LAPORAN PRAKTIKUM
Modul 4
“Single Linked List (Bagian Pertama)”

Disusun Oleh:
Dimastian Aji Wibowo (2311104058)
SE-07-02

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO

2024

1. **Tujuan**
    - Memahami penggunaan *linked list* dengan *pointer* operator – operator dalam program.
    - Memahami operasi – operasi dasar dalam linked list.
    - Membuat program dengan menggunakan linked list dengan *prototype* yang ada.
2. **Landasan Teori**

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 :

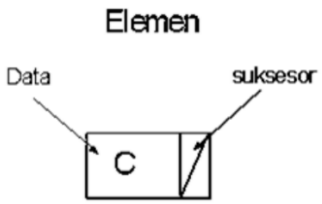
1. *Single Linked List*
2. *Double Linked List*
3. *Circular Linked List*
4. *Multi Linked List*
5. *Stack* (Tumpukan)
6. *Queue* (Antrian)
7. *Tree*
8. *Graph*

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).

A. Single Linked List

Single Linked list merupakan model ADT Linked list yang hanya memiliki satu arah pointer. Komponen elemen dalam 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.

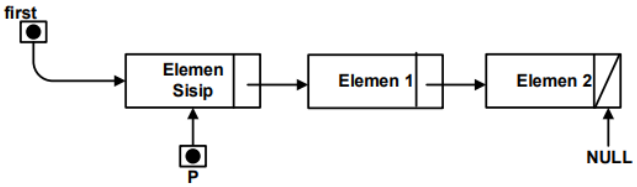
Istilah-istilah dalam Single Linked list :

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:

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

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

6. 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;
```

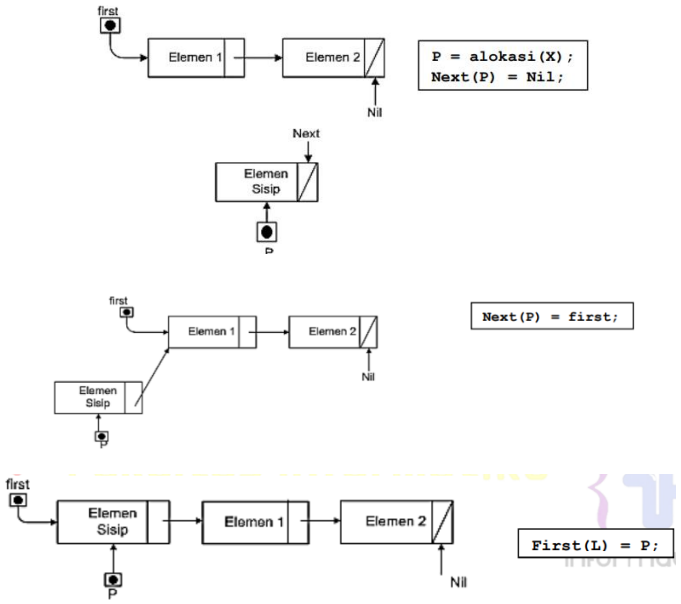
d. Pengecekan List

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().

7. Insert

a. Insert First

Merupakan metode memasukkan elemen data ke dalam list yang diletakkan pada awal list. Langkah-langkah dalam proses insert first:

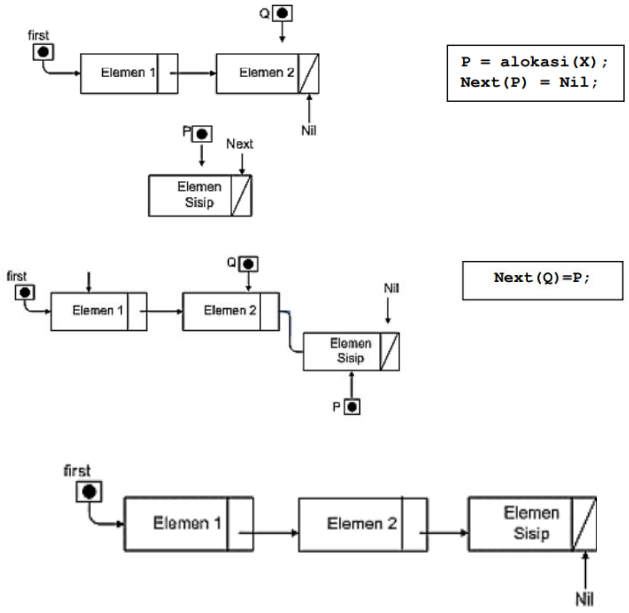



