

**LAPORAN PRAKTIKUM STRUKTUR  
DATA**

**MODUL V  
SINGLY LINKED LIST II**



**Disusun Oleh :**

NAMA : Muhamad Naufal Ammar

NIM : 103112430036

**Dosen**

WAHYU ANDI SAPUTRA

**PROGRAM STUDI STRUKTUR DATA  
FAKULTAS INFORMATIKA  
TELKOM UNIVERSITY PURWOKERTO  
2025**

## A. Dasar Teori

Pada dasarnya, SLL merupakan struktur data dinamis yang terdiri dari simpul-simpul (nodes) di mana setiap simpul menyimpan dua bagian: satu bagian untuk data (misalnya info) dan satu bagian untuk penunjuk ke simpul berikutnya (next). Karena memori dialokasikan secara dinamis, SLL dapat ‘tumbuh’ atau ‘menyusut’ selama waktu eksekusi tanpa harus menetapkan ukuran di awal—ini berbeda dengan array yang ukuran dan alokasi memori harus diketahui sebelum dipakai. Operasi dasar pada SLL meliputi pembuatan list kosong (inisialisasi pointer head atau First menjadi NULL), penambahan simpul di awal (insertFirst), pencarian simpul (findElm), penelusuran semua simpul (traversal atau printInfo), dan penghitungan informasi yang mungkin terkandung di semua simpul (seperti sumInfo). Struktur ini memungkinkan kita melakukan penyisipan atau penghapusan simpul tanpa harus menggeser elemen-elemen lain seperti pada array, sehingga operasi tersebut dapat lebih efisien dalam kondisi tertentu. Namun SLL juga memiliki keterbatasan, misalnya hanya dapat ditelusuri satu arah (dari head ke tail), dan pencarian elemen tertentu secara langsung (random access) tidak secepat array karena harus melakukan traversal dari simpul pertama. Dalam program yang kamu buat, fungsi findElm dan sumInfo merupakan aplikasi langsung dari teori ini: findElm melakukan penelusuran hingga menemukan simpul dengan info = x, sedangkan sumInfo melakukan akumulasi seluruh nilai data dalam list melalui traversal. Dengan memahami teori tersebut, kita sebagai mahasiswa bisa lebih jelas bagaimana alokasi memori, pointer next, dan operasi traversal bekerja pada level struktur data.

B. Guided (berisi screenshot source code & output program disertai penjelasannya)

Guided 1

```
list.h
/* file : list.h */
/* Contoh ADT list berkait dengan representasi fisik pointer */
/* Representasi address dengan pointer */
/* Info tipe adalah integer */

#ifndef list_H
#define list_H

#include <stdio.h>
#include <stdlib.h>

/* Definisi boolean */
#define boolean int
#define true 1
#define false 0

#define Nil NULL

/* Deklarasi record dan struktur data list */
typedef int infotype;
typedef struct elmlist *address;

struct elmlist {
    infotype info;        // Menyimpan data/informasi elemen
    address next;        // Pointer ke elemen berikutnya
};

/* Definisi list */
/* List kosong jika L.first = Nil */
/* Setiap elemen address P dapat diacu P->info atau P->next */
struct list {
    address first;        // Pointer ke elemen pertama list
};

/***** PENGECEKAN APAKAH LIST KOSONG *****/
boolean ListEmpty(list L);
/* Mengembalikan nilai true jika list kosong */

/***** PEMBUATAN LIST KOSONG *****/
void CreateList(list *L);
/* I.S. sembarang */
/* F.S. terbentuk list kosong */

/***** MANAJEMEN MEMORI *****/
void dealokasi(address P);
/* I.S. P terdefinisi */
/* F.S. memori yang digunakan P dikembalikan ke sistem */
```

```

/***** MANAJEMEN MEMORI *****/
void dealokasi(address P);
/* I.S. P terdefinisi */
/* F.S. memori yang digunakan P dikembalikan ke sistem */

address alokasi(inftype X);
/* Mengalokasikan memori untuk elemen baru dengan nilai X */
/* Mengembalikan address elemen yang dialokasi */

/***** PENCAIRAN SEBUAH ELEMEN LIST *****/
address findElm(list L, infotype X);
/* Mencari apakah ada elemen list dengan P+info = X */
/* Jika ada, mengembalikan address elemen tsb, dan Nil jika sebaliknya */

boolean fFindElm(list L, address P);
/* Mencari apakah ada elemen list dengan alamat P */
/* Mengembalikan true jika ada dan false jika tidak ada */

address findBefore(list L, address P);
/* Mengembalikan address elemen sebelum P */
/* Jika prec berada pada awal list, maka mengembalikan nilai Nil */

/***** PENAMBAHAN ELEMEN *****/
void insertFirst(list *L, address P);
/* I.S. sembarang, P sudah dialokasi */
/* F.S. menempatkan elemen beralamat P pada awal list */

void insertAfter(list *L, address P, address Prec);
/* I.S. sembarang, P dan Prec alamat salah satu elemen list */
/* F.S. menempatkan elemen beralamat P sesudah elemen beralamat Prec */

void insertLast(list *L, address P);
/* I.S. sembarang, P sudah dialokasi */
/* F.S. menempatkan elemen beralamat P pada akhir list */

/***** PENGHAPUSAN SEBUAH ELEMEN *****/
void delFirst(list *L, address *P);
/* I.S. list tidak kosong */
/* F.S. P adalah alamat dari elemen pertama list sebelum dihapus */
/* Elemen pertama list hilang dan list mungkin menjadi kosong */

void dellast(list *L, address *P);
/* I.S. list tidak kosong */
/* F.S. P adalah alamat dari elemen terakhir list sebelum dihapus */
/* Elemen terakhir list hilang dan list mungkin menjadi kosong */

