



Pertemuan 11

~Linked List~

Tim Ajar Algoritma dan Struktur Data 2021



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 memesan sejumlah memori sesuai pada saat deklarasi, 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, untuk menyimpan alamat tertentu.
 - Pointer disebut juga sebagai **link**



Definisi Linked Lists

- Jika linked list hanya berisi satu node maka pointer-nya akan menunjuk ke NULL.
- Jika linked list memiliki lebih dari satu node maka pointer menyimpan alamat dari node berikutnya. Sehingga antara node satu dengan node yang lain akan terhubung, kecuali node terakhir.
- Node terakhir pada linked list akan menunjuk null (nilai null merepresentasikan nilai tidak ada/nothing/unset reference).
- Awal dari struktur *Linked List* terdapat *head*.
- **Catatan:** *head* bukanlah *node* yang terpisah, melainkan *node* yang menunjuk ke *node* pertama.
- Jika *Linked List* kosong, maka *head* akan merujuk ke *null*.
- Pada penerapan yang berbeda terkadang dikenalkan juga *tail* (tetapi untuk menyederhanakan pembahasan dalam materi kali ini, implementasi hanya akan menggunakan *head*).



Array VS Linked List

ARRAY	LINKED LIST
Statis	Dinamis
Penambahan/ Penghapusan data terbatas	Penambahan/ Penghapusan data terbatas
Random Access	Sequential access
Penghapusan array tidak mungkin	Penghapusan Linked List mudah

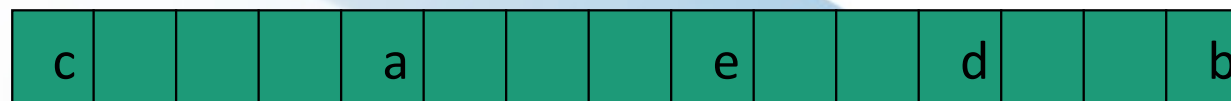


Array VS Linked List

- Menyimpan koleksi elemen secara non-contiguously.
 - Elemen dapat 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



Array VS Linked List

- Mengizinkan operasi penambahan atau penghapusan elemen ditengah-tengah koleksi dengan hanya membutuhkan jumlah perpindahan elemen yang konstan.
 - Bandingkan dengan array. Berapa banyak elemen yang harus dipindahkan bila akan menyisipi elemen ditengah-tengah array?



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 dari head dan mengikuti penunjuk next sampai didapatkan data/index yang diinginkan.



Jenis-jenis Linked Lists

- Single Linked List: seperti pada ilustrasi slide sebelumnya;
- Double Linked List: mempunyai dua penunjuk, yaitu next dan prev;




Gambaran Struktur Node

Single linked-list



Double linked-list



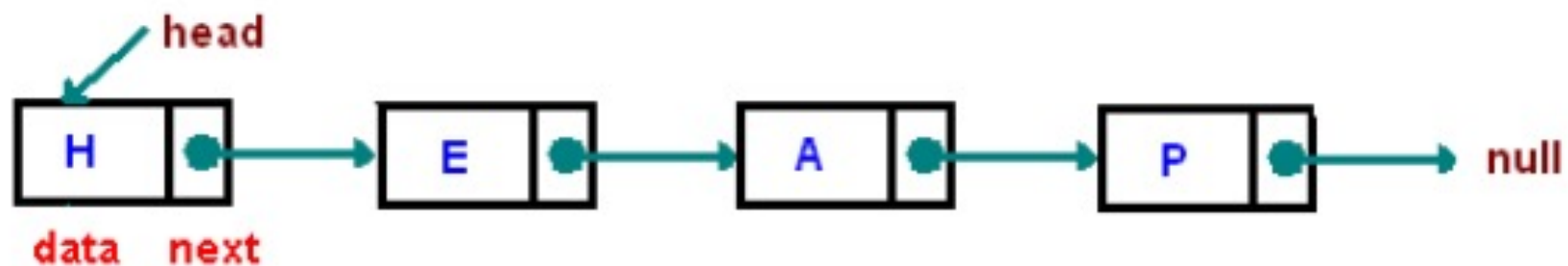
 **Link atau pointer**

 **data**



Konsep Single Linked Lists

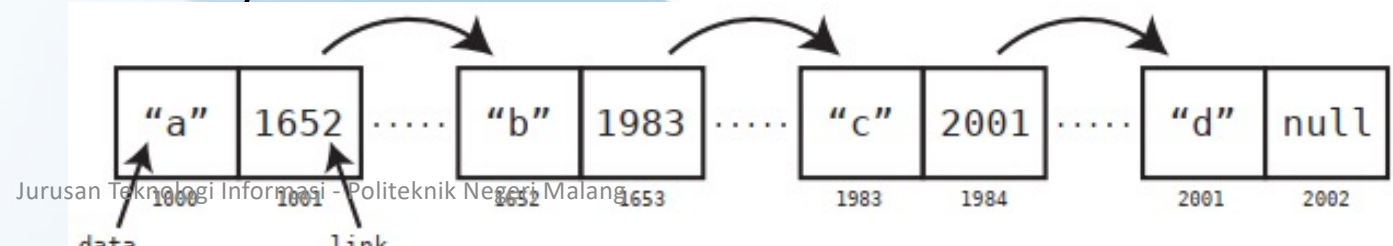
- *Linked List* merupakan struktur data dinamis.
- Jumlah *node* dapat bertambah sesuai dengan kebutuhan.
- Program yang tidak diketahui jumlah datanya, sebaiknya menggunakan struktur data *Linked List*.
- Gambar berikut menunjukkan ilustrasi *single linked lists*.





