

LAPORAN PRAKTIKUM
Modul 4
SINGLE LINKED LIST (BAGIAN PERTAMA)



Disusun Oleh:
Aulia Jasifa Br Ginting 2311104060
S1SE-07-02

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024

1. Tujuan

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

2. Landasan Teori

4.1 Linked List dengan Pointer

Linked list dengan pointer adalah sebuah struktur data di mana elemen-elemennya dihubungkan satu sama lain menggunakan pointer. Setiap elemen dalam linked list disebut **node**, dan setiap node berisi dua bagian utama:

1. **Data (Value)**: Bagian yang menyimpan informasi atau nilai dari elemen tersebut.
2. **Pointer (Next)**: Bagian yang menyimpan alamat atau referensi ke node berikutnya dalam urutan.

Dengan menggunakan pointer, setiap node dalam linked list dapat "menunjuk" ke node selanjutnya, sehingga menciptakan struktur data yang dinamis dan fleksibel. Berikut adalah beberapa karakteristik penting dari linked list dengan pointer:

4.2 Single Linked List

Single Linked List adalah salah satu jenis linked list di mana setiap node hanya memiliki satu pointer yang menunjuk ke node berikutnya dalam daftar. Struktur ini bersifat linear, yang berarti Anda dapat menelusuri linked list hanya dari satu arah, yaitu dari head (node pertama) ke tail (node terakhir), sampai menemukan null atau akhir dari daftar.

4.2.1 Single Linked List

A. Pembentukan List

Adalah proses pembuatan dan penambahan elemen-elemen (node) ke dalam linked list dari awal hingga membentuk struktur lengkap. Pembentukan linked list melibatkan pengaturan node pertama (head) dan penambahan node-node lainnya secara berurutan menggunakan pointer.

B. Pengalokasian Memori

Adalah proses reservasi atau penyediaan ruang memori yang diperlukan oleh suatu program atau data selama eksekusi program. Dalam pemrograman, pengalokasian memori digunakan untuk menyimpan variabel, objek, struktur data (seperti array dan linked list), dan elemen-elemen lainnya sehingga program dapat berjalan dengan benar.

C. Dealokasi

Adalah proses mengembalikan atau membebaskan kembali ruang memori yang sebelumnya telah dialokasikan untuk program atau struktur data, sehingga memori tersebut dapat digunakan lagi oleh sistem. Dalam bahasa pemrograman seperti C dan C++, dealokasi sangat penting untuk mencegah kebocoran memori (**memory leak**) yang terjadi ketika memori yang dialokasikan tidak pernah dibebaskan, menyebabkan sistem kehabisan memori yang tersedia.

D. Pengecekan List

Adalah proses untuk memeriksa atau menentukan kondisi tertentu dari linked list, seperti apakah list kosong, apakah suatu elemen terdapat dalam list, atau berapa banyak elemen yang ada dalam list. Pengecekan ini penting untuk memastikan integritas dan validitas operasi yang akan dilakukan pada linked list.

4.2.2 Insert

A. Insert First

Insert First pada **singly linked list** adalah operasi untuk menambahkan sebuah node baru di **awal** linked list. Ini berarti node baru akan menjadi node pertama (head) dari linked list, dan node sebelumnya yang menjadi head akan bergeser ke posisi kedua.

B. Insert Last

Insert Last pada **singly linked list** adalah operasi untuk menambahkan sebuah node baru di **akhir** linked list. Setelah operasi ini, node baru menjadi node terakhir dalam daftar, dan pointer next dari node sebelumnya yang tadinya merupakan node terakhir akan menunjuk ke node baru. Pointer next dari node baru akan diatur ke null, karena itu adalah akhir dari linked list.

C. Insert After

Insert After pada **singly linked list** adalah operasi untuk menambahkan sebuah node baru **setelah node tertentu** yang sudah ada dalam linked list. Dengan operasi ini, node baru akan dimasukkan tepat setelah node yang ditentukan, dan node baru akan menjadi penerus node tersebut.

