LAPORAN PRAKTIKUM

MODUL 4

SINGLE LINKED LIST (BAGIAN PERTAMA)

Disusun Oleh :

Farhan Kurniawan (2311104073)

Kelas:

SE-07-2

Dosen :

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

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. 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

II. LANDASAN TEORI

Linked list adalah struktur data yang terdiri dari node-node yang saling berhubungan. Setiap node terdiri dari dua bagian: data dan pointer yang menunjuk ke node berikutnya. Penggunaan pointer dalam linked list memungkinkan memori dialokasikan secara dinamis, berbeda dengan array yang memiliki ukuran tetap. Operator seperti dereferensi (*) dan referensi (&) sering digunakan untuk mengakses dan mengelola pointer dalam linked list.

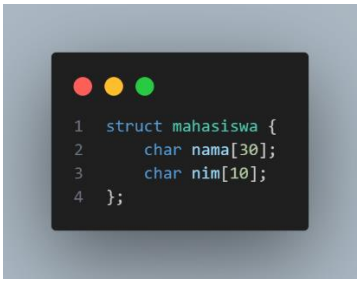
Membuat program menggunakan linked list sering dimulai dengan mendefinisikan prototype dari struktur linked list dan fungsi-fungsi yang diperlukan, seperti fungsi untuk penambahan (insertion), penghapusan (deletion), dan pencarian (searching) node. Implementasi prototipe yang efisien akan membantu dalam mengoptimalkan memori dan kecepatan akses data.

III. GUIDE

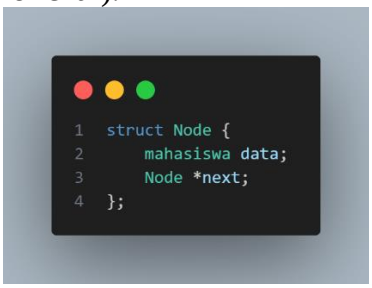
3.1. Linked List (1)

Berikut Implementasi Linked List sederhana menggunakan bahasa C++.Pada kodingan ini, struktur data Linked List digunakan untuk menyimpan data mahasiswa yang terdiri dari nama dan NIM. Mari kita bahas tiap bagiannya secara detail:

3.1.1. Struct Mahasiswa dan Node

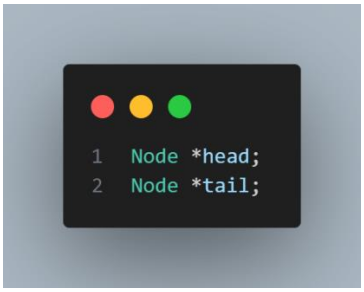


Fungsi ini digunakan untuk menyimpan data mahasiswa, yang terdiri dari dua field: nama (array of char) dan nim (array of char).

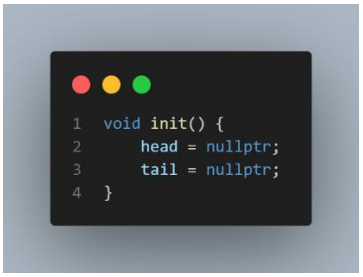


Fungsi ini digunakan untuk mendefinisikan elemen dari Linked List. Setiap Node menyimpan: data: informasi tentang mahasiswa yang bertipe mahasiswa. next: pointer ke node berikutnya di dalam Linked List.

3.1.2. Inisialisasi Linked List



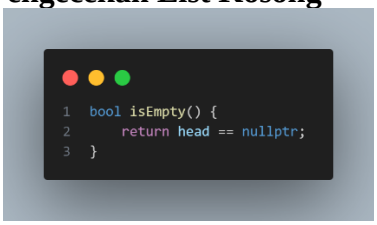
Pointer head digunakan untuk menunjuk node pertama (awal) dari Linked List. Pointer tail digunakan untuk menunjuk node terakhir dari Linked List.



Fungsi init() menginisialisasi Linked List dengan mengatur head dan tail menjadi nullptr, yang berarti list dimulai dalam keadaan kosong.

`

3.1.3. Pengecekan List Kosong



Fungsi isEmpty() memeriksa apakah Linked List kosong dengan mengecek apakah head bernilai nullptr. Jika ya, maka list kosong.