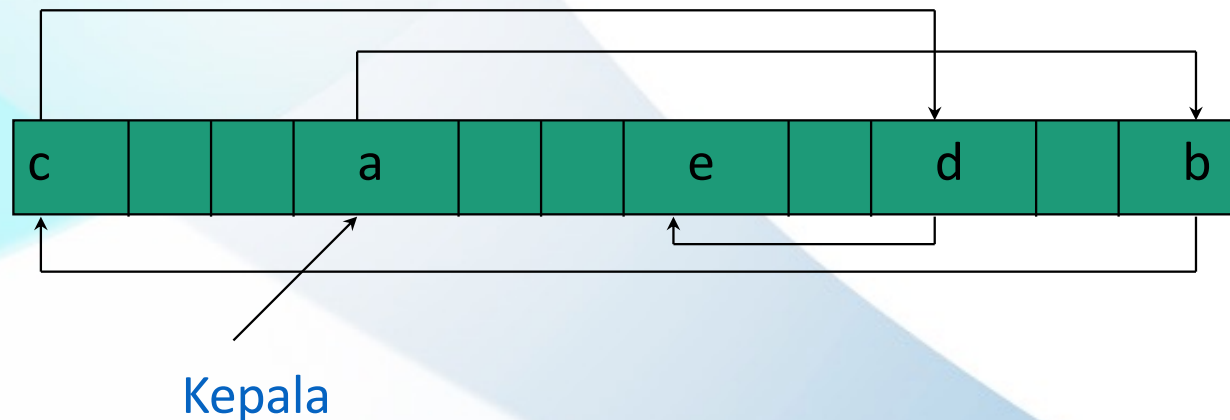
Konsep Single Linked Lists

- 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

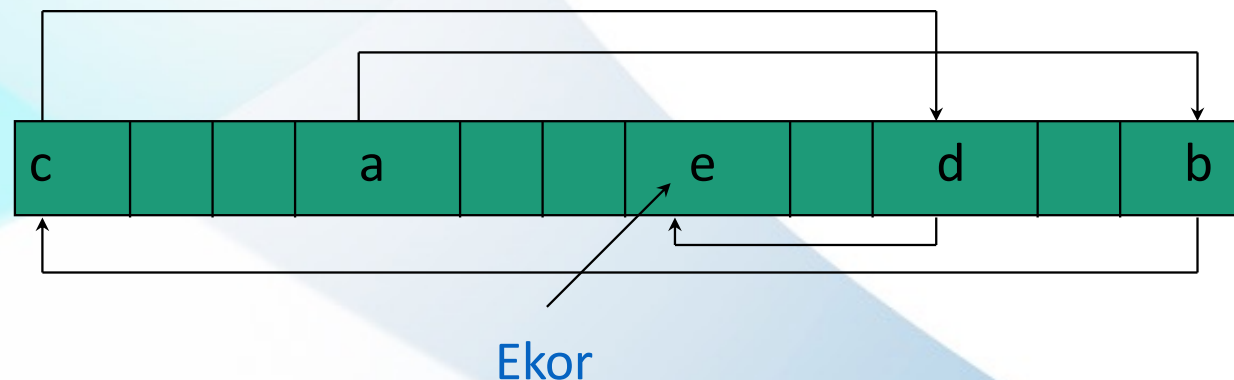
- Ilustrasi single linked list pada memory :



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

Ilustrasi Single Linked List

- Ilustrasi single linked list pada memory :



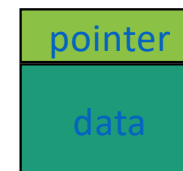
- Node e tidak menunjuk ke node manapun sehingga pointer dari node e adalah NULL. Dapat disimpulkan bahwa node ini adalah node yang paling belakang (node ekor).



“Single” Representation

```
class Node
{
    Object data;
    Node pointer;
}
```

Ilustrasi :



Penjelasan:

- Pembuatan class bernama Node yang berisi 2 field/variabel, yaitu data bertipe Object dan pointer yang bertipe class Node.
- Field data : digunakan untuk menyimpan data/nilai pada linked list.
Field pointer : digunakan untuk menyimpan alamat node berikutnya.



Implementasi Linked Lists (Node)

- Untuk merepresentasikan elemen data, diperlukan Node. Implementasi dalam bahasa Java sebagai berikut:

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

- Terdapat dua atribut utama pada node yaitu data dan penunjuk next yang menghubungkan dengan data berikutnya.



