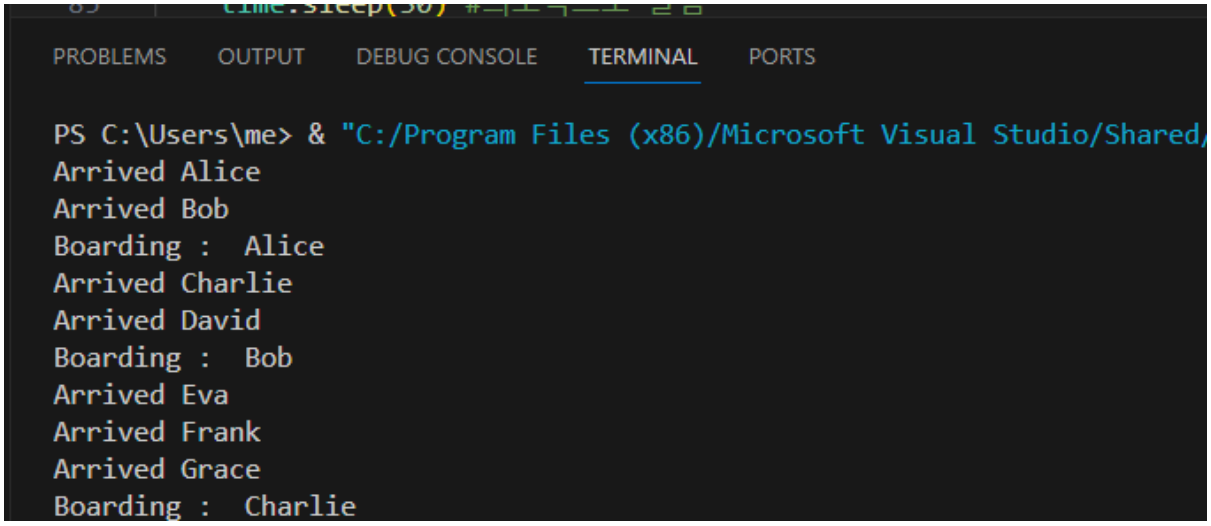
2. 아래의 코드를 실행했을 때 왼쪽과 같은 결과가 나오는데, 오른쪽과 같은 결과가 나오도록 Queue 를 사용하여 데이터를 전달하시오.

소스 코드

```
from listQueue import ListQueue
import threading
import time
import queue

class Producer:
    def __init__(self, items):
        self.__alive = True
        self.items = items
        self.pos = 0
        self.worker = threading.Thread(target=self.run)

    def get_item(self):
        if self.pos < len(self.items):
            item = self.items[self.pos]
            self.pos += 1
            return item
        else:
            return None

    def run(self):
        while True:
            time.sleep(0.2)
            if self.__alive:
                item = self.get_item()
                if item == None:
                    continue
                print("Arrived", item[1])
                globalQueue.put(item)
            else:
                break

    def start(self):
        self.worker.start()

    def finish(self):
        self.__alive = False
        self.worker.join()

class Consumer:
    def __init__(self):
```

```
        self.__alive = True
        self.worker = threading.Thread(target=self.run)

    def run(self):
        while True:
            time.sleep(0.5)
            if self.__alive:
                if not globalQueue.empty():
                    print("Boarding : ", globalQueue.get()[1])

            else:
                break
        print("Consumer is dying.")
    def start(self):
        self.worker.start()

    def finish(self):
        self.__alive = False
        self.worker.join()

if __name__ == "__main__":

    customers = []
    with open("H:/ds_2024-main/producer_consumer/customer.txt", 'r') as file:
        lines = file.readlines()
        for line in lines:
            customer = line.split()
            customers.append(customer)

    # FIFO
    names = queue.Queue()
    globalQueue = queue.Queue()
    #names = []
    for c in customers:
        names.put(c[1])

    producer = Producer(names)

    # Priority
    producer = Producer(customers)
    consumer = Consumer()
    producer.start()
    consumer.start()
    time.sleep(30) #의도적으로 늘림
    producer.finish()
    consumer.finish()
```

실행화면



간단한 설명

여러 쓰레드가 참조 할 수 있는 공용 큐를 두고, 생산자는 원소를 생산하고 소모자는 원소를 소모하도록 했습니다.

3. customers 리스트에는 고객의 등급 정보가 함께 포함되어 있다 (일반: 1, 골드: 2, 플래티넘: 3). 아래의 A 항공사의 규칙대로 탑승을 하도록 Producer 와 Consumer 의 코드를 수정하시오.

