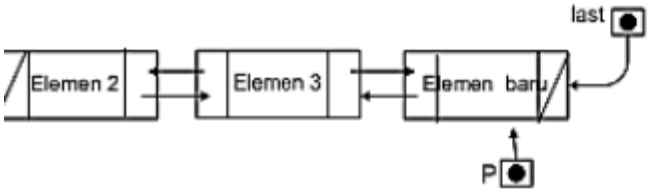

```
prev(P) = last(L);
```

Gambar 6-8 *Double Linked list Insert Last* 2




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

Gambar 6-9 *Double Linked list Insert Last* 3

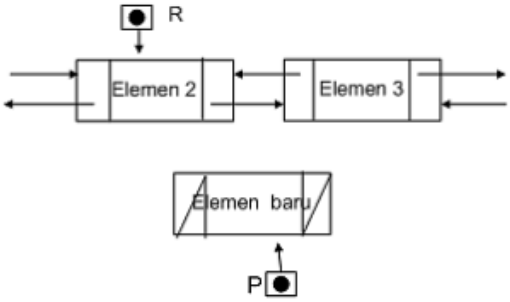



```
last(L) = P;
```

Insert after

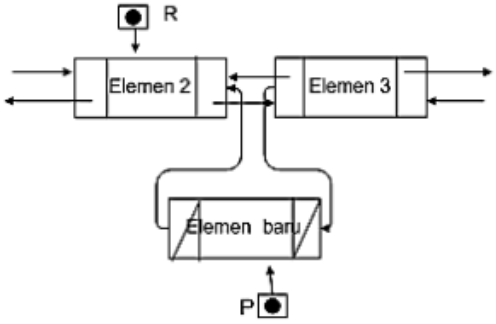
proses menambahkan elemen baru setelah elemen tertentu di dalam list.

Langkah Langkah insert after:



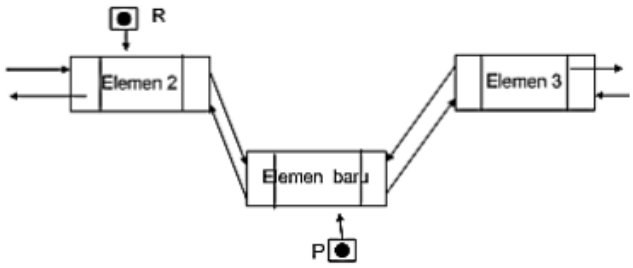

```
P = alokasi(X);
next(P) = Nil;
prev(P) = Nil;
```

Gambar 6-11 *Double Linked list Insert After* 1




```
next(P) = next(R);
prev(P) = R;
```

Gambar 6-12 *Double Linked list Insert After* 2




```
prev(next(R)) = P;
prev(R) = P;
```

insert before

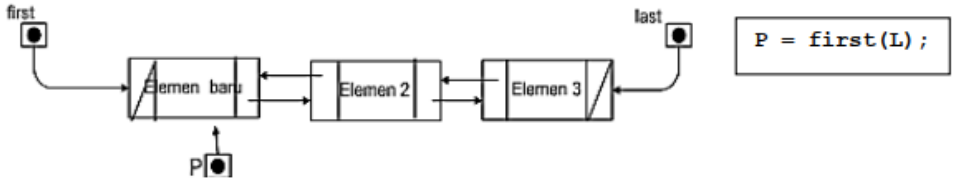
proses menambahkan elemen baru sebelum elemen tertentu di dalam list.

2.1.1 DELETE

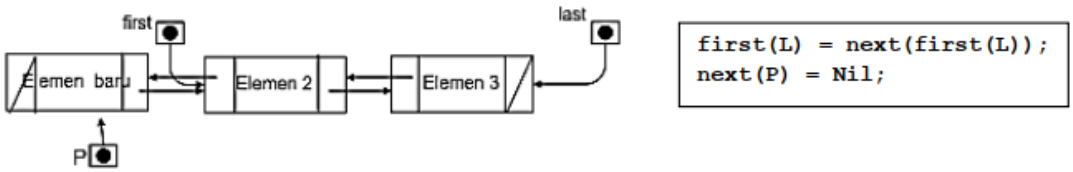
DELETE FIRST

proses menghapus elemen pertama (head) dari list. Pada double linked list.