Implementasi Linked Lists (Node) Menggunakan Tipe Data Generic

- Untuk penyimpanan tipe data yang lebih fleksibel, dapat menggunakan konsep generic. Perhatikan contoh kode program berikut ini.

```
/**
 * Implementasi Node dengan Tipe Data Generic
 * @author Habibie Ed Dien
 * @param <T>
 */
public class Node<T> {

    T data;
    Node<T> next;

    public Node(T data, Node<T> next) {
        this.data = data;
        this.next = next;
    }

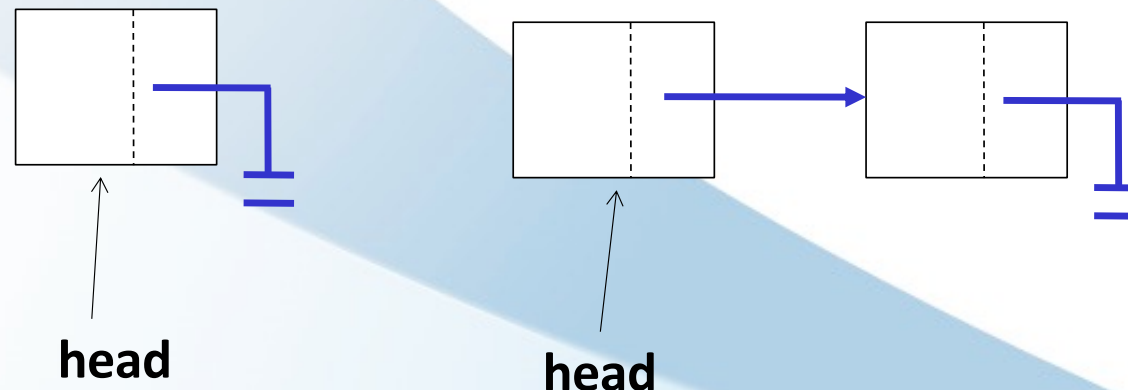
}
```

- Node bisa menerima berbagai tipe data: Integer, Float, String, Boolean



Pointer Head

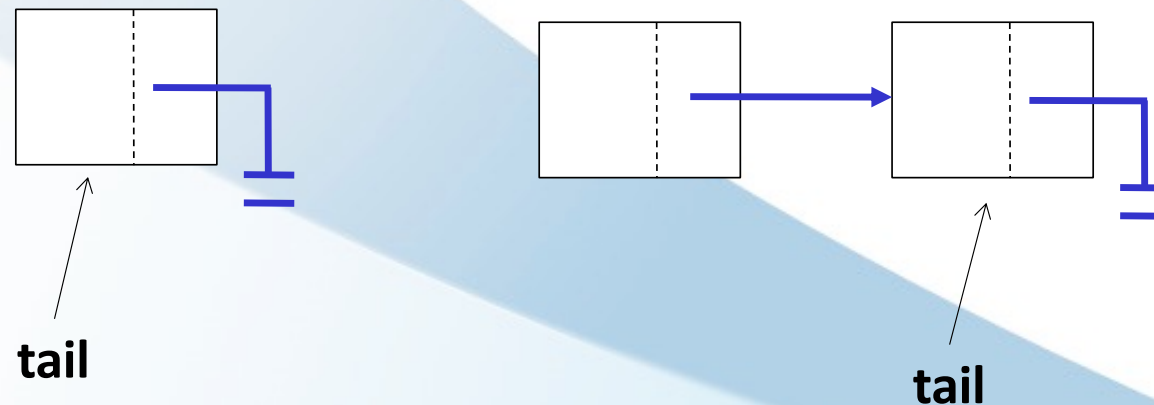
- Untuk mengingat node yg paling depan (node kepala) digunakan sebuah pointer yang akan menyimpan alamat dari node depan.
- Pointer ini biasanya diberi nama head.





Pointer Tail

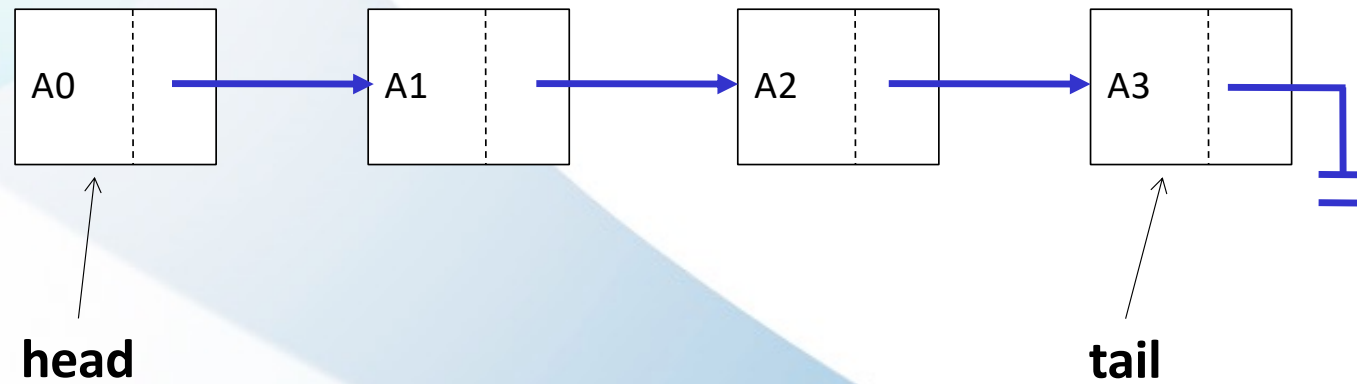
- Untuk mengingat node yg paling belakang (node ekor) digunakan sebuah pointer yang akan menyimpan alamat dari node belakang.
- Pointer ini biasanya diberi nama tail.





