LAPORAN PRAKTIKUM
Modul 06
“DOUBLE LINKED LIST (BAGIAN PERTAMA)”

Disusun Oleh:
Aji Prasetyo Nugroho - 2211104049
S1SE-07-2

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO

2024

A. Tujuan

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

B. Landasan Teori

Double linked list adalah struktur data berbentuk rantai yang memungkinkan navigasi dua arah melalui pointer pada setiap elemennya, yaitu pointer next yang menunjuk elemen setelahnya dan pointer prev yang menunjuk elemen sebelumnya. Setiap double linked list memiliki dua pointer utama, yaitu first yang menunjuk ke elemen pertama dan last yang menunjuk ke elemen terakhir dalam daftar. Struktur ini memudahkan manipulasi data seperti penyisipan dan penghapusan elemen dari kedua ujung rantai, karena setiap elemen memiliki akses langsung ke elemen di depannya dan di belakangnya.

Operasi dasar dalam double linked list meliputi beberapa fungsi penting, yaitu:

1. Inisialisasi: Membuat list kosong dengan mengatur pointer first dan last menjadi null. Operasi ini penting sebagai langkah awal sebelum memulai manipulasi pada list.
2. Penambahan (Insert):
    - Insert at Beginning: Menambahkan elemen baru di awal list. Elemen baru akan menjadi elemen pertama, di mana pointer next elemen baru menunjuk ke elemen yang sebelumnya menjadi first, dan pointer prev elemen pertama lama menunjuk ke elemen baru.
    - Insert at End: Menambahkan elemen baru di akhir list. Elemen baru akan menjadi last, di mana pointer prev elemen baru menunjuk ke elemen yang sebelumnya menjadi last, dan pointer next elemen terakhir lama menunjuk ke elemen baru.
    - Insert After a Specific Node: Menambahkan elemen setelah elemen tertentu. Elemen baru akan ditempatkan setelah elemen yang ditentukan, dengan pointer next dan prev yang diatur agar tetap konsisten.
3. Penghapusan (Delete):
    - Delete from Beginning: Menghapus elemen pertama (yang ditunjuk oleh

first). Setelah penghapusan, elemen kedua akan menjadi first, dan pointer prev elemen baru first diatur ke null.

- Delete from End: Menghapus elemen terakhir (yang ditunjuk oleh last). Setelah penghapusan, elemen sebelumnya akan menjadi last, dan pointer next elemen baru last diatur ke null.
- Delete a Specific Node: Menghapus elemen tertentu dengan menyesuaikan pointer next dari elemen sebelumnya dan pointer prev dari elemen setelahnya.

4. Traversal (Penelusuran): Mengakses setiap elemen dalam list secara berurutan, baik dari first ke last (ke depan) atau dari last ke first (ke belakang). Traversal ini berguna untuk melihat semua elemen atau mencari elemen tertentu dalam list.
5. Searching (Pencarian): Mencari elemen tertentu berdasarkan nilai atau kondisi yang diinginkan. Pencarian ini dilakukan dengan menelusuri elemen-elemen dalam list, biasanya menggunakan traversal.

