

# **LAPORAN PRAKTIKUM**

## **Modul 4**

### **“Single Linked List (Bagian Pertama)”**



**Disusun Oleh:**

**Rengganis Tantri Pramudita - 2311104065**

**S1SE0702**

**Dosen :**

**Wahyu Andi Saputra**

**PROGRAM STUDI S1 SOFTWARE ENGINEERING**

**FAKULTAS INFORMATIKA**

**TELKOM UNIVERSITY**

**PURWOKERTO**

**2024**

## 1. Tujuan

- Memahami penggunaan linked list dengan pointer operator- operator dalam program.
- Memahami operasi-operasi dasar dalam linked list.
- Membuat program dengan menggunakan linked list dengan prototype yang ada

## 2. Landasan Teori

Single linked list adalah struktur data dinamis yang terdiri dari serangkaian node yang saling terhubung melalui pointer. Setiap node memiliki dua komponen utama: data dan pointer ke node berikutnya. Linked list dengan pointer memanfaatkan konsep ini untuk menciptakan hubungan logis antar elemen, memungkinkan alokasi memori yang fleksibel dan efisien. Pembentukan komponen-komponen list melibatkan definisi struktur node, yang biasanya terdiri dari variabel untuk menyimpan data dan pointer ke node selanjutnya, serta inisialisasi head pointer yang menunjuk ke node pertama dalam list.

Operasi dasar pada single linked list meliputi insert (penyisipan), view (penelusuran), delete (penghapusan), dan update. Penyisipan dapat dilakukan di awal, akhir, atau di tengah list, memerlukan manipulasi pointer untuk mempertahankan integritas struktur. Penelusuran atau view melibatkan traversal list dari awal hingga akhir untuk mengakses atau menampilkan data. Penghapusan node membutuhkan penyesuaian pointer untuk menjaga kontinuitas list setelah node dihapus. Update melibatkan lokasi node target dan modifikasi datanya.

Dealokasi adalah proses penting dalam manajemen memori linked list, di mana memori yang tidak lagi digunakan oleh node yang dihapus dikembalikan ke sistem untuk mencegah kebocoran memori. Ini umumnya dilakukan secara manual dalam bahasa seperti C, sementara bahasa dengan garbage collection otomatis seperti Java atau Python menanganinya secara implisit.

Keunggulan utama single linked list terletak pada fleksibilitasnya dalam alokasi memori dinamis dan efisiensi dalam operasi penyisipan dan penghapusan, terutama di awal list. Namun, struktur ini memiliki keterbatasan dalam akses acak dan memerlukan traversal untuk mencapai node tertentu, yang dapat menjadi tidak efisien untuk list yang sangat panjang. Pemahaman mendalam tentang manipulasi pointer dan manajemen memori sangat penting dalam implementasi dan penggunaan efektif single linked list, membentuk dasar untuk struktur data yang lebih kompleks seperti double linked list dan circular linked list.

