LAPORAN PRAKTIKUM

Modul 04

“SINGLE LINKED LIST”

Disusun Oleh:

Faishal Arif Setiawan 2311104066

Kelas:

SE 07 02

Dosen :

WAHYU ANDI SAPUTRA, S.PD.,M.ENG

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

2.1 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*.
Setiap elemen dalam linked list disebut sebagai node, dan setiap node terdiri dari dua komponen:

- Data: Nilai yang disimpan di dalam node.
- Pointer (next**)**: Penunjuk ke node berikutnya dalam urutan linked list.

2.2. Single linked list
adalah struktur data linear di mana setiap elemen (node) hanya memiliki satu penunjuk (pointer) yang mengarah ke node berikutnya. Elemen pertama dalam linked list disebut head, dan node terakhir memiliki penunjuk yang bernilai null, yang menunjukkan bahwa node tersebut merupakan elemen terakhir dalam linked list.
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*

3.GUIDED

Program C++ di atas adalah implementasi linked list untuk menyimpan data mahasiswa menggunakan single linked list. Program ini menyediakan beberapa operasi dasar, seperti menambahkan node di depan dan belakang, menghapus node di depan dan belakang, menampilkan isi list, serta menghapus seluruh isi linked list.

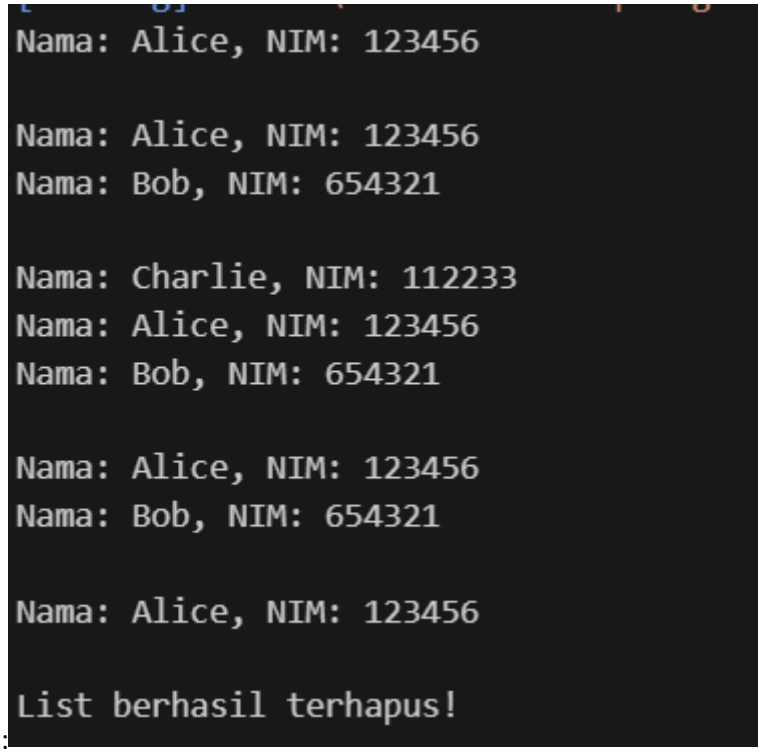
Struct mahasiswa:Menyimpan data mahasiswa berupa nama dan nim

Struct Node: Merepresentasikan node pada linked list, yang terdiri dari data mahasiswa dan pointer next untuk menunjuk ke node berikutnya.

init(): Inisialisasi list dengan mengatur head dan tail menjadi nullptr, menandakan bahwa list kosong.

isEmpty(): Memeriksa apakah linked list kosong. Mengembalikan true jika list kosong.

Output:



:

Program C++ di atas adalah implementasi **single linked list** sederhana yang mencakup beberapa operasi dasar, seperti menambahkan elemen di awal dan akhir list, menampilkan elemen, menghitung jumlah elemen, serta menghapus seluruh elemen list.

Struct Node: Digunakan untuk mendefinisikan node pada linked list, yang terdiri dari dua komponen:

- data: Menyimpan nilai elemen.

- next: Pointer yang menunjuk ke node berikutnya.

alokasi(): Fungsi untuk mengalokasikan memori bagi node baru. Jika berhasil, node baru akan diisi dengan nilai yang diberikan dan pointer next diatur ke nullptr.

dealokasi(): Fungsi untuk menghapus memori yang digunakan oleh node. Ini digunakan saat node dihapus.

isListEmpty(): Fungsi untuk memeriksa apakah linked list kosong dengan memeriksa apakah head bernilai nullptr.

4.UNGUIDED

1.

