LAPORAN PRAKTIKUM

MODUL 6

DOUBLE LINKED LIST (BAGIAN PERTAMA)

Disusun Oleh:

Rizaldy Aulia Rachman (2311104051)

S1SE-07-02

Dosen :

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

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

2.1 Double Linked List

Double Linked List adalah salah satu bentuk struktur data linked list yang memungkinkan traversal (penjelajahan) dalam dua arah: maju (forward) dan mundur (backward). Berbeda dengan **single linked list** yang hanya memungkinkan penjelajahan ke arah depan, double linked list memiliki dua referensi atau pointer pada setiap node, yaitu ke node **berikutnya** (next) dan **sebelumnya** (prev). Struktur ini memudahkan beberapa operasi seperti penghapusan atau penyisipan elemen dari kedua ujung list.

Komponen Utama Double Linked List:

1. **Node**: Setiap elemen dalam double linked list disebut node. Setiap node memiliki tiga komponen:
    - **Info**: Menyimpan informasi atau data (misalnya, dalam latihan, data kendaraan: nomor polisi, warna, dan tahun pembuatan).
    - **Next**: Pointer yang menunjuk ke elemen berikutnya dalam list.
    - **Prev**: Pointer yang menunjuk ke elemen sebelumnya dalam list.
2. **First**: Pointer yang menunjuk ke node pertama dalam list.
3. **Last**: Pointer yang menunjuk ke node terakhir dalam list.

2.2 Operasi Utama pada Double Linked List

1. **Insert (Sisipkan)**:
    - **Insert First**: Menambahkan node di awal list.
    - **Insert Last**: Menambahkan node di akhir list.
    - **Insert After**: Menambahkan node setelah node tertentu.
    - **Insert Before**: Menambahkan node sebelum node tertentu.
2. **Delete (Hapus)**:
    - **Delete First**: Menghapus node pertama dari list.

- **Delete Last**: Menghapus node terakhir dari list.
- **Delete After**: Menghapus node setelah node tertentu.
- **Delete Before**: Menghapus node sebelum node tertentu.

3. **Search (Pencarian)**: Pencarian elemen dalam double linked list dilakukan dengan menelusuri elemen dari awal ke akhir atau dari akhir ke awal.

Keuntungan Double Linked List:

- **Traversal Dua Arah**: Dengan adanya dua pointer (next dan prev), list ini bisa dijelajahi dari depan ke belakang ataupun sebaliknya. Ini memudahkan operasi seperti penghapusan atau penyisipan pada kedua ujung list.
- **Penghapusan Lebih Mudah**: Karena memiliki pointer ke elemen sebelumnya, penghapusan elemen di double linked list lebih mudah daripada di single linked list, di mana kita harus melacak elemen sebelumnya secara manual.

Kerugian Double Linked List:

- **Penggunaan Memori Lebih Banyak**: Setiap node memerlukan dua pointer (prev dan next), sehingga membutuhkan lebih banyak memori dibandingkan single linked list yang hanya membutuhkan satu pointer.
- **Kompleksitas Kode**: Operasi yang melibatkan manipulasi dua pointer bisa lebih kompleks dan berpotensi menyebabkan kesalahan.

III. GUIDED

1. Guided

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;
    }
};