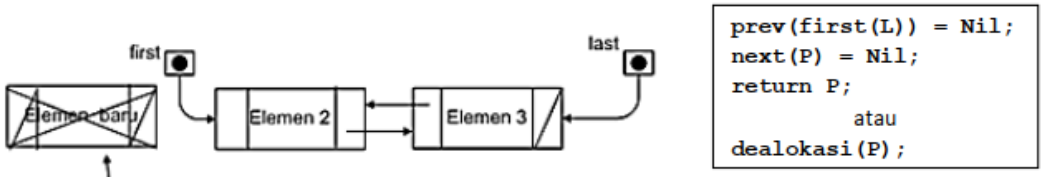
Langkah Langkah delete first:



Gambar 6-14 *Double Linked list Delete First* 1



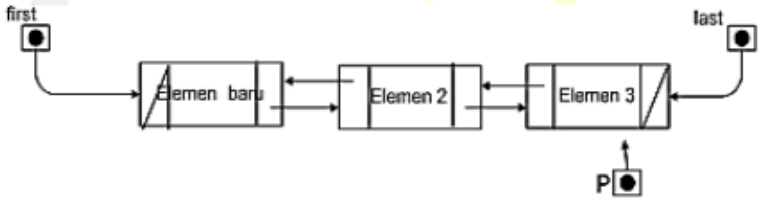
Gambar 6-15 *Double Linked list Delete First* 2



DELETE LAST

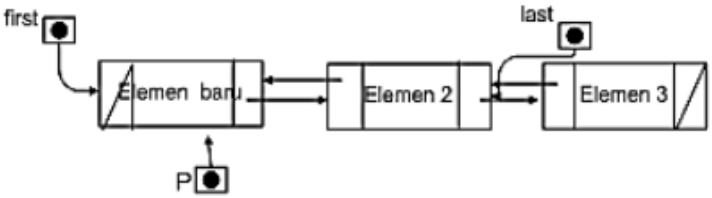
Proses penghapusan elemen akhir(tail)dari list.

Langkah Langkah delete first:



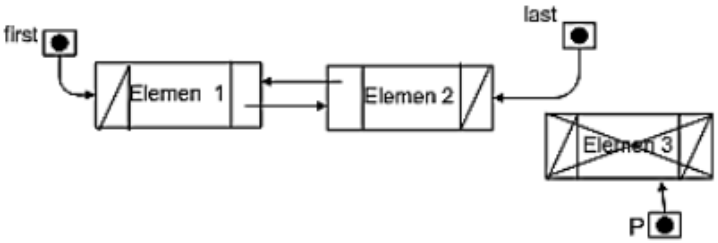

```
P = last(L);
```

Gambar 6-17 *Double Linked list Delete Last* 1




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

Gambar 6-18 *Double Linked list Delete Last* 2




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

DELETE AFTER

proses menghapus elemen tertentu yang berada setelah elemen yang ditentukan dalam list.

DELETE BEFORE

menghapus elemen sebelum elemen tertentu

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*.

