TUGAS MENJELASKAN PROGRAM

Tugas Ini Dibuat Guna Memenuhi Mata Kuliah Struktur Data

Dosen pengampu:

Adam bachtiar, s.kom, M.MT

Disusun Oleh :

Nama : Mutiah Aryani

NIM : 24241040

Kelas : PTI B

PROGRAM STUDI PENDIDIKAN TEKNOLOGI INFORMASI
FAKULTAS SAINS, TEKNIK DAN TERAPAN
UNIVERSITAS PENDIDIKAN MANDALIKA MATARAM

2025

Modul 2 Linked-List

❖ **Praktek 22**

- Bagian 1:

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

Penjelasan:

1. Baris 2: Mendefinisikan fungsi buat_node yang menerima parameter data.
2. Baris 3: Fungsi mengembalikan dictionary berisi dua elemen:
   a. 'data': menyimpan nilai yang diberikan.
   b. 'next': diset ke None, artinya belum terhubung ke node lain.

- Bagian 2

```
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
```

Penjelasan:

1. Baris 6: Fungsi menerima head (awal linked list) dan data yang ingin ditambahkan.
2. Baris 7: Membuat node baru menggunakan buat_node.
3. Baris 8–9: Jika head kosong (None), artinya list belum ada. Maka, node baru langsung menjadi head.
4. Baris 10: Jika head tidak kosong, mulai dari node pertama.
5. Baris 11–12: Iterasi sampai node terakhir (yaitu node dengan next == None).
6. Baris 13: Hubungkan node terakhir ke node baru.
7. Baris 14: Kembalikan head agar tetap menunjuk ke awal list.

- Bagian 3

```
16  # menampilkan linked-list
17  def cetak_linked_list(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
```

Penjelasan:

1. Baris 17: Definisi fungsi cetak_linked_list dengan parameter head.
2. Baris 18: Inisialisasi variabel current untuk iterasi dari head.
3. Baris 19: Cetak label "Head" sebagai awal tampilan.
4. Baris 20–22:
   a. Selama current tidak None, cetak datanya.

b. Lanjut ke node berikutnya dengan current = current['next'].

5. Baris 23: Setelah semua node dicetak, tampilkan NULL sebagai akhir list.

- Bagian 4

```
25  # Contoh Penerapan
26  # Head awal dari linked-list
27  head = None
28
```

```
29  # Tambah node
30  head = tambah_node(head, 10)
31  head = tambah_node(head, 11)
32  head = tambah_node(head, 12)
33
34  # cetak linked-list
35  print('Linked-List : ')
36  cetak_linked_list(head)
```

Penjelasan:

1. Baris 25–27: Komentar dan inisialisasi. Awalnya linked list kosong (head = None).
2. Baris 28–31:
   a. Menambahkan node ke dalam linked list satu per satu.
   b. Nilai yang ditambahkan: 10, 11, 12.
3. Baris 33–35:
   a. Cetak judul "Linked-List :"
   b. Panggil fungsi cetak_linked_list untuk mencetak isi linked list.

Output:

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS
PS C:\PENYIMPANAN DATA\Struktur Data\Modul 2 Linked list> & C:/Users/user/AppData/Local/Pr
 DATA/Struktur Data/Modul 2 Linked list/modul2LinkedList/main.py"
Linked-List :
Head → 10 → 11 → 12 → NULL
PS C:\PENYIMPANAN DATA\Struktur Data\Modul 2 Linked list>
```

❖ **Praktek 23**

- Bagian 1

```
1  # function untuk membuat node
2  def buat_node(data):
3      return {'data': data, 'next': None}
4
```

Penjelasan:

1. Baris 2: Mendefinisikan fungsi buat_node yang menerima satu parameter: data.
2. Baris 3: Mengembalikan dictionary dengan dua elemen:
   a. 'data': nilai node.
   b. 'next': inisialisasi None karena belum terhubung ke node lain.

- Bagian 2

```
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
```

Penjelasan:

1. Baris 6: Definisi fungsi untuk menambahkan node ke akhir list.
2. Baris 7: Membuat node baru.
3. Baris 8–9: Jika linked list masih kosong (head == None), maka node baru menjadi node pertama (head).
4. Baris 10–12: Iterasi ke node terakhir (next == None) agar kita bisa menambahkan node di sana.
5. Baris 13–14: Tautkan node terakhir ke node baru. Kembalikan head agar tetap menunjuk ke awal list.

- Bagian 3

