

**LAPORAN PRAKTIKUM STRUKTUR DATA**  
**PERTEMUAN 4**  
**SINGLE LINKED LIST (BAGIAN PERTAMA)**



**Nama :**

Reyner Atira Prasetyo (2311104057)

S1SE-07-02

**Dosen :**

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

**PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK**  
**FAKULTAS INFORMATIKA**  
**TELKOM UNIVERSITY PURWOKERTO**  
**2024**

## I. TUJUAN

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

## II. TOOL

1. Visual Studio Code
2. GCC

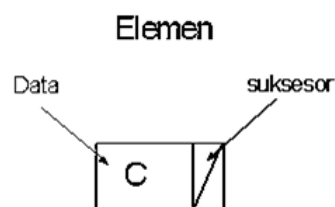
## III. DASAR TEORI

Linked list (biasa disebut list saja) adalah salah satu bentuk struktur data (representasi penyimpanan) berupa serangkaian elemen data yang saling berkait (berhubungan) dan bersifat fleksibel karena dapat tumbuh dan mengerut sesuai kebutuhan. Data yang disimpan dalam Linked list bisa berupa data tunggal atau data majemuk. Data tunggal merupakan data yang hanya terdiri dari satu data (variabel), misalnya: nama bertipe string. Sedangkan data majemuk merupakan sekumpulan data (record) yang di dalamnya terdiri dari berbagai tipe data, misalnya: Data Mahasiswa, terdiri dari Nama bertipe string, NIM bertipe long integer, dan Alamat bertipe string.

### 1. Single Linked List

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

Komponen elemen dalam single linked list:



Keterangan:

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

Data: informasi utama yang tersimpan dalam sebuah elemen.

Suksesor: bagian elemen yang berfungsi sebagai penghubung antar elemen.

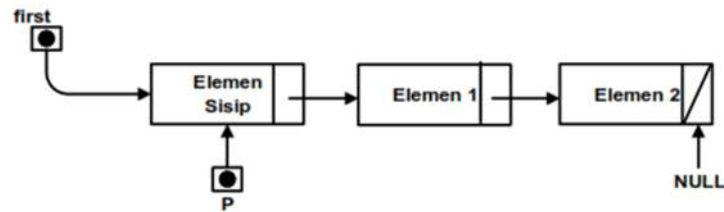
Sifat dari Single Linked list:

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

Istilah-istilah dalam Single Linked list :

1. first/head: pointer pada list yang menunjuk alamat elemen pertama list.
2. next: pointer pada elemen yang berfungsi sebagai successor (penunjuk) alamat elemen di depannya.
3. Null/Nil: artinya tidak memiliki nilai, atau tidak mengacu ke mana pun, atau kosong.
4. Node/simpul/elemen: merupakan tempat penyimpanan data pada suatu memori tertentu.

Gambaran sederhana single linked list dengan 3 elemen:



#### IV. GUIDED

##### 1. Guided1.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() {
    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;
        } else {
            baru->next = head;
            head = baru;
        }
    }

// Tambah Belakang
void insertBelakang(const mahasiswa &data) {
    Node *baru = new Node;
    baru->data = data;
    baru->next = nullptr;
    if (isEmpty()) {
        head = tail = baru;
    } else {
        tail->next = baru;
        tail = baru;
    }
}

// Hitung Jumlah List
int hitungList() {
    Node *current = head;
    int jumlah = 0;
    while (current != nullptr) {
        jumlah++;
        current = current->next;
    }
    return jumlah;
}

// Hapus Depan
void hapusDepan() {
    if (!isEmpty()) {
        Node *hapus = head;
        head = head->next;
        delete hapus;
        if (head == nullptr) {
            tail = nullptr; // Jika list menjadi kosong
        }
    } else {
        cout << "List kosong!" << endl;
    }
}

// Hapus Belakang
void hapusBelakang() {
    if (!isEmpty()) {
        if (head == tail) {
            delete head;
            head = tail = nullptr; // List menjadi kosong
        } else {
            Node *bantu = head;
            while (bantu->next != tail) {
                bantu = bantu->next;
            }
        }
    }
}

```

```

        }
        delete tail;
        tail = bantu;
        tail->next = nullptr;
    }
} else {
    cout << "List kosong!" << endl;
}
}

// Tampilkan List
void tampil() {
    Node *current = head;
    if (!isEmpty()) {
        while (current != nullptr) {
            cout << "Nama: " << current->data.nama << ", NIM: " <<
current->data.nim << endl;
            current = current->next;
        }
    } else {
        cout << "List masih kosong!" << endl;
    }
    cout << endl;
}

// Hapus List
void clearList() {
    Node *current = head;
    while (current != nullptr) {
        Node *hapus = current;
        current = current->next;
        delete hapus;
    }
    head = tail = nullptr;
    cout << "List berhasil terhapus!" << endl;
}

// Main function
int main() {
    init();

