LAPORAN PRAKTIKUM

PERTEMUAN 4

SINGLE LINKED LIST

Nama :

Yehuda Melvin Sugiarto (2311104055)

Dosen :

Wahyu Andi Saputra

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. TUJUAN

a. Memahami penggunaan *linked list* dengan *pointer* operator- operator dalam program.
b. Memahami operasi-operasi dasar dalam *linked list*.
c. Membuat program dengan menggunakan *linked list* dengan *prototype* yang ada

II. TOOL

a. Laptop

b. Code::Block

c. C++

III. DASAR TEORI

1. Linked List with Pointer

Linked list, atau sering disebut list, adalah salah satu bentuk struktur data yang terdiri dari rangkaian elemen data yang saling terhubung. Struktur ini bersifat fleksibel karena dapat berkembang dan menyusut sesuai kebutuhan. Data yang disimpan dalam linked list bisa berupa data tunggal atau data kompleks. Data tunggal adalah data yang hanya terdiri dari satu variabel, misalnya nama dengan tipe string. Sementara itu, data kompleks adalah kumpulan data yang terdiri dari berbagai tipe, seperti data mahasiswa yang mencakup Nama (string), NIM (long integer), dan Alamat (string). Linked list dapat diimplementasikan menggunakan Array dan Pointer. Namun untuk kali ini, kita akan menggunakan pointer untuk praktikum kita karena beberapa alasan, antara lain :
1. Array bersifat tetap, sedangkan pointer bersifat dinamis.

2. Dalam linked list, elemen-elemen data saling terhubung, sehingga lebih praktis menggunakan pointer.

3. Fleksibilitas linked list lebih sesuai dengan karakteristik pointer yang dapat disesuaikan berdasarkan kebutuhan.

4. Array lebih sulit digunakan dalam pengelolaan linked list, sedangkan pointer lebih mudah digunakan.

5. Array lebih cocok untuk kumpulan data dengan jumlah elemen yang sudah diketahui sejak awal.

Dalam implementasinya, pengaksesan elemen pada Linked list menggunakan pointer dapat dilakukan dengan (->) atau tanda titik (.). Beberapa model ADT Linked list yang kita pelajari meliputi:

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

Masing-masing model ADT Linked list memiliki karakteristik yang berbeda dan penggunaannya disesuaikan dengan kebutuhan. Secara umum, operasi-operasi ADT pada Linked list meliputi:

1. Pembuatan dan inisialisasi list (Create List)
2. Penyisipan elemen dalam list (Insert)
3. Penghapusan elemen dari list (Delete)
4. Penelusuran dan penampilan elemen list (View)
5. Pencarian elemen dalam list (Searching)
6. Pengubahan nilai elemen dalam list (Update)

2. Single Linked List

Single Linked list merupakan model ADT Linked list yang hanya memiliki satu arah pointer. SLL memiliki komponen seperti:
a. Data : Informasi utama suatu elemen
b. Elemen : segmen data yang terdapat dalam suatu list
c. Suksesor : Penghubung antar elemen

Ciri-ciri dari Single Linked List antara lain:

1. Hanya membutuhkan satu pointer.
2. Node terakhir mengarah ke Nil, kecuali pada list circular.
3. Hanya bisa dibaca secara maju.
4. Pencarian dilakukan secara berurutan jika data tidak terurut.
5. Lebih mudah untuk menyisipkan atau menghapus elemen di tengah list.

Istilah-istilah yang digunakan dalam Single Linked List:

1. first/head: pointer yang menunjuk ke elemen pertama dalam list.
2. next: pointer di setiap elemen yang berfungsi sebagai penunjuk ke elemen berikutnya.
3. Null/Nil: tidak memiliki nilai, tidak mengarah ke mana pun, atau kosong.
4. Node/simpul/elemen: tempat penyimpanan data pada lokasi memori tertentu.

3. Pembentukan Komponen List

a. **Pembentukan List:** Proses ini bertujuan untuk membuat list baru. Biasanya, digunakan fungsi *createList()* yang menginisialisasi nilai awal *first(list)* dan *last(list)* menjadi *Nil*.
b. **Pengalokasian Memori:** Memori dialokasikan untuk setiap elemen data dalam list. Fungsi *alokasi()* digunakan untuk ini. Pada bahasa C, pengalokasian memori dilakukan dengan `malloc`, sedangkan pada C++, bisa menggunakan `new` untuk menyederhanakan proses. Contoh sintaks di C:

```
P = (address) malloc ( sizeof (elmlist));
```

Contoh di C++:

```
P = new elmlist;
```

c. **Dealokasi:** Penghapusan memori yang telah dialokasikan dilakukan menggunakan *free* di C dan *delete* di C++:

- Sintaks C: free(p);

- Sintaks C++: delete p;

d. **Pengecekan List:** Fungsi *isEmpty()* digunakan untuk mengecek apakah list kosong. Fungsi ini mengembalikan nilai *true* jika list kosong, dan *false* jika tidak kosong.

