STRUKTUR DATA
STACK AND QUEUE

Oleh:
DANUARY BIMA HAMMAM MAYFALAH
24091397007

Program Studi D4 Manajemen Informatika
Fakultas Vokasi
Universitas Negeri Surabaya
2025

A. LAPORAN PRAKTIKUM

TUGAS NOMOR 1

1. Buatlah program Stack dengan Double Linked List

a) Gunakan input angka integer berbeda untuk menguji push(), pop(), dan peek().

b) Skenario:

• Input pertama: Tambah elemen [100, 200, 300] lalu lakukan pop() dua kali.

• Input kedua: Tambah elemen [50, 150, 250, 350] lalu lakukan peek() dan pop().

• Bandingkan hasilnya setelah dilakukan operasi pop() beberapa kali.

Source code

```python
# soal nomer 1
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
        self.prev = None

class StackDoubleLinkedList:
        def __init__(self):
            self.top = None

        def is_empty(self):
            return self.top is None

        def push(self, item):
            new_node = Node(item)
            if self.top:
                new_node.prev = self.top
            self.top = new_node

        def pop(self):
            if self.is_empty():
                print("Stack kosong!")
                return None
            item = self.top.data
            self.top = self.top.prev
            return item

        def peek(self):
            if self.is_empty():
                return None
            return self.top.data

stack = StackDoubleLinkedList()
print("Pertama")
stack.push(100)
stack.push(200)
stack.push(300)
print("Pop: ",stack.pop())
print("Pop: ",stack.pop())

print("kedua")
stack.push(50)
stack.push(150)
stack.push(250)
stack.push(350)
print("top stack:", stack.peek())
print("pop: ",stack.pop())

print("=" * 50)
print("=" * 50)
```

Output

```
Pertama
Pop:  300
Pop:  200
kedua
top stack: 350
pop:  350
```

Penjelasan tiap code

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
        self.prev = None
```

- Node adalah unit dasar dalam Double Linked List.
- Setiap Node memiliki tiga atribut:
  - data: Menyimpan nilai data.
  - next: Pointer ke Node berikutnya dalam linked list.
  - prev: Pointer ke Node sebelumnya dalam linked list

```
class StackDoubleLinkedList:
    def __init__(self):
        self.top = None
```

- **__init__**: Konstruktor untuk membuat stack kosong dengan atribut top yang merepresentasikan elemen teratas.

```
    def is_empty(self):
        return self.top is None
```

- Mengecek apakah stack kosong atau tidak.
- Jika self.top == None, berarti stack kosong, maka return True.

```
    def push(self, item):
        new_node = Node(item)
        if self.top:
            new_node.prev = self.top
        self.top = new_node
```

- Menambahkan item ke dalam stack.
- Membuat Node baru dengan data item.
- Jika stack tidak kosong, maka:
  - new_node.prev = self.top → Menghubungkan node baru ke node teratas sebelumnya.
- Node baru menjadi top.

```
        def pop(self):
            if self.is_empty():
                print("Stack kosong!")
                return None
            item = self.top.data
            self.top = self.top.prev
            return item
```

- Menghapus elemen teratas dari stack.
- Jika stack kosong, tampilkan pesan "Stack kosong!" dan return None.
- Simpan nilai self.top.data dalam variabel item.
- Geser top ke self.top.prev (node sebelumnya).
- Kembalikan nilai item.

```
        def peek(self):
            if self.is_empty():
                return None
            return self.top.data
```

- Mengembalikan nilai elemen teratas tanpa menghapusnya.
- Jika stack kosong, kembalikan None.

```
stack = StackDoubleLinkedList()
print("Pertama")
stack.push(100)
stack.push(200)
stack.push(300)
print("Pop: ",stack.pop())
print("Pop: ",stack.pop())
print("kedua")
stack.push(50)
stack.push(150)
stack.push(250)
stack.push(350)
print("top stack:", stack.peek())
print("pop: ",stack.pop())

print("=" * 50)
print("=" * 50)
```

TUGAS 2

a. Queue Biasa dengan Array

Gunakan input angka integer yang berbeda untuk menguji enqueue() dan dequeue().

Skenario:

➢ Input pertama: [10, 20, 30, 40, 50]

➢ Input kedua: [5, 15, 25, 35, 45, 55]

➢ Bandingkan hasilnya.

Source Code

