LAPORAN PRAKTIKUM
Modul 06
"DOUBLE LINKED LIST (BAGIAN PERTAMA)"

Disusun Oleh:
Ganesha Rahman Gibran -2211104058
Kelas S1SE-07-02

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024

Tujuan

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

Landasan Teori

Double Linked List

Double Linked List (DLL) adalah struktur data yang terdiri dari elemen-elemen yang terhubung melalui dua pointer: prev yang menunjuk ke elemen sebelumnya dan next yang menunjuk ke elemen berikutnya. Hal ini memungkinkan traversal baik dari awal ke akhir maupun dari akhir ke awal, memberikan fleksibilitas lebih dibandingkan dengan Single Linked List yang hanya memiliki pointer ke depan. Pada DLL, setiap node memiliki struktur yang mencakup data serta dua pointer next dan prev.

Operasi Dasar dalam Double Linked List

1. **CreateList**: Inisialisasi list kosong dengan pointer head dan tail menunjuk ke nullptr.
2. **Insert**: Menambahkan node baru pada posisi tertentu:
    - InsertFirst: Menambahkan node di awal list.
    - InsertLast: Menambahkan node di akhir list.
    - InsertAfter: Menambahkan node setelah node tertentu.
3. **Delete**: Menghapus node dari posisi tertentu:
    - DeleteFirst: Menghapus node di awal list.
    - DeleteLast: Menghapus node di akhir list.
    - DeleteNode: Menghapus node tertentu berdasarkan nilai atau posisi.
4. **Search**: Mencari node dengan nilai tertentu.

Implementasi Double Linked List

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

struct Node {
    int data;
    Node* next;
    Node* prev;
    Node(int val) : data(val), next(nullptr), prev(nullptr) {} // Konstruktor
sederhana
};

class DoubleLinkedList {
private:
    Node* head;
    Node* tail;

public:
    DoubleLinkedList() : head(nullptr), tail(nullptr) {}

    // CreateList: menginisialisasi list kosong
    bool isEmpty() { return head == nullptr; }
```

```
// InsertFirst: menambahkan node di awal
void insertFirst(int val) {
    Node* newNode = new Node(val);
    if (isEmpty()) {
        head = tail = newNode;
    } else {
        newNode->next = head;
        head->prev = newNode;
        head = newNode;
    }
}

// InsertLast: menambahkan node di akhir
void insertLast(int val) {
    Node* newNode = new Node(val);
    if (isEmpty()) {
        head = tail = newNode;
    } else {
        tail->next = newNode;
        newNode->prev = tail;
        tail = newNode;
    }
}

// DeleteFirst: menghapus node di awal
void deleteFirst() {
    if (!isEmpty()) {
        Node* temp = head;
        if (head == tail) { // hanya ada satu elemen
            head = tail = nullptr;
        } else {
            head = head->next;
            head->prev = nullptr;
        }
        delete temp;
    }
}

// DeleteLast: menghapus node di akhir
void deleteLast() {
    if (!isEmpty()) {
        Node* temp = tail;
        if (head == tail) { // hanya ada satu elemen
            head = tail = nullptr;
        } else {
            tail = tail->prev;
            tail->next = nullptr;
        }
        delete temp;
    }
}

// Menampilkan list dari awal
void displayForward() {
```

```
        Node* current = head;
        while (current != nullptr) {
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }

    // Menampilkan list dari akhir
    void displayBackward() {
        Node* current = tail;
        while (current != nullptr) {
            cout << current->data << " ";
            current = current->prev;
        }
        cout << endl;
    }
};

int main() {
    DoubleLinkedList dll;
    dll.insertFirst(10);
    dll.insertLast(20);
    dll.insertFirst(5);

    cout << "List forward: ";
    dll.displayForward();

    cout << "List backward: ";
    dll.displayBackward();

    dll.deleteFirst();
    cout << "After deleting first element: ";
    dll.displayForward();

    dll.deleteLast();
    cout << "After deleting last element: ";
    dll.displayForward();

    return 0;
}
```

Penjelasan Kode

- **Node**: Struktur Node menyimpan data data serta pointer next dan prev untuk menunjuk ke node berikutnya dan sebelumnya.
- **DoubleLinkedList**: Kelas ini mengelola operasi-operasi Double Linked List.
    1. **InsertFirst dan InsertLast**: Metode ini menambahkan node di awal atau akhir list.
    2. **DeleteFirst dan DeleteLast**: Metode ini menghapus node dari awal atau akhir list.
    3. **displayForward dan displayBackward**: Metode ini menampilkan isi list dari awal ke akhir atau sebaliknya.

Telkom University Purwokerto

Guided

Input :

