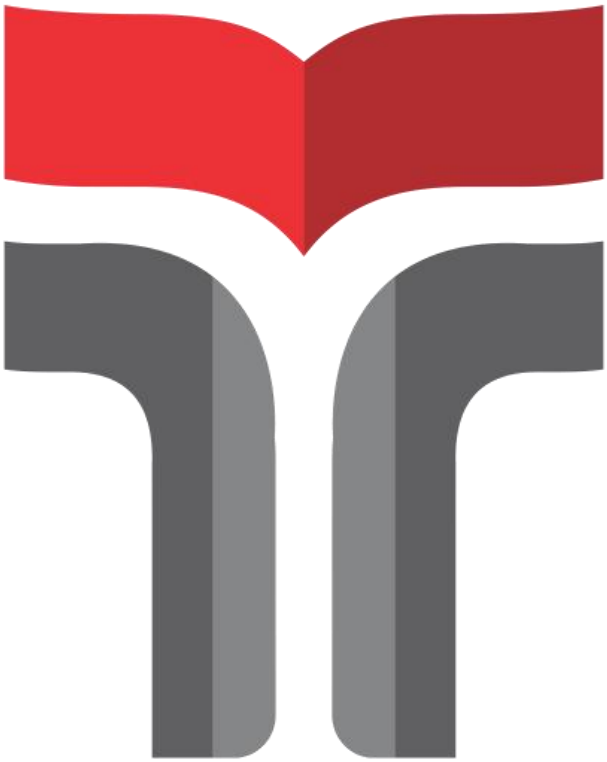
LAPORAN PRAKTIKUM
MODUL 04
"SINGLE LINKED LIST(BAGIAN PERTAMA)"



Oleh:

NAMA : Alvin Bagus Firmansyah

NIM : 2311104070

KELAS : SE-07-02

Dosen:

Wahyu Andi Saputra, S.Pd., M.Eng

PRODI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

INSTITUT TEKNOLOGI TELKOM PURWOKERTO

2024

I. 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

II. DASAR TEORI

4.1 *Linked List dengan Pointer*

Linked list (biasa disebut *list* saja) adalah salah satu bentuk struktur data (representasi penyimpanan) berupa serangkaian elemen data yang saling berkait (berhubungan) dan bersifat fleksibel karena dapat tumbuh dan mengerut sesuai kebutuhan. Data yang disimpan dalam *Linked list* bisa berupa data tunggal atau data majemuk. Data tunggal merupakan data yang hanya terdiri dari satu data (variabel), misalnya: nama bertipe *string*. Sedangkan data majemuk merupakan sekumpulan data (*record*) yang di dalamnya terdiri dari berbagai tipe data, misalnya: Data Mahasiswa, terdiri dari Nama bertipe *string*, NIM bertipe *long integer*, dan Alamat bertipe *string*.

Linked list dapat diimplementasikan menggunakan *Array* dan *Pointer* (*Linked list*).

Yang akan kita gunakan adalah *pointer*, karena beberapa alasan, yaitu :

1. *Array* bersifat statis, sedangkan *pointer* dinamis.
2. Pada *linked list* bentuk datanya saling bergandengan (berhubungan) sehingga lebih mudah memakai *pointer*.
3. Sifat *linked list* yang fleksibel lebih cocok dengan sifat *pointer* yang dapat diatur sesuai kebutuhan.
4. Karena *array* lebih susah dalam menangani *linked list*, sedangkan *pointer* lebih mudah.
5. *Array* lebih cocok pada kumpulan data yang jumlah elemen maksimumnya sudah diketahui dari awal.

Dalam implementasinya, pengaksesan elemen pada *Linked list* dengan *pointer* bisa menggunakan

(->) atau tanda titik (.).

Model-model dari ADT *Linked list* yang kita pelajari adalah :

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*

Setiap model ADT *Linked list* di atas memiliki karakteristik tertentu dan dalam penggunaannya disesuaikan dengan kebutuhan.

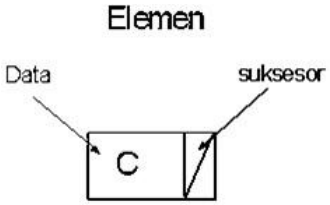
Secara umum operasi-operasi ADT pada *Linked list*, yaitu :

1. Penciptaan dan inisialisasi *list* (*Create List*).
2. Penyisipan elemen *list* (*Insert*).
3. Penghapusan elemen *list* (*Delete*).
4. Penelusuran elemen *list* dan menampilkannya (*View*).
5. Pencarian elemen *list* (*Searching*).
6. Pengubahan isi elemen *list* (*Update*).

4.2 Single Linked List

Single Linked list merupakan model ADT *Linked list* yang hanya memiliki satu arah *pointer*.

Komponen elemen dalam *single linked list*:



Keterangan:

Elemen: segmen-segmen data yang terdapat dalam suatu *list*.

Data: informasi utama yang tersimpan dalam sebuah elemen.

Suksesor: bagian elemen yang berfungsi sebagai penghubung antar elemen.

