

**LAPORAN PRAKTIKUM**  
**Modul 04**  
**“SINGLE LINKED LIST (BAGIAN PERTAMA)”**



**Disusun Oleh:**  
**Muhammad Ralfi - 2211104054**  
**Kelas S1SE-07-02**

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

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

### **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

### **Landasan Teori**

#### **Linked List dengan Pointer**

Struktur data yang terdiri dari rangkaian elemen yang saling terkait dikenal dengan nama linked list atau list. Karakteristik utama linked list adalah sifatnya yang fleksibel dan dapat menyesuaikan ukuran, tidak seperti array yang memiliki ukuran tetap. Dengan kata lain, jumlah elemen dalam linked list bisa bertambah atau berkurang sesuai keperluan. Dalam linked list, data yang disimpan bisa dibedakan menjadi dua jenis. Pertama, data tunggal yang hanya terdiri dari satu variabel, contohnya seperti nama yang bertipe string. Kedua, data majemuk yang merupakan gabungan dari beberapa variabel dengan tipe berbeda-beda, misalnya data mahasiswa yang memuat informasi nama (string), NIM (long integer), dan alamat (string).

Dalam pengimplementasian linked list, terdapat dua pilihan yaitu menggunakan Array atau Pointer. Pointer dipilih sebagai implementasi yang lebih baik karena beberapa keunggulan. Pertama, pointer memiliki sifat dinamis berbeda dengan array yang statis. Kedua, struktur data linked list yang saling terhubung lebih sesuai dengan konsep pointer. Ketiga, fleksibilitas linked list sangat cocok dengan karakteristik pointer yang dapat disesuaikan. Keempat, penanganan linked list menggunakan pointer lebih sederhana dibandingkan dengan array. Kelima, array lebih tepat digunakan untuk data yang jumlah maksimum elemennya sudah diketahui sejak awal.

Untuk mengakses elemen dalam linked list yang menggunakan pointer, dapat digunakan operator panah (->) atau titik (.). Terdapat delapan model ADT (Abstract Data Type) Linked list yang dipelajari, yaitu:

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 tersebut memiliki ciri khas tersendiri dan penggunaannya dapat disesuaikan dengan kebutuhan spesifik.

Operasi pada Linked List :

1. **Penciptaan dan inisialisasi list:** Proses menciptakan list baru (create list).
2. **Penyisipan elemen list:** Menambahkan elemen baru ke dalam list (insert).

3. **Penghapusan elemen list:** Menghapus elemen dari list (delete).
4. **Penelusuran elemen list:** Menelusuri dan menampilkan elemen dalam list (view).
5. **Pencarian elemen list:** Mencari elemen tertentu dalam list (search).
6. **Pengubahan elemen list:** Mengubah data dalam elemen list (update).

### Single Linked List

Single Linked List merupakan sebuah struktur data yang memiliki karakteristik pointer satu arah, dimana tiap elemen hanya dapat menunjuk ke elemen selanjutnya. Dalam Single Linked List terdapat dua komponen utama:

1. Data, yang merupakan informasi yang disimpan dalam setiap elemen list
2. Suksesor (next), yaitu pointer yang berfungsi sebagai penunjuk ke elemen berikutnya dalam list

Ada beberapa istilah penting yang digunakan dalam Single Linked List:

1. First atau head, yaitu pointer yang mengarah ke elemen pertama dalam list
2. Next, merupakan pointer pada suatu elemen yang mengarah ke elemen selanjutnya
3. Null atau nil, yang menandakan akhir dari linked list atau mengindikasikan elemen yang kosong

### Contoh deklarasi 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;

typedef int infotype;
typedef struct elmlist *address;

struct elmlist {
    infotype info;
    address next;
};

struct list {
    address first;
};

#endif // LIST_H_INCLUDED
```

### Deklarasi 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;

struct mahasiswa {
    char nama[30];
    char nim[10];
};

typedef mahasiswa infotype;
typedef struct elmlist *address;

struct elmlist {
    infotype info;
    address next;
};

struct list {
    address first;
};

#endif // LIST_H_INCLUDED
```

### Pembentukan List

- **Pembentukan list:** Fungsi createList() digunakan untuk menciptakan list baru dan menginisialisasi pointer first ke Nil.
- **Pengalokasian memori:** Setiap elemen baru yang ditambahkan harus dialokasikan memori menggunakan malloc di C atau new di C++.