4. Insert
Insert merupakan methode untuk memasukan elemen kedalam list, terdapat 3 jenis insert, yaitu:
a. Insert First : Input elemen diawal list (paling awal)
b. Insert After : Input elemen ditengah list setelah node tertentu
c. Insert Last : Input elemen diakhir list (paling akhir)

5. View
Sebuah metode yang digunakan untuk menampilkan list yang dimiliki.

6. Delete
merupakan metode yang digunakan untuk menghapus sebuah elemen atau node yang dimiliki suatu list, ada 4 jenis, yaitu:
a. Delete First : Menghapus elemen diawal list
b. Delete After : Menghapus elemen ditengah list setelah node tertentu
c. Delete Last : Menghapus elemen diakhir list
d. Delete Elemen : Digunakan untuk menghapus dan membebaskan memori yang dipakai suatu elemen. Memiliki 2 fungsi, yaitu:
a. fungsi dealokasi(P)
b. fungsi delAll(L)

7. Update
metode yang digunakan untuk mengubah isi/data dari suatu elemen.

IV. GUIDE

A. Guided 1

```
#include <iostream>
#include <cstring>
#include <iomanip>

using namespace std;

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

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

Node *head;
Node *tail;

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

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

void insertDepan(const mahasiswa &data) {
    Node *baru = new Node;
    baru->data = data;
    baru->next = head;
    head = baru;
```

```
    if (tail == nullptr) {
        tail = baru;
    }
}

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;
    }
}

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

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

```
        if (head == nullptr) {
            tail = nullptr;
        }
    } else {
        cout << "List Kosong!!" << endl;
    }
}

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

void tampil() {
    Node *current = head;
    if (!isEmpty()) {
        cout << "Daftar Mahasiswa:" << endl;
        cout << setw(30) << left << "Nama" << setw(10) << left << "NIM" << endl;
        cout << string(40, '-') << endl;
```

```
void tampil() {
    Node *current = head;
    if (!isEmpty()) {
        cout << "Daftar Mahasiswa:" << endl;
        cout << setw(30) << left << "Nama" << setw(10) << left << "NIM" << endl;
        cout << string(40, '-') << endl;
        while (current != nullptr) {
            cout << setw(30) << left << current->data.nama << setw(10) << left << current->data.nim << endl;
            current = current->next;
        }
    } else {
        cout << "List masih kosong!" << endl;
    }
}

void clearList() {
    Node *current = head;
    while (current != nullptr) {
        Node *hapus = current;
        current = current->next;
        delete hapus;
    }
    head = tail = nullptr;
    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;
}
```

```
Daftar Mahasiswa:
Nama                            NIM
---------------------------------------
Alice                           123456
Daftar Mahasiswa:
Nama                            NIM
---------------------------------------
Alice                           123456
Bob                             654321
Daftar Mahasiswa:
Nama                            NIM
---------------------------------------
Charlie                         112233
Alice                           123456
Bob                             654321
Daftar Mahasiswa:
Nama                            NIM
---------------------------------------
Alice                           123456
Bob                             654321
Daftar Mahasiswa:
Nama                            NIM
---------------------------------------
Alice                           123456
List berhasil terhapus!
```

B. Guided 2

```cpp
#include <iostream>

using namespace std;

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

Node* alokasi(int value) {
    Node* newNode = new Node;
    if (newNode != nullptr) {
        newNode->data = value;
        newNode->next = nullptr;
    } else {
        cout << "Memory allocation failed!" << endl;
    }
    return newNode;
}

void dealokasi(Node* node) {
    delete node;
}

bool isListEmpty(Node* head) {
    return head == nullptr;
}

void insertFirst(Node*& head, int value) {
    Node* newNode = alokasi(value);
    if (newNode != nullptr) {
        newNode->next = head;
        head = newNode;
    }
}

void insertLast(Node*& head, int value) {
    Node* newNode = alokasi(value);
    if (newNode != nullptr) {
        if (isListEmpty(head)) {
            head = newNode;
        } else {
            Node* temp = head;
            while (temp->next != nullptr) {
                temp = temp->next;
            }
            temp->next = newNode;
        }
    }
}
```