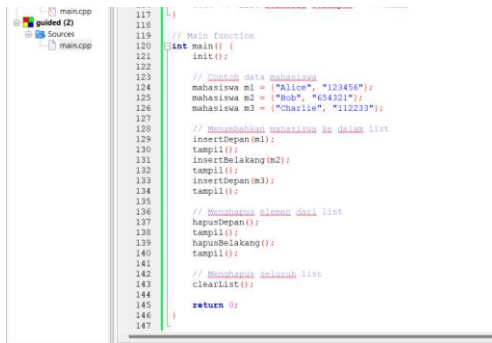
### 3. Guided

#### 1. Pertama

```
main.cpp (guided (1)) - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DockerDocks Settings Help
<global>
main.cpp X main.cpp X main.cpp X
1 #include <iostream>
2 #include <cstring>
3 using namespace std;
4
5 // Deklarasi struct untuk mahasiswa
6 struct mahasiswa {
7     char nama[50];
8     char nim[10];
9 };
10 // Deklarasi struct Node
11 struct Node {
12     mahasiswa data;
13     Node *next;
14 };
15
16 Node *head;
17 Node *tail;
18 // Inisialisasi list
19 void init() {
20     head = nullptr;
21     tail = nullptr;
22 }
23 // Fungsi untuk menambahkan node ke list
24 bool isEmpty() {
25     return head == nullptr;
26 }
27 // Fungsi untuk menambahkan node ke list
28 void insertDepan(const mahasiswa &data) {
29     Node *baru = new Node;
30     baru->data = data;
31     baru->next = nullptr;
32     if (isEmpty()) {
33         head = tail = baru;
34     } else {
35         baru->next = head;
36         head = baru;
37     }
38 }
39
```

```
main.cpp (guided (1)) - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DockerDocks Settings Help
<global>
main.cpp X main.cpp X main.cpp X
39 // Fungsi untuk menghapus node dari list
40 void insertBelakang(const mahasiswa &data) {
41     Node *baru = new Node;
42     baru->data = data;
43     baru->next = nullptr;
44     if (isEmpty()) {
45         head = tail = baru;
46     } else {
47         tail->next = baru;
48         tail = baru;
49     }
50 }
51 // Fungsi untuk menghitung jumlah node
52 int hitungList() {
53     Node *current = head;
54     int jumlah = 0;
55     while (current != nullptr) {
56         jumlah++;
57         current = current->next;
58     }
59     return jumlah;
60 }
61 // Fungsi untuk menghapus node dari list
62 void hapusDepan() {
63     if (isEmpty()) {
64         return;
65     }
66     delete head;
67     head = head->next;
68     if (head == nullptr) {
69         tail = nullptr;
70     }
71     cout << "List kosong!" << endl;
72 }
73 // Fungsi untuk menghapus node dari list
74 void hapusBelakang() {
75     if (isEmpty()) {
76         return;
77     }
78     if (head == tail) {
79         delete head;
80         head = tail = nullptr;
81     } else {
82         Node *current = head;
83         while (current->next != tail) {
84             current = current->next;
85         }
86         delete current;
87         current->next = nullptr;
88     }
89     cout << "List kosong!" << endl;
90 }
91 // Fungsi untuk menampilkan list
92 void tampil() {
93     if (isEmpty()) {
94         return;
95     }
96     Node *current = head;
97     while (current != nullptr) {
98         cout << "Nama: " << current->data.nama << ", NIM: " << current->data.nim << endl;
99         current = current->next;
100     }
101     if (current == nullptr) {
102         cout << "List kosong!" << endl;
103     }
104 }
105 // Fungsi untuk membersihkan list
106 void clearList() {
107     Node *current = head;
108     while (current != nullptr) {
109         delete current;
110         current = current->next;
111     }
112     head = tail = nullptr;
113     cout << "List berhasil dibersihkan!" << endl;
114 }
115
```

```
main.cpp (guided (1)) - Code::Blocks 20.03
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DockerDocks Settings Help
<global>
main.cpp X main.cpp X main.cpp X
78 // Fungsi untuk membersihkan list
79 void clearList() {
80     Node *current = head;
81     while (current != nullptr) {
82         delete current;
83         current = current->next;
84     }
85     head = tail = nullptr;
86     cout << "List berhasil dibersihkan!" << endl;
87 }
88 // Fungsi untuk menampilkan list
89 void tampil() {
90     if (isEmpty()) {
91         return;
92     }
93     Node *current = head;
94     while (current != nullptr) {
95         cout << "Nama: " << current->data.nama << ", NIM: " << current->data.nim << endl;
96         current = current->next;
97     }
98     if (current == nullptr) {
99         cout << "List kosong!" << endl;
100     }
101 }
102 // Fungsi untuk membersihkan list
103 void clearList() {
104     Node *current = head;
105     while (current != nullptr) {
106         delete current;
107         current = current->next;
108     }
109     head = tail = nullptr;
110     cout << "List berhasil dibersihkan!" << endl;
111 }
112
```



```
117 }
118
119 // Main function
120 int main() {
121     init();
122
123     // Contoh data mahasiswa
124     mahasiswa m1 = {"Alice", "123456"};
125     mahasiswa m2 = {"Bob", "654321"};
126     mahasiswa m3 = {"Charlie", "112233"};
127
128     // Menambahkan mahasiswa ke dalam list
129     insertDepan(m1);
130     tampil();
131     insertBelakang(m2);
132     tampil();
133     insertDepan(m3);
134     tampil();
135
136     // Menghapus elemen dari list
137     hapusDepan();
138     tampil();
139     hapusBelakang();
140     tampil();
141
142     // Menghapus seluruh list
143     clearList();
144
145     return 0;
146 }
147
```

