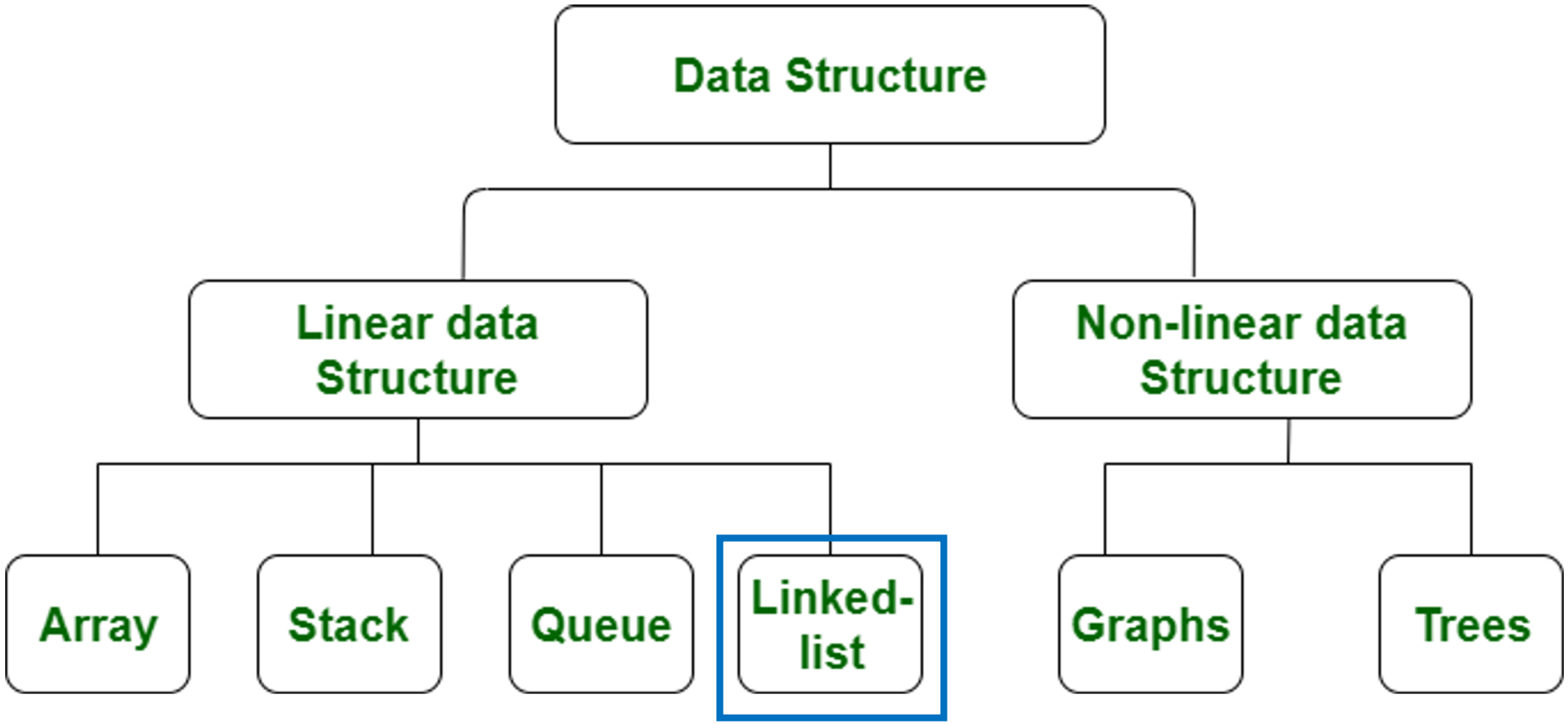
DOUBLE LINKED LIST*

Tim Ajar Algoritma dan Struktur Data
Genap 2024/2025

Jenis Struktur Data



Capaian Pembelajaran

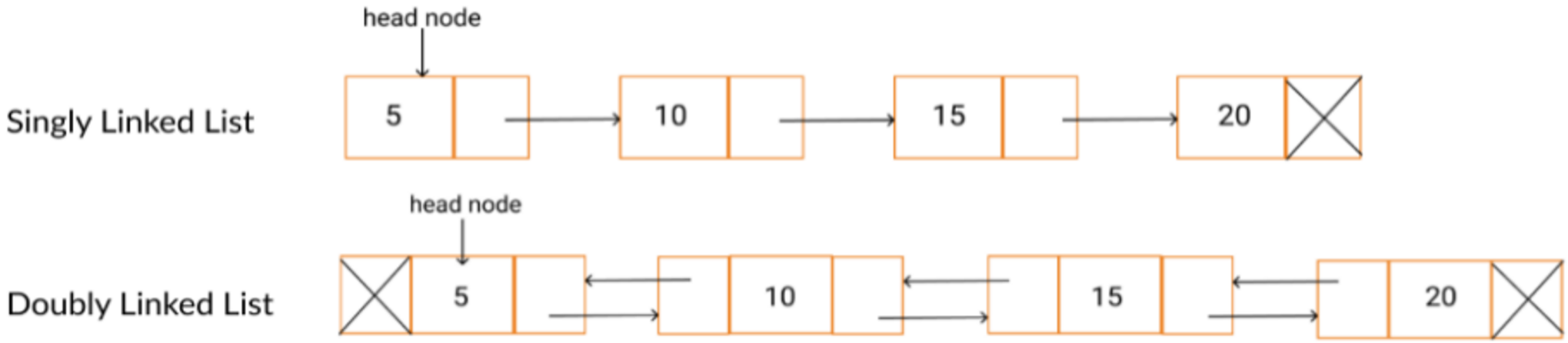
- Mahasiswa memahami konsep double linked list
- Mahasiswa memahami tahapan pembuatan double linked list untuk menyelesaikan masalah

Double Linked List

- Double Linked Lists memiliki dua buah pointer yaitu pointer **next** dan **prev**.
- Pointer next menunjuk pada node setelahnya dan pointer prev menunjuk pada node sebelumnya.
- Double memiliki arti field pointer-nya dua buah dan dua arah, ke node sebelum dan sesudahnya.
- Sedangkan Linked List artinya adalah node-node yang saling terhubung satu sama lain.

Ilustrasi Double Linked List

- Berbeda dengan single linked lists yang hanya memiliki satu pointer next saja.
- Double linked lists memiliki **dua pointer** (prev dan next) pada setiap Node.
- Data pada awal linked lists memiliki prev sama dengan null, sedangkan data akhir memiliki next sama dengan null.



