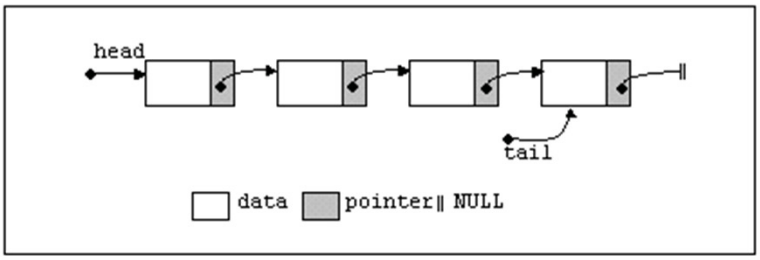
Single Linked List

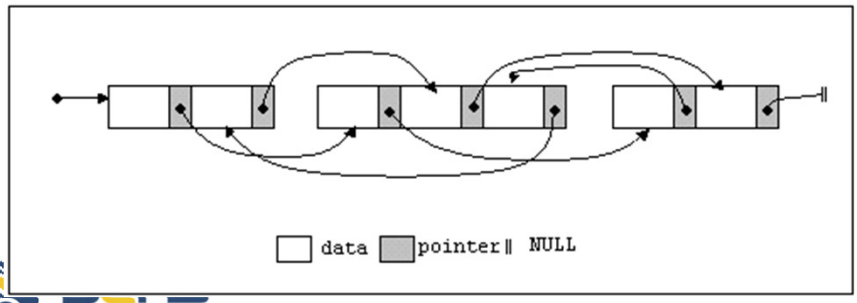
Single Linked List

- ***Single linked list*** **atau linked list**
- **Tiap elemen terdiri dari dua bagian, yaitu sebuah data dan sebuah pointer/link yang disebut dengan link *next*.**



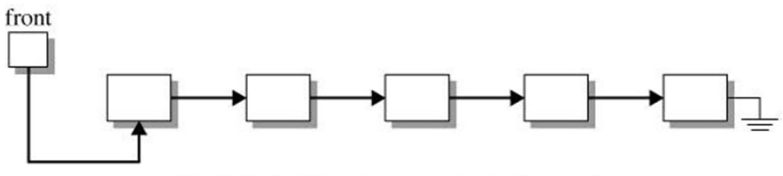
Single Linked List

- Linked list terpencar-pencar di memori
- Link dari elemen ke elemen berarti sebagai penjamin bahwa semua elemen dapat diakses.



Single Linked List

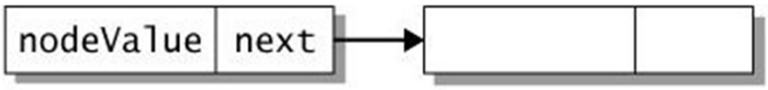
- Single Linked List tidak bisa diakses secara langsung. Dapat diakses secara sequential dengan mengakses maju satu persatu node.



Singly linked list where an element is a node.

Pembuatan Single Linked List

- Sebuah node pada linked list mempunyai dua instance variabel :
  - nodeValue dengan tipe generics T.
  - next dengan tipe Node yang merupakan link ke node berikutnya.



Pembuatan Single Linked List Class Node

- Class Node memiliki dua constructor yaitu:
  - Default constructor : menginisialisasi variabel nodeValue dan next dengan null
  - Constructor dengan parameter : memberikan nilai pada variabel nodeValue dan memberikan nilai pada variabel next dengan null.

Pembuatan Single Linked List Class Node

```
public class Node<T>
{
   // variabel node untuk menyimpan data
   public T nodeValue;
   // link node untuk menghubungkan ke node berikutnya
   public Node<T> next;

   // default constructor
   public Node()
   {
      nodeValue = null;
      next = null;
   }
```

Pembuatan Single Linked List Class Node

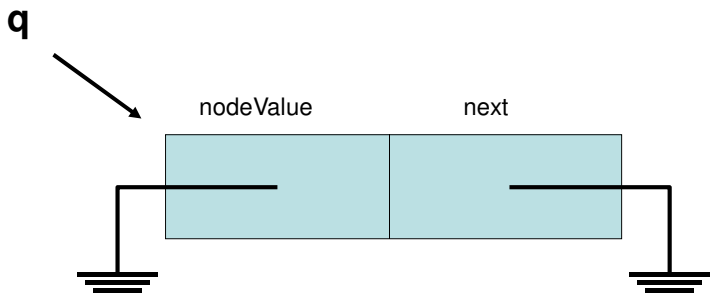
```
   // menginisialisasi nodeValue dengan item
   //dan memberikan nilai next dengan null
   public Node(T item)
   {
      nodeValue = item;
      next = null;
   }
}
```

Pembuatan Single Linked List

- Memerlukan variabel reference front untuk menandai node pertama pada list.
- Pada saat kita di node pertama, maka kita dapat menggunakan variabel next untuk menghubungkan dengan node ke dua, node ke tiga dan node berikutnya.

