BAB II
Senarai Berantai (Linked List)

Tujuan

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

---

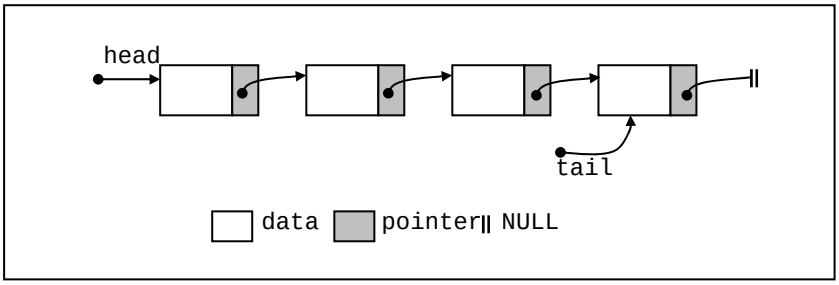
Dalam pemakaian sehari-hari istilah senarai berantai (*linked list*) adalah kumpulan linear sejumlah data. Gambar 2.1 a di bawah ini menunjukkan senarai berantai yang berisi daftar belanjaan, yang berupa barang pertama, kedua, ketiga dan seterusnya. Untuk hari berikutnya, maka daftar tersebut bisa berubah sesuai dengan barang yang harus dibeli lagi atau barang yang tidak perlu dibeli lagi. Gambar 2.1 b menunjukkan daftar belanjaan semula setelah ditambah 3 barang lain dan menghapus 2 barang (dengan mencoret) yang tidak perlu dibeli lagi

2.1 Definisi Linked List

Pengolahan data yang kita lakukan menggunakan komputer seringkali mirip dengan ilustrasi di atas, yang antara lain berupa penyimpanan data dan pengolahan lain dari sekelompok data yang telah terorganisir dalam sebuah urutan tertentu. Salah satu cara untuk menyimpan sekumpulan data yang kita miliki adalah menggunakan larik. Telah kita bicarakan dalam Bab 1, keuntungan dan kerugian pemakaian larik untuk menyimpan sekelompok data yang banyaknya selalu berubah dan tidak diketahui dengan pasti kapan penambahan atau penghapusan akan berakhir.

2.2 Single Linked List

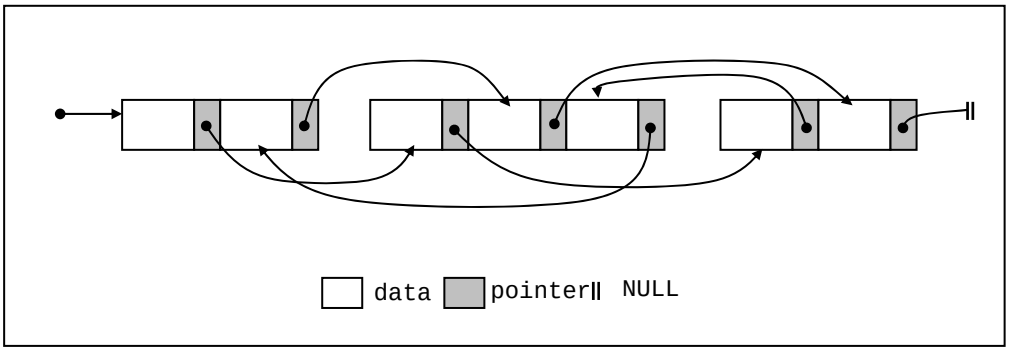
Single linked list atau biasa disebut linked list terdiri dari elemen-elemen individu, dimana masing-masing dihubungkan dengan pointer tunggal. Masing-masing elemen terdiri dari dua bagian, yaitu sebuah data dan sebuah pointer yang disebut dengan pointer *next*. Dengan menggunakan struktur *two-member* seperti ini, linked list dibentuk dengan cara menunjuk pointer *next* suatu elemen ke elemen yang mengikutinya seperti gambar 2.1. Pointer *next* pada elemen terakhir merupakan NULL, yang menunjukkan akhir dari suatu list. Elemen pada awal suatu list disebut *head*, dan elemen terakhir dari suatu list disebut *tail*.



