

Linked-List

Singel Linked-List

Praktek 22

```
1 # function untuk membuat node
2 def buat_node(data):
3     return {'data': data, 'next': None}
4 # menambahkan node di akhir list
5 def tambah_node(head, data):
6     new_node = buat_node(data)
7     if head is None:
8         return new_node
9     current = head
10    while current['next'] is not None:
11        current = current['next']
12    current['next'] = new_node
13    return head
14
15 # menampilkan linked-list
16 def cetak_linked_list(head):
17     current = head
18     print('Head', end=' → ')
19     while current is not None:
20         print(current['data'], end=' → ')
21         current = current['next']
22     print("NULL")
23
24 # Contoh Penerapan
25 # Head awal dari linked-list
26 head = None
27
28 # Tambah node
29 head = tambah_node(head, 10)
30 head = tambah_node(head, 11)
31 head = tambah_node(head, 12)
32
33 # cetak linked-list
34 print('Linked-List : ')
35 cetak_linked_list(head)
36
```

Outputnya

```
Linked-List :
Head → 10 → 11 → 12 → NULL
```

Penjelasan :

Baris 1: Komentar – memberi tahu bahwa fungsi berikut digunakan untuk membuat node.

Baris 2: Mendefinisikan fungsi `buat_node` dengan parameter `data`.

Baris 3: Mengembalikan sebuah dictionary yang berisi `data` dan `next` (penunjuk ke node berikutnya, awalnya `None`).

Baris 4: Komentar – memberi tahu bahwa fungsi berikut digunakan untuk menambahkan node di akhir linked list.

Baris 5: Mendefinisikan fungsi `tambah_node` dengan parameter `head` dan `data`.

Baris 6: Membuat node baru dengan memanggil `buat_node(data)`.

Baris 7: Mengecek apakah linked list masih kosong (`head` adalah `None`).

Baris 8: Jika kosong, langsung kembalikan node baru sebagai `head`.

Baris 9: Jika tidak kosong, buat variabel `current` untuk mulai dari `head`.

Baris 10: Mulai loop untuk menelusuri node hingga ke node terakhir.

Baris 11: Pindah ke node berikutnya selama masih ada node selanjutnya.

Baris 12: Setelah ketemu node terakhir, sambungkan node baru ke sana.

Baris 13: Kembalikan `head` agar linked list tetap utuh.

Baris 15: Komentar – memberi tahu bahwa fungsi berikut digunakan untuk mencetak isi linked list.

Baris 16: Mendefinisikan fungsi `cetak_linked_list` dengan parameter `head`.

Baris 17: Mulai dari node pertama (`head`) dan simpan dalam `current`.

Baris 18: Cetak teks awal "`Head →`" tanpa pindah baris.

Baris 19: Mulai loop selama `current` masih ada (tidak `None`).

Baris 20: Cetak nilai `data` dari node sekarang, lalu pindah ke node berikutnya.

Baris 21: Setelah loop selesai, cetak "`NULL`" sebagai akhir list.

Baris 23: Komentar – memberi tahu bahwa bagian ini adalah contoh penerapan.

Baris 24: Komentar – menjelaskan bahwa linked list dimulai dari kondisi kosong.

Baris 25: Inisialisasi `head` sebagai `None`, artinya list kosong.

Baris 27: Menambahkan node pertama dengan nilai 10.

Baris 28: Menambahkan node kedua dengan nilai 11.

Baris 29: Menambahkan node ketiga dengan nilai 12.

Baris 31: Komentar – memberi tahu bahwa isi list akan dicetak.

Baris 32: Mencetak label "Linked-List :".

Baris 33: Memanggil fungsi cetak_linked_list untuk mencetak isi linked list dari head sampai akhir.