```
1 /* contoh syntax insert first */
```

```
2 void insertFirst(List &L, address &P){
3  next (P) = first(L);
4  first(L) = P;
5 }
```

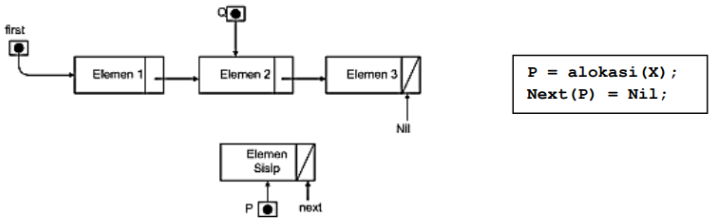
b. Insert Last

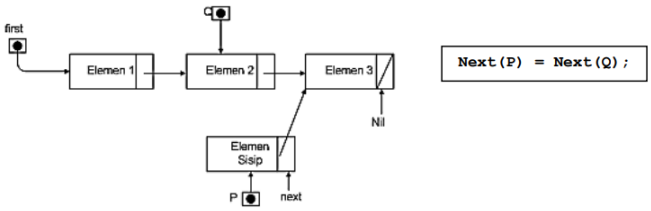
Merupakan metode memasukkan elemen data ke dalam list yang diletakkan pada akhir list. Langkah dalam insert last :

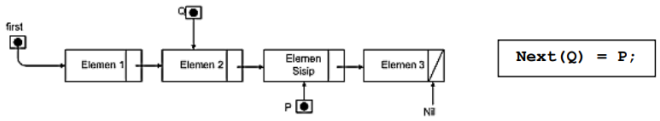


c. Insert After

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







8. 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
 3 pointer*/
 4 /* representasi address dengan pointer*/
 5 /* info tipe adalah integer */
 6 #ifndef list_H
 7 #define list_H
 8 #include "boolean.h"
 9 #include <stdio.h>
10 #define Nil NULL
11 #define info(P) (P)->info
12 #define next(P) (P)->next
13 #define first(L) ((L).first)
14 /*deklarasi record dan struktur data list*/
15 typedef int infotype;
16 typedef struct elmlist *address;
17 struct elmlist{
18 infotype info;
19 address next;
20 };
21 /* definisi list : */
22 /* list kosong jika First(L)=Nil */
23 /* setiap elemen address P dapat diacu info(P) atau
```

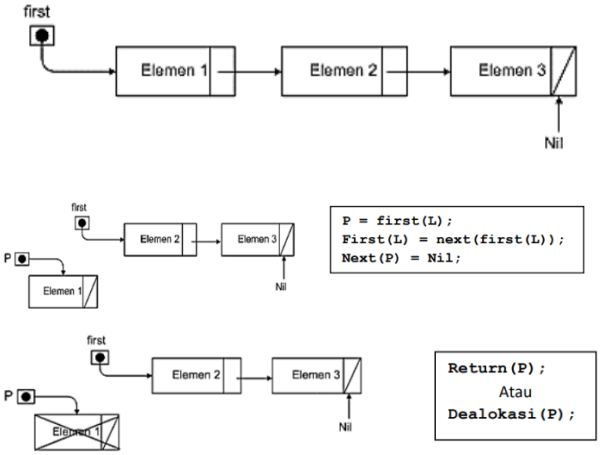
```
24 next(P) */
25 struct list {
26 address first;
27 };
28 /********* pengecekan apakah list kosong
29 ***********/
30 boolean ListEmpty(list L);
31 /*mengembalikan nilai true jika list kosong*/
32 /********* pembuatan list kosong *********/
33 void CreateList(list &L);
34 /* I.S. sembarang
35  F.S. terbentuk list kosong*/
36
37 /********* manajemen memori *********/
38 void dealokasi(address P);
39 /* I.S. P terdefinisi
40  F.S. memori yang digunakan P dikembalikan ke sistem
41 */
42
43 /********* penambahan elemen **********/
44 void insertFirst(list &L, address P);
45 /* I.S. sembarang, P sudah dialokasikan
46  F.S. menempatkan elemen beralamat P pada awal list
47 */
48
49 void insertAfter(list &L, address P, address Prec);
50 /* I.S. sembarang, P dan Prec alamt salah satu
51 elemen list
52  F.S. menempatkan elemen beralamat P sesudah elemen
53 beralamat Prec */
54 void insertLast(list &L, address P);
55 /* I.S. sembarang, P sudah dialokasikan
56  F.S. menempatkan elemen beralamat P pada akhir list
57 */

   /********** 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
```