### Sintaks Pengalokasian Memori

```
P = new elmlist;
```

**Dealokasi Memori:** Untuk menghapus elemen dan melepaskan memori yang digunakan:

```
delete P;
```

### Insert

1. **Insert First:** Menambahkan elemen baru di awal list

```
void insertFirst(list &L, address P) {
    next(P) = first(L);
    first(L) = P;
}
```

2. **Insert Last:** Menambahkan elemen baru di akhir list.

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

```
        first(L) = P;
    } else {
        address Q = first(L);
        while (next(Q) != NULL) { // Menelusuri hingga elemen terakhir
            Q = next(Q);
        }
        next(Q) = P; // Menambahkan elemen baru di akhir
    }
    next(P) = NULL; // Pastikan elemen baru menjadi elemen terakhir
}
```

3. **Insert After:** Menambahkan elemen baru setelah elemen tertentu.

```
void insertAfter(List &L, address P, address Prec) {
    if (Prec != NULL) {
        next(P) = next(Prec); // Hubungkan elemen baru dengan
        // elemen setelah Prec
        next(Prec) = P; // Hubungkan Prec dengan elemen baru
    }
}
```

### Operasi View

Menampilkan isi dari seluruh elemen dalam linked list dengan cara menelusuri list dari elemen pertama hingga terakhir.

```
void printInfo(list L) {
    address P = first(L);
    while (P != Nil) {
        cout << info(P) << " ";
        P = next(P);
    }
}
```

### Delete

1. **Delete First :** Merupakan penghapusan elemen dari awal *Linked List*. Langkah-langkah dalam *delete first*

```
/* contoh sintaks delete first */
void deleteFirst(List &L, address &P){
    P = first(L); // menyimpan alamat elemen pertama
    first(L) = next(first(L)); // memindahkan pointer ke elemen
    // berikutnya
    next(P) = Nil; // memutuskan hubungan ke elemen lainnya
}
```

2. **Delete Last :** Merupakan penghapusan elemen dari akhir *Linked List*. Langkah-langkah dalam *delete last*:

```
/* contoh sintaks delete last */
void deletelast(List &L, address &P){
    P = first(L); // menyimpan alamat elemen pertama
    if (next(P) == Nil) {
        first(L) = Nil; // jika hanya satu elemen, hapus langsung
    } else {
        address Q = P;
    }
}
```

```
while (next(next(Q)) != Nil) { // mencari elemen terakhir
    Q = next(Q);
}
P = next(Q); // menyimpan elemen terakhir
next(Q) = Nil; // memutus hubungan ke elemen terakhir
}
```

3. Delete After : Merupakan penghapusan elemen setelah node tertentu. Langkah-langkah dalam *delete after*:

```
/* contoh sintaks delete after */
void deleteAfter(List &L, address &P, address Prec){
    P = next(Prec); // menyimpan elemen setelah Prec
    next(Prec) = next(P); // menyambungkan elemen Prec ke elemen
    setelah P
    next(P) = Nil; // memutuskan hubungan P dengan elemen lainnya
}
```

### Searching

Operasi pencarian pada *Single Linked List* dapat dilakukan dengan menggunakan metode *sequential search*

```
/* contoh sintaks pencarian elemen */
address search(List L, infotype X){
    address P = first(L);
    while (P != Nil && info(P) != X) {
        P = next(P); // menelusuri setiap elemen
    }
    return P; // mengembalikan alamat elemen jika ditemukan, atau Nil jika
    tidak
}
```

### Update

Merupakan proses pengubahan nilai dari suatu elemen yang ada pada *Single Linked List*. Langkah-langkah dalam *update*:

```
/* contoh sintaks update */
void update(List &L, address P, infotype X){
    info(P) = X; // mengubah nilai elemen yang ditunjuk P
}
```

### View

Proses menampilkan seluruh elemen yang ada pada *Single Linked List*. Penelusuran dilakukan dari elemen pertama hingga elemen terakhir.

```
/* contoh sintaks view */
void printInfo(List L){
    address P = first(L);
    while (P != Nil) {
        cout << info(P) << " "; // menampilkan informasi tiap elemen
    }
}
```

```
        P = next(P); // pindah ke elemen berikutnya
    }
    cout << endl;
}
```

