LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL VI
DOUBLE LINKED LIST

Disusun Oleh :

NAMA : Muhamad Naufal Ammar

NIM : 103112430036

Dosen

WAHYU ANDI SAPUTRA

PROGRAM STUDI STRUKTUR DATA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Dasar teori dari program ini berlandaskan pada konsep Doubly Linked List (DLL), yaitu struktur data dinamis yang memungkinkan penyimpanan dan manipulasi data secara fleksibel melalui dua pointer pada setiap node, yaitu next untuk menunjuk ke elemen berikutnya dan prev untuk menunjuk ke elemen sebelumnya. Menurut Acharjya, Koley, dan Barman (2022), struktur linked list, termasuk DLL, memiliki keunggulan dalam efisiensi pengelolaan memori dan kemudahan dalam proses penambahan maupun penghapusan elemen dibandingkan struktur data statis seperti array. Selain itu, penelitian oleh Khan et al. (2022) menunjukkan bahwa penerapan Doubly Linked List juga relevan dalam sistem pengenalan pola dan pengolahan data yang membutuhkan akses dua arah, karena kemampuannya dalam mempercepat proses traversal data. Dengan memahami teori ini, mahasiswa dapat mengetahui bagaimana prinsip kerja pointer dan hubungan antar-node berperan penting dalam menciptakan struktur data yang efisien dan adaptif terhadap berbagai kebutuhan komputasi modern..

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

Guided 1

```
#ifndef DOUBLYLIST_H
#define DOUBLYLIST_H

#include <iostream>
using namespace std;

#define Nil NULL

/*deklarasi record dan struktur data Doubly linked list*/
typedef int infotype;
typedef struct elmlist *address;
struct elmlist {
    infotype info;
    address next;
    address prev;
};

/* definisi list: */
/* list kosong jika L.first=Nil */
struct List {
    address first;
    address last;
};

/** Deklarasi fungsi primitif **/
address alokasi(infotype x);
void dealokasi(address &P);
void createList(List &L);
bool isEmpty(List L);

// Insert operations
void insertFirst(List &L, address P);
void insertLast(List &L, address P);
void insertAfter(address Prec, address P);
void insertBefore(address Prec, address P);

// Delete operations
void deleteFirst(List &L, address &P);
void deleteLast(List &L, address &P);
void deleteAfter(address Prec, address &P);
void deleteBefore(address Prec, address &P);

// View operations
void printForward(List L);
void printBackward(List L);

#endif
```

```cpp
#include "DoublyList.h"
// Alokasi memori untuk elemen baru
address alokasi(infotype x) {
    address P = new elmlist;
    P->info = x;
    P->next = Nil;
    P->prev = Nil;
    return P;
}

// Dealokasi memori
void dealokasi(address &P) {
    delete P;
    P = Nil;
}

// Inisialisasi list kosong
void createList(List &L) {
    L.first = Nil;
    L.last = Nil;
}

// Cek apakah list kosong
bool isEmpty(List L) {
    return (L.first == Nil);
}

// INSERT FIRST -
void insertFirst(List &L, address P) {
    if (isEmpty(L)) {
        // Jika list kosong
        L.first = P;
        L.last = P;
    } else {
        // Jika list tidak kosong
        P->next = L.first;      // Step 1: P->next menunjuk first lama
        L.first->prev = P;      // Step 2: first lama->prev menunjuk P
        L.first = P;            // Step 3: first baru = P
    }
}

// INSERT LAST -
void insertLast(List &L, address P) {
    if (isEmpty(L)) {
        // Jika list kosong
        L.first = P;
        L.last = P;
    } else {
        // Jika list tidak kosong
        P->prev = L.last;       // Step 1: P->prev menunjuk last lama
        L.last->next = P;       // Step 2: last lama->next menunjuk P
        L.last = P;             // Step 3: last baru = P
    }
}
```

```
// INSERT AFTER -
void insertAfter(address Prec, address P) {
    if (Prec == Nil || P == Nil) return;

    P->next = Prec->next;       // Step 1: P->next menunjuk elemen setelah Prec
    P->prev = Prec;             // Step 2: P->prev menunjuk Prec

    if (Prec->next != Nil) {
        Prec->next->prev = P;   // Step 3: elemen setelah Prec->prev menunjuk P
    }
    Prec->next = P;             // Step 4: Prec->next menunjuk P
}

// INSERT BEFORE - kebalikan insert after
void insertBefore(address Prec, address P) {
    if (Prec == Nil || P == Nil) return;

    P->next = Prec;             // Step 1: P->next menunjuk Prec
    P->prev = Prec->prev;       // Step 2: P->prev menunjuk elemen sebelum Prec

    if (Prec->prev != Nil) {
        Prec->prev->next = P;   // Step 3: elemen sebelum Prec->next menunjuk P
    }
    Prec->prev = P;             // Step 4: Prec->prev menunjuk P
}

// DELETE FIRST -
void deleteFirst(List &L, address &P) {
    if (isEmpty(L)) {
        P = Nil;
        return;
    }

    P = L.first;                // Step 1: P menunjuk elemen pertama

    if (L.first == L.last) {
        // Hanya ada satu elemen
        L.first = Nil;
        L.last = Nil;
    } else {
        // Lebih dari satu elemen
        L.first = L.first->next;    // Step 2: first menunjuk elemen kedua
        L.first->prev = Nil;        // Step 3: first baru->prev = Nil
    }

    // Isolasi elemen yang dihapus
    P->next = Nil;
    P->prev = Nil;
}
```

