

Praktek 22

```
# function untuk membuat node
def buat_node(data):
    return {'data': data, 'next': None}

# menambahkan node di akhir list
def tambah_node(head, data):
    new_node = buat_node(data)
    if head is None:
        return new_node
    current = head
    while current['next'] is not None:
        current = current['next']
    current['next'] = new_node
    return head

# menampilkan linked-list
def cetak_linked_list(head):
    current = head
    print('Head', end=' → ')
    while current is not None:
        print(current['data'], end=' → ')
        current = current['next']
    print("NULL")

# Contoh Penerapan
# Head awal dari linked-list
head = None

# Tambah node
head = tambah_node(head, 10)
head = tambah_node(head, 11)
head = tambah_node(head, 12)

# cetak linked-list
print('Linked-List : ')
cetak_linked_list(head)
```


- **Parameter:** data - nilai yang akan disimpan dalam node.
- **Return:** Mengembalikan sebuah dictionary yang merepresentasikan node, dengan kunci 'data' untuk menyimpan nilai dan kunci 'next' yang diatur ke **None** (menunjukkan bahwa node ini tidak terhubung ke node lain).
- **Parameter:** head - node pertama dari linked list, data - nilai yang akan ditambahkan.
- **buat_node** untuk membuat node baru dengan data yang diberikan.
- **Inisialisasi:** current diatur ke head untuk mulai traversing (menelusuri) linked list.
- Selama **current['next']** tidak **None**, artinya masih ada node berikutnya, maka **current** akan bergerak ke node berikutnya.
- **Return:** Mengembalikan head yang tidak berubah.

- **Print:** Mencetak 'Head' diikuti dengan panah.
- **Print:** Mencetak nilai dari node saat ini (`current['data']`) diikuti dengan panah.
- Setelah loop selesai, mencetak "NULL" untuk menunjukkan bahwa tidak ada node lagi setelah yang terakhir.
- **Inisialisasi:** `head` diatur ke `None`, menandakan bahwa linked list kosong.
- **Menambahkan node:** Memanggil `tambah_node` untuk menambahkan tiga node dengan nilai 10, 11, dan 12 ke dalam linked list. Setiap kali, `head` diperbarui untuk tetap menunjuk ke node pertama.
- **Menampilkan linked list:** Memanggil `cetak_linked_list` untuk menampilkan isi dari linked list yang telah dibangun.

output

```
Linked-List :  
Head → 10 → 11 → 12 → NULL  
PS D:\python\tugas-PRE-PRAKTIKUM>
```

Praktek 23



```

1  # function untuk membuat node
2  def buat_node(data):
3      return {'data': data, 'next': None}
4
5  # menambahkan node di akhir list
6  def tambah_node(head, data):
7      new_node = buat_node(data)
8      if head is None:
9          return new_node
10     current = head
11     while current['next'] is not None:
12         current = current['next']
13     current['next'] = new_node
14     return head
15
16 # traversal untuk cetak isi linked-list
17 def traversal_to_display(head):
18     current = head
19     print('Head', end=' → ')
20     while current is not None:
21         print(current['data'], end=' → ')
22         current = current['next']
23     print("NULL")
24
25 # traversal untuk menghitung jumlah elemen dalam linked-list
26 def traversal_to_count_nodes(head):
27     count = 0
28     current = head
29     while current is not None:
30         count += 1
31         current = current['next']
32     return count
33
34 # traversal untuk mencari dimana tail (node terakhir)
35 def traversal_to_get_tail(head):
36     if head is None:
37         return None
38     current = head
39     while current['next'] is not None:
40         current = current['next']
41     return current
42
43 # Penerapan
44 head = None
45 head = tambah_node(head, 10)
46 head = tambah_node(head, 15)
47 head = tambah_node(head, 117)
48 head = tambah_node(head, 19)
49
50 # cetak isi linked-list
51 print("Isi Linked-List")
52 traversal_to_display(head)
53
54 # cetak jumlah node
55 print("Jumlah Nodes = ", traversal_to_count_nodes(head))
56
57 # cetak HEAD node
58 print("HEAD Node : ", head['data'])
59
60 # cetak TAIL NODE
61 print("TAIL Node : ", traversal_to_get_tail(head)['data'])

