Pertemuan 4

LINKED LIST

BAGAIMANA LINKED LIST MENYIMPAN DATA

Linked List, di sisi lain, berbeda dengan array. Linked List, tidak menyimpan data di lokasi memori yang berdekatan. Untuk setiap item di lokasi memori, Linked List menyimpan nilai item dan referensi atau penunjuk ke item berikutnya. Sepasang item Linked List dan referensi ke item berikutnya merupakan sebuah node.

Misal, jika sebuah node terdiri dari 34 | 10, artinya nilai node tersebut adalah 30, sedangkan item berikutnya disimpan di lokasi memori "10". Untuk melintasi Linked List, Anda hanya perlu mengetahui lokasi memori atau referensi dari node pertama, node lainnya dapat dilintasi secara berurutan menggunakan referensi ke elemen berikutnya di setiap node.

Referensi ke node pertama juga dikenal sebagai node awal.

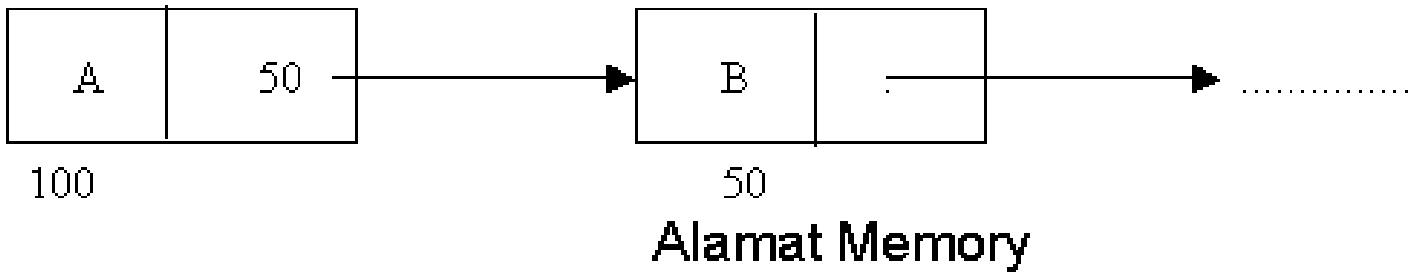
LINKED LIST vs ARRAY

- Linked List adalah struktur data dinamis yang berarti bahwa memori yang dicadangkan untuk Linked List dapat ditingkatkan atau dikurangi saat runtime. Tidak ada memori yang dialokasikan untuk struktur data Linked List sebelumnya. Setiap kali item baru diperlukan untuk ditambahkan ke link, memori untuk node baru dibuat pada saat berjalan. Di sisi lain, dalam kasus larik, memori harus dialokasikan terlebih dahulu untuk sejumlah item tertentu. Dalam kasus di mana item yang cukup tidak tersedia untuk mengisi semua indeks larik, ruang memori akan terbuang.
- Karena array membutuhkan lokasi memori yang berdekatan, sangat sulit untuk menghapus atau menyisipkan item dalam array karena lokasi memori dari sejumlah besar item harus diperbarui. Di sisi lain, item daftar tertaut tidak disimpan di lokasi memori yang berdekatan, oleh karena itu Anda dapat dengan mudah memperbarui Linked List .
- Karena fleksibilitasnya, Linked List lebih cocok untuk mengimplementasikan struktur data seperti tumpukan, antrian, dan daftar.

LINKED LIST

Salah satu Struktur Data Dinamis yang paling sederhana adalah Linked List atau Struktur Berkait atau Senarai Berantai, *yaitu suatu kumpulan komponen yang disusun secara berurutan dengan bantuan Pointer*.

Linked List (Senarai Berantai) disebut juga dengan Senarai Satu Arah (One-Way List). Masing-masing komponen dinamakan dengan Simpul (Node).