## Keterangan

- Persiapan dan Deklarasi:
  - Program dimulai dengan include header yang diperlukan dan menggunakan namespace std.
  - Struct 'mahasiswa' didefinisikan untuk menyimpan data nama dan NIM.
  - Struct 'Node' didefinisikan, yang berisi data mahasiswa dan pointer ke node berikutnya.
  - Pointer global 'head' dan 'tail' dideklarasikan untuk mengelola linked list.
- Inisialisasi dan Pengecekan:
  - Fungsi 'init()' digunakan untuk menginisialisasi list kosong.
  - Fungsi 'isEmpty()' mengecek apakah list kosong.
- Operasi Penyisipan:
  - 'insertDepan()' menambahkan node baru di awal list.
  - 'insertBelakang()' menambahkan node baru di akhir list.
- Penghitungan dan Penghapusan:
  - 'hitungList()' menghitung jumlah node dalam list.
  - 'hapusDepan()' menghapus node pertama dari list.
  - 'hapusBelakang()' menghapus node terakhir dari list.
- Tampilan dan Pembersihan:
  - 'tampil()' menampilkan seluruh isi list.
  - 'clearList()' menghapus seluruh isi list dan membebaskan memori.
- Fungsi Main:
  - List diinisialisasi.
  - Beberapa data mahasiswa dibuat.
  - Operasi penyisipan, penghapusan, dan tampilan didemonstrasikan.
  - Akhirnya, seluruh list dibersihkan.
- Alur Eksekusi:

- Program menambahkan tiga mahasiswa ke list menggunakan insertDepan dan insertBelakang.
- Setelah setiap operasi, isi list ditampilkan.
- Dua operasi penghapusan dilakukan (depan dan belakang), dengan tampilan list setelah setiap operasi.
- Terakhir, seluruh list dihapus menggunakan clearList().

## Output

```

C:\codeblocks\performance\main.cpp
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
List berhasil terhapus!
Process returned 0 (0x0)   execution time : 0.158 s
Press any key to continue.