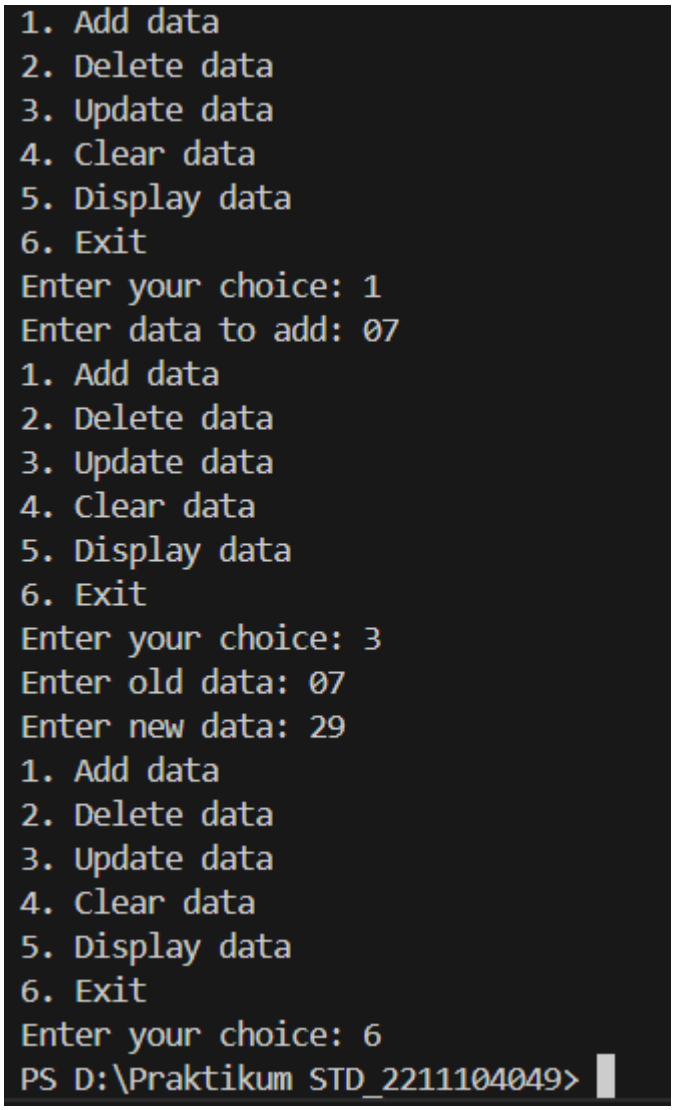
C. Guided

Source Code :

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

class Node {
public:
    int data;
    Node* prev;
    Node* next;
};

class DoublyLinkedList {
public:
    Node* head;
    Node* tail;

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

    // Fungsi untuk menambahkan elemen di depan list
    void insert(int data) {
        Node* newNode = new Node;
        newNode->data = data;
        newNode->prev = nullptr;
        newNode->next = head;

        if (head != nullptr) {
            head->prev = newNode;
        } else {
            tail = newNode; // Jika list kosong, tail juga mengarah ke node baru
        }
        head = newNode;
    }

    // Fungsi untuk menghapus elemen dari depan list
    void deleteNode() {
        if (head == nullptr) {
            return; // Jika list kosong
        }
        Node* temp = head;
        head = head->next;
        if (head != nullptr) {
            head->prev = nullptr;
        } else {
            tail = nullptr; // Jika hanya satu elemen di list
        }
        delete temp; // Hapus elemen
    }

    // Fungsi untuk mengupdate data di list
    bool update(int oldData, int newData) {
        Node* current = head;
        while (current != nullptr) {
            if (current->data == oldData) {
                current->data = newData;
                return true; // Jika data ditemukan dan diupdate
            }
            current = current->next;
        }
        return false; // Jika data tidak ditemukan
    }

    // Fungsi untuk menghapus semua elemen di list
    void deleteAll() {
        Node* current = head;
        while (current != nullptr) {
            Node* temp = current;
            current = current->next;
            delete temp;
        }
        head = nullptr;
        tail = nullptr;
    }

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

```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;
            }
            case 4: {
                list.deleteAll();
                break;
            }
            case 5: {
                list.display();
                break;
            }
            case 6: {
                return 0;
            }
            default: {
                cout << "Invalid choice" << endl;
                break;
            }
        }
    }
    return 0;
}
```

Output :


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: 07
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 3
Enter old data: 07
Enter new data: 29
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 6
PS D:\Praktikum STD_2211104049>

D. Unguided

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
```

Source Code :

a. main.cpp

```cpp
#include "doublelist.h"

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

    infotype data;
    address p;
    char choice = 'y';

    while (choice == 'y' || choice == 'Y') {
        cout << "masukkan nomor polisi: ";
        cin >> data.nopol;
        cout << "masukkan warna kendaraan: ";
        cin >> data.warna;
        cout << "masukkan tahun kendaraan: ";
        cin >> data.tahun;

        p = alokasi(data);
        insertLast(L, p);

        cout << "apakah ingin menambah data lagi? (y/n): ";
        cin >> choice;
    }

    cout << endl << "DATA LIST" << endl;
    printInfo(L);

    return 0;
}
```

b. 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 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;
    }
}

void printInfo(List L) {
    address p = L.last;
    int i = 1;
    while (p != nullptr) {
        cout << "DATA LIST " << i << endl;
        cout << "no polisi : " << p->info.nopol << endl;
        cout << "warna     : " << p->info.warna << endl;
        cout << "tahun     : " << p->info.tahun << endl << endl;
        p = p->prev;
        i++;
    }
}
```

