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

Disusun Oleh:
Benedictus Qsota Noventino Baru - 2311104029
S1SE07A

Dosen :
Yudha Islami Sulistya, S.Kom., M.Cs

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM
PURWOKERTO

2024

LATIHAN MODUL

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

struct infotype {
    string nopoli;
    string warna;
    int tahun;
};

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

struct List {
    ElmList *first;
    ElmList *last;
};

void createList(List &L) {
    L.first = nullptr;
    L.last = nullptr;
}

ElmList* allocate(infotype x) {
    ElmList* P = new ElmList;
    P->info = x;
    P->next = nullptr;
    P->prev = nullptr;
    return P;
}

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

void insertLast(List &L, ElmList* 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) {
    ElmList* P = L.first;
    cout << "DATA LIST 1" << endl;
    while (P != nullptr) {
        cout << "No polisi  : " << P->info.nopoli << endl;
        cout << "Warna      : " << P->info.warna << endl;
        cout << "Tahun      : " << P->info.tahun << endl;
        cout << endl;
        P = P->next;
    }
}


```

```cpp
ElmList* findElm(List L, string nopoli) {
    ElmList* P = L.first;
    while (P != nullptr) {
        if (P->info.nopoli == nopoli) {
            return P;
        }
        P = P->next;
    }
    return nullptr;
}

void deleteFirst(List &L, ElmList* &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, ElmList* &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(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;
    }
}

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

    int n;
    cout << "Masukkan jumlah data kendaraan: ";
    cin >> n;

    for (int i = 0; i < n; i++) {
        infotype x;
        cout << "Masukkan nomor polisi: ";
        cin >> x.nopoli;
        cout << "Masukkan warna kendaraan: ";
        cin >> x.warna;
        cout << "Masukkan tahun kendaraan: ";
        cin >> x.tahun;

        ElmList* P = allocate(x);
        insertLast(L, P);
    }

    printInfo(L);

    // Mencari elemen dengan nomor polisi tertentu
    string search_nopoli;
    cout << "Masukkan Nomor Polisi yang dicari: ";
    cin >> search_nopoli;
    ElmList* found = findElm(L, search_nopoli);
    if (found != nullptr) {
        cout << "Nomor Polisi : " << found->info.nopoli << endl;
        cout << "Warna        : " << found->info.warna << endl;
        cout << "Tahun        : " << found->info.tahun << endl;
    } else {
        cout << "Nomor polisi tidak ditemukan." << endl;
    }

    // Menghapus elemen dengan nomor polisi tertentu
    string delete_nopoli;
    cout << "Masukkan Nomor Polisi yang akan dihapus: ";
    cin >> delete_nopoli;
    found = findElm(L, delete_nopoli);
    if (found != nullptr) {
        deleteFirst(L, found); // Contoh penggunaan deleteFirst
        deallocate(found);
        cout << "Data dengan nomor polisi " << delete_nopoli << " berhasil dihapus." << endl;
    }

    printInfo(L);

