

**LAPORAN PRAKTIKUM**  
**Modul IV**  
**“SINGLE LINKED LIST (BAGIAN PERTAMA)”**



**Disusun Oleh:**  
**Dewi Atika Muthi -2211104042**  
**SE-07-02**

**Dosen:**  
**Wahyu Andi Saputra**

**PROGRAM STUDI S1 SOFTWARE ENGINEERING**  
**FAKULTAS INFORMATIKA**  
**TELKOM UNIVERSITY**  
**PURWOKERTO**  
**2024**

## 1. Tujuan

Tujuan praktikum ini adalah:

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.

## 2. Landasan Teori

Linked list adalah struktur data yang terdiri dari serangkaian elemen data yang saling terhubung. Setiap elemen (node) dalam linked list memiliki dua bagian utama: data dan pointer ke elemen berikutnya.

- Data: Menyimpan informasi yang ingin disimpan dalam list.
- Pointer: Menunjuk ke node berikutnya dalam list.

Single Linked List adalah jenis linked list di mana setiap node hanya memiliki satu pointer yang menunjuk ke node berikutnya.

Karakteristik Single Linked List:

- Hanya memerlukan satu pointer per node.
- Node terakhir menunjuk ke NULL.
- Hanya dapat melakukan traversal ke arah depan.
- Lebih efisien dalam operasi penyisipan dan penghapusan di tengah list dibandingkan array.

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).

## 3. Guided

### A. Pembentukan Komponen-Komponen List

#### a) Create Empty List

Biasanya nama fungsi yang digunakan `createList()`. Fungsi ini akan mengeset nilai awal list yaitu `first(list)` dan `last(list)` dengan nilai

```
void CreateList(List &L) {  
    first(L) = nullptr;  
}
```

Nil.

b) Allocation

Adalah proses memberikan bagian seseorang dari sejumlah total sesuatu untuk digunakan dengan cara tertentu, dalam kasus ini adalah proses untuk mengalokasikan memori untuk setiap elemen data ke dalam list. Biasanya menggunakan fungsi `alokasi()`.

```
address alokasi(infotype x) {
    address P = new ElmList;
    info(P) = x;
    next(P) = nullptr;
    return P;
}
```

c) Deallocation

Dealokasi adalah untuk menghapus sebuah memori address yang tersimpan atau yang sudah teralokasikan. Dalam pemrograman C menggunakan sintak `free`, dan di dalam C++ menggunakan sintak `delete`.

```
void dealokasi(address &P) {
    delete P;
}
```

d) Pengecekan List

Ini adalah fungsi untuk mengecek apakah list tersebut kosong atau tidak, fungsi yang digunakan adalah `isEmpty()`.

B. Insert Operation

a) Insert First

Adalah metode untuk memasukkan elemen data ke dalam list yang diletakkan **di awal list**

```
void insertFirst(List &L, address P) {
    next(P) = first(L);
    first(L) = P;
}
```

b) Insert Last

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

```
void insertLast(List &L, address P) {
    if (isEmpty(L)) {
        insertFirst(L, P);
    } else {
        address Q = first(L);
        while (next(Q) != nullptr) {
            Q = next(Q);
        }
        next(Q) = P;
    }
}
```

c) Insert After

Merupakan metode untuk memasukkan data ke dalam list yang diletakkan **setelah node tertentu** yang ditunjukkan oleh user

```
void insertAfter(List &L, address Prec, address P) {
    if (Prec != nullptr) {
        next(P) = next(Prec);
        next(Prec) = P;
    }
}
```

C. View

The printInfo function traverses the list from the first node to the end, printing the data of each node.

```
void printInfo(List L) {
    address P = first(L);
    while (P != nullptr) {
        cout << info(P) << " ";
        P = next(P);
    }
    cout << endl;
}
```

D. Delete Operation

1) Delete First

Yaitu operasi pengambilan atau penghapusan sebuah elemen pada awal list

```
void deleteFirst(List &L, address &P) {
    P = first(L);
    first(L) = next(first(L));
    next(P) = nullptr;
}
```