```

***list.cpp***

```

/* file : list.c */
/* Implementasi ADT Single Linked List */

#include "list.h"
#include <stdio.h>
#include <stdlib.h>

/***** PENGECEKAN APAKAH LIST KOSONG *****/
boolean ListEmpty(list L) {
    /* Keterangan: Mengembalikan true jika list kosong (first = NULL) */
    return (L.first == Nil);
}

/***** PEMBUATAN LIST KOSONG *****/
void CreateList(list *L) {
    /* Keterangan: Inisialisasi list menjadi kosong */
    (*L).first = Nil;
}

/***** MANAJEMEN MEMORI *****/
void dealokasi(address P) {
    /* Keterangan: Membebaskan memori yang digunakan oleh elemen P */
    free(P);
}

address alokasi(inftype X) {
    /* Keterangan: Mengalokasikan memori untuk elemen baru dengan nilai X */
    address P = (address)malloc(sizeof(struct elmList));
    if (P != Nil) {
        P->info = X;
        P->next = Nil;
    }
    return P;
}

/***** PENCARIAN SEBUAH ELEMEN LIST *****/
address findElm(list L, infotype X) {
    /* Keterangan: Mencari elemen dengan nilai X dalam list */
    /* Proses searching: mengunjungi setiap node sampai ditemukan atau sampai akhir list */
    address P = L.first;

    while (P != Nil) {
        if (P->info == X) {
            return P; // Elemen ditemukan
        }
        P = P->next;
    }
    return Nil; // Elemen tidak ditemukan
}

```

430099\_Monkainribuwana\_Modul5 > Guided > list.cpp > fFindElm(list, address)

```
boolean fFindElm(list L, address P) {  
    address current = L.first;  
  
    while (current != Nil) {  
        if (current == P) {  
            return true;  
        }  
        current = current->next;  
    }  
    return false;  
}  
  
address findBefore(list L, address P) {  
    /* Keterangan: Mencari elemen sebelum P dalam list */  
    if (L.first == P) {  
        return Nil; // P adalah elemen pertama  
    }  
  
    address current = L.first;  
    while (current != Nil && current->next != P) {  
        current = current->next;  
    }  
    return current;  
}  
  
/***** PENAMBAHAN ELEMEN *****/  
void insertFirst(list *L, address P) {  
    /* Keterangan: Menyisipkan elemen P di awal list */  
    P->next = (*L).first;  
    (*L).first = P;  
}  
  
void insertAfter(list *L, address P, address Prec) {  
    /* Keterangan: Menyisipkan elemen P setelah elemen Prec */  
    P->next = Prec->next;  
    Prec->next = P;  
}
```



```

void insertLast(list *L, address P) {
    /* Keterangan: Menyisipkan elemen P di akhir list */
    if (ListEmpty(*L)) {
        (*L).first = P;
    } else {
        address last = (*L).first;
        while (last->next != Nil) {
            last = last->next;
        }
        last->next = P;
    }
    P->next = Nil;
}

/***** PENGHAPUSAN SEBUAH ELEMEN *****/
void delFirst(list *L, address *P) {
    /* Keterangan: Menghapus elemen pertama dari list */
    *P = (*L).first;
    (*L).first = (*L).first->next;
    (*P)->next = Nil;
}

void dellast(list *L, address *P) {
    /* Keterangan: Menghapus elemen terakhir dari list */
    if ((*L).first->next == Nil) {
        // Hanya ada satu elemen
        *P = (*L).first;
        (*L).first = Nil;
    } else {
        address last = (*L).first;
        while (last->next->next != Nil) {
            last = last->next;
        }
        *P = last->next;
        last->next = Nil;
    }
}

void delAfter(list *L, address *P, address Prec) {
    /* Keterangan: Menghapus elemen setelah Prec */
    *P = Prec->next;
    Prec->next = (*P)->next;
    (*P)->next = Nil;
}

