I. COVER

LAPORAN PRAKTIKUM

Modul 06

DOUBLE LINKED LIST BAGIAN SATU

Disusun Oleh:

Haza Zaidan Zidna Fann

2311104056

Dosen :

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

II. TUJUAN

Memahami konsep modul *linked list*.
Mengaplikasikan konsep *double linked list* dengan menggunakan *pointer* dan dengan bahasa C++

III. Landasan Teori

Linked List: Struktur data linier yang terdiri dari node-node yang saling terhubung. Setiap node menyimpan data dan pointer ke node berikutnya. Tidak seperti array, linked list tidak membutuhkan memori berurutan.

Double Linked List (DLL): Tipe linked list di mana setiap node memiliki dua pointer satu menunjuk ke node berikutnya (next) dan satu lagi ke node sebelumnya (prev). Ini memungkinkan penelusuran dalam dua arah.

Struktur Node: Setiap node dalam DLL terdiri dari data, pointer next untuk node berikutnya, dan pointer prev untuk node sebelumnya.

Operasi Dasar DLL:
Insert (Penyisipan): Menambahkan elemen di awal (insert first), akhir (insert last), atau setelah node tertentu (insert after).

Delete (Penghapusan): Menghapus elemen dari awal (delete first), akhir (delete last), atau setelah node tertentu (delete after).

Traversal: Menelusuri dan menampilkan semua elemen dalam list.

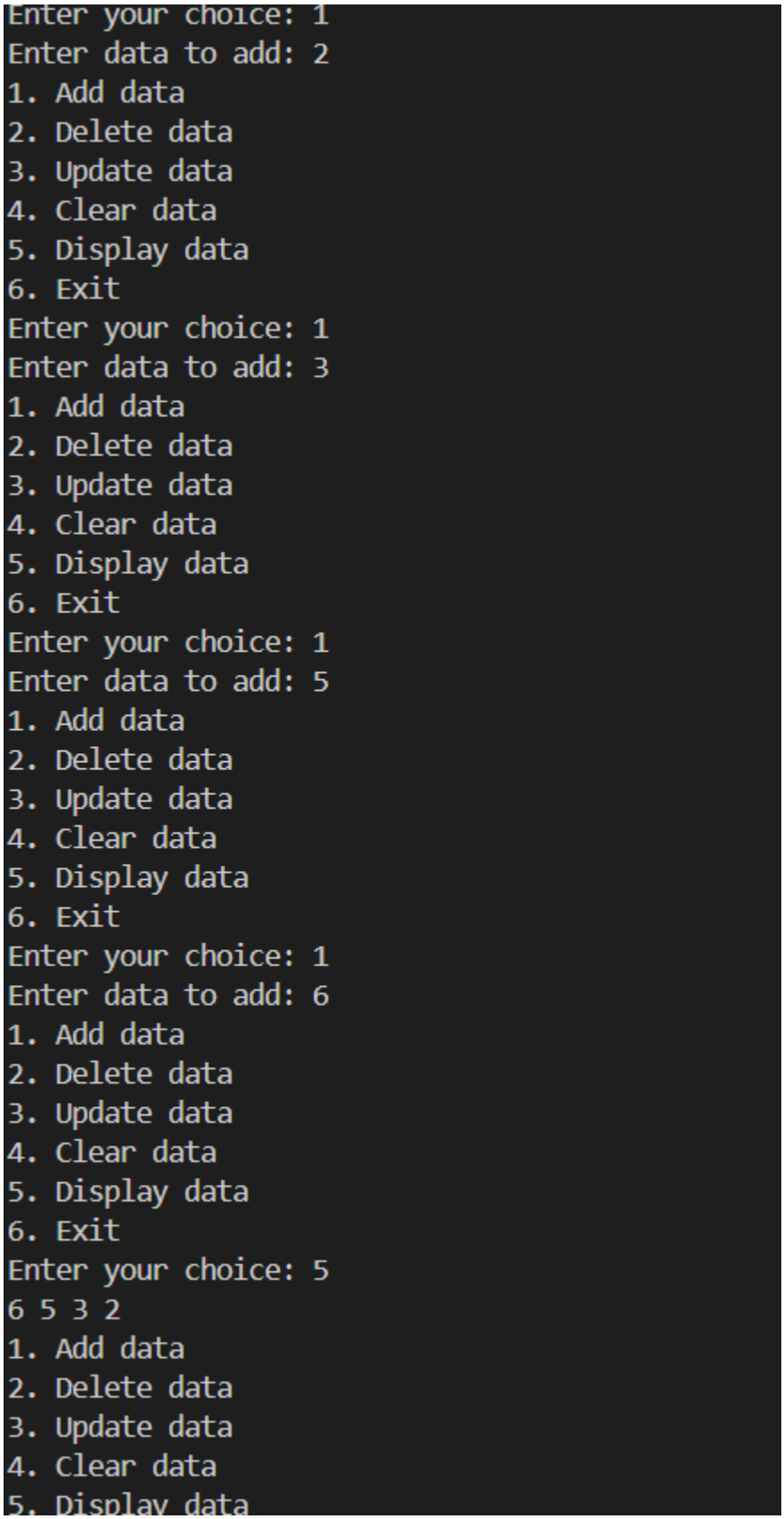
Search (Pencarian): Mencari elemen tertentu dalam list berdasarkan kriteria, misalnya nomor polisi.

