LAPORAN PRAKTIKUM
MODUL 4
“SINGLE LINGKED LIST(BAGIAN PERTAMA)”

Oleh:

NAMA :

Muhammad Daniel Anugrah Pratama

2311104063

SE-07-02

Dosen:

Wahyu Andi Saputra, S.Pd, M.Eng

PRODI S1 REKAYASA PERANGKAT LUNAK
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
2024

I. TUJUAN

1. Memahami penggunaan linked list dengan pointer operator- operator dalam program.
2. Memahami operasi-operasi dasar dalam linked list.
3. Membuat program dengan menggunakan linked list dengan prototype yang ada

II. DASAR TEORI

4.1 Linked List dengan Pointer

Linked list (biasa disebut list saja) adalah salah satu bentuk struktur data (representasi penyimpanan) berupa serangkaian elemen data yang saling berkait (berhubungan) dan bersifat fleksibel karena dapat tumbuh dan mengerut sesuai kebutuhan. Data yang disimpan dalam Linked list bisa berupa data tunggal atau data majemuk. Data tunggal merupakan data yang hanya terdiri dari satu data (variabel),

misalnya: nama bertipe string. Sedangkan data majemuk merupakan sekumpulan data (record) yang di dalamnya terdiri dari berbagai tipe data, misalnya: Data Mahasiswa, terdiri dari Nama bertipe string, NIM bertipe long integer, dan Alamat bertipe string. Linked list dapat diimplementasikan menggunakan Array dan Pointer (Linked list). Yang akan kita gunakan adalah pointer, karena beberapa alasan, yaitu :

1. Array bersifat statis, sedangkan pointer dinamis.
2. Pada linked list bentuk datanya saling bergandengan (berhubungan) sehingga lebih mudah memakai pointer.
3. Sifat linked list yang fleksibel lebih cocok dengan sifat pointer yang dapat diatur sesuai kebutuhan.
4. Karena array lebih susah dalam menangani linked list, sedangkan pointer lebih mudah.
5. Array lebih cocok pada kumpulan data yang jumlah elemen maksimumnya sudah diketahui dari awal.Dalam implementasinya, pengaksesan elemen pada Linked list dengan pointer bisa menggunakan

(->) atau tanda titik (.).

Model-model dari ADT Linked list yang kita pelajari adalah :

1. Single Linked list
2. Double Linked list
3. Circular Linked list

4. Multi Linked list
5. Stack (Tumpukan)
6. Queue (Antrian)
7. Tree
8. Graph

Setiap model ADT Linked list di atas memiliki karakteristik tertentu dan dalam penggunaannya disesuaikan dengan kebutuhan.

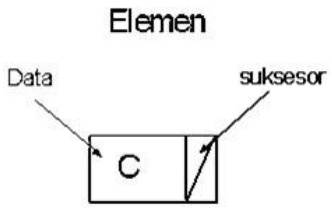
Secara umum operasi-operasi ADT pada Linked list, yaitu :

1. Penciptaan dan inisialisasi list (Create List).
2. Penyisipan elemen list (Insert).
3. Penghapusan elemen list (Delete).
4. Penelusuran elemen list dan menampilkannya (View).
5. Pencarian elemen list (Searching).
6. Pengubahan isi elemen list (Update).

4.2 *Single Linked List*

Single Linked list merupakan model ADT *Linked list* yang hanya memiliki satu arah *pointer*.

Komponen elemen dalam *single linked list*:



Gambar 4-1 Elemen *Single Linked list*

Keterangan:

Elemen: segmen-segmen data yang terdapat dalam suatu *list*.

Data: informasi utama yang tersimpan dalam sebuah elemen.

Suksesor: bagian elemen yang berfungsi sebagai penghubung antar elemen.

Sifat dari *Single Linked list*:

1. Hanya memerlukan satu buah *pointer*.
2. *Node* akhir menunjuk ke Nil kecuali untuk *list circular*.
3. Hanya dapat melakukan pembacaan maju.
4. Pencarian sequensial dilakukan jika data tidak terurut.
5. Lebih mudah ketika melakukan penyisipan atau penghapusan di tengah *list*.

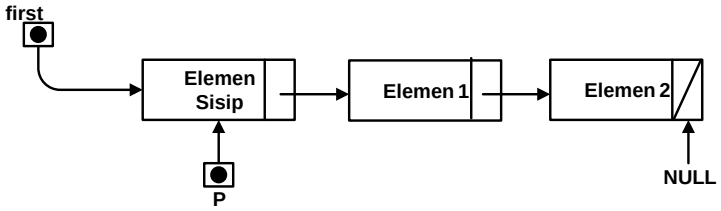
Istilah-istilah dalam *Single Linked list* :

1. *first*/*head*: *pointer* pada *list* yang menunjuk alamat elemen pertama *list*.
2. *next*: *pointer* pada elemen yang berfungsi sebagai *successor* (penunjuk) alamat elemen di depannya.
3. Null/Nil: artinya tidak memiliki nilai, atau tidak mengacu ke mana pun, atau kosong.
4. *Node*/simpul/elemen: merupakan tempat penyimpanan data pada suatu memori tertentu.

