LAPORAN PRAKTIKUM
Modul 6
DOUBLE LINKED LIST (BAGIAN PERTAMA)

Disusun Oleh:
Aulia Jasifa Br Ginting 2311104060
S1SE-07-02

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**

   6.1 Double Linked List

   Double Linked List adalah struktu data ini memungkinkan navigasi yang terdiri dari kumpulan elemen-elemen dan traversal yang saling terhubung secara dua arah, dalam list dilakukan dalam dua arah, baik ke depan (next) maupun ke belakang (prev). Dalam implementasinya, Double Linked List biasanya terdiri dari struktur data untuk mewakili elemen (ElmList) dan struktur data untuk mewakili keseluruhan list (List).

   6.1.1 Insert

   A. *Insert Fisrt*

   Insert First adalah fungsi untuk menambahkan elemen vari di awal list. Fungsi ini akan menambahkan elemen baru pada posisi pertama (sebelum elemen lain yang sudah ada).

   B. *Insert Last*

   Insert last adalah fungsi untuk menambahkan elemen baru di akhir Double Linked List. Fungsi ini menambahkan node baru sebagai elemen terakhir di list. Jika list kosong, elemen tersebut menjadi elemen pertama, jika tidak, elemen tersebut ditempatkan di akhir list dengan menghubungkan node sebelumnya dengan node baru ini.

   C. *Insert After*

   Insert After pada Double Linked List digunakan untuk menambahkan elemen bar setelah elemen tertentu yang ada di dalam list. Fungsi ini bergguna jiika kita menempatkan node baru diposisi tertentu, setelah node tertentu yang telah diketahui.

   D. *Insert Before*

   Insert Before pada Double Linked List digunakan untuk menambahkan elemen baru sebelum elemen tertentu dalam list. Fungsi ini sangat berguna ketika kita ingin menambahkan node di posisi yang spesifik, sebelum elemen yang sudah ada.

   6.1.2 Delete

   A. Delete First

Fungsi Delete First pada Double Linked List digunakan untuk menghapus elemen pertama dari list. Fungsi ini menghapus node di posisi awal list dan memperbarui pointer agar list tetap terhubung dengan benar setelah penghapusan.

B. Delete Last

Fungsi delete Last pada Double Linked List digunakan untuk menghapus elemen terakhir. Fungsi ini juga dapat mengelola penujuk (pointer) yang mengarah ke node terakhir.

C. Delete After

Fungsi delete after pada Doubel Linked List digunakan untuk meghapus suatu node yang berada setelah node tertentu, lau menghapus node yang ada setelahnya

D. Delete Before

Fungsi delete before pada double linked list digunakan untuk menghapus node yang berada tepat sebelum node tertentu. Dengan kata lain, jika kita memiliki node X, fungsi ini akan menghapus node yang ada sebelum X, yaitu node X.prev.

E. Update, View, dan Searching

Dalam konteks struktur data seperti *double linked list* (DLL) atau dalam sistem manajemen data lainnya, Update, View, dan Searching adalah operasi dasar yang memungkinkan kita untuk memanipulasi, melihat, dan mencari data dalam struktur tersebut.

Update

Adalah operasi yang memungkinkan kita untuk mengubah data yang ada di dalam node tertentu dari struktur data. Dalam *double linked list*, kita dapat memperbarui nilai suatu node setelah menemukannya.

View

Adalah operasi untuk melihat atau menampilkan data yang ada dalam struktur. Dalam *double linked list*, kita biasanya menampilkan semua elemen dengan cara traversing atau menjelajahi list dari head hingga akhir list.

Searching

Adalah operasi untuk mencari node tertentu berdasarkan nilai data yang disimpan di dalamnya. Dalam *double linked list*, kita dapat mencari nilai tertentu dengan menjelajahi setiap node hingga menemukan node yang memiliki nilai yang sesuai.

Telkom University Purwokerto

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;
        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;
}
```

Outputnya;