Gambar 2.1 Elemen yang Dihubungkan Bersama Dalam Bentuk Linked List

Untuk mengakses elemen dalam linked list, dimulai dari head dan menggunakan pointer next dari elemen selanjutnya untuk berpindah dari elemen ke elemen berikutnya sampai elemen yang diminta dicapai. Dengan single linke list, list dapat dilintasi hanya satu arah dari head ke tail karena masing-masing elemen tidak terdapat link dengan elemen sebelumnya. Sehingga, apabila kita mulai dari head dan berpindah ke beberapa elemen dan berharap dapat mengakses elemen sebelumnya, kita harus mulai dari head.

Secara konseptual, linked list merupakan deretan elemen yang berdampingan. Akan tetapi, karena elemen-elemen tersebut dialokasikan secara dinamis (menggunakan malloc), sangat penting untuk diingat bahwa kenyataannya, linked list akan terpencar-pencar di memori seperti Gambar 2.2. Pointer dari elemen ke elemen berarti sebagai penjamin bahwa semua elemen dapat diakses.



Gambar 2.2 Elemen Pada Linked List Dihubungkan Secara Terpencar-Pencar pada Alamat Memori

2.2.1 Representasi Simpul (*Node*)

Struktur node pada linked list merupakan suatu simpul(node) yang berisi pointer ke suatu data yang merupakan data dirinya sendiri. Model struktur dari linked list tersebut dalam C adalah sebagai berikut:

```
typedef struct node *list;
struct node {
      int datalist;
      struct node *next;
};
```

dilanjutkan dengan deklarasi dari pointer ke struktur di atas sebagai berikut:

```
struct node *head;
```

atau

```
list head;
```

2.2.2 Alokasi Simpul

Ketika sebuah variabel dideklarasikan, terlebih dahulu harus diinisialisasi. Demikian juga dengan pengalokasian secara dinamis. Sehingga, fungsi untuk mengalokasikan sebuah node baru, fungsi `allocate_node()` 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 `allocate_node` menghasilkan 0, bila berhasil maka menghasilkan 1. Untuk membebaskan node digunakan fungsi free_node. Fungsi dari alokasi node adalah sebagai berikut :

```
int allocate_node(int data, list *new)
{
      new = (list) malloc (sizeof(node));
      if(new==NULL)
            return 0;

      new->datalist = data;
```

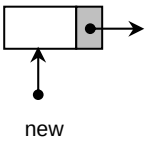
```
      new->next=NULL;
      return 1;
}
```

Untuk inisialisasi list setelah alokasi untuk node pertama maka ditambahkan statement sebagai berikut:

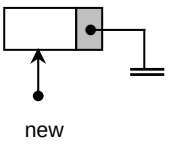
```
head = new;
```

Ilustrasi dari fungsi i `allocate_node()` adalah sebagai berikut

```
      new = (list) malloc (sizeof(node));
      new->datalist = data;
```

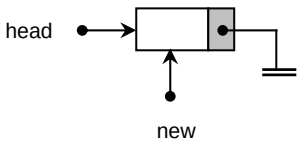



```
      new->next=NULL;
```



Untuk inisialisasi list setelah alokasi untuk node pertama ilustrasinya adalah sebagai berikut:

```
      head = new;
```



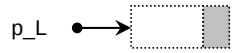
Fungsi `free_node()`menggunakan memori dinamis dengan fungsi `free()`. `free()` akan menghancurkan node yang menunjuk ke pointer yang dilewati, sehingga tempat yang ada dapat dipakai untuk lainnya. Akan tetapi, pointer yang melewati `free()` tidak otomatis menjadi null, tetapi akan menunjuk ke node yang sudah tidak ada. Karena itu pointer harus didefinisikan dengan `NULL`.

```
void free_node(list p_L)
{
      free(p_L);
```

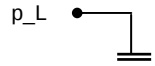
```
	p_L=NULL;
}
```

Ilustrasi dari fungsi `free_node()`dapat dilihat pada gambar berikut :

```
	free(*p_L);
```




```
	*p_L=NULL;
```



2.2.3 Operasi pada Linked List

Pada sub bab ini akan dijelaskan mengenai operasi yang terpenting pada linked list, yaitu menambahkan node (insert) dan menghapus node (delete).

