SINGLE LINKED LIST

ALGORITMA DAN STRUKTUR DATA

Institut Teknologi Sumatera

POKOK BAHASAN

- Pengenalan Linked List dan Single Linked List
- Pendefinisian dalam Single Linked List
- Skema Dasar Pemrosesan List
- Fungsional dalam Single Linked List

TUJUAN KULIAH

- Mahasiswa mampu memahami konsep senarai berantai tunggal / single linked list
- Mahasiswa mampu memahami definisi dan fungsi dalam single linked list
- Mahasiswa mampu implementasi single linked list dalam bahasa pemrograman C++

PENGENALAN LINKED LIST

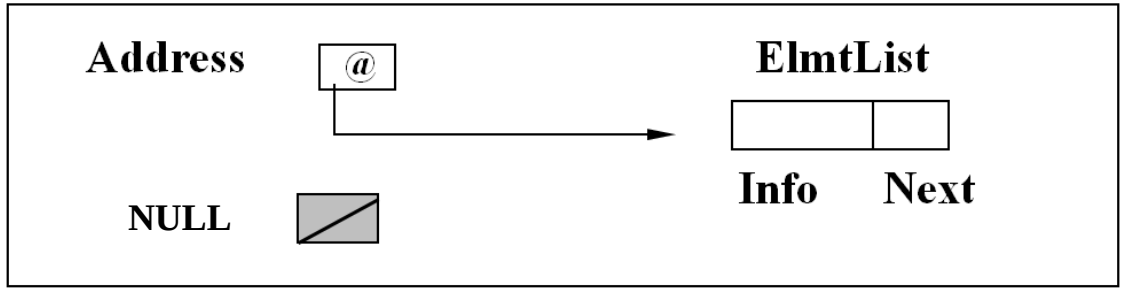
- Linked list dapat menyimpan data dengan jumlah yang dinamis
- Merupakan salah satu struktur data dinamis yang paling sederhana
  - Berisi kumpulan komponen yang disusun berdasarkan aturan tertentu dengan bantuan petunjuk
- Komponen tersebut biasanya dikenal dengan istilah **Node** / **Elemen.**

MENGAPA HARUS LINKED LIST ??

- Dalam kehidupan nyata jumlah data yang harus diproses tidaklah menentu, sehingga relatif memakan waktu jika harus menggunakan array (ataupun array dinamis dengan redeclare dan copy memory).
- Linked list memiliki waktu yang konstan dalam proses pemasukan atau penghapusan data (terlebih operasi-operasi yang melibatkan data di bagian tengah).
- Dua hal ini lah yang sering dipertimbangkan ketika akan menggunakan linked list dari pada array, ***Dynamic Size*** dan ***Ease of Insertion/Deletion***.

SINGLE LINKED LIST

- Single Linked list merupakan salah satu varian dari linked list dengan penyusun nodenya adalah
  - Informasi tentang data atau isi dari elemen (Info)
  - Informasi tentang alamat elemen selanjutnya (Next)
- Penggambaran dari single linked list



SINGLE LINKED LIST

- Sebuah elemen dalam single linked list dikenali dari:
  - Alamat dari elemen pertama dalam list yang biasanya disimpan pada **FIRST**.
  - Alamat elemen berikutnya yang dapat diperoleh dari informasi **NEXT** pada suatu elemen
  - Alamat dari elemen terakhir dalam list yang biasanya disimpan pada **LAST** (bersifat opsional)
- Setiap elemen mempunyai alamat, sebagai tempat elemen yang disimpan dapat diacu/dikunjungi
- Untuk mengacu sebuah elemen, alamat harus terdefinisi

DEFINISI

- Jika **L** adalah List, dan **P** adalah address:
  - ✓ Untuk mengacu alamat elemen pertama list **L**, digunakan notasi **First(L)**
  - ✓ Elemen yang diacu oleh **P** dapat diakses informasinya dengan notasi **Selektor:**
    - **Info(P)** : nilai yang disimpan
    - **Next(P)** : alamat elemen berikutnya
- Definisi List Kosong
  - ✓ List **L** adalah list kosong jika **First(L) = NULL**
- Definisi Elemen Terakhir
  - ✓ Elemen **P** dikatakan sebagai elemen terakhir jika **Next(P) = NULL**