Implementasi class Node

Untuk merepresentasikan elemen data, diperlukan Node pada double link list.
Class Node memiliki 3 atribut:

- data: menyimpan nilai
- prev: menyimpan node sebelumnya
- next: menyimpan node berikutnya

```
public class Node01 {
    Mahasiswa01 data; //bisa berupa tipe data primitive atau tipe data kelas
    Node00 prev;
    Node00 next;

    public Node00(Mahasiswa01 data) {
        this.data = data;
        this.prev = null;
        this.next = null;
    }
}
```

Implementasi class DoubleLinkedList

- Untuk melakukan operasi pada double link list dibuat class double link list. Implementasi atribut dan konstruktor class double link list dalam bahasa Java sebagai berikut:

```
public class DoubleLinkedList01 {
    Node01 head;
    Node01 tail;

    public DoubleLinkedList01() {
        head = null;
        tail = null;
    }
}
```

- Head → node pertama
- Tail → node terakhir

Operasi pada Double Linked Lists

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

Fungsi isEmpty()

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

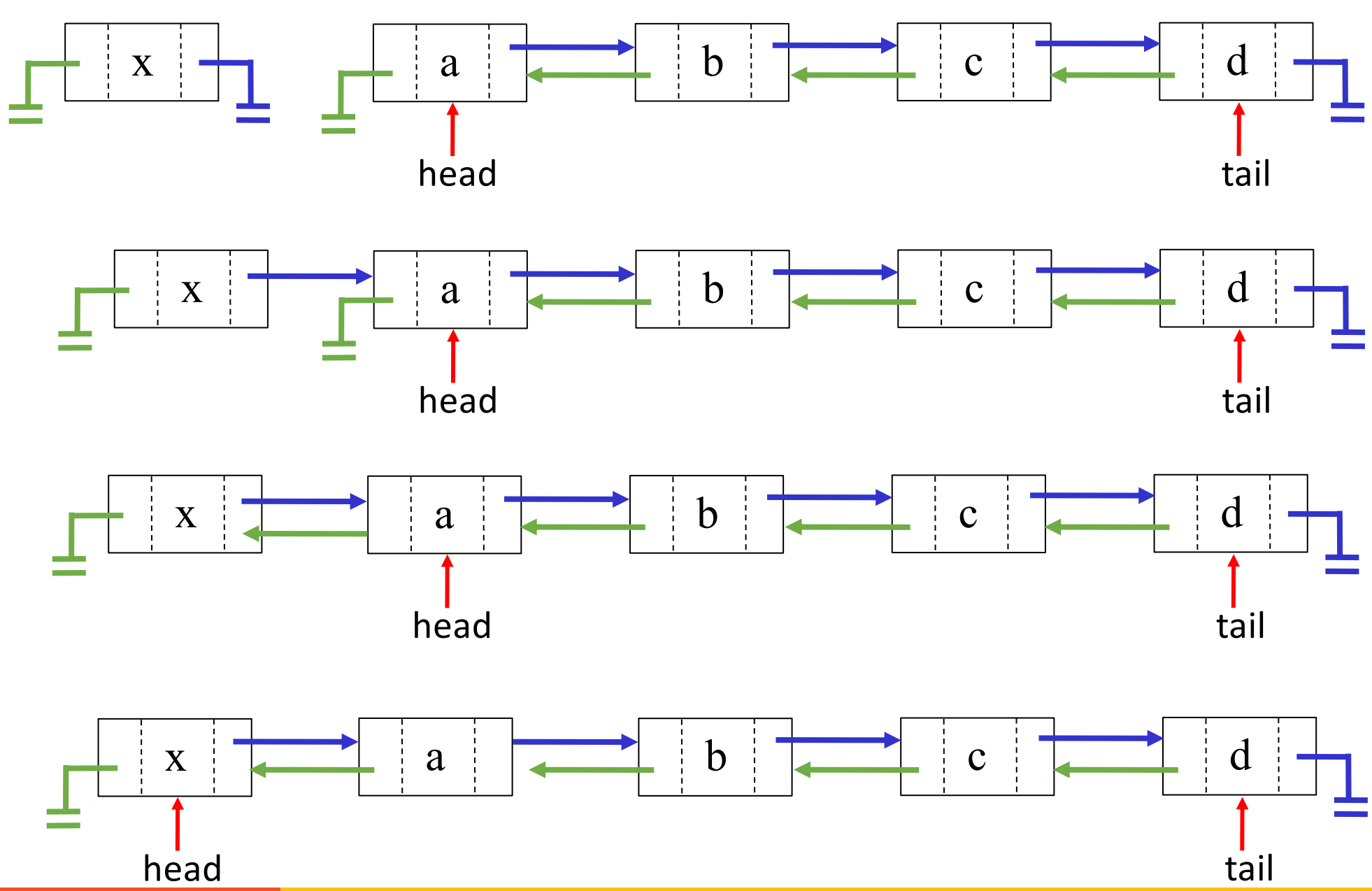
```
public Boolean isEmpty() {
    return (head == null);
}
```

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 double linked list
- Jika double linked list kosong maka node input akan dijadikan sebagai head sekaligus tail
- Jika tidak kosong, maka:
  - Atribut next pada node input akan menunjuk head node
  - Atribut prev pada head node akan menunjuk ke node input
  - Node input akan dijadikan sebagai head yang baru

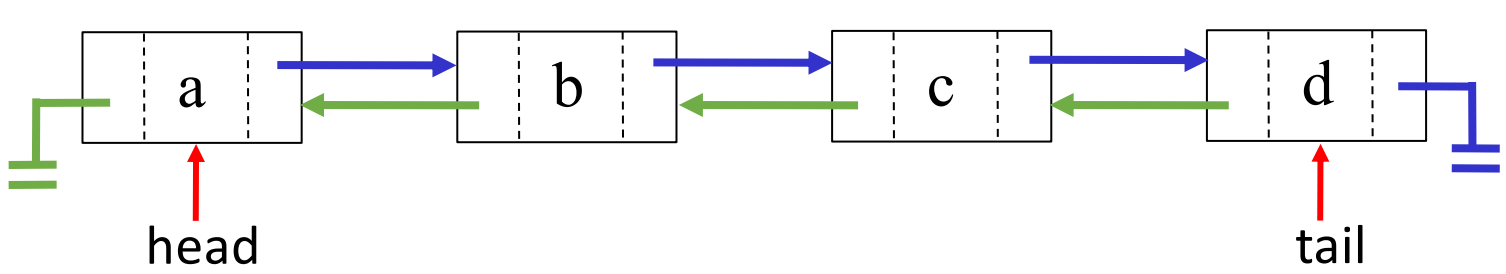

x
a
b
c
d
head
tail
x
a
b
c
d
head
tail
x
a
b
c
d
head
tail
x
a
b
c
d
head
tail

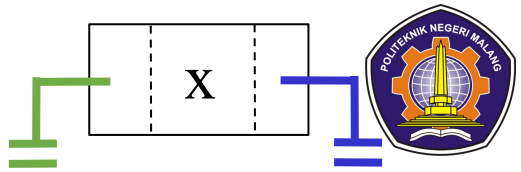
Fungsi addFirst()

