

LAPORAN PRAKTIKUM
Double Linked List
Bagian Pertama



Disusun Oleh:

Ryan Gabriel Togar Simamora (2311104045)

Kelas : SE0702

Dosen :

Wahyu Andi Saputra

PROGRAM STUDI S1 SOFTWARE ENGINEERING

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. Landasan Teori

A. Double Linked List

Double Linked List adalah jenis linked list di mana setiap elemen atau node memiliki dua pointer:

- ❖ Prev (sebelumnya): menunjuk ke node sebelum node tersebut.
- ❖ Next (berikutnya): menunjuk ke node setelah node tersebut.

Dengan adanya dua pointer ini, Double Linked List memungkinkan traversal atau penelusuran dari dua arah, baik maju (dari awal ke akhir) maupun mundur (dari akhir ke awal). Ini menjadikannya lebih fleksibel dibandingkan Single Linked List, yang hanya bisa ditelusuri dalam satu arah.

B. Karakteristik Double Linked List

- ❖ Node: Merupakan elemen yang menyimpan data dan dua pointer, yaitu pointer next dan prev.
- ❖ First (Awal): Menunjuk ke node pertama dalam list.
- ❖ Last (Akhir): Menunjuk ke node terakhir dalam list.

C. Struktur Data Double Linked List

Untuk membuat Double Linked List dalam C++, kita perlu mendefinisikan beberapa hal seperti tipe data elemen, pointer ke elemen-elemen sebelumnya dan selanjutnya, dan struktur dari list itu sendiri. Berikut adalah contoh struktur data dasar untuk Double Linked List:

```
#ifndef doublelist_H
#define doublelist_H
#include "boolean.h" // Digunakan jika kita memerlukan tipe data boolean
#define Nil NULL
#define info(P) (P)->info
#define next(P) (P)->next
#define prev(P) (P)->prev
#define first(L) ((L).first)
#define last(L) ((L).last)
```

```
typedef int infotype; // Mendefinisikan tipe data elemen list sebagai integer
typedef struct elmlist *address;
```

```
// Definisi node atau elemen dalam Double Linked List
```

```
struct elmlist {
    infotype info; // Menyimpan data/info dari elemen
    address next; // Pointer ke elemen berikutnya
    address prev; // Pointer ke elemen sebelumnya
};
```

```
// Definisi struktur Double Linked List
```

```
struct list {
    address first; // Pointer ke elemen pertama dalam list
    address last; // Pointer ke elemen terakhir dalam list
};
#endif
```

D. Operasi Dasar pada Double Linked List

Operasi-operasi dasar pada Double Linked List mirip dengan Single Linked List, namun dengan tambahan fleksibilitas akses dua arah.

1. Operasi Insert (Menambahkan Elemen)

Operasi Insert digunakan untuk menambahkan elemen baru ke dalam list. Terdapat beberapa jenis Insert dalam Double Linked List:

- ❖ Insert First (Menambahkan di Awal List): Proses ini menambahkan elemen baru di posisi awal list. Berikut adalah langkah-langkahnya:

```
P = alokasi(X);      // Alokasi elemen baru P dengan info X
next(P) = first(L);  // Set pointer next dari P ke elemen pertama yang ada
prev(first(L)) = P;  // Set prev elemen pertama ke P
first(L) = P;        // P menjadi elemen pertama (first)
```

- ❖ Insert Last (Menambahkan di Akhir List): Operasi ini menambahkan elemen baru di posisi akhir list.

```
P = alokasi(X);      // Alokasi elemen baru P dengan info X
prev(P) = last(L);   // Set pointer prev dari P ke elemen terakhir yang ada
next(last(L)) = P;   // Set next elemen terakhir ke P
last(L) = P;         // P menjadi elemen terakhir (last)
```

- ❖ Insert After (Menambahkan Setelah Elemen Tertentu): Menambahkan elemen baru setelah elemen tertentu, misalnya setelah elemen R.

```
P = alokasi(X);      // Alokasi elemen baru P dengan info X
next(P) = next(R);   // Set pointer next dari P ke elemen setelah R
prev(P) = R;         // Set pointer prev dari P ke R
prev(next(R)) = P;   // Set pointer prev dari elemen setelah R ke P
next(R) = P;         // Set pointer next dari R ke P
```

- ❖ Insert Before (Menambahkan Sebelum Elemen Tertentu): Mirip dengan *Insert After*, hanya saja posisi elemen baru berada sebelum elemen tertentu. Cara ini bisa dilakukan dengan mencari elemen yang ingin ditambahkan sebelum posisi P.