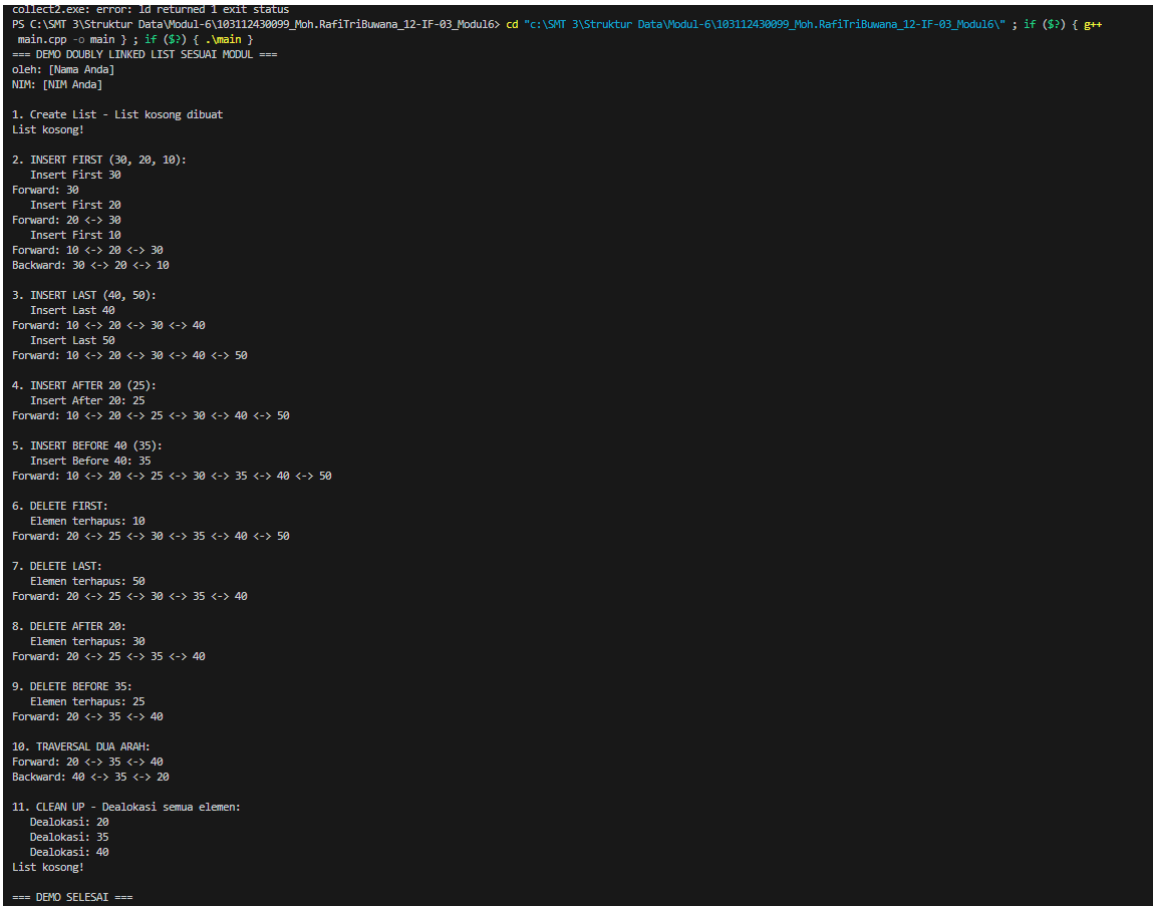
```
// DELETE LAST -
void deleteLast(List &L, address &P) {
    if (isEmpty(L)) {
        P = Nil;
        return;
    }

    P = L.last;                 // Step 1: P menunjuk elemen terakhir

    if (L.first == L.last) {
        // Hanya ada satu elemen
        L.first = Nil;
        L.last = Nil;
    } else {
        // Lebih dari satu elemen
        L.last = L.last->prev;      // Step 2: last menunjuk elemen sebelum last
        L.last->next = Nil;         // Step 3: last baru->next = Nil
    }

    // Isolasi elemen yang dihapus
    P->next = Nil;
    P->prev = Nil;
}

// DELETE AFTER -
void deleteAfter(address Prec, address &P) {
    if (Prec == Nil || Prec->next == Nil) {
        P = Nil;
        return;
    }

    P = Prec->next;             // Step 1: P menunjuk elemen setelah Prec

    if (P->next != Nil) {
        P->next->prev = Prec;   // Step 2: elemen setelah P->prev menunjuk Prec
    }
    Prec->next = P->next;       // Step 3: Prec->next menunjuk elemen setelah P

    // Isolasi elemen yang dihapus
    P->next = Nil;
    P->prev = Nil;
}
```

```cpp
// DELETE BEFORE - kebalikan delete after