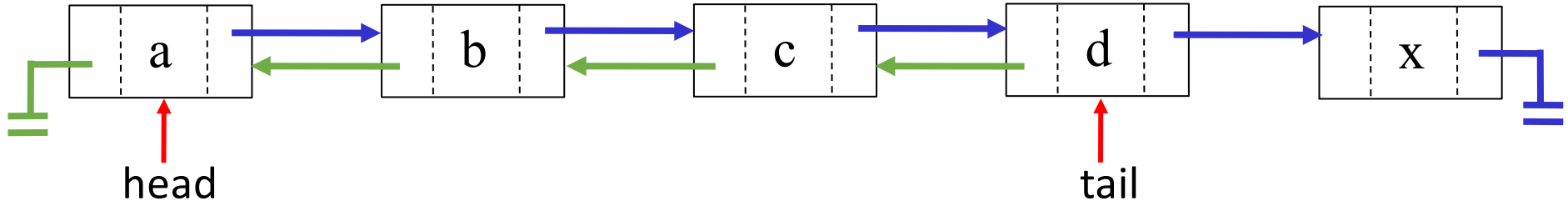
```
public void addFirst(Mahasiswa01 data) {
        Node01 newNode = new Node01(data);
        if (isEmpty()) {
            head = tail = newNode;
        } else {
            newNode.next = head;
            head.prev = newNode;
            head = newNode;
        }
    }
```

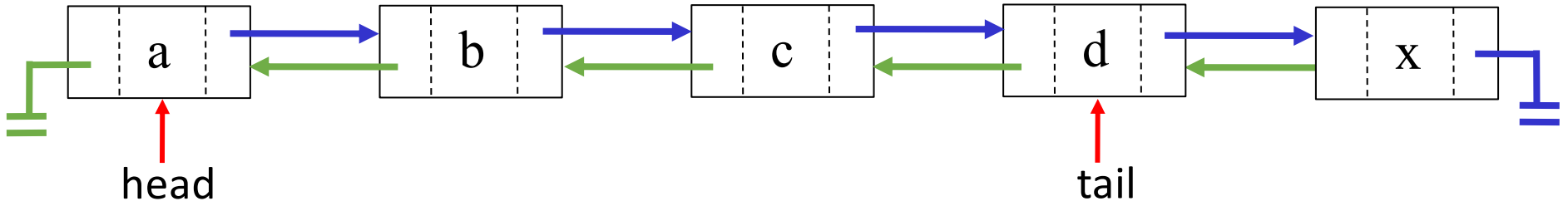
Fungsi addLast()

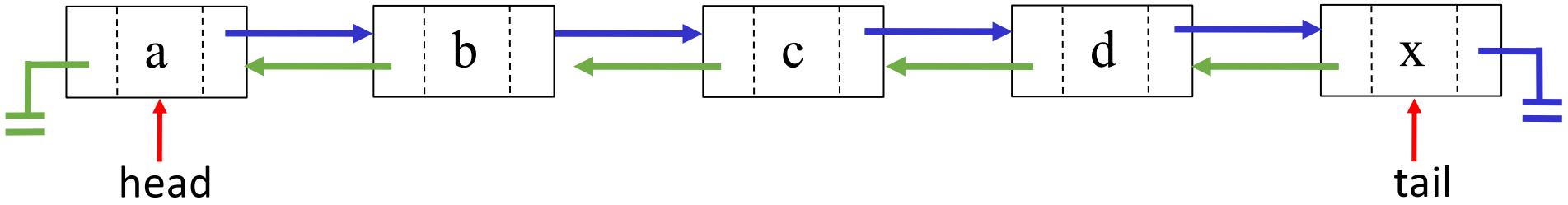
- Digunakan untuk menambahkan node baru di akhir double linked lists.
- Jika double linked list kosong maka node baru akan dijadikan sebagai head sekaligus tail
- Jika linked list tidak kosong, maka:
  - Atribut next pada tail node akan menunjuk ke node input
  - Atribut prev pada node input akan menunjuk tail node
  - Node input akan dijadikan sebagai tail yang baru


a
b
c
d
head
tail


x


a
b
c
d
x
head
tail


a
b
c
d
x
head
tail


a
b
c
d
x
head
tail

Fungsi addLast()

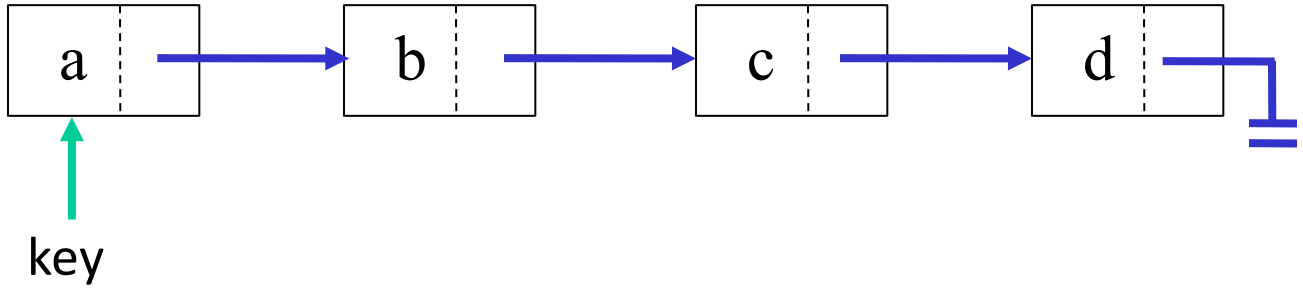
```
public void addLast(Mahasiswa01 data) {
      Node00 newNode = new Node01(data);
      if (isEmpty()) {
         head = tail = newNode;
      } else {
         tail.next = newNode;
         newNode.prev = tail;
         tail = newNode;
      }
   }
```

Fungsi insertAfter()

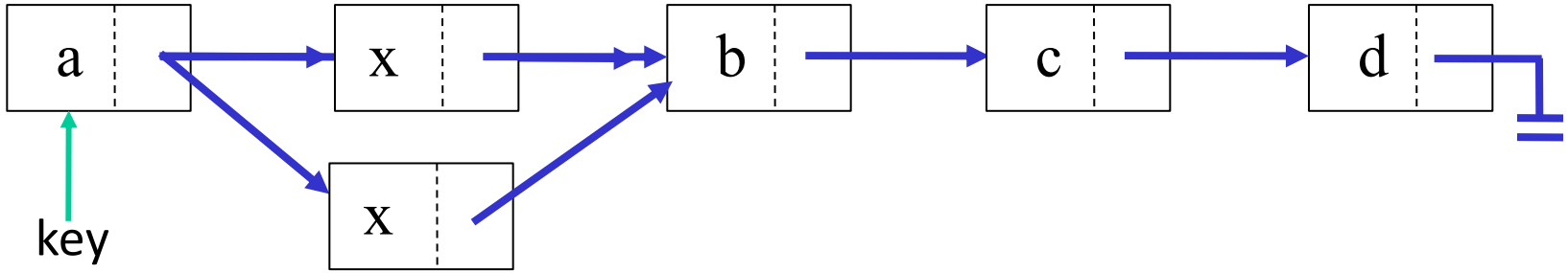
- Digunakan untuk menambahkan node baru setelah node tertentu yang memiliki data key
- Yang pertama dilakukan adalah proses pencarian node yang memiliki data yang sama dengan key
- Kemudian dilakukan penambahan node setelah node yang memiliki data yang sama dengan key ketemu

insertAfter() Linked List

- Kondisi Awal



Menyisipkan x pada lokasi setelah *key*.