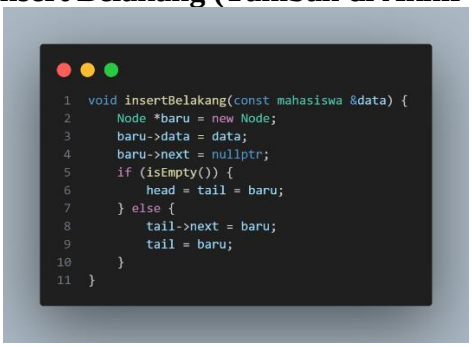
3.1.4. Insert Depan (Tambah di Awal)

Fungsi insertDepan() menambahkan node baru di bagian depan Linked List:

- Membuat node baru baru.
- Jika list kosong, node baru menjadi head dan tail.
- Jika list tidak kosong, node baru ditempatkan di depan, dan

head menunjuk ke node baru.

3.1.5. Insert Belakang (Tambah di Akhir

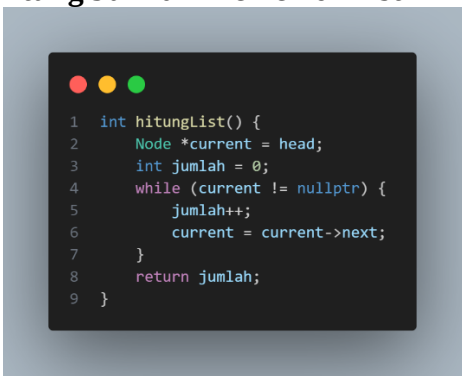


```
void insertBelakang(const mahasiswa &data) {
    Node *baru = new Node;
    baru->data = data;
    baru->next = nullptr;
    if (isEmpty()) {
        head = tail = baru;
    } else {
        tail->next = baru;
        tail = baru;
    }
}
```

Fungsi insertBelakang() menambahkan node baru di bagian akhir Linked List:

- Jika list kosong, node baru menjadi head dan tail.
- Jika list tidak kosong, node baru ditempatkan di belakang, dan tail diperbarui untuk menunjuk ke node baru.

3.1.6. Hitung Jumlah Elemen di List



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

Fungsi hitungList() menghitung jumlah node dalam Linked List dengan melakukan iterasi dari head hingga nullptr.

3.1.7. 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;
    }
}
```

Fungsi hapusDepan() menghapus node di bagian depan Linked List:

- Jika list kosong, pesan "List kosong!" ditampilkan.
- Jika tidak, node pertama dihapus, dan head diperbarui ke node berikutnya.
- Jika setelah penghapusan list menjadi kosong, tail juga diatur ke nullptr.

3.1.8. Hapus Belakang

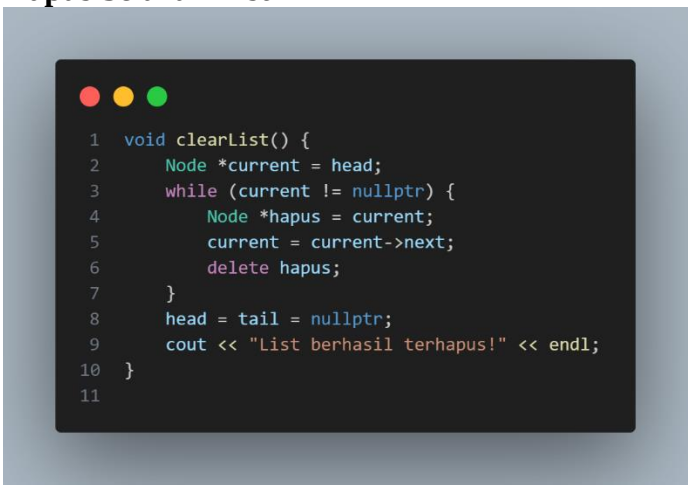
```
void hapusBelakang() {
    if (!isEmpty()) {
        if (head == tail) {
            delete head;
            head = tail = nullptr; // List menjadi kosong
        } else {
            Node *bantu = head;
            while (bantu->next != tail) {
                bantu = bantu->next;
            }
            delete tail;
            tail = bantu;
            tail->next = nullptr;
        }
    } else {
        cout << "List kosong!" << endl;
    }
}
```

3.1.9. 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;
    }
}
```

Fungsi tampil() menampilkan semua elemen dalam Linked List dengan menelusuri dari head hingga nullptr dan mencetak data mahasiswa pada setiap node.

