Praktikum 7

Double Linked List (1)

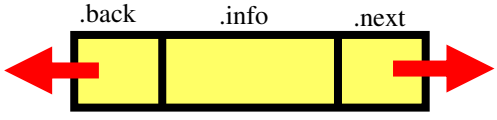
A. TUJUAN PEMBELAJARAN

Setelah mempelajari materi dalam bab ini, mahasiswa diharapkan mampu:

1. Memahami konsep membangun *double linked list*
2. Memahami konsep operasi menyisipkan sebagai simpul ujung(awal) dari *double linked list*
3. Memahami konsep operasi membaca sebuah simpul tertentu
4. Memahami konsep operasi mencari sebuah simpul tertentu
5. Memahami konsep operasi menghapus simpul tertentu
6. Mengimplementasikan semua operasi *double linked list* dalam pemrograman

B. DASAR TEORI

Double linked list dibentuk dengan menyusun sejumlah elemen sehingga pointer *next* menunjuk ke elemen yang mengikutinya dan pointer *back* menunjuk ke elemen yang mendahuluinya. Dalam gambar 7.1 ini diilustrasikan sebuah simpul dalam *double linked list.* Info adalah data yang digunakan dalam simpul, back adalah pointer yang menunjuk pada simpul sebelumnya, dan next adalah pointer yang menunjuk pada simpul sesudahnya



Gambar 7.1 Ilustrasi sebuah simpul dalam *Double linked list*

B.1 Bagaimana Membangun *Double linked list*

Untuk membangun sebuah linked list, maka terdapat beberapa langkah yang harus disiapkan. Langkah tersebut adalah sebagai berikut

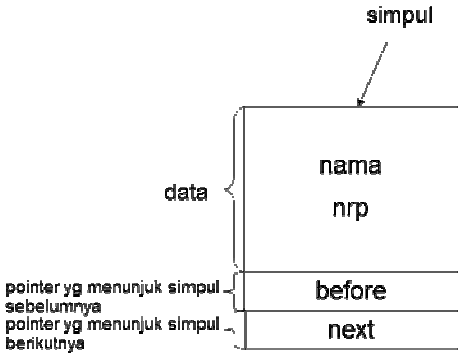
1. Deklarasi
2. Alokasi memori
3. Mengisi data
4. Menyiapkan untuk dihubungkan dengan data baru berikutnya

B.1.1 Deklarasi Simpul

Double linked list terdiri dari elemen-elemen individu, dimana masing-masing dihubungkan dengan dua pointer. Masing-masing elemen terdiri dari tiga bagian, yaitu sebuah data dan sebuah pointer yang berisi alamat data berikutnya disebut dengan *next* dan pointer yang berisi alamat data sebelumnya disebut *before*. Dengan menggunakan struktur *two-member* seperti ini, *linked list* dibentuk dengan cara menunjuk pointer *next* suatu elemen ke elemen yang mengikutinya. Pointer *before* pada elemen terakhir merupakan NULL, yang menunjukkan awal dari suatu list. Pointer *next* pada elemen terakhir merupakan NULL, yang menunjukkan akhir dari suatu list. Pada Gambar 7.2 ditunjukkan deklarasi untuk sebuah simpul menggunakan struktur yang diikuti deklarasi sebuah variable yang menggunakan struktur tersebut. Dari Gambar 7.2 didapatkan gambaran sebuah simpul pada Gambar 7.3.

```
struct simpul
{
      char nama[25];
      int nrp;
      struct simpul *before;
      struct simpul *next;
};
struct simpul *baru;
```

Gambar 7.2 Deklarasi Simpul dalam *Double linked list*



Gambar 7.3 Gambaran Simpul yang Dibangun dengan Deklarasi pada Gambar 7.2

B.1.2 Alokasi Memori

Ketika sebuah variabel dideklarasikan, terlebih dahulu harus diinisialisasi. Demikian juga dengan pengalokasian secara dinamis. Sehingga, fungsi untuk mengalokasikan sebuah node baru, fungsi `alokasi_simpul()` menggunakan `malloc()` untuk mendapatkan memori aktual, yang akan menginisialisasi suatu field data. next selalu diinisialisasi sebagai NULL.

Untuk melihat kemungkinan alokasi memori gagal, maka fungsi `alokasi_simpul()` menghasilkan NULL, bila berhasil maka menghasilkan sebuah simpul. Fungsi dari `alokasi_simpul()` adalah sebagai berikut :

```
struct simpul* alokasi_simpul()
{
      struct simpul * new;
      new = (struct simpul*)malloc(sizeof(struct simpul));
      if(new==NULL)
            return NULL;
      else
      {
           new->next=NULL;
           new->before=NULL;
           return new;
      }
}
```

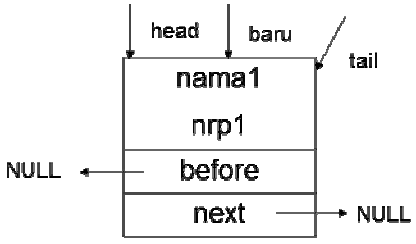
B.1.3 Mengisi Data Dan Menyiapkan Dihubungkan Dengan Data Berikutnya

Setelah melakukan deklarasi untuk simpul dan terdapat fungsi untuk melakukan alokasi memori maka keduanya kita gunakan untuk membangun sebuah *double linked list* dengan perintah sebagai berikut:

```
baru=alokasi_simpul();
if(baru==NULL)printf("Alokasi gagal");
else
{
      printf("Nama      :");scanf("%s",&baru->nama);
      printf("NRP :");scanf("%d",&baru->nrp);
      if(j==0)/*kondisi jika data masih kosong*/
      {
           head=baru;
           tail=baru;
      }
}
```

Gambar 7.4 Perintah untuk Membangun *Linked list*

Dengan perintah pada Gambar 7.4 didapatkan ilustrasi alokasi memori untuk sebuah simpul seperti Gambar 7.5.



