Praktek 22

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

Penjelasan :

Fungsi ini digunakan untuk membuat sebuah node baru dalam bentuk dictionary. Setiap node memiliki dua elemen:

- 'data': menyimpan nilai atau informasi (dalam hal ini data yang diberikan sebagai parameter).
- 'next': menunjuk ke node berikutnya. Awalnya diatur ke None karena node ini belum terhubung ke node lain.

```
# 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
```

Penjelasan :

Fungsi ini menambahkan node baru di akhir linked list. Langkah-langkahnya:

- Membuat node baru dengan buat_node(data).
- Jika head bernilai None, artinya linked list masih kosong, maka node baru langsung menjadi head.
- Jika tidak, dilakukan iterasi dengan pointer current untuk mencapai node terakhir (di mana 'next' adalah None).
- Setelah sampai di node terakhir, 'next' dari node tersebut dihubungkan ke new_node.
- Fungsi mengembalikan head agar tetap menunjuk ke awal 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")
```

Penjelasan :

Fungsi ini digunakan untuk menampilkan seluruh isi linked list dari awal sampai akhir:

- Mulai dari head, ditampilkan kata 'Head → '.
- Selama current tidak None, cetak nilai data dari node dan lanjutkan ke node berikutnya.
- Setelah semua node dicetak, tampilkan "NULL" untuk menandakan akhir dari list.

```
# 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)
```

Penjelasan :

Bagian ini menunjukkan cara penggunaan fungsi-fungsi di atas:

- head diinisialisasi dengan None, menandakan linked list masih kosong.
- Tiga node dengan data 10, 11, dan 12 ditambahkan secara berurutan.
- Kemudian, seluruh isi linked list ditampilkan. Hasil yang dicetak adalah:

Praktek 23 :

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

Penjelasan :

Fungsi ini membuat sebuah node dalam bentuk dictionary. Setiap node memiliki dua bagian:

- 'data': berisi nilai yang akan disimpan dalam node.
- 'next': penunjuk ke node berikutnya, yang awalnya diset ke 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
```

Penjelasan :

Fungsi ini menambahkan node baru ke akhir linked list:

- Pertama, membuat node baru dari data yang diberikan.
- Jika head masih None, berarti list kosong. Node baru menjadi node pertama (head).
- Jika tidak, pointer current digunakan untuk berjalan ke node terakhir (next-nya adalah None).
- Setelah mencapai akhir, node baru dihubungkan ke node terakhir.
- Fungsi mengembalikan head agar linked list tetap bisa diakses dari awal.

```
# 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")
```

Penjelasan :

Fungsi ini mencetak isi seluruh linked list:

- Mulai dari head, lalu iterasi ke seluruh node satu per satu.
- Menampilkan data tiap node diikuti tanda panah.
- Setelah mencapai akhir list (current menjadi None), tampilkan "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
```

Penjelasan :

Fungsi ini menghitung berapa banyak node dalam linked list:

- Inisialisasi penghitung (count) ke 0.
- Lalu, iterasi tiap node, menaikkan count setiap langkah.
- Setelah selesai, nilai count dikembalikan.

```
# 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
```

Penjelasan :

Fungsi ini mengembalikan node terakhir (tail) dari linked list:

- Jika linked list kosong (head is None), kembalikan None.
- Jika tidak, iterasi sampai node terakhir (di mana 'next' adalah None) dan kembalikan node itu.

```
        # Penerapan

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

Penjelasan :

- Membuat linked list dengan 4 node secara bertahap: 10 → 15 → 117 → 19.

- Node baru ditambahkan menggunakan tambah_node.

```
# 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'])
```

Penjelasan :

- Mencetak isi linked list menggunakan fungsi traversal_to_display.
- Menghitung dan menampilkan jumlah node dalam linked list.
- Menampilkan data pada node head (pertama).
- Mengambil node terakhir dan mencetak isinya.