Sifat dari *Single Linked list*:

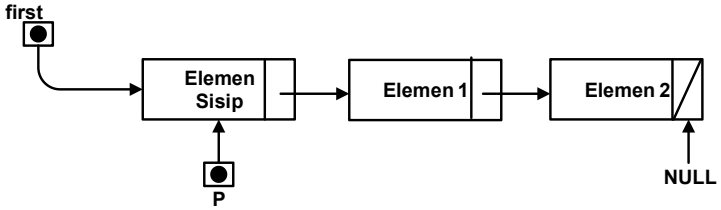
1. Hanya memerlukan satu buah *pointer*.
2. *Node* akhir menunjuk ke Nil kecuali untuk *list circular*.
3. Hanya dapat melakukan pembacaan maju.
4. Pencarian sequensial dilakukan jika data tidak terurut.
5. Lebih mudah ketika melakukan penyisipan atau penghapusan di tengah *list*.

Istilah-istilah dalam *Single Linked list* :

1. *first*/*head*: *pointer* pada *list* yang menunjuk alamat elemen pertama *list*.
2. *next*: *pointer* pada elemen yang berfungsi sebagai *successor* (penunjuk) alamat elemen di depannya.
3. Null/Nil: artinya tidak memiliki nilai, atau tidak mengacu ke mana pun, atau kosong.
4. *Node*/simpul/elemen: merupakan tempat penyimpanan data pada suatu memori tertentu.

Gambaran sederhana *single linked list* dengan elemen kosong:

sederhana *single linked list* dengan 3 elemen:



Contoh deklarasi struktur data *single linked list*:

```
/*file : list.h*/
#ifndef LIST_H_INCLUDED
#define LIST_H_INCLUDED


#define Nil NULL
#define info(P) (P)->info
#define next(P) (P)->next #define
first(L) ((L).first) using
namespace std;
/*deklarasi record dan struktur data
list*/ typedef int infotype; typdef struct
elmlist *address; struct elmlist
{    infotype info;    address next;
};
struct
list{    address
first;
};
#endif // TEST_H_INCLUDED
```

Misal untuk data mahasiswa yang terdiri dari nama dan nim.

```
/*file : list.h*/
#ifndef LIST_H_INCLUDED
#define LIST_H_INCLUDED


#define Nil NULL
#define info(P) (P)->info
#define next(P) (P)->next
#define first(L) ((L).first)

using namespace std;
/*deklarasi record dan struktur data list*/
truct mahasiswa{    char nama[30]    char nim[10]
}
typedef mahasiswa infotype;


typdef struct elmlist *address;
struct elmlist {    infotype
info;    address next;
};
struct
list{    address
first;
};
#endif // TEST_H_INCLUDED
```

```
3
2
4
2
5
2
6
2
7
```

4.2.1 Pembentukan Komponen-Komponen List

A. Pembentukan List

Adalah sebuah proses untuk membetuk sebuah *list* baru. Biasanya nama fungsi yang digunakan **`createList()`**. Fungsi ini akan mengeset nilai awal *list* yaitu *first*(*list*) dan *last*(*list*) dengan nilai Nil.

B. Pengalokasian Memori

Adalah proses untuk mengalokasikan memori untuk setiap elemen data yang ada dalam *list*. Fungsi yang biasanya digunakan adalah nama fungsi yang biasa digunakan `alokasi()`.

Sintak alokasi pada C:

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

Keterangan:

P	= variabel *pointer* yang mengacu pada elemen yang dialokasikan.
address	= tipe data *pointer* dari tipe data elemen yang akan dialokasikan.

Elmlist = tipe data atau *record* elemen yang dialokasikan.

Contoh deklarasi struktur data *single linked list*:

Misal untuk data mahasiswa yang terdiri dari nama dan nim.

```
address alokasi(mahasiswa m){      address p
= (address)malloc(sizeof(elmlist));
info(p) = m;       return p;
}
```

Namun pada Cpp. Penggunaan **malloc** dapat dipersingkat menggunakan sintak **new.**

Sintak alokasi pada Cpp:

```
P = new elmlist;
```

Keterangan:

P = variabel *pointer* yang mengacu pada elemen yang dialokasikan. *address* = tipe data *pointer* dari tipe data elemen yang akan dialokasikan.

Contoh deklarasi struktur data *single linked list*:
Misal untuk data mahasiswa yang terdiri dari nama dan nim.

```
address alokasi(mahasiswa m){
    address p = new elmlist;
    info(p) = m;
    return p;
}
```


C. Dealokasi

Untuk menghapus sebuah *memory address* yang tersimpan atau telah dialokasikan dalam bahasa pemrograman C digunakan sintak ***free***, sedangkan pada Cpp digunakan sintak ***delete***, seperti berikut.

