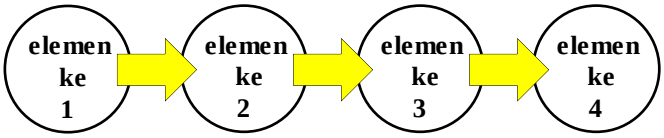
III. STRUKTUR LINIER

3.1. KONSEP DASAR.

Struktur Linier (atau sering juga disebut List Linier) terjadi karena hubungan antar "*elemen*" yang seakan akan membentuk sebuah "*penggal garis*" tanpa cabang. Istilah "*elemen*" sering mendapat sebutan lain meskipun maksudnya sama saja, seperti : "*node*" (simpul), "*bead*", "*record*", "*item*" dsb.
Abstraksi terhadap suatu himpunan data dapat "*menemukan*" munculnya struktur linier seperti misalnya :

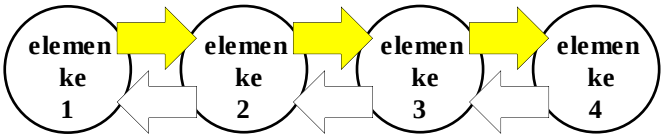
elemen ke 1 hanya mempunyai "hubungan" dengan elemen ke 2, elemen ke 2 mempunyai hubungan yang sama dengan elemen ke 3 dst. Jika hubungan antar elemen tersebut diabstraksikan, maka muncul gambaran sebuah *penggal garis* yang berawal dari 1 dan berakhir di 4.



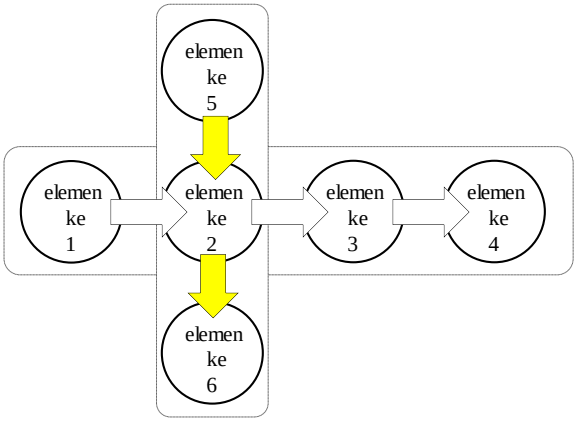
"Hubungan" disini mempunyai semacam "*arah*".

Catatan :
HUBUNGAN dalam kasus diatas janganlah diartikan bahwa ada POINTER yang menunjuk dari suatu elemen ke elemen lain, akan tetapi ditafsirkan sebagai sesuatu yang benar benar abstrak.

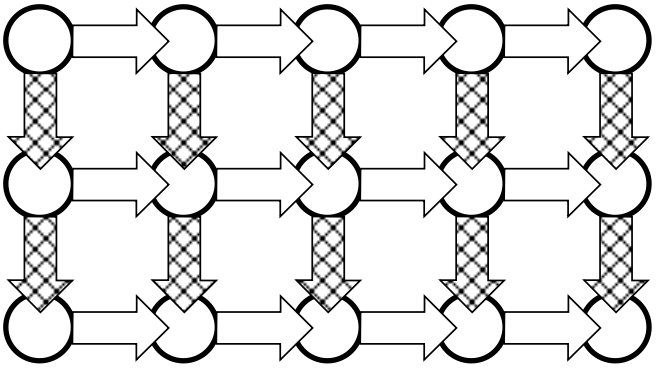
Dapat pula terjadi 2 macam "hubungan" yang saling berlawanan arah seperti terlihat pada gambar.
Perhatikan bahwa secara keseluruhan bentuk abstraksi tidak kehilangan sifat garisnya.



Atau bahkan bentuk abstraksi seperti pada gambar disamping ini dimana terlihat adanya 2 buah struktur linier yang kebetulan salah satu anggautanya sama, ialah elemen ke 2.
Struktur linier pertama beranggautakan elemen ke 1, 2, 3 dan 4; sedangkan struktur linier kedua beranggautakan elemen ke 5, 2 dan 6.



Struktur linier dapat pula dijumpai dalam bentuk 2 dimensi atau lebih, seperti sbb :



Struktur semacam ini sering disebut sebagai "LIST ORTHOGONAL".

3.2. JENIS STRUKTUR LINIER.

Struktur linier dapat dibedakan dari berbagai sudut pandang. Ada pembedaan yang bersifat konsep, ada juga yang bersifat representatif. Salah satu pembedaan yang bersifat konseptual ialah dari karakteristik proses ***insertion*** (penambahan anggauta baru) dan ***deletion*** (pemecatan anggauta) nya. Dari sudut pandang ini dibedakan jenis sbb :

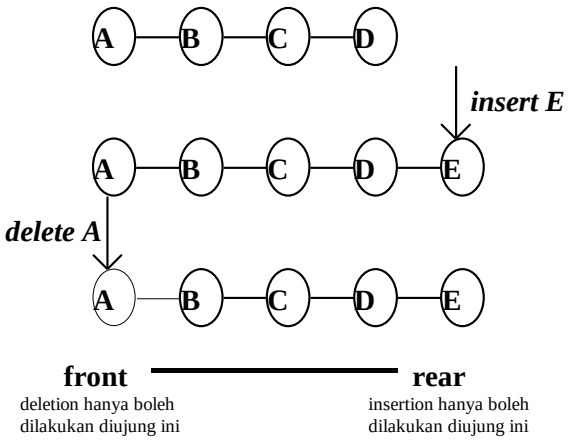
- ***Queue*** (atau Antrian)
- ***Stack***
- ***Deque*** (disingkat dari double ended queue)
- ***Non-restricted linear-list*** (List linier bebas/sembarang)

QUEUE

Setiap ujung dari suatu queue (ingat : hanya ada 2 buah ujung) hanya boleh mengalami 1 jenis proses saja. Jika pada ujung pertama hanya boleh dilakukan *deletion*, maka pada ujung kedua hanya boleh dilakukan *insertion*. Jadi, deletion dan insertion selalu terjadi pada "ujung" yang berbeda.
Sebagai ilustrasi :

misalnya mula mula anggauta sebuah queue adalah 4 buah elemen (node), sebut saja sebagai A, B, C dan D.
Kemudian terjadi insertion elemen baru E, terjadi pada ujung kanan. Maka angguta queue sekarang menjadi 5 buah node dengan bentuk seperti pada gambar.