Gambar 7.5 Simpul yang Terbentuk dengan Perintah pada Gambar 5.4

B.2 Operasi Pada *Double Linked List*

Terdapat beberapa Operasi yang penting pada *double linked list*, yaitu:

1. Menyisipkan sebagai simpul ujung(awal) dari linked list.
2. Membaca atau menampilkan
3. Mencari sebuah simpul tertentu
4. Menghapus simpul tertentu (simpul depan)
5. Menghapus simpul tertentu (simpul di tengah)

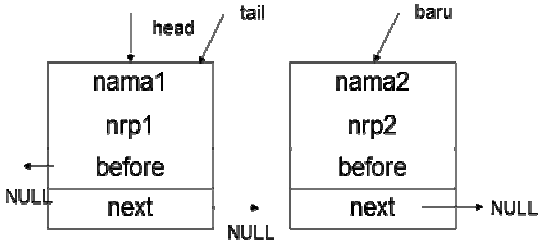
6. Menghapus simpul tertentu (simpul terakhir)
7. Menyisipkan sebelum simpul tertentu
8. Menyisipkan setelah simpul tertentu

B.2.1 Menyisipkan Sebagai Simpul Akhir Dari *Double linked list*

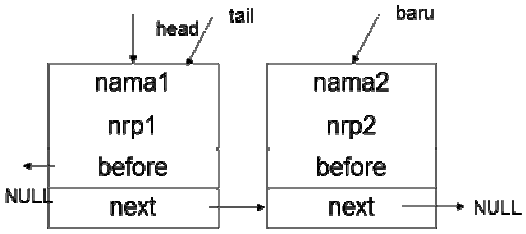
Setelah mendeklarasikan, mengisi data dan menyiapkan untuk dihubungkan dengan simpul berikutnya untuk menciptakan simpul pertama seperti yang dilakukan sebelumnya. Kita bisa melanjutkan dengan operasi menyisipkan sebagai simpul ujung dari linked list. Perintah untuk menyisipkan simpul kedua dst sebagai simpul paling ujung dari linked list adalah sebagai berikut:

```
1. baru = alokasi_simpul ();
2. printf("Nama   :");scanf("%s",&baru->nama);
3. printf("NRP    :");scanf("%d",&baru->nrp);
4. if(j!=0)
5. {
6.      tail->next=baru;
7.      baru->before=tail;
8.       tail=baru;
9. }
```

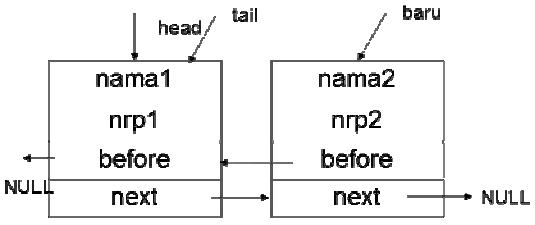
Setelah perintah baris ke-3



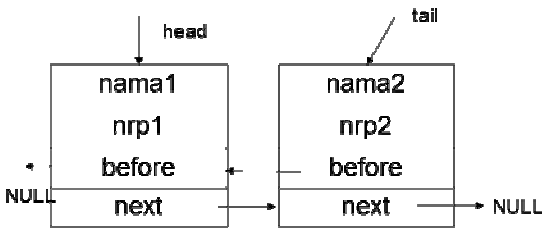
Setelah perintah baris ke-6



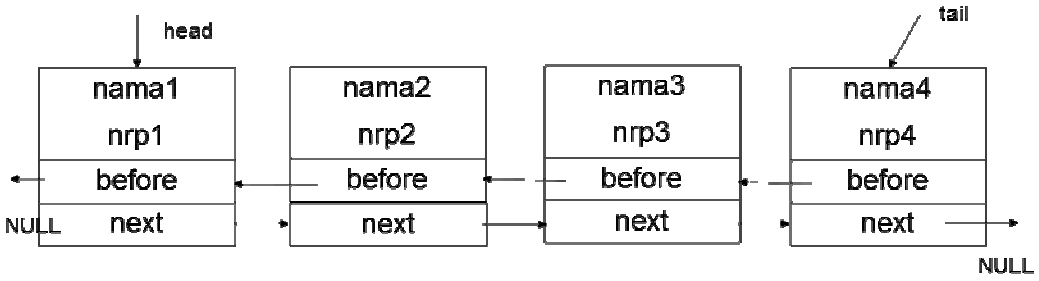
Setelah perintah baris ke-7



Setelah perintah baris ke-8



Setelah iterasi ke-4 membuat simpul



B.2.2 Membaca Atau Menampilkan

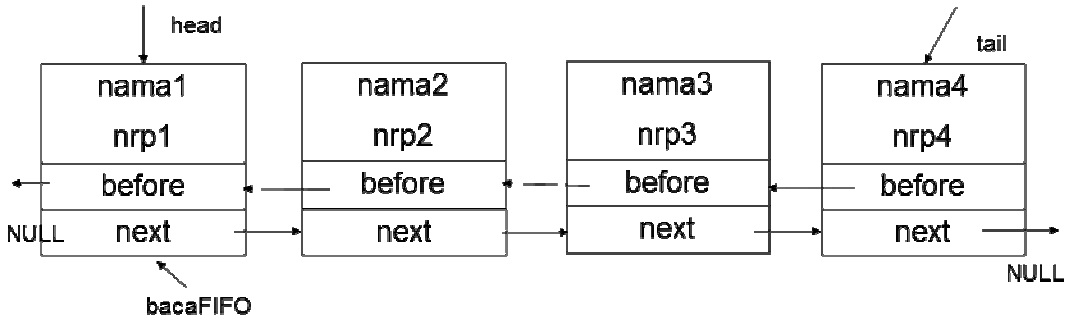
Berbeda dengan single linked list, untuk *double linked list* kita dapat menampilkan data secara FIFO(*First In First Out*) melalui *head* atau secara LIFO(*Last In First Out*) melalui *tail*. Langkah-langkah untuk membaca atau menampilkan double linked list dengan prinsip FIFO adalah sebagai berikut:

1. Inisialisasi sebuah variabel bertipe `struct simpul* (bacaFIFO)` dengan `head`
2. Tampilkan data nama dan nrp yang ada pada tampil
3. Jalankan `bacaFIFO = bacaFIFO->next`
4. Ulangi langkah 2 selama tampil tidak sama dengan NULL

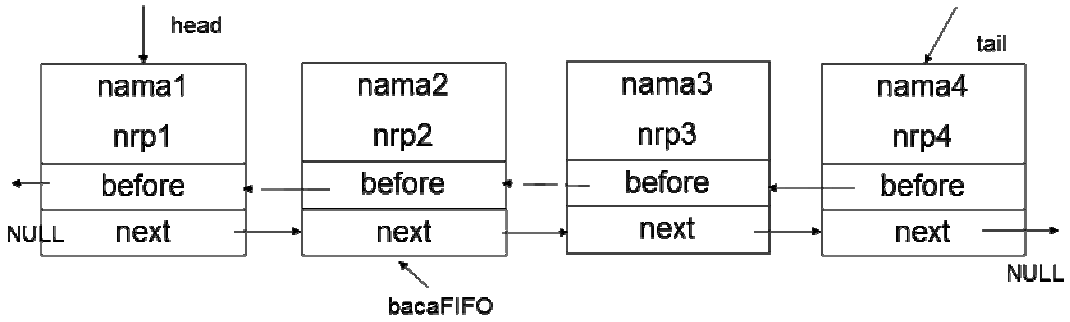
Berikut ini adalah perintah untuk membaca atau menampilkan data pada double linked list dengan prinsip FIFO

```
1. bacaFIFO = head;
2. while (bacaFIFO!=NULL)
3. {
4.      printf("%s",tampil->nama);
5.      printf("%d",tampil->nrp);
6.      bacaFIFO = bacaFIFO -> next;
7. }
```

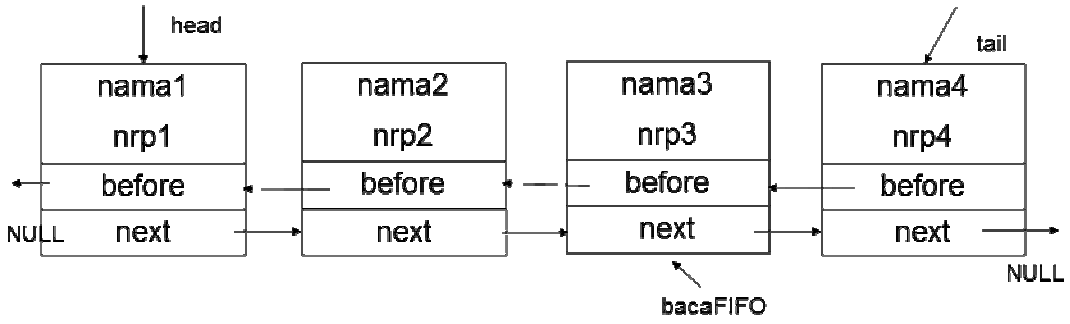
Setelah perintah baris 1



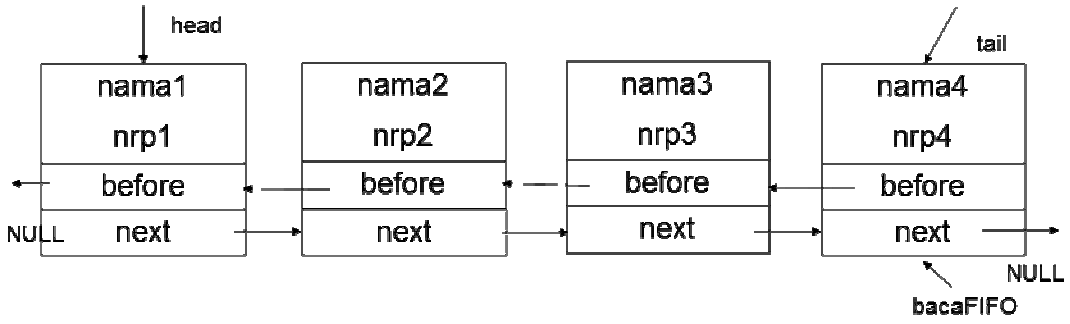
Setelah perintah baris ke-6 iterasi pertama



Setelah perintah baris ke-6 iterasi kedua



Setelah perintah baris ke-6 iterasi ketiga



Sedangkan langkah-langkah untuk membaca atau menampilkan double linked list dengan prinsip LIFO adalah sebagai berikut:

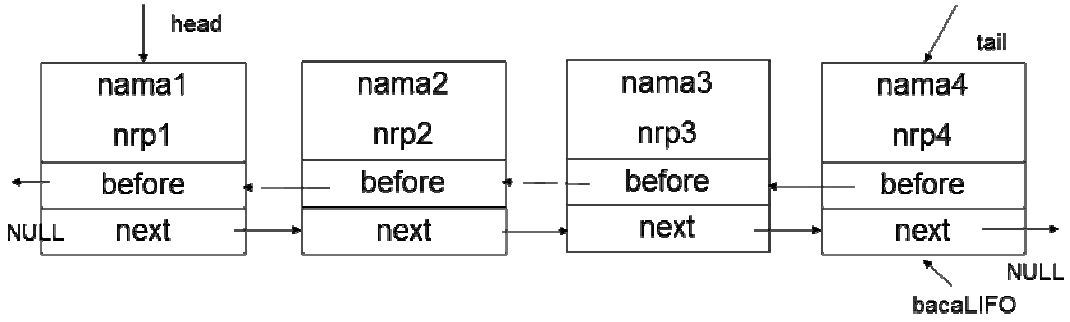
1. Inisialisasi sebuah variabel bertipe `struct simpul* (bacaLIFO)` dengan `tail`