void deleteBefore(address Prec, address &P) {
    if (Prec == Nil || Prec->prev == Nil) {
        P = Nil;
        return;
    }

    P = Prec->prev;             // Step 1: P menunjuk elemen sebelum Prec

    if (P->prev != Nil) {
        P->prev->next = Prec;   // Step 2: elemen sebelum P->next menunjuk Prec
    }
    Prec->prev = P->prev;       // Step 3: Prec->prev menunjuk elemen sebelum P

    // Isolasi elemen yang dihapus
    P->next = Nil;
    P->prev = Nil;
}

// PRINT FORWARD (dari first ke last)
void printForward(List L) {
    if (isEmpty(L)) {
        cout << "List kosong!" << endl;
        return;
    }

    address P = L.first;
    cout << "Forward: ";
    while (P != Nil) {
        cout << P->info;
        if (P->next != Nil) cout << " <-> ";
        P = P->next;
    }
    cout << endl;
}

// PRINT BACKWARD (dari last ke first)
void printBackward(List L) {
    if (isEmpty(L)) {
        cout << "List kosong!" << endl;
        return;
    }

    address P = L.last;
    cout << "Backward: ";
    while (P != Nil) {
        cout << P->info;
        if (P->prev != Nil) cout << " <-> ";
        P = P->prev;
    }
    cout << endl;
}
```

