Laporan Praktikum
Modul 6
DOUBLE LINKED LIST (BAGIAN PERTAMA)

Disusun Oleh:
Dwi Candra Pratama / 2211104035
SE06-02

Dosen Pengampu:
Wahyu Andi Saputra S.Kom. MSc

PROGRAM STUDI REKAYASA PERANGKAT LUNAK
FAKULTAS INFORMATIKA TELKOM UNIVERSITY
PURWOKERTO
2024

GUIDED

1. Tujuan Praktikum

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

A. Double Linked List

Double Linked list adalah linked list yang masing – masing elemen nya memiliki 2 successor, yaitu successor yang menunjuk pada elemen sebelumnya (prev) dan successor yang menunjuk pada elemen sesudahnya (next).

Komponen-komponen dalam double linked list:

1. First : pointer pada list yang menunjuk pada elemen pertama list.
2. Last : pointer pada list yang menunjuk pada elemen terakhir list.
3. Next : pointer pada elemen sebagai successor yang menunjuk pada elemen didepannya.
4. Prev :pointer pada elemen sebagai successor yang menunjuk pada elemen dibelakangnya.

Berikut adalah implementasinya yang mencakup INSERT(Insert First, Insert Last, Insert After, & Insert Before) & DELETE(Delete First, Delete Last, Delete After, Delete Before, Update, View, dan Searching)

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;
}
```

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

1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 2
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 3
Enter old data: 1
Enter new data: 5
Data not found
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice:
```

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”.

SourCodenya:

DoubleList.cpp

```
#include "doublelist.h"
#include <iostream>

void CreateList(List &L) {
   L.First = nullptr;
   L.Last = nullptr;
}

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

void dealokasi(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(const List &L) {
    ElmList* P = L.First;
    while (P != nullptr) {
        std::cout << "no polisi : " << P->info.nopol << std::endl;
        std::cout << "warna     : " << P->info.warna << std::endl;
        std::cout << "tahun     : " << P->info.thnBuat << std::endl;
        P = P->next;
    }
}

ElmList* findElm(const List &L, infotype x) {
    ElmList* P = L.First;
    while (P != nullptr) {
        if (P->info.nopol == x.nopol) {
            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;
    }
}
```

DoubleList.h

```
#ifndef DOUBLELIST_H
#define DOUBLELIST_H

#include <string>

struct infotype {
    std::string nopol;  // Nomor Polisi
    std::string warna;  // Warna Kendaraan
    int thnBuat;        // Tahun Pembuatan
};

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

struct List {
    ElmList* First;
    ElmList* Last;
};

// Deklarasi fungsi
void CreateList(List &L);
ElmList* alokasi(infotype x);
void dealokasi(ElmList* &P);
void printInfo(const List &L);
void insertLast(List &L, ElmList* P);
ElmList* findElm(const List &L, infotype x);
void deleteFirst(List &L, ElmList* &P);
void deleteLast(List &L, ElmList* &P);
void deleteAfter(ElmList* Prec, ElmList* &P);
```

```
#endif
```