```

## 2. Kedua

### Code

```

main.cpp X main.cpp X *main.cpp X
1 #include <iostream>
2 using namespace std;
3 // Definisi struktur untuk elemen list
4 struct Node {
5     int data; // Menyimpan nilai elemen
6     Node* next; // Pointer ke elemen berikutnya
7 };
8 // Fungsi untuk mengalokasikan memori untuk node baru
9 Node* alokasi(int value) {
10     Node* newNode = new Node; // Alokasi memori untuk elemen baru
11     if (newNode != nullptr) { // Jika alokasi berhasil
12         newNode->data = value; // Mengisi data node
13         newNode->next = nullptr; // Set next ke nullptr
14     }
15     return newNode; // Mengembalikan pointer node baru
16 }
17 // Fungsi untuk dealokasi memori node
18 void dealokasi(Node* node) {
19     delete node; // Mengembalikan memori yang digunakan oleh node
20 }
21 // Mengecek apakah list kosong
22 bool isEmpty(Node* head) {
23     return head == nullptr; // List kosong jika head adalah nullptr
24 }
25 // Menambahkan elemen di awal list
26 void insertFirst(Node* &head, int value) {
27     Node* newNode = alokasi(value); // Alokasi memori untuk elemen baru
28     if (newNode != nullptr) {
29         newNode->next = head; // Menghubungkan elemen baru ke elemen pertama
30         head = newNode; // Menetapkan elemen baru sebagai elemen pertama
31     }
32 }
33 // Menambahkan elemen di akhir list
34 void insertLast(Node* &head, int value) {
35     Node* newNode = alokasi(value); // Alokasi memori untuk elemen baru
36     if (newNode != nullptr) {
37         if (isEmpty(head)) { // Jika list kosong
38             head = newNode; // Elemen baru menjadi elemen pertama
39         } else {
40             Node* temp = head;
41             while (temp->next != nullptr) { // Mencari elemen terakhir
42                 temp = temp->next;
43             }
44             temp->next = newNode; // Menambahkan elemen baru di akhir list
45         }
46     }
47 }
48 // Menampilkan semua elemen dalam list
49 void printList(Node* head) {
50     if (isEmpty(head)) {
51         cout << "List kosong!" << endl;
52     } else {
53         Node* temp = head;
54         while (temp != nullptr) { // Selama belum mencapai akhir list
55             cout << temp->data << " "; // Menampilkan data elemen
56             temp = temp->next; // Melanjutkan ke elemen berikutnya
57         }
58         cout << endl;
59     }
60 }
61 // Menghitung jumlah elemen dalam list
62 int countElements(Node* head) {
63     int count = 0;
64     Node* temp = head;
65     while (temp != nullptr) {
66         count++; // Menambah jumlah elemen
67         temp = temp->next; // Melanjutkan ke elemen berikutnya
68     }
69     return count; // Mengembalikan jumlah elemen
70 }
71 // Menghapus semua elemen dalam list dan dealokasi memori
72 void clearList(Node* &head) {
73     while (head != nullptr) {
74         Node* temp = head; // Simpan pointer ke node saat ini
75         head = head->next; // Zondasikan ke node berikutnya
76         dealokasi(temp); // Dealokasi node
77     }
78 }

```

```

79 int main() {
80     Node* head = nullptr; // Membuat list kosong
81     // Menambahkan elemen ke dalam list
82     insertFirst(head, 10); // Menambahkan elemen 10 di awal list
83     insertLast(head, 20); // Menambahkan elemen 20 di akhir list
84     insertLast(head, 30); // Menambahkan elemen 30 di akhir list
85     // Menampilkan isi list
86     cout << "Isi List: ";
87     printList(head);
88     // Menampilkan jumlah elemen
89     cout << "Jumlah elemen: " << countElements(head) << endl;
90     // Menghapus semua elemen dalam list
91     clearList(head);
92     // Menampilkan isi list setelah penghapusan
93     cout << "Isi list setelah penghapusan: ";
94     printList(head);
95     return 0;
96 }
97

```

## Keterangan

- *struct Node* mendefinisikan elemen-elemen dari linked list. Setiap node memiliki dua bagian:
  - o *int data*: Menyimpan nilai data dari elemen.
  - o *Node\* next*: Menyimpan pointer yang menunjuk ke elemen berikutnya.
- *Node alokasi(int value)*: Fungsi ini mengalokasikan memori untuk node baru, mengisi nilai data, dan mengatur pointer next ke nullptr. Jika berhasil, mengembalikan pointer ke node baru.
- *void dealokasi(Node\* node)*: Fungsi ini menghapus node dengan membebaskan memori yang digunakan.
- *bool isEmpty(Node\* head)*: Fungsi ini memeriksa apakah list kosong dengan mengecek apakah head menunjuk ke nullptr. Jika ya, list kosong.
- *void insertFirst(Node\* &head, int value)*: Fungsi ini menambahkan elemen baru di awal list. Node baru dihubungkan ke node yang saat ini ada di head, kemudian node baru tersebut menjadi head yang baru.
- *void insertLast(Node\* &head, int value)*: Fungsi ini menambahkan elemen baru di akhir list. Jika list kosong, node baru menjadi elemen pertama. Jika tidak, fungsi mencari node terakhir dan menghubungkannya dengan node baru.
- *void printList(Node\* head)*: Fungsi ini mencetak semua elemen dalam list, mulai dari head hingga node terakhir. Jika list kosong, akan mencetak pesan "List kosong!".
- *int countElements(Node\* head)*: Fungsi ini menghitung berapa banyak elemen dalam list dengan mengiterasi melalui seluruh node dan menghitung setiap node yang ditemukan.
- *void clearList(Node\* &head)*: Fungsi ini menghapus semua elemen dalam list dengan mengiterasi melalui semua node, memindahkan pointer head, dan mendelokasi setiap node satu per satu.
- Di dalam *main()*, program melakukan beberapa operasi:
  1. Membuat list kosong (head diinisialisasi dengan nullptr).

