

**LAPORAN PRAKTIKUM**

**MODUL 4**

**SINGLE LINKED LIST (BAGIAN PERTAMA)**



**Disusun Oleh:**

**Muhammad Ikhsan Al Hakim (2311104064)**

**S1SE-07-02**

**Dosen :**

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

**PROGRAM STUDI S1 SOFTWARE ENGINEERING**

**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

### 2.1 Linked List dengan Pointer

**Linked List** adalah sebuah struktur data yang digunakan untuk menyimpan koleksi data dalam bentuk yang teratur. Tidak seperti array yang menyimpan elemen-elemen secara berurutan dalam blok memori yang berdekatan, *linked list* menyimpan elemen-elemen dalam node-node yang saling terhubung melalui pointer (penunjuk). Data yang disimpan dalam *Linked list* bisa berupa data tunggal atau data majemuk. Data tunggal merupakan data yang hanya terdiri dari satu data (variabel), misalnya: nama bertipe *string*. Sedangkan data majemuk merupakan sekumpulan data (*record*) yang di dalamnya terdiri dari berbagai tipe data, misalnya: Data Mahasiswa, terdiri dari Nama bertipe *string*, NIM bertipe *long integer*, dan Alamat bertipe *string*.

*Linked list* dapat diimplementasikan menggunakan *Array* dan *Pointer* (*Linked list*).

Yang akan kita gunakan adalah *pointer*, karena beberapa alasan, yaitu :

1. *Array* bersifat statis, sedangkan *pointer* dinamis.
2. Pada *linked list* bentuk datanya saling bergandengan (berhubungan) sehingga lebih mudah memakai *pointer*.
3. Sifat *linked list* yang fleksibel lebih cocok dengan sifat *pointer* yang dapat diatur sesuai kebutuhan.
4. Karena *array* lebih susah dalam menangani *linked list*, sedangkan *pointer* lebih mudah.
5. *Array* lebih cocok pada kumpulan data yang jumlah elemen maksimumnya sudah diketahui dari awal.

Dalam implementasinya, pengaksesan elemen pada *Linked list* dengan *pointer* bisa menggunakan

(->) atau tanda titik (.).

Model-model dari ADT *Linked list* yang kita pelajari adalah :

1. *Single Linked list*
2. *Double Linked list*
3. *Circular Linked list*
4. *Multi Linked list*

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

Setiap model ADT *Linked list* di atas memiliki karakteristik tertentu dan dalam penggunaannya disesuaikan dengan kebutuhan.

Secara umum operasi-operasi ADT pada *Linked list*, yaitu :

1. Penciptaan dan inisialisasi *list* (*Create List*).
2. Penyisipan elemen *list* (*Insert*).
3. Penghapusan elemen *list* (*Delete*).
4. Penelusuran elemen *list* dan menampilkannya (*View*).
5. Pencarian elemen *list* (*Searching*).
6. Pengubahan isi elemen *list* (*Update*).

## 2.2 Single Linked List

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

Keterangan:

Elemen: segmen-segmen data yang terdapat dalam suatu *list*.

Data: informasi utama yang tersimpan dalam sebuah elemen.

Suksesor: bagian elemen yang berfungsi sebagai penghubung antar elemen.

Sifat dari *Single Linked list*:

1. Hanya memerlukan satu buah *pointer*.
2. *Node* akhir menunjuk ke Nil kecuali untuk *list circular*.
3. Hanya dapat melakukan pembacaan maju.
4. Pencarian sequensial dilakukan jika data tidak terurut.
5. Lebih mudah ketika melakukan penyisipan atau penghapusan di tengah *list*.

Istilah-istilah dalam *Single Linked list* :

1. *first/head: pointer* pada *list* yang menunjuk alamat elemen pertama *list*.
2. *next: pointer* pada elemen yang berfungsi sebagai *successor* (penunjuk) alamat elemen di depannya.
3. Null/Nil: artinya tidak memiliki nilai, atau tidak mengacu ke mana pun, atau kosong.

*Node/simpul/elemen*: merupakan tempat penyimpanan data pada suatu memori tertentu.

### 2.2.1 Pembentukan Komponen-Komponen *List*

#### A. Pembentukan *List*

Adalah sebuah proses untuk membentuk sebuah *list* baru. Biasanya nama fungsi yang digunakan **createList()**. Fungsi ini akan mengeset nilai awal *list* yaitu *first(list)* dan *last(list)* dengan nilai Nil.

