# LAPORAN PRAKTIKUM

# MODUL

# SINGLE LINKED LIST (BAGIAN PERTAMA)


**Disusun Oleh:**

**Tiurma Grace Angelina**

**2311104042 SE-07-02**

**Dosen :**

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

**PROGRAM STUDI S1 SOFTWARE ENGINEERING**

**FAKULTAS INFORMATIKA**

**TELKOM UNIVERSITY**

**PURWOKERTO**


**2024**

## I. TUJUAN

**a.** Memahami penggunaan *linked list* dengan *pointer* operator-operator dalam program.

**b.** Memahami operasi-operasi dasar dalam *linked list.*

**c.** Membuat program dengan menggunakan *linked list* dengan *prototype* yang ada.

## II. LANDASAN TEORI

### 1. *Linked List dengan Pointer*

Linked list adalah salah satu struktur data dinamis yang terdiri dari rangkaian elemen yang saling terhubung melalui pointer. Setiap elemen, yang dikenal sebagai *node*, terdiri dari dua bagian utama: data dan pointer yang menunjuk ke elemen berikutnya dalam daftar. Berbeda dengan array yang memerlukan alokasi memori secara statis, linked list bersifat fleksibel karena ukurannya dapat berubah-ubah secara dinamis sesuai kebutuhan program.

Dalam implementasi single linked list, setiap node hanya memiliki satu arah pointer, yaitu menunjuk ke elemen berikutnya. Node terakhir dalam linked list akan menunjuk ke NULL sebagai penanda bahwa daftar telah berakhir. Fleksibilitas ini memungkinkan linked list untuk mengelola penambahan atau penghapusan elemen secara lebih mudah, terutama jika dibandingkan dengan array yang memiliki batasan ukuran tetap.

Operasi dasar pada single linked list meliputi pembuatan linked list (create), penambahan elemen di awal (insert first) dan di akhir (insert last), penghapusan elemen (delete), serta penelusuran elemen (traverse). Selain itu, linked list juga mendukung pencarian secara sekuensial untuk menemukan elemen berdasarkan nilai tertentu.

Beberapa fungsi penting dalam program yang menggunakan linked list antara lain:

- cariNode, digunakan untuk mencari apakah suatu nilai terdapat dalam linked list.
- hitungPanjang, digunakan untuk menghitung jumlah node atau panjang dari linked list.

Keunggulan utama dari linked list adalah efisiensinya dalam penggunaan memori, karena hanya membutuhkan memori sesuai jumlah data yang disimpan. Hal ini membuat linked list menjadi pilihan yang tepat dalam berbagai aplikasi yang memerlukan struktur data yang dapat bertambah atau berkurang ukurannya secara dinamis.

## III. GUIDED

1. guided_1

```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() {
```

```cpp
// 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;