```

- Fungsi `traversal_to_display()` bertujuan untuk **menampilkan isi linked list** dari awal sampai akhir dengan cara menelusuri (traversal) setiap node satu per satu.
- Kita buat variabel `current` untuk mulai penelusuran dari node pertama (head). Selanjutnya kita akan mencetak penanda untuk head ("Head ->") dari linked-list, agar secara visual kita dapat mengetahui node awal.
- potongan kode ini menunjukkan selama `current` belum mencapai akhir dari linked-list (yaitu `current == None`), kita akan terus menelusuri node-node yang ada dalam linked-list.
- selama kita menelusuri tiap node yang ada dalam linked-list, maka cetak tiap node bagian data dari node tersebut, tambahkan tanda `→` untuk menunjukkan adanya hubungan antar node.

Output

```
Isi Linked-List
Head → 10 → 15 → 117 → 19 → NULL
Jumlah Nodes = 4
HEAD Node : 10
TAIL Node : 19
PS D:\python\tugas-PRE-PRAKTIKUM>
```

Praktek 24

```

# membuat node baru
def sisip_depan(head, data):
    new_node = {'data': data, 'next': head}
    return new_node

# menampilkan linked-list
def cetak_linked_list(head):
    current = head
    print('Head', end=' → ')
    while current is not None:
        print(current['data'], end=' → ')
        current = current['next']
    print("NULL")

# Penerapan membuat linked-list awal
head = None
head = sisip_depan(head, 30)
head = sisip_depan(head, 20)
head = sisip_depan(head, 10)

# cetak isi linked-list awal
print("Isi Linked-List Sebelum Penyisipan di Depan")
cetak = cetak_linked_list(head)

# Penyisipan node
data = 99
head = sisip_depan(head, data)

print("\nData Yang Disisipkan : ", data)

# cetak isi setelah penyisipan node baru di awal
print("\nIsi Linked-List Setelah Penyisipan di Depan")
cetak_linked_list(head)

```

- fungsi `sisip_depan(head, data)` adalah sebuah fungsi yang memiliki dua buah parameter yaitu `head` dan `data`. Kedua parameter itu disimpan dalam variabel `new_node` dan bertipe data *dictionary*. Melalui fungsi ini node baru akan dibuat dan selalu diletakkan pada posisi awal *linked-list*.

- **Proses:** Membuat node baru (`new_node`) yang menyimpan `data` dan menunjuk ke `head` yang lama.
- **Inisialisasi:** `head` diatur ke `None`, menandakan linked list kosong.
- **Menyisipkan node baru:** Menyisipkan node dengan nilai 99 di depan linked list, memperbarui `head`.
- **Menampilkan:** Memanggil `cetak_linked_list` untuk menampilkan isi linked list yang telah diperbarui.

Output

```
Isi Linked-List Sebelum Penyisipan di Depan
Head → 10 → 20 → 30 → NULL

Data Yang Disisipkan : 99

Isi Linked-List Setelah Penyisipan di Depan
Head → 99 → 10 → 20 → 30 → NULL
PS D:\python\tugas-PRE-PRAKTIKUM>
```

