LAPORAN PRAKTIKUM
MODUL 6
“DOUBLE LINKED LIST (BAGIAN PERTAMA)”

Disusun Oleh:

Rengganis Tantri Pramudita - 2311104065

S1SE0702

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 dan dengan bahasa C
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.

   Operasi Dasar yang ada pada Double Linked List

   - Insert First

     menambahkan node baru di awal list dengan mengatur pointer next pada node baru ke node pertama yang lama, lalu menjadikan node baru sebagai node pertama (head) pada list.
   - Insert Last

     menambahkan node di akhir list dengan menghubungkan pointer next pada node terakhir ke node baru, dan pointer prev pada node baru mengarah ke node terakhir, sehingga node baru menjadi node terakhir (tail).
   - Insert After

     node baru disisipkan setelah node tertentu. Proses ini melibatkan pengaturan pointer next dari node baru ke node setelah node target, dan pointer prev dari node baru ke node target itu sendiri, lalu menghubungkan node sebelum dan sesudah node baru agar terintegrasi di dalam rantai list.
   - Insert Before

menambahkan node baru sebelum node tertentu. Ini dilakukan dengan mengatur pointer next node baru ke node target dan prev node baru ke node sebelum node target, lalu memastikan hubungan antara node sebelumnya, node baru, dan node target

Operasi dasar penghapusan node dalam Double Linked List

- Delete First

  node pertama (head) dihapus dengan cara mengubah head ke node berikutnya dan memutuskan koneksi prev node baru tersebut agar tidak lagi terhubung ke node pertama yang lama. Jika hanya ada satu node, list menjadi kosong.
- Delete last

  menghapus node terakhir (tail) dengan memindahkan tail ke node sebelumnya dan memutuskan koneksi next pada node baru tersebut ke node terakhir yang lama.
- Delete after

  menghapus node setelah node tertentu dengan cara mengambil node setelah node target, menghubungkan kembali node target ke node setelah node yang dihapus, dan memastikan koneksi prev tetap benar untuk mencegah node yang dihapus tetap terhubung.
- Delete before, node sebelum node tertentu dihapus dengan cara yang mirip, yaitu menghubungkan node sebelum node yang dihapus ke node target dan memperbarui pointer next dan prev agar node yang dihapus terputus sempurna dari rantai list.

Dalam Double Linked List (DLL), operasi *Update*, *View*, dan *Searching* merupakan operasi dasar yang memungkinkan manipulasi dan pengelolaan data di dalam list.

- Operasi Update

  digunakan untuk mengganti data dalam node tertentu. Setelah node yang diinginkan ditemukan, nilai data pada node tersebut diperbarui tanpa mengubah pointer next dan prev, sehingga hubungan antar-node tetap konsisten.
- Operasi View

  berfungsi untuk menampilkan isi list, baik dari awal ke akhir atau sebaliknya, dengan memulai dari head dan mengikuti pointer next untuk tampilan maju, atau mulai dari tail dan mengikuti pointer prev untuk tampilan mundur.
- Operasi Searching

  dilakukan untuk mencari data tertentu dalam list. Pencarian dapat dimulai dari head atau tail, tergantung letak perkiraan data yang dicari.

3. 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;
```

```cpp
        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;
    }
```

ain.cpp

```cpp
};
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;
            }
```

