

1. PRAKTEK 22

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

BAGIAN 1: MEMBUAT NODE

function untuk membuat node

```
def buat_node(data):
    return {'data': data, 'next': None}
```

1. def buat_node(data):
→ Membuat sebuah **fungsi** bernama buat_node yang menerima **data** sebagai input.
2. return {'data': data, 'next': None}
→ Fungsi ini mengembalikan sebuah dictionary yang mewakili satu **node** dalam linked list:
 - o 'data': data → menyimpan nilai dari node.
 - o 'next': None → node ini belum menunjuk ke node selanjutnya (masih akhir/ujung).

```
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
```

BAGIAN 2: MENAMBAHKAN NODE DI AKHIR LIST

menambahkan node di akhir list

```
def tambah_node(head, data):
```

3. Fungsi tambah_node menerima:
 - o head: node pertama dari linked list.
 - o data: nilai baru yang ingin dimasukkan.

```
    new_node = buat_node(data)
```

4. Membuat node baru berisi data yang ingin ditambahkan.

```
    if head is None:
```

```
return new_node
```

5. Jika list masih kosong (head masih None), maka node baru langsung jadi kepala (head).

```
current = head
```

6. Kalau head sudah ada, kita mulai dari awal (current jadi node pertama).

```
while current['next'] is not None:
```

```
    current = current['next']
```

7. Lakukan **perulangan** untuk berjalan ke node berikutnya sampai menemukan node terakhir (yang next-nya None).

```
current['next'] = new_node
```

8. Sambungkan node terakhir ke node baru dengan mengatur 'next'-nya.

```
return head
```

9. Kembalikan node awal (head) agar tetap bisa diakses.

```
# 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")
```

BAGIAN 3: MENAMPILKAN LINKED LIST

```
# menampilkan linked-list
```

```
def cetak_linked_list(head):
```

10. Fungsi untuk mencetak isi dari linked list, mulai dari head.

```
    current = head
```

```
    print('Head', end=' → ')
```

11. Mulai dari head, dan cetak "Head → " sebagai penanda awal.

```
    while current is not None:
```

```
        print(current['data'], end=' → ')
```

```
        current = current['next']
```

12. Selama node belum habis (current tidak None):

- Cetak data di dalam node.
- Lanjut ke node berikutnya (current = current['next']).

```
print("NULL")
```

13. Setelah sampai akhir list, cetak NULL sebagai penanda ujung list.

```
25 # Contoh Penerapan
26 # Head awal dari linked-list
27 head = None
28
29 # Tambah node
30 head = tambah_node(head, 1)
31 head = tambah_node(head, 5)
32 head = tambah_node(head, 7)
33
34 # cetak linked-list
35 print('Linked-List : ')
36 cetak_linked_list(head)
```

BAGIAN 4: CONTOH PENGGUNAAN

Contoh Penerapan

Head awal dari linked-list

head = None

14. Awalnya, linked list masih kosong (head belum ada isinya).

Tambah node

head = tambah_node(head, 1)

head = tambah_node(head, 5)

head = tambah_node(head, 7)

15–17. Tambahkan tiga node ke dalam linked list:

- Pertama berisi 1, jadi kepala.
- Kedua berisi 5, ditambahkan di belakang.
- Ketiga berisi 7, ditambahkan di belakang juga.

cetak linked-list

print('Linked-List : ')

cetak_linked_list(head)

18–19. Cetak seluruh isi dari linked list dari head hingga NULL.

HASIL OUTPUT YANG AKAN MUNCUL

Linked-List :

Head → 10 → 11 → 12 → NULL

Kalau kamu ingin menambahkan fitur seperti **hapus node**, **sisip di tengah**, atau ubah ke versi **berbasis class (OOP)**, tinggal bilang saja!

