LAPORAN

MODUL 6

DOUBLE LINKED LIST

Disusun Oleh :

NAMA :

Devi Nurliana

NIM :

103112400144

Dosen :

FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

A. DASAR TEORI

Double linked list merupakan salah satu struktur data linier yang terdiri dari sekumpulan elemen yang saling terhubung menggunakan pointer. Berbeda dengan *single linked list*, setiap node pada *double linked list* memiliki dua pointer, yaitu pointer ke node sebelumnya (*prev*) dan pointer ke node berikutnya (*next*). Selain pointer, setiap node juga menyimpan data atau informasi tertentu. Dengan adanya dua arah hubungan ini, proses penelusuran data dapat dilakukan dari depan ke belakang maupun dari belakang ke depan, sehingga lebih fleksibel dalam pengolahan data.

Pada implementasinya, *double linked list* biasanya memiliki node awal (*head*) dan node akhir (*tail*) untuk memudahkan pengelolaan struktur data. Operasi dasar yang dapat dilakukan pada *double linked list* meliputi penyisipan data di awal, di akhir, atau di tengah list, penghapusan data, serta penelusuran isi list. Walaupun penggunaan dua pointer pada setiap node membutuhkan memori lebih besar dibandingkan *single linked list*, keunggulan *double linked list* terletak pada kemudahan manipulasi data, terutama saat melakukan penghapusan atau penelusuran dua arah.

B. GUIDED

a. Source Code

- dll.cpp

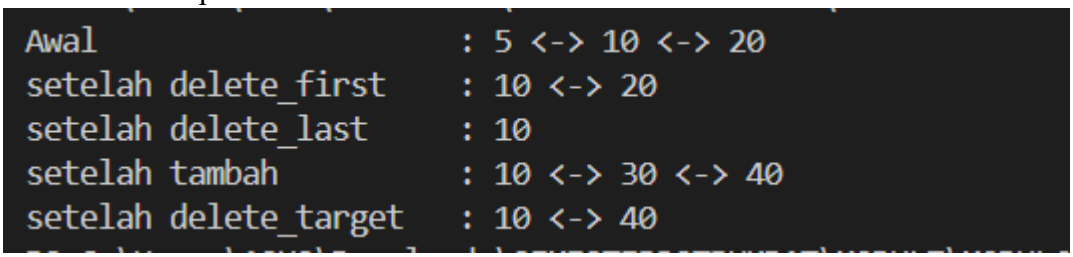
```
#include <iostream>
using namespace std;
struct Node {
int data;
Node *prev;
Node *next;
Node(int val, Node *p = nullptr, Node *n = nullptr)
: data(val), prev(p), next(n) {}
};
Node *ptr_first = nullptr;
Node *ptr_last  = nullptr;
void delete_last();
void add_first(int Value) {
Node *newNode = new Node(Value, nullptr, ptr_first);
if (ptr_first == nullptr) {
ptr_last = newNode;
} else {
ptr_first->prev = newNode;
   }
   ptr_first = newNode;
}

void add_last(int Value) {
   Node *newNode = new Node(Value, ptr_last, nullptr);

   if (ptr_last == nullptr) {
      ptr_first = newNode;
   } else {
```

```
        ptr_last->next = newNode;
    }
    ptr_last = newNode;
}

void add_target(int targetValue, int newValue) {
    Node *current = ptr_first;

    while (current != nullptr && current->data != targetValue)
{
        current = current->next;
    }

    if (current != nullptr) {
        if (current == ptr_last) {
            add_last(newValue);
        } else {
            Node *newNode = new Node(newValue, current,
current->next);
            current->next->prev = newNode;
            current->next = newNode;
        }
    }
}

void view() {
    Node *current = ptr_first;

    if (current == nullptr) {
        cout << "list kosong\n";
        return;
    }

    while (current != nullptr) {
        cout << current->data;
        if (current->next != nullptr) {
            cout << " <-> ";
        }
        current = current->next;
    }
    cout << endl;
}

void delete_first() {
    if (ptr_first == nullptr) return;

    Node *temp = ptr_first;

    if (ptr_first == ptr_last) {
        ptr_first = nullptr;
```

```
            ptr_last = nullptr;
        } else {
            ptr_first = ptr_first->next;
            ptr_first->prev = nullptr;
        }
        delete temp;
    }

    void delete_last() {
        if (ptr_last == nullptr) return;

        Node *temp = ptr_last;

        if (ptr_first == ptr_last) {
            ptr_first = nullptr;
            ptr_last = nullptr;
        } else {
            ptr_last = ptr_last->prev;
            ptr_last->next = nullptr;
        }
        delete temp;
    }

    void delete_target(int targetValue) {
        Node *current = ptr_first;

        while (current != nullptr && current->data != targetValue)
    {
            current = current->next;
        }

        if (current != nullptr) {
            if (current == ptr_first) {
                delete_first();
            } else if (current == ptr_last) {
                delete_last();
            } else {
                current->prev->next = current->next;
                current->next->prev = current->prev;
                delete current;
            }
        }
    }

    void edit_node(int targetValue, int newValue) {
        Node *current = ptr_first;

        while (current != nullptr && current->data != targetValue)
    {
            current = current->next;
```

```
        }

        if (current != nullptr) {
            current->data = newValue;
        }
    }

    int main() {
        add_first(10);
        add_first(5);
        add_last(20);

        cout << "Awal\t\t\t: ";
        view();

        delete_first();
        cout << "setelah delete_first\t: ";
        view();

        delete_last();
        cout << "setelah delete_last\t: ";
        view();

        add_last(30);
        add_last(40);
        cout << "setelah tambah\t\t: ";
        view();

        delete_target(30);
        cout << "setelah delete_target\t: ";
        view();

        return 0;
}
```