3.1.10. Hapus Seluruh 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;
}
```

Fungsi clearList() menghapus seluruh elemen dalam Linked List, membersihkan memori, dan mengatur head dan tail ke nullptr.

3.1.11. Main Function

```cpp
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;
}
```

Pada main(), fungsi-fungsi yang telah didefinisikan di atas dipanggil untuk:

- Menambah data mahasiswa.
- Menampilkan isi list.
- Menghapus elemen di depan dan di belakang.
- Menghapus seluruh list.

Maka akan menghasilkan output sebagai berikut:

```
PS C:\Users\Farhan Kurniawan> cd "c:\Users\Farhan Kurniawan\Desktop\pemograman\c++\" ; 
uided4 } ; if ($?) { .\Guided4 }
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 C:\Users\Farhan Kurniawan\Desktop\pemograman\c++>
```

3.2. Linked List (2)

contoh implementasi Linked List dasar dalam bahasa C++, di mana operasi seperti menambah elemen di awal dan akhir list, menampilkan elemen, menghitung jumlah elemen, serta menghapus semua elemen dalam list, telah diimplementasikan. Mari kita bahas fungsionalitas dari setiap bagian kode ini secara rinci.

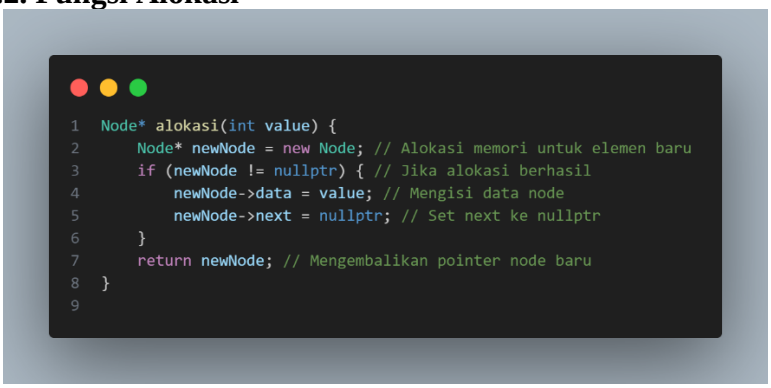
3.2.1. Definisi Struktur Node

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

Struct Node digunakan untuk merepresentasikan setiap elemen dalam Linked List.

- data menyimpan nilai elemen.
- next adalah pointer yang menunjuk ke elemen (node) berikutnya di Linked List.

3.2.2. Fungsi Alokasi



```
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 alokasi() bertanggung jawab untuk mengalokasikan memori untuk node baru.

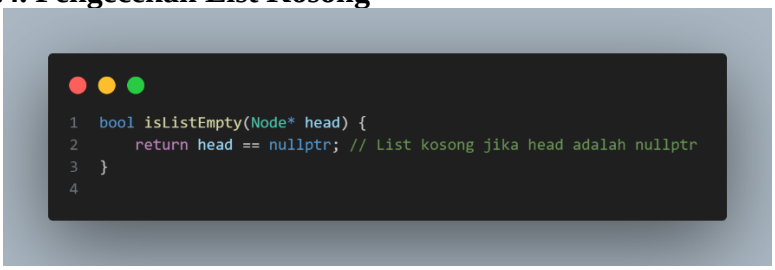
- Menggunakan new untuk membuat node baru.
- Jika alokasi berhasil, fungsi akan mengisi node dengan nilai value yang diberikan dan mengatur pointer next menjadi nullptr.
- Mengembalikan pointer ke node baru.

3.2.3. Fungsi Dealokasi

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

Fungsi dealokasi() bertugas untuk menghapus node dari memori menggunakan delete, melepaskan memori yang telah dialokasikan sebelumnya.