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

void demoModul() {
    cout << "=== DEMO DOUBLY LINKED LIST SESUAI MODUL ===" << endl;
    cout << "oleh: [Nama Anda]" << endl;
    cout << "NIM: [NIM Anda]" << endl << endl;

    List L;
    address P, Prec;

    // Inisialisasi list
    createList(L);
    cout << "1. Create List - List kosong dibuat" << endl;
    printForward(L);
    cout << endl;

    // Insert First sesuai modul
    cout << "2. INSERT FIRST (30, 20, 10):" << endl;
    insertFirst(L, alokasi(30));
    cout << "   Insert First 30" << endl;
    printForward(L);

    insertFirst(L, alokasi(20));
    cout << "   Insert First 20" << endl;
    printForward(L);

    insertFirst(L, alokasi(10));
    cout << "   Insert First 10" << endl;
    printForward(L);
    printBackward(L);
    cout << endl;

    // Insert Last sesuai modul
    cout << "3. INSERT LAST (40, 50):" << endl;
    insertLast(L, alokasi(40));
    cout << "   Insert Last 40" << endl;
    printForward(L);

    insertLast(L, alokasi(50));
    cout << "   Insert Last 50" << endl;
    printForward(L);
    cout << endl;

    // Insert After sesuai modul
    cout << "4. INSERT AFTER 20 (25):" << endl;
    Prec = L.first->next; // Menunjuk ke elemen 20
    insertAfter(Prec, alokasi(25));
    cout << "   Insert After 20: 25" << endl;
    printForward(L);
    cout << endl;
```

```
// Insert Before
cout << "5. INSERT BEFORE 40 (35):" << endl;
Prec = L.last->prev; // Menunjuk ke elemen 40
insertBefore(Prec, alokasi(35));
cout << "   Insert Before 40: 35" << endl;
printForward(L);
cout << endl;

// Delete First sesuai modul
cout << "6. DELETE FIRST:" << endl;
deleteFirst(L, P);
cout << "   Elemen terhapus: " << P->info << endl;
dealokasi(P);
printForward(L);
cout << endl;

// Delete Last sesuai modul
cout << "7. DELETE LAST:" << endl;
deleteLast(L, P);
cout << "   Elemen terhapus: " << P->info << endl;
dealokasi(P);
printForward(L);
cout << endl;

// Delete After sesuai modul
cout << "8. DELETE AFTER 20:" << endl;
Prec = L.first->next; // Menunjuk ke elemen 20
deleteAfter(Prec, P);
cout << "   Elemen terhapus: " << P->info << endl;
dealokasi(P);
printForward(L);
cout << endl;

// Delete Before
cout << "9. DELETE BEFORE 35:" << endl;
Prec = L.last->prev; // Menunjuk ke elemen 35
deleteBefore(Prec, P);
cout << "   Elemen terhapus: " << P->info << endl;
dealokasi(P);
printForward(L);
cout << endl;

// Traversal dua arah
cout << "10. TRAVERSAL DUA ARAH:" << endl;
printForward(L);
printBackward(L);
cout << endl;

    // Bersihkan memory
    cout << "11. CLEAN UP - Dealokasi semua elemen:" << endl;
    while (!isEmpty(L)) {
        deleteFirst(L, P);
        cout << "   Dealokasi: " << P->info << endl;
        dealokasi(P);
    }
    printForward(L);

    cout << endl << "=== DEMO SELESAI ===" << endl;
}

int main() {
    demoModul();
    return 0;
}
```

Screenshots Output



Deskripsi:

Program ini merupakan demonstrasi penerapan Doubly Linked List dalam praktikum struktur data. Melalui fungsi demoModul(), program menampilkan urutan operasi utama seperti pembuatan list, penyisipan elemen di awal, akhir, dan posisi tertentu, serta penghapusan elemen dari berbagai posisi menggunakan fungsi seperti insertFirst(), insertLast(), insertAfter(), deleteFirst(), dan deleteLast(). Setiap perubahan pada list ditampilkan menggunakan printForward() dan printBackward() agar pengguna dapat melihat isi list dari dua arah. Program ini membantu mahasiswa memahami cara kerja pointer dalam manipulasi node secara efisien dan menekankan pentingnya manajemen memori melalui proses *dealokasi* untuk mencegah kebocoran memori..

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

Unguided 1

```
#ifndef DOUBLYLIST_H
#define DOUBLYLIST_H

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

struct Kendaraan {
    string nopol;
    string warna;
    int tahunBuat;
};

typedef Kendaraan Infotype;
struct ElmList {
    Infotype info;
    ElmList* next;
    ElmList* prev;
};

typedef ElmList* Address;
struct List {
    Address first;
    Address last;
};
void createList(List &L);
Address alokasi(Infotype x);
void dealokasi(Address &P);
void printInfo(List L);
void insertLast(List &L, Address P);
Address findElm(List L, Infotype x);
void deleteFirst(List &L, Address &P);
void deleteLast(List &L, Address &P);
void deleteAfter(Address Prec, Address &P);