Gambaran sederhana *single linked list* dengan elemen kosong:

Gambar 4-2 *Single Linked list* dengan Elemen Kosong Gambaran

sederhana *single linked list* dengan 3 elemen:



Gambar 4-3 *Single Linked list* dengan 3 Elemen

Contoh deklarasi struktur data *single linked list*:

```
1  /*file : list.h*/
2  #ifndef LIST_H_INCLUDED
3  #define LIST_H_INCLUDED
4
5
6  #define Nil NULL
7  #define info(P) (P)->info
8  #define next(P) (P)->next #define
9  first(L) ((L).first) using
10 namespace std;
11 /*deklarasi record dan struktur data
12 list*/ typedef int infotype; typdef struct
13 elmlist *address; struct elmlist {
14 infotype info;    address next;
15 };
16 struct list{
17 address first;
18 };
19 #endif // TEST_H_INCLUDED
20
```

```
21
```

Misal untuk data mahasiswa yang terdiri dari nama dan nim.

```
1  /*file : list.h*/
2  #ifndef LIST_H_INCLUDED
3  #define LIST_H_INCLUDED
4
5
6  #define Nil NULL
7  #define info(P) (P)->info
8  #define next(P) (P)->next
9  #define first(L) ((L).first)
10
11 using namespace std;
12 /*deklarasi record dan struktur data list*/
13 struct mahasiswa{    char nama[30]    char nim[10]
14 }
15 typedef mahasiswa infotype;
16
17 typdef struct elmlist *address;
18 struct elmlist {    infotype
19 info;    address next;
20 };
21 struct list{
22 address first;
23 };
24 #endif // TEST_H_INCLUDED
25
26
27
```

4.2.1 Pembentukan Komponen-Komponen *List*

A. Pembentukan *List*

Adalah sebuah proses untuk membetuk sebuah *list* baru. Biasanya nama fungsi yang digunakan `createList()`. Fungsi ini akan mengeset nilai awal *list* yaitu *first*(*list*) dan *last*(*list*) dengan nilai Nil.

B. Pengalokasian Memori

Adalah proses untuk mengalokasikan memori untuk setiap elemen data yang ada dalam *list*. Fungsi yang biasanya digunakan adalah nama fungsi yang biasa digunakan `alokasi()`.

Sintak alokasi pada C:

```
P = (address) malloc ( sizeof (elmlist));
```

Keterangan:

P	= variabel *pointer* yang mengacu pada elemen yang dialokasikan.
address	= tipe data *pointer* dari tipe data elemen yang akan dialokasikan.
Elmlist	= tipe data atau *record* elemen yang dialokasikan.

Contoh deklarasi struktur data *single linked list*:

Misal untuk data mahasiswa yang terdiri dari nama dan nim.

```
address alokasi(mahasiswa m){      address p
= (address)malloc(sizeof(elmlist));
info(p) = m;      return p;
}
```

Namun pada Cpp. Penggunaan **malloc** dapat dipersingkat menggunakan sintak **new.**

Sintak alokasi pada Cpp:

```
P = new elmlist;
```

Keterangan:

P	= variabel *pointer* yang mengacu pada elemen yang dialokasikan.
address	= tipe data *pointer* dari tipe data elemen yang akan dialokasikan.

Contoh deklarasi struktur data *single linked list*:
Misal untuk data mahasiswa yang terdiri dari nama dan nim.

```
address alokasi(mahasiswa m){
    address p = new elmlist;
    info(p) = m;
    return p;
}
```


C. Dealokasi

Untuk menghapus sebuah *memory address* yang tersimpan atau telah dialokasikan dalam bahasa pemrograman C digunakan sintak ***free***, sedangkan pada Cpp digunakan sintak ***delete***, seperti berikut.

Sintak pada C:

```
free( p );
```

Sintak pada Cpp:

```
delete p;
```

D. Pengecekan *List*

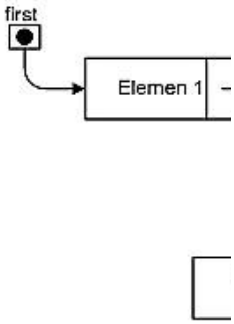
Adalah fungsi untuk mengecek apakah *list* tersebut kosong atau tidak. Akan mengembalikan nilai true jika *list* kosong dan nilai *false* jika *list* tidak kosong. Fungsi yang digunakan adalah `isEmpty()`.

4.2.2 Insert

A. *Insert First*

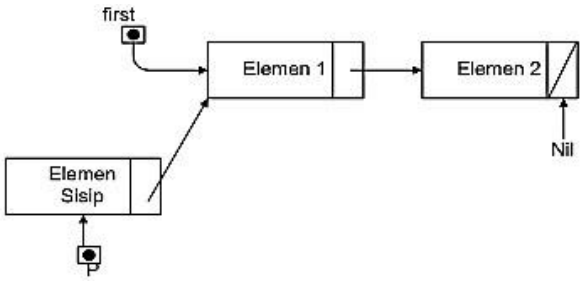
Merupakan metode memasukkan elemen data ke dalam *list* yang diletakkan pada awal *list*.