2. PRAKTEK 23

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

Bagian 1: Membuat Node Baru

function untuk membuat node

def buat_node(data):

return {'data': data, 'next': None}

1. Fungsi buat_node(data) membuat sebuah **node** (kotak) yang berisi:
 - o data: nilainya,
 - o next: sambungan ke node berikutnya, awalnya None karena belum terhubung.

```
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
```

Bagian 2: Menambahkan Node di Akhir

menambahkan node di akhir list

def tambah_node(head, data):

2. Fungsi tambah_node menerima:
 - o head: node pertama dari linked list,
 - o data: nilai baru yang ingin dimasukkan ke dalam linked list.

new_node = buat_node(data)

3. Buat node baru dengan nilai data.

if head is None:

return new_node

4. Kalau list masih kosong (head kosong), node baru langsung jadi head.

current = head

while current['next'] is not None:

current = current['next']

5. Kalau head sudah ada, cari node terakhir (yang next-nya None).

```
current['next'] = new_node
```

6. Sambungkan node terakhir dengan node baru.

```
return head
```

7. Kembalikan head agar tetap bisa digunakan.

```
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")
```

Bagian 3: Menampilkan Isi Linked List

```
# traversal untuk cetak isi linked-list
```

```
def traversal_to_display(head):
```

8. Fungsi ini akan **menelusuri dan menampilkan isi** dari linked list.

```
current = head
```

```
print('Head', end=' → ')
```

9. Mulai dari head, tampilkan tulisan "Head →".

```
while current is not None:
```

```
    print(current['data'], end=' → ')
```

```
    current = current['next']
```

10. Cetak isi setiap node sampai habis (sampai None).

```
print("NULL")
```

11. Tampilkan NULL sebagai akhir list.

```
# traversal untuk menghitung jumlah elemen dalam linked-list
def traversal_to_count_nodes(head):
    count = 0
    current = head
    while current is not None:
        count += 1
        current = current['next']
    return count
```

Bagian 4: Menghitung Jumlah Node

```
# traversal untuk menghitung jumlah elemen dalam linked-list
```

```
def traversal_to_count_nodes(head):
```

```
count = 0
```

12. Buat variabel count untuk menghitung jumlah node.

```
current = head
```

```
while current is not None:
```

```
    count += 1
```

```
    current = current['next']
```

13. Mulai dari head, tambahkan 1 untuk setiap node yang ditemukan.

```
return count
```

14. Kembalikan hasil hitungan jumlah node.

Bagian 5: Mencari Node Terakhir (Tail)

```
# traversal untuk mencari dimana tail (node terakhir)
def traversal_to_get_tail(head):
    if head is None:
        return None
    current = head
    while current['next'] is not None:
        current = current['next']
    return current
```

traversal untuk mencari dimana tail (node terakhir)

```
def traversal_to_get_tail(head):
```

15. Fungsi ini mencari node terakhir (tail).

```
    if head is None:
```

```
        return None
```

16. Kalau list kosong, langsung kembalikan None.

```
    current = head
```

```
    while current['next'] is not None:
```

```
        current = current['next']
```

17. Telusuri dari head sampai menemukan node yang next-nya kosong.

```
    return current
```

18. Kembalikan node terakhir.

```
# Penerapan
head = None
head = tambah_node(head, 10)
head = tambah_node(head, 15)
head = tambah_node(head, 117)
head = tambah_node(head, 19)
```

Bagian 6: Penerapan dan Output

Penerapan

head = None

19. Awalnya list kosong (head = None).

head = tambah_node(head, 10)

head = tambah_node(head, 15)

head = tambah_node(head, 117)

head = tambah_node(head, 19)

20–23. Tambahkan 4 node satu per satu:

- 10
- 15
- 117
- 19

Semua disambung jadi satu linked list.

```
# cetak isi linked-list
print("Isi Linked-List")
traversal_to_display(head)

# cetak jumlah node
print("Jumlah Nodes = ", traversal_to_count_nodes(head))

# cetak HEAD node
print("HEAD Node : ", head['data'])

# cetak TAIL NODE
print("TAIL Node : ", traversal_to_get_tail(head)['data'])
```

Bagian 7: Cetak dan Tampilkan Informasi

cetak isi linked-list

print("Isi Linked-List")

traversal_to_display(head)

24–25. Cetak isi semua node dari awal sampai akhir.

Head → 10 → 15 → 117 → 19 → NULL

cetak jumlah node

```
print("Jumlah Nodes = ", traversal_to_count_nodes(head))
```

26. Tampilkan jumlah total node:

Jumlah Nodes = 4

cetak HEAD node

```
print("HEAD Node : ", head['data'])
```

27. Tampilkan data dari node pertama (head):

HEAD Node : 10

cetak TAIL NODE

```
print("TAIL Node : ", traversal_to_get_tail(head)['data'])
```

28. Tampilkan data dari node terakhir (tail):

TAIL Node : 19

Kesimpulan:

Kamu telah membuat:

- Fungsi untuk **buat node**,
- Tambah node di akhir,
- **Traversal** untuk:
 - Menampilkan isi,
 - Menghitung jumlah,
 - Menemukan node terakhir.