Praktek 24

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

Penjelasan :

Fungsi ini digunakan untuk menyisipkan node baru **di depan (awal)** linked list:

- new_node dibuat sebagai dictionary dengan dua kunci:
  - 'data': menyimpan nilai dari parameter data.
  - 'next': menunjuk ke head yang lama.
- Ini artinya, node baru akan menjadi head yang baru, dan node lama menjadi node kedua.
- Fungsi mengembalikan node baru sebagai head yang 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")
```

Penjelasan :

Fungsi ini untuk mencetak isi linked list:

- Iterasi dimulai dari head.
- Selama node belum None, cetak nilai data dan lanjut ke node berikutnya.
- Setelah selesai, tampilkan "NULL" sebagai akhir dari linked list.

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

Penjelasan :

 Pertama, head diinisialisasi None untuk menandai linked list masih kosong.

 Kemudian tiga node disisipkan **berturut-turut di depan**:

- 30 → linked list: 30
- 20 → linked list: 20 → 30
- 10 → linked list: 10 → 20 → 30

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

Penjelasan :

Mencetak isi linked list saat ini, yaitu: Head → 10 → 20 → 30 → NULL

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

Penjelasan :

 Nilai 99 akan disisipkan di awal.

 Node baru akan menunjuk ke node yang sebelumnya head (10), sehingga linked list menjadi: 99 → 10 → 20 → 30

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

Penjelasan :

Menampilkan informasi bahwa 99 telah disisipkan ke depan linked list.

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

```
cetak_linked_list(head)
```

Prnjelasan :

Menampilkan isi linked list terbaru: Head → 99 → 10 → 20 → 30 → NULL

Praktek 25

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

Penjelasan :

Fungsi ini menyisipkan node baru di awal (depan) linked list:

- Membuat new_node dengan data yang diberikan dan menunjuk ke head saat ini.
- Node baru akan menjadi head yang baru.
- Fungsi ini berguna jika kita ingin menambahkan data di posisi ke-0.

```
# 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)
```

Penjelasan :

 Fungsi ini menyisipkan node baru **di posisi tertentu** dalam linked list.

 Jika position adalah 0, maka penyisipan dilakukan di awal menggunakan fungsi sisip_depan().

```
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
```

Penjelasan :

 Traversal dilakukan untuk mencapai node **sebelum** posisi yang diinginkan (position - 1).

 Misalnya jika ingin menyisipkan di posisi 3, maka traversal berhenti di node indeks 2.

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

Penjelasan :

 Jika traversal mencapai akhir linked list sebelum mencapai posisi yang diinginkan, berarti posisi tersebut **tidak valid**.

 Fungsi mengembalikan head tanpa perubahan dan mencetak pesan peringatan

- Node baru disisipkan dengan mengatur:
  - 'next' dari node baru mengarah ke node setelah current.
  - 'next' dari current diubah untuk menunjuk ke node baru.
- Ini menyisipkan node di antara current dan node setelahnya.

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

Penjelasan :

- Node baru disisipkan dengan mengatur:
  - 'next' dari node baru mengarah ke node setelah current.
  - 'next' dari current diubah untuk menunjuk ke node baru.
- Ini menyisipkan node di antara current dan node setelahnya.

```
# 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")
```

Penjelasan :

 Fungsi ini digunakan untuk mencetak isi linked list dari awal sampai akhir.

 Menampilkan node satu per satu sampai current menjadi None.

 Terakhir mencetak "NULL" sebagai penanda akhir.