## 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"};

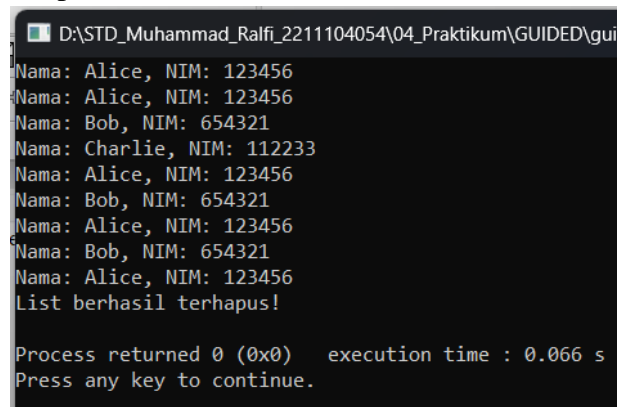
    // Menambahkan mahasiswa ke dalam list
    insertDepan(m1);
    tampil();
    insertBelakang(m2);
    tampil();
    insertDepan(m3);
    tampil();

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

    // Menghapus seluruh list
    clearList();

    return 0;
}
```

### Output:



```
D:\STD_Muhammad_Ralfi_2211104054\04_Praktikum\GUIDED\gui
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 : 0.066 s
Press any key to continue.
```

### 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 isEmpty(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 (isEmpty(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 (isEmpty(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);

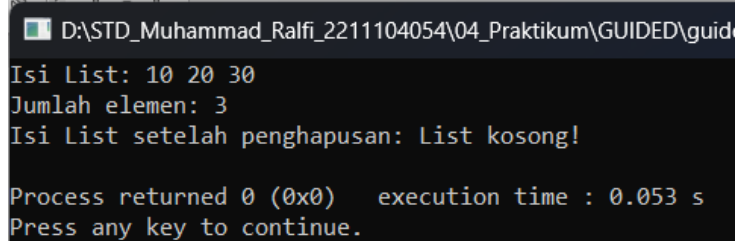
    // Menampilkan jumlah elemen
    cout << "Jumlah elemen: " << countElements(head) << endl;

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

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

    return 0;
}
```

Output:



```
D:\STD_Muhammad_Ralfi_2211104054\04_Praktikum\GUIDED\guid
Isi List: 10 20 30
Jumlah elemen: 3
Isi List setelah penghapusan: List kosong!

Process returned 0 (0x0)   execution time : 0.053 s
Press any key to continue.
```

## Unguided

### 1. Membuat Single Linked List

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

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

#### Contoh input dan output:

Input:

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

Output: 5 -> 10 -> 20

Code:

```
#include <iostream>

using namespace std;

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

void insertAtFront(Node*& head, int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = head;
    head = newNode;
}

void insertAtBack(Node*& head, int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;

    if (head == nullptr) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
```

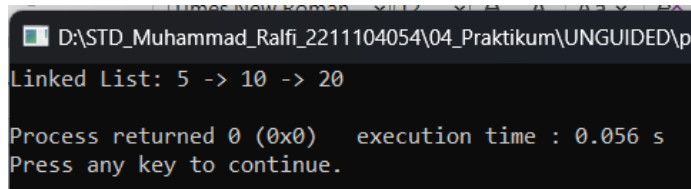
```
void printList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data;
        if (temp->next != nullptr) {
            cout << " -> ";
        }
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    Node* head = nullptr;

    // Contoh operasi
    insertAtFront(head, 10);
    insertAtBack(head, 20);
    insertAtFront(head, 5);

    cout << "Linked List: ";
    printList(head);
    return 0;
}
```

Output :



## 2. Menghapus Node pada Linked List

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

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

**Contoh input/output:**

Input:

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

Output: 5 -> 20

Input :

```
#include <iostream>

using namespace std;

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

void insertAtFront(Node*& head, int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = head;
    head = newNode;
}

void insertAtBack(Node*& head, int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;

    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) return;

    if (head->data == value) {
        Node* temp = head;
        head = head->next;
        delete temp;
        return;
    }

    Node* current = head;
    Node* previous = nullptr;
    while (current != nullptr && current->data != value) {
        previous = current;
```

```
        current = current->next;
    }

    if (current != nullptr) {
        previous->next = current->next;
        delete current;
    }
}

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;

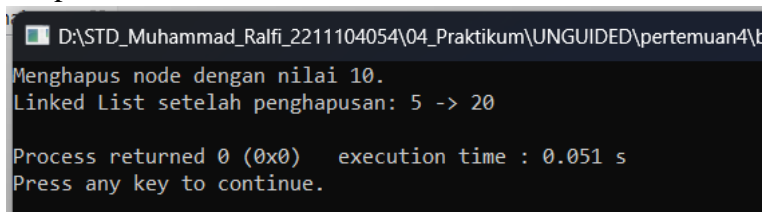
    // Contoh operasi
    insertAtFront(head, 10);
    insertAtBack(head, 20);
    insertAtFront(head, 5);

    cout << "Menghapus node dengan nilai 10." << endl;
    deleteNode(head, 10);

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

    return 0;
}
```