2. Operasi Delete (Menghapus Elemen)

Operasi Delete digunakan untuk menghapus elemen tertentu dari list. Beberapa jenis Delete dalam Double Linked List adalah sebagai berikut:

- ❖ Delete First (Menghapus Elemen Pertama): Menghapus elemen pertama dari list dan menjadikan elemen berikutnya sebagai elemen pertama.

```
P = first(L);        // Menyimpan elemen pertama ke variabel P
first(L) = next(first(L)); // Update first ke elemen berikutnya
prev(first(L)) = Nil; // Set prev elemen baru pertama menjadi NULL
return P;            // Kembalikan P atau dealokasi(P)
```

- ❖ Delete Last (Menghapus Elemen Terakhir): Menghapus elemen terakhir dari list dan menjadikan elemen sebelumnya sebagai elemen terakhir.

```
P = last(L);         // Menyimpan elemen terakhir ke variabel P
last(L) = prev(last(L)); // Update last ke elemen sebelumnya
next(last(L)) = Nil;  // Set next elemen baru terakhir menjadi NULL
return P;            // Kembalikan P atau dealokasi(P)
```

- ❖ Delete After (Menghapus Elemen Setelah Elemen Tertentu): Menghapus elemen yang berada setelah elemen tertentu.

```
next(R) = next(P);      // Update next dari R ke elemen setelah P
prev(next(P)) = R;     // Set prev dari elemen setelah P ke R
return P;              // Kembalikan P atau dealokasi(P)
```

- ❖ Delete Before (Menghapus Elemen Sebelum Elemen Tertentu): Mirip dengan Delete After, tetapi menghapus elemen sebelum elemen tertentu.

3. Operasi Lainnya: Update, View, dan Searching

- ❖ View: Menampilkan elemen-elemen dalam list. Dengan Double Linked List, penelusuran dapat dilakukan dari first ke last atau sebaliknya.
- ❖ Searching: Mencari elemen berdasarkan nilai tertentu. Dengan adanya pointer prev dan next, pencarian bisa dilakukan dua arah.
- ❖ Update: Mengubah nilai elemen tertentu. Operasi ini hampir sama dengan Single Linked List, tetapi lebih fleksibel karena kita bisa mengakses elemen dari dua arah.

E. Keunggulan dan Kekurangan Double Linked List

Keunggulan:

1. Navigasi Dua Arah: Dapat menelusuri list dari depan ke belakang atau sebaliknya.
2. Kemudahan Manipulasi Data: Lebih mudah untuk menambahkan atau menghapus elemen dibandingkan Single Linked List karena adanya pointer prev dan next.

Kekurangan:

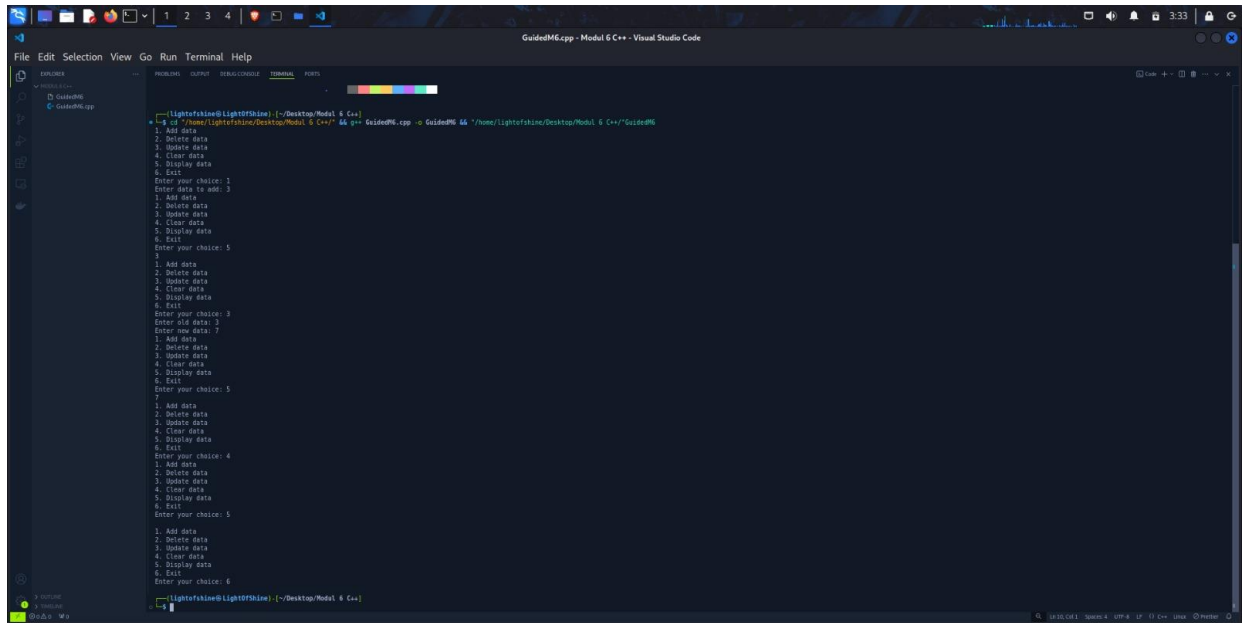
1. Memerlukan Memori Lebih: Karena setiap node memiliki dua pointer, membutuhkan memori lebih besar.
2. Kompleksitas Operasi: Manipulasi node memerlukan penanganan dua pointer, sehingga lebih kompleks dibandingkan Single Linked List.

