# LAPORAN PRAKTIKUM
# Modul 6
# DOUBLE LINKED LIST BAGIAN 1

**Disusun Oleh:**
**Zhafir Zaidan Avail**
**S1-SE-07-2**

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

**PROGRAM STUDI S1 SOFTWARE ENGINEERING**
**FAKULTAS INFORMATIKA**
**TELKOM UNIVERSITY**
**PURWOKERTO**
**2024**

## 1. Tujuan

1. Menguasai konsep double linked list.
2. Menerapkan konsep double linked list dengan menggunakan pointer dan bahasa C.

## 2. Landasan Teori

- ***Double LinkedList***

  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.

  Contoh pendeklarasian struktur data untuk double linked list:

```
#ifndef doublelist_H
#define doublelist_H
#include "boolean.h"
#define Nil NULL
#define info(P) (P)->info
#define next(P) (P)->next
#define prev(P) (P)->prev
#define first(L) ((L).first)
#define last(L) ((L).last)

/*deklarasi record dan struktur data double linked list*/
typedef int infotype;
typedef struct elmlist *address;
struct elmlist {
 infotype info;
 address next;
 address prev;
};
/* definisi list: */
/* list kosong jika First(L)=Nil */
struct list{
address first;
address last;
};
#endif
```

- ***Insert***
  - ***Insert First***

```
void insertFirst(list &L, address &P) {
    // Sambungkan node baru ke depan list
    next(P) = first(L);
    if (first(L) != NULL) {
        prev(first(L)) = P;
    }
    first(L) = P;
    if (last(L) == NULL) {
        last(L) = P;
    }
}
```

  - ***Insert Last***

```
void insertLast(list &L, address &P) {
    // Jika list kosong
    if (first(L) == NULL) {
```

```
        first(L) = P;
        last(L) = P;
    } else {
        // Sambungkan node baru di akhir list
        prev(P) = last(L);
        next(last(L)) = P;
        last(L) = P;
    }
}
```

- ***Insert After***

```
void insertAfter(list &L, address &P, address &prevNode) {
    // Periksa apakah prevNode valid (ada dalam list)
    if (prevNode == NULL) {
        // Jika prevNode tidak ditemukan, kembalikan (atau lakukan
tindakan lain)
        return;
    }

    // Sambungkan node baru setelah prevNode
    next(P) = next(prevNode);
    prev(P) = prevNode;
    if (next(prevNode) != NULL) {
        prev(next(prevNode)) = P;
    }
    next(prevNode) = P;
}
```

- ***Insert Before***

```
void insertBefore(list &L, address &P, address &nextNode) {
    // Periksa apakah nextNode valid (ada dalam list)
    if (nextNode == NULL) {
        // Jika nextNode tidak ditemukan, kembalikan (atau lakukan
tindakan lain)
        return;
    }

    // Sambungkan node baru sebelum nextNode
    prev(P) = prev(nextNode);
    next(P) = nextNode;
    if (prev(nextNode) != NULL) {
        next(prev(nextNode)) = P;
    }
    prev(nextNode) = P;
}
```

- ***Delete***
  - ***Delete First***

```
void deleteFirst(list &L, address &P) {
    P = first(L); // Simpan pointer ke elemen pertama pada P

    first(L) = next(first(L)); // Ubah first(L) menjadi elemen kedua

    // Jika list tinggal satu elemen
    if (first(L) == NULL) {
        last(L) = NULL; // Jika elemen pertama adalah satu-satunya, maka
last juga NULL
    } else {
        prev(first(L)) = NULL; // Atur prev dari elemen baru pertama
menjadi NULL
    }

    // Putuskan node yang dihapus
    next(P) = NULL;
    prev(P) = NULL;
}
```