```

```
void delP(list *L, infotype X) {
    if (P != Nil) {
        if (P == (*L).first) {
            delFirst(L, &P);
        } else {
            address Prec = findBefore(*L, P);
            delAfter(L, &P, Prec);
        }
        dealokasi(P);
    }
}

/***** PROSES SEMUA ELEMEN LIST *****/
void printInfo(list L) {
    /* Keterangan: Menampilkan semua elemen dalam list */
    address P = L.first;

    if (ListEmpty(L)) {
        printf("List kosong\n");
    } else {
        printf("Isi list: ");
        while (P != Nil) {
            printf("%d", P->info);
            P = P->next;
            if (P != Nil) {
                printf(" -> ");
            }
        }
        printf("\n");
    }
}

int nblist(list L) {
    /* Keterangan: Menghitung jumlah elemen dalam list */
    int count = 0;
    address P = L.first;

    while (P != Nil) {
        count++;
        P = P->next;
    }
    return count;
}
```



```
0033_monhammadibrahima_nodulis - Salda - list.cpp - n index(list, address)
/***** PROSES TERHADAP LIST *****/
```

```
void delAll(list *L) {
    /* Keterangan: Menghapus semua elemen dalam list */
    address P;
    while (!ListEmpty(*L)) {
        delFirst(L, &P);
        dealokasi(P);
    }
}
```

```
void invertList(list *L) {
    /* Keterangan: Membalik urutan elemen dalam list */
    address prev = Nil;
    address current = (*L).first;
    address next = Nil;

    while (current != Nil) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    (*L).first = prev;
}
```

```
void copyList(list l1, list *L2) {
    /* Keterangan: Membuat salinan list l1 ke L2 */
    Createlist(L2);

    if (!ListEmpty(l1)) {
        address P1 = l1.first;
        address last = Nil;

        while (P1 != Nil) {
            address P2 = alokasi(P1->info);
            if (ListEmpty(*L2)) {
                (*L2).first = P2;
                last = P2;
            } else {
                last->next = P2;
                last = P2;
            }
            P1 = P1->next;
        }
    }
}
```

```
list fCopyList(list L) {
    /* Keterangan: Mengembalikan salinan list L */
    list L2;
    copyList(L, &L2);
    return L2;
}
```

```

void delAfter(list *L, address *P, address Prec);
/* I.S. list tidak kosong, Prec alamat salah satu elemen list */
/* F.S. P adalah alamat dari Prec→next, menghapus Prec→next dari list */

void delP(list *L, infotype X);
/* I.S. sembarang */
/* F.S. jika ada elemen list dengan P→info = X, maka P dihapus dan di-dealokasi */
/* Jika tidak ada maka list tetap, list mungkin akan menjadi kosong karena penghapusan */

/***** PROSES SEMUA ELEMEN LIST *****/
void printInfo(list L);
/* I.S. list mungkin kosong */
/* F.S. jika list tidak kosong menampilkan semua info yang ada pada list */

int nbList(list L);
/* Mengembalikan jumlah elemen pada list */

/***** PROSES TERHADAP LIST *****/
void delAll(list *L);
/* Menghapus semua elemen list dan semua elemen di-dealokasi */

void invertList(list *L);
/* I.S. sembarang */
/* F.S. elemen-elemen list dibalik */

void copyList(list l1, list *l2);
/* I.S. l1 sembarang */
/* F.S. l2 = l1, l2 merupakan salinan dari l1 */

list fCopyList(list L);
/* Mengembalikan list yang merupakan salinan dari L */

#endif