```

```
55          head = tail = baru;
56
57      } else {
58
59          baru->next = head;
60
61          head = baru;
62      }
63  }
64
65
66  // Tambah Belakang
67  void insertBelakang(const mahasiswa &data) {
68
69      Node *baru = new Node;
70
71      baru->data = data;
72
73      baru->next = nullptr;
74
75      if (isEmpty()) {
76
77          head = tail = baru;
78
79      } else {
80
81          tail->next = baru;
82
83          tail = baru;
```

```
82
83          tail = baru;
84      }
85  }
86
87
88  // Hitung Jumlah List
89  int hitungList() {
90
91      Node *current = head;
92
93      int jumlah = 0;
94
95      while (current != nullptr) {
96
97          jumlah++;
98
99          current = current->next;
100     }
101     return jumlah;
102 }
103
104
105 // Hapus Depan
106 void hapusDepan() {
107
108     if (!isEmpty()) {
109
110         Node *hapus = head;
```

```
109
110         Node *hapus = head;
111
112         head = head->next;
113
114         delete hapus;
115
116         if (head == nullptr) {
117
118             tail = nullptr; // Jika list menjadi kosong
119
120         }
121
122          } else {
123
124         cout << "List kosong!" << endl;
125
126     }
127 }
128
129
130  // Hapus Belakang
131 void hapusBelakang() {
132
133     if (!isEmpty()) {
134
135         if (head == tail) {
136
137             delete head;
```

```
136
137             delete head;
138
139             head = tail = nullptr; // List menjadi kosong
140
141          } else {
142
143             Node *bantu = head;
144
145             while (bantu->next != tail) {
146
147                 bantu = bantu->next;
148             }
149
150             delete tail;
151
152             tail = bantu;
153
154             tail->next = nullptr;
155         }
156
157      } else {
158
159          cout << "List kosong!" << endl;
160     }
161 }
162
163
164  // Tampilkan List
```

```
163
164  // Tampilkan List
165 void tampil() {
166
167     Node *current = head;
168
169     if (!isEmpty()) {
170
171         while (current != nullptr) {
172
173             cout << "Nama: " << current->data.nama << ", NIM: " << current->data.nim << endl;
174
175             current = current->next;
176
177         }
178
179     } else {
180
181         cout << "List masih kosong!" << endl;
182     }
183 }
184
185
186  // Hapus List
187 void clearList() {
188
189     Node *current = head;
190
191     while (current != nullptr) {
```

```
190
191     while (current != nullptr) {
192
193         Node *hapus = current;
194
195         current = current->next;
196
197         delete hapus;
198     }
199
200     head = tail = nullptr;
201
202     cout << "List berhasil terhapus!" << endl;
203 }
204
205
206 // Main function
207 int main() {
208     init();
209
210     // Contoh data mahasiswa
211     mahasiswa m1 = {"Alice", "123456"};
212
213     mahasiswa m2 = {"Bob", "654321"};
214
215     mahasiswa m3 = {"Charlie", "112233"};
216
217     // Menambahkan mahasiswa ke dalam list
218     insertDepan(m1);
```

```
218     insertDepan(m1);
219
220     tampil();
221
222     insertBelakang(m2);
223
224     tampil();
225
226     insertDepan(m3);
227
228     tampil();
229
230     // Menghapus elemen dari list
231     hapusDepan();
232
233     tampil();
234
235     hapusBelakang();
236
237     tampil();
238
239     // Menghapus seluruh list
240     clearList();
241
242     return 0;
243 }
```

Output

```
C:\Users\USER\Documents\cb_std4\guided1\bin\Debug\guided1.exe
Nama: Alice, NIM: 123456
Nama: Alice, NIM: 123456
Nama: Bob, NIM: 654321
Nama: Charlie, NIM: 112233
Nama: Alice, NIM: 123456
Nama: Bob, NIM: 654321
Nama: Alice, NIM: 123456
Nama: Bob, NIM: 654321
Nama: Alice, NIM: 123456
List berhasil terhapus!

Process returned 0 (0x0)   execution time : 0.095 s
Press any key to continue.
```

Penjelasan :

Program ini dirancang untuk mengelola data mahasiswa menggunakan struktur data single linked list. Single linked list adalah struktur data yang terdiri dari serangkaian elemen yang saling terhubung, di mana setiap elemen (atau node) berisi data dan pointer yang menunjuk ke node berikutnya. Dalam konteks ini, data yang disimpan mencakup informasi mahasiswa, seperti nama dan NIM.

a. Struktur Data

- *Struct 'mahasiswa'*: Struct ini digunakan untuk mendefinisikan data mahasiswa dengan dua anggota:
  - `Nama`: Array karakter dengan panjang maksimal 30, digunakan untuk menyimpan nama mahasiswa.
  - `nim`: Array karakter dengan panjang maksimal 10, digunakan untuk menyimpan NIM mahasiswa.
- Struct 'Node': Struct ini digunakan untuk mendefinisikan node dalam linked list. Setiap node memiliki dua bagian:
  - `data`: Berisi informasi mahasiswa, bertipe struct 'mahasiswa'.
  - `next`: Pointer yang menunjuk ke node berikutnya dalam list.

b. Variabel Global
- `Node *head`: Pointer yang menunjuk ke node pertama dalam linked list.
- `Node *tail`: Pointer yang menunjuk ke node terakhir dalam linked list.

c. Fungsi Inisialisasi
- `void init()`: Fungsi ini digunakan untuk menginisialisasi linked list dengan mengatur `head` dan `tail` menjadi `nullptr`, menandakan bahwa list kosong.