- ***Delete Last***

```
void deleteLast(list &L, address &P) {
    if (last(L) == NULL) {
        // Jika list kosong, tidak ada yang perlu dihapus
        return;
    }

    P = last(L); // Simpan pointer ke elemen terakhir pada P

    if (first(L) == last(L)) {
        // Jika hanya ada satu elemen, maka first dan last menjadi NULL
        first(L) = NULL;
        last(L) = NULL;
    } else {
        last(L) = prev(last(L)); // Ubah last(L) menjadi elemen sebelum
terakhir
        next(last(L)) = NULL; // Atur next dari elemen terakhir baru
menjadi NULL
    }

    // Putuskan node yang dihapus
    next(P) = NULL;
    prev(P) = NULL;
}
```

- ***Delete After***

```
void deleteAfter(list &L, address &prevNode) {
    if (prevNode == NULL || next(prevNode) == NULL) {
        // Jika prevNode atau next(prevNode) tidak ada, tidak ada yang
dihapus
        return;
    }

    address temp = next(prevNode); // Simpan pointer ke elemen yang akan
dihapus

    next(prevNode) = next(temp);
    if (next(temp) != NULL) {
        prev(next(temp)) = prevNode;
    }

    // Putuskan node yang dihapus
    next(temp) = NULL;
    prev(temp) = NULL;
}
```

- ***Delete Before***

```
void deleteBefore(list &L, address &nextNode) {
    if (nextNode == NULL || prev(nextNode) == NULL) {
        // Jika nextNode atau prev(nextNode) tidak ada, tidak ada yang
dihapus
        return;
    }

    address temp = prev(nextNode); // Simpan pointer ke elemen yang akan
dihapus

    prev(nextNode) = prev(temp);
    if (prev(temp) != NULL) {
        next(prev(temp)) = nextNode;
    }

    // Putuskan node yang dihapus
    next(temp) = NULL;
    prev(temp) = NULL;
}
```

- ***Update, View, dan Searching***
  - ***Update***

```
void updateNode(Node* head, int oldValue, int newValue) {
    Node* current = head;
    while (current != nullptr) {
        if (current->data == oldValue) {
            current->data = newValue;
            return;
        }
        current = current->next;

    }
```

  - ***View***

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

  - ***Searching***

```
Node* search(Node* head, int value) {
    Node* current = head;
    while (current != nullptr) {
        if (current->data == value) {
            return current; // Node ditemukan
        }
        current = current->next;
    }
```

## 3. Guided

Code:

```
#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;
        cout << "List Tersimpan: ";
        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;
}
```

Output Clear

```
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:
```

Output Delete

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

Output Insert

```
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: 23
```

Output Update

```
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 3
Enter old data: 2
Enter new data: 9
```

## 4. Unguided

### File doublelist.h

```
#ifndef doublelist_H
#define doublelist_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define Nil NULL
#define info(P) (P)->info
#define next(P) (P)->next
#define prev(P) (P)->prev
#define first(L) ((L).first)
#define last(L) ((L).last)

typedef struct {
    char nopol[10];
    char warna[10];
    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);
address Alokasi(infotype X);
void Dealokasi(address P);
void InsertLast(list *L, address P);
void PrintInfo(list L);
address findElm(list L, char *nopol);
void deleteFirst(list *L, address *P);
void deleteLast(list *L, address *P);
void deleteAfter(address Prec, address *P);
```

```
#endif
```