3.GUIDED

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

class Node {
public:
    int data;
    Node* prev;
    Node* next;
};

class DoublyLinkedList {
public:
    Node* head;
    Node* tail;

    // Constructor untuk inisialisasi head dan tail
    DoublyLinkedList() {
        head = nullptr;
        tail = nullptr;
    }

    // Fungsi untuk menambahkan elemen di depan list
    void insert(int data) {
        Node* newNode = new Node;
        newNode->data = data;
        newNode->prev = nullptr;
        newNode->next = head;

        if (head != nullptr) {
            head->prev = newNode;
        } else {
            tail = newNode; // Jika list kosong, tail juga mengarah ke node baru
        }
        head = newNode;
    }

    // Fungsi untuk menghapus elemen dari depan list
    void deleteNode() {
        if (head == nullptr) {
            return; // Jika list kosong
        }
        Node* temp = head;
        head = head->next;
        if (head != nullptr) {
            head->prev = nullptr;
        } else {
            tail = nullptr; // Jika hanya satu elemen di list
        }
        delete temp; // Hapus elemen
    }

    // Fungsi untuk mengupdate data di list
    bool update(int oldData, int newData) {
        Node* current = head;
        while (current != nullptr) {
            if (current->data == oldData) {
                current->data = newData;
                return true; // Jika data ditemukan dan diupdate
            }
            current = current->next;
        }
        return false; // Jika data tidak ditemukan
    }

    // Fungsi untuk menghapus semua elemen di list
    void deleteAll() {
        Node* current = head;
        while (current != nullptr) {
            Node* temp = current;
            current = current->next;
            delete temp;
        }
        head = nullptr;
        tail = nullptr;
    }

    // Fungsi untuk menampilkan semua elemen di list
    void display() {
        Node* current = head;
        while (current != nullptr) {
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }
};

int main() {
    DoublyLinkedList list;
    while (true) {
        cout << "1. Add data" << endl;
        cout << "2. Delete data" << endl;
        cout << "3. Update data" << endl;
        cout << "4. Clear data" << endl;
        cout << "5. Display data" << endl;
        cout << "6. Exit" << endl;

        int choice;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1: {
                int data;
                cout << "Enter data to add: ";
                cin >> data;
                list.insert(data);
                break;
            }
            case 2: {
                list.deleteNode();
                break;
            }
            case 3: {
                int oldData, newData;
                cout << "Enter old data: ";
                cin >> oldData;
                cout << "Enter new data: ";
                cin >> newData;
                bool updated = list.update(oldData, newData);
                if (!updated) {
                    cout << "Data not found" << endl;
                }
                break;
            }
            case 4: {
                list.deleteAll();
                break;
            }
            case 5: {
                list.display();
                break;
            }
            case 6: {
                return 0;
            }
            default: {
                cout << "Invalid choice" << endl;
                break;
            }
        }
    }
    return 0;
}
```

OUTPUT:

```
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 1
Enter data to add: 3
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 1
Enter data to add: 4
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice:
```

4.UNGUIDED

1.doublelist.h

```h
#ifndef DOUBLELIST_H
#define DOUBLELIST_H

#include <string>
using namespace std;

struct infotype {
    string nopol;
    string warna;
    int tahun;
};

struct ElmList {
    infotype info;
    ElmList* next;
    ElmList* prev;
};

typedef ElmList* address;

struct List {
    address first;
    address last;
};

// Prototipe fungsi dan prosedur
void createList(List &L);
address allocate(infotype x);
void deallocate(address &P);
void printInfo(const List &L);
void insertLast(List &L, address P);
bool isNopolExists(const List &L, const string &nopol);  // Pastikan ini ada di file header

#endif
```

Doublelist.cpp

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

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

address allocate(infotype x) {
    address newElement = new ElmList;
    newElement->info = x;
    newElement->next = nullptr;
    newElement->prev = nullptr;
    return newElement;
}

void deallocate(address &P) {
    delete P;
    P = nullptr;
}

bool isNopolExists(const List &L, const string &nopol) {
    address P = L.first;
    while (P != nullptr) {
        if (P->info.nopol == nopol) {
            return true;
        }
        P = P->next;
    }
    return false;
}

void printInfo(const List &L) {
    address P = L.first;
    int index = 1;
    cout << "DATA LIST " << index++ << endl;
    while (P != nullptr) {
        cout << "no polisi : " << P->info.nopol << endl;
        cout << "warna     : " << P->info.warna << endl;
        cout << "tahun     : " << P->info.tahun << endl;
        cout << "--------------------------" << endl;
        P = P->next;
    }
}

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

Main.cpp