Contoh

- Linked list yang memiliki 4 node :



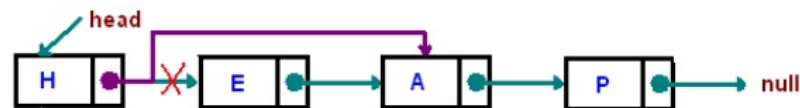


Cara Kerja Linked Lists

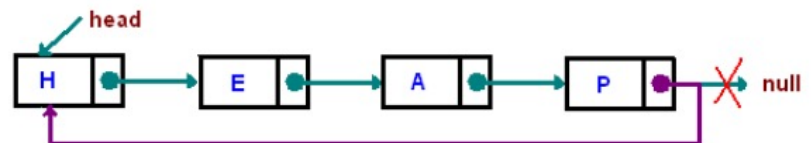
```
head = head.next;
```



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



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





Operasi pada Linked Lists

- **isEmpty**: mengecek apakah head = null (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
 - Di lokasi tertentu
- Operasi Linked List dengan Index
 - Pengaksesan data node
 - Pengaksesan index node
 - Penambahan data
 - Penghapusan data



Operasi isEmpty()

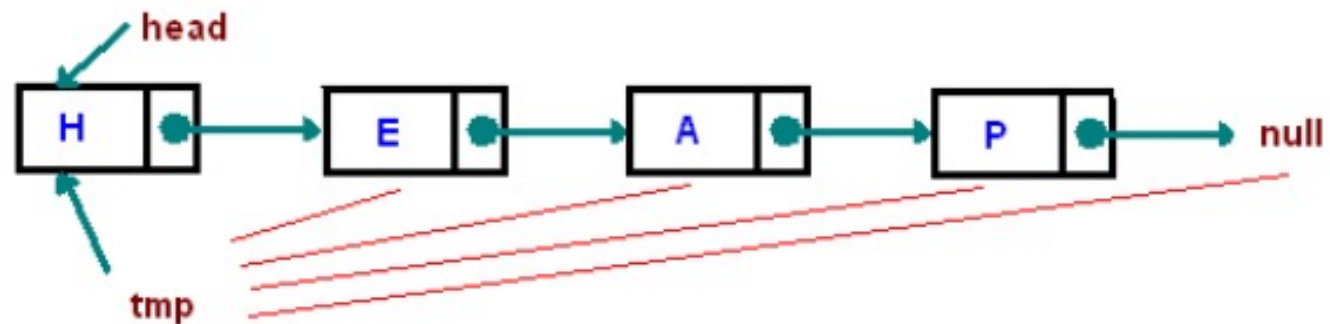
- Digunakan untuk mengetahui linked dalam kondisi kosong.
- Kondisi kosong : jika head=tail=null.

```
boolean isEmpty()  
{  
    return size==0;  
}
```




Proses Traverse pada Linked List

- Proses melakukan kunjungan pada setiap node tepat satu kali. Dengan melakukan kunjungan secara lengkap, maka akan didapatkan urutan informasi secara linier yang tersimpan dalam Linked List.
- Proses ini dilakukan pada operasi cetak data, penambahan data di akhir Linked Lists dan pengaksesan Linked List menggunakan index
- Proses ini dimulai dari awal data (head) sampai menjumpai null. Proses ini tidak merubah referensi dari head.





Operasi print()

- Untuk mencetak data seluruh node mulai dari yang paling depan(head) hingga tail

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



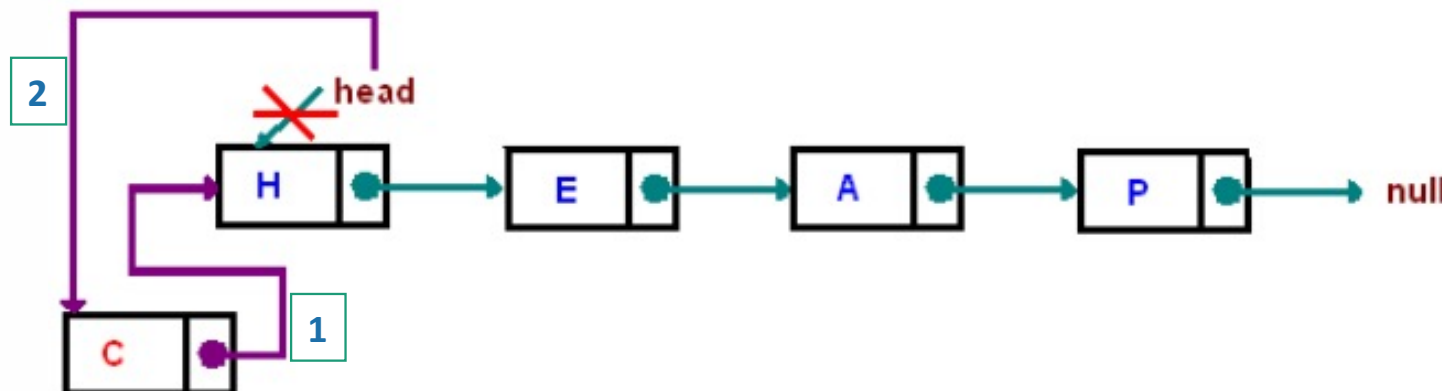
Operasi Penambahan

- Penambahan dari depan (AddFirst)
- Penambahan di belakang (AddLast)
- Penambahan setelah key tertentu (InsertAfter)



Penambahan dari Depan(addFirst)

- Jika kondisi awal node kosong maka head dan tail akan sama-sama menunjuk ke node input.
- Jika pada linked list telah ada node, maka:
 - Next pada node input menunjuk node yang ditunjuk oleh head 1
 - Kemudian head akan menunjuk ke node input 2

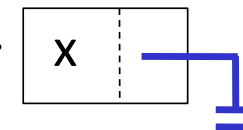




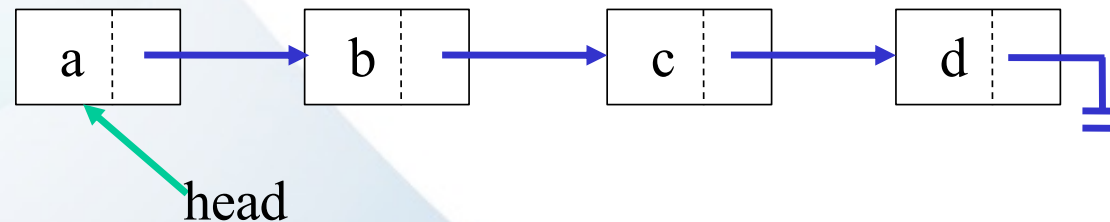
Ilustrasi : addFirst(x)

- Menambahkan **x** pada lokasi paling depan.