Sintak pada C:

```
free( p );
```

Sintak pada Cpp:

```
delete p;
```

D. Pengecekan List

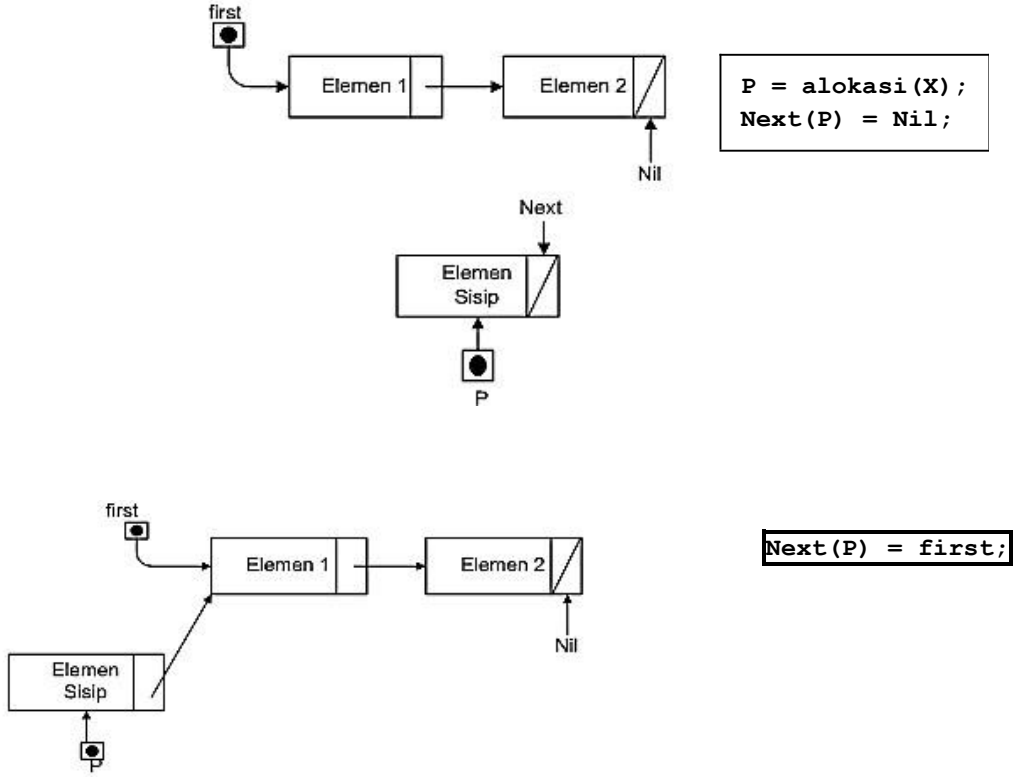
Adalah fungsi untuk mengecek apakah *list* tersebut kosong atau tidak. Akan mengembalikan nilai true jika *list* kosong dan nilai *false* jika *list* tidak kosong. Fungsi yang digunakan adalah `isEmpty()`.

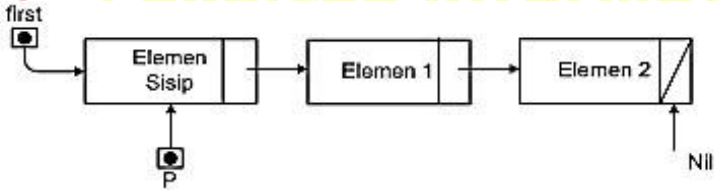
4.2.2 Insert

A. Insert First

Merupakan metode memasukkan elemen data ke dalam *list* yang diletakkan pada awal *list*.

Langkah-langkah dalam proses *insert first*:






```
First(L) = P;
```

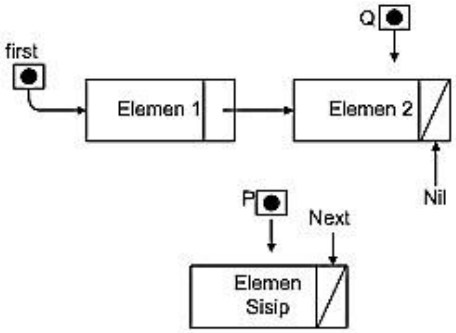
Gambar 4-6 *Single Linked list Insert First* (3)

```
/* contoh syntax insert first */ void insertFirst(List &L,
address &P){     next (P) = first(L);     first(L) = P; }
```

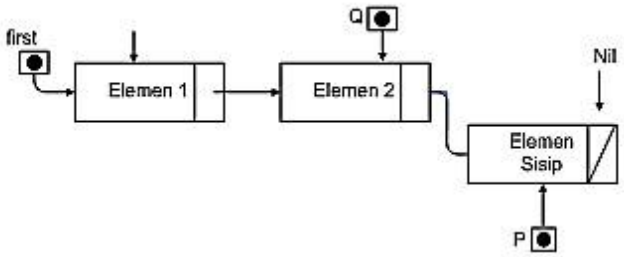
B. *Insert Last*

Merupakan metode memasukkan elemen data ke dalam *list* yang diletakkan pada akhir *list*.

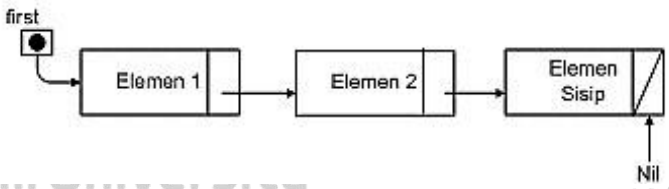
Langkah dalam *insert last* :




```
P = alokasi(X);
Next(P) = Nil;
```

