LAPORAN PRAKTIKUM

PERTEMUAN 3

04_SINGLELINKLIST(BAGIANPERTAMA)

Nama :

Ilham Lii Assidaq (2311104068)

Dosen :

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

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. 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

II. TOOL

Dev C++

III. DASAR TEORI

Linked list adalah struktur data yang terdiri dari serangkaian elemen yang saling terhubung dan bersifat fleksibel, karena ukurannya dapat bertambah atau berkurang sesuai kebutuhan. Data dalam linked list bisa berupa data tunggal (seperti nama bertipe string) atau data majemuk (seperti data mahasiswa yang terdiri dari nama, NIM, dan alamat). Linked list biasanya diimplementasikan menggunakan pointer karena beberapa alasan:

1. Array bersifat statis, sedangkan pointer dinamis.
2. Elemen-elemen dalam linked list saling terhubung, sehingga lebih mudah menggunakan pointer.
3. Pointer lebih sesuai dengan sifat fleksibel linked list.
4. Pointer lebih efisien dibanding array dalam menangani linked list.
5. Array lebih cocok untuk kumpulan data dengan jumlah elemen yang sudah diketahui sejak awal.

Dalam implementasinya, pengaksesan elemen pada *Linked list* dengan *pointer* bisa menggunakan
(->) atau tanda titik (.).
Model-model dari ADT *Linked list* yang kita pelajari adalah :
1. *Single Linked list*
2. *Double Linked list*
3. *Circular Linked list*
4. *Multi Linked list*
5. *Stack* (Tumpukan)
6. *Queue* (Antrian)
7. *Tree*

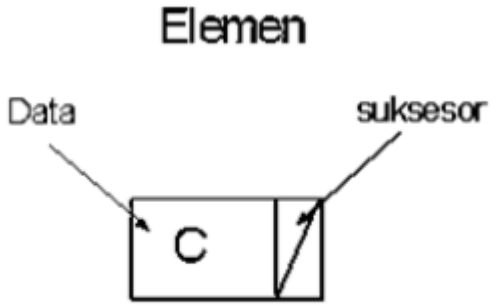
8. *Graph*

Setiap model ADT *Linked list* di atas memiliki karakteristik tertentu dan dalam penggunaannya
disesuaikan dengan kebutuhan.

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*)

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



Gambar 4-1 Elemen *Single Linked list*

Keterangan:

- Elemen: Segmen data dalam sebuah list.
- Data: Informasi utama yang tersimpan dalam elemen.
- Suksesor: Bagian elemen yang menghubungkan elemen satu dengan lainnya.

Sifat Single Linked List:

1. Hanya membutuhkan satu pointer.
2. Node terakhir menunjuk ke Nil, kecuali pada list circular.
3. Pembacaan hanya bisa dilakukan maju.
4. Pencarian dilakukan secara berurutan jika data tidak terurut.
5. Memudahkan penyisipan atau penghapusan di tengah list.

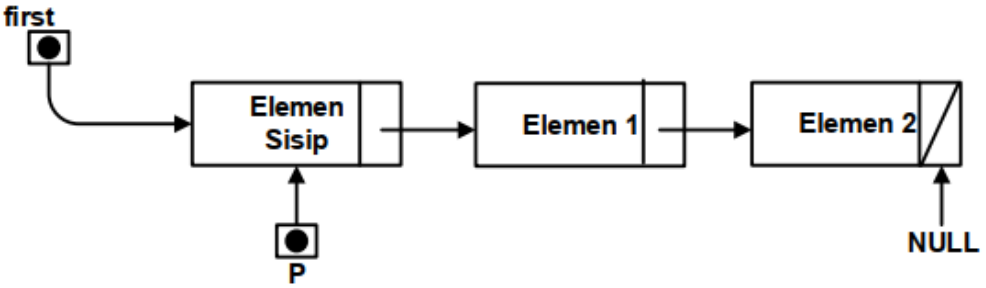
Istilah dalam Single Linked List:

1. First/Head: Pointer yang menunjuk ke elemen pertama list.
2. Next: Pointer dalam elemen yang mengarahkan ke elemen berikutnya.
3. Null/Nil: Menunjukkan tidak ada nilai atau tidak mengarah ke mana pun.
4. Node/Elemen/Simpul: Tempat penyimpanan data di memori.

Gambaran sederhana single linked list dengan elemen kosong:

Gambar 4-2 *Single Linked list* dengan Elemen Kosong

Gambaran sederhana *single linked list* dengan 3 elemen:



Gambar 4-3 *Single Linked list* dengan 3 Elemen

IV. GUIDED

1. Gui1

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

struct mahasiswa {
    char nama[30];
    char nim[10];
};

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

Node *head;
Node *tail;

void init() {
    head = NULL;
    tail = NULL;
}

bool isEmpty() {
    return head == NULL;
}

void insertDepan(const mahasiswa &data) {
    Node *baru = new Node;
    baru->data = data;
    baru->next = NULL;
    if (isEmpty()) {
        head = tail = baru;
    } else {
        baru->next = head;
        head = baru;
    }
}

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

int hitungList() {
    Node *current = head;
    int jumlah = 0;
    while (current != NULL) {
        jumlah++;
        current = current->next;
    }
    return jumlah;
}

void hapusDepan() {
    if (!isEmpty()) {
        Node *hapus = head;
        head = head->next;
```

```
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!