Praktek 25

```

1  # membuat node baru
2  def sisip_depan(head, data):
3      new_node = {'data': data, 'next': head}
4      return new_node
5
6  # sisip node diposisi mana saja
7  def sisip_dimana_aja(head, data, position):
8      new_node = {'data': data, 'next': None}
9
10     # cek jika posisi di awal pakai fungsi sisip_depan()
11     if position == 0:
12         return sisip_depan(head, data)
13
14     current = head
15     index = 0
16
17     # traversal menuju posisi yang diinginkan dan bukan posisi 0
18     while current is not None and index < position - 1:
19         current = current['next']
20         index += 1
21
22     if current is None:
23         print("Posisi melebihi panjang linked list!")
24         return head
25
26     # ubah next dari node sebelumnya menjadi node baru
27     new_node['next'] = current['next']
28     current['next'] = new_node
29     return head
30
31 ## menampilkan linked-list
32 def cetak_linked_list(head):
33     current = head
34     print('Head', end=' → ')
35     while current is not None:
36         print(current['data'], end=' → ')
37         current = current['next']
38     print("NULL")
39
40 # Penerapan
41 # membuat linked-list awal
42 head = None
43 head = sisip_depan(head, 30)
44 head = sisip_depan(head, 20)
45 head = sisip_depan(head, 10)
46 head = sisip_depan(head, 50)
47 head = sisip_depan(head, 70)
48
49 # cetak isi linked-list awal
50 print("Isi Linked-List Sebelum Penyisipan")
51 cetak = cetak_linked_list(head)
52
53 # Penyisipan node
54 data = 99
55 pos = 3
56 head = sisip_dimana_aja(head, data, pos)
57
58 print("\nData Yang Disisipkan : ", data)
59 print("Pada posisi : ", pos, "")
60
61 # cetak isi setelah penyisipan node baru di awal
62 print("\nIsi Linked-List Setelah Penyisipan di tengah")
63 cetak_linked_list(head)

```

- Fungsi `traversal_to_display()` bertujuan untuk **menampilkan isi linked list** dari awal sampai akhir dengan cara menelusuri (traversal) setiap node satu per satu.
- Fungsi ini hanya memiliki 1 parameter yaitu `head`, yaitu node pertama (awal) dari linked list. Selanjutnya perhatikan kode berikut ini :
- Kita buat variabel `current` untuk mulai penelusuran dari node pertama (`head`). Selanjutnya kita akan mencetak penanda untuk `head` ("`Head ->`") dari linked-list, agar secara visual kita dapat mengetahui node awal.
- potongan kode ini menunjukkan selama `current` belum mencapai akhir dari linked-list (yaitu `current == None`), kita akan terus menelusuri node-node yang ada dalam linked-list.
- selama kita menelusuri tiap node yang ada dalam linked-list, maka cetak tiap node bagian data dari node tersebut, tambahkan tanda `→` untuk menunjukkan adanya hubungan antar node.
- jika kita sudah keluar dari while maka kita sudah berada di ujung dari linked-list (`current == None`) dan semua node sudah tercetak, sehingga kita bisa mencetak nilai `next` dari node terakhir berupa `NULL`. Fungsi `traversal_to_count_nodes(head)` digunakan untuk **menghitung jumlah node** (simpul) dalam sebuah *singly linked-list*. Jadi, berapa banyak data yang ada dalam *linked-list* itulah yang dihitung.

Output

```
Isi Linked-List Sebelum Penyisipan
Head → 70 → 50 → 10 → 20 → 30 → NULL

Data Yang Disisipkan : 99
Pada posisi : 3

Isi Linked-List Setelah Penyisipan di tengah
Head → 70 → 50 → 10 → 99 → 20 → 30 → NULL
PS D:\python\tugas-PRE-PRAKTIKUM>
```