Fungsi insertAfter()

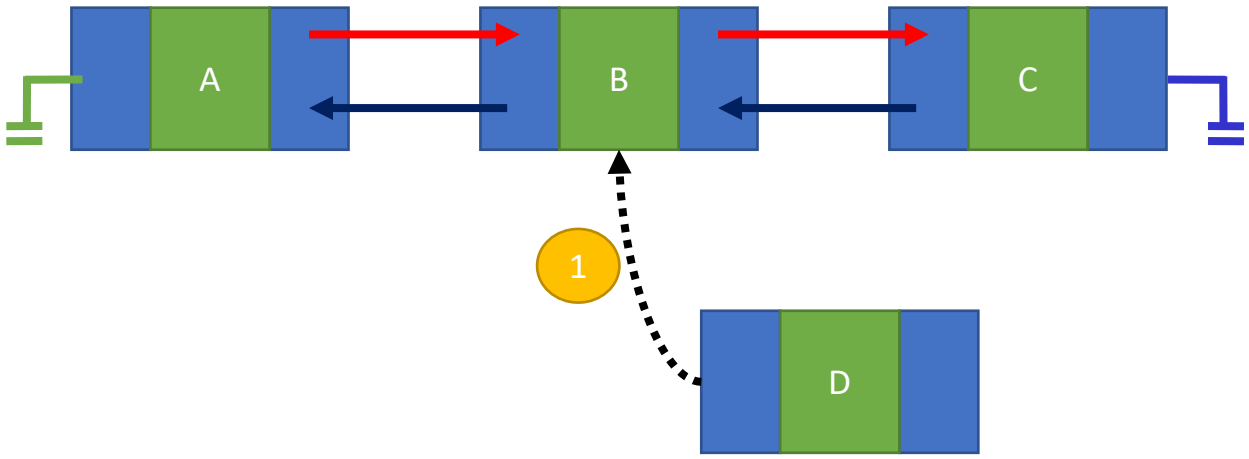
```
public void insertAfter(String keyNim, Mahasiswa00 data) {
    Node00 current = head;

//missal pencarian berdasarkan NIM
while (current != null && !current.data.nim.equals(keyNim)) {
        current = current.next;
    }

    if (current == null) {
        System.out.println("Node dengan NIM " + keyNim + " tidak ditemukan.");
        return;
    }

    Node00 newNode = new Node00(data);

    // Jika current adalah tail, panggil fungsi addLast()
    if (current == tail) {
        addlast(data);
} else {
        // Sisipkan di tengah
        newNode.next = current.next;
        newNode.prev = current;
        current.next.prev = newNode;
        current.next = newNode;
    }

    System.out.println("Node berhasil disisipkan setelah NIM " + keyNim);
}
```

Fungsi add()

- Operasi untuk menambah node berdasarkan indeks.
- Indeks yang akan ditambah dapat disisipkan di awal ataupun di akhir dari double linked lists.
- Terdapat **empat langkah utama** dalam penambahan data yaitu:
  - memposisikan location prev indeks data yang akan dimasukkan sebagai Node baru bagian prev,
  - location terletak pada posisi New Node bagian next,
  - Node baru terletak pada location bagian prev.next, dan
  - Node baru terletak pada bagian location bagian prev.

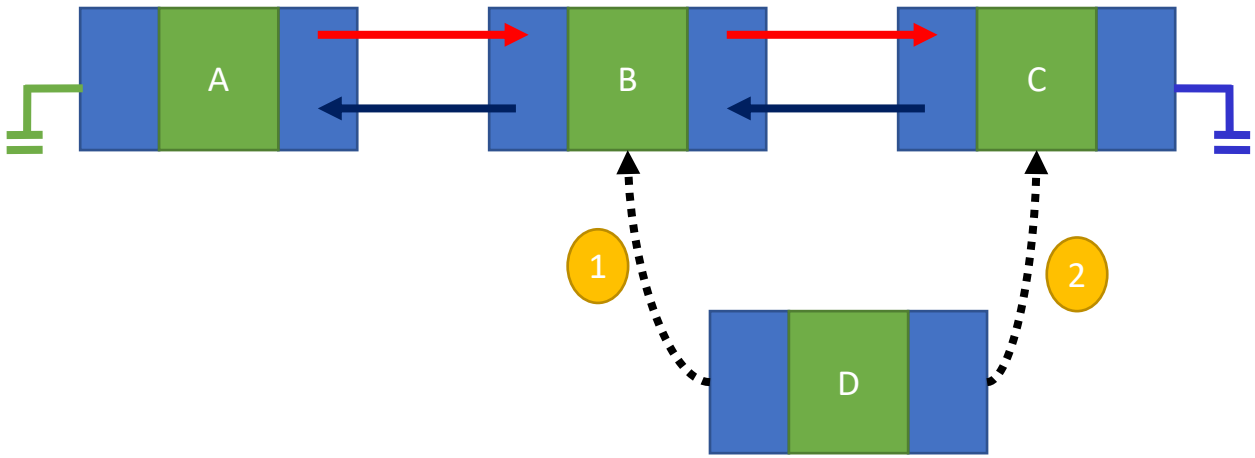
Fungsi add()

1. memposisikan location prev indeks data yang akan dimasukkan sebagai Node baru bagian prev,