**File doublelist.cpp**

```
#include "doublelist.h"

void CreateList(list *L) {
    first(*L) = Nil;
    last(*L) = Nil;
}

address Alokasi(infotype X) {
    address P = (address) malloc(sizeof(struct elmlist));
    if (P != Nil) {
        info(P) = X;
        next(P) = Nil;
        prev(P) = Nil;
    }
    return P;
}

void Dealokasi(address P) {
    free(P);
}

void InsertLast(list *L, address P) {
    if (first(*L) == Nil) {
        first(*L) = P;
        last(*L) = P;
    } else {
        next(last(*L)) = P;
        prev(P) = last(*L);
        last(*L) = P;
    }
}

void PrintInfo(list L) {
    address P = first(L);
    while (P != Nil) {
        printf("Nomor Polisi: %s\n", info(P).nopol);
        printf("Warna Kendaraan: %s\n", info(P).warna);
        printf("Tahun Kendaraan: %d\n", info(P).thnBuat);
        printf("\n");
        P = next(P);
    }
}

address findElm(list L, char *nopol) {
    address P = first(L);
    while (P != Nil) {
        if (strcmp(info(P).nopol, nopol) == 0) {
            return P;  // Mengembalikan alamat elemen jika ditemukan
        }
        P = next(P);
    }
    return Nil;  // Jika tidak ditemukan
}

void deleteFirst(list *L, address *P) {
    *P = first(*L);
    if (first(*L) != Nil) {
        if (first(*L) == last(*L)) {
            first(*L) = Nil;
            last(*L) = Nil;
        } else {
            first(*L) = next(first(*L));
            prev(first(*L)) = Nil;
```

```
        }
        next(*P) = Nil;
        prev(*P) = Nil;
    }
}

void deleteLast(list *L, address *P) {
    *P = last(*L);
    if (first(*L) != Nil) {
        if (first(*L) == last(*L)) {
            first(*L) = Nil;
            last(*L) = Nil;
        } else {
            last(*L) = prev(last(*L));
            next(last(*L)) = Nil;
        }
        prev(*P) = Nil;
        next(*P) = Nil;
    }
}

void deleteAfter(address Prec, address *P) {
    if (next(Prec) != Nil) {
        *P = next(Prec);
        next(Prec) = next(next(Prec));
        if (next(Prec) != Nil) {
            prev(next(Prec)) = Prec;
        }
        next(*P) = Nil;
        prev(*P) = Nil;
    }
}
```

**File main.cpp**

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

using namespace std;

int main() {
    list L;
    CreateList(&L);

    int jumlahKendaraan;
    cout << "Masukkan jumlah kendaraan yang ingin dimasukkan: ";
    cin >> jumlahKendaraan;

    char nopol[10];
    char warna[10];
    int thnBuat;
    infotype kendaraan;
    address P;

    for (int i = 0; i < jumlahKendaraan; i++) {
        cout << "Masukkan nomor polisi : ";
        cin >> nopol;

        // Mengecek apakah nomor polisi sudah ada
        if (findElm(L, nopol) != Nil) {
            cout << "NOMOR POLISI SUDAH TERDAFTAR !" << endl;
            i--;  // Meminta input ulang jika nomor polisi sudah
terdaftar
            continue;
        }

        cout << "Masukkan warna kendaraan : ";
```

```
        cin >> warna;
        cout << "Masukkan tahun kendaraan : ";
        cin >> thnBuat;

        // Mengisi data kendaraan
        strcpy(kendaraan.nopol, nopol);
        strcpy(kendaraan.warna, warna);
        kendaraan.thnBuat = thnBuat;

        // Menambahkan data ke dalam list
        P = Alokasi(kendaraan);
        InsertLast(&L, P);
    }

    // Menampilkan data yang ada di dalam list
    cout << "\nData List 1:" << endl;
    PrintInfo(L);

    // Mencari elemen berdasarkan nomor polisi
    char cariNopol[10];
    cout << "\nMasukkan nomor polisi yang ingin dicari: ";
    cin >> cariNopol;

    address ditemukan = findElm(L, cariNopol);
    if (ditemukan != Nil) {
        cout << "Data ditemukan:\n";
        cout << "Nomor Polisi: " << info(ditemukan).nopol << endl;
        cout << "Warna: " << info(ditemukan).warna << endl;
        cout << "Tahun: " << info(ditemukan).thnBuat << endl;
    } else {
        cout << "Nomor polisi " << cariNopol << " tidak ditemukan!" <<
endl;
    }

    // Menghapus elemen dengan nomor polisi tertentu
    char hapusNopol[10];
    cout << "\nMasukkan nomor polisi yang ingin dihapus: ";
    cin >> hapusNopol;