Traversal Single Linked-List

Praktek 23

```
1  # function untuk membuat node
2  def buat_node(data):
3      return {'data': data, 'next': None}
4  # menambahkan node di akhir list
5  def tambah_node(head, data):
6      new_node = buat_node(data)
7      if head is None:
8          return new_node
9      current = head
10     while current['next'] is not None:
11         current = current['next']
12     current['next'] = new_node
13     return head
14
15 # traversal untuk cetak isi linked-list
16 def traversal_to_display(head):
17     current = head
18     print('Head', end=' → ')
19     while current is not None:
20         print(current['data'], end=' → ')
21         current = current['next']
22     print("NULL")
23
24 # traversal untuk menghitung jumlah elemen dalam linked-list
25 def traversal_to_count_nodes(head):
26     count = 0
27     current = head
28     while current is not None:
29         count += 1
30         current = current['next']
31     return count
32
33 # traversal untuk mencari dimana tail (node terakhir)
34 def traversal_to_get_tail(head):
35     if head is None:
36         return None
37     current = head
38     while current['next'] is not None:
39         current = current['next']
40     return current
41
```

```

42 # Penerapan
43 head = None
44 head = tambah_node(head, 10)
45 head = tambah_node(head, 15)
46 head = tambah_node(head, 117)
47 head = tambah_node(head, 19)
48
49 # cetak isi linked-list
50 print("Isi Linked-List")
51 traversal_to_display(head)
53 # cetak jumlah node
54 print("Jumlah Nodes = ", traversal_to_count_nodes(head))
55
56 # cetak HEAD node
57 print("HEAD Node : ", head['data'])
58
59 # cetak TAIL NODE
60 print("TAIL Node : ", traversal_to_get_tail(head)['data'])

```

Outputnya

```

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

```

Pejelasanannya

Baris 1: Komentar – memberi tahu bahwa fungsi berikut digunakan untuk membuat node.

Baris 2: Mendefinisikan fungsi buat_node dengan parameter data.

Baris 3: Mengembalikan dictionary node dengan data dan next (bernilai None).

Baris 4: Komentar – menjelaskan fungsi untuk menambahkan node di akhir list.

Baris 5: Mendefinisikan fungsi tambah_node dengan parameter head dan data.

Baris 6: Membuat node baru menggunakan fungsi buat_node.

Baris 7: Jika head kosong, langsung kembalikan node baru sebagai head.

Baris 8: Kembalikan node baru sebagai head.

Baris 9: Menyimpan node awal (head) ke dalam current.

Baris 10: Melakukan perulangan selama current['next'] tidak kosong.

Baris 11: Pindah ke node berikutnya.

Baris 12: Setelah ketemu node terakhir, sambungkan node baru ke sana.

Baris 13: Kembalikan head untuk mempertahankan linked list utuh.

Baris 15: Komentar – fungsi untuk traversal menampilkan isi linked list.

Baris 16: Mendefinisikan fungsi `traversal_to_display` dengan parameter `head`.

Baris 17: Simpan `head` ke variabel `current`.

Baris 18: Cetak teks "Head →" sebagai awalan.

Baris 19: Perulangan selama masih ada node (`current` tidak kosong).

Baris 20: Cetak data pada node dan lanjut ke node berikutnya.

Baris 21: Pindah ke node selanjutnya.

Baris 22: Setelah selesai, cetak "NULL" sebagai penutup list.

Baris 24: Komentar – traversal untuk menghitung jumlah node.

Baris 25: Mendefinisikan fungsi `traversal_to_count_nodes` dengan parameter `head`.

Baris 26: Inisialisasi variabel `count` untuk menghitung jumlah node.

Baris 27: Simpan `head` ke dalam `current`.

Baris 28: Perulangan selama masih ada node.

Baris 29: Tambahkan 1 ke `count` setiap menemukan node.

Baris 30: Pindah ke node selanjutnya.

Baris 31: Setelah loop, kembalikan nilai `count`.

Baris 33: Komentar – traversal untuk menemukan node terakhir (`tail`).

Baris 34: Mendefinisikan fungsi `traversal_to_get_tail` dengan parameter `head`.

Baris 35: Jika `head` kosong, kembalikan `None`.

Baris 36: Simpan `head` ke dalam `current`.

Baris 37: Perulangan selama masih ada node berikutnya.

Baris 38: Pindah ke node selanjutnya.

Baris 39: Setelah loop, kembalikan node terakhir (`tail`).

Baris 41: Komentar – bagian penerapan kode dimulai.

Baris 42: Inisialisasi `head` sebagai linked list kosong.

Baris 43: Tambahkan node dengan nilai 10.

Baris 44: Tambahkan node dengan nilai 15.

Baris 45: Tambahkan node dengan nilai 117.