Alokasi dan Dealokasi: Alokasi memesan memori untuk node baru, sedangkan dealokasi membebaskan memori saat node dihapus untuk mencegah kebocoran memori.

Kelebihan DLL: Double linked list memungkinkan penelusuran dua arah serta penyisipan dan penghapusan elemen di berbagai posisi lebih efisien dibandingkan singly linked list.

IV. GUIDED

```cpp
#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;
}
```



Struktur Kelas

Node: Kelas ini merepresentasikan node dalam list, yang menyimpan data int dan pointer prev serta next untuk menunjuk node sebelumnya dan berikutnya.
DoubleLinkedList: Kelas ini mengelola list dengan pointer head (awal list) dan tail (akhir list), serta berisi metode untuk berbagai operasi pada list.

Metode dalam Kelas DoubleLinkedList
insert(int data): Menambahkan node baru dengan data di awal list. Jika list kosong, head dan tail diarahkan ke node baru.
deleteNode(): Menghapus node pertama. Jika hanya ada satu node, tail juga diatur ke nullptr.
update(int oldData, int newData): Mencari node dengan nilai oldData dan mengubahnya menjadi newData. Mengembalikan true jika berhasil, false jika tidak ditemukan.
deleteAll(): Menghapus seluruh node dalam list, membebaskan memori.
display(): Menampilkan semua data dalam list dari head hingga tail.

Fungsi main()
Program memiliki loop utama yang menampilkan menu pilihan:
Add data: Menambahkan data ke awal list.
Delete data: Menghapus data dari depan list.
Update data: Memperbarui nilai tertentu dalam list.
Clear data: Menghapus seluruh elemen dalam list.
Display data: Menampilkan seluruh elemen di list.
Exit: Keluar dari program.

Alur Program
Tampilkan Menu.
Minta Input Pilihan Pengguna.
Eksekusi Operasi Sesuai Pilihan:
Menambah data: Memanggil insert.
Menghapus data: Memanggil deleteNode.
Memperbarui data: Memanggil update.
Menghapus semua data: Memanggil deleteAll.
Menampilkan data: Memanggil display.
Keluar: Mengakhiri program.

V. UNGUIDED

Membuat ADT Double Linked List

```cpp
#ifndef DOUBLELIST_H
#define DOUBLELIST_H

#include <string>
using namespace std;

// Struktur data untuk menyimpan informasi kendaraan
struct kendaraan {
    string nopol;   // Nomor polisi
    string warna;   // Warna kendaraan
    int thnBuat;    // Tahun pembuatan kendaraan
};

typedef kendaraan infotype;
struct ElmList;    // Deklarasi forward untuk struct ElmList
typedef ElmList* address; // Define `address` sebagai pointer ke ElmList

// Struktur elemen dalam linked list
struct ElmList {
    infotype info;    // Informasi kendaraan
    address next;     // Pointer ke elemen berikutnya
    address prev;     // Pointer ke elemen sebelumnya
};

// Struktur untuk mengelola linked list
struct List {
    address First;   // Pointer ke elemen pertama
    address Last;    // Pointer ke elemen terakhir
};

// Deklarasi fungsi-fungsi untuk manipulasi linked list
void CreateList(List &L);
address alokasi(infotype x);
void dealokasi(address &P);
void printInfo(const List &L);
void insertLast(List &L, address P);

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

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

#endif // DOUBLELIST_H

