BAB III
Double Linked List

Tujuan

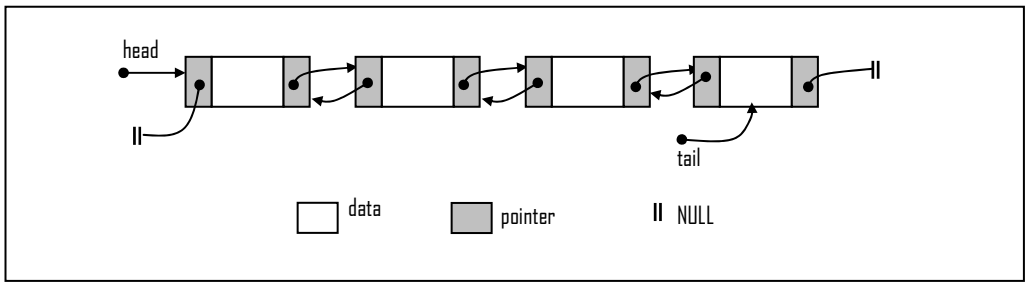
1. Memahami pengertian double linked list, gunanya dan dapat mengimplementasikan dalam pemrograman
2. Dapat mengidentifikasi permasalahan-permasalahan pemrograman yang harus diselesaikan dengan menggunakan double linked list, sekaligus menyelesaikannya

---

3.1 Double Linked List

Elemen-elemen dihubungkan dengan dua pointer dalam satu elemen. Struktur ini menyebabkan list bisa melintas baik ke depan maupun ke belakang. Masing-masing elemen pada double linked list terdiri atas tiga bagian, di samping data/informasi dan pointer `next`, masing-masing elemen dilengkapi dengan pointer `prev` yang menunjuk ke elemen sebelumnya. Double linked list dibentuk dengan menyusun sejumlah elemen sehingga pointer `next` menunjuk ke elemen yang mengikutinya dan pointer `prev` menunjuk ke elemen yang mendahuluinya.

Untuk menunjukkan `head` dari double linked list, maka pointer `prev` dari elemen pertama menunjuk `NULL`. Untuk menunjukkan `tail` dari double linked list tersebut, maka pointer `next` dari elemen terakhir menunjuk `NULL`. Susunan elemen yang dihubungkan dalam bentuk double linked list dapat dilihat pada Gambar 3.1

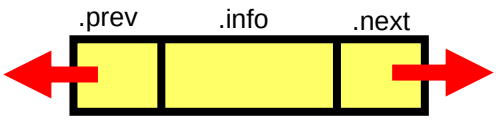


Gambar 3.1 Elemen yang Dihubungkan dalam Bentuk *Double Linked List*

Untuk melintas kembali melalui double linked list, kita gunakan pointer `prev` dari elemen yang berurutan pada arah `tail` ke `head`. Double linked list mempunyai fleksibilitas yang lebih tinggi daripada single linked list dalam perpindahan pada list. Bentuk ini sangat berguna ketika akan meletakkan suatu elemen pada list dan dapat memilih dengan lebih bijaksana bagaimana

memindahkannya. Sebagai contoh, salah satu fleksibilitas dari double linked list adalah dalam hal memindahkan elemen daripada menggunakan single linked list.

Double linked list dibentuk dengan menyusun sejumlah elemen sehingga pointer `next` menunjuk ke elemen yang mengikutinya dan pointer `prev` menunjuk ke elemen yang mendahuluinya. Dalam gambar 3.2 ini diilustrasikan sebuah simpul dalam double linked list. Info adalah data yang digunakan/disimpan dalam simpul, `prev` adalah pointer yang menunjuk pada simpul sebelumnya, dan `next` adalah pointer yang menunjuk pada simpul sesudahnya



Gambar 3.2 Ilustrasi sebuah simpul dalam Double Linked List

Deklarasi Simpul

Cara mendeklarasikan sebuah simpul dalam bentuk double linked list adalah sebagai berikut:

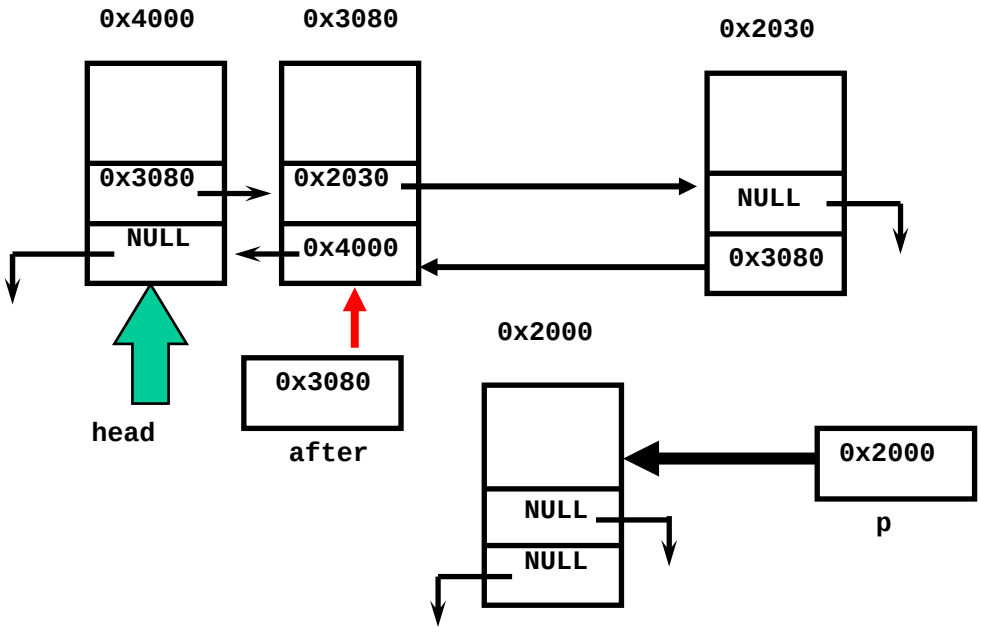
```
typedef struct simpul DNode;
struct simpul
{
      DNode *prev;
      int data;
      DNode *next;
};
DNode *head = NULL, *tail, *p;
```

3.2 Operasi dalam Double Linked List

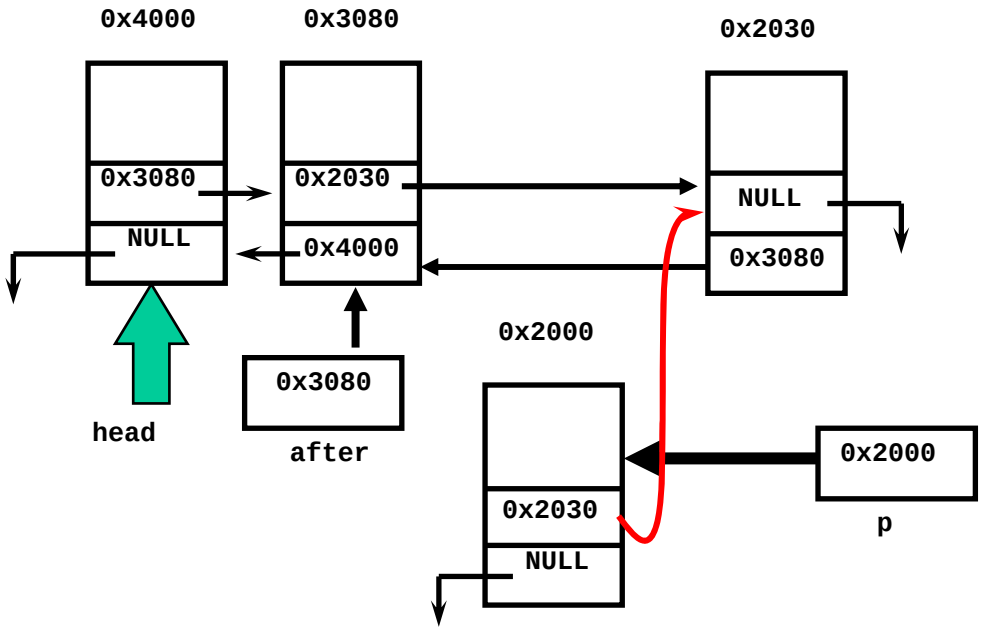
Sama seperti operasi yang ada dalam Single Linked List, operasi yang ada dalam Double Linked List dibagi menjadi 2, yaitu menyisipkan simpul dan menghapus simpul. Untuk menyisipkan simpul, secara garis besar juga sama seperti dalam Single Linked List. Namun untuk memahami konsepnya, kali ini kita membahas 2 macam operasi, yaitu menyisipkan sebagai simpul terakhir dan menyisipkan simpul di tengah.