Output :



```
D:\STD_Muhammad_Ralfi_2211104054\04_Praktikum\UNGUIDED\pertemuan4\t
Menghapus node dengan nilai 10.
Linked List setelah penghapusan: 5 -> 20
Process returned 0 (0x0)   execution time : 0.051 s
Press any key to continue.
```

### 3. Mencari dan Menghitung Panjang Linked List

Buatlah program C++ yang dapat melakukan operasi berikut:

- Cari Node dengan Nilai Tertentu: Fungsi untuk mencari apakah sebuah nilai ada di dalam linked list.

- Hitung Panjang Linked List: Fungsi untuk menghitung jumlah node yang ada di dalam linked list.

**Contoh input/output:**

Input:

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

Output:

Node dengan nilai 20 ditemukan

Panjang linked list: 3

Code:

```
// UNGUIDED 3
#include <iostream>

using namespace std;

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

void insertAtFront(Node*& head, int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = head;
    head = newNode;
}

void insertAtBack(Node*& head, int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;

    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* current = head;
    while (current != nullptr) {
        if (current->data == value) {
            return true;
        }
        current = current->next;
    }
    return false;
}

int countLength(Node* head) {
    int count = 0;
    Node* current = head;
    while (current != nullptr) {
        count++;
        current = current->next;
    }
    return count;
}

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;

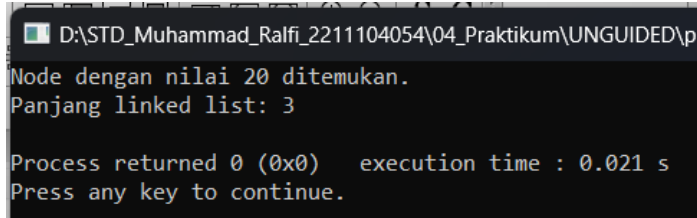
    insertAtFront(head, 10);
    insertAtBack(head, 20);
    insertAtFront(head, 5);

    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 = countLength(head);
}
```

```
cout << "Panjang linked list: " << length << endl;  
  
return 0;  
}
```

Output :



```
D:\STD_Muhammad_Ralfi_2211104054\04_Praktikum\UNGUIDED\p  
Node dengan nilai 20 ditemukan.  
Panjang linked list: 3  
Process returned 0 (0x0) execution time : 0.021 s  
Press any key to continue.
```

### Kesimpulan

Linked List adalah struktur data yang terdiri dari elemen-elemen yang saling terhubung, dengan karakteristik utama berupa sifatnya yang dinamis dan fleksibel dalam hal ukuran. Data dalam linked list dapat berupa data tunggal (satu variabel) atau data majemuk (beberapa variabel dengan tipe berbeda). Implementasi linked list lebih efektif menggunakan pointer dibandingkan array karena beberapa keunggulan: sifatnya yang dinamis, konsep pointer yang sesuai dengan struktur linked list, fleksibilitas yang tinggi, penanganan yang lebih sederhana, dan kemampuan menyesuaikan ukuran sesuai kebutuhan.

Linked list memiliki delapan model ADT yang berbeda (Single, Double, Circular, Multi, Stack, Queue, Tree, dan Graph), dengan enam operasi dasar yang meliputi penciptaan dan inisialisasi, penyisipan elemen, penghapusan elemen, penelusuran elemen, pencarian elemen, dan pengubahan elemen. Khusus untuk Single Linked List, struktur ini menggunakan pointer satu arah dengan komponen utama berupa data dan suksesor (next). Dalam implementasinya, Single Linked List menggunakan tiga istilah penting yaitu first/head yang berfungsi sebagai penunjuk elemen pertama, next sebagai penunjuk elemen berikutnya, dan null/nil yang berperan sebagai penanda akhir list.