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

Disusun Oleh:
Aji Prasetyo Nugroho - 2211104049
S1SE-07-2

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024

A. 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.

B. Landasan Teori

Linked List Dengan Pointer

Linked list, sering disebut sebagai list, adalah salah satu jenis struktur data yang terdiri dari serangkaian elemen yang saling terhubung. Linked list bersifat fleksibel karena dapat bertambah atau berkurang sesuai kebutuhan. Data yang disimpan dalam linked list bisa berupa data sederhana atau kompleks. Data sederhana hanya berisi satu elemen, seperti variabel nama bertipe string. Sedangkan data kompleks terdiri dari beberapa elemen dengan berbagai tipe, seperti data mahasiswa yang terdiri dari Nama (string), NIM (long integer), dan Alamat (string).

Linked list dapat diimplementasikan menggunakan array atau pointer. Namun, penggunaan pointer lebih sering dipilih karena beberapa alasan:

1. Array bersifat statis, sedangkan pointer bersifat dinamis.
2. Bentuk data dalam linked list saling terhubung, sehingga lebih cocok menggunakan pointer.
3. Fleksibilitas linked list sesuai dengan sifat pointer yang dapat disesuaikan dengan kebutuhan.
4. Penanganan linked list lebih sulit dengan array dibandingkan dengan pointer.
5. Array lebih sesuai digunakan pada kumpulan data yang jumlah elemennya diketahui sejak awal.

Dalam implementasinya, akses ke elemen dalam linked list dengan pointer bisa menggunakan tanda panah (->) atau tanda titik (.).

Beberapa model dari ADT Linked List yang dipelajari meliputi:

1. Single Linked List
2. Double Linked List
3. Circular Linked List
4. Multi Linked List

5. Stack (Tumpukan)
6. Queue (Antrian)
7. Tree (Pohon)
8. Graph (Graf)

Setiap model ADT Linked List memiliki karakteristik khusus yang digunakan sesuai kebutuhan.

Secara umum, operasi dasar pada ADT Linked List mencakup:

1. Membuat dan menginisialisasi list (Create List).
2. Menyisipkan elemen baru (Insert).
3. Menghapus elemen list (Delete).
4. Menelusuri dan menampilkan elemen list (View).
5. Mencari elemen tertentu dalam list (Search).
6. Mengubah isi elemen list (Update).

Single Linked List

Single Linked List adalah model ADT Linked List yang memiliki arah pointer tunggal. Setiap elemen dalam single linked list terdiri dari beberapa komponen:

- Elemen: Segmen data yang membentuk suatu list.
- Data: Informasi utama yang tersimpan dalam sebuah elemen.
- Suksesor: Bagian yang bertindak sebagai penghubung antar elemen dalam list.

Karakteristik dari Single Linked List adalah sebagai berikut:

1. Hanya menggunakan satu pointer.
2. Node terakhir menunjuk ke nilai Null, kecuali pada circular linked list.
3. Arah pembacaan hanya bisa dilakukan maju.
4. Pencarian data dilakukan secara berurutan (sequential) jika data tidak diurutkan.
5. Mempermudah penyisipan atau penghapusan elemen di tengah-tengah list.

Istilah-istilah dalam Single Linked List:

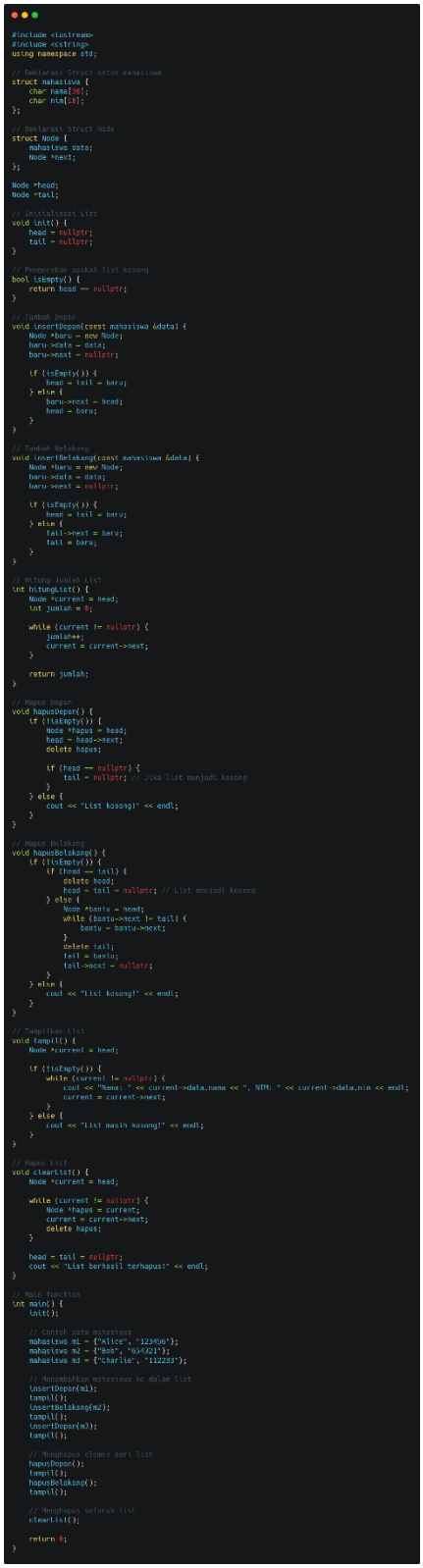
1. First/Head: Pointer yang menunjuk ke elemen pertama dari list.
2. Next: Pointer dalam elemen yang berfungsi sebagai penunjuk ke elemen berikutnya.
3. Null/Nil: Menunjukkan bahwa pointer tidak mengacu ke mana pun, atau elemen

