

**LAPORAN PRAKTIKUM
STRUKTUR DATA
MODUL 4
“SINGLE LINKED LIST (BAGIAN PERTAMA) ”**



Disusun Oleh:
Dhiya Ulhaq Ramadhan 2211104053
Kelas :
S1SE-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 dalam program.
- Memahami operasi-operasi dasar dalam linked list.
- Membuat program dengan menggunakan linked list dengan prototype yang ada.

2. Landasan Teori

Linked list adalah struktur data yang terdiri dari serangkaian elemen data yang saling terhubung dan bersifat fleksibel. Single Linked List merupakan jenis linked list dengan satu arah pointer, di mana setiap elemen (node) terdiri dari data dan pointer ke elemen berikutnya. Komponen utamanya meliputi elemen, data, dan suksesor (pointer). Istilah penting mencakup first/head, next, dan Null/Nil.

Operasi dasar Single Linked List meliputi pembuatan list, penyisipan (insert), penghapusan (delete), penelusuran (view), pencarian, dan pengubahan (update) elemen. Implementasinya memerlukan fungsi seperti `createList()`, `alokasi()`, `dealokasi()`, dan `isEmpty()`. Penyisipan dan penghapusan dapat dilakukan di awal, akhir, atau tengah list.

3. Guided

Guided 1 SLL

Source code :

```
#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 :

```
"D:\bersama berkarya\SEMES" X
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 returned 0 (0x0)
Press any key to continue.
```

1. Inisialisasi List:

- Program dimulai dengan memanggil `init()` untuk membuat list kosong.

2. Menambahkan Data:

- Tiga data mahasiswa (`m1`, `m2`, `m3`) dibuat.
- `insertDepan(m1)` menambahkan Alice ke depan list.
- `tampil()` menampilkan list (hanya berisi Alice).
- `insertBelakang(m2)` menambahkan Bob ke belakang list.
- `tampil()` menampilkan list (Alice, Bob).
- `insertDepan(m3)` menambahkan Charlie ke depan list.
- `tampil()` menampilkan list (Charlie, Alice, Bob).

3. Menghapus Data:

- `hapusDepan()` menghapus node depan (Charlie).
- `tampil()` menampilkan list (Alice, Bob).
- `hapusBelakang()` menghapus node belakang (Bob).
- `tampil()` menampilkan list (hanya berisi Alice).

4. Membersihkan List:

- `clearList()` menghapus seluruh isi list.

Guided 2 SLL

Source code :

```
#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
Node* alokasi(int value) {
    Node* newNode = new Node; // Alokasi memori
    if (newNode != nullptr) { // Jika alokasi berhasil
        newNode->data = value; // Mengisi data
        newNode->next = nullptr; // Set next ke null
    }
    return newNode; // Mengembalikan pointer ke node baru
}

// Fungsi untuk dealokasi memori node
void dealokasi(Node* node) {
    delete node; // Mengembalikan memori ke sistem
}

// Pengecekan apakah list kosong
bool isListEmpty(Node* head) {
    return head == nullptr; // List kosong jika head null
}

// Menambahkan elemen di awal list
void insertFirst(Node* &head, int value) {
    Node* newNode = alokasi(value); // Alokasi memori
    if (newNode != nullptr) {
        newNode->next = head; // Menghubungkan ke list
        head = newNode; // Menetapkan head ke node baru
    }
}
```



```
// Menambahkan elemen di akhir list
void insertLast(Node* &head, int value) {
    Node* newNode = alokasi(value); // Aloka
    if (newNode != nullptr) {
        if (isListEmpty(head)) { // Jika li
            head = newNode; // Elemen
        } else {
            Node* temp = head;
            while (temp->next != nullptr) {
                temp = temp->next;
            }
            temp->next = newNode; // Menamba
        }
    }
}