Setelah itu misalnya akan dilakukan deletion, maka harus dilakukan pada ujung kiri. Maka node A akan dipecat, sehingga anggauta queue sekarang berturut turut adalah B, C, D dan E.



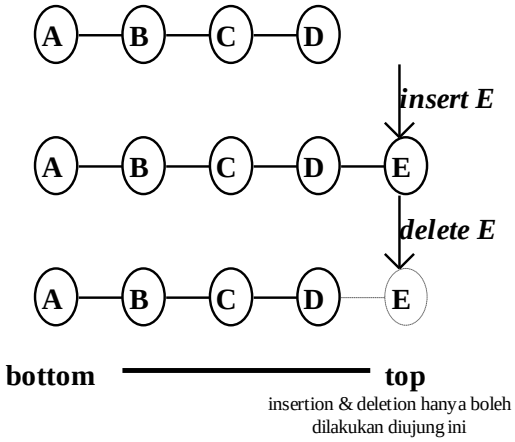
Perhatikan bahwa sifat insertion dan deletion seperti diatas akan menyebabkan node yang pertama masuk menjadi anggauta akan menjadi node yang pertama dipecat dari keanggautaan. Sifat ini sering disebut sebagai "yang pertama masuk adalah yang pertama keluar", atau *FIFO* (First In First Out).
Ujung yang mengalami deletion (pada ilustrasi berarti ujung kiri) sering disebut *FRONT*, dan ujung yang mengalami insertion disebut *REAR*.
Struktur data ini banyak dipakai dalam informatika, misalnya untuk merepresentasikan :

- Antrian job yang harus ditangani oleh sistem operasi;
- Antrian dalam dunia nyata;

STACK

Pada sebuah stack, INSERTION dan DELETION hanya boleh dilakukan pada satu ujung yang sama.
Sebagai ilustrasi :

Misalnya mula mula isi stack adalah 4 buah node, ialah : A, B, C dan D.
Kemudian dilakukan insertion node E pada ujung kanan, sehingga anggauta stack adalah : A, B, C, D dan E.
Berikutnya jika akan dilakukan deletion, maka harus dilakukan pada ujung yang sama dengan ujung yang mengalami insertion, ialah ujung kanan. Dengan demikian E akan didelete, sehingga bentuk stack menjadi seperti pada keadaan awal.
Terlihat bahwa insertion dan deletion hanya boleh terjadi pada ujung kanan saja, ujung ini disebut sebagai *TOP*, sedangkan ujung kiri disebut *BOTTOM*.



Stack merupakan Linier list yang :

- Dikenali elemen puncaknya (TOP)
- Aturan penyisipan dan penghapusan elemennya tertentu, yaitu :
    1. Penyisipan selalu dilakukan “di atas” TOP, melalui operasi *PUSH*
    2. Penghapusan selalu dilakukan di TOP, melalui operasi *POP*

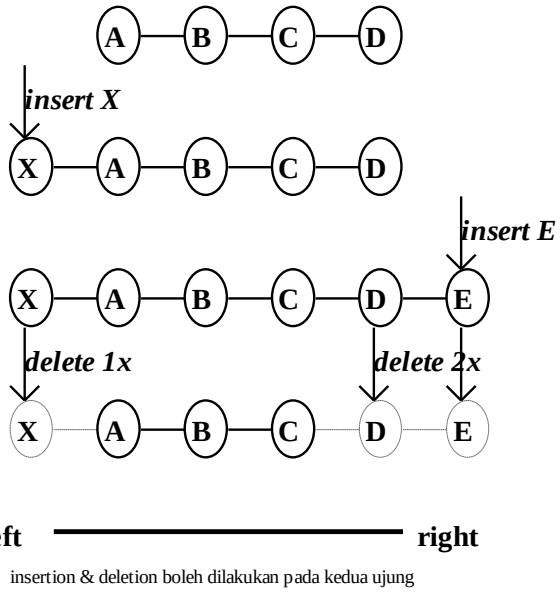
Struktur data ini banyak dipakai dalam informatika, misalnya untuk merepresentasikan :

- Pemanggilan prosedur;
- Perhitungan ekspresi aritmatika;
- Rekursifitas;
- Backtracking;
- dan Algoritma lanjut lainnya.

DEQUE

Pada list linier jenis ini insertion dan deletion boleh terjadi pada ujung yang manapun.
Sebagai ilustrasi :

Misalnya mula mula deque terdiri dari 4 elemen berturut turut : A, B, C dan D.
Kemudian dilakukan insertion X pada *ujung kiri*, sehingga deque mempunyai 5 anggauta berturut turut : X, A, B, C dan D.
Berikutnya dilakukan insertion E pada *ujung kanan*, sehingga anggauta deque : X, A, B, C, D dan E.
Misalkan kemudian dilakukan 2 x deletion pada ujung kanan, diikuti oleh 1 x deletion pada ujung kiri, maka akhirnya deque akan terdiri dari 3 node : A, B dan C.
Pada contoh ini terlihat bahwa kedua ujung deque, yang umum disebut sebagai *left* dan *front*, boleh mengalami insertion dan deletion.

