Bab 5. Queue

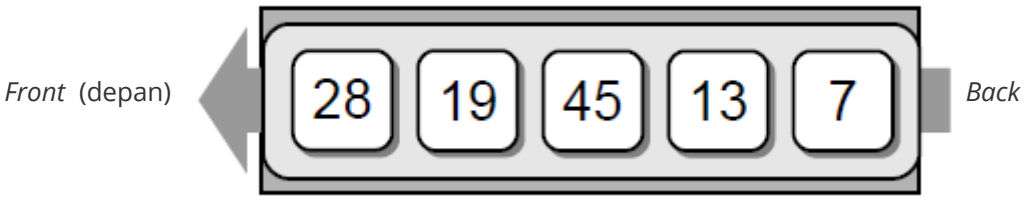
OBJEKTIF:

1. Mahasiswa mampu memahami struktur Queue, mengimplementasikan ADT Queue menggunakan `List`, dan mengimplementasikan ADT Queue menggunakan Linked List.

5.1 Queue

Queue dalam Bahasa Indonesia diterjemahkan sebagai "antrian". Struktur queue adalah struktur data yang memiliki cara kerja seperti sebuah antrian orang yang sedang menunggu untuk dilayani. Gambar berikut mengilustrasikan antrian:

Proses menambah dan mengurangi orang pada antrian menerapkan prinsip *First-In First-Out* (FIFO) yang berarti yang pertama masuk adalah yang pertama keluar. Konsep FIFO ini berarti menambah orang pada antrian hanya dapat dilakukan di sisi belakang dan mengurangi orang hanya dapat dilakukan di sisi depan. Gambar berikut mengilustrasikan proses penambahan dan pengurangan pada queue:



Pada struktur **queue**, data baru ditambahkan dari bagian ***back*** (belakang) queue sedangkan pengambilan data dilakukan dari bagian ***front*** (depan). Proses penambahan data pada queue disebut **enqueue** dan proses mengeluarkan data dari queue disebut **dequeue**. Meskipun pada gambar di atas elemen-elemen queue dapat dilihat urutannya, namun elemen-elemen di queue ini tidak dapat diakses secara langsung.

ADT Queue

Definisi ADT Queue

Queue adalah sebuah struktur data yang menyimpan koleksi *linear* data yang dimana aksesnya terbatas pada konsep *first-in first-out*. Elemen baru disisipkan pada bagian belakang dan elemen lama diambil dari bagian depan struktur queue. Susunan kumpulan elemen tidak berubah sejak elemen pertama ditambahkan pada struktur queue. Operasi-operasi yang dimiliki oleh queue adalah sebagai berikut:

- `Queue()` : Membuat struktur queue baru yang kosong.
- `isEmpty()` : Mengembalikan nilai boolean yang menandakan apakah struktur queue kosong.
- `length()` : Mengembalikan jumlah banyaknya elemen yang ada didalam queue.
- `enqueue(data)` : Menambahkan data baru ke bagian belakang queue.
- `dequeue()` : Menghapus dan mengembalikan elemen bagian depan dari queue. Sebuah data tidak dapat dihapus dan dikembalikan dari queue yang kosong.

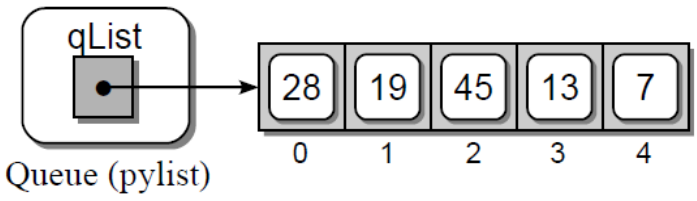
Tabel berikut mencontohkan penggunaan operasi-operasi pada queue beserta nilai kembali dan perubahan dari elemen struktur queue:

Operasi	Nilai Kembali	Isi Queue (kiri adalah front)
`Q = Queue()`	-	[ ]
`Q.enqueue(2)`	-	[2]
`Q.enqueue(7)`	-	[2, 7]
`Q.enqueue(5)`	-	[2, 7, 5]
`len(Q)`	3	[2, 7, 5]
`Q.dequeue()`	2	[7, 5]
`Q.dequeue()`	7	[5]
`Q.dequeue()`	5	[ ]
`Q.dequeue()`	Error	[ ]

ADT Queue dapat diimplementasikan menggunakan `list` atau linked list. Kita akan membahas keduanya.