Langkah-langkah dalam proses *insert first*:



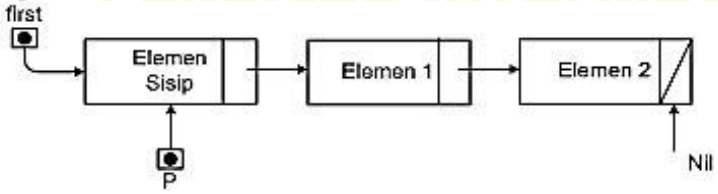

```
P = alokasi(X);
Next(P) = Nil;
```

Gambar 4-4 *Single Linked list Insert First* (1)




```
Next(P) = first;
```

Gambar 4-5 *Single Linked list Insert First* (2)




```
First(L) = P;
```

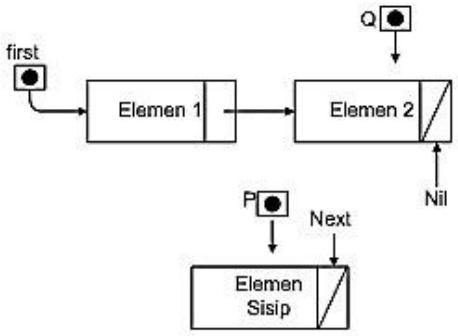
Gambar 4-6 *Single Linked list Insert First* (3)

```
/* contoh syntax insert first */ void insertFirst(List &L,
address &P){     next (P) = first(L);     first(L) = P; }
```

B. *Insert Last*

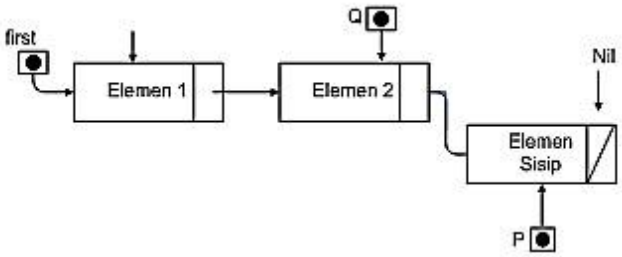
Merupakan metode memasukkan elemen data ke dalam *list* yang diletakkan pada akhir *list*.

Langkah dalam *insert last* :



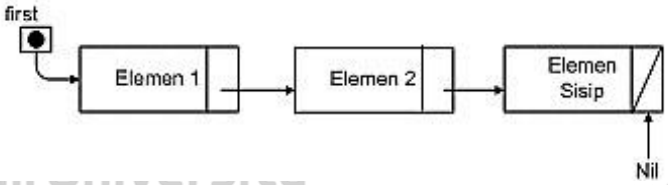

```
P = alokasi(X);
Next(P) = Nil;
```

Gambar 4-7 *Single Linked list Insert Last* 1




```
Next(Q)=P;
```

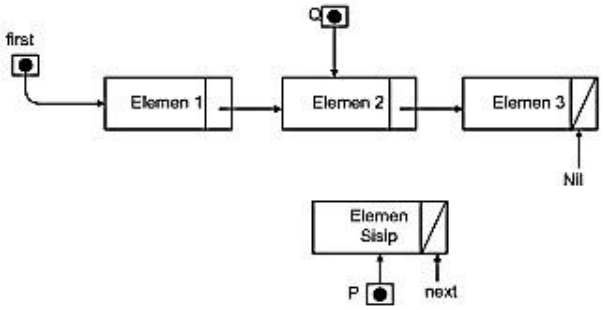
Gambar 4-8 *Single Linked list Insert Last* 2



Gambar 4-9 *Single Linked list Insert Last* 3

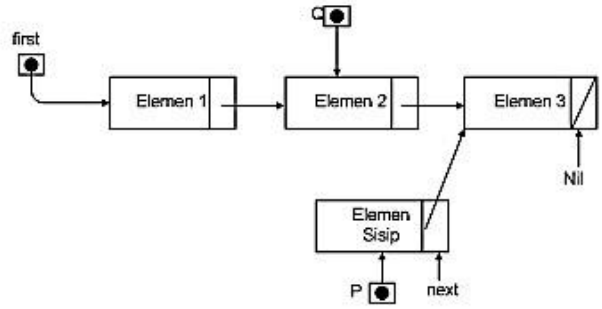
C. *Insert After*

Merupakan metode memasukkan data ke dalam *list* yang diletakkan setelah *node* tertentu yang ditunjuk oleh *user*. Langkah dalam *insert after*:



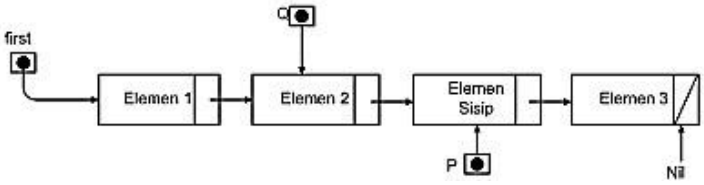

```
P = alokasi(X);
Next(P) = Nil;
```

Gambar 4-10 *Single Linked list Insert After* 1




```
Next(P) = Next(Q);
```

Gambar 4-11 *Single Linked list Insert After* 2




```
Next(Q) = P;
```

Gambar 4-12 *Single Linked list Insert After* 3

4.2.3 View

Merupakan operasi dasar pada *list* yang menampilkan isi *node*/simpul dengan suatu penelusuran *list*. Mengunjungi setiap *node* kemudian menampilkan data yang tersimpan pada *node* tersebut.