b. Screenshot Output



c. Deskripsi

Program ini dibuat untuk mengimplementasikan struktur data *doubly linked list* menggunakan bahasa *C++*. Setiap node pada list memiliki dua pointer, yaitu *prev* dan *next*, sehingga memungkinkan penelusuran data dilakukan dari arah depan maupun

belakang. Program ini mendukung operasi dasar pengelolaan data seperti penambahan, penghapusan, pengubahan, dan penampilan isi list secara berurutan.

1. Pendefinisian struktur data
   Program mendefinisikan struktur *Node* yang berisi data bertipe *integer* serta dua pointer *prev* dan *next*. Selain itu, digunakan pointer global *ptr_first* dan *ptr_last* untuk menandai awal dan akhir list.

2. Proses penambahan data
   Penambahan node dilakukan melalui prosedur *add_first*, *add_last*, dan *add_target* untuk menyisipkan data di awal, akhir, atau setelah node tertentu pada *doubly linked list*.

3. Proses penghapusan dan pengeditan data
   Program menyediakan prosedur *delete_first*, *delete_last*, dan *delete_target* untuk menghapus node, serta *edit_node* untuk mengubah nilai data pada node tertentu.

4. Penampilan isi list
   Prosedur *view* digunakan untuk menampilkan seluruh data dalam list dari awal hingga akhir dengan format penunjuk dua arah, sehingga struktur *doubly linked list* dapat terlihat dengan jelas.

C. UNGUIDED

1. Unguided 1
   a. Source Code

- doublylist.h

```
#ifndef DOUBLYLIST_H
#define DOUBLYLIST_H

#include <iostream>
using namespace std;

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

typedef struct ElmList *address;

typedef struct ElmList {
   infotype info;
   address next;
   address prev;
} ElmList;

typedef struct {
   address First;
```

```
    address Last;
} List;

void CreateList(List &L);
address alokasi(infotype x);
void dealokasi(address &P);

void insertLast(List &L, address P);
void printInfo(List L);

address findElm(List L, infotype x);

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

#endif
```

