

**LAPORAN PRAKTIKUM**  
**Modul V**  
**“SINGLE LINKED LIST BAGIAN 2”**



**Disusun Oleh:**  
**Dewi Atika Muthi -2211104042**  
**SE-07-02**

**Dosen:**  
**Wahyu Andi Saputra**

**PROGRAM STUDI S1 SOFTWARE ENGINEERING**  
**FAKULTAS INFORMATIKA**  
**TELKOM UNIVERSITY**  
**PURWOKERTO**  
**2024**

## 1. Tujuan

Tujuan praktikum materi ini adalah:

- Memahami penggunaan linked list dengan pointer dan operator-operator dalam program
- Memahami operasi-operasi dasar dalam linked list
- Membuat program dengan menggunakan linked list sesuai prototype yang ada

## 2. Landasan Teori

Single Linked List merupakan struktur data yang terdiri dari **sekumpulan node yang saling terhubung** secara **linear** melalui pointer. Setiap node memiliki dua komponen utama:

- **Data/Info:** Menyimpan nilai atau informasi
- **Next:** Pointer yang menunjuk ke node berikutnya

Operasi-operasi dasar dalam Single Linked List meliputi:

- **Searching:** Mencari node tertentu dalam list
- **Insertion:** Menambah node baru (di awal, di akhir, atau setelah node tertentu)
- **Deletion:** Menghapus node (di awal, di akhir, atau setelah node tertentu)
- **Traversal:** Mengakses seluruh node secara berurutan

## 3. Guided

### 1) Implementation of Single Linked List Basic Operations

**Source code:**

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

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

// Fungsi untuk menambahkan elemen baru ke awal linked list
void insertFirst(Node*& head, Node*& tail, int new_data) {
    Node* new_node = new Node();
    new_node->data = new_data;
    new_node->next = head;
    head = new_node;

    if(tail == nullptr){
        tail = new_node;
    }
}

// Fungsi untuk menambahkan elemen baru ke akhir linked list
void insertLast(Node*& head, Node*& tail, int new_data) {
    Node* new_node = new Node();
    new_node->data = new_data;
    new_node->next = head;
```

```

        head = new_node;

        if(tail == nullptr){
            tail = new_node;
        }
    }

    // Fungsi untuk mencari elemen dalam linked list
    int findElement(Node* head, int x) {
        Node* current = head;
        int index = 0;

        while (current != nullptr) {
            if (current->data == x) {
                return index;
            }
            current = current->next;
            index++;
        }
        return -1;
    }

    //Fungsi untuk menampilkan elemen dalam linked list
    void display(Node* node) {
        while (node != nullptr){
            cout << node->data << " ";
            node = node->next;
        }
        cout << endl;
    }

    // Fungsi untuk menghapus elemen dari linked list
    void deleteElement(Node*& head, int x) {
        if (head == nullptr){
            cout << "Linked list kosong" << endl;
            return;
        }

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

        Node* current = head;
        while (current->next != nullptr){
            if(current->next->data == x){
                Node* temp = current->next;
                current->next = current->next->next;
                delete temp;
                return;
            }
            current = current->next;
        }
    }

    int main() {
        Node* head = nullptr;
        Node* tail = nullptr;
    }

```

```

insertFirst(head, tail, 3);
insertFirst(head, tail, 5);
insertFirst(head, tail, 7);

insertLast(head, tail, 11);
insertLast(head, tail, 14);
insertLast(head, tail, 15);

cout << "Elemen dalam linked list: ";
display(head);

int x;
cout << endl << "Masukkan elemen yang ingin dicari: ";
cin >> x;

int result = findElement(head, x);

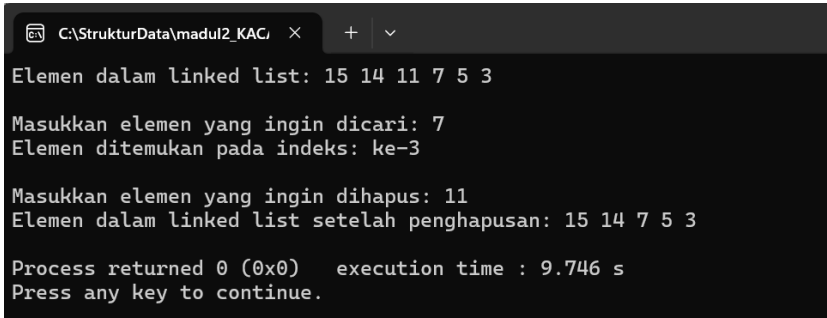
if(result == -1)
cout << "Elemen tidak ditemukan dalam linked list!" << endl;
else
cout << "Elemen ditemukan pada indeks: ke-" << result << endl;

cout << endl << "Masukkan elemen yang ingin dihapus: ";
cin >> x;
deleteElement(head, x);

cout << "Elemen dalam linked list setelah penghapusan: ";
display(head);
return 0;
}

```

### Output:



```

C:\StrukturData\madul2_KAC/ x + v
Elemen dalam linked list: 15 14 11 7 5 3

Masukkan elemen yang ingin dicari: 7
Elemen ditemukan pada indeks: ke-3

Masukkan elemen yang ingin dihapus: 11
Elemen dalam linked list setelah penghapusan: 15 14 7 5 3

Process returned 0 (0x0)   execution time : 9.746 s
Press any key to continue.

```

### Deskripsi program:

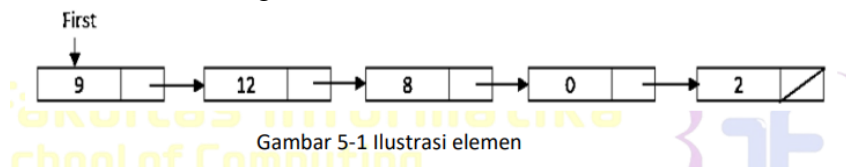
Program ini mengimplementasikan operasi dasar Single Linked List dengan fungsi-fungsi:

1. insertFirst(): Menambah node baru di awal list
  - Membuat node baru
  - Mengatur pointer next ke head lama
  - Mengupdate head ke node baru
2. insertLast(): Menambah node baru di akhir list
  - Membuat node baru
  - Mengatur pointer tail ke node baru

- Mengupdate tail
- 3. findElement(): Mencari elemen dalam list
  - Mengiterasi list dari head
  - Mengembalikan indeks jika elemen ditemukan
  - Mengembalikan -1 jika tidak ditemukan
- 4. deleteElement(): Menghapus node dengan nilai tertentu
  - Mencari node yang akan dihapus
  - Mengupdate pointer next
  - Menghapus node yang dituju
- 5. display(): Menampilkan seluruh elemen list
  - Mengiterasi dan mencetak nilai setiap node

#### 4. Unguided

##### 1) Membuat ADT Single Linked List



#### Source Code:

##### Singlelist.h:

```
#ifndef SINGLELIST_H
#define SINGLELIST_H
#include <iostream>

typedef int infotype;
typedef struct ElmList *address;

struct ElmList {
    infotype info;
    address next;
};

struct List {
    address First;
};

// Operasi Basic
void CreateList(List &L);
address alokasi(infotype x);
void dealokasi(address &P);
void printInfo(List L);
void insertFirst(List &L, address P);

#endif
```

##### singlelist.list:

```
#include "singlelist.h"
#include <iostream>
using namespace std;
```

```

void CreateList(List &L) {
    L.First = NULL;
}

address alokasi(infotype x) {
    address P = new ElmList;
    P->info = x;
    P->next = NULL;
    return P;
}

void dealokasi(address &P) {
    delete P;
    P = NULL;
}

void printInfo(List L) {
    address P = L.First;
    cout << "Isi List: ";
    while (P != NULL) {
        cout << P->info << " ";
        P = P->next;
    }
    cout << endl;
}

void insertFirst(List &L, address P) {
    P->next = L.First;
    L.First = P;
}

```

### main.cpp:

```

#include "singlelist.h"
#include <iostream>
using namespace std;

int main() {
    List L;
    address P;

    // Membuat list dan memasukkan elemen
    CreateList(L);

    // Memasukkan elements: 9, 12, 8, 0, 2
    P = alokasi(2);
    insertFirst(L, P);
    P = alokasi(0);
    insertFirst(L, P);
    P = alokasi(8);
    insertFirst(L, P);
    P = alokasi(12);
}

```

```

insertFirst(L, P);
P = alokasi(9);
insertFirst(L, P);

// Exercise 2: Display all elements
cout << "\nSoal no.1: \n";
printInfo(L);

return 0;
}

```

### Output:

```

C:\StrukturData\prak3\bin\De
Soal no.1:
Isi List: 9 12 8 0 2

Process returned 0 (0x0)   execution time : 0.079 s
Press any key to continue.

```

### Penjelasan Program:

Program ini mengimplementasikan Abstract Data Type (ADT) Single Linked List dengan komponen-komponen sebagai berikut:

#### Struktur Data:

- **ElmList**: Struktur untuk node yang berisi data (info) dan pointer ke node berikutnya (next)
- **List**: Struktur untuk list yang memiliki pointer ke elemen pertama (First)

#### Operasi-operasi dasar:

- **CreateList**: Inisialisasi list kosong dengan mengatur `First = NULL`
- **alokasi**: Membuat node baru dan mengalokasikan memori dengan nilai yang diberikan
- **dealokasi**: Menghapus node dan membebaskan memori
- **printInfo**: Menampilkan seluruh elemen list secara berurutan
- **insertFirst**: Menambahkan node baru di awal list

Program mendemonstrasikan penggunaan ADT dengan memasukkan lima nilai (9, 12, 8, 0, 2) ke dalam list dan menampilkannya.

## 2) Membuat fungsi mencari elemen (masukkan dari program)

### Source Code:

#### singlelist.h:

```

// Operasi tambahan untuk fungsi pencarian
address findElm(List L, infotype x);

```

### singlelist.cpp:

```
// Soal 2: mencari elemen
address findElm(List L, infotype x) {
    address P = L.First;
    while (P != NULL) {
        if (P->info == x) {
            return P;
        }
        P = P->next;
    }
    return NULL;
}
```

### main.cpp:

```
// Soal 2: Mencari Elemen 8
cout << "\nSoal 2 - Mencari Elemen engan info 8" << endl;
address found = findElm(L, 8);
if (found != NULL) {
    cout << "Elemen 8 Ditemukan dalam lit" << endl;
} else {
    cout << "Elemen 8 tidak ditemukan dalam list" << endl;
}
```

### Output:

```
Elemen 8 Ditemukan dalam lit
```

```
Process returned 0 (0x0)   execution time : 0.095 s
Press any key to continue.
```

### Penjelasan Program:

Program ini mengimplementasikan fungsi pencarian (`findElm`) dalam Single Linked List dengan karakteristik:

#### 1. Algoritma Pencarian:

- Menggunakan pointer `P` untuk traversal list
- Membandingkan nilai `info` setiap node dengan nilai yang dicari
- Mengembalikan address node jika ditemukan, `NULL` jika tidak

#### 2. Implementasi:

- Parameter input: List `L` dan nilai yang dicari (`x`)
- Menggunakan loop `while` untuk mengakses setiap node
- Pengecekan kondisi dengan `if` untuk membandingkan nilai

#### 3. Kompleksitas:

- Time Complexity:  $O(n)$ , dimana  $n$  adalah jumlah node dalam list
- Space Complexity:  $O(1)$



### 3) Membuat fungsi pentotalan seluruh elemen

#### Source Code:

##### Singlelist.h:

```
// Operasi tambahan untuk fungsi pentotalan
int sumInfo(List L);
```

##### Singlelist.cpp:

```
// Soal 3: Menghitung semua jumlah elemen
int sumInfo(List L) {
    address P = L.First;
    int sum = 0;
    while (P != NULL) {
        sum += P->info;
        P = P->next;
    }
    return sum;
}
```

##### Main.cpp:

```
//Soal 3 - Total dari semua elemen
cout << "\nSoal 3 - Total dari semua elemen" << endl;
int total = sumInfo(L);
cout << "Total jumlah dari semua elemen: " << total <<
endl;
```

#### Output:

```
Soal 3 - Total dari semua elemen
Total jumlah dari semua elemen: 31

Process returned 0 (0x0)   execution time : 0.116 s
Press any key to continue.
|
```

#### Penjelasan Program:

Program ini mengimplementasikan fungsi penjumlahan (`sumInfo`) untuk menghitung total nilai seluruh elemen dalam list dengan karakteristik:

##### 1. Algoritma Penjumlahan:

- Inisialisasi variabel `sum = 0`
- Traversal seluruh list menggunakan pointer `P`
- Akumulasi nilai info setiap node ke dalam `sum`
- Mengembalikan total hasil penjumlahan

##### 2. Implementasi:

- Menggunakan variabel `sum` sebagai accumulator
- Loop `while` untuk mengakses setiap node
- Pertambahan nilai menggunakan operator `+=`

### 3. Hasil:

- Untuk list dengan elemen [9, 12, 8, 0, 2]
- Total yang dihasilkan adalah 31 ( $9+12+8+0+2$ )

## 5. Kesimpulan

Single Linked List adalah struktur data yang efektif untuk menyimpan data secara dinamis dengan penggunaan memori yang efisien. Operasi dasar seperti insertion, deletion, dan searching dapat diimplementasikan dengan mengatur pointer antar node.

Implementasi ADT Single Linked List memungkinkan pengorganisasian kode yang lebih terstruktur dan modular. Pemahaman tentang pointer dan manajemen memori sangat penting dalam implementasi Single Linked List. Single Linked List cocok digunakan ketika operasi insertion dan deletion di awal list sering dilakukan, namun kurang efisien untuk operasi random access.