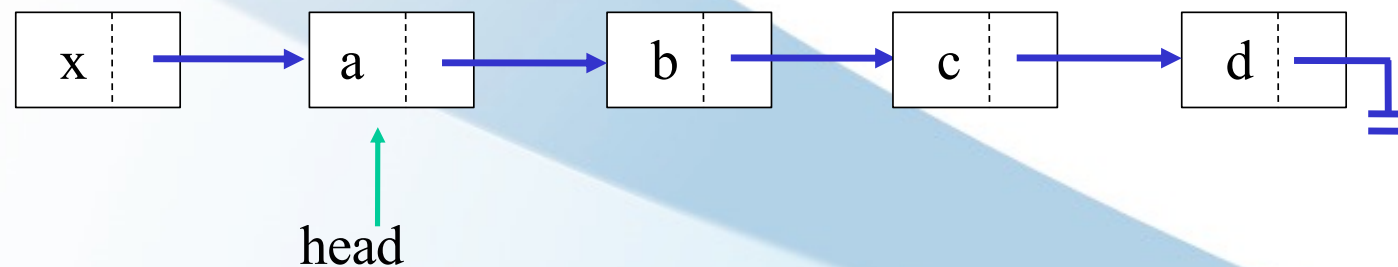
Node input



Kondisi awal pada linked list :



Setelah penambahan node x didepan:





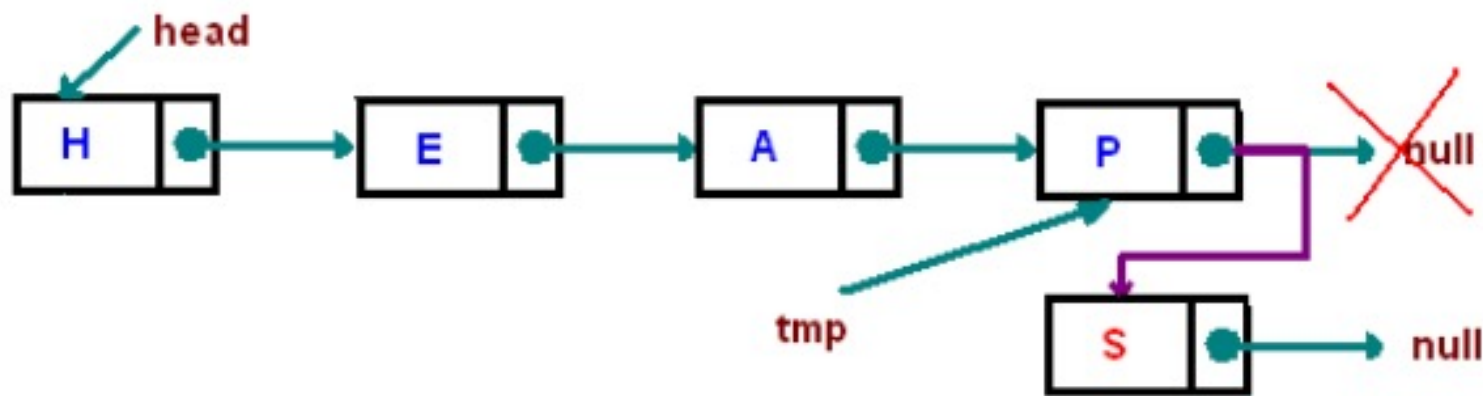
Operasi Linked Lists: AddFirst

```
void addFirst(Node input) {  
    if (isEmpty()) {  
        head=input;  
        tail=input;  
    }  
    else  
    {  
        input.next = head;  
        head = input;  
    }  
}
```



Penambahan dari Belakang (addLast)

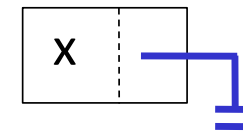
- Operasi untuk menambah node di akhir elemen Linked Lists.
- Jika kondisi awal node kosong maka head dan tail akan sama-sama menunjuk ke node input.
- Jika pada linked list telah ada node, maka:
 - Next pada node yang ditunjuk oleh tail menunjuk ke node input
 - kemudian tail akan menunjuk ke node input



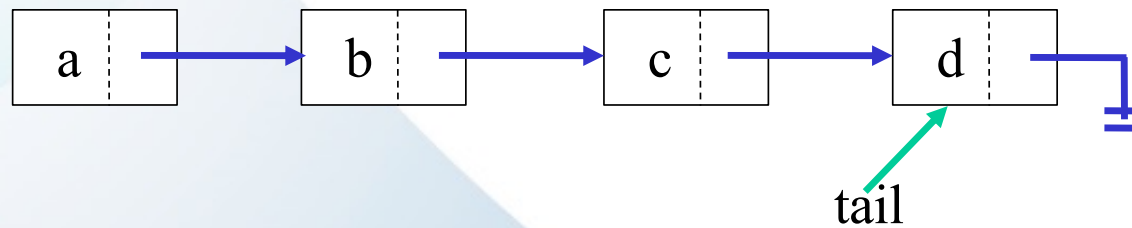
Ilustrasi : addLast(x)