    return 0;
}
```

Struktur Data

1. **Struct `infotype`**:
    - Menyimpan informasi kendaraan dengan tiga atribut:
        - `nopoli`: Nomor polisi kendaraan (tipe data string).
        - `warna`: Warna kendaraan (tipe data string).
        - `tahun`: Tahun kendaraan (tipe data integer).
2. **Struct `ElmList`**:
    - Node dalam Doubly Linked List yang menyimpan informasi dari `infotype`.
    - Memiliki pointer `next` ke elemen berikutnya dan pointer `prev` ke elemen sebelumnya.
3. **Struct `List`**:
    - Menyimpan pointer `first` ke elemen pertama dan `last` ke elemen terakhir dari list.

Fungsi-fungsi Utama

1. **`createList(List &L)`**:
    - Inisialisasi list dengan `first` dan `last` diatur ke `nullptr`.
2. **`allocate(infotype x)`**:
    - Mengalokasikan memori untuk node baru dan mengisi informasi dari `infotype`.
3. **`deallocate(ElmList* &P)`**:
    - Menghapus node dan mengatur pointer ke `nullptr`.
4. **`insertLast(List &L, ElmList* P)`**:
    - Menambahkan elemen ke akhir list. Jika list kosong, elemen baru menjadi `first` dan `last`.
5. **`printInfo(List L)`**:
    - Menampilkan semua data kendaraan dalam list.
6. **`findElm(List L, string nopoli)`**:
    - Mencari dan mengembalikan pointer ke node yang memiliki nomor polisi tertentu.
7. **`deleteFirst(List &L, ElmList* &P)`**:
    - Menghapus elemen pertama dari list dan mengatur pointer sesuai kondisi list (apakah kosong atau memiliki lebih dari satu elemen).
8. **`deleteLast(List &L, ElmList* &P)`**:
    - Menghapus elemen terakhir dari list.
9. **`deleteAfter(ElmList* Prec, ElmList* &P)`**:
    - Menghapus elemen setelah node tertentu.

Fungsi `main()`

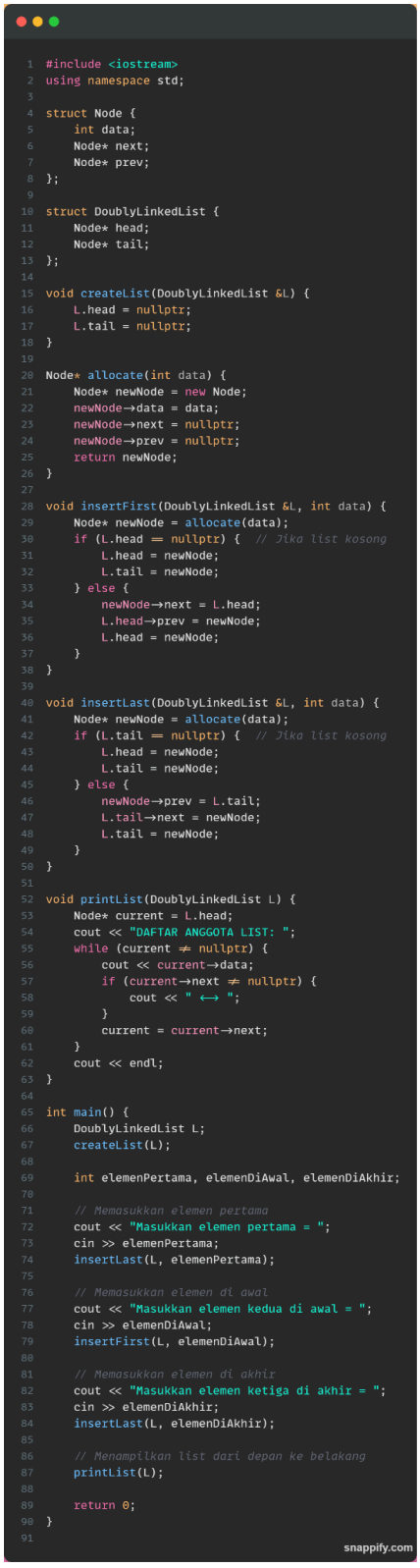
1. **Inisialisasi**: Membuat list baru dengan `createList`.
2. **Input Data**: Meminta pengguna untuk memasukkan jumlah data kendaraan dan informasi kendaraan (nomor polisi, warna, tahun) secara berurutan. Data dimasukkan ke dalam list menggunakan `insertLast`.
3. **Menampilkan Data**: Memanggil `printInfo` untuk menampilkan semua data kendaraan yang telah dimasukkan.
4. **Pencarian Data**: Meminta pengguna untuk memasukkan nomor polisi yang ingin dicari dan menampilkan informasi jika ditemukan.
5. **Penghapusan Data**: Meminta pengguna untuk memasukkan nomor polisi yang akan dihapus. Jika ditemukan, elemen dihapus menggunakan `deleteFirst` (meskipun seharusnya menggunakan `deleteAfter` atau fungsi lain yang lebih tepat) dan kemudian memanggil `deallocate` untuk membebaskan memori.
6. **Menampilkan Data Terakhir**: Memanggil `printInfo` lagi untuk menampilkan list setelah penghapusan.

Kesimpulan

Program ini mengelola data kendaraan menggunakan struktur data Doubly Linked List, memungkinkan pengguna untuk menambah, mencari, dan menghapus data kendaraan secara efisien. Beberapa fungsi dapat disempurnakan lebih lanjut, terutama untuk penghapusan elemen tertentu yang lebih tepat.

Soal TP

Soal 1



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

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

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

void createList(DoublyLinkedList &L) {
    L.head = nullptr;
    L.tail = nullptr;
}

Node* allocate(int data) {
    Node* newNode = new Node;
    newNode->data = data;
    newNode->next = nullptr;
    newNode->prev = nullptr;
    return newNode;
}

void insertFirst(DoublyLinkedList &L, int data) {
    Node* newNode = allocate(data);
    if (L.head == nullptr) {  // Jika list kosong
        L.head = newNode;
        L.tail = newNode;
    } else {
        newNode->next = L.head;
        L.head->prev = newNode;
        L.head = newNode;
    }
}

void insertLast(DoublyLinkedList &L, int data) {
    Node* newNode = allocate(data);
    if (L.tail == nullptr) {  // Jika list kosong
        L.head = newNode;
        L.tail = newNode;
    } else {
        newNode->prev = L.tail;
        L.tail->next = newNode;
        L.tail = newNode;
    }
}

void printList(DoublyLinkedList L) {
    Node* current = L.head;
    cout << "DAFTAR ANGGOTA LIST: ";
    while (current != nullptr) {
        cout << current->data;
        if (current->next != nullptr) {
            cout << " <-> ";
        }
        current = current->next;
    }
    cout << endl;
}

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

    int elemenPertama, elemenDiAwal, elemenDiAkhir;

    // Memasukkan elemen pertama
    cout << "Masukkan elemen pertama = ";
    cin >> elemenPertama;
    insertLast(L, elemenPertama);

    // Memasukkan elemen di awal
    cout << "Masukkan elemen kedua di awal = ";
    cin >> elemenDiAwal;
    insertFirst(L, elemenDiAwal);

    // Memasukkan elemen di akhir
    cout << "Masukkan elemen ketiga di akhir = ";
    cin >> elemenDiAkhir;
    insertLast(L, elemenDiAkhir);

    // Menampilkan list dari depan ke belakang
    printList(L);

    return 0;
}
```

