LAPORAN PRAKTIKUM

PERTEMUAN 6

Nama : Razhendriya Vania Ramadhan Suganjarsarwat

Nama Mahasiswa (2311104048)

Dosen :

WAHYU ANDI SAPUTRA

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. TUJUAN

Tujuan dari praktikum ini adalah:

1. Memahami konsep dari linked list.
2. Mengaplikasikan konsep double linked list menggunakan pointer dan bahasa C++

II. DASAR TEORI

Double Linked List adalah tipe linked list yang memiliki dua penunjuk (pointer) di setiap elemennya: satu penunjuk ke elemen sebelumnya (prev) dan satu lagi ke elemen berikutnya (next). Struktur ini memungkinkan traversal baik ke depan maupun ke belakang, menjadikannya lebih fleksibel dibandingkan single linked list

III. 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;
}
```

Kode ini adalah implementasi dasar Double Linked List dengan fitur:

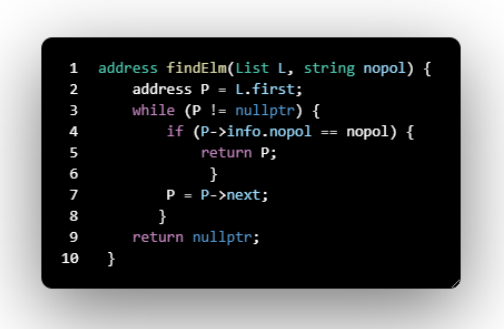
- Menambahkan elemen di depan list.
- Menghapus elemen dari depan list.
- Memperbarui data dalam node tertentu.
- Menghapus semua elemen dalam list.
- Menampilkan semua elemen.

IV. UNGUIDED

ADT Double Linked List di dalam file doublelist.h: Pada file doublelist.h, deklarasi tipe data dan fungsi-fungsi primitif untuk double linked list dapat dituliskan sebagai berikut:

```
#ifndef DOUBLELIST_H
#define DOUBLELIST_H

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

typedef struct kendaraan {
    string nopol;
    string warna;
    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) {
    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 printInfo(List L) {
    address P = L.first;
    while (P != nullptr) {
        cout << "Nomor Polisi: " << P->info.nopol << endl;
        cout << "Warna: " << P->info.warna << endl;
        cout << "Tahun Buat: " << P->info.thnBuat << endl;
        P = P->next;
    }
}

void insertLast(List &L, address P) {
    if (L.first == nullptr) {
        L.first = P;
        L.last = P;
    } else {
        P->prev = L.last;
        L.last->next = P;
        L.last = P;
    }
}

#endif
```

Fungsi findElm untuk mencari elemen dengan nomor polisi tertentu: Fungsi findElm mencari elemen berdasarkan nomor polisi dalam list dan mengembalikan alamat elemen jika ditemukan, atau nullptr jika tidak ditemukan.



```
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;
}
```

Prosedur untuk menghapus elemen dengan nomor polisi tertentu (deleteFirst, deleteLast, dan deleteAfter):

- **Prosedur deleteFirst:**

```
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;
       }
}
```

Prosedur deleteLast:

```
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;
       }
}
```

Prosedur deleteAfter:

```
1  void deleteAfter(address Prec, address &P) {
2      if (Prec != nullptr && Prec->next != nullptr) {
3          P = Prec->next;
4          Prec->next = P->next;
5          if (P->next != nullptr) {
6              P->next->prev = Prec;
7               }
8          P->next = nullptr;
9          P->prev = nullptr;
10        }
11   }
```

V. KESIMPULAN

Double linked list memungkinkan pengelolaan data dengan lebih mudah karena elemen dapat diakses dari dua arah. Operasi seperti insert, delete, dan search juga dapat dilakukan dengan lebih efisien dibandingkan single linked list, karena traversal bisa dilakukan maju dan mundur