#### B. Pengalokasian Memori

Adalah proses untuk mengalokasikan memori untuk setiap elemen data yang ada dalam *list*. Fungsi yang biasanya digunakan adalah nama fungsi yang biasa digunakan `alokasi()`.

### C. Dealokasi

Untuk menghapus sebuah *memory address* yang tersimpan atau telah dialokasikan dalam bahasa pemrograman C digunakan sintak ***free***, sedangkan pada Cpp digunakan sintak ***delete***.

### D. Pengecekan *List*

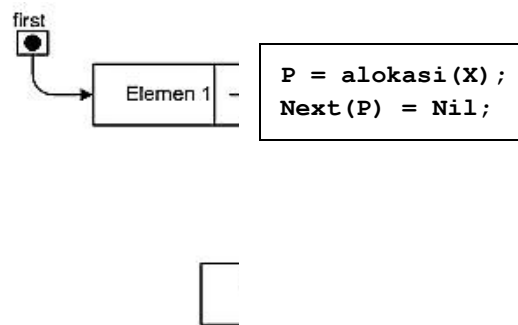
Adalah fungsi untuk mengecek apakah *list* tersebut kosong atau tidak. Akan mengembalikan nilai *true* jika *list* kosong dan nilai *false* jika *list* tidak kosong. Fungsi yang digunakan adalah `isEmpty()`.

## 2.2.2 Insert

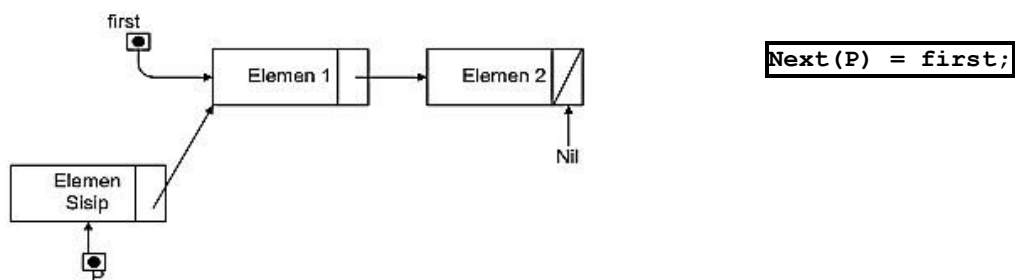
### A. Insert First

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

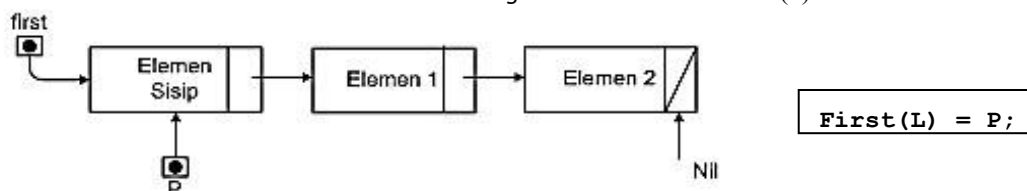
Langkah-langkah dalam proses *insert first*:



Gambar 4-4 Single Linked list Insert First (1)



Gambar 4-5 Single Linked list Insert First (2)



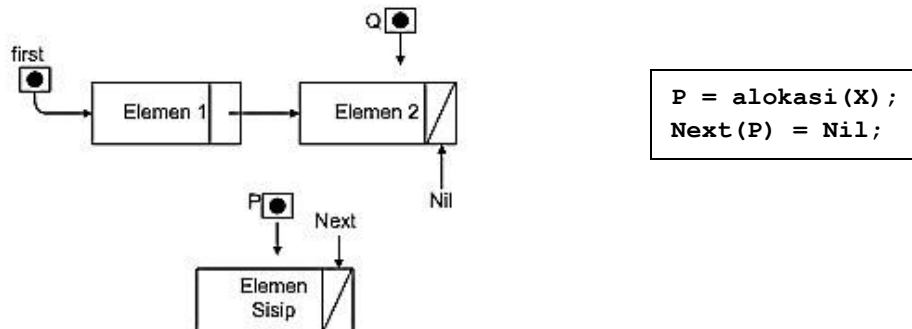
Gambar 4-6 Single Linked list Insert First (3)

```
/* contoh syntax insert first */ void insertFirst(List &L,
address &P){      next (P) = first(L);      first(L) = P; }
```

