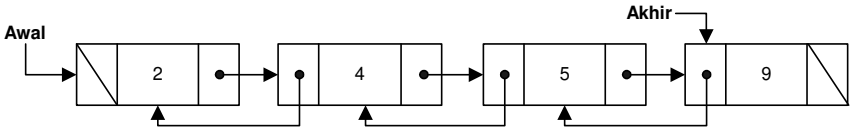
II. Double Linked List

Double Linked List adalah suatu linked list yang mempunyai 2 penunjuk yaitu penunjuk ke simpul sebelumnya dan ke simpul berikutnya. Perhatikan gambar di bawah ini :



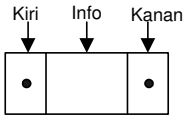
Deklarasi secara umum double linked list :

```
Type
  nama_pointer = ↑Simpul
  Simpul  = Record
      medan_data  : tipedata
      medan_sambungan_kiri, medan_sambungan_kanan : Namapointer
  EndRecord
  nama_var_pointer : nama_pointer
```

Contoh:

```
Type
      Point =   ↑Data
      Data  = Record
          <  info   : char
             prev, next  : Point >
      Endrecord
      awal, akhir: Point
```

Jadi satu simpul di double linked list adalah sebagai berikut :

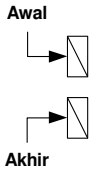


Dari gambar di atas, untuk setiap simpul terdiri dari 3 buah field yaitu medan sambungan kiri (prev), medan data (info), dan medan sambungan kanan (next).

Beberapa operasi yang dapat dilakukan dalam double linked list adalah :

1. Penciptaan

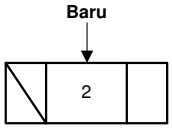
Penciptaan adalah memberikan nilai nil terhadap variabel pointer awal dan variabel pointer akhir.



2. Penyisipan

2.a <u>Penyisipan di depan/awal</u>

Operasi ini berguna untuk menambahkan satu simpul baru di posisi pertama. Langkah pertama untuk penambahan data adalah pembuatan simpul baru dan mengisinya dengan data pada field info-nya. Pointer yang menunjuk ke simpul tersebut dipanggil dengan nama **baru**. Kondisi setelah ada pembuatan simpul baru tersebut adalah :

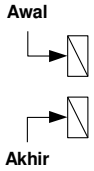


Ada 2 kondisi yang harus diperhatikan dalam penambahan data di awal yaitu :

a. Ketika linked list masih kosong

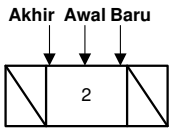
Kalau kondisi linked list masih kosong, maka simpul baru akan menjadi simpul awal dan sekaligus simpul akhir dari double linked list. Perhatikan gambar di bawah ini :

- Kondisi sebelum disisipkan



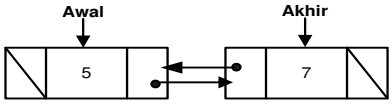
- Kondisi setelah operasi penyisipan

  Operasi penyisipan pertama kali ketika linked list masih kosong adalah dengan mengisikan alamat pointer baru ke pointer awal dan pointer akhir. Dan memberikan nilai nil pada medan sambungan kiri (prev) dan medan sambungan kanan(next). Lihat gambar di bawah ini:



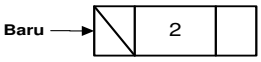
b. Ketika linked list tidak kosong

Misalnya mula-mula keadaan list sebagai berikut:

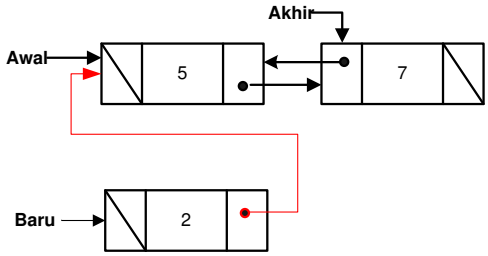


Proses penyisipan simpul di awal linked list adalah :

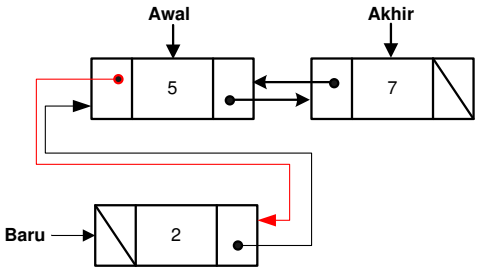
- Tempatkan sebuah pointer baru di alamat yang bertipe record untuk double linked list, kemudian medan sambungan kirinya diberi harga nil.