d. Fungsi Pengecekan Kosong
- `bool isEmpty()`: Fungsi ini mengembalikan nilai boolean. Jika `head` bernilai `nullptr`, maka linked list dianggap kosong dan fungsi mengembalikan `true`, sebaliknya `false` jika tidak kosong.

e. Operasi Penambahan Node
- `void insertDepan(const mahasiswa &data)`: Fungsi ini menambahkan node baru di awal linked list.
  - Membuat node baru dengan data mahasiswa yang diberikan.
  - Jika linked list kosong, node baru menjadi head dan tail.
  - Jika tidak kosong, node baru ditambahkan di depan, dengan pointer `next` dari node baru menunjuk ke head sebelumnya.

- `void insertBelakang(const mahasiswa &data)`: Fungsi ini menambahkan node baru di akhir linked list.
  - Membuat node baru dengan data mahasiswa yang diberikan.
  - Jika list kosong, node baru menjadi head dan tail.
  - Jika tidak kosong, pointer `next` dari tail menunjuk ke node baru, dan tail diperbarui ke node tersebut.

f. Fungsi Penghitung Jumlah Node
- `int hitungList()`: Fungsi ini menghitung jumlah node dalam linked list.
  - Fungsi ini berjalan dari head hingga tail dan menghitung setiap node yang ada.

g. Fungsi Penghapusan Node

- `void hapusDepan()`: Fungsi ini menghapus node dari awal linked list.
  - Jika linked list tidak kosong, node yang ditunjuk oleh head dihapus, dan head diperbarui ke node berikutnya.
  - Jika setelah penghapusan `head` menjadi `nullptr`, `tail` juga diatur menjadi `nullptr`, menandakan list kosong.

- `void hapusBelakang()`: Fungsi ini menghapus node dari akhir linked list.
  - Jika hanya ada satu node, setelah penghapusan, `head` dan `tail` diatur menjadi `nullptr`.
  - Jika lebih dari satu node, fungsi ini mencari node sebelum tail dan memperbarui tail.

h. Fungsi Menampilkan Data
- `void tampil()`: Fungsi ini menampilkan seluruh data dalam linked list.
  - Fungsi ini mengiterasi dari head hingga tail dan mencetak nama serta NIM dari setiap node.
  - Jika linked list kosong, pesan "List masih kosong" akan ditampilkan.

i. Fungsi Penghapusan Seluruh List
- `void clearList()`: Fungsi ini menghapus seluruh node dalam linked list.
  - Fungsi ini mengiterasi dari head, menghapus setiap node satu per satu, hingga `head` dan `tail` diatur menjadi `nullptr`.

Program ini memberikan contoh penggunaan *single linked list* untuk pengelolaan data mahasiswa. Dengan struktur ini, operasi seperti penambahan, penghapusan, dan penampilan data mahasiswa dapat dilakukan dengan mudah dan efisien. Setiap perubahan dalam linked list dapat dilihat secara langsung melalui output program setelah setiap operasi dilakukan.

