



MODUL IX Struktur Data

Judul	Struktur data Linked List pada STACK dan QUEUE	
Penyusun	Distribusi	Perkuliahan
Nixon Erzed	Teknik Informatika Universitas Esa Unggul	Pertemuan – IX online

Tujuan :

1. Mahasiswa memahami struktur stack dan representasi dinamik stack dengan linked list
2. Mahasiswa memahami struktur queue dan representasi queue secara dinamik dengan linked list

Materi :

Bagian A : Struktur Stack

1. Representasi Stack dengan Linked List
2. Operasi-operasi pada stack
3. Stack sebagai Ordered Linked List

Bagian B : Struktur Queue

4. Definisi
5. Karakteristik Penting Queue
6. Representasi Queue dengan Array
7. Operasi Dasar
8. Representasi Statik Linear Queue dengan Array

STACK DENGAN LINKED LIST (REPRESENTASI DINAMIS)

Selain implmentasi stack dengan array, seperti sudah dibahas pada bagian sebelumnya, stack dapat juga diimplementasikan dengan *linked list*. Keunggulannya dibandingkan array adalah dalam hal alokasi memory yang bersifat dinamis. Misalnya jika sebuah stack dialokasikan 200 elemen, sedangkan ketika dipakai oleh porogram, umumnya hanya diisi 50 elemen, sehingga terjadi pemborosan memory untuk sisa yang 150 elemen lagi. Pada kasus lain ketika dibuat alokasi yang sedikit (misalnya 7 elemen saja), ketika pada suatu saat ternyata proses memerlukan jumlah yang lebih besar dari itu, maka mengakibatkan proses terhenti.

Dengan linked list, maka stack yang diimplementasikan tidak mengenal istilah full.

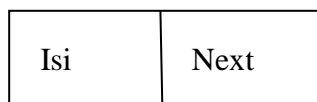
Contoh :

deklarasi Stack dengan linked list :

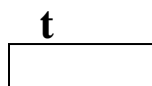
```
Type
  Tipedata      = integer;
  Point         = ^ Stack
  Stack         = record
                    Isi   : tipe data
                    Next  : point
                end
Var
  t             : point
```

Hasil deklarasi tersebut dapat digambarkan sebagai berikut :

Struktur Simpul



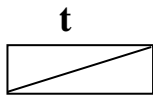
Dan pointer TOP



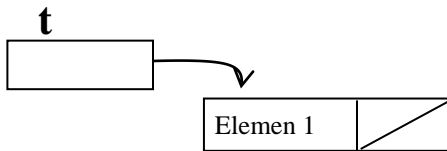
Implementasi Sifat Stack pada Linked List

Untuk mengimplementasikan sifat stack (LIFO) pada linked list, maka operasi penyisipan dan penghapusan dilakukan **pada simpul pertama**.