```
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: 24
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: 10
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: 15
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 3
Enter old data: 24
Enter new data: 15
```

4. Unguided

Telkom University Purwokerto

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

// Define the structure for police data
struct infotype {
    string nopol;
    string warna;
    int tahun;
};

// Define the structure for list element
struct ElmList {
    infotype info;
    ElmList* next;
    ElmList* prev;
};

// Define the structure for list
struct List {
    ElmList* first;
    ElmList* last;
};

// Function prototypes
void createList(List& L);
ElmList* alokasi(infotype x);
void dealokasi(ElmList* p);
void insertLast(List& L, ElmList* p);
void printInfo(List L);
ElmList* findElm(List L, infotype x);
void deleteFirst(List& L, ElmList*& p);
void deleteLast(List& L, ElmList*& p);
void deleteAfter(ElmList* prec, ElmList*& p);
void deleteElement(List& L, string nopol);

int main() {
    List L;
    infotype x;

    // Initialize list
    createList(L);

    // Input data
    char more;
    do {
        cout << "Masukkan nomor polisi: ";
        cin >> x.nopol;
        cout << "Masukkan warna kendaraan: ";
        cin >> x.warna;
        cout << "Masukkan tahun kendaraan: ";
        cin >> x.tahun;

        ElmList* p = alokasi(x);
        insertLast(L, p);

        cout << "Input more data? (y/n): ";
        cin >> more;
    } while (more == 'y' || more == 'Y');

    // Print initial list
    cout << "\nDATA LIST 1\n";
    printInfo(L);

    // Search for element D001
    cout << "\nMencari nomor polisi D001:\n";
    x.nopol = "D001";
    ElmList* found = findElm(L, x);
    if (found != NULL) {
        cout << "Nomor Polisi : " << found->info.nopol << endl;
        cout << "Warna        : " << found->info.warna << endl;
        cout << "Tahun        : " << found->info.tahun << endl;
    } else {
        cout << "Data tidak ditemukan\n";
    }

    // Delete element D003
    cout << "\nMenghapus nomor polisi D003:\n";
    deleteElement(L, "D003");

    // Print final list
    cout << "\nDATA LIST 1 (setelah penghapusan)\n";
    printInfo(L);

    return 0;
}

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

ElmList* alokasi(infotype x) {
    ElmList* p = new ElmList;
    p->info = x;
    p->next = NULL;
    p->prev = NULL;
    return p;
}

void dealokasi(ElmList* p) {
    delete p;
}

void insertLast(List& L, ElmList* p) {
    if (L.first == NULL) {
        L.first = p;
        L.last = p;
    } else {
        p->prev = L.last;
        L.last->next = p;
        L.last = p;
    }
}

void printInfo(List L) {
    ElmList* p = L.first;
    while (p != NULL) {
        cout << "Nomor Polisi : " << p->info.nopol << endl;
        cout << "Warna        : " << p->info.warna << endl;
        cout << "Tahun        : " << p->info.tahun << endl;
        cout << endl;
        p = p->next;
    }
}

ElmList* findElm(List L, infotype x) {
    ElmList* p = L.first;
    while (p != NULL) {
        if (p->info.nopol == x.nopol) {
            return p;
        }
        p = p->next;
    }
    return NULL;
}

void deleteFirst(List& L, ElmList*& p) {
    p = L.first;
    if (L.first == L.last) {
        L.first = NULL;
        L.last = NULL;
    } else {
        L.first = p->next;
        L.first->prev = NULL;
        p->next = NULL;
    }
}

void deleteLast(List& L, ElmList*& p) {
    p = L.last;
    if (L.first == L.last) {
        L.first = NULL;
        L.last = NULL;
    } else {
        L.last = p->prev;
        L.last->next = NULL;
        p->prev = NULL;
    }
}

void deleteAfter(ElmList* prec, ElmList*& p) {
    p = prec->next;
    prec->next = p->next;
    if (p->next != NULL) {
        p->next->prev = prec;
    }
    p->next = NULL;
    p->prev = NULL;
}

void deleteElement(List& L, string nopol) {
    ElmList* p;
    if (L.first->info.nopol == nopol) {
        deleteFirst(L, p);
    } else if (L.last->info.nopol == nopol) {
        deleteLast(L, p);
    } else {
        ElmList* prec = L.first;
        while (prec->next != NULL && prec->next->info.nopol != nopol) {
            prec = prec->next;
        }
        if (prec->next != NULL) {
            deleteAfter(prec, p);
        }
    }
    if (p != NULL) {
        cout << "Data dengan nomor polisi " << nopol << " berhasil dihapus.\n";
        dealokasi(p);
    } else {
        cout << "Data tidak ditemukan\n";
    }
}
```

Outputnya:

Telkom University Purwokerto

```
Masukkan nomor polisi: D001
Masukkan warna kendaraan: hitam
Masukkan tahun kendaraan: 90
Input more data? (y/n): y
Masukkan nomor polisi: D003
Masukkan warna kendaraan: putih
Masukkan tahun kendaraan: 70
Input more data? (y/n): y
Masukkan nomor polisi: D001
Masukkan warna kendaraan: merah
Masukkan tahun kendaraan: 80
Input more data? (y/n): y
Masukkan nomor polisi: D004
Masukkan warna kendaraan: kuning
Masukkan tahun kendaraan: 90
Input more data? (y/n): n

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

Nomor Polisi : D003
Warna        : putih
Tahun        : 70

Nomor Polisi : D001
Warna        : merah
Tahun        : 80

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

```
Mencari nomor polisi D001:
Nomor Polisi : D001
Warna        : hitam
Tahun        : 90
```

```
Menghapus nomor polisi D003:
Data dengan nomor polisi D003 berhasil dihapus.

DATA LIST 1 (setelah penghapusan)
Nomor Polisi : D001
Warna        : hitam
Tahun        : 90

Nomor Polisi : D001
Warna        : merah
Tahun        : 80

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

5. **Kesimpulan**

Pada praktikum ini, dapat disimpulkan bahwa double linked list adalah struktur data yang memungkinkan navigasi dua arah, sehingga setiap elemen dapat diakses dari depan atau belakang. Praktikum ini juga mengajarkan penggunaan pointer dalam bahasa C untuk mengimplementasikan berbagai operasi dasar pada double linked list, seperti penambahan (Insert First, Insert Last, Insert After, Insert Before) dan penghapusan elemen (Delete First, Delete Last, Delete After, Delete Before).