3.2.4. Pengecekan List Kosong



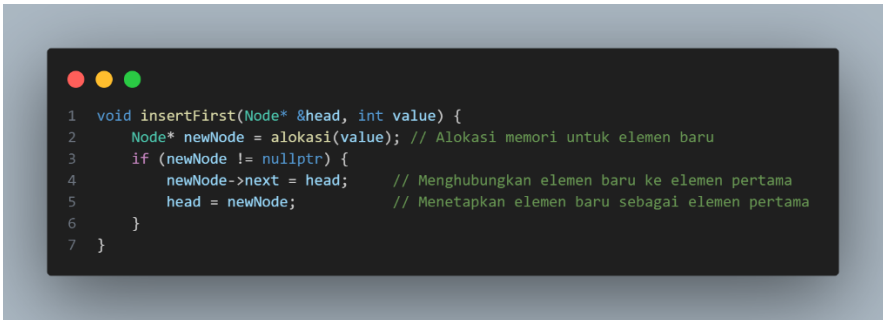
```
bool isListEmpty(Node* head) {
    return head == nullptr; // List kosong jika head adalah nullptr
}

```

Fungsi isListEmpty() memeriksa apakah list kosong dengan mengecek apakah head bernilai nullptr.

- Jika head == nullptr, artinya list kosong, sehingga fungsi mengembalikan nilai true.

3.2.5. Menambah 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
    }
}
```

Fungsi insertFirst() menambahkan node baru di bagian depan Linked List:

- Memanggil fungsi alokasi() untuk membuat node baru.
- Mengatur next dari node baru untuk menunjuk ke node yang saat ini menjadi head.
- head kemudian diperbarui untuk menunjuk ke node baru, sehingga node baru menjadi elemen pertama dalam list.

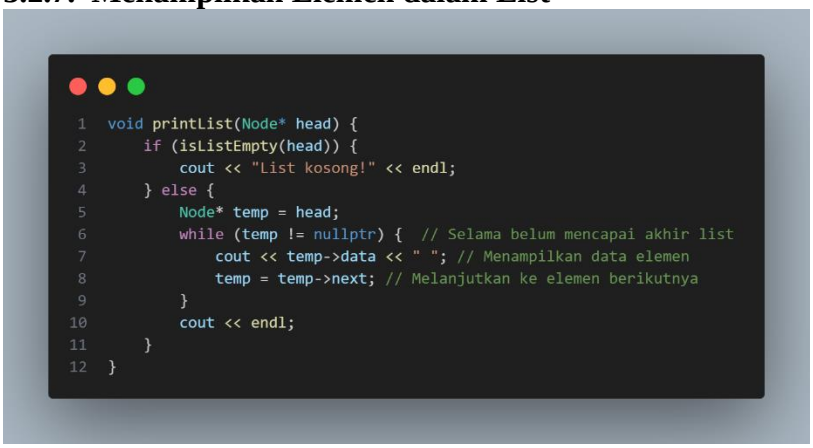
3.2.6. Menambah Elemen di Akhir List

```
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
        }
    }
}
```

Fungsi insertLast() menambahkan node baru di bagian akhir Linked List:

- Jika list kosong (head == nullptr), node baru menjadi head.
- Jika list tidak kosong, fungsi mencari node terakhir (temp->next == nullptr) dan menghubungkan node baru di akhir list.

3.2.7. Menampilkan Elemen dalam List

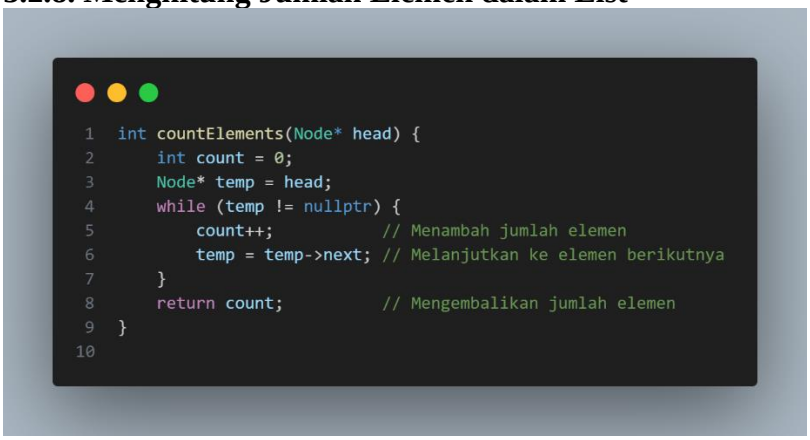