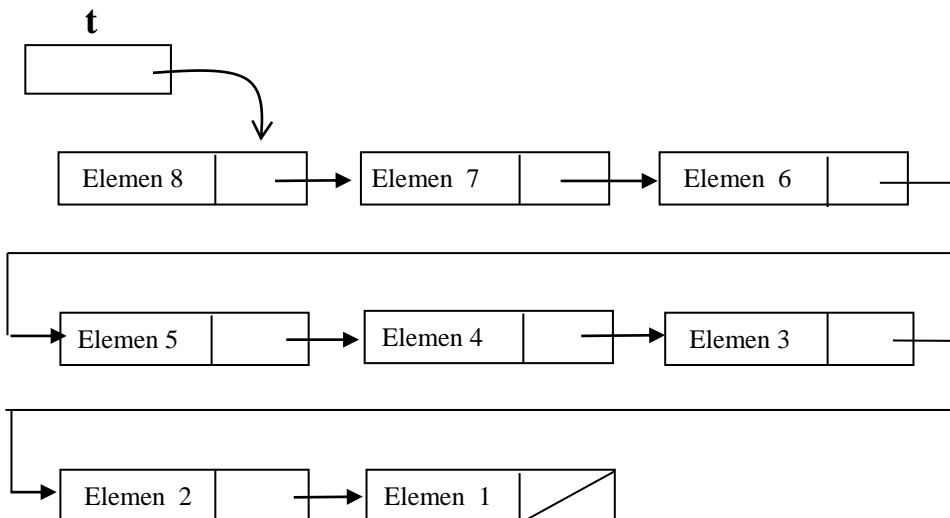
Stack Kosong



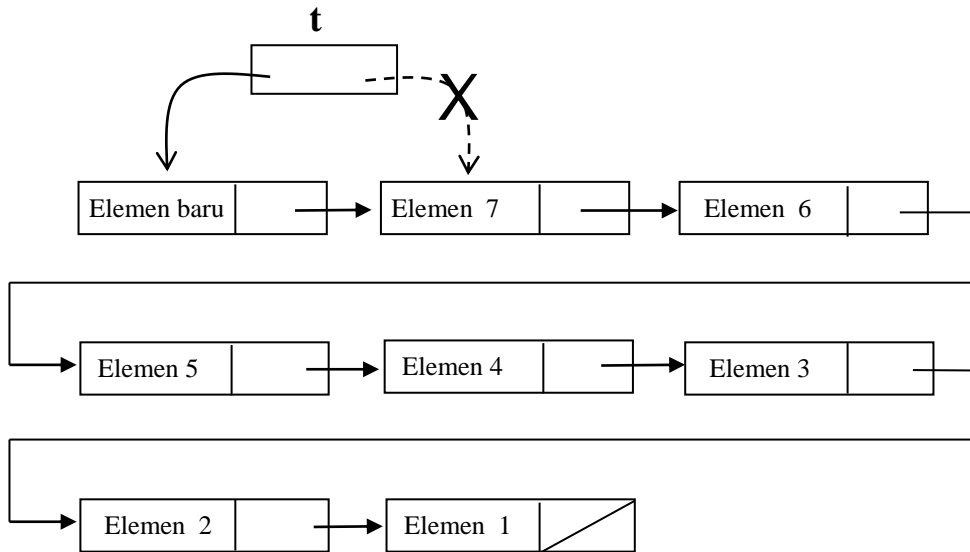
Stack dengan satu elemen



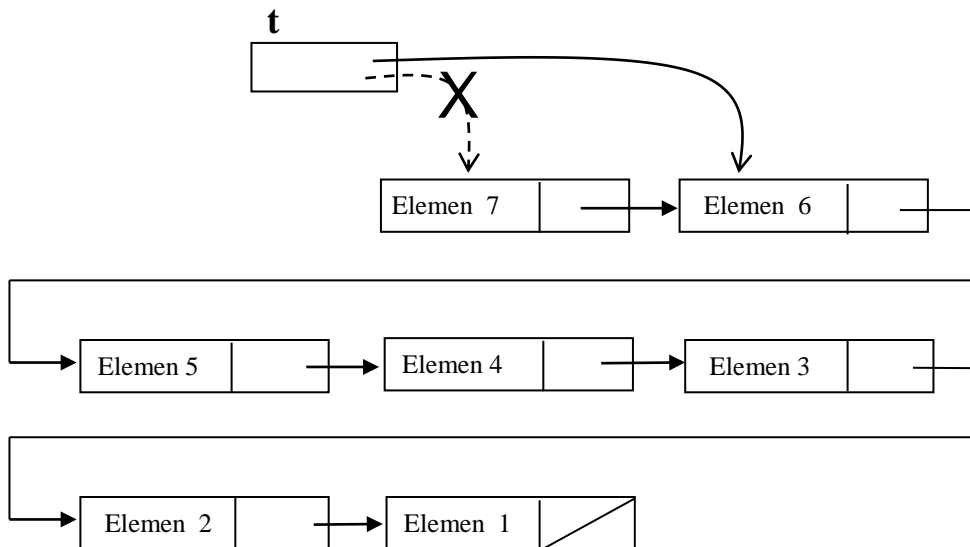
Stack dengan sejumlah elemen.