```
# 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)
```

Penjelasan :

Membuat linked list awal dengan penyisipan berurutan di depan:

- Urutan akhir setelah semua penyisipan: 70 → 50 → 10 → 20 → 30

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

Penjelasan : Menampilkan isi linked list sebelum penyisipan di posisi tertentu.

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

Penjelasan :

 Menyisipkan data = 99 di posisi ke-3 (berarti setelah indeks 2).

 Node baru akan berada setelah 10, dan sebelum 20.

 Linked list akan menjadi: 70 → 50 → 10 → 99 → 20 → 30

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

Penjelasan :

Menampilkan informasi tentang data dan posisi yang disisipkan.

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

Penjelasan :

Menampilkan isi linked list setelah proses penyisipan selesai.

Praktak 26

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

Penjelasan :

Fungsi ini digunakan untuk **menyisipkan node baru di awal linked list**. Node baru berupa dictionary yang menyimpan `data` dan `next` (penunjuk ke node berikutnya). `next` diarahkan ke `head` lama agar node baru menjadi head baru.

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

    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
        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
```

Penjelasan :

Fungsi ini menyisipkan node baru **di posisi tertentu** dalam linked list:

- Jika `position == 0`, artinya menyisip di awal, maka gunakan fungsi `sisip_depan`.
- Jika `position > 0`, lakukan traversal hingga ke node sebelum posisi target (`position - 1`).
- Jika posisi melebihi panjang list (yaitu node sebelum posisi tidak ditemukan), cetak pesan kesalahan.
- Setelah sampai pada posisi yang tepat, hubungkan node baru dengan node berikutnya, dan node sebelumnya dengan node baru.

```python
# 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']
```

Penjelasan :

Fungsi ini menghapus **node pertama (head)** dari linked list. Jika `head` adalah `None` (list kosong), fungsi mencetak pesan dan mengembalikan `None`. Jika tidak kosong, fungsi mencetak node yang dihapus dan mengembalikan node berikutnya sebagai head baru.

```python
## 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")
```

Penjelsan :

Fungsi ini digunakan untuk **menampilkan isi linked list** dari awal (head) hingga akhir. Ia mencetak setiap data node, diakhiri dengan `"NULL"` untuk menunjukkan akhir list.

```python
# 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
```

Penjelasan :

Kode ini membuat sebuah linked list dengan menyisipkan node dari belakang ke depan. Urutan sisipan akan membuat linked list seperti berikut: 70 → 50 → 10 → 20 → 30 → NULL

Karena setiap elemen disisipkan di depan, maka elemen terakhir yang dimasukkan (70) menjadi head.

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

Penjelsan :

Bagian ini mencetak isi linked list sebelum dilakukan penghapusan. Hasil yang ditampilkan: Head → 70 → 50 → 10 → 20 → 30 → NULL

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

Penjelasan :

Memanggil fungsi `hapus_head()` untuk menghapus node pertama (head), yaitu node dengan nilai `70`. Setelah penghapusan, `head` akan menunjuk ke node `50`.

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

Penjelasan :

Bagian ini mencetak isi linked list setelah head dihapus. Hasil akhir: Head → 50 → 10 → 20 → 30 → NULL

Praktek 27

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

Penjelsan :

Fungsi ini digunakan untuk **menyisipkan node baru di awal (head)** dari linked list. Fungsi menerima dua parameter: `head` (node awal saat ini) dan `data` (nilai yang ingin disisipkan). Node baru akan menunjuk ke `head` lama, lalu node baru dikembalikan sebagai head yang baru. Ini adalah cara sederhana untuk membangun linked list dari depan.

```
# 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
```

Penjelsan :

Fungsi ini bertugas untuk **menghapus node terakhir (tail)** dari linked list. Berikut penjelasan langkah-langkah dalam fungsi ini:

- **Baris 1–3**: Jika linked list kosong (`head == None`), maka tidak ada yang bisa dihapus. Fungsi akan mencetak pesan dan mengembalikan `None`.
- **Baris 5–7**: Jika hanya ada satu node (artinya `head['next'] == None`), maka node tersebut dihapus, dan linked list akan menjadi kosong. Fungsi mengembalikan `None`.
- **Baris 9–12**: Jika ada lebih dari satu node, program akan melakukan perulangan sampai menemukan node **sebelum** node terakhir (`current['next']['next'] == None`). Setelah ditemukan:
    - Data dari node terakhir akan dicetak sebagai konfirmasi penghapusan.
    - Node `current['next']` diubah menjadi `None`, yang berarti node terakhir telah diputus dari list.
    - Fungsi mengembalikan `head` agar srtuktur list tetap utuh

```
## 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")
```

