

TUGAS STRUKTUR DATA

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
```

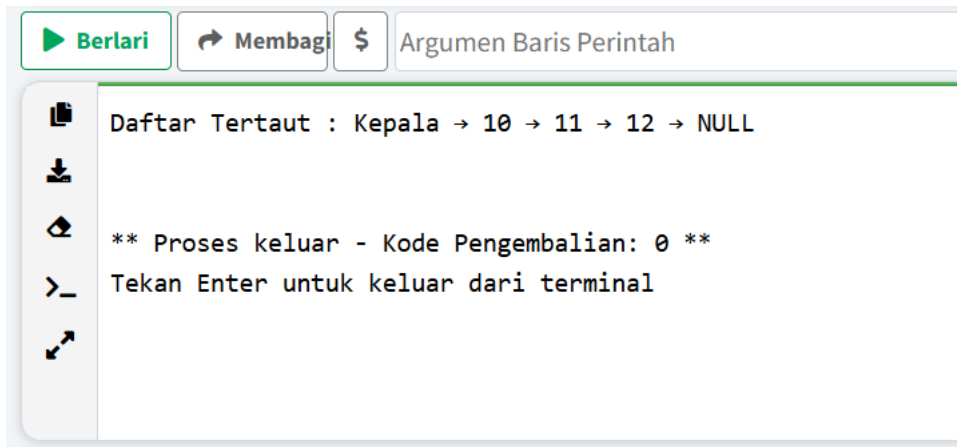
```
# menampilkan linked-list
def cetak_linked_list(head):
    current = head
    print('Head', end=' → ')
    while current is not None:
        print(current['data'], end=' → ')
        current = current['next']
    print("NULL")

# 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)
```

Hasilnya :



```
▶ Berlari  ↻ Membagi  $ Argumen Baris Perintah

Daftar Tertaut : Kepala → 10 → 11 → 12 → NULL

** Proses keluar - Kode Pengembalian: 0 **

Tekan Enter untuk keluar dari terminal
```

Penjelasannya :

Baris 1 & 2 : Membuat node baru berupa dictionary dengan data dan penunjuk next ke node berikutnya (awalnya None).

Baris 3 & 4 : Buat node baru dengan data yang diberikan.

Baris 5 & 6 : Jika linked list masih kosong, node baru jadi kepala (head).

Baris 7, 8 & 9 : Telusuri hingga node terakhir (next adalah None).

Baris 10 & 11 : Hubungkan node terakhir ke node baru, lalu kembalikan head.

Baris 12, 13 & 14 : Mulai mencetak dari head, tampilkan "Head →".

Baris 15, 16 & 19 : Cetak nilai tiap node sampai habis (current jadi None).

Baris 20 : Cetak penanda akhir linked list.

Baris 21 : Inisialisasi linked list kosong.

Baris 22, 23 & 24 : Tambahkan 3 node berturut-turut: 10 → 11 → 12.

Baris 25 & 26 : Tampilkan isi linked list di konsol.

Praktek 23

```
# 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

# traversal untuk cetak isi linked-list
def traversal_to_display(head):
    current = head
    print('Head', end=' → ')
    while current is not None:
        print(current['data'], end=' → ')
        current = current['next']
    print("NULL")
```

