

LAPORAN PRAKTIKUM
Modul 4
Single Linked List Bagian 1



Disusun Oleh :
Satria Ariq Adelard Dompas/2211104033
SE 06 2

Asisten Praktikum :
Aldi Putra
Andini Nur Hidayah

Dosen Pengampu :
Wahyu Andi Saputra

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

1. Tujuan

- Mahasiswa mampu memahami penggunaan linked list dengan pointer operator dalam program.
- Mahasiswa mampu memahami operasi dasar dalam linked list.
- Mahasiswa mampu membuat program dengan menggunakan linked list dengan prototype yang ada.

2. Landasan Teori

a. Single Linked List

Single linked list adalah struktur data dasar yang terdiri dari rangkaian node, di mana setiap node berisi data dan referensi ke node berikutnya. Struktur ini efisien untuk operasi penyisipan dan penghapusan, tetapi memiliki kelemahan dalam akses data karena perlu penelusuran dari awal. Selain itu, linked list hanya memiliki penunjuk ke node berikutnya, sehingga tidak bisa mengakses node sebelumnya tanpa memulai dari awal. Struktur ini sering digunakan untuk antrian, tumpukan, dan tabel hash, serta berguna ketika ukuran data tidak pasti.

3. Guided

a. Linked List Guided 1

```
OUTPUT  DEBUG CONSOLE  PROBLEMS  TERMINAL  PORTS  COMMENTS
PS D:\Praktikum Struktur Data\Pertemuan4> cd 'd:\Praktikum Struktur Data\Pertemuan4\GUIDED\output'
PS D:\Praktikum Struktur Data\Pertemuan4\GUIDED\output> & .\modul_04_1.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!
PS D:\Praktikum Struktur Data\Pertemuan4\GUIDED\output> |
```

```

#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;
}

```

b. Linked List Guided 2

```

PS D:\Praktikum Struktur Data\Pertemuan4\GUIDED\output> cd 'd:\Praktikum Struktur Data\Pertemuan4\GUIDED\output'
PS D:\Praktikum Struktur Data\Pertemuan4\GUIDED\output> & .\modul_04_2.exe
Isi List: 10 20 30
Jumlah elemen: 3
Isi List setelah penghapusan: List kosong!
PS D:\Praktikum Struktur Data\Pertemuan4\GUIDED\output>

```

```

#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)
{
    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;
}

```

4. Unguided

- a. 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.

```
PS D:\Praktikum Struktur Data\Pertemuan4\GUIDED\output> cd 'd:\Praktikum Struktur Data\Pertemuan4\UNGUIDED\output'
PS D:\Praktikum Struktur Data\Pertemuan4\UNGUIDED\output> & .\soal_01.exe
Isi linked list: 5 -> 10 -> 20
PS D:\Praktikum Struktur Data\Pertemuan4\UNGUIDED\output> 
```

```
● ● ●

#include <iostream>
using namespace std;

// Struktur untuk node
struct Node {
    int data;
    Node *next;
};

// Fungsi untuk menambah node di depan
void insertDepan(Node *&head, int nilai) {
    Node *baru = new Node{nilai, head};
    head = baru;
}

// Fungsi untuk menambah node di belakang
void insertBelakang(Node *&head, int nilai) {
    Node *baru = new Node{nilai, nullptr};
    if (head == nullptr) {
        head = baru;
    } else {
        Node *temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = baru;
    }
}

// Fungsi untuk mencetak isi linked list
void cetakList(Node *head) {
    Node *temp = head;
    while (temp != nullptr) {
        cout << temp->data;
        if (temp->next != nullptr) cout << " -> ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    Node *head = nullptr;
    insertDepan(head, 10);
    insertBelakang(head, 20);
    insertDepan(head, 5);
    cout << "Isi linked list: ";
    cetakList(head);
    return 0;
}
```

- b. 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.

```
OUTPUT  DEBUG CONSOLE  PROBLEMS  TERMINAL  PORTS  COMMENTS

PS D:\Praktikum Struktur Data\Pertemuan4> cd 'd:\Praktikum Struktur Data\Pertemuan4\UNGUIDED\output'
PS D:\Praktikum Struktur Data\Pertemuan4\UNGUIDED\output> & .\soal_02.exe
Isi linked list: 5 -> 20
PS D:\Praktikum Struktur Data\Pertemuan4\UNGUIDED\output>

#include <iostream>
using namespace std;

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

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

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

void hapusNode(Node *&head, int nilai) {
    Node *temp = head, *prev = nullptr;
    while (temp != nullptr && temp->data != nilai) {
        prev = temp;
        temp = temp->next;
    }
    if (temp != nullptr) {
        if (prev != nullptr) prev->next = temp->next;
        else head = temp->next;
        delete temp;
    }
}

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

int main() {
    Node *head = nullptr;
    insertDepan(head, 10);
    insertBelakang(head, 20);
    insertDepan(head, 5);
    hapusNode(head, 10);
    cout << "Isi linked list: ";
    cetakList(head);
    return 0;
}
```


- c. 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.

```
OUTPUT  DEBUG CONSOLE  PROBLEMS  TERMINAL  PORTS  COMMENTS
PS D:\Praktikum Struktur Data\Pertemuan4> cd 'd:\Praktikum Struktur Data\Pertemuan4\UNGUIDED\output'
PS D:\Praktikum Struktur Data\Pertemuan4\UNGUIDED\output> & .\'soal_03.exe'
Node dengan nilai 20 ditemukan.
Panjang linked list: 3
PS D:\Praktikum Struktur Data\Pertemuan4\UNGUIDED\output>

#include <iostream>
using namespace std;

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

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

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

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

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

int main() {
    Node *head = nullptr;
    insertDepan(head, 10);
    insertBelakang(head, 20);
    insertDepan(head, 5);

    int nilaiDicari = 20;
    if (cariNode(head, nilaiDicari))
        cout << "Node dengan nilai " << nilaiDicari << " ditemukan." << endl;
    else
        cout << "Node dengan nilai " << nilaiDicari << " tidak ditemukan." << endl;

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