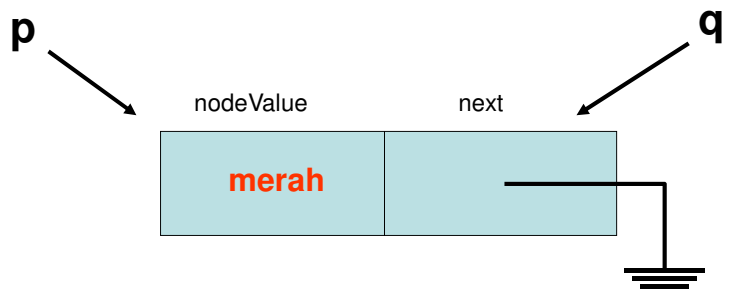
Membuat Node

- `Node<String> q = new Node<String>();`

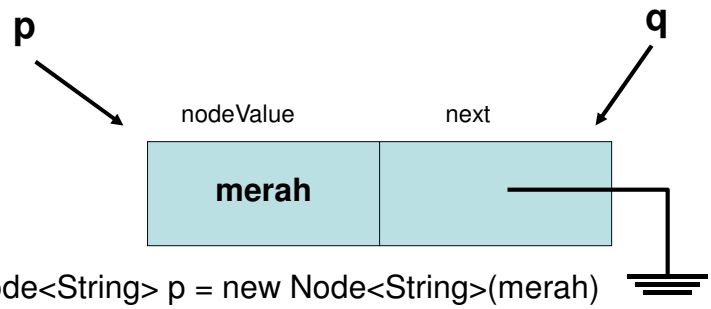


Membuat Node

- `Node<String> p = new Node<String>("merah");`
- `Node<String> q = p ;`



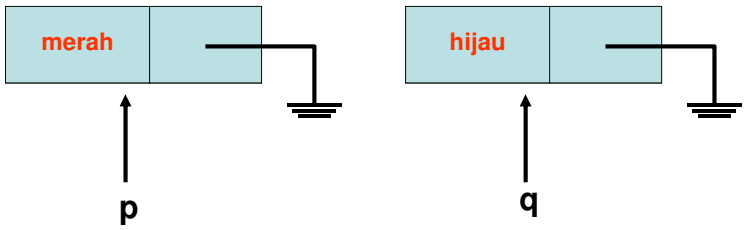
Cara Mengakses Node



- Node<String> p = new Node<String>(merah)
- p.nodeValue (merah)
- p.next (null)
- q = p
- q
- q.nodeValue (merah)
- q.next (null)

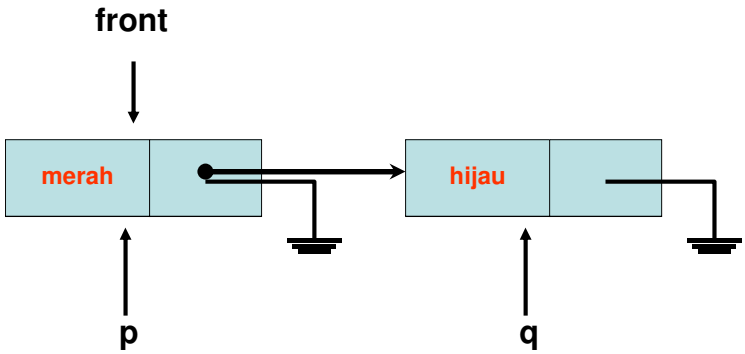
Contoh

```
Node<String> front, p, q;
p = new Node<String>("merah");
q = new Node<String>("hijau");
```



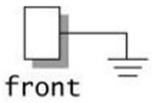
Contoh

```
p.next = q;
front = p;
```



Pembuatan Single Linked List

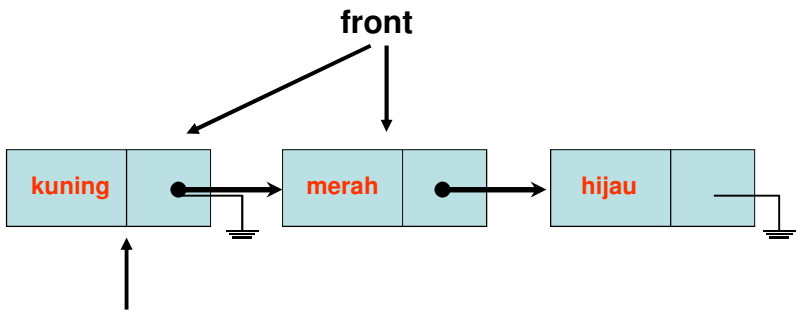
- Jika linked list kosong, maka front bernilai null.