2. Tampilkan data nama dan nrp yang ada pada `tampil`
3. Jalankan `bacaLIFO = bacaLIFO->before`
4. Ulangi langkah 2 selama tampil tidak sama dengan NULL

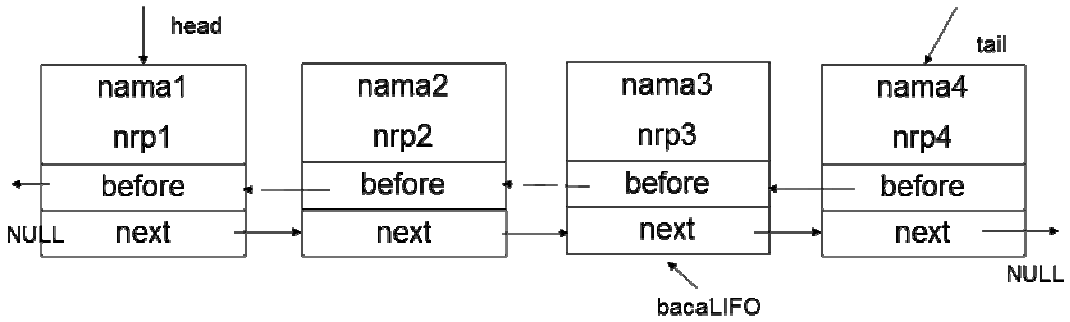
Berikut ini adalah perintah untuk membaca atau menampilkan data pada double linked list dengan prinsip LIFO

```
1. bacaLIFO = tail;
2. while (bacaLIFO!=NULL)
3. {
4.      printf("%s",tampil->nama);
5.      printf("%d",tampil->nrp);
6.      bacaLIFO = bacaLIFO -> before;
7. }
```

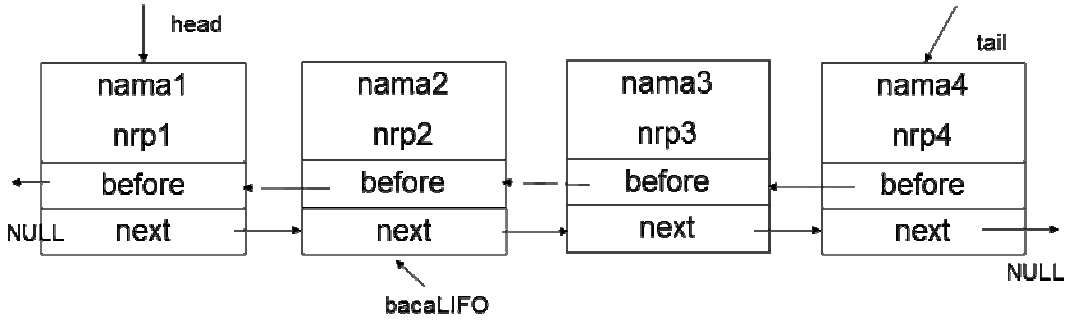
Setelah perintah baris 1



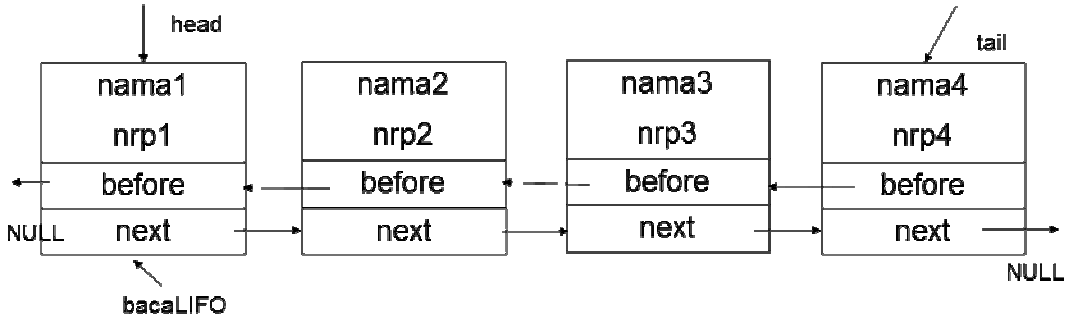
Setelah perintah baris ke-6 iterasi pertama



Setelah perintah baris ke-6 iterasi kedua



Setelah perintah baris ke-6 iterasi ketiga



B.2.3 Mencari Simpul Tertentu

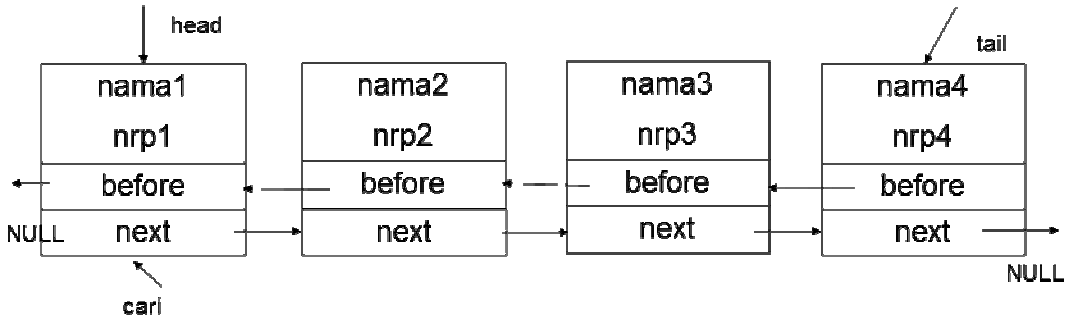
Langkah-langkah untuk mencari simpul tertentu pada linked list yang sudah terbentuk di atas adalah sebagai berikut:

1. Inisialisasi sebuah variabel bertipe `struct simpul* (cari)` dengan `head`
2. Ulangi langkah 3 jika data pada simpul cari tidak sama dengan data yang dicari atau cari tidak sama NULL
3. Jalankan `cari = cari->next`

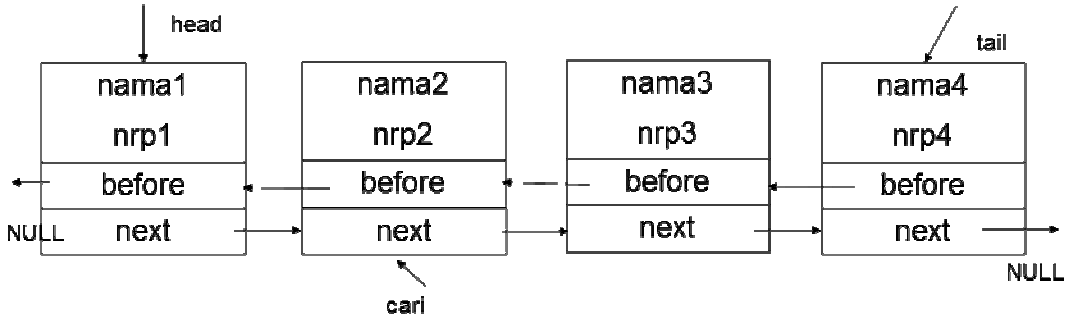
Berikut ini adalah perintah untuk mencari sebuah data pada single linked list

```
1.  cari = head;
2.  while (cari->nama!=nama3)
3.       cari = cari->next;
```

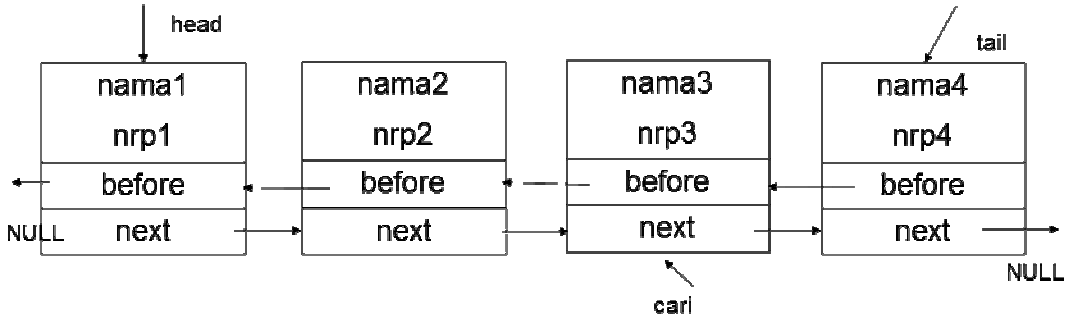
Setelah perintah baris 1



Setelah iterasi ke-2



Setelah iterasi sampai pada data yang dicari



B.2.4 Menghapus Simpul Tertentu (Simpul Depan)

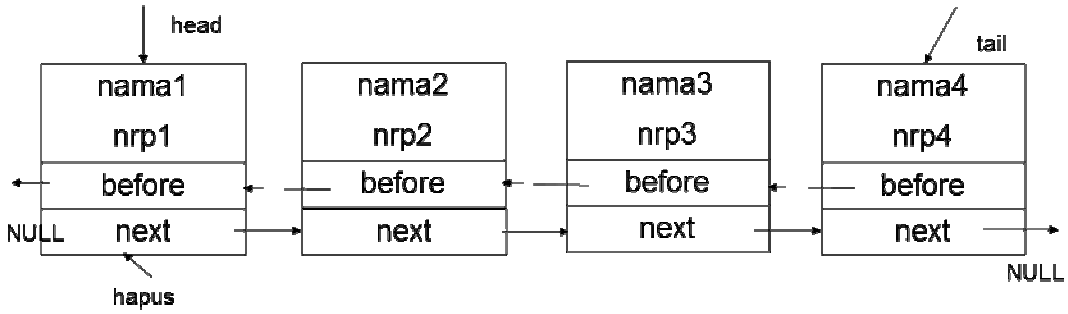
Langkah-langkah untuk menghapus simpul tertentu (depan) dari *double linked list* adalah sebagai berikut:

1. Inisialisasi sebuah variabel bertipe `struct simpul* (hapus)` dengan `head`
2. Jika `hapus->nama` sama dengan `head->nama` lakukan langkah 3-5
3. Arahkan `head` ke `head->next`
4. Arahkan `head->before` ke NULL
5. Bebaskan simpul `hapus`

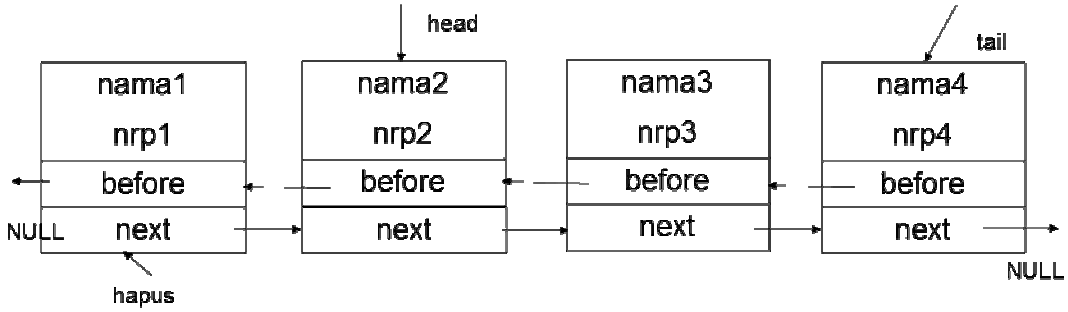
Berikut ini adalah perintah untuk menghapus simpul depan dari *double linked list*:

```
1. hapus = head;
2. if (hapus->nama==head->nama)
3. {
4.     head=head->next;
5.     head->before=NULL;
6.     free(hapus);
7. }
```