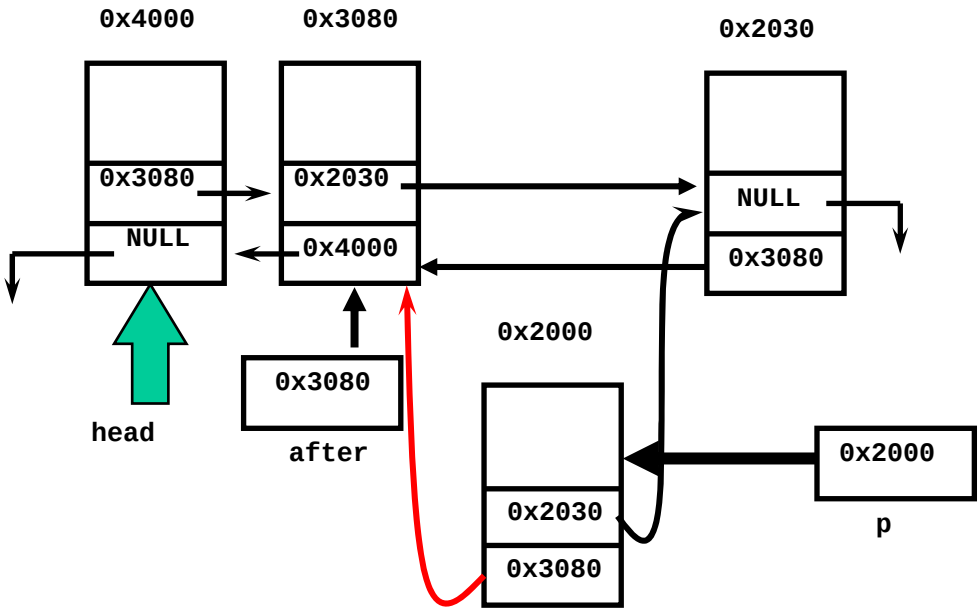
3.2.1 Operasi Penyisipan Node di Tengah

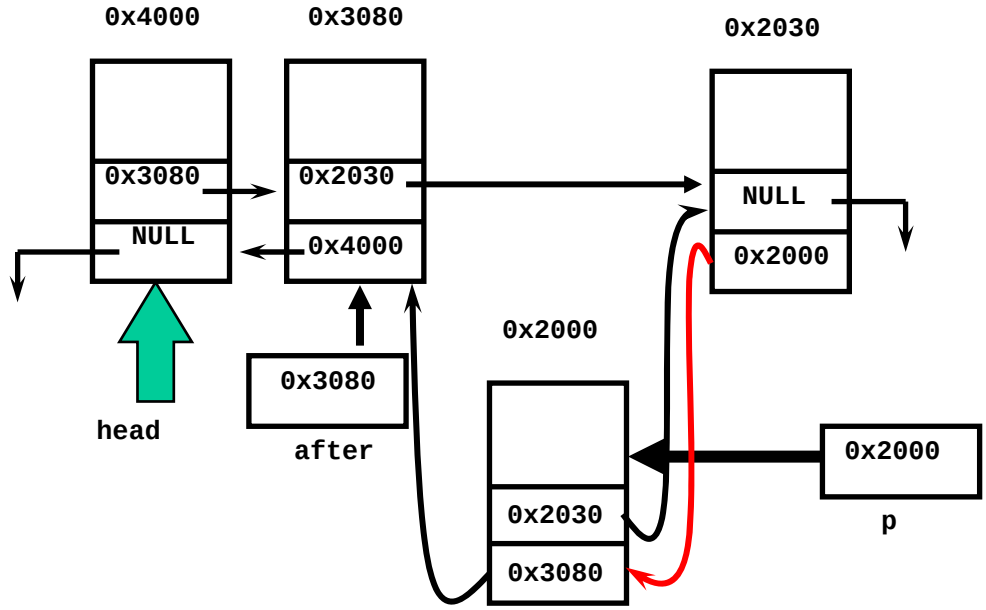
Dalam gambar-gambar di bawah ini akan diilustrasikan langkah-langkah untuk menyisipkan node di tengah.