```
void printList(Node* head) {
    if (isListEmpty(head)) {
        cout << "List kosong!" << endl;
    } else {
        Node* temp = head;
        while (temp != nullptr) {  // Selama belum mencapai akhir list
            cout << temp->data << " "; // Menampilkan data elemen
            temp = temp->next;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

Fungsi printList() menampilkan semua elemen dalam Linked List:

- Jika list kosong, pesan "List kosong!" ditampilkan.
- Jika tidak, fungsi melakukan iterasi dari node pertama hingga akhir, mencetak data dari setiap node.

3.2.8. 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
}

```

Fungsi countElements() menghitung jumlah elemen dalam LinkedList:

- Melakukan iterasi melalui seluruh node, menambah counter setiap kali node ditemukan.
- Mengembalikan jumlah total elemen setelah iterasi selesai.

3.2.9. Menghapus Semua Elemen dalam List

```
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
    }
}

```

Fungsi clearList() menghapus semua elemen dalam Linked List:

- Selama list belum kosong, fungsi mengambil node pertama (head), memindahkan head ke node berikutnya, lalu dealokasi node yang dihapus.

3.2.10. Main Function

```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;
}
```

Pada main(), beberapa fungsi yang telah dibuat digunakan untuk melakukan operasi pada Linked List:

- Penambahan elemen: Elemen 10, 20, dan 30 ditambahkan ke dalam list.
- Menampilkan list: Isi list ditampilkan setelah penambahan elemen.
- Menghitung elemen: Jumlah total elemen dihitung dan ditampilkan.
- Menghapus list: Seluruh elemen dalam list dihapus menggunakan clearList(), kemudian list ditampilkan untuk memastikan list kosong.

Maka akan menghasilkan Output sebagai berikut:

```
PS C:\Users\Farhan Kurniawan\Desktop\pemograman\c++> cd
 { g++ guided24.cpp -o guided24 } ; if ($?) { .\guided24
Isi List: 10 20 30
Jumlah elemen: 3
Isi List setelah penghapusan: List kosong!
PS C:\Users\Farhan Kurniawan\Desktop\pemograman\c++>
```

IV. UNGUIDED

1. Input Program:

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

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

// Fungsi untuk menambah node di depan linked list
void insertDepan(Node* &head, int value) {
    Node* newNode = new Node;  // Alokasi memori untuk node baru
    newNode->data = value;     // Mengisi nilai pada node baru
    newNode->next = head;      // Menunjuk ke node pertama sebelumnya
    head = newNode;            // Node baru menjadi head
}

// Fungsi untuk menambah node di belakang linked list
void insertBelakang(Node* &head, int value) {
    Node* newNode = new Node;  // Alokasi memori untuk node baru
    newNode->data = value;     // Mengisi nilai pada node baru
    newNode->next = nullptr;   // Node baru menjadi node terakhir (next = nullptr)

    if (head == nullptr) {     // Jika list kosong
        head = newNode;        // Node baru menjadi node pertama
    } else {
        Node* temp = head;     // Memulai dari head
        while (temp->next != nullptr) {
            temp = temp->next; // Menemukan node terakhir
        }
        temp->next = newNode;  // Menambahkan node baru di akhir
    }
}

// Fungsi untuk mencetak seluruh isi linked list
void cetakList(Node* head) {
    if (head == nullptr) {
        cout << "Linked List kosong!" << endl;
        return;
    }

    Node* temp = head;  // Mulai dari node pertama
    while (temp != nullptr) {
        cout << temp->data;     // Cetak nilai node
        if (temp->next != nullptr) {
            cout << " -> ";     // Cetak panah jika masih ada node berikutnya
        }
        temp = temp->next;      // Pindah ke node berikutnya
    }
    cout << endl;
}

// Fungsi utama
int main() {
    Node* head = nullptr;  // Inisialisasi linked list dengan head sebagai nullptr (kosong)

    // Menambah node berdasarkan contoh input
    insertDepan(head, 10);    // Tambah node di depan dengan nilai 10
    insertBelakang(head, 20); // Tambah node di belakang dengan nilai 20
    insertDepan(head, 5);     // Tambah node di depan dengan nilai 5

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

    return 0;
}
```