5.2 Implementasi ADT Queue Menggunakan `list`

Implementasi queue paling sederhana adalah menggunakan `list` pada Python. `list` pada Python menyediakan proses penambahan dan penghapusan elemen queue yang dibutuhkan. Dengan menggunakan proses tersebut, kita dapat mengapus elemen data dari bagian depan `list` dan menambahkan elemen baru dibagian akhir `list`. Berikut adalah gambar ilustrasi implementasi ADT Queue menggunakan `list`:



Implementasi queue kita mulai dengan mendefinisikan class `Queue`:

```
class Queue:

 # .... Implementasi method-method
```

Constructor `Queue`

Class `Queue` hanya membutuhkan sebuah *field* yang mendefinisikan sebuah `list` kosong untuk menyimpan data dari queue. Berikut adalah definisi constructor `Queue`:

```
def __init__(self):
    self._qList = list()
```

Method `isEmpty()`

Method `isEmpty()` mengembalikan `True` jika queue kosong dan mengembalikan `False` jika tidak kosong. Kita hanya perlu mengembalikan sebuah nilai boolean yang menandakan apakah `list` yang digunakan untuk menyimpan elemen data queue mempunyai panjang sama dengan 0. Definisi method `isEmpty()` dapat dituliskan seperti berikut:

```
def isEmpty(self):
    return len(self) == 0
```

Method `len()`

Method `len()` mengembalikan nilai dari panjang elemen data yang ada pada `list` penyimpan struktur queue. Definisi method `len()` dapat dituliskan seperti berikut:

```
def __len__(self):
    return len(self._qList)
```

Method `enqueue(data)`

Method `enqueue(data)` menambahkan elemen data baru pada `list` dari queue. Kita dapat menggunakan method `append` dari `list` untuk melakukan ini. Sehingga, definisi method `enqueue(data)` dapat dituliskan seperti berikut:

```
def enqueue(self, data):
    self._qList.append(data)
```

Method `dequeue()`

Method `dequeue()` menghapus dan mengembalikan nilai elemen bagian depan queue. Kondisi yang perlu diperhatikan pada implementasi method `dequeue()` adalah jika method ini dipanggil pada queue kosong, maka program akan meng-*raise exception* generik dengan pesan queue kosong. Lalu, kita dapat menggunakan method `pop` pada `list` untuk menghapus data dari queue. Sehingga kita dapat menuliskan definisi method `dequeue()` seperti berikut:

```
def dequeue(self):
    if self.isEmpty():
        raise Exception('Queue kosong. Tidak ada data yang dapat di-dequeue.')
    else:
        return self._qList.pop(0)
```

Kode Lengkap Implementasi ADT Queue dengan `list`

Berikut adalah kode lengkap dari class `Queue` yang disimpan dalam *module* bernama `listqueue.py`:

Module *`listqueue.py`*

```
# Implementasi ADT Queue menggunakan list
class Queue:

      # Constructor untuk membuat queue baru dengan menggunakan sebuah list
kosong.
    # Field 1 (_qList): sebuah list untuk menyimpan queue.
    def __init__(self):
        self._qList = list()

    # Method isEmpty() mengembalikan nilai True jika queue kosong atau
    # False jika queue memiliki data.
    def isEmpty(self):
        return len(self) == 0

    # Mengembalikan banyak elemen yang ada di dalam queue.
    def __len__(self):
        return len(self._qList)

    # Menambahkan elemen baru pada bagian belakang queue.
    def enqueue(self, data):
        self._qList.append(data)

    # Menghapus dan mengembalikan nilai bagian elemen depan queue. Jika
    # queue kosong, raise exception generik.
    def dequeue( self ):
        if self.isEmpty():
                raise Exception('Queue kosong. Tidak ada data yang dapat di-
dequeue.')
        else:
            return self._qList.pop(0)
```