2. guided_2

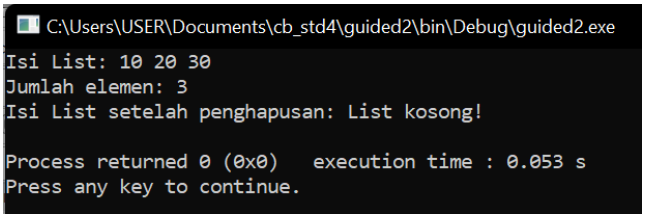
```cpp
#include <iostream>

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) {
```

```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
        }
    }
}

// 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
```

```cpp
        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
```

```cpp
            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:




Penjelasan:

Program ini digunakan untuk membuat dan mengelola linked list sederhana yang menyimpan nilai integer. Linked list adalah struktur data di mana elemen-elemen (atau node) saling terhubung satu sama lain. Dalam program ini, terdapat berbagai operasi dasar seperti menambah elemen, menampilkan isi list, menghitung jumlah elemen, dan menghapus semua elemen dari list.

a. Definisi Struktur

- Struct Node: Setiap elemen di dalam linked list diwakili oleh node.
  - `data`: Menyimpan nilai integer sebagai data dalam node.
  - `next`: Pointer yang menunjuk ke node berikutnya. Jika `next` bernilai `nullptr`, artinya node tersebut adalah node terakhir dalam list.

b. Fungsi Utama

- `Node alokasi(int value)`:
  - Fungsi ini mengalokasikan memori untuk node baru.
  - Jika alokasi berhasil, data pada node diisi dengan nilai yang diberikan, dan pointer `next` diatur ke `nullptr`.
  - Fungsi ini mengembalikan pointer ke node yang baru dialokasikan.
- `void dealokasi(Node node)`:
  - Fungsi ini membebaskan memori yang digunakan oleh node dengan menggunakan perintah `delete`, mengembalikannya ke sistem.

c. Fungsi Pengecekan Kosong

- `bool isListEmpty(Node head)`:
  - Fungsi ini mengecek apakah linked list kosong dengan memeriksa apakah pointer `head` bernilai `nullptr`.
  - Jika `head` bernilai `nullptr`, berarti tidak ada node dalam list.

d. Operasi Menambah Node

- `void insertFirst(Node &head, int value)`:
  - Fungsi ini menambah node baru di awal linked list.
  - Node baru dialokasikan menggunakan fungsi `alokasi()`. Jika berhasil, node tersebut dihubungkan ke node yang saat ini menjadi `head`, dan node baru ini menjadi `head` yang baru.
- `void insertLast(Node* &head, int value)`:
  - Fungsi ini menambah node baru di akhir linked list.
  - Jika list kosong, node baru akan menjadi `head`.
  - Jika list tidak kosong, fungsi ini akan mencari node terakhir dan menambahkan node baru di akhir.

e. Menampilkan Isi Linked List

- `void printList(Node head)`:
  - Fungsi ini mencetak semua elemen dalam linked list.

- Jika list kosong, fungsi akan mencetak pesan "List kosong!".
- Jika tidak kosong, fungsi ini akan mengiterasi setiap node dan mencetak nilai dari setiap node.

f. Menghitung Jumlah Elemen

- `int countElements(Node head)`:
  - Fungsi ini menghitung berapa banyak node yang ada dalam linked list.
  - Fungsi ini akan mengiterasi dari node pertama (head) sampai node terakhir dan menghitung setiap node yang ditemukan.

g. Menghapus Semua Elemen

- `void clearList(Node &head)`:
  - Fungsi ini menghapus semua elemen dari linked list.
  - Fungsi ini akan mengiterasi setiap node, menggeser `head` ke node berikutnya, dan memanggil fungsi `dealokasi()` untuk membebaskan memori dari node yang dihapus.

## IV. UNGUIDED

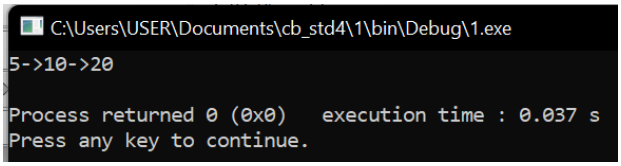
1. Task 1

Program ini adalah implementasi dari struktur data **linked list** (daftar berantai) dalam bahasa C++. Setiap node terdiri dari dua bagian: **data** (nilai) dan **pointer** (penunjuk) ke node berikutnya. Program ini memungkinkan kita untuk menambahkan elemen baru di depan atau di belakang daftar, serta mencetak isi dari linked list tersebut.

```cpp
#include <iostream>

using namespace std;

// Struktur Node untuk linked list
struct Node {
    int data;
    Node* next;
    Node(int val) : data(val), next(nullptr) {}
};

class LinkedList {
private:
    Node* head;

public:
    LinkedList() : head(nullptr) {}

    // Insert Node di Depan
    void insertFront(int val) {
        Node* newNode = new Node(val);
        newNode->next = head;
        head = newNode;
    }

    // Insert Node di Belakang
    void insertBack(int val) {
        Node* newNode = new Node(val);
        if (!head) {
            head = newNode;
            return;
        }
```

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

    // Cetak Linked List
    void printList() {
        Node* temp = head;
        while (temp) {
            cout << temp->data;
            if (temp->next) cout << "->";
            temp = temp->next;
        }
        cout << endl;
    }
};

int main() {
    LinkedList list;

    // Contoh input untuk Soal 1
    list.insertFront(10);
    list.insertBack(20);
    list.insertFront(5);
    list.printList(); // Output: 5->10->20

    return 0;
}
```

Output:




Penjelasan :

Di sini, kita membuat sebuah *struct* bernama Node. Setiap node terdiri dari dua bagian:

- data: Menyimpan nilai integer.
- next: Pointer yang menunjuk ke node berikutnya.