Struktur Data

1. **Struct `Node`**:
    - Merepresentasikan sebuah node dalam Doubly Linked List.
    - Memiliki tiga atribut:
        - `data`: Menyimpan nilai (tipe data integer).
        - `next`: Pointer ke node berikutnya.
        - `prev`: Pointer ke node sebelumnya.
2. **Struct `DoublyLinkedList`**:
    - Menyimpan dua pointer:
        - `head`: Pointer ke elemen pertama dalam list.
        - `tail`: Pointer ke elemen terakhir dalam list.

Fungsi-fungsi Utama

1. **`createList(DoublyLinkedList &L)`**:
    - Menginisialisasi list dengan menetapkan `head` dan `tail` ke `nullptr`, menandakan bahwa list kosong.
2. **`allocate(int data)`**:
    - Mengalokasikan memori untuk node baru dan menginisialisasi nilai `data` untuk node tersebut. Fungsi ini mengembalikan pointer ke node yang baru dibuat.
3. **`insertFirst(DoublyLinkedList &L, int data)`**:
    - Menambahkan elemen baru di awal list. Jika list kosong, node baru menjadi `head` dan `tail`. Jika tidak, node baru diatur sebagai `head` dan pointer `prev` dari node pertama diupdate.
4. **`insertLast(DoublyLinkedList &L, int data)`**:
    - Menambahkan elemen baru di akhir list. Jika list kosong, node baru menjadi `head` dan `tail`. Jika tidak, node baru dihubungkan sebagai `next` dari node terakhir yang ada, dan pointer `prev` diupdate.
5. **`printList(DoublyLinkedList L)`**:
    - Menampilkan semua elemen dalam list dari `head` hingga `tail`. Menggunakan loop untuk traversing, dan mencetak data setiap node dengan format yang sesuai.

Fungsi `main()`

1. **Inisialisasi**: Membuat list baru dengan memanggil `createList`.
2. **Input Elemen**:
    - Meminta pengguna untuk memasukkan elemen pertama yang ditambahkan ke akhir list.
    - Meminta pengguna untuk memasukkan elemen kedua yang akan ditambahkan di awal list.

    - Meminta pengguna untuk memasukkan elemen ketiga yang akan ditambahkan di akhir list.
3. **Menampilkan List**: Memanggil fungsi `printList` untuk menampilkan elemen yang telah dimasukkan ke dalam list.

Soal 2

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

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

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

void createList(DoublyLinkedList &L) {
    L.head = nullptr;
    L.tail = nullptr;
}