Output Program:

```
PS C:\Users\Farhan Kurniawan\Desktop\pemograman\c++\Unguided(4)>
\Unguided(4)\" ; if ($?) { g++ nomor1.cpp -o nomor1 } ; if ($?)
Isi Linked List: 5 -> 10 -> 20
PS C:\Users\Farhan Kurniawan\Desktop\pemograman\c++\Unguided(4)>
```

2. Input Program:

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

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

// Fungsi untuk menambah node di depan linked list
void insertDepan(Node* &head, int value) {
    Node* newNode = new Node;  // Alokasi memori untuk node baru
    newNode->data = value;     // Mengisi nilai pada node baru
    newNode->next = head;      // Menunjuk ke node pertama sebelumnya
    head = newNode;            // Node baru menjadi head
}

// Fungsi untuk menambah node di belakang linked list
void insertBelakang(Node* &head, int value) {
    Node* newNode = new Node;  // Alokasi memori untuk node baru
    newNode->data = value;     // Mengisi nilai pada node baru
    newNode->next = nullptr;   // Node baru menjadi node terakhir (next = nullptr)

    if (head == nullptr) {     // Jika list kosong
        head = newNode;        // Node baru menjadi node pertama
    } else {
        Node* temp = head;     // Memulai dari head
        while (temp->next != nullptr) {
            temp = temp->next; // Menemukan node terakhir
        }
        temp->next = newNode;  // Menambahkan node baru di akhir
    }
}

// Fungsi untuk mencetak seluruh isi linked list
void cetakList(Node* head) {
    if (head == nullptr) {
        cout << "Linked List kosong!" << endl;
        return;
    }

    Node* temp = head;  // Mulai dari node pertama
    while (temp != nullptr) {
        cout << temp->data;     // Cetak nilai node
        if (temp->next != nullptr) {
            cout << " -> ";     // Cetak panah jika masih ada node berikutnya
        }
        temp = temp->next;      // Pindah ke node berikutnya
    }
    cout << endl;
}

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

    Node* temp = head;
    Node* prev = nullptr;

    // Jika node yang akan dihapus adalah node pertama
    if (head->data == value) {
        head = head->next;   // Pindahkan head ke node berikutnya
        delete temp;         // Hapus node pertama
        return;
    }

    // Cari node yang akan dihapus dan simpan pointer ke node sebelumnya
    while (temp != nullptr && temp->data != value) {
        prev = temp;
        temp = temp->next;
    }

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

    // Hapus node yang ditemukan
    prev->next = temp->next;
    delete temp;
}

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

    // Menambah node berdasarkan contoh input
    insertDepan(head, 10);    // Tambah node di depan dengan nilai 10
    insertBelakang(head, 20); // Tambah node di belakang dengan nilai 20
    insertDepan(head, 5);     // Tambah node di depan dengan nilai 5

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

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

    // Mencetak isi linked list setelah penghapusan
    cout << "Isi Linked List setelah penghapusan: ";
    cetakList(head);  // Output: 5 -> 20

    return 0;
}
```

Output Program:

