LAPORAN PRAKTIKUM

PERTEMUAN 6

DOUBLE LINKED LIST (BAGIAN PERTAMA)

Nama :

Alvin Bagus Firmansyah-2311104070

Dosen :

Nama Dosen

Wahyu Andi Saputra, S.PD, M.Eng

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. TUJUAN

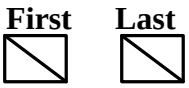
1. Memahami konsep modul *linked list*.
2. Mengaplikasikan konsep *double linked list* dengan menggunakan *pointer* dan dengan bahasa C

II. DASAR TEORI

6.1 Double Linked List

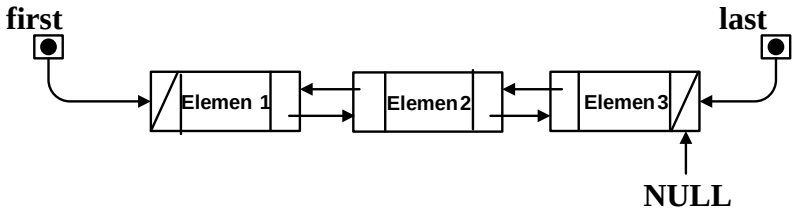
Double Linked List (DLL) adalah sebuah struktur data linear di mana setiap elemen (node) tidak hanya merujuk ke elemen berikutnya (next), tetapi juga merujuk ke elemen sebelumnya (prev). Hal ini berbeda dengan Single Linked List yang hanya memiliki satu pointer ke elemen berikutnya.

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



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*:

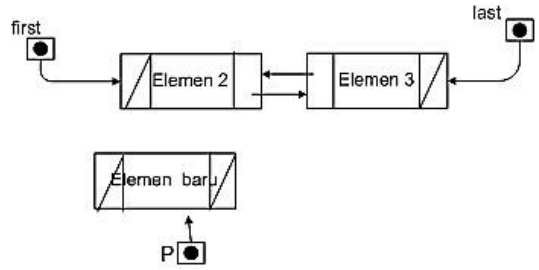
```
1   #ifndef doublelist_H
2   #define doublelist_H
3   #include "boolean.h"
4   #define Nil NULL
5   #define info(P) (P)->info
6   #define next(P) (P)->next
7   #define prev(P) (P)->prev
8   #define first(L) ((L).first)
9   #define last(L) ((L).last)
10
11  /*deklarasi record dan struktur data double linked list*/ typedef int
12  infotype;
13  typedef struct elmlist *address;
14  struct elmlist {  infotype info;
15  address next;  address prev;
16  };
17
18
19
```

```
20  /* definisi list: */
21  /* list kosong jika First(L)=Nil */
22  struct list{  address first;
23   address last;
24  };
25  #endif
26
```

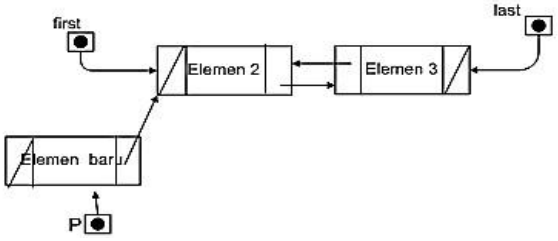
6.1.1 Insert

A. Insert First

Langkah-langkah dalam proses *insert first*:

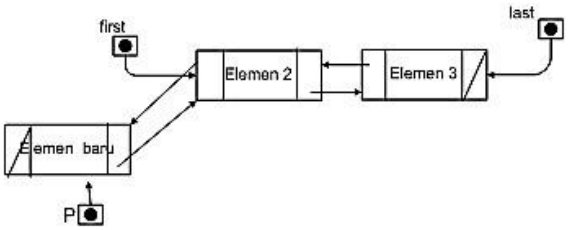


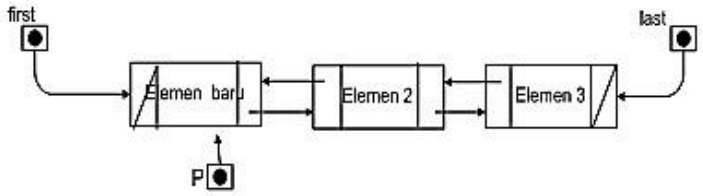

```
P = alokasi(X);
next(P) = Nil;
prev(P) = Nil;
```