```

**main.cpp**

```

/* file : main.c */
/* Program Single Linked List Lengkap dalam Satu File */

#include <stdio.h>
#include <stdlib.h>

/* Definisi boolean */
#define boolean int
#define true 1
#define false 0

#define Nil NULL

/* Deklarasi record dan struktur data list */
typedef int infotype;
typedef struct elmlist *address;

struct elmlist {
    infotype info;        // Menyimpan data/informasi elemen
    address next;         // Pointer ke elemen berikutnya
};

/* Definisi list */
struct list {
    address first;        // Pointer ke elemen pertama list
};

/***** PENGECEKAN APAKAH LIST KOSONG *****/
boolean ListEmpty(list L) {
    /* Keterangan: Mengembalikan true jika list kosong (first = NULL) */
    return (L.first == Nil);
}

/***** PEMBUATAN LIST KOSONG *****/
void CreateList(list *L) {
    /* Keterangan: Inisialisasi list menjadi kosong */
    (*L).first = Nil;
}

/***** MANAJEMEN MEMORI *****/
void dealokasi(address P) {
    /* Keterangan: Membebaskan memori yang digunakan oleh elemen P */
    free(P);
}

address alokasi(infotype X) {
    /* Keterangan: Mengalokasikan memori untuk elemen baru dengan nilai X */
    address P = (address)malloc(sizeof(struct elmlist));
    if (P != Nil) {
        P->info = X;
        P->next = Nil;
    }
    return P;
}

```

```

/***** PENCARIAN SEBUAH ELEMEN LIST *****/
address findElm(list L, infotype X) {
    /* Keterangan: Mencari elemen dengan nilai X dalam list */
    /* Proses searching: mengunjungi setiap node sampai ditemukan atau sampai akhir list */
    address P = L.first;

    while (P != Nil) {
        if (P->info == X) {
            return P; // Elemen ditemukan
        }
        P = P->next;
    }
    return Nil; // Elemen tidak ditemukan
}

boolean fFindElm(list L, address P) {
    /* Keterangan: Mengecek apakah alamat P ada dalam list */
    address current = L.first;

    while (current != Nil) {
        if (current == P) {
            return true;
        }
        current = current->next;
    }
    return false;
}

address findBefore(list L, address P) {
    /* Keterangan: Mencari elemen sebelum P dalam list */
    if (L.first == P) {
        return Nil; // P adalah elemen pertama
    }

    address current = L.first;
    while (current != Nil && current->next != P) {
        current = current->next;
    }
    return current;
}

/***** PENAMBAHAN ELEMEN *****/
void insertFirst(list *L, address P) {
    /* Keterangan: Menyisipkan elemen P di awal list */
    P->next = (*L).first;
    (*L).first = P;
}

void insertAfter(list *L, address P, address Prec) {
    /* Keterangan: Menyisipkan elemen P setelah elemen Prec */
    P->next = Prec->next;
    Prec->next = P;
}

```

```

void insertLast(list *L, address P) {
    /* Keterangan: Menyisipkan elemen P di akhir list */
    if (ListEmpty(*L)) {
        (*L).first = P;
    } else {
        address last = (*L).first;
        while (last->next != Nil) {
            last = last->next;
        }
        last->next = P;
    }
    P->next = Nil;
}

/***** PENGHAPUSAN SEBUAH ELEMEN *****/
void delFirst(list *L, address *P) {
    /* Keterangan: Menghapus elemen pertama dari list */
    *P = (*L).first;
    (*L).first = (*L).first->next;
    (*P)->next = Nil;
}

void dellast(list *L, address *P) {
    /* Keterangan: Menghapus elemen terakhir dari list */
    if ((*L).first->next == Nil) {
        // Hanya ada satu elemen
        *P = (*L).first;
        (*L).first = Nil;
    } else {
        address last = (*L).first;
        while (last->next->next != Nil) {
            last = last->next;
        }
        *P = last->next;
        last->next = Nil;
    }
}

void delAfter(list *L, address *P, address Prec) {
    /* Keterangan: Menghapus elemen setelah Prec */
    *P = Prec->next;
    Prec->next = (*P)->next;
    (*P)->next = Nil;
}