Semua fungsi dasar diatas merupakan bagian dari ADT dari singgle *linked list*, dan aplikasi pada bahasa pemrograman Cp semua ADT tersebut tersimpan dalam *file* *.c dan *file* *.h.

```
1   /*file : list .h*/
2   /* contoh ADT list berkait dengan representasi fisik pointer*/
3   /* representasi address dengan pointer*/
4   /* info tipe adalah integer */
5   #ifndef list_H
6   #define list_H
7   #include "boolean.h"
8   #include <stdio.h>
9   #define Nil NULL
10  #define info(P) (P)->info
11  #define next(P) (P)->next
12  #define first(L) ((L).first)
13
14
15  /*deklarasi record dan struktur data list*/
16  typedef int infotype;
17  typedef struct elmlist
18  *address; struct elmlist{
19  infotype info;
20   address next;
21  };
22
23  /* definisi list : */
24  /* list kosong jika First(L)=Nil */
25  /* setiap elemen address P dapat diacu info(P) atau next(P)
26  */ struct list {  address first;
27  };
28  /********* pengecekan apakah list kosong ***********/
29  boolean ListEmpty(list L);
30  /*mengembalikan nilai true jika list kosong*/
31  /********* pembuatan list kosong *********/
32  void CreateList(list &L);
33
```

```
/* I.S. sembarang
   F.S. terbentuk list kosong*/

/********* manajemen memori *********/
void dealokasi(address P);
/* I.S. P terdefinisi
   F.S. memori yang digunakan P dikembalikan ke sistem */

/********* penambahan elemen
**********/ void insertFirst(list &L,
address P); /* I.S. sembarang, P sudah
dialokasikan
   F.S. menempatkan elemen beralamat P pada awal list */
void insertAfter(list &L, address P, address Prec); /*
I.S. sembarang, P dan Prec alamt salah satu elemen list
   F.S. menempatkan elemen beralamat P sesudah elemen beralamat Prec */
void insertLast(list &L, address P);
/* I.S. sembarang, P sudah
dialokasikan
   F.S. menempatkan elemen beralamat P pada akhir list */

/********** proses semau elemen list *********/ void printInfo(list L); /*
I.S. list mungkin kosong    F.S. jika list tidak kosong menampilkan semua
info yang ada pada list */
int nbList(list L);
/* mengembalikan jumlah elemen pada list */

#endif
```

III. GUIDED

1. Single Linked

Kode Program:

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

// Deklarasi Struct untuk mahasiswa
struct mahasiswa {
    char nama[30];
    char nim[10];
};

// Deklarasi Struct Node
struct Node {
    mahasiswa data;
    Node *next;
};

Node *head;
Node *tail;

// Inisialisasi List
void init() {
    head = nullptr;
    tail = nullptr;
}

// Pengecekan apakah list kosong
bool isEmpty() {
    return head == nullptr;
}

// Tambah Depan
void insertDepan(const mahasiswa &data) {
    Node *baru = new Node;
    baru->data = data;
    baru->next = nullptr;
    if (isEmpty()) {
        head = tail = baru;
    } else {
        baru->next = head;
        head = baru;
    }
}

// Tambah Belakang
void insertBelakang(const mahasiswa &data) {
    Node *baru = new Node;
    baru->data = data;
    baru->next = nullptr;
    if (isEmpty()) {
        head = tail = baru;
    } else {
        tail->next = baru;
        tail = baru;
    }
}

// Hitung Jumlah List
int hitungList() {
    Node *current = head;
    int jumlah = 0;
    while (current != nullptr) {
        jumlah++;
        current = current->next;
    }
    return jumlah;
}

// Hapus Depan
void hapusDepan() {
    if (!isEmpty()) {
        Node *hapus = head;
        head = head->next;
        delete hapus;
        if (head == nullptr) {
            tail = nullptr; // Jika list menjadi kosong
        }
    } else {
        cout << "List kosong!" << endl;
    }
}

// Hapus Belakang
void hapusBelakang() {
    if (!isEmpty()) {
        if (head == tail) {
            delete head;
            head = tail = nullptr; // List menjadi kosong
        } else {
            Node *bantu = head;
            while (bantu->next != tail) {
                bantu = bantu->next;
            }
            delete tail;
            tail = bantu;
            tail->next = nullptr;
        }
    } else {
        cout << "List kosong!" << endl;
    }
}

// Tampilkan List
void tampil() {
    Node *current = head;
    if (!isEmpty()) {
        while (current != nullptr) {
            cout << "Nama: " << current->data.nama << ", NIM: " << current->data.nim << endl;
            current = current->next;
        }
    } else {
        cout << "List masih kosong!" << endl;
    }
}

// Hapus list
void clearList() {
    Node *current = head;
    while (current != nullptr) {
        Node *hapus = current;
        current = current->next;
        delete hapus;
    }
    head = tail = nullptr;
    cout << "List berhasil terhapus!" << endl;
}

// Main function
int main() {
    init();