- menambahkan X pada akhir list :

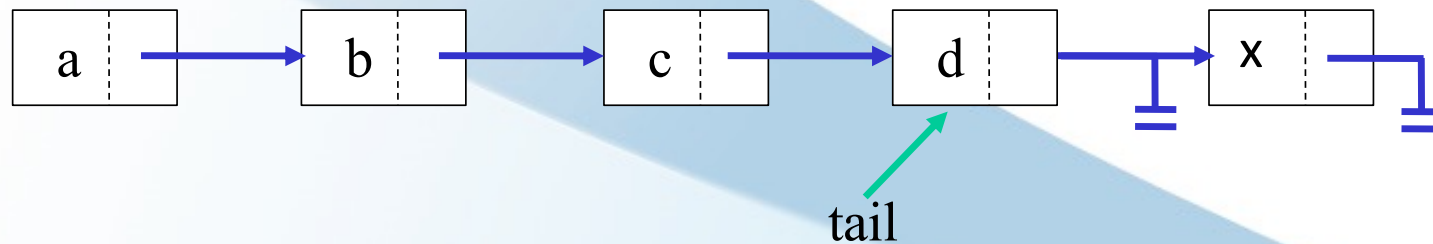
Node input



Kondisi awal pada linked list :



Setelah penambahan node x dibelakang :





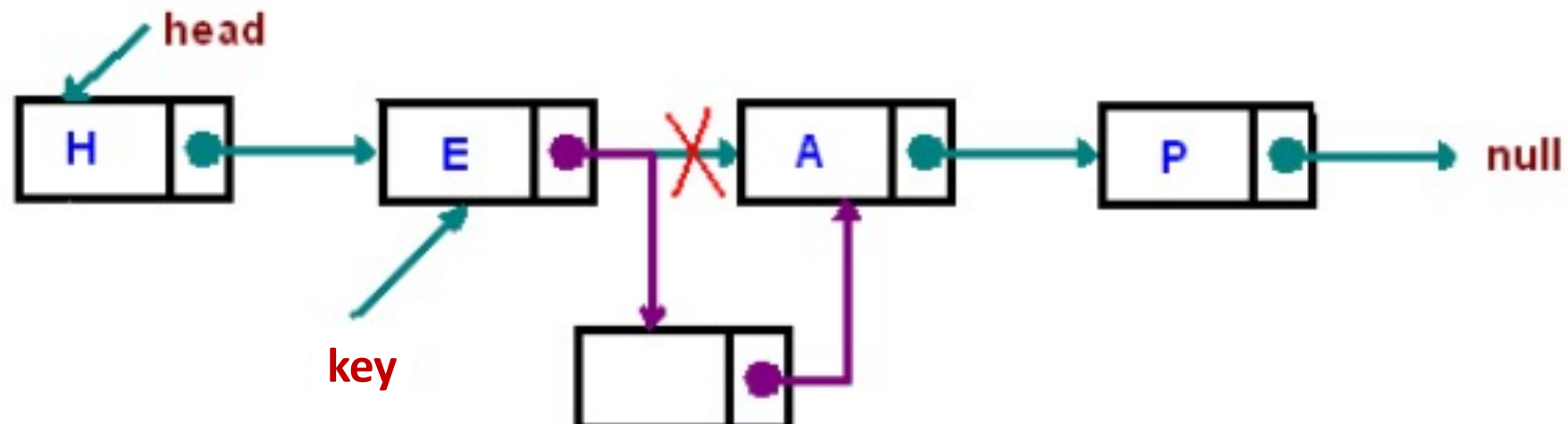
Operasi Linked Lists: addLast

```
void addLast(Node input) {  
    if (isEmpty()) {  
        head = input;  
        tail = input;  
    }  
    else  
    {  
        tail.pointer = input;  
        tail = input;  
    }  
}
```



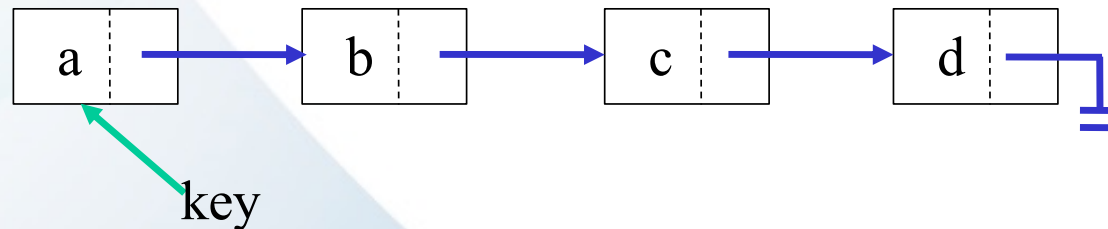
Penambahan setelah Node tertentu(insertAfter)

- Dilakukan pencarian node yang memiliki data yang sama dengan key.
- Kemudian dilakukan penambahan node setelah node yang memiliki data sama dengan key
- Contoh di bawah ini merupakan penambahan setelah key E :
insertAfter(E)

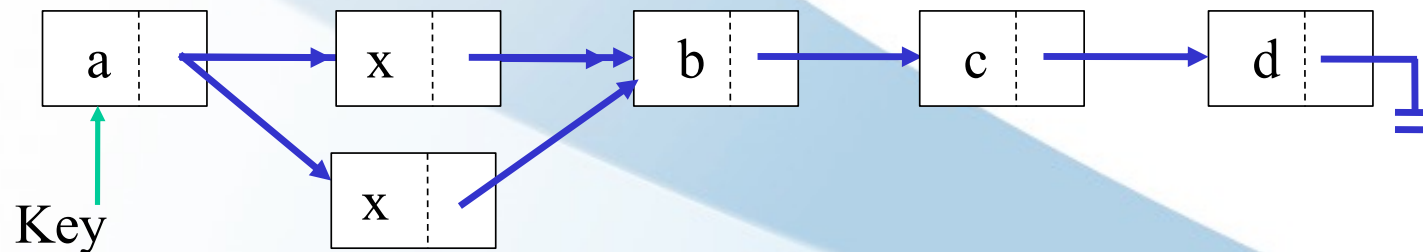


Ilustrasi : Insert After(a)

- Kondisi Awal



- Menyisipkan **X** pada lokasi setelah **key**.