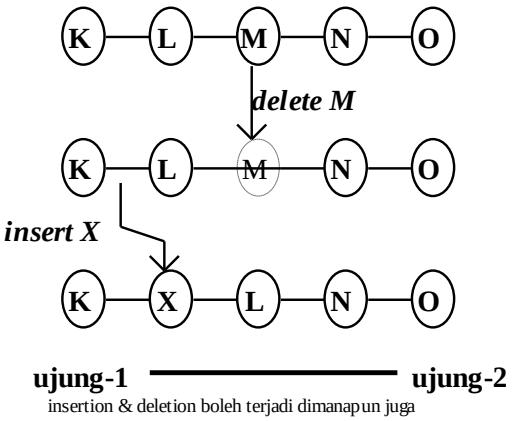


insertion & deletion boleh dilakukan pada kedua ujung

LIST LINIER BEBAS/ SEMBARANG

Nama yang lebih resminya ialah "NON RESTRICTED LINIER LIST", karena tidak ada larangan atau pembatasan mengenai insertion dan deletion. Kedua jenis proses *dapat terjadi dimanapun juga*, baik dikedua ujung maupun di tengah list.
Sebagai contoh :

Misalnya mula mula isi list berturut turut : K, L, M, N dan O.
Dapat terjadi deletion suatu anggauta ditengah list tersebut, misalnya saja M, sehingga isi list sekarang adalah : K, L, N dan O.
Kemudian boleh pula terjadi insertion ditengah, misalnya insertion X diantara K dan L, sehingga keadaan akhirnya adalah : K, X, L, N dan O.



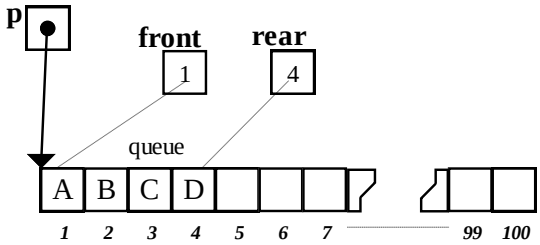
insertion & deletion boleh terjadi dimanapun juga

3.3. ALOKASI SEQUENTIAL UNTUK STRUKTUR LINIER.

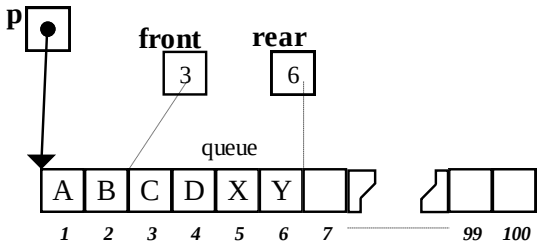
Alokasi sequential dilakukan dengan jalan memesan suatu ruang memori yang tak terputus putus (contiguous) dengan panjang tertentu.

Alokasi Sequential untuk QUEUE

Misalnya alokasi sequential mulai dari lokasi yang ditunjuk oleh pointer p dst sepanjang 100 satuan. Dan misalnya pula pada keadaan awal queue sudah berisi 4 elemen, masing masing A, B, C, dan D.
Diketahui pula 2 integer *rear* dan *front* yang berisi nilai "*offset*" posisi queue dari pointer p.



Jika terjadi berturut turut 2 x deletion (A dan B) dan 2 x insertion (X dan Y), maka keadaan queue dapat direpresentasikan oleh perubahan nilai variabel front dan rear seperti pada gambar.
Terlihat bahwa posisi queue (sekarang dari C sampai dengan Y) telah bergeser kekanan pada medan yang dialokasikan.



Jika hal ini terus berlangsung maka lama kelamaan ujung rear (yang diwakili oleh nilai rear) dari queue tsb akan sampai ke lokasi bernomor 100, yang merupakan batas kanan dari medan alokasi.
Queue tidak bisa mengalami insertion lagi, dan terlihat bahwa ada ruang memori (dari nomor 1 s/d posisi yang diwakili oleh nilai var front) yang sia sia.

Untuk mengatasi hal tersebut diatas maka perlu dilakukan penambahan ketentuan sbb :

> dianggap bahwa alokasi sequential tsb *MELINGKAR*, sehingga jika rear sudah sampai di posisi 100 dan ada insertion, maka elemen terbaru akan dimasukkan ke elemen ke 1 (rear akan menunjuk ke posisi 1).

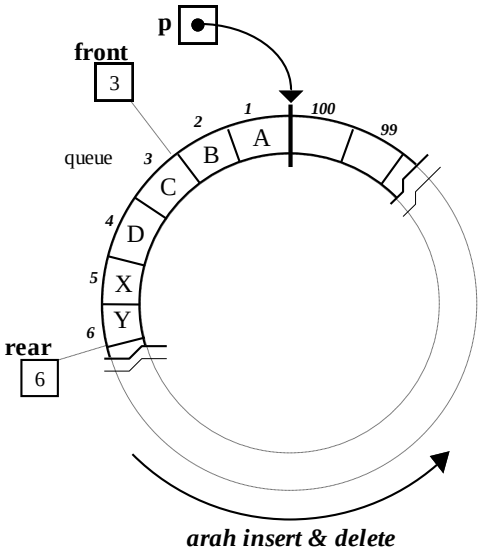
Cara ini disebut "*CIRCULAR SEQUENTIAL LIST*".

