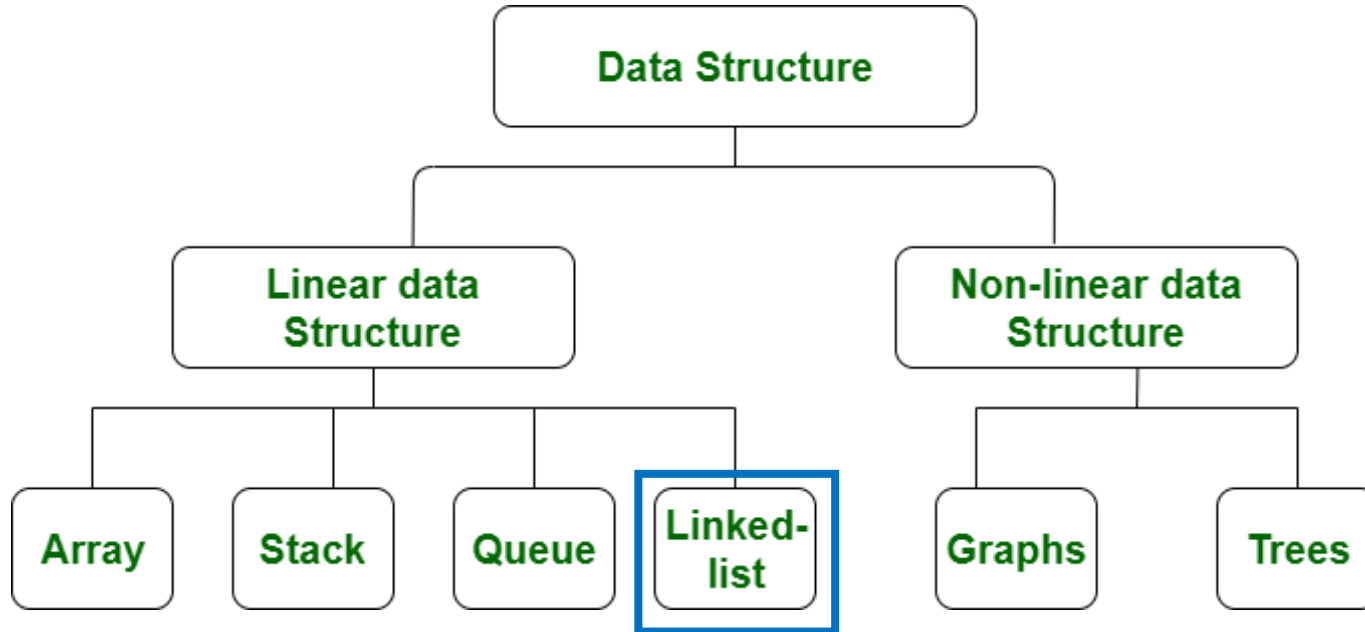




Linked List

Tim Ajar Algoritma dan Struktur Data
Genap 2023/2024

Jenis Struktur Data



Capaian Pembelajaran

- Mahasiswa memahami konsep linked list
- Mahasiswa memahami tahapan pembuatan linked list untuk menyelesaikan masalah

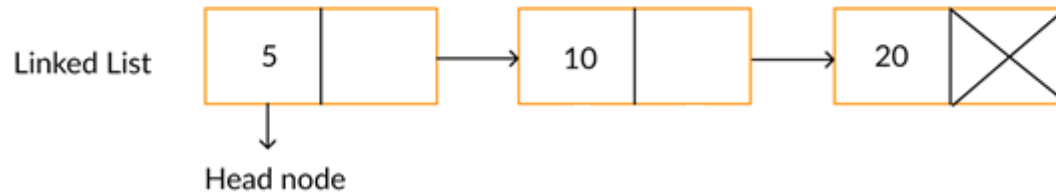
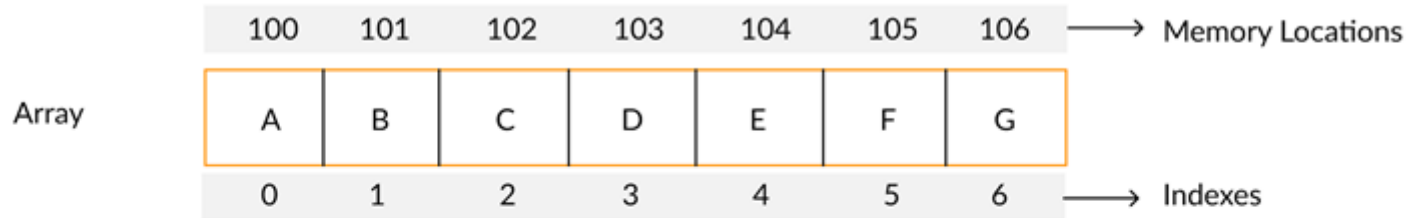
Pengantar