// Menampilkan semua elemen dalam list
void printList(Node* head) {
    if (isListEmpty(head)) {
        cout << "List kosong!" << endl;
    } else {
        Node* temp = head;
        while (temp != nullptr) { // Selama
            cout << temp->data << " "; // Me
            temp = temp->next; // Melanjutka
        }
        cout << endl;
    }
}
```

```
int countElements(Node* head) {
    int count = 0;
    Node* temp = head;
    while (temp != nullptr) {
        count++; // Menambah jumlah
        temp = temp->next; // Melanjutkan ke e
    }
    return count; // Mengembalikan ju
}

void clearList(Node* &head) {
    while (head != nullptr) {
        Node* temp = head; // Simpan pointer
        head = head->next; // Pindahkan ke no
        dealokasi(temp); // Dealokasi node
    }
}

int main() {
    Node* head = nullptr; // Membuat list kos

    // Menambahkan elemen ke dalam list
    insertFirst(head, 10); // Menambahkan elem
    insertLast(head, 20); // Menambahkan elem
    insertLast(head, 30); // Menambahkan elem

    // Menampilkan isi list
    cout << "Isi List: ";
    printList(head);

    // Menampilkan jumlah elemen
    cout << "Jumlah elemen: " << countElements

    // Menghapus semua elemen dalam list
    clearList(head);

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

    return 0;
}
```

Output :

```
"D:\bersama berkarya\SEMES" x
Isi List: 10 20 30
Jumlah elemen: 3
Isi List setelah penghapusan:

Process returned 0 (0x0)   ex
Press any key to continue.
|
```

Penjelasan

- insertFirst(head, 10) membuat node baru dengan nilai 10 dan menjadikannya head.
- insertLast(head, 20) traverses list hingga akhir dan menambahkan node baru dengan nilai 20.
- insertLast(head, 30) melakukan hal yang sama untuk nilai 30.
- printList(head) menelusuri list dari head, mencetak setiap nilai.
- countElements(head) menghitung node saat menelusuri list.
- clearList(head) menghapus setiap node satu per satu, membebaskan memori.
- Pemanggilan printList(head) terakhir menemukan list kosong dan mencetak pesan yang sesuai.

4. Unguided

- Membuat Single Linked List 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.Contoh input dan output: Input:
 1. Tambah node di depan (nilai: 10)
 2. Tambah node di belakang (nilai: 20)
 3. Tambah node di depan (nilai: 5)
 4. Cetak linked list Output: 5 -> 10 -> 20

Jawaban :

```
#include <iostream>

struct Node {
    int data;
    Node* next;
    Node(int val) : data(val), next(nullptr) {}
};

class LinkedList {
private:
    Node* head;

public:
    LinkedList() : head(nullptr) {}

    void insertFront(int val) {
        Node* newNode = new Node(val);
        newNode->next = head;
        head = newNode;
    }

    void insertBack(int val) {
        Node* newNode = new Node(val);
        if (!head) {
            head = newNode;
            return;
        }
        Node* temp = head;
        while (temp->next) {
            temp = temp->next;
        }
        temp->next = newNode;
    }

    void printList() {
        Node* temp = head;
        while (temp) {
            std::cout << temp->data;
            if (temp->next) std::cout << " -> ";
            temp = temp->next;
        }
        std::cout << std::endl;
    }

    ~LinkedList() {
        while (head) {
            Node* temp = head;
            head = head->next;
            delete temp;
        }
    }
};

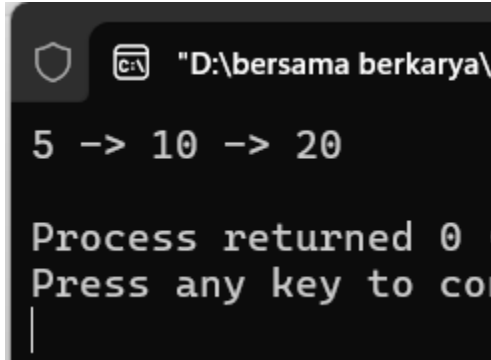
int main() {
    LinkedList list;

    list.insertFront(10);
    list.insertBack(20);
    list.insertFront(5);

    list.printList();

    return 0;
}
```

Output :



