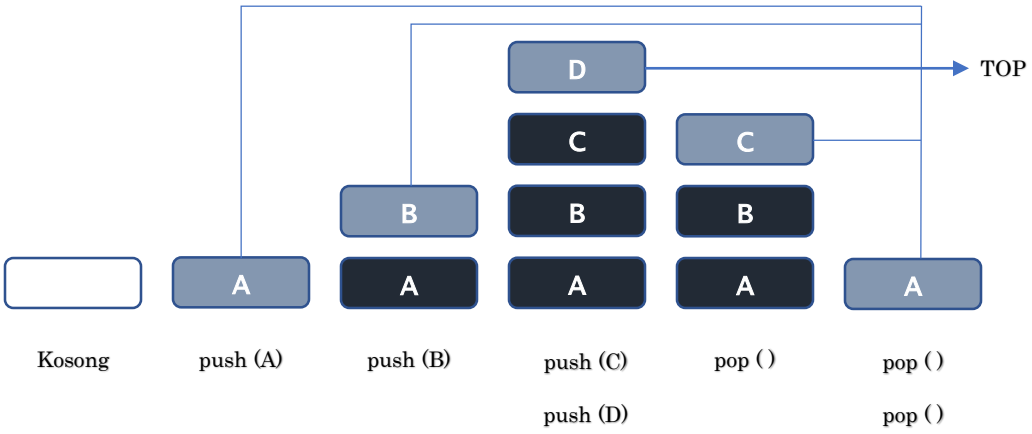
Bab 4. Stack & Queue

4.1 Tujuan Pembelajaran

- Mahasiswa mampu memahami konsep dasar *Stack & Queue*
- Mahasiswa mampu menganalisis permasalahan menggunakan konsep *Stack & Queue*
- Mahasiswa mampu membuat program penyelesaian masalah menggunakan konsep *Stack & Queue*

4.2 Stack

Stack adalah struktur data linier yang menggunakan konsep LIFO (Last In First Out), dimana data yang dimasukkan akan diletakkan diatas data yang sudah ada. Untuk operasinya, akses data pada stack hanya terdapat pada satu point yang disebut dengan top. Data baru yang dimasukkan akan otomatis menumpuk data yang sudah ada sebelumnya dan posisinya akan menjadi top. Sebaliknya, ketika data pada pop diambil maka posisi top akan digantikan dengan data dibawah top sebelumnya.



Gambar 4.1 Ilustrasi struktur data stack

Pada ilustrasi stack gambar 4.1, operasi stack yang digunakan bisa dijabarkan pada tabel berikut :

Operasi	Pre	Post	Return Value	Ket
Inisialisasi	Stack kosong	Stack kosong		Inisialisasi awal stack
s.push (A)	Stack kosong	A		Top = A
s.push (B)	A	A, B		Top = B
s.push (C)	A, B	A, B, C		Top = C
s.peek	A, B, C	A, B, C	C	Menampilkan nilai Top
s.push (D)	A, B, C	A, B, C, D		Top = D
len(s)	A, B, C, D	A, B, C, D	3	Menampilkan jumlah data stack
s.pop ( )	A, B, C, D	A, B, C		Top = C
s.isEmpty ( )	A, B, C	A, B, C	False	Stack tidak kosong
s.pop ( )	A, B, C	A, B		Top = B
s.pop ( )	A, B	A		Top = A

Tabel 4.1 Proses operasi pada stack

Selain operasi push (menambah) dan pop (mengambil) data yang digunakan pada stack, ada beberapa operasi lainnya pada penggunaan stack, yaitu :

Operasi	Fungsi
s.push ( )	menambah data pada top stack
s.pop ( )	mengambil data top pada stack
s.peek ( )	menampilkan data top pada stack (stack tidak boleh kosong)
s.isEmpty ( )	memeriksa stack dalam keadaan kosong (True), atau (False) jika tidak kosong
s.__len__ ( )	menghitung jumlah data pada stack