Perhatikan bahwa :

Jika terjadi *insertion* terus menerus, mungkin saja akhirnya panjang queue akan sama dengan jumlah ruang yang di alokasikan (atau rear akan tepat berada "*sebelum*" front sesuai arah putar "*berlawanan jarum jam*"). Jika ternyata masih ada insertion lagi, maka terjadi situasi yang disebut *OVERFLOW*.

Demikian juga sebaliknya untuk *deletion* pada saat queue sudah kosong, terjadi situasi *UNDERFLOW*.

Pada anggapan circular ini ada alternatif lain pemberian nilai front dan rear. Perbedaannya adalah bahwa nilai front diisi dengan nilai elemen "sebelum" elemen terakhir. Maka untuk contoh pada gambar disamping : front = 2.



Bentuk Algoritma Insertion & Deletion.

Bentuk algoritma akan sangat tergantung dari asumsi yang diambil mengenai nilai variabel rear dan front, dan juga kondisi "queue kosong" dan "queue penuh". Dapat diambil beberapa cara asumsi yang agak berbeda, sebagai contoh perhatikan 2 cara sbb :

Cara ke 1 : (sama dengan cara seperti pada ilustrasi gambar diatas)

isi variabel rear	≡	nomor elemen pada "ujung rear"
isi variabel front	≡	nomor elemen pada "ujung front"
kondisi queue "kosong"	≡	**(rear=0) and (front=0)**
kondisi queue "penuh"	≡	**(rear=front - 1) or ((rear=panjang_max) and (front=1))**

Cara ke 2 : idem cara ke 1 kecuali bahwa

kondisi queue "kosong"	≡	**(rear=**"*nilai sembarang#0*"**) and (front=0)**

Cara ke 3 :

isi variabel rear	≡	nomor elemen pada "ujung rear"
isi variabel front	≡	nomor elemen di "depan" dari "ujung front"
kondisi queue "kosong"	≡	**(rear=**"*nilai sembarang#0*"**) and (front=0)**
kondisi queue "penuh"	≡	**rear=front**

Untuk setiap cara dapat disusun algoritma insertion & deletion yang sesuai, sebagai contoh misalnya untuk asumsi seperti pada cara ke 1 maka bentuk algoritmanya adalah sbb :

INSERTION :

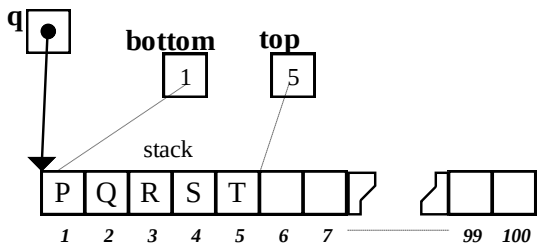
```
bantu ← rear
if ( rear = panjang_max )
| then  rear ← 1
| else  rear ← rear + 1
endif
if ( rear = front )
| then  {OVERFLOW, penuh}
|       tolak insertion/restore nilai rear....
|       rear ← bantu
| else  if ( front = 0 )
|       | then  {tadinya queue kosong}
|       |       front ← 1
|       | else
|       endif
|       {isi data pada posisi p+rear}
endif
```

DELETION :

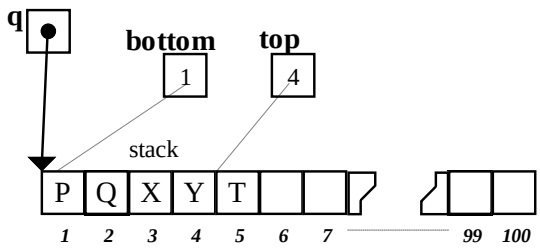
```
if ( rear=0 )
| then  {UNDERFLOW,queue masih kosong}
|       tolak deletion....
| else  if ( front = rear )
|       | then  {tadinya tinggal satu}
|       |       rear ← 0
|       |       front ← 0
|       | else  if (front = panjang_max)
|       |       | then  front ← 1
|       |       | else  front ← front + 1
|       |       endif
|       endif
endif
```

Alokasi Sequential untuk STACK

Misalnya alokasi sequential mulai dari lokasi yang ditunjuk oleh pointer *q* dst sepanjang 100 satuan. Dan misalnya pula pada keadaan awal stack sudah berisi 4 elemen, masing masing *P*, *Q* *R* , *S*dan *T*. Diketahui pula 2 integer *bottom* dan *top* mewakili posisi stack tersebut.



Jika terjadi berturut turut 3 x deletion (T, S dan R) serta 2 x insertion (X dan Y), maka bentuk stack akan seperti pada gambar disamping ini.
Terlihat bahwa posisi stack sekarang dari P sampai dengan Y. Terlihat bahwa ujung bottom tidak terganggu oleh delete/ insert. Jadi dalam hal ini *TIDAK PERLU* dilakukan penganggapan sebagai *CIRCULAR LIST*.



Akan tetapi : kemungkinan *OVERFLOW* dan *UNDERFLOW* tetap ada ! Perhatikan bahwa :

- Jika terjadi insertion terus menerus maka pointer t akan menunjuk ke posisi 100. Jika ternyata masih terjadi insertion lagi maka diperlukan tambahan alokasi, kondisi ini disebut *OVERFLOW*.
- Sebaliknya jika terjadi deletion terus menerus, juga akan terjadi *UNDERFLOW*.