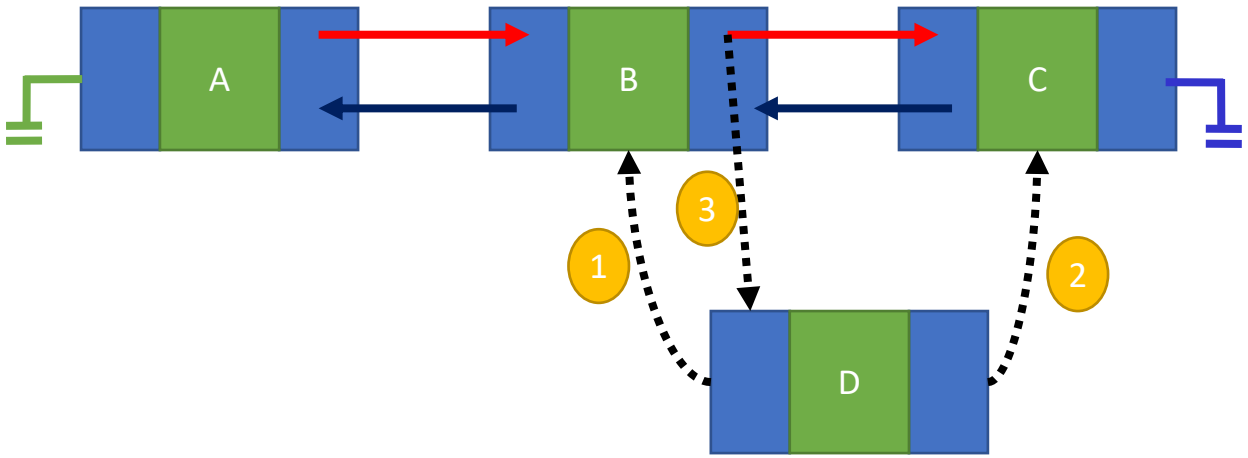
Fungsi add()

1. memposisikan location prev indeks data yang akan dimasukkan sebagai Node baru bagian prev,
2. location terletak pada posisi New Node bagian next,



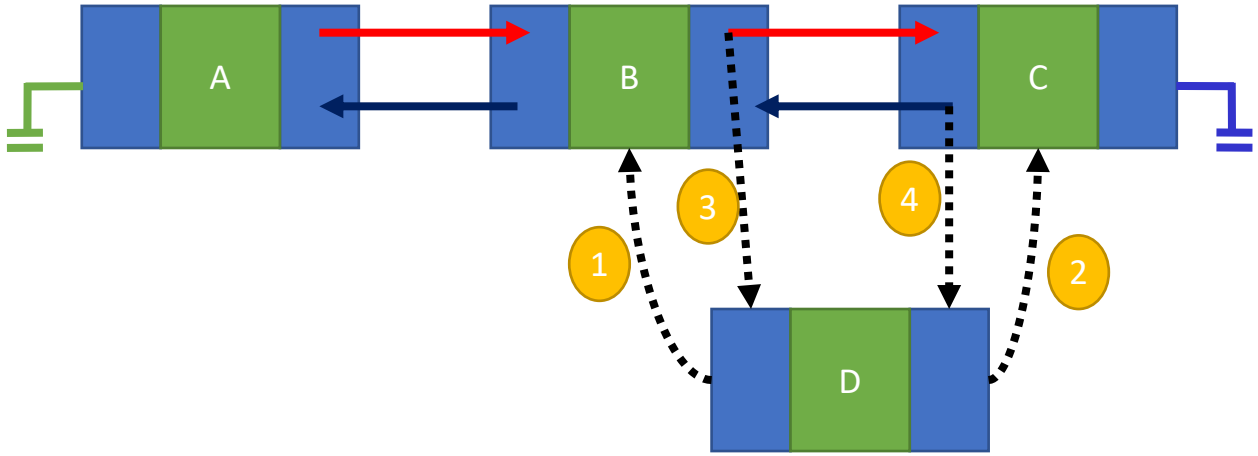
Fungsi add()

1. memposisikan location prev indeks data yang akan dimasukkan sebagai Node baru bagian prev,
2. location terletak pada posisi New Node bagian next,
3. Node baru terletak pada location bagian prev.next,

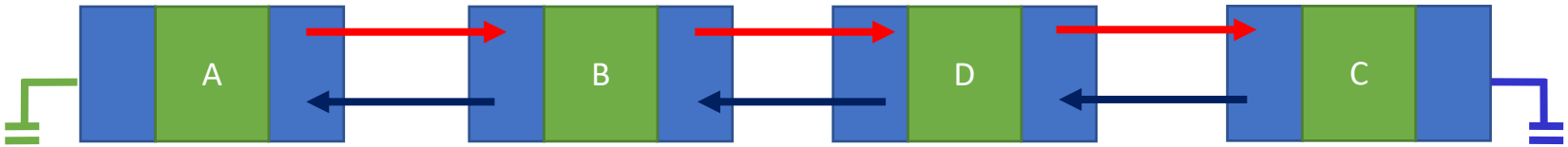


Fungsi add()

1. memposisikan location prev indeks data yang akan dimasukkan sebagai Node baru bagian prev,
2. location terletak pada posisi New Node bagian next,
3. Node baru terletak pada location bagian prev.next, dan
4. Node baru terletak pada bagian location bagian prev.



Fungsi add()



Operasi Penghapusan

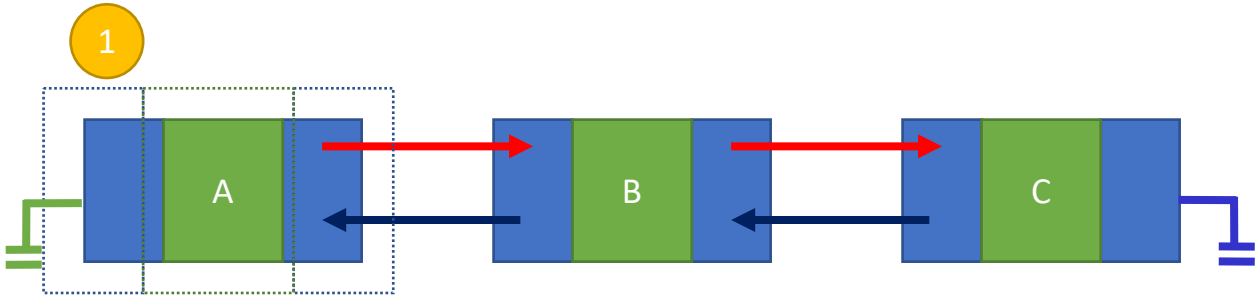
- removeFirst(): menghapus node pertama
- removeLast(): menghapus node terakhir
- remove(): menghapus node tertentu

Langkah-langkah menghapus elemen

- Proses menghapus dilakukan dengan mengabaikan elemen yang hendak dihapus
- 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.