Penambahan elemen baru pada Stack (Operasi Push (S))



Penghapusan sebuah elemen dari Stack (Operasi Pop (S))



Operasi-operasi pada Stack dengan Linked List:

Create S

→ Membuat stack S baru yang masih kosong.

Secara logika adalah dengan menginisialisasi variabel penunjuk elemen puncak stack (top pointer) $t = \text{nil}$.

```
Procedure Create;  
Begin  
    t ← nil;  
end;
```

Empty (S)

→ fungsi untuk memeriksa apakah stack kosong atau tidak
Secara logika stack kosong jika pointer top $t = \text{nil}$.

```
Function Empty : boolean  
Begin  
    Empty := false  
    if t = nil then Empty := true;  
End;
```

Push (S,x,t)

→ fungsi untuk menempatkan data x di stack S.
Operasi Push adalah insert simpul baru pada awal linked list

```
Procedure Push ( S : stack; x : typedata; var t : point)  
Var Baru : point  
Begin  
    New (Baru)  
    Baru^.Isi := x  
    If Empty then  
        Baru^.next := nil  
    Else  
        Baru^.next := t  
    End-if  
    t := Baru  
End;
```

Pop (S,x,t)

- fungsi untuk mengambil elemen teratas dari stack S.
Operasi Pop (S,x,t) akan ditolak jika Stack S kosong

```
Procedure Pop ( S : stack; var x : typedata; var t : integer)
Var   Hapus : point
Begin
    If not Empty then
    Begin
        x := t^.isi
        Hapus := t
        t := t^.next
        Dispose(hapus)
    End-if
End;
```

Clear (S, t)

- fungsi untuk mengosongkan seluruh elemen dari stack S.

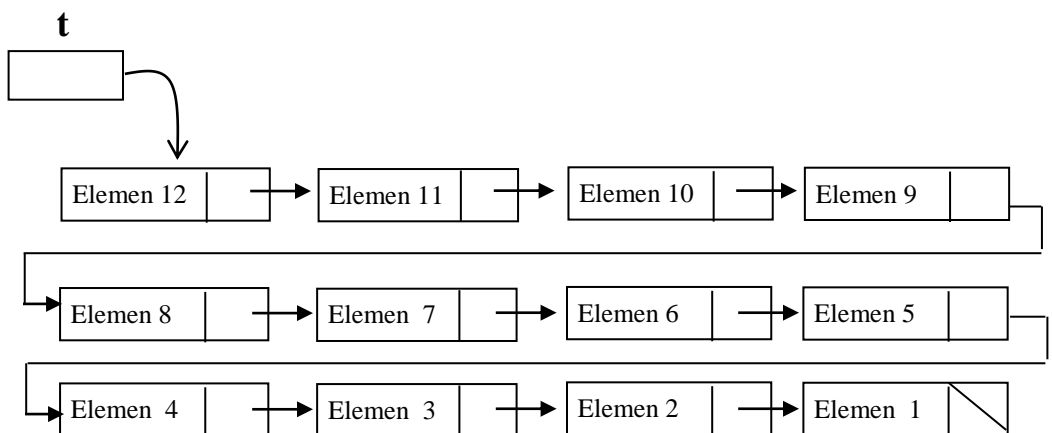
Konsep clear yang diterapkan adalah bahwa seluruh simpul secara bertahap dimusnahkan melalui operasi Pop hingga stack Empty.

```
Procedure Clear ( S : stack; var t : point)
Var   trash : Tipedata
Begin
    While not empty do Pop(S,trash,t)
End;
```

OPERASI-OPERASI KHAS PADA STACK SEBAGAI OREDRED LINKED LIST

Penelusuran ordered linked list ialah mengunjungi seluruh elemen linked list dari simpul yang pertama sampai dengan simpul terakhir. Terminasi penelusuran adalah jika ditemukan simpul dengan field pointer bernilai **Nil**.