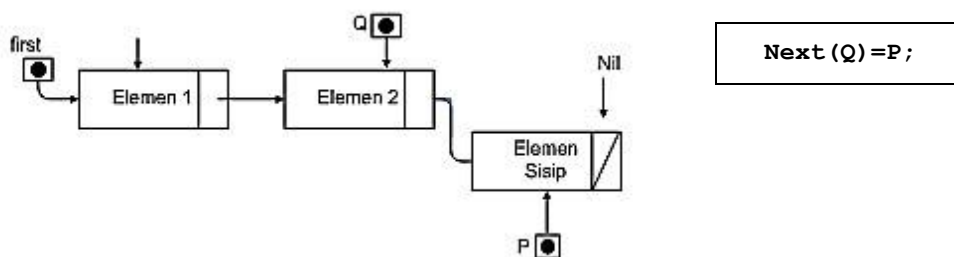
## B. Insert Last

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

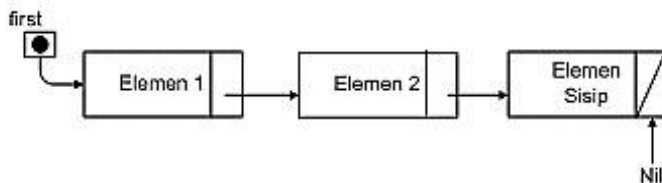
Langkah dalam *insert last* :



Gambar 4-7 Single Linked list Insert Last 1



Gambar 4-8 Single Linked list Insert Last 2

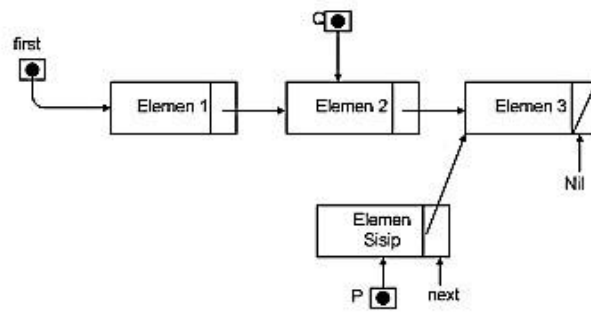


Gambar 4-9 Single Linked list Insert Last 3

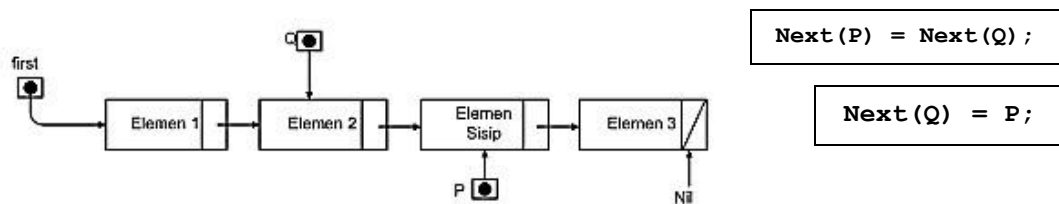
## C. Insert After

Merupakan metode memasukkan data ke dalam *list* yang diletakkan setelah *node* tertentu yang ditunjuk oleh *user*. Langkah dalam *insert after*:

P = alokasi(X);  
Next(P) = Nil;



Gambar 4-11 Single Linked list Insert After 2



Gambar 4-12 Single Linked list Insert After 3

### III. GUIDED

#### 1. Guided 1

Code:

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 // Deklarasi struct untuk mahasiswa
6 struct mahasiswa {
7     char nama[30];
8     char nim[10];
9 };
10
11 // Deklarasi struct Node
12 struct Node {
13     mahasiswa data;
14     Node *next;
15 };
16
17 Node *head;
18 Node *tail;
19
20 // Inisialisasi list
21 void init() {
22     head = nullptr;
23     tail = nullptr;
24 }
25
26 // Pengisian apakah list kosong
27 bool isEmpty() {
28     return head == nullptr;
29 }
30
31 // Tambah Depan
32 void insertDepan(const mahasiswa &data) {
33     Node *baru = new Node;
34     baru->data = data;
35     baru->next = nullptr;
36     if (isEmpty()) {
37         head = tail = baru;
38     } else {
39         baru->next = head;
40         head = baru;
41     }
42 }
43
44 // Tambah Belakang
45 void insertBelakang(const mahasiswa &data) {
46     Node *baru = new Node;
47     baru->data = data;
48     baru->next = nullptr;
49     if (isEmpty()) {
50         head = tail = baru;
51     } else {
52         tail->next = baru;
53         tail = baru;
54     }
55 }
56
57 // Hitung Jumlah list
58 int hitungList() {
59     Node *current = head;
60     int jumlah = 0;
61     while (current != nullptr) {
62         jumlah++;
63         current = current->next;
64     }
65     return jumlah;
66 }
67
68 // Hapus Depan
69 void hapusDepan() {
70     if (!isEmpty()) {
71         Node *hapus = head;
72         head = head->next;
73         delete hapus;
74         if (head == nullptr) {
75             tail = nullptr; // Jika list menjadi kosong
76         }
77     } else {
78         cout << "List kosong!" << endl;
79     }
80 }
81
82 // Hapus Belakang
83 void hapusBelakang() {
84     if (!isEmpty()) {
85         if (head == tail) {
86             delete head;
87             head = tail = nullptr; // list menjadi kosong
88         } else {
89             Node *bantu = head;
90             while (bantu->next != tail) {
91                 bantu = bantu->next;
92             }
93             delete tail;
94             tail = bantu;
95             tail->next = nullptr;
96         }
97     } else {
98         cout << "List kosong!" << endl;
99     }
100 }
101
102 // Tampilkan list
103 void tampil() {
104     Node *current = head;
105     if (!isEmpty()) {
106         while (current != nullptr) {
107             cout << "Nama: " << current->data.nama << ", NIM: " << current->data.nim << endl;
108             current = current->next;
109         }
110     } else {
111         cout << "List masih kosong!" << endl;
112     }
113     cout << endl;
114 }
115
116 // Hapus list
117 void clearList() {
118     Node *current = head;
119     while (current != nullptr) {
120         Node *hapus = current;
121         current = current->next;
122         delete hapus;
123     }
124     head = tail = nullptr;
125     cout << "List berhasil terhapus!" << endl;
126 }
127
128 // Main function
129 int main() {
130     init();
131
132     // Contoh data mahasiswa
133     mahasiswa m1 = {"Alice", "123456"};
134     mahasiswa m2 = {"Bob", "654321"};
135     mahasiswa m3 = {"Charlie", "112233"};
136
137     // Menambahkan mahasiswa ke dalam list
138     insertDepan(m1);
139     tampil();
140     insertBelakang(m2);
141     tampil();
142     insertDepan(m3);
143     tampil();
144
145     // Menghapus elemen dari list
146     hapusDepan();
147     tampil();
148     hapusBelakang();
149     tampil();
150
151     // Menghapus seluruh list
152     clearList();
153
154     return 0;
155 }
```

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
```