Operasi Linked Lists: insertAfter

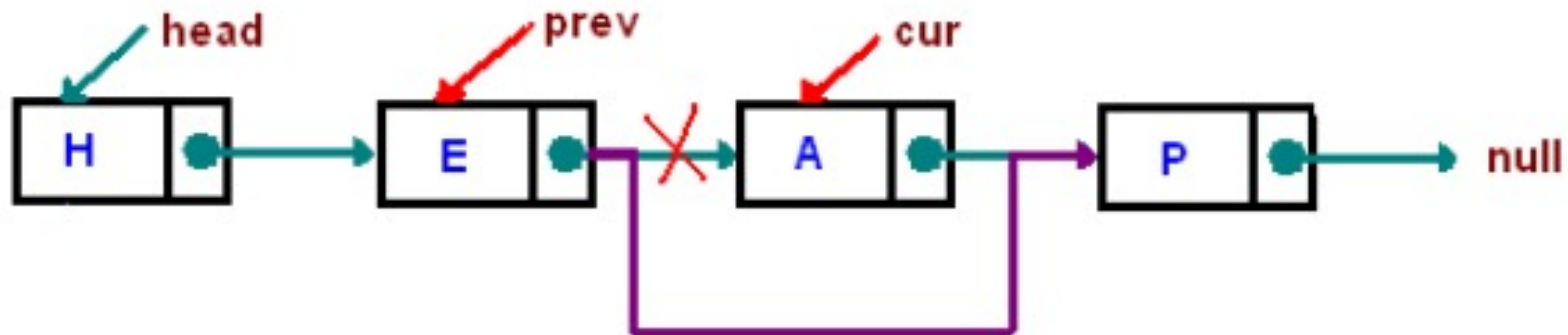
```
public void insertAfter(Object key, Node input) {  
    Node temp = head;  
    do {  
        if (temp.data == key) {  
            input.next = temp.next;  
            temp.next = input;  
            System.out.println("Insert data is succeed.");  
            break;  
        }  
        temp = temp.next;  
    } while (temp != null);  
}
```



Operasi Penghapusan

Dibedakan menjadi :

- Hapus node depan (removeFirst)
- Hapus node belakang (removeLast)
- Hapus node tertentu (remove)

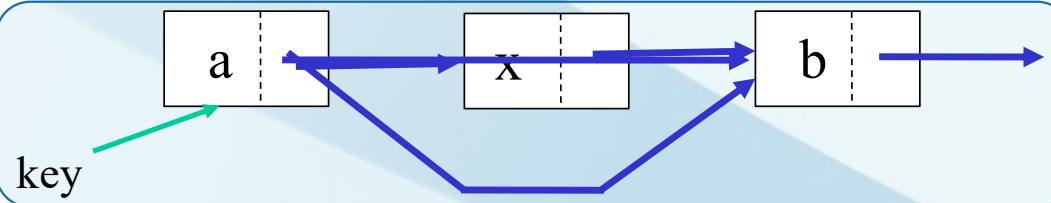
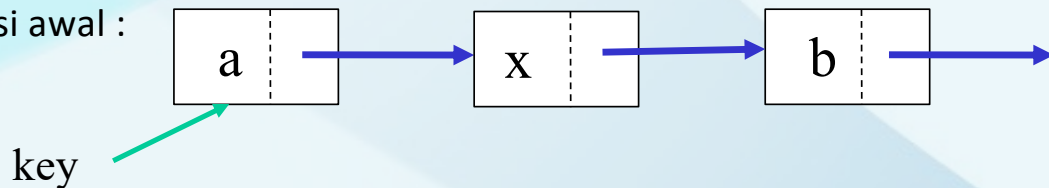




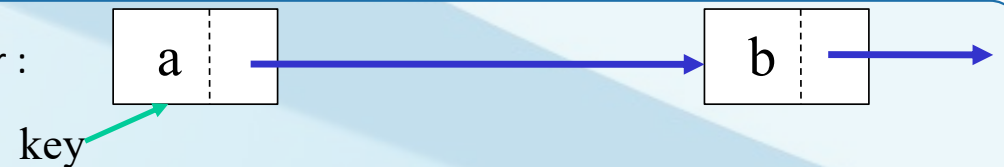
Linked Lists: menghapus elemen X

- Proses menghapus dilakukan dengan mengabaikan elemen yang hendak dihapus dengan cara melewati pointer (reference) dari elemen tersebut langsung pada elemen selanjutnya.
- Elemen x dihapus dengan meng-assign field next pada elemen a dengan alamat b

Kondisi awal :