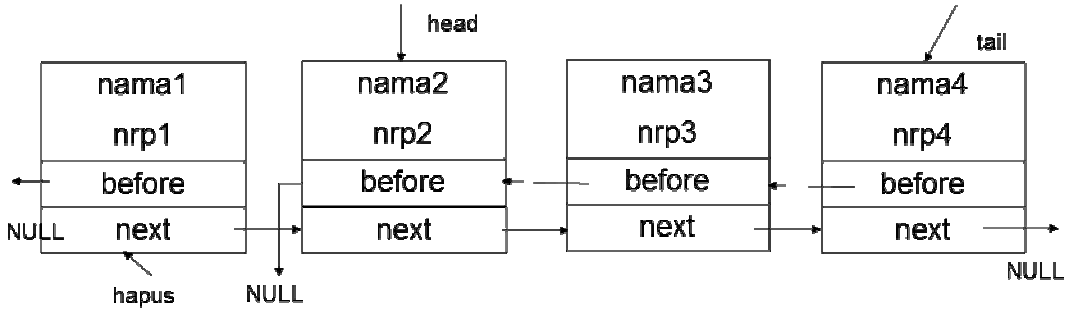
Setelah perintah baris ke-1



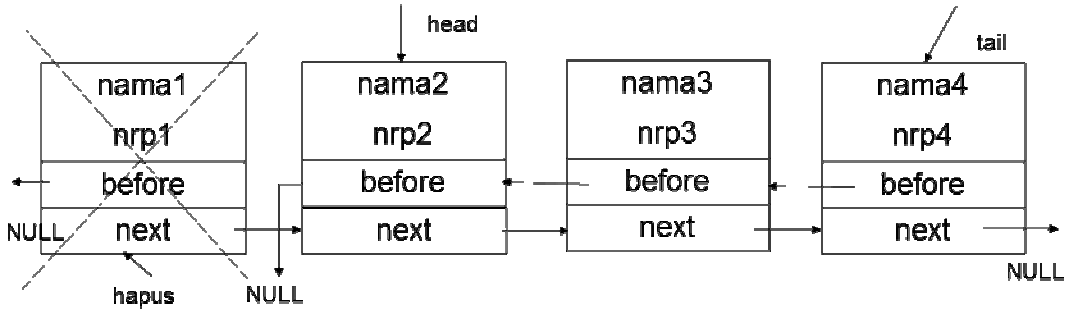
Setelah perintah baris ke-4



Setelah perintah baris ke-5



Setelah perintah baris ke-6



B.2.5 MENGHAPUS SIMPUL TERTENTU (SIMPUL TENGAH)

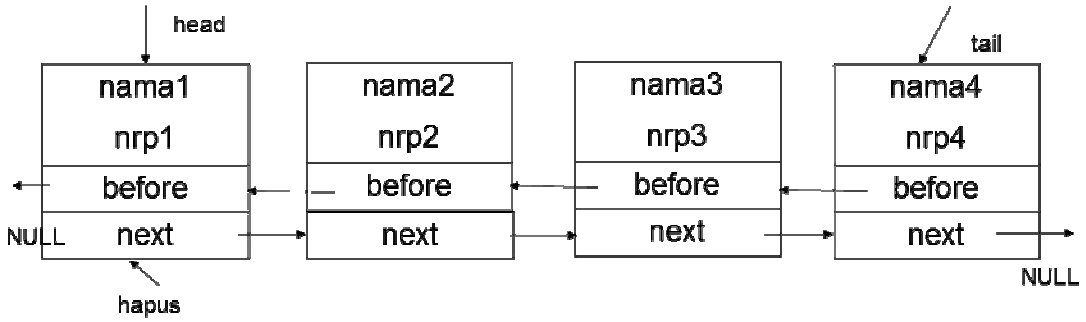
Langkah-langkah untuk menghapus simpul tertentu (belakang) dari *double linked list* adalah sebagai berikut:

1. Inisialisasi sebuah variabel bertipe `struct simpul* (hapus)` dengan `head`
2. Lakukan langkah 3 selama data pada simpul hapus tidak sama dengan data yang dicari
3. Arahkan hapus ke `hapus->next`
4. Arahkan `hapus->before->next` ke `hapus->next`
5. Arahkan `hapus->next->before` ke `hapus->before`
6. Bebaskan simpul `hapus`

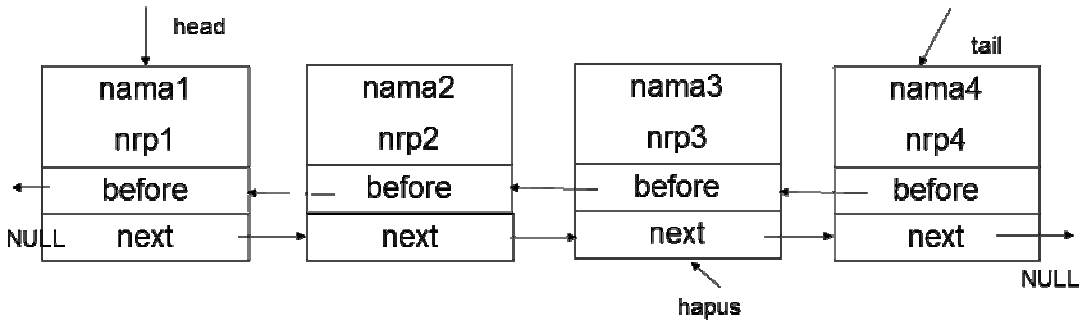
Berikut ini adalah perintah untuk menghapus simpul depan dari *double linked list*:

```
1.  hapus = head;
2.  while (hapus->nama!=nama3)
3.        hapus = hapus -> next;
4.  hapus->before->next=hapus->next;
5.  hapus->next->before=hapus->before;
6.  free(hapus);
```