Tabel 4.2 Operasi lain pada stack

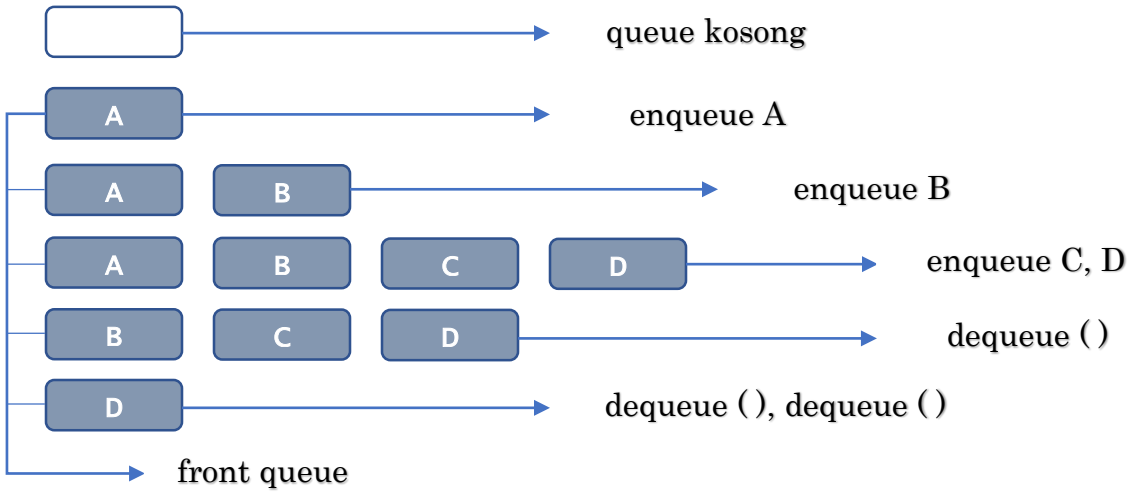
4.3 Implementasi Stack Pada Python

```
class Stack:
    def __init__(self):
        self.items = []

    def pop(self):
        if self.isEmpty():
            raise RuntimeError("Attempt to pop an empty stack")

        topIdx = len(self.items)-1
        item = self.items[topIdx]
        del self.items[topIdx]
        return item

    def push(self,item):
        self.items.append(item)

    def top(self):
        if self.isEmpty():
            raise RuntimeError("Attempt to get top of empty stack")
        topIdx = len(self.items)-1
        return self.items[topIdx]

    def isEmpty(self):
        return len(self.items) == 0

def main():
    s = Stack()
    lst = list(range(10))
    lst2 = []
    for k in lst:
        s.push(k)
    if s.top() == 9:
            print("Test 1 Passed")
    else:
            print("Test 1 Failed")

    while not s.isEmpty():
            lst2.append(s.pop())

    lst2.reverse()
    if lst2 != lst:
        print("Test 2 Failed")
    else:
        print("Test 2 Passed")

    try:
        s.pop()
        print("Test 3 Failed")

    except RuntimeError:
        print("Test 3 Passed")
    except:
```

```
        print("Test 3 Failed")

    try:
        s.top()
        print("Test 4 Failed")

    except RuntimeError:
        print("Test 4 Passed")
    except:
        print("Test 4 Failed")

if __name__=="__main__":
    main()
```

4.4 Queue

Hampir sama seperti stack, Queue merupakan struktur data linier yang yang juga dibatasi pada satu point dalam operasi akses nya. Perbedaanyya yaitu queue menggunakan konsep FIFO (First In First Out), yaitu data yang dimasukkan (enqueue) akan diletakkan pada posisi paling belakang (rear), dan data yang diambil (dequeue) adalah data pada posisi (front) atau yang paling awal pada queue.



Gambar 4.2 Ilustrasi struktur data Queue

Pada ilustrasi queue gambar 4.1, operasi queue yang digunakan bisa dijabarkan pada tabel berikut :

Operasi	Pre	Post	Return Value	Ket
Inisialisasi	Queue kosong	Queue kosong		Inisialisasi awal queue
q.enqueue (A)	Queue kosong	A		front = A
q.enqueue (B)	A	A, B		front = A, rear = B
q.enqueue (C)	A, B	A, B, C		front = A, rear = C
q.peek	A, B, C	A, B, C	A	Menampilkan nilai rear
q.enqueue (D)	A, B, C	A, B, C, D		front = A, rear = D
len(q)	A, B, C, D	A, B, C, D	4	Menampilkan jumlah data queue
q.dequeue ( )	A, B, C, D	B, C, D		front = B, rear = D
q.isEmpty ( )	B, C, D	B, C, D	False	queue tidak kosong
q.dequeue ( )	B, C, D	C, D		front = C, rear = D
q.dequeue ( )	C, D	D		front = D