```

File doublelist.h

Struktur Data

1. **struct kendaraan**:
    - Menyimpan informasi kendaraan dengan tiga atribut: nopol (nomor polisi), warna, dan thnBuat (tahun pembuatan).
    - Tipe kendaraan diberi alias infotype untuk mempermudah pemakaian.
2. **struct ElmList**:
    - Mewakili satu elemen/node dari linked list dengan:
        - **info**: Menyimpan data tipe infotype.
        - **next**: Pointer ke node berikutnya.
        - **prev**: Pointer ke node sebelumnya.
    - address didefinisikan sebagai pointer ke ElmList.
3. **struct List**:
    - Menyimpan pointer ke elemen pertama (First) dan terakhir (Last) dari linked list.

Fungsi-fungsi

1. **void CreateList(List &L)**:
    - Menginisialisasi linked list dengan mengatur First dan Last ke nullptr, menandakan list kosong.
2. **address alokasi(infotype x)**:
    - Mengalokasikan node baru dengan informasi yang diberikan (x) dan mengembalikan alamat node baru tersebut.
3. **void dealokasi(address &P)**:
    - Membebaskan memori node yang ditunjuk oleh pointer P.
4. **void printInfo(const List &L)**:
    - Menampilkan seluruh data dari elemen-elemen dalam list.
5. **void insertLast(List &L, address P)**:
    - Menambahkan node P di akhir list L.
6. **address findElm(const List &L, const infotype &x)**:
    - Mencari elemen dengan data x dalam list L. Jika ditemukan, mengembalikan pointer ke elemen tersebut; jika tidak, mengembalikan nullptr.
7. **Fungsi Hapus**:
    - **void deleteFirst(List &L, address &P)**: Menghapus elemen pertama dari list dan menyimpan alamatnya di P.
    - **void deleteLast(List &L, address &P)**: Menghapus elemen terakhir dari list dan menyimpan alamatnya di P.

- **void deleteAfter(address Prec, address &P)**: Menghapus elemen setelah Prec dan menyimpan alamatnya di P.

Alur Umum

- **Inisialisasi** dengan CreateList.
- **Penambahan elemen** melalui insertLast.
- **Pencarian elemen** menggunakan findElm.
- **Penghapusan elemen** dengan deleteFirst, deleteLast, atau deleteAfter.
- **Dealokasi node** untuk membebaskan memori dengan dealokasi.

```cpp
#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 printInfoReverse(const List &L) {
    address P = L.Last;
    int i = 1;
    cout << "DATA LIST 1" << endl << endl;
    while (P != nullptr) {
        cout << "no polisi    : " << P->info.nopol << endl;
        cout << "warna        : " << P->info.warna << endl;
        cout << "tahun        : " << P->info.thnBuat << endl;
        P = P->prev;
    }
}

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;
    }
}

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;
}

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;
    }
}

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;
    }
}

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;
    }
}

void deleteData(List &L, string nopol) {
    infotype searchData;
    searchData.nopol = nopol;

    address P = findElm(L, searchData);
    if (P != NULL) {
        if (P == L.First) {
            deleteFirst(L, P);
        } else if (P == L.Last) {
            deleteLast(L, P);
        } else {
            deleteAfter(P->prev, P);
        }
        cout << "Data dengan nomor polisi " << nopol << " berhasil dihapus." << endl;
        dealokasi(P);
    } else {
        cout << "Data tidak ditemukan!" << endl;
    }
}