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;
}
```

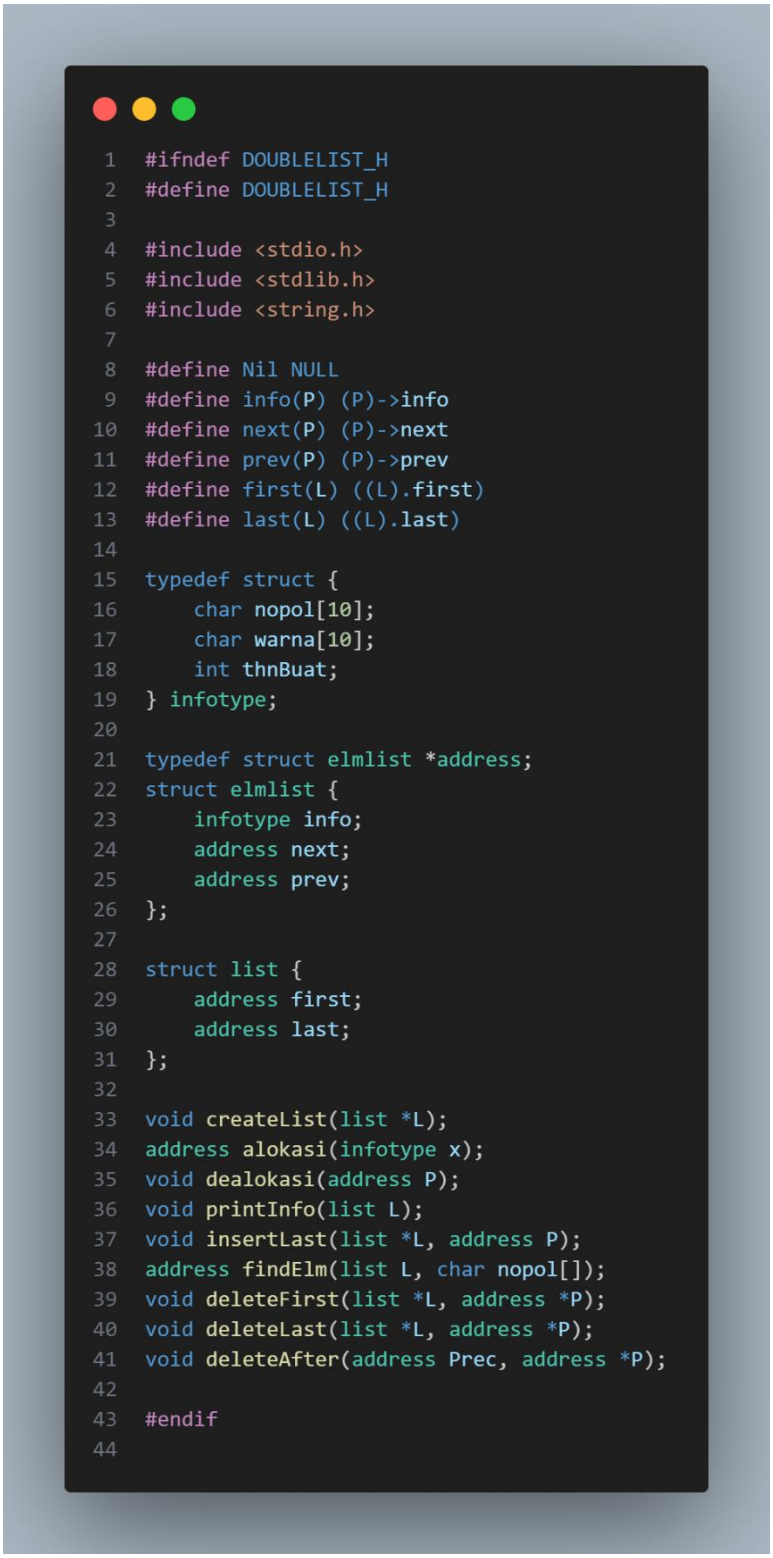
Output:

```
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: 25
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 3
Enter old data: 25
Enter new data: 11
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
11
```

IV. UNGUIDED

1. Doublelist.h

   Code:



```h
#ifndef DOUBLELIST_H
#define DOUBLELIST_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#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 struct {
    char nopol[10];
    char warna[10];
    int thnBuat;
} 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, char nopol[]);
void deleteFirst(list *L, address *P);
void deleteLast(list *L, address *P);
void deleteAfter(address Prec, address *P);

#endif
```

2. Doublelist.cpp
   Code:

```c
#include "doublelist.h"

void createList(list *L) {
    first(*L) = Nil;
    last(*L) = Nil;
}

address alokasi(infotype x) {
    address P = (address)malloc(sizeof(struct elmlist));
    if (P != Nil) {
        strcpy(info(P).nopol, x.nopol);
        strcpy(info(P).warna, x.warna);
        info(P).thnBuat = x.thnBuat;
        next(P) = Nil;
        prev(P) = Nil;
    }
    return P;
}

void dealokasi(address P) {
    free(P);
}

void printInfo(list L) {
    address P = first(L);
    while (P != Nil) {
        printf("NoPol: %s, Warna: %s, Tahun: %d\n", info(P).nopol, info(P).warna, info(P).thnBuat);
        P = next(P);
    }
}

void insertLast(list *L, address P) {
    if (first(*L) == Nil) {
        first(*L) = P;
        last(*L) = P;
    } else {
        prev(P) = last(*L);
        next(last(*L)) = P;
        last(*L) = P;
    }
}

address findElm(list L, char nopol[]) {
    address P = first(L);
    while (P != Nil) {
        if (strcmp(info(P).nopol, nopol) == 0) {
            return P;
        }
        P = next(P);
    }
    return Nil;
}

void deleteFirst(list *L, address *P) {
    *P = first(*L);
    if (first(*L) != Nil) {
        if (next(first(*L)) == Nil) {
            first(*L) = Nil;
            last(*L) = Nil;
        } else {
            first(*L) = next(*P);
            prev(first(*L)) = Nil;
        }
        next(*P) = Nil;
        prev(*P) = Nil;
    }
}

void deleteLast(list *L, address *P) {
    *P = last(*L);
    if (last(*L) != Nil) {
        if (prev(last(*L)) == Nil) {
            first(*L) = Nil;
            last(*L) = Nil;
        } else {
            last(*L) = prev(*P);
            next(last(*L)) = Nil;
        }
        next(*P) = Nil;
        prev(*P) = Nil;
    }
}

void deleteAfter(address Prec, address *P) {
    if (Prec != Nil && next(Prec) != Nil) {
        *P = next(Prec);
        if (next(*P) != Nil) {
            next(Prec) = next(*P);
            prev(next(*P)) = Prec;
        } else {
            next(Prec) = Nil;
        }
        next(*P) = Nil;
        prev(*P) = Nil;
    }
}

```

3. Main.cpp
   Code:

```c
#include "doublelist.h"

int main() {
    list L;
    infotype kendaraan;
    address P;

    createList(&L);

    // Input kendaraan 1
    strcpy(kendaraan.nopol, "D001");
    strcpy(kendaraan.warna, "Hitam");
    kendaraan.thnBuat = 2015;
    P = alokasi(kendaraan);
    insertLast(&L, P);

    // Input kendaraan 2
    strcpy(kendaraan.nopol, "D002");
    strcpy(kendaraan.warna, "Merah");
    kendaraan.thnBuat = 2018;
    P = alokasi(kendaraan);
    insertLast(&L, P);

    // Input kendaraan 3
    strcpy(kendaraan.nopol, "D003");
    strcpy(kendaraan.warna, "Putih");
    kendaraan.thnBuat = 2020;
    P = alokasi(kendaraan);
    insertLast(&L, P);

    // Cetak semua data
    printf("Data Kendaraan:\n");
    printInfo(L);

    // Cari kendaraan dengan NoPol D001
    P = findElm(L, "D001");
    if (P != Nil) {
        printf("Kendaraan dengan NoPol D001 ditemukan: Warna %s, Tahun %d\n", info(P).warna, info(P).thnBuat);
    } else {
        printf("Kendaraan dengan NoPol D001 tidak ditemukan\n");
    }

    // Hapus kendaraan dengan NoPol D003
    deleteLast(&L, &P);
    printf("Kendaraan dengan NoPol D003 telah dihapus\n");

    // Cetak ulang data
    printf("Data Kendaraan setelah penghapusan:\n");
    printInfo(L);

    return 0;
}
```

Output Program:

```
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: 25
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 3
Enter old data: 25
Enter new data: 11
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
11
```