Untuk menguji implementasi ADT Queue yang telah kita tulis, kita dapat menuliskan program berikut:

```
from listqueue import Queue

def main():
    # Buat objek Queue
    myQueue = Queue()

    # Uji apakah queue kosong
    if myQueue.isEmpty():
        print('Queue kosong.')
```

```
    else:
        print('Queue tidak kosong')

    # Meminta input dari pengguna
    input_data = 'Masukkan nilai integer (nilai negatif untuk mengakhiri): '
    # Input pengguna untuk memasukkan data ke queue
    nilai = int(input(input_data))
    while nilai > 0:
        myQueue.enqueue(nilai)
        nilai = int(input(input_data))

    # Tampilkan panjang queue
    print('Panjang queue: ', end='')
    print(len(myQueue))

    # Cetak isi queue
    print('Isi queue: ')
    while not myQueue.isEmpty():
        nilai = myQueue.dequeue()
        if not myQueue.isEmpty():
            print(nilai, end=' - ')
        else:
            print(nilai)
main()
```

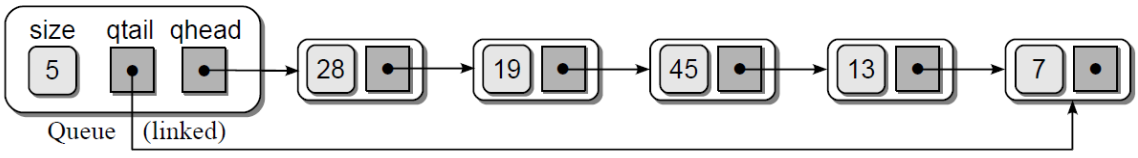
Contoh *output* dari program uji queue diatas adalah sebagai berikut:

```
Queue kosong.
Masukkan nilai integer (nilai negatif untuk mengakhiri): 54
Masukkan nilai integer (nilai negatif untuk mengakhiri): 23
Masukkan nilai integer (nilai negatif untuk mengakhiri): 86
Masukkan nilai integer (nilai negatif untuk mengakhiri): 22
Masukkan nilai integer (nilai negatif untuk mengakhiri): 98
Masukkan nilai integer (nilai negatif untuk mengakhiri): 71
Masukkan nilai integer (nilai negatif untuk mengakhiri): -1
Panjang queue: 6
Isi queue:
54 - 23 - 86 - 22 - 98 - 71
```

Dapat kita lihat pada *output*, di baris 'Isi queue: ' nilai 54 yang menjadi nilai paling depan pada queue (*front*) dan 71 yang menjadi nilai paling belakang pada queue (*back*).

5.3 Implementasi ADT Queue dengan Linked List

Kekurangan dari implementasi ADT Queue dengan `list` adalah pada operasi dari enqueue dan dequeue dibutukan reservasi ruang memori yang lebih besar dibandingkan kebutuhan. Ini tidak efisien. Solusi implementasi ADT Queue yang lebih baik adalah dengan menggunakan sebuah linked list yang mempunyai referensi *head* dan referensi *tail*. Menambahkan referensi *tail* memungkinkan operasi enqueue dapat dilakukan tanpa meng-*traverse* keseluruhan linked list terlebih dahulu untuk mendapatkan node terakhir. Berikut adalah gambar ilustrasi dari implementasi ADT Queue dengan linked List.