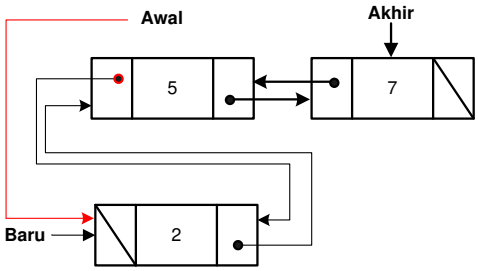
- Hubungkan medan sambungan kanan dari simpul yang ditunjuk oleh pointer baru ke simpul yang ditunjuk oleh pointer awal.



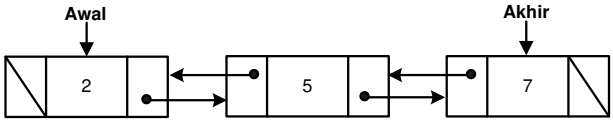
- Hubungkan medan sambungan kiri (prev) dari simpul yang ditunjuk oleh pointer awal ke simpul yang baru.



- Pindahkan pointer awal ke simpul yang baru



Maka bentuk linked list setelah terjadi penyisipan di awal adalah:



2.b. <u>Penyisipan di tengah</u>

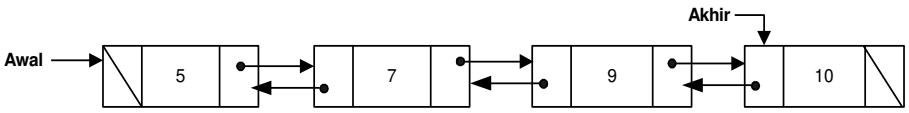
Operasi penyisipan data di tengah linked list adalah suatu operasi menambah data di posisi tertentu di dalam linked list. Karena double linked list memiliki dua pointer sambungan, maka penyisipan bisa dilakukan sebelum data tertentu atau sesudah data tertentu, berbeda dengan single linked list yang hanya memiliki satu pointer sambungan yaitu sambungan kanan(next).

Untuk proses tersebut ada 2 hal yang harus diperhatikan yaitu :

- **Kondisi linked list masih kosong** prosesnya sama seperti penyisipan di depan/awal.

- **Kondisi linked list sudah mempunyai data (tidak kosong)**

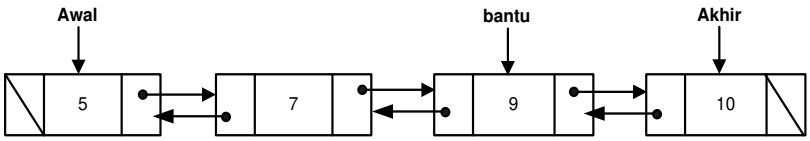
  Mula-mula keadaan linked list sebagai berikut:



  Misalnya akan menyisipkan data 6 sebelum data 9 (untuk menyisipkan data setelahnya lihat kembali pada single linked list)

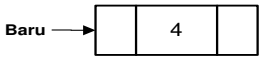
  Langkah untuk penyisipan data ketika list tidak kosong adalah:

  - Cari data yang akan disisipkan setelahnya pada double linked (data 9), dimana simpul yang ada data 9 ditunjuk oleh pointer bantuan (**bantu**).

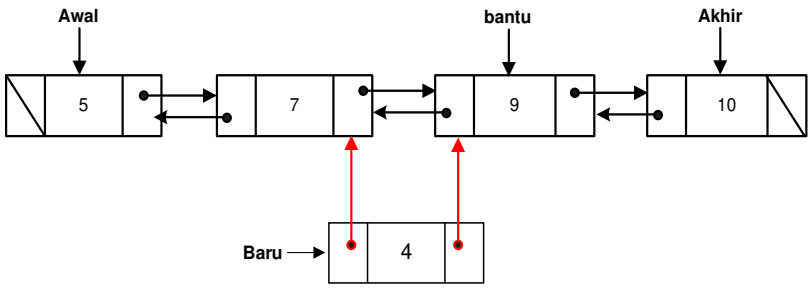


    Pointer bantu bisa berada di simpul yang ada data 9, karena melalui proses pencarian (searching). Metode searching yang digunakan bisa sequential search dengan boolean atau sequential search tanpa sentinel.

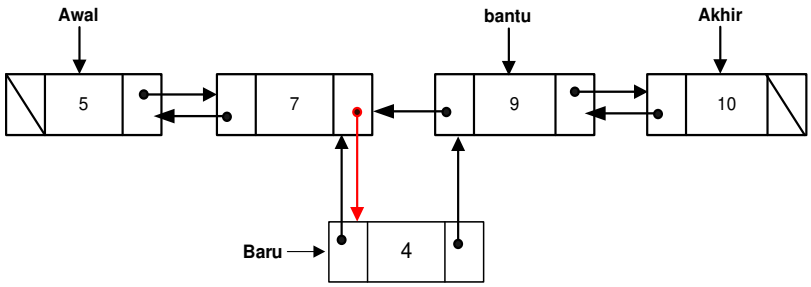
  - Jika data yang dicari ditemukan, maka akan terjadi penyisipan dengan langkah sebagai berikut:
    - Tempatkan pointer baru pada sebuah simpul baru berupa double linked yang akan disisipkan.