III. Guided

File GuidedM6.cpp

```
GuidedM6.cpp - Modul 6 C++ - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
  MODULE 6 C++
    GuidedM6
      GuidedM6.cpp
GuidedM6.cpp x
1 #include <iostream>
2 using namespace std;
3
4 class Node {
5 public:
6     int data;
7     Node* prev;
8     Node* next;
9 };
10
11 class DoublyLinkedList {
12 public:
13     Node* head;
14     Node* tail;
15
16     // Constructor untuk inisialisasi head dan tail
17     DoublyLinkedList() {
18         head = nullptr;
19         tail = nullptr;
20     }
21
22     // Fungsi untuk menambahkan elemen di depan list
23     void insert(int data) {
24         Node* newNode = new Node;
25         newNode->data = data;
26         newNode->prev = nullptr;
27         newNode->next = head;
28
29         if (head != nullptr) {
30             head->prev = newNode;
31         } else {
32             tail = newNode; // Jika list kosong, tail juga mengarah ke node baru
33         }
34         head = newNode;
35     }
36
37     // Fungsi untuk menghapus elemen dari depan list
38     void deleteNode() {
39         if (head == nullptr) {
40             return; // Jika list kosong
41         }
42     }
43 }
```

Outputnya



```
Lightofshin@Lightofshin: [~/Desktop/Modul 6 C++]
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 1
Enter data to add: 3
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 2
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 3
Enter old data: 3
Enter new data: 7
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
5. Display data
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 4
4. Clear data
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
5. Display data
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 6
6. Exit
```

Untuk Source Codenya Lebih Lengkap Dibawah ini :



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

// Node structure
struct Node {
    int data;
    Node* prev;
    Node* next;
};

// DoublyLinkedList structure
struct DoublyLinkedList {
    Node* head;
    Node* tail;
};

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

// Fungsi untuk menambahkan elemen di depan list
void insertFront(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 deleteFront() {
    if (head == nullptr) {
        return; // Jika list kosong
    }
    Node* temp = head;
    head = head->next;
    if (head == nullptr) {
        head = nullptr;
    } else {
        head->prev = nullptr;
    }
    delete temp;
}

// Fungsi untuk menambahkan elemen di belakang list
void insertBack(int data) {
    Node* newNode = new Node;
    newNode->data = data;
    newNode->prev = tail;
    newNode->next = nullptr;
    if (tail != nullptr) {
        tail->next = newNode;
    } else {
        head = newNode;
    }
    tail = newNode;
}

// Fungsi untuk menghapus elemen dari belakang list
void deleteBack() {
    if (tail == nullptr) {
        return; // Jika list kosong
    }
    Node* temp = tail;
    tail = tail->prev;
    if (tail == nullptr) {
        head = nullptr;
    }
    delete temp;
}

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

// Fungsi untuk menampilkan status list
void status() {
    cout << "Head: " << head << endl;
    cout << "Tail: " << tail << endl;
}