Node<T> front = null

Menambahkan Node di Depan List

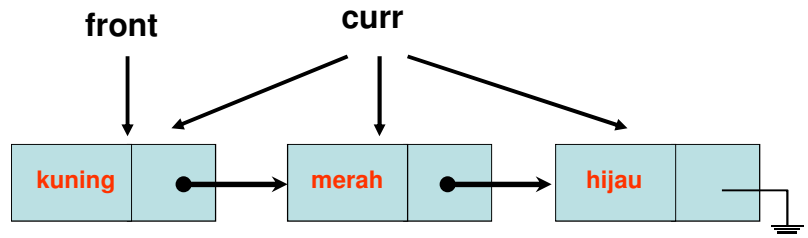
Node<String> newNode = new Node<String>(“kuning”);
newNode.next = front;
front = newNode;



Membaca Linked List

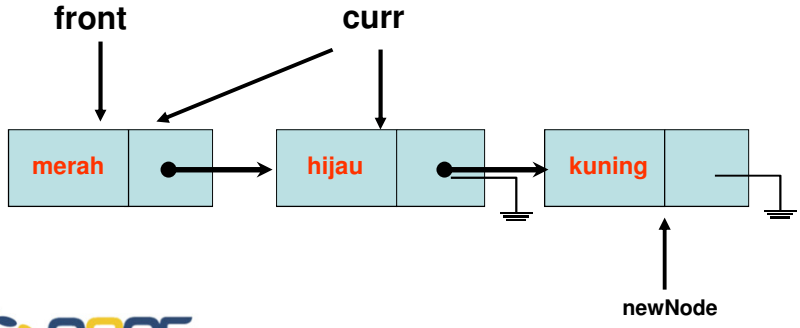
[kuning ,merah,hijau]

```
Node<T> curr = front;
String str = "[" + curr.nodeValue;
while(curr.next != null)
{       curr = curr.next;
        str += ", " + curr.nodeValue; }
str += "]";
```



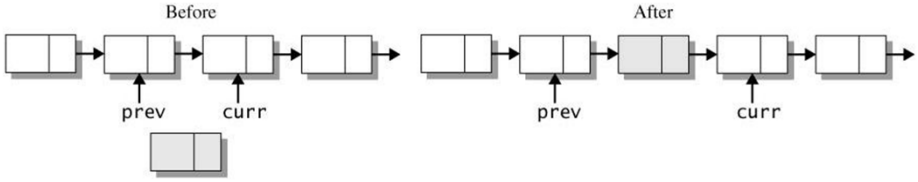
Menambah Node di Akhir List

```
Node<T> curr = front ;
 while(curr.next != null){
     curr = curr.next ;
 }
 curr.next = newNode ;
```



Menyisipkan Node di Linked List

- Untuk menyisipkan node baru sebelum node yang diacu oleh **curr**, maka perlu menandai node sebelum curr yaitu node **prev** karena node baru tersebut diletakkan setelah node prev dan sebelum node curr.

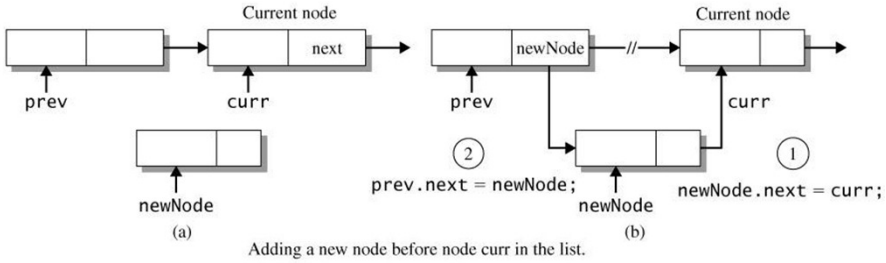


Menyisipkan Node di Linked List

- Membuat node baru (newNode) dengan value item.
- Menghubungkan newNode pada list yang memerlukan perubahan nilai pada newNode.next dan prev.next.