- Konsep struktur data linked list mengatasi kelemahan dari struktur data array.
- Salah satu kekurangan ketika menggunakan data array sebagai penyimpanan data adalah sifatnya yang statis.
- Array akan mengalokasikan memori dengan size tertentu saat instansiasi, walaupun slot memori tersebut belum terpakai untuk menyimpan data.

Definisi

- **Linked list:** struktur data linier yang dibangun dari satu atau lebih node yang saling terhubung yang menempati alokasi memori secara dinamis.
- **Node:** tempat penyimpanan data yang terdiri dari dua bagian/field.
 - ✓ **Field 1** adalah data, digunakan untuk menyimpan data/nilai.
 - ✓ **Field 2** adalah pointer ke node selanjutnya
- Node pertama pada Linked List disebut head.
- Node terakhir pada Linked List disebut tail
- Jika Linked List kosong, maka head dan tail akan bernilai null.

Array vs Linked List



Array VS Linked List

ARRAY	LINKED LIST
Statis	Dinamis
Penambahan/penghapusan data terbatas	Penambahan/penghapusan data tidak terbatas
Random access	Sequential access
Elemen-elemennya terletak pada lokasi memory yang berurutan	Elemen-elemennya bisa jadi terletak pada lokasi memory yang saling berjauhan

Array Vs Linked List

- Menyimpan koleksi elemen secara non-contiguously.
 - Elemen-elemen bisa jadi terletak pada lokasi memory yang saling berjauhan. Bandingkan dengan array dimana tiap-tiap elemen akan terletak pada lokasi memory yang berurutan.



Array representation



Linked list representation

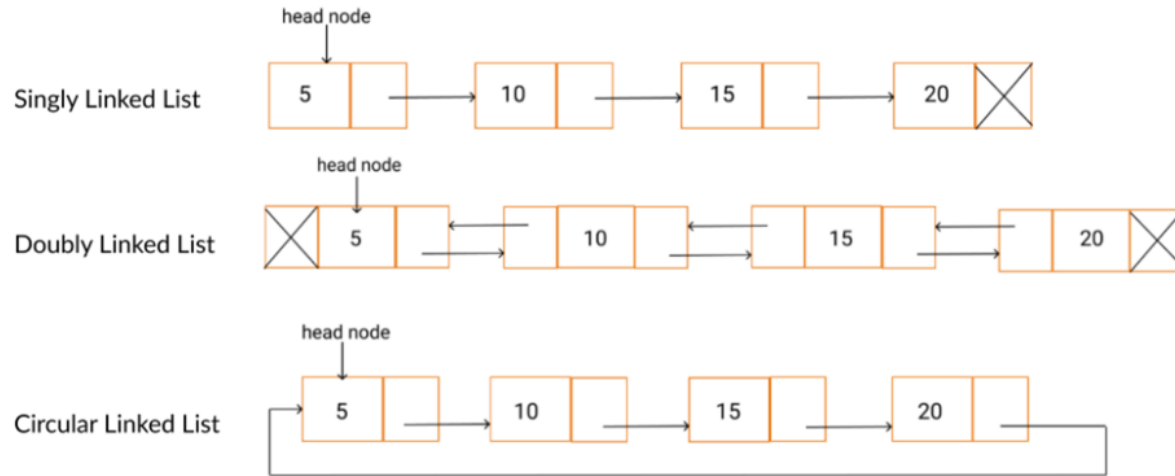


Kelebihan & Kekurangan Linked Lists

- Kelebihan: Struktur data yang dinamis, jumlah node dapat bertambah sesuai kebutuhan data.
- Kekurangan: Struktur data ini tidak dapat mengakses data berdasarkan index. Jika dibutuhkan pendekatan seperti ini, maka perlu dilakukan proses traverse mulai dari head ke node-node berikutnya secara urut sampai didapatkan data/index yang diinginkan.