Baris 46: Tambahkan node dengan nilai 19.

Baris 48: Komentar – mencetak isi linked list.

Baris 49: Cetak label "Isi Linked-List".

Baris 50: Panggil fungsi untuk mencetak isi linked list.

Baris 52: Komentar – mencetak jumlah node.

Baris 53: Cetak jumlah node menggunakan fungsi penghitung.

Baris 55: Komentar – mencetak nilai head node.

Baris 56: Cetak nilai data dari head node.

Baris 58: Komentar – mencetak nilai tail node.

Baris 59: Cetak data dari node terakhir dengan memanggil fungsi tail.

Insert Single Linekd-List

1. Penyisipan Diawal Linked-List

Praktek 24

```
1  # membuat node baru
2  def sisip_depan(head, data):
3      new_node = {'data': data, 'next': head}
4      return new_node
5
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
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
21 # cetak isi linked-list awal
22 print("Isi Linked-List Sebelum Penyisipan di Depan")
23 cetak = cetak_linked_list(head)
24
25 # Penyisipan node
26 data = 99
27 head = sisip_depan(head, data)
28
29 print("\nData Yang Disisipkan : ", data)
30
31 # cetak isi setelah penyisipan node baru di awal
32 print("\nIsi Linked-List Setelah Penyisipan di Depan")
33 cetak_linked_list(head)
```

Outputnya

```
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
```

Penjelasannya

Baris 1: Komentar – menjelaskan bahwa fungsi berikut digunakan untuk membuat node baru di depan.

Baris 2: Mendefinisikan fungsi `sisip_depan` dengan parameter `head` dan `data`.

Baris 3: Membuat node baru dengan nilai `data` dan menghubungkannya langsung ke `head` lama.

Baris 4: Mengembalikan node baru sebagai `head` yang baru.

Baris 6: Komentar – menjelaskan fungsi untuk mencetak isi linked list.

Baris 7: Mendefinisikan fungsi `cetak_linked_list` dengan parameter `head`.

Baris 8: Menyimpan node awal ke dalam `current`.

Baris 9: Mencetak teks pembuka "Head →".

Baris 10: Melakukan loop selama `current` tidak kosong.

Baris 11: Mencetak nilai `data` pada node dan lanjut ke node berikutnya.

Baris 12: Pindah ke node berikutnya.

Baris 13: Setelah selesai, mencetak "NULL" sebagai penutup list.

Baris 15: Komentar – penanda bahwa proses membuat linked list awal akan dimulai.

Baris 16: Menginisialisasi linked list kosong (`head = None`).

Baris 17: Menyisipkan node dengan nilai 30 di depan.

Baris 18: Menyisipkan node dengan nilai 20 di depan (sebelum 30).

Baris 19: Menyisipkan node dengan nilai 10 di depan (sebelum 20 dan 30).

Baris 21: Komentar – menandai proses cetak isi linked list awal.

Baris 22: Menampilkan label isi linked list sebelum penyisipan baru.

Baris 23: Memanggil fungsi cetak untuk menampilkan isi linked list awal.

Baris 25: Komentar – bagian penyisipan node baru dimulai.

Baris 26: Menyimpan nilai data yang akan disisipkan di variabel data.

Baris 27: Menyisipkan data tersebut di depan (sebagai head baru).

Baris 29: Menampilkan data yang disisipkan ke pengguna.

Baris 31: Komentar – menampilkan hasil akhir linked list setelah penyisipan node baru.

Baris 32: Menampilkan label linked list setelah penyisipan node baru.

Baris 33: Memanggil fungsi cetak untuk menampilkan linked list terbaru.