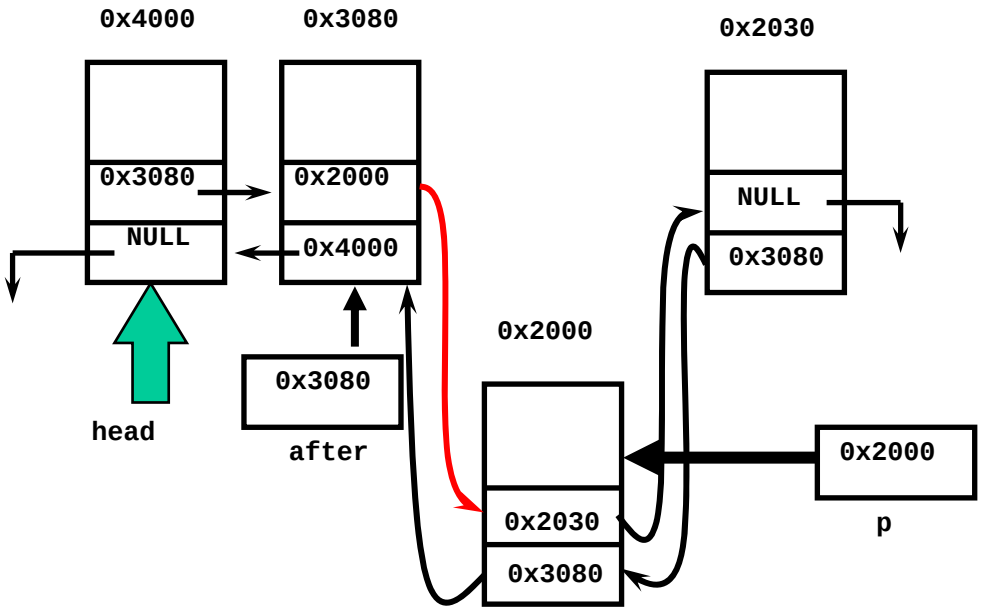


Gambar 3.3 Ilustrasi *Double Linked List* sebelum Disisipi pada Posisi Tengah




0x4000
0x3080
0x2030
0x3080
NULL
0x2030
0x4000
NULL
0x3080
0x2000
0x3080
head
after
0x2000
0x2030
0x3080
p


0x4000
0x3080
0x2030
0x3080
NULL
0x2030
0x4000
NULL
0x2000
0x2000
0x3080
head
after
0x2000
0x2030
0x3080
p



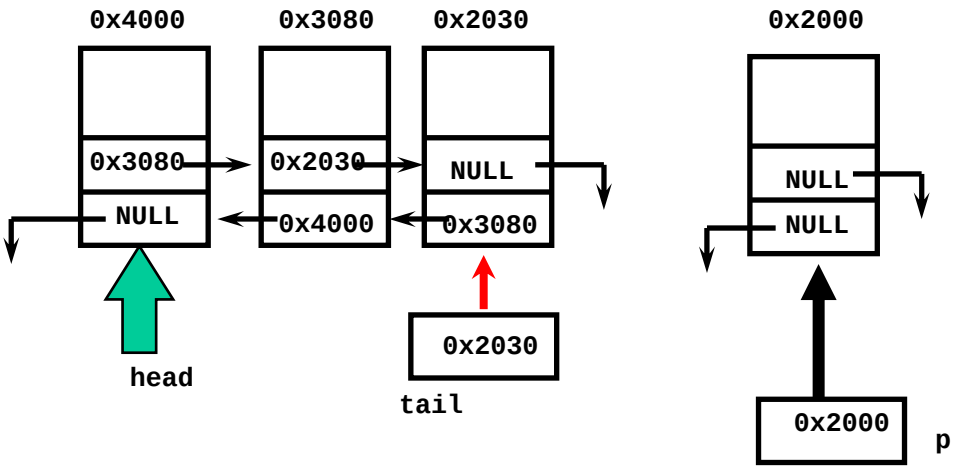
Gambar 3.4 Ilustrasi Proses Penyisipan sebagai Node di Tengah pada *Double Linked List*