```

D:\bersama berkarya\
5 -> 10 -> 20

Process returned 0
Press any key to continue

```

2. Menghapus Node pada Linked List 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.

Contoh input/output: Input:

1. Tambah node di depan (nilai: 10)
2. Tambah node di belakang (nilai: 20)
3. Tambah node di depan (nilai: 5)
4. Hapus node dengan nilai (nilai: 10) 5. Cetak linked list Output: 5 -> 20

Jawaban :

```

#include <iostream>
struct Node {
    int data;
    Node* next;
    Node(int val) : data(val), next(nullptr) {}
};

class LinkedList {
private:
    Node* head;
public:
    LinkedList() : head(nullptr) {}

    void insertFront(int val) {
        Node* newNode = new Node(val);
        newNode->next = head;
        head = newNode;
    }

    void insertBack(int val) {
        Node* newNode = new Node(val);
        if (!head) {
            head = newNode;
            return;
        }
        Node* temp = head;
        while (temp->next) {
            temp = temp->next;
        }
    }
};

```

```

    temp->next = newNode;
}

void deleteNode(int val) {
    if (!head) return;

    if (head->data == val) {
        Node* temp = head;
        head = head->next;
        delete temp;
        return;
    }

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

    if (current) {
        prev->next = current->next;
        delete current;
    }
}

void printList() {
    Node* temp = head;
    while (temp) {
        std::cout << temp->data;

        if (temp->next) std::cout << " -> ";
        temp = temp->next;
    }
    std::cout << std::endl;
}

~LinkedList() {
    while (head) {
        Node* temp = head;
        head = head->next;
        delete temp;
    }
}

};

int main() {
    LinkedList list;

    list.insertFront(10);
    list.insertBack(20);
    list.insertFront(5);

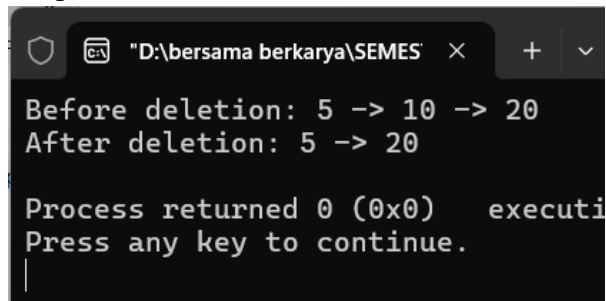
    std::cout << "Before deletion: ";
    list.printList();

    list.deleteNode(10);

    std::cout << "After deletion: ";
    list.printList();
    return void LinkedList::printList()
}

```

Output :



```
"D:\bersama berkarya\SEMES" x + v
Before deletion: 5 -> 10 -> 20
After deletion: 5 -> 20