    // Contoh data mahasiswa
    mahasiswa m1 = {"Alice", "123456"};
    mahasiswa m2 = {"Bob", "654321"};
    mahasiswa m3 = {"Charlie", "112233"};

    // Menambahkan mahasiswa ke dalam list
    insertDepan(m1);
    tampil();
    insertBelakang(m2);
    tampil();
    insertDepan(m3);
    tampil();

    // Menghapus elemen dari list
    hapusDepan();
    tampil();
    hapusBelakang();
    tampil();

    // Menghapus seluruh list
    clearList();

    return 0;
}
```

Outputnya:

```
PS D:\Pertemuan4sd> cd "d:\Pertemuan4sd\"
Nama: Alice, NIM: 123456
Nama: Alice, NIM: 123456
Nama: Bob, NIM: 654321
Nama: Charlie, NIM: 112233
Nama: Alice, NIM: 123456
Nama: Bob, NIM: 654321
Nama: Alice, NIM: 123456
Nama: Bob, NIM: 654321
Nama: Alice, NIM: 123456
List berhasil terhapus!
PS D:\Pertemuan4sd>
```

Kode ini adalah implementasi **Linked List** yang menyimpan data mahasiswa (berupa nama dan NIM). Operasi dasar yang bisa dilakukan adalah menambah elemen di depan dan belakang, menghapus elemen dari depan dan belakang, menghitung jumlah elemen, menampilkan elemen, dan menghapus seluruh list.

Komponen Utama

1. **Struct mahasiswa**:
    - Menyimpan data mahasiswa, yaitu nama dan NIM.
2. **Struct Node**:
    - Menyimpan data mahasiswa dan pointer ke node berikutnya dalam list.
3. **Pointer head dan tail**:
    - head: menunjuk ke elemen pertama dari list.
    - tail: menunjuk ke elemen terakhir dari list.

Fungsi-Fungsi Utama

1. **init()**:
    - Menginisialisasi list kosong, dengan head dan tail diset ke nullptr.
2. **isEmpty()**:
    - Mengecek apakah list kosong (yaitu, jika head == nullptr).
3. **insertDepan()**:
    - Menambahkan node baru di depan list.
    - Jika list kosong, node pertama menjadi head dan tail.

4. **insertBelakang()**:

    - Menambahkan node baru di belakang list.
    - Jika list kosong, node pertama menjadi head dan tail.

5. **hitungList()**:

    - Menghitung jumlah elemen yang ada dalam list dengan cara menghitung node satu per satu.

6. **hapusDepan()**:

    - Menghapus elemen pertama dari list.
    - Jika list menjadi kosong setelah penghapusan, tail juga di-set ke nullptr.

7. **hapusBelakang()**:

    - Menghapus elemen terakhir dari list.
    - Jika hanya ada satu elemen, head dan tail diset ke nullptr.

8. **tampil()**:

    - Menampilkan semua elemen dalam list (nama dan NIM mahasiswa).
    - Jika list kosong, menampilkan pesan "List masih kosong!"

9. **clearList()**:

    - Menghapus seluruh node dari list dan mengosongkan list sepenuhnya.

2. Single link list

   Kode Program:

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

// Deklarasi Struct untuk mahasiswa
struct mahasiswa {
    char nama[30];
    char nim[10];
};

// Deklarasi Struct Node
struct Node {
    mahasiswa data;
    Node *next;
};

Node *head;
Node *tail;

// Inisialisasi list
void init() {
    head = nullptr;
    tail = nullptr;
}

// Pengecekan apakah list kosong
bool isEmpty() {
    return head == nullptr;
}

// Tambah Depan
void insertDepan(const mahasiswa &data) {
    Node *baru = new Node;
    baru->data = data;
    baru->next = nullptr;
    if (isEmpty()) {
        head = tail = baru;
    } else {
        baru->next = head;
        head = baru;
    }
}

// Tambah Belakang
void insertBelakang(const mahasiswa &data) {
    Node *baru = new Node;
    baru->data = data;
    baru->next = nullptr;
    if (isEmpty()) {
        head = tail = baru;
    } else {
        tail->next = baru;
        tail = baru;
    }
}

// Hitung Jumlah List
int hitungList() {
    Node *current = head;
    int jumlah = 0;
    while (current != nullptr) {
        jumlah++;
        current = current->next;
    }
    return jumlah;
}

// Hapus Depan
void hapusDepan() {
    if (!isEmpty()) {
        Node *hapus = head;
        head = head->next;
        delete hapus;
        if (head == nullptr) {
            tail = nullptr; // Jika list menjadi kosong
        }
    } else {
        cout << "List kosong!" << endl;
    }
}

// Hapus Belakang
void hapusBelakang() {
    if (!isEmpty()) {
        if (head == tail) {
            delete head;
            head = tail = nullptr; // list menjadi kosong
        } else {
            Node *bantu = head;
            while (bantu->next != tail) {
                bantu = bantu->next;
            }
            delete tail;
            tail = bantu;
            tail->next = nullptr;
        }
    } else {
        cout << "List kosong!" << endl;
    }
}

// Tampilkan List
void tampil() {
    Node *current = head;
    if (!isEmpty()) {
        while (current != nullptr) {
            cout << "Nama: " << current->data.nama << ", NIM: " << current->data.nim << endl;
            current = current->next;
        }
    } else {
        cout << "List masih kosong!" << endl;
    }
}

// Hapus List
void clearList() {
    Node *current = head;
    while (current != nullptr) {
        Node *hapus = current;
        current = current->next;
        delete hapus;
    }
    head = tail = nullptr;
    cout << "List berhasil terhapus!" << endl;
}

// Main function
int main() {
    init();

    // Contoh data mahasiswa
    mahasiswa m1 = {"Alice", "1234567"};
    mahasiswa m2 = {"Bob", "7654321"};
    mahasiswa m3 = {"Charlie", "1122333"};

    // Menambahkan mahasiswa ke dalam list
    insertDepan(m1);
    tampil();
    insertBelakang(m2);
    tampil();
    insertDepan(m3);
    tampil();

    // Menghapus elemen dari list
    hapusDepan();
    tampil();
    hapusBelakang();
    tampil();

    // Menghapus seluruh list
    clearList();

    return 0;
}
```