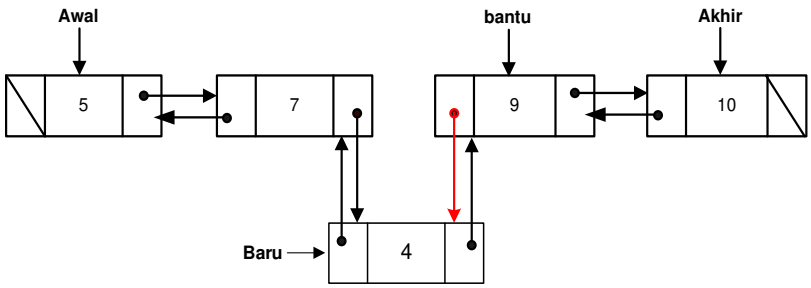
    - Sisipkan simpul yang baru atau data yang baru disebelum simpul yang dicari tadi, dengan langkah:
      a. Medan sambungan kanan (next) dan medan sambungan kiri (prev) dari simpul yang baru dihubungkan ke simpul yang ada data 9 dengan simpul yang ada data 7.



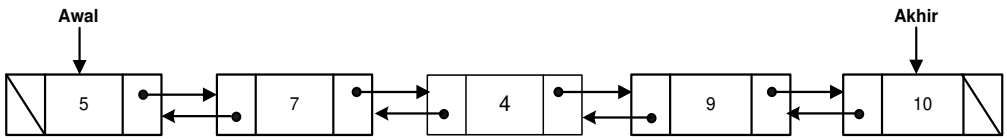
b. Hubungkan medan sambungan kanan (next) dari simpul tetangga kirinya bantu ke simpul yang baru.



c. Terakhir hubungkan medan sambungan kiri (prev) dari simpul yang ditunjuk oleh bantu ke simpul yang baru.

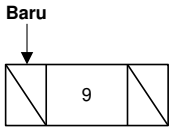


Maka bentuk double linked list setelah mengalami penyisipan di tengah seperti gambar berikut:



2.c. <u>Penyisipan di belakang/akhir</u>

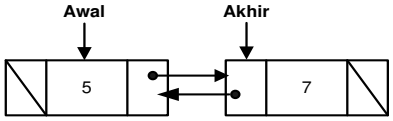
Operasi ini berguna untuk menambahkan elemen baru di posisi akhir. Langkah pertama untuk penambahan data adalah pembuatan elemen baru dan pengisian nilai infonya. Pointer yang menunjuk ke data tersebut dipanggil dengan nama **baru**. Kondisi di setelah ada pembuatan elemen baru tersebut adalah :



Ada 2 kondisi yang harus diperhatikan dalam penambahan data di akhir yaitu :

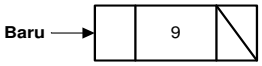
- **Kondisi linked list masih kosong** prosesnya sama seperti penyisipan di depan/awal.
- **Ketika linked list sudah mempunyai data**

  Mula-mula list sebagai berikut:

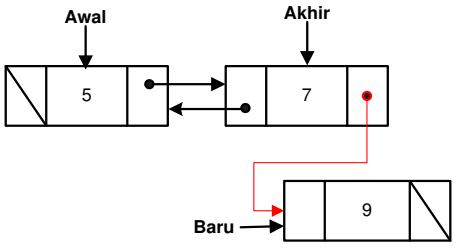


Proses penambahan data di akhir linked list adalah :

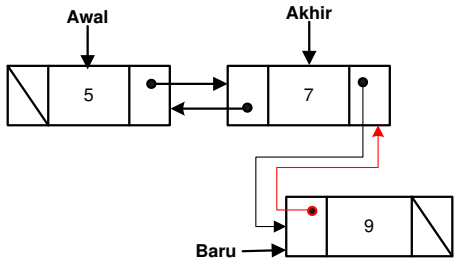
- Tempatkan pointer baru pada sebuah simpul double linked list yang baru, karena simpul yang baru ini nantinya menjadi simpul yang terakhir, maka medan sambungan dari simpul yang baru diberi harga nil.