Node* allocate(int data) {
    Node* newNode = new Node;
    newNode->data = data;
    newNode->next = nullptr;
    newNode->prev = nullptr;
    return newNode;
}

void insertLast(DoublyLinkedList &L, int data) {
    Node* newNode = allocate(data);
    if (L.tail == nullptr) {  // Jika list kosong
        L.head = newNode;
        L.tail = newNode;
    } else {
        newNode->prev = L.tail;
        L.tail->next = newNode;
        L.tail = newNode;
    }
}

void deleteFirst(DoublyLinkedList &L) {
    if (L.head == nullptr) {
        cout << "List kosong, tidak ada elemen yang bisa dihapus." << endl;
        return;
    }
    Node* temp = L.head;
    if (L.head == L.tail) {  // Jika hanya ada satu elemen
        L.head = nullptr;
        L.tail = nullptr;
    } else {
        L.head = L.head->next;
        L.head->prev = nullptr;
    }
    delete temp;
}

void deleteLast(DoublyLinkedList &L) {
    if (L.tail == nullptr) {
        cout << "List kosong, tidak ada elemen yang bisa dihapus." << endl;
        return;
    }
    Node* temp = L.tail;
    if (L.head == L.tail) {  // Jika hanya ada satu elemen
        L.head = nullptr;
        L.tail = nullptr;
    } else {
        L.tail = L.tail->prev;
        L.tail->next = nullptr;
    }
    delete temp;
}

void printList(DoublyLinkedList L) {
    if (L.head == nullptr) {
        cout << "DAFTAR ANGGOTA LIST SETELAH PENGHAPUSAN: List kosong." << endl;
        return;
    }
    Node* current = L.head;
    cout << "DAFTAR ANGGOTA LIST SETELAH PENGHAPUSAN: ";
    while (current != nullptr) {
        cout << current->data;
        if (current->next != nullptr) {
            cout << " <-> ";
        }
        current = current->next;
    }
    cout << endl;
}

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

    int elemenPertama, elemenKedua, elemenKetiga;

    // Memasukkan elemen-elemen ke dalam list
    cout << "Masukkan elemen pertama = ";
    cin >> elemenPertama;
    insertLast(L, elemenPertama);

    cout << "Masukkan elemen kedua di akhir = ";
    cin >> elemenKedua;
    insertLast(L, elemenKedua);

    cout << "Masukkan elemen ketiga di akhir = ";
    cin >> elemenKetiga;
    insertLast(L, elemenKetiga);

    // Menghapus elemen pertama dan terakhir
    deleteFirst(L);
    deleteLast(L);

    // Menampilkan list setelah penghapusan
    printList(L);

    return 0;
}
```

Struktur Data

1. **Struct `Node`**:
   - Mewakili sebuah node dalam Doubly Linked List.
   - Memiliki tiga atribut:
     - `data`: Menyimpan nilai node (tipe data integer).
     - `next`: Pointer yang menunjuk ke node berikutnya.
     - `prev`: Pointer yang menunjuk ke node sebelumnya.
2. **Struct `DoublyLinkedList`**:
   - Menyimpan dua pointer:
     - `head`: Pointer ke elemen pertama (head) dalam list.
     - `tail`: Pointer ke elemen terakhir (tail) dalam list.

Fungsi-fungsi Utama

1. **`createList(DoublyLinkedList &L)`**:
   - Menginisialisasi list dengan menetapkan `head` dan `tail` ke `nullptr`, menandakan bahwa list kosong.
2. **`allocate(int data)`**:
   - Mengalokasikan memori untuk node baru dan menginisialisasi nilai `data`. Fungsi ini mengembalikan pointer ke node yang baru dibuat.
3. **`insertLast(DoublyLinkedList &L, int data)`**:
   - Menambahkan elemen baru di akhir list.
   - Jika list kosong, node baru menjadi `head` dan `tail`. Jika tidak, node baru dihubungkan dengan node terakhir yang ada.
4. **`deleteFirst(DoublyLinkedList &L)`**:
   - Menghapus elemen pertama dari list.
   - Jika list kosong, mencetak pesan bahwa tidak ada elemen yang bisa dihapus.
   - Jika hanya ada satu elemen, mengatur `head` dan `tail` menjadi `nullptr`.
   - Jika ada lebih dari satu elemen, mengupdate `head` dan menghapus node pertama.
5. **`deleteLast(DoublyLinkedList &L)`**:
   - Menghapus elemen terakhir dari list.
   - Jika list kosong, mencetak pesan bahwa tidak ada elemen yang bisa dihapus.
   - Jika hanya ada satu elemen, mengatur `head` dan `tail` menjadi `nullptr`.
   - Jika ada lebih dari satu elemen, mengupdate `tail` dan menghapus node terakhir.
6. **`printList(DoublyLinkedList L)`**:
   - Menampilkan semua elemen dalam list dari `head` hingga `tail`.
   - Jika list kosong, mencetak pesan bahwa list kosong.

Fungsi `main()`

1. **Inisialisasi**: Membuat list baru dengan memanggil `createList`.
2. **Input Elemen**:
    - Meminta pengguna untuk memasukkan tiga elemen yang akan ditambahkan ke akhir list menggunakan `insertLast`.
3. **Penghapusan Elemen**:
    - Menghapus elemen pertama dengan memanggil `deleteFirst`.
    - Menghapus elemen terakhir dengan memanggil `deleteLast`.
4. **Menampilkan List**:
    - Memanggil `printList` untuk menampilkan elemen yang tersisa dalam list setelah penghapusan.

Soal 3

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

// Node class untuk Doubly Linked List
class Node {
public:
    int data;
    Node* next;
    Node* prev;

    Node(int data) {
        this->data = data;
        next = nullptr;
        prev = nullptr;
    }
};

// DoublyLinkedList class
class DoublyLinkedList {
private:
    Node* head;

public:
    DoublyLinkedList() {
        head = nullptr;
    }

    // Fungsi untuk menambahkan elemen di akhir list
    void append(int data) {
        Node* newNode = new Node(data);
        if (!head) {
            head = newNode;
            return;
        }
        Node* last = head;
        while (last->next) {
            last = last->next;
        }
        last->next = newNode;
        newNode->prev = last;
    }

    // Fungsi untuk menampilkan elemen dari depan ke belakang
    void displayForward() {
        Node* current = head;
        cout << "Daftar elemen dari depan ke belakang: ";
        while (current) {
            cout << current->data;
            if (current->next) {
                cout << " <-> ";
            }
            current = current->next;
        }
        cout << endl;
    }
```

```
// Fungsi untuk menampilkan elemen dari belakang ke depan
    void displayBackward() {
        Node* current = head;
        if (!current) return;

        // Pergi ke node terakhir
        while (current->next) {
            current = current->next;
        }

        cout << "Daftar elemen dari belakang ke depan: ";
        while (current) {
            cout << current->data;
            if (current->prev) {
                cout << " <-> ";
            }
            current = current->prev;
        }
        cout << endl;
    }
};

int main() {
    DoublyLinkedList dll;

    // Memasukkan 4 elemen
    cout << "Masukkan 4 elemen secara berurutan:" << endl;
    for (int i = 0; i < 4; i++) {
        int element;
        cout << "Masukkan elemen: ";
        cin >> element;
        dll.append(element);
    }

    // Menampilkan elemen dari depan ke belakang dan sebaliknya
    dll.displayForward();
    dll.displayBackward();

    return 0;
}