2.2.3.1 Insert

Fungsi insert pada linked list meliputi :

- insert sebagai node awal (head) dari linked list
- insert setelah node tertentu
- insert sebelum node tertentu
- insert sebagai node akhir (tail) dari linked list

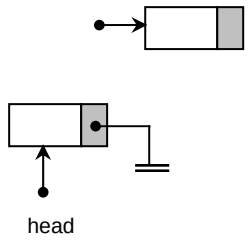
Insert sebagai node awal (head) dari linked list

Statement kelanjutan dengan deklarasi seperti diatas untuk insert sebagai node awal (head) dari linked list adalah sebagai berikut:

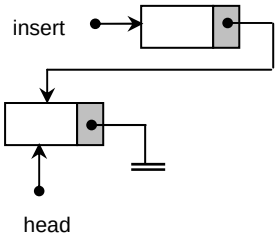
```
void insertashead(list insert)
{
	insert->next=head;
	head = insert;
}
```

ilustrasi dari fungsi diatas adalah sebagai berikut:

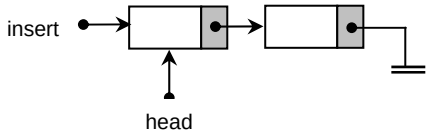
kondisi awal




```
insert->next=head;
```




```
head = insert;
```



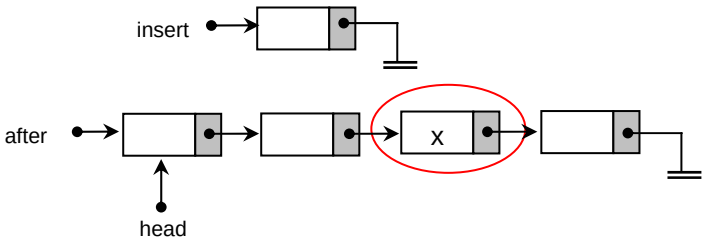
Insert setelah node tertentu

Statement untuk insert setelah node tertentu dari linked list adalah sebagai berikut:

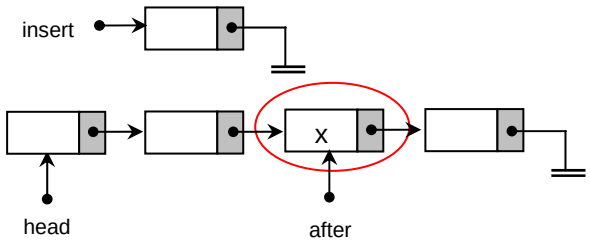
```
void insertafternode(int x, list insert)
{
      list after;
      after = head;
      do
            after = after->next;
      while (after->datalist != x);
      insert->next = after->next;
      after->next = insert;
}
```

ilustrasi dari fungsi diatas adalah sebagai berikut:

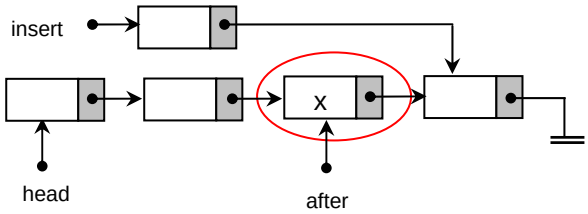
```
after = head;
```



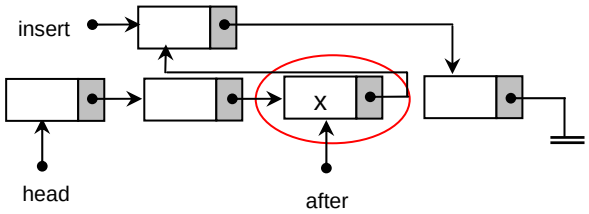
```
do
      after = after->next;
while (after->datalist != x);
```




```
insert->next = after->next;
```




```
after->next = insert;
```