```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
    }
```

Telkom University Purwokerto

```
    // 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;
    }
};
```

Telkom University Purwokerto

```cpp
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 :

Telkom University Purwokerto

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

Unguided

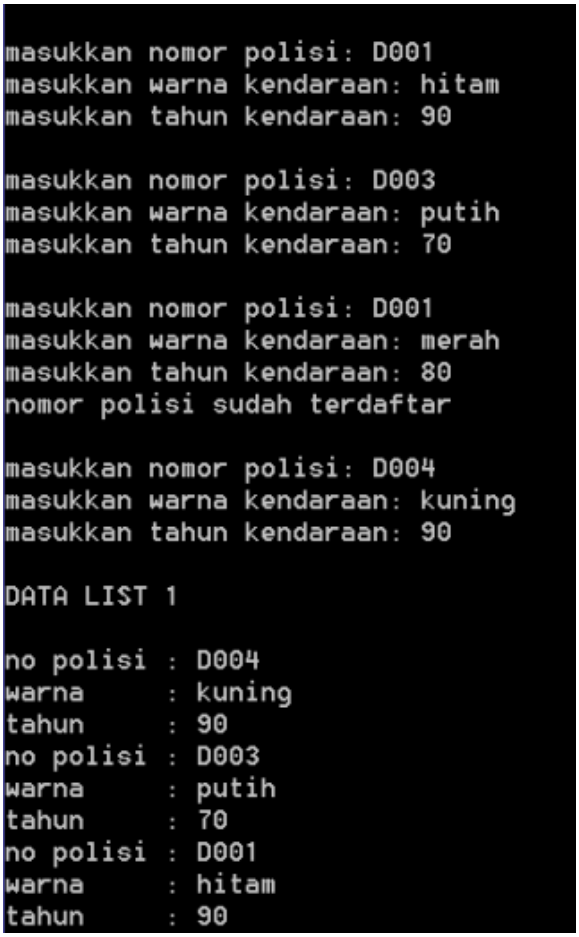
1. Buatlah ADT Double Linked list sebagai berikut di dalam file “doublelist.h”:

```
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”.

Contoh Output :



Input :

doublelist.h

```
#ifndef DOUBLELIST_H
#define DOUBLELIST_H

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

struct kendaraan {
```

```
    string nopol;
    string warna;
    int thnBuat;
};

typedef kendaraan 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);

#endif
```

doublelist.cpp

```
#include "doublelist.h"

// Initialize an empty list
void CreateList(List &L) {
    L.First = nullptr;
    L.Last = nullptr;
}

// Allocate memory for a new element
address alokasi(infotype x) {
    address P = new ElmList;
    P->info = x;
    P->next = nullptr;
    P->prev = nullptr;
    return P;
}

// Deallocate memory
void dealokasi(address &P) {
    delete P;
    P = nullptr;
}

// Insert element at the end of the list
void insertLast(List &L, address P) {
    if (L.First == nullptr) { // List is empty
        L.First = P;
```

```
            L.Last = P;
        } else {
            P->prev = L.Last;
            L.Last->next = P;
            L.Last = P;
        }
    }

    // Print all elements in the list
    void printInfo(List L) {
        address P = L.First;
        int i = 1;
        while (P != nullptr) {
            cout << "DATA LIST " << i << endl;
            cout << "no polisi : " << P->info.nopol << endl;
            cout << "warna     : " << P->info.warna << endl;
            cout << "tahun     : " << P->info.thnBuat << endl;
            cout << endl;
            P = P->next;
            i++;
        }
    }
```

main.cpp

```
#include "doublelist.h"

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

    // Sample data to insert
    infotype data1 = {"D001", "hitam", 90};
    infotype data2 = {"D003", "putih", 70};
    infotype data3 = {"D004", "kuning", 90};

    // Insert data into list
    insertLast(L, alokasi(data1));
    insertLast(L, alokasi(data2));
    insertLast(L, alokasi(data3));

    // Print the list contents
    printInfo(L);

    return 0;
}
```

Output :

```
masukkan nomor polisi: R44RR
masukkan warna kendaraan: Pelangi
masukkan tahun kendaraan: 2019

DATA LIST
no polisi : R44RR
warna     : Pelangi
tahun     : 2019

Lanjut (y/n)? y
masukkan nomor polisi: R14RQ
masukkan warna kendaraan: RGB
masukkan tahun kendaraan: 2020

DATA LIST
no polisi : R44RR
warna     : Pelangi
tahun     : 2019
no polisi : R14RQ
warna     : RGB
tahun     : 2020

Lanjut (y/n)?
y
masukkan nomor polisi: A12BC
masukkan warna kendaraan: Coquette
masukkan tahun kendaraan: 2024