```
main.cpp
99                  list.insert(data);
100                 break;
101             }
102             case 2: {
103                 list.deleteNode();
104                 break;
105             }
106             case 3: {
107                 int oldData, newData;
108                 cout << "Enter old data: ";
109                 cin >> oldData;
110                 cout << "Enter new data: ";
111                 cin >> newData;
112                 bool updated = list.update(oldData, newData);
113                 if (!updated) {
114                     cout << "Data not found" << endl;
115                 }
116                 break;
117             }
118             case 4: {
119                 list.deleteAll();
120                 break;
121             }
122             case 5: {
123                 list.display();
124                 break;
125             }
126             case 6: {
127                 return 0;
128             }
129             default: {
130                 cout << "Invalid choice" << endl;
131                 break;
132             }
133         }
134     }
135     return 0;
136 }
137
```

Keterangan

- **Class Node**: Mendefinisikan struktur node dalam Double Linked List yang memiliki data, pointer prev untuk node sebelumnya, dan pointer next untuk node berikutnya.
- **Class DoublyLinkedList**:
  - ✓ **Atribut head dan tail**: Pointer untuk node pertama (head) dan node terakhir (tail) dalam list.
  - ✓ **Constructor**: Menginisialisasi head dan tail dengan nullptr, menandakan list kosong saat awal dibuat.
- **Fungsi insert**: Menambahkan node baru di awal list. Jika list kosong, node baru menjadi tail. Jika tidak kosong, node baru ditambahkan sebagai head dan node lama diperbarui agar prev menunjuk ke node baru.
- **Fungsi deleteNode**: Menghapus node pertama. Jika list hanya memiliki satu node, tail diset ke nullptr. Jika ada beberapa node, head dipindahkan ke node berikutnya.
- **Fungsi update**: Mencari data tertentu (oldData) di dalam list. Jika ditemukan, data diganti dengan newData dan mengembalikan true. Jika tidak ditemukan, mengembalikan false.
- **Fungsi deleteAll**: Menghapus semua node dalam list dengan menghapus setiap node satu per satu dan menyetel head serta tail ke nullptr.
- **Fungsi display**: Menampilkan seluruh isi list dari head ke tail.
- **main Function**: Menyediakan antarmuka menu sederhana bagi pengguna untuk menambahkan data, menghapus data, memperbarui data, menghapus semua data, menampilkan data, dan keluar dari program.

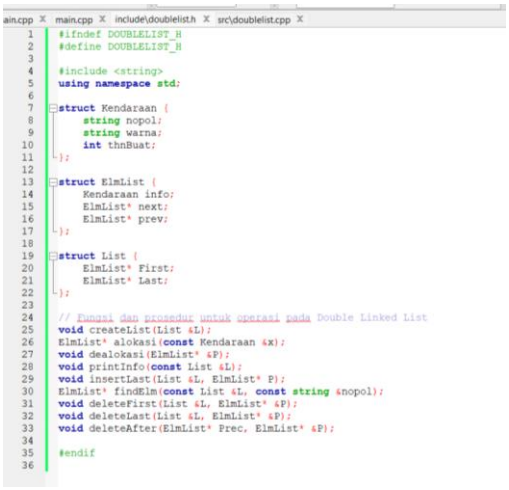
Outputnya

```
"C:\codeblocks\pertemuan ke
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: 7
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
7
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 6

Process returned 0 (0x0)   execution time : 13.683 s
Press any key to continue.
```

4. Unguided

1. - Doublelist.h



```
ain.cpp | main.cpp | include\doublelist.h | src\doublelist.cpp
#ifndef DOUBLELIST_H
#define DOUBLELIST_H

#include <string>
using namespace std;

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

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

struct List {
    ElmList* First;
    ElmList* Last;
};

// Fungsi dan prosedur untuk operasi pada Double Linked List
void createList(List &L);
ElmList* alokasi(const Kendaraan &x);
void dealokasi(ElmList* &P);
void printInfo(const List &L);
void insertLast(List &L, ElmList* P);
ElmList* findElm(const List &L, const string &nopol);
void deleteFirst(List &L, ElmList* &P);
void deleteLast(List &L, ElmList* &P);
void deleteAfter(ElmList* Prec, ElmList* &P);