Praktek 26


```

1  # membuat node baru
2  def sisip_depan(head, data):
3      new_node = {'data': data, 'next': head}
4      return new_node
5
6  # sisip node diposisi mana saja
7  def sisip_dimana_aja(head, data, position):
8      new_node = {'data': data, 'next': None}
9
10     # cek jika posisi di awal pakai fungsi sisip_depan()
11     if position == 0:
12         return sisip_depan(head, data)
13
14     current = head
15     index = 0
16
17     # traversal menuju posisi yang diinginkan dan bukan posisi 0
18     while current is not None and index < position - 1:
19         current = current['next']
20         index += 1
21
22     if current is None:
23         print("Posisi melebihi panjang linked list!")
24         return head
25
26     # ubah next dari node sebelumnya menjadi node baru
27     new_node['next'] = current['next']
28     current['next'] = new_node
29     return head
30
31 # menghapus head node dan mengembalikan head baru
32 def hapus_head(head):
33     # cek apakah list kosong
34     if head is None:
35         print("Linked-List kosong, tidak ada yang bisa")
36         return None
37     print(f"\nNode dengan data '{head['data']}' dihapus dari head linked-list")
38     return head['next']
39
40 ## menampilkan linked-list
41 def cetak_linked_list(head):
42     current = head
43     print('Head', end=' → ')
44     while current is not None:
45         print(current['data'], end=' → ')
46         current = current['next']
47     print("NULL")
48
49 # Penerapan
50 # membuat linked-list awal
51 head = None
52 head = sisip_depan(head, 30) # tail
53 head = sisip_depan(head, 20)
54 head = sisip_depan(head, 10)
55 head = sisip_depan(head, 50)
56 head = sisip_depan(head, 70) # head
57
58 # cetak isi linked-list awal
59 print("Isi Linked-List Sebelum Penghapusan")
60 cetak_linked_list(head)
61
62 # Penghapusan head linked-list
63 head = hapus_head(head)
64
65 # cetak isi setelah hapus head linked-list
66 print("Isi Linked-List Setelah Penghapusan Head ")
67 cetak_linked_list(head)

```

- **Proses:** Membuat node baru (**new_node**) yang menyimpan **data** dan menunjuk ke **head** yang lama.

- **Inisialisasi:** Membuat node baru (**new_node**) dengan **data** dan **next** diatur ke **None**.

- **Pemeriksaan:** Jika **position** adalah 0, maka menggunakan fungsi **sisip_depan** untuk menyisipkan node di depan.

- **Inisialisasi:** **current** diatur ke **head** untuk mulai traversing linked list, dan **index** diatur ke 0 untuk melacak posisi saat ini.
- **Loop:** Menelusuri linked list hingga mencapai posisi yang diinginkan (sebelum posisi yang ditentukan) atau hingga mencapai akhir list.

- **Pemeriksaan:** Jika **current** adalah **None**, berarti posisi yang diminta melebihi panjang linked list. Mencetak pesan kesalahan dan mengembalikan **head** yang tidak berubah.

- **Menghapus head:** Mencetak data dari node yang dihapus, lalu mengembalikan node berikutnya sebagai head yang baru.

- **Loop:** Menelusuri linked list dan mencetak nilai dari setiap node diikuti dengan panah.

- **Akhir:** Mencetak "NULL" untuk menunjukkan akhir dari linked list.

- **Menyisipkan node:** Menyisipkan lima node dengan nilai 30, 20, 10, 50, dan 70 di depan linked list. Node terakhir yang disisipkan (70) menjadi head

- **Menghapus head:** Memanggil **hapus_head** untuk menghapus node head dari linked list dan memperbarui **head**.

- **Print:** Mencetak judul untuk menunjukkan isi linked list setelah penghapusan.

- **Menampilkan:** Memanggil **cetak_linked_list** untuk menampilkan isi linked list yang telah diperbarui.

Output

```
Head → 70 → 50 → 10 → 20 → 30 → NULL

Node dengan data '70' dihapus dari head linked-list
Isi Linked-List Setelah Penghapusan Head
Head → 50 → 10 → 20 → 30 → NULL
PS D:\python\tugas-PRE-PRAKTIKUM>
```