kosong.

4. Node/Simpul/Elemen: Tempat penyimpanan data dalam memori.

Guided

Source Code :



```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", "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!
PS D:\Praktikum STD_2211104049>
```

Source Code :

```cpp
#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!
PS D:\Praktikum STD_2211104049>
```

C. 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

Source Code :

```cpp
#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* createNode(int value) {
    Node* newNode = new Node;  // Alokasi memori untuk elemen baru
    newNode->data = value;     // Mengisi data node
    newNode->next = nullptr;   // Set next ke nullptr
    return newNode;            // Mengembalikan pointer node baru
}

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

// Menambahkan elemen di depan list
void insertAtFront(Node* &head, int value) {
    Node* newNode = createNode(value);  // Membuat node baru
    newNode->next = head;  // Hubungkan elemen baru ke elemen pertama
    head = newNode;        // Set elemen baru sebagai elemen pertama
}

// Menambahkan elemen di belakang list
void insertAtEnd(Node* &head, int value) {
    Node* newNode = createNode(value);  // Membuat node baru
    if (isListEmpty(head)) {  // Jika list kosong
        head = newNode;       // Elemen baru menjadi elemen pertama
    } else {
        Node* temp = head;
        while (temp->next != nullptr) { // Cari elemen terakhir
            temp = temp->next;
        }
        temp->next = newNode;  // Hubungkan elemen baru di akhir list
    }
}

// Menampilkan seluruh 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;         // Tampilkan data elemen
            if (temp->next != nullptr)  // Tambahkan "->" jika masih ada elemen berikutnya
                cout << " -> ";
            temp = temp->next;          // Lanjut ke elemen berikutnya
        }
        cout << endl;
    }
}

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

    // Input operasi
    insertAtFront(head, 10);   // Tambah node di depan (nilai: 10)
    insertAtEnd(head, 20);     // Tambah node di belakang (nilai: 20)
    insertAtFront(head, 5);    // Tambah node di depan (nilai: 5)

    // Cetak linked list
    cout << "Isi Linked List: ";
    printList(head);  // Output: 5 -> 10 -> 20

    return 0;
}
```

Output :

```
Isi Linked List: 5 -> 10 -> 20
PS D:\Praktikum STD_2211104049>
```

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

Source Code :

```cpp
#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* createNode(int value) {
    Node* newNode = new Node;  // Alokasi memori untuk elemen baru
    newNode->data = value;     // Mengisi data node
    newNode->next = nullptr;   // Set next ke nullptr
    return newNode;            // Mengembalikan pointer node baru
}

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

// Menambahkan elemen di depan list
void insertAtFront(Node* &head, int value) {
    Node* newNode = createNode(value);  // Membuat node baru
    newNode->next = head;  // Hubungkan elemen baru ke elemen pertama
    head = newNode;        // Set elemen baru sebagai elemen pertama
}

// Menambahkan elemen di belakang list
void insertAtEnd(Node* &head, int value) {
    Node* newNode = createNode(value);  // Membuat node baru
    if (isListEmpty(head)) {  // Jika list kosong
        head = newNode;       // Elemen baru menjadi elemen pertama
    } else {
        Node* temp = head;
        while (temp->next != nullptr) { // Cari elemen terakhir
            temp = temp->next;
        }
        temp->next = newNode;  // Hubungkan elemen baru di akhir list
    }
}

// Menghapus node dengan nilai tertentu
void deleteNodeByValue(Node* &head, int value) {
    if (isListEmpty(head)) {  // Jika list kosong
        cout << "List kosong! Tidak ada yang bisa dihapus." << endl;
        return;
    }

    // Jika node yang ingin dihapus adalah node pertama
    if (head->data == value) {
        Node* temp = head;    // Simpan pointer ke node yang akan dihapus
        head = head->next;    // Pindahkan head ke node berikutnya
        delete temp;          // Dealokasi memori node yang dihapus
        return;
    }

    // Mencari node yang memiliki nilai yang akan dihapus
    Node* temp = head;
    Node* prev = nullptr;
    while (temp != nullptr && temp->data != value) {
        prev = temp;
        temp = temp->next;
    }

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

    // Hapus node dengan nilai yang ditemukan
    prev->next = temp->next;
    delete temp;  // Dealokasi memori node yang dihapus
}

// Menampilkan seluruh 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;         // Tampilkan data elemen
            if (temp->next != nullptr)  // Tambahkan "->" jika masih ada elemen berikutnya
                cout << " -> ";
            temp = temp->next;          // Lanjut ke elemen berikutnya
        }
        cout << endl;
    }
}

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

    // Input operasi
    insertAtFront(head, 10);   // Tambah node di depan (nilai: 10)
    insertAtEnd(head, 20);     // Tambah node di belakang (nilai: 20)
    insertAtFront(head, 5);    // Tambah node di depan (nilai: 5)

    // Cetak linked list sebelum penghapusan
    cout << "Isi Linked List: ";
    printList(head);  // Output: 5 -> 10 -> 20

    // Hapus node dengan nilai 10
    deleteNodeByValue(head, 10);

    // Cetak linked list setelah penghapusan
    cout << "Isi Linked List setelah penghapusan node dengan nilai 10: ";
    printList(head);  // Output: 5 -> 20

    return 0;
}
```