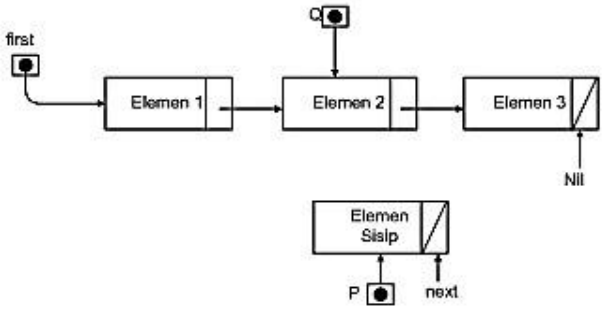



```
Next(Q)=P;
```

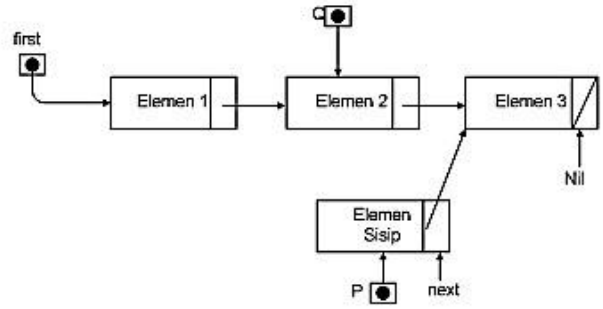


C. *Insert After*

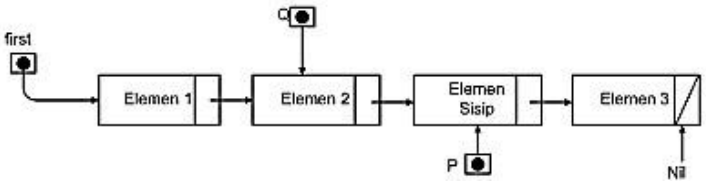
Merupakan metode memasukkan data ke dalam *list* yang diletakkan setelah *node* tertentu yang ditunjuk oleh *user*. Langkah dalam *insert after*:




```
P = alokasi(X);
Next(P) = Nil;
```




```
Next(P) = Next(Q);
```




```
Next(Q) = P;
```

4.2.3 View

Merupakan operasi dasar pada *list* yang menampilkan isi *node*/simpul dengan suatu penelusuran *list*. Mengunjungi setiap *node* kemudian menampilkan data yang tersimpan pada *node* tersebut.

Semua fungsi dasar diatas merupakan bagian dari ADT dari singgle *linked list*, dan aplikasi pada bahasa pemrograman Cp semua ADT tersebut tersimpan dalam *file* *.c dan *file* *.h.

```
1   /*file : list .h*/
2   /* contoh ADT list berkait dengan representasi fisik pointer*/
3   /* representasi address dengan pointer*/
4   /* info tipe adalah integer */
5   #ifndef list_H
6   #define list_H
7   #include "boolean.h"
8   #include <stdio.h>
9   #define Nil NULL
10  #define info(P) (P)->info
11  #define next(P) (P)->next
12  #define first(L) ((L).first)
13
14
15  /*deklarasi record dan struktur data list*/
16  typedef int infotype;
17  typedef struct elmlist
18  *address; struct
19  elmlist{  infotype info;
20   address next;
21  };
22
23
24  /* definisi list : */
25  /* list kosong jika First(L)=Nil */
26  /* setiap elemen address P dapat diacu info(P) atau next(P)
27  */ struct list {  address first;
28  };
29  /********* pengecekan apakah list kosong ***********/
30  boolean ListEmpty(list L);
31  /*mengembalikan nilai true jika list kosong*/
32  /********* pembuatan list kosong *********/
33  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 */


/********* penambahan elemen
**********/ void insertFirst(list &L,
address P); /* I.S. sembarang, P sudah
dialokasikan
   F.S. menempatkan elemen beralamat P pada awal list */
void insertAfter(list &L, address P, address Prec); /*
I.S. sembarang, P dan Prec alamt 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
dialokasikan
   F.S. menempatkan elemen beralamat P pada akhir list */


/********** proses semau 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 */


#endif
```

```
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 )
```

III. GUIDED

1.Single Linked List

Single Linked List (atau *Singly Linked List*) adalah struktur data linear yang terdiri dari serangkaian node di mana setiap node hanya memiliki satu referensi atau pointer ke node berikutnya. Dengan kata lain, dalam single linked list, setiap elemen dihubungkan ke elemen berikutnya, namun tidak ada referensi ke elemen sebelumnya.

Komponen Single Linked List:

1. Node:
    - Setiap elemen dalam linked list disebut node.
    - Setiap node berisi dua bagian:
        - Data: Menyimpan nilai (misalnya, data mahasiswa).
        - Next: Pointer yang menunjuk ke node berikutnya dalam daftar.
2. Head:
    - Head adalah pointer yang menunjuk ke node pertama dalam linked list. Jika linked list kosong, head akan bernilai nullptr.
3. Tail (opsional dalam kasus single linked list):
    - Tail adalah pointer yang menunjuk ke node terakhir dalam linked list. Kadang-kadang, dalam implementasi single linked list, kita juga menyimpan pointer ke elemen terakhir (tail) untuk mempercepat operasi penambahan di akhir list.

Karakteristik Single Linked List:

- Dinamika Memori: Berbeda dengan array, yang memiliki ukuran tetap, linked list dapat dinamis, yaitu dapat tumbuh dan menyusut sesuai dengan kebutuhan. Ini berarti kita bisa terus menambahkan node tanpa harus mengalokasikan ukuran tetap sebelumnya.
- Arah Satu Arah: Single linked list hanya bergerak maju dari satu node ke node berikutnya. Kita tidak bisa bergerak mundur.

- Ukuran Variabel: Tidak seperti array, ukuran linked list bisa berubah seiring penambahan atau penghapusan elemen.