Praktek 27

```

1  # membuat node baru
2  def sisip_depan(head, data):
3      new_node = {'data': data, 'next': head}
4      return new_node
5
6  # menghapus head node dan mengembalikan head baru
7  def hapus_tail(head):
8      # cek apakah head node == None
9      if head is None:
10         print('Linked-List Kosong, tidak ada yang bisa dihapus!')
11         return None
12
13     # cek node hanya 1
14     if head['next'] is None:
15         print(f"Node dengan data '{head['data']}' dihapus. Linked list sekarang kosong.")
16         return None
17
18     current = head
19     while current['next']['next'] is not None:
20         current = current['next']
21
22     print(f"\nNode dengan data '{current['next']['data']}' dihapus dari akhir.")
23     current['next'] = None
24     return head
25
26  ## menampilkan linked-list
27  def cetak_linked_list(head):
28     current = head
29     print('Head', end=' → ')
30     while current is not None:
31         print(current['data'], end=' → ')
32         current = current['next']
33     print("NULL")
34
35  # Penerapan
36  # membuat linked-list awal
37  head = None
38  head = sisip_depan(head, 30) # tail
39  head = sisip_depan(head, 20)
40  head = sisip_depan(head, 10)
41  head = sisip_depan(head, 50)
42  head = sisip_depan(head, 70) # head
43
44  # cetak isi linked-list awal
45  print("Isi Linked-List Sebelum Penghapusan")
46  cetak_linked_list(head)
47
48  # Penghapusan tail linked-list
49  head = hapus_tail(head)
50
51  # cetak isi setelah hapus Tail linked-list
52  print("Isi Linked-List Setelah Penghapusan Tail ")
53  cetak_linked_list(head)

```

- Jika kita bedah isi dari fungsi `hapus_tail()` diatas, pertama fungsi ini menerima hanya satu parameter saja yaitu `head` yang merepresentasikan node pertama dari *linked-list*.

- Didalam fungsi ini diawali dengan cek apakah *linked-list* saat ini kosong atau tidak, dan jika ya maka cetak pesan bahwa *linked-list* kosong serta kembalikan nilai None, yang artinya list kosong.
- Selanjutnya fungsi ini mengecek apakah *linked-list* hanya memiliki 1 node saja, head['next'] is None berarti node tersebut tidak menunjuk ke node lain, maksudnya tidak ada node berikutnya
- jika benar node tidak memiliki pointer ke node berikutnya, maka cetak pesan bahwa node dengan isi dari node head('data') akan dihapus dari *linked-list*. Jika sudah terhapus maka *linked-list* kosong dan None dikembalikan sebagai nilai baru head
- Selanjutnya jika *linked-list* tidak kosong, dan tidak hanya punya satu node, maka kita perlu melakukan traversal (penelusuran) untuk menemukan tail atau ekor dari *linked-list*

Output

```
Isi Linked-List Sebelum Penghapusan
Head → 70 → 50 → 10 → 20 → 30 → NULL

Node dengan data '30' dihapus dari akhir.
Isi Linked-List Setelah Penghapusan Tail
Head → 70 → 50 → 10 → 20 → NULL
PS D:\python\tugas-PRE-PRAKTIKUM>
```

Praktek 28

```

1  # Praktek 28 : Menghapus node di posisi manapun (tengah)
2  # membuat node baru
3  def sisip_depan(head, data):
4      new_node = {'data': data, 'next': head}
5      return new_node
6
7  # menghapus head node dan mengembalikan head baru
8  def hapus_head(head):
9      # cek apakah list kosong
10     if head is None:
11         print("Linked-List kosong, tidak ada yang bisa")
12         return None
13     print(f"\nNode dengan data '{head['data']}' dihapus dari head linked-list")
14     return head['next']
15
16 # menghapus node pada posisi manapun (tengah)
17 def hapus_tengah(head, position):
18     # cek apakah head node == None
19     if head is None:
20         print('\nLinked-List Kosong, tidak ada yang bisa dihapus!')
21         return None
22
23     # cek apakah posisi < 0
24     if position < 0:
25         print('\nPosisi Tidak Valid')
26         return head
27
28     # Cek apakah posisi = 0
29     if position == 0:
30         print(f"Node dengan data '{head['data']}' dihapus dari posisi 0.")
31         hapus_head(head)
32         return head['next']
33
34     current = head
35     index = 0
36
37     # cari node sebelum posisi target
38     while current is not None and index < position - 1:
39         current = current['next']
40         index += 1
41
42     # Jika posisi yang diinputkan lebih besar dari panjang list
43     if current is None or current['next'] is None:
44         print("\nPosisi melebihi panjang dari linked-list")
45         return head
46
47     print(f"\nNode dengan data '{current['next']['data']}' dihapus dari posisi {position}.")
48     current['next'] = current['next']['next']
49     return head
50
51 ## menampilkan linked-list
52 def cetak_linked_list(head):
53     current = head
54     print('Head', end=' → ')
55     while current is not None:
56         print(current['data'], end=' → ')
57         current = current['next']
58     print("NULL")
59
60 # Penerapan
61 # membuat linked-list awal
62 head = None
63 head = sisip_depan(head, 30) # tail
64 head = sisip_depan(head, 20)
65 head = sisip_depan(head, 10)
66 head = sisip_depan(head, 50)
67 head = sisip_depan(head, 70) # head
68
69 # cetak isi linked-list awal
70 print("Isi Linked-List Sebelum Penghapusan")
71 cetak_linked_list(head)
72
73 # Penghapusan ditengah linked-list
74 head = hapus_tengah(head, 2)
75
76 # cetak isi setelah hapus tengah linked-list
77 print("\nIsi Linked-List Setelah Penghapusan Tengah ")
78 cetak_linked_list(head)