4.2.3 View

Merupakan operasi dasar pada list yang menampilkan isi node/simpul dengan suatu penelusuran list. Mengunjungi setiap node kemudian menampilkan data yang tersimpan pada node tersebut.

3. Guided

1. Guided 1

Programnya

```

1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 // Struktur untuk simpul
6 struct simpul {
7     char nama[50];
8     struct simpul *next;
9 };
10
11 // Struktur untuk List
12 struct List {
13     simpul *head;
14     simpul *tail;
15 };
16
17 // Fungsi untuk menginisialisasi list
18 void init(List &l) {
19     l.head = NULL;
20     l.tail = NULL;
21 }
22
23 // Fungsi untuk menambah simpul ke list
24 void tambah_simpul(List &l, char *nama) {
25     struct simpul *baru = new simpul;
26     baru->nama = nama;
27     baru->next = NULL;
28     if (l.head == NULL) {
29         l.head = baru;
30         l.tail = baru;
31     } else {
32         l.tail->next = baru;
33         l.tail = baru;
34     }
35 }
36
37 // Fungsi untuk menghapus simpul dari list
38 void hapus_simpul(List &l, char *nama) {
39     struct simpul *baru = new simpul;
40     baru->nama = nama;
41     baru->next = NULL;
42     if (l.head == NULL) {
43         l.head = baru;
44         l.tail = baru;
45     } else {
46         l.tail->next = baru;
47         l.tail = baru;
48     }
49 }
50
51 // Fungsi untuk mencari simpul
52 int hitung(List l) {
53     struct simpul *p = l.head;
54     int jumlah = 0;
55     while (p != NULL) {
56         jumlah++;
57         p = p->next;
58     }
59     return jumlah;
60 }
61
62 // Fungsi untuk menampilkan list
63 void tampil(List l) {
64     if (l.head == NULL) {
65         cout << "List kosong" << endl;
66     } else {
67         struct simpul *p = l.head;
68         while (p != NULL) {
69             cout << p->nama << " ";
70             p = p->next;
71         }
72         cout << endl;
73     }
74 }
75
76 // Fungsi untuk menghapus list
77 void hapus(List l) {
78     if (l.head == NULL) {
79         cout << "List kosong" << endl;
80     } else {
81         struct simpul *p = l.head;
82         while (p != NULL) {
83             delete p;
84             p = p->next;
85         }
86         l.head = NULL;
87         l.tail = NULL;
88     }
89 }
90
91 // Fungsi untuk mencari simpul
92 int hitung(List l) {
93     struct simpul *p = l.head;
94     int jumlah = 0;
95     while (p != NULL) {
96         jumlah++;
97         p = p->next;
98     }
99     return jumlah;
100 }
101
102 // Fungsi untuk menampilkan list
103 void tampil(List l) {
104     if (l.head == NULL) {
105         cout << "List kosong" << endl;
106     } else {
107         struct simpul *p = l.head;
108         while (p != NULL) {
109             cout << p->nama << " ";
110             p = p->next;
111         }
112         cout << endl;
113     }
114 }
115
116 // Fungsi untuk menghapus list
117 void hapus(List l) {
118     if (l.head == NULL) {
119         cout << "List kosong" << endl;
120     } else {
121         struct simpul *p = l.head;
122         while (p != NULL) {
123             delete p;
124             p = p->next;
125         }
126         l.head = NULL;
127         l.tail = NULL;
128     }
129 }
130
131 // Fungsi untuk mencari simpul
132 int hitung(List l) {
133     struct simpul *p = l.head;
134     int jumlah = 0;
135     while (p != NULL) {
136         jumlah++;
137         p = p->next;
138     }
139     return jumlah;
140 }
141
142 // Fungsi untuk menampilkan list
143 void tampil(List l) {
144     if (l.head == NULL) {
145         cout << "List kosong" << endl;
146     } else {
147         struct simpul *p = l.head;
148         while (p != NULL) {
149             cout << p->nama << " ";
150             p = p->next;
151         }
152         cout << endl;
153     }
154 }
155
156 // Fungsi untuk menghapus list
157 void hapus(List l) {
158     if (l.head == NULL) {
159         cout << "List kosong" << endl;
160     } else {
161         struct simpul *p = l.head;
162         while (p != NULL) {
163             delete p;
164             p = p->next;
165         }
166         l.head = NULL;
167         l.tail = NULL;
168     }
169 }
170
171 // Fungsi untuk mencari simpul
172 int hitung(List l) {
173     struct simpul *p = l.head;
174     int jumlah = 0;
175     while (p != NULL) {
176         jumlah++;
177         p = p->next;
178     }
179     return jumlah;
180 }
181
182 // Fungsi untuk menampilkan list
183 void tampil(List l) {
184     if (l.head == NULL) {
185         cout << "List kosong" << endl;
186     } else {
187         struct simpul *p = l.head;
188         while (p != NULL) {
189             cout << p->nama << " ";
190             p = p->next;
191         }
192         cout << endl;
193     }
194 }
195
196 // Fungsi untuk menghapus list
197 void hapus(List l) {
198     if (l.head == NULL) {
199         cout << "List kosong" << endl;
200     } else {
201         struct simpul *p = l.head;
202         while (p != NULL) {
203             delete p;
204             p = p->next;
205         }
206         l.head = NULL;
207         l.tail = NULL;
208     }
209 }