```
class QueueArray:
    def __init__(self, size):
        self.queue = [None] * size
        self.front = 0
        self.rear = -1
        self.size = size
        self.count = 0

    def is_empty(self):
        return self.count == 0

    def is_full(self):
        return self.count == self.size

    def enqueue(self, item):
        if self.is_full():
            print("Queue penuh!")
            return
        self.rear += 1
        self.queue[self.rear] = item
        self.count += 1
        print(f"Setelah nilai {item}: masuk ---> {self.queue}")

    def dequeue(self):
        if self.is_empty():
            print("Queue kosong!")
            return None
        item = self.queue[self.front]
        self.queue[self.front] = None
        self.front += 1
        self.count -= 1
        print(f"Setelah nilai {item}: masuk ---> {self.queue}")
        return item

queue1 = QueueArray(5)
queue1.enqueue(10)
queue1.enqueue(20)
queue1.enqueue(30)
queue1.enqueue(40)
queue1.enqueue(50)
queue1.dequeue()
queue1.dequeue()
print("Queue setelah skenario 1:", queue1.queue)

queue2 = QueueArray(6)
queue2.enqueue(5)
queue2.enqueue(15)
queue2.enqueue(25)
queue2.enqueue(35)
queue2.enqueue(45)
queue2.enqueue(55)
queue2.dequeue()
queue2.dequeue()
print("Queue setelah skenario 2:", queue2.queue)


print("=" * 50)
print("=" * 50)
```

Output

```
Setelah nilai 10: masuk ---> [10, None, None, None, None]
Setelah nilai 20: masuk ---> [10, 20, None, None, None]
Setelah nilai 30: masuk ---> [10, 20, 30, None, None]
Setelah nilai 40: masuk ---> [10, 20, 30, 40, None]
Setelah nilai 50: masuk ---> [10, 20, 30, 40, 50]
Setelah nilai 10: masuk ---> [None, 20, 30, 40, 50]
Setelah nilai 20: masuk ---> [None, None, 30, 40, 50]
Queue setelah skenario 1: [None, None, 30, 40, 50]
Setelah nilai 5: masuk ---> [5, None, None, None, None, None]
Setelah nilai 15: masuk ---> [5, 15, None, None, None, None]
Setelah nilai 25: masuk ---> [5, 15, 25, None, None, None]
Setelah nilai 35: masuk ---> [5, 15, 25, 35, None, None]
Setelah nilai 45: masuk ---> [5, 15, 25, 35, 45, None]
Setelah nilai 55: masuk ---> [5, 15, 25, 35, 45, 55]
Setelah nilai 5: masuk ---> [None, 15, 25, 35, 45, 55]
Setelah nilai 15: masuk ---> [None, None, 25, 35, 45, 55]
Queue setelah skenario 2: [None, None, 25, 35, 45, 55]
================================================
================================================
```

Penjelasan tiap code

```
class QueueArray:
    def __init__(self, size):
        self.queue = [None] * size
        self.front = 0
        self.rear = -1
        self.size = size
        self.count = 0
```

- self.queue = [None] * size → Membuat array dengan kapasitas size, yang diisi dengan None sebagai nilai awal.
- self.front = 0 → Pointer yang menunjuk ke elemen depan antrian.
- self.rear = -1 → Pointer yang menunjuk ke elemen belakang antrian (karena antrian kosong, maka dimulai dari -1).
- self.size = size → Menyimpan ukuran maksimum dari queue.
- self.count = 0 → Menyimpan jumlah elemen saat ini dalam queue.

```
    def is_empty(self):
        return self.count == 0
```

- Mengecek apakah queue kosong.
- Mengembalikan True jika **count** (jumlah elemen) adalah 0.

```
    def is_full(self):
        return self.count == self.size
```

- Mengecek apakah queue penuh.
- Mengembalikan True jika count sama dengan size.

```
    def enqueue(self, item):
        if self.is_full():
            print("Queue penuh!")
            return
        self.rear += 1
        self.queue[self.rear] = item
        self.count += 1
        print(f"Setelah nilai {item}: masuk ---> {self.queue}")
```

- Cek apakah queue penuh: Jika penuh, cetak "Queue penuh!" dan hentikan proses.
- Geser pointer rear (self.rear += 1) untuk menunjukkan tempat baru di queue.
- Tambahkan item ke dalam self.queue[self.rear].
- Tingkatkan jumlah elemen (self.count += 1).
- Cetak queue setelah item masuk.

```
    def dequeue(self):
        if self.is_empty():
            print("Queue kosong!")
            return None
        item = self.queue[self.front]
        self.queue[self.front] = None
        self.front += 1
        self.count -= 1
        print(f"Setelah nilai {item}: masuk ---> {self.queue}")
        return item
```

- Cek apakah queue kosong: Jika kosong, cetak "Queue kosong!" dan return None.
- Ambil elemen dari depan (self.queue[self.front]).
- Hapus elemen di depan dengan mengubahnya menjadi None.
- Geser front (self.front += 1) agar menunjuk ke elemen berikutnya.
- Kurangi jumlah elemen (self.count -= 1).
- Cetak queue setelah elemen dikeluarkan.

```
queue1 = QueueArray(5)
queue1.enqueue(10)
queue1.enqueue(20)
queue1.enqueue(30)
queue1.enqueue(40)
queue1.enqueue(50)
queue1.dequeue()
queue1.dequeue()
print("Queue setelah skenario 1:", queue1.queue)
```

- Membuat queue dengan kapasitas 5.
- Memasukkan elemen: 10, 20, 30, 40, 50 → Queue penuh setelah itu.
- Menghapus dua elemen (dequeue() dua kali) → 10 dan 20 dihapus.
- Cetak queue setelah proses ini.

