LAPORAN PRAKTIKUM
Modul 4.
SINGLE 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. 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

2. Landasan Teori

- ***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. ADT 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).

- ***Single Linked List***

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

  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.

  Contoh deklarasi struktur data single linked list:

```
/*file : list.h*/
#ifndef LIST_H_INCLUDED
#define LIST_H_INCLUDED

#define Nil NULL
#define info(P) (P)->info
#define next(P) (P)->next
#define first(L) ((L).first)
using namespace std;
/*deklarasi record dan struktur data list*/
typedef int infotype;
typdef struct elmlist *address;
struct elmlist {
   infotype info;
   address next;
};

struct list{
   address first;
};
#endif // TEST_H_INCLUDED
```

  Misal untuk data mahasiswa yang terdiri dari nama dan nim.

```
/*file : list.h*/
#ifndef LIST_H_INCLUDED
#define LIST_H_INCLUDED

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

using namespace std;
/*deklarasi record dan struktur data list*/
struct mahasiswa{
   char nama[30]
   char nim[10]
}
typedef mahasiswa infotype;

typdef struct elmlist *address;
struct elmlist {
   infotype info;
   address next;
```

```
};

struct list{
   address first;
};
#endif // TEST_H_INCLUDED
```

- ***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()`.

- ***Insert***

A. Insert First

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

```
* 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.

C. Insert After

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

- ***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.

```
/*file : list .h*/
/* contoh ADT list berkait dengan representasi fisik pointer*/
/* representasi address dengan pointer*/
/* info tipe adalah integer */
#ifndef list_H
#define list_H
#include "boolean.h"
#include <stdio.h>
#define Nil NULL
#define info(P) (P)->info
#define next(P) (P)->next
#define first(L) ((L).first)

/*deklarasi record dan struktur data list*/
typedef int infotype;
typedef struct elmlist *address;
struct elmlist{
 infotype info;
 address next;
};

/* definisi list : */
/* list kosong jika First(L)=Nil */
/* setiap elemen address P dapat diacu info(P) atau next(P) */
struct list {
 address first;
};
/********* pengecekan apakah list kosong ***********/
boolean ListEmpty(list L);
/*mengembalikan nilai true jika list kosong*/

/********* pembuatan list kosong *********/
void CreateList(list &L);
/* 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
```

3. Guided

1. Guided 1

Code:

```
#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", "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;
}
```

Output:

```
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!

Process returned 0 (0x0)   execution time : 0.291 s
Press any key to continue.
```

2. Guided 2

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

// 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
}

// 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
        }
    }
}

// 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
}

// 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
    }
}

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;
}
```

Output:

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

// 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
}

// 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
        }
    }
}

// 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
}

// 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
    }
}

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;
}
```

Output:

```
Isi List: 10 20 30
Jumlah elemen: 3
Isi List setelah penghapusan: List kosong!

Process returned 0 (0x0)   execution time : 0.064 s
Press any key to continue.
```

4. Unguided

1. Unguided 1

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

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;
    }
}
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;
}
```

Output:

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

Process returned 0 (0x0)   execution time : 0.064 s
Press any key to continue.
```

2. Unguided 2

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

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;
    }
}
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";
}
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;
}
```

Output:

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

Process returned 0 (0x0)   execution time : 0.060 s
Press any key to continue.
```

3. Unguided 3

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

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;
    }
}
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;
}
```

```
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);
    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;
}
```

Output:

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

Process returned 0 (0x0)   execution time : 0.080 s
Press any key to continue.
```

5. Kesimpulan

Linked List merupakan struktur data dinamis berbasis pointer yang terdiri dari serangkaian elemen (node) yang saling terhubung. Setiap node dalam linked list menyimpan data dan pointer yang menunjuk ke elemen berikutnya. Salah satu keunggulan utama linked list dibandingkan array adalah sifatnya yang fleksibel karena mampu menambah atau mengurangi elemen secara dinamis tanpa perlu mengalokasikan ukuran memori sejak awal. Hal ini membuat linked list ideal untuk situasi di mana jumlah data tidak diketahui atau bisa berubah sepanjang eksekusi program.
Single Linked List, sebagai salah satu varian linked list, hanya memiliki pointer satu arah, di mana setiap node menunjuk ke elemen selanjutnya dan node terakhir mengarah ke `Nil` (kosong).
Operasi dasar pada linked list meliputi penciptaan list, penambahan elemen (di awal, di tengah, atau di akhir), penghapusan elemen, dan penelusuran atau pencetakan elemen. Implementasi linked list menggunakan pointer memberikan fleksibilitas lebih dalam manajemen memori karena memori dapat dialokasikan atau dibebaskan sesuai kebutuhan, menggunakan sintaks seperti `new` dan

`delete` dalam C++.
Penggunaan linked list dengan pointer lebih efisien dan mudah untuk kebutuhan struktur data yang memerlukan fleksibilitas tinggi, seperti stack, queue, tree, atau graph. Meski memiliki beberapa kelebihan, linked list juga memiliki keterbatasan, seperti waktu akses yang lebih lambat dibandingkan array karena harus melakukan traversal secara berurutan. Namun, dalam kasus yang membutuhkan penambahan dan penghapusan elemen secara dinamis, linked list dengan pointer merupakan solusi yang efektif dan optimal.