

**LAPORAN PRAKTIKUM**  
**Modul 05**  
**“SINGLE LINKED LIST (BAGIAN KEDUA)”**



**Disusun Oleh:**  
**Aji Prasetyo Nugroho - 2211104049**  
**S1SE-07-2**

**Dosen :**  
**Wahyu Andi Saputra, S.Pd., M.Eng**

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

## **A. Tujuan**

1. Memahami penggunaan linked list dengan pointer operator- operator dalam program.
2. Memahami operasi-operasi dasar dalam linked list.
3. Membuat program dengan menggunakan linked list dengan prototype yang ada

## **B. Landasan Teori**

### **a. Searching**

Searching merupakan salah satu operasi dasar dalam struktur data berbasis list, di mana proses pencarian dilakukan untuk menemukan node atau elemen tertentu dalam list. Operasi ini penting karena menjadi dasar bagi berbagai manipulasi data lainnya, seperti insert after, delete after, dan update. Dalam konteks list, baik list berurut (ordered list) maupun tidak berurut (unordered list), searching dilakukan dengan mengunjungi setiap node satu per satu sampai elemen yang dicari ditemukan atau akhir list tercapai.

#### **1. Pengertian Searching dalam List**

Searching adalah proses pencarian data tertentu di dalam list. Pada implementasi list berbasis node, seperti dalam list terhubung (linked list), proses pencarian ini berjalan dengan mengunjungi setiap node mulai dari awal hingga node yang dicari ditemukan atau list berakhir. Proses ini bersifat iteratif dan linear, sehingga membutuhkan waktu  $O(n)$  dalam kompleksitas waktu untuk menemukan node di posisi tertentu. Pada list berukuran besar, kompleksitas ini menjadi perhatian khusus karena membutuhkan waktu yang signifikan seiring bertambahnya jumlah elemen.

#### **2. Metode Searching pada List**

Ada dua metode utama dalam melakukan searching pada list, yaitu:

- **Sequential Search (Pencarian Berurutan):** Dilakukan dengan mengunjungi setiap node satu per satu hingga elemen yang dicari ditemukan. Metode ini umum digunakan pada list tidak berurut.
- **Binary Search (Pencarian Biner):** Digunakan pada list yang sudah terurut. Metode ini memotong jumlah elemen yang dicari menjadi setengah setiap iterasi, sehingga lebih efisien dibandingkan sequential search, dengan kompleksitas waktu  $O(\log n)$ . Namun, binary search tidak cocok untuk struktur data linked list karena tidak memiliki akses acak (random access) ke elemen tengah.

### 3. Pentingnya Searching dalam Operasi Lain

Operasi searching sangat mendasar karena memungkinkan manipulasi data lain seperti:

- Insert After: Setelah node yang dicari ditemukan, node baru dapat disisipkan setelah node tersebut.
- Delete After: Setelah menemukan node target, node setelahnya dapat dihapus.
- Update: Setelah node yang diinginkan ditemukan, nilai data dalam node tersebut dapat diubah.

Dengan demikian, searching menjadi operasi yang esensial untuk mendukung manipulasi data yang lebih kompleks dalam list, sehingga membuat struktur data lebih fleksibel dan dinamis dalam pengelolaan informasi.

## C. Guided

### a. Searching

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 = nullptr;

    if (head == nullptr) {
        head = new_node;
        tail = new_node;
    } else {
        tail->next = new_node;
        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, 18);

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

    int x;
    cout << "Masukan 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 dalam index " << result << endl;

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

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

    return 0;
}

```

Output :

```

Elemen dalam linked list: 7 5 3 11 14 18
Masukan elemen yang ingin dicari: 7
Elemen ditemukan dalam index 0
Masukan elemen yang ingin dihapus: 11
Elemen dalam linked list setelah penghapusan: 7 5 3 14 18
PS D:\Praktikum STD_2211104049>