```

- Fungsi ini memiliki tujuan untuk menghapus node pada posisi tertentu (misalnya node ke-1, ke-2, dst.) dalam *linked-list*. Posisi dimulai dari 0 (seperti indeks pada list Python).
- fungsi `hapus_tengah()` menerima dua parameter yaitu `head` dan `position`. Dimana `head` adalah node pertama dalam *linked-list* sementara `position` sebagai posisi index dari node yang ingin dihapus.
- jika `head == None`, maka list kosong, dan tidak ada node yang bisa dihapus. Tetapi jika tidak lanjutkan ke pengecekan validitas berikutnya

output

```
Isi Linked-List Sebelum Penghapusan
Head → 70 → 50 → 10 → 20 → 30 → NULL

Node dengan data '10' dihapus dari posisi 2.

Isi Linked-List Setelah Penghapusan Tengah
Head → 70 → 50 → 20 → 30 → NULL
PS D:\python\tugas-PRE-PRAKTIKUM>
```

Praktek 29

```

1  # Praktek 29 : Membuat Double Linked-List
2  # membuat node baru
3  def buat_node_double(data):
4      return {'data': data, 'prev': head, 'next': None}
5
6  # Menambahkan node baru di awal double linked-list
7  def tambah_node_depan(head, data):
8      new_node = buat_node_double(data)
9      new_node['next'] = head
10     new_node['prev'] = None
11
12     if head is not None:
13         head['prev'] = new_node
14
15     return new_node
16
17 # Mencetak double linked-list dengan traversal maju
18 def cetak_dll(head):
19     current = head
20     print('HEAD', end=' <-> ')
21     while current:
22         print(current['data'], end=' <-> ')
23         current = current['next']
24     print('NULL')
25
26 # Penerapannya
27 # Head awal dari linked-list
28 head = None
29
30 # Tambah Node
31 head = tambah_node_depan(head, 16) # 16
32 head = tambah_node_depan(head, 19) # 16 <-> 19
33
34 # Cetak double linked-list sebelum penyisipan di awal node
35 print("Double Linked-list Awal Sebelum Penyisipan : \n", end='')
36 cetak_dll(head)
37
38 # Tambah Node didepan double linked-list
39 head = tambah_node_depan(head, 22) # 16 <-> 19 <-> 22
40 head = tambah_node_depan(head, 99) # 16 <-> 19 <-> 22 <-> 99
41
42 # Cetak double linked-list setelah penyisipan di awal node
43 print("\nDouble Linked-list Awal Setelah Penyisipan: \n", end='')
44 cetak_dll(head)