Misalnya diberikan sebuah Linked List sebagai berikut :



Algoritma umum untuk menelusuri Stack tersebut :

```

Algoritma Penelusuran_Stack
  var k : ptr           { var. pointer untuk penelusuran }
  Begin
    k := Head           { menginisialisasi k dengan alamat
                        simpul yang pertama }
    While k <> nil do   { looping akan dilakukan selama pointer
      begin              tidak nil }
        Visit ( k )    { aksi yang didefinisikan terhadap simpul
                        yang dikunjungi }
        k := k^.next  { memperbaharui variabel penelusuran
                        dengan alamat simpul berikutnya }
      end-while
    end-algoritma
  
```

Untuk menyisipkan dan menghapuskan sebuah simpul bukan diawal stack langkah pertama yang harus dilakukan adalah menemukan lokasi peyisipan atau data yang akan dihapus.

```
Algoritma Menyisipkan_simpul_di_Sembarang_posisi_pada_stack
var k, ujung : ptr
Begin
  k := Head
  ujung := Head
  While k <> nil do
    begin
      |   ujung := k
      |   k := k^.next
    } loop untuk menemukan alamat
    } simpul yang terakhir
  end-while
  Read ( A )
  New ( Ptr )
  Ptr^.Data := A
  Ptr^.Next := nil
  if ujung = nil
  then
    Head := Ptr
    { Stack yang akan disisipi data
    adalah linked list kosong }
  else
    ujung^.next := Ptr
    { Stack tdk kosong dan simpul
    disisipkan sebagai simpul yang
    terakhir }
  end-if
end-algoritma
```


CONTOH PENGGUNAAN STACK

KONVERSI BILANGAN DESIMAL KE BINER

Misal dimiliki bilangan desimal $x = 325$ akan dihitung nilai dalam basis 2 (biner). Untuk mendapat nilai konversi, maka terhadap x akan dilakukan operasi mod 2 untuk mendapatkan nilai biner dan div 2 secara berulang hingga dicapai $x = 0$.

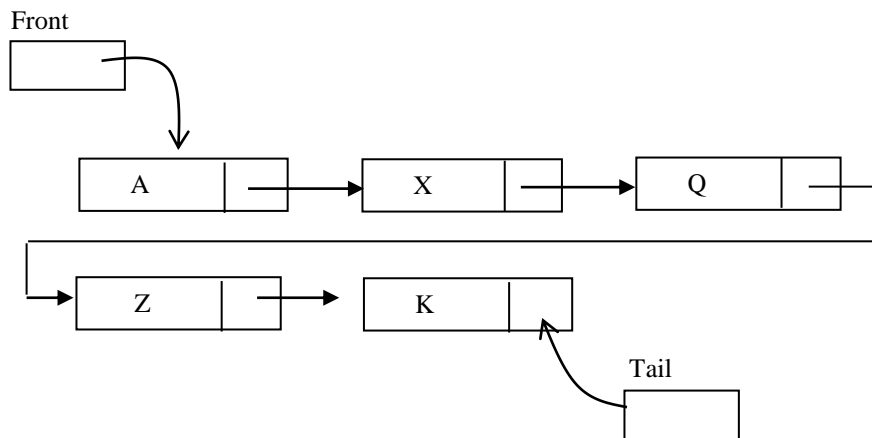
Hasil operasi MOD 2 akan dipush ke Stack S, hingga $x = 0$.