```

file doublelist.cpp

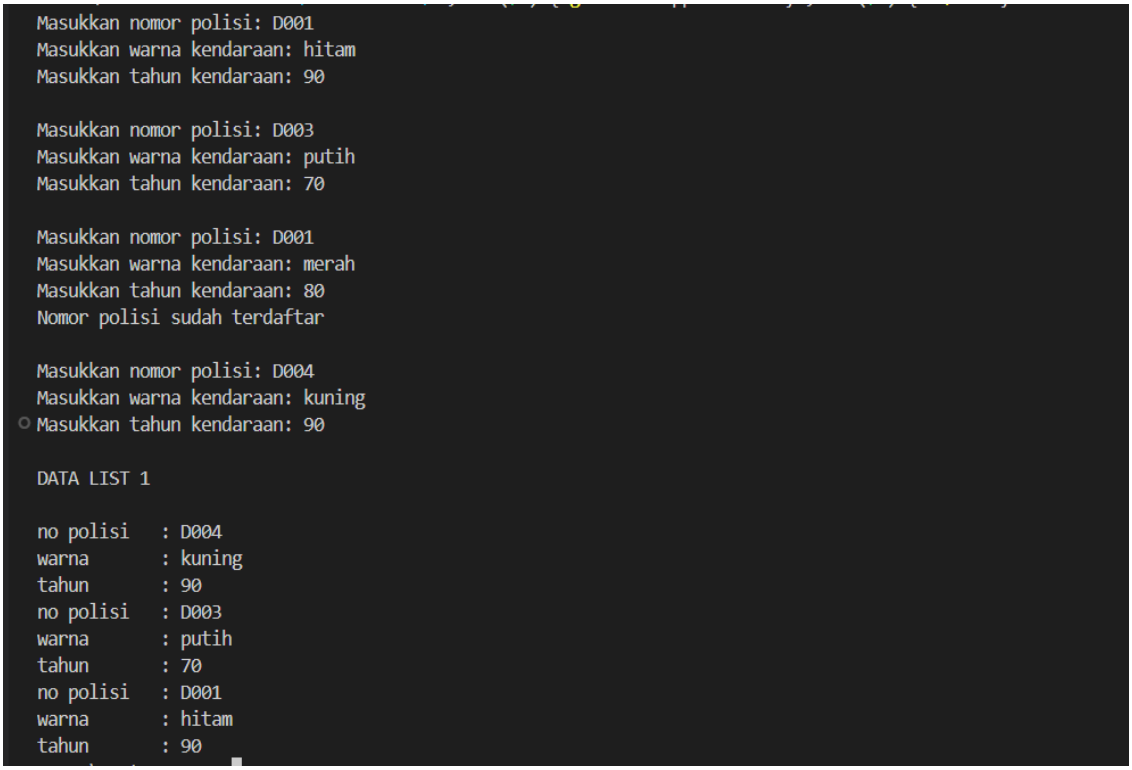
1. **CreateList(List &L)**: Menginisialisasi list kosong dengan mengatur First dan Last ke nullptr.
2. **address alokasi(infotype x)**: Mengalokasikan memori untuk elemen baru P dan menginisialisasinya dengan data x. Mengembalikan pointer ke elemen yang baru dibuat.
3. **void dealokasi(address &P)**: Membebaskan memori dari elemen P dan mengatur P menjadi nullptr.
4. **void printInfoReverse(const List &L)**: Mencetak data kendaraan dalam list dari elemen terakhir (Last) ke awal (First).
5. **void insertLast(List &L, address P)**: Menambahkan elemen P di akhir list. Jika list kosong, P menjadi elemen pertama dan terakhir.
6. **address findElm(const List &L, const infotype &x)**: Mencari elemen dengan nomor polisi x.nopol. Jika ditemukan, mengembalikan alamat elemen tersebut; jika tidak, mengembalikan nullptr.
7. **Fungsi Penghapusan**:
    - **void deleteFirst(List &L, address &P)**: Menghapus elemen pertama dalam list.
    - **void deleteLast(List &L, address &P)**: Menghapus elemen terakhir dalam list.
    - **void deleteAfter(address Prec, address &P)**: Menghapus elemen setelah elemen Prec.
8. **void deleteData(List &L, string nopol)**: Mencari elemen berdasarkan nopol. Jika ditemukan, elemen dihapus menggunakan fungsi penghapusan yang sesuai (pertama, terakhir, atau tengah). Mengeluarkan pesan sesuai hasil pencarian dan penghapusan.

Alur Program

1. **Inisialisasi** list dengan CreateList.
2. **Penambahan** elemen ke akhir list dengan insertLast.
3. **Pencarian** elemen dengan nomor polisi tertentu melalui findElm.
4. **Penghapusan** elemen menggunakan fungsi deleteFirst, deleteLast, deleteAfter, atau deleteData.
5. **Cetak** seluruh elemen secara terbalik dengan printInfoReverse.

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

using namespace std;

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

    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 found = findElm(L, data);
        if (found != nullptr) {
            cout << "Nomor polisi sudah terdaftar" << endl << endl;
            continue; // Skip this iteration and proceed to the next input
        }

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

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

File Main.cpp



1. **CreateList(L)**: Menginisialisasi list kosong.
2. **findElm(L, data)**: Mencari apakah data kendaraan dengan nopol tertentu sudah ada di dalam list.
3. **alokasi(data)**: Mengalokasikan elemen baru yang berisi data kendaraan.
4. **insertLast(L, P)**: Menambahkan elemen baru P di akhir list.
5. **printInfoReverse(L)**: Menampilkan data dalam list dari elemen terakhir hingga pertama.