```
# 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

# 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
```

```

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

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

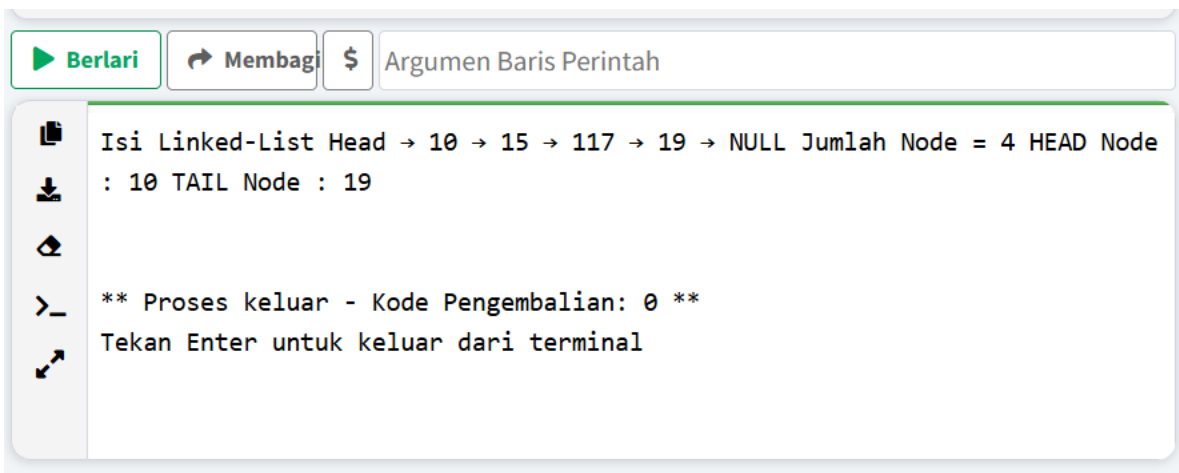
# cetak jumlah node
print("Jumlah Nodes = ", traversal_to_count_nodes(head))

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

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

```

Hasilnya :



The screenshot shows a terminal window with a header bar containing buttons for 'Berlari', 'Membagi', and a search icon, followed by a text input field labeled 'Argumen Baris Perintah'. The terminal output is as follows:

```

Isi Linked-List Head → 10 → 15 → 117 → 19 → NULL Jumlah Node = 4 HEAD Node
: 10 TAIL Node : 19

** Proses keluar - Kode Pengembalian: 0 **
Tekan Enter untuk keluar dari terminal

```

Penjelasannya :

Baris 1 – 2 : Membuat fungsi `buat_node(data)` yang menghasilkan sebuah node berupa dictionary dengan key 'data' untuk menyimpan nilai, dan 'next' untuk penunjuk ke node berikutnya (awalnya None).

Baris 3 – 4 : Fungsi `tambah_node()` memanggil `buat_node(data)` untuk membuat node baru dari data yang diberikan.

Baris 5 – 6 : Jika head masih None (linked list kosong), node baru langsung dikembalikan sebagai head.

Baris 7 – 9 : Jika list sudah berisi, lakukan perulangan untuk mencari node terakhir (node dengan next = None).

Baris 10 – 11 : Setelah node terakhir ditemukan, next-nya dihubungkan ke node baru. Lalu, head dikembalikan sebagai hasil fungsi.

Baris 12 – 13 : Fungsi `traversal_to_display()` mencetak teks "Head →" untuk menandai awal linked list.

Baris 14 – 16 : Menelusuri node satu per satu dan mencetak data setiap node.

Baris 17 : Setelah mencapai node terakhir, cetak "NULL" sebagai penanda akhir list.

Baris 18 – 19 : Fungsi `traversal_to_count_nodes()` memulai perhitungan jumlah node dengan count = 0.

Baris 20 – 22 : Telusuri node satu per satu dan tambahkan 1 ke count untuk setiap node yang dilewati.

Baris 23 : Mengembalikan hasil akhir dari jumlah node.

Baris 24 – 25 : Fungsi `traversal_to_get_tail()` mengecek apakah list kosong. Jika iya, kembalikan None.

Baris 26 – 28 : Jika list tidak kosong, telusuri hingga next adalah None, artinya itu adalah tail.

Baris 29 : Mengembalikan node terakhir (tail).

Baris 30 : Inisialisasi linked list kosong, yaitu head = None.

Baris 31 – 34 : Menambahkan 4 node ke dalam linked list secara berurutan: 10 → 15 → 117 → 19.

Baris 35 : Mencetak teks "Isi Linked-List" sebagai penanda sebelum isi list ditampilkan.

Baris 36 : Memanggil fungsi `traversal_to_display()` untuk mencetak isi list dari awal sampai akhir.

Baris 37 : Mencetak jumlah node dengan memanggil `traversal_to_count_nodes(head)`.

Baris 38 : Mencetak isi dari head node menggunakan `head['data']`.

Baris 39 : Mencetak isi dari tail node dengan memanggil `traversal_to_get_tail(head)['data']`.

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)
```