Langkah-langkah untuk Menyisipkan Simpul di Tengah adalah sebagai berikut:

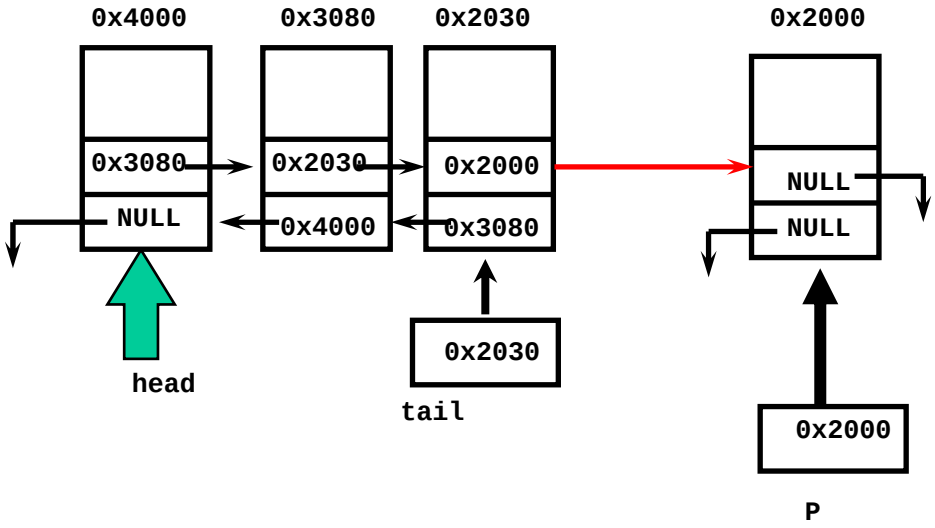
1. Siapkan node yang akan disisipkan (p adalah pointer bantuan untuk menunjuk kepada node baru yang akan disisipkan)
    a. Alokasikan memory-nya
    b. Jika berhasil, masukkan datanya dan set pointer `next` & `prev`-nya dengan `NULL`
2. Cari posisi yang akan disisipi sesudahnya.
    a. Gunakan pointer bantuan `after` untuk menunjuk node yang akan disisipi sesudahnya
    b. Gerakkan pointer `after` mulai dari `head` sampai ketemu posisi node yang akan disisipi selanjutnya atau node yang dicari tidak ada.
3. Sambungkan node baru dengan exsisting DLL.
    a. Arahkan pointer `p->next` ke `after->next`.
    b. Arahkan pointer `p->prev` ke `after`.
    c. Arahkan pointer `after->next->prev` ke `p`.
    d. Arahkan pointer `after->next` ke `p`.