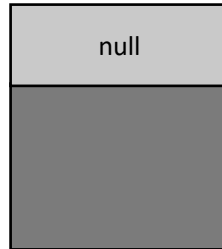
Jenis-Jenis Linked Lists

- Single Linked List: Memiliki penunjuk ke node berikutnya (next)
- Double Linked List: mempunyai dua penunjuk, yaitu next dan prev
- Circular linked list

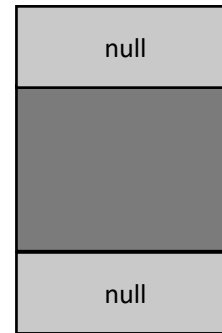


Gambaran Struktur Node

Single linked-list



Double linked-list



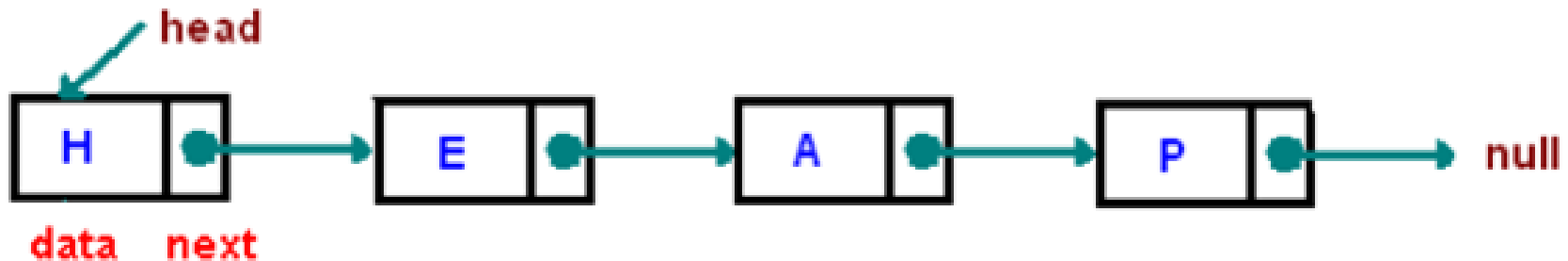
Link atau pointer



data

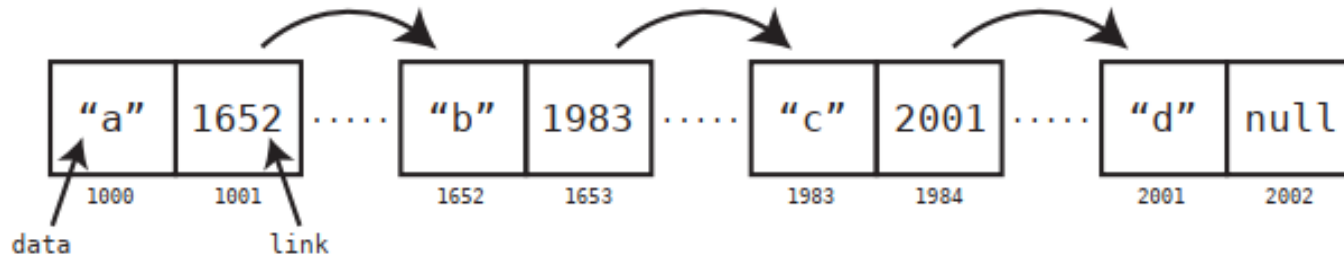
Konsep Single Linked Lists

- Linked List merupakan struktur data dinamis.
- Jumlah node dapat bertambah sesuai dengan kebutuhan.
- Program yang tidak diketahui jumlah datanya, sebaiknya menggunakan struktur data Linked List.