```
PS C:\Users\Farhan Kurniawan\Desktop\pemograman\c++\Unguided(4)>
\Unguided(4)\" ; if ($?) { g++ nomor2.cpp -o nomor2 } ; if ($?) {
Isi Linked List sebelum penghapusan: 5 -> 10 -> 20
Isi Linked List setelah penghapusan: 5 -> 20
PS C:\Users\Farhan Kurniawan\Desktop\pemograman\c++\Unguided(4)>
```

3. Input Program:

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

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

// Fungsi untuk menambah node di depan linked list
void insertDepan(Node* &head, int value) {
    Node* newNode = new Node;  // Alokasi memori untuk node baru
    newNode->data = value;     // Mengisi nilai pada node baru
    newNode->next = head;      // Menunjuk ke node pertama sebelumnya
    head = newNode;            // Node baru menjadi head
}

// Fungsi untuk menambah node di belakang linked list
void insertBelakang(Node* &head, int value) {
    Node* newNode = new Node;  // Alokasi memori untuk node baru
    newNode->data = value;     // Mengisi nilai pada node baru
    newNode->next = nullptr;   // Node baru menjadi node terakhir (next = nullptr)

    if (head == nullptr) {     // Jika list kosong
        head = newNode;        // Node baru menjadi node pertama
    } else {
        Node* temp = head;     // Memulai dari head
        while (temp->next != nullptr) {
            temp = temp->next; // Menemukan node terakhir
        }
        temp->next = newNode;  // Menambahkan node baru di akhir
    }
}

// Fungsi untuk mencari node dengan nilai tertentu
bool cariNode(Node* head, int value) {
    Node* temp = head;
    while (temp != nullptr) {
        if (temp->data == value) {
            return true;  // Node dengan nilai ditemukan
        }
        temp = temp->next;
    }
    return false;  // Node dengan nilai tidak ditemukan
}

// Fungsi untuk menghitung panjang linked list
int hitungPanjang(Node* head) {
    int panjang = 0;
    Node* temp = head;
    while (temp != nullptr) {
        panjang++;         // Tambah panjang untuk setiap node
        temp = temp->next;
    }
    return panjang;
}

// Fungsi untuk mencetak seluruh isi linked list
void cetakList(Node* head) {
    if (head == nullptr) {
        cout << "Linked List kosong!" << endl;
        return;
    }

    Node* temp = head;  // Mulai dari node pertama
    while (temp != nullptr) {
        cout << temp->data;     // Cetak nilai node
        if (temp->next != nullptr) {
            cout << " -> ";     // Cetak panah jika masih ada node berikutnya
        }
        temp = temp->next;      // Pindah ke node berikutnya
    }
    cout << endl;
}

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

    // Menambah node berdasarkan contoh input
    insertDepan(head, 10);    // Tambah node di depan dengan nilai 10
    insertBelakang(head, 20); // Tambah node di belakang dengan nilai 20
    insertDepan(head, 5);     // Tambah node di depan dengan nilai 5

    // Mencari node dengan nilai tertentu
    int cari = 20;
    if (cariNode(head, cari)) {
        cout << "Node dengan nilai " << cari << " ditemukan." << endl;
    } else {
        cout << "Node dengan nilai " << cari << " tidak ditemukan." << endl;
    }

    // Menghitung panjang linked list
    cout << "Panjang linked list: " << hitungPanjang(head) << endl;

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

    return 0;
}
```

Output Program:

```
PS C:\Users\Farhan Kurniawan\Desktop\pemograman\c++\Unguided(4)>
\Unguided(4)\" ; if ($?) { g++ nomor3.cpp -o nomor3 } ; if ($?)
Node dengan nilai 20 ditemukan.
Panjang linked list: 3
Isi Linked List: 5 -> 10 -> 20
PS C:\Users\Farhan Kurniawan\Desktop\pemograman\c++\Unguided(4)>
```

V. KESIMPULAN

Penggunaan linked list dengan pointer dan operator merupakan konsep fundamental dalam pemrograman yang memungkinkan pengelolaan data secara dinamis. Melalui pemahaman ini, kita dapat mengimplementasikan berbagai operasi dasar pada linked list, seperti penambahan dan penghapusan node di awal maupun akhir list, serta pencarian nilai di dalam list. Operasi-operasi ini memperlihatkan fleksibilitas linked list dalam memanipulasi data secara efisien. Dengan memanfaatkan prototipe yang ada, kita dapat menyusun program yang mengelola data secara dinamis, yang sangat berguna dalam situasi di mana ukuran data tidak diketahui atau terus berubah.