Operasi Dasar Single Linked List:

1. Penambahan Node (Insert):
    - Insert di depan (insertDepan): Node baru ditambahkan di awal linked list. Operasi ini meng-update head untuk menunjuk ke node baru.
    - Insert di belakang (insertBelakang): Node baru ditambahkan di akhir linked list. Node terakhir akan mengarahkan ke node baru, dan tail akan diperbarui.
2. Penghapusan Node (Delete):
    - Hapus di depan (hapusDepan): Node pertama dihapus, dan head akan menunjuk ke node kedua.
    - Hapus di belakang (hapusBelakang): Node terakhir dihapus. Proses ini mengharuskan traversal dari head hingga mencapai node sebelum yang terakhir.
3. Traversing (Menjelajah List):
    - Traversing berarti mengunjungi setiap node dari head sampai tail untuk membaca data atau melakukan operasi tertentu (misalnya, mencetak isi list).
4. Pengecekan Kosong:
    - Fungsi isEmpty() digunakan untuk memeriksa apakah list kosong (yaitu head menunjuk ke nullptr).
5. Menghitung Panjang List:
    - Fungsi hitungList() menghitung jumlah node dengan melakukan iterasi dari node pertama hingga node terakhir.
6. Penghapusan Seluruh List:
    - Fungsi clearList() menghapus semua node dalam list dan membebaskan memori yang digunakan oleh node tersebut.

Kelebihan Single Linked List:

1. Ukuran Dinamis: Memori dialokasikan sesuai kebutuhan (penambahan node baru), sehingga kita tidak perlu menetapkan ukuran tetap seperti pada array.
2. Penghapusan dan Penambahan Mudah: Penambahan dan penghapusan node di awal atau di akhir dapat dilakukan dengan efisien dibandingkan dengan array yang memerlukan penggeseran elemen.

3. Penggunaan Memori Lebih Efisien: Karena memori dialokasikan secara dinamis, tidak ada ruang yang terbuang seperti pada array statis.

   Kekurangan Single Linked List:

1. Akses Lambat: Untuk mengakses elemen di tengah linked list, kita harus melakukan traversal dari awal hingga mencapai elemen yang diinginkan, sehingga akses secara acak tidak efisien (berbeda dengan array yang memiliki akses acak dalam O(1)).
2. Kebutuhan Memori Lebih Besar: Setiap node membutuhkan tambahan memori untuk menyimpan pointer ke node berikutnya, yang berarti lebih banyak memori dibandingkan array jika hanya menyimpan data.
3. Arah Satu Arah: Tidak bisa bergerak mundur, sehingga traversal mundur (dari akhir ke awal) memerlukan linked list dengan dua arah (doubly linked list).

   Kode Program:

main.cpp | STRUKTUR DATA 4\unguided\unguided2.c | main.cpp | main.cpp | main.cpp

```cpp
#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;
```

main.cpp | STRUKTUR DATA 4\unguided\unguided2.c | main.cpp | main.cpp | main.cpp

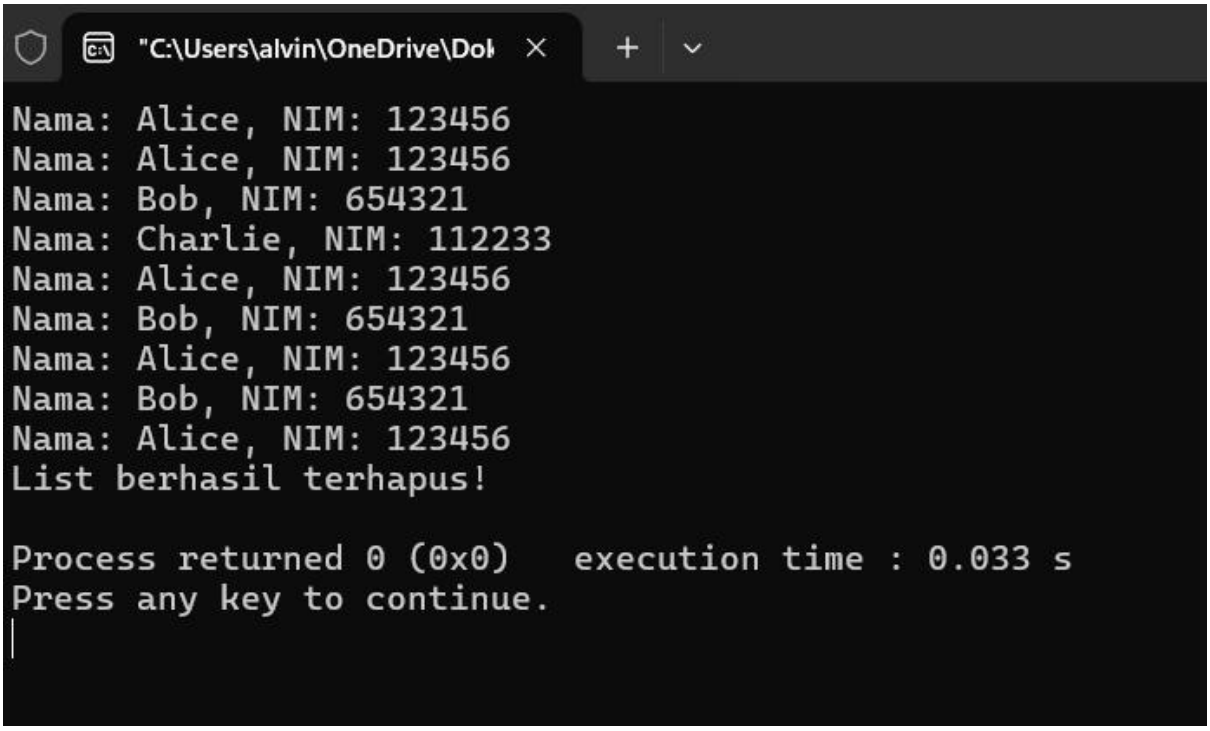
```cpp
            head = baru;
        }
    }

    // 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;
```