Konstruktor Node(int val) digunakan untuk menginisialisasi node baru dengan nilai val dan pointer next diatur ke nullptr, menandakan bahwa node ini belum terhubung ke node lain. Kita mendefinisikan kelas LinkedList yang memiliki satu data anggota yaitu head, yang menunjuk ke node pertama dari linked list.

Konstruktor LinkedList() akan menginisialisasi head menjadi nullptr, menandakan bahwa list awalnya kosong. Fungsi ini digunakan untuk menambahkan node baru di depan linked list.

Langkah-langkahnya:

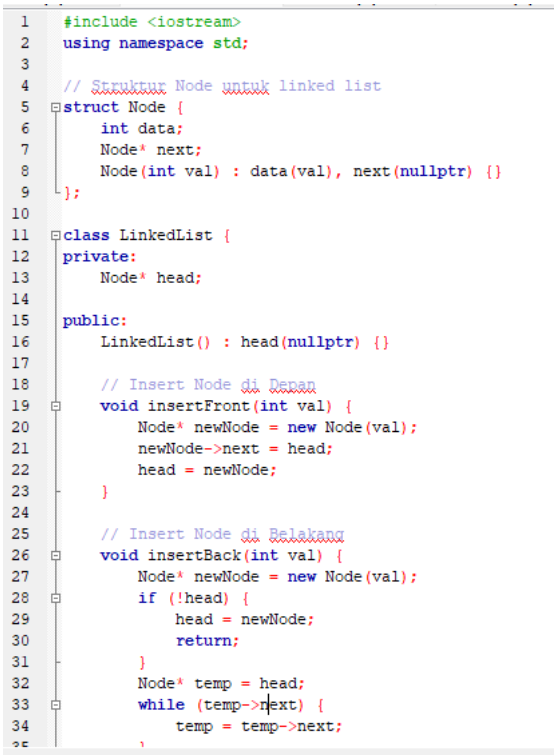
1. Membuat node baru dengan nilai val.
2. Mengatur next dari node baru tersebut agar menunjuk ke head (node yang saat ini berada di awal).
3. Mengubah head agar menunjuk ke node baru, sehingga node ini menjadi node pertama.

Fungsi insertBack(int val) menambahkan node baru di bagian akhir linked list. Jika list masih kosong (head adalah `nullptr`), maka node baru langsung menjadi `head`. Jika list tidak kosong, fungsi ini akan mengiterasi dari node pertama (`head`) hingga node terakhir (yang `next`-nya adalah `nullptr`), lalu menghubungkan node baru ke node terakhir. Fungsi printList() digunakan untuk mencetak semua elemen dalam linked list. Fungsi ini mengiterasi dari node pertama (head) hingga node terakhir, mencetak nilai data dari setiap node, dan menampilkan tanda "->" di antara node-nodenya jika masih ada node berikutnya.  Dalam fungsi main(), kita membuat sebuah objek LinkedList bernama list. Kemudian, kita menambahkan nilai

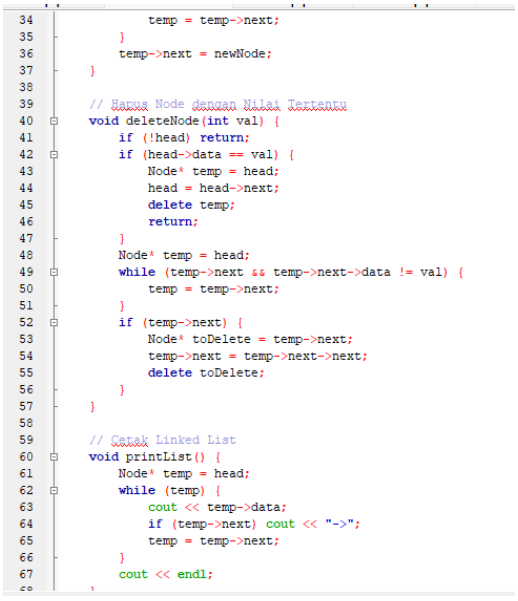
10 di depan, 20 di belakang, dan 5 di depan list. Fungsi printList() akan menampilkan isi linked list dengan urutan 5->10->20.