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

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



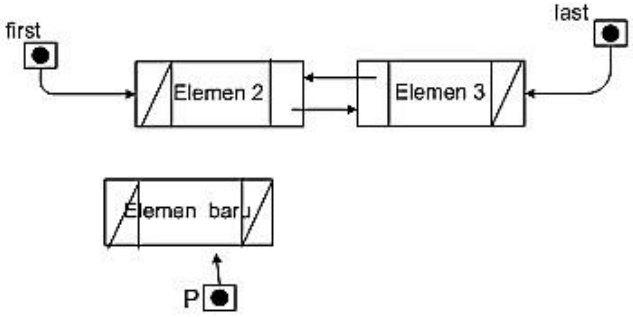


first(L) = P;

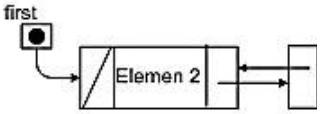
```
void insertFirst(list &L, address &P){
 next(P) = first(L);  prev(first(L))
= P;  first(L) = P;
}
```

B. Insert Last

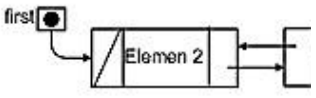
Langkah-langkah dalam proses *insert last*:



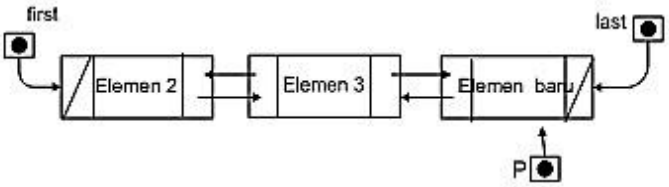
P = alokasi(X);
next(P) = Nil;
last(P) = Nil;



prev(P) = last(L);



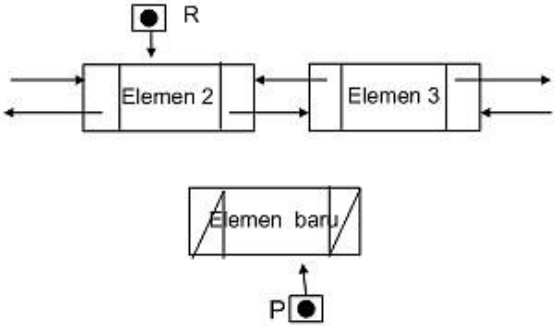
next(last(L)) = P;



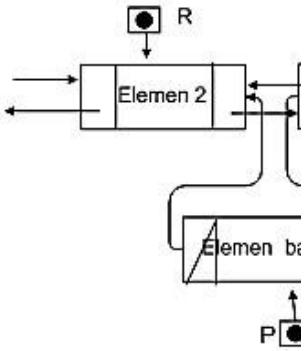
last(L) = P;

C. Insert After

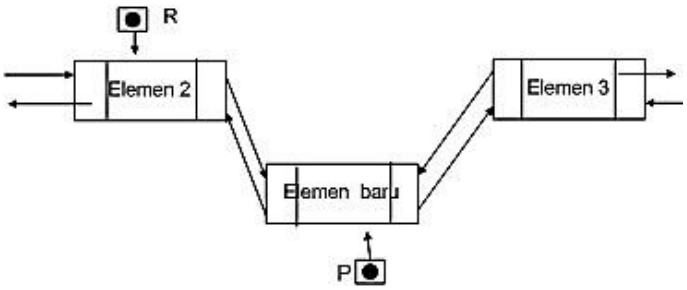
Langkah-langkah dalam proses *insert after*:



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



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



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

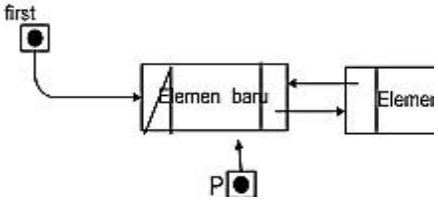
D. Insert Before

Perbedaan utama antara operasi insert after dan insert before pada Double Linked List terletak pada posisi penempatan node baru relatif terhadap node yang sudah ada.