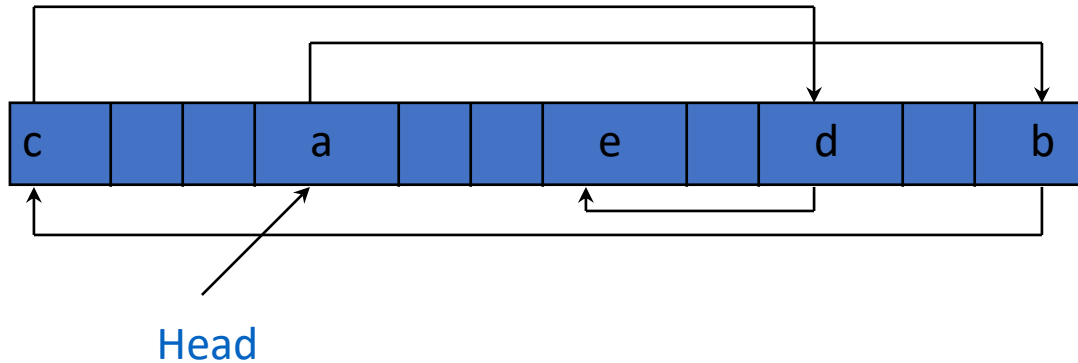
Konsep Single Linked Lists

- Single: pointer-nya hanya satu buah dan satu arah, yaitu menunjuk ke node sesudahnya.
- Node terakhir akan menunjuk ke NULL yang akan digunakan sebagai kondisi berhenti pada saat pembacaan isi linked list.
- Linked List tidak menggunakan memory cell secara berderet (row). Tetapi, ia memanfaatkan memory secara acak.
- Lalu bagaimana komputer mengetahui bahwa node itu merupakan satu linked lists yang sama ?
 - Kuncinya adalah data yang disimpan ke dalam node, setiap node juga menyimpan memory address untuk node berikutnya dalam satu linked list.



Ilustrasi Single Linked List

- Ilustrasi single linked list pada memory :



- Karena node a tidak ditunjuk oleh node manapun maka node ini adalah node yang paling depan (head).

Implementasi Node

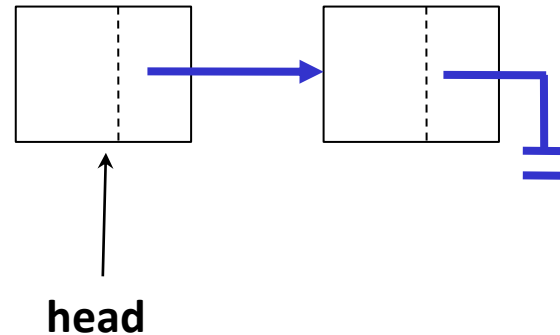
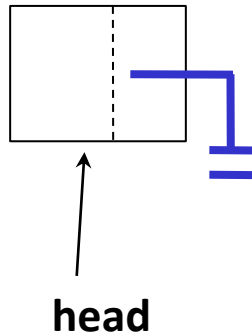
Class Node memiliki 2 atribut:

- Atribut data: menyimpan data/nilai pada setiap node
- Atribut pointer: menyimpan node berikutnya

```
public class Node {  
    int data;  
    Node next;  
  
    public Node(int data, Node next) {  
        this.data = data;  
        this.next = next;  
    }  
}
```

Head

- Dalam implementasinya, atribut head pada class LinkedList diisi dengan node pertama





Implementasi LinkedList

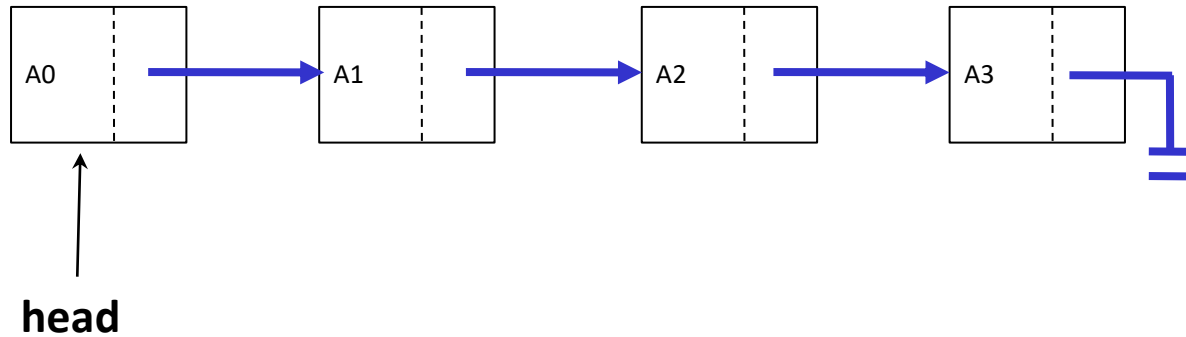
Class LinkedList hanya memiliki atribut head saja