Outputnya:

```
PS D:\Per4SD> cd "d:\Per4SD\" ; if ($?) {
Isi List: 10 20 30
Jumlah elemen: 3
Isi List setelah penghapusan: List kosong
PS D:\Per4SD>
```

Kode ini merupakan implementasi dasar **Single Linked List** dalam C++, yang menyediakan fungsi untuk:

1. Menambahkan elemen di awal dan akhir list.
2. Menampilkan semua elemen list.
3. Menghitung jumlah elemen dalam list.
4. Menghapus semua elemen dan mengosongkan list.

Komponen Utama

1. **Struct `Node`:**
    - Mewakili elemen list yang terdiri dari `data` (nilai elemen) dan `next` (pointer ke elemen berikutnya).
2. **Fungsi `alokasi()`:**
    - Mengalokasikan memori untuk node baru dan mengembalikannya.
3. **Fungsi `dealokasi()`:**
    - Menghapus node yang sudah tidak digunakan dari memori.
4. **Fungsi `isListEmpty()`:**
    - Mengecek apakah list kosong dengan memeriksa apakah `head` adalah `nullptr`.
5. **Fungsi `insertFirst()`:**
    - Menambahkan elemen baru di awal list (depan). Elemen baru akan menjadi elemen pertama.
6. **Fungsi `insertLast()`:**
    - Menambahkan elemen baru di akhir list (belakang). Jika list kosong, elemen baru menjadi elemen pertama.
7. **Fungsi `printList()`:**

- Menampilkan semua elemen dalam list. Jika list kosong, mencetak pesan "List kosong!".

8. **Fungsi countElements():**

   - Menghitung jumlah elemen dalam list dengan menelusuri semua elemen satu per satu.

9. **Fungsi clearList()**

   - Menghapus semua elemen dalam list dengan cara menghapus node satu per satu dan dealokasi memori.

IV. UNGUIDED

1. Kode Program:

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

// Definisi struktur untuk Node dalam Linked List
struct Node {
    int data;       // Menyimpan nilai elemen
    Node* next;     // Pointer ke elemen berikutnya
};

// Fungsi untuk mengalokasikan memori untuk node baru
Node* createNode(int value) {
    Node* newNode = new Node;   // Alokasi memori untuk node baru
    newNode->data = value;      // Menyimpan nilai node
    newNode->next = nullptr;    // Set next menjadi nullptr
    return newNode;             // Mengembalikan pointer ke node baru
}

// Fungsi untuk menambah node di depan (di awal linked list)
void insertDepan(Node* &head, int value) {
    Node* newNode = createNode(value);  // Membuat node baru
    newNode->next = head;   // Node baru menunjuk ke node pertama sebelumnya
    head = newNode;         // Node baru menjadi node pertama
}

// Fungsi untuk menambah node di belakang (di akhir linked list)
void insertBelakang(Node* &head, int value) {
    Node* newNode = createNode(value); // Membuat node baru
    if (head == nullptr) {             // Jika linked list kosong
        head = newNode;                // Node baru menjadi node pertama
    } else {
        Node* temp = head;
        while (temp->next != nullptr) { // Menelusuri sampai node terakhir
            temp = temp->next;
        }
        temp->next = newNode;          // Menambahkan node baru di akhir
    }
}

// Fungsi untuk mencetak seluruh isi linked list
void printList(Node* head) {
    if (head == nullptr) {             // Jika linked list kosong
        cout << "Linked List kosong!" << endl;
        return;
    }

    Node* temp = head;
    while (temp != nullptr) {          // Menelusuri seluruh linked list
        cout << temp->data;
        if (temp->next != nullptr) {
            cout << " -> ";            // Cetak panah antara elemen
        }
        temp = temp->next;
    }
    cout << endl;                      // Baris baru setelah selesai mencetak
}

// Main Program
int main() {
    Node* head = nullptr;              // Inisialisasi linked list kosong (head = nullptr)

    // Input operasi pada linked list
    insertDepan(head, 10);             // Menambah node dengan nilai 10 di depan
    insertBelakang(head, 20);          // Menambah node dengan nilai 20 di belakang
    insertDepan(head, 5);              // Menambah node dengan nilai 5 di depan