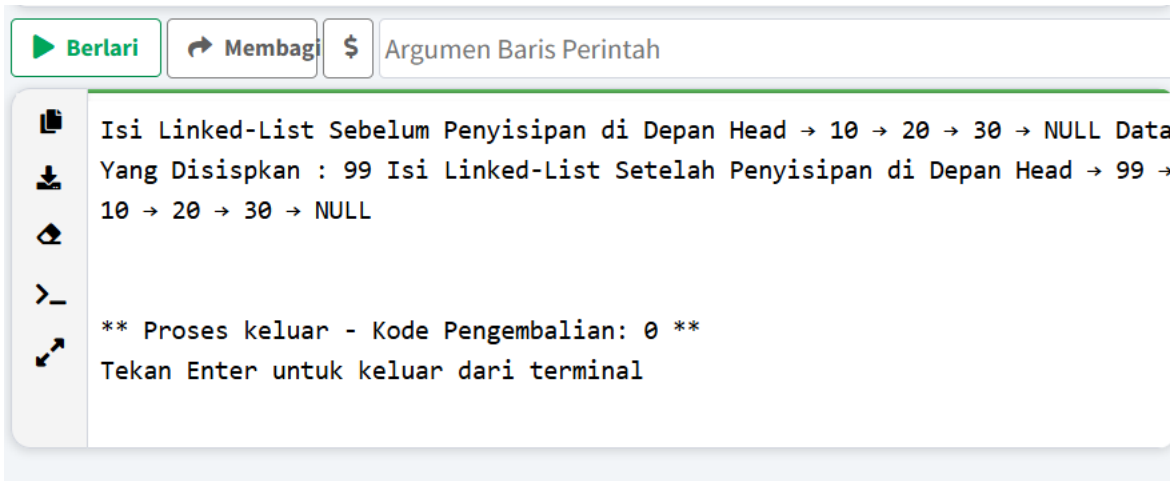
```
# cetak isi linked-list awal
print("Isi Linked-List Sebelum Penyisipan di Depan")
cetak = cetak_linked_list(head)

# 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)
```

Hasilnya :



```
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  
  
** Proses keluar - Kode Pengembalian: 0 **  
Tekan Enter untuk keluar dari terminal
```

Penjelasannya :

Baris 1 – 2 : Membuat fungsi `sisip_depan()` yang menerima `head` dan `data`. Node baru dibuat dengan `data` yang diberikan dan `next` menunjuk ke `head` lama (jadi node ini akan jadi node pertama sekarang).

Baris 3 : Fungsi `sisip_depan()` mengembalikan `new_node` sebagai `head` yang baru.

Baris 5 – 6 : Fungsi `cetak_linked_list()` dimulai dengan `current = head`, dan mencetak "`Head →`" sebagai penanda awal.

Baris 7 – 9 : Selama `current` tidak `None`, cetak nilai `current['data']` dan pindah ke node berikutnya (`current = current['next']`).

Baris 10 : Setelah node terakhir, cetak "`NULL`" sebagai penanda akhir dari linked list.

Baris 12 : Inisialisasi `head` sebagai `None`, artinya linked list masih kosong.

Baris 13 – 15 : Menambahkan 3 node ke depan list: `30 → 20 → 10`, menggunakan `sisip_depan()`. Karena penyisipan selalu di depan, urutan jadi `10 → 20 → 30`.

Baris 17 : Mencetak teks sebelum isi list ditampilkan.

Baris 18 : Memanggil fungsi `cetak_linked_list(head)` untuk mencetak isi linked list awal.