main.cpp | STRUKTUR DATA 4\unguided\unguided2.c | main.cpp | main.cpp | main.cpp

```cpp
    }
}

    // 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;
    }
}

    // Hapus List
void clearList() {
    Node *current = head;
```

```
118     while (current != nullptr) {
119         Node *hapus = current;
120         current = current->next;
121         delete hapus;
122     }
123     head = tail = nullptr;
124     cout << "List berhasil terhapus!" << endl;
125 }
126
127 // Main function
128 int main() {
129     init();
130
131     // Contoh data mahasiswa
132     mahasiswa m1 = {"Alice", "123456"};
133     mahasiswa m2 = {"Bob", "654321"};
134     mahasiswa m3 = {"Charlie", "112233"};
135
136     // Menambahkan mahasiswa ke dalam list
137     insertDepan(m1);
138     tampil();
139     insertBelakang(m2);
140     tampil();
141     insertDepan(m3);
142     tampil();
143
144     // Menghapus elemen dari list
145     hapusDepan();
146     tampil();
147     hapusBelakang();
148     tampil();
149
150     // Menghapus seluruh list
151     clearList();
152
153     return 0;
154 }
155
```

Kode Program:



2. Single Linked List

Single Linked List adalah struktur data yang terdiri dari node-node yang saling terhubung. Setiap **node** memiliki dua bagian:

1. **Data**: Menyimpan nilai.
2. **Pointer**: Menunjuk ke node berikutnya.

Operasi dasar pada Single Linked List meliputi:

- **Insert First**: Menambah node di awal list.
- **Insert Last**: Menambah node di akhir list.
- **Traversal**: Menjelajah seluruh node untuk mengakses atau menghitung jumlah elemen.
- **Delete**: Menghapus node tertentu atau seluruh list.

Kelebihannya adalah fleksibilitas dalam menambah atau menghapus elemen secara dinamis, sementara kekurangannya adalah akses lambat karena harus traversal dari awal untuk menemukan elemen.

main.cpp | STRUKTUR DATA 4\unguided\unguided2.c | main.cpp | main.cpp | main.cpp | main.cpp

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

// Fungsi untuk dealokasi memori node
void dealokasi(Node* node) {
    delete node; // Mengembalikan memori yang digunakan oleh node
}

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

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

// Menambahkan elemen di akhir list
```

main.cpp | STRUKTUR DATA 4\unguided\unguided2.c | main.cpp | main.cpp | main.cpp | main.cpp

```cpp
void insertLast(Node* &head, int value) {
    Node* newNode = alokasi(value); // Alokasi memori untuk elemen baru
    if (newNode != nullptr) {
        if (isListEmpty(head)) {  // Jika list kosong
            head = newNode;       // Elemen baru menjadi elemen pertama
        } else {
            Node* temp = head;
            while (temp->next != nullptr) { // Mencari elemen terakhir
                temp = temp->next;
            }
            temp->next = newNode; // Menambahkan elemen baru di akhir list
        }
    }
}

// Menampilkan semua elemen dalam list
void printList(Node* head) {
    if (isListEmpty(head)) {
        cout << "List kosong!" << endl;
    } else {
        Node* temp = head;
        while (temp != nullptr) {  // Selama belum mencapai akhir list
            cout << temp->data << " "; // Menampilkan data elemen
            temp = temp->next; // Melanjutkan ke elemen berikutnya
        }
        cout << endl;
    }
}

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

```
79
80  // Menghapus semua elemen dalam list dan dealokasi memori
81  void clearList(Node* &head) {
82      while (head != nullptr) {
83          Node* temp = head;  // Simpan pointer ke node saat ini
84          head = head->next;  // Pindahkan ke node berikutnya
85          dealokasi(temp);    // Dealokasi node
86      }
87  }
88
89  int main() {
90      Node* head = nullptr;  // Membuat list kosong
91
92      // Menambahkan elemen ke dalam list
93      insertFirst(head, 10); // Menambahkan elemen 10 di awal list
94      insertLast(head, 20);  // Menambahkan elemen 20 di akhir list
95      insertLast(head, 30);  // Menambahkan elemen 30 di akhir list
96
97      // Menampilkan isi list
98      cout << "Isi List: ";
99      printList(head);
100
101     // Menampilkan jumlah elemen
102     cout << "Jumlah elemen: " << countElements(head) << endl;
103
104     // Menghapus semua elemen dalam list
105     clearList(head);
106
107     // Menampilkan isi list setelah penghapusan
108     cout << "Isi List setelah penghapusan: ";
109     printList(head);
110
111     return 0;
112 }
113
```

Hasil Output:

```
"C:\Users\alvin\OneDrive\Dok
Isi List: 10 20 30
Jumlah elemen: 3
Isi List setelah penghapusan: List kosong!

Process returned 0 (0x0)   execution time : 0.024 s
Press any key to continue.
```

IV. UNGUIDED

1. Membuat Single Linked List

Kode

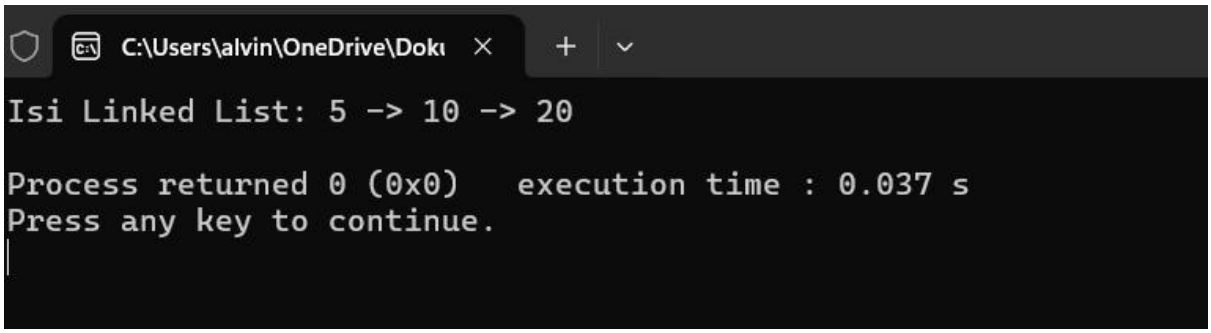
Progam:

main.cpp

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

// Struktur node
struct Node {
    int data;       // Data yang disimpan dalam node
    Node* next;     // Pointer ke node berikutnya
};