#endif
```

Kode di atas adalah deklarasi header untuk Double Linked List (DLL) yang digunakan untuk menyimpan data kendaraan.

Keterangan:

- Struct Kendaraan

  Struktur Kendaraan mendefinisikan tipe data kendaraan yang berisi:

  - nopol (nomor polisi kendaraan) bertipe string.
  - warna (warna kendaraan) bertipe string.
  - thnBuat (tahun pembuatan kendaraan) bertipe int.

- Struct ElmList

  Struktur ElmList berfungsi sebagai *node* dalam *Double Linked List*, yang berisi:

  - info bertipe Kendaraan, untuk menyimpan data kendaraan di *node* tersebut.
  - next adalah pointer yang menunjuk ke *node* berikutnya dalam list.

- prev adalah pointer yang menunjuk ke *node* sebelumnya dalam list.

- Struc List

  Struktur List berfungsi sebagai penanda awal dan akhir dari *Double Linked List*, dengan:

  - First, yaitu pointer yang menunjuk ke *node* pertama dalam list.
  - Last, yaitu pointer yang menunjuk ke *node* terakhir dalam list.

- Fungsi dan Prosedur

  - Menginisialisasi list L dengan mengatur First dan Last menjadi nullptr, yang menunjukkan bahwa list kosong.
  - Menerima data Kendaraan sebagai parameter dan mengalokasikan memori untuk *node* baru (ElmList).
  - Mengembalikan pointer ke *node* baru yang sudah terisi dengan data Kendaraan.
  - Membebaskan memori yang dialokasikan untuk *node* P, sehingga mencegah kebocoran memori.
  - Menampilkan semua data kendaraan di dalam *Double Linked List* mulai dari First hingga Last.
  - Menambahkan *node* P ke posisi terakhir pada *Double Linked List* L.
  - Mencari *node* dengan nomor polisi (nopol) tertentu di dalam *Double Linked List* L.
  - Mengembalikan pointer ke *node* yang ditemukan atau nullptr jika tidak ada.
  - Menghapus *node* pertama dalam *Double Linked List* L dan mengembalikannya dalam parameter P.
  - Menghapus *node* terakhir dalam *Double Linked List* L dan mengembalikannya dalam parameter P.
  - Menghapus *node* setelah *node* Prec di dalam list, dan mengembalikan *node* yang dihapus melalui parameter P.

- Doublelist.cpp

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

void createList(List &L) {
    L.First = nullptr;
    L.Last = nullptr;
}

ElmList* alokasi(const Kendaraan &x) {
    ElmList* P = new ElmList;
    P->info = x;
    P->next = nullptr;
    P->prev = nullptr;
    return P;
}

void dealokasi(ElmList* &P) {
    delete P;
    P = nullptr;
}

void printInfo(const List &L) {
    ElmList* P = L.First;
    while (P != nullptr) {
        cout << "Nomor Polisi : " << P->info.nopol << endl;
        cout << "Warna        : " << P->info.warna << endl;
        cout << "Tahun        : " << P->info.thnBuat << endl;
        cout << endl;
        P = P->next;
    }
}

void insertLast(List &L, ElmList* P) {
    if (L.First == nullptr) {  // Jika list kosong
        L.First = P;
        L.Last = P;
    } else {
        P->prev = L.Last;
        L.Last->next = P;
        L.Last = P;
    }
}

ElmList* findElm(const List &L, const string &nopol) {
    ElmList* P = L.First;
    while (P != nullptr) {
        if (P->info.nopol == nopol) {
            return P;  // Mengembalikan pointer ke elemen yang ditemukan
        }
        P = P->next;
    }
    return nullptr;  // Mengembalikan nullptr jika elemen tidak ditemukan
}

void deleteFirst(List &L, ElmList* &P) {
    if (L.First != nullptr) {
        P = L.First;
        if (L.First == L.Last) {  // Jika hanya satu elemen
            L.First = nullptr;
            L.Last = nullptr;
        } else {
            L.First = L.First->next;
            L.First->prev = nullptr;
        }
        P->next = nullptr;
    }
}

void deleteLast(List &L, ElmList* &P) {
    if (L.Last != nullptr) {
        P = L.Last;
        if (L.First == L.Last) {  // Jika hanya satu elemen
            L.First = nullptr;
            L.Last = nullptr;
        } else {
            L.Last = L.Last->prev;
            L.Last->next = nullptr;
        }
        P->prev = nullptr;
    }
}

void deleteAfter(ElmList* Prec, ElmList* &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;
    }
}
```