2. Task 2

Program ini adalah implementasi dari struktur data **linked list** dalam bahasa C++. Program ini memungkinkan kita untuk menambah elemen (node) di depan atau dibelakang linked list, menghapus node yang memiliki nilai tertentu, dan mencetak isi dari linked list:



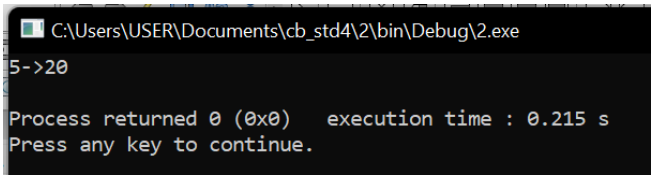
```cpp
#include <iostream>
using namespace std;

// Struktur Node untuk linked list
struct Node {
    int data;
    Node* next;
    Node(int val) : data(val), next(nullptr) {}
};

class LinkedList {
private:
    Node* head;

public:
    LinkedList() : head(nullptr) {}

    // Insert Node di Depan
    void insertFront(int val) {
        Node* newNode = new Node(val);
        newNode->next = head;
        head = newNode;
    }

    // Insert Node di Belakang
    void insertBack(int val) {
        Node* newNode = new Node(val);
        if (!head) {
            head = newNode;
            return;
        }
        Node* temp = head;
        while (temp->next) {
            temp = temp->next;
```



```cpp
            temp = temp->next;
        }
        temp->next = newNode;
    }

    // Hapus Node dengan Nilai Tertentu
    void deleteNode(int val) {
        if (!head) return;
        if (head->data == val) {
            Node* temp = head;
            head = head->next;
            delete temp;
            return;
        }
        Node* temp = head;
        while (temp->next && temp->next->data != val) {
            temp = temp->next;
        }
        if (temp->next) {
            Node* toDelete = temp->next;
            temp->next = temp->next->next;
            delete toDelete;
        }
    }

    // Cetak Linked List
    void printList() {
        Node* temp = head;
        while (temp) {
            cout << temp->data;
            if (temp->next) cout << "->";
            temp = temp->next;
        }
        cout << endl;
```

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

int main() {
    LinkedList list;

    // Contoh input untuk Soal 2
    list.insertFront(10);
    list.insertBack(20);
    list.insertFront(5);
    list.deleteNode(10);
    list.printList(); // Output: 5->20