9. Delete

a. Delete First

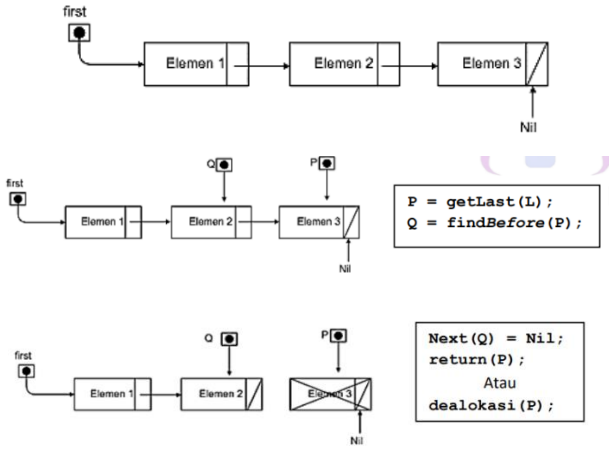
Adalah pengambilan atau penghapusan sebuah elemen pada awal list. Langkah-langkah dalam delete first:




```
1 /* contoh syntax delete first */
2 void deleteFirst(List &L, address &P){
3  P = first(L);
4  first(L) = next(first(L));
5  next (P) = null;
6 }
```

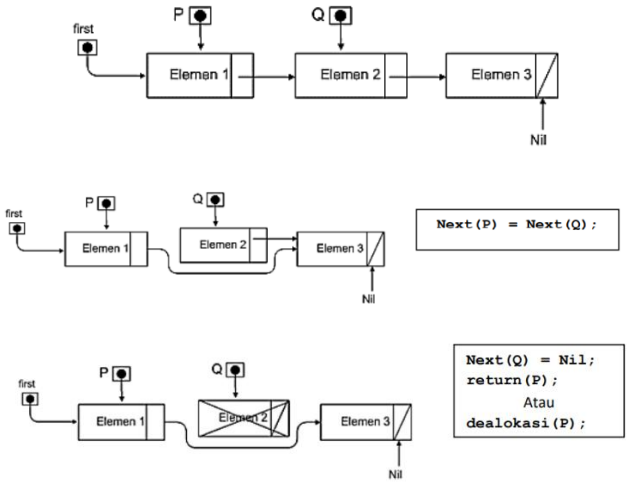
b. Delete Last

Merupakan pengambilan atau penghapusan suatu elemen dari akhir list. Langkah-langkah dalam delete last:



c. Delete After

Merupakan pengambilan atau penghapusan node setelah node tertentu. Langkah-langkah dalam delete after:



d. Delete Elemen

Adalah operasi yang digunakan untuk menghapus dan membebaskan memori yang dipakai oleh elemen tersebut.

Fungsi yang biasanya dipakai:

1. fungsi dealokasi(P) : membebaskan memori yang dipakai oleh elemen P.
2. fungsi delAll(L) : membebaskan semua memori yang dipakai elemen – elemen yang ada pada list L. Hasil akhir list L menjadi kosong.

Semua operasi-operasi dasar list biasa disebut dengan operasi primitif. Primitif-primitif dalam list ini merupakan bagian dari ADT list yang tersimpan dalam file *.h dan file *.cpp, dengan rincian file *.h untuk menyimpan prototipe primitif-primitif atau fungsi-fungsi dan menyimpan tipe data yang dipergunakan dalam primitif list tersebut.