```

```

void delP(list *L, infotype X) {
    /* Keterangan: Menghapus elemen dengan nilai X dari list */
    address P = findElm(*L, X);
    if (P != Nil) {
        if (P == (*L).first) {
            delFirst(L, &P);
        } else {
            address Prec = findBefore(*L, P);
            delAfter(L, &P, Prec);
        }
        dealokasi(P);
    }
}

/***** PROSES SEMUA ELEMEN LIST *****/
void printInfo(list L) {
    /* Keterangan: Menampilkan semua elemen dalam list */
    address P = L.first;

    if (ListEmpty(L)) {
        printf("List kosong\n");
    } else {
        printf("Isi list: ");
        while (P != Nil) {
            printf("%d", P->info);
            P = P->next;
            if (P != Nil) {
                printf(" -> ");
            }
        }
        printf("\n");
    }
}

int nbList(list L) {
    /* Keterangan: Menghitung jumlah elemen dalam list */
    int count = 0;
    address P = L.first;

    while (P != Nil) {
        count++;
        P = P->next;
    }
    return count;
}

/***** PROSES TERHADAP LIST *****/
void delAll(list *L) {
    /* Keterangan: Menghapus semua elemen dalam list */
    address P;
    while (!ListEmpty(*L)) {
        delFirst(L, &P);
        dealokasi(P);
    }
}

```



```

void invertList(list *L) {
    /* Keterangan: Membalik urutan elemen dalam list */
    address prev = Nil;
    address current = (*L).first;
    address next = Nil;

    while (current != Nil) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    (*L).first = prev;
}

void copyList(list L1, list *L2) {
    /* Keterangan: Membuat salinan list L1 ke L2 */
    CreateList(L2);

    if (!ListEmpty(L1)) {
        address P1 = L1.first;
        address last = Nil;

        while (P1 != Nil) {
            address P2 = alokasi(P1->info);
            if (ListEmpty(*L2)) {
                (*L2).first = P2;
                last = P2;
            } else {
                last->next = P2;
                last = P2;
            }
            P1 = P1->next;
        }
    }
}

list fCopyList(list L) {
    /* Keterangan: Mengembalikan salinan list L */
    list L2;
    copyList(L, &L2);
    return L2;
}

```

```

/***** FUNGSI UNTUK DEMONSTRASI *****/
void demoSearching() {
    printf("\n=== DEMO OPERASI SEARCHING ===\n");

    list L;
    CreateList(&L);

    // Tambahkan beberapa elemen
    insertFirst(&L, alokasi(30));
    insertFirst(&L, alokasi(20));
    insertFirst(&L, alokasi(10));
    insertLast(&L, alokasi(40));
    insertLast(&L, alokasi(50));

    printf("List awal: ");
    printInfo(L);

    // Demo searching
    infotype nilaiCari;
    address hasil;

    nilaiCari = 20;
    hasil = findElm(L, nilaiCari);
    if (hasil != Nil) {
        printf("Nilai %d DITEMUKAN pada address: %p\n", nilaiCari, hasil);
    } else {
        printf("Nilai %d TIDAK DITEMUKAN\n", nilaiCari);
    }

    nilaiCari = 99;
    hasil = findElm(L, nilaiCari);
    if (hasil != Nil) {
        printf("Nilai %d DITEMUKAN pada address: %p\n", nilaiCari, hasil);
    } else {
        printf("Nilai %d TIDAK DITEMUKAN\n", nilaiCari);
    }

    // Demo findBefore
    address elemen20 = findElm(L, 20);
    if (elemen20 != Nil) {
        address sebelum20 = findBefore(L, elemen20);
        if (sebelum20 != Nil) {
            printf("Elemen sebelum 20 adalah: %d\n", sebelum20->info);
        } else {
            printf("20 adalah elemen pertama\n");
        }
    }

    delAll(&L);
}

