데이터 구조 입문

김종현

04

스택과 큐

04-1 스택이란?

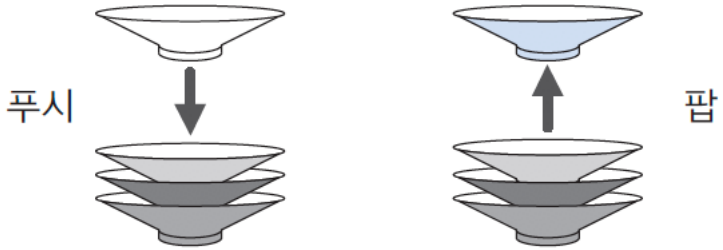
04-2 큐란?

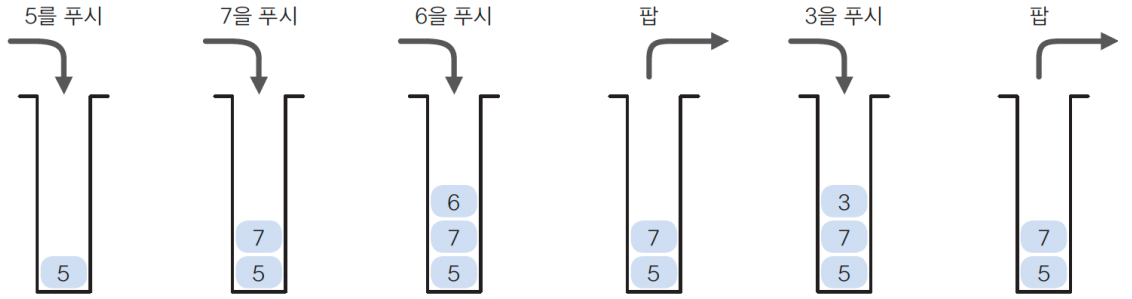
04-1 스택이란?

스택 알아보기

- **스택**stack
  - 데이터를 임시 저장할 때 사용하는 자료구조
  - 데이터 입력과 출력 순서는 후입선출LIFO 방식

  - **푸시**push: 스택에 데이터를 넣는 작업
  - **팝**pop: 스택에서 데이터를 꺼내는 작업
  - **꼭대기**top: 푸시하고 팝하는 윗부분
  - **바닥**bottom: 푸시하고 팝하는 아랫부분

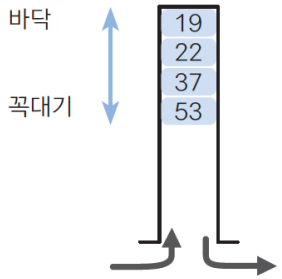


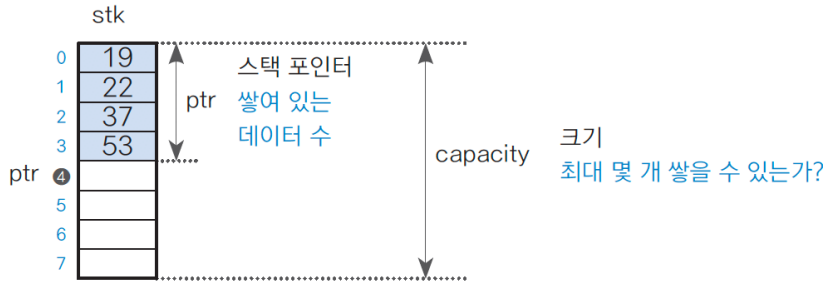


04-1 스택이란?

스택 구현하기 – (1)

- **스택 배열: stk**
  - 푸시한 데이터를 저장하는 스택 본체인 list형 배열
  - 인덱스가 0인 원소를 스택의 바닥이라고 함
  - 가장 먼저 푸시하여 데이터를 저장하는 곳은 stk[0]
- **스택 크기: capacity**
  - 스택의 최대 크기를 나타내는 int형 정수
  - 이 값은 배열 stk의 원소 수인 len(stk)와 일치
- **스택 포인터: ptr**
  - **스택 포인터**stack pointer: 스택에 쌓여 있는 데이터의 개수를 나타내는 정숫값
  - 스택이 비어 있으면 ptr의 값은 0이 되고, 가득 차 있으면 capacity와 같은 값





04-1 스택이란?