    // Mencetak seluruh isi linked list
    cout << "Isi Linked List: ";
    printList(head);

    return 0;
}
```

Outputnya:

```
PS D:\Per4SD> cd "d:\Per4SD\" ;
Isi Linked List: 5 -> 10 -> 20
PS D:\Per4SD>
```

2. Kode Program:

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

// Definisi struktur untuk Node dalam Linked List
struct Node {
    int data;       // Menyimpan nilai elemen
    Node* next;     // Pointer ke elemen berikutnya
};

// Fungsi untuk mengalokasikan memori untuk node baru
Node* createNode(int value) {
    Node* newNode = new Node;   // Alokasi memori untuk node baru
    newNode->data = value;      // Menyimpan nilai node
    newNode->next = nullptr;    // Set next menjadi nullptr
    return newNode;             // Mengembalikan pointer ke node baru
}

// Fungsi untuk menambah node di depan (di awal linked list)
void insertDepan(Node* &head, int value) {
    Node* newNode = createNode(value);  // Membuat node baru
    newNode->next = head;               // Node baru menunjuk ke node pertama sebelumnya
    head = newNode;                     // Node baru menjadi node pertama
}

// Fungsi untuk menambah node di belakang (di akhir linked list)
void insertBelakang(Node* &head, int value) {
    Node* newNode = createNode(value); // Membuat node baru
    if (head == nullptr) {             // Jika linked list kosong
        head = newNode;                // Node baru menjadi node pertama
    } else {
        Node* temp = head;
        while (temp->next != nullptr) { // Menelusuri sampai node terakhir
            temp = temp->next;
        }
        temp->next = newNode;          // Menambahkan node baru di akhir
    }
}

// Fungsi untuk mencetak seluruh isi linked list
void printList(Node* head) {
    if (head == nullptr) {             // Jika linked list kosong
        cout << "Linked List kosong!" << endl;
        return;
    }

    Node* temp = head;
    while (temp != nullptr) {          // Menelusuri seluruh linked list
        cout << temp->data;
        if (temp->next != nullptr) {
            cout << " -> ";            // Cetak panah antara elemen
        }
        temp = temp->next;
    }
    cout << endl;                      // Baris baru setelah selesai mencetak
}

// Fungsi untuk menghapus node dengan nilai tertentu
void deleteNode(Node* &head, int value) {
    if (head == nullptr) {  // Jika list kosong
        cout << "Linked List kosong, tidak ada yang bisa dihapus!" << endl;
        return;
    }

    // Jika node pertama yang harus dihapus
    if (head->data == value) {
        Node* temp = head;
        head = head->next;   // Geser head ke node berikutnya
        delete temp;         // Dealokasi node pertama
        cout << "Node dengan nilai " << value << " telah dihapus." << endl;
        return;
    }

    // Mencari node dengan nilai tertentu
    Node* current = head;
    Node* prev = nullptr;
    while (current != nullptr && current->data != value) {
        prev = current;         // Simpan node sebelumnya
        current = current->next; // Pindah ke node berikutnya
    }

    if (current == nullptr) {
        // Jika node dengan nilai tidak ditemukan
        cout << "Node dengan nilai " << value << " tidak ditemukan!" << endl;
        return;
    }

    // Hapus node yang ditemukan
    prev->next = current->next; // Lewati node yang akan dihapus
    delete current;             // Dealokasi node yang dihapus
    cout << "Node dengan nilai " << value << " telah dihapus." << endl;
}

// Main Program
int main() {
    Node* head = nullptr;  // Membuat linked list kosong (head = nullptr)

    // Menambahkan elemen ke dalam list
    insertDepan(head, 10);  // Menambah node dengan nilai 10 di depan
    insertBelakang(head, 20); // Menambah node dengan nilai 20 di belakang
    insertDepan(head, 5);   // Menambah node dengan nilai 5 di depan

    // Mencetak seluruh isi linked list sebelum penghapusan
    cout << "Isi Linked List sebelum penghapusan: ";
    printList(head);

    // Menghapus node dengan nilai tertentu
    deleteNode(head, 10);  // Menghapus node dengan nilai 10

    // Mencetak seluruh isi linked list setelah penghapusan
    cout << "Isi Linked List setelah penghapusan: ";
    printList(head);

    return 0;
}
```

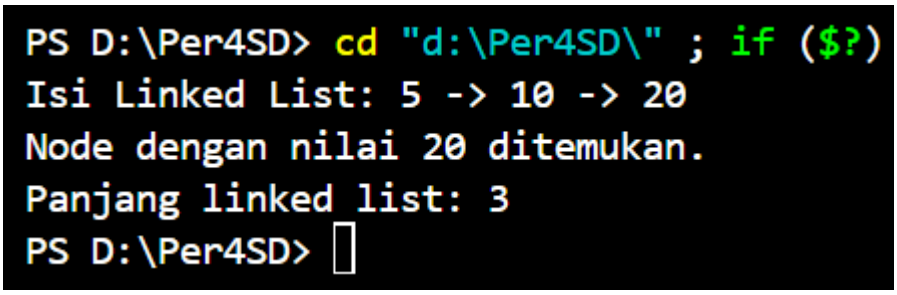
Outputnya:

```
PS D:\Per4SD> cd "d:\Per4SD\" ; if ($?) { g++ UGUIDED2.cpp
Isi Linked List sebelum penghapusan: 5 -> 10 -> 20
Node dengan nilai 10 telah dihapus.
Isi Linked List setelah penghapusan: 5 -> 20
PS D:\Per4SD>
```