    return 0;
}
```

Output :




Penjelasan:

**1. Struktur Node**

Program tetap menggunakan struktur Node yang berfungsi sebagai elemen dari linked list. Setiap node menyimpan:

- data: nilai integer.
- next: pointer ke node berikutnya dalam linked list.

**2. Kelas LinkedList**

Kelas LinkedList menyimpan pointer head yang menunjuk ke node pertama dalam linked list. Pointer ini digunakan untuk melacak awal dari list.

**3. Fungsi insertFront(int val)**

Fungsi ini berfungsi menambahkan node baru di depan linked list. Prosesnya mirip dengan program sebelumnya, yaitu membuat node baru, kemudian menghubungkannya ke head yang lama, dan menjadikan node baru sebagai head.

**4. Fungsi insertBack(int val)**

Fungsi ini menambahkan node baru di bagian belakang linked list. Jika list masih kosong, node baru langsung menjadi head. Jika tidak, fungsi ini akan mengiterasi hingga node terakhir, kemudian menambahkan node baru di akhir.

**5. Fungsi deleteNode(int val)**

Fungsi ini adalah tambahan baru dalam program ini, digunakan untuk menghapus node berdasarkan nilai tertentu (val).

- Pertama, jika linked list kosong (head == nullptr), maka fungsi langsung mengembalikan kontrol tanpa melakukan apapun.
- Jika nilai yang ingin dihapus ada di head, maka node pertama dihapus dan head diatur ke node berikutnya.
- Jika nilai yang ingin dihapus tidak ada di head, fungsi ini mengiterasi melalui list untuk menemukan node yang memiliki nilai yang cocok, lalu memutuskan node tersebut dari linked list, dan menghapusnya dari memori.
  Fungsi ini sangat berguna untuk mengelola elemen-elemen di dalam linked list

dan memastikan bahwa kita bisa menghapus elemen yang tidak lagi diperlukan.

**6. Fungsi printList()**

Fungsi ini mencetak seluruh elemen dalam linked list, dimulai dari head. Jika terdapat lebih dari satu node, maka setiap node akan dicetak dengan simbol -> di antara mereka. Jika list kosong, tidak ada elemen yang dicetak.

**7. Fungsi main()**

Dalam fungsi main(), beberapa operasi dilakukan pada linked list:

- Pertama, kita menambahkan nilai 10 di depan list.
- Kemudian, kita menambahkan nilai 20 di bagian belakang list.
- Selanjutnya, kita menambahkan nilai 5 lagi di depan.
- Setelah itu, kita menghapus node yang memiliki nilai 10 dari linked list.
- Terakhir, kita mencetak isi linked list, dan hasilnya adalah 5->20, karena node dengan nilai 10 sudah dihapus.

3. Task 3

Program ini memungkinkan untuk melakukan beberapa operasi penting termasuk menambah elemen (node) di depan atau di belakang linked list, mencari node tertentu di dalam linked list, dan menghiung panjang atau jumlah elemen dalam linked list

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

// Struktur Node untuk linked list
struct Node {
    int data;
    Node* next;
    Node(int val) : data(val), next(nullptr) {}
};

class LinkedList {
private:
    Node* head;

public:
    LinkedList() : head(nullptr) {}

    // Insert Node di Depan
    void insertFront(int val) {
        Node* newNode = new Node(val);
        newNode->next = head;
        head = newNode;
    }

    // Insert Node di Belakang
    void insertBack(int val) {
        Node* newNode = new Node(val);
        if (!head) {
            head = newNode;
            return;
        }
        Node* temp = head;
        while (temp->next) {
            temp = temp->next;
```

```
            temp = temp->next;
        }
        temp->next = newNode;
    }

    // Cari Node dengan Nilai Tertentu
    bool searchNode(int val) {
        Node* temp = head;
        while (temp) {
            if (temp->data == val) return true;
            temp = temp->next;
        }
        return false;
    }

    // Hitung Panjang Linked List
    int length() {
        int count = 0;
        Node* temp = head;
        while (temp) {
            count++;
            temp = temp->next;
        }
        return count;
    }

    // Cetak Linked List
    void printList() {
        Node* temp = head;
        while (temp) {
            cout << temp->data;
            if (temp->next) cout << "->";
            temp = temp->next;
        }
```

```
    // Cetak Linked List
    void printList() {
        Node* temp = head;
        while (temp) {
            cout << temp->data;
            if (temp->next) cout << "->";
            temp = temp->next;
        }
        cout << endl;
    }
};

int main() {
    LinkedList list;

    // Contoh input untuk Soal 3
    list.insertFront(10);
    list.insertBack(20);
    list.insertFront(5);

    // Mencari node dengan nilai tertentu
    int searchValue = 20;
    if (list.searchNode(searchValue)) {
        cout << "Node dengan nilai " << searchValue << " ditemukan." << endl;
    } else {
        cout << "Node dengan nilai " << searchValue << " tidak ditemukan." << endl;
    }

    // Menghitung panjang linked list
    cout << "Panjang linked list: " << list.length() << endl;

    return 0;
}
```

Output:

```
C:\Users\USER\Documents\cb_std4\3\bin\Debug\3.exe
Node dengan nilai 20 ditemukan.
Panjang linked list: 3

Process returned 0 (0x0)   execution time : 0.078 s
Press any key to continue.
```

Penjelasan :

**1. Struktur Node**

Masih sama seperti sebelumnya, struktur Node digunakan untuk merepresentasikan elemen-elemen dalam linked list. Setiap node menyimpan:

- data: nilai integer yang disimpan dalam node.
- next: pointer yang menunjuk ke node berikutnya. Jika node tersebut adalah node terakhir, next akan bernilai nullptr.