#endif
```

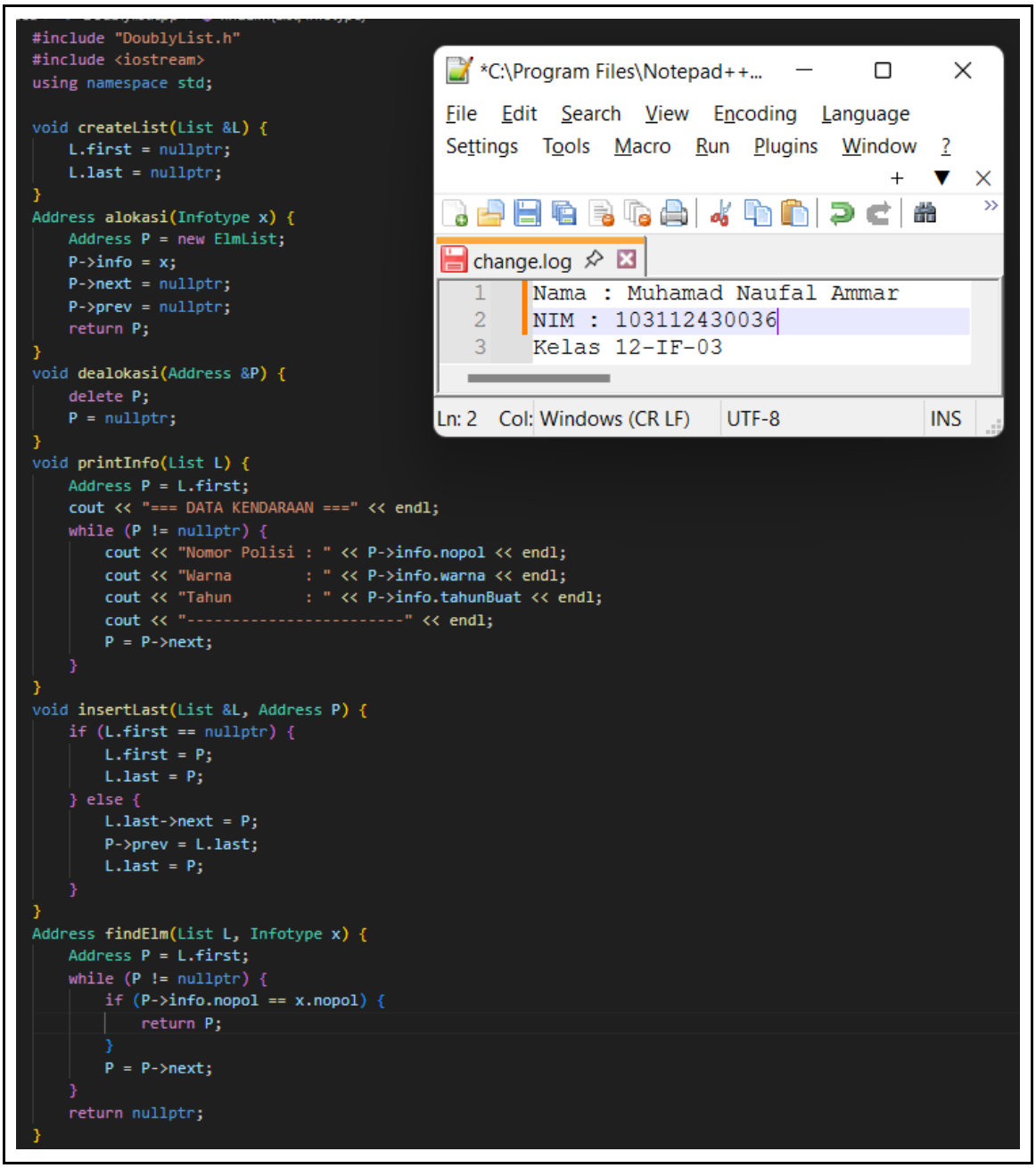
*C:\Program Files\Notepad++\...

File Edit Search View Encoding Language
Settings Tools Macro Run Plugins Window ?

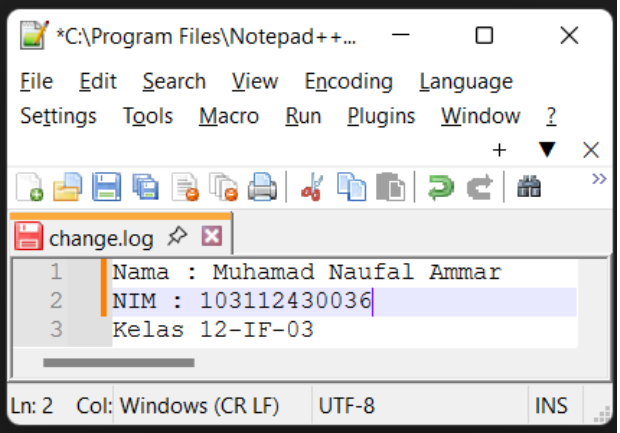
change.log