3. Kode Program:

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

// Definisi struktur untuk Node dalam Linked List
struct Node {
    int data;       // Menyimpan nilai elemen
    Node* next;     // Pointer ke elemen berikutnya
};

// Fungsi untuk mengalokasikan memori untuk node baru
Node* createNode(int value) {
    Node* newNode = new Node;   // Alokasi memori untuk node baru
    newNode->data = value;      // Menyimpan nilai node
    newNode->next = nullptr;    // Set next menjadi nullptr
    return newNode;             // Mengembalikan pointer ke node baru
}

// Fungsi untuk menambah node di depan (di awal linked list)
void insertDepan(Node* &head, int value) {
    Node* newNode = createNode(value);  // Membuat node baru
    newNode->next = head;               // Node baru menunjuk ke node pertama sebelumnya
    head = newNode;                     // Node baru menjadi node pertama
}

// Fungsi untuk menambah node di belakang (di akhir linked list)
void insertBelakang(Node* &head, int value) {
    Node* newNode = createNode(value); // Membuat node baru
    if (head == nullptr) {             // Jika linked list kosong
        head = newNode;                // Node baru menjadi node pertama
    } else {
        Node* temp = head;
        while (temp->next != nullptr) { // Menelusuri sampai node terakhir
            temp = temp->next;
        }
        temp->next = newNode;          // Menambahkan node baru di akhir
    }
}

// Fungsi untuk mencari node dengan nilai tertentu
bool cariNode(Node* head, int value) {
    Node* temp = head;
    while (temp != nullptr) {          // Menelusuri seluruh linked list
        if (temp->data == value) {     // Jika nilai ditemukan
            return true;               // Mengembalikan true jika ditemukan
        }
        temp = temp->next;             // Pindah ke node berikutnya
    }
    return false;                      // Mengembalikan false jika tidak ditemukan
}

// Fungsi untuk menghitung panjang linked list
int hitungPanjang(Node* head) {
    int panjang = 0;
    Node* temp = head;
    while (temp != nullptr) {          // Menelusuri seluruh linked list
        panjang++;                     // Menambah hitungan panjang
        temp = temp->next;             // Pindah ke node berikutnya
    }
    return panjang;                    // Mengembalikan panjang linked list
}

// Fungsi untuk mencetak seluruh isi linked list
void printList(Node* head) {
    if (head == nullptr) {             // Jika linked list kosong
        cout << "Linked List kosong!" << endl;
        return;
    }

    Node* temp = head;
    while (temp != nullptr) {          // Menelusuri seluruh linked list
        cout << temp->data;
        if (temp->next != nullptr) {
            cout << " -> ";            // Cetak panah antar elemen
        }
        temp = temp->next;
    }
    cout << endl;                      // Baris baru setelah selesai mencetak
}

// Main Program
int main() {
    Node* head = nullptr;  // Membuat linked list kosong (head = nullptr)

    // Menambahkan elemen ke dalam list
    insertDepan(head, 10);   // Menambah node dengan nilai 10 di depan
    insertBelakang(head, 20); // Menambah node dengan nilai 20 di belakang
    insertDepan(head, 5);    // Menambah node dengan nilai 5 di depan

    // Mencetak seluruh isi linked list
    cout << "Isi Linked List: ";
    printList(head);

    // Mencari node dengan nilai tertentu
    int cariNilai = 20;
    if (cariNode(head, cariNilai)) {
        cout << "Node dengan nilai " << cariNilai << " ditemukan." << endl;
    } else {
        cout << "Node dengan nilai " << cariNilai << " tidak ditemukan." << endl;
    }

    // Menghitung dan menampilkan panjang linked list
    int panjang = hitungPanjang(head);
    cout << "Panjang linked list: " << panjang << endl;

    return 0;
}
```

Outputnya:



V. KESIMPULAN

Kesimpulan dari praktikum ini adalah bahwa linked list merupakan salah satu struktur data yang fleksibel karena dapat tumbuh dan menyusut sesuai kebutuhan, berbeda dengan array yang bersifat statis. Dalam linked list, setiap elemen terhubung melalui pointer, dan kita dapat dengan mudah melakukan operasi dasar seperti penambahan elemen (insert), penghapusan elemen (delete), penelusuran elemen (view), serta pengecekan apakah list kosong. Single linked list, yang hanya memiliki satu arah pointer, memudahkan dalam penyisipan dan penghapusan elemen, baik di awal, tengah, maupun akhir list. Implementasi single linked list menggunakan bahasa pemrograman C++ memperlihatkan bagaimana memanfaatkan pointer untuk mengelola memori secara dinamis, melakukan inisialisasi list kosong, pengecekan, serta penghapusan node. Penggunaan pointer memberikan efisiensi dan fleksibilitas yang lebih baik dalam pengelolaan data dibandingkan dengan struktur data statis seperti array.