Output :

```
Isi Linked List: 5 -> 10 -> 20
Isi Linked List setelah penghapusan node dengan nilai 10: 5 -> 20
PS D:\Praktikum STD_2211104049>
```

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

Source Code :

```cpp
#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* createNode(int value) {
    Node* newNode = new Node;  // Alokasi memori untuk elemen baru
    newNode->data = value;     // Mengisi data node
    newNode->next = nullptr;   // Set next ke nullptr
    return newNode;            // Mengembalikan pointer node baru
}

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

// Menambahkan elemen di depan list
void insertAtFront(Node* &head, int value) {
    Node* newNode = createNode(value);  // Membuat node baru
    newNode->next = head;  // Hubungkan elemen baru ke elemen pertama
    head = newNode;        // Set elemen baru sebagai elemen pertama
}

// Menambahkan elemen di belakang list
void insertAtEnd(Node* &head, int value) {
    Node* newNode = createNode(value);  // Membuat node baru
    if (isListEmpty(head)) {  // Jika list kosong
        head = newNode;       // Elemen baru menjadi elemen pertama
    } else {
        Node* temp = head;
        while (temp->next != nullptr) { // Cari elemen terakhir
            temp = temp->next;
        }
        temp->next = newNode;  // Hubungkan elemen baru di akhir list
    }
}

// Fungsi untuk mencari apakah node dengan nilai tertentu ada di linked list
bool searchNode(Node* head, int value) {
    Node* temp = head;
    while (temp != nullptr) {  // Selama belum mencapai akhir list
        if (temp->data == value) {  // Jika nilai ditemukan
            return true;
        }
        temp = temp->next;  // Lanjut ke node berikutnya
    }
    return false;  // Nilai tidak ditemukan
}

// Fungsi untuk menghitung panjang linked list
int countNodes(Node* head) {
    int count = 0;
    Node* temp = head;
    while (temp != nullptr) {  // Selama belum mencapai akhir list
        count++;  // Tambah jumlah node
        temp = temp->next;  // Lanjut ke node berikutnya
    }
    return count;  // Mengembalikan jumlah node
}

// Menampilkan seluruh 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;         // Tampilkan data elemen
            if (temp->next != nullptr)  // Tambahkan "->" jika masih ada elemen berikutnya
                cout << " -> ";
            temp = temp->next;          // Lanjut ke elemen berikutnya
        }
        cout << endl;
    }
}

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

    // Input operasi
    insertAtFront(head, 10);   // Tambah node di depan (nilai: 10)
    insertAtEnd(head, 20);     // Tambah node di belakang (nilai: 20)
    insertAtFront(head, 5);    // Tambah node di depan (nilai: 5)

    // Cetak linked list
    cout << "Isi Linked List: ";
    printList(head);  // Output: 5 -> 10 -> 20

    // Cari node dengan nilai 20
    int valueToSearch = 20;
    if (searchNode(head, valueToSearch)) {
        cout << "Node dengan nilai " << valueToSearch << " ditemukan." << endl;
    } else {
        cout << "Node dengan nilai " << valueToSearch << " tidak ditemukan." << endl;
    }

    // Hitung panjang linked list
    int length = countNodes(head);
    cout << "Panjang linked list: " << length << endl;  // Output: 3

    return 0;
}
```

Output :

```
Isi Linked List: 5 -> 10 -> 20
Node dengan nilai 20 ditemukan.
Panjang linked list: 3
PS D:\Praktikum STD_2211104049>
```