STRUKTUR DARI ELEMEN LIST

```
typedef int infotype;
typedef struct TElmtList *address;
typedef struct TElmtList {
   infotype info;
   address next;
} ElmtList;

typedef struct List {
   address first;
};
```

DEFINISI FUNGSIONAL SINGLE LINKED LIST

- **CreateEmpty** {*Input : List, Output : List dengan kondisi kosong*}
- **IsEmpty** {*Input : List, Output : Boolean status kondisi list kosong*}
- **Allocation** {*Input : Suatu nilai, Output : Elemen dalam list*}
- **Deallocation** {*Input : Alamat, Output : Alamat yang sudah free*}
- **Insert** {*Input : List dan suatu nilai, Output : Nilai masuk ke elemen baru dan diposisikan ke dalam list*}
- **Delete** {*Input : List dan variabel untuk menyimpan nilai yang dihapus, Output : List yang sudah berkurang elemen di dalamnya*}

FUNGSIONAL SINGLE LINKED LIST

CreateEmpty {*Input : List, Output : List dengan kondisi kosong*}

```
void CreateEmpty(List *L) {
     (*L).first = NULL;
}
```

FUNGSIONAL SINGLE LINKED LIST

IsEmpty {*Input : List, Output : Boolean status kondisi list kosong*}

```
bool IsEmpty(List L) {
    return ((L).first == NULL);
}
```

FUNGSIONAL SINGLE LINKED LIST

Allocation {*Input : Suatu nilai, Output : Elemen dalam list*}

```
address Allocation(infotype x) {
    address NewElmt;
    NewElmt = (ElmtList*) malloc (sizeof(ElmtList));

    NewElmt->info = x;
    NewElmt->next = NULL;

    return NewElmt;
}
```

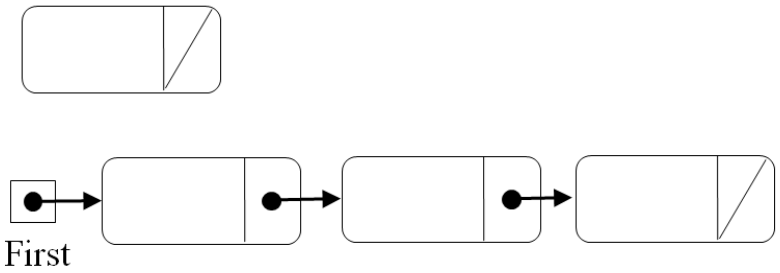
FUNGSIONAL SINGLE LINKED LIST

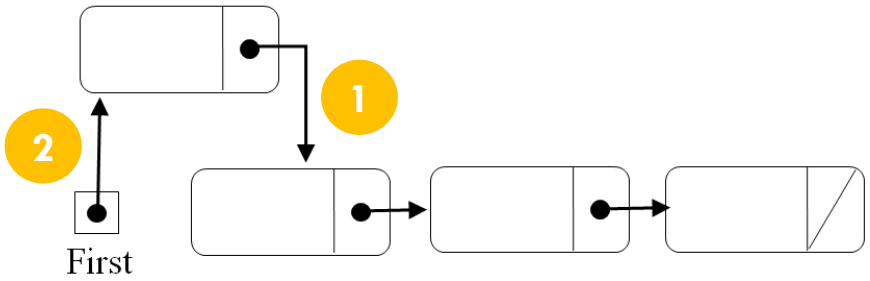
Deallocation {*Input : Alamat, Output : Alamat yang sudah free*}

```
void Deallocation(address hapus) {
    free(hapus);
}
```