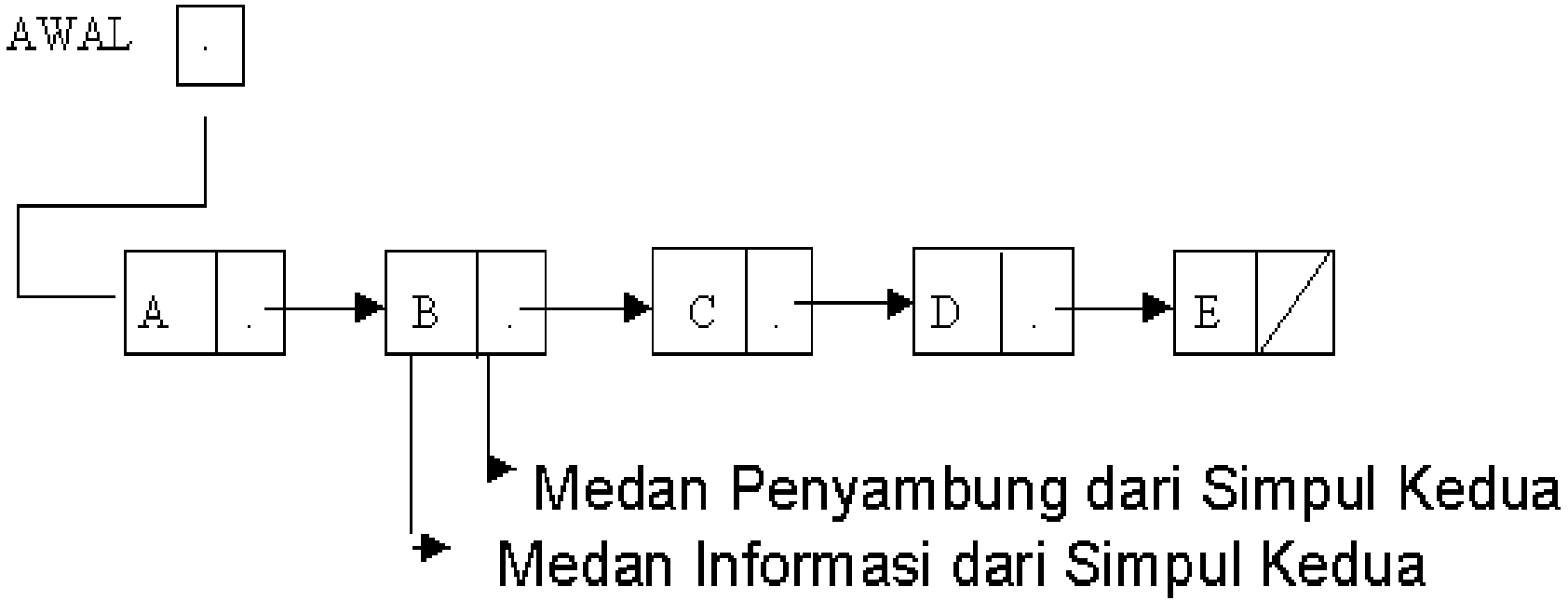
Setiap simpul dalam suatu Linked List terbagi menjadi dua bagian,yaitu :