```
int main() {
    List L;
    createList(L);
    char tambahLagi;

    do {
        infotype kendaraan;

        cout << "masukkan nomor polisi: ";
        cin >> kendaraan.nopol;

        // Cek apakah nomor polisi sudah terdaftar
        if (isNopolExists(L, kendaraan.nopol)) {
            cout << "nomor polisi sudah terdaftar" << endl;
            continue; // Kembali ke awal loop jika nomor polisi sudah ada
        }

        cout << "masukkan warna kendaraan: ";
        cin >> kendaraan.warna;
        cout << "masukkan tahun kendaraan: ";
        cin >> kendaraan.tahun;

        // Masukkan data kendaraan ke dalam list
        insertLast(L, allocate(kendaraan));

        cout << "Apakah Anda ingin menambah kendaraan lagi? (y/n): ";
        cin >> tambahLagi;

    } while (tambahLagi == 'y' || tambahLagi == 'Y');

    // Tampilkan data kendaraan
    printInfo(L);

    return 0;
}
```

Output:

```
masukkan nomor polisi: D001
masukkan warna kendaraan: hitam
masukkan tahun kendaraan: 70
Apakah Anda ingin menambah kendaraan lagi? (y/n): y
masukkan nomor polisi: D003
masukkan warna kendaraan: putih
masukkan tahun kendaraan: 70
Apakah Anda ingin menambah kendaraan lagi? (y/n): y
masukkan nomor polisi: D001
nomor polisi sudah terdaftar
masukkan nomor polisi: D004
masukkan warna kendaraan: kuning
masukkan tahun kendaraan: 90
Apakah Anda ingin menambah kendaraan lagi? (y/n): n
DATA LIST 1
no polisi : D001
warna     : hitam
tahun     : 70
--------------------------
no polisi : D003
warna     : putih
tahun     : 70
--------------------------
no polisi : D004
warna     : kuning
tahun     : 90
--------------------------
```

2.

```cpp
#include <iostream>
#include <string>

using namespace std;

struct Mobil {
    string nomorPolisi;
    string warna;
    int tahun;
    Mobil* next;
};

Mobil* findElm(Mobil* head, const string& targetNomor) {
    Mobil* current = head;
    while (current != nullptr) {
        if (current->nomorPolisi == targetNomor) {
            return current;
        }
        current = current->next;
    }
    return nullptr;
}

int main() {
    Mobil* mobil1 = new Mobil{"B001", "hitam", 90, nullptr};
    Mobil* mobil2 = new Mobil{"B002", "biru", 85, nullptr};
    Mobil* mobil3 = new Mobil{"B003", "merah", 95, nullptr};

    mobil1->next = mobil2;
    mobil2->next = mobil3;

    string targetNomor;
    cout << "Masukkan Nomor Polisi yang dicari : ";
    cin >> targetNomor;

    Mobil* mobilDitemukan = findElm(mobil1, targetNomor);
    if (mobilDitemukan != nullptr) {
        cout << "Nomor Polisi : " << mobilDitemukan->nomorPolisi << endl;
        cout << "Warna        : " << mobilDitemukan->warna << endl;
        cout << "Tahun        : " << mobilDitemukan->tahun << endl;
    } else {
        cout << "Nomor Polisi tidak ditemukan." << endl;
    }

    delete mobil1;
    delete mobil2;
    delete mobil3;