Bentuk Algoritma Insertion & Deletion.
Asumsi yang diambil adalah :

stack "penuh" ≡ **top = panjang_max**
stack "kosong" ≡ **top = 0**

Operasi INSERTION :

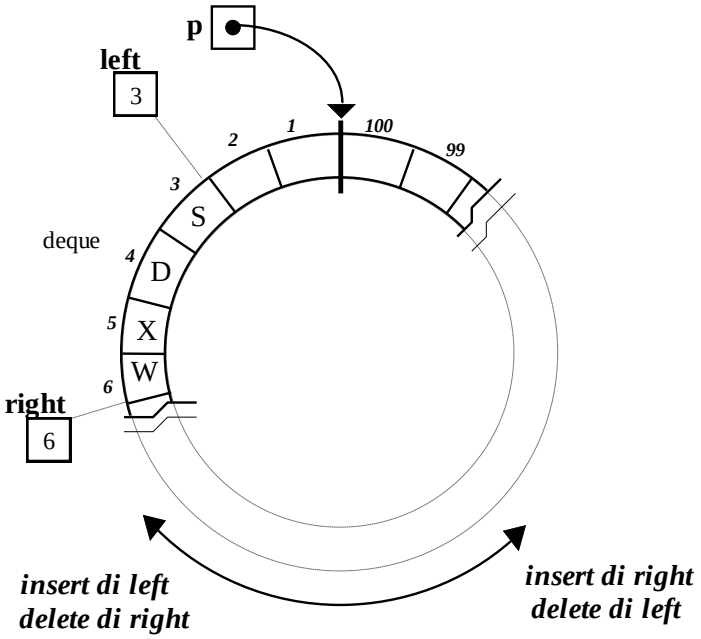
```
if ( top = panjang_max )
| then {OVERFLOW, tempat penuh}
|      tolak insertion/exit to....
| else top ← top + 1
|      {isikan data pada posisi q+top-1}
endif
```

Operasi DELETION :

```
if ( top = 0 )
| then {UNDERFLOW, stack kosong}
|      tolak deletion/exit to....
| else {delete data pada posisi q+top-1}
|      top ← top - 1
endif
```

Alokasi Sequential untuk DEQUE

Situasinya hampir sama dengan queue, hanya bedanya *pengembangan dan pengerutan list terjadi pada kedua ujung*, yang diwakili oleh integer **left** dan **right**. Penganggapan sebagai circular list juga harus tetap dilaksanakan. Kedua posisi dapat bergerak baik kekiri (*counter clock wise*) atau kekanan (*clock wise*).
Demikian pula dapat terjadi baik "overflow" maupun "underflow".



Bentuk Algoritma Insertion & Deletion.

Ada 2 macam insertion : pada ujung right dan ujung left; demikian juga ada 2 macam deletion masing masing satu pada kedua ujung tersebut. Jika diambil asumsi pemberian nilai variabel left dan right seperti pada ilustrasi gambar, dan jumlah elemen maksimum = panjang_max ; dan :

kondisi deque "kosong" ≡ **(right=0) and (left=0)**

Maka bentuk algoritmanya adalah sbb :

Operasi INSERTION :

pada ujung right

```
if ( right = panjang_max )
| then  right ← 1
| else  right ← right + 1
endif
if ( right = left )
| then  {OVERFLOW, penuh}
|       tolak insertion/restore nilai right....
| else  if ( left = 0 )
|       | then  {tadinya deque kosong}
|       |       left ← 1
|       | else
|       endif
|       {isi data pada posisi p+right}
endif
```

pada ujung left

```
if ( left = 1 )
| then  left ← panjang_max
| else  left ← left - 1
endif
if ( right = left )
| then  {OVERFLOW, penuh}
|       tolak insertion/restore nilai left....
| else  if ( right = 0 )
|       | then  {tadinya deque kosong}
|       |       left ← 1
|       |       right ← 1
|       | else
|       endif
|       {isi data pada posisi p+left}
endif
```

Operasi DELETION :

pada ujung right

```
if ( right = 0 ) and ( left = 0 )
| then  {Underflow, deque masih kosong}
|       tolak deletion.....
| else  if ( left = right )
|       | then  { tinggal satu }
|       |       left ← 0
|       |       right ← 0
|       | else  if (left=panjang_max)
|       |       | then  left ← 1
|       |       | else  left ← left + 1
|       |       endif
|       endif
endif
```

pada ujung left

```
if ( right = 0 ) and ( left = 0 )
| then  {Underflow, deque masih kosong}
|       tolak deletion.....
| else  if ( left = right )
|       | then  { tadinya tinggal satu }
|       |       right ← 0
|       |       left ← 0
|       | else  if (right = 1 )
|       |       | then  right←panjang_max
|       |       | else  right ← right - 1
|       |       endif
|       endif
endif
```

Alokasi Sequential untuk NON RESTRICTED LINEAR LIST

Pada umumnya tidak memanfaatkan alokasi sequential , karena insertion maupun deletion yang terjadi ditengah list akan menyebabkan harus dilakukannya "pemindahan/ penggeseran data" (data movement) secara besar besaran. Keadaan ini mengakibatkan buruknya performance. Jika masih ingin ditempuh alokasi sequential (meskipun alternatif ini hampir tidak pernah dipilih) maka harus dapat diatur cara untuk :

- "*menandai*" elemen yang diDELETE.
- "*memakai kembali*" tempat elemen yang sudah didelete tsb, jika ada INSERTION, disertai usaha minimal untuk melakukan "*penggeseran*".