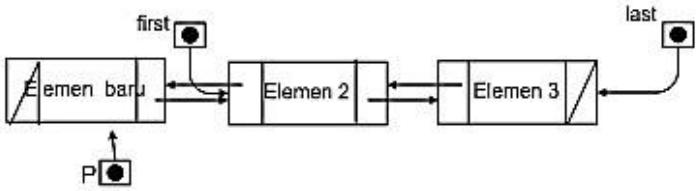
6.1.2 Delete

A. Delete First

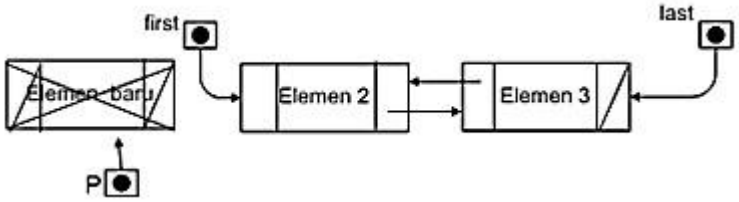
Langkah-langkah dalam proses *delete first*:



P = first(L);




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

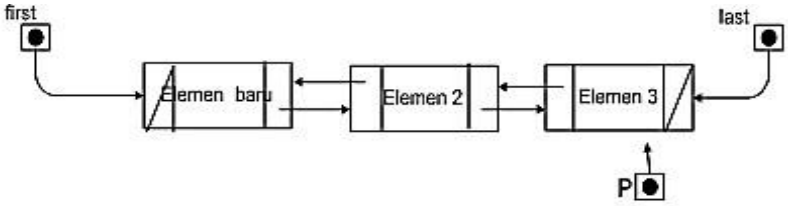



```
prev(first(L)) = Nil;
next(P) = Nil; return
P;
        atau
dealokasi(P);
```

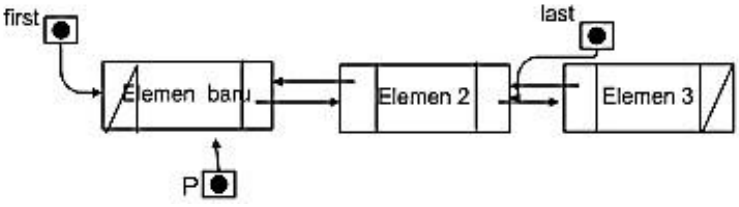
```
/* contoh sintak delet first */ void
deleteFirst(list &L, address &P){  P =
first(L);
 first(L) = next(first(L));  prev (P) =
null;  prev(first(L)) = null;  next(P)
= null;
}
```

B. Delete Last

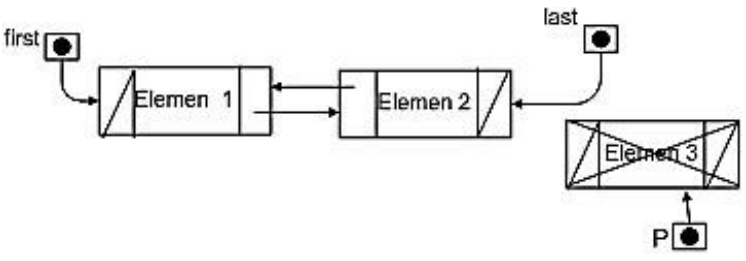
Langkah-langkah dalam proses *delete last*:




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



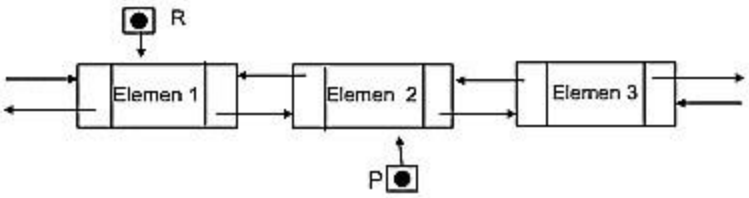

```
last(L) = prev(last(L));
```

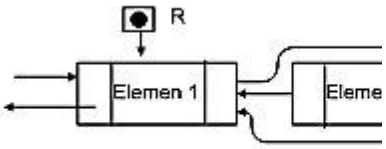



```
prev(P) = Nil;
next(last(L)) = Nil;
return P;
        atau
dealokasi(P);
```

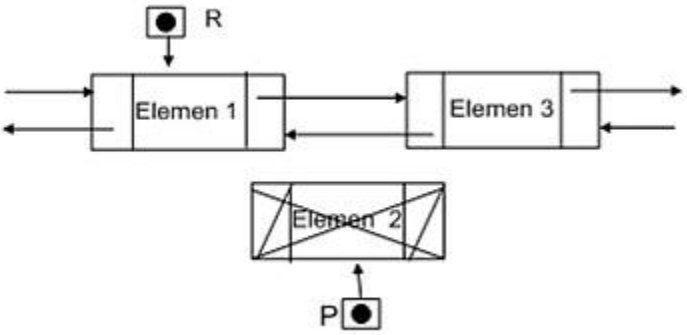
C. Delete After

Langkah-langkah dalam proses *delete after*:





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



prev(P) = Nil;
next(P) = Nil; return P;

atau

dealokasi(P);

D. 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.

E. 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*.

```
1   /*file : doublelist .h*/
2   /* contoh ADT list berkait dengan representasi fisik pointer*/
3   /* representasi address dengan pointer*/
4   /* info tipe adalah integer */
5   #ifndef doublelist_H
6   #define doublelist_H
7
8   #include <stdio.h>
9   #define Nil NULL
10  #define info(P) (P)->info
11  #define next(P) (P)->next
12  #define prev(P) (P)->prev
13  #define first(L) ((L).first)
14  #define last(L) ((L).last)
15
16  typedef int infotype;
17  typedef struct elmlist *address;
18  /* pendefinisian tipe data bentukan elemen list
19  dengan dua successor, yaitu next dan prev */
20  struct elmlist{  infotype info;  address prev;
21  address next;
22  };
23
24  /* definisi double linked list : list kosong jika first(L)=Nil    setiap
25  elemen address P dapat diacu info(P) atau next(P)    elemen terakhir
26  adalah last
27     nama tipe list yang dipakai adalah 'list', sama dengan pada singe list*/ struct list
28  {
29   address first,last;
30  };
31
32  /** Deklarasi fungsi primitif lain **/
33  /** Sama dengan Single Linked list **/
34
35
```

III. GUIDE

1.Analisis Kode Doubly Linked List

Kode C++ yang kita analisis ini dirancang untuk membangun sebuah struktur data yang disebut Double Linked List. Tujuan utamanya adalah untuk:

- Mewakili data secara berurutan: Data disimpan dalam bentuk node-node yang saling terhubung, membentuk sebuah rantai.
- Memungkinkan akses dua arah: Kita bisa menelusuri data baik dari awal ke akhir maupun sebaliknya.
- Menyediakan fleksibilitas dalam operasi: Kita bisa menambahkan, menghapus, mencari, dan mengubah data dengan mudah.