Process returned 0 (0x0) executing
Press any key to continue.
```

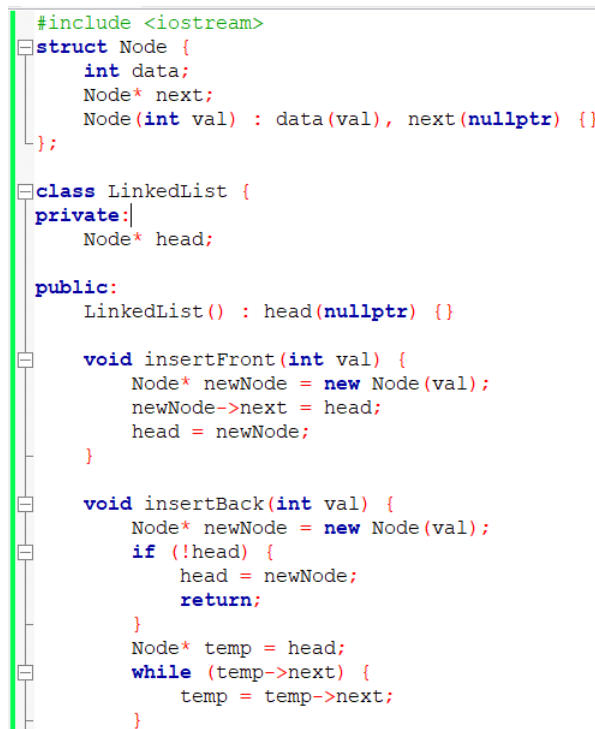
3. Mencari dan Menghitung Panjang Linked List Buatlah program C++ yang dapat melakukan operasi berikut:

- 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.

Contoh input/output: Input:

1. Tambah node di depan (nilai: 10)
2. Tambah node di belakang (nilai: 20)
3. Tambah node di depan (nilai: 5)
4. Cari node dengan nilai 20
5. Cetak panjang linked list Output: Node dengan nilai 20 ditemukan. Panjang linked list: 3

Jawaban :



```
#include <iostream>
struct Node {
    int data;
    Node* next;
    Node(int val) : data(val), next(nullptr) {}
};

class LinkedList {
private:
    Node* head;

public:
    LinkedList() : head(nullptr) {}

    void insertFront(int val) {
        Node* newNode = new Node(val);
        newNode->next = head;
        head = newNode;
    }

    void insertBack(int val) {
        Node* newNode = new Node(val);
        if (!head) {
            head = newNode;
            return;
        }
        Node* temp = head;
        while (temp->next) {
            temp = temp->next;
        }
    }
};
```

```

        temp->next = newNode;
    }

    bool searchNode(int val) {
        Node* temp = head;
        while (temp) {
            if (temp->data == val) return true;
            temp = temp->next;
        }
        return false;
    }

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

    void printList() {
        Node* temp = head;
        while (temp) {
            std::cout << temp->data;
            if (temp->next) std::cout << " -> ";
            temp = temp->next;
        }
        std::cout << std::endl;
    }

    ~LinkedList() {
        while (head) {
            Node* temp = head;
            head = head->next;
            delete temp;
        }
    };

int main() {
    LinkedList list;

    list.insertFront(10);
    list.insertBack(20);
    list.insertFront(5);

    std::cout << "Linked List: ";
    list.printList();

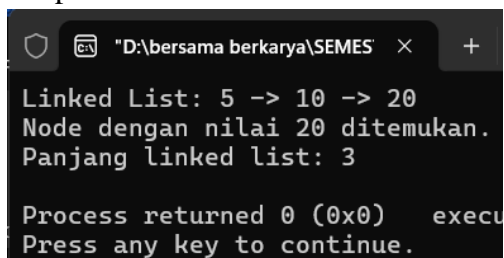
    int searchValue = 20;
    if (list.searchNode(searchValue)) {
        std::cout << "Node dengan nilai " << searchValue << " ditemukan." << std::endl;
    } else {
        std::cout << "Node dengan nilai " << searchValue << " tidak ditemukan." << std::endl;
    }
    int main::searchValue

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

    return 0;
}

```

Output :



```

D:\bersama berkarya\SEMES x +
Linked List: 5 -> 10 -> 20
Node dengan nilai 20 ditemukan.
Panjang linked list: 3

Process returned 0 (0x0) execu
Press any key to continue.

```

5. Kesimpulan

Single Linked List merupakan struktur data penting untuk mengelola data dinamis. Pemahaman konsep dan implementasinya, termasuk operasi dasar, sangat penting bagi pengembang perangkat lunak. Meskipun memiliki keterbatasan, Single Linked List tetap menjadi pilihan yang baik untuk aplikasi yang memerlukan struktur data yang dapat tumbuh dan menyusut secara dinamis sesuai kebutuhan.