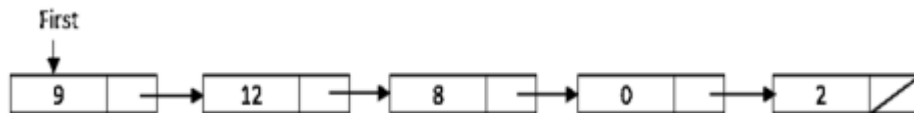
```

## D. Unguided

1. Buatlah ADT Single Linked list sebagai berikut di dalam file “singlelist.h”:

```
Type infotype : int
Type address : pointer to ElmList
Type ElmList <
    info : infotype
    next : address
>
Type List : < First : address >
prosedur CreateList( in/out L : List )
fungsi alokasi( x : infotype ) : address
prosedur dealokasi( in/out P : address )
prosedur printInfo( in L : List )
prosedur insertFirst( in/out L : List, in P : address )
```

Kemudian buat implementasi ADT Single Linked list pada file “singlelist.cpp”. Adapun isi data



Cobalah hasil implementasi ADT pada *file* “main.cpp”

```
int main()
{
    List L;
    address P1, P2, P3, P4, P5 = NULL;
    createList(L);

    P1 = alokasi(2);
    insertFirst(L,P1);

    P2 = alokasi(0);
    insertFirst(L,P2);

    P3 = alokasi(8);
    insertFirst(L,P3);

    P4 = alokasi(12);
    insertFirst(L,P4);

    P5 = alokasi(9);
    insertFirst(L,P5);


    printInfo(L)
    return 0;
}
```

Contoh output :

```
9 12 8 0 2
Process returned 0 (0x0)   execution time : 0.019 s
Press any key to continue.
```

Source Code :

a. Main.cpp



```
#include "singlelist.h"
#include <iostream>

using namespace std;

int main() {
    List L;
    address P1, P2, P3, P4, P5 = nullptr;

    createList(L);

    P1 = alokasi(9);
    insertFirst(L, P1);

    P2 = alokasi(12);
    insertFirst(L, P2);

    P3 = alokasi(8);
    insertFirst(L, P3);


    P4 = alokasi(10);
    insertFirst(L, P4);

    P5 = alokasi(2);
    insertFirst(L, P5);

    printInfo(L);

    return 0;
}
```

b. Singlelist.cpp



```
#include "singlelist.h"
#include <iostream>

using namespace std;

void createList(List &L) {
    L.first = nullptr;
}

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

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

void printInfo(List L) {
    address P = L.first;
    while (P != nullptr) {
        cout << P->info << " ";
        P = P->next;
    }
    cout << endl;
}
```



c. Singlelist.h

```
#ifndef SINGLELIST_H
#define SINGLELIST_H

typedef int infotype;
typedef struct ElmtList *address;

struct ElmtList {
    infotype info;
    address next;
};

struct List {
    address first;
};

void createList(List &L);
address alokasi(infotype x);
void insertFirst(List &L, address P);
void printInfo(List L);

#endif
```

Output :

```
PS D:\Praktikum STD_2211104049\2211104049_Aji Prasetyo Nugroho\Unguided> g++ main.cpp singlelist.cpp -o main
PS D:\Praktikum STD_2211104049\2211104049_Aji Prasetyo Nugroho\Unguided> ./main
2 10 8 12 9
PS D:\Praktikum STD_2211104049\2211104049_Aji Prasetyo Nugroho\Unguided> █
```

2. Carilah elemen dengan info 8 dengan membuat fungsi baru.  
fungsi findElm( L : List, x : infotype ) : address

```
8 ditemukan dalam list
Process returned 0 (0x0)   execution time : 0.020 s
Press any key to continue.
```

Source Code :

- a. main.cpp

```
#include "singlelist.h"
#include <iostream>

using namespace std;

int main() {
    List L;
    address P1, P2, P3, P4, P5 = nullptr;

    createList(L);

    P1 = alokasi(9);
    insertFirst(L, P1);

    P2 = alokasi(12);
    insertFirst(L, P2);

    P3 = alokasi(8);
    insertFirst(L, P3);

    P4 = alokasi(10);
    insertFirst(L, P4);