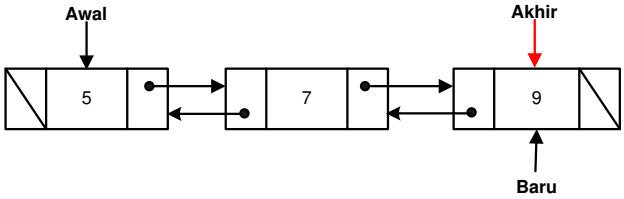
- Hubungkan medan sambungan kanan (next) dari simpul yang ditunjuk oleh pointer akhir ke simpul yang baru.



- Hubungkan medan sambungan kiri(prev) dari simpul yang baru ke simpul yang ditunjuk oleh pointer akhir.



- Pindahkan pointer akhir ke simpul yang baru



3. Penghapusan

a. Penghapusan di awal

Operasi ini berguna untuk menghapus data pada posisi pertama. Ada 3 keadaan yang mungkin terjadi ketika akan melakukan proses hapus yaitu :

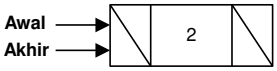
- Kondisi linked list masih kosong

  Jika kondisi ini terjadi, maka proses penghapusan data tidak bisa dilakukan karena linked list masih kosong.

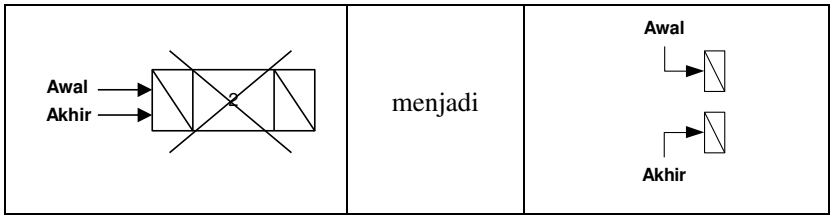
- Kondisi linked list hanya memiliki 1 data

  Langkah yang perlu dilakukan ketika ingin melakukan proses penghapusan linked list yang memiliki hanya 1 data adalah dengan langsung menghapus data dari memori dan kemudian pointer awal dan akhir diberi harga nil. Untuk lebih jelas perhatikan urutan penghapusannya di bawah ini :

  - Kondisi list sebelum dihapus



- Kondisi list setelah ada proses penghapusan

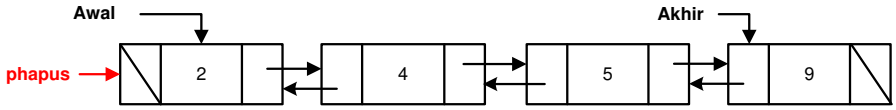


Kemudian pointer awal dan akhir diisi dengan nil.

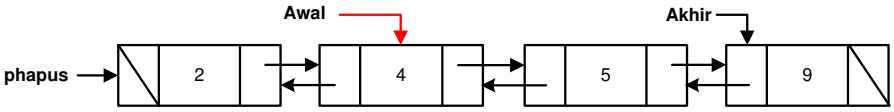
- Kondisi linked list memiliki lebih dari satu data atau satu simpul

  Untuk operasi penghapusan data di posisi pertama pada double linked list yang mempunyai data lebih dari satu buah adalah :

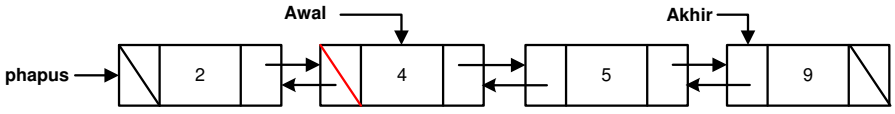
  - Tempatkan pointer bantuan (**phapus**) ke simpul (alamat) yang ditunjuk oleh pointer awal.



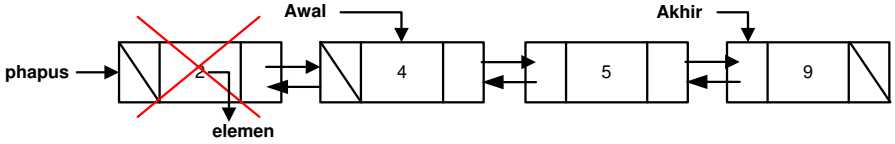
  - Pindahkan pointer awal ke simpul tetangga kanannya (next)



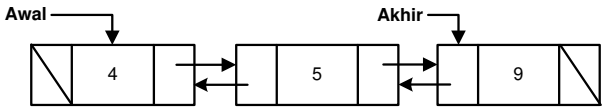
  - Sambungan kiri (prev) dari simpul yang ditunjuk oleh pointer awal diberi harga nil



  - Selamatkan data yang akan dihapus ke dalam variabel elemen, kemudian hapus simpul yang ditunjuk oleh pointer phapus



- Setelah simpul dihapus, maka kondisi linked list adalah seperti di gambar di bawah ini:



b. <u>Penghapusan di tengah</u>

Untuk melakukan proses penghapusan di tengah linked list, ada 3 kondisi yang perlu diperhatikan yaitu :

a. Kondisi ketika linked list masih kosong atau ketika posisi hapus lebih kecil dari 1.