Baris 21 : Menentukan `data` baru yang ingin disisipkan di depan, yaitu 99.

Baris 22 : Menyisipkan node baru di depan, sehingga urutan menjadi: `99 → 10 → 20 → 30`.

Baris 24 : Menampilkan nilai `data` yang baru disisipkan.

Baris 27 : Mencetak isi linked list setelah penyisipan node baru.

Baris 28 : Memanggil kembali cetak_linked_list(head) untuk melihat hasil akhir list.

Praktek 25

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

# 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
    while current is not None and index < position - 1:
        current = current['next']
        index += 1

    if current is None:
        print("Posisi melebihi panjang linked list!")
        return head

    # ubah next dari node sebelumnya menjadi node baru
    new_node['next'] = 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")
```



```

# 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)

# 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)

```

Hasilnya :

Berlari

Membagi

\$ Argumen Baris Perintah

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

** Proses keluar - Kode Pengembalian: 0 **

Tekan Enter untuk keluar dari terminal

|

Penjelasannya :

Baris 1 – 3 : Fungsi `sisip_depan()` untuk menyisipkan node baru di depan. Node baru berisi data, dan penunjuk next mengarah ke head lama.

Baris 5 – 23 : Fungsi `sisip_dimana_aja()` untuk menyisipkan node di posisi tertentu:

Baris 6 : Buat node baru dengan data dan next default None.

Baris 8 – 9 : Jika posisi yang diminta adalah 0, maka gunakan `sisip_depan()` (artinya node baru jadi kepala).

Baris 11 – 14 : Traversal ke posisi sebelum yang diminta (`position - 1`) untuk mendapatkan node sebelumnya.

Baris 16 – 18 : Jika traversal gagal (posisi terlalu jauh), tampilkan pesan kesalahan dan kembalikan head awal.

Baris 20 – 21 : Sisipkan node baru dengan menyambungkannya di antara node sebelumnya dan node sesudahnya.

Baris 25 – 31 : Fungsi `cetak_linked_list()` untuk mencetak isi linked list dari head hingga akhir:

Baris 26 : Mulai dari head.

Baris 27 – 29 : Selama current belum None, cetak nilai dan pindah ke node berikutnya.

Baris 30 : Cetak "NULL" sebagai penanda akhir list.

Baris 33 : Inisialisasi head = None, artinya linked list masih kosong.

Baris 34 – 38 : Menyisipkan lima node di depan: urutannya menjadi $70 \rightarrow 50 \rightarrow 10 \rightarrow 20 \rightarrow 30$.

Baris 41 : Tampilkan teks sebelum isi linked list.

Baris 42 : Cetak linked list sebelum penyisipan tengah.

Baris 45 – 46 : Menentukan data baru (99) dan posisi penyisipan (`pos = 3`).

Baris 47 : Panggil fungsi `sisip_dimana_aja()` untuk menyisipkan 99 di posisi ke-3.

Baris 49 – 50 : Cetak data dan posisi penyisipan.

Baris 53 : Cetak teks penanda list setelah disisipkan node baru di tengah.

Baris 54 : Cetak hasil akhir linked list setelah disisipkan.

Praktek 26

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

# 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
    while current is not None and index < position - 1:
        current = current['next']
        index += 1

    if current is None:
        print("Posisi melebihi panjang linked list!")
        return head

    # ubah next dari node sebelumnya menjadi node baru
    new_node['next'] = current['next']
    current['next'] = new_node
    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']
```

```

## 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) # 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 head linked-list
head = hapus_head(head)



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

```

Hasilnya :

 Berlari
  Membagi
  Argumen Baris Perintah

 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


 >_
 

** Proses keluar - Kode Pengembalian: 0 **
 Tekan Enter untuk keluar dari terminal
 |

Penjelasannya :

Baris 1 – 3 : Fungsi `sisip_depan()` untuk membuat node baru dan menyisipkannya di awal linked-list. Node baru menunjuk ke head sebelumnya.

