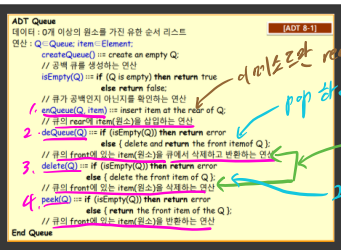
- Queue ADT



ADT Queue (ADT 8-1)

데이터 : 0개 이상의 원소를 가진 유한 순서 리스트

연산 : Q∈Queue; item∈Element;

createQueue() ::= create an empty Q;
// 공백 큐를 생성하는 연산

isEmpty(Q) ::= if (Q is empty) then return true
else return false;
// 큐가 공백인지 아닌지를 확인하는 연산

1. enQueue(Q, item) ::= insert item at the rear of Q;
// 큐의 rear에 item(원소)을 삽입하는 연산

2. deQueue(Q) ::= if (isEmpty(Q)) then return error
else { delete and return the front itemof Q };
// 큐의 front에 있는 item(원소)을 큐에서 삭제하고 반환하는 연산

3. delete(Q) ::= if (isEmpty(Q)) then return error
else { delete the front item of Q };
// 큐의 front에 있는 item(원소)을 삭제하는 연산

4. peek(Q) ::= if (isEmpty(Q)) then return error
else { return the front item of the Q };
// 큐의 front에 있는 item(원소)을 반환하는 연산

End Queue

어떻게든 rear과 관련있을.

맨 앞은 삭제

둘이 다르다!!!

그냥삭제

```
class Queue:
    def __init__(self):
        self.q = []

    def enQueue(self, item):
        self.q.append(item)

    def deQueue(self):
        if self.isEmpty() == False:
            return self.q.pop(0)

    def size(self):
        return len(self.q)

    def isEmpty(self):
        if len(self.q) > 0:
            return False
        else:
            return True

    def peek(self):
        if self.isEmpty() == False:
            return self.q[0]

    def delete(self, item):
        if item in self.q:
            self.q.remove(item)
        else:
            print("해당 아이템이 존재하지 않습니다.")

q = Queue()
q.enQueue('a')
q.enQueue('b')
q.enQueue('c')

q.delete('d')

print(q.deQueue())
print(q.deQueue())
print(q.deQueue())
print(q.deQueue())
```

Queue도 stack과 마찬가지로 list를 활용하여 구현한다.

일반적인 'append' 메소드와 같은 맥락이다.

stack의 pop 메소드와 비슷하다. (deQueue 메소드는 맨 앞의 원소를 pop한다.)

Queue의 peek 메소드는 맨 앞의 원소를 반환한다.

list의 remove 메소드를 사용함.

stack 메소드와 차이점을 보이는 부분.