```

```

void demoInsertDelete() {
    printf("\n=== DEMO OPERASI INSERT DAN DELETE ===\n");

    list L;
    Createlist(&L);

    printf("\n1. Insert First (10, 20, 30):\n");
    insertFirst(&L, alokasi(10));
    insertFirst(&L, alokasi(20));
    insertFirst(&L, alokasi(30));
    printInfo(L);

    printf("\n2. Insert Last (40, 50):\n");
    insertLast(&L, alokasi(40));
    insertLast(&L, alokasi(50));
    printInfo(L);

    printf("\n3. Insert After 20 (25):\n");
    address elemen20 = findElm(L, 20);
    if (elemen20 != Nil) {
        insertAfter(&L, alokasi(25), elemen20);
    }
    printInfo(L);

    printf("\n4. Delete First:\n");
    address deleted;
    delFirst(&L, &deleted);
    printf("Elemen yang dihapus: %d\n", deleted->info);
    dealokasi(deleted);
    printInfo(L);

    printf("\n5. Delete elemen 25:\n");
    delP(&L, 25);
    printInfo(L);

    printf("\n6. Delete Last:\n");
    dellast(&L, &deleted);
    printf("Elemen yang dihapus: %d\n", deleted->info);
    dealokasi(deleted);
    printInfo(L);

    delAll(&L);
}

```

```

void demoAdvancedOperations() {
    printf("\n=== DEMO OPERASI LANJUT ===\n");

    list L;
    CreateList(&L);

    // Buat list: 1 -> 2 -> 3 -> 4 -> 5
    for (int i = 1; i <= 5; i++) {
        insertLast(&L, alokasi(i));
    }

    printf("List awal: ");
    printInfo(L);
    printf("Jumlah elemen: %d\n", nbList(L));

    printf("\n1. Invert List:\n");
    invertList(&L);
    printf("Setelah diinvert: ");
    printInfo(L);

    printf("\n2. Copy List:\n");
    list L2;
    copyList(L, &L2);
    printf("List asli: ");
    printInfo(L);
    printf("List salinan: ");
    printInfo(L2);

    printf("\n3. Modifikasi list salinan:\n");
    insertFirst(&L2, alokasi(99));
    printf("List asli (tetap): ");
    printInfo(L);
    printf("List salinan (berubah): ");
    printInfo(L2);

    delAll(&L);
    delAll(&L2);
}

int main() {
    printf("=== PROGRAM SINGLE LINKED LIST LENGKAP ===\n");
    printf("=== PRAKTIKUM STRUKTUR DATA ===\n\n");

    // Demo berbagai operasi
    demoSearching();
    demoInsertDelete();
    demoAdvancedOperations();

    printf("\n=== PROGRAM SELESAI ===\n");

    return 0;
}

```

Screenshots Output

```

PS C:\SMT 3\Struktur Data\modul-amer-2\103112430036_MuhamadNaufalAmmar_12-IF-03_Modul5> cd "c:\SMT 3\Struktur Data\modul-amer-2\103112430036_MuhamadNaufalAmmar_12-IF-03_Modul5\Guided\" ;
g++ main.cpp -o main } ; if ($?) { .\main }
=== PROGRAM SINGLE LINKED LIST LENGKAP ===
=== PRAKTIKUM STRUKTUR DATA ===

=== DEMO OPERASI SEARCHING ===
List awal: Isi list: 10 -> 20 -> 30 -> 40 -> 50
Nilai 20 DITEMUKAN pada address: 000002217F5DC400
Nilai 99 TIDAK DITEMUKAN
Elemen sebelum 20 adalah: 10

=== DEMO OPERASI INSERT DAN DELETE ===
1. Insert First (10, 20, 30):
Isi list: 30 -> 20 -> 10

2. Insert Last (40, 50):
Isi list: 30 -> 20 -> 10 -> 40 -> 50

3. Insert After 20 (25):
Isi list: 30 -> 20 -> 25 -> 10 -> 40 -> 50

4. Delete First:
Elemen yang dihapus: 30
Isi list: 20 -> 25 -> 10 -> 40 -> 50

5. Delete elemen 25:
Isi list: 20 -> 10 -> 40 -> 50

6. Delete Last:
Elemen yang dihapus: 50
Isi list: 20 -> 10 -> 40

=== DEMO OPERASI LAMOUT ===
List awal: Isi list: 1 -> 2 -> 3 -> 4 -> 5
Jumlah elemen: 5

1. Invert List:
Setelah diinvert: Isi list: 5 -> 4 -> 3 -> 2 -> 1

2. Copy List:
List asli: Isi list: 5 -> 4 -> 3 -> 2 -> 1
List salinan: Isi list: 5 -> 4 -> 3 -> 2 -> 1

3. Modifikasi list salinan:
List asli (tetap): Isi list: 5 -> 4 -> 3 -> 2 -> 1
List salinan (berubah): Isi list: 99 -> 5 -> 4 -> 3 -> 2 -> 1

=== PROGRAM SELESAI ===