Untuk bisa mengakses semua primitif tersebut yaitu dengan meng-include terhadap file *.h-nya.

10. Update

Merupakan operasi dasar pada list yang digunakan untuk mengupdate data yang ada di dalam list. Dengan operasi update ini kita dapat meng-update data-data node yang ada di dalam list. Proses update biasanya diawali dengan proses pencarian terhadap data yang akan di-update

3. Guided

A. Guided 1

1. Membuat struktur data mahasiswa yang menyimpan char nama[30] untuk menyimpan nama mahasiswa dan char nim[10] untuk menyimpan NIM.
2. Membuat struktur node dengan mahasiswa data untuk menyimpan data mahasiswa dan node *next yaitu pointer yang menunjuk ke pointer berikutnya.
3. Membuat fungsi init() untuk menginisialisasi linked list dengan menetapkan head dan tail ke nullptr, menandakan bahwa list kosong.
4. Membuat fungsi isEmpty() untuk mengembalikan true jika head adalah nullptr, menandakan bahwa list kosong.

```
// 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;
}
```

5. Membuat fungsi insertDepan(const mahasiswa &data) untuk menambahkan node baru di belakang linked list. Jika list kosong, head dan tail akan menunjuk ke node baru. Jika tidak, tail->next akan diatur ke node baru, dan tail akan diperbarui.
6. Membuat fungsi insertBelakang(const mahasiswa &data) untuk menambahkan node baru di belakang linked list. Jika list kosong, head dan tail akan menunjuk ke node baru. Jika tidak, tail->next akan diatur ke node baru, dan tail akan diperbarui.

```
// 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;
    }
}
```

7. Membuat fungsi hitungList() untuk menghitung dan mengembalikan jumlah node dalam linked list dengan menelusuri dari head hingga nullptr.
8. hapusDepan() untuk menghapus node dari depan linked list. Jika list tidak kosong, node head dihapus, dan head diperbarui ke node berikutnya. Jika setelah penghapusan list menjadi kosong, tail juga diatur ke nullptr.

```
// 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;
    }
}
```

9. hapusBelakang() untuk menghapus node dari belakang linked list. Jika list hanya memiliki satu elemen, node dihapus, dan head serta tail diatur ke nullptr. Jika tidak, list ditelusuri hingga node sebelum tail untuk menghapus tail.

```
// 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;
    }
}
```

10. Membuat fungsi tampil() untuk menampilkan semua data mahasiswa

dalam linked list. Jika list kosong, menampilkan pesan bahwa list kosong.

11. Membuat fungsi clearList() untuk menghapus seluruh elemen dalam linked list dan mengatur head dan tail ke nullptr.

```
// 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;
    }
    cout << 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;
}
```

12. Fungsi main() menginisialisasi linked list, kemudian menambahkan beberapa data mahasiswa (m1, m2, m3) menggunakan fungsi insertDepan dan insertBelakang, serta menampilkan isi linked list setelah setiap penambahan, kemudian beberapa elemen dihapus menggunakan hapusDepan dan hapusBelakang, dan isi list ditampilkan kembali setelah setiap penghapusan, dan seluruh list dihapus menggunakan clearList.

```
// 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;
}
```