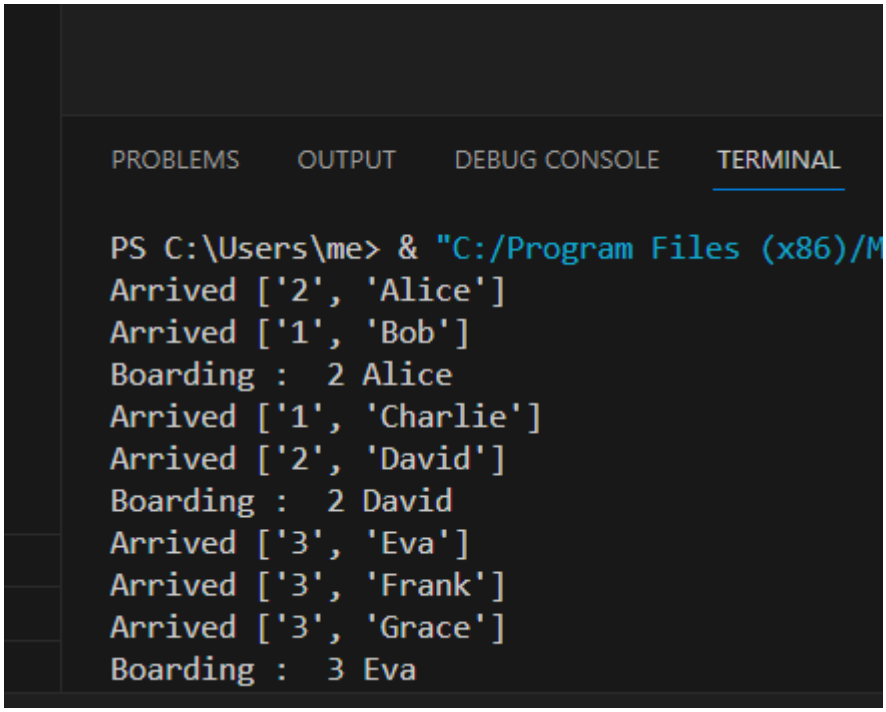
소스코드

```
from listQueue import ListQueue
import threading
import time
import queue

class Producer:
    def __init__(self, items):
        self.__alive = True
        self.items = items
        self.pos = 0
        self.worker = threading.Thread(target=self.run) #쓰레드 생성(?) 아마도

    def get_item(self):
        if self.pos < len(self.items):
            item = self.items[self.pos]
            self.pos += 1
            return item
        else:
```

```
            return None

    def run(self):
        while True:
            time.sleep(0.2)
            if self.__alive:
                item = self.get_item()
                if item == None:
                    continue
                print("Arrived", item) #버그 다발 지역(왜인지는 몰루? 내 직감이 이
부분을 말함)
                #print("Arrived", item[1])
                globalQueue.put(((-(int)(item[0])), item[1]))

            else:
                break

    def start(self):
        self.worker.start()

    def finish(self):
        self.__alive = False
        self.worker.join() #join

class Consumer:
    def __init__(self):
        self.__alive = True
        self.worker = threading.Thread(target=self.run)

    def run(self):
        while True:
            time.sleep(0.5) #디버깅에 용이하도록 의도적으로 늘림 + 크래쉬
걱정때문에 늘림
            if self.__alive:
                if not globalQueue.empty():
                    item =globalQueue.get()
                    print("Boarding : ", -(int)(item[0]) , item[1])
                    #print("Boarding :", item[1]) #버그 조심!


            else:
                break
        print("Consumer is dying.")
    def start(self):
        self.worker.start()

    def finish(self):
        self.__alive = False
```

```
        self.worker.join()

if __name__ == "__main__":

    customers = []
    with open("H:/ds_2024-main/producer_consumer/customer.txt", 'r') as
file:  #경로조심
        lines = file.readlines()
        for line in lines:
            customer = line.split()
            customers.append(customer)

    # FIFO
    customerList = queue.Queue()
    globalQueue = queue.PriorityQueue() #여러 쓰레드가 참조할 글로벌 큐 선언 ->
내부 구조가 뭐지? RW lock 인가? 아니면 LockFreeQueue? 뭐지?
                                        #사실상 무조건 크래쉬 터저야 정상인데
파이썬이라 상관 없는건가?
    #names = []
    for c in customers:
        customerList.put(c)

    producer = Producer(customerList)

    # Priority
    producer = Producer(customers)
    consumer = Consumer()
    producer.start()
    consumer.start()
    time.sleep(30) #의도적으로 늘림
    producer.finish()
    consumer.finish()
```

실행결과



간단한 설명

공용 큐를 우선순위 큐로 바꿔주었습니다.

파이썬의 우선순위 큐 내부 정렬 규칙을 맘대로 바꿀 수 없어서 우선순위가 큰 값이 먼저 나오도록 해주기 위해서 큐에 입력을 [-3, Eva] 와 같은 방식으로 입력하여 가장 작게 만들어서 먼저 나오도록 해주 었습니다. 복잡한 기교를 할 필요 없이 간단하게 구현했습니다.