스택 구현하기 – (2)

- **실습 4-1 [A]**
  - **Empty**: 예외 처리 클래스
    - pop( ) 함수 또는 peek( ) 함수를 호출할 때 스택이 비어 있으면 내보내는 예외 처리
  - **Full**: 예외 처리 클래스
    - push( ) 함수를 호출할 때 스택이 가득 차 있으면 내보내는 예외 처리
  - **__init__( )**: 초기화하는 함수
    - 스택 배열을 생성하는 등의 준비 작업을 수행
    - 매개변수 capacity로 전달 받아 원소 수가 capacity이고 모든 원소가 None인 리스트형 stk를 생성
    - 스택이 비어 있으므로 스택 포인터 ptr는 0

Do it! 실습 4-1 [A] • 완성 파일 chap04/fixed_stack.py

```
01: # 고정 길이 스택 클래스 FixedStack 구현하기
02:
03: from typing import Any
04:
05: class FixedStack:
06:     """고정 길이 스택 클래스"""
07:
08:     class Empty(Exception):
09:         """비어 있는 FixedStack에 팝 또는 피크할 때 내보내는 예외 처리"""
10:         pass
11:
12:     class Full(Exception):
13:         """가득 찬 FixedStack에 푸시할 때 내보내는 예외 처리"""
14:         pass
15:
16:     def __init__(self, capacity: int = 256) -> None:
17:         """스택 초기화"""
18:         self.stk = [None] * capacity    # 스택 본체
19:         self.capacity = capacity        # 스택의 크기
20:         self.ptr = 0                    # 스택 포인터
```

04-1 스택이란?

스택 구현하기 – (3)

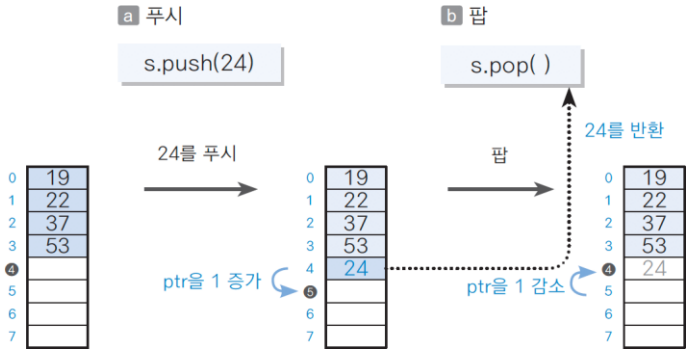
- **실습 4-1 [A]**
  - **__len__( )**: 쌓여 있는 데이터 개수를 알아내는 함수
    - 스택에 쌓여 있는 데이터 개수를 반환(ptr값 반환)
  - **is_empty( )**: 스택이 비어 있는지를 판단하는 함수
    - 데이터가 하나도 쌓여 있지 않은 상태,
      즉 스택이 비어 있는지 판단
    - 스택이 비어 있으면 True, 아니면 False 반환
  - **is_full( )**: 스택이 가득 차 있는지를 판단하는 함수
    - 더 이상 데이터를 푸시할 수 없는 상태,
      즉 스택이 가득 차 있는지 판단
    - 스택이 가득 차 있으면 True, 아니면 False 반환

```
22:     def __len__(self) -> int:
23:         """스택에 쌓여 있는 데이터 개수를 반환"""
24:         return self.ptr
25:
26:     def is_empty(self) -> bool:
27:         """스택이 비어 있는지 판단"""
28:         return self.ptr <= 0
29:
30:     def is_full(self) -> bool:
31:         """스택이 가득 차 있는지 판단"""
32:         return self.ptr >= self.capacity
```

04-1 스택이란?

스택 구현하기 – (4)

- **실습 4-1 [B]**
  - **push( )**: 데이터를 푸시하는 함수
    - 스택에 데이터를 추가
    - 스택이 가득 차서 더 이상 푸시할 수 없는 경우에는 FixedStack.Full을 통하여 예외 처리를 내보냄
  - **pop( )**: 데이터를 팝하는 함수
    - 스택의 꼭대기에서 데이터를 꺼내서 그 값을 반환
    - 스택이 비어서 팝할 수 없는 경우에는 FixedStack.Empty를 통하여 예외 처리를 내보냄