2. Penyisipan di Posisi Tertentu

Praktek 25

```
1  # membuat node baru
2  def sisip_depan(head, data):
3      new_node = {'data': data, 'next': head}
4      return new_node
5
6  # sisip node diposisi mana saja
7  def sisip_dimana_aja(head, data, position):
8      new_node = {'data': data, 'next': None}
9
10     # cek jika posisi di awal pakai fungsi sisip_depan()
11     if position == 0:
12         return sisip_depan(head, data)
13
14     current = head
15     index = 0
16
17     # traversal menuju posisi yang diinginkan dan bukan posisi 0
18     while current is not None and index < position - 1:
19         current = current['next']
20         index += 1
21
22     if current is None:
23         print("Posisi melebihi panjang linked list!")
24         return head
25
26     # ubah next dari node sebelumnya menjadi node baru
27     new_node['next'] = current['next']
28     current['next'] = new_node
29     return head
30
31     ## menampilkan linked-list
32     def cetak_linked_list(head):
33         current = head
34         print('Head', end=' → ')
35         while current is not None:
36             print(current['data'], end=' → ')
37             current = current['next']
38         print("NULL")
```



```

39
40 # Penerapan
41 # membuat linked-list awal
42 head = None
43 head = sisip_depan(head, 30)
44 head = sisip_depan(head, 20)
45 head = sisip_depan(head, 10)
46 head = sisip_depan(head, 50)
47 head = sisip_depan(head, 70)
48
49 # cetak isi linked-list awal
50 print("Isi Linked-List Sebelum Penyisipan")
51 cetak = cetak_linked_list(head)
52
53 # Penyisipan node
54 data = 99
55 pos = 3
56 head = sisip_dimana_aja(head, data, pos)
57
58 print("\nData Yang Disisipkan : ", data)
59 print("Pada posisi : ", pos, "")
60
61 # cetak isi setelah penyisipan node baru di awal
62 print("\nIsi Linked-List Setelah Penyisipan di tengah")
63 cetak_linked_list(head)

```

Outputnya

```

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

```

Penjelasannya

Baris 1: Komentar – memberi tahu bahwa fungsi berikut untuk menyisipkan node di depan.

Baris 2: Mendefinisikan fungsi `sisip_depan` dengan parameter `head` dan `data`.

Baris 3: Membuat node baru yang menunjuk ke `head` lama.

Baris 4: Mengembalikan node baru sebagai `head` yang baru.

Baris 6: Komentar – menjelaskan fungsi selanjutnya untuk menyisipkan node di posisi mana pun.

Baris 7: Mendefinisikan fungsi `sisip_dimana_aja` dengan parameter `head`, `data`, dan `position`.

Baris 8: Membuat node baru, `next`-nya di-set `None`.

Baris 10: Mengecek jika posisi penyisipan adalah di depan (posisi 0).

Baris 11: Jika benar, gunakan fungsi `sisip_depan`.

Baris 13: Menyimpan head dalam variabel `current` untuk proses traversal.

Baris 14: Inisialisasi `index` dengan 0 untuk menghitung posisi.

Baris 16: Perulangan untuk mencapai node sebelum posisi yang ditentukan.

Baris 17: Pindah ke node berikutnya.

Baris 18: Menambahkan nilai `index`.

Baris 20: Jika `current` kosong, artinya posisi melebihi panjang list.

Baris 21: Tampilkan pesan bahwa posisi tidak valid.

Baris 22: Kembalikan head tanpa perubahan.

Baris 24: Set next node baru ke node setelah `current`.

Baris 25: Set `current.next` ke node baru, menyisipkan node di posisi yang diinginkan.

Baris 26: Kembalikan head yang telah diperbarui.

Baris 28: Komentar – penjelasan bahwa fungsi berikut untuk mencetak linked list.

Baris 29: Mendefinisikan fungsi `cetak_linked_list`.

Baris 30: Menyimpan head ke variabel `current`.

Baris 31: Cetak teks pembuka "Head →".

Baris 32: Perulangan untuk mencetak seluruh isi list.

Baris 33: Cetak data pada node saat ini.

Baris 34: Pindah ke node berikutnya.

Baris 35: Cetak "NULL" sebagai penanda akhir list.

Baris 37: Komentar – menunjukkan bagian penerapan kode.

Baris 38: Inisialisasi head sebagai list kosong.

Baris 39: Sisipkan 30 di depan.

Baris 40: Sisipkan 20 di depan (jadi di atas 30).

Baris 41: Sisipkan 10 di depan.

Baris 42: Sisipkan 50 di depan.

Baris 43: Sisipkan 70 di depan.

Baris 45: Komentar – bagian untuk mencetak isi list sebelum penyisipan.

Baris 46: Tampilkan label isi list sebelum disisipkan node baru.

Baris 47: Cetak linked list awal.

Baris 49: Komentar – mulai proses penyisipan baru.