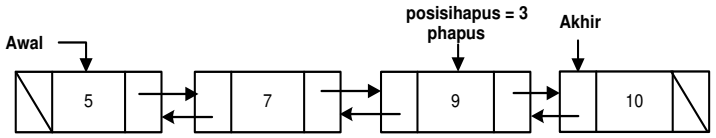
   Ketika kondisi ini terjadi, maka proses penghapusan tidak bisa dilakukan karena data masih kosong atau karena posisi hapus diluar jangkauan linked list (posisi kurang dari 1).

b. Kondisi ketika list memiliki satu simpul atau posisi hapus sama dengan satu (hapus data pertama)

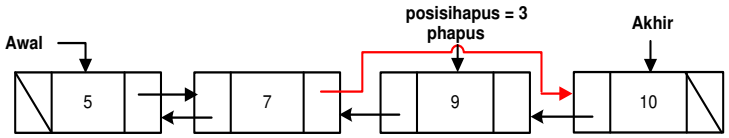
   Ketika kondisi ini terjadi, maka proses yang dilakukan adalah proses penghapusan di posisi awal (hapus_awal).

c. Kondisi ketika list lebih dari satu simpul atau posisi hapus lebih besar dari satu.

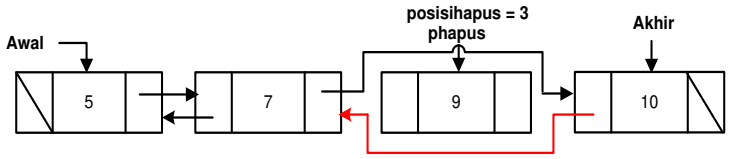
   Misalkan akan menghapus simpul yang ke-3:



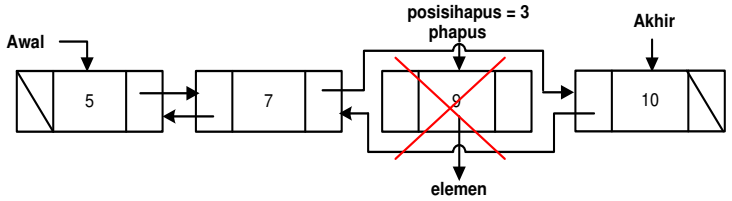
Hubungan sambungan kanan (next) dari simpul tetangga kirinya phapus dengan simpul tetangga kanannya phapus.



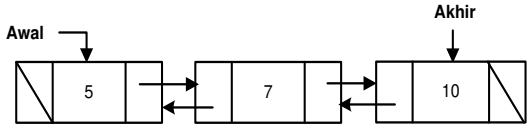
Kemudian hubungkan sambungan kiri dari tetangga kanannya phapus ke simpul tetangga kirinya phapus.



Simpan data yang akan dihapus ke dalam sebuah variabel **elemen**, kemudian simpul yang ditunjuk oleh phapus dapat dihapus.



Setelah terjadi penghapusan di tengah, maka linked list seperti gambar berikut:



c. <u>Penghapusan di akhir</u>

Operasi ini berguna untuk menghapus data pada posisi terakhir. Ada 3 keadaan yang mungkin terjadi ketika akan melakukan proses hapus yaitu :

a. Kondisi linked list masih kosong

Jika kondisi ini terjadi, maka proses penghapusan data tidak bisa dilakukan karena linked list masih kosong.

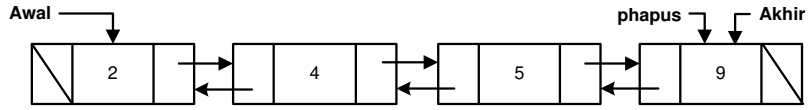
b. Kondisi linked list hanya memiliki satu data atau satu simpul

Penghapusan di akhir prosesnya sama seperti penghapusan di depan

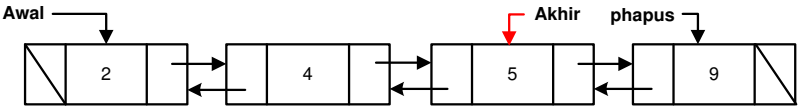
c. Kondisi linked list memiliki lebih dari satu data atau lebih dari 1 simpul

Untuk operasi penghapusan data di posisi terakhir pada double linked list yang mempunyai data lebih dari 1 buah adalah :

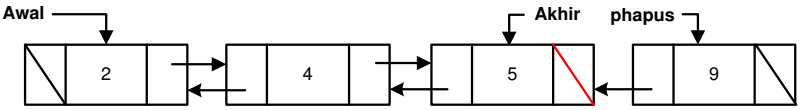
- Tempatkan pointer bantuan (**phapus**) di simpul yang terakhir