Penjelasan :

Fungsi ini digunakan untuk **mencetak seluruh isi linked list**. Dimulai dari `head`, program akan menelusuri setiap node dan mencetak nilai `data` dari tiap node secara berurutan, dipisahkan dengan panah (→). Traversal berhenti saat node `current` menjadi `None`, dan program mencetak `"NULL"` sebagai penutup.

```
# 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
```

Penjelasan :

Linked list dibentuk dari belakang ke depan, dengan menyisipkan data 30, 20, 10, 50, dan terakhir 70. Karena selalu disisipkan di depan, struktur akhirnya adalah: Head → 70 → 50 → 10 → 20 → 30 → NULL Node dengan data `30` menjadi **tail**, dan node dengan data `70` menjadi **head**.

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

Penjelasan :

Fungsi ini mencetak isi linked list sebelum node tail dihapus. Output-nya akan: Head → 70 → 50 → 10 → 20 → 30 → NULL

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

Penjelasan :

- Fungsi `hapus_tail()` akan menelusuri sampai node sebelum node terakhir (`20`).
- Node terakhir dengan data `30` akan dihapus.
- `current['next']` pada node `20` akan di-set ke `None`, menjadikan node `20` sebagai tail yang baru.

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

Penjelasan :

Fungsi mencetak isi list setelah node `30` dihapus. Struktur sekarang menjadi: Head → 70 → 50 → 10 → 20 → NULL

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
```

Penjelsan :

Fungsi ini digunakan untuk **menyisipkan node baru di depan (head)** dari linked-list. Node baru dibuat sebagai dictionary dengan `data` dan `next`. Properti `next` akan menunjuk ke node sebelumnya (yaitu `head` saat ini), lalu fungsi mengembalikan node baru tersebut sebagai head yang baru.

```
# 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']
```

Penjelsan :

Fungsi ini digunakan untuk **menghapus node pertama (head)** dari linked-list. Jika `head` kosong, fungsi mencetak pesan bahwa tidak ada node yang bisa dihapus. Jika tidak kosong, ia mencetak data node yang dihapus dan mengembalikan node setelah head sebagai head baru.

```
# 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 melebih 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
```

Penjelsan :

Fungsi ini bertugas menghapus **node di posisi tertentu** dalam linked-list:

- Pertama, dicek apakah list kosong.
- Jika posisi < 0, maka posisi tidak valid.
- Jika posisi == 0, berarti node yang dihapus adalah head, maka fungsi `hapus_head` dipanggil.
- Jika posisi > 0, maka fungsi mencari node di posisi sebelum target (`position - 1`) dan mengubah pointer `next` agar melewati node target.
- Jika `position` lebih besar dari panjang list, dicetak pesan kesalahan.

Setelah penghapusan berhasil, list tetap terhubung dengan baik.

```
## 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")
```

Penjelsan :

Fungsi ini mencetak isi linked-list dari head hingga akhir. Setiap node ditampilkan mulai dari "Head →", diikuti data setiap node, hingga mencapai `NULL` sebagai akhir dari list.

```
# 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
```

Penjelsan :

Bagian ini membangun linked-list awal dengan menyisipkan elemen secara terbalik, sehingga linked-list akhir adalah: 70 → 50 → 10 → 20 → 30 → NULL

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

```
cetak_linked_list(head)
```

Penjelsan :

Menampilkan isi linked-list sebelum dilakukan penghapusan node.

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

Penjelsan :

Menghapus node di posisi ke-2 (indeks mulai dari 0), yaitu node dengan data **10**. Maka setelah penghapusan, pointer node 50 akan langsung menunjuk ke node 20, melewati 10.

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

Penjelsan :

Menampilkan linked-list setelah node di posisi ke-2 dihapus. Hasil akhir: 70 → 50 → 20 → 30 → NULL