Kode Progam:

main.cpp | main.cpp | include\doublelist.h | src\doublelist.cpp | *Untitled2 | main.cpp | main.cpp | main.cpp

```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) {
```

main.cpp | main.cpp | include\doublelist.h | src\doublelist.cpp | *Untitled2 | main.cpp | main.cpp | main.cpp

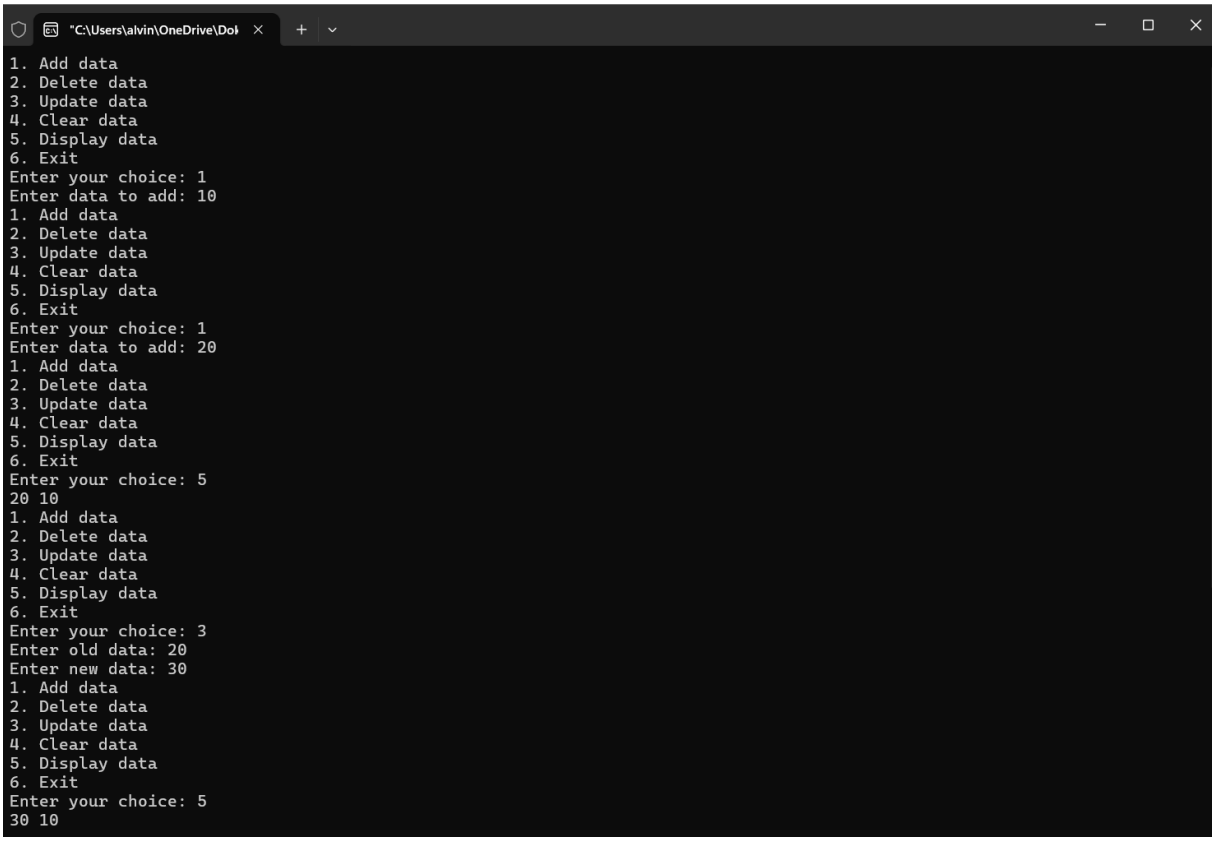
```cpp
            return; // Jika list kosong
        }
        Node* temp = head;
        head = head->next;
        if (head != nullptr) {
            head->prev = nullptr;
        } else {
            tail = nullptr; // Jika hanya satu elemen di list
        }
        delete temp; // Hapus elemen
    }

    // 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) {
            No Node* DoublyLinkedList::deleteAll::current
            current = current->next;
            delete temp;
        }
        head = nullptr;
        tail = nullptr;
    }

    // Fungsi untuk menampilkan semua elemen di list
    void display() {
```

main.cpp | main.cpp | include\doublelist.h | src\doublelist.cpp | *Untitled2 | main.cpp | main.cpp | main.cpp

```cpp
        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;
}
```

Hasil Outputnya:



IV. UNGUIDED

1.Program Manajemen Data Kendaraan Sederhana

Program ini merupakan contoh sederhana dari penerapan struktur data vektor dalam bahasa C++ untuk mengelola data kendaraan. Konsep yang digunakan dalam program ini dapat diaplikasikan dalam berbagai jenis program yang membutuhkan pengelolaan data.