```
queue2 = QueueArray(6)
queue2.enqueue(5)
queue2.enqueue(15)
queue2.enqueue(25)
queue2.enqueue(35)
queue2.enqueue(45)
queue2.enqueue(55)
queue2.dequeue()
queue2.dequeue()
print("Queue setelah skenario 2:", queue2.queue)
```

- Membuat queue dengan kapasitas 6.
- Memasukkan elemen: 5, 15, 25, 35, 45, 55 → Queue penuh setelah itu.
- Menghapus dua elemen (dequeue() dua kali) → 5 dan 15 dihapus.
- Cetak queue setelah proses ini.

b. Circularqueue

Gunakan ukuran queue yang berbeda dan input angka integer yang berbeda.

Skenario

- **Queue ukuran 5 dengan input: [1, 2, 3, 4, 5]**
- **Queue ukuran 7 dengan input: [10, 20, 30, 40, 50, 60, 70]**
- **Uji kondisi saat queue penuh dan kosong.**

Source code

```
class CircularQueue :
    def __init__(self, size):
        self.queue = [None] * size
        self.front = -1
        self.rear = -1
        self.size = size

    def is_empty(self):
        return self.front == -1

    def is_full(self):
        return (self.rear + 1) % self.size == self.front

    def enqueue(self, item):
        if self.is_full():
            print("Queue penuh!")
            return
        if self.is_empty():
            self.front = 0
        self.rear = (self.rear + 1) % self.size
        self.queue[self.rear] = item
        print (f"Setelah nilai {item}: masuk ---> {self.queue}")

    def dequeue(self):
        if self.is_empty():
            print("Queue kosong!")
            return None
        item = self.queue[self.front]
        self.queue[self.front] = None
        if self.front == self.rear:
            self.front = -1
            self.rear = -1
        else:
            self.front = (self.front + 1) % self.size
        print(f"Setelah nilai {item}: keluar ---> {self.queue}")
        return item

print("========== queue ukuran 5 ==========")
cq1 = CircularQueue(5)
print("queue kosong!")
cq1.enqueue(1)
cq1.enqueue(2)
cq1.enqueue(3)
cq1.dequeue()
cq1.enqueue(4)
cq1.enqueue(5)

print("========== queue ukuran 7 ==========")
cq2 = CircularQueue(7)
cq2.enqueue(10)
cq2.enqueue(20)
cq2.enqueue(30)
cq2.enqueue(40)
cq2.enqueue(50)
cq2.enqueue(60)
cq2.enqueue(70)
print("queue penuh!")
cq2.dequeue()
cq2.dequeue()
```

Output

```
========== queue ukuran 5 ==========
queue kosong!
Setelah nilai 1: masuk ---> [1, None, None, None, None]
Setelah nilai 2: masuk ---> [1, 2, None, None, None]
Setelah nilai 3: masuk ---> [1, 2, 3, None, None]
Setelah nilai 1: keluar ---> [None, 2, 3, None, None]
Setelah nilai 4: masuk ---> [None, 2, 3, 4, None]
Setelah nilai 5: masuk ---> [None, 2, 3, 4, 5]
========== queue ukuran 7 ==========
Setelah nilai 10: masuk ---> [10, None, None, None, None, None, None]
Setelah nilai 20: masuk ---> [10, 20, None, None, None, None, None]
Setelah nilai 30: masuk ---> [10, 20, 30, None, None, None, None]
Setelah nilai 40: masuk ---> [10, 20, 30, 40, None, None, None]
Setelah nilai 50: masuk ---> [10, 20, 30, 40, 50, None, None]
Setelah nilai 60: masuk ---> [10, 20, 30, 40, 50, 60, None]
Setelah nilai 70: masuk ---> [10, 20, 30, 40, 50, 60, 70]
queue penuh!
Setelah nilai 10: keluar ---> [None, 20, 30, 40, 50, 60, 70]
Setelah nilai 20: keluar ---> [None, None, 30, 40, 50, 60, 70]
```

Penjelasan tiap code

```
class CircularQueue :
    def __init__(self, size):
        self.queue = [None] * size
        self.front = -1
        self.rear = -1
        self.size = size
```

- self.queue = [None] * size → Membuat array dengan kapasitas size, semua elemen awalnya None.
- self.front = -1 → Menunjuk ke elemen depan queue, awalnya -1 (artinya queue kosong).
- self.rear = -1 → Menunjuk ke elemen belakang queue, awalnya -1.
- self.size = size → Menyimpan ukuran maksimum queue.

```
    def is_empty(self):
        return self.front == -1
```

- Mengembalikan True jika front == -1, artinya queue kosong.