Semua ini sudah membentuk struktur **single linked list** manual menggunakan dictionary.

Kalau mau lanjut, kamu bisa coba buat fitur:

- Hapus node,
- Sisip di tengah,
- Atau buat versi class (OOP). Siap bantu juga!

3. PRAKTEK 24

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

Bagian 1: Fungsi Penyisipan di Depan

membuat node baru

```
def sisip_depan(head, data):
```

```
    new_node = {'data': data, 'next': head}
```

```
    return new_node
```

1. `def sisip_depan(head, data):`
→ Mendefinisikan fungsi untuk menyisipkan node baru **di depan** linked list.
2. `new_node = {'data': data, 'next': head}`
→ Membuat node baru:
 - o data: berisi nilai yang dimasukkan,
 - o next: menunjuk ke head saat ini, agar node baru jadi node pertama (head).
3. `return new_node`
→ Node baru sekarang menjadi kepala (head) dari linked list.

```
6 # menampilkan linked-list
7 def cetak_linked_list(head):
8     current = head
9     print('Head', end=' → ')
10    while current is not None:
11        print(current['data'], end=' → ')
12        current = current['next']
13    print("NULL")
```

Bagian 2: Menampilkan Linked List

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")
```

4. Fungsi cetak_linked_list bertugas **menampilkan isi linked list** dari awal (head) hingga akhir (NULL):
 - Mulai dari head,
 - Cetak setiap isi node (data),
 - Berjalan ke node berikutnya hingga current menjadi None.

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

Bagian 3: Penerapan – Membuat Linked List Awal

```
# Penerapan membuat linked-list awal
```

```
head = None
```

5. Awalnya list kosong (head = None).

```
head = sisip_depan(head, 30)
```

```
head = sisip_depan(head, 20)
```

```
head = sisip_depan(head, 10)
```

6. Tambahkan node satu per satu **di depan**:
 - Tambah 30: head jadi node 30 → NULL,
 - Tambah 20: head jadi node 20 → 30 → NULL,
 - Tambah 10: head jadi node 10 → 20 → 30 → NULL.

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

Bagian 4: Cetak Linked List Sebelum Penyisipan

```
# cetak isi linked-list awal
```

```
print("Isi Linked-List Sebelum Penyisipan di Depan")
```

```
cetak = cetak_linked_list(head)
```

7. Cetak isi linked list sebelum ada penyisipan baru:

Head → 10 → 20 → 30 → NULL

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

print("\nData Yang Disispkan : ", data)
```

Bagian 5: Penyisipan Node Baru di Depan

Penyisipan node

data = 99

head = sisip_depan(head, data)

8. Menyisipkan nilai baru 99 ke paling depan:

- o head sekarang menjadi 99 → 10 → 20 → 30 → NULL.

```
Head → 10 → 20 → 30 → NULL

Data Yang Disispkan : 99

Isi Linked-List Setelah Penyisipan di Depan
Head → 99 → 10 → 20 → 30 → NULL
```

Bagian 6: Tampilkan Data yang Disisipkan

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

9. Cetak nilai 99 yang baru saja disisipkan.

Bagian 7: Cetak Linked List Setelah Penyisipan

cetak isi setelah penyisipan node baru di awal

print("\nIsi Linked-List Setelah Penyisipan di Depan")

cetak_linked_list(head)

10. Cetak ulang isi linked list **setelah** disisipkan:

Head → 99 → 10 → 20 → 30 → NULL

Kesimpulan

Kode ini memperlihatkan:

- Cara **menyisipkan node di awal** linked list,
- Cara menampilkan seluruh isi list dari head ke tail,
- Hasil penyisipan terlihat langsung dari perbandingan **sebelum dan sesudah**.