Baris 5 – 23 : Fungsi `sisip_dimana_aja()` untuk menyisipkan node baru di posisi tertentu.

Baris 6 : Membuat node baru dengan data dan next default None.

Baris 8 – 9 : Jika posisi yang diminta adalah 0, langsung gunakan fungsi `sisip_depan()`.

Baris 11 – 14 : Traversal ke posisi `position - 1` untuk mendapatkan node sebelumnya.

Baris 16 – 18 : Jika node sebelumnya tidak ditemukan (posisi melebihi panjang list), tampilkan pesan error dan kembalikan head.

Baris 20 – 21 : Node baru dimasukkan di antara node sebelumnya dan sesudahnya.

Baris 25 – 31 : Fungsi `hapus_head()` untuk menghapus node paling depan (head).

Baris 27 : Jika linked list kosong, tampilkan pesan dan return None.

Baris 28 : Tampilkan data node yang dihapus.

Baris 29 : Return node setelah head sebagai head baru.

Baris 33 – 39 : Fungsi `cetak_linked_list()` untuk menampilkan isi linked list dari awal ke akhir.

Baris 34 : Mulai dari head.

Baris 35 – 37 : Cetak data masing-masing node satu per satu.

Baris 38 : Cetak penanda NULL di akhir.

Baris 41 : Inisialisasi `head = None`, linked-list masih kosong.

Baris 42 – 46 : Sisipkan 5 node ke depan: urutannya menjadi $70 \rightarrow 50 \rightarrow 10 \rightarrow 20 \rightarrow 30$

Baris 49 : Cetak teks keterangan sebelum penghapusan head.

Baris 50 : Cetak isi linked list sebelum penghapusan.

Baris 53 : Hapus head node, yaitu node dengan data 70.

Baris 56 : Cetak teks keterangan setelah penghapusan.

Baris 57 : Cetak isi linked list setelah node pertama dihapus.

Praktek 27

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

# 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
```

```
## 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) # 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)

```

hasilnya :

```

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

** Proses keluar - Kode Pengembalian: 0 **
Tekan Enter untuk keluar dari terminal

```

penjelasannya :

Baris 1 & 2: Mendefinisikan fungsi sisip_depan untuk menyisipkan node baru di depan linked list.

Baris 3: Membuat node baru berupa dictionary berisi data dan next yang menunjuk ke head lama.

Baris 4: Mengembalikan node baru sebagai head baru dari linked list.

Baris 6 & 7: Mendefinisikan fungsi hapus_tail untuk menghapus node terakhir dari linked list.

Baris 8 & 9: Jika head bernilai None, tampilkan pesan bahwa linked list kosong dan kembalikan None.

Baris 11 & 12: Jika hanya ada satu node, tampilkan pesan penghapusan dan kembalikan None karena list jadi kosong.

Baris 14: Inisialisasi variabel current sebagai penunjuk ke node awal (head).

Baris 15 & 16: Telusuri hingga menemukan node sebelum tail (node yang next.next-nya adalah None).

Baris 18: Cetak data node yang akan dihapus.

Baris 19: Hapus node terakhir dengan menyetel `current['next']` menjadi `None`.

Baris 20: Kembalikan `head` sebagai `head` baru dari linked list.

Baris 23: Definisikan fungsi `cetak_linked_list` untuk menampilkan isi linked list.

Baris 24: Inisialisasi `current` ke `head`.

Baris 25: Cetak label "Head →" sebagai awalan tampilan.

Baris 26 & 27: Selama masih ada node, cetak data lalu pindah ke `next`.

Baris 28: Cetak "NULL" sebagai akhir dari linked list.

Baris 31: Inisialisasi linked list kosong (`head = None`).

Baris 32–36: Tambahkan node ke depan secara bertahap: `30 → 20 → 10 → 50 → 70`.

Baris 38: Cetak isi linked list sebelum penghapusan tail.

Baris 39: Panggil fungsi `hapus_tail` untuk menghapus node paling akhir.

Baris 41: Cetak isi linked list setelah penghapusan tail.

Praktek 28