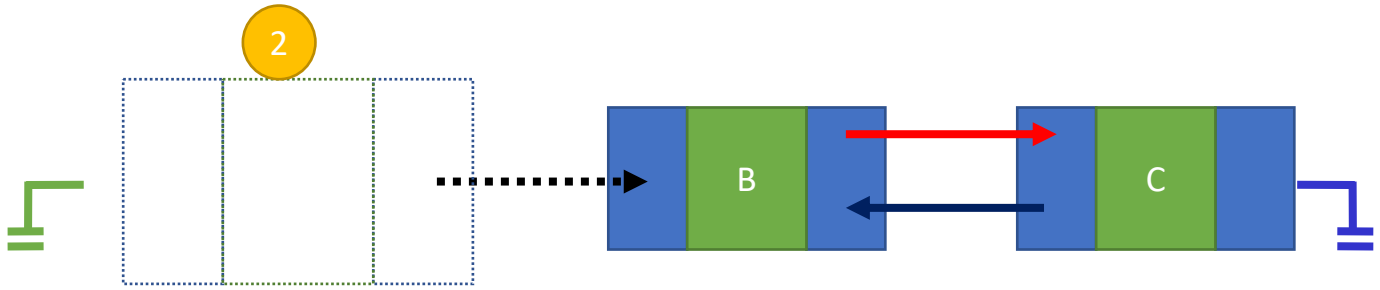
Fungsi removeFirst()

1. Menghapus data pada bagian awal dilakukan dengan **pencarian lokasi awal double linked list,**

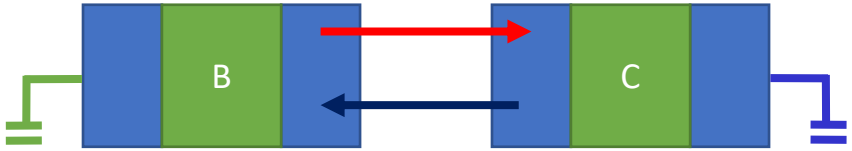


Fungsi removeFirst()

1. Menghapus data pada bagian awal dilakukan dengan pencarian lokasi awal double linked list,
2. kemudian melakukan removing dengan cara **menjadikan data pada bagian next menjadi bagian head**.



Fungsi removeFirst()



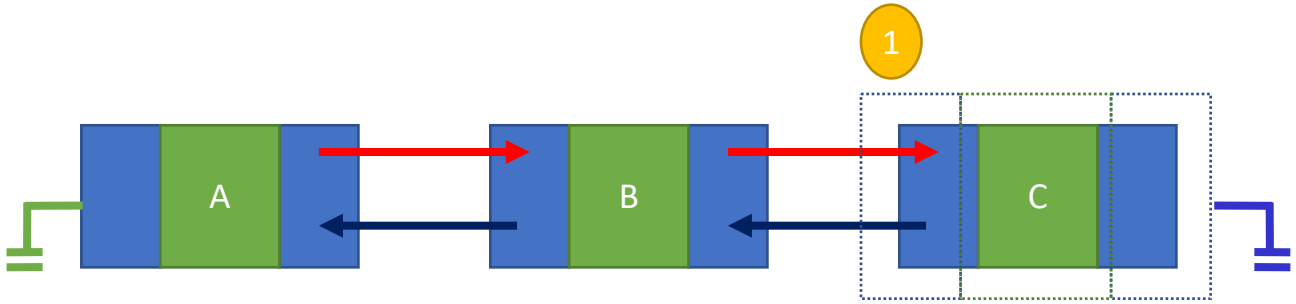

```
public void removeFirst() {
      if (isEmpty()) {
         System.out.println("List kosong, tidak bisa dihapus.");
         return;
      }
      if (head == tail) {
         head = tail = null;
      } else {
         head = head.next;
         head.prev = null;
      }
   }
```

Fungsi removeLast()

- Menghapus node pada bagian akhir Double Linked List
- Jika linked list kosong, tampilkan warning
- Jika linked list berisi 1 node saja, kosongkan dengan mengubah atribut head dan tail menjadi null
- Jika linked list berisi lebih dari 1 node:
  - Jadikan prev node dari tail sebagai tail yang baru
  - Set next dari tail menjadi null

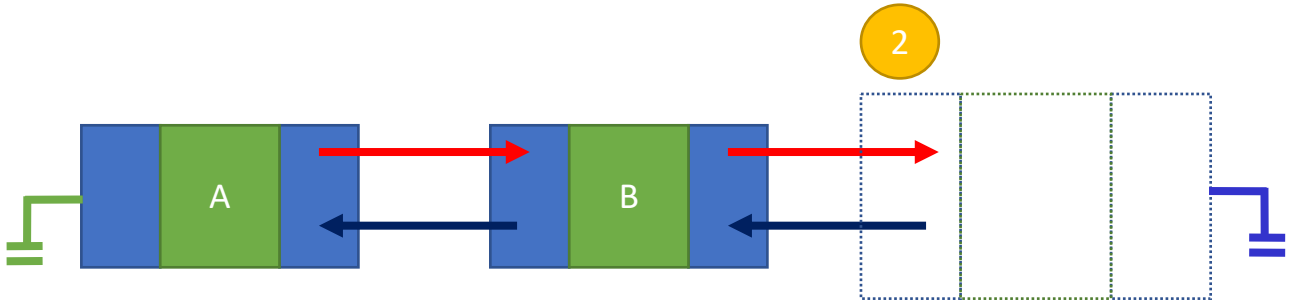
Fungsi removeLast()

1. Menghapus data pada bagian akhir elemen diawali **dengan memastikan posisi yang diinginkan berada di bagian akhir**,

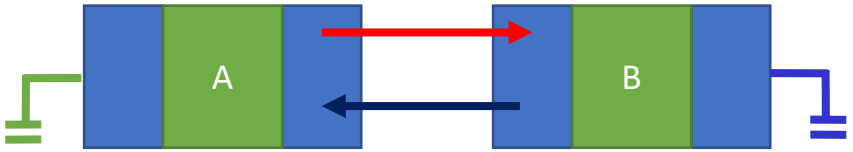


Fungsi removeLast()

1. Menghapus data pada bagian akhir elemen diawali dengan memastikan posisi yang diinginkan berada di bagian akhir,
2. kemudian menghapus item yang berada di posisi tersebut.



Fungsi removeLast()




```
public void removeLast() {
      if (isEmpty()) {
         System.out.println("List kosong, tidak bisa dihapus.");
         return;
      }
      if (head == tail) {
         head = tail = null;
      } else {
         tail = tail.prev;
         tail.next = null;
      }
   }
```