```
1  Nama : Muhamad Naufal Ammar
2  NIM : 103112430036
3  Kelas 12-IF-03
```

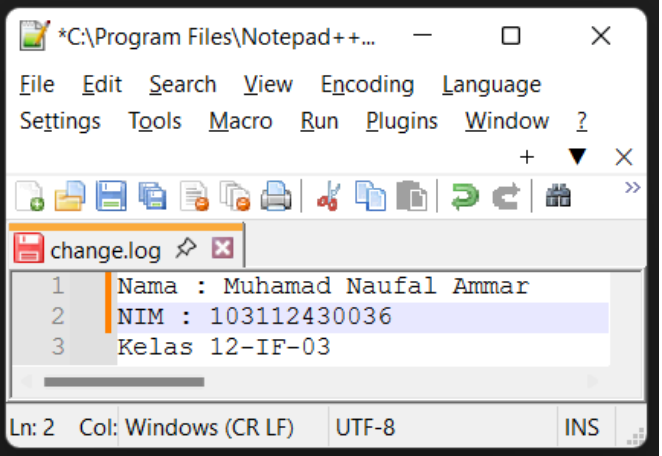
Ln: 2 Col: Windows (CR LF) UTF-8 INS


#include "DoublyList.h"
#include <iostream>
using namespace std;
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(List L) {
Address P = L.first;
cout << "=== DATA KENDARAAN ===" << endl;
while (P != nullptr) {
cout << "Nomor Polisi : " << P->info.nopol << endl;
cout << "Warna : " << P->info.warna << endl;
cout << "Tahun : " << P->info.tahunBuat << 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;
}
}
Address findElm(List L, Infotype x) {
Address P = L.first;
while (P != nullptr) {
if (P->info.nopol == x.nopol) {
return P;
}
P = P->next;
}
return nullptr;
}
*C:\Program Files\Notepad++...
File Edit Search View Encoding Language
Settings Tools Macro Run Plugins Window ?
change.log
1 Nama : Muhamad Naufal Ammar
2 NIM : 103112430036
3 Kelas 12-IF-03