Informasi node-node lainnya disimpan di mana? → head menyimpan pointer ke node kedua. Node kedua memiliki pointer ke node ketiga, dst

```
public class LinkedList {  
    Node head;  
}
```

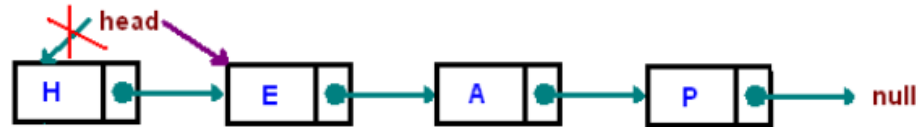
Contoh

- Linked list yang memiliki 4 node :

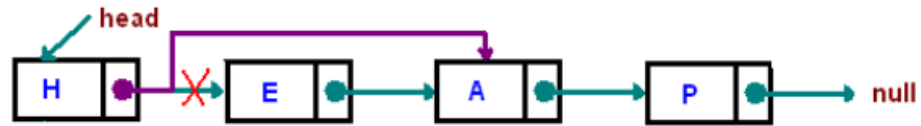


Cara Kerja Linked Lists

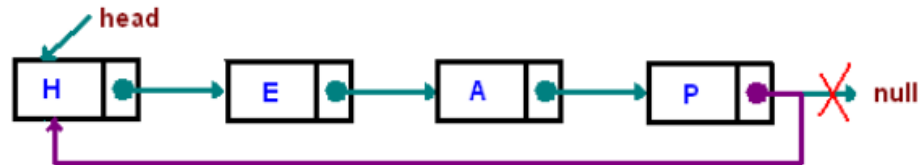
```
head = head.next;
```



```
head.next = head.next.next;
```



```
head.next.next.next.next = head;
```



Operasi pada Linked Lists

- isEmpty(): mengecek apakah linked list kosong
- print(): menampilkan seluruh elemen pada Linked Lists
- Operasi penambahan node
 - Di awal
 - Di akhir
 - Setelah node tertentu
- Operasi menghapus node
 - Di awal
 - Di akhir
 - Key tertentu
- Operasi Linked List dengan Index
 - Pengaksesan data node
 - Pengaksesan index node
 - Penambahan data
 - Penghapusan data

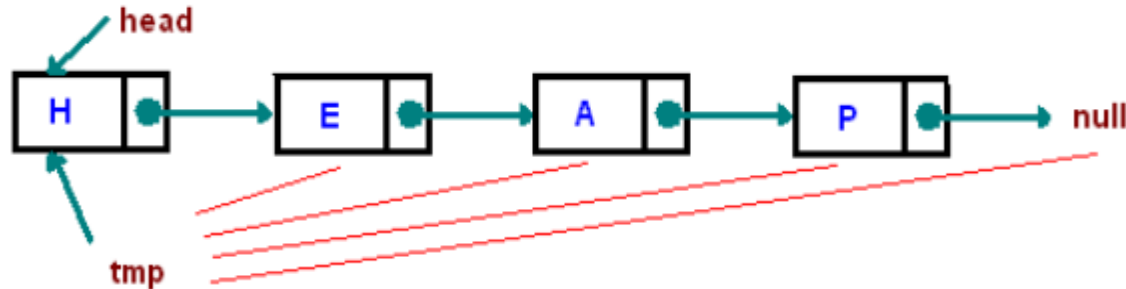
Operasi isEmpty()

- Digunakan untuk mengecek apakah linked list kosong.
- Linked List kosong jika head=null

```
public boolean isEmpty() {  
    return (head == null);  
}
```

Proses Traverse pada Linked List

- Proses melakukan kunjungan pada setiap node tepat satu kali. Dengan melakukan kunjungan secara lengkap, maka akan didapatkan urutan data yang tersimpan dalam linked list secara linier
- Proses ini dilakukan pada proses cetak data, penambahan data di akhir linked list dan pengaksesan linked list menggunakan index
- Proses ini dimulai dari head sampai hingga node terakhir (next bernilai null)
- Proses ini tidak mengubah nilai dari head.