Menangani OVERFLOW/ UNDERFLOW pada alokasi sequential.

menangani UNDERFLOW :

Tolak operasi DELETION tersebut !
Kondisi UNDERFLOW bukanlah kondisi ERROR, akan tetapi kondisi dimana operasi DELETION tidak mungkin dilakukan lagi, karena list sudah kosong. Kondisi ini mungkin menjadi tanda bahwa harus dilakukan suatu operasi tertentu !

menangani OVERFLOW :

lakukan *re-alokasi yang diperbesar* (perubahan medan alokasi dengan medan yang lebih luas), atau perluasan biasa saja.
Misalnya untuk kasus diatas jika dapat alokasi ditambah sampai dengan posisi 101. Setelah itu baru dilakukan insertion ybs.
Jika ternyata posisi 101 sudah dialokasikan untuk keperluan lain, maka :

- mungkin tambahan alokasi (dapat diminta lebih dari 1) dilakukan di posisi bukan 101, sehingga terjadi suatu alokasi yang tidak sequential. Untuk keperluan ini harus ditambahkan suatu cara untuk "*menyambung*" kedua penggal alokasi, yang akan mengakibatkan berubahnya bentuk proses insertion dan deletion.
- mungkin pula seluruh medan alokasi (sebanyak 101 satuan atau lebih) dipindahkan ke daerah lain yang masih "*kosong*" dan konsekwensinya harus dilakukan pemindahan 100 satuan data.

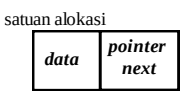
3.4. ALOKASI TERKAIT (LINKED ALLOCATION) UNTUK STRUKTUR LINIER.

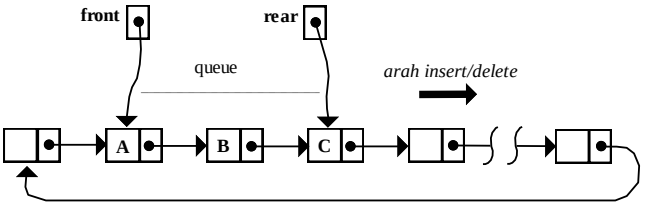
Alokasi linked untuk QUEUE

Biasa dipergunakan bentuk "*singly linked list*" biasa atau "*circularly singly linked list*". Satuan alokasinya sama adalah sbb :

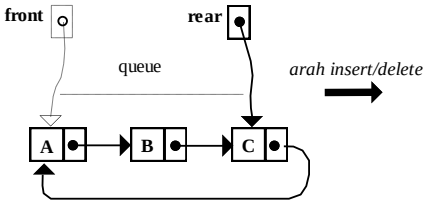
- *data* (bisa terbagi bagi lagi menjadi beberapa item data)
- *pointer next*, pointer yang berfungsi menunjuk elemen berikut.

Sebagai ilustrasi perhatikan pada gambar dimana terlihat beberapa cara representasi yang agak berbeda satu dengan yang lainnya. Satuan alokasi yang sama dipergunakan pada ke 3 buah cara pada gambar tersebut. Pada cara-1 dan cara-2 dipilih anggapan circular, sedangkan pada cara-3 tidak demikian.
Perhatikan pula bahwa pada cara-1 jumlah unit alokasi yang menjadi anggota circular list tersebut lebih banyak dari anggauta queue. Ini merupakan hasil dari strategi dimana deletion tidaklah berarti "melepaskan" unit ybs, tetapi cukup dengan menggeser pointer front. Unit unit sisa deletion nantinya akan dipergunakan pada insertion. Sedangkan pada ke 2 cara yang lain tidaklah demikian. Deletion berarti pula bahwa unit alokasi elemen ybs "dilepaskan" yang dapat berarti : dikembalikan pada sistem atau diurus oleh mekanisme lain. Pada semua cara proses insertion dan deletion mempunyai "arah" yang sama.

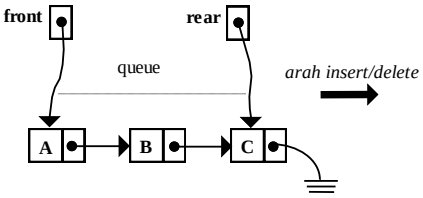




cara-1 :
bentuk : "circularly singly linked list"
jika ada deletion, satuan alokasi ybs tidak "dilepaskan"

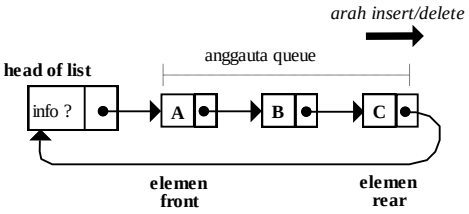


cara-2 :
bentuk : "circularly singly linked list"
jika ada deletion, satuan alokasi ybs "dilepaskan",
sehingga jumlah alokasi selalu sama dengan jumlah elemen
pointer front sebenarnya tidak perlu ! (posisinya selalu "didepan" rear)



cara-3 :
bentuk : "singly linked list" (tidak circular)
jika ada deletion, satuan alokasi ybs "dilepaskan",
sehingga jumlah alokasi selalu sama dengan jumlah elemen

Pada beberapa jenis aplikasi seringkali lebih baik untuk melakukan cara representasi dimana pada queue ybs. minimal ada 1 satuan alokasi yang berfungsi sebagai "*head of list*". Dengan demikian pada saat "queue" kosong masih ada 1 unit alokasi. Alokasi head of list biasanya bersifat "fixed", dan ada bagian dimana diletakkan info yang dianggap penting mengenai list ybs. Pada saat queue "kosong", pointer pada head of list menunjuk dirinya sendiri.



cara-4 :
bentuk : "circularly singly linked list"
jika ada deletion, satuan alokasi ybs "dilepaskan",
sehingga jumlah alokasi selalu sama dengan jumlah elemen
head of list : satuan alokasi berbeda, biasanya memori yang "fixed"

Bentuk Algoritma Insertion & Deletion.