FUNGSIONAL SINGLE LINKED LIST

Insert – Penambahan Data di Bagian Awal



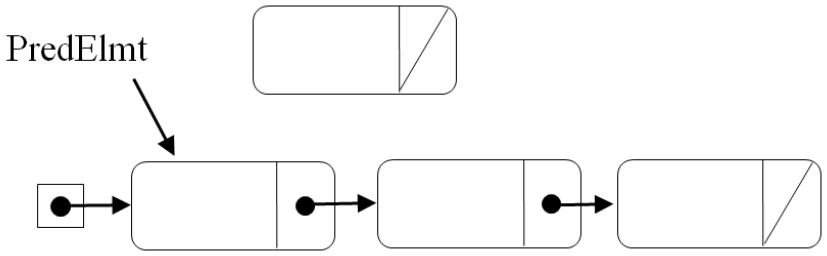


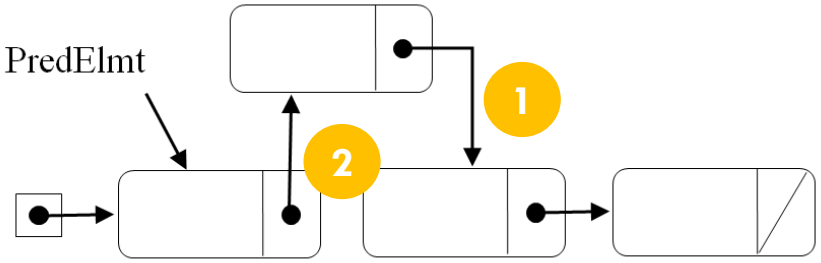

```
void InsertFirst(List *L, infotype x) {
    address NewElmt;
    NewElmt = Allocation(x);

    if (NewElmt != NULL) {
      1 NewElmt->next = (*L).first;
      2 (*L).first = NewElmt;
    }
}
```

FUNGSIONAL SINGLE LINKED LIST

Insert – Penambahan Data di Posisi Setelah Suatu Elemen List






```
void InsertAfter(address *PredElmt, infotype x) {
    address NewElmt;
    NewElmt = Allocation(x);

    if (NewElmt != NULL) {
      1 NewElmt->next = (*PredElmt)->next;
      2 (*PredElmt)->next = NewElmt;
    }
}
```

FUNGSIONAL SINGLE LINKED LIST

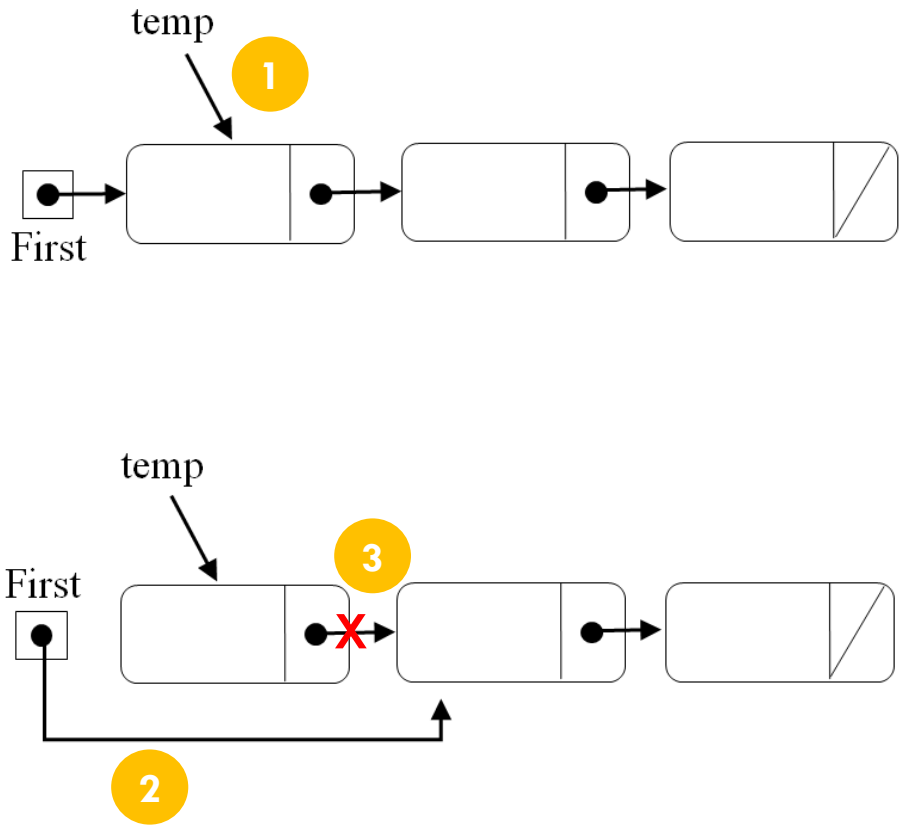
Insert – Penambahan Data pada Bagian Akhir