Baris 50: Simpan data yang akan disisipkan, yaitu 99.

Baris 51: Simpan posisi target penyisipan, yaitu indeks ke-3.

Baris 52: Sisipkan node 99 di posisi ke-3.

Baris 54: Cetak data yang disisipkan.

Baris 55: Cetak posisi penyisipan.

Baris 57: Komentar – cetak isi list setelah penyisipan.

Baris 58: Cetak label untuk list hasil setelah penyisipan.

Baris 59: Cetak isi list setelah node baru dimasukkan.

Deleting Single Linked-List

1. Penghapusan Pada Awal (Head) Node Dalam Linked-List

Praktek 26

```
1  # membuat node baru
2  def sisip_depan(head, data):
3      new_node = {'data': data, 'next': head}
4      return new_node
5
6  # sisip node diposisi mana saja
7  def sisip_dimana_aja(head, data, position):
8      new_node = {'data': data, 'next': None}
9
10     # cek jika posisi di awal pakai fungsi sisip_depan()
11     if position == 0:
12         return sisip_depan(head, data)
13
14     current = head
15     index = 0
16
17     # traversal menuju posisi yang diinginkan dan bukan posisi 0
18     while current is not None and index < position - 1:
19         current = current['next']
20         index += 1
21
22     if current is None:
23         print("Posisi melebihi panjang linked list!")
24         return head
25
26     # ubah next dari node sebelumnya menjadi node baru
27     new_node['next'] = current['next']
28     current['next'] = new_node
29     return head
30
```

```

31 # menghapus head node dan mengembalikan head baru
32 def hapus_head(head):
33     # cek apakah list kosong
34     if head is None:
35         print("Linked-List kosong, tidak ada yang bisa")
36         return None
37     print(f"\nNode dengan data '{head['data']}' dihapus dari head linked-list")
38     return head['next']
39
40 ## menampilkan linked-list
41 def cetak_linked_list(head):
42     current = head
43     print('Head', end=' → ')
44     while current is not None:
45         print(current['data'], end=' → ')
46         current = current['next']
47     print("NULL")
48
49 # Penerapan
50 # membuat linked-list awal
51 head = None
52 head = sisip_depan(head, 30) # tail
53 head = sisip_depan(head, 20)
54 head = sisip_depan(head, 10)
55 head = sisip_depan(head, 50)
56 head = sisip_depan(head, 70) # head
57
58 # cetak isi linked-list awal
59 print("Isi Linked-List Sebelum Penghapusan")
60 cetak_linked_list(head)
61
62 # Penghapusan head linked-list
63 head = hapus_head(head)
64
65 # cetak isi setelah hapus head linked-list
66 print("Isi Linked-List Setelah Penghapusan Head ")
67 cetak_linked_list(head)

```

Outputnya

```

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

```

Penjelasannya

Baris 1: Komentar: Menandakan bahwa fungsi di bawah digunakan untuk membuat node baru.

Baris 2: Mendefinisikan fungsi `sisip_depan()` dengan parameter `head` dan `data`.

Baris 3: Membuat node baru berupa dictionary dengan kunci `'data'` dan `'next'`, di mana `'next'` menunjuk ke `head` lama.

Baris 4: Mengembalikan node baru tersebut sebagai `head` yang baru.

Baris 6: Komentar: Fungsi berikut digunakan untuk menyisipkan node di posisi mana saja dalam linked list.

Baris 7: Mendefinisikan fungsi `sisip_dimana_aja()` dengan parameter `head`, `data`, dan `position`.

Baris 8: Membuat node baru dengan `data` yang diberikan dan `next` di-set ke `None`.

Baris 10: Mengecek apakah posisi yang dimasukkan adalah 0 (berarti menyisip di depan).

Baris 11: Jika ya, maka langsung panggil fungsi `sisip_depan()` untuk menyisipkan di depan dan return hasilnya.

Baris 13: Inisialisasi variabel `current` untuk melakukan traversal mulai dari `head`.

Baris 14: Inisialisasi variabel `index` ke 0 untuk menghitung posisi saat traversal.

Baris 16: Perulangan: Selama `current` tidak `None` dan belum mencapai posisi target (`posisi - 1`), lanjutkan traversal.