Berikut adalah algoritma konversi secara lengkap :

```
Procedure Konversi ( S : stack; data : desimal; var b : biner)
Var t, d : integer
Begin
    t := 0
    x := data
    While x > 0 do
        PUSH(S, x MOD 2, t)
        x := x DIV 2
    End-while
    While t > 0 do
        POP(S, d, t)
        Pack (d, b)
    End-while
    Write(" hasil konversi", data, "adalah ", b)
End;
```

REPRESENTASI DINAMIK QUEUE DENGAN LINKED LIST

Sesuai dengan sifat dinamik struktur Linked List, maka pada representasi Queue dengan Linked List tidak mengenal panjang maksimum antrian (MaxQueue). Untuk implementasi antrian (Queue) digunakan Single Linked List dengan penunjuk Kepala dan Ekor.

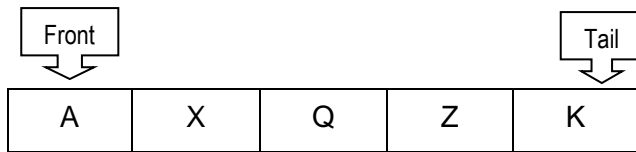


Dalam implementasinya operasi penambahan (EnQueue) dilakukan disalah satu ujung Linked List dan operasi penghapusan (DeQueue) dilakukan diujung lainnya.

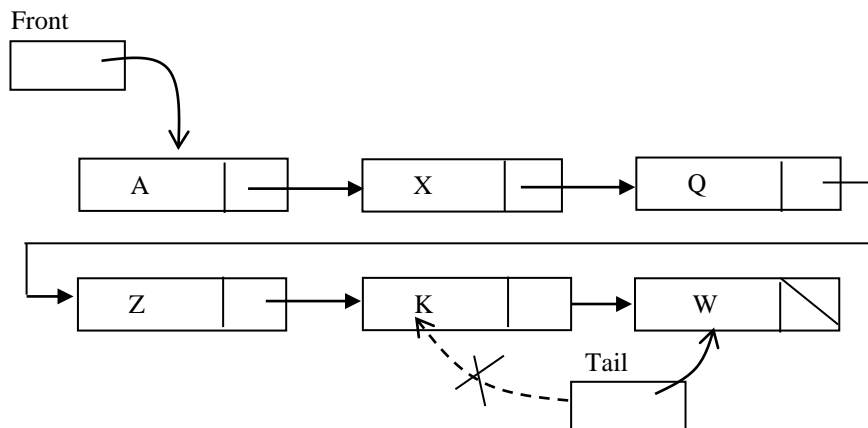
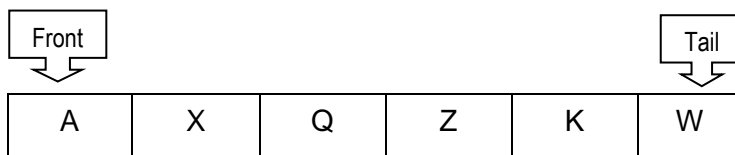
Mengingat sifat penghapusan dari Linked List dengan Dua Pointer, maka lebih efisien jika diterapkan operasi penghapusan pada elemen pertama Linked List. Sehingga implementasi Queue menggunakan konvensi sebagai berikut :

1. Front menunjuk elemen terdepan dari Queue,
2. Elemen terdepan Queue diimplementasikan pada elemen pertama Linked List
3. Operasi DeQueue merupakan operasi penghapusan elemen pertama Linked List
4. Tail menunjuk elemen terakhir dari Queue,
5. Elemen terakhir Queue diimplementasikan pada elemen terakhir Linked List
6. Operasi EnQueue merupakan operasi penyisipan simpul baru sesudah elemen terakhir Linked List

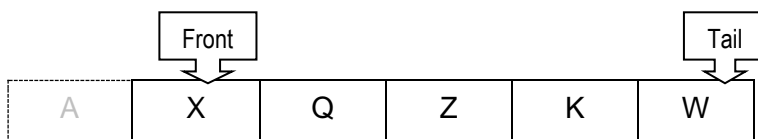
Linked List pada gambar diatas merepresentasikan Queue :