## 2. Guided 2

Code:



```

1 #include <iostream>
2 using namespace std;
3
4 // Definisi struktur untuk elemen list
5 struct Node {
6     int data; // Menyimpan nilai elemen
7     Node* next; // Pointer ke elemen berikutnya
8 };
9
10 // Fungsi untuk mengalokasikan memori untuk node baru
11 Node* alokasi(int value) {
12     Node* newNode = new Node; // Alokasi memori untuk elemen baru
13     if (newNode != nullptr) { // Jika alokasi berhasil
14         newNode->data = value; // Mengisi data node
15         newNode->next = nullptr; // Set next ke nullptr
16     }
17     return newNode; // Mengembalikan pointer node baru
18 }
19
20 // Fungsi untuk dealokasi memori node
21 void dealokasi(Node* node) {
22     delete node; // Mengembalikan memori yang digunakan oleh node
23 }
24
25 // Pengecekan apakah list kosong
26 bool isEmpty(Node* head) {
27     return head == nullptr; // List kosong jika head adalah nullptr
28 }
29
30 // Menambahkan elemen di awal list
31 void insertFirst(Node* &head, int value) {
32     Node* newNode = alokasi(value); // Alokasi memori untuk elemen baru
33     if (newNode != nullptr) {
34         newNode->next = head; // Menghubungkan elemen baru ke elemen pertama
35         head = newNode; // Menetapkan elemen baru sebagai elemen pertama
36     }
37 }
38
39 // Menambahkan elemen di akhir list
40 void insertLast(Node* &head, int value) {
41     Node* newNode = alokasi(value); // Alokasi memori untuk elemen baru
42     if (newNode != nullptr) {
43         if (isEmpty(head)) { // Jika list kosong
44             head = newNode; // Elemen baru menjadi elemen pertama
45         } else {
46             Node* temp = head;
47             while (temp->next != nullptr) { // Mencari elemen terakhir
48                 temp = temp->next;
49             }
50             temp->next = newNode; // Menambahkan elemen baru di akhir list
51         }
52     }
53 }
54
55 // Menampilkan semua elemen dalam list
56 void printList(Node* head) {
57     if (isEmpty(head)) {
58         cout << "List kosong!" << endl;
59     } else {
60         Node* temp = head;
61         while (temp != nullptr) { // Selama belum mencapai akhir list
62             cout << temp->data << " "; // Menampilkan data elemen
63             temp = temp->next; // Melanjutkan ke elemen berikutnya
64         }
65         cout << endl;
66     }
67 }
68
69 // Menghitung jumlah elemen dalam list
70 int countElements(Node* head) {
71     int count = 0;
72     Node* temp = head;
73     while (temp != nullptr) {
74         count++; // Menambah jumlah elemen
75         temp = temp->next; // Melanjutkan ke elemen berikutnya
76     }
77     return count; // Mengembalikan jumlah elemen
78 }
79
80 // Menghapus semua elemen dalam list dan dealokasi memori
81 void clearList(Node* &head) {
82     while (head != nullptr) {
83         Node* temp = head; // Simpan pointer ke node saat ini
84         head = head->next; // Pindahkan ke node berikutnya
85         dealokasi(temp); // Dealokasi node
86     }
87 }
88
89 int main() {
90     Node* head = nullptr; // Membuat list kosong
91
92     // Menambahkan elemen ke dalam list
93     insertFirst(head, 10); // Menambahkan elemen 10 di awal list
94     insertLast(head, 20); // Menambahkan elemen 20 di akhir list
95     insertLast(head, 30); // Menambahkan elemen 30 di akhir list
96
97     // Menampilkan isi list
98     cout << "Isi list: ";
99     printList(head);
100
101     // Menampilkan jumlah elemen
102     cout << "Jumlah elemen: " << countElements(head) << endl;
103
104     // Menghapus semua elemen dalam list
105     clearList(head);
106
107     // Menampilkan isi list setelah penghapusan
108     cout << "Isi list setelah penghapusan: ";
109     printList(head);
110
111     return 0;
112 }

```

Output:

```

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

```

#### IV. UNGUIDED

##### 1. Membuat Single Linklist

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.

Code:

```

1  #include <iostream>
2  using namespace std;
3
4  struct Node {
5      int data;
6      Node* next;
7  };
8
9  // Function to insert a node at the front
10 void insertFront(Node** head, int value) {
11     Node* newNode = new Node();
12     newNode->data = value;
13     newNode->next = *head;
14     *head = newNode;
15 }
16
17 // Function to insert a node at the back
18 void insertBack(Node** head, int value) {
19     Node* newNode = new Node();
20     newNode->data = value;
21     newNode->next = nullptr;
22
23     if (*head == nullptr) {
24         *head = newNode;
25         return;
26     }
27
28     Node* temp = *head;
29     while (temp->next != nullptr) {
30         temp = temp->next;
31     }
32     temp->next = newNode;
33 }
34
35 // Function to print the linked list
36 void printList(Node* head) {
37     Node* temp = head;
38     while (temp != nullptr) {
39         cout << temp->data ;
40         if (temp->next != nullptr) {
41             cout << " -> "; // Only print arrow if there's a next node
42         }
43         temp = temp->next;
44     }
45     cout << endl; // New line after printing the list
46 }
47
48
49
50
51 int main() {
52     Node* head = nullptr;
53
54     insertFront(&head, 10);
55     insertBack(&head, 20);
56     insertFront(&head, 5);
57
58     cout << "Linked List: ";
59     printList(head);
60
61     return 0;
62 }
63

```

Output:

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

## 2. Menghapus Node pada Linkedlist

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.