    return 0;
}
```

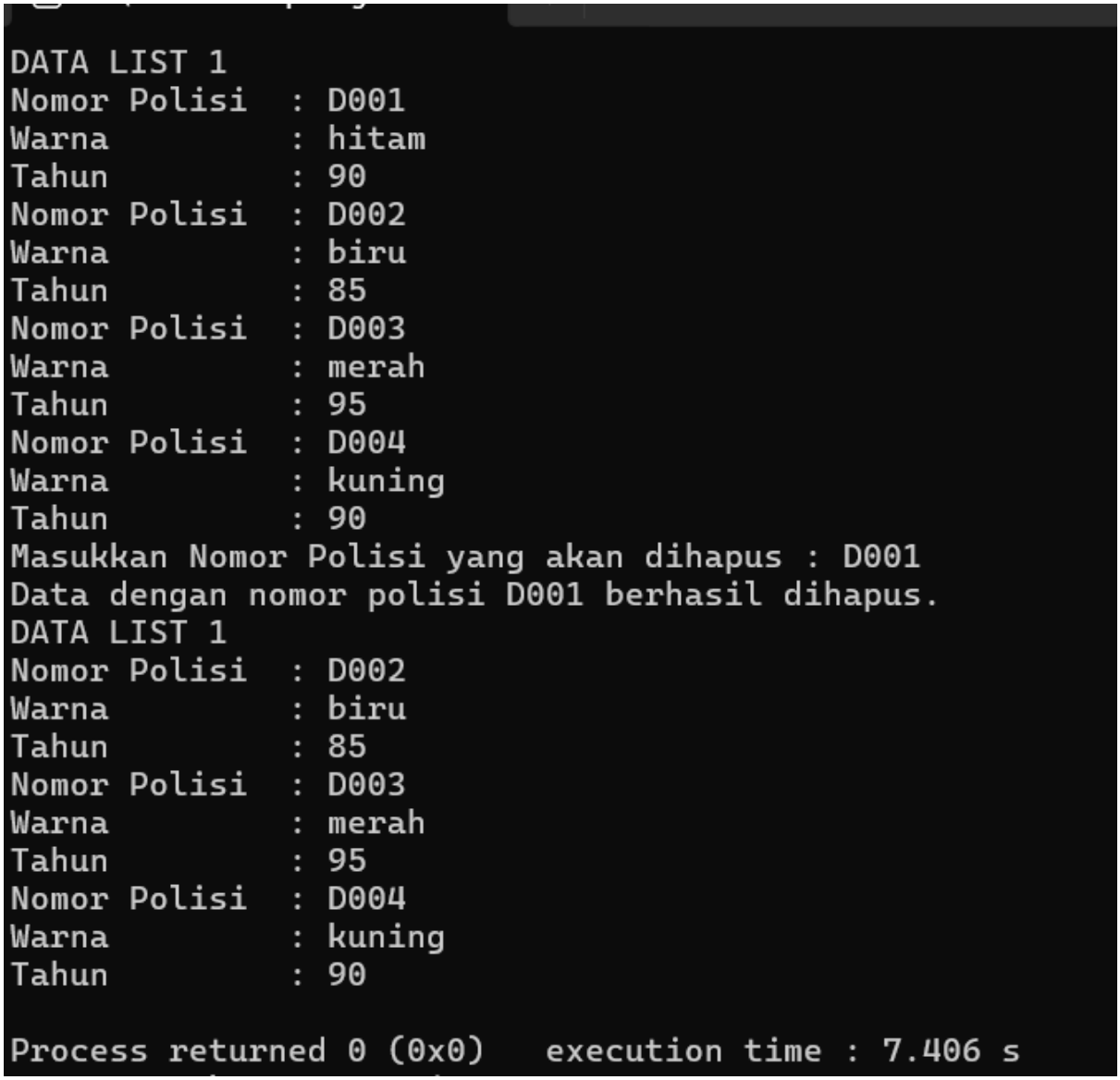
Output:

```
lasukkan Nomor Polisi yang dicari : D001
lomor Polisi : D001
Warna        : hitam
Tahun        : 90
```

3.