4. PRAKTEK 25

```
1 # membuat node baru
2 def sisip_depan(head, data):
3     new_node = {'data': data, 'next': head}
4     return new_node
```

BAGIAN 1: Fungsi Menyisipkan Node di Depan

def sisip_depan(head, data):

new_node = {'data': data, 'next': head}

return new_node

1. **sisip_depan(head, data):** fungsi untuk menambahkan node di **paling depan**.
2. Membuat node baru (new_node) berisi:
 - o data: nilai yang diberikan,
 - o next: menunjuk ke node pertama saat ini (head).
3. Node baru dikembalikan dan menjadi head yang baru.

```
# sisip node diposisi mana saja
def sisip_dimana_aja(head, data, position):
    new_node = {'data': data, 'next': None}

    # cek jika posisi di awal pakai fungsi sisip_depan()
    if position == 0:
        return sisip_depan(head, data)

    current = head
    index = 0
```

BAGIAN 2: Fungsi Menyisipkan Node di Posisi Tertentu

def sisip_dimana_aja(head, data, position):

new_node = {'data': data, 'next': None}

4. Fungsi sisip_dimana_aja() akan menyisipkan node di **posisi yang ditentukan** (bukan hanya di awal).
5. Buat new_node berisi data, dan next awalnya kosong (None).

if position == 0:

return sisip_depan(head, data)

6. Kalau posisi yang diinginkan adalah 0 (di depan), pakai fungsi sisip_depan() saja.

current = head

index = 0

7. Siapkan current untuk menelusuri list, mulai dari head.

8. Gunakan index untuk mencatat posisi saat ini.

while current is not None and index < position - 1:

current = current['next']

index += 1

9. Loop berjalan untuk menemukan node **sebelum posisi yang dituju**.

- o Misal posisi yang dituju = 3, maka loop berhenti di node ke-2 (index = 2).

if current is None:

print("Posisi melebihi panjang linked list!")

return head

10. Jika posisi terlalu besar (melebihi panjang list), cetak pesan peringatan dan **jangan sisipkan** apa pun.

new_node['next'] = current['next']

current['next'] = new_node

11. Sambungkan new_node ke node setelahnya,

12. Lalu, sambungkan node sebelumnya (current) ke new_node.

→ Proses sisip selesai.

return head

13. Kembalikan head agar list tetap utuh.

```
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
```

```

## 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")

```

BAGIAN 3: 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")
```

14. Fungsi ini akan **menampilkan isi linked list dari depan hingga akhir**.

15. Loop mencetak setiap node sampai current menjadi None.

```

# Penerapan
# membuat linked-list awal
head = None
head = sisip_depan(head, 30)
head = sisip_depan(head, 20)
head = sisip_depan(head, 10)
head = sisip_depan(head, 50)
head = sisip_depan(head, 70)

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

```

BAGIAN 4: Penerapan – Membuat Linked List Awal

```
# Penerapan
```

```
# membuat linked-list awal
```

```
head = None
```

```
head = sisip_depan(head, 30)
```

```
head = sisip_depan(head, 20)
```

```
head = sisip_depan(head, 10)
```

```
head = sisip_depan(head, 50)
```

```
head = sisip_depan(head, 70)
```

16. Awalnya linked list kosong (head = None).
- 17–21. Tambahkan 5 node satu per satu di depan:
 - 70 → 50 → 10 → 20 → 30 → NULL

BAGIAN 5: Cetak Linked List Sebelum Penyisipan

```
print("Isi Linked-List Sebelum Penyisipan")
```

```
cetak = cetak_linked_list(head)
```

22. Cetak isi linked list **sebelum penyisipan node baru**.

BAGIAN 6: Proses Penyisipan

```
data = 99
```

```
pos = 3
```

```
head = sisip_dimana_aja(head, data, pos)
```

23. Siapkan data 99 untuk disisipkan.
24. Tentukan posisi (pos = 3), artinya data 99 akan disisipkan setelah node ke-2 (pada index ke-3).
25. Panggil sisip_dimana_aja() untuk menyisipkan node tersebut.

```
# Penyisipan node
data = 99
pos = 3
head = sisip_dimana_aja(head, data, pos)

print("\nData Yang Disisipkan : ", data)
print("Pada posisi : ", pos, "")

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

BAGIAN 7: Tampilkan Info Penyisipan

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

```
print("Pada posisi : ", pos, "")
```

- 26–27. Tampilkan nilai yang disisipkan dan posisinya.

BAGIAN 8: Cetak Linked List Setelah Penyisipan

```
print("\nIsi Linked-List Setelah Penyisipan di tengah")
```

```
cetak_linked_list(head)
```

- 28–29. Cetak isi list setelah penyisipan:
Jika sebelumnya:

Head → 70 → 50 → 10 → 20 → 30 → NULL

Maka sesudah penyisipan 99 di posisi ke-3:

Head → 70 → 50 → 10 → 99 → 20 → 30 → NULL

KESIMPULAN

Fungsi `sisip_dimana_aja()` bisa menyisipkan node:

- Di awal (posisi 0),
- Di tengah mana saja,
- Dan menolak jika posisi terlalu besar.