2) Delete Last

Yaitu proses pengambilan atau penghapusan sebuah elemen pada akhir list

```
void deleteLast(List &L, address &P) {
    if (next(first(L)) == nullptr) {
        deleteFirst(L, P);
    } else {
        address Q = first(L);
        while (next(next(Q)) != nullptr) {
            Q = next(Q);
        }
        P = next(Q);
        next(Q) = nullptr;
    }
}
```

```
}
```

### 3) Delete After

Yaitu proses pengambilan atau penghapusan node setelah node tertentu

```
void deleteAfter(List &L, address Prec, address &P) {  
    if (Prec != nullptr) {  
        P = next(Prec);  
        next(Prec) = next(P);  
        next(P) = nullptr;  
    }  
}
```

### E. Update

Operasi pada list untuk mengupdate data yang ada di dalam list, diawali dengan pencarian data yang akan kita update.

```
void updateElement(List &L, infotype x, infotype newInfo) {  
    address P = first(L);  
    while (P != nullptr && info(P) != x) {  
        P = next(P);  
    }  
    if (P != nullptr) {  
        info(P) = newInfo;  
    }  
}
```

## F. PRAKTIKUM 1

Sourcecode:

```
1  #include <iostream>  
2  #include <cstring>  
3  using namespace std;  
4  
5  // Deklarasi Struct untuk mahasiswa  
6  struct mahasiswa {  
7      char nama[30];  
8      char nim[10];  
9  };  
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 // Pengecekan apakah list kosong  
27 bool isEmpty() {  
28     return head == nullptr;  
29 }
```

```

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 }

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 }

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 }

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 }

102 // Tampilkan List
103 void tampil() {
104     cout << "\n=== Daftar Mahasiswa ===\n" << endl;
105     Node *current = head;
106     if (!isEmpty()) {
107         while (current != nullptr) {
108             cout << "Nama: " << current->data.nama
109                 << ", NIM: " << current->data.nim << endl;
110             current = current->next;
111         }
112     } else {
113         cout << "List masih kosong!" << endl;
114     }
115     cout << "\n===== \n" << endl;
116 }

```

```

118 // Harus List
119 void clearList() {
120     Node *current = head;
121     while (current != nullptr) {
122         Node *hapus = current;
123         current = current->next;
124         delete hapus;
125     }
126     head = tail = nullptr;
127     cout << "List berhasil terhapus!" << endl;
128 }

130 // Main function
131 int main() {
132     init();
133
134     // Contoh data mahasiswa
135     mahasiswa m1 = {"Alice", "123456"};
136     mahasiswa m2 = {"Bob", "654321"};
137     mahasiswa m3 = {"Charlie", "112233"};
138
139     cout << "Menambahkan Alice di depan:" << endl;
140     insertDepan(m1);
141     tampil();
142
143     cout << "Menambahkan Bob di belakang:" << endl;
144     insertBelakang(m2);
145     tampil();
146
147     cout << "Menambahkan Charlie di depan:" << endl;
148     insertDepan(m3);
149     tampil();
150
151     cout << "Menghapus elemen dari depan:" << endl;
152     hapusDepan();
153     tampil();
154
155     cout << "Menghapus elemen dari belakang:" << endl;
156     hapusBelakang();
157     tampil();
158
159     // Menghapus seluruh list
160     clearList();
161
162     return 0;
163 }

```

Output:

```

C:\StrukturData\madul2_KAC
Menambahkan Alice di depan:
==== Daftar Mahasiswa ====
Nama: Alice, NIM: 123456
=====
Menambahkan Bob di belakang:
==== Daftar Mahasiswa ====
Nama: Alice, NIM: 123456
Nama: Bob, NIM: 654321
=====