Baris 17: Pindah ke node selanjutnya dalam linked list.

Baris 18: Tambahkan nilai `index` sebanyak 1.

Baris 20: Mengecek apakah `current` menjadi `None`, yang berarti posisi melebihi panjang linked list.

Baris 21: Menampilkan pesan bahwa posisi melebihi panjang linked list.

Baris 22: Mengembalikan `head` tanpa perubahan karena penyisipan gagal.

Baris 24: Jika posisi valid, atur `next` dari node baru untuk menunjuk ke node setelah `current`.

Baris 25: Mengatur `next` dari `current` agar menunjuk ke `new_node`, menyisipkan node baru.

Baris 26: Mengembalikan `head` karena `head` tidak berubah.

Baris 28: Komentar: Fungsi ini untuk menghapus node paling depan (`head`) dari linked list.

Baris 29: Mendefinisikan fungsi `hapus_head()` dengan parameter `head`.

Baris 30: Mengecek apakah linked list kosong (`head == None`).

Baris 31: Menampilkan pesan bahwa linked list kosong dan tidak ada yang bisa dihapus.

Baris 32: Mengembalikan `None` karena list memang kosong.

Baris 33: Jika tidak kosong, tampilkan pesan bahwa node dengan `data`

tertentu dihapus.

Baris 34: Mengembalikan node setelah head sebagai head yang baru.

Baris 36: Komentar: Fungsi ini digunakan untuk mencetak semua isi dari linked list.

Baris 37: Mendefinisikan fungsi cetak_linked_list() dengan parameter head.

Baris 38: Inisialisasi variabel current sebagai pointer ke node pertama (head).

Baris 39: Mencetak label awal "Head → " untuk memulai tampilan linked list.

Baris 40: Perulangan: Selama current tidak None, cetak data dari node yang sedang ditunjuk.

Baris 41: Mencetak nilai data dari node yang sedang ditunjuk diikuti tanda panah.

Baris 42: Pindah ke node berikutnya (current = current['next']).

Baris 43: Mencetak "NULL" sebagai akhir dari linked list.

Baris 45: Komentar: Bagian ini merupakan penerapan program (main program).

Baris 46: Komentar: Inisialisasi awal linked list.

Baris 47: Menginisialisasi head dengan None, artinya linked list masih kosong.

Baris 48: Menyisipkan data 30 di depan; ini menjadi node terakhir (tail).

Baris 49: Menyisipkan data 20 di depan node 30.

Baris 50: Menyisipkan data 10 di depan node 20.

Baris 51: Menyisipkan data 50 di depan node 10.

Baris 52: Menyisipkan data 70 di depan node 50, menjadi head baru.

Baris 54: Menampilkan teks "Isi Linked-List Sebelum Penghapusan".

Baris 55: Memanggil fungsi cetak_linked_list() untuk mencetak isi linked list saat ini.

Baris 57: Komentar: Melakukan penghapusan node pertama (head).

Baris 58: Memanggil hapus_head() dan menyimpan hasilnya sebagai head yang baru.

Baris 60: Menampilkan teks "Isi Linked-List Setelah Penghapusan Head".

Baris 61: Memanggil kembali cetak_linked_list() untuk mencetak kondisi terbaru linked list.

2. Penghapusan Pada Akhir (Tail) Linked-List

Praktek 27

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

```

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

```

Outputnya

```

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

```

Penjelasannya

Baris 1: Komentar: Menandakan fungsi untuk membuat node baru.

Baris 2: Definisi fungsi sisip_depan dengan parameter head dan data.

Baris 3: Membuat node baru berupa dictionary dengan kunci 'data' berisi data, dan 'next' menunjuk ke head (node sebelumnya).

Baris 4: Mengembalikan node baru sebagai head linked list.

Baris 6: Komentar: Fungsi untuk menghapus node terakhir (tail) dan mengembalikan head baru.

Baris 7: Definisi fungsi hapus_tail dengan parameter head.

Baris 8-10: Mengecek apakah linked list kosong (head None), jika ya cetak pesan dan kembalikan None.

Baris 12-15: Mengecek apakah linked list hanya memiliki satu node, jika ya hapus node tersebut, cetak pesan, dan kembalikan None.

Baris 17: Inisialisasi variabel current dengan head sebagai starting point traversal.