- doublylist.cpp

```
#include "Doublylist.h"

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

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

void dealokasi(address &P) {
    delete P;
    P = NULL;
}

void insertLast(List &L, address 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) {
   address P = L.Last;
   while (P != NULL) {
      cout << "Nomor Polisi : " << P->info.nopol << endl;
      cout << "Warna       : " << P->info.warna << endl;
      cout << "Tahun       : " << P->info.thnBuat << endl;
      P = P->prev;
   }
}

address findElm(List L, infotype x) {
   address P = L.First;
   while (P != NULL) {
      if (P->info.nopol == x.nopol) {
         return P;
      }
      P = P->next;
   }
   return NULL;
}

void deleteFirst(List &L, address &P) {
   P = L.First;
   if (P != NULL) {
      if (L.First == L.Last) {
         L.First = NULL;
         L.Last = NULL;
      } else {
         L.First = P->next;
         L.First->prev = NULL;
      }
      P->next = NULL;
      dealokasi(P);
   }
}

void deleteLast(List &L, address &P) {
   P = L.Last;
   if (P != NULL) {
      if (L.First == L.Last) {
         L.First = NULL;
         L.Last = NULL;
      } else {
         L.Last = P->prev;
         L.Last->next = NULL;
      }
      P->prev = NULL;
      dealokasi(P);
   }
}
```

```
        void deleteAfter(address Prec, address &P) {
            if (Prec != NULL && Prec->next != NULL) {
                P = Prec->next;
                Prec->next = P->next;
                if (P->next != NULL) {
                    P->next->prev = Prec;
                }
                P->next = NULL;
                P->prev = NULL;
                dealokasi(P);
            }
}
```

- main.cpp

```
        #include "doublylist.h"
        #include "doublylist.cpp"

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

            infotype x;
            address P;
            int n = 4;

            for (int i = 1; i <= n; i++) {
                cout << "masukkan nomor polisi: ";
                cin >> x.nopol;
                cout << "masukkan warna kendaraan: ";
                cin >> x.warna;
                cout << "masukkan tahun kendaraan: ";
                cin >> x.thnBuat;

                bool ada = false;
                address Q = L.First;
                while (Q != NULL) {
                    if (Q->info.nopol == x.nopol) {
                        ada = true;
                        break;
                    }
                    Q = Q->next;
                }

                if (ada) {
                    cout << "nomor polisi sudah terdaftar\n\n";
                } else {
                    insertLast(L, alokasi(x));
                    cout << endl;
                }
```

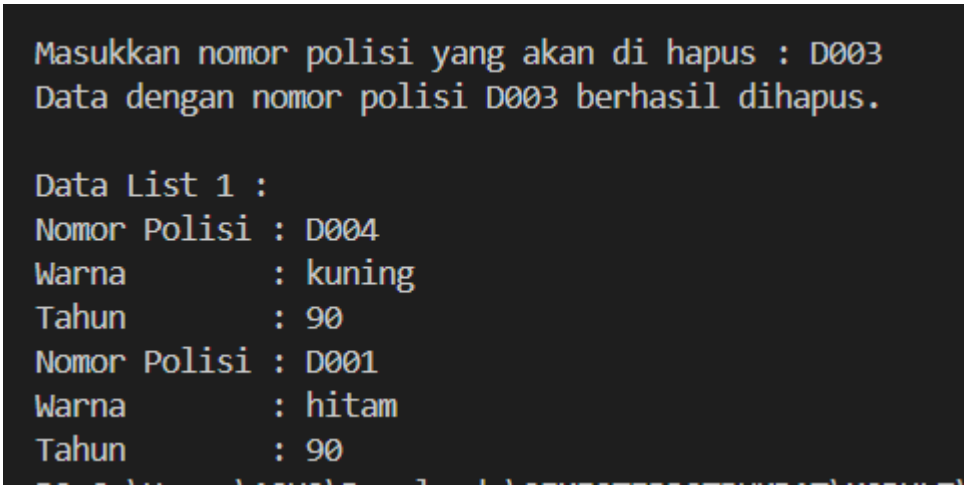
```
}

cout << "DATA LIST 1\n";
printInfo(L);

/* ====== CARI DATA ====== */
infotype cari;
address hasil;

cout << "\nMasukkan Nomor Polisi yang dicari : ";
cin >> cari.nopol;

hasil = findElm(L, cari);

if (hasil != NULL) {
   cout << "\nNomor Polisi : " << hasil->info.nopol << endl;
   cout << "Warna       : " << hasil->info.warna << endl;
   cout << "Tahun       : " << hasil->info.thnBuat << endl;
} else {
   cout << "\nNomor polisi tidak ditemukan" << endl;
}

/* ====== HAPUS DATA ====== */
infotype hapus;
address target, prec;

cout << "\nMasukkan nomor polisi yang akan di hapus : ";
cin >> hapus.nopol;

target = findElm(L, hapus);

if (target != NULL) {
   if (target == L.First) {
      deleteFirst(L, P);
   } else if (target == L.Last) {
      deleteLast(L, P);
   } else {
      prec = target->prev;
      deleteAfter(prec, P);
   }

   cout << "Data dengan nomor polisi " << hapus.nopol
        << " berhasil dihapus.\n";
} else {
   cout << "Data tidak ditemukan\n";
}

cout << "\nData List 1 :\n";
printInfo(L);
```

```
        return 0;
}
```