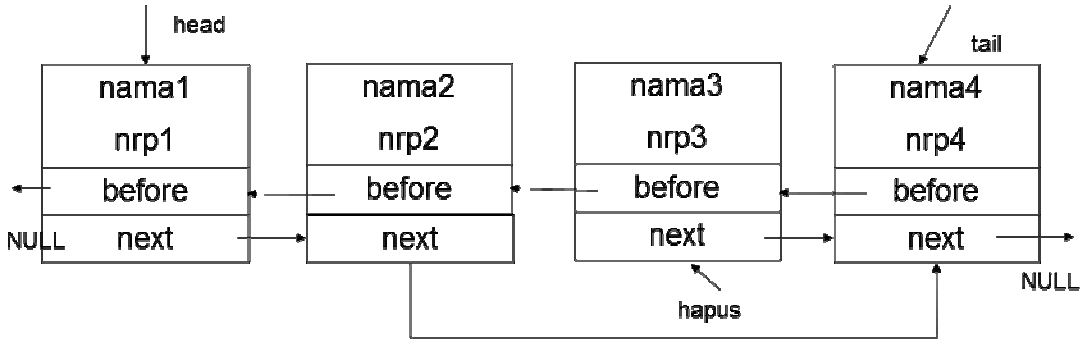
Setelah perintah `hapus = head`



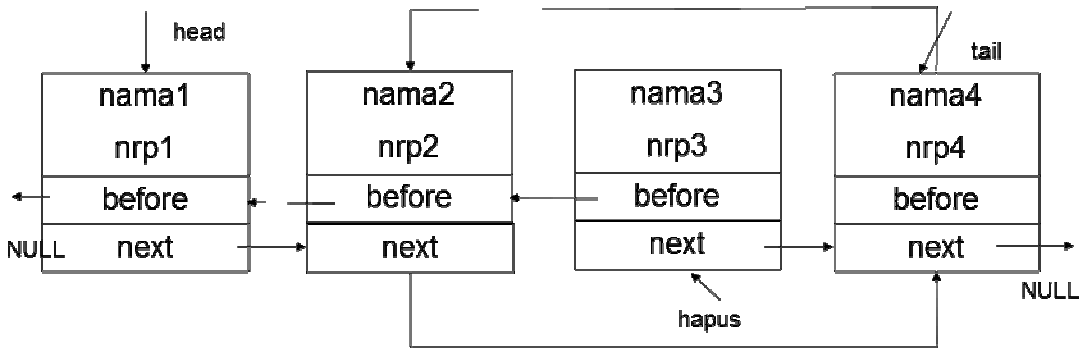
Setelah perintah `hapus = hapus -> next` pada iterasi terakhir



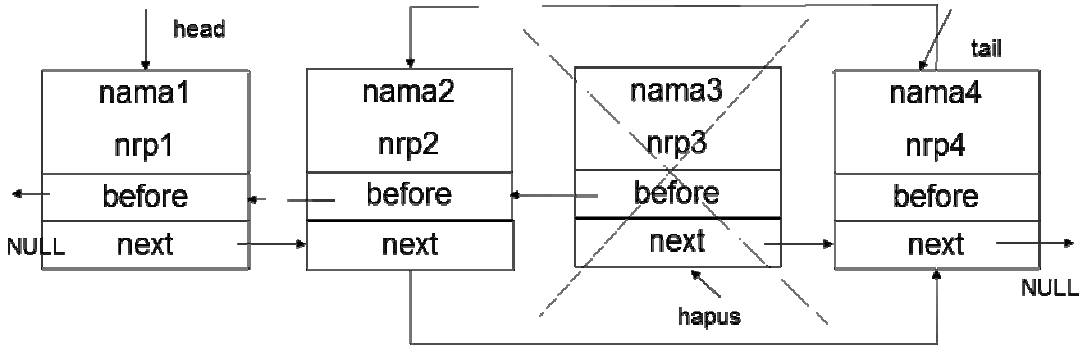
Setelah perintah `hapus->before->next=hapus->next;`



Setelah perintah `hapus->next->before=hapus->before;`



Setelah perintah `free(hapus)`



B.2.6 Menghapus Simpul Tertentu (Simpul Terakhir)

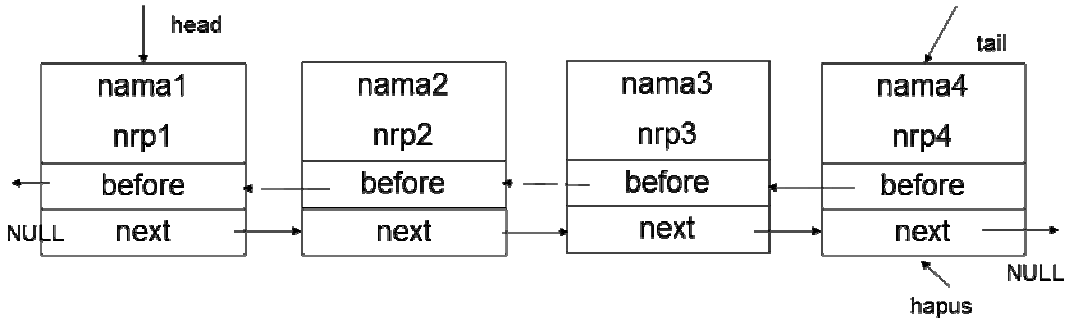
Langkah-langkah untuk menghapus simpul tertentu (depan) dari *double linked list* adalah sebagai berikut:

1. Inisialisasi sebuah variabel bertipe `struct simpul* (hapus)` dengan `head`
2. Lakukan langkah 3 selama data pada simpul hapus tidak sama dengan data yang dicari
3. Arahkan `hapus` ke `hapus->next`
4. Jika `hapus->nama` sama dengan `tail->nama` lakukan langkah 3-5
5. Arahkan `tail` ke `tail->before`
6. Arahkan `tail->next` ke NULL
7. Bebaskan simpul `hapus`