Alur Program

1. **Inisialisasi** list kosong dengan CreateList(L).
2. **Input Data**: Meminta pengguna untuk memasukkan informasi kendaraan (nomor polisi, warna, dan tahun).
    - **Pengecekan Duplikasi**: Menggunakan findElm untuk memastikan nomor polisi belum ada di list. Jika sudah ada, tampilkan pesan "Nomor polisi sudah terdaftar" dan lewati input tersebut.
    - Jika tidak ada duplikasi, alokasikan elemen baru dengan alokasi(data) dan tambahkan elemen tersebut ke akhir list dengan insertLast.
3. **Cetak List Terbalik**: Setelah semua input, tampilkan seluruh data kendaraan secara terbalik menggunakan printInfoReverse(L).

Mencari elemen

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

using namespace std;

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

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

    address foundElement = findElm(L, searchData);
    cout << endl;
    if (foundElement != nullptr) {
        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;
    }

    cout << endl;

    return 0;

}
```

```
Masukkan nomor polisi yang ingin dicari: D001

Nomor Polisi  : D001
Warna         : hitam
Tahun         : 90
```

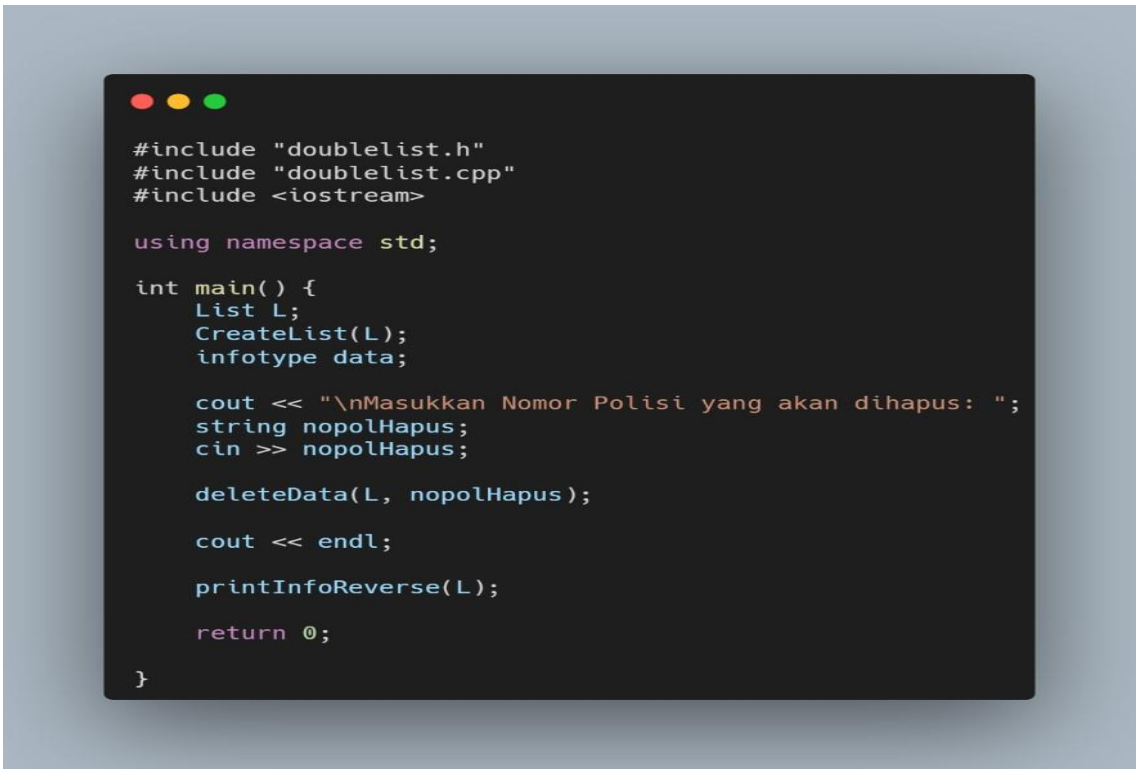
1. **inisialisasi**:
    - List L; mendeklarasikan variabel L sebagai list ganda.
    - CreateList(L); memanggil fungsi untuk membuat atau menginisialisasi list kosong.
2. **Input Data Pencarian**:
    - cout meminta pengguna memasukkan nomor polisi (nopol) yang ingin dicari.
    - cin >> searchData.nopol; menyimpan nomor polisi yang dimasukkan ke dalam searchData.nopol.
3. **Pencarian**:
    - findElm(L, searchData); mencari elemen di list L berdasarkan nomor polisi yang dimasukkan.
    - Hasil pencarian disimpan di foundElement.