// Main function
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.insertFront(data);
                break;
            }
            case 2: {
                list.deleteFront();
                break;
            }
            case 3: {
                int oldData, newData;
                cout << "Enter old data: ";
                cin >> oldData;
                cout << "Enter new data: ";
                cin >> newData;
                list.update(oldData, newData);
                if (list.update(oldData, newData) == false) {
                    cout << "Data not found" << endl;
                }
                break;
            }
            case 4: {
                list.deleteBack();
                break;
            }
            case 5: {
                list.display();
                break;
            }
            case 6: {
                return 0;
            }
            default: {
                cout << "Invalid choice" << endl;
                break;
            }
        }
    }
}
```

V. Unguided

6.1 Latihan

1. Buatlah ADT *Double Linked list* sebagai berikut di dalam file “doublelist.h”:

```
Type infotype : kendaraan <
  nopol : string
  warna : string
  thnBuat : integer
>
Type address : pointer to ElmList
Type ElmList <
  info : infotype
  next : address
  prev : address
>
Type List <
  First : address
  Last : address
>
prosedur CreateList( in/out L : List )
fungsi alokasi( x : infotype ) : address
prosedur dealokasi( in/out P : address )
prosedur printInfo( in L : List )
prosedur insertLast( in/out L : List, in P : address )
```

Buatlah implementasi ADT *Double Linked list* pada file “doublelist.cpp” dan coba hasil implementasi ADT pada file “main.cpp”.

Contoh Output :

```
masukkan nomor polisi: D001
masukkan warna kendaraan: hitam
masukkan tahun kendaraan: 90

masukkan nomor polisi: D003
masukkan warna kendaraan: putih
masukkan tahun kendaraan: 70

masukkan nomor polisi: D001
masukkan warna kendaraan: merah
masukkan tahun kendaraan: 80
nomor polisi sudah terdaftar

masukkan nomor polisi: D004
masukkan warna kendaraan: kuning
masukkan tahun kendaraan: 90

DATA LIST 1

no polisi : D004
warna      : kuning
tahun      : 90
no polisi : D003
warna      : putih
tahun      : 70
no polisi : D001
warna      : hitam
tahun      : 90
```

2. Carilah elemen dengan nomor polisi D001 dengan membuat fungsi baru. fungsi findElm(L : List, x : infotype) : address

```
Masukkan Nomor Polisi yang dicari : D001
Nomor Polisi : D001
Warna : hitam
Tahun : 90
```

3. Hapus elemen dengan nomor polisi D003 dengan prosedur *delete*.
- prosedur deleteFirst(in/out L : List, in/out P : address)
 - prosedur deleteLast(in/out L : List, in/out P : address)
 - prosedur deleteAfter(in Prec : address, in/out: P : address)

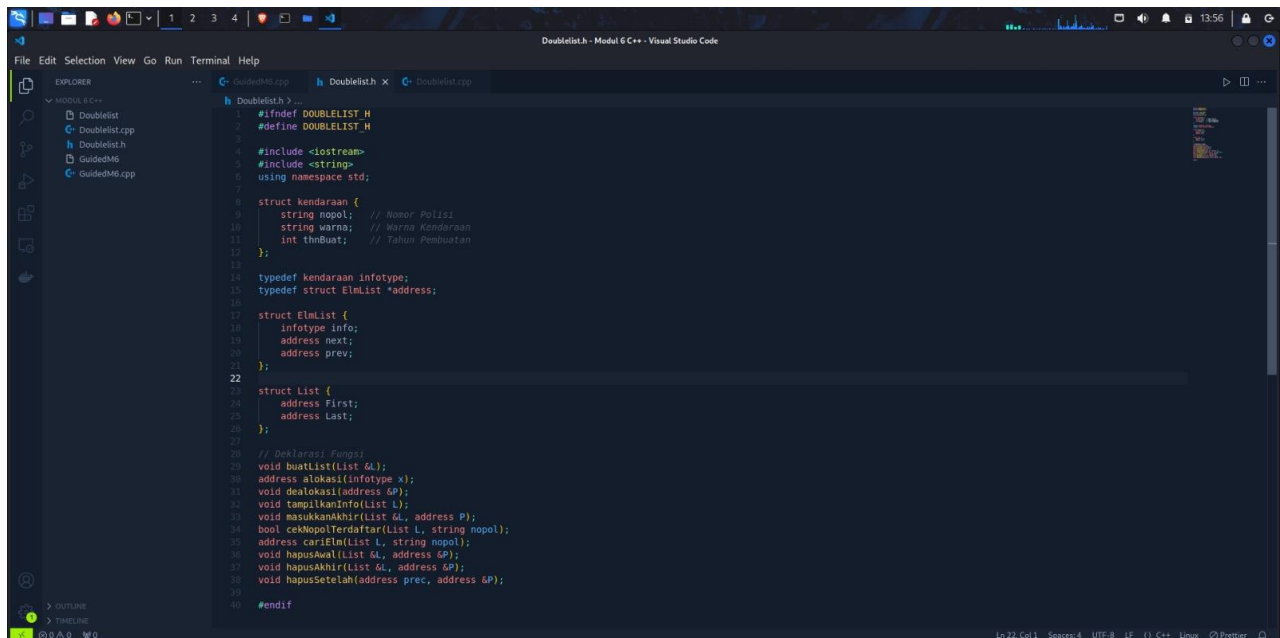
```
Masukkan Nomor Polisi yang akan dihapus : D003
Data dengan nomor polisi D003 berhasil dihapus.