```
    def is_full(self):
        return (self.rear + 1) % self.size == self.front
```

- Cek apakah queue penuh menggunakan rumus modulus:
  - Jika rear + 1 sama dengan front dalam format circular, maka queue penuh.
  - (self.rear + 1) % self.size == self.front memastikan jika rear sudah berada di posisi terakhir, maka ia akan kembali ke index awal.

```
    def enqueue(self, item):
        if self.is_full():
            print("Queue penuh!")
            return
        if self.is_empty():
            self.front = 0
        self.rear = (self.rear + 1) % self.size
        self.queue[self.rear] = item
        print (f"Setelah nilai {item}: masuk ---> {self.queue}")
```

- Cek apakah queue penuh (is_full()).
- Jika kosong (is_empty()), set front = 0.
- Menentukan posisi rear:
  - (self.rear + 1) % self.size membuat queue bersifat melingkar.
- Masukkan item ke dalam queue di indeks rear.
- Cetak kondisi queue setelah enqueue().

```
    def dequeue(self):
        if self.is_empty():
            print("Queue kosong!")
            return None
        item = self.queue[self.front]
        self.queue[self.front] = None
        if self.front == self.rear:
                self.front = -1
                self.rear = -1
        else:
            self.front = (self.front + 1) % self.size
        print(f"Setelah nilai {item}: keluar ---> {self.queue}")
        return item
```

- Cek apakah queue kosong (is_empty()).
- Ambil elemen di front.
- Set posisi front ke None (menghapus data).
- Jika hanya ada satu elemen (front == rear), reset queue ke keadaan kosong (front = -1, rear = -1).
- Jika tidak, front berpindah ke posisi berikutnya secara circular:
  - (self.front + 1) % self.size.

```
print("============ queue ukuran 5 ==========")
cq1 = CircularQueue(5)
print("queue kosong!")
cq1.enqueue(1)
cq1.enqueue(2)
cq1.enqueue(3)
cq1.dequeue()
cq1.enqueue(4)
cq1.enqueue(5)
```

- Membuat Circular Queue dengan ukuran 5.
- Cetak "queue kosong!", karena awalnya queue kosong.
- Memasukkan angka 1, 2, 3.
- Menghapus angka 1 (dequeue()).
- Memasukkan angka 4 dan 5.

```
print("=========== queue ukuran 7 ==========")
cq2 = CircularQueue(7)
cq2.enqueue(10)
cq2.enqueue(20)
cq2.enqueue(30)
cq2.enqueue(40)
cq2.enqueue(50)
cq2.enqueue(60)
cq2.enqueue(70)
print("queue penuh!")
cq2.dequeue()
cq2.dequeue()
```

- Membuat Circular Queue dengan ukuran 7.
- Memasukkan angka 10, 20, 30, 40, 50, 60, 70 (Queue penuh).
- Cetak "queue penuh!" karena queue sudah terisi maksimal.
- Menghapus angka 10 dan 20 (dequeue() dua kali).

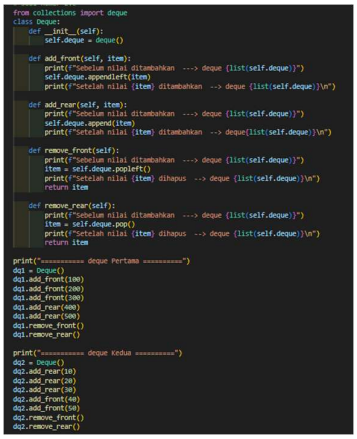
c. Deque (Double-ended Queue)

Gunakan input angka integer berbeda untuk add_front(), add_rear(), remove_front(), dan remove_rear().

• Skenario

Input pertama: Tambah dari depan [100, 200, 300] → Tambah dari belakang[400, 500]
Input kedua: Tambah dari belakang [10, 20, 30] → Tambah dari depan [40, 50]
Bandingkan hasilnya setelah dilakukan operasi remove.

Source code



Output

```
========== deque Pertama ==========
Sebelum nilai ditambahkan  ---> deque []
Setelah nilai 100 ditambahkan  --> deque [100]

Sebelum nilai ditambahkan  ---> deque [100]
Setelah nilai 200 ditambahkan  --> deque [200, 100]

Sebelum nilai ditambahkan  ---> deque [200, 100]
Setelah nilai 300 ditambahkan  --> deque [300, 200, 100]

Sebelum nilai ditambahkan  ---> deque [300, 200, 100]
Setelah nilai 400 ditambahkan  --> deque[300, 200, 100, 400]

Sebelum nilai ditambahkan  ---> deque [300, 200, 100, 400]
Setelah nilai 500 ditambahkan  --> deque[300, 200, 100, 400, 500]

Sebelum nilai ditambahkan  ---> deque [300, 200, 100, 400, 500]
Setelah nilai 300 dihapus  --> deque [200, 100, 400, 500]

Sebelum nilai ditambahkan  ---> deque [200, 100, 400, 500]
Setelah nilai 500 dihapus  --> deque [200, 100, 400]
```

