# LAPORAN PRAKTIKUM

## Modul 4

## DAFTAR BERANTAI TUNGGAL (*SINGLE LINKED LIST*)


**Disusun Oleh:**

**Adhiansyah Muhammad Pradana Farawowan - 2211104038**

**S1SE-07-02**

**Asisten Praktikum:**

**Aldi Putra**

**Andini Nur Hidayah**

**Dosen:**

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

**PROGRAM STUDI S1 REKAYASAN PERANGKAT LUNAK**

**FAKULTAS INFORMATIKA**

**UNIVERSITAS TELKOM PURWOKERTO**


**2024**

## A. Tujuan

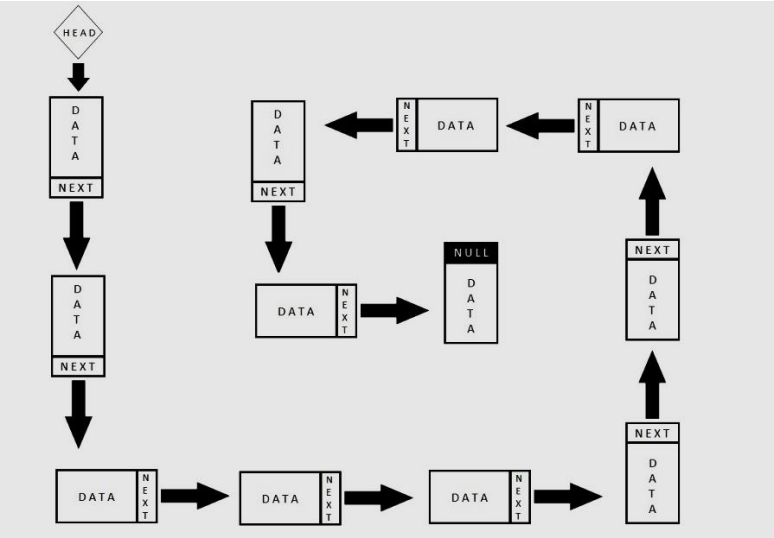
Laporan praktikum ini bertujuan untuk memperkenalkan, memahami, mengelola, dan menuntun cara mengimplementasikan daftar berantai tunggal.

## B. Landasan Teori

### a. Daftar berantai

Daftar berantai (*linked list*) adalah struktur data berupa koleksi yang elemennya menunjuk pada elemen lain yang sedemikian sehingga membentuk sebuah urutan yang berantai.

Sebuah elemen daftar terdiri dari isi data/informasi itu sendiri dan sebuah tipe yang menunjuk ke elemen selanjutnya (*next*). Dalam C++, *next* ini bisa diimplementasikan dengan penunjuk (*pointer*).




Ilustrasi daftar berantai. Sumber: Vhcomptech di commonswiki.

Daftar berantai memiliki dua jenis, yaitu tunggal dan ganda. Daftar berantai ganda memiliki penunjuk ke elemen sebelumnya dan setelahnya. Laporan ini akan membahas daftar berantai tunggal.