2. Menambahkan elemen 10 di awal list.
3. Menambahkan elemen 20 dan 30 di akhir list.
4. Menampilkan isi list dengan memanggil printList().
5. Menampilkan jumlah elemen dengan countElements().
6. Menghapus semua elemen dengan clearList().
7. Menampilkan isi list setelah penghapusan untuk memastikan bahwa list sudah kosong.

## Output

```

"C:\codeblocks\pertemuan4\y X + v
Isi List: 10 20 30
Jumlah elemen: 3
Isi List setelah penghapusan: List kosong!

Process returned 0 (0x0)   execution time : 0.583 s
Press any key to continue.

```

## 4. Unguided

### - Membuat Single Linked List

#### Code

```

main.cpp X main.cpp X main.cpp X *main.cpp X
1 #include <iostream>
2 using namespace std;
3
4 struct Node {
5     int data;
6     Node* next;
7 };
8 // Fungsi untuk membuat node baru
9 Node* createNode(int value) {
10     Node* newNode = new Node(value);
11     newNode->next = nullptr;
12     return newNode;
13 }
14 // Fungsi untuk menambahkan node di depan (awal) linked list
15 void insertAtFront(Node* head, int value) {
16     Node* newNode = createNode(value);
17     newNode->next = head;
18     head = newNode;
19 }
20 // Fungsi untuk menambahkan node di belakang linked list
21 void insertAtEnd(Node* head, int value) {
22     Node* newNode = createNode(value);
23     if (head == nullptr) {
24         head = newNode;
25     } else {
26         Node* temp = head;
27         while (temp->next != nullptr) {
28             temp = temp->next;
29         }
30         temp->next = newNode;
31     }
32 }
33 // Fungsi untuk mencetak seluruh isi linked list
34 void printList(Node* head) {
35     if (head == nullptr) {
36         cout << "List kosong!" << endl;
37     } else {
38         Node* temp = head;
39         while (temp != nullptr) {
40             cout << temp->data;
41             if (temp->next != nullptr) {
42                 cout << " -> ";
43             }
44             temp = temp->next;
45         }
46         cout << endl;
47     }
48 }
49
50 int main() {
51     Node* head = nullptr;
52     // operasi berdasarkan contoh
53     insertAtFront(head, 10);
54     insertAtEnd(head, 20);
55     insertAtFront(head, 30);
56     // cetak isi linked list
57     cout << "Isi linked list: ";
58     printList(head);
59     return 0;
60 }
61

```

#### Keterangan

- Struktur *Node*: Setiap node menyimpan nilai data dan pointer next yang menunjuk ke node berikutnya.
- Fungsi *createNode*: Digunakan untuk membuat node baru dengan nilai tertentu.
- Fungsi *insertAtFront*: Menambahkan node baru di depan linked list. Node baru ini akan menjadi node pertama, dan menunjuk ke node sebelumnya sebagai node kedua.
- Fungsi *insertAtEnd*: Menambahkan node baru di akhir linked list. Fungsi mencari node terakhir dan menghubungkannya ke node baru.
- Fungsi *printList*: Mencetak seluruh isi linked list dalam format "nilai -> nilai -> ...".