```
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
```

Penjelasan:

1. Baris 17–18: Definisi fungsi dan inisialisasi pointer current ke node pertama. Tampilkan label "Head".
2. Baris 19–21: Selama belum mencapai akhir list (None), tampilkan nilai node dan lanjutkan. Akhiri dengan "NULL".

- Bagian 4

```
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
```

Penjelasan:

1. Baris 26–28: Inisialisasi counter dan mulai traversal.
2. Baris 29–32: Hitung setiap node, lanjutkan traversal, dan kembalikan jumlah total node.

- Bagian 5

```
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
```

Penjelasan:

1. Baris 35–37: Jika list kosong, langsung kembalikan None.
2. Baris 38–41: Iterasi sampai menemukan node dengan next == None (yaitu node terakhir) lalu kembalikan node tersebut.

- Bagian 6

```
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'])
```

Penjelasan:

1. Baris 44: Linked list dimulai dari kosong.
2. Baris 45–48: Tambahkan empat node secara bertahap ke akhir linked list.
3. Baris 51–52: Cetak isi dari linked list.
4. Baris 55: Hitung dan tampilkan jumlah node.
5. Baris 58: Tampilkan nilai node pertama.
6. Baris 61: Ambil node terakhir dan tampilkan nilainya.

Output:

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS
PS C:\PENYIMPANAN DATA\Struktur Data\Modul 2 Linked list> & C:/Users/user/AppDat
 DATA/Struktur Data/Modul 2 Linked list/modul2LinkedList/main.py"
Isi Linked-List
Head → 10 → 15 → 117 → 19 → NULL
Jumlah Nodes =  4
HEAD Node :  10
TAIL Node :  19
PS C:\PENYIMPANAN DATA\Struktur Data\Modul 2 Linked list>
```

❖ **Praktek 24**

- Bagian 1

```
1  # membuat node baru
2  def sisip_depan(head, data):
3      new_node = {'data': data, 'next': head}
4      return new_node
5
```

Penjelasan:

1. Baris 2: Mendefinisikan fungsi sisip_depan dengan dua parameter: head (linked list saat ini) dan data (nilai node baru).
2. Baris 3: Membuat node baru yang:
   a. menyimpan nilai data,
   b. menunjuk ke node head lama (jadi node baru akan menjadi head baru).
3. Baris 4: Mengembalikan node baru sebagai head yang baru.

- Bagian 2

```
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")
14
```

Penjelasan:

1. Baris 7: Inisialisasi pointer current ke awal list (head).
2. Baris 8–12: Iterasi dari head ke tail dan tampilkan nilai setiap node.
3. Baris 13: Cetak "NULL" sebagai penutup list.

- Bagian 3

```
15  # Penerapan membuat linked-list awal
16  head = None
17  head = sisip_depan(head, 30)
18  head = sisip_depan(head, 20)
19  head = sisip_depan(head, 10)
20
```

Penjelasan:

1. Baris 16: Inisialisasi linked list kosong.
2. Baris 17–19: Menyisipkan node secara berurutan ke depan:
    a. sisip_depan(head, 30) → list: 30 → NULL
    b. sisip_depan(head, 20) → list: 20 → 30 → NULL
    c. sisip_depan(head, 10) → list: 10 → 20 → 30 → NULL

- Bagian 4

```
21  # cetak isi linked-list awal
22  print("Isi Linked-List Sebelum Penyisipan di Depan")
23  cetak = cetak_linked_list(head)
24
```

Penjelasan:

1. Baris 20: Panggil fungsi cetak_linked_list untuk menampilkan isi.
    a. Catatan: Penugasan cetak = tidak berguna karena cetak_linked_list tidak mengembalikan nilai. Ini bisa dihapus tanpa efek.