Main.cpp

```
#include "doublelist.h"
#include "doublelist.cpp"
#include <iostream>

void inputKendaraan(infotype &kendaraan) {
   std::cout << "masukkan nomor polisi: ";
   std::cin >> kendaraan.nopol;
   std::cout << "masukkan warna kendaraan: ";
   std::cin >> kendaraan.warna;
   std::cout << "masukkan tahun kendaraan: ";
   std::cin >> kendaraan.thnBuat;
}

void tampilkanMenu() {
   std::cout << "\nMENU\n";
   std::cout << "1. Tambah Data Kendaraan\n";
   std::cout << "2. Cari Data Kendaraan\n";
   std::cout << "3. Hapus Data Kendaraan Pertama\n";
   std::cout << "4. Hapus Data Kendaraan Terakhir\n";
   std::cout << "5. Hapus Data Setelah Kendaraan Tertentu\n";
   std::cout << "6. Tampilkan Semua Data Kendaraan\n";
   std::cout << "0. Keluar\n";
   std::cout << "Pilih menu: ";
}

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

   int pilihan;
   do {
      tampilkanMenu();
      std::cin >> pilihan;
      std::cin.ignore();  // Mengabaikan newline setelah pilihan

      switch (pilihan) {
         case 1: {
            // Tambah data kendaraan
            infotype kendaraan;
            inputKendaraan(kendaraan);

            // Cek apakah nomor polisi sudah ada
            if (findElm(L, kendaraan) != nullptr) {
               std::cout << "nomor polisi sudah terdaftar\n";
            } else {
               insertLast(L, alokasi(kendaraan));
```

```
            std::cout << "Data kendaraan berhasil ditambahkan.\n";
        }
        break;
    }
    case 2: {
        // Cari data kendaraan
        infotype cari;
        std::cout << "Masukkan Nomor Polisi yang dicari : ";
        std::cin >> cari.nopol;

        ElmList* found = findElm(L, cari);
        if (found != nullptr) {
            std::cout << "Nomor Polisi: " << found->info.nopol << std::endl;
            std::cout << "Warna       : " << found->info.warna << std::endl;
            std::cout << "Tahun       : " << found->info.thnBuat << std::endl;
        } else {
            std::cout << "nomor polisi tidak ditemukan.\n";
        }
        break;
    }
    case 3: {
        // Hapus data kendaraan pertama
        ElmList* deleted;
        deleteFirst(L, deleted);
        if (deleted != nullptr) {
            std::cout << "Data kendaraan pertama berhasil dihapus.\n";
            dealokasi(deleted);
        } else {
            std::cout << "List kosong, tidak ada data untuk dihapus.\n";
        }
        break;
    }
    case 4: {
        // Hapus data kendaraan terakhir
        ElmList* deleted;
        deleteLast(L, deleted);
        if (deleted != nullptr) {
            std::cout << "Data kendaraan terakhir berhasil dihapus.\n";
            dealokasi(deleted);
        } else {
            std::cout << "List kosong, tidak ada data untuk dihapus.\n";
        }
        break;
    }
    case 5: {
        // Hapus data setelah kendaraan tertentu
        infotype cari;
         std::cout << "Masukkan nomor polisi kendaraan sebelum data yang ingin
dihapus: ";
        std::cin >> cari.nopol;
```

```
            ElmList* prec = findElm(L, cari);
            if (prec != nullptr && prec->next != nullptr) {
                ElmList* deleted;
                deleteAfter(prec, deleted);
                if (deleted != nullptr) {
                    std::cout << "Data dengan nomor polisi " << deleted->info.nopol <<
" berhasil dihapus.\n";
                    dealokasi(deleted);
                }
            } else {
                    std::cout << "Tidak ada data untuk dihapus setelah nomor polisi
tersebut.\n";
            }
            break;
        }
        case 6: {
            // Tampilkan semua data kendaraan
            std::cout << "DATA LIST KENDARAAN\n";
            printInfo(L);
            break;
        }
        case 0: {
            std::cout << "Keluar dari program.\n";
            break;
        }
        default:
            std::cout << "Pilihan tidak valid. Silakan coba lagi.\n";
            break;
      }
   } while (pilihan != 0);

   return 0;
}
```

```
MENU
1. Tambah Data Kendaraan
2. Cari Data Kendaraan
3. Hapus Data Kendaraan Pertama
4. Hapus Data Kendaraan Terakhir
5. Hapus Data Setelah Kendaraan Tertentu
6. Tampilkan Semua Data Kendaraan
0. Keluar
Pilih menu: 6
DATA LIST KENDARAAN
no polisi : G001
warna     : Merah
tahun     : 2009
no polisi : G002
warna     : Biru
tahun     : 2016
```

2. Carilah elemen dengan nomor polisi D001 dengan membuat fungsi baru.
   fungsi findElm(L : List, x : infotype) : address

```
MENU
1. Tambah Data Kendaraan
2. Cari Data Kendaraan
3. Hapus Data Kendaraan Pertama
4. Hapus Data Kendaraan Terakhir
5. Hapus Data Setelah Kendaraan Tertentu
6. Tampilkan Semua Data Kendaraan
0. Keluar
Pilih menu: 2
Masukkan Nomor Polisi yang dicari : G002
Nomor Polisi: G002
Warna       : Biru
Tahun       : 2016
```

3. apus 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)

```
1. Tambah Data Kendaraan
2. Cari Data Kendaraan
3. Hapus Data Kendaraan Pertama
4. Hapus Data Kendaraan Terakhir
5. Hapus Data Setelah Kendaraan Tertentu
6. Tampilkan Semua Data Kendaraan
0. Keluar
Pilih menu: 4
Data kendaraan terakhir berhasil dihapus.

MENU
1. Tambah Data Kendaraan
2. Cari Data Kendaraan
3. Hapus Data Kendaraan Pertama
4. Hapus Data Kendaraan Terakhir
5. Hapus Data Setelah Kendaraan Tertentu
6. Tampilkan Semua Data Kendaraan
0. Keluar
Pilih menu: 6
DATA LIST KENDARAAN
no polisi : G001
warna     : Merah
tahun     : 2009
```