b. Screenshot Output

```
masukkan nomor polisi: D001
masukkan warna kendaraan: hitam
masukkan tahun kendaraan: 90

masukkan nomor polisi: D003
masukkan warna kendaraan: putih
masukkan tahun kendaraan: 70

masukkan nomor polisi: D001
masukkan warna kendaraan: merah
masukkan tahun kendaraan: 80
nomor polisi sudah terdaftar

masukkan nomor polisi: D004
masukkan warna kendaraan: kuning
masukkan tahun kendaraan: 90

DATA LIST 1
Nomor Polisi : D004
Warna        : kuning
Tahun        : 90
Nomor Polisi : D003
Warna        : putih
Tahun        : 70
Nomor Polisi : D001
Warna        : hitam
Tahun        : 90
```

```
Masukkan Nomor Polisi yang dicari : D001

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



c. Deskripsi
Program ini merupakan implementasi struktur data *doubly linked list* menggunakan bahasa pemrograman *C++* untuk mengelola data kendaraan.

1. Setiap data kendaraan disimpan dalam sebuah node yang berisi *nomor polisi*, *warna*, dan *tahun pembuatan*, serta memiliki dua pointer yaitu *next* dan *prev* untuk menunjuk elemen sesudah dan sebelumnya.

2. Program menyediakan proses penambahan data menggunakan prosedur *insertLast* setelah list dibuat kosong dengan *CreateList*, serta menampilkan seluruh data menggunakan *printInfo*.

3. Pencarian data kendaraan dilakukan berdasarkan *nomor polisi* menggunakan fungsi *findElm*, sehingga data dapat ditemukan secara efisien.

4. Program juga mendukung penghapusan data kendaraan menggunakan prosedur *deleteFirst*, *deleteLast*, dan *deleteAfter* sesuai dengan posisi elemen dalam list.

D. KESIMPULAN

Struktur data *doubly linked list* merupakan struktur data linier yang efektif untuk pengelolaan data karena setiap node memiliki dua pointer, yaitu *prev* dan *next*, sehingga memungkinkan penelusuran data secara dua arah. Implementasi *doubly linked list* pada program ini berhasil mendukung operasi dasar seperti penambahan, pencarian, penghapusan, dan penampilan data kendaraan dengan baik. Meskipun membutuhkan penggunaan memori yang lebih besar dibandingkan *single linked list*, struktur ini memberikan kemudahan dalam manipulasi data, terutama saat melakukan pencarian dan penghapusan elemen berdasarkan posisi tertentu.

E. REFERENSI

- Kurniawan, A. (2019). *Struktur Data dan Algoritma dengan C++*. Jakarta: Elex Media Komputindo.

- Munir, R. (2018). *Algoritma dan Struktur Data*. Bandung: Informatika.
- Weiss, M. A. (2014). *Data Structures and Algorithm Analysis in C++*. Boston: Pearson Education.
- Modul Praktikum Struktur Data. (2025). *Doubly Linked List*. Program Studi Teknik Informatika.