// Kelas untuk single linked list
class SingleLinkedList {
private:
    Node* head;     // Pointer ke node pertama (head)

public:
    // Constructor
    SingleLinkedList() {
        head = nullptr;
    }

    // Fungsi untuk menambah node di depan
    void insertDepan(int nilai) {
        Node* newNode = new Node(); // Buat node baru
        newNode->data = nilai;      // Set nilai data
        newNode->next = head;       // Hubungkan node baru ke head saat ini
        head = newNode;             // Set head ke node baru
    }

    // Fungsi untuk menambah node di belakang
    void insertBelakang(int nilai) {
        Node* newNode = new Node(); // Buat node baru
        newNode->data = nilai;      // Set nilai data
        newNode->next = nullptr;    // Node baru akan menjadi node terakhir

        // Jika list kosong, node baru menjadi head
        if (head == nullptr) {
            head = newNode;
        } else {
            Node* temp = head;
            // Cari node terakhir
            while (temp->next != nullptr) {
                temp = temp->next;
            }
```

```
            temp->next = newNode;  // Hubungkan node terakhir ke node baru
        }
    }

    // Fungsi untuk mencetak seluruh isi linked list
    void cetakList() {
        if (head == nullptr) {
            cout << "Linked list kosong!" << endl;
        } else {
            Node* temp = head;
            while (temp != nullptr) {
                cout << temp->data;
                if (temp->next != nullptr) {
                    cout << " -> ";
                }
                temp = temp->next;
            }
            cout << endl;
        }
    }
};

// Fungsi utama
int main() {
    SingleLinkedList list;  // Buat objek single linked list

    // Operasi pada linked list sesuai contoh
    list.insertDepan(10);      // Tambah node di depan (nilai: 10)
    list.insertBelakang(20);   // Tambah node di belakang (nilai: 20)
    list.insertDepan(5);       // Tambah node di depan (nilai: 5)

    // Cetak isi linked list
    cout << "Isi Linked List: ";
    list.cetakList();

    return 0;
}
```

Hasil Output:



2. Menghapus Node pada Linked List

Kode Progam:

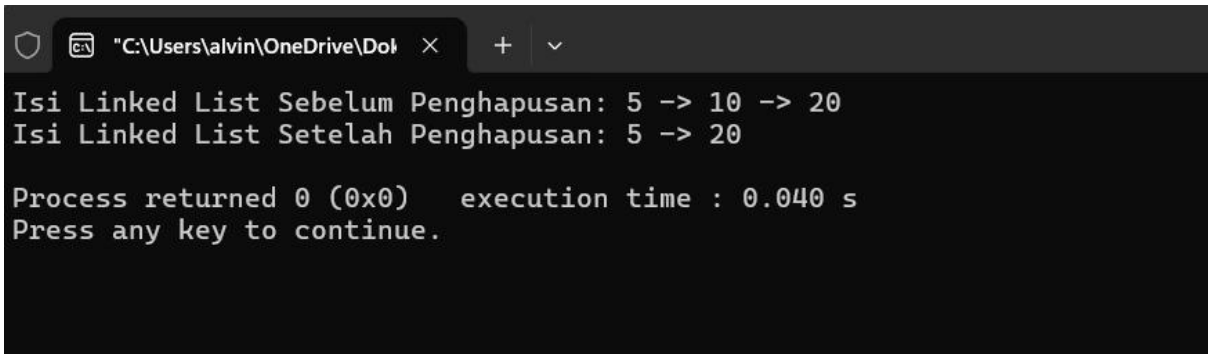
main.cpp | STRUKTUR DATA 4\unguided\unguided2.c | main.cpp

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

// Struktur node
struct Node {
    int data;       // Data yang disimpan dalam node
    Node* next;     // Pointer ke node berikutnya
};

// Kelas untuk single linked list
class SingleLinkedList {
private:
    Node* head;     // Pointer ke node pertama (head)

public:
    // Constructor
    SingleLinkedList() {
        head = nullptr;
    }

    // Fungsi untuk menambah node di depan
    void insertDepan(int nilai) {
        Node* newNode = new Node(); // Buat node baru
        newNode->data = nilai;      // Set nilai data
        newNode->next = head;       // Hubungkan node baru ke head saat ini
        head = newNode;             // Set head ke node baru
    }

    // Fungsi untuk menambah node di belakang
    void insertBelakang(int nilai) {
        Node* newNode = new Node(); // Buat node baru
        newNode->data = nilai;      // Set nilai data
        newNode->next = nullptr;    // Node baru akan menjadi node terakhir

        // Jika list kosong, node baru menjadi head
        if (head == nullptr) {
            head = newNode;
        } else {
            Node* temp = head;
            // Cari node terakhir
            while (temp->next != nullptr) {
                temp = temp->next;
            }
            temp->next = newNode;  // Hubungkan node terakhir ke node baru
        }
    }
```

```
47
48      // Fungsi untuk menghapus node dengan nilai tertentu
49      void hapusNode(int nilai) {
50          if (head == nullptr) {
51              cout << "Linked list kosong!" << endl;
52              return;
53          }
54
55          // Jika node yang akan dihapus adalah head
56          if (head->data == nilai) {
57              Node* temp = head;
58              head = head->next; // Ubah head ke node berikutnya
59              delete temp;       // Hapus node lama
60              return;
61          }
62
63          // Mencari node yang akan dihapus
64          Node* temp = head;
65          Node* prev = nullptr;
66
67          while (temp != nullptr && temp->data != nilai) {
68              prev = temp;
69              temp = temp->next;
70          }
71
72          // Jika node tidak ditemukan
73          if (temp == nullptr) {
74              cout << "Node dengan nilai " << nilai << " tidak ditemukan!" << endl;
75              return;
76          }
77
78          // Hapus node
79          prev->next = temp->next;
80          delete temp;
81      }
82
83      // Fungsi untuk mencetak seluruh isi linked list
84      void cetakList() {
85          if (head == nullptr) {
86              cout << "Linked list kosong!" << endl;
87          } else {
88              Node* temp = head;
89              while (temp != nullptr) {
90                  cout << temp->data;
91                  if (temp->next != nullptr) {
```

```
92                      cout << " -> ";
93                  }
94                  temp = temp->next;
95              }
96              cout << endl;
97          }
98      }
99  };
100
101 // Fungsi utama
102 int main() {
103     SingleLinkedList list;  // Buat objek single linked list
104
105     // Operasi pada linked list sesuai contoh
106     list.insertDepan(10);      // Tambah node di depan (nilai: 10)
107     list.insertBelakang(20);   // Tambah node di belakang (nilai: 20)
108     list.insertDepan(5);       // Tambah node di depan (nilai: 5)
109          void SingleLinkedList::insertDepan(int nilai)
110     // Cetak isi linked list sebelum penghapusan
111     cout << "Isi Linked List Sebelum Penghapusan: ";
112     list.cetakList();
113
114     // Hapus node dengan nilai tertentu (nilai: 10)
115     list.hapusNode(10);
116
117     // Cetak isi linked list setelah penghapusan
118     cout << "Isi Linked List Setelah Penghapusan: ";
119     list.cetakList();
120
121     return 0;
122 }
123
```