    // Contoh data mahasiswa
    mahasiswa m1 = {"Alice", "123456"};
    mahasiswa m2 = {"Bob", "654321"};
    mahasiswa m3 = {"Charlie", "112233"};

    // Menambahkan mahasiswa ke dalam list
    insertDepan(m1);
    tampil();
    insertBelakang(m2);
    tampil();
    insertDepan(m3);
    tampil();

    // Menghapus elemen dari list
    hapusDepan();

```

```

    tampil();
    hapusBelakang();
    tampil();

    // Menghapus seluruh list
    clearList();

    return 0;
}

```

Output :

```

Nama: Alice, NIM: 123456

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

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

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

Nama: Alice, NIM: 123456

List berhasil terhapus!
PS D:\PRAKTIKUM\Struktur Data\pertemuan4>

```

## 2. Guided2.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 isEmpty(Node* head) {
    return head == nullptr; // List kosong jika head adalah nullptr
}

```

```

// 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
            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

    // 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 :

```

PS D:\PRAKTIKUM\Struktur Data\pertemuan4> &
r.exe' '--stdin=Microsoft-MIEngine-In-5xgr310
icrosoft-MIEngine-Pid-1eocwh2e.s5f' '--dbgExe
Isi List: 10 20 30
Jumlah elemen: 3
Isi List setelah penghapusan: List kosong!
PS D:\PRAKTIKUM\Struktur Data\pertemuan4>

```

## V. UNGUIDED

### 1. Membuat Single Linked List



```

1  #include <iostream>
2  using namespace std;
3
4  struct Node {
5      int data;
6      Node* next;
7  };
8
9  // Function untuk menambah node di depan
10 void insertDepan(Node*& head, int nilai) {
11     Node* newNode = new Node();
12     newNode->data = nilai;
13     newNode->next = head;
14     head = newNode;
15 }
16
17 // Function untuk menambah node di belakang
18 void insertBelakang(Node*& head, int nilai) {
19     Node* newNode = new Node();
20     newNode->data = nilai;
21     newNode->next = nullptr;
22
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
34 // Function untuk mencetak seluruh isi Linked List
35 void cetakList(Node* head) {
36     Node* temp = head;
37     while (temp != nullptr) {
38         cout << temp->data;
39         if (temp->next != nullptr) {
40             cout << " -> ";
41         }
42         temp = temp->next;
43     }
44     cout << endl;
45 }
46
47 int main() {
48     Node* head = nullptr;
49
50     insertDepan(head, 10);
51     insertBelakang(head, 20);
52     insertDepan(head, 5);
53
54     cetakList(head); // Output: 5 -> 10 -> 20
55
56     return 0;
57 }

```

Output :

```
PS D:\PRAKTIKUM\Struktur Data\pertemuan4> &
r.exe' '--stdin=Microsoft-MIEngine-In-0uz53ob
icrosoft-MIEngine-Pid-v0dkwkud.54d' '--dbgExe:
5 -> 10 -> 20
PS D:\PRAKTIKUM\Struktur Data\pertemuan4>
```

## 2. Menghapus Node pada Linked List

```

1  #include <iostream>
2  using namespace std;
3
4  struct Node {
5      int data;
6      Node* next;
7  };
8
9  // Function untuk menambah node di depan
10 void insertDepan(Node*& head, int nilai) {
11     Node* newNode = new Node();
12     newNode->data = nilai;
13     newNode->next = head;
14     head = newNode;
15 }
16
17 // Function untuk menambah node di belakang
18 void insertBelakang(Node*& head, int nilai) {
19     Node* newNode = new Node();
20     newNode->data = nilai;
21     newNode->next = nullptr;
22
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
34 // Function untuk mencetak seluruh isi Linked List
35 void cetakList(Node* head) {
36     Node* temp = head;
37     while (temp != nullptr) {
38         cout << temp->data;
39         if (temp->next != nullptr) {
40             cout << " -> ";
41         }
42         temp = temp->next;
43     }
44     cout << endl;
45 }
46
47 // Function untuk menghapus node dengan nilai tertentu
48 void hapusNode(Node*& head, int nilai) {
49     if (head == nullptr) return;
50
51     // Jika node pertama adalah node yang akan dihapus
52     if (head->data == nilai) {
53         Node* temp = head;
54         head = head->next;
55         delete temp;
56         return;
57     }
58
59     Node* temp = head;
60     while (temp->next != nullptr && temp->next->data != nilai) {
61         temp = temp->next;
62     }
63
64     if (temp->next == nullptr) {
65         cout << "Node dengan nilai " << nilai << " tidak ditemukan." << endl;
66     } else {
67         Node* toDelete = temp->next;
68         temp->next = toDelete->next;
69         delete toDelete;
70     }
71 }
72
73 int main() {
74     Node* head = nullptr;
75
76     insertDepan(head, 10);
77     insertBelakang(head, 20);
78     insertDepan(head, 5);
79
80     hapusNode(head, 10);
81     cetakList(head); // Output: 5 -> 20
82
83     return 0;
84 }
85

```

Output :