Tabel 4.3 Proses operasi pada queue

Selain operasi enqueue (menambah) dan dequeue (mengambil) data yang digunakan pada queue, ada beberapa operasi lainnya pada penggunaan queue, yaitu :

Operasi	Fungsi
q.enqueue ( )	menambah data pada belakang (rear) queue
q.dequeue ( )	mengambil data depan (front) queue dan mengubah [front+1] menjadi front
q.peek ( )	menampilkan data front pada queue (queue tidak boleh kosong)
q.isEmpty ( )	memeriksa queue dalam keadaan kosong (True), atau (False) jika tidak kosong
q.__len__ ( )	menghitung jumlah data pada queue

Tabel 4.4 Proses operasi pada queue

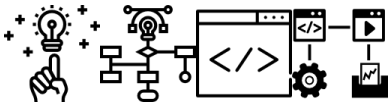
4.5 Implementasi Queue Pada Python

```
class Queue:
    def __init__(self):
        self.items = []
        self.frontIdx = 0

    def __compress(self):
        newlst = []
        for i in range(self.frontIdx,len(self.items)):
            newlst.append(self.items[i])

        self.items = newlst
        self.frontIdx = 0

    def dequeue(self):
        if self.isEmpty():
            raise RuntimeError("Attempt to dequeue an empty queue")

        # When queue is half full, compress it. This
        # achieves an amortized complexity of O(1) while
        # not letting the list continue to grow unchecked.
        if self.frontIdx * 2 > len(self.items):
            self.__compress()

        item = self.items[self.frontIdx]
        self.frontIdx += 1
        return item

    def enqueue(self,item):
        self.items.append(item)

    def front(self):
        if self.isEmpty():
            raise RuntimeError("Attempt to access front of empty queue")

        return self.items[self.frontIdx]

    def isEmpty(self):
        return self.frontIdx == len(self.items)

def main():
    q = Queue()
    lst = list(range(10))
    lst2 = []

    for k in lst:
        q.enqueue(k)

    if q.front() == 0:
        print("Test 1 Passed")
    else:
        print("Test 1 Failed")
```

```
    while not q.isEmpty():
        lst2.append(q.dequeue())

    if lst2 != lst:
        print("Test 2 Failed")
    else:
        print("Test 2 Passed")

    for k in lst:
        q.enqueue(k)

    lst2 = []

    while not q.isEmpty():
        lst2.append(q.dequeue())

    if lst2 != lst:
        print("Test 3 Failed")
    else:
        print("Test 3 Passed")

    try:
        q.dequeue()
        print("Test 4 Failed")

    except RuntimeError:
        print("Test 4 Passed")
    except:
        print("Test 4 Failed")

    try:
        q.front()
        print("Test 5 Failed")

    except RuntimeError:
        print("Test 5 Passed")
    except:
        print("Test 5 Failed")

if __name__=="__main__":
    main()
```

4.6 Latihan

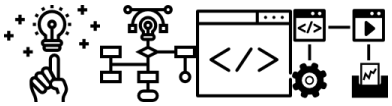
1. Pada program implementasi stack dan queue diatas, coba jalankan dan jelaskan program tersebut beserta fungsi-fungsi nya.



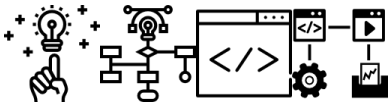
2. Buatlah program yang memiliki fungsi untuk membalik kata atau kalimat.

   Input : informatika
   Output : akitamrofni

4.7 Studi Kasus



1. Buatlah program yang memiliki fungsi mengubah bilangan decimal menjadi bilangan biner.



2. Buatlah program penyimpanan data dengan menggunakan konsep stack/queue yang memiliki fungsi paling tidak : menambah, menghapus, dan menampilkan data. tulis juga deskripsi program yang kalian buat.