Keterangan:

- Menginisialisasi list L dengan mengatur First dan Last menjadi nullptr, sehingga list kosong.
- Mengalokasikan memori untuk *node* baru dan mengisi info dengan data Kendaraan yang diberikan. Pointer next dan prev diatur ke nullptr. Mengembalikan pointer ke *node* yang baru dibuat.
- Membebaskan memori yang dialokasikan untuk *node* P dan mengatur P ke nullptr untuk mencegah kebocoran memori.
- Menampilkan data kendaraan dalam list mulai dari First hingga Last. Mencetak nopol, warna, dan thnBuat dari setiap *node* di dalam list.

- Menambahkan *node* P ke posisi terakhir dalam list L. Jika list kosong, First dan Last diatur menjadi P. Jika tidak, P ditambahkan setelah *node* terakhir.
- Mencari *node* dengan nomor polisi (nopol) tertentu. Jika ditemukan, mengembalikan pointer ke *node* tersebut. Jika tidak ditemukan, mengembalikan nullptr.
- Menghapus *node* pertama (First) dalam list L dan mengembalikan *node* yang dihapus melalui parameter P. Jika hanya satu elemen, First dan Last diatur ke nullptr.
- Menghapus *node* terakhir (Last) dalam list L dan mengembalikan *node* yang dihapus melalui parameter P. Jika hanya satu elemen, First dan Last diatur ke nullptr.
- Menghapus *node* setelah *node* Prec dalam list dan mengembalikan *node* yang dihapus melalui parameter P. Jika *node* yang dihapus bukan yang terakhir, pointer prev dari *node* setelah P diperbarui ke Prec.

- Main.cpp

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

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

    // Menambahkan beberapa data kendaraan
    insertLast(L, alokasi("D001", "hitam", 90));
    insertLast(L, alokasi("D003", "putih", 70));
    insertLast(L, alokasi("D004", "kuning", 90));

    // Cetak semua data
    cout << "DATA LIST 1" << endl;
    printInfo(L);

    // Mencari elemen berdasarkan nomor polisi
    string cariNopol;
    cout << "\nMasukkan Nomor Polisi yang dicari: ";
    cin >> cariNopol;

    ElmList* hasil = findElm(L, cariNopol);
    if (hasil != nullptr) {
        cout << "\nNomor Polisi : " << hasil->info.nopol << endl;
        cout << "Warna        : " << hasil->info.warna << endl;
        cout << "Tahun        : " << hasil->info.thnBuat << endl;
    } else {
        cout << "\nNomor polisi " << cariNopol << " tidak ditemukan." << endl;
    }

    // Menghapus elemen dengan nomor polisi yang diminta
    cout << "\nMasukkan Nomor Polisi yang akan dihapus: ";
    cin >> cariNopol;

    ElmList* P = findElm(L, cariNopol);
    if (P != nullptr) {
        if (P == L.First) {
            deleteFirst(L, P);
        } else if (P == L.Last) {
            deleteLast(L, P);
        } else {
            deleteAfter(P->prev, P);
        }
        dealokasi(P);
        cout << "\nData dengan nomor polisi " << cariNopol << " berhasil dihapus." << endl;
    } else {
        cout << "\nNomor polisi " << cariNopol << " tidak ditemukan." << endl;
    }

    // Cetak data setelah penghapusan
    cout << "\nDATA LIST 1" << endl;
    printInfo(L);

    return 0;
}
```

Outputnya

```
"C:\codeblocks\pertemuan ke
DATA LIST 1
Nomor Polisi : D001
Warna        : hitam
Tahun        : 90

Nomor Polisi : D003
Warna        : putih
Tahun        : 70

Nomor Polisi : D004
Warna        : kuning
Tahun        : 90


Masukkan Nomor Polisi yang dicari: D003

Nomor Polisi : D003
Warna        : putih
Tahun        : 70

Masukkan Nomor Polisi yang akan dihapus: D004

Data dengan nomor polisi D004 berhasil dihapus.