1. InsertLast akan sama seperti InsertFirst ketika list masih dalam kondisi kosong.
2. Ketika List tidak dalam kondisi kosong, maka akan dilakukan penelusuran hingga mendapatkan elemen terakhir dalam list.
3. Selanjutnya melakukan InsertAfter dengan inputan PredElmt adalah elemen terakhir dalam list.

```
void InsertLast(List *L, infotype x) {
    if (IsEmpty(*L)) {
      1 InsertFirst(&(*L), x);
    } else {
        address temp;
        temp = (*L).first;
      2 while (temp->next != NULL) {
            temp = temp->next;
        }
      3 InsertAfter(&temp, x);
    }
}
```

FUNGSIONAL SINGLE LINKED LIST

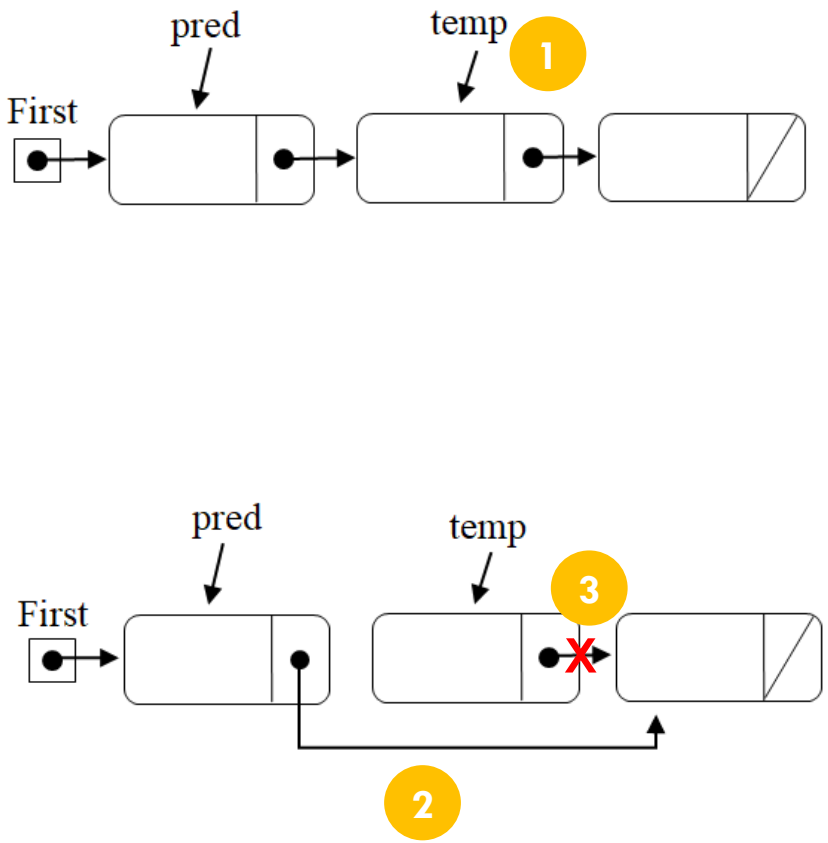
Delete – Penghapusan Data di Bagian Awal




```
void DeleteFirst(List *L, infotype *hapus) {
    if (!IsEmpty(*L)) {
        address temp;
    1   temp = (*L).first;
        *hapus = temp->info;
    2   (*L).first = (*L).first->next;
    3   temp->next = NULL;
        Deallocation(temp);
    }
}
```

FUNGSIONAL SINGLE LINKED LIST

Delete – Penghapusan Data di Bagian Tengah dari List




```
void DeleteAfter(address pred, infotype *hapus) {
    if (pred->next != NULL) {
        address temp;
    1   temp = pred->next;
        *hapus = temp->info;
    2   pred->next = temp->next;
    3   temp->next = NULL;
        Deallocation(temp);
    }
}
```