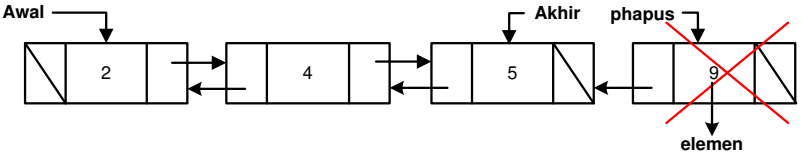
- Pindahkan pointer akhir ke simpul sebelumnya



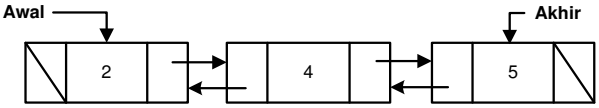
- Field kanan dari pointer akhir diberi harga nil



- Simpan data yang akan dihapus ke variabel elemen, kemudian hapus simpul yang ditunjuk leh pointer phapus.



- Setelah simpul dihapus, maka kondisi linked list adalah seperti di gambar di bawah ini.



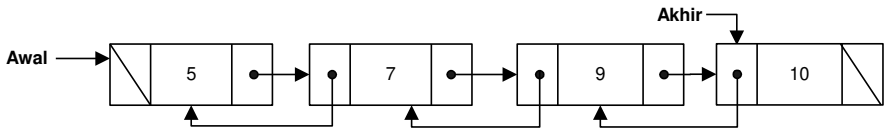
4. Penelusuran/traversal

Prosesnya secara umum sama seperti penelusuran pada single liked list.

5. Pencarian /Seaching

Pencarian dilakukan dengan memeriksa data yang ada dalam linked list dengan data yang dicari. Pencarian dilakukan dari data pertama sampai data ditemukan atau pointer pencari (bantu) telah mencapai akhir dari list yang menandakan bahwa data yang dicari tidak ditemukan.

Agar lebih jelas perhatikan ilustrasi di bawah ini, dengan contoh data adalah :

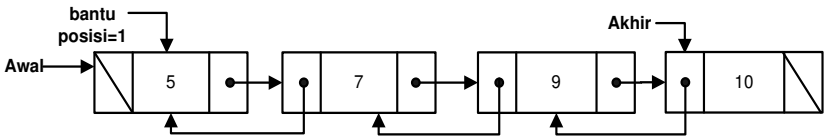


Ada 2 kondisi yang dihasilkan oleh proses pencarian yaitu

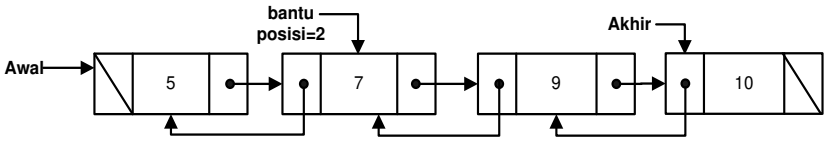
a. Pencarian dimana data yang dicari dapat ditemukan

Kasus : data yang akan dicari adalah data 9.

- Tempatkan pointer bantuan (**bantu**) di simpul pertama dan beri harga pada variabel posisi dengan angka 1



- Jika field info yang ditunjuk oleh pointer bantu tidak sama dengan data yang dicari, maka pointer bantu pindah ke simpul berikutnya dan variabel posisi harganya bertambah 1, ulang terus menerus sampai data yang dicari ditemukan atau sampai seluruh list ditelusuri.

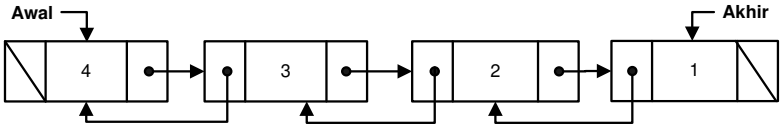


- Jika ditemukan, maka akan ada pernyataan “data yang dicari ada pada simpul ke sekian”, tetapi jika data yang dicari tidak ditemukan, maka akan ada pernyataan “data yang dicari tidak ditemukan/tidak ada”.

6. Pengurutan/sorting

Proses penyusunan data acak menjadi tersusun baik secara ascending ataupun secara descending pada dasarnya sama seperti pada single linked list, hanya saja pada double linked list dapat dilakukan dari kiri ke kanan atau dari kanan ke kiri dalam hal menyusuri list.