```
========== deque Kedua ==========
Sebelum nilai ditambahkan  ---> deque []
Setelah nilai 10 ditambahkan  --> deque[10]

Sebelum nilai ditambahkan  ---> deque [10]
Setelah nilai 20 ditambahkan  --> deque[10, 20]

Sebelum nilai ditambahkan  ---> deque [10, 20]
Setelah nilai 30 ditambahkan  --> deque[10, 20, 30]

Sebelum nilai ditambahkan  ---> deque [10, 20, 30]
Setelah nilai 40 ditambahkan  --> deque [40, 10, 20, 30]

Sebelum nilai ditambahkan  ---> deque [40, 10, 20, 30]
Setelah nilai 50 ditambahkan  --> deque [50, 40, 10, 20, 30]

Sebelum nilai ditambahkan  ---> deque [50, 40, 10, 20, 30]
Setelah nilai 50 dihapus  --> deque [40, 10, 20, 30]

Sebelum nilai ditambahkan  ---> deque [40, 10, 20, 30]
Setelah nilai 30 dihapus  --> deque [40, 10, 20]
```

Penjelasan tiap code

```
from collections import deque
class Deque:
    def __init__(self):
        self.deque = deque()
```

- from collections import deque → Mengimpor deque, yang merupakan implementasi deque yang lebih efisien dibandingkan list biasa.
- self.deque = deque() → Membuat objek deque kosong untuk menyimpan elemen.

```
    def add_front(self, item):
        print(f"Sebelum nilai ditambahkan  ---> deque {list(self.deque)}")
        self.deque.appendleft(item)
        print(f"Setelah nilai {item} ditambahkan  --> deque {list(self.deque)}\n")
```

- self.deque.appendleft(item) → Menambahkan item ke depan (kiri) deque.
- Cetak kondisi deque sebelum dan sesudah penambahan.

```
def add_rear(self, item):
    print(f"Sebelum nilai ditambahkan  ---> deque {list(self.deque)}")
    self.deque.append(item)
    print(f"Setelah nilai {item} ditambahkan  --> deque{list(self.deque)}\n")
```

- self.deque.append(item) → Menambahkan item ke belakang (kanan) deque.
- Cetak kondisi deque sebelum dan sesudah penambahan.

```
def remove_front(self):
    print(f"Sebelum nilai ditambahkan  ---> deque {list(self.deque)}")
    item = self.deque.popleft()
    print(f"Setelah nilai {item} dihapus  --> deque {list(self.deque)}\n")
    return item
```

- self.deque.popleft() → Menghapus elemen dari depan (kiri) deque.
- Cetak kondisi deque sebelum dan sesudah penghapusan.
- Return elemen yang dihapus.

```
def remove_rear(self):
    print(f"Sebelum nilai ditambahkan  ---> deque {list(self.deque)}")
    item = self.deque.pop()
    print(f"Setelah nilai {item} dihapus  --> deque {list(self.deque)}\n")
    return item
```

- self.deque.pop() → Menghapus elemen dari belakang (kanan) deque.
- Cetak kondisi deque sebelum dan sesudah penghapusan.
- Return elemen yang dihapus.

```
print("========== deque Pertama ==========")
dq1 = Deque()
dq1.add_front(100)
dq1.add_front(200)
dq1.add_front(300)
dq1.add_rear(400)
dq1.add_rear(500)
dq1.remove_front()
dq1.remove_rear()
```

- Membuat deque kosong (dq1 = Deque()).
- Menambahkan elemen dari depan:
  - dq1.add_front(100) → [100]
  - dq1.add_front(200) → [200, 100]
  - dq1.add_front(300) → [300, 200, 100]
- Menambahkan elemen dari belakang:
  - dq1.add_rear(400) → [300, 200, 100, 400]
  - dq1.add_rear(500) → [300, 200, 100, 400, 500]
- Menghapus elemen dari depan (remove_front()) → 300 dihapus → [200, 100, 400, 500]
- Menghapus elemen dari belakang (remove_rear()) → 500 dihapus → [200, 100, 400]

```
print("========== deque Kedua ==========")
dq2 = Deque()
dq2.add_rear(10)
dq2.add_rear(20)
dq2.add_rear(30)
dq2.add_front(40)
dq2.add_front(50)
dq2.remove_front()
dq2.remove_rear()
```

- Membuat deque kosong (dq2 = Deque()).
- Menambahkan elemen dari belakang:
  - dq2.add_rear(10) → [10]
  - dq2.add_rear(20) → [10, 20]
  - dq2.add_rear(30) → [10, 20, 30]
- Menambahkan elemen dari depan:
  - dq2.add_front(40) → [40, 10, 20, 30]
  - dq2.add_front(50) → [50, 40, 10, 20, 30]
- Menghapus elemen dari depan (remove_front()) → 50 dihapus → [40, 10, 20, 30]
- Menghapus elemen dari belakang (remove_rear()) → 30 dihapus → [40, 10, 20]

d. Priority Queue

Uji dengan angka integer yang berbeda dan lihat bagaimana elemen diproses berdasarkan prioritasnya.

Contoh skenario:

Input pertama: [(2, 100), (1, 200), (3, 50)] → Prioritas lebih kecil lebih dulu diproses.

Input kedua: [(5, 10), (3, 30), (4, 20)]

Bandingkan urutan elemen yang diproses.

Source Code