4. **Hasil Pencarian**:
    - Jika foundElement ditemukan (tidak nullptr), tampilkan nopol, warna, dan thnBuat.

- Jika tidak ditemukan, tampilkan pesan bahwa kendaraan tidak ada di list.

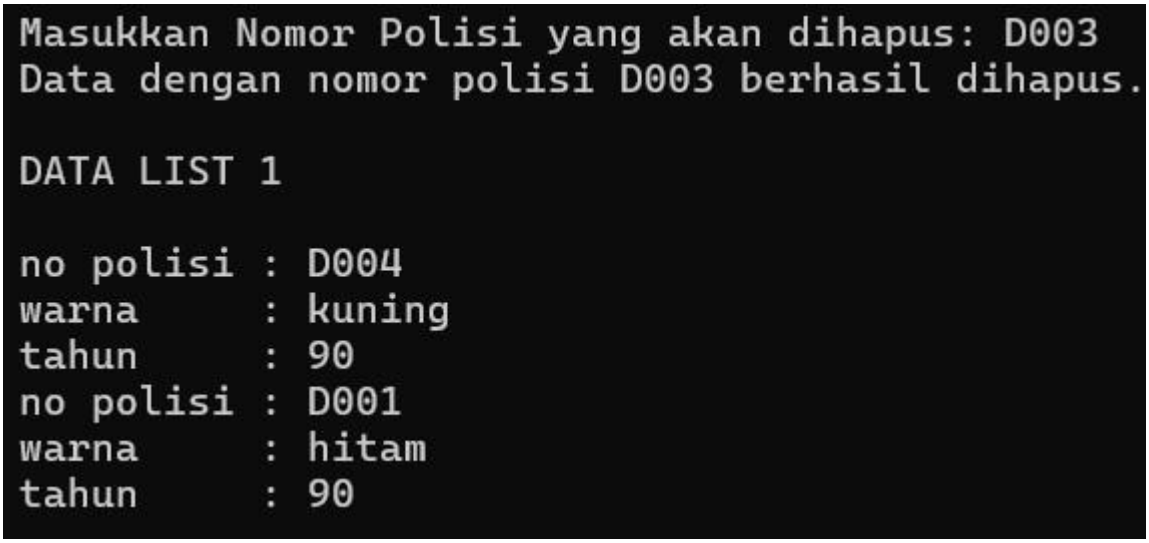
5. **Akhiri Program**:
   - return 0; menandakan akhir program.

   Alur Program: Inisialisasi list → Input nomor polisi → Cari di list → Tampilkan hasil (ditemukan/tidak)

   Menghapus elemen



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

using namespace std;

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

    cout << "\nMasukkan Nomor Polisi yang akan dihapus: ";
    string nopolHapus;
    cin >> nopolHapus;

    deleteData(L, nopolHapus);

    cout << endl;

    printInfoReverse(L);

    return 0;

}
```



```
Masukkan Nomor Polisi yang akan dihapus: D003
Data dengan nomor polisi D003 berhasil dihapus.

DATA LIST 1

no polisi : D004
warna     : kuning
tahun     : 90
no polisi : D001
warna     : hitam
tahun     : 90
```

1. **Inisialisasi**:
   - List L; mendeklarasikan list L.
   - CreateList(L); membuat atau menginisialisasi list kosong L.

2. **Input Data untuk Dihapus**:
    - cout meminta pengguna memasukkan nomor polisi yang akan dihapus.
    - cin >> nopolHapus; menyimpan nomor polisi ke variabel nopolHapus.
3. **Hapus Data**:
    - deleteData(L, nopolHapus); mencari dan menghapus elemen dengan nomor polisi nopolHapus dari list L.
4. **Cetak List Terbalik**:
    - printInfoReverse(L); mencetak semua elemen di list L dari belakang ke depan.
5. **Akhiri Program**:
    - return 0; menandakan akhir program.

Alur Program: Inisialisasi list → Input nomor polisi untuk dihapus → Hapus data dari list → Tampilkan list secara terbalik