Baris 18-20: Traversal menuju node sebelum tail dengan kondisi while sampai current['next']['next'] None (node sebelum terakhir).

Baris 22: Cetak data node yang akan dihapus (node terakhir).

Baris 23: Menghapus node terakhir dengan membuat pointer next pada current menjadi None.

Baris 24: Mengembalikan head yang sekarang sudah tanpa node terakhir.

Baris 26: Komentar: Fungsi untuk menampilkan isi linked list.

Baris 27: Definisi fungsi cetak_linked_list dengan parameter head.

Baris 28: Inisialisasi variabel current dengan head.

Baris 29: Mencetak string awal 'Head' tanpa pindah baris.

Baris 30-32: Loop selama current bukan None, cetak data node, lalu pindah ke node berikutnya.

Baris 33: Mencetak 'NULL' menandakan akhir linked list.

Baris 35: Komentar: Bagian penerapan program.

Baris 36: Menginisialisasi linked list kosong dengan head = None.

Baris 37-41: Menambahkan node ke linked list menggunakan fungsi sisip_depan dengan data masing-masing,urut dari tail sampai head.

Baris 37: Menambahkan node 30 sebagai tail.

Baris 38: Menambahkan node 20 di depan node 30.

Baris 39: Menambahkan node 10 di depan node 20.

Baris 40: Menambahkan node 50 di depan node 10.

Baris 41: Menambahkan node 70 sebagai head linked list.

Baris 43: Mencetak teks "Isi Linked-List Sebelum Penghapusan".

Baris 44: Memanggil fungsi cetak_linked_list untuk menampilkan isi linked list saat ini.

Baris 46: Menghapus node terakhir dengan memanggil fungsi `hapus_tail`.

Baris 47: Menyimpan hasil penghapusan node terakhir ke variabel `head`.

Baris 49: Mencetak teks "Isi Linked-List Setelah Penghapusan Tail".

Baris 50: Memanggil fungsi `cetak_linked_list` untuk menampilkan isi linked list setelah penghapusan node terakhir.

3. Penghapusan Pada Posisi Tengah Linked-List

Praktek 28

```
1 # Praktek 28 : Menghapus node di posisi manapun (tengah)
2 # membuat node baru
3 def sisip_depan(head, data):
4     new_node = {'data': data, 'next': head}
5     return new_node
6
7 # menghapus head node dan mengembalikan head baru
8 def hapus_head(head):
9     # cek apakah list kosong
10    if head is None:
11        print("Linked-List kosong, tidak ada yang bisa")
12        return None
13    print(f"\nNode dengan data '{head['data']}' dihapus dari head linked-list")
14    return head['next']
15
16 # menghapus node pada posisi manapun (tengah)
17 def hapus_tengah(head, position):
18     # cek apakah head node == None
19    if head is None:
20        print('\nLinked-List Kosong, tidak ada yang bisa dihapus!')
21        return None
22
23    # cek apakah posisi < 0
24    if position < 0:
25        print('\nPosisi Tidak Valid')
26        return head
27
28    # Cek apakah posisi = 0
29    if position == 0:
30        print(f"Node dengan data '{head['data']}' dihapus dari posisi 0.")
31        hapus_head(head)
32        return head['next']
33
34    current = head
35    index = 0
36
```

```

37     # cari node sebelum posisi target
38     while current is not None and index < position - 1:
39         current = current['next']
40         index += 1
41
42     # Jika posisi yang diinputkan lebih besar dari panjang list
43     if current is None or current['next'] is None:
44         print("\nPosisi melebihi panjang dari linked-list")
45         return head
46
47     print(f"\nNode dengan data '{current['next']['data']}' dihapus dari posisi {position}.")
48     current['next'] = current['next']['next']
49     return head
50
51 ## menampilkan linked-list
52 def cetak_linked_list(head):
53     current = head
54     print('Head', end=' → ')
55     while current is not None:
56         print(current['data'], end=' → ')
57         current = current['next']
58     print("NULL")
59
60 # Penerapan
61 # membuat linked-list awal
62 head = None
63 head = sisip_depan(head, 30) # tail
64 head = sisip_depan(head, 20)
65 head = sisip_depan(head, 10)
66 head = sisip_depan(head, 50)
67 head = sisip_depan(head, 70) # head
68
69 # cetak isi linked-list awal
70 print("Isi Linked-List Sebelum Penghapusan")
71 cetak_linked_list(head)
72
73 # Penghapusan ditengah linked-list
74 head = hapus_tengah(head, 2)
75
76 # cetak isi setelah hapus tengah linked-list
77 print("\nIsi Linked-List Setelah Penghapusan Tengah ")
78 cetak_linked_list(head)