DATA LIST 1

Nomor Polisi : D004
Warna : kuning
Tahun : 90
Nomor Polisi : D001
Warna : hitam
Tahun : 90
```

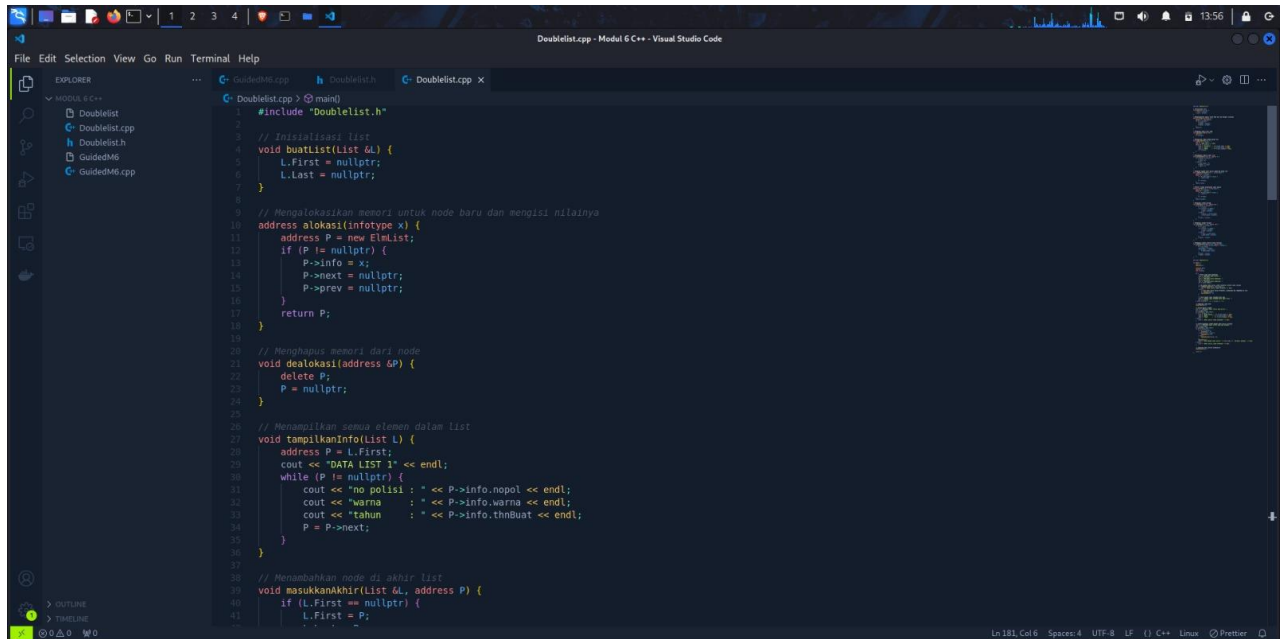
Jawab :

File Doublelist.h



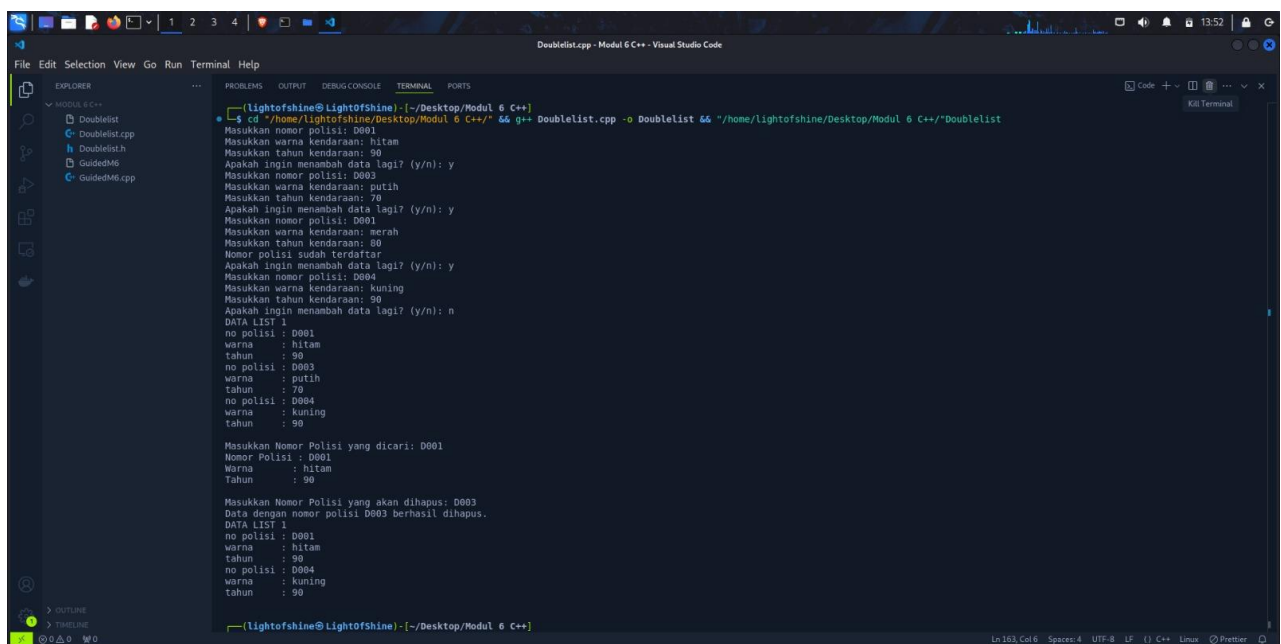
```
1 #ifndef DOUBLELIST_H
2 #define DOUBLELIST_H
3
4 #include <iostream>
5 #include <string>
6 using namespace std;
7
8 struct kendaraan {
9     string nopol; // Nomor Polisi
10    string warna; // Warna Kendaraan
11    int thnBuat; // Tahun Pembuatan
12 };
13
14 typedef kendaraan infotype;
15 typedef struct Elmlist *address;
16
17 struct Elmlist {
18     infotype info;
19     address next;
20     address prev;
21 };
22
23 struct List {
24     address First;
25     address Last;
26 };
27
28 // Deklarasi Fungsi
29 void buatList(List &L);
30 address alokasi(infotype x);
31 void dealokasi(address &P);
32 void tampilkanInfo(List L);
33 void masukkanAkhir(List &L, address P);
34 bool cekNopolTerdaftar(List L, string nopol);
35 address cariElm(List L, string nopol);
36 void hapusAwal(List &L, address &P);
37 void hapusAkhir(List &L, address &P);
38 void hapusSetelah(address prec, address &P);
39
40 #endif
```

File Doublelist.cpp