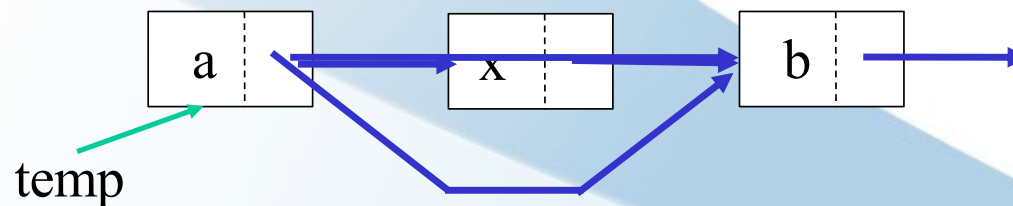
Hasil akhir :





Langkah-langkah menghapus elemen

- Tidak ada elemen lain yang menyimpan alamat node x.
- Node x tidak bisa diakses lagi.
- Java Garbage Collector akan membersihkan alokasi memory yang tidak dipakai lagi atau tidak bisa diakses.
- Dengan kata lain, menghapus node x.





Operasi Linked Lists: Remove/Delete

Ketika proses penghapusan perlu diperhatikan juga beberapa kondisi tambahan dari data Linked Lists, yaitu:

- Linked List kosong, maka tidak dapat dilakukan penghapusan;
- Penghapusan node yang merupakan head, maka head harus menunjuk ke node berikutnya;
- Node yang ingin dihapus harus ada dalam Linked List.



Hapus node depan

```
public void removeFirst(){  
    if(isEmpty()) System.out.println("Linked List masih Kosong!");  
    else if(head==tail){  
        head = tail = null;  
    }  
    else{  
        head = head.next;  
    }  
}
```



Hapus node belakang

```
public void removeLast() {  
    if(isEmpty()) System.out.println("Linked List masih Kosong!");  
    else if(head==tail){  
        head = tail = null;  
    }  
    else{  
        Node current = head;  
        while(current.next != tail){  
            current = current.next;  
        }  
        current.next = null;  
        tail=current;  
    }  
}
```



Hapus node tertentu

```
public void remove(int key){
    if(isEmpty()) System.out.println("Linked List masih Kosong, tidak dapat dihapus!");
    else{
        Node temp = head;
        while (temp != null){
            if ((temp.data == key)&&(temp == head)){
                this.removeFirst();
                break;
            }
            else if (temp.next.data == key){
                temp.next = temp.next.next;
                if(temp.next == null)
                    tail=temp;
                break;
            }
            temp = temp.next;
        }
    }
}
```



Operasi Linked List dengan Index

- Pengaksesan data node
- Pengaksesan index node
- Penambahan data
- Penghapusan data



Pengaksesan data node: getData(int index)

```
public int getData(int index)
{
    Node tmp = head;
    for (int i = 0; i < index; i++)
        tmp = tmp.next;
    return tmp.data;
}
```



Operasi pencarian indeks data pada node : indexOf(key)

```
public int indexOf(int key) {  
    Node tmp = head;  
    int index = 0;  
    while (tmp != null && tmp.data != key) {  
        tmp = tmp.next;  
        index++;  
    }  
  
    if (tmp == null) {  
        return -1;  
    } else {  
        return index;  
    }  
}
```



Method remove(int index)

```
public void removeAt(int index) {  
    if (index == 0) {  
        removeFirst();  
    } else {  
        Node temp = head;  
        for (int i = 0; i < index - 1; i++) {  
            temp = temp.next;  
        }  
        temp.next = temp.next.next;  
        if (temp.next == null) {  
            tail = temp;  
        }  
    }  
}
```



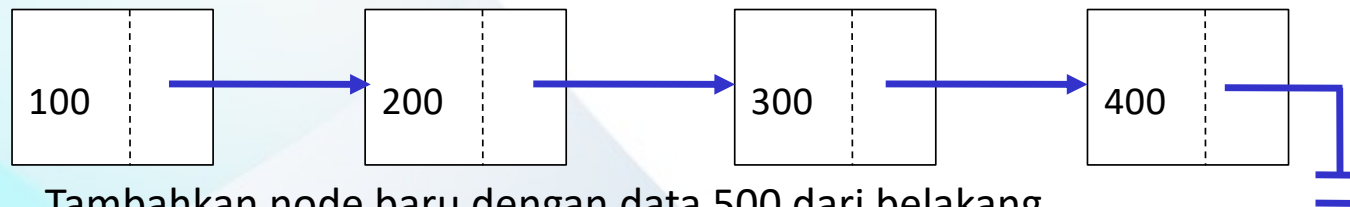
Method add(int index, Object theElement)

```
public void insertAt(int index, int input) {  
    if (index < 0) {  
        System.out.println("indeks salah");  
    } else if (index == 0)  
    {  
        addFirst(input);  
    } else {  
        Node temp = head;  
        for (int i = 0; i < index - 1; i++) {  
            temp = temp.next;  
        }  
        temp.next = new Node(input, temp.next);  
    }  
}
```




Latihan

Jelaskan Langkah-langkah dari 4 node berikut dengan kondisi awal linked list kosong:



1. Tambahkan node baru dengan data 500 dari belakang.
2. Tambahkan node baru dengan data 50 dari depan.
3. Tambahkan node dengan data 250 setelah node 200.
4. Tambahkan node dengan data 150 pada indeks ke-1
5. Hapus node depan
6. hapus node belakang
7. hapus node yg memiliki data 300.
8. Hapus node pada indeks ke-3

*Tampilkan semua data dari seluruh node pada linked list untuk setiap penambahan/penghapusan



Terima Kasih 😊