1. Medan Informasi
   Berisi informasi yang akan disimpan dan diolah.

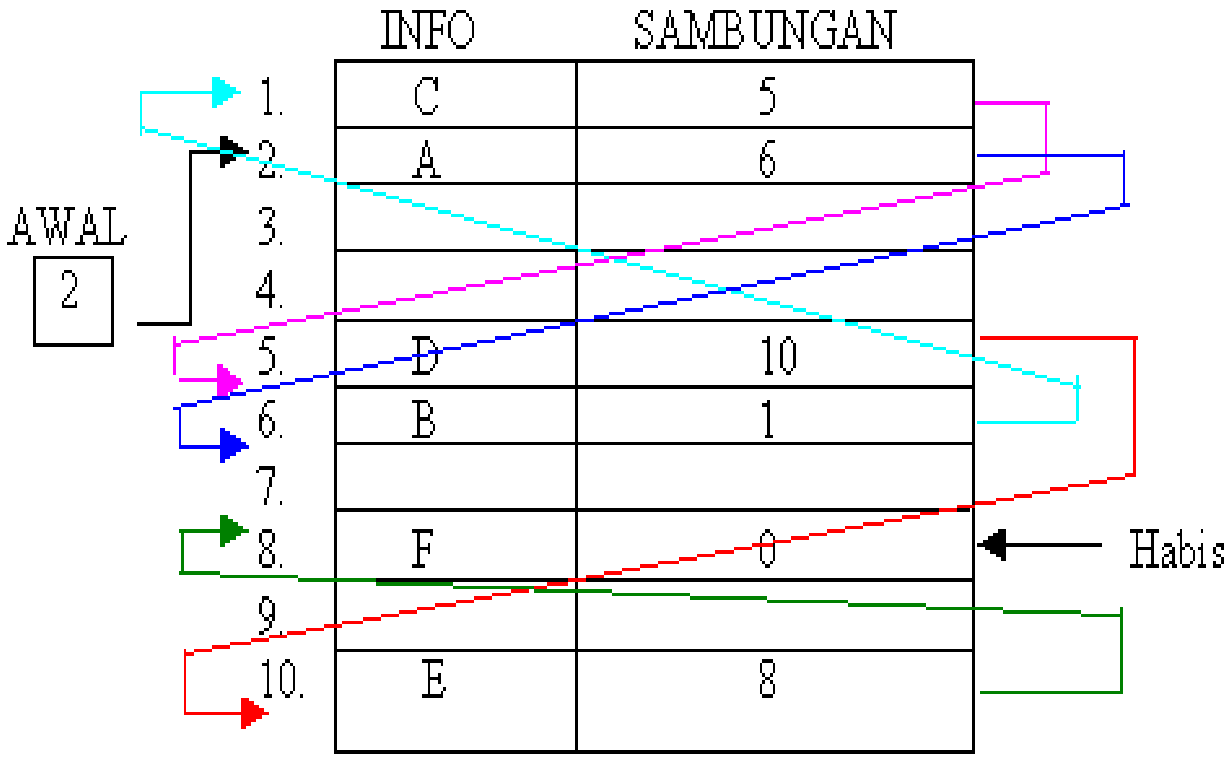
2. Medan Penyambung (Link Field)
   Berisi alamat berikutnya. Bernilai 0, Jika Link tersebut tidak menunjuk ke Data (Simpul) lainnya. Penunjuk ini disebut Penunjuk Nol.

LINKED LIST (Lanjutan)

Linked juga mengandung sebuah variabel penunjuk List, yang biasanya diberi nama START (AWAL), yang berisi alamat dari simpul pertama dalam List.



Penyajian Linked List dalam Memory



Penyajian Linked List dalam Memory (Lanjutan)

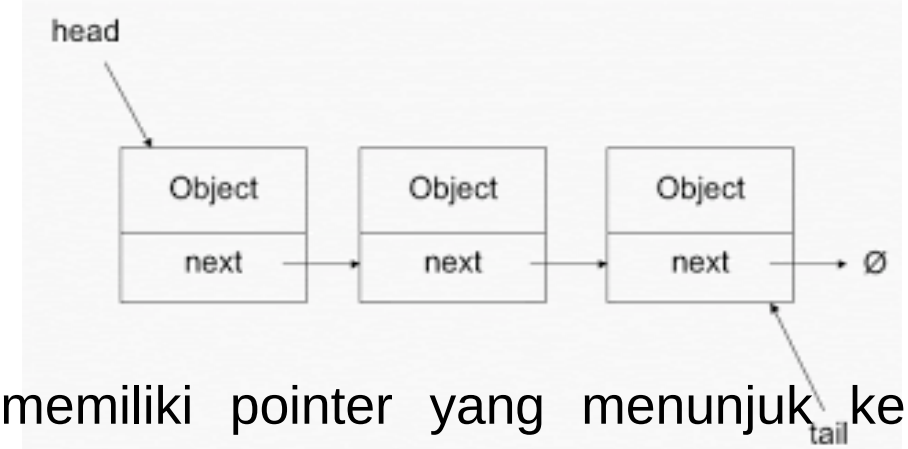
Keterangan :

AWAL	=	2	,	Maka	INFO[2]	=	'A'
SAMBUNGAN[2]	=	6	,	Maka	INFO[6]	=	'B'
SAMBUNGAN[6]	=	1	,	Maka	INFO[1]	=	'C'
SAMBUNGAN[1]	=	5	,	Maka	INFO[5]	=	'D'
SAMBUNGAN[5]	=	10	,	Maka	INFO[10]	=	'E'
SAMBUNGAN[10]	=	8	,	Maka	INFO[8]	=	'F'
SAMBUNGAN[8]	=	0	,	Maka	Akhir Linked List		

