LAPORAN PRAKTIKUM
Modul 04
"SINGLE LINKED LIST (BAGIAN PERTAMA)"

Disusun Oleh:
Ganesha Rahman Gibran -2211104058
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

Liked List dengan Pointer

Linked list (sering disebut sebagai list) adalah salah satu bentuk struktur data yang terdiri dari serangkaian elemen yang saling terhubung. Berbeda dengan array yang memiliki ukuran tetap, linked list bersifat dinamis, yang berarti ukurannya dapat berubah sesuai dengan kebutuhan, baik bertambah maupun berkurang. Data dalam linked list bisa berupa data tunggal atau data majemuk. Data tunggal adalah data yang terdiri dari satu variabel, misalnya nama (string). Data majemuk adalah data yang terdiri dari beberapa variabel dengan tipe yang berbeda, misalnya data mahasiswa yang mencakup nama (string), NIM (long integer), dan alamat (string).

Penggunaan pointer dalam implementasi linked list memiliki beberapa keuntungan dibandingkan array:

1. **Array bersifat statis**: Ukurannya tetap setelah dibuat, sedangkan pointer bersifat dinamis, sehingga lebih fleksibel.
2. **Struktur terhubung**: Linked list terdiri dari elemen-elemen yang saling terhubung, sehingga pointer lebih cocok digunakan dibandingkan array.
3. **Kemudahan pengelolaan memori**: Pointer memungkinkan memori dialokasikan sesuai kebutuhan, tidak seperti array yang memerlukan alokasi memori pada awal
4. **Efisiensi penambahan dan penghapusan elemen**: Penambahan atau penghapusan elemen di tengah linked list lebih mudah dilakukan dengan pointer dibandingkan array

Model Linked List Beberapa model ADT (Abstract Data Type) Linked List yang umum digunakan:

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

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 adalah struktur linked list yang hanya memiliki satu arah pointer, di mana setiap elemen hanya menunjuk ke elemen berikutnya.

Komponen dalam Single Linked List

- **Data**: Informasi utama yang tersimpan dalam elemen list.
- **Suksesor (next)**: Pointer yang menunjuk ke elemen berikutnya dalam list.

Istilah dalam Single Linked List

1. **first/head**: Pointer yang menunjuk elemen pertama dalam list.
2. **next**: Pointer pada elemen yang menunjuk ke elemen berikutnya.
3. **null/nil**: Menunjukkan akhir dari linked list atau elemen 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

```
// 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

Input :

```
#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;
    }
}
```

Telkom University Purwokerto

```cpp
// 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;
    }
}
```

```cpp
// 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;
    }
    cout << endl;
}
```

Telkom University Purwokerto

```
// 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 E:\Struktur Data\2211104058 Ganesha Rahman Gibran SE-06-02>
```

Input :

```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
}
```

```cpp
// 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
        }
    }
}
```

Telkom University Purwokerto

```cpp
// 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
    }
}
```

```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;
}
```

Output :

```
Isi List: 10 20 30
Jumlah elemen: 3
Isi List setelah penghapusan: List kosong!
PS E:\Struktur Data\2211104058_Ganesha_Rahman_Gibran_SE-06-02>
```

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

   Input :

```
#include <iostream>

using namespace std;

// Deklarasi struktur elemen linked list
struct Node {
    int data;        // Data yang disimpan di node
    Node* next;     // Pointer ke node berikutnya
};

// Fungsi untuk menambahkan node di depan linked list
void insertAtFront(Node*& head, int value) {
    Node* newNode = new Node(); // Alokasikan memori untuk node baru
    newNode->data = value;      // Set data
    newNode->next = head;       // Hubungkan node baru dengan head yang ada
    head = newNode;             // Pindahkan head ke node baru
}

// Fungsi untuk menambahkan node di belakang linked list
void insertAtBack(Node*& head, int value) {
    Node* newNode = new Node(); // Alokasikan memori untuk node baru
    newNode->data = value;      // Set data
    newNode->next = nullptr;    // Node baru akan menjadi node terakhir

    if (head == nullptr) {       // Jika list kosong, node baru menjadi head
        head = newNode;
```