```

```

=====
Menambahkan Charlie di depan:

==== Daftar Mahasiswa ====

Nama: Charlie, NIM: 112233
Nama: Alice, NIM: 123456
Nama: Bob, NIM: 654321

=====

Menghapus elemen dari depan:

==== Daftar Mahasiswa ====

Nama: Alice, NIM: 123456
Nama: Bob, NIM: 654321

=====

Menghapus elemen dari belakang:

==== Daftar Mahasiswa ====

Nama: Alice, NIM: 123456

=====

List berhasil terhapus!

Process returned 0 (0x0)   execution time : 0.270 s
Press any key to continue.
|

```

Penjelasan program:

Penambahan Data dengan `insertDepan(mahasiswa m)` dan `insertBelakang(mahasiswa m)`. lalu penghapusan Data dengan `hapusDepan()` dan `hapusBelakang()`. `clearList()`: untuk menghapus satu persatu elemen menggunakan `hapusDepan()` sampai list kosong. Dan menampilkan Data dengan `tampil()`: dengan format "Nama: [Nama], NIM: [NIM]".

## G. PRAKTIKUM 2

Sourcecode:

```

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 }

```



```

25 // Pengacekan apakah list kosong
26 bool isListEmpty(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 (isListEmpty(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 (isListEmpty(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 }

```

```

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:

```

C:\StrukturData\madul2_KAC/ x + v
Isi List: 10 20 30
Jumlah elemen: 3
Isi List setelah penghapusan: List kosong!

Process returned 0 (0x0)   execution time : 0.098 s
Press any key to continue.

```

### Deskripsi program:

Alur program di ini adalah membuat list kosong (head diinisialisasi sebagai nullptr). Kemudian menambahkan elemen ke list (Elemen 10 di awal list, elemen 20 dan 30 di akhir list.) Kemudian menampilkan isi list setelah penambahan elemen, menghitung dan menampilkan jumlah elemen dalam list, lalu menghapus seluruh elemen dari list dan membersihkan memori. Terakhir menampilkan isi list setelah semua elemen dihapus.

## 4. Unguided

1. Membuat ilustrasi SDT single linked list:

Source Code:

```

#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* next;
};

class LinkedList {

```

```

private:
    Node* head;

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

    void insertDepan(int nilai) {
        Node* newNode = new Node{nilai, head};
        head = newNode;
    }

    void insertBelakang(int nilai) {
        Node* newNode = new Node{nilai, nullptr};
        if (head == nullptr) {
            head = newNode;
            return;
        }
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
    }

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

int main() {
    LinkedList list;
    list.insertDepan(10);
    list.insertBelakang(20);
    list.insertDepan(5);
    list.cetakList();
    return 0;
}

```

Output run:

```

C:\StrukturData\madul2_KAC  ×  +  ▾
5 -> 10 -> 20
Process returned 0 (0x0)   execution time : 0.084 s
Press any key to continue.

```

Deskripsi program:

Inti dari program ini:

- Menggunakan struktur Node untuk merepresentasikan setiap elemen dalam linked list.
- Kelas LinkedList berisi operasi dasar:
  - insertDepan: Menambah node baru di awal list. (10 dan 5)
  - insertBelakang: Menambah node baru di akhir list. (20)
  - cetakList: Menampilkan seluruh isi list.

## 2. Menghapus Node pada Linked List

Source Code:

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

struct Node {
    int data;
    Node* next;
};

class LinkedList {
private:
    Node* head;

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

    void insertDepan(int nilai) {
        Node* newNode = new Node{nilai, head};
        head = newNode;
    }

    void insertBelakang(int nilai) {
        Node* newNode = new Node{nilai, nullptr};
        if (head == nullptr) {
            head = newNode;
            return;
        }
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
    }

    void hapusNode(int nilai) {
        if (head == nullptr) return;
        if (head->data == nilai) {
            Node* temp = head;
            head = head->next;
        }
    }
}
```

```

        delete temp;
        return;
    }
    Node* temp = head;
    while (temp->next != nullptr && temp->next->data !=
nilai) {
        temp = temp->next;
    }
    if (temp->next != nullptr) {
        Node* toDelete = temp->next;
        temp->next = temp->next->next;
        delete toDelete;
    }
}

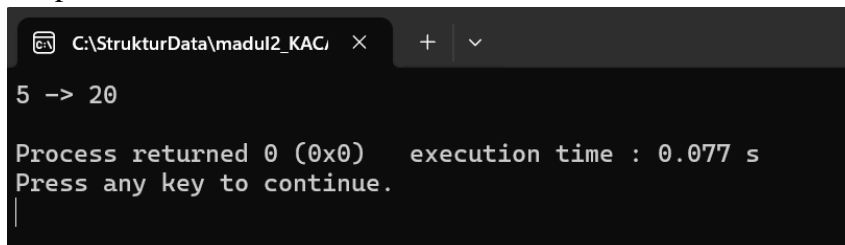
void cetakList() {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data;
        if (temp->next != nullptr) cout << " -> ";
        temp = temp->next;
    }
    cout << endl;
}

};

int main() {
    LinkedList list;
    list.insertDepan(10);
    list.insertBelakang(20);
    list.insertDepan(5);
    list.hapusNode(10);
    list.cetakList();
    return 0;
}