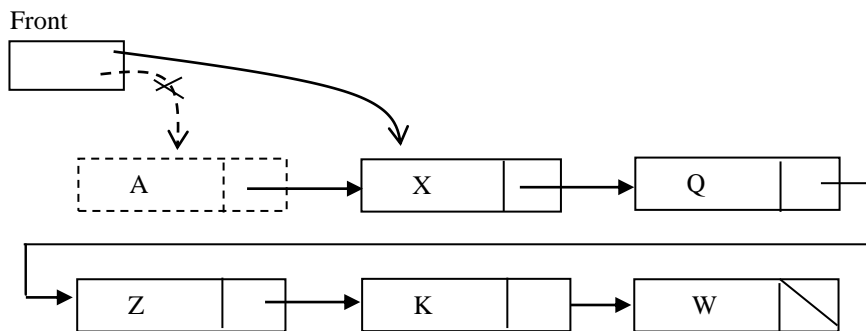
Jika dilakukan EnQueue sebuah data baru (misalnya W)



Jika dilakukan DeQueue terhadap data pertama dalam antrian (yaitu : A)



\



Berikut ini adalah contoh deklarasi data untuk suatu Queue :

```
Type
  TipeData = char;
  QP = ^ Queue
  Queue = record of
    Data : TipeData
    Next : QP
  end

Var
  Front, Tail : QP
  DataBaru : TipeData
```

Konvensi :

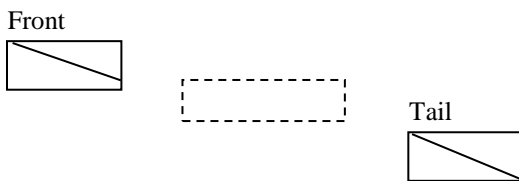
1. Front selalu menunjuk posisi Queue-1 atau elemen pertama dari Liked List
2. Tail menunjuk elemen terakhir pada Queue, sehingga Tail^.Next = nil
3. Queue adalah kosong jika Front = Tail = Nil

OPERASI-OPERASI PADA QUEUE DENGAN LINKED LIST

1. Create

Procedure Create berguna untuk menciptakan Queue baru dan Kosong.

```
Procedure Create;  
Begin  
    Head := nil;  
    Tail := nil;  
End;
```



2. CekStatus

Operasi CekStatus berguna untuk memeriksa apakah Queue :

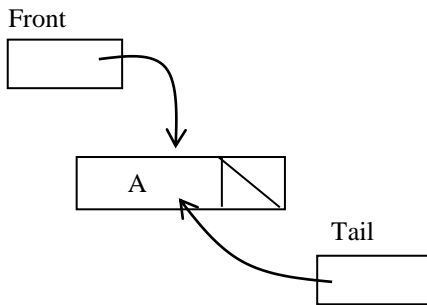
- Kosong / empty, jika Tail = Front = nil,

```
Function Empty : Boolean ;  
Begin  
    If Front = nil  
    then Empty := true  
    else Empty := false  
end;
```

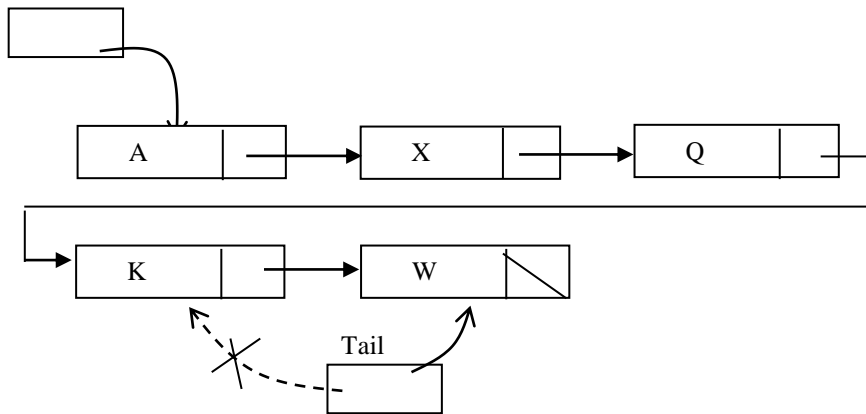
3. EnQueue

Operasi EnQueue berguna untuk memasukkan sebuah elemen baru ke antrian. Operasi ini hanya dibedakan atas penyisipan pada antrian kosong dan penyisipan pada antrian tidak kosong.