c. doublelist.h

```h
#ifndef DOUBLELIST_H
#define DOUBLELIST_H

#include <iostream>
#include <string>

using namespace std;

typedef struct kendaraan {
    string nopol;
    string warna;
    int tahun;
} infotype;

typedef struct ElmList *address;

struct ElmList {
    infotype info;
    address next;
    address prev;
};

struct List {
    address first;
    address last;
};

// Function Prototypes
void createList(List &L);
address alokasi(infotype x);
void dealokasi(address &p);
void printInfo(List L);
void insertLast(List &L, address p);

#endif
```

Output :

```
masukkan nomor polisi: D001
masukkan warna kendaraan: hitam
masukkan tahun kendaraan: 2019
apakah ingin menambah data lagi? (y/n): y
masukkan nomor polisi: D002
masukkan warna kendaraan: putih
masukkan tahun kendaraan: 2020
apakah ingin menambah data lagi? (y/n): y
masukkan nomor polisi: D003
masukkan warna kendaraan: biru
masukkan tahun kendaraan: 2017
apakah ingin menambah data lagi? (y/n): n

DATA LIST
DATA LIST 1
no polisi : D003
warna     : biru
tahun     : 2017

DATA LIST 2
no polisi : D002
warna     : putih
tahun     : 2020

DATA LIST 3
no polisi : D001
warna     : hitam
tahun     : 2019

PS D:\Praktikum STD_2211104049\2211104049_Aji Prasetyo Nugroho\Unguided\soal 1>
```

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
```

a. main.cpp

```cpp
#include "doublelist.h"

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

    infotype data;
    address p;
    char choice = 'y';

    while (choice == 'y' || choice == 'Y') {
        cout << "masukkan nomor polisi: ";
        cin >> data.nopol;
        cout << "masukkan warna kendaraan: ";
        cin >> data.warna;
        cout << "masukkan tahun kendaraan: ";
        cin >> data.tahun;

        p = alokasi(data);
        insertLast(L, p);

        cout << "apakah ingin menambah data lagi? (y/n): ";
        cin >> choice;
    }

    cout << endl << "DATA LIST" << endl;
    printInfo(L);

    string searchNopol;
    cout << "\nMasukkan Nomor Polisi yang dicari: ";
    cin >> searchNopol;

    address found = findElm(L, searchNopol);
    if (found != nullptr) {
        cout << "\nNomor Polisi : " << found->info.nopol << endl;
        cout << "Warna        : " << found->info.warna << endl;
        cout << "Tahun        : " << found->info.tahun << endl;
    } else {
        cout << "\nData tidak ditemukan.\n";
    }

    return 0;
}
```

b. 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 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;
    }
}

void printInfo(List L) {
    address p = L.first;
    int i = 1;
    while (p != nullptr) {
        cout << "DATA LIST " << i << endl;
        cout << "no polisi : " << p->info.nopol << endl;
        cout << "warna     : " << p->info.warna << endl;
        cout << "tahun     : " << p->info.tahun << endl << endl;
        p = p->next;
        i++;
    }
}

address findElm(List L, string nopol) {
    address p = L.first;
    while (p != nullptr) {
        if (p->info.nopol == nopol) {
            return p;
        }
        p = p->next;
    }
    return nullptr;
}
```

c. doublelist.h

```h
#ifndef DOUBLELIST_H
#define DOUBLELIST_H

#include <iostream>
#include <string>

using namespace std;

typedef struct kendaraan {
    string nopol;
    string warna;
    int tahun;
} infotype;

typedef struct 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 insertLast(List &L, address p);
address findElm(List L, string nopol);

#endif
```

Output :

```
masukkan nomor polisi: D001
masukkan warna kendaraan: biru
masukkan tahun kendaraan: 2013
apakah ingin menambah data lagi? (y/n): y
masukkan nomor polisi: D002
masukkan warna kendaraan: hijau
masukkan tahun kendaraan: 2020
apakah ingin menambah data lagi? (y/n): n

DATA LIST
DATA LIST 1
no polisi : D001
warna     : biru
tahun     : 2013

DATA LIST 2
no polisi : D002
warna     : hijau
tahun     : 2020


Masukkan Nomor Polisi yang dicari: D001

Nomor Polisi : D001
Warna        : biru
Tahun        : 2013
PS D:\Praktikum STD_2211104049\2211104049_Aji Prasetyo Nugroho\Unguided\soal 2>
```