Misalkan akan mengurutkan data acak secara **ascending** dengan menggunakan metode **selection sort** yang jenisnya **minimum sort**:

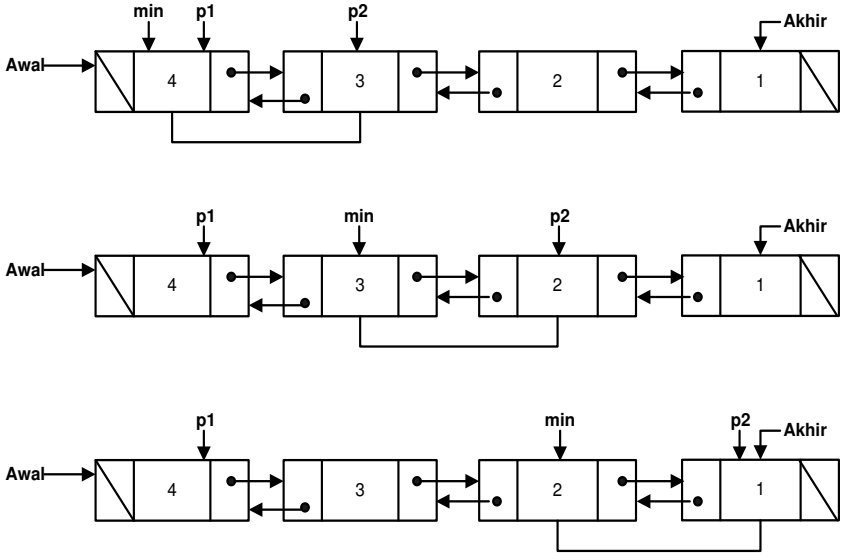


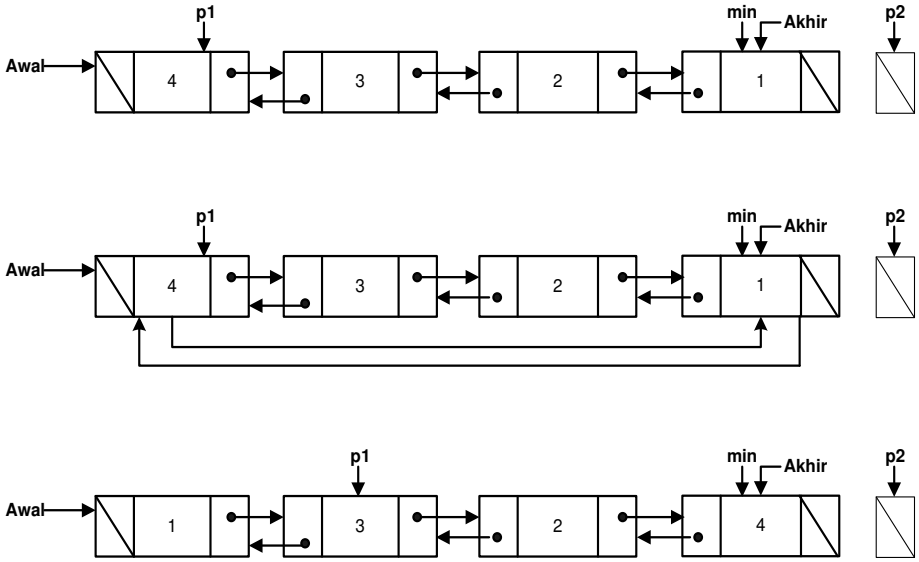
Proses pengurutannya adalah :

Bandingkan data di simpul yg ditunjuk oleh pointer p2 dengan data di simpul yang ditunjuk oleh pointer min, jika lebih kecil pindahkan pointer min ke simpul yang ditunjuk oleh pointer p2, kemudian pointer p2 pindah ke simpul tetangga kanannya, ulangi sampai pointer p2 bernilai nil (asumsi data terkecil sudah ditemukan dan simpulnya ditunjuk oleh pointer min).