```
PS D:\PRAKTIKUM\Struktur Data\pertemuan4> &
r.exe' '--stdin=Microsoft-MIEngine-In-pc2usc
icrosoft-MIEngine-Pid-alq113wn.kuq' '--dbgEx
5 -> 20
PS D:\PRAKTIKUM\Struktur Data\pertemuan4>
```

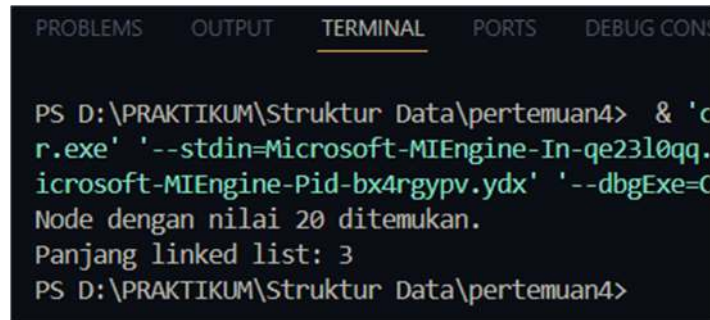
### 3. Mencari dan Menghitung Panjang Linked List

```

1  #include <iostream>
2  using namespace std;
3
4  struct Node {
5      int data;
6      Node* next;
7  };
8
9  // Function untuk menambah node di depan
10 void insertDepan(Node*& head, int nilai) {
11     Node* newNode = new Node();
12     newNode->data = nilai;
13     newNode->next = head;
14     head = newNode;
15 }
16
17 // Function untuk menambah node di belakang
18 void insertBelakang(Node*& head, int nilai) {
19     Node* newNode = new Node();
20     newNode->data = nilai;
21     newNode->next = nullptr;
22
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
34 // Function untuk mencetak seluruh isi Linked List
35 void cetakList(Node* head) {
36     Node* temp = head;
37     while (temp != nullptr) {
38         cout << temp->data;
39         if (temp->next != nullptr) {
40             cout << " -> ";
41         }
42         temp = temp->next;
43     }
44     cout << endl;
45 }
46
47 // Function untuk mencari node dengan nilai tertentu
48 bool cariNode(Node* head, int nilai) {
49     Node* temp = head;
50     while (temp != nullptr) {
51         if (temp->data == nilai) {
52             return true;
53         }
54         temp = temp->next;
55     }
56     return false;
57 }
58
59 // Function untuk menghitung panjang Linked List
60 int panjangList(Node* head) {
61     int panjang = 0;
62     Node* temp = head;
63     while (temp != nullptr) {
64         panjang++;
65         temp = temp->next;
66     }
67     return panjang;
68 }
69
70 int main() {
71     Node* head = nullptr;
72
73     insertDepan(head, 10);
74     insertBelakang(head, 20);
75     insertDepan(head, 5);
76
77     if (cariNode(head, 20)) {
78         cout << "Node dengan nilai 20 ditemukan." << endl;
79     } else {
80         cout << "Node dengan nilai 20 tidak ditemukan." << endl;
81     }
82
83     cout << "Panjang linked list: " << panjangList(head) << endl;
84
85     return 0;
86 }
87

```

Output :



```
PROBLEMS  OUTPUT  TERMINAL  PORTS  DEBUG CONSOLE

PS D:\PRAKTIKUM\Struktur Data\pertemuan4> & 'c
r.exe' '--stdin=Microsoft-MIEngine-In-qe23l0qq.
icrosoft-MIEngine-Pid-bx4rgypv.ydx' '--dbgExe=C
Node dengan nilai 20 ditemukan.
Panjang linked list: 3
PS D:\PRAKTIKUM\Struktur Data\pertemuan4>
```

## VI. KESIMPULAN

Pada praktikum ini, telah berhasil dibuat dan diimplementasikan tiga operasi dasar pada single linked list menggunakan bahasa C++:

1. **Membuat Linked List dan Operasi Insert** : Program dapat menambah node baru di awal dan di akhir linked list. Operasi ini berfungsi dengan baik dalam menambahkan elemen sesuai urutan yang diinginkan dan mencetak hasilnya.
2. **Penghapusan Node Berdasarkan Nilai** : Program dapat menghapus node berdasarkan nilai yang diberikan oleh pengguna. Proses penghapusan berjalan dengan baik, baik untuk node di awal, tengah, maupun akhir linked list, serta mampu menangani kasus ketika nilai yang dicari tidak ada.
3. **Pencarian dan Penghitungan Panjang Linked List** : Program dapat mencari node dengan nilai tertentu serta menghitung jumlah node yang ada dalam linked list. Fungsi pencarian dapat menemukan node yang diinginkan dan mencetak hasil pencarian, sedangkan fungsi penghitungan memberikan total panjang linked list dengan akurat.

Dengan demikian, praktikum ini memberikan pemahaman mendalam tentang implementasi single linked list dan berbagai operasinya, mulai dari penambahan, penghapusan, pencarian, hingga penghitungan panjang.