Dari contoh diatas diperoleh untai '**ABCDEF**'

SINGLE LINGKED LIST

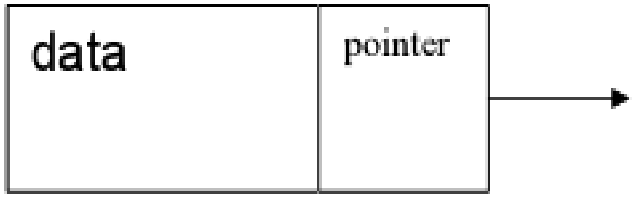
Single linked list adalah yang paling sederhana dari semua varian linked list. Setiap node dalam satu linked list berisi item dan referensi ke item berikutnya dan hanya itu. Di bagian ini, kita akan melihat cara membuat node untuk sinlgled linked list bersama dengan fungsi untuk berbagai jenis penyisipan, traversal, dan penghapusan.



Setiap node memiliki pointer yang menunjuk ke simpul berikutnya sehingga terbentuk satu untaian, dengan demikian hanya diperlukan sebuah variabel pointer. Susunan berupa untaian semacam ini disebut **Single Linked List.** Biasanya Linked List pada node terakhir akan menunjuk ke NULL, dimana NULL memilik nilai khusus yang artinya tidak menunjuk ke mana-mana.

Bentuk Node
Single Linked List

- Single : field pointer-nya hanya satu dan satu arah,pada akhir node pointernya menunjuk NULL



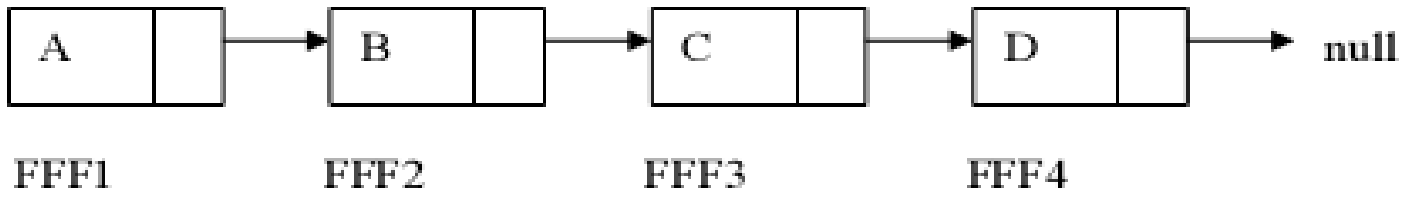
Menempati alamat memori tertentu

- Linked List : node-node tersebut saling terhubung satu sama lain.

Single Linked List (Lanjutan)

Setiap node pada linked list mempunyai field yang berisi pointer ke node berikutnya, dan juga memiliki field yang berisi data.

Node terakhir akan menunjuk ke NULL yang akan digunakan sebagai kondisi berhenti pada saat pembacaan isi linked list.



Pembuatan
Single Linked List

Deklarasi Node :

```
    class Node:
def __init__(self, data):
self.item = data
self.ref = None
```

Keterangan:

- class node kami akan berisi dua variabel item dan ref. Nilai item akan ditetapkan oleh nilai yang diteruskan melalui konstruktor, sedangkan referensi awalnya akan disetel ke nol.

Pembuatan Class pada Single Linked List

Selanjutnya, kita perlu membuat class untuk linked list. Kelas ini akan berisi metode untuk menyisipkan, menghapus, melintasi, dan mengurutkan daftar. Awalnya, kelas hanya akan berisi satu anggota start_node yang akan menunjuk ke simpul awal atau pertama dari daftar. Nilai start_node akan disetel ke nol menggunakan konstruktor karena daftar tertaut akan kosong pada saat pembuatan. Skrip berikut membuat kelas untuk daftar tertaut.

```
class LinkedList:
        def __init__(self):
                self.start_node = None
```