6. PRAKTEK 26

```
1 # membuat node baru
2 def sisip_depan(head, data):
3     new_node = {'data': data, 'next': head}
4     return new_node
```

BAGIAN 1: Fungsi `sisip_depan`

```
def sisip_depan(head, data):
    new_node = {'data': data, 'next': head}
    return new_node
```

1. Fungsi `sisip_depan()` digunakan untuk **menyisipkan node di depan**.
2. `new_node` adalah dictionary (objek node) yang menyimpan:
 - 'data': nilai yang diberikan,
 - 'next': menunjuk ke head lama (node sebelumnya).
3. Fungsi mengembalikan node baru yang menjadi head sekarang.

```
# sisip node diposisi mana saja
def sisip_dimana_aja(head, data, position):
    new_node = {'data': data, 'next': None}

    # cek jika posisi di awal pakai fungsi sisip_depan()
    if position == 0:
        return sisip_depan(head, data)

    current = head
    index = 0

    # traversal menuju posisi yang diinginkan dan bukan posisi 0
    while current is not None and index < position - 1:
        current = current['next']
        index += 1
```

BAGIAN 2: Fungsi `sisip_dimana_aja`


```
def sisip_dimana_aja(head, data, position):
```

```
    new_node = {'data': data, 'next': None}
```

4. Membuat node baru (new_node) untuk disisipkan di posisi tertentu.

```
    if position == 0:
```

```
        return sisip_depan(head, data)
```

5. Jika posisi yang diminta adalah 0, langsung gunakan fungsi sisip_depan().

```
    current = head
```

```
    index = 0
```

6. Gunakan variabel current untuk menyusuri node, dan index untuk menghitung posisi.

```
    while current is not None and index < position - 1:
```

```
        current = current['next']
```

```
        index += 1
```

7. Loop ini akan berjalan hingga current berada **sebelum** posisi yang dituju.

- o Misalnya position = 3, maka current akan berada di posisi ke-2 (karena index < 2).

```
    if current is None:
```

```
        print("Posisi melebihi panjang linked list!")
```

```
        return head
```

8. Jika posisi melebihi jumlah node dalam list, tampilkan pesan dan **jangan lakukan penyisipan**.

```
    new_node['next'] = current['next']
```

```
    current['next'] = new_node
```

```
    return head
```

9. Hubungkan new_node ke node setelah current.

10. Lalu hubungkan current ke new_node.

11. Return head agar linked list tetap utuh.

```
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
```

```
# menghapus head node dan mengembalikan head baru
def hapus_head(head):
    # cek apakah list kosong
    if head is None:
        print("Linked-List kosong, tidak ada yang bisa")
        return None
    print(f"\nNode dengan data '{head['data']}' dihapus dari head linked-list")
    return head['next']
```

BAGIAN 3: Fungsi hapus_head

```
def hapus_head(head):
```

```
    if head is None:
```

```
        print("Linked-List kosong, tidak ada yang bisa")
```

```
        return None
```

12. Fungsi hapus_head() akan menghapus node paling depan.

13. Cek dulu: jika head kosong (linked list kosong), cetak pesan dan kembalikan None.

```
    print(f"\nNode dengan data '{head['data']}' dihapus dari head linked-list")
```

```
    return head['next']
```

14. Cetak data yang dihapus.

15. Kembalikan head['next'], artinya **node kedua jadi head baru**.

```
## 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")
```

BAGIAN 4: Fungsi cetak_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")
```

16. Fungsi untuk **menampilkan isi linked list** dari depan sampai akhir.

17. Gunakan loop untuk cetak satu per satu data node hingga habis (None).