DATA LIST 1
Nomor Polisi : D001
Warna        : hitam
Tahun        : 90

Nomor Polisi : D003
Warna        : putih
Tahun        : 70


Process returned 0 (0x0)   execution time : 32.202 s
Press any key to continue.
```

2. – Doublelist.h

ain.cpp | doublelist.h | doublelist.cpp

```cpp
#ifndef DOUBLELIST_H
#define DOUBLELIST_H

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

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

struct ElmList;
typedef 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(List L);
void insertFirst(List &L, address P); // Menggantí insertLast dengan insertFirst
address findElm(List L, infotype x);

#endif
```

- Doublelist.cpp

in.cpp | doublelist.h | doublelist.cpp

```cpp

#include "doublelist.h"

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 << "\n";
    while (P != nullptr) {
        cout << "No Polisi: " << P->info.nopol << endl;
        cout << "Warna    : " << P->info.warna << endl;
        cout << "Tahun    : " << P->info.thnBuat << endl;
        P = P->next;
    }
}

void insertFirst(List &L, address P) {
    if (L.First == nullptr) {
        L.First = P;
        L.Last = P;
    } else {
        P->next = L.First;
        L.First->prev = P;
        L.First = 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;
}
```

- Main.cpp

```
#include "doublelist.h"

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

    // Data yang sudah tersimpan sebelumnya
    infotype kendaraan1 = {"D001", "hitam", 90};
    infotype kendaraan2 = {"D003", "putih", 70};
    infotype kendaraan3 = {"D004", "kuning", 90};

    insertFirst(L, alokasi(kendaraan1));
    insertFirst(L, alokasi(kendaraan2));
    insertFirst(L, alokasi(kendaraan3));

    // Tampilkan data list awal
    cout << "DATA LIST 1\n";
    printInfo(L);

    // Input nomor polisi yang ingin dicari
    infotype kendaraanCari;
    cout << "\nMasukkan Nomor Polisi yang dicari: ";
    cin >> kendaraanCari.nopol;

    // Cari data berdasarkan nomor polisi
    address hasilCari = findElm(L, kendaraanCari);
    if (hasilCari != nullptr) {
        cout << "\n";
        cout << "Nomor Polisi : " << hasilCari->info.nopol << endl;
        cout << "Warna        : " << hasilCari->info.warna << endl;
        cout << "Tahun        : " << hasilCari->info.thnBuat << endl;
    } else {
        cout << "\nData dengan nomor polisi " << kendaraanCari.nopol << " tidak ditemukan.\n";
    }

    return 0;
}
```

Outputnya

```
"C:\codeblocks\pertemuan ke
DATA LIST 1

No Polisi: D004
Warna    : kuning
Tahun    : 90
No Polisi: D003
Warna    : putih
Tahun    : 70
No Polisi: D001
Warna    : hitam
Tahun    : 90

Masukkan Nomor Polisi yang dicari: D003

Nomor Polisi : D003
Warna        : putih
Tahun        : 70

Process returned 0 (0x0)   execution time : 5.543 s
Press any key to continue.
```

3.– doublelist.h

```
#ifndef DOUBLELIST_H
#define DOUBLELIST_H

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

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

struct ElmList;
typedef 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(List L);
void insertFirst(List &L, address P);
address findElm(List L, infotype x);

// Deklarasi fungsi untuk menghapus elemen
void deleteFirst(List &L, address &P);
void deleteLast(List &L, address &P);
void deleteAfter(List &L, address Prec, address &P);