```
1 #include "Doublelist.h"
2
3 // Inisialisasi list
4 void buatList(List &L) {
5     L.First = nullptr;
6     L.Last = nullptr;
7 }
8
9 // Mengalokasikan memori untuk node baru dan mengisi nilainya
10 address alokasi(infotype x) {
11     address P = new Elmlist;
12     if (P != nullptr) {
13         P->info = x;
14         P->next = nullptr;
15         P->prev = nullptr;
16     }
17     return P;
18 }
19
20 // Menghapus memori dari node
21 void dealokasi(address &P) {
22     delete P;
23     P = nullptr;
24 }
25
26 // Menampilkan semua elemen dalam list
27 void tampilkanInfo(List L) {
28     address P = L.First;
29     cout << "DATA LIST 1" << endl;
30     while (P != nullptr) {
31         cout << "no polisi : " << P->info.nopol << endl;
32         cout << "warna : " << P->info.warna << endl;
33         cout << "tahun : " << P->info.thnBuat << endl;
34         P = P->next;
35     }
36 }
37
38 // Menambahkan node di akhir list
39 void masukkanAkhir(List &L, address P) {
40     if (L.First == nullptr) {
41         L.First = P;
42     }
```

Output



```
Lightofshine@Lightofshine:~/Desktop/Modul 6 C++$ g++ Doublelist.cpp -o Doublelist && ./Doublelist
Masukkan nomor polisi: D001
Masukkan warna kendaraan: hitam
Masukkan tahun kendaraan: 90
Apakah ingin menambah data lagi? (y/n): y
Masukkan nomor polisi: D003
Masukkan warna kendaraan: putih
Masukkan tahun kendaraan: 70
Apakah ingin menambah data lagi? (y/n): y
Masukkan nomor polisi: D001
Masukkan warna kendaraan: merah
Masukkan tahun kendaraan: 80
Nomor polisi sudah terdaftar
Apakah ingin menambah data lagi? (y/n): y
Masukkan nomor polisi: D004
Masukkan warna kendaraan: kuning
Masukkan tahun kendaraan: 90
Apakah ingin menambah data lagi? (y/n): n
DATA LIST 1
no polisi : D001
warna : hitam
tahun : 90
no polisi : D003
warna : putih
tahun : 70
no polisi : D004
warna : kuning
tahun : 90

Masukkan Nomor Polisi yang dicari: D001
Nomor 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 : D001
warna : hitam
tahun : 90
no polisi : D004
warna : kuning
tahun : 90
```


Untuk Source code Doublelist.cpp lebih lengkapnya dibawah ini :

```
#include "Doublelist.h"

// Inisialisasi list
void buatList(list &li) {
    li.first = nullptr;
    li.last = nullptr;
}

// Mengalokasikan memori untuk node baru dan menginisialisasinya
address alokasiIntType xi {
    address P = new Element;
    if (P != nullptr) {
        P->info = xi;
        P->next = nullptr;
        P->prev = nullptr;
    }
    return P;
}

// Menghapus memori dari node
void dealokasi(address &P) {
    delete P;
    P = nullptr;
}

// Menampilkan semua elemen dalam list
void tampilkanInfoList(list &li) {
    address P = li.first;
    cout << "DATA LIST : " << endl;
    while (P != nullptr) {
        cout << "no polisi : " << P->info.noPolisi << endl;
        cout << "nama : " << P->info.nama << endl;
        cout << "tahun : " << P->info.tahun << endl;
        P = P->next;
    }
}

// Menambahkan node di akhir list
void tambahAkhir(list &li, address P) {
    if (li.first == nullptr) {
        li.first = P;
        li.last = P;
    } else {
        li.last->next = P;
        P->prev = li.last;
        li.last = P;
    }
}

// Mengcek apakah nomor polisi sudah ada dalam list
bool cekDuplikatPolisi(list &li, string ngpol) {
    address P = li.first;
    while (P != nullptr) {
        if (P->info.noPolisi == ngpol) {
            return true;
        }
        P = P->next;
    }
    return false;
}

// Mencari elemen berdasarkan nomor polisi
address cariPolisi(list &li, string ngpol) {
    address P = li.first;
    while (P != nullptr) {
        if (P->info.noPolisi == ngpol) {
            return P;
        }
        P = P->next;
    }
    return nullptr;
}

// Menghapus elemen pertama
void hapusPertama(list &li, address &P) {
    if (li.first == nullptr) {
        P = li.first;
        if (li.first == li.last) {
            li.first = nullptr;
            li.last = nullptr;
        } else {
            li.first = li.first->next;
            li.first->prev = nullptr;
        }
        P->next = nullptr;
    }
}

// Menghapus elemen terakhir
void hapusAkhir(list &li, address &P) {
    if (li.last == nullptr) {
        P = li.last;
        if (li.first == li.last) {
            li.first = nullptr;
            li.last = nullptr;
        } else {
            li.last = li.last->prev;
            li.last->next = nullptr;
        }
        P->prev = nullptr;
    }
}

// Menghapus elemen setelah elemen tertentu
void hapusSetelah(address &prev, address &P) {
    if (prev != nullptr && prev->next != nullptr) {
        prev->next = prev->next->next;
        if (prev->next == nullptr) {
            P->prev = prev;
        }
        P->next = nullptr;
        P->prev = nullptr;
    }
}

#include "Doublelist.h"

int main() {
    list li;
    buatList(li);
    int type data;
    address P;
    char pilihan;

    do {
        // Minta input data baru
        cout << "Masukkan nomor polisi : ";
        cin >> data.noPolisi;
        cout << "Masukkan nama kendaraan : ";
        cin >> data.nama;
        cout << "Masukkan tahun kendaraan : ";
        cin >> data.tahun;