Fungsi print()

- Untuk mencetak data seluruh node mulai dari head hingga tail

```
public void print() {  
    if (!isEmpty()) {  
        Node currentNode = head;  
  
        while (currentNode != null) {  
            System.out.print(currentNode.data + "\t");  
            currentNode = currentNode.next;  
        }  
    } else {  
        System.out.println("Linked list kosong");  
    }  
}
```

Operasi Penambahan

- `addFirst()`: menambahkan node baru di awal linked list
- `addLast()`: menambahkan node baru di akhir linked list
- `insertAfter()`: menambahkan node baru setelah node tertentu



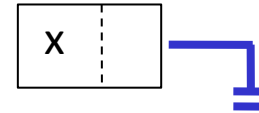
Fungsi addFirst()

- Digunakan untuk menambahkan node baru di awal Linked List
- Jika linked list kosong maka node input akan dijadikan sebagai head
- Jika pada linked list telah berisi node, maka:
 - Atribut next pada node input akan menunjuk head node
 - Node input akan dijadikan sebagai head yang baru

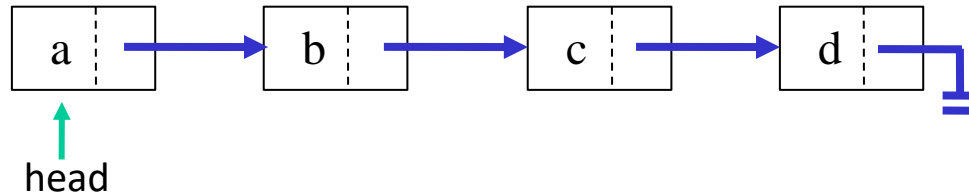
Ilustrasi : addFirst(x)

Menambahkan x di awal

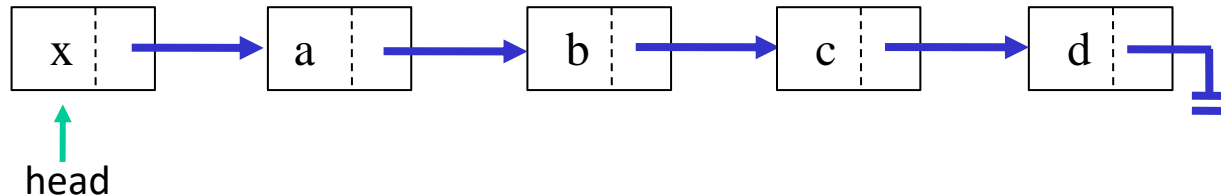
Node input



Kondisi awal linked list :



Setelah penambahan node x di awal:



Fungsi addFirst()

```
public void addFirst(int input) {  
    Node newNode = new Node(input, null);  
  
    if (isEmpty()) {  
        head = newNode;  
    } else {  
        newNode.next = head;  
        head = newNode;  
    }  
}
```

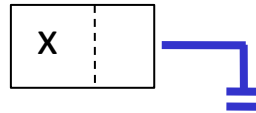
Fungsi addLast()

- Digunakan untuk menambahkan node baru di akhir Linked Lists.
- Jika kondisi awal node kosong maka node baru akan dijadikan sebagai head
- Jika linked list telah berisi node, maka:
 - Lakukan iterasi hingga node terakhir (tail) ditemukan
 - Set node baru sebagai next node dari tail

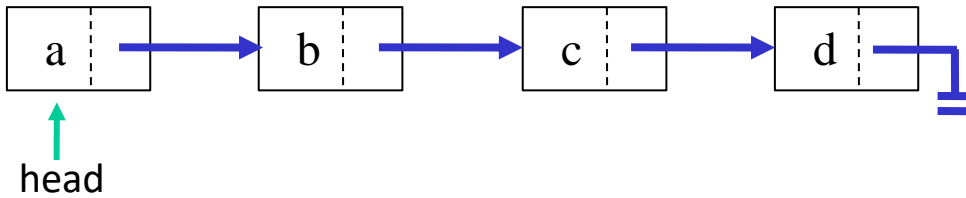
Ilustrasi : addLast()

Menambahkan x di akhir

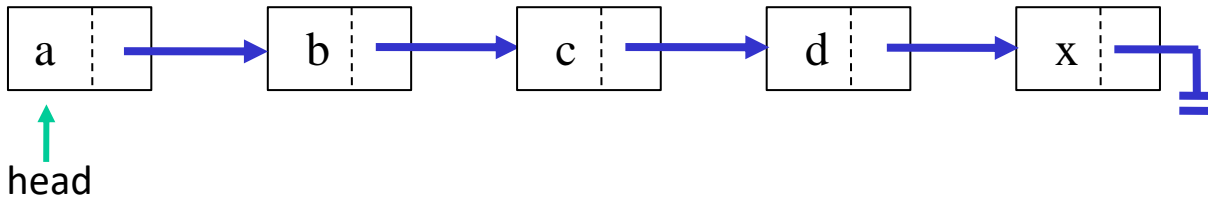
Node input



Kondisi awal linked list:



Setelah penambahan node x di akhir:



Fungsi addLast()

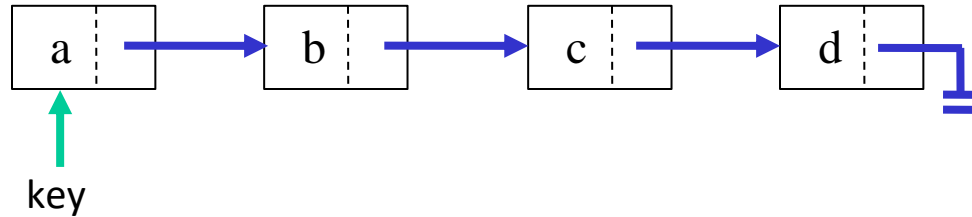
```
public void addLast(int input) {  
    Node newNode = new Node(input, null);  
  
    if (isEmpty()) {  
        head = newNode;  
    } else {  
        Node currentNode = head;  
  
        while (currentNode.next != null) {  
            currentNode = currentNode.next;  
        }  
  
        currentNode.next = newNode;  
    }  
}
```

Fungsi insertAfter()

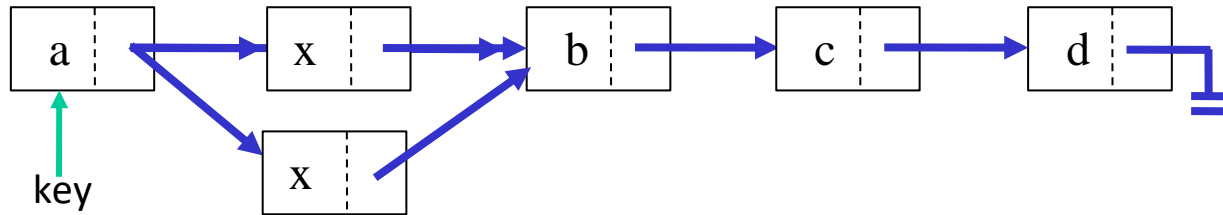
- Dilakukan untuk menambahkan node baru pada posisi setelah node yang berisi data tertentu (key)
- `insertAfter(key, input)` berarti tambahkan node baru berisi input pada posisi setelah node berisi data yang sama dengan key
- Jika linked list kosong, tampilkan warning
- Jika linked list berisi node, maka:
 - Lakukan iterasi hingga node yang berisi data sama dengan key ditemukan
 - Jadikan next node dari key node sebagai next node dari node baru
 - Jadikan node baru sebagai next node dari key node

Ilustrasi : Insert After()

- Kondisi Awal



Menyisipkan x pada lokasi setelah *key*.





Fungsi insertAfter()

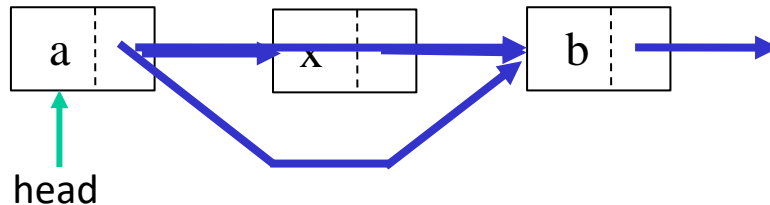
```
public void insertAfter(int key, int input) {  
    Node newNode = new Node(input, null);  
  
    if (!isEmpty()) {  
        Node currentNode = head;  
  
        do {  
            if (currentNode.data == key) {  
                newNode.next = currentNode.next;  
                currentNode.next = newNode;  
                System.out.println("Insert data is succeed.");  
                break;  
            }  
  
            currentNode = currentNode.next;  
        } while (currentNode != null);  
    } else {  
        System.out.println("Linked list kosong");  
    }  
}
```

Operasi Penghapusan

- `removeFirst()`: menghapus node pertama
- `removeLast()`: menghapus node terakhir
- `remove()`: menghapus node tertentu

Langkah-langkah menghapus elemen

- Proses menghapus dilakukan dengan mengabaikan elemen yang hendak dihapus
- Pointer yang seharusnya mengarah ke elemen tersebut diubah sehingga mengarah pada elemen selanjutnya
- Jika tidak ada pointer yang mengarah ke node x, maka node tersebut tidak dapat diakses lagi.
- Java Garbage Collector akan membersihkan alokasi memory yang tidak dipakai lagi atau tidak bisa diakses. Dengan kata lain, menghapus node x.