```cpp
#include <iostream>
#include <string>

using namespace std;

// Struktur untuk informasi mobil
struct Mobil {
    string nomorPolisi;
    string warna;
    int tahun;
    Mobil* next;
};

// Prosedur untuk menghapus elemen pertama
void deleteFirst(Mobil*& head, Mobil*& p) {
    if (head != nullptr) {
        p = head;
        head = head->next;
        delete p;
    }
}

// Prosedur untuk menghapus elemen terakhir
void deleteLast(Mobil*& head, Mobil*& p) {
    if (head == nullptr) return;

    if (head->next == nullptr) {
        p = head;
        head = nullptr;
        delete p;
    } else {
        Mobil* temp = head;
        while (temp->next->next != nullptr) {
            temp = temp->next;
        }
        p = temp->next;
        temp->next = nullptr;
        delete p;
    }
}

// Prosedur untuk menghapus elemen setelah elemen tertentu
void deleteAfter(Mobil* prec, Mobil*& p) {
    if (prec != nullptr && prec->next != nullptr) {
        p = prec->next;
        prec->next = p->next;
        delete p;
    }
}

// Prosedur untuk menghapus elemen berdasarkan nomor polisi
void deleteByPlate(Mobil*& head, const string& targetNomor) {
    Mobil* p = nullptr;

    if (head != nullptr && head->nomorPolisi == targetNomor) {
        deleteFirst(head, p);
    } else {
        Mobil* temp = head;
        while (temp->next != nullptr && temp->next->nomorPolisi != targetNomor) {
            temp = temp->next;
        }
        if (temp->next != nullptr) {
            deleteAfter(temp, p);
        }
    }

    if (p != nullptr) {
        cout << "Data dengan nomor polisi " << targetNomor << " berhasil dihapus." << endl;
    } else {
        cout << "Data dengan nomor polisi " << targetNomor << " tidak ditemukan." << endl;
    }
}

// Fungsi untuk menampilkan data list
void printList(Mobil* head) {
    Mobil* temp = head;
    cout << "DATA LIST :" << endl;
    while (temp != nullptr) {
        cout << "Nomor Polisi  : " << temp->nomorPolisi << endl;
        cout << "Warna         : " << temp->warna << endl;
        cout << "Tahun         : " << temp->tahun << endl;
        temp = temp->next;
    }
}

int main() {
    // Membuat linked list dari beberapa mobil
    Mobil* mobil1 = new Mobil{"D001", "hitam", 90, nullptr};
    Mobil* mobil2 = new Mobil{"D002", "biru", 85, nullptr};
    Mobil* mobil3 = new Mobil{"D003", "merah", 95, nullptr};
    Mobil* mobil4 = new Mobil{"D004", "kuning", 90, nullptr};

    mobil1->next = mobil2;
    mobil2->next = mobil3;
    mobil3->next = mobil4;

    // Menampilkan list sebelum penghapusan
    printList(mobil1);

    // Input nomor polisi yang ingin dihapus
    string targetNomor;
    cout << "Masukkan Nomor Polisi yang akan dihapus : ";
    cin >> targetNomor;

    // Menghapus elemen dengan nomor polisi yang diberikan
    deleteByPlate(mobil1, targetNomor);

    // Menampilkan list setelah penghapusan
    printList(mobil1);

    // Membersihkan memori
    while (mobil1 != nullptr) {
        Mobil* temp = mobil1;
        mobil1 = mobil1->next;
        delete temp;
    }

    return 0;
}
```

Output:



5.KESIMPULAN

Dalam praktikum ini, kami mempelajari double linked list, yaitu struktur data yang memungkinkan penyimpanan dan pengelolaan data dengan dua arah, berkat adanya dua pointer pada setiap simpul yang menghubungkan ke simpul sebelumnya dan berikutnya. Kami melakukan berbagai operasi dasar, seperti penyisipan, penghapusan (termasuk operasi delete before), dan traversal, yang menunjukkan keunggulan double linked list dalam hal fleksibilitas dan efisiensi pengelolaan data dibandingkan dengan single linked list.