```

Outputnya:

```
PS C:\Users\LENOVO\Documents\STUDYING\SEMESTER 1>
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!
```

2. Guided 2 Programnya

```

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     cout << endl;
88 }
89
90 int main() {
91     Node* head = nullptr; // Membuat list kosong
92
93     // Menambahkan elemen ke dalam list
94     insertFirst(head, 10); // Menambahkan elemen 10 di awal list
95     insertLast(head, 20); // Menambahkan elemen 20 di akhir list
96     insertLast(head, 30); // Menambahkan elemen 30 di akhir list
97
98     // Menampilkan isi list
99     cout << "Isi list: ";
100    printList(head);
101
102    // Menampilkan jumlah elemen
103    cout << "Jumlah elemen: " << countElements(head) << endl;
104
105    // Menghapus semua elemen dalam list
106    clearList(head);
107
108    // Menampilkan isi list setelah penghapusan
109    cout << "Isi list setelah penghapusan: ";
110    printList(head);
111
112    return 0;
113 }

```

Outputnya:

```
PS C:\Users\LENOVO\Documents\STUDYING\SEMESTER 3\  
Isi List: 10 20 30  
Jumlah elemen: 3  
  
Isi List setelah penghapusan: List kosong!
```

4. Unguided

1. Membuat Single Linked List

Buatlah program C++ untuk membuat sebuah single linked list dengan operasi dasar sebagai berikut:

- **Insert Node di Depan:** Fungsi untuk menambah node baru di awal linked list.
- **Insert Node di Belakang:** Fungsi untuk menambah node baru di akhir linked list.
- **Cetak Linked List:** Fungsi untuk mencetak seluruh isi linked list.

```
1 #include <iostream>
2 using namespace std;
3
4 // Struktur untuk node dalam linked list
5 struct Node {
6     int data;
7     Node* next;
8 };
9
10 class LinkedList {
11 private:
12     Node* head;
13
14 public:
15     LinkedList() : head(nullptr) {}
16
17     // Fungsi untuk menambah node di depan
18     void insertFront(int value) {
19         Node* newNode = new Node();
20         newNode->data = value;
21         newNode->next = head;
22         head = newNode;
23     }
24
25     // Fungsi untuk menambah node di belakang
26     void insertBack(int value) {
27         Node* newNode = new Node();
28         newNode->data = value;
29         newNode->next = nullptr;
30
31         if (head == nullptr) {
32             head = newNode;
33             return;
34         }
35
36         Node* temp = head;
37         while (temp->next != nullptr) {
38             temp = temp->next;
39         }
40         temp->next = newNode;
41     }
42
43     // Fungsi untuk mencetak linked list
44     void printList() {
45         Node* temp = head;
46         while (temp != nullptr) {
47             cout << temp->data;
48             if (temp->next != nullptr) {
49                 cout << " -> ";
50             }
51             temp = temp->next;
52         }
53         cout << endl;
54     }
55 };
56
57 int main() {
58     LinkedList list;
59
60     int choice, value;
61     do {
62         cout << "\nMenu:\n";
63         cout << "1. Tambah node di depan\n";
64         cout << "2. Tambah node di belakang\n";
65         cout << "3. Cetak linked list\n";
66         cout << "4. Keluar\n";
67         cout << "Pilihan Anda: ";
68         cin >> choice;
69
70         switch (choice) {
71             case 1:
72                 cout << "Masukkan nilai: ";
73                 cin >> value;
74                 list.insertFront(value);
75                 break;
76             case 2:
77                 cout << "Masukkan nilai: ";
78                 cin >> value;
79                 list.insertBack(value);
80                 break;
81             case 3:
82                 cout << "Linked List: ";
83                 list.printList();
84                 break;
85             case 4:
86                 cout << "Program selesai.\n";
87                 break;
88             default:
89                 cout << "Pilihan tidak valid.\n";
90         }
91     } while (choice != 4);
92
93     return 0;
94 }
```


Outputnya:

```
Menu:
1. Tambah node di depan
2. Tambah node di belakang
3. Cetak linked list
4. Keluar
Pilihan Anda: 1
Masukkan nilai: 10

Menu:
1. Tambah node di depan
2. Tambah node di belakang
3. Cetak linked list
4. Keluar
Pilihan Anda: 2
Masukkan nilai: 20

Menu:
1. Tambah node di depan
2. Tambah node di belakang
3. Cetak linked list
4. Keluar
Pilihan Anda: 1
Masukkan nilai: 5

Menu:
1. Tambah node di depan
2. Tambah node di belakang
3. Cetak linked list
4. Keluar
Pilihan Anda: 3
Linked List: 5 -> 10 -> 20
```

2. Menghapus Node pada Linked List

Buatlah program C++ yang dapat menghapus node tertentu dalam single linked list berdasarkan nilai yang diberikan oleh pengguna. Tugas ini mencakup operasi:

- **Delete Node dengan Nilai Tertentu:** Fungsi untuk menghapus node yang memiliki nilai tertentu.
- **Cetak Linked List:** Setelah penghapusan, cetak kembali isi linked list.

```
1 #include <iostream>
2 using namespace std;
3
4 // Struktur untuk node dalam linked list
5 struct Node {
6     int data;
7     Node* next;
8     Node(int val) : data(val), next(nullptr) {}
9 };
10
11 class LinkedList {
12 private:
13     Node* head;
14
15 public:
16     LinkedList() : head(nullptr) {}
17
18     // Fungsi untuk menambah node di depan
19     void tambahDepan(int nilai) {
20         Node* newNode = new Node(nilai);
21         newNode->next = head;
22         head = newNode;
23     }
24
25     // Fungsi untuk menambah node di belakang
26     void tambahBelakang(int nilai) {
27         Node* newNode = new Node(nilai);
28         if (!head) {
29             head = newNode;
30             return;
31         }
32         Node* temp = head;
33         while (temp->next) {
34             temp = temp->next;
35         }
36         temp->next = newNode;
37     }
38
39     // Fungsi untuk menghapus node dengan nilai tertentu
40     void hapusNode(int nilai) {
41         if (!head) return;
42
43         if (head->data == nilai) {
44             Node* temp = head;
45             head = head->next;
46             delete temp;
47             return;
48         }
49
50         Node* current = head;
51         Node* prev = nullptr;
52         while (current && current->data != nilai) {
53             prev = current;
54             current = current->next;
55         }
56
57         if (!current) return;
58
59         prev->next = current->next;
60         delete current;
61     }
62
63     // Fungsi untuk mencetak linked list
64     void cetakList() {
65         Node* temp = head;
66         while (temp) {
67             cout << temp->data;
68             if (temp->next) cout << " -> ";
69             temp = temp->next;
70         }
71         cout << endl;
72     }
73 };
74
75 int main() {
76     LinkedList list;
77     int pilihan, nilai;
78
79     while (true) {
80         cout << "\nMenu:\n";
81         cout << "1. Tambah node di depan\n";
82         cout << "2. Tambah node di belakang\n";
83         cout << "3. Hapus node dengan nilai tertentu\n";
84         cout << "4. Cetak linked list\n";
85         cout << "5. Keluar\n";
86         cout << "Pilih operasi: ";
87         cin >> pilihan;
88
89         switch (pilihan) {
90             case 1:
91                 cout << "Masukkan nilai: ";
92                 cin >> nilai;
93                 list.tambahDepan(nilai);
94                 break;
95             case 2:
96                 cout << "Masukkan nilai: ";
97                 cin >> nilai;
98                 list.tambahBelakang(nilai);
99                 break;
100             case 3:
101                 cout << "Masukkan nilai yang akan dihapus: ";
102                 cin >> nilai;
103                 list.hapusNode(nilai);
104                 break;
105             case 4:
106                 cout << "Tsi linked list: ";
107                 list.cetakList();
108                 break;
109             case 5:
110                 cout << "Program selesai.\n";
111                 return 0;
112             default:
113                 cout << "Pilihan tidak valid.\n";
114         }
115     }
116
117     return 0;
118 }
```

Outputnya:

```
Menu:
1. Tambah node di depan
2. Tambah node di belakang
3. Hapus node dengan nilai tertentu
4. Cetak linked list
5. Keluar
Pilih operasi: 1
Masukkan nilai: 10
```

```
Menu:
1. Tambah node di depan
2. Tambah node di belakang
3. Hapus node dengan nilai tertentu
4. Cetak linked list
5. Keluar
Pilih operasi: 2
Masukkan nilai: 20
```

```
Menu:
1. Tambah node di depan
2. Tambah node di belakang
3. Hapus node dengan nilai tertentu
4. Cetak linked list
5. Keluar
Pilih operasi: 1
Masukkan nilai: 5
```

```
Menu:
1. Tambah node di depan
2. Tambah node di belakang
3. Hapus node dengan nilai tertentu
4. Cetak linked list
5. Keluar
Pilih operasi: 3
Masukkan nilai yang akan dihapus: 10
```

```
Menu:
1. Tambah node di depan
2. Tambah node di belakang
3. Hapus node dengan nilai tertentu
4. Cetak linked list
5. Keluar
Pilih operasi: 4
Isi linked list: 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.

Programnya