Langkah-langkah untuk proses di atas adalah sebagai berikut:

1. Pointer next dari elemen baru menunjuk elemen setelah elemen tertentu
2. Pointer next elemen sebelumnya menunjuk ke elemen baru

Insert sebelum node tertentu

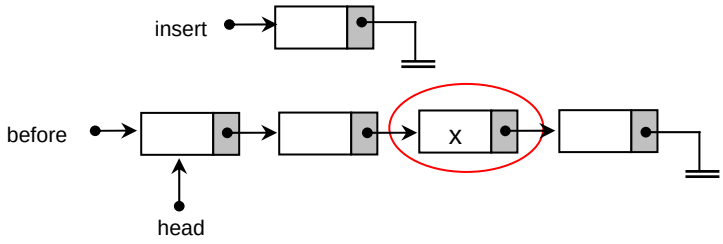
Statement untuk insert setelah node tertentu dari linked list adalah sebagai berikut:

```
void insertbeforenode(int x, list insert)
```

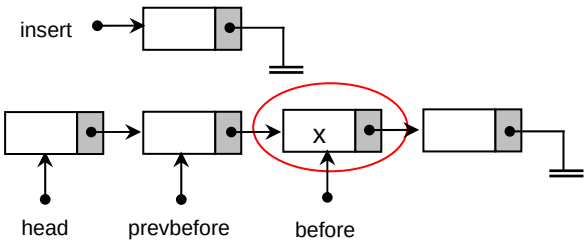
```
{
      list before, prevbefore;
      if (head->datalist = x)
            insertashead(insert)
      else
      {
            before = head;
            do
                  prevbefore = before;
                  before = before->next;
            while (before->datalist != x);
            insert->next = before;
            prevbefore->next = insert;
      }
}
```

ilustrasi dari fungsi diatas adalah sebagai berikut:

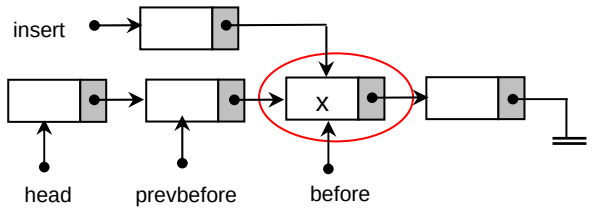
```
before = head;
```



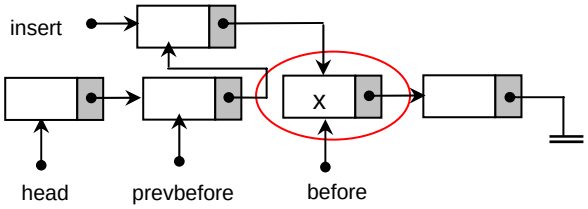

```
do
      prevbefore = before;
      before = before->next;
while (before->datalist != x);
```




```
insert->next = before;
```




```
prevbefore->next = insert;
```



Langkah-langkah untuk proses di atas adalah sebagai berikut:

1. Telusuri list sampai elemen tertentu, catat juga elemen sebelumnya
2. Lakukan penyisipan sebelum elemen tertentu tersebut

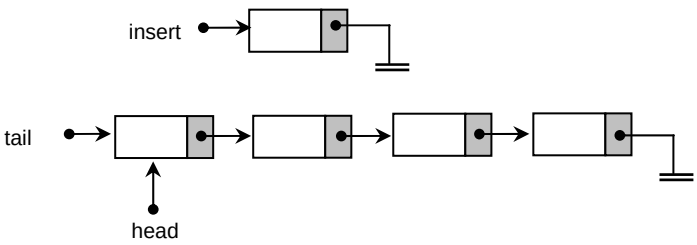
Insert di akhir (tail) dari linked list

Statement untuk insert di akhir dari list adalah sebagai berikut:

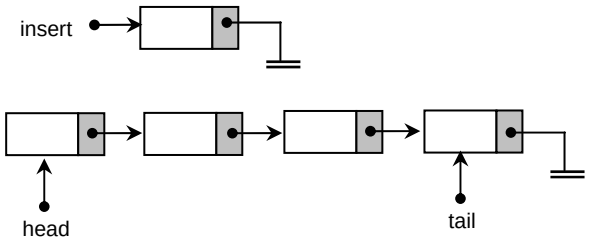
```
void insertastail(list insert)
{
      list tail;
      tail = head;
      do
            tail = tail->next;
      while (tail->next != NULL);
      tail->next = insert;
      tail = tail->next;
}
```

ilustrasi dari fungsi diatas adalah sebagai berikut:

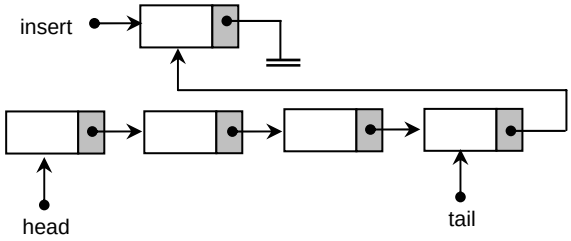
```
tail = head;
```



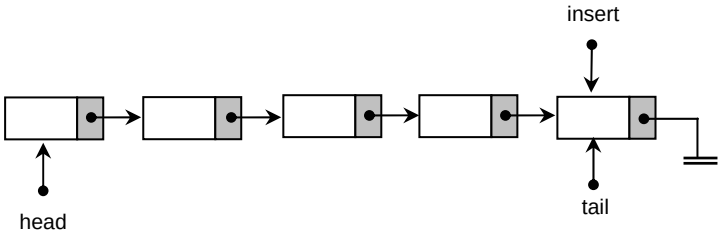
```
do
      tail = tail->next;
while (tail->next != NULL);
```




```
tail->next = insert;
```




```
tail = tail->next;
```



Langkah-langkah untuk proses di atas adalah sebagai berikut:

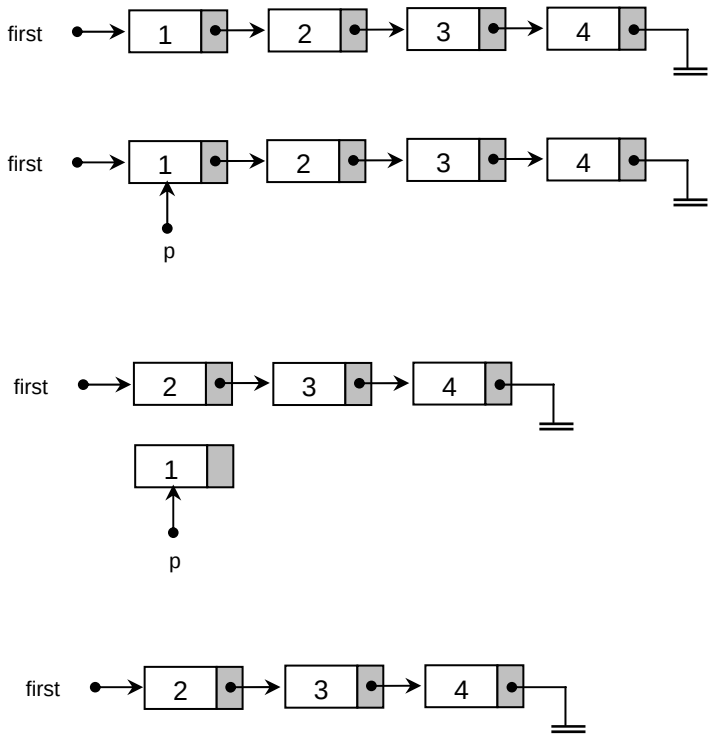
1. Telusuri list sampai elemen terakhir (tail->next=NULL)
2. Lakukan pentisipan setelah elemen terakhir