        // Cek apakah nomor polisi sudah terdaftar setelah input selesai
        if (cekDuplikatPolisi(li, data.noPolisi)) {
            cout << "Nomor polisi sudah terdaftar" << endl;
        } else {
            // Jika nomor polisi belum terdaftar, alokasikan dan tambahkan ke list
            P = alokasiIntType(data);
            tambahAkhir(li, P);
        }

        // Tanya apakah ingin menambah data lagi (y/n)
        cout << "Apakah ingin menambah data lagi? (y/n) : ";
        cin >> pilihan;
    } while (pilihan == 'y' || pilihan == 'Y');

    // Tampilkan semua data
    tampilkanInfoList(li);

    // Carilah nomor polisi
    cout << "Masukkan Nomor Polisi yang dicari : ";
    cin >> data.noPolisi;
    P = cariPolisi(li, data.noPolisi);
    if (P != nullptr) {
        cout << "Nomor Polisi : " << P->info.noPolisi << endl;
        cout << "nama : " << P->info.nama << endl;
        cout << "tahun : " << P->info.tahun << endl;
    } else {
        cout << "Nomor polisi tidak ditemukan" << endl;
    }

    // Carilah hapus elemen dengan nomor polisi tertentu
    cout << "Masukkan Nomor Polisi yang akan dihapus : ";
    cin >> data.noPolisi;
    P = cariPolisi(li, data.noPolisi);
    if (P != nullptr) {
        if (P == li.first) {
            hapusPertama(li, P);
        } else if (P == li.last) {
            hapusAkhir(li, P);
        } else {
            hapusSetelah(P->prev, P);
        }
        dealokasi(P);
        cout << "Data dengan nomor polisi " << data.noPolisi << " berhasil dihapus" << endl;
    } else {
        cout << "Nomor polisi tidak ditemukan" << endl;
    }

    // Tampilkan data setelah penghapusan
    tampilkanInfoList(li);

    return 0;
}
```

VI. Kesimpulan

Double Linked List adalah struktur data yang memungkinkan penelusuran dua arah, yaitu dari awal ke akhir dan sebaliknya. Setiap node dalam Double Linked List memiliki dua pointer: prev yang menunjuk ke node sebelumnya, dan next yang menunjuk ke node berikutnya. Ini memberikan fleksibilitas yang lebih besar dibandingkan Single Linked List, terutama dalam hal penambahan dan penghapusan elemen di posisi mana pun dalam list.

Berikut beberapa kesimpulan penting dari Double Linked List:

Struktur Node dan List:

Setiap node memiliki data dan dua pointer (prev dan next) untuk menunjang akses dua arah.

List memiliki dua pointer utama, yaitu first yang menunjuk ke node pertama dan last yang menunjuk ke node terakhir.

Operasi Dasar:

- ❖ Insert: Double Linked List memungkinkan penambahan elemen di awal (Insert First), di akhir (Insert Last), di setelah node tertentu (Insert After), atau sebelum node tertentu (Insert Before).
- ❖ Delete: Double Linked List memungkinkan penghapusan elemen pertama (Delete First), elemen terakhir (Delete Last), elemen setelah node tertentu (Delete After), dan elemen sebelum node tertentu (Delete Before).
- ❖ Update, View, dan Searching: Operasi lain seperti menampilkan, mencari, atau memperbarui elemen menjadi lebih fleksibel berkat navigasi dua arah.

Keunggulan:

1. Navigasi Dua Arah: Memudahkan traversal list dari awal ke akhir atau sebaliknya, meningkatkan efisiensi akses data.
2. Kemudahan Manipulasi Data: Penambahan dan penghapusan elemen di posisi mana pun lebih mudah karena adanya pointer dua arah (prev dan next).

Kekurangan:

1. Memerlukan Memori Lebih: Setiap node membutuhkan dua pointer, sehingga memori yang digunakan lebih besar dibandingkan Single Linked List.
2. Kompleksitas Operasi: Manipulasi node memerlukan penanganan dua pointer, membuat operasinya lebih kompleks dibandingkan Single Linked List.

Secara keseluruhan, Double Linked List merupakan pilihan yang ideal untuk aplikasi yang memerlukan fleksibilitas tinggi dalam pengelolaan data dan traversal dua arah, meskipun membutuhkan lebih banyak memori dan penanganan yang lebih kompleks.