Do it! 실습 4-1 [B] • 완성 파일 chap04/fixed_stack.py

```
35:     def push(self, value: Any) -> None:
36:         """스택에 value를 푸시(데이터를 넣음)"""
37:         if self.is_full():          # 스택이 가득 차 있는 경우
38:             raise FixedStack.Full   # 예외 처리 발생
39:         self.stk[self.ptr] = value
40:         self.ptr += 1
41:
42:     def pop(self) -> Any:
43:         """스택에서 데이터를 팝(꼭대기 데이터를 꺼냄)"""
44:         if self.is_empty():         # 스택이 비어 있는 경우
45:             raise FixedStack.Empty  # 예외 처리 발생
46:         self.ptr -= 1
47:         return self.stk[self.ptr]
48:
49:     def peek(self) -> Any:
50:         """스택에서 데이터를 피크(꼭대기 데이터를 들여다봄)"""
51:         if self.is_empty():         # 스택이 비어 있음
52:             raise FixedStack.Empty  # 예외 처리 발생
53:         return self.stk[self.ptr - 1]
54:
55:     def clear(self) -> None:
56:         """스택을 비움(모든 데이터를 삭제)"""
57:         self.ptr = 0
```

04-1 스택이란?

스택 구현하기 – (5)

- **실습 4-1 [B]**
  - **peek( )**: 데이터를 들여다보는 함수
    - 스택의 꼭대기 데이터를 들여다 봄
    - 스택이 비어 있는 경우에는 FixedStack.Empty를 통하여 예외 처리를 내보냄
    - 데이터의 입출력이 없으므로 스택 포인터는 반환하지 않음
  - **clear( )**: 스택의 모든 데이터를 삭제하는 함수
    - 스택에 쌓여 있는 데이터를 모두 삭제하여 빈 스택을 만듦
    - 스택 포인터 ptr 값을 0으로 하면 끝!

Do it! 실습 4-1 [B]

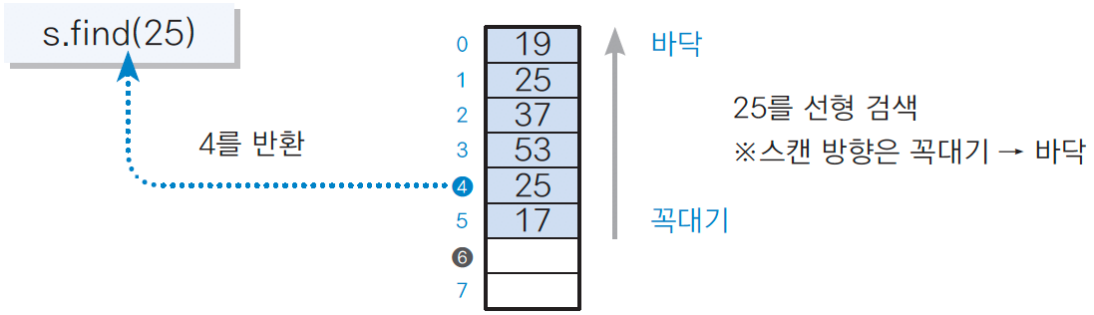
• 완성 파일 chap04/fixed_stack.py

```
35:     def push(self, value: Any) -> None:
36:         """스택에 value를 푸시(데이터를 넣음)"""
37:         if self.is_full():          # 스택이 가득 차 있는 경우
38:             raise FixedStack.Full   # 예외 처리 발생
39:         self.stk[self.ptr] = value
40:         self.ptr += 1
41:
42:     def pop(self) -> Any:
43:         """스택에서 데이터를 팝(꼭대기 데이터를 꺼냄)"""
44:         if self.is_empty():         # 스택이 비어 있는 경우
45:             raise FixedStack.Empty  # 예외 처리 발생
46:         self.ptr -= 1
47:         return self.stk[self.ptr]
48:
49:     def peek(self) -> Any:
50:         """스택에서 데이터를 피크(꼭대기 데이터를 들여다봄)"""
51:         if self.is_empty():         # 스택이 비어 있음
52:             raise FixedStack.Empty  # 예외 처리 발생
53:         return self.stk[self.ptr - 1]
54:
55:     def clear(self) -> None:
56:         """스택을 비움(모든 데이터를 삭제)"""
57:         self.ptr = 0
```

04-1 스택이란?