Menyisipkan Node di Linked List

```
Node curr, prev, newNode;
// membuat node baru dan memberikan nilai
newNode = new Node(item);
// update link
newNode.next = curr;        // step 1
prev.next = newNode;        // step 2
```



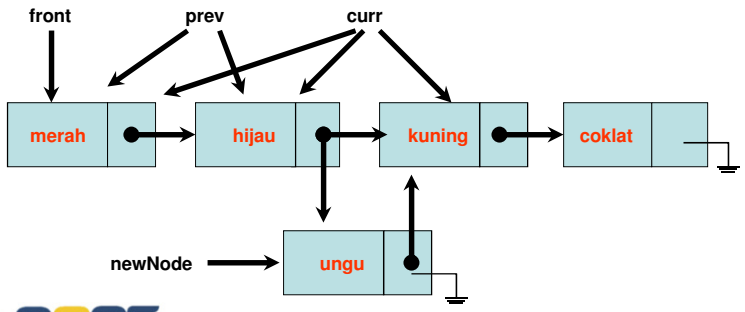
Adding a new node before node curr in the list.

```
curr = front ;
while (curr != null && !foundItem) {
     if (target.nodeValue.equals(curr.nodeValue)) {
            // menambah di tengah
             prev.next = newNode;
             newNode.next = curr ;
             foundItem = true;
       } else {
          // advance curr and prev
          prev = curr;
          curr = curr.next;
       }
}
```

true

false

Menyisipkan sebelum target.nodeValue = kuning

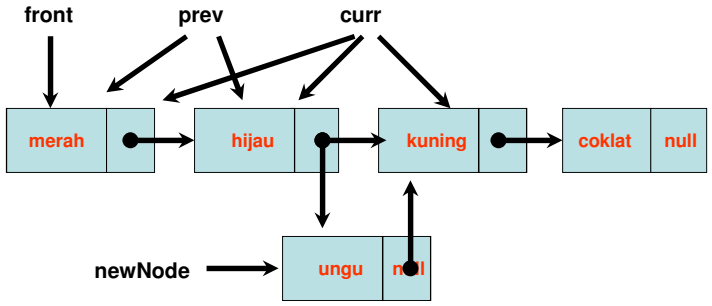


```
curr = front ;
while (curr != null && !foundItem) {
      if (target.nodeValue.equals(curr.nodeValue)) {
            // menambah di tengah
             prev.next = newNode;
             newNode.next = curr ;
             foundItem = true;
        } else {
           // advance curr and prev
           prev = curr;
           curr = curr.next;
        }
}
```

true

false

Menyisipkan sebelum target.nodeValue = kuning

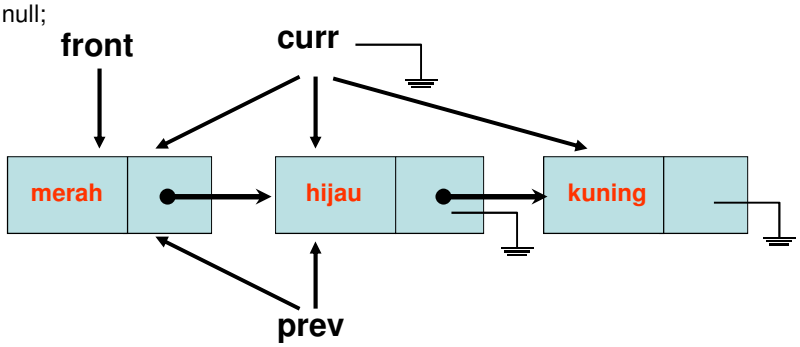


Menghapus Node di Akhir List

```
curr = front;
while (curr.next != null) {
    prev = curr;
    curr = curr.Next;}
prev.next = null;
curr = null;
```

true

false



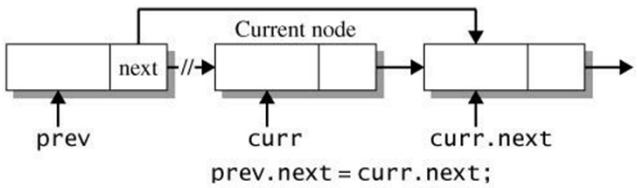
Menghapus Node

- Menghapus node pada posisi curr juga memerlukan pengaksesan ke node sebelumnya yang ditunjuk oleh prev.
- Ubah link dari prex.next menuju curr.next

Menghapus Node Sesuai Target

```
Node curr, prev;

// reconnect prev to curr.next
prev.next = curr.next;
```



Removing a node at position curr in the list.

Menghapus Node Sesuai Target

```
curr = front;
while (!curr.valueNode.equals(target.valueNode) &&
      curr.next != null){
      prev = curr;
      curr = curr.next;}
prev.next = curr.next;
curr.next = null;
curr = null;
```

Target.nodeValue = hijau