## C. Bimbingan (*guided*)

Implementasi 1

```
#include <iostream>

struct MAHASISWA {
	char nama[10];
	char nim[10];
};

struct NODE {
	struct MAHASISWA data;
	NODE *next;
};
```

```
NODE *head;
NODE *tail;

void init() {
	head = nullptr;
	tail = nullptr;
}

bool daftar_ternyata_kosong() {
	return head == nullptr;
}

void sisip_paling_depan(const struct MAHASISWA& data) {
	NODE *baru = new NODE;
	baru->data = data;
	baru->next = nullptr;

	if (daftar_ternyata_kosong()) {
		head = baru;
		tail = baru;
	} else {
		baru->next = head;
		head = baru;
	}
}

void sisip_paling_belakang(const struct MAHASISWA& data) {
	NODE *baru = new NODE;
	baru->data = data;
	baru->next = nullptr;

	if (daftar_ternyata_kosong()) {
		head = baru;
		tail = baru;
	} else {
		tail->next = baru;
		tail = baru;
	}
}

int jumlah_elemen_daftar() {
	NODE *c = head;
	int jumlah = 0;

	while (c->next != nullptr) {
		jumlah = jumlah + 1;
		c = c->next;
	}


	return jumlah;
}

void hapus_paling_depan() {
	if (daftar_ternyata_kosong()) {
		std::cout << "Daftarnya kosong" << std::endl;
	} else {
		NODE *elemen_target = head;
		head = head->next;

		delete elemen_target;
```

```
		if (head == nullptr) {
			tail = nullptr;
		}
	}
}

void hapus_paling_belakang() {
	if (head == tail) {
		delete head;
		head = nullptr;
		tail = nullptr;
	} else {
		NODE *elemen_terakhir_baru = head;

		while (elemen_terakhir_baru->next != tail) {
			elemen_terakhir_baru = elemen_terakhir_baru->next;
		}

		delete tail;
		tail = elemen_terakhir_baru;
		tail->next = nullptr;
	}
}

void tampilkan_daftar() {
	if (daftar_ternyata_kosong()) {
		std::cout << "Daftarnya kosong" << std::endl;
	} else {
		NODE *elms = head;

		while (elms != nullptr) {
			std::cout << elms->data.nama << "+" << elms->data.nim << " =>
";
			elms = elms->next;
		}
	}

	std::cout << '\n';
}

void hapus_daftar() {
	NODE *c = head;

	while (c != nullptr) {
		NODE *target = c;
		c = c->next;
		delete target;
	}

	head = nullptr;
	tail = nullptr;

	std::cout << '\n';
	std::cout << "Berhasil dihapus" << std::endl;
}

// Main function
int main() {
    head = nullptr;
	tail = nullptr;

    struct MAHASISWA m1 = {"Alice", "123456"};
    struct MAHASISWA m2 = {"Bob", "654321"};
```

```
    struct MAHASISWA m3 = {"Charlie", "112233"};

       sisip_paling_depan(m1);
       tampilkan_daftar();
       sisip_paling_belakang(m2);
       tampilkan_daftar();
       sisip_paling_depan(m3);
       tampilkan_daftar();

       hapus_paling_depan();
       tampilkan_daftar();
       hapus_paling_belakang();
       tampilkan_daftar();

    hapus_daftar();

    return 0;
}
```

```
                                                                >a.exe
Alice+123456 =>
Alice+123456 => Bob+654321 =>
Charlie+112233 => Alice+123456 => Bob+654321 =>
Alice+123456 => Bob+654321 =>
Alice+123456 =>

Berhasil dihapus
```

Implementasi 2

```
#include <iostream>

struct NODE {
    int data;
    NODE* next;
};

// Fungsi alokasi (buat elemen)
NODE* alok(int isi_data) {
    NODE* elemen_terakhir_baru = new(NODE);

    if (elemen_terakhir_baru != nullptr) {
        elemen_terakhir_baru->data = isi_data;
        elemen_terakhir_baru->next = nullptr;
    }

    return elemen_terakhir_baru;
}

// Fungsi dealokasi (hapus) elemen
void dealok(NODE* NODE) {
    delete NODE;
}




// Cek apakah isi daftar kosong
```

```
bool isi_ternyata_kosong(NODE* head) {
    return head == nullptr;
}

// Sisip di awal daftar
void sisip_di_awal(NODE* &head, int isi_nilai) {
    NODE* elemen_terawal_baru = alok(isi_nilai);

    if (elemen_terawal_baru != nullptr) {
        elemen_terawal_baru->next = head;
        head = elemen_terawal_baru;
    }
}

// Sisip di akhir daftar
void sisip_di_akhir(NODE* &head, int value) {
    NODE* elemen_terakhir_baru = alok(value);

    if (elemen_terakhir_baru != nullptr) {
        if (isi_ternyata_kosong(head)) {
            head = elemen_terakhir_baru;
        } else {
            NODE* elemen_terakhir = head;

            while (elemen_terakhir->next != nullptr) {
                elemen_terakhir = elemen_terakhir->next;
            }

            elemen_terakhir->next = elemen_terakhir_baru;
        }
    }
}

// Menampilkan semua elemen
void cetak_daftar(NODE* head) {
    if (isi_ternyata_kosong(head)) {
        std::cout << "Daftar kosong" << std::endl;
    } else {
        NODE* elemen = head;

        while (elemen != nullptr) {
            std::cout << elemen->data << " => ";
            elemen = elemen->next;
        }
        std::cout << "NULL";
        std::cout << std::endl;
    }
}

// Menghitung jumlah semua elemen dalam daftar
int jumlah_elemen(NODE* head) {
    int jumlah = 0;
    NODE* elemen = head;

    while (elemen != nullptr) {
        jumlah = jumlah + 1;
        elemen = elemen->next;
    }

    return jumlah;
}
```

```
// Menghapus semua elemen dalam daftar
void hapus_semua_elemen(NODE* &head) {
    while (head != nullptr) {
        NODE* elemen = head;
        head = head->next;
        dealok(elemen);
    }
}

int main() {
    // Daftar kosong
    NODE* head = nullptr;

    // Menambahkan elemen
    sisip_di_awal(head, 10);
    sisip_di_akhir(head, 20);
    sisip_di_akhir(head, 30);

    // Menampilkan isi daftar
    std::cout << "Isi daftar: ";
    cetak_daftar(head);

    // Menampilkan jumlah elemen
    std::cout << "Jumlah elemen: " << jumlah_elemen(head) << std::endl;

    // Menghapus semua elemen
    hapus_semua_elemen(head);

    // Menampilkan isi daftar setelah penghapusan
    std::cout << "Isi daftar setelah penghapusan: ";
    cetak_daftar(head);

    return 0;
}
```