```
import heapq
class PriorityQueue:
    def __init__(self):
        self.queue = []

    def enqueue(self, item, priority):
        print(f"Sebelum nilai masuk   --> {self.queue}")
        heapq.heappush(self.queue, (priority, item))
        print(f"Setelah nilai {item} masuk, menambahkan {priority} prioritas  --> {self.queue}\n")

    def dequeue(self):
        print(f"Sebelum nilai masuk   --> {self.queue}")
        item = heapq.heappop(self.queue)[1]
        print(f"Setelah nilai {item} dihapus --> {self.queue}\n")
        return item

print("==============Priority Pertama==============")
pq1 = PriorityQueue()
pq1.enqueue(100, 2)
pq1.enqueue(200, 1)
pq1.enqueue(300, 3)
pq1.dequeue()
pq1.dequeue()

print("==============Priority Kedua==============")
pq1 = PriorityQueue()
pq1.enqueue(10, 5)
pq1.enqueue(30, 3)
pq1.enqueue(20, 4)
pq1.enqueue(50, 2)
pq1.dequeue()
```

Output

```
==============Priority Pertama==============
Sebelum nilai masuk   --> []
Setelah nilai 100 masuk, menambahkan 2 prioritas  --> [(2, 100)]

Sebelum nilai masuk   --> [(2, 100)]
Setelah nilai 200 masuk, menambahkan 1 prioritas  --> [(1, 200), (2, 100)]

Sebelum nilai masuk   --> [(1, 200), (2, 100)]
Setelah nilai 300 masuk, menambahkan 3 prioritas  --> [(1, 200), (2, 100), (3, 300)]

Sebelum nilai masuk   --> [(1, 200), (2, 100), (3, 300)]
Setelah nilai 200 dihapus --> [(2, 100), (3, 300)]

Sebelum nilai masuk   --> [(2, 100), (3, 300)]
Setelah nilai 100 dihapus --> [(3, 300)]

==============Priority Kedua==============
Sebelum nilai masuk   --> []
Setelah nilai 10 masuk, menambahkan 5 prioritas  --> [(5, 10)]

Sebelum nilai masuk   --> [(5, 10)]
Setelah nilai 30 masuk, menambahkan 3 prioritas  --> [(3, 30), (5, 10)]

Sebelum nilai masuk   --> [(3, 30), (5, 10)]
Setelah nilai 20 masuk, menambahkan 4 prioritas  --> [(3, 30), (5, 10), (4, 20)]

Sebelum nilai masuk   --> [(3, 30), (5, 10), (4, 20)]
Setelah nilai 50 masuk, menambahkan 2 prioritas  --> [(2, 50), (3, 30), (4, 20), (5, 10)]

Sebelum nilai masuk   --> [(2, 50), (3, 30), (4, 20), (5, 10)]
Setelah nilai 50 dihapus --> [(3, 30), (5, 10), (4, 20)]
```

Penjelasan tiap code

```python
import heapq
class PriorityQueue:
    def __init__(self):
        self.queue = []
```

- import heapq → Mengimpor heapq, modul yang mengimplementasikan min-heap di Python.
- self.queue = [] → Membuat list kosong untuk menyimpan elemen dalam bentuk heap.

```python
    def enqueue(self, item, priority):
        print(f"Sebelum nilai masuk   --> {self.queue}")
        heapq.heappush(self.queue, (priority, item))
        print(f"Setelah nilai {item} masuk, menambahkan {priority} prioritas  --> {self.queue}\n")
```

- heapq.heappush(self.queue, (priority, item)) → Menambahkan elemen ke dalam heap, di mana priority menentukan urutan.
- Elemen dengan priority terkecil akan berada di puncak heap.
- Cetak kondisi queue sebelum dan sesudah penambahan.

```python
    def dequeue(self):
        print(f"Sebelum nilai masuk   --> {self.queue}")
        item = heapq.heappop(self.queue)[1]
        print(f"Setelah nilai {item} dihapus --> {self.queue}\n")
        return item
```

- heapq.heappop(self.queue) → Menghapus elemen dengan prioritas tertinggi (priority terkecil).
- Cetak kondisi queue sebelum dan sesudah penghapusan.
- Return item yang dihapus (bukan prioritasnya).