Ln: 2 Col: Windows (CR LF) UTF-8 INS

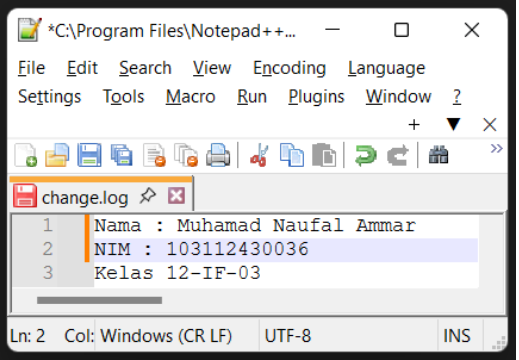
```
void deleteFirst(List &L, Address &P) {
    if (L.first != nullptr) {
        P = L.first;
        if (L.first == L.last) {
            L.first = nullptr;
            L.last = nullptr;
        } else {
            L.first = P->next;
            L.first->prev = nullptr;
            P->next = nullptr;
        }
    }
}
void deleteLast(List &L, Address &P) {
    if (L.last != nullptr) {
        P = L.last;
        if (L.first == L.last) {
            L.first = nullptr;
            L.last = nullptr;
        } else {
            L.last = P->prev;
            L.last->next = nullptr;
            P->prev = nullptr;
        }
    }
}
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;
    }
}
```



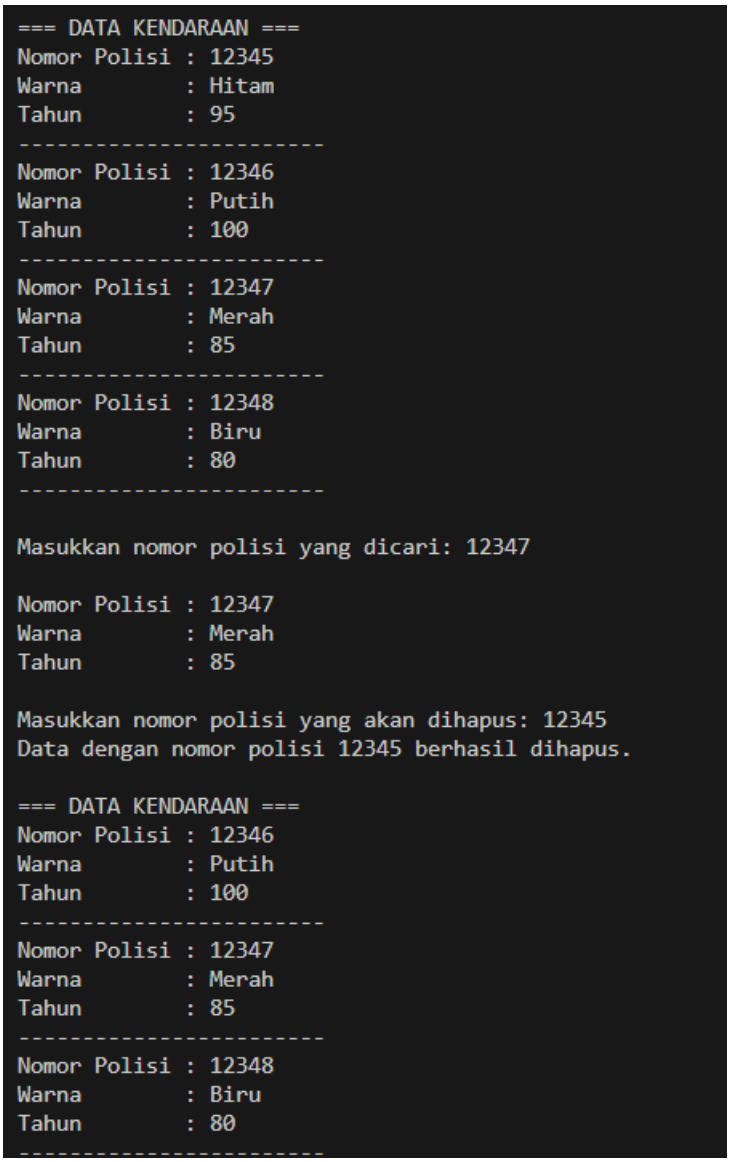
```cpp
#include "DoublyList.h"
#include "DoublyList.cpp"
#include <iostream>
using namespace std;