- Bagian 5

```
25  # Penyisipan node
26  data = 99
27  head = sisip_depan(head, data)
28
```

Penjelasan:

1. Baris 26: Data yang akan disisipkan adalah 99.
2. Baris 27: Node baru disisipkan di depan list: 99 → 10 → 20 → 30 → NULL

- Bagian 6

```
28
29  print("\nData Yang Disispkan : ", data)
30
```

Penjelasan: Cetak nilai yang baru disisipkan

- Bagian 7

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

Penjelasan:

1. Baris 32: Judul tampilan.
2. Baris 33: Tampilkan isi list setelah node 99 disisipkan di depan.

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS C:\PENYIMPANAN DATA\Struktur Data\Modul 2 Linked list> & C:/Users/user/A
 DATA/Struktur Data/Modul 2 Linked list/modul2LinkedList/main.py"
Isi Linked-List Sebelum Penyisipan di Depan
Head → 10 → 20 → 30 → NULL

Data Yang Disispkan :  99

Isi Linked-List Setelah Penyisipan di Depan
Head → 99 → 10 → 20 → 30 → NULL
PS C:\PENYIMPANAN DATA\Struktur Data\Modul 2 Linked list>
```

❖ Praktek 25

- Bagian 1

```
1  # membuat node baru
2  def sisip_depan(head, data):
3      new_node = {'data': data, 'next': head}
4      return new_node
5
```

Penjelasan:

1. Baris 2: Mendefinisikan fungsi sisip_depan() untuk menambahkan node di awal linked list.
2. Baris 3: Membuat node baru berupa dictionary dengan dua kunci:
   a. 'data' → nilai data yang diberikan
   b. 'next' → menunjuk ke head saat ini (node sebelumnya)
3. Baris 4: Mengembalikan node baru sebagai head dari linked list yang diperbarui.

- Bagian 2

```
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
```

Penjelasan:

1. Baris 7: Mendefinisikan fungsi untuk menyisipkan node pada posisi tertentu.
2. Baris 8: Membuat node baru dengan data dan next default None.

3. Baris 11–12: Jika posisi adalah 0 (depan), langsung gunakan sisip_depan().
4. Baris 14–15: Mulai traversal dari head, menggunakan variabel current dan penghitung index.
5. Baris 18–20: Menelusuri node sampai tepat sebelum posisi penyisipan (position - 1).
6. Baris 22–24: Jika traversal habis sebelum mencapai posisi, berarti posisi invalid.
7. Baris 27–29:
    a. Hubungkan new_node ke node berikutnya (current['next'])
    b. Lalu ubah current['next'] untuk menunjuk ke new_node

- Bagian 3

```
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
```

Penjelasan:

1. Baris 32: Mendefinisikan fungsi untuk mencetak seluruh linked list.
2. Baris 33–34: Inisialisasi traversal dan cetak label “Head”.
3. Baris 35–37: Cetak data dari setiap node dan lanjut ke node berikutnya.
4. Baris 38: Cetak "NULL" sebagai penutup list.

- Bagian 4

```
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
```

Penjelasan:

1. Baris 42: Menginisialisasi linked list kosong (head = None).
2. Baris 43–47:
    a. Menyisipkan data satu per satu di depan.
    b. Urutan hasil akhir: 70 → 50 → 10 → 20 → 30 → NULL

- Bagian 5

```
49  # cetak isi linked-list awal
50  print("Isi Linked-List Sebelum Penyisipan")
51  cetak = cetak_linked_list(head)
52
```

Penjelasan:

Baris 50–51: Cetak isi linked list sebelum ada penyisipan di posisi tertentu.

- Bagian 6

```
53  # Penyisipan node
54  data = 99
55  pos = 3
56  head = sisip_dimana_aja(head, data, pos)
57
```

Penjelasan:

1. Baris 54–56:
    a. Menyisipkan nilai 99 pada posisi ke-3 (mulai dari 0).
    b. Node baru akan disisipkan setelah node ke-2.
    c. Hasil akhir: 70 → 50 → 10 → 99 → 20 → 30 → NULL

- Bagian 7

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

Penjelasan:

1. Baris 58–59: Menampilkan informasi data dan posisi yang disisipkan.
2. Baris 62–63: Menampilkan hasil akhir linked list setelah penyisipan.

Output:

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS
PS C:\PENYIMPANAN DATA\Struktur Data\Modul 2 Linked list> & C:/Users/user/Appl
 DATA/Struktur Data/Modul 2 Linked list/modul2LinkedList/main.py"
Isi Linked-List Sebelum Penyisipan
Head → 70 → 50 → 10 → 20 → 30 → NULL

Data Yang Disispkan :  99
Pada posisi :  3

Isi Linked-List Setelah Penyisipan di tengah
Head → 70 → 50 → 10 → 99 → 20 → 30 → NULL
PS C:\PENYIMPANAN DATA\Struktur Data\Modul 2 Linked list>
```