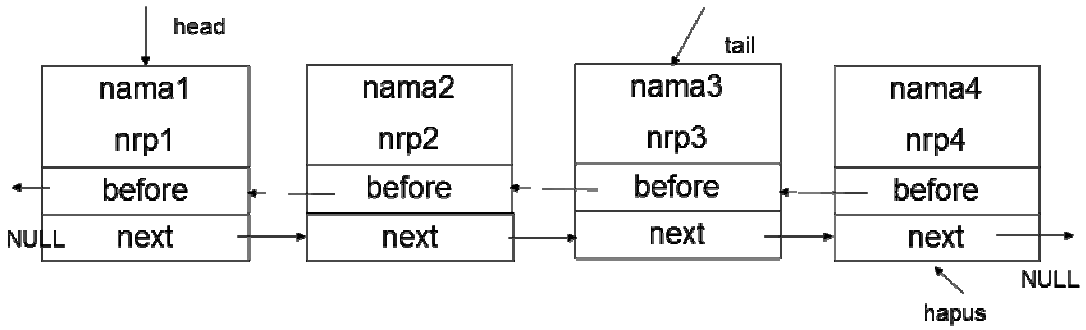
Berikut ini adalah perintah untuk menghapus simpul depan dari *double linked list*:

```
1.    hapus = head;
2.    while (hapus->nama!=nama3)
3.          hapus = hapus -> next;
4.    if (hapus->nama==tail->nama)
5.    {
6.          tail=tail->before;
7.          tail->next=NULL;
8.          free(hapus);
9.    }
```

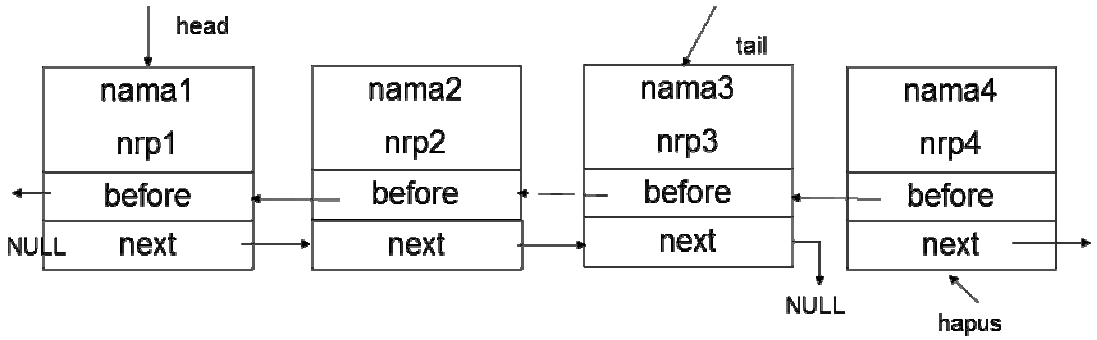
Setelah perintah `hapus = hapus -> next` pada iterasi terakhir



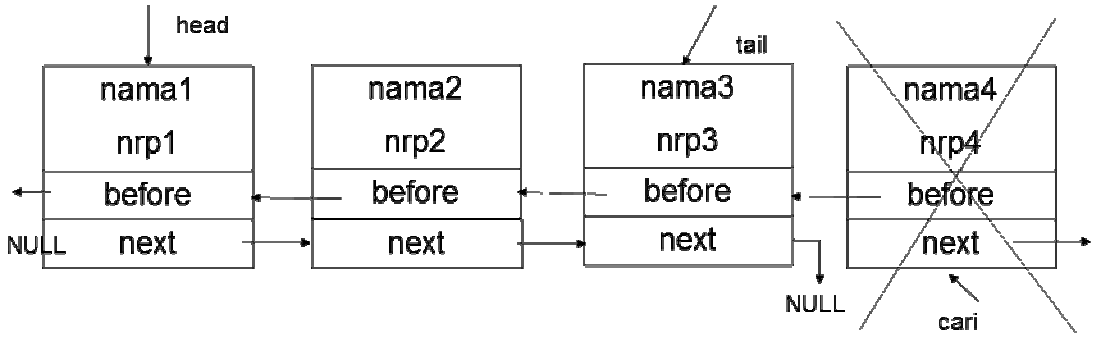
Setelah perintah `tail= tail->before`



Setelah perintah `tail->next = NULL`



Setelah perintah `free(hapus)`



C. TUGAS PENDAHULUAN

Untuk semua operasi dasar single linked list persoalan di bawah ini, desainlah algoritma dan flowchartnya :

1. Menyisipkan sebagai simpul ujung(awal) dari linked list.
2. Membaca atau menampilkan
3. Mencari sebuah simpul tertentu
4. Menghapus simpul tertentu (simpul depan)
5. Menghapus simpul tertentu (simpul di tengah)
6. Menghapus simpul tertentu (simpul terakhir)

D. PERCOBAAN

1. Implementasikan operasi dasar *Double linked list* : Menyisipkan sebagai simpul ujung(awal) dari linked list.
2. Implementasikan operasi dasar *Double linked list* : Membaca atau menampilkan
3. Implementasikan operasi dasar *Double linked list* : Mencari sebuah simpul tertentu. Tambahkan kondisi jika yang dicari adalah data yang paling depan.
4. Implementasikan operasi dasar *Double linked list* : Menghapus simpul tertentu. Tambahkan kondisi jika yang dihapus adalah data yang paling depan atau data yang paling terakhir.
5. Gabungkan semua operasi di atas dalam sebuah Menu Pilihan.

E. LATIHAN

1. Bangunlah *double linked list* di atas adalah *double linked list* dengan prinsip LIFO.
2. Bangunlah sebuah *single linked list* dentan prinsip FIFO(First In First Out)

F. LAPORAN RESMI

1. Kerjakan hasil percobaan(D) dan latihan(E) di atas dan tambahkan analisa.
2. Tuliskan kesimpulan dari percobaan dan latihan yang telah anda lakukan.