Code:

```

1  #include <iostream>
2  using namespace std;
3
4  struct Node {
5      int data;
6      Node* next;
7  };
8
9  // Function to insert a node at the front
10 void insertFront(Node** head, int value) {
11     Node* newNode = new Node();
12     newNode->data = value;
13     newNode->next = *head;
14     *head = newNode;
15 }
16
17 // Function to insert a node at the back
18 void insertBack(Node** head, int value) {
19     Node* newNode = new Node();
20     newNode->data = value;
21     newNode->next = nullptr;
22
23     if (*head == nullptr) {
24         *head = newNode;
25         return;
26     }
27
28     Node* temp = *head;
29     while (temp->next != nullptr) {
30         temp = temp->next;
31     }
32     temp->next = newNode;
33 }
34
35 // Function to delete a node with a given value
36 void deleteNode(Node** head, int value) {
37     Node* temp = *head;
38     Node* prev = nullptr;
39
40     // If head node itself holds the value to be deleted
41     if (temp != nullptr && temp->data == value) {
42         *head = temp->next; // Change head
43         delete temp; // Free old head
44         return;
45     }
46
47     // Search for the value to be deleted
48     while (temp != nullptr && temp->data != value) {
49         prev = temp;
50         temp = temp->next;
51     }
52
53     // If value was not present in the list
54     if (temp == nullptr) return;
55
56     // Unlink the node from linked list
57     prev->next = temp->next;
58
59     delete temp; // Free memory
60 }
61
62 // Function to print the linked list
63 void printList(Node* head) {
64     Node* temp = head;
65     while (temp != nullptr) {
66         cout << temp->data;
67         temp = temp->next;
68         if (temp != nullptr) {
69             cout << " -> ";
70         }
71     }
72     cout << endl;
73 }
74
75 int main() {
76     Node* head = nullptr;
77
78     insertFront(&head, 10);
79     insertBack(&head, 20);
80     insertFront(&head, 5);
81
82     cout << "Linked List before deletion: ";
83     printList(head);
84
85     deleteNode(&head, 10);
86
87     cout << "Linked List after deletion: ";
88     printList(head);
89
90     return 0;
91 }
92

```

Output:

```

Linked List before deletion: 5 -> 10 -> 20
Linked List after deletion: 5 -> 20

```

### 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.

Jawaban:

Code:

```

1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     int data;
6     Node* next;
7 };
8
9 // Function to insert a node at the front
10 void insertFront(Node** head, int value) {
11     Node* newNode = new Node();
12     newNode->data = value;
13     newNode->next = *head;
14     *head = newNode;
15 }
16
17 // Function to insert a node at the back
18 void insertBack(Node** head, int value) {
19     Node* newNode = new Node();
20     newNode->data = value;
21     newNode->next = nullptr;
22
23     if (*head == nullptr) {
24         *head = newNode;
25         return;
26     }
27
28     Node* temp = *head;
29     while (temp->next != nullptr) {
30         temp = temp->next;
31     }
32     temp->next = newNode;
33 }
34
35 // Function to search for a node with a given value
36 bool searchNode(Node* head, int value) {
37     Node* temp = head;
38     while (temp != nullptr) {
39         if (temp->data == value) {
40             return true;
41         }
42         temp = temp->next;
43     }
44     return false;
45 }
46
47 // Function to count the length of the linked list
48 int countLength(Node* head) {
49     int count = 0;
50     Node* temp = head;
51     while (temp != nullptr) {
52         count++;
53         temp = temp->next;
54     }
55     return count;
56 }
57
58 // Function to print the linked list
59 void printList(Node* head) {
60     Node* temp = head;
61     while (temp != nullptr) {
62         cout << temp->data;
63         temp = temp->next;
64         if (temp != nullptr) {
65             cout << " -> ";
66         }
67     }
68     cout << endl;
69 }
70
71 int main() {
72     Node* head = nullptr;
73
74     insertFront(&head, 10);
75     insertBack(&head, 20);
76     insertFront(&head, 5);
77
78     cout << "Linked List: ";
79     printList(head);
80
81     // Search for a value
82     int searchValue = 20;
83     if (searchNode(head, searchValue)) {
84         cout << "Node dengan nilai " << searchValue << " ditemukan." << endl;
85     } else {
86         cout << "Node dengan nilai " << searchValue << " tidak ditemukan." << endl;
87     }
88
89     // Count length of the list
90     int length = countLength(head);
91     cout << "Panjang linked list: " << length << endl;
92
93     return 0;
94 }
95

```

Output:

```

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

```