```

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)

```

BAGIAN 5: Penerapan (Main Program)

head = None

head = sisip_depan(head, 30) # tail

head = sisip_depan(head, 20)

head = sisip_depan(head, 10)

head = sisip_depan(head, 50)

head = sisip_depan(head, 70) # head

18. Awalnya, head = None (linked list kosong).

19. Tambahkan node dari belakang ke depan (karena pakai sisip_depan()):

- Hasil akhir:
- Head → 70 → 50 → 10 → 20 → 30 → NULL

print("Isi Linked-List Sebelum Penghapusan")

cetak_linked_list(head)

20. Cetak isi linked list **sebelum node pertama dihapus**.

head = hapus_head(head)

21. Hapus node paling depan (70), dan head sekarang menunjuk ke 50.

print("Isi Linked-List Setelah Penghapusan Head ")

cetak_linked_list(head)

22. Cetak isi linked list **setelah node head dihapus**.

OUTPUT YANG DITAMPILKAN

Isi Linked-List Sebelum Penghapusan

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

6. PRAKTEK 27

```
1 # membuat node baru
2 def sisip_depan(head, data):
3     new_node = {'data': data, 'next': head}
4     return new_node
```

FUNGSI sisip_depan()

def sisip_depan(head, data):

new_node = {'data': data, 'next': head}

return new_node

1. Fungsi ini menyisipkan node di depan dari linked list.
2. data adalah nilai yang ingin disimpan.
3. Node baru (new_node) akan menunjuk ke head lama.
4. Fungsi mengembalikan node baru sebagai head yang baru.
5. FUNGSI hapus_tail()

```
# menghapus head node dan mengembalikan head baru
def hapus_tail(head):
    # cek apakah head node == None
    if head is None:
        print('Linked-List Kosong, tidak ada yang bisa dihapus!')
        return None

    # cek node hanya 1
    if head['next'] is None:
        print(f"Node dengan data '{head['data']}' dihapus. Linked list sekarang kosong.")
        return None

    current = head
    while current['next']['next'] is not None:
        current = current['next']

    print(f"\nNode dengan data '{current['next']['data']}' dihapus dari akhir.")
    current['next'] = None
    return head
```

```
def hapus_tail(head):
```

5. Fungsi ini untuk menghapus node paling akhir (tail).

if head is None:

```
    print('Linked-List Kosong, tidak ada yang bisa dihapus!')
```

```
    return None
```

6. Jika linked list kosong (head = None), tampilkan pesan dan kembalikan None.

if head['next'] is None:

```
    print(f'Node dengan data '{head['data']}' dihapus. Linked list sekarang kosong.")
```

```
    return None
```

7. Jika hanya ada satu node saja, maka node itu dihapus dan linked list jadi kosong.

```
current = head
```

while current['next']['next'] is not None:

```
    current = current['next']
```

8. current digunakan untuk menelusuri node.

9. Loop ini berjalan hingga current berada di node sebelum tail (dua langkah sebelum None).

```
print(f'\nNode dengan data '{current['next']['data']}' dihapus dari akhir.")
```

```
current['next'] = None
```

```
return head
```

10. Cetak node mana yang dihapus.

11. Putuskan koneksi ke node terakhir (current['next'] = None) — sekarang dia menjadi tail.

12. Kembalikan head supaya linked list tetap bisa diakses.

```
## 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")
```

FUNGSI cetak_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")

```

13. Menampilkan seluruh isi linked list dari awal hingga akhir (NULL).

14. Gunakan loop untuk cetak data dari setiap node satu per satu.

```

# Penerapan
# membuat linked-list awal
head = None
head = sisip_depan(head, 30) # tail
head = sisip_depan(head, 20)
head = sisip_depan(head, 10)
head = sisip_depan(head, 50)
head = sisip_depan(head, 70) # head

# cetak isi linked-list awal
print("Isi Linked-List Sebelum Penghapusan")
cetak_linked_list(head)

# Penghapusan tail linked-list
head = hapus_tail(head)

# cetak isi setelah hapus Tail linked-list
print("Isi Linked-List Setelah Penghapusan Tail ")
cetak_linked_list(head)

```

PENERAPAN (MAIN PROGRAM)

```

head = None
head = sisip_depan(head, 30) # tail
head = sisip_depan(head, 20)
head = sisip_depan(head, 10)
head = sisip_depan(head, 50)
head = sisip_depan(head, 70) # head

```

15. Membuat linked list dengan data $70 \rightarrow 50 \rightarrow 10 \rightarrow 20 \rightarrow 30$.

- Urutannya dari belakang karena disisipkan di depan.