```
print("===============Priority Pertama===============")
pq1 = PriorityQueue()
pq1.enqueue(100, 2)
pq1.enqueue(200, 1)
pq1.enqueue(300, 3)
pq1.dequeue()
pq1.dequeue()
```

- Membuat priority queue kosong (pq1 = PriorityQueue()).
- Menambahkan elemen dengan prioritas:
  - pq1.enqueue(100, 2) → [(2, 100)]
  - pq1.enqueue(200, 1) → [(1, 200), (2, 100)]
  - pq1.enqueue(300, 3) → [(1, 200), (2, 100), (3, 300)]
- Menghapus elemen dengan prioritas tertinggi:
  - pq1.dequeue() → 200 (priority 1) keluar → [(2, 100), (3, 300)]
  - pq1.dequeue() → 100 (priority 2) keluar → [(3, 300)]

```
print("===============Priority Kedua===============")
pq1 = PriorityQueue()
pq1.enqueue(10, 5)
pq1.enqueue(30, 3)
pq1.enqueue(20, 4)
pq1.enqueue(50, 2)
pq1.dequeue()
```

- Membuat priority queue kosong (pq1 = PriorityQueue()).
- Menambahkan elemen dengan prioritas:
  - pq1.enqueue(10, 5) → [(5, 10)]
  - pq1.enqueue(30, 3) → [(3, 30), (5, 10)]
  - pq1.enqueue(20, 4) → [(3, 30), (5, 10), (4, 20)]
  - pq1.enqueue(50, 2) → [(2, 50), (3, 30), (4, 20), (5, 10)]
- Menghapus elemen dengan prioritas tertinggi:
  - pq1.dequeue() → 50 (priority 2) keluar → [(3, 30), (5, 10), (4, 20)]

e. Queue dengan Linked List

Gunakan input angka integer yang berbeda untuk menguji enqueue() dan dequeue().

Contoh skenario:

- **Input pertama: [8, 16, 24, 32, 40]**
- **Input kedua: [11, 22, 33, 44, 55] Bandingkan hasil setelah operasi dequeue dilakukan.**

Source code

```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class QueueLinkedList:
    def __init__(self):
        self.front = self.rear = None

    def is_empty(self):
        return self.front is None

    def enque
        print                                    ", end="")
        self.display()
        new_node = Node(item)
        if self.rear is None:
            self.front = self.rear = new_node
        else:
            self.rear.next = new_node
            self.rear = new_node
        print(f"Setelah nilai {item} ditambahkan  --> ", end="")
        self.display()

    def dequeue(self):
        print("\nSebelum nilai dihapus  --> ", end="")
        self.display()
        if self.is_empty():
            print("Queue kosong!")
            return
        item = self.front.data
        self.front = self.front.next
        if self.front is None:
            self.rear = None
        print(f"Setelah nilai {item} dihapus  --> ", end="")
        self.display()

    def display(self):
        temp = self.front
        while temp:
            print(temp.data, end=" ")
            temp = temp.next
        print("None")

print("==============Queue dengan linked List pertama==============")
ql1 = QueueLinkedList()
ql1.enqueue(8)
ql1.enqueue(16)
ql1.enqueue(24)
ql1.dequeue()
ql1.enqueue(32)
ql1.enqueue(40)
ql1.dequeue()

print("==============Queue dengan linked List kedua==============")
ql2 = QueueLinkedList()
ql2.dequeue()
ql2.enqueue(11)
ql2.enqueue(22)
ql2.enqueue(33)
ql2.dequeue()
ql2.enqueue(44)
ql2.enqueue(55)
ql2.dequeue()
print("=" * 50)
print("=" * 50)
```

(method) def display() -> None

Output

```
==============Queue dengan linked List pertama==============

Sebelum nilai ditambahkan  --> None
Setelah nilai 8 ditambahkan  --> 8 None

Sebelum nilai ditambahkan  --> 8 None
Setelah nilai 16 ditambahkan  --> 8 16 None

Sebelum nilai ditambahkan  --> 8 16 None
Setelah nilai 24 ditambahkan  --> 8 16 24 None

Sebelum nilai dihapus  --> 8 16 24 None
Setelah nilai 8 dihapus  --> 16 24 None

Sebelum nilai ditambahkan  --> 16 24 None
Setelah nilai 32 ditambahkan  --> 16 24 32 None

Sebelum nilai ditambahkan  --> 16 24 32 None
Setelah nilai 40 ditambahkan  --> 16 24 32 40 None

Sebelum nilai dihapus  --> 16 24 32 40 None
Setelah nilai 16 dihapus  --> 24 32 40 None
==============Queue dengan linked List kedua==============

Sebelum nilai dihapus  --> None
Queue kosong!