```

1 #include <iostream>
2 using namespace std;
3
4 // Struktur untuk node linked list
5 struct Node {
6     int data;
7     Node* next;
8 }
9
10 // Constructor
11 Node(int value) {
12     data = value;
13     next = NULL;
14 }
15
16 class LinkedList {
17 private:
18     Node* head;
19
20 public:
21     // Constructor
22     LinkedList() {
23         head = NULL;
24     }
25
26     // Fungsi untuk menambah node di depan
27 void tambahDepan(int nilai) {
28     Node* newNode = new Node(nilai);
29     newNode->next = head;
30     head = newNode;
31     cout << "Node dengan nilai " << nilai << " ditambahkan di depan.\n";
32 }
33
34 // Fungsi untuk menambah node di belakang
35 void tambahBelakang(int nilai) {
36     Node* newNode = new Node(nilai);
37
38     if (head == NULL) {
39         head = newNode;
40         cout << "Node dengan nilai " << nilai << " ditambahkan di belakang.\n";
41         return;
42     }
43
44     Node* temp = head;
45     while (temp->next != NULL) {
46         temp = temp->next;
47     }
48     temp->next = newNode;
49     cout << "Node dengan nilai " << nilai << " ditambahkan di belakang.\n";
50 }
51
52 // Fungsi untuk mencari node dengan nilai tertentu
53 bool cariNode(int nilai) {
54     Node* current = head;
55     while (current != NULL) {
56         if (current->data == nilai) {
57             cout << "Node dengan nilai " << nilai << " ditemukan.\n";
58             return true;
59         }
60         current = current->next;
61     }
62     cout << "Node dengan nilai " << nilai << " tidak ditemukan.\n";
63     return false;
64 }
65
66 // Fungsi untuk menghitung panjang linked list
67 int hitungPanjang() {
68     int count = 0;
69     Node* current = head;
70     while (current != NULL) {
71         count++;
72         current = current->next;
73     }
74     cout << "Panjang linked list: " << count << endl;
75     return count;
76 }
77
78 int main() {
79     LinkedList list;
80     int pilihan, nilai;
81
82     do {
83         cout << "Menu Operasi Linked List:\n";
84         cout << "1. Tambah node di depan\n";
85         cout << "2. Tambah node di belakang\n";
86         cout << "3. Cari node\n";
87         cout << "4. Hitung panjang linked list\n";
88         cout << "5. Keluar\n";
89         cout << "Pilihan Anda: ";
90         cin >> pilihan;
91
92         switch (pilihan) {
93             case 1:
94                 cout << "Masukkan nilai: ";
95                 cin >> nilai;
96                 list.tambahDepan(nilai);
97                 break;
98
99             case 2:
100                 cout << "Masukkan nilai: ";
101                 cin >> nilai;
102                 list.tambahBelakang(nilai);
103                 break;
104
105             case 3:
106                 cout << "Masukkan nilai yang dicari: ";
107                 cin >> nilai;
108                 list.cariNode(nilai);
109                 break;
110
111             case 4:
112                 list.hitungPanjang();
113                 break;
114
115             case 5:
116                 cout << "Program selesai.\n";
117                 break;
118
119             default:
120                 cout << "Pilihan tidak valid!\n";
121         }
122     } while (pilihan != 5);
123
124     return 0;
125 }

```

Outputnya:

```
Menu Operasi Linked List:
1. Tambah node di depan
2. Tambah node di belakang
3. Cari node
4. Hitung panjang linked list
5. Keluar
Pilihan Anda: 1
Masukkan nilai: 10
Node dengan nilai 10 ditambahkan di depan.

Menu Operasi Linked List:
1. Tambah node di depan
2. Tambah node di belakang
3. Cari node
4. Hitung panjang linked list
5. Keluar
Pilihan Anda: 2
Masukkan nilai: 20
Node dengan nilai 20 ditambahkan di belakang.

Menu Operasi Linked List:
1. Tambah node di depan
2. Tambah node di belakang
3. Cari node
4. Hitung panjang linked list
5. Keluar
Pilihan Anda: 1
Masukkan nilai: 5
Node dengan nilai 5 ditambahkan di depan.

Menu Operasi Linked List:
1. Tambah node di depan
2. Tambah node di belakang
3. Cari node
4. Hitung panjang linked list
5. Keluar
Pilihan Anda: 3
Masukkan nilai yang dicari: 20
Node dengan nilai 20 ditemukan.

Menu Operasi Linked List:
1. Tambah node di depan
2. Tambah node di belakang
3. Cari node
4. Hitung panjang linked list
5. Keluar
Pilihan Anda: 4
Panjang linked list: 3
```

5. Kesimpulan

Pada materi ini saya dapat memahami dan cara penggunaan Linked List struktur data yang sangat berguna, terutama ketika kita tidak tahu sebelumnya berapa banyak data yang akan disimpan atau ketika kita sering melakukan penambahan dan penghapusan elemen di tengah data. Serta dapat memahami mengenai kelebihan dan kekurangannya sebelum memilih struktur data dalam suatu aplikasi.