Output

```

main.cpp [unguided 1] - Code::Blocks 20.03
Isi linked list: 5 -> 10 -> 20
Process returned 0 (0x0)   execution time : 0.259 s
Press any key to continue.

```

## - Menghapus Node pada Linked List

Code

```

1 #include <iostream>
2
3 struct Node {
4     int data;
5     Node* next;
6 };
7 // Fungsi untuk menambah node di depan
8 void addNodeAtFront(Node*& head, int value) {
9     Node* newNode = new Node();
10    newNode->data = value;
11    newNode->next = head;
12    head = newNode;
13 }
14 // Fungsi untuk menambah node di belakang
15 void addNodeAtBack(Node*& head, int value) {
16     Node* newNode = new Node();
17     newNode->data = value;
18     newNode->next = nullptr;
19
20     if (head == nullptr) {
21         head = newNode;
22     } else {
23         Node* temp = head;
24         while (temp->next != nullptr) {
25             temp = temp->next;
26         }
27         temp->next = newNode;
28     }
29 }
30 // Fungsi untuk menghapus node dengan nilai tertentu
31 void deleteNode(Node*& head, int value) {
32     Node* temp = head;
33     Node* prev = nullptr;
34     // Jika node yang akan dihapus adalah head
35     if (temp != nullptr && temp->data == value) {
36         head = temp->next; // Ubah head
37         delete temp; // Hapus node
38         return;
39     }

```



```

40
41 // Mencari node yang akan dihapus
42 while (temp != nullptr && temp->data != value) {
43     prev = temp;
44     temp = temp->next;
45 }
46 // Jika node tidak ditemukan
47 if (temp == nullptr) return;
48 // Menghapus node
49 prev->next = temp->next;
50 delete temp;
51 }
52 // Fungsi untuk mencetak linked list
53 void printList(Node* head) {
54     Node* temp = head;
55     while (temp != nullptr) {
56         std::cout << temp->data;
57         if (temp->next != nullptr) {
58             std::cout << " -> ";
59         }
60         temp = temp->next;
61     }
62     std::cout << std::endl;
63 }
64
65 int main() {
66     Node* head = nullptr;
67     // Menambah node
68     addNodeAtFront(head, 10);
69     addNodeAtBack(head, 20);
70     addNodeAtFront(head, 5);
71     // Menghapus node dengan nilai tertentu
72     deleteNode(head, 10);
73     // Mencetak linked list
74     std::cout << "Isi Linked List: ";
75     printList(head);
76     return 0;
77 }
78

```

## Keterangan

- Struktur Node: Struktur *Node* memiliki dua atribut: *data* (nilai node) dan *next* (pointer ke node berikutnya).
- Fungsi *addNodeAtFront*: Menambah node baru di depan linked list.
- Fungsi *addNodeAtBack*: Menambah node baru di belakang linked list.
- Fungsi *deleteNode*: Menghapus node dengan nilai tertentu dari linked list. Fungsi ini juga menangani kasus di mana node yang dihapus adalah head.
- Fungsi *printList*: Mencetak semua nilai di linked list dengan format yang diinginkan.
- Fungsi *main*: Mengatur alur program, menambah node, menghapus node, dan mencetak isi linked list.

## Output

```

C:\codeblocks\pertemuan4\ >
Isi Linked List: 5 -> 20
Process returned 0 (0x0)   execution time : 0.243 s
Press any key to continue.

```

- **Mencari dan menghitung panjang Linked List**

## Code

```

*main.cpp X
1 #include <iostream>
2
3 struct Node {
4     int data;
5     Node* next;
6 };
7 // Fungsi untuk menambah node di depan
8 void addNodeAtFront(Node*& head, int value) {
9     Node* newNode = new Node();
10    newNode->data = value;
11    newNode->next = head;
12    head = newNode;
13 }
14 // Fungsi untuk menambah node di belakang
15 void addNodeAtBack(Node*& head, int value) {
16     Node* newNode = new Node();
17     newNode->data = value;
18     newNode->next = nullptr;
19
20     if (head == nullptr) {
21         head = newNode;
22     } else {
23         Node* temp = head;
24         while (temp->next != nullptr) {
25             temp = temp->next;
26         }
27         temp->next = newNode;
28     }
29 }
30 // Fungsi untuk mencari node dengan nilai tertentu
31 bool searchNode(Node* head, int value) {
32     Node* temp = head;
33     while (temp != nullptr) {
34         if (temp->data == value) {
35             return true; // Node ditemukan
36         }
37         temp = temp->next;
38     }
39     return false; // Node tidak ditemukan