--------------------------------
Process exited after 9.691 seconds with return value 0
Press any key to continue . . .
```

```
        delete hapus;
        if (head == NULL) {
            tail = NULL;
        }
    } else {
        cout << "List kosong euy!" << endl;
    }
}

void hapusBelakang() {
    if (!isEmpty()) {
        if (head == tail) {
            delete head;
            head = tail = NULL;
        } else {
            Node *bantu = head;
            while (bantu->next != tail) {
                bantu = bantu->next;
            }
            delete tail;
            tail = bantu;
            tail->next = NULL;
        }
    } else {
        cout << "List kosong!" << endl;
    }
}

void tampil() {
    Node *current = head;
    if (!isEmpty()) {
```

```
            cout << "Nama: " << current->data.nama << ", NIM: " << current->data.nim << endl;
            current = current->next;
        }
    } else {
        cout << "List masih kosong!" << endl;
    }
}


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


int main() {
    init();

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


    insertDepan(m1);
    tampil();
    insertBelakang(m2);
    tampil();
    insertDepan(m3);
    tampil();


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


    clearList();

    return 0;
}
```

2. Gui2

```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 != NULL) { // Jika alokasi berhasil
        newNode->data = value; // Mengisi data node
        newNode->next = NULL; // Set next ke NULL
    }
    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 == NULL; // List kosong jika head adalah NULL
}

// Menambahkan elemen di awal list
void insertFirst(Node* &head, int value) {
    Node* newNode = alokasi(value); // Alokasi memori untuk elemen baru
    if (newNode != NULL) {
        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 != NULL) {
        if (isListEmpty(head)) {  // Jika list kosong
            head = newNode;       // Elemen baru menjadi elemen pertama
        } else {
            Node* temp = head;
            while (temp->next != NULL) { // 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 != NULL) {  // 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 != NULL) {
        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 != NULL) {
        Node* temp = head;  // Simpan pointer ke node saat ini
        head = head->next;  // Pindahkan ke node berikutnya
        dealokasi(temp);    // Dealokasi node
    }
}

int main() {
    Node* head = NULL;  // 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;
}
```

```
Isi List: 10 20 30
Jumlah elemen: 3
Isi List setelah penghapusan: List kosong!

--------------------------------
Process exited after 9.609 seconds with return value 0
Press any key to continue . . .
```

V. UNGUIDED

1. Membuat Single Linked List

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

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

Node* head = NULL;

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

void insertBelakang(int nilai) {
    Node* newNode = new Node();
    newNode->data = nilai;
    newNode->next = NULL;

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


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

int main() {
    insertDepan(10);
    insertBelakang(20);
    insertDepan(5);
    cetakList();

    return 0;
}
```

```
D:\ITT SM 3\04_SINGLELINKLI
5 -> 10 -> 20

--------------------------------
Process exited after 6.704 seconds with return value 0
Press any key to continue . . .
```

2. Menghapus Node pada Linked List

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

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

Node* head = NULL;

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

void insertBelakang(int nilai) {
    Node* newNode = new Node();
    newNode->data = nilai;
    newNode->next = NULL;

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

void hapusNode(int nilai) {
    if (head == NULL) {
        cout << "List kosong, tidak ada yang bisa dihapus!" << endl;
        return;
    }

    if (head->data == nilai) {
        Node* temp = head;
        head = head->next;
        delete temp;
        cout << "Node dengan nilai " << nilai << " berhasil dihapus!" << endl;
        return;
    }

    Node* current = head;
    Node* prev = NULL;
    while (current != NULL && current->data != nilai) {
        prev = current;
        current = current->next;
    }

    if (current == NULL) {
        cout << "Node dengan nilai " << nilai << " tidak ditemukan!" << endl;
        return;
    }

    prev->next = current->next;
    delete current;
    cout << "Node dengan nilai " << nilai << " berhasil dihapus!" << endl;
}

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

int main() {
    insertDepan(10);
    insertBelakang(20);
    insertDepan(5);
    cetakList();

    hapusNode(10);
    cetakList();

    return 0;
}
```

```
5 -> 10 -> 20
Node dengan nilai 10 berhasil dihapus!
5 -> 20

--------------------------------
Process exited after 6.102 seconds with return value 0
Press any key to continue . . .
```

3. Mencari dan Menghitung Panjang Linked List

```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 = NULL;

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

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

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

int main() {
    Node* head = NULL;

    insertDepan(head, 10);
    insertBelakang(head, 20);
    insertDepan(head, 5);

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

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

    return 0;
}
```

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

--------------------------------
Process exited after 6.05 seconds with return value 0
Press any key to continue . . .
```

KESIMPULAN

Kesimpulannya adalah praktikum kali ini menjelaskan konsep dasar dan implementasi dari Single Linked List, sebuah struktur data yang menggunakan pointer untuk menyimpan elemen data secara dinamis dan fleksibel. Dengan memahami penggunaan pointer, pengguna dapat melakukan operasi dasar seperti pembuatan, penyisipan, penghapusan, penelusuran, dan pembaruan elemen dalam list. Single Linked List memiliki keunggulan dalam penyisipan dan penghapusan elemen di tengah list, serta mengelola data yang tidak terurut.