Akan sangat tergantung dari cara representasi yang dipilih. Misalnya dipilih cara seperti cara-1 pada gambar diatas, maka terlebih dahulu harus disusun semua kondisi yang mungkin ada pada cara tersebut. Dalam hal ini ada 5 macam "kondisi" queue yang perlu diperhatikan sehubungan dengan insertion dan atau deletion. Kondisi seperti dibawah ini disusun dengan asumsi bahwa linked-list mungkin "kosong".

(a) List "penuh" terisi oleh anggauta queue; atau **front=rear↑.next.**
(b) List ada, tapi queue "kosong" (tidak punya anggauta); dapat dipilih salah satu dari 2 asumsi sbb :
*(***front=***NIL) and (***rear<>***NIL)* , atau
*(***front<>***NIL) and (***rear=***NIL)*
(c) List belum dialokasikan, queue "kosong"; atau (**front=***NIL) and (***rear=***NIL)*
(d) Queue hanya mempunyai 1 anggauta; atau (**front<>***NIL*) *and (***front=rear***)*

Bentuk kondisi akan berubah jika diambil asumsi bahwa queue mempunyai semacam elemen "header", sehingga minimal harus ada 1 elemen pada linked-list yang mewakili queue ybs.
Perhatikan pula bahwa jika list penuh terisi atau belum dialokasikan , pada proses insertion mungkin akan terjadi hal sbb :

- minta alokasi baru sebanyak 1 satuan.
- jika ditolak oleh sistem berarti kondisi OVERFLOW; maka insertion ditolak !

Jika untuk (b) diambil asumsi yang pertama, maka bentuk algoritmanya adalah :

```
INSERTION :
 case of (front)
  NIL  : { kondisi (b) atau (c)}
         if rear=NIL
          then    { belum ada alokasi (c) }
                  p ← malloc ( 1, elemen_list )
                  if   p = NIL
                   then   {alokasi gagal }
                          call prosedur error .....
                   else   { "bentuk circular" }
                          front ← p
                          rear ← p
                          p↑.next ← p
                          isikan data pada rear↑...
                  endif
          else    { queue kosong (b) }
                  front ← rear
                  isikan data pada rear↑...
         endif
  rear↑.next : { kondisi (a) }
         p ← malloc ( 1, elemen_list )
         if   p = NIL
          then   {alokasi gagal }
                 call prosedur error .....
          else   { "penyambungan" }
                 p↑.next ← front
                 rear↑.next ← p
                 rear ← p
                 isikan data pada rear↑...
         endif
  otherwise : { kondisi (d) atau normal }
         rear ← rear↑.next
         isikan data pada rear↑...
 endcase
```

```
DELETION :
 if  front = NIL
  then    { queue "kosong"/ underflow }
          tolak deletion.....
  else    if  front=rear
           then   {tinggal satu }
                  front ← NIL
           else   {majukan front }
                  front  ← front↑.next
          endif
 endif
```

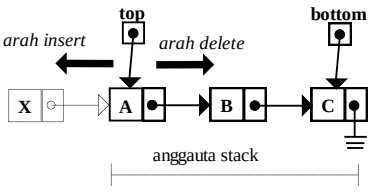
Alokasi linked untuk STACK

Jarang direpresentasikan dengan alokasi linked, representasi dengan alokasi sequential sudah dapat menghasilkan performance yang cukup baik.
Stack dapat direpresentasikan dengan singly linked list, dengan arah pointer ke posisi bottom. Insertion dilakukan sbb :

- minta alokasi baru, isikan datanya.
- set pointernya sehingga menunjuk elemen lama pada posisi *top*.
- pindahkan pointer *top* sehingga menunjuk elemen baru.

Sedangkan deletion dilakukan sbb :

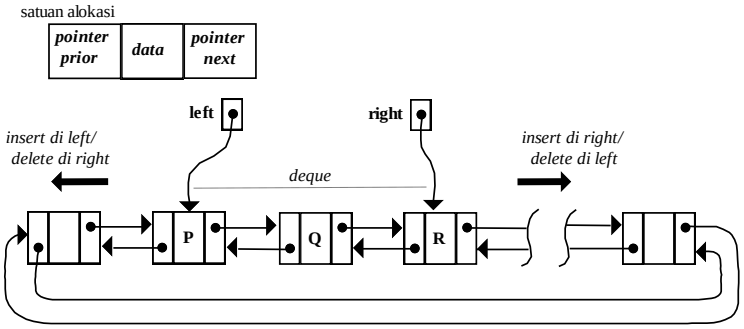
- gunakan pointer bantu untk menunjuk elemen pada posisi *top*.
- pindahkan *top* ke elemen berikut.
- dealokasi elemen yang ditunjuk oleh pointre bantu tsb diatas.



Alokasi linked untuk DEQUE

Oleh karena kedua ujung (left dan right) dapat bergerak ke *dua arah*, maka biasanya dipergunakan alokasi linked yang berbeda dengan queue. Dipergunakan suatu bentuk dimana setiap elemen mempunyai 2 buah pointer, ialah :

- *pointer next* yang berfungsi menunjuk alamat elemen berikut, dan
- *pointer prior* yang berfungsi menunjuk alamat elemen sebelum.



Bentuk yang terjadi disebut "*doubly linked list*". Jika diatur seakan akan melingkar seperti dalam kasus ini, maka secara tepat disebut "*circularly doubly linked list*".