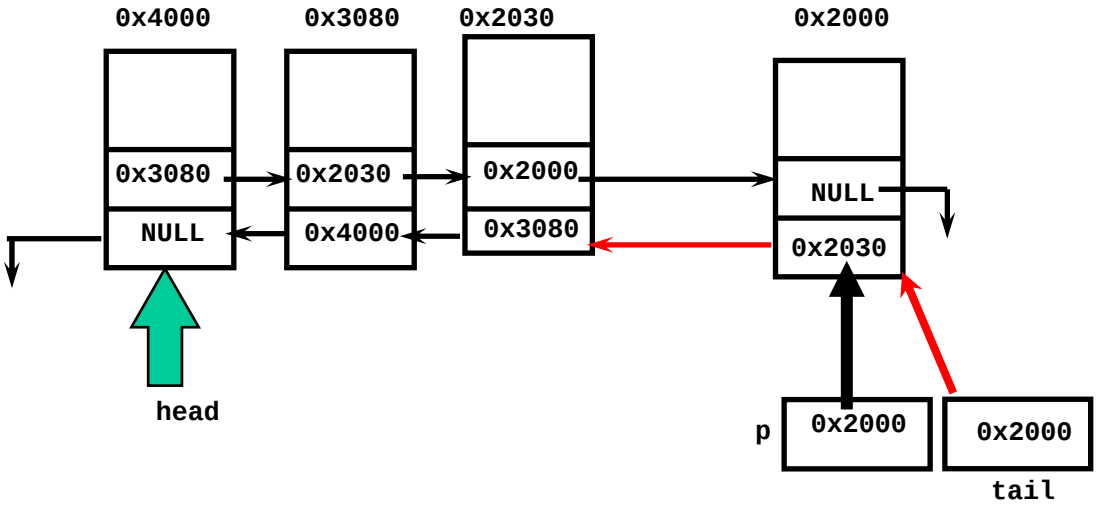
3.2.2 Operasi Penyisipan Node di Akhir

Dalam gambar-gambar di bawah ini akan diilustrasikan langkah-langkah untuk menyisipkan node di akhir.



Gambar 3.5 Ilustrasi *Double Linked List* sebelum Disisipi pada Node Terakhir





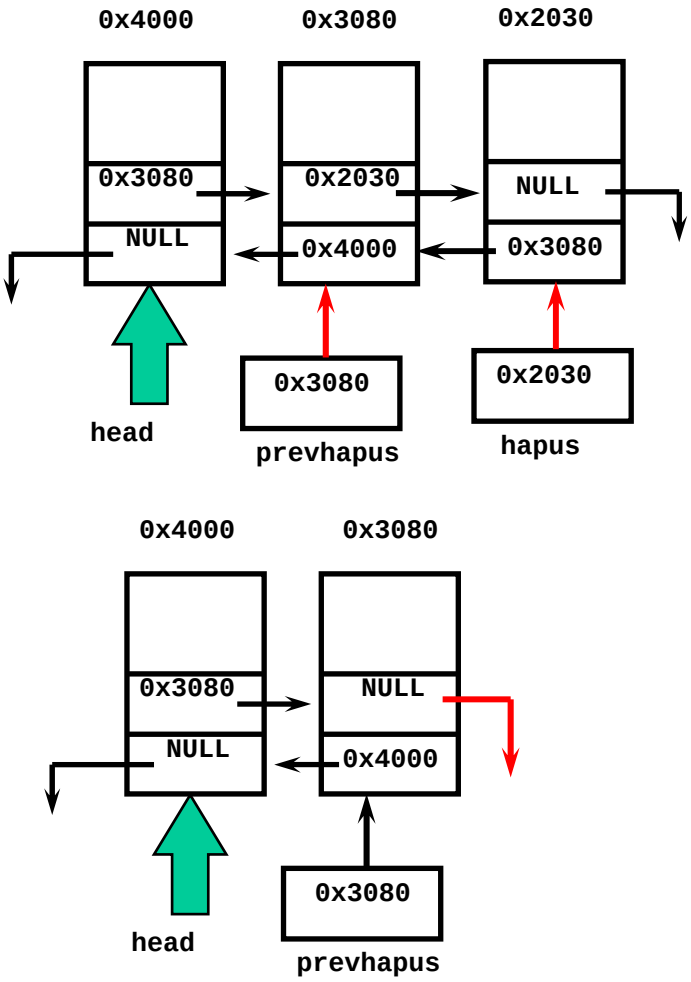
Gambar 3.6 Ilustrasi Proses Penyisipan sebagai Node Terakhir pada *Double Linked List*

Langkah-langkah untuk Menyisipkan sebagai Node Terakhir di atas adalah sebagai berikut:

1. Siapkan node yang akan disisipkan (`p` adalah pointer bantuan untuk menunjuk kepada node baru yang akan disisipkan)
    a. Alokasikan memory-nya
    b. Jika berhasil, masukkan datanya dan set pointer `next` & `prev`-nya dengan `NULL`
2. Cari posisi yang akan disisipi sesudahnya.
    a. Gunakan pointer bantuan `tail` untuk menunjuk node yang terakhir
    b. Gerakkan pointer `tail` mulai dari `head` sampai ketemu posisi node yang terakhir, yaitu node yang pointer `next` nya menunjuk ke `NULL`.
3. Sambungkan node baru dengan exsisting DLL.
    a. Arahkan pointer `tail->next` ke `p`
    b. Arahkan pointer `p->prev` ke `tail`.
    c. Update posisi `tail` ke supaya menunjuk ke `p`.