```

- Kode diatas adalah sebuah fungsi dengan nama `buat_node_double(data)`, yang menerima satu parameter yaitu `data`. Dimana node baru ini nantinya akan memiliki komponen seperti ini
- Langkah 1 : membuat node baru dengan memanggil fungsi `buat_node_double(data)`
- Langkah 2 : setelah node baru terbuat, ubah nilai dari bagian `next` dari node untuk menyambungkannya dengan `head` sebelumnya.
- Langkah 3 : `prev` dari node baru diberi nilai `None`
- Langkah 4 : jika list tidak kosong, `head['prev']` node sebelumnya arahkan ke node baru
- Langkah 5 : node baru menjadi `head` baru

output

```
Double Linked-list Awal Sebelum Penyisipan :
HEAD <-> 19 <-> 16 <-> NULL

Double Linked-list Awal Setelah Penyisipan:
HEAD <-> 99 <-> 22 <-> 19 <-> 16 <-> NULL
PS D:\python\tugas-PRE-PRAKTIKUM>
```

praktek 30



```
1 # Praktek 30 : Membuat Double Linked-List di Akhir
2 # membuat node baru
3 def buat_node_double(data):
4     return {'data': data, 'prev': head, 'next': None}
5
6 # Menambahkan node baru di akhir double linked-list
7 def tambah_node_akhir(head, data):
8     new_node = buat_node_double(data)
9
10    # Jika list kosong
11    if head is None:
12        return new_node
13
14    # Jika list tidak kosong, cari node terakhir
15    current = head
16    while current['next'] is not None:
17        current = current['next']
18
19    # Sambungkan node terakhir ke node baru
20    current['next'] = new_node
21    new_node['prev'] = current
22
23    return head
24
25
26 # Mencetak double linked-list dengan traversal maju
27 def cetak_dll(head):
28     current = head
29     print('HEAD', end=' <-> ')
30     while current:
31         print(current['data'], end=' <-> ')
32         current = current['next']
33     print('NULL')
34
35 # Penerapannya
36 # Head awal dari linked-list
37 head = None
38
39 # Tambah Node
40 head = tambah_node_depan(head, 16) # 16
41 head = tambah_node_depan(head, 19) # 19 <-> 16
42
43 # Cetak double linked-list sebelum penyisipan di akhir node
44 print("Double Linked-list Sebelum Penyisipan diakhir: \n", end='')
45 cetak_dll(head)
46
47 # Tambah Node diakhir double linked-list
48 head = tambah_node_akhir(head, 22) # 19 <-> 16 <-> 22
49 head = tambah_node_akhir(head, 99) # 19 <-> 16 <-> 22 <-> 19
50
51 # Cetak double linked-list setelah penyisipan di akhir node
52 print("\nDouble Linked-list Setelah Penyisipan diakhir: \n", end='')
53 cetak_dll(head)
```

- head yang merupakan node pertama (awal) dari *linked-list*
- data yang merupakan nilai (isi) yang akan dimasukkan dalam node baru
- untuk membuat node baru dengan memanggil fungsi diluar yaitu `buat_node_double(data)`. Variabel `new_node` menjadi *dictionary* yang mewakili node baru,
- node baru dengan nilai `data = 50`, `prev = None`, dan `next = None`. Nilai `prev` dan `next` masih bernilai `None` karena belum terhubung dengan node lainnya.
- Potongan kode ini berfungsi untuk pengecekan validitas, apakah list dalam kondisi kosong atau tidak, jika iya maka `new_node` yang baru dibuat, langsung menjadi head dari *double linked-list*, dan kemudian node baru dikembalikan `return`
- dengan potongan kode diatas kita akan menjelajahi (traversal) dari node pertama sama akhir. Sebelum penjelajahan dimulai pertama kita buat variabel bantuan yaitu `current` yang akan menampung pointer dari setiap node dalam hal ini karena penjelajahan dilakukan maju jadi yang dibutuhkan hanya bagian pointer `next` saja.
- Selanjutnya `while current['next'] is not None:` atau selama isi dari `next` pointer tidak `None`, maka perintah `current = current['next']` dijalankan, artinya pindah ke node berikutnya. Perulangan ini akan dilakukan sampai dengan `current['next'] == None` atau sudah tidak ada node lagi (node terakhir).
- kita akan menyambungkan pointer `next` dari node terakhir yang awalnya `None` menjadi `new_node`, `current['next'] = new_node`, dan pointer `prev` dari `new_node` menjadi `current`, `new_node['prev'] = current`.