    P = findElm(L, hapusNopol);
    if (P != Nil) {
        if (P == first(L)) {
            deleteFirst(&L, &P);
        } else if (P == last(L)) {
            deleteLast(&L, &P);
        } else {
            address Prec = prev(P);  // Mencari elemen sebelum P
            deleteAfter(Prec, &P);
        }
        cout << "Data dengan nomor polisi " << hapusNopol << " berhasil
dihapus.\n";
    } else {
        cout << "Nomor polisi " << hapusNopol << " tidak ditemukan!\n";
    }

    // Menampilkan kembali data list setelah penghapusan
    cout << "\nData List Setelah Penghapusan:" << endl;
    PrintInfo(L);

    return 0;
}
```

Output:

```
Masukkan jumlah kendaraan yang ingin dimasukkan: 4
Masukkan nomor polisi : D001
Masukkan warna kendaraan : hitam
```

```
Masukkan tahun kendaraan : 90
Masukkan nomor polisi : D002
Masukkan warna kendaraan : putih
Masukkan tahun kendaraan : 70
Masukkan nomor polisi : D001
NOMOR POLISI SUDAH TERDAFTAR !
Masukkan nomor polisi : D003
Masukkan warna kendaraan : hijau
Masukkan tahun kendaraan : 80
Masukkan nomor polisi : D004
Masukkan warna kendaraan : hitam
Masukkan tahun kendaraan : 98

Data List 1:
Nomor Polisi: D001
Warna Kendaraan: hitam
Tahun Kendaraan: 90

Nomor Polisi: D002
Warna Kendaraan: putih
Tahun Kendaraan: 70

Nomor Polisi: D003
Warna Kendaraan: hijau
Tahun Kendaraan: 80

Nomor Polisi: D004
Warna Kendaraan: hitam
Tahun Kendaraan: 98


Masukkan nomor polisi yang ingin dicari: D001
Data ditemukan:
Nomor Polisi: D001
Warna: hitam
Tahun: 90

Masukkan nomor polisi yang ingin dihapus: D003
Data dengan nomor polisi D003 berhasil dihapus.

Data List Setelah Penghapusan:
Nomor Polisi: D001
Warna Kendaraan: hitam
Tahun Kendaraan: 90

Nomor Polisi: D002
Warna Kendaraan: putih
Tahun Kendaraan: 70

Nomor Polisi: D004
Warna Kendaraan: hitam
Tahun Kendaraan: 98


Process returned 0 (0x0)   execution time : 198.245 s
Press any key to continue.
```

## 5. Kesimpulan

Kode C++ yang diberikan secara komprehensif mengimplementasikan struktur data double linked list. Struktur ini memungkinkan setiap elemen dalam list memiliki dua pointer: next yang menunjuk ke elemen selanjutnya dan prev yang menunjuk ke elemen sebelumnya. Hal ini memberikan fleksibilitas dalam melakukan operasi penyisipan dan penghapusan baik di awal, akhir, maupun di tengah list.

Kode tersebut telah mendefinisikan berbagai fungsi penting untuk mengelola double linked list,

seperti:

- **Fungsi untuk membuat list kosong:** CreateList
- **Fungsi untuk mengalokasikan node baru:** alokasi
- **Fungsi untuk dealokasi node:** dealokasi
- **Fungsi untuk menampilkan seluruh elemen list:** printInfo
- **Fungsi untuk menambahkan elemen di akhir list:** insertLast
- **Fungsi untuk mencari elemen berdasarkan nilai:** findElm
- **Fungsi untuk menghapus elemen pertama, terakhir, setelah, dan sebelum elemen tertentu:** deleteFirst, deleteLast, deleteAfter, deleteBefore

Selain fungsi-fungsi dasar di atas, kode juga memberikan contoh implementasi untuk memperbarui nilai suatu node, menampilkan seluruh elemen list, dan mencari suatu node berdasarkan nilai. Implementasi double linked list ini dapat menjadi dasar untuk membangun berbagai macam aplikasi yang membutuhkan struktur data dinamis dan fleksibel. Dengan pemahaman yang baik tentang konsep double linked list dan kode yang telah diberikan, Anda dapat mengembangkan aplikasi yang lebih kompleks dan efisien.