a. Penyisipan pada antrian kosong



b. Penyisipan pada antrian tidak kosong



```

Procedure EnQueue ( Databaru:TipeData )
Begin
    New (QP)
    QP^.Data := Databaru
    QP^.Next := nil

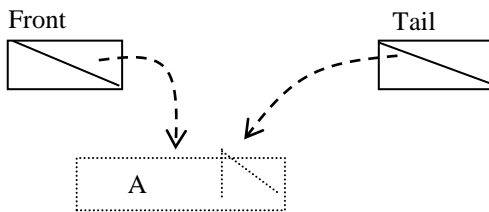
    If not Empty then
        begin
            Tail^.Next := QP           { antrian tidak kosong }
            Tail := QP
        Else
            Tail := QP               { antrian kosong }
            Front := QP
        End;
    End;

```

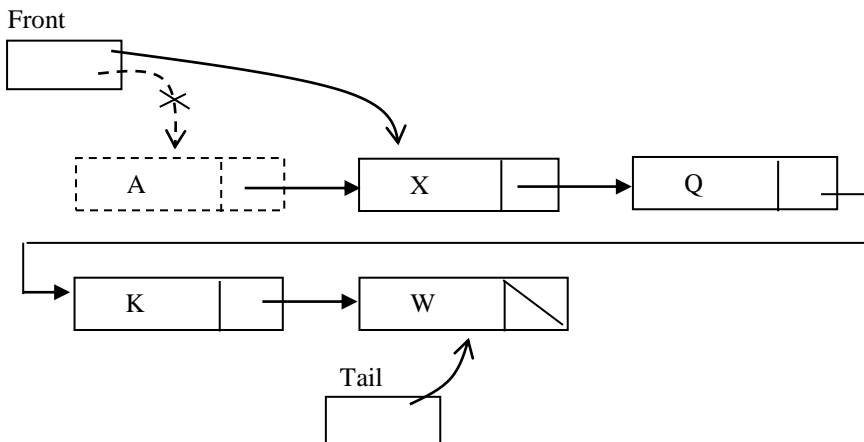
4. DeQueue

Operasi DeQueue berguna untuk mengambil sebuah elemen dari antrian. Dilakukan terhadap elemen terdepan (front). Dibedakan atas penghapusan pada antrian dengan 1 elemen dan pada antrian dengan lebih dari 1 elemen.

a. Pengambilan pada antrian dengan 1 elemen



b. Pengambilan pada antrian dengan lebih dari 1 elemen



```

Procedure DeQueue ( var DataServe :TipeData )
Var Hapus : QP
Begin
  If not Empty then
    begin
      DataServe := Front ^ . Data
      Hapus := Front
      Front := Front ^ . Next
      If Front = nil
        then Tail := Nil           { antrian dengan satu elemen }
      Dispose (Hapus)
    End;
  End;

```

5. Clear

Mengosongkan antrian (Queue) pada implementasi dengan linked list adalah melakukan operasi DeQueue secara berulang hingga Queue kosong (empty).

```
Procedure Clear ;  
Var trash : tippedata;  
Begin  
    While not Empty do  
        Begin  
            DeQueue ( trash )  
        End;  
    End;  
End;
```

6. Panjang Antrian

Panjang antrian (Queue) pada Linked List adalah jumlah elemen Linked List tersebut .

```
Function Panjang : integer ;  
Var x : integer  
    Cari : QP  
Begin  
    x := 0  
    Cari := Front  
    While Cari <> nil do  
        Begin  
            x := x + 1  
            Cari := Cari ^ Next  
        End;  
    Panjang := x  
End;
```