## D. *Unguided*

a. Membuat Single Linked List

Buatlah program C++ untuk membuat sebuah single linked list dengan operasi dasar sebagai berikut:

Insert Node di Depan: Fungsi untuk menambah node baru di awal linked list.
Insert Node di Belakang: Fungsi untuk menambah node baru di akhir linked list.
Cetak Linked List: Fungsi untuk mencetak seluruh isi linked list.

Contoh input dan output:
Input:
1. Tambah node di depan (nilai: 10)
2. Tambah node di belakang (nilai: 20)
3. Tambah node di depan (nilai: 5)
4. Cetak linked list

Output:
5 -> 10 -> 20

Kode sumber lengkap tersedia di UNGUIDED/unguided_1.cxx

```
SIMPUL* jawaban_1 = elemen_baru_kosong();
tambah_baru_di_awal(jawaban_1, 10);
tambah_baru_di_akhir(jawaban_1, 20);
tambah_baru_di_awal(jawaban_1, 5);
cetak_daftar(jawaban_1);

std::cout << '\n';
```

Output:

```
>a.exe
5 -> 10 -> 20 -> nullptr  OUTPUT SOAL 1
```

b. Menghapus Node pada Linked List
Buatlah program C++ yang dapat menghapus node tertentu dalam single linked list berdasarkan nilai yang diberikan oleh pengguna. Tugas ini mencakup operasi:

Delete Node dengan Nilai Tertentu: Fungsi untuk menghapus node yang memiliki nilai tertentu.
Cetak Linked List: Setelah penghapusan, cetak kembali isi linked list.

Contoh input/output:
Input:
1. Tambah node di depan (nilai: 10)
2. Tambah node di belakang (nilai: 20)
3. Tambah node di depan (nilai: 5)
4. Hapus node dengan nilai (nilai: 10)
5. Cetak linked list

Output:

5 -> 2

Kode sumber lengkap tersedia di UNGUIDED/unguided_1.cxx

```
SIMPUL* jawaban_2 = elemen_baru_kosong();
tambah_baru_di_awal(jawaban_2, 10);
tambah_baru_di_akhir(jawaban_2, 20);
tambah_baru_di_awal(jawaban_2, 5);
hapus_pada(jawaban_2, 10);
cetak_daftar(jawaban_2);
```

Output:

```
5 -> 20 -> nullptr OUTPUT SOAL 2
```

c. Mencari dan Menghitung Panjang Linked List
Buatlah program C++ yang dapat melakukan operasi berikut:
Cari Node dengan Nilai Tertentu: Fungsi untuk mencari apakah sebuah nilai ada di dalam linked list.
Hitung Panjang Linked List: Fungsi untuk menghitung jumlah node yang ada di dalam linked list.

Contoh input/output:
Input:
1. Tambah node di depan (nilai: 10)
2. Tambah node di belakang (nilai: 20)
3. Tambah node di depan (nilai: 5)
4. Cari node dengan nilai 20
5. Cetak panjang linked list

Output:
Node dengan nilai 20 ditemukan.
Panjang linked list: 3

Kode sumber lengkap tersedia di UNGUIDED/unguided_1.cxx

```
SIMPUL* jawaban_3 = elemen_baru_kosong();
tambah_baru_di_awal(jawaban_3, 10);
tambah_baru_di_akhir(jawaban_3, 20);
tambah_baru_di_awal(jawaban_3, 5);
cari_elemen(jawaban_3, 20);
std::cout << "Panjang daftar berantai: " << jumlah_elemen_daftar(jawaban_2)
             << '\n';
```

Output:

```
Simpul dengan nilai 20 ditemukan.
Panjang daftar berantai: 3
```