- Jadi 70 adalah head, 30 adalah tail.

```
print("Isi Linked-List Sebelum Penghapusan")
```

```
cetak_linked_list(head)
```

16. Menampilkan isi linked list sebelum dilakukan penghapusan tail.

```
head = hapus_tail(head)
```

17. Menghapus node terakhir (30) dari linked list.

```
print("Isi Linked-List Setelah Penghapusan Tail ")
```

```
cetak_linked_list(head)
```

18. Menampilkan linked list setelah node tail dihapus.

HASIL YANG DITAMPILKAN

Misalnya hasilnya seperti ini:

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

7. 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
```

FUNGSI sisip_depan(head, data)

```
def sisip_depan(head, data):
```

```
    new_node = {'data': data, 'next': head}
```

```
    return new_node
```

1. Membuat **node baru** dengan data.
2. next menunjuk ke head yang lama.
3. Node baru dikembalikan sebagai head baru.

```

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

```

FUNGSI hapus_head(head)

def hapus_head(head):

if head is None:

print("Linked-List kosong, tidak ada yang bisa")

return None

print(f"\nNode dengan data '{head['data']}' dihapus dari head linked-list")

return head['next']

4. Menghapus **node pertama (head)**.
5. Jika kosong, tampilkan pesan dan kembalikan None.
6. Jika tidak kosong, tampilkan data yang dihapus, lalu kembalikan node setelah head.

```

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

```

FUNGSI hapus_tengah(head, position)

def hapus_tengah(head, position):

7. Fungsi ini digunakan untuk **menghapus node di posisi tertentu** (tengah).

if head is None:

print("\nLinked-List Kosong, tidak ada yang bisa dihapus!")

return None

8. Jika linked list kosong, tampilkan pesan dan keluar dari fungsi.

if position < 0:

print("\nPosisi Tidak Valid")

return head

9. Posisi tidak boleh negatif.

if position == 0:

print(f"Node dengan data '{head['data']}' dihapus dari posisi 0.")

hapus_head(head)

return head['next']

10. Jika posisi adalah 0, berarti kita ingin hapus head. Panggil hapus_head().

Catatan penting: hapus_head(head) sudah mengembalikan head['next'], jadi baris return head['next'] ini **tidak tepat**, seharusnya cukup:

return hapus_head(head)

current = head

index = 0

11. Siapkan variabel untuk traversing ke node sebelum node yang mau dihapus.

while current is not None and index < position - 1:

current = current['next']

index += 1

12. Loop untuk mencari node **sebelum posisi target**.

if current is None or current['next'] is None:

print("\nPosisi melebihi panjang dari linked-list")

return head

13. Cek apakah posisi melebihi panjang list.

print(f"\nNode dengan data '{current['next']['data']}' dihapus dari posisi {position}.")

current['next'] = current['next']['next']

return head

14. Hapus node di posisi tersebut dengan **melewatkan** node itu.

15. Kembalikan head.

```

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")

```

FUNGSI cetak_linked_list(head)

```
def cetak_linked_list(head):
```

```
    current = head
```

```
    print('Head', end=' → ')
```

```
    while current is not None:
```

```
        print(current['data'], end=' → ')
```

```
        current = current['next']
```

```
    print("NULL")
```

16. Menampilkan isi linked list dari awal sampai akhir.

PENERAPAN

```
head = None
```

```
head = sisip_depan(head, 30) # tail
```

```
head = sisip_depan(head, 20)
```

```
head = sisip_depan(head, 10)
```

```
head = sisip_depan(head, 50)
```

```
head = sisip_depan(head, 70) # head
```

17. Membuat linked list seperti ini:

70 → 50 → 10 → 20 → 30

```
print("Isi Linked-List Sebelum Penghapusan")
```

```
cetak_linked_list(head)
```

18. Tampilkan isi sebelum penghapusan.

```
head = hapus_tengah(head, 2)
```

19. Hapus node di posisi ke-2 (yaitu node dengan nilai 10).

```
print("\nIsi Linked-List Setelah Penghapusan Tengah ")
```

```
cetak_linked_list(head)
```

20. Cetak isi linked list setelah penghapusan.

OUTPUT YANG DITAMPILKAN

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

PENINGKATAN YANG DISARANKAN

Di bagian ini:

```
if position == 0:
```

```
    print(f"Node dengan data '{head['data']}' dihapus dari posisi 0.")
```

```
    hapus_head(head)
```

```
    return head['next']
```

Harusnya cukup:

```
if position == 0:
```

```
    return hapus_head(head)
```

Alif Akbar Fane

24241072

Struktur Data