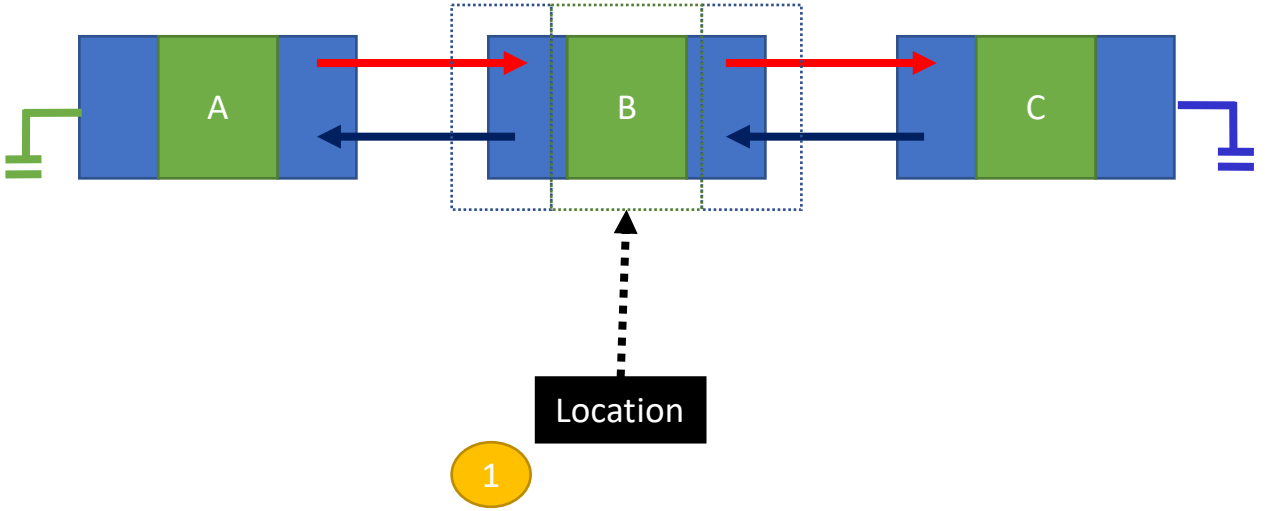
Fungsi remove()

- Operasi untuk menghapus node **berdasarkan indeks**.
- Indeks yang akan dihapus dapat disisipkan di awal ataupun di akhir dari double linked list.
- Perintah ini akan merubah posisi Node pada bagian next menjadi Node pada bagian next next.

Fungsi remove()

Operasi untuk menghapus node berdasarkan indeks.

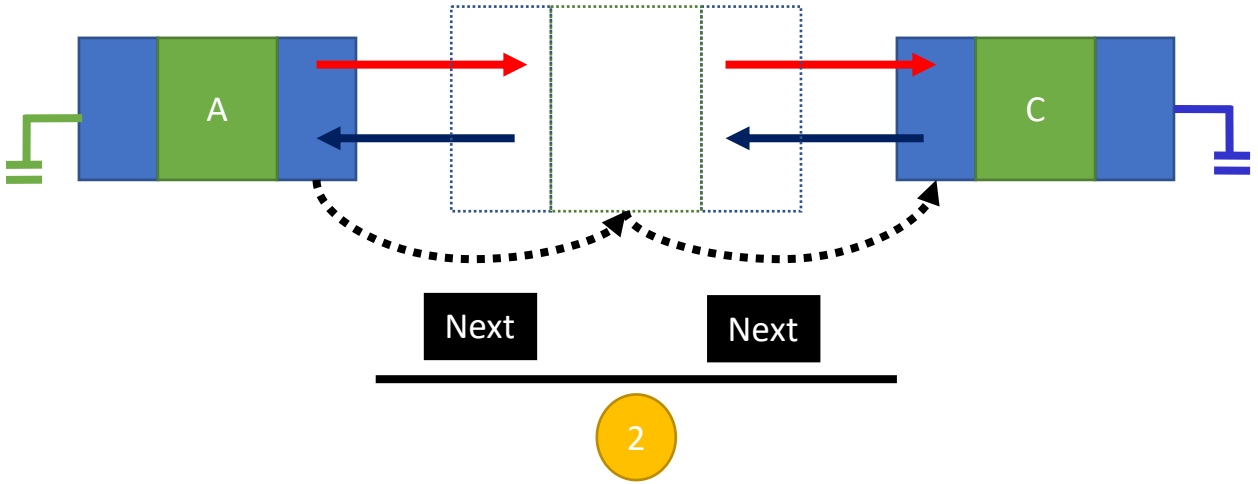
1. Indeks yang akan dihapus dapat disisipkan di awal ataupun di akhir dari double linked list (location).



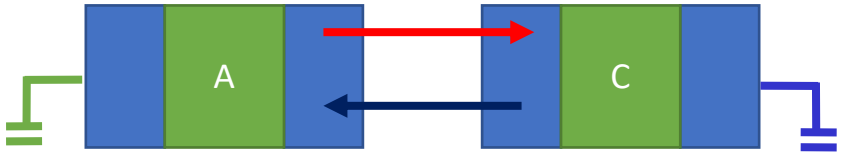
Fungsi remove()

Operasi untuk menghapus node berdasarkan indeks.

1. Indeks yang akan dihapus dapat disisipkan di awal ataupun di akhir dari double linked list (location).
2. Perintah ini akan **merubah posisi Node pada bagian next menjadi Node pada bagian next next.**



Fungsi remove()



Operasi Lain

- Mencetak seluruh data pada double linked list
- Menambahkan data setelah key tertentu
- Menghapus data dengan key tertentu
- Pengaksesan data node: mengetahui data yang terletak pada indeks tertentu (missal node awal atau node akhir
- 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

Fungsi print()

- Print digunakan untuk mencetak isi double linked list

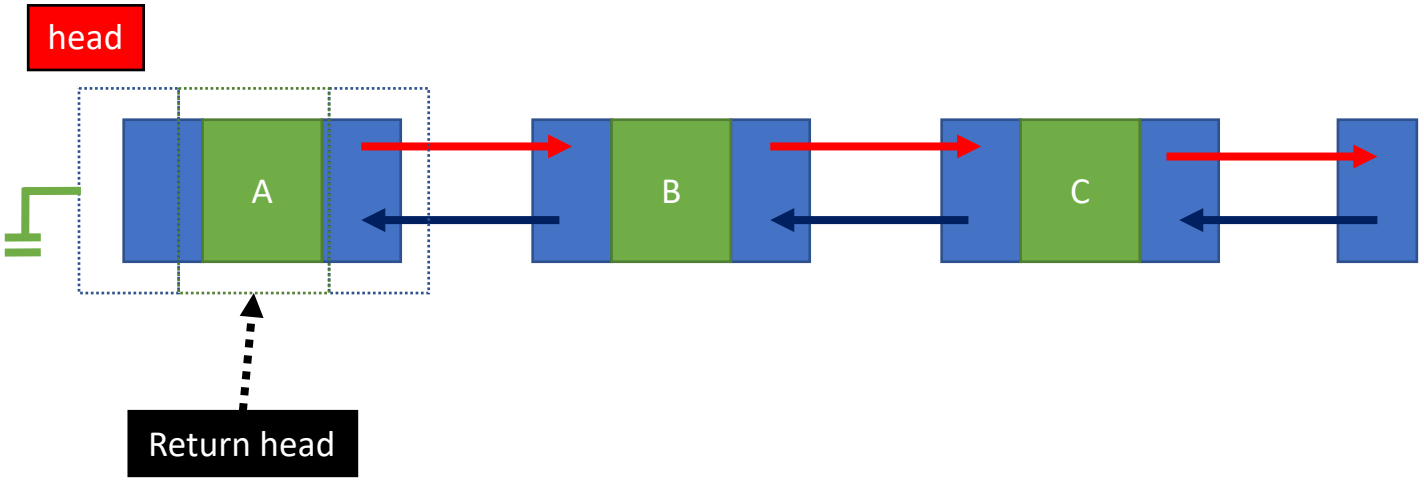
```
public void print() {
        Node00 current = head;
        while (current != null) {
            current.data.tampil();
            current = current.next;
        }
    }
```

Operasi Double Linked Lists: getFirst / getLast

- Fungsi **getFirst** digunakan untuk **mengambil data di elemen paling depan (head)**.
- Prosedur **getFirst** pada double linked lists adalah dengan cara **mengembalikan nilai data pada head** untuk dapat ditampilkan.
- Berbeda dengan pengambilan data pada akhir elemen (**getLast**) digunakan untuk mengambil data pada double linked lists yang mana posisi data adalah **pada indeks terakhir atau data paling belakang**.