Fungsi removeFirst()

- Digunakan untuk menghapus node pertama pada Linked Lists.
- Jika linked list kosong, tampilkan warning
- Jadikan next node dari head sebagai head yang baru (Jika hanya terdapat 1 node, maka nilai head akan bernilai null)

Fungsi removeFirst()

```
public void removeFirst() {  
    if (!isEmpty()) {  
        head = head.next;  
    } else {  
        System.out.println("Linked list kosong");  
    }  
}
```

Fungsi removeLast()

- Digunakan untuk menghapus node terakhir pada Linked Lists.
- Jika linked list kosong, tampilkan warning
- Jika linked list berisi 1 node saja, kosongkan linked list dengan mengubah atribut head menjadi null
- Lakukan iterasi untuk menemukan node terakhir kedua, kemudian set next nodenya dengan null

Fungsi removeLast()

```
public void removeLast() {  
    if (isEmpty()) {  
        System.out.println("Linked list kosong");  
    } else if (head.next == null) {  
        head = null;  
    } else {  
        Node currentNode = head;  
  
        while (currentNode.next != null) {  
            if (currentNode.next.next == null) {  
                currentNode.next = null;  
                break;  
            }  
  
            currentNode = currentNode.next;  
        }  
    }  
}
```





Fungsi remove()

- Fungsi `remove(int key)` digunakan untuk menghapus node dengan data sesuai key
- Jika linked list kosong, tampilkan warning
- Jika data pada head sama dengan key, panggil `removeFirst()` untuk menghapus head
- Lakukan iterasi untuk menemukan node yang next node nya berisi key. Arahkan next node ke node setelah key



Fungsi remove()

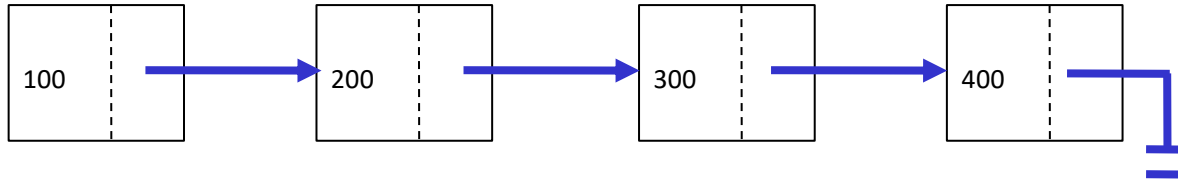
```
public void remove(int key) {  
    if (isEmpty()) {  
        System.out.println("Linked list kosong");  
    } else if (head.data == key) {  
        removeFirst();  
    } else {  
        Node currentNode = head;  
  
        while (currentNode.next != null) {  
            if (currentNode.next.data == key) {  
                currentNode.next = currentNode.next.next;  
                break;  
            }  
  
            currentNode = currentNode.next;  
        }  
    }  
}
```

Operasi Linked List dengan Index

- Pengaksesan data node: mengetahui data yang terletak pada indeks tertentu
- Pengaksesan index node: mengetahui index suatu node yang berisi data yang dicari
- Penambahan data: menambahkan data pada index tertentu
- Penghapusan data: menghapus data pada index tertentu

Latihan

1. Suatu link list berisi 4 node berikut:



- Tambahkan node baru dengan data 500 dari belakang.
- Tambahkan node baru dengan data 50 dari depan.
- Tambahkan node dengan data 250 setelah node 200
- Hapus node depan
- Hapus node belakang
- Hapus node yg memiliki data 300

* Gambarkan kondisi linked list untuk setiap perubahan di atas



Latihan (2)

2. Buat pseudocode/flowchart untuk 2 fungsi berikut
 - a. Fungsi insertAt(int index, int key) untuk menambahkan data baru pada index tertentu
 - b. Fungsi removeAt(index) untuk menghapus data pada index tertentu