Output

```
Double Linked-list Sebelum Penyisipan diakhir:
HEAD <-> 19 <-> 16 <-> NULL

Double Linked-list Setelah Penyisipan diakhir:
HEAD <-> 19 <-> 16 <-> 22 <-> 99 <-> NULL
```

Praktek 31

```

1  # membuat node baru
2  def buat_node_double(data):
3      return {'data': data, 'prev': head, 'next': None}
4
5  # Menambahkan node baru di awal double linked-list
6  def tambah_node_depan(head, data):
7      new_node = buat_node_double(data)
8      new_node['next'] = head
9      new_node['prev'] = None
10
11     if head is not None:
12         head['prev'] = new_node
13
14     return new_node
15
16 # sisip node diposisi mana saja
17 def sisip_double_dimana_aja(head, data, position):
18     # membuat node baru
19     new_node = buat_node_double(data)
20
21     # cek jika posisi = 0 gunakan fungsi tambah_node_depan()
22     if position == 0:
23         return tambah_node_depan(head, data)
24
25     # cek jika posisi < 0, input tidak valid
26     if position < 0:
27         print('\nPosisi Tidak Valid')
28         return head
29
30     # deklarasi node pertama
31     current = head
32     index = 1
33
34     # traversal menuju posisi yang diinginkan dan bukan posisi 0
35     while current is not None and index < position - 1:
36         current = current['next']
37         index += 1
38
39     # validasi posisi
40     if current is None:
41         print("Posisi melebihi panjang linked list!")
42         return head
43
44     # sisipkan node diantara current dan current.next
45     next_node = current['next']
46     current['next'] = new_node
47     new_node['prev'] = current
48     new_node['next'] = next_node
49     if next_node is not None:
50         next_node['prev'] = new_node
51
52
53     return head
54
55 # Mencetak double linked-list dengan traversal maju
56 def cetak_dll(head):
57     current = head
58     print('HEAD', end=' <-> ')
59     while current:
60         print(current['data'], end=' <-> ')
61         current = current['next']
62     print('NULL')
63
64 # Penerapannya
65 # Head awal dari linked-list
66 head = None
67
68 # Tambah Node
69 head = tambah_node_depan(head, 16) # 16
70 head = tambah_node_depan(head, 19) # 16 <-> 19
71
72 # Cetak double linked-list sebelum penyisipan di awal node
73 print("Double Linked-list Awal Sebelum Penyisipan Tengah: \n", end='')
74 cetak_dll(head)
75
76 # Tambah Node pada posisi mana saja, di double linked-list
77 head = sisip_double_dimana_aja(head, 22, 1) # 19 <-> 22 <-> 16
78 head = sisip_double_dimana_aja(head, 10, 2) # 19 <-> 10 <-> 22 <-> 16
79 head = sisip_double_dimana_aja(head, 30, 3) # 9 <-> 10 <-> 30 <-> 22 <-> 16
80
81 # Cetak double linked-list setelah penyisipan di awal node
82 print("\nDouble Linked-list Awal Setelah Penyisipan Tengah: \n", end='')
83 cetak_dll(head)

```

- head, merupakan node pertama dari *linked-list*
- data, merupakan nilai pada node yang akan disisipkan
- position, merupakan posisi (indeks, mulai dari 0) dimana lokasi node baru akan disisipkan ke ``*linked-list*
- perintah ini digunakan untuk membuat node baru dengan memanggil fungsi `buat_node_double(data)` diluar fungsi ini, dan akan mengembalikan nilai berupa *dictionary*
- node baru dengan nilai data = 50, prev = None, dan next = None. Nilai prev dan next masih bernilai None karena belum terhubung dengan node lainnya.

Output

```
Double Linked-list Awal Sebelum Penyisipan Tengah:
HEAD <-> 19 <-> 16 <-> NULL

Double Linked-list Awal Setelah Penyisipan Tengah:
HEAD <-> 19 <-> 10 <-> 30 <-> 22 <-> 16 <-> NULL
PS D:\python\tugas-PRE-PRAKTIKUM>
```