❖ **Praktek 26**

- Bagian 1

```
1   # membuat node baru
2   def sisip_depan(head, data):
3       new_node = {'data': data, 'next': head}
4       return new_node
5
```

Penjelasan:

1. Baris 2: Definisi fungsi sisip_depan() untuk menambahkan node baru di depan linked list.
2. Baris 3-4: Membuat node baru (tipe dictionary) yang menunjuk ke head lama dan mengembalikannya sebagai head baru.

- Bagian 2

```
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
```

Penjelasan:

1. Baris 7–8: Membuat fungsi untuk menyisipkan data pada posisi tertentu.
2. Baris 11–12: Jika posisi adalah 0, gunakan sisip_depan().
3. Baris 14–15: Siapkan variabel traversal.
4. Baris 18–20: Menelusuri node sampai posisi yang dimaksud.
5. Baris 22–24: Jika posisi tidak valid (melebihi panjang), tampilkan pesan.
6. Baris 27–29: Lakukan penyisipan di posisi yang valid.

- Bagian 3

```
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
```

Penjelasan:

1. Baris 32: Fungsi untuk menghapus node paling depan (head).
2. Baris 34–36: Jika linked list kosong, tidak ada yang dihapus.
3. Baris 37–38: Tampilkan node yang dihapus, dan pindahkan head ke node berikutnya.

- Bagian 4

```
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
```

Penjelasan:

1. Baris 41–43: Inisialisasi fungsi untuk menampilkan isi linked list.
2. Baris 44–47: Looping menampilkan isi node hingga akhir.

- Bagian 5

```
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)
```

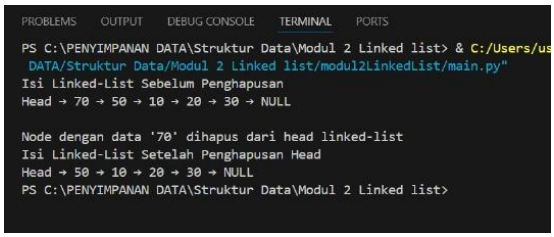
```
62   # Penghapusan head linked-list
63   head = hapus_head(head)
64
65   # cetak isi setelah hapus head linked-list
66   print("Isi Linked-List Setelah Penghapusan Head ")
67   cetak_linked_list(head)
```

Penjelasan:

1. Baris 51: Inisialisasi linked list kosong.

2. Baris 52–56: Menyisipkan data ke depan satu per satu.
3. Baris 59–60: Menampilkan linked list sebelum node head dihapus.
4. Baris 63: Menghapus node pertama (head), yaitu node dengan data 70.
5. Baris 66–67:
    a. Menampilkan isi linked list setelah penghapusan head.
    b. Hasil akhir: 50 → 10 → 20 → 30 → NULL

Output:



❖ **Praktek 27**

- Bagian 1

```
1  # membuat node baru
2  def sisip_depan(head, data):
3      new_node = {'data': data, 'next': head}
4      return new_node
5
```

Penjelasan:

Baris 3–4: Membuat node baru (dictionary) dan menjadikannya head baru, karena node ini menunjuk ke head lama. Fungsi ini digunakan untuk menambahkan elemen ke depan linked list.

- Bagian 2

```
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
```

Penjelasan:

1. Baris 7: Mendefinisikan fungsi hapus_tail() untuk menghapus node terakhir dari linked list.
2. Baris 9–11: Jika linked list kosong, tampilkan pesan dan kembalikan None.
3. Baris 14–16: Jika hanya ada satu node, langsung hapus node tersebut dan linked list menjadi kosong.
4. Baris 18–20: Traversal hingga mencapai node sebelum tail (yaitu node dengan next.next == None).