```

Output run:



```

C:\StrukturData\madul2_KAC>
5 -> 20

Process returned 0 (0x0)   execution time : 0.077 s
Press any key to continue.

```

Deskripsi program:

Program ini sama dengan program sebelumnya, namun ada penambahan operasi fungsi hapusNode, mencari node dengan nilai tertentu (10) lalu menghapusnya.

### 3. Mencari dan Mengitung Panjang Linked List

Source Code:

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

struct Node {
    int data;
    Node* next;
};

class LinkedList {
private:
    Node* head;

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

    void insertDepan(int nilai) {
        Node* newNode = new Node{nilai, head};
        head = newNode;
    }

    void insertBelakang(int nilai) {
        Node* newNode = new Node{nilai, nullptr};
        if (head == nullptr) {
            head = newNode;
            return;
        }
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
    }

    bool cariNode(int nilai) {
        Node* temp = head;
        while (temp != nullptr) {
            if (temp->data == nilai) return true;
            temp = temp->next;
        }
        return false;
    }

    int hitungPanjang() {
        int panjang = 0;
        Node* temp = head;
        while (temp != nullptr) {
            panjang++;
            temp = temp->next;
        }
    }
}
```

```

        return panjang;
    }
};

int main() {
    LinkedList list;
    list.insertDepan(10);
    list.insertBelakang(20);
    list.insertDepan(5);

    if (list.cariNode(20)) {
        cout << "Node dengan nilai 20 ditemukan." << endl;
    } else {
        cout << "Node dengan nilai 20 tidak ditemukan." <<
endl;
    }

    cout << "Panjang linked list: " << list.hitungPanjang()
<< endl;
    return 0;
}

```

Output run:

```

C:\StrukturData\madul2_KAC>
Node dengan nilai 20 ditemukan.
Panjang linked list: 3

Process returned 0 (0x0)   execution time : 0.094 s
Press any key to continue.

```

Deskripsi program:

Sama dengan program sebelumnya, program ini menambahkan operasi fungsi tambahan seperti fungsi cariNode untuk mencari nilai tertentu (20), dan fungsi hitungPanjang untuk menghitung jumlah total node dalam list (3).

## 5. Kesimpulan

Praktikum Single Linked List ini memberikan pemahaman mendasar tentang struktur data dinamis dalam pemrograman. Melalui implementasi operasi dasar seperti penyisipan, penghapusan, dan traversal, mahasiswa mengembangkan keterampilan penting dalam manipulasi pointer dan manajemen memori. Pengalaman ini tidak hanya meningkatkan kemampuan pemrograman, tetapi juga memperkuat pemikiran algoritmik dan pemecahan masalah, membentuk dasar untuk memahami struktur data yang lebih kompleks.