    P5 = alokasi(2);
    insertFirst(L, P5);

    printInfo(L);

    address found = findElm(L, 8);
    if (found != nullptr) {
        cout << found->info << " ditemukan dalam list" << endl;
    } else {
        cout << "Elemen tidak ditemukan dalam list" << endl;
    }

    return 0;
}
```

b. singlelist.cpp

```
#include "singlelist.h"
#include <iostream>

using namespace std;

void createList(List &L) {
    L.first = nullptr;
}

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

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

void printInfo(List L) {
    address P = L.first;
    while (P != nullptr) {
        cout << P->info << " ";
        P = P->next;
    }
    cout << endl;
}

address findElm(List L, infotype x) {
    address P = L.first;
    while (P != nullptr) {
        if (P->info == x) {
            return P;
        }
        P = P->next;
    }
    return nullptr;
}
```

c. singlelist.h

```
#ifndef SINGLELIST_H
#define SINGLELIST_H

typedef int infotype;
typedef struct ElmtList *address;

struct ElmtList {
    infotype info;
    address next;
};

struct List {
    address first;
};

void createList(List &L);
address alokasi(infotype x);
void insertFirst(List &L, address P);
void printInfo(List L);
address findElm(List L, infotype x); // Add this line

#endif
```

Output :

```
PS D:\Praktikum STD_2211104049\2211104049_Aji Prasetyo Nugroho\Unguided\soal 2> g++ main.cpp singlelist.cpp -o main
PS D:\Praktikum STD_2211104049\2211104049_Aji Prasetyo Nugroho\Unguided\soal 2> ./main
2 10 8 12 9
8 ditemukan dalam list
PS D:\Praktikum STD_2211104049\2211104049_Aji Prasetyo Nugroho\Unguided\soal 2> |
```

3. Hitunglah jumlah total info seluruh elemen ( $9+12+8+0+2=31$ ).

```
Total info dari kelima elemen adalah 31
Process returned 0 (0x0)   execution time : 0.019 s
Press any key to continue.
```

Source Code :

a. main.cpp

```
#include "singlelist.h"
#include <iostream>

using namespace std;

int main() {
    List L;
    address P1, P2, P3, P4, P5 = nullptr;

    createList(L);

    P1 = alokasi(9);
    insertFirst(L, P1);

    P2 = alokasi(12);
    insertFirst(L, P2);

    P3 = alokasi(8);
    insertFirst(L, P3);

    P4 = alokasi(10);
    insertFirst(L, P4);

    P5 = alokasi(2);
    insertFirst(L, P5);

    printInfo(L);

    int total = sumInfo(L);
    cout << "Total info dari kelima elemen adalah " << total << endl;

    return 0;
}
```

b. singlelist.cpp

```
#include "singlelist.h"
#include <iostream>

using namespace std;

void createList(List &L) {
    L.first = nullptr;
}

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


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

void printInfo(List L) {
    address P = L.first;
    while (P != nullptr) {
        cout << P->info << " ";
        P = P->next;
    }
    cout << endl;
}

address findElm(List L, infotype x) {
    address P = L.first;
    while (P != nullptr) {
        if (P->info == x) {
            return P;
        }
        P = P->next;
    }
    return nullptr;
}

int sumInfo(List L) {
    int sum = 0;
    address P = L.first;
    while (P != nullptr) {
        sum += P->info;
        P = P->next;
    }
    return sum;
}
```

c. singlelist.h



```
#ifndef SINGLELIST_H
#define SINGLELIST_H

typedef int infotype;
typedef struct ElmtList *address;

struct ElmtList {
    infotype info;
    address next;
};

struct List {
    address first;
};

void createList(List &L);
address alokasi(infotype x);
void insertFirst(List &L, address P);
void printInfo(List L);
address findElm(List L, infotype x);
int sumInfo(List L); // Add this line

#endif
```

Output :

```
PS D:\Praktikum STD_2211104049\2211104049_Aji Prasetyo Nugroho\Unguided\soal 3> g++ main.cpp singlelist.cpp -o main
PS D:\Praktikum STD_2211104049\2211104049_Aji Prasetyo Nugroho\Unguided\soal 3> ./main
2 10 8 12 9
Total info dari kelima elemen adalah 41
PS D:\Praktikum STD_2211104049\2211104049_Aji Prasetyo Nugroho\Unguided\soal 3> |
```