3.2.3 Operasi Penghapusan Node di Akhir

Dalam gambar-gambar di bawah ini akan diilustrasikan langkah-langkah untuk menghapus node di akhir.



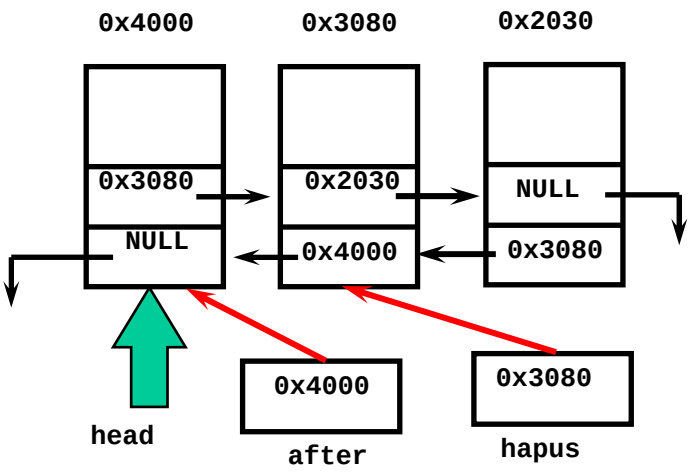
Gambar 3.7 Ilustrasi Operasi Penghapusan Node di Akhir

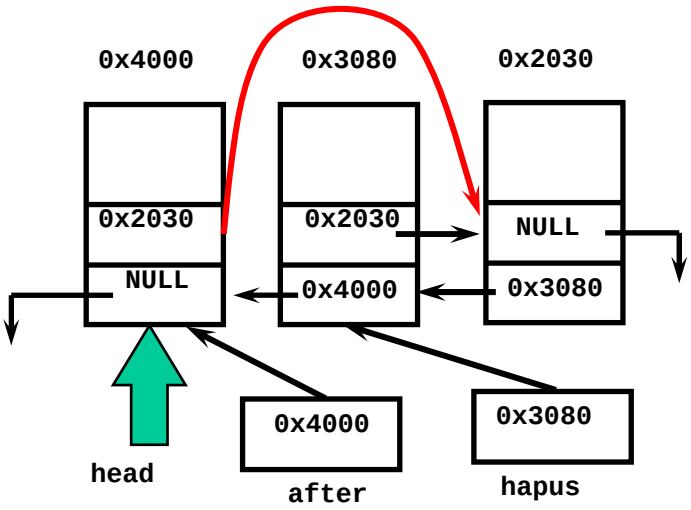
Langkah-langkah untuk Menhapus Node Terakhir di atas adalah sebagai berikut:

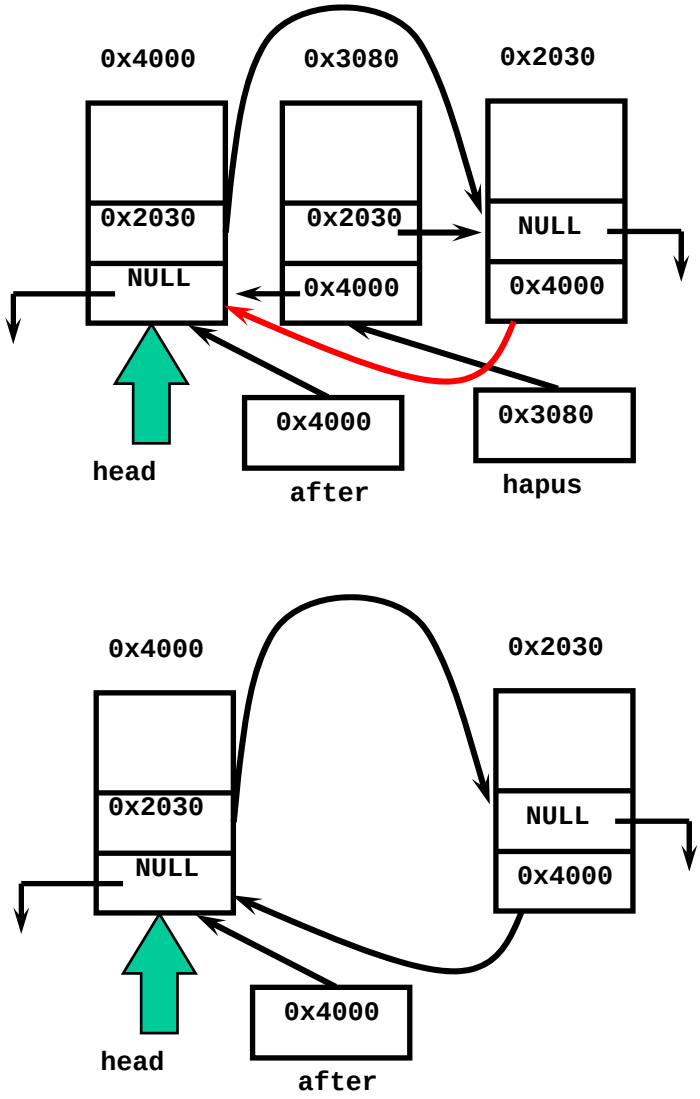
1. Cari posisi node yang akan dihapus, tandai dengan pointer `hapus` → Jalankan pointer `hapus` mulai dari `head` sampai ketemu node yang `next`nya menunjuk ke `NULL`
2. Selamatkan DLL-nya
    a. Gunakan pointer bantuan `prevhapus` untuk menunjuk node sebelum node yang terakhir (`prevhapus = hapus->prev`)
    b. Arahkan `prevhapus->next` ke `NULL`.
3. Hapus node yang ditandai dengan pointer `hapus` dan bebaskan memorynya.
    a. `free(hapus)`
    b. `hapus = NULL`