DATA LIST
no polisi : R44RR
warna     : Pelangi
tahun     : 2019
no polisi : R14RQ
warna     : RGB
tahun     : 2020
no polisi : A12BC
warna     : Coquette
tahun     : 2024

Lanjut (y/n)?
```

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

```
Masukkan Nomor Polisi yang dicari   : D001

Nomor Polisi    : D001
Warna           : hitam
Tahun           : 90
```

doublelist.h

```
#ifndef DOUBLELIST_H
#define DOUBLELIST_H

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

struct kendaraan {
    string nopol;
    string warna;
```

```
    int thnBuat;
};

typedef kendaraan 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, string nopol); // New function prototype

#endif
```

doublelist.cpp

```
#include "doublelist.h"

// Initialize an empty list
void CreateList(List &L) {
    L.First = nullptr;
    L.Last = nullptr;
}

// Allocate memory for a new element
address alokasi(infotype x) {
    address P = new ElmList;
    P->info = x;
    P->next = nullptr;
    P->prev = nullptr;
    return P;
}

// Deallocate memory
void dealokasi(address &P) {
    delete P;
    P = nullptr;
}

// Insert element at the end of the list
void insertLast(List &L, address P) {
    if (L.First == nullptr) { // List is empty
        L.First = P;
        L.Last = P;
```

```
    } else {
        P->prev = L.Last;
        L.Last->next = P;
        L.Last = P;
    }
}

// Print all elements in the list
void printInfo(List L) {
    address P = L.First;
    int i = 1;
    while (P != nullptr) {
        cout << "DATA LIST " << i << endl;
        cout << "no polisi : " << P->info.nopol << endl;
        cout << "warna     : " << P->info.warna << endl;
        cout << "tahun     : " << P->info.thnBuat << endl;
        cout << endl;
        P = P->next;
        i++;
    }
}

// Find an element by nomor polisi
address findElm(List L, string nopol) {
    address P = L.First;
    while (P != nullptr) {
        if (P->info.nopol == nopol) {
            return P; // Element found
        }
        P = P->next;
    }
    return nullptr; // Element not found
}
```

main.cpp

```
#include "doublelist.h"

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

    // Sample data to insert
    infotype data1 = {"D001", "hitam", 90};
    infotype data2 = {"D003", "putih", 70};
    infotype data3 = {"D004", "kuning", 90};

    // Insert data into list
    insertLast(L, alokasi(data1));
    insertLast(L, alokasi(data2));
    insertLast(L, alokasi(data3));

    // Search for an element
    string targetNopol = "D001";
    address found = findElm(L, targetNopol);
```

```
    if (found != nullptr) {
        cout << "Nomor Polisi : " << found->info.nopol << endl;
        cout << "Warna        : " << found->info.warna << endl;
        cout << "Tahun        : " << found->info.thnBuat << endl;
    } else {
        cout << "Nomor Polisi " << targetNopol << " tidak ditemukan." <<
    endl;
    }

    return 0;
}
```

Output :

```
Lanjut (y/n)? y
masukkan nomor polisi: A12BC
masukkan warna kendaraan: Coquette
masukkan tahun kendaraan: 2024

DATA LIST
no polisi : R44RR
warna     : Pelangi
tahun     : 2019
no polisi : R14RQ
warna     : RGB
tahun     : 2020
no polisi : A12BC
warna     : Coquette
tahun     : 2024

Lanjut (y/n)? n

Masukkan nomor polisi yang dicari: A12BC

Data Kendaraan ditemukan:
no polisi : A12BC
warna     : Coquette
tahun     : 2024

Process returned 0 (0x0)   execution time
Press any key to continue.
```

3. Hapus 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)

```
Masukkan Nomor Polisi yang akan dihapus   : D003
Data dengan nomor polisi D003 berhasil dihapus.

DATA LIST 1

Nomor Polisi    : D004
Warna           : kuning
Tahun           : 90
Nomor Polisi    : D001
Warna           : hitam
Tahun           : 90
```

Input :

doublelist.h

```
  #ifndef DOUBLELIST_H
```

```
#define DOUBLELIST_H

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

struct kendaraan {
    string nopol;
    string warna;
    int thnBuat;
};

typedef kendaraan 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, string nopol);

// Delete procedures
void deleteFirst(List &L, address &P);
void deleteLast(List &L, address &P);
void deleteAfter(address Prec, address &P);

#endif
```

doublelist.cpp

```
#include "doublelist.h"

// Initialize an empty list
void CreateList(List &L) {
    L.First = nullptr;
    L.Last = nullptr;
}

// Allocate memory for a new element
address alokasi(infotype x) {
    address P = new ElmList;
    P->info = x;
    P->next = nullptr;
    P->prev = nullptr;
```