```

```

*main.cpp X
40 }
41 // Fungsi untuk menghitung panjang linked list
42 int countLength(Node* head) {
43     int count = 0;
44     Node* temp = head;
45     while (temp != nullptr) {
46         count++;
47         temp = temp->next;
48     }
49     return count;
50 }
51 // Fungsi untuk mencetak linked list
52 void printList(Node* head) {
53     Node* temp = head;
54     while (temp != nullptr) {
55         std::cout << temp->data;
56         if (temp->next != nullptr) {
57             std::cout << " -> ";
58         }
59         temp = temp->next;
60     }
61     std::cout << std::endl;
62 }
63
64 int main() {
65     Node* head = nullptr;
66     // Menambah node
67     addNodeAtFront(head, 10);
68     addNodeAtBack(head, 20);
69     addNodeAtFront(head, 5);
70     // Mencari node dengan nilai tertentu
71     int searchValue = 20;
72     if (searchNode(head, searchValue)) {
73         std::cout << "Node dengan nilai " << searchValue << " ditemukan." << std::endl;
74     } else {
75         std::cout << "Node dengan nilai " << searchValue << " tidak ditemukan." << std::endl;
76     }
77     // Menghitung panjang linked list
78     int length = countLength(head);
79
79     std::cout << "Panjang linked list: " << length << std::endl;
80     return 0;
81 }
82

```

## Keterangan

- **Struktur Node:** Struktur **Node** memiliki dua atribut: **data** (nilai node) dan **next** (pointer ke node berikutnya).
- **Fungsi addNodeAtFront:** Menambah node baru di depan linked list.
- **Fungsi addNodeAtBack:** Menambah node baru di belakang linked list.
- **Fungsi searchNode:** Mencari apakah sebuah nilai ada dalam linked list. Mengembalikan **true** jika ditemukan, dan **false** jika tidak.
- **Fungsi countLength:** Menghitung dan mengembalikan jumlah node dalam linked list.
- **Fungsi printList:** Mencetak semua nilai di linked list dengan format yang diinginkan.
- **Fungsi main:** Mengatur alur program, menambah node, mencari node, dan menghitung panjang linked list.

## Output

A screenshot of a terminal window with a dark background. The text displayed is as follows:

```
"C:\codeblocks\pertemuan4\q" x + v
Node dengan nilai 20 ditemukan.
Panjang linked list: 3
Process returned 0 (0x0) execution time : 0.234 s
Press any key to continue.
```

## 5. Kesimpulan

Kesimpulan dari praktikum single linked list menunjukkan bahwa struktur data ini sangat berguna untuk menyimpan koleksi elemen secara dinamis. Melalui implementasi berbagai operasi seperti penambahan, penghapusan, pencarian, dan perhitungan panjang linked list, kita dapat memahami bagaimana linked list bekerja dalam pengelolaan memori dan manipulasi data. Kelebihan utama dari single linked list adalah kemampuannya untuk mengalokasikan memori secara dinamis dan efisien dalam menambah atau menghapus elemen tanpa perlu memindahkan elemen lainnya. Namun, linked list juga memiliki kekurangan, seperti akses elemen yang lebih lambat dibandingkan array, karena memerlukan traversal dari awal hingga node yang diinginkan. Secara keseluruhan, praktik ini memberikan pemahaman yang lebih baik tentang penggunaan dan implementasi linked list dalam pemrograman.