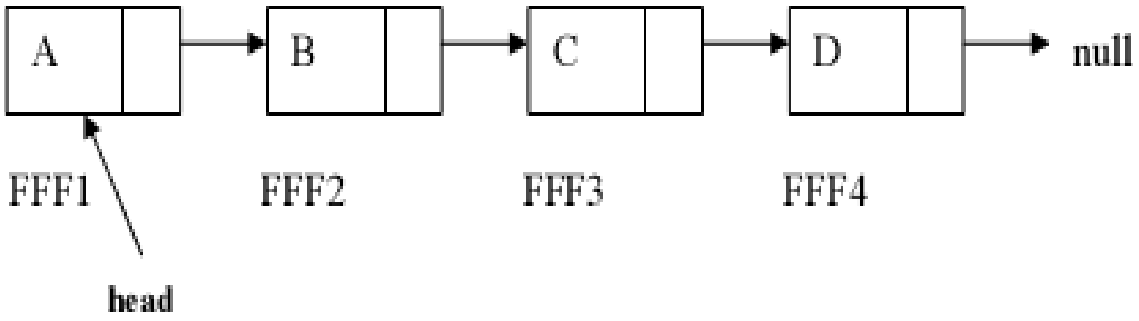
Pembuatan Class pada Single Linked List (lanjutan)

- Selanjutnya, kita perlu membuat class untuk linked list. Kelas ini akan berisi metode untuk menyisipkan, menghapus, melintasi, dan mengurutkan daftar. Awalnya, kelas hanya akan berisi satu anggota start_node yang akan menunjuk ke simpul awal atau pertama dari daftar. Nilai start_node akan disetel ke nol menggunakan konstruktor karena daftar tertaut akan kosong pada saat pembuatan. Skrip berikut membuat kelas untuk daftar tertaut.

```
class LinkedList:
    def __init__(self):
        self.start_node = None
```


Menambahkan item pada Singled Linked List di Awal

- Tergantung pada lokasi tempat Anda ingin menyisipkan item, ada berbagai cara untuk menyisipkan item dalam singled .



- Cara paling sederhana untuk menyisipkan item dalam satu linked list adalah dengan menambahkan item di awal. Fungsi berikut menyisipkan item di awal linked list. Tambahkan fungsi ini ke kelas LinkedList yang kita buat sebelumnya.

Deklarasi Penambahan di Awal pada Single Linked List

```
def insert_at_start(self, data):
        new_node = Node(data)
        new_node.ref=self.start_node
        self.start_node= new_node
```

Pada skrip di atas, kita membuat metode insert_at_start (), metode tersebut menerima satu parameter, yang pada dasarnya adalah nilai item yang ingin kita masukkan. Di dalam metode ini, kita cukup membuat objek dari kelas Node dan menetapkan referensinya ke start_node karena start_node sebelumnya menyimpan simpul pertama, yang setelah penyisipan simpul baru di awal akan menjadi simpul kedua. Oleh karena itu, kami menambahkan referensi start_node ke variabel ref dari node baru. Sekarang karena new_node adalah node pertama, kami menetapkan nilai variabel start_node ke new_node.

Menambahkan Item pada Singled Linked List di Akhir

Memasukkan Item di Akhir Fungsi berikut digunakan untuk menambahkan item di akhir linked list.

```
def insert_at_end(self, data):
new_node = Node(data)
if self.start_node is None:
 self.start_node = new_node
return
n = self.start_node
while n.ref is not None:
n= n.ref
n.ref = new_node;
```

Menambahkan Item pada Singled Linked List di Akhir

Dalam skrip di atas, membuat fungsi insert_at_end (), yang menyisipkan elemen di akhir linked list. Nilai item yang ingin disisipkan sebagai argumen ke fungsi. Fungsinya terdiri dari dua bagian. Pertama kita periksa apakah linked list kosong atau tidak, jika linked list kosong, yang harus kita lakukan hanyalah mengatur nilai variabel start_node ke objek new_node. Di sisi lain, jika list sudah berisi beberapa node. Inisialisasi variabel n dengan simpul awal. Kemudian melakukan iterasi melalui semua node dalam list menggunakan loop sementara. Perulangan berakhir saat mencapai simpul terakhir. Kemudian menetapkan referensi node terakhir ke new_node yang baru dibuat. Tambahkan fungsi insert_at_end () ke kelas LinkedList.