```
# Praktek 28 : Menghapus node di posisi manapun (tengah)
# membuat node baru
def sisip_depan(head, data):
    new_node = {'data': data, 'next': head}
    return new_node

# 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']

# menghapus node pada posisi manapun (tengah)
def hapus_tengah(head, position):
    # cek apakah head node == None
    if head is None:
        print('\nLinked-List Kosong, tidak ada yang bisa dihapus!')
        return None
```



```

# cek apakah posisi < 0
if position < 0:
    print('\nPosisi Tidak Valid')
    return head

# Cek apakah posisi = 0
if position == 0:
    print(f"Node dengan data '{head['data']}' dihapus dari posisi 0.")
    hapus_head(head)
    return head['next']

current = head
index = 0

# cari node sebelum posisi target
while current is not None and index < position - 1:
    current = current['next']
    index += 1

```

```

# Jika posisi yang diinputkan lebih besar dari panjang list
if current is None or current['next'] is None:
    print("\nPosisi melebihi panjang dari linked-list")
    return head

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

```

```

# 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 ditengah linked-list
head = hapus_tengah(head, 2)

# cetak isi setelah hapus tengah linked-list
print("\nIsi Linked-List Setelah Penghapusan Tengah ")
cetak_linked_list(head)

```

Hasilnya :

Berlari

Membagi

Argumen Baris Perintah

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

>_

** Proses keluar - Kode Pengembalian: 0 **

Tekan Enter untuk keluar dari terminal

Penjelasannya :

Baris 2 & 3: Mendefinisikan fungsi `sisip_depan` untuk menambahkan node baru di depan linked list.

Baris 4: Membuat node baru berisi data dan penunjuk next ke head.

Baris 5: Mengembalikan node baru sebagai head baru dari linked list.

Baris 7–9: Mendefinisikan fungsi `hapus_head` untuk menghapus node paling depan.

Baris 10–12: Jika linked list kosong, tampilkan pesan dan kembalikan None.

Baris 13: Cetak informasi penghapusan data pada head.

Baris 14: Kembalikan `head['next']` sebagai head baru.

Baris 16: Fungsi `hapus_tengah` untuk menghapus node pada posisi tertentu (misalnya tengah).

Baris 17–19: Jika linked list kosong, tampilkan pesan dan kembalikan None.

Baris 21–23: Jika posisi negatif, tampilkan pesan kesalahan dan batalkan penghapusan.

Baris 25–27: Jika posisi 0, panggil `hapus_head()` dan kembalikan head baru (`head['next']`).

Baris 29–30: Inisialisasi `current` sebagai head dan `index` sebagai penghitung posisi.

Baris 32–34: Telusuri hingga node sebelum posisi target (`index = posisi - 1`).

Baris 36–38: Jika posisi melebihi panjang linked list, tampilkan pesan kesalahan dan batalkan proses.

Baris 40: Cetak informasi bahwa node di posisi tertentu akan dihapus.

Baris 41: Ubah koneksi next untuk melewati node yang dihapus.

Baris 42: Kembalikan head yang sudah diperbarui.

Baris 45–49: Fungsi `cetak_linked_list` untuk menampilkan isi linked list.

Baris 46: Mulai dari head.

Baris 47: Cetak label "Head →" sebagai tanda awal.

Baris 48–49: Cetak data setiap node hingga akhir, kemudian tampilkan "NULL".

Baris 52: Inisialisasi linked list kosong (`head = None`).

Baris 53–57: Tambahkan lima node ke depan: `30 → 20 → 10 → 50 → 70` (jadi head).

Baris 59: Cetak isi linked list sebelum penghapusan.

Baris 60: Hapus node pada posisi ke-2 (node dengan data 10).

Baris 62: Cetak isi linked list setelah penghapusan node di tengah.