5. Baris 22–24: Tampilkan data tail yang dihapus, dan ubah pointer next dari node sebelumnya menjadi None.

- Bagian 3

```
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
```

Penjelasan:

Baris 27–33: Fungsi untuk menampilkan semua data node di linked list secara berurutan dari head hingga NULL.

- Bagian 4

```
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)
```

Penjelasan:

1. Baris 37–42: Membentuk linked list dengan penyisipan di depan. Urutan akhir: 70 → 50 → 10 → 20 → 30 → NULL
    a. 70 jadi head
    b. 30 jadi tail
2. Baris 45–46: Menampilkan isi linked list sebelum tail dihapus.
3. Baris 49: Menghapus node paling akhir (30).
4. Baris 46–47:
    a. Menampilkan isi linked list setelah tail dihapus.
    b. Hasil akhir: 70 → 50 → 10 → 20 → NULL

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS C:\PENYIMPANAN DATA\Struktur Data\Modul 2 Linked list> & C:/Users/user/AppData/Lo
 DATA/Struktur Data/Modul 2 Linked list/modul2LinkedList/main.py"
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 C:\PENYIMPANAN DATA\Struktur Data\Modul 2 Linked list>
```

❖ **Praktek 28**

- Bagian 1

```
1  # 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
```

Penjelasan:

1. Baris 3: Definisi fungsi untuk menyisipkan node di depan.
2. Baris 4: Membuat dictionary new_node dengan data dan next mengarah ke head lama.
3. Baris 5: Mengembalikan new_node sebagai head baru.

- Bagian 2

```
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
```

Penjelasan:

1. Baris 8: Fungsi untuk menghapus node paling depan.
2. Baris 10-12: Jika list kosong, cetak pesan dan return None.
3. Baris 13: Cetak data node yang dihapus.
4. Baris 14: Kembalikan node setelah head sebagai head baru.

- Bagian 3

```
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 melebih 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
```

Penjelasan:

1. Baris 17: Definisi fungsi hapus di posisi tertentu.

2. Baris 19-21: Cek jika linked list kosong.
3. Baris 24-26: Tangani jika posisi negatif.
4. Baris 29-32: Jika posisi = 0, hapus head.

   Catatan: hapus_head(head) tidak dipakai hasil return-nya — bisa diperbaiki jadi return hapus_head(head).
5. Baris 34-35: Inisialisasi traversal dari head.
6. Baris 38-40: Pindah ke node sebelum node yang ingin dihapus.
7. Baris 43-45: Jika posisi lebih panjang dari panjang list, cetak pesan.
8. Baris 47: Cetak node yang akan dihapus.
9. Baris 48: Melewati node target (menghapus).
10. Baris 39: Return head setelah perubahan.

- Bagian 4

```
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
```

Penjelasan:

1. Baris 52-53: Mulai dari head.
2. Baris 54: Cetak label awal.
3. Baris 55-57: Traversal dan cetak isi node.
4. Baris 58: Akhiri dengan NULL.

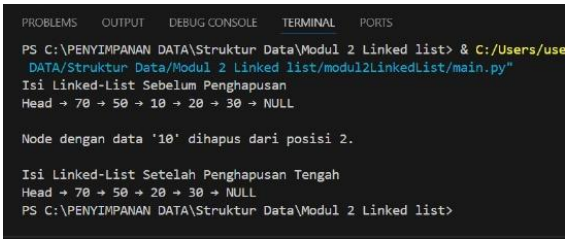
- Bagian 5

```
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)
```

Penjelasan:

1. Baris 62-67: Membentuk linked list 70 → 50 → 10 → 20 → 30 → NULL.
2. Baris 70-71: Cetak list sebelum penghapusan.
3. Baris 74: Hapus node posisi ke-2 (data 10).
4. Baris 77-78: Cetak isi list setelah penghapusan.

Output:


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PS C:\PENYIMPANAN DATA\Struktur Data\Modul 2 Linked list> & C:/Users/use
DATA/Struktur Data/Modul 2 Linked list/modul2LinkedList/main.py"
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 C:\PENYIMPANAN DATA\Struktur Data\Modul 2 Linked list>