```
    } else {
        Node* temp = head;       // Temp untuk menelusuri list
        while (temp->next != nullptr) { // Menelusuri hingga akhir list
            temp = temp->next;
        }
        temp->next = newNode;    // Hubungkan node terakhir dengan node baru
    }
}

// Fungsi untuk mencetak seluruh isi linked list
void printList(Node* head) {
    Node* temp = head;           // Temp untuk menelusuri list
    while (temp != nullptr) {    // Selama belum mencapai akhir list
        cout << temp->data;      // Cetak data node
        if (temp->next != nullptr) {
            cout << " -> ";      // Cetak panah jika bukan node terakhir
        }
        temp = temp->next;       // Pindah ke node berikutnya
    }
    cout << endl;                // Cetak newline setelah selesai
}

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

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

    // Cetak linked list
    cout << "Linked List: ";
    printList(head);              // Menampilkan isi linked list

    return 0;
}
```

Output :

```
o' '--stderr=Microsoft-MIEngine-Error-odcjchn3.mkv' '--pid=Micr
 '--interpreter=mi'
Linked List: 5 -> 10 -> 20
PS E:\Struktur Data\2211104058_Ganesha_Rahman_Gibran_SE-06-02>
```

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;

// Deklarasi struktur elemen linked list
struct Node {
    int data;        // Data yang disimpan di node
    Node* next;     // Pointer ke node berikutnya
};

// Fungsi untuk menambahkan node di depan linked list
void insertAtFront(Node*& head, int value) {
    Node* newNode = new Node(); // Alokasikan memori untuk node baru
    newNode->data = value;      // Set data
    newNode->next = head;       // Hubungkan node baru dengan head yang ada
    head = newNode;             // Pindahkan head ke node baru
}

// Fungsi untuk menambahkan node di belakang linked list
void insertAtBack(Node*& head, int value) {
    Node* newNode = new Node(); // Alokasikan memori untuk node baru
    newNode->data = value;      // Set data
    newNode->next = nullptr;    // Node baru akan menjadi node terakhir

    if (head == nullptr) {       // Jika list kosong, node baru menjadi head
        head = newNode;
    } else {
        Node* temp = head;       // Temp untuk menelusuri list
        while (temp->next != nullptr) { // Menelusuri hingga akhir list
            temp = temp->next;
        }
        temp->next = newNode;    // Hubungkan node terakhir dengan node baru
    }
}

// Fungsi untuk menghapus node dengan nilai tertentu
void deleteNode(Node*& head, int value) {
    if (head == nullptr) return;  // Jika list kosong, tidak ada yang dihapus

    // Jika node yang dihapus adalah head
    if (head->data == value) {
        Node* temp = head;       // Simpan node head yang akan dihapus
```

```
        head = head->next;       // Pindahkan head ke node berikutnya
        delete temp;             // Hapus node
        return;
    }

    // Menelusuri list untuk menemukan node yang akan dihapus
    Node* current = head;
    Node* previous = nullptr;
    while (current != nullptr && current->data != value) {
        previous = current;      // Simpan node sebelumnya
        current = current->next; // Pindah ke node berikutnya
    }

    // Jika node ditemukan
    if (current != nullptr) {
        previous->next = current->next; // Hubungkan node sebelumnya dengan node
setelah current
        delete current;                  // Hapus node
    }
}

// Fungsi untuk mencetak seluruh isi linked list
void printList(Node* head) {
    Node* temp = head;           // Temp untuk menelusuri list
    while (temp != nullptr) {    // Selama belum mencapai akhir list
        cout << temp->data;      // Cetak data node
        if (temp->next != nullptr) {
            cout << " -> ";      // Cetak panah jika bukan node terakhir
        }
        temp = temp->next;       // Pindah ke node berikutnya
    }
    cout << endl;                // Cetak newline setelah selesai
}

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

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

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

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

    return 0;
}
```