FUNGSIONAL SINGLE LINKED LIST

Delete – Penghapusan Data pada Bagian Akhir

1. Penghapusan data pada bagian akhir diawali dengan pencarian elemen terakhir dan menyimpan lokasi dari elemen terakhir dan sebelumnya.
2. Jika elemen terakhir sama dengan elemen pertama, maka dilakukan DeleteFirst.
3. Namun jika tidak, maka dilakukan DeleteAfter dengan menggunakan elemen sebelum yang terakhir sebagai parameter.

```
void DeleteLast(List *L, infotype *hapus) {
    if (!IsEmpty(*L)) {
        address temp, predTemp;
        predTemp = NULL;
        temp = (*L).first;
    1   while (temp->next != NULL) {
            predTemp = temp;
            temp = temp->next;
        }

        if (temp == (*L).first) {
        2   DeleteFirst(&(*L), &(*hapus));
        } else {
        3   DeleteAfter(predTemp, &(*hapus));
        }
    }
}
```

FUNGSIONAL SINGLE LINKED LIST

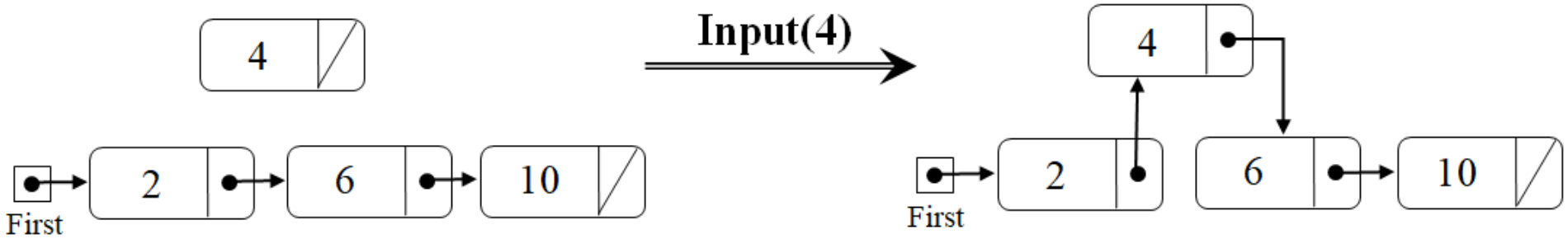
Pembacaan Data

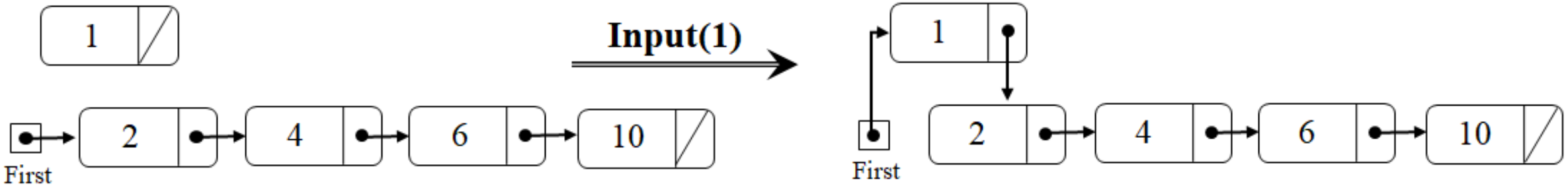
```
address temp = data.first;
while (temp != NULL) {
    cout << temp->info << endl;
    temp = temp->next;
}
```

LATIHAN ATAU TUGAS MANDIRI 1

- Buatlah program yang dapat menerima data masukan berupa angka (integer), yang menerapkan konsep Single Linked List.
- Setiap data yang dimasukkan, langsung akan diletakkan pada urutan yang benar.
- Yang mana data yang tersimpan dalam List selalu dalam kondisi terurut dari angka terkecil hingga terbesar.
- *Simulasi berada pada Slide Berikutnya.*

LATIHAN ATAU TUGAS MANDIRI 1 (SIMULASI)





LATIHAN ATAU TUGAS MANDIRI 2

- Tambahkan fitur/layanan penghapusan data dari kode program yang telah dibuat pada Latihan/Tugas Mandiri 1 sebelumnya.
- Yang mana, layanan ini akan dimulai dengan menanyakan angka yang akan dihapus dalam List.
- Jika program tidak menemukan angka dalam list maka diberikan informasi bahwa angka tersebut tidak tersimpan pada List.
- Namun, jika angka ditemukan maka lakukan pendeteksian lokasi dari angka tersebut, serta terapkan penghapusan data sesuai dengan lokasi data (DeleteFirst, DeleteAfter, ataupun DeleteLast).

TERIMA KASIH