```
    return P;
}

// Deallocate memory
void dealokasi(address &P) {
    delete P;
    P = nullptr;
}

// Insert element at the end of the list
void insertLast(List &L, address P) {
    if (L.First == nullptr) { // List is empty
        L.First = P;
        L.Last = P;
    } else {
        P->prev = L.Last;
        L.Last->next = P;
        L.Last = P;
    }
}

// Print all elements in the list
void printInfo(List L) {
    address P = L.First;
    int i = 1;
    while (P != nullptr) {
        cout << "DATA LIST " << i << endl;
        cout << "no polisi : " << P->info.nopol << endl;
        cout << "warna     : " << P->info.warna << endl;
        cout << "tahun     : " << P->info.thnBuat << endl;
        cout << endl;
        P = P->next;
        i++;
    }
}

// Find an element by nomor polisi
address findElm(List L, string nopol) {
    address P = L.First;
    while (P != nullptr) {
        if (P->info.nopol == nopol) {
            return P; // Element found
        }
        P = P->next;
    }
    return nullptr; // Element not found
}

// Delete the first element in the list
void deleteFirst(List &L, address &P) {
    if (L.First != nullptr) { // List is not empty
        P = L.First;
        if (L.First == L.Last) { // Only one element
            L.First = nullptr;
            L.Last = nullptr;
```

```
        } else {
            L.First = L.First->next;
            L.First->prev = nullptr;
        }
        P->next = nullptr;
    }
}

// Delete the last element in the list
void deleteLast(List &L, address &P) {
    if (L.Last != nullptr) { // List is not empty
        P = L.Last;
        if (L.First == L.Last) { // Only one element
            L.First = nullptr;
            L.Last = nullptr;
        } else {
            L.Last = L.Last->prev;
            L.Last->next = nullptr;
        }
        P->prev = nullptr;
    }
}

// Delete the element after a specified node
void deleteAfter(address Prec, address &P) {
    if (Prec != nullptr && Prec->next != nullptr) { // Prec and Prec->next
    are not null
        P = Prec->next;
        Prec->next = P->next;
        if (P->next != nullptr) {
            P->next->prev = Prec;
        } else {
            // If P is the last element
            Prec->next = nullptr;
        }
        P->prev = nullptr;
        P->next = nullptr;
    }
}
```

main.cpp

```
#include "doublelist.h"

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

    // Sample data to insert
    infotype data1 = {"D001", "hitam", 90};
    infotype data2 = {"D003", "putih", 70};
    infotype data3 = {"D004", "kuning", 90};

    // Insert data into list
    insertLast(L, alokasi(data1));
    insertLast(L, alokasi(data2));
```

```
    insertLast(L, alokasi(data3));

    // Display list before deletion
    cout << "List before deletion:" << endl;
    printInfo(L);

    // Search and delete element with nomor polisi "D003"
    string targetNopol = "D003";
    address P = findElm(L, targetNopol);

    if (P != nullptr) {
        if (P == L.First) {
            deleteFirst(L, P);
        } else if (P == L.Last) {
            deleteLast(L, P);
        } else {
            deleteAfter(P->prev, P);
        }
        dealokasi(P);
        cout << "Data dengan nomor polisi " << targetNopol << " berhasil
    dihapus." << endl;
    } else {
        cout << "Nomor polisi " << targetNopol << " tidak ditemukan." <<
    endl;
    }

    // Display list after deletion
    cout << "List after deletion:" << endl;
    printInfo(L);

    return 0;
}
```

Output :

```
Masukkan nomor polisi yang akan dihapus: R44RR
Data berhasil dihapus

Data setelah penghapusan:

DATA LIST
no polisi : R14RQ
warna     : RGB
tahun     : 2020
no polisi : A12BC
warna     : Coquette
tahun     : 2024

Menu:
1. Cari data kendaraan
2. Hapus data kendaraan
3. Tampilkan semua data
4. Keluar
Pilihan: 4
```

Kesimpulan

Double Linked List adalah struktur data yang fleksibel dengan kemampuan traversal dua arah melalui dua pointer, prev dan next, sehingga memungkinkan penambahan dan

penghapusan elemen di awal, akhir, maupun di antara elemen lain dengan lebih mudah dibandingkan Single Linked List. Struktur ini efektif untuk aplikasi yang membutuhkan navigasi bolak-balik, seperti pada fitur undo-redo atau browser history. Implementasinya melibatkan penggunaan node dengan pointer next dan prev yang saling terhubung, serta operasi dasar seperti Insert dan Delete yang memungkinkan manipulasi data secara efisien di berbagai posisi dalam list.