Sebelum nilai ditambahkan  --> None
Setelah nilai 11 ditambahkan  --> 11 None

Sebelum nilai ditambahkan  --> 11 None
Setelah nilai 22 ditambahkan  --> 11 22 None

Sebelum nilai ditambahkan  --> 11 22 None
Setelah nilai 33 ditambahkan  --> 11 22 33 None

Sebelum nilai dihapus  --> 11 22 33 None
Setelah nilai 11 dihapus  --> 22 33 None

Sebelum nilai ditambahkan  --> 22 33 None
Setelah nilai 44 ditambahkan  --> 22 33 44 None

Sebelum nilai ditambahkan  --> 22 33 44 None
Setelah nilai 55 ditambahkan  --> 22 33 44 55 None

Sebelum nilai dihapus  --> 22 33 44 55 None
Setelah nilai 22 dihapus  --> 33 44 55 None
```

Penjelasan tiap code

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
```

- Mendefinisikan node untuk menyimpan data dan pointer ke node berikutnya.
- self.data menyimpan nilai dari node.
- self.next menunjuk ke node berikutnya dalam antrian.

```
class QueueLinkedList:
    def __init__(self):
        self.front = self.rear = None
```

- self.front menunjuk ke elemen pertama dalam antrian.
- self.rear menunjuk ke elemen terakhir dalam antrian.

```
    def is_empty(self):
        return self.front is None
```

- Mengembalikan True jika antrian kosong (tidak ada elemen).
- self.front akan None jika tidak ada elemen dalam antrian.

```
def enqueue(self, item):
    print("\nSebelum nilai ditambahkan  --> ", end="")
    self.display()
    new_node = Node(item)
    if self.rear is None:
        self.front = self.rear = new_node
    else:
        self.rear.next = new_node
        self.rear = new_node
    print(f"Setelah nilai {item} ditambahkan  --> ", end="")
    self.display()
```

- Menampilkan kondisi antrian sebelum elemen ditambahkan.
- Membuat node baru (new_node) dengan data item.
  - Jika antrian kosong (self.rear is None):
- Node baru menjadi front dan rear.
  - Jika antrian tidak kosong:
- Node baru ditambahkan setelah rear, dan rear diperbarui.
- Menampilkan kondisi antrian setelah penambahan.

```
def dequeue(self):
    print("\nSebelum nilai dihapus  --> ", end="")
    self.display()
    if self.is_empty():
        print("Queue kosong!")
        return
    item = self.front.data
    self.front = self.front.next
    if self.front is None:
        self.rear = None
    print(f"Setelah nilai {item} dihapus  --> ", end="")
    self.display()
```

- Menampilkan kondisi antrian sebelum elemen dihapus.
- Jika antrian kosong, cetak "Queue kosong!".
- Ambil nilai dari front (elemen pertama).
- Geser front ke elemen berikutnya (self.front.next).
- Jika setelah penghapusan front menjadi None, maka rear juga harus None.
- Menampilkan kondisi antrian setelah penghapusan.

```
def display(self):
    temp = self.front
    while temp:
        print(temp.data, end=" ")
        temp = temp.next
    print("None")
```

- Iterasi mulai dari front hingga None, menampilkan data dari setiap node.

```
print("==============Queue dengan linked List pertama==============")
ql1 = QueueLinkedList()
ql1.enqueue(8)
ql1.enqueue(16)
ql1.enqueue(24)
ql1.dequeue()
ql1.enqueue(32)
ql1.enqueue(40)
ql1.dequeue()
```

- ql2 = QueueLinkedList() → Membuat antrian kosong.
- ql2.dequeue() → Antrian kosong! Cetak: "Queue kosong!"
- ql2.enqueue(11) → Tambah 11, antrian: 11 → None
- ql2.enqueue(22) → Tambah 22, antrian: 11 → 22 → None
- ql2.enqueue(33) → Tambah 33, antrian: 11 → 22 → 33 → None
- ql2.dequeue() → Hapus 11, antrian: 22 → 33 → None
- ql2.enqueue(44) → Tambah 44, antrian: 22 → 33 → 44 → None
- ql2.enqueue(55) → Tambah 55, antrian: 22 → 33 → 44 → 55 → None
- ql2.dequeue() → Hapus 22, antrian: 33 → 44 → 55 → None

```
print("===============Queue dengan linked List kedua===============")
ql2 = QueueLinkedList()
ql2.dequeue()
ql2.enqueue(11)
ql2.enqueue(22)
ql2.enqueue(33)
ql2.dequeue()
ql2.enqueue(44)
ql2.enqueue(55)
ql2.dequeue()
```

- ql2 = QueueLinkedList() → Membuat antrian kosong.
- ql2.dequeue() → Antrian kosong! Cetak "Queue kosong!"
- ql2.enqueue(11) → Tambah 11, antrian: 11 → None
- ql2.enqueue(22) → Tambah 22, antrian: 11 → 22 → None
- ql2.enqueue(33) → Tambah 33, antrian: 11 → 22 → 33 → None
- ql2.dequeue() → Hapus 11, antrian: 22 → 33 → None
- ql2.enqueue(44) → Tambah 44, antrian: 22 → 33 → 44 → None
- ql2.enqueue(55) → Tambah 55, antrian: 22 → 33 → 44 → 55 → None
- ql2.dequeue() → Hapus 22, antrian: 33 → 44 → 55 → None