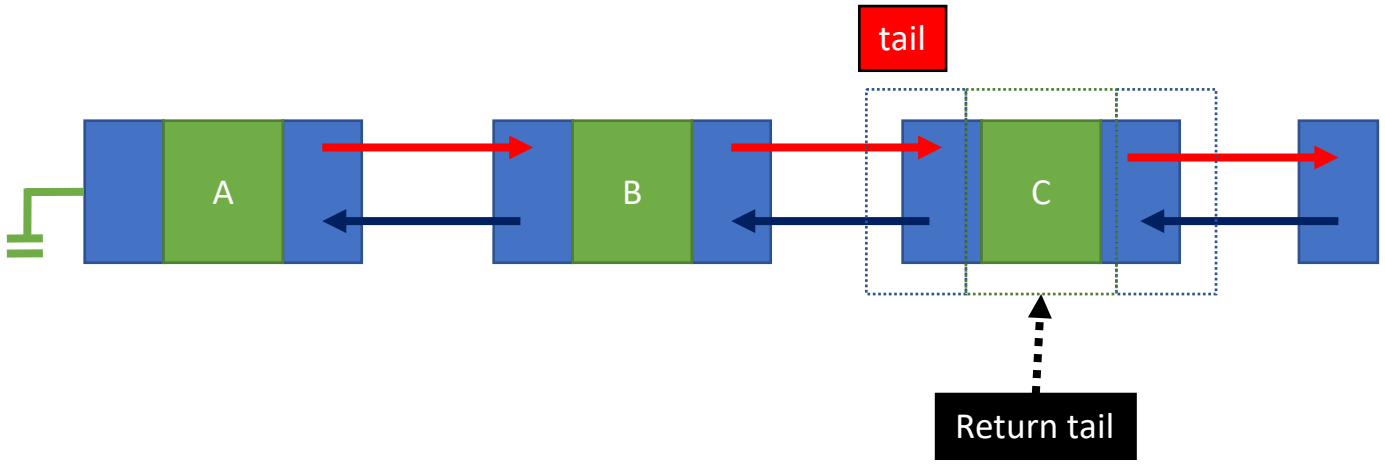
Operasi Double Linked Lists: getFirst / getLast

- Fungsi **getFirst** digunakan untuk mengambil data di elemen **paling depan (head).**
- Prosedur getFirst pada double linked lists adalah **dengan cara mengembalikan nilai data pada head** untuk dapat ditampilkan.



Operasi Double Linked Lists: getFirst / getLast

Pengambilan data pada akhir elemen (getLast) digunakan untuk **mengambil data pada double linked lists yang mana posisi data adalah pada indeks terakhir atau data paling belakang**.



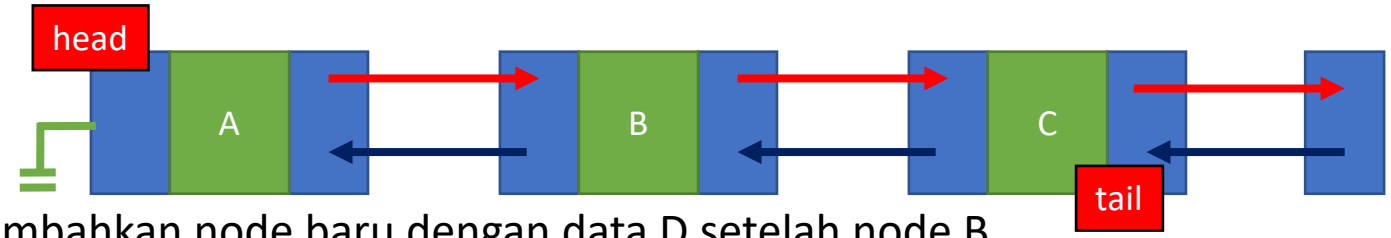
Operasi Double Linked Lists: get

- Fungsi get(index) digunakan jika ingin mengambil data yang dipilih pada indeks tertentu.
- Prosedur pengambilan data pada indeks tertentu adalah sebagai berikut:

```
public int get(int index) throws Exception {
    if (isEmpty() || index >= size) {
        throw new Exception("Nilai indeks di luar batas.");
    }
    Node tmp = head;
    for (int i = 0; i < index; i++) {
        tmp = tmp.next;
    }
    return tmp.data;
}
```

Latihan

Jelaskan Langkah-langkah dari 3 node berikut dengan kondisi awal double linked list kosong secara berkelanjutan!



1. Tambahkan node baru dengan data D setelah node B.
2. Tambahkan node baru dengan data E setelah node C.
3. Tambahkan node dengan data F setelah node D.
4. Tambahkan node dengan data G pada indeks ke-3.
5. Tambahkan node dengan data H pada posisi sebelum head (sebagai head baru).
6. Tambahkan node dengan data I pada posisi setelah head baru.
7. Hapus node depan
8. Hapus node belakang
9. Hapus node yg memiliki data A.
10. Hapus node pada indeks ke-5

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


thank you
danke
謝謝
ngiyabonga
شكراً جزيلاً
рахмат
teşekkür ederim
спасибо
gracias
tapadh leat
Баярлалаа
faafetai lava
dank je
vinaka
спасибі
blagodaram
misaotra
matondo
paldies
grazzi
mahalo
хвала
asante
manana
obrigada
tenki
murakoze
chokrane
mamnun
дякую
kiitos
dankie
nanni
nandri
dhanyavad
bedankt
mauruuru
hvala
köszönöm
gràcie
bayarlalaa
enkosi
akun
dankon
ačiū
djiere dieuf
tau
mochchakkeram
dziękuję
chnorakaloutioun
gratias ago
grācies
sulpáy
go raibh maith agat
taiku
sukriya
kop khun krap
sobodi
dēkuji
obrigado
sagolun
mèsi
didi madloba
kam sah hamnida
rahmat
najis tuke
terima kasih
ありがとう
tanemirt
rahmet
grazie
arigatô
takk
dakujem
trugarez
merce
shukriya
мерси
diolch
dhanyavadagalu
ευχαριστώ
xièxie
감사합니다
merci