Menambahkan Item Setelah Item Lainnya

Menambahkan item setelah item lain dalam singled linked list. Untuk melakukannya, kita bisa menggunakan fungsi insert_after_item () seperti yang didefinisikan di bawah ini:

```
def insert_after_item(self, x, data):
    n = self.start_node
    print(n.ref)
    while n is not None:
            if n.item == x:
               break
            n = n.ref
            if n is None:
                print("item not in the list")
            else:
    new_node = Node(data)
    new_node.ref = n.ref
    n.ref = new_node
```


Menambahkan Item Setelah Item Lainnya (Lanjutan)

Fungsi insert_after_item () menerima dua parameter: x dan data. Parameter pertama adalah item setelah itu Anda ingin memasukkan node baru sedangkan parameter kedua berisi nilai untuk node baru. Mulai dengan membuat variabel baru n dan menugaskan variabel start_node untuk itu. Selanjutnya, menelusuri linked list menggunakan while loop. Perulangan while dijalankan hingga n menjadi None. Selama iterasi, memeriksa apakah nilai yang disimpan di node saat ini sama dengan nilai yang dilewatkan oleh parameter x. Jika perbandingan mengembalikan nilai true, maka perulangan dihentikan. Selanjutnya, jika item ditemukan, variabel n tidak akan menjadi None. Referensi new_node disetel ke referensi yang disimpan oleh n dan referensi n disetel ke kode_baru. Tambahkan fungsi insert_after_item () ke kelas LinkesList.

Menambahkan Item Setelah Item Lainnya (Lanjutan)

```
def insert_before_item(self, x, data):
    if self.start_node is None:
      print("List has no element")
      return
    if x == self.start_node.item:
      new_node = Node(data)
      new_node.ref = self.start_node
      self.start_node = new_node
      return
    n = self.start_node print(n.ref)
    while n.ref is not None:
      if n.ref.item == x:
        break
      n = n.ref
      if n.ref is None:
        print("item not in the list")
      else:
        new_node = Node(data)
        new_node.ref = n.ref
        n.ref = new_node
```


Menambahkan Linked List

Meskipun kita dapat menambahkan item satu per satu menggunakan salah satu fungsi penyisipan. Mari buat fungsi yang meminta pengguna untuk memasukkan jumlah elemen di node dan kemudian elemen individu dan memasukkan elemen itu ke dalam linked list.

```
def make_new_list(self):
    nums = int(input("How many nodes do you want to create: "))
    if nums == 0:
      return
    for i in range(nums):
      value = int(input("Enter the value for the node:"))
      self.insert_at_end(value)
```


Menambahkan Linked List (Lanjutan)

Dalam skrip di atas, fungsi make_new_list () pertama-tama meminta pengguna untuk jumlah item dalam daftar. Selanjutnya menggunakan for-loop, pengguna diminta untuk memasukkan nilai untuk setiap node, yang kemudian dimasukkan ke dalam daftar tertaut menggunakan fungsi insert_at_end ().

Menghapus Element di Awal

Menghapus elemen atau item dari awal daftar tertaut sangatlah mudah. Kita harus mengatur referensi start_node ke node kedua yang dapat kita lakukan hanya dengan menetapkan nilai referensi dari node awal (yang menunjuk ke node kedua) ke node awal seperti yang ditunjukkan di bawah ini:

```
def delete_at_start(self):
    if self.start_node is None:
        print("The list has no element to delete")
        return
    self.start_node = self.start_node.ref
```

Pada skrip di atas, periksa dulu apakah daftar itu kosong atau tidak. Jika daftar kosong tampilkan pesan bahwa daftar tidak memiliki elemen untuk dihapus. Jika tidak, tetapkan nilai start_node.ref ke start_node tersebut. Start_node sekarang akan mengarah ke elemen kedua. Tambahkan fungsi delete_at_start () ke class LinkedList.