Hasil Output:



3. Mencari dan Menghitung Panjang Linked List

Kode Progam:

main.cpp | STRUKTUR DATA 4\unguided\unguided2.c | main.cpp | main.cpp

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

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

// Function to insert a node at the front
void insertFront(Node** head, int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = *head;
    *head = newNode;
}

// Function to insert a node at the back
void insertBack(Node** head, int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;

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

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

// Function to search for a node with a given value
bool searchNode(Node* head, int value) {
    Node* temp = head;
    while (temp != nullptr) {
        if (temp->data == value) {
```

```
main.cpp X  STRUKTUR DATA 4\unguided\unguided2.c X  main.cpp X  main.cpp X
40              return true;
41          }
42          temp = temp->next;
43      }
44      return false;
45  }
46
47  // Function to count the length of the linked list
48  int countLength(Node* head) {
49      int count = 0;
50      Node* temp = head;
51      while (temp != nullptr) {
52          count++;
53          temp = temp->next;
54      }
55      return count;
56  }
57
58  // Function to print the linked list
59  void printList(Node* head) {
60      Node* temp = head;
61      while (temp != nullptr) {
62          cout << temp->data;
63          temp = temp->next;
64          if (temp != nullptr) {
65              cout << " -> ";
66          }
67      }
68      cout << endl;
69  }
70
71  int main() {
72      Node* head = nullptr;
73
74      insertFront(&head, 10);
75      insertBack(&head, 20);
76      insertFront(&head, 5);
77
78      cout << "Linked List: ";
79      printList(head);
80
81      // Search for a value
82      int searchValue = 20;
83      if (searchNode(head, searchValue)) {
84          cout << "Node dengan nilai " << searchValue << " ditemukan." << endl;
85      } else {
86          cout << "Node dengan nilai " << searchValue << " tidak ditemukan." << endl;
87      }
88
89      // Count length of the list
90      int length = countLength(head);
91      cout << "Panjang linked list: " << length << endl;
92
93      return 0;
94  }
95
```

Hasil Output:

```
"C:\Users\alvin\OneDrive\Dok
Linked List: 5 -> 10 -> 20
Node dengan nilai 20 ditemukan.
Panjang linked list: 3

Process returned 0 (0x0)   execution time : 0.043 s
Press any key to continue.
```

V. KESIMPULAN

Single Linked List adalah struktur data yang efisien untuk menyimpan elemen yang jumlahnya dapat berubah secara dinamis. Dengan menggunakan node yang terdiri dari data dan pointer ke node berikutnya, linked list memungkinkan operasi penambahan dan penghapusan elemen dengan cepat. Meskipun akses ke elemen tertentu membutuhkan traversal dari awal, keunggulan dalam fleksibilitas dan pengelolaan memori menjadikannya pilihan yang baik untuk aplikasi yang memerlukan struktur data yang dapat berkembang. Namun, penggunaannya harus disesuaikan dengan kebutuhan spesifik, mengingat kekurangan dalam kecepatan akses acak.