#endif
```

- doublelist.cpp

```cpp
#include "doublelist.h"

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 << "\n";
    while (P != nullptr) {
        cout << "No Polisi : " << P->info.nopol << endl;
        cout << "Warna     : " << P->info.warna << endl;
        cout << "Tahun     : " << P->info.thnBuat << endl;
        P = P->next;
    }
}

void insertFirst(List &L, address P) {
    if (L.First == nullptr) {
        L.First = P;
        L.Last = P;
    } else {
        P->next = L.First;
        L.First->prev = P;
        L.First = 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;
}

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 = L.First->next;
            L.First->prev = nullptr;
        }
        dealokasi(P);
    }
}

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 = L.Last->prev;
            L.Last->next = nullptr;
        }
        dealokasi(P);
    }
}

void deleteAfter(List &L, 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;
        } else {
            L.Last = Prec;
        }
        dealokasi(P);
    }
}
```

- Main.cpp

```cpp
#include "doublelist.h"

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

    infotype kendaraan1 = {"D001", "hitam", 90};
    infotype kendaraan2 = {"D003", "putih", 70};
    infotype kendaraan3 = {"D004", "kuning", 90};

    insertFirst(L, alokasi(kendaraan1));
    insertFirst(L, alokasi(kendaraan2));
    insertFirst(L, alokasi(kendaraan3));

    cout << "DATA LIST 1\n";
    printInfo(L);

    infotype kendaraanHapus;
    cout << "Masukkan Nomor Polisi yang akan dihapus: ";
    cin >> kendaraanHapus.nopol;

    address P = findElm(L, kendaraanHapus);
    if (P != nullptr) {
        if (P == L.First) {
            deleteFirst(L, P);
        } else if (P == L.Last) {
            deleteLast(L, P);
        } else {
            address Prec = P->prev;
            deleteAfter(L, Prec, P);
        }
        cout << "Data dengan nomor polisi " << kendaraanHapus.nopol << " berhasil dihapus.\n";
    } else {
        cout << "Data dengan nomor polisi " << kendaraanHapus.nopol << " tidak ditemukan.\n";
    }

    cout << "\nDATA LIST 1\n";
    printInfo(L);

    return 0;
}
```

Outputnya

```
"C:\codeblocks\pertemuan ke
DATA LIST 1

No Polisi : D004
Warna     : kuning
Tahun     : 90
No Polisi : D003
Warna     : putih
Tahun     : 70
No Polisi : D001
Warna     : hitam
Tahun     : 90
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

Process returned 0 (0x0)   execution time : 5.462 s
Press any key to continue.
```

5. Kesimpulan

Double Linked List (DLL) adalah struktur data dinamis yang terdiri dari rangkaian elemen (*node*) yang saling terhubung secara berurutan. Setiap elemen dalam DLL memiliki dua pointer, yaitu pointer *next* yang menunjuk ke elemen berikutnya dan pointer *prev* yang menunjuk ke elemen sebelumnya. Dengan adanya dua pointer ini, DLL memungkinkan penelusuran baik dari awal ke akhir (*forward traversal*) maupun dari akhir ke awal (*backward traversal*), yang membuatnya lebih fleksibel dibandingkan *singly linked list* yang hanya memiliki satu arah. Dalam operasi dasar seperti penambahan dan penghapusan elemen, DLL relatif efisien karena kita dapat langsung mengakses elemen sebelumnya atau berikutnya tanpa harus melalui seluruh elemen dari awal. Struktur ini sangat berguna untuk aplikasi yang membutuhkan navigasi dua arah, seperti implementasi *undo-redo*, sistem manajemen memori, atau antrian dengan akses bolak-balik. Meskipun lebih fleksibel,

kelemahan utama dari DLL adalah konsumsi memori yang lebih tinggi dibandingkan dengan *singly linked list*, karena setiap *node* menyimpan dua pointer tambahan.