```

Outputnya

```

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

```

Penjelasannya

Baris 1: Komentar: Judul praktek, menjelaskan tujuan kode adalah menghapus node di posisi manapun (tengah).

Baris 2: Komentar: Menandai fungsi untuk membuat node baru di depan linked list.

Baris 3: Fungsi sisip_depan dimulai, membuat node baru.

Baris 4: Membuat dictionary node baru yang berisi data dan pointer ke head.

Baris 5: Mengembalikan node baru sebagai head.

Baris 7: Komentar: Fungsi untuk menghapus head node dan mengembalikan head baru.

Baris 8: Fungsi hapus_head dimulai.

Baris 9: Cek apakah linked list kosong (head None).

Baris 10: Jika kosong, tampilkan pesan.

Baris 11: Mengembalikan None karena list kosong.

Baris 12: Jika tidak kosong, cetak pesan node yang dihapus.

Baris 13: Kembalikan node setelah head sebagai head baru.

Baris 16: Komentar: Fungsi untuk menghapus node pada posisi manapun.

Baris 17: Fungsi hapus_tengah dimulai dengan parameter posisi.

Baris 18: Cek apakah linked list kosong.

Baris 19: Jika kosong, tampilkan pesan.

Baris 20: Kembalikan None karena list kosong.

Baris 22: Cek apakah posisi negatif (tidak valid).

Baris 23: Jika posisi tidak valid, tampilkan pesan.

Baris 24: Kembalikan linked list tanpa perubahan.

Baris 26: Cek apakah posisi sama dengan 0 (hapus head).

Baris 27: Tampilkan pesan penghapusan pada posisi 0.

Baris 28: Panggil fungsi hapus_head untuk menghapus head.

Baris 29: Kembalikan head baru setelah penghapusan.

Baris 31: Inisialisasi variabel untuk traversal linked list.

Baris 32: Inisialisasi indeks traversal.

Baris 34: Traversal untuk mencapai node sebelum posisi yang ingin dihapus.

Baris 35: Pindah ke node berikutnya.

Baris 36: Tambah indeks.

Baris 38: Cek apakah posisi melebihi panjang linked list.

Baris 39: Jika iya, tampilkan pesan peringatan.

Baris 40: Kembalikan linked list tanpa perubahan.

Baris 42: Tampilkan pesan node yang akan dihapus pada posisi yang ditentukan.

Baris 43: Ubah pointer node sebelumnya untuk melewati node yang dihapus.

Baris 44: Kembalikan linked list setelah penghapusan.

Baris 48: Komentar: Fungsi untuk menampilkan seluruh isi linked list.

Baris 49: Fungsi cetak_linked_list dimulai.

Baris 50: Inisialisasi variabel traversal.

Baris 51: Cetak awalan string "Head".

Baris 52: Loop untuk mencetak semua node selama node tidak None.

Baris 53: Cetak data node saat ini.

Baris 54: Pindah ke node berikutnya.

Baris 55: Setelah loop selesai, cetak "NULL" sebagai penanda akhir.

Baris 58: Komentar: Membuat linked list awal.

Baris 59: Inisialisasi linked list kosong (head None).

Baris 60-64: Menyisipkan node baru di depan linked list secara berurutan.

Baris 66: Komentar: Cetak isi linked list sebelum penghapusan.

Baris 67: Cetak pesan info.

Baris 68: Panggil fungsi cetak_linked_list untuk menampilkan isi.

Baris 70: Komentar: Hapus node di tengah linked list.

Baris 71: Panggil fungsi hapus_tengah pada posisi 2.

Baris 73: Komentar: Cetak isi linked list setelah penghapusan.

Baris 74: Cetak pesan info.

Baris 75: Panggil fungsi cetak_linked_list untuk menampilkan isi setelah penghapusan.