**2. Kelas LinkedList**

Kelas LinkedList berfungsi untuk mengelola operasi-operasi dasar yang dapat dilakukan pada linked list. Kelas ini memiliki pointer head untuk menunjuk ke node pertama dalam linked list.

**3. Fungsi insertFront(int val)**

Fungsi ini digunakan untuk menambahkan node baru di awal linked list. Prosesnya adalah:

- Membuat node baru dengan nilai yang diberikan (val).
- Menghubungkan node baru tersebut ke node yang sekarang menjadi head.
- Menjadikan node baru tersebut sebagai head yang baru.

**4. Fungsi insertBack(int val)**

Fungsi ini digunakan untuk menambahkan node baru di bagian belakang linked list. Prosesnya adalah:

- Jika linked list kosong (tidak ada node sama sekali), node baru akan langsung menjadi head.
- Jika tidak, fungsi akan mencari node terakhir (yang next-nya adalah nullptr), kemudian menghubungkan node baru tersebut di bagian akhir.

**5. Fungsi searchNode(int val)**

Fungsi ini berfungsi untuk mencari node yang memiliki nilai tertentu (val).

- Fungsi ini akan mengiterasi dari node pertama (head) sampai akhir list, membandingkan nilai setiap node dengan nilai yang dicari (val).
- Jika ditemukan, fungsi akan mengembalikan nilai true, menandakan bahwa node dengan nilai tersebut ada dalam linked list.
- Jika tidak ditemukan, maka akan mengembalikan false.

**6. Fungsi length()**

Fungsi ini digunakan untuk menghitung jumlah node dalam linked list.

- Fungsi akan mengiterasi dari node pertama hingga node terakhir, menghitung jumlah node yang ditemukan.
- Hasil dari perhitungan ini adalah panjang linked list, yang kemudian akan dikembalikan sebagai nilai integer.

**7. Fungsi printList()**

Fungsi ini bertugas mencetak semua nilai yang ada dalam linked list dari node pertama hingga node terakhir. Setiap node akan dicetak dengan format yang dipisahkan oleh simbol -> jika ada node berikutnya. Jika tidak ada node berikutnya, list akan diakhiri tanpa simbol tambahan.

**8. Fungsi main()**

Dalam fungsi main(), berikut beberapa operasi yang dilakukan:

- Menambahkan nilai 10 di depan list, kemudian menambahkan nilai 20 di belakang, dan menambahkan nilai 5 lagi di depan.
- Mencari node dengan nilai 20 menggunakan fungsi searchNode(). Jika ditemukan, program akan mencetak pesan bahwa node tersebut ada di dalam list, jika tidak ditemukan akan mencetak bahwa node tersebut tidak ada.
- Menghitung panjang linked list menggunakan fungsi length() dan mencetak hasilnya.

## V. KESIMPULAN

Dalam praktikum mengenai *single linked list*, telah dipelajari berbagai operasi dasar yang dapat dilakukan pada struktur data ini, seperti penambahan node di awal dan di akhir, pencarian elemen, penghapusan node, serta pencetakan isi linked list. Linked list memungkinkan pengelolaan data secara dinamis tanpa batasan ukuran yang tetap seperti pada array, sehingga lebih fleksibel dan efisien dalam penggunaan memori, terutama ketika data sering berubah ukuran. Selain itu, operasi dasar seperti menambah node, menghapus node, dan mencari elemen dapat dilakukan lebih efisien tanpa perlu menggeser elemen seperti pada array. Linked list menggunakan memori secara efektif karena hanya memerlukan ruang sebanyak node yang dibutuhkan, namun pengelolaan memori yang tepat sangat penting untuk menghindari kebocoran memori. Meskipun akses elemen pada linked list dilakukan secara beruntun, yang membuat waktu pencarian relatif lebih lama dibandingkan array, struktur ini tetap unggul dalam hal fleksibilitas. Praktikum ini memberikan pengalaman langsung dalam mengimplementasikan linked list menggunakan C++, yang mencakup inisialisasi list, penambahan elemen, pencarian, hingga penghitungan jumlah elemen dalam list. Dengan pemahaman ini, linked list dapat diaplikasikan dalam berbagai program yang membutuhkan struktur data dinamis dan fleksibel, serta memperkuat pemahaman mengenai pengelolaan data dalam memori komputer.