B. Guided 2

1. Membuat struktur node dengan int data untuk nilai dari elemen list dan node* next adalah pointer yang menunjuk ke node berikutnya.
2. Node* alokasi(int value) adalah fungsi yang digunakan untuk untuk mengalokasikan memori untuk sebuah node baru dan fungsi ini menerima parameter value yang akan disimpan dalam node.
3. void dealokasi(Node* node) digunakan untuk menghapus node dan melepaskan memori yang digunakan oleh node tersebut dengan delete.
4. bool isListEmpty(Node* head) digunakan untuk mengecek apakah list kosong jika head adalah nullptr, maka list kosong dan fungsi ini mengembalikan true dan sebaliknya, jika ada elemen dalam list, maka mengembalikan false.

```
// Definisi struktur untuk elemen list
struct Node {
    int data;       // Menyimpan nilai elemen
    Node* next;     // Pointer ke elemen berikutnya
};

// Fungsi untuk mengalokasikan memori untuk node baru
Node* alokasi(int value) {
    Node* newNode = new Node; // Alokasi memori untuk elemen baru
    if (newNode != nullptr) { // Jika alokasi berhasil
        newNode->data = value; // Mengisi data node
        newNode->next = nullptr; // Set next ke nullptr
    }
    return newNode; // Mengembalikan pointer node baru
}

// Fungsi untuk dealokasi memori node
void dealokasi(Node* node) {
    delete node; // Mengembalikan memori yang digunakan oleh node
}

// Pengecekan apakah list kosong
bool isListEmpty(Node* head) {
    return head == nullptr; // List kosong jika head adalah nullptr
}
```

5. void insertFirst(Node* &head, int value) digunakan untuk menambahkan node baru dengan value di awal list.
6. void insertLast(Node* &head, int value) digunakan untuk menambahkan node baru di akhir list.

```
// Menambahkan elemen di awal list
void insertFirst(Node* &head, int value) {
    Node* newNode = alokasi(value); // Alokasi memori untuk elemen baru
    if (newNode != nullptr) {
        newNode->next = head;     // Menghubungkan elemen baru ke elemen pertama
        head = newNode;           // Menetapkan elemen baru sebagai elemen pertama
    }
}

// Menambahkan elemen di akhir list
void insertLast(Node* &head, int value) {
    Node* newNode = alokasi(value); // Alokasi memori untuk elemen baru
    if (newNode != nullptr) {
        if (isListEmpty(head)) {  // Jika list kosong
            head = newNode;       // Elemen baru menjadi elemen pertama
        } else {
            Node* temp = head;
            while (temp->next != nullptr) { // Mencari elemen terakhir
                temp = temp->next;
            }
            temp->next = newNode; // Menambahkan elemen baru di akhir list
        }
    }
}
```

7. void printList(Node* head) digunakan untuk menampilkan semua elemen dalam linked list.
8. int countElements(Node* head) digunakan untuk menghitung jumlah elemen dalam linked list.

```
// Menampilkan semua elemen dalam list
void printList(Node* head) {
    if (isListEmpty(head)) {
        cout << "List kosong!" << endl;
    } else {
        Node* temp = head;
        while (temp != nullptr) {  // Selama belum mencapai akhir list
            cout << temp->data << " "; // Menampilkan data elemen
            temp = temp->next; // Melanjutkan ke elemen berikutnya
        }
        cout << endl;
    }
}

// Menghitung jumlah elemen dalam list
int countElements(Node* head) {
    int count = 0;
    Node* temp = head;
    while (temp != nullptr) {
        count++;           // Menambah jumlah elemen
        temp = temp->next; // Melanjutkan ke elemen berikutnya
    }
    return count;          // Mengembalikan jumlah elemen
}
```

9. void clearList(Node* &head) digunakan untuk menghapus semua elemen dalam linked list.

```
// Menghapus semua elemen dalam list dan dealokasi memori
void clearList(Node* &head) {
    while (head != nullptr) {
        Node* temp = head;  // Simpan pointer ke node saat ini
        head = head->next;  // Pindahkan ke node berikutnya
        dealokasi(temp);    // Dealokasi node
    }
}
```

10. Membuat fungsi main() menginisialisasi linked list dimulai dari menginisialisasi list kosong dengan head yang menunjuk ke nullptr, menambahkan elemen ke list, menampilkan isi list, menampilkan jumlah elemen, menghapus semua elemen, dan menampilkan isi list setelah penghapusan.

```cpp
int main() {
    Node* head = nullptr;  // Membuat list kosong

    // Menambahkan elemen ke dalam list
    insertFirst(head, 10); // Menambahkan elemen 10 di awal list
    insertLast(head, 20);  // Menambahkan elemen 20 di akhir list
    insertLast(head, 30);  // Menambahkan elemen 30 di akhir list

    // Menampilkan isi list
    cout << "Isi List: ";
    printList(head);

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

    // Menghapus semua elemen dalam list
    clearList(head);

    // Menampilkan isi list setelah penghapusan
    cout << "Isi List setelah penghapusan: ";
    printList(head);

    return 0;
}
```