Output :

```
Menghapus node dengan nilai 10.
Linked List setelah penghapusan: 5 -> 20
PS E:\Struktur Data\2211104058_Ganesha_Rahman_Gibran_SE-06-02>
```

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

   Input :

```
#include <iostream>

using namespace std;

// Deklarasi struktur elemen linked list
struct Node {
    int data;        // Data yang disimpan di node
    Node* next;     // Pointer ke node berikutnya
};

// Fungsi untuk menambahkan node di depan linked list
void insertAtFront(Node*& head, int value) {
    Node* newNode = new Node(); // Alokasikan memori untuk node baru
    newNode->data = value;      // Set data
    newNode->next = head;       // Hubungkan node baru dengan head yang ada
    head = newNode;             // Pindahkan head ke node baru
}

// Fungsi untuk menambahkan node di belakang linked list
void insertAtBack(Node*& head, int value) {
    Node* newNode = new Node(); // Alokasikan memori untuk node baru
    newNode->data = value;      // Set data
    newNode->next = nullptr;    // Node baru akan menjadi node terakhir

    if (head == nullptr) {       // Jika list kosong, node baru menjadi head
```

```
        head = newNode;
    } else {
        Node* temp = head;       // Temp untuk menelusuri list
        while (temp->next != nullptr) { // Menelusuri hingga akhir list
            temp = temp->next;
        }
        temp->next = newNode;    // Hubungkan node terakhir dengan node baru
    }
}

// Fungsi untuk mencari node dengan nilai tertentu
bool searchNode(Node* head, int value) {
    Node* current = head;  // Menelusuri dari head
    while (current != nullptr) {
        if (current->data == value) {
            return true;    // Jika nilai ditemukan
        }
        current = current->next; // Pindah ke node berikutnya
    }
    return false;           // Jika nilai tidak ditemukan
}

// Fungsi untuk menghitung panjang linked list
int countLength(Node* head) {
    int count = 0;          // Inisialisasi penghitung
    Node* current = head;   // Menelusuri dari head
    while (current != nullptr) {
        count++;            // Tambah penghitung untuk setiap node
        current = current->next; // Pindah ke node berikutnya
    }
    return count;          // Kembalikan panjang linked list
}

// Fungsi untuk mencetak seluruh isi linked list
void printList(Node* head) {
    Node* temp = head;           // Temp untuk menelusuri list
    while (temp != nullptr) {    // Selama belum mencapai akhir list
        cout << temp->data;      // Cetak data node
        if (temp->next != nullptr) {
            cout << " -> ";      // Cetak panah jika bukan node terakhir
        }
        temp = temp->next;       // Pindah ke node berikutnya
    }
    cout << endl;                // Cetak newline setelah selesai
}

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

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

    // Cari node dengan nilai 20
```

```
    int searchValue = 20;
    if (searchNode(head, searchValue)) {
        cout << "Node dengan nilai " << searchValue << " ditemukan." << endl;
    } else {
        cout << "Node dengan nilai " << searchValue << " tidak ditemukan." << endl;
    }

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

    return 0;
}
```

Output :

```
Node dengan nilai 20 ditemukan.
Panjang linked list: 3
PS E:\Struktur Data\2211104058_Ganesha_Rahman_Gibran_SE-06-02>
```

Kesimpulan

Linked list adalah struktur data dinamis yang terdiri dari serangkaian elemen yang saling terhubung, berbeda dengan array yang memiliki ukuran tetap. Penggunaan pointer dalam implementasi linked list memungkinkan fleksibilitas dalam pengelolaan memori, memudahkan penambahan dan penghapusan elemen, serta mendukung efisiensi dalam operasi dasar seperti penyisipan, penghapusan, pencarian, dan pembaruan data. Dalam konteks Single Linked List, setiap elemen terdiri dari data dan pointer ke elemen berikutnya, memudahkan penelusuran dan pengelolaan. Dengan memahami cara kerja linked list dan operasi-operasi dasarnya, kita dapat menciptakan program yang efisien dan mampu menangani data secara dinamis sesuai kebutuhan.