Menghapus Element di Akhir

Untuk menghapus elemen dari akhir daftar, kita hanya perlu mengulang melalui linked list sampai elemen terakhir kedua, dan kemudian perlu mengatur referensi elemen terakhir kedua ke none, yang akan mengubah elemen terakhir kedua menjadi elemen terakhir. Script untuk fungsi delete_at_end adalah sebagai berikut:

```
def delete_at_end(self):
    if self.start_node is None:
        print("The list has no element to delete")
        return
        n = self.start_node
        while n.ref.ref is not None:
        n = n.ref
        n.ref = None
```

Menghapus Berdasarkan Nilai Element

Untuk menghapus elemen berdasarkan nilai, pertama-tama kita harus menemukan node yang berisi item dengan nilai yang ditentukan dan kemudian menghapus node tersebut. Menemukan item dengan nilai yang ditentukan sangat mirip dengan pencarian item. Setelah item yang akan dihapus ditemukan, referensi node sebelum item diatur ke node yang ada setelah item dihapus.

```
def delete_element_by_value(self, x):
    if self.start_node is None:
      print("The list has no element to delete")
      return
# Deleting first node
    if self.start_node.item == x:
      self.start_node = self.start_node.ref
      return
      n = self.start_node
   while n.ref is not None
if n.ref.item == x:
       break
       n = n.ref
    if n.ref is None:
      print("item not found in the list")
    else:
      n.ref = n.ref.ref
```


Contoh program single linked list

```
class node:
    def __init__(self,initdata):
        self.data = initdata
        self.next = None
    def getdata (self):
        return self.data
    def getnext(self):
        return self.next
    def setdata (self,newdata):
        self.data = newdata
    def setnext (self,newnext):
        self.next = newnext
```

→

```
class orderedlist:
    def __init__(self):
        self.head = None

    def show (self):
        current = self.head
        print("Head -> ", end = "")
        while current != None:
            print(current.getdata(), end="-> ")
            current = current.getnext()
        print("None")
```

→

```
    def isempty(self):
        return self.head == None

    def add(self,item):
        temp = node(item)
        temp.setnext(self.head)
        self.head = temp

    def size(self):
        current = self.head
        count = 0
        while current != None :
```

```
        count += 1
        current = current.getnext()
    return count


def search(self):
    current = self.head
    s = int(input("masukkan data yang akan dicari : "))
    found = False
    while current != None and not found:
        if current.getdata() == s:
            found = True
        else:
```

```
            current = current.getnext()
    if found == True :
        print("Data ditemukan")
    else :
        print("Data tidak ditemukan")


def remove (self):
    current = self.head
    s = int(input("masukkan data yang akan dihapus : "))
    previous = None
    found = False
    while not found and current != None:
```



```
    if current.getdata() == s:
        found = True
    else:
        previous = current
        current = current.getnext()
if found == True :
    if previous == None:
        self.head = current.getnext()
    else:
        previous.setnext(current.getnext())
    print("Data sudah dihapus")
else :
        print("Data yang akan dihapus tidak ditemukan")
```

```
def ganjil(self,item):
    for i in range(len(item)):
        if item[i] % 2 == 1:
            ganjil.append(item[i])
    ol.sort(ganjil)
    print(ganjil)

def genap(self,item):
    for i in range(len(item)):
```

```
            if item[i] % 2 == 0:
                genap.append(item[i])
    ol.sort(genap)
    print(genap)

def sort(self,item):
    for x in range(len(item)-1, 0, -1):
        for y in range(x):
            if item[y] > item[y+1]:
                item[y], item[y+1] = item[y+1], item[y]
def tambah(self,item):
    pil = "y"
```



```
while pil == "y":
    tam = int(input("masukkan data : "))
    lis.append(tam)
    pil = str(input("apakah anda mau memasukkan data lagi ? (y/n)"))
ol.ganjil(lis)
ol.genap(lis)
genap.reverse()
merge = ganjil + genap
for i in merge:
    ol.add(i)
```

```
ol = orderedlist()
lis = [170411100001,170411100002,170411100003,170411100004,170411100005,170411100006]
genap=[]
ganjil=[]
merge = ganjil + genap
ol.tambah(ol)
ol.show()
print("panjang datanya adalah",orderedlist.size(ol))
orderedlist.search(ol)
orderedlist.remove(ol)
ol.show()
```

Latihan

(Dikumpulkan pada pertemuan selanjutnya)

Buatlah Ilustrasi / Penggambaran untuk menambah dan menghapus node di posisi tengah pada :

1. Single Linked List dengan Head
2. Single Linked List dengan Head & Trail