4. Unguided

A. Unguided 1

1. Membuat struktur node dengan dua elemen yaitu data yang menyimpan nilai dari node dalam int dan next adalah pointer yang merujuk ke node berikutnya dalam linked list.
2. Membuat fungsi createNode untuk membuat node baru dengan alokasi memori dinamis yang mengisi elemen data dari node tersebut dengan nilai yang diterima sebagai parameter (value), lalu mengatur next menjadi nullptr karena node baru belum terhubung dengan node lain, dan mengembalikan pointer ke node baru yang telah dibuat.
3. Membuat fungsi insertFront dengan membuat node baru menggunakan fungsi createNode, set pointer next dari node baru menunjuk ke node yang saat ini menjadi head, dan set head menjadi node baru sehingga node baru ini menjadi node pertama dalam linked list.
4. Membuat fungsi insertBack dengan membuat node baru menggunakan fungsi createNode, jika linked list kosong (head == nullptr) maka node baru menjadi node pertama dengan langsung

mengatur head ke node baru, jika linked list tidak kosong maka cari node terakhir dengan menggunakan pointer temp dan traversing melalui linked list (menggunakan while (temp->next != nullptr)), dan setelah node terakhir ditemukan, tautkan node baru tersebut dengan mengatur next dari node terakhir ke node baru.

```
struct Node {
    int data;
    Node* next;
};
Node* createNode(int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;
    return newNode;
}
void insertFront(Node*& head, int value) {
    Node* newNode = createNode(value);
    newNode->next = head;
    head = newNode;
}
void insertBack(Node*& head, int value) {
    Node* newNode = createNode(value);
    if (head == nullptr) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
```

5. Membuat fungsi printList menggunakan pointer temp yang awalnya diset ke head, kemudian traversing melalui linked list dan setiap elemen data dari node dicetak. Jika masih ada node selanjutnya (temp->next != nullptr), cetak simbol " -> " sebagai pemisah.
6. Membuat fungsi main dengan head adalah pointer ke node pertama dari linked list dan diinisialisasi dengan nullptr (linked list awalnya kosong) dan melakukan ketiga operasi tersebut diikuti dengan fungsi untuk menampilkan hasil dari operasi tersebut.

```
void printList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data;
        if (temp->next != nullptr) {
            cout << " -> ";
        }
        temp = temp->next;
    }
    cout << endl;
}
int main() {
    Node* head = nullptr;
    insertFront(head, 10);
    insertBack(head, 20);
    insertFront(head, 5);
    cout << "Linked List: ";
    printList(head);
    return 0;
}
```

7. Berikut merupakan output dari program tersebut.

```
Linked List: 5 -> 10 -> 20
```

B. Unguided 2

1. Membuat struktur node dengan dua elemen yaitu data yang menyimpan nilai dari node dalam int dan next adalah pointer yang merujuk ke node berikutnya dalam linked list.
2. Membuat fungsi createNode untuk membuat node baru dengan alokasi memori dinamis yang mengisi elemen data dari node tersebut dengan nilai yang diterima sebagai parameter (value), lalu mengatur next menjadi nullptr karena node baru belum terhubung dengan node lain, dan mengembalikan pointer ke node baru yang telah dibuat.
3. Membuat fungsi insertFront dengan membuat node baru menggunakan fungsi createNode, set pointer next dari node baru menunjuk ke node yang saat ini menjadi head, dan set head menjadi node baru sehingga node baru ini menjadi node pertama dalam linked list.
4. Membuat fungsi insertBack dengan membuat node baru

menggunakan fungsi createNode, jika linked list kosong (head == nullptr) maka node baru menjadi node pertama dengan langsung mengatur head ke node baru, jika linked list tidak kosong maka cari node terakhir dengan menggunakan pointer temp dan traversing melalui linked list (menggunakan while (temp->next != nullptr)), dan setelah node terakhir ditemukan, tautkan node baru tersebut dengan mengatur next dari node terakhir ke node baru.

```
struct Node {
    int data;
    Node* next;
};
Node* createNode(int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;
    return newNode;
}
void insertFront(Node*& head, int value) {
    Node* newNode = createNode(value);
    newNode->next = head;
    head = newNode;
}
void insertBack(Node*& head, int value) {
    Node* newNode = createNode(value);
    if (head == nullptr) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
```