```

## Deskripsi:

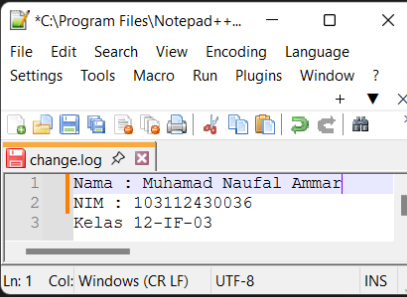
Program Single Linked List ini merupakan implementasi struktur data dinamis yang menyimpan data secara berantai menggunakan pointer, di mana setiap node berisi nilai data (info) dan penunjuk ke node berikutnya (next). Program terdiri dari tiga file utama: list.h untuk deklarasi tipe data dan fungsi, list.cpp untuk implementasi operasi linked list, serta main.cpp untuk demonstrasi penggunaannya. Operasi dasarnya meliputi pembuatan list kosong (CreateList), pengecekan list (ListEmpty), manajemen memori (alokasi dan dealokasi), serta manipulasi data seperti penambahan (insertFirst, insertAfter, insertLast), penghapusan (delFirst, delAfter, delLast, delP), dan pencarian elemen (findElm, findBefore). Fungsi tambahan seperti invertList dan copyList juga disertakan untuk membalik serta menyalin isi list. Melalui program ini, mahasiswa dapat mempelajari prinsip kerja pointer, pengelolaan memori dinamis, dan efisiensi operasi dalam struktur data linear non-statis.

### C. Unguired (berisi screenshot source code & output program disertai penjelasannya)

#### Unguired 1

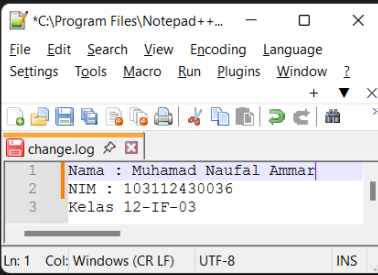
##### Singlylist.h

```
Unguired > C: Singlylist.h > ...
C:\SMT 3\Struktur Data\modul-amer-2\103112430036_MuhamadNaufalAmmar_12-IF-03_Modul5\Unguired
3 #include <iostream>
4 using namespace std;
5
6 #define Nil NULL
7
8 typedef int infotype;
9 typedef struct Elmlist *address;
10
11 struct Elmlist {
12     infotype info;
13     address next;
14 };
15
16 struct List {
17     address First;
18 };
19
20 void createlist(List &L);
21 address alokasi(infotype x);
22 void dealokasi(address &P);
23 void insertFirst(List &L, address P);
24 void printInfo(List L);
25 address findElm(List L, infotype x);
26 int sumInfo(List L);
27
28 #endif
```



##### Singlylist.cpp

```
Unguired > C: Singlylist.cpp > alokasi(infotype)
1 #include "Singlylist.h"
2 void createlist(List &L) {
3     L.First = Nil;
4 }
5 address alokasi(infotype x) {
6     address P = new Elmlist;
7     P->info = x;
8     P->next = Nil;
9     return P;
10 }
11 void dealokasi(address &P) {
12     delete P;
13     P = Nil;
14 }
15 void insertFirst(List &L, address P) {
16     P->next = L.First;
17     L.First = P;
18 }
19 void printInfo(List L) {
20     address P = L.First;
21     while (P != Nil) {
22         cout << P->info;
23         if (P->next != Nil) cout << " -> ";
24         P = P->next;
25     }
26     cout << endl;
27 }
28 address findElm(List L, infotype x) {
29     address P = L.First;
30     while (P != Nil) {
31         if (P->info == x) {
32             return P;
33         }
34         P = P->next;
35     }
36     return Nil;
37 }
38
39 int sumInfo(List L) {
40     int total = 0;
41     address P = L.First;
42     while (P != Nil) {
43         total += P->info;
44         P = P->next;
45     }
46     return total;
47 }
48
```