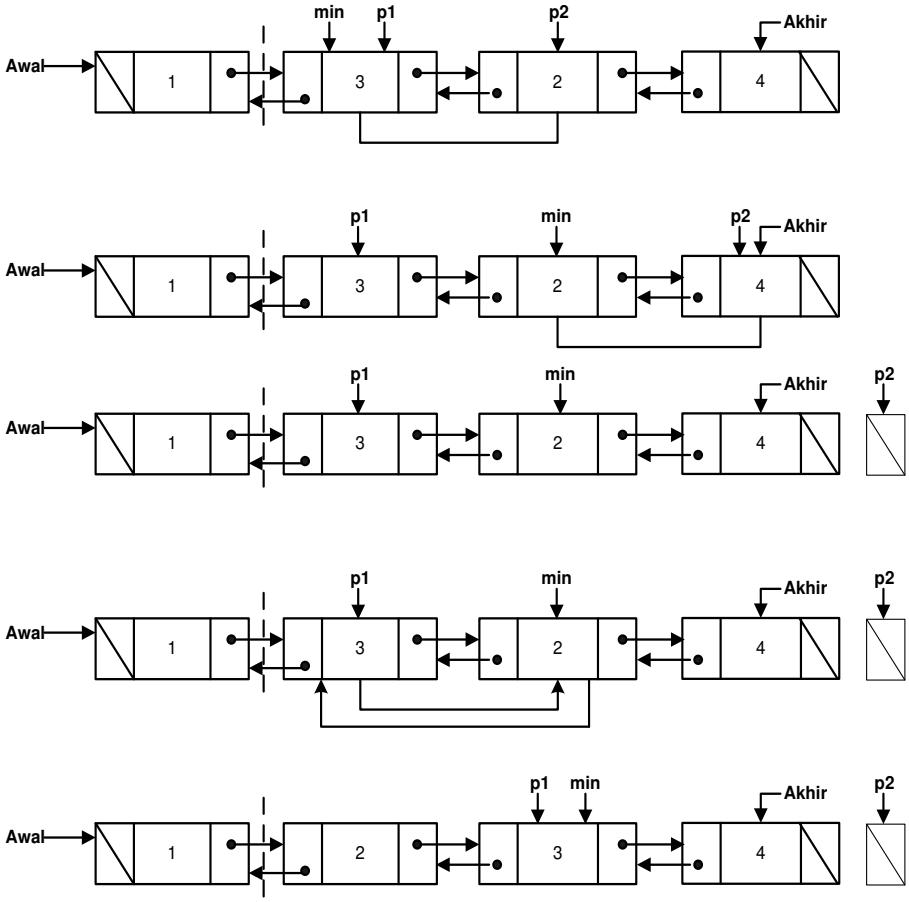
Setelah itu tukarkan data terkecil tadi dengan data di simpul yang ditunjuk oleh pointer p1.

Tahap I :

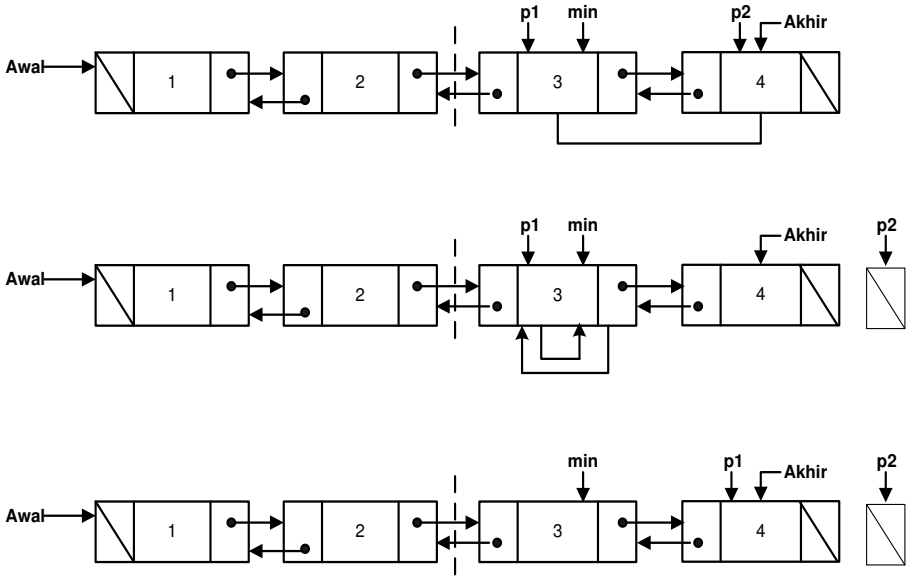



p1
min
Akhir
p2
Awal
4
3
2
1
p1
min
Akhir
p2
Awal
4
3
2
1
p1
min
Akhir
p2
Awal
1
3
2
4

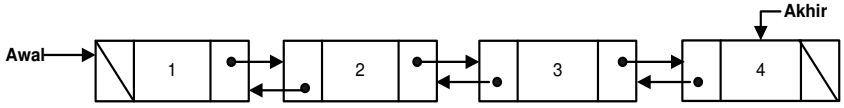
Tahap II :


min
p1
p2
Akhir
Awal
1
3
2
4
p1
min
p2
Akhir
Awal
1
3
2
4
p1
min
Akhir
p2
Awal
1
3
2
4
p1
min
Akhir
p2
Awal
1
3
2
4
p1
min
Akhir
p2
Awal
1
2
3
4

Tahap III :



Data acak akhirnya tersusun secara ascending sebagai berikut:



7. Penghancuran/destroy

Proses penghancuran bisa dengan cara memanggil modul/subrutin penghapusan di awal atau penghapusan di akhir secara terus menerus sampai list kosong, atau dengan proses penghancuran seperti pada single linked list.