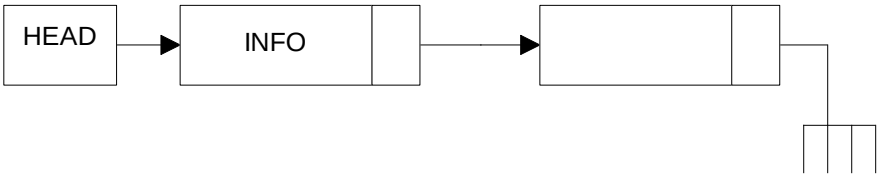
INSERTION pada ujung right akan menggeser pointer right kearah next, pada ujung left akan menggeser pointer left kearah prior.
DELETION pada ujung right akan menggeser pointer right kearah prior, sedangkan pada ujung left akan menggeser pointer left kearah next.
Seperti juga pada queue, dapat ditempuh strategi alternatif sedemikian rupa sehingga ukuran list tepat sama dengan ukuran deque.

Alokasi linked untuk NON RESTRICTED LINEAR LIST

Dapat dipergunakan berbagai bentuk linked list, singly linked, doubly linked, baik dengan anggapan melingkar ataupun tidak. Pemilihan dilakukan sesuai dengan keperluan proses pada masalah aplikasi.
Demikian pula pada strategi alokasi dan dealokasi dapat diterapkan 2 macam cara seperti telah disebutkan pada pembahasan mengenai queue dan stack. Perbedaan terutama dijumpai pada insertion atau deletion ditengah tengah list.

Contoh rerpesentasi menggunakan *Singly Linked List:*

- Suatu Linked List memiliki sejumlah elemen.
- Elemen paling awal disebut HEAD (Kepala) yang tidak digunakan untuk menyimpan data.

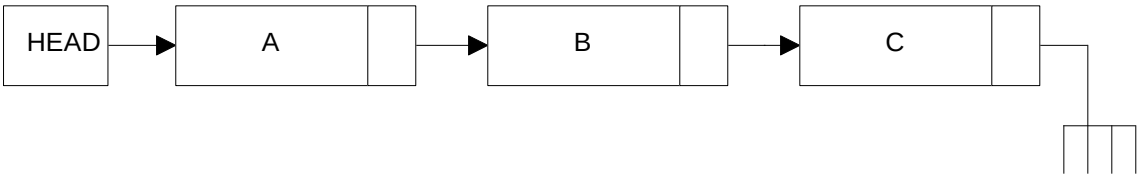


Head, menunjukkan ke awal list.
Untuk pengisian elemen pertama perintahnya :

```
HEAD↑.Info ←'A'
```

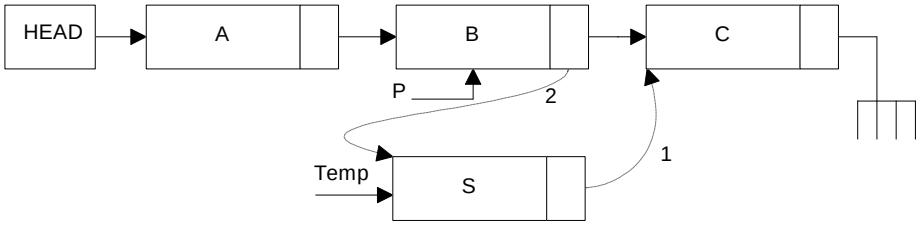
Untuk elemen lainnya :

```
HEAD↑.Next↑. Next↑.Info ←'C'
```



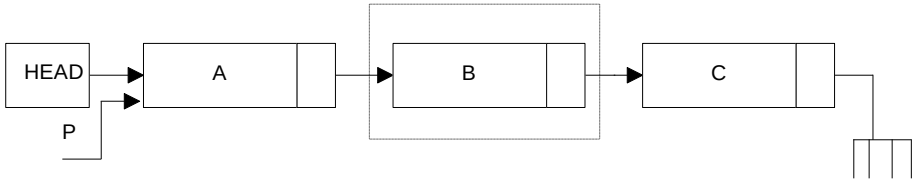
Penulisan linked list di atas dapat berbentuk :
List (‘A’,’B’,’C’)

Konsep sisip/menambah dan hapus/mengurangi List:
a. Sisip :



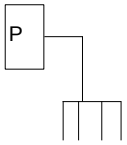
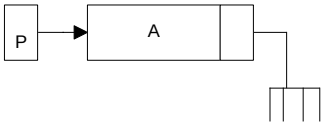
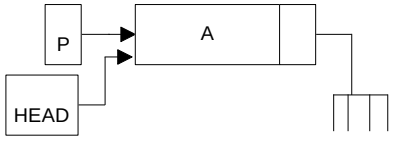

```
Temp↑.Next ←P↑.Next;
P↑.Next ← Temp;
```

b. Hapus :




```
P↑.Next ← P↑.Next↑.Next;
```

Prosedur - prosedur dasar pada Linked List :
a. Membuat List Baru :

`New(p);`	
`p↑.Info ← 'A';` `p↑.Next ← Nil;`	
`Head ← p;`	

b. Menambah/sisip :

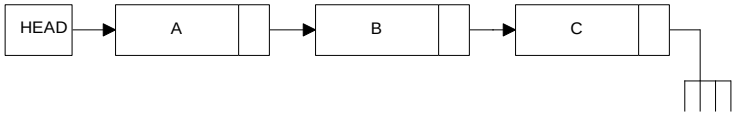
-. Di Head :

```
New(p);
p↑.Info ← 'B';
p↑.Next ← Head;
Head ← p;
```

-. Di Tail (Ujung) :

```
New(p);
p↑.Info ← 'S';
p↑.Next ← Nil;
Head↑.Next ← p;
```

c. Mencari lokasi elemen tertentu :



Cari elemen yang isinya = 'B'

```
p ← HEAD;
While ( p ≠ Nil) do
     if (p↑.Info = 'B')
          then Posisi ← p
          else p ← p↑.Next
     Endif
Endwhile
```

d. Menghapus, harus ada satu variabel pointer pembantu, untuk menunjukkan elemen sebelum elemen yang akan dihapus.