3.2.4 Operasi Penghapusan Node di Tengah

Dalam gambar-gambar di bawah ini akan diilustrasikan langkah-langkah untuk menghapus node di tengah (sesudah node tertentu)







Gambar 3.8 Ilustrasi Operasi Penghapusan Node di Tengah pada Double Linked List

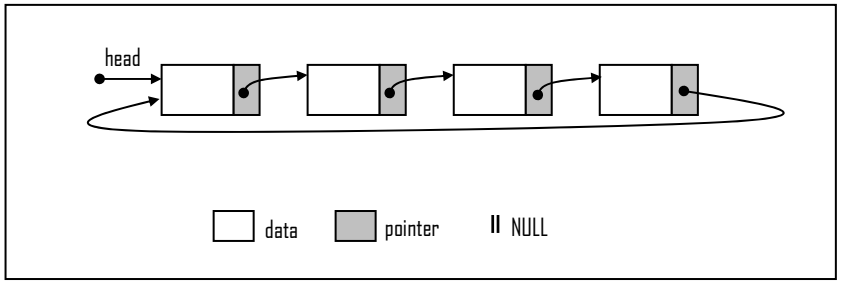
Langkah-langkah untuk Menhapus Node Terakhir di atas adalah sebagai berikut:

1. Cari posisi node yang akan dihapus, tandai dengan pointer `after` :
    a. Jalankan pointer `after` mulai dari `head` sampai ketemu node yang akan dihapus node sesudahnya.
    b. Gunakan pointer `hapus` yang diarahkan ke node sesudah after (`hapus = after->next`)
2. Selamatkan DLL-nya
    a. Arahkan pointer `after->next ke hapus->next`
    b. Arahkan `hapus->next->prev` ke `after`
3. Hapus node yang ditandai dengan pointer `hapus` dan bebaskan memorynya.
    a. `free(hapus)`
    b. `hapus = NULL`

3.3 Circular List

Circular list adalah bentuk lain dari linked list yang memberikan fleksibilitas dalam melewatkan elemen. *Circular list* bisa berupa single linked list atau double linked list, tetapi tidak mempunyai `tail`. Pada *circular list*, pointer `next` dari elemen terakhir menunjuk ke elemen pertama dan bukan menunjuk `NULL`. Pada *double linked circular list*, pointer `prev` dari elemen pertama menunjuk ke elemen terakhir.

Gambar 3.8 menunjukkan bagaimana susunan dari *single linked circular list*. *Circular list* yang akan dijelaskan pada bab ini merupakan *single linked circular list*. Kita hanya menangani link dari elemen terakhir kembali ke elemen pertama.



Gambar 3.8 Single Linked Circular List