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

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

void insertDepan(Node*& head, int nilai) {
    Node* newNode = new Node();
    newNode->data = nilai;
    newNode->next = head;
    head = newNode;
}

void insertBelakang(Node*& head, int nilai) {
    Node* newNode = new Node();
    newNode->data = nilai;
    newNode->next = nullptr;

    if (head == nullptr) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

void cetakLinkedList(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: ";
    cetakLinkedList(head);

    return 0;
}
```

Output:

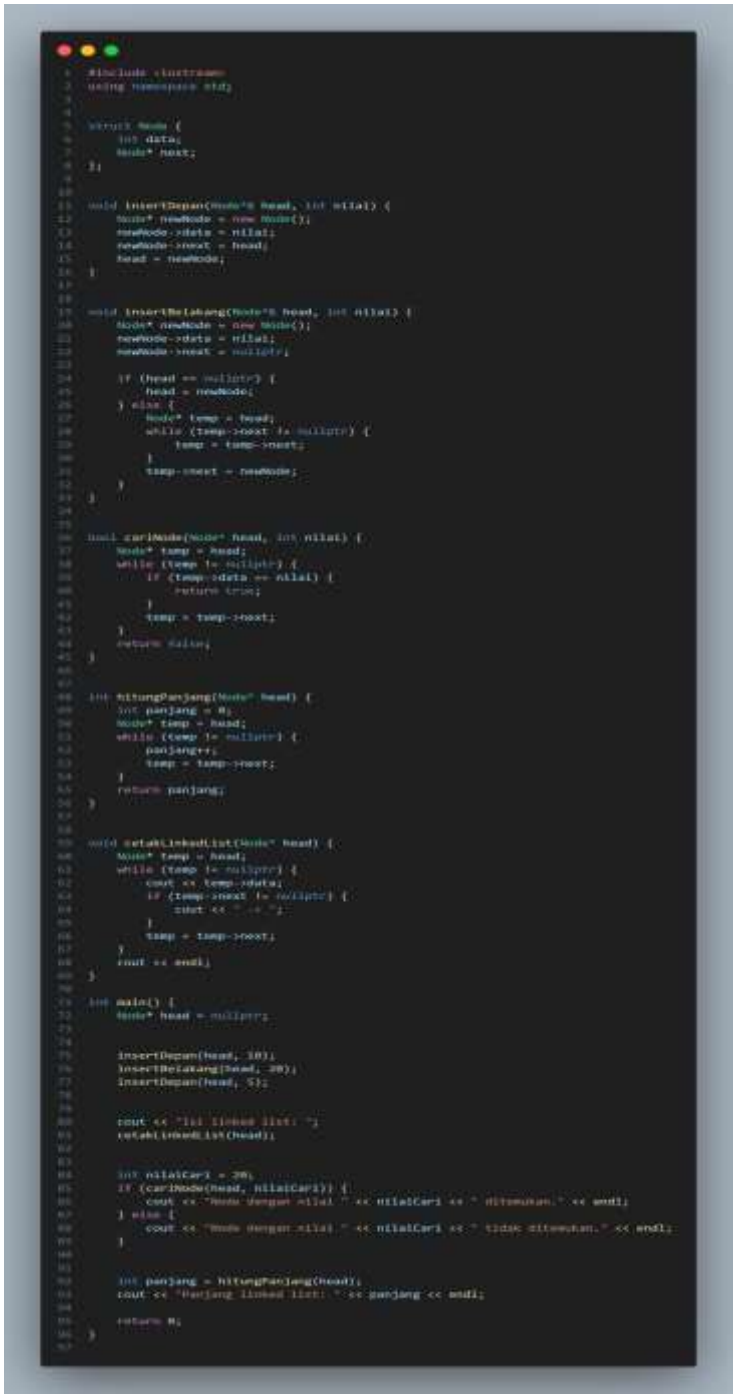
```
[Running] cd "d:\pemograman java\" && g++ unguidedliaprak4.cpp -o unguidedliaprak4 && "d:\pemograman java\"unguidedliaprak4
Isi linked list: 5 -> 10 -> 20
```

2.

Output:

```
[Running] cd "d:\pemograman java\" && g++ unguidedno2laprak4.cpp -o unguidedno2laprak4 && "d:\pemograman java\"unguidedno2laprak4
5 -> 20
```

3.



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

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

void insertDepan(Node*& head, int nilai) {
    Node* newNode = new Node();
    newNode->data = nilai;
    newNode->next = head;
    head = newNode;
}

void insertBelakang(Node*& head, int nilai) {
    Node* newNode = new Node();
    newNode->data = nilai;
    newNode->next = nullptr;

    if (head == nullptr) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

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;
}

void cetakLinkedList(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: ";
    cetakLinkedList(head);

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

    int panjang = hitungPanjang(head);
    cout << "Panjang linked list: " << panjang << endl;

    return 0;
}
```

Output:

```
Node dengan nilai 20 ditemukan.
Panjang linked list: 3
```

5.KESIMPULAN

Single linked list memberikan fleksibilitas yang lebih tinggi dibandingkan array dalam hal pengelolaan memori dinamis. Meskipun memiliki beberapa kekurangan, seperti akses yang lebih lambat dan penggunaan memori tambahan untuk pointer, linked list sangat efisien dalam operasi penambahan dan penghapusan elemen. Memahami dan mengimplementasikan linked list dengan benar adalah langkah penting dalam pemahaman dasar struktur data dan algoritma.