5. Membuat fungsi deleteNode dengan suatu nilai pada value
6. Menggunakan if else untuk jika linked list kosong (head == nullptr), tampilkan pesan bahwa tidak ada node yang bisa dihapus dan jika node yang ingin dihapus adalah node pertama (head), maka ubah head menjadi node berikutnya (head = head->next) dan hapus node

pertama, lalu jika node yang ingin dihapus bukan node pertama, traversing linked list menggunakan pointer current untuk mencari node dengan nilai yang sesuai dan Setelah node ditemukan, unlink node tersebut dengan mengubah pointer previous->next sehingga melewati node yang akan dihapus diikuti hapus node tersebut dengan delete current.

```
void deleteNode(Node*& head, int value) {
    if (head == nullptr) {
        cout << "List is empty, no node to delete.\n";
        return;
    }
    if (head->data == value) {
        Node* temp = head;
        head = head->next;
        delete temp;
        cout << "Node with value " << value << " deleted from the list.\n";
        return;
    }
    Node* current = head;
    Node* previous = nullptr;

    while (current != nullptr && current->data != value) {
        previous = current;
        current = current->next;
    }
    if (current == nullptr) {
        cout << "Node with value " << value << " not found in the list.\n";
        return;
    }
    previous->next = current->next;
    delete current;
    cout << "Node with value " << value << " deleted from the list.\n";
}
```

7. Membuat fungsi printList untuk mencetak semua elemen pada linked list, menggunakan if else untuk jika linked list kosong (head == nullptr) maka tampilkan pesan bahwa list kosong dan jika tidak kosong, traversing melalui node-node dalam linked list dan mencetak nilai dari setiap node. Jika masih ada node berikutnya, tampilkan separator " -> " di antara nilai-nilai node.
8. Membuat fungsi main dengan linked list dimulai dengan head = nullptr, kemudian melakukan tiga operasi awal, lalu menampilkan linked list sebelum penghapusan, melakukan penghapusan node, dan menampilkan hasil setelah penghapusan.

```
void printList(Node* head) {
    if (head == nullptr) {
        cout << "The list is empty.\n";
        return;
    }
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data;
        if (temp->next != nullptr) {
            cout << " -> ";
        }
        temp = temp->next;
    }
    cout << endl;
}
int main() {
    Node* head = nullptr;
    insertFront(head, 10);
    insertFront(head, 5);
    insertBack(head, 20);

    cout << "Linked List before deletion: ";
    printList(head);
    deleteNode(head, 10);
    cout << "Linked List after deletion: ";
    printList(head);
    return 0;
}
```

9. Berikut merupakan output dari program tersebut.

```
Linked List before deletion: 5 -> 10 -> 20
Node with value 10 deleted from the list.
Linked List after deletion: 5 -> 20
```

C. Unguided 3

1. Membuat struktur node dengan dua elemen yaitu data yang menyimpan nilai dari node dalam int dan next adalah pointer yang merujuk ke node berikutnya dalam linked list.
2. Membuat fungsi createNode untuk membuat node baru dengan alokasi memori dinamis yang mengisi elemen data dari node tersebut dengan nilai yang diterima sebagai parameter (value), lalu mengatur next menjadi nullptr karena node baru belum terhubung dengan node

lain, dan mengembalikan pointer ke node baru yang telah dibuat.