## Main.cpp

```
#include "Singlylist.h"
#include "Singlylist.cpp"

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

    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);

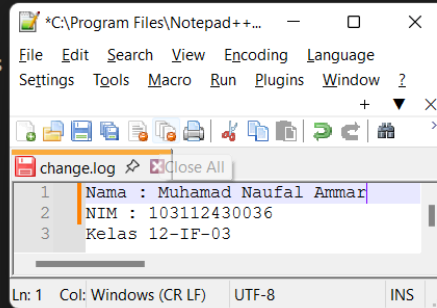
    P5 = alokasi(9);
    insertFirst(L, P5);

    cout << "Isi list: ";
    printInfo(L);

    address hasilCari = findElm(L, 8);
    if (hasilCari != Nil) {
        cout << "Elemen dengan info 8 ditemukan di alamat: " << hasilCari << endl;
    } else {
        cout << "Elemen dengan info 8 tidak ditemukan." << endl;
    }

    int total = sumInfo(L);
    cout << "Jumlah total info: " << total << endl;

    return 0;
}
```



## Screenshots Output

```
PS C:\SMT 3\Struktur Data\modul-amer-2\103112430036_MuhamadNaufalAmmar_12-IF-03_Modul5\Guided> cd "c:\SMT 3\Struktur Data\modul-amer-2\103112430036_MuhamadNaufalAmmar_12-IF-03_Modul5\Unguided\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
Isi list: 9 -> 12 -> 8 -> 0 -> 2
Elemen dengan info 8 ditemukan di alamat: 0x1e5d8991990
Jumlah total info: 31
```

## Deskripsi:

Program ini merupakan implementasi dari ADT (Abstract Data Type) Singly Linked List yang digunakan untuk menyimpan dan mengelola sekumpulan data secara dinamis menggunakan pointer. Dalam program ini, setiap elemen list terdiri dari dua bagian, yaitu nilai data (info) dan penunjuk ke elemen berikutnya (next). Fungsi-fungsi seperti createList, alokasi, insertFirst, dan printInfo digunakan untuk membuat list, menambahkan elemen, serta menampilkan isi list secara berurutan. Selain itu, ditambahkan juga fungsi findElm untuk mencari elemen tertentu berdasarkan nilai info, dan sumInfo untuk menghitung total nilai dari seluruh elemen dalam list. Melalui program ini, mahasiswa dapat memahami bagaimana struktur data linked list bekerja di memori, serta bagaimana pengelolaan elemen dilakukan tanpa menggunakan array statis, sehingga lebih efisien dan fleksibel.

#### D. Kesimpulan

Berdasarkan hasil analisis dan implementasi, dapat disimpulkan bahwa Singly Linked List merupakan struktur data dinamis yang efisien untuk menyimpan dan mengelola data yang ukurannya dapat berubah-ubah. Melalui penggunaan pointer, setiap elemen (node) dapat saling terhubung tanpa harus disimpan secara berurutan di memori. Implementasi fungsi seperti **insertFirst**, **findElm**, dan **sumInfo** menunjukkan bagaimana operasi dasar linked list bekerja sesuai dengan teori di mana penyisipan di awal list memiliki kompleksitas waktu  $O(1)$ , sementara pencarian dan penjumlahan seluruh elemen membutuhkan waktu  $O(n)$ . Struktur ini memberikan fleksibilitas tinggi dalam pengelolaan data, namun memiliki kelemahan dalam akses acak karena memerlukan traversal dari awal list. Secara keseluruhan, konsep Singly Linked List pada kode program ini mendukung pemahaman mahasiswa tentang manajemen memori dinamis dan efisiensi algoritma dalam pemrograman berbasis struktur data.

#### E. Referensi

Design UDL Virtual. (2023). *What is singly linked list in data structure?*

International Journal of Humanities, Information Technology, and Innovation (IJHIT). (2022). *Implementation of dynamic data structures using linked list.*

PubHTML5. (2021). *Data structure and algorithms: Linked list concepts*. Retrieved

International Journal of Engineering Research & Management Technology (IJERMT). (2014). *Comparative study of array and linked list data structures.*