스택 구현하기 – (6)

- **실습 4-1 [C]**
  - **find( )**: 데이터를 검색하는 함수
    - 스택 본체의 배열 stk 안에 value와 값이 같은 데이터가 포함되어 있는지 확인하고, 포함되어 있다면 배열의 어디에 들어 있는지 검색



Do it! 실습 4-1 [C]

• 완성 파일 chap04/fixed_stack.py

```
60:     def find(self, value: Any) -> Any:
61:         """스택에서 value를 찾아 인덱스를 반환(없으면 -1을 반환)"""
62:         for i in range(self.ptr - 1, -1, -1):  # 꼭대기 쪽부터 선형 검색
63:             if self.stk[i] == value:
64:                 return i             # 검색 성공
65:         return -1                    # 검색 실패
66:
67:     def count(self, value: Any) -> bool:
68:         """스택에 있는 value의 개수를 반환"""
69:         c = 0
70:         for i in range(self.ptr):    # 바닥 쪽부터 선형 검색
71:             if self.stk[i] == value: # 검색 성공
72:                 c += 1
73:         return c
74:
75:     def __contains__(self, value: Any) -> bool:
76:         """스택에 value가 있는지 판단"""
77:         return self.count(value)
78:
79:     def dump(self) -> None:
80:         """덤프(스택 안의 모든 데이터를 바닥부터 꼭대기 순으로 출력)"""
81:         if self.is_empty():          # 스택이 비어 있음
82:             print('스택이 비어 있습니다.')
83:         else:
84:             print(self.stk[:self.ptr])
```

04-1 스택이란?

스택 구현하기 – (7)

- **실습 4-1 [C]**
  - **count( )**: 데이터 개수를 세는 함수
    - 스택에 쌓여 있는 데이터(value)의 개수 반환
  - **__contains__( )**: 데이터가 포함되어 있는지 판단하는 함수
    - 스택에 데이터(value)가 있는지 판단
    - 있으면 True, 없으면 False를 반환
    - 예) 스택 s에 데이터 x가 포함되어 있는지 판단하기 s.__contains__(x)
  - **dump( )**: 스택의 모든 데이터를 출력하는 함수
    - 스택에 쌓여 있는 ptr개의 모든 데이터를 바닥부터 꼭대기까지 순서대로 출력
    - 스택이 비어 있으면 '스택이 비어 있습니다.'를 출력

Do it! 실습 4-1 [C] • 완성 파일 chap04/fixed_stack.py

```
60:     def find(self, value: Any) -> Any:
61:         """스택에서 value를 찾아 인덱스를 반환(없으면 -1을 반환)"""
62:         for i in range(self.ptr - 1, -1, -1):  # 꼭대기 쪽부터 선형 검색
63:             if self.stk[i] == value:
64:                 return i             # 검색 성공
65:         return -1                    # 검색 실패
66:
67:     def count(self, value: Any) -> bool:
68:         """스택에 있는 value의 개수를 반환"""
69:         c = 0
70:         for i in range(self.ptr):    # 바닥 쪽부터 선형 검색
71:             if self.stk[i] == value: # 검색 성공
72:                 c += 1
73:         return c
74:
75:     def __contains__(self, value: Any) -> bool:
76:         """스택에 value가 있는지 판단"""
77:         return self.count(value)
78:
79:     def dump(self) -> None:
80:         """덤프(스택 안의 모든 데이터를 바닥부터 꼭대기 순으로 출력)"""
81:         if self.is_empty():          # 스택이 비어 있음
82:             print('스택이 비어 있습니다.')
83:         else:
84:             print(self.stk[:self.ptr])
```

04-1 스택이란?

스택 프로그램 만들기

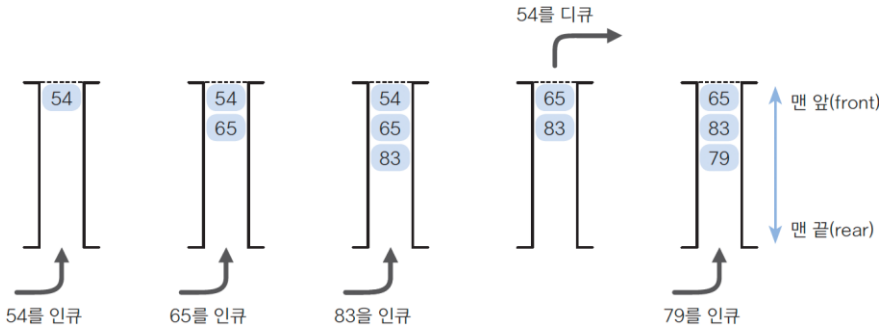
- **실습 4-2**
  - 고정 길이 스택 FixedStack 클래스를 사용한 프로그램
  - **chap04/fixed_stack_test.py 참조** (p.162)

04-2 큐란?