2.2.3.2 Delete

Fungsi delete pada linked list meliputi :

- delete sebagai simpul pertama(head) dari linked list
- delete setelah simpul tertentu
- delete simpul terakhir

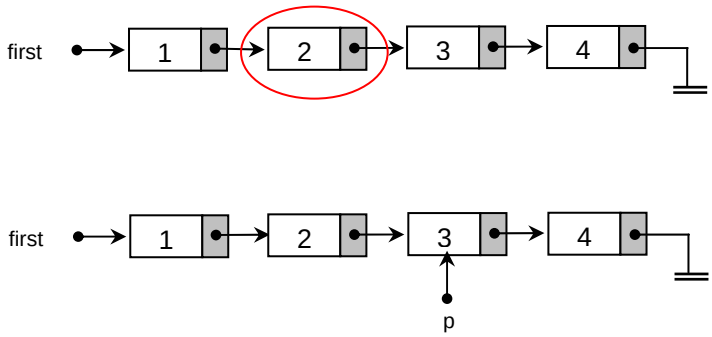
Penghapusan Simpul Pertama:

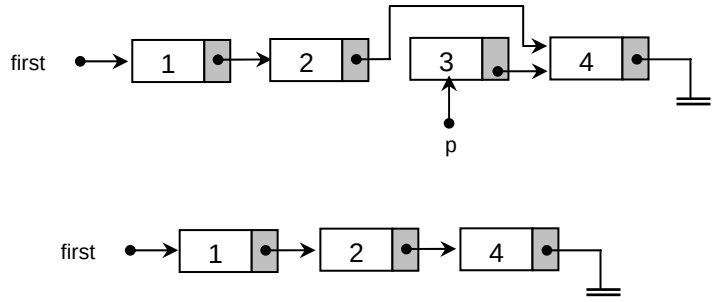


Langkah-langkah untuk proses di atas adalah sebagai berikut:

1. Pointer first diarahkan pada data ke-2
2. Pointer p diarahkan pada data ke-1
3. Bebaskan pointer p (secara otomatis data ke-1 terhapus)

Penghapusan Setelah Simpul Tertentu

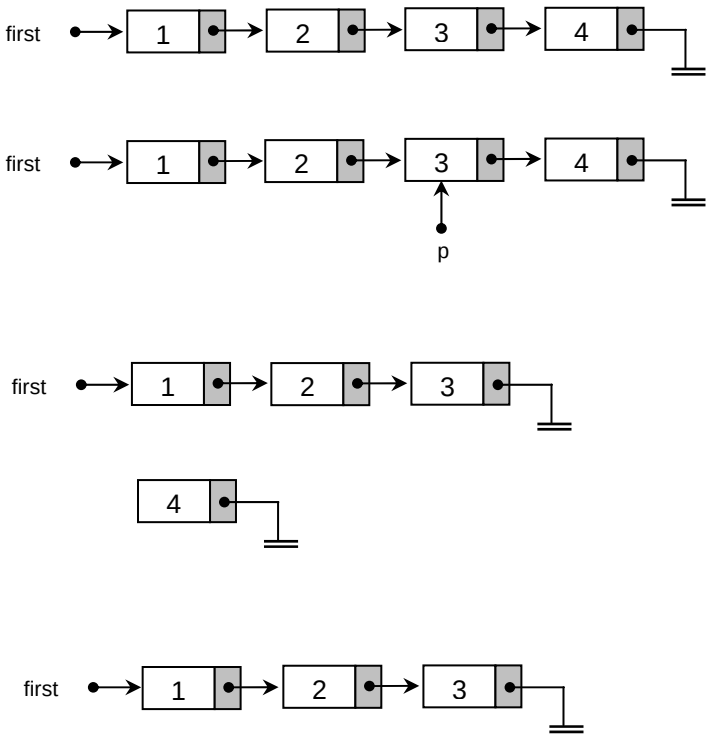




Langkah-langkah untuk proses di atas adalah sebagai berikut:

1. Arahkan pointer first s/d data yang ditunjuk
2. Pointer p diarahkan pada first->next
3. Arahkan pointer first->next pada p->next
4. Bebaskan pointer p (secara otomatis data setelah simpul tertentu terhapus)

Penghapusan Simpul Terakhir



Langkah-langkah untuk proses di atas adalah sebagai berikut:

1. Telusuri simpul s/d first->next = NULL
2. Arahkan pointer p ke first
3. Bebaskan pointer p->next (Simpul Terakhir)

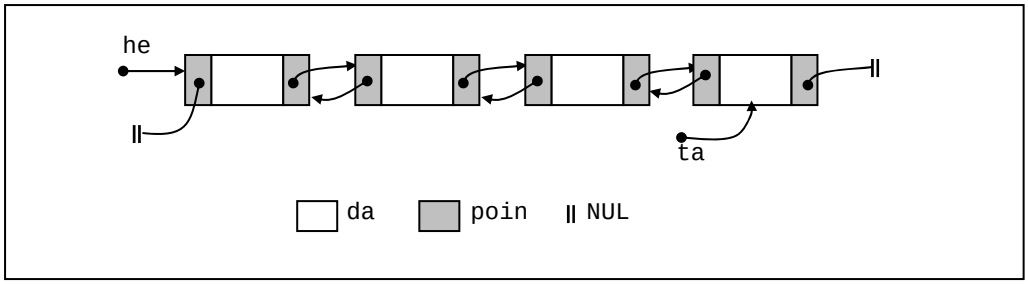
4. Arahkan p->next ke NULL

2.3 Double Linked List

Elemen-elemen dihubungkan dengan dua pointer dalam satu elemen. Struktur ini menyebabkan list melintas baik ke depan maupun ke belakang.

Masing-masing elemen pada double linked list terdiri dari tiga bagian, disamping data 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 2.3



Gambar 2.3 Elemen yang Dihubungkan dalam Bentuk *Double Linked List*

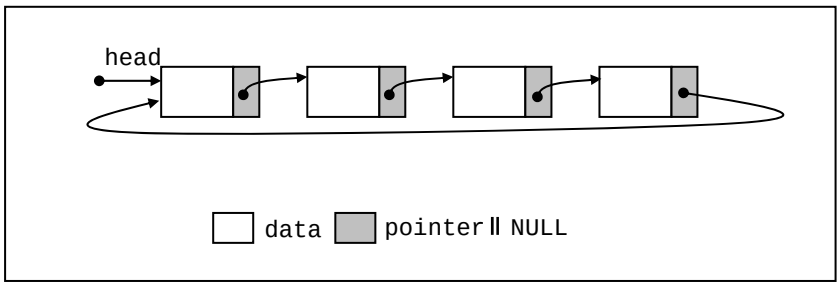
Untuk melintas kembali melalui *double linked list*, kita gunakan pointer *prev* dari elemen yang berurutan pada arah *tail* ke *head*. *Double linked list* mempunyai fleksibilitas yang lebih tinggi daripada *single linked list* dalam perpindahan pada list. Bentuk ini sangat berguna ketika akan meletakkan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

2.4 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 2.4 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 2.4 Single Linked Circular List

2.5 Kesimpulan

1. Linked list adalah sebuah struktur untuk menyimpan data yang bersifat dinamik
2. Beberapa operasi dapat diterapkan pada linked list seperti sisip(insert), hapus(delete)
3. Operasi-operasi yang ada pada linked list relatif lebih sulit jika dibandingkan dengan operasi-operasi pada struktur yang statis

2.6 Latihan

1. Buatlah linked list untuk data mahasiswa, tambahkan prosedur untuk menambah dan menghapus data
2. Tambahkan tampilan di output pada program di atas untuk menghitung nilai rata-rata, dimana

   nrata=total/jumlah_siswa;

   total didapatkan dari menambahkan nilai yang didapat tiap mahasiswa