Kode Progam:

main.cpp

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

using namespace std;

struct Kendaraan {
    string nomorPolisi;
    string warna;
    int tahun;
};

int main() {
    vector<Kendaraan> dataKendaraan;

    while (true) {
        Kendaraan kendaraanBaru;
        cout << "Masukkan nomor polisi: ";
        cin >> kendaraanBaru.nomorPolisi;

        // Cek duplikat
        bool adaDuplikat = false;
        for (const Kendaraan& kendaraan : dataKendaraan) {
            if (kendaraan.nomorPolisi == kendaraanBaru.nomorPolisi) {
                cout << "Nomor polisi sudah terdaftar." << endl;
                adaDuplikat = true;
                break;
            }
        }

        if (!adaDuplikat) {
            cout << "Masukkan warna kendaraan: ";
            cin >> kendaraanBaru.warna;
            cout << "Masukkan tahun kendaraan: ";
            cin >> kendaraanBaru.tahun;

            dataKendaraan.push_back(kendaraanBaru);
        }

        cout << endl;

        // Tampilkan opsi untuk melanjutkan atau tidak
        char pilihan;
        cout << "Ingin menambahkan data lagi? (y/n): ";
        cin >> pilihan;
        if (pilihan != 'y' && pilihan != 'Y') {
            break;
        }
    }

    // Tampilkan semua data kendaraan
    cout << "DATA LIST 1" << endl;
    for (const Kendaraan& kendaraan : dataKendaraan) {
        cout << "Nomor Polisi: " << kendaraan.nomorPolisi << endl;
        cout << "Warna       : " << kendaraan.warna << endl;
        cout << "Tahun       : " << kendaraan.tahun << endl;
    }

    return 0;
}
```

Hasil Outputnya:

```
"C:\Users\alvin\OneDrive\Dok
Masukkan nomor polisi: D001
Masukkan warna kendaraan: hitam
Masukkan tahun kendaraan: 90

Ingin menambahkan data lagi? (y/n): y
Masukkan nomor polisi: D003
Masukkan warna kendaraan: putih
Masukkan tahun kendaraan: 70

Ingin menambahkan data lagi? (y/n): y
Masukkan nomor polisi: D001
Nomor polisi sudah terdaftar.

Ingin menambahkan data lagi? (y/n): y
Masukkan nomor polisi: D004
Masukkan warna kendaraan: kuning
Masukkan tahun kendaraan: 90

Ingin menambahkan data lagi? (y/n): n
DATA LIST 1
Nomor Polisi: D001
Warna       : hitam
Tahun       : 90
Nomor Polisi: D003
Warna       : putih
Tahun       : 70
Nomor Polisi: D004
Warna       : kuning
Tahun       : 90