Class Node

Kita perlu mengimplementasikan class Node yang digunakan sebagai node-node dari elemen-elemen queue. Class node ini kita namanakan dengan `_QueueNode` dan diimplementasikan seperti berikut:

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

Class `Queue`

Berikut adalah kerangka dari class `Queue` beserta class `_QueueNode`:

```
class Queue:

 # .... Implementasi method-method

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

Constructor `Queue`

Class `Queue` mendefinisikan tiga buah *field* yaitu *field* `_head` yang digunakan untuk referensi ke elemen *front* (*head*), *field* `_tail` yang digunakan untuk referensi ke elemen *back* (*tail*), dan *field* `_count` yang digunakan sebagai variabel penghitung jumlah dari elemen yang ada di dalam queue. Berikut adalah definisi constructor `Queue`:

```
def __init__(self):
    self._head = None
    self._tail = None
    self._count = 0
```

Method `isEmpty()`

Method `isEmpty()` mengembalikan `True` jika queue kosong dan mengembalikan `False` jika tidak kosong. Kita hanya perlu mengembalikan sebuah nilai boolean yang menandakan apakah linked list yang digunakan untuk menyimpan elemen data queue mempunyai banyak elemen sama dengan 0. Kita dapat melakukannya dengan menggunakan ekspresi yang mencaritahu apakah *field* `_head` mereferensikan `None`. Jika ya, maka berarti queue kosong dan ekspresi ini mengembalikan `True`. Sebaliknya, jika *field* `_head` tidak mereferensikan `None` maka *field* ini mereferensikan suatu node yang berarti queue tidak kosong dan ekspresi ini mengembalikan `False`. Sehingga, implementasi *method* `isEmpty()` dapat dituliskan seperti berikut:

```
def isEmpty(self):
    return self._head is None
```

Method `len()`

Method `len()` mengembalikan nilai dari panjang elemen data yang ada pada linked list penyimpan struktur queue. Berikut adalah implementasi dari *method* `len()`:

```
def __len__(self):
    return self._count
```

Method `enqueue(data)`

Method `enqueue(data)` menambahkan elemen data baru pada queue. Langkah pertama dari proses penambahan data baru adalah dengan mebuat sebuah node baru berisi data baru tersebut. Lalu, kita harus memperhatikan dua kondisi: ketika queue kosong dan ketika queue tidak kosong. Jika kondisi queue kosong kita perlu menetapkan *field* `_head` untuk mereferensikan node baru ini, lalu menetapkan *field* `_tail` juga untuk mereferensikan node baru ini, kemudian menginkrementasi *field* `_count`. Jika kondisi queue tidak kosong, kita perlu mengubah *field* `next` dari node terakhir untuk mereferensikan ke node baru, lalu menetapkan *field* `_tail` juga untuk mereferensikan node baru ini, kemudian menginkrementasi *field* `_count`.

Implementasi *method* `enqueue(data)` dapat dituliskan seperti berikut:

```
def enqueue(self, data):
    newNode = _QueueNode(data)
    if self.isEmpty() :
        self._head = newNode
    else:
        self._tail.next = newNode
    self._tail = newNode
    self._count += 1
```

Method `dequeue()`

Method `dequeue()` menghapus dan mengembalikan nilai elemen paling depan (*front*) dari queue. Operasi penghapusan tidak dapat dilakukan pada queue kosong, sehingga kita perlu meng-*raise exception* generik dengan pesan queue kosong jika *method* ini dipanggil pada queue kosong. Jika queue tidak kosong, kita menghapus node pertama dalam queue dengan dengan mengubah *field* `_head` dari queue untuk mereferensikan node kedua lalu mendekrementasi *field* `_count`. Untuk mengembalikan nilai data node yang dihapus, sebelum mengubah *field* `_head` kita mereferensikan node paling dengan ke suatu variabel lalu setelah megubah *field* `_head`, kita mengembalikan nilai data pada node tersebut.

Implementasi *method* `dequeue()` dapat dituliskan seperti berikut:

```
def dequeue(self):
    if self.isEmpty():
        raise Exception('Queue kosong. Tidak ada data yang dapat di-dequeue.')
    else:
        node = self._head
        self._head = self._head.next
        self._count -= 1
        return node.data
```

Kode Lengkap Implementasi ADT Queue dengan Linked List

Berikut adalah kode lengkap dari ADT Queue dengan linked list yang disimpan dalam *module* bernama `linkedlistqueue.py`:

Module *`linkedlistqueue.py`*

```
# Implementasi ADT Queue menggunakan linked list
class Queue:

    # Constructor untuk membuat field pada queue baru.
    # Field 1: Mereferensikan elemen pertama.
    # Field 2: Mereferensikan elemen terakhir.
    # Field 3: Mereferensikan penghitung jumlah elemen queue.
    def __init__(self):
        self._head = None
        self._tail = None
        self._count = 0

    # Method isEmpty() mengembalikan nilai True jika queue kosong atau
    # False jika queue memiliki data.
    def isEmpty(self):
        return self._head is None

    # Mengembalikan banyak elemen yang ada di dalam queue.
    def __len__(self):
        return self._count

    # Menambahkan elemen baru pada bagian belakang queue.
    def enqueue(self, data):
        newNode = _QueueNode(data)
        if self.isEmpty() :
            self._head = newNode
        else:
            self._tail.next = newNode
        self._tail = newNode
        self._count += 1

    # Menghapus dan mengembalikan nilai bagian elemen depan queue. Jika
    # queue kosong, raise exception generik.
    def dequeue(self):
        if self.isEmpty():
                raise Exception('Queue kosong. Tidak ada data yang dapat di-
dequeue.')
        else:
            node = self._head
            self._head = self._head.next
            self._count -= 1
```

```
            return node.data

# Class private untuk menyimpan node linked list dari elemen-elemen pada queue
class _QueueNode():
    def __init__(self, data):
        self.data = data
        self.next = None
```

Untuk menguji implementasi ADT Queue yang telah kita tulis, kita dapat menuliskan program berikut:

```
from linkedlistqueue import Queue

def main():
    # Buat objek Queue
    myQueue = Queue()

    # Uji apakah queue kosong
    if myQueue.isEmpty():
        print('Queue kosong.')
    else:
        print('Queue tidak kosong. Isi: ', len(myQueue))

    # Meminta input dari pengguna
    input_data = 'Masukkan nilai integer (nilai negatif untuk mengakhiri): '
    # Input pengguna untuk memasukkan data ke queue
    nilai = int(input(input_data))
    while nilai > 0:
        myQueue.enqueue(nilai)
        nilai = int(input(input_data))

    # Tampilkan panjang queue
    print('Panjang queue: ', end='')
    print(len(myQueue))
    print()

    # Cetak isi queue dengan menyisakan 3 elemen terakhir
    print('Isi queue: ')
    for i in range(0,len(myQueue)-3):
        nilai = myQueue.dequeue()
        if len(myQueue) > 3:
            print(nilai, end=' - ')
        else:
            print(nilai)

    # Uji apakah queue kosong
    if myQueue.isEmpty():
        print('Queue kosong.')
    else:
        print('Queue tidak kosong. Isi: ', len(myQueue))

    # Cetak isi queue
    print()
    print('Sisa isi queue: ')
    while not myQueue.isEmpty():
        nilai = myQueue.dequeue()
        if not myQueue.isEmpty():
```

```
            print(nilai, end=' - ')
        else:
            print(nilai)

    # Uji apakah queue kosong
    if myQueue.isEmpty():
        print('Queue kosong.')
    else:
        print('Queue tidak kosong. Isi: ', len(myQueue))

main()
```

Contoh *output* dari program uji queue diatas adalah sebagai berikut:

```
Queue kosong.
Masukkan nilai integer (nilai negatif untuk mengakhiri): 13
Masukkan nilai integer (nilai negatif untuk mengakhiri): 24
Masukkan nilai integer (nilai negatif untuk mengakhiri): 35
Masukkan nilai integer (nilai negatif untuk mengakhiri): 8
Masukkan nilai integer (nilai negatif untuk mengakhiri): 77
Masukkan nilai integer (nilai negatif untuk mengakhiri): 54
Masukkan nilai integer (nilai negatif untuk mengakhiri): 3
Masukkan nilai integer (nilai negatif untuk mengakhiri): -4
Panjang queue: 7

Isi queue:
13 - 24 - 35 - 8
Queue tidak kosong. Isi:  3

Sisa isi queue:
77 - 54 - 3
Queue kosong.
```

REFERENSI

[1] Necaise, Rance D. 2011. Data structures and algorithms using Python .