```

Struktur Data

1. **Kelas `Node`**:
    - Mewakili sebuah node dalam Doubly Linked List.
    - Memiliki tiga atribut:
        - `data`: Menyimpan nilai node (tipe data integer).
        - `next`: Pointer yang menunjuk ke node berikutnya.
        - `prev`: Pointer yang menunjuk ke node sebelumnya.
    - Konstruktor untuk menginisialisasi nilai `data` dan menetapkan pointer `next` dan `prev` ke `nullptr`.
2. **Kelas `DoublyLinkedList`**:
    - Memiliki satu atribut:
        - `head`: Pointer ke node pertama dalam list.
    - Konstruktor untuk menginisialisasi `head` menjadi `nullptr`.

Fungsi-fungsi Utama

1. **`append(int data)`**:
    - Menambahkan elemen baru di akhir list.
    - Jika list kosong (tidak ada `head`), maka node baru menjadi `head`.
    - Jika tidak kosong, program akan traversing dari `head` hingga menemukan node terakhir dan menghubungkan node baru di akhir.
2. **`displayForward()`**:
    - Menampilkan semua elemen dalam list dari `head` hingga node terakhir.
    - Menggunakan loop untuk traversing dan mencetak nilai `data` dari setiap node, dipisahkan dengan tanda `<->`.
3. **`displayBackward()`**:
    - Menampilkan semua elemen dalam list dari node terakhir hingga `head`.
    - Pertama, traversing dari `head` hingga node terakhir.
    - Kemudian, mencetak nilai `data` dari setiap node saat traversing kembali ke `head`, dipisahkan dengan tanda `<->`.

Fungsi `main()`

1. **Inisialisasi**: Membuat objek `DoublyLinkedList` yang baru.
2. **Input Elemen**:
    - Meminta pengguna untuk memasukkan empat elemen secara berurutan.
    - Menggunakan loop untuk mengambil input dan memanggil fungsi `append` untuk menambahkan elemen ke dalam list.
3. **Menampilkan List**:
    - Memanggil `displayForward` untuk menampilkan elemen dari depan ke belakang.

- Memanggil `displayBackward` untuk menampilkan elemen dari belakang ke depan.