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
```

Source Code :

a. main.cpp

```cpp
#include "doublelist.h"

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

    infotype data;
    address p;
    char choice = 'y';

    while (choice == 'y' || choice == 'Y') {
        cout << "masukkan nomor polisi: ";
        cin >> data.nopol;
        cout << "masukkan warna kendaraan: ";
        cin >> data.warna;
        cout << "masukkan tahun kendaraan: ";
        cin >> data.tahun;

        p = alokasi(data);
        insertLast(L, p);

        cout << "apakah ingin menambah data lagi? (y/n): ";
        cin >> choice;
    }

    cout << endl << "DATA LIST" << endl;
    printInfo(L);

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

    address toDelete = findElm(L, deleteNopol);
    if (toDelete != nullptr) {
        if (toDelete == L.first) {
            deleteFirst(L, toDelete);
        } else if (toDelete == L.last) {
            deleteLast(L, toDelete);
        } else {
            deleteAfter(toDelete->prev, toDelete);
        }
        cout << "\nData dengan nomor polisi " << deleteNopol << " berhasil dihapus.\n";
    } else {
        cout << "\nData tidak ditemukan.\n";
    }

    cout << endl << "DATA LIST" << endl;
    printInfo(L);

    return 0;
}
```

b. 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 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;
    }
}

void printInfo(List L) {
    address p = L.first;
    int i = 1;
    while (p != nullptr) {
        cout << "DATA LIST " << i << endl;
        cout << "no polisi : " << p->info.nopol << endl;
        cout << "warna     : " << p->info.warna << endl;
        cout << "tahun     : " << p->info.tahun << endl << endl;
        p = p->next;
        i++;
    }
}

address findElm(List L, string nopol) {
    address p = L.first;
    while (p != nullptr) {
        if (p->info.nopol == 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;
        }
        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 = L.last->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;
    }
}
```

c. doublelist.h

```
#ifndef DOUBLELIST_H
#define DOUBLELIST_H

#include <iostream>
#include <string>

using namespace std;

typedef struct kendaraan {
    string nopol;
    string warna;
    int tahun;
} infotype;

typedef struct 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 insertLast(List &L, address p);
address findElm(List L, string nopol);

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

#endif
```

Output :

```
masukkan nomor polisi: D001
masukkan warna kendaraan: putih
masukkan tahun kendaraan: 2012
apakah ingin menambah data lagi? (y/n): y
masukkan nomor polisi: D002
masukkan warna kendaraan: hitam
masukkan tahun kendaraan: 2015
apakah ingin menambah data lagi? (y/n): y
masukkan nomor polisi: D003
masukkan warna kendaraan: hijau
masukkan tahun kendaraan: 2020
apakah ingin menambah data lagi? (y/n): n

DATA LIST
DATA LIST 1
no polisi : D001
warna     : putih
tahun     : 2012

DATA LIST 2
no polisi : D002
warna     : hitam
tahun     : 2015

DATA LIST 3
no polisi : D003
warna     : hijau
tahun     : 2020
```

```
Masukkan Nomor Polisi yang akan dihapus: D003

Data dengan nomor polisi D003 berhasil dihapus.

DATA LIST
DATA LIST 1
no polisi : D001
warna     : putih
tahun     : 2012

DATA LIST 2
no polisi : D002
warna     : hitam
tahun     : 2015

PS D:\Praktikum STD_2211104049\2211104049_Aji Prasetyo Nugroho\Unguided\soal 3>
```

E. Kesimpulan

Program secara keseluruhan membangun sebuah ADT (Abstract Data Type) Double Linked List yang berfungsi untuk mengelola data kendaraan. Dengan fitur penambahan, pencarian, dan penghapusan elemen, program ini memungkinkan pengguna untuk melakukan berbagai operasi manipulasi data dalam list secara efisien. Double Linked List ini cocok digunakan dalam kasus di mana data perlu ditelusuri, ditambah, atau dihapus dari kedua arah dengan kompleksitas yang lebih rendah. Implementasi ini menunjukkan fleksibilitas dan efisiensi struktur data Double Linked List dalam mengelola data secara dinamis.