bool isExist(List L, string nopol) {
    Address P = L.first;
    while (P != nullptr) {
        if (P->info.nopol == nopol) {
            return true;
        }
        P = P->next;
    }
    return false;
}
int main() {
    List L;
    createList(L);

    Infotype x;
    Address P;
    int n = 4;
    string nopolInput, warnaInput;
    int tahunInput;

    for (int i = 0; i < n; i++) {
        cout << "Masukkan nomor polisi: ";
        cin >> nopolInput;
        if (isExist(L, nopolInput)) {
            cout << "Nomor polisi sudah terdaftar!" << endl;
            continue;
        }

        cout << "Masukkan warna kendaraan: ";
        cin >> warnaInput;
        cout << "Masukkan tahun kendaraan: ";
        cin >> tahunInput;

        x.nopol = nopolInput;
        x.warna = warnaInput;
        x.tahunBuat = tahunInput;

        P = alokasi(x);
        insertLast(L, P);
    }

    cout << endl;
    printInfo(L);
    cout << endl;
    cout << "Masukkan nomor polisi yang dicari: ";
    cin >> x.nopol;
```



```
cout << endl;
printInfo(L);
cout << endl;
cout << "Masukkan nomor polisi yang dicari: ";
cin >> x.nopol;

P = findElm(L, x);
if (P != nullptr) {
    cout << endl;
    cout << "Nomor Polisi : " << P->info.nopol << endl;
    cout << "Warna        : " << P->info.warna << endl;
    cout << "Tahun        : " << P->info.tahunBuat << endl;
} else {
    cout << "Data dengan nomor polisi tersebut tidak ditemukan." << endl;
}
cout << endl;
cout << "Masukkan nomor polisi yang akan dihapus: ";
cin >> x.nopol;

Address prec = nullptr;
P = L.first;
while (P != nullptr && P->info.nopol != x.nopol) {
    prec = P;
    P = P->next;
}
if (P == nullptr) {
    cout << "Data dengan nomor polisi " << x.nopol << " tidak ditemukan." << endl;
} else {
    if (P == L.first) {
        deleteFirst(L, P);
    } else if (P == L.last) {
        deleteLast(L, P);
    } else {
        deleteAfter(prec, P);
    }
    cout << "Data dengan nomor polisi " << x.nopol << " berhasil dihapus." << endl;
    dealokasi(P);
}

cout << endl;
printInfo(L);

return 0;
}
```



Screenshots Output



Deskripsi:

program yang digunakan untuk mengimplementasikan struktur data Doubly Linked List dalam bahasa C++. File DoublyList.h berfungsi sebagai header file yang berisi deklarasi tipe data, struktur node, serta prototipe fungsi yang digunakan dalam pengelolaan list, seperti pembuatan list, alokasi, penyisipan, pencarian, dan penghapusan data. File DoublyList.cpp berisi implementasi dari fungsi-fungsi tersebut secara detail, termasuk cara kerja pointer next dan prev dalam menghubungkan antar-node, serta prosedur penghapusan yang memperhatikan kondisi list kosong atau hanya memiliki satu elemen. Sementara itu, file main.cpp berperan sebagai bagian utama program yang mengatur alur eksekusi, seperti memasukkan data kendaraan, menampilkan isi list, mencari data berdasarkan nomor polisi, dan menghapus data yang diinginkan. Secara keseluruhan, ketiga file ini menunjukkan bagaimana mahasiswa dapat memahami dan menerapkan konsep struktur data dinamis dua arah, manipulasi pointer, serta pengelolaan memori dalam pemrograman C++ dengan cara yang terstruktur dan sistematis.

D. Kesimpulan

Kesimpulannya, program merupakan penerapan konsep Doubly Linked List dalam pengelolaan data secara dinamis menggunakan bahasa C++. Melalui pembagian kode menjadi file header, implementasi, dan main, program ini menunjukkan bagaimana struktur data dua arah bekerja dalam proses penyisipan, pencarian, dan penghapusan elemen dengan memanfaatkan pointer next dan prev. Selain itu, program ini juga menekankan pentingnya manajemen memori dinamis serta pemahaman terhadap hubungan antar-node dalam list. Secara keseluruhan, implementasi ini membantu mahasiswa memahami bagaimana konsep teoretis struktur data dapat diubah menjadi program yang efisien, fleksibel, dan mudah dikembangkan dalam praktik pemrograman nyata.

E. Referensi

Khan, M. S., Ware, A., Habib, U., Khalid, M. J., & Bahoo, N. (2022). Skeleton based human action recognition using doubly linked list. *International Journal of Computer Trends and Technology (IJCTT)*, 70(2), 18-21.

Acharjya, P. P., Koley, S., & Barman, S. (2022). Analysis of the complexity of various linked lists. *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, 11(2), 886-895.