```cpp
void printList(Node* head) {
    if (isListEmpty(head)) {
        cout << "List kosong!" << endl;
    } else {
        Node* temp = head;
        while (temp != nullptr) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
}

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

void clearList(Node*& head) {
    while (head != nullptr) {
        Node* temp = head;
        head = head->next;
        dealokasi(temp);
    }
    head = nullptr;
}

int main() {
    Node* head = nullptr;

    insertFirst(head, 10);
    insertLast(head, 20);
    insertLast(head, 30);

    cout << "Isi List: ";
    printList(head);

    cout << "Jumlah elemen: " << countElements(head) << endl;

    clearList(head);

    cout << "Isi List setelah penghapusan: ";
    printList(head);

    return 0;
}
```

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

V. UNGUIDED

1. Unguided 1

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

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

Node* createNode(int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;
    return newNode;
}

void insertFirst(Node*& head, int value) {
    Node* newNode = createNode(value);
    newNode->next = head;
    head = newNode;
}

void InsertLast(Node*& head, int value) {
    Node* newNode = createNode(value);

    if (head == nullptr) {
        head = newNode;
        return;
    }

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

```
    }
    temp->next = newNode;
}

void printList(Node* head) {
    if (head == nullptr) {
        cout << "Linked List kosong." << endl;
        return;
    }

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

int main() {
    Node* head = nullptr;

    insertFirst(head, 10);
    InsertLast(head, 20);
    insertFirst(head, 5);

    cout << "Isi Linked List: ";
    printList(head);

    return 0;
}
```

```
Isi Linked List: 5 -> 10 -> 20
```

2. Unguided 2

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

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

Node* createNode(int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;
    return newNode;
}

void insertFirst(Node*& head, int value) {
    Node* newNode = createNode(value);
    newNode->next = head;
    head = newNode;
}

void insertLast(Node*& head, int value) {
    Node* newNode = createNode(value);

    if (head == nullptr) {
        head = newNode;
        return;
    }
```

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

void printList(Node* head) {
    if (head == nullptr) {
        cout << "Linked List kosong." << endl;
        return;
    }

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

void deleteNode(Node*& head, int value) {
    if (head == nullptr) {
        cout << "Linked List kosong, tidak bisa menghapus." << endl;
        return;
    }

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

    Node* current = head;
    Node* previous = nullptr;

    while (current != nullptr && current->data != value) {
        previous = current;
        current = current->next;
    }

    if (current == nullptr) {
        cout << "Node dengan nilai " << value << " tidak ditemukan." << endl;
        return;
    }

    previous->next = current->next;
    delete current;
    cout << "Node dengan nilai " << value << " berhasil dihapus." << endl;
}
```

```
int main() {
    Node* head = nullptr;

    insertFirst(head, 10);
    insertLast(head, 20);
    insertFirst(head, 5);

    cout << "Isi Linked List sebelum penghapusan: ";
    printList(head);

    deleteNode(head, 10);

    cout << "Isi Linked List setelah penghapusan: ";
    printList(head);

    return 0;
}
```

```
Isi Linked List sebelum penghapusan: 5 -> 10 -> 20
Node dengan nilai 10 berhasil dihapus.
Isi Linked List setelah penghapusan: 5 -> 20
```

3. Unguided 3

```
#include <iostream>

using namespace std;

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

class LinkedList {
private:
    Node* head;

public:
    LinkedList() {
        head = nullptr;
    }

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

    void insertLast(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = nullptr;
```

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

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

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

int main() {
    LinkedList list;

    list.insertFirst(10);
    list.insertLast(20);
    list.insertFirst(5);

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

    cout << "Panjang linked list: " << list.length() << endl;

    return 0;
}
```

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

VI. KESIMPULAN

Dari laporan ini dapat disimpulkan bahwa linked list/list merupakan suatu bentuk struktur data dengan rangkaian elemen. Dan pada laporan ini kita dapat mengetahui salah satu jenis dari linked list yaitu single linked list, yang dimana kita juga jadi mengetahui cara mengimplementasikan SLL pada kodingan kita, dan kita juga mengetahui cara menambahkan dan menghapus elemen pada linked list.