3. Membuat fungsi insertFront dengan membuat node baru menggunakan fungsi createNode, set pointer next dari node baru menunjuk ke node yang saat ini menjadi head, dan set head menjadi node baru sehingga node baru ini menjadi node pertama dalam linked list.
4. Membuat fungsi insertBack dengan membuat node baru menggunakan fungsi createNode, jika linked list kosong (head == nullptr) maka node baru menjadi node pertama dengan langsung mengatur head ke node baru, jika linked list tidak kosong maka cari node terakhir dengan menggunakan pointer temp dan traversing melalui linked list (menggunakan while (temp->next != nullptr)), dan setelah node terakhir ditemukan, tautkan node baru tersebut dengan mengatur next dari node terakhir ke node baru.

```
struct Node {
    int data;
    Node* next;
};
Node* createNode(int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;
    return newNode;
}
void insertFront(Node*& head, int value) {
    Node* newNode = createNode(value);
    newNode->next = head;
    head = newNode;
}
void insertBack(Node*& head, int value) {
    Node* newNode = createNode(value);
    if (head == nullptr) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
```

5. Membuat fungsi searchNode menelusuri setiap node dari head hingga node terakhir. Jika nilai ditemukan maka fungsi akan mengembalikan true dan jika tidak ditemukan hingga akhir list maka fungsi mengembalikan false.
6. Membuat fungsi lengthOfList dengan cara menghitung jumlah node (atau panjang) dari linked list dengan menelusuri dari head hingga node terakhir dan setiap kali menelusuri sebuah node, penghitung (count) ditambah 1.

```
bool searchNode(Node* head, int value) {
    Node* temp = head;
    while (temp != nullptr) {
        if (temp->data == value) {
            return true;
        }
        temp = temp->next;
    }
    return false;
}
int lengthOfList(Node* head) {
    int count = 0;
    Node* temp = head;
    while (temp != nullptr) {
        count++;
        temp = temp->next;
    }
    return count;
}
```

7. Membuat fungsi printList untuk mencetak semua elemen pada linked list, menggunakan if else untuk jika linked list kosong (head == nullptr) maka tampilkan pesan bahwa list kosong dan jika tidak kosong, traversing melalui node-node dalam linked list dan mencetak nilai dari setiap node. Jika masih ada node berikutnya, tampilkan separator " -> " di antara nilai-nilai node.

```
void printList(Node* head) {
    if (head == nullptr) {
        cout << "The list is empty.\n";
        return;
    }
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data;
        if (temp->next != nullptr) {
            cout << " -> ";
        }
        temp = temp->next;
    }
    cout << endl;
}
```

8. Membuat fungsi main dengan head adalah pointer ke node pertama dari linked list dan diinisialisasi dengan nullptr (linked list awalnya kosong) dan melakukan tiga operasi dan mencetak linked list.

9. Memanggil fungsi earchNode(head, 20) untuk mencari node dengan nilai 20 dan jika ditemukan maka program mencetak pesan bahwa node ditemukan.
10. Memanggil fungsi lengthOfList(head) menghitung panjang linked list dan menampilkan hasilnya.

```
int main() {
    Node* head = nullptr;
    insertFront(head, 10);
    insertBack(head, 20);
    insertFront(head, 5);

    cout << "Linked List: ";
    printList(head);

    int searchValue = 20;
    if (searchNode(head, searchValue)) {
        cout << "Node dengan nilai " << searchValue << " ditemukan." << endl;
    } else {
        cout << "Node dengan nilai " << searchValue << " tidak ditemukan." << endl;
    }
    int length = lengthOfList(head);
    cout << "Panjang linked list: " << length << endl;
    return 0;
}
```

11. Berikut merupakan output dari program tersebut.

```
Linked List: 5 -> 10 -> 20
Node dengan nilai 20 ditemukan.
Panjang linked list: 3
```

5. Kesimpulan

Dalam praktikum ini, telah dipelajari cara kerja single linked list dengan operasi dasar seperti insert, delete, dan dealokasi memori menggunakan bahasa pemrograman C++. Linked list memungkinkan penambahan dan penghapusan elemen secara dinamis, baik di awal maupun di akhir list, dengan keunggulan fleksibilitas dalam manajemen data. Namun, karena setiap node hanya menunjuk ke node berikutnya, pencarian dan penghapusan elemen memerlukan penelusuran dari awal list, yang bisa memakan waktu. Penting juga untuk melakukan dealokasi memori agar terhindar dari kebocoran memori, sehingga penggunaan memori tetap efisien.