Process returned 0 (0x0)   execution time : 76.152 s
Press any key to continue.
```

2. Program Pencarian Data Kendaraan Berdasarkan Nomor Polisi

Program ini merupakan contoh sederhana dari penerapan pencarian data dalam vektor. Konsep yang digunakan dalam program ini dapat diaplikasikan dalam berbagai jenis program yang membutuhkan pencarian data.

Kode Progam:

*main.cpp

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

using namespace std;

struct Kendaraan {
    string nomorPolisi;
    string warna;
    int tahun;
};
int main() {
    vector<Kendaraan> dataKendaraan;

    dataKendaraan.push_back({"D001", "hitam", 1998});
    dataKendaraan.push_back({"B1234", "merah", 2005});
    dataKendaraan.push_back({"D5678", "biru", 2010});

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

    bool ditemukan = false;
    for (const Kendaraan& kendaraan : dataKendaraan) {
        if (kendaraan.nomorPolisi == nomorPolisiCari) {
            cout << "Nomor Polisi: " << kendaraan.nomorPolisi << endl;
            cout << "Warna       : " << kendaraan.warna << endl;
            cout << "Tahun       : " << kendaraan.tahun << endl;
            ditemukan = true;
            break;
        }
    }

    if (!ditemukan) {
        cout << "Kendaraan tidak ditemukan." << endl;
    }

    return 0;
}
```

Hasil Outputnya:

```
"C:\Users\alvin\OneDrive\Dok
Masukkan Nomor Polisi yang dicari: D001
Nomor Polisi: D001
Warna       : hitam
Tahun       : 1998

Process returned 0 (0x0)   execution time : 8.174 s
Press any key to continue.
```

3. Program Penghapusan Data Kendaraan Berdasarkan Nomor Polisi

Program ini memberikan contoh sederhana tentang cara menghapus data dari sebuah vektor dalam bahasa C++. Konsep yang digunakan dalam program ini dapat diaplikasikan dalam berbagai jenis program yang membutuhkan pengelolaan data.

Kode Progam:

main.cpp

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

using namespace std;

struct Kendaraan {
    string nomorPolisi;
    string warna;
    int tahun;
};

int main() {
    vector<Kendaraan> dataKendaraan;

    // Contoh data kendaraan
    dataKendaraan.push_back({"D001", "hitam", 1998});
    dataKendaraan.push_back({"B1234", "merah", 2005});
    dataKendaraan.push_back({"D5678", "biru", 2010});
    dataKendaraan.push_back({"D003", "kuning", 1990});
    dataKendaraan.push_back({"D004", "hitam", 1990});

    string nomorPolisiHapus;
    cout << "Masukkan Nomor Polisi yang akan dihapus: ";
    cin >> nomorPolisiHapus;

    bool ditemukan = false;
    for (auto it = dataKendaraan.begin(); it != dataKendaraan.end(); ++it) {
        if (it->nomorPolisi == nomorPolisiHapus) {
            dataKendaraan.erase(it);
            ditemukan = true;
            break;
        }
    }

    if (ditemukan) {
        cout << "Data dengan nomor polisi " << nomorPolisiHapus << " berhasil dihapus." << endl;
    } else {
        cout << "Kendaraan tidak ditemukan." << endl;
    }

    // Menampilkan data yang tersisa
    cout << "DATA LIST 1" << endl;
    for (const Kendaraan& kendaraan : dataKendaraan) {
        cout << "Nomor Polisi: " << kendaraan.nomorPolisi << endl;
        cout << "Warna       : " << kendaraan.warna << endl;
        cout << "Tahun       : " << kendaraan.tahun << endl;
    }

    return 0;
}
```

Hasil Outputnya:

```
"C:\Users\alvin\OneDrive\Dok
Masukkan Nomor Polisi yang akan dihapus: D003
Data dengan nomor polisi D003 berhasil dihapus.
DATA LIST 1
Nomor Polisi: D001
Warna       : hitam
Tahun       : 1998
Nomor Polisi: B1234
Warna       : merah
Tahun       : 2005
Nomor Polisi: D5678
Warna       : biru
Tahun       : 2010
Nomor Polisi: D004
Warna       : hitam
Tahun       : 1990

Process returned 0 (0x0)   execution time : 8.257 s
Press any key to continue.
```

V. KESIMPULAN

Program ini berhasil mengelola data kendaraan secara sederhana. Program bisa menambah, mencari, dan menghapus data kendaraan. Konsep utama yang digunakan adalah struktur data, vektor, dan iterasi. Program ini bisa dikembangkan lebih lanjut untuk fitur yang lebih kompleks.