큐 알아보기

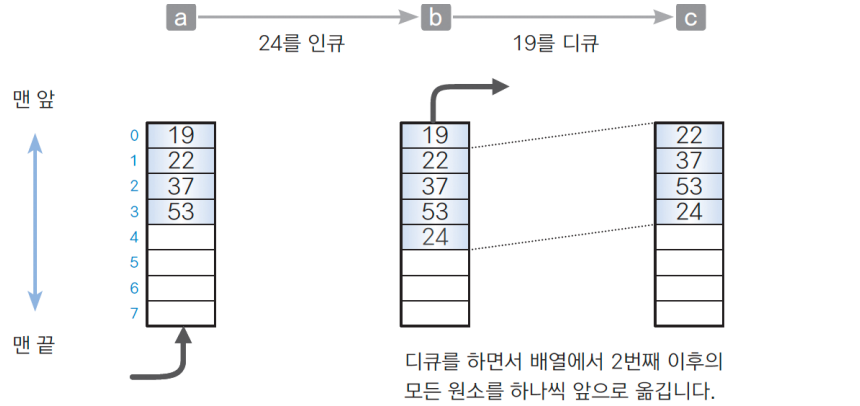
- **큐**queue
  - 스택과 같이 데이터를 임시 저장하는 자료구조
  - 가장 먼저 넣은 데이터를 가장 먼저 꺼내는 선입선출FIFO 구조

  - **인큐**enqueue: 큐에 데이터를 추가하는 작업
  - **디큐**dequeue: 큐에서 데이터를 꺼내는 작업
  - **프런트**front: 큐에서 데이터를 꺼내는 쪽
  - **리어**rear: 큐에 데이터를 넣는 쪽



배열로 큐 구현하기

- **배열로 큐를 구현한 예**
  - 24를 인큐하기
    - 맨 끝 데이터 que[3]의 다음 원소인 que[4]에 24를 저장
    - 처리 복잡도는 O(1)로, 비교적 적은 비용cost로 구현 가능
  - 19를 디큐하기
    - 19(que[0])를 꺼내고 뒤의 모든 원소를 앞으로 옮겨야 함
    - 처리 복잡도는 O(n)으로 데이터를 꺼낼 때마다 이런 처리 작업을 수행해야 한다면 프로그램의 효율성 ↓



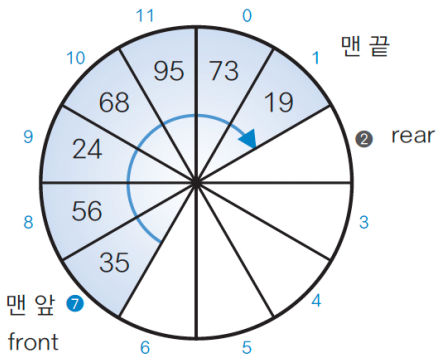
04-2 큐란?

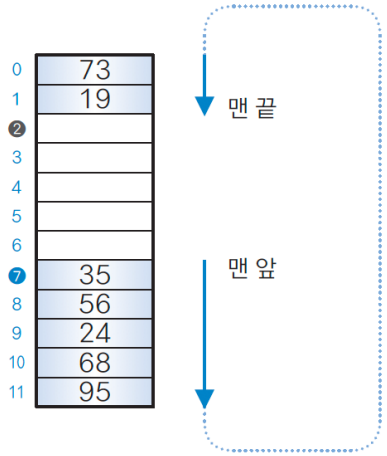
링 버퍼로 큐 구현하기 – (1)

- **링 버퍼**ring buffer
  - 배열 맨 끝의 원소 뒤에 맨 앞의 원소가 연결되는 자료구조
  - 링 버퍼로 큐를 구현하면 원소를 옮길 필요 없이
    front와 rear의 값을 업데이트 하는 것만으로 인큐와 디큐를 수행할 수 있음 → **모든 처리의 복잡도는 O(1)**

● 프런트(front): 맨 앞 원소의 인덱스

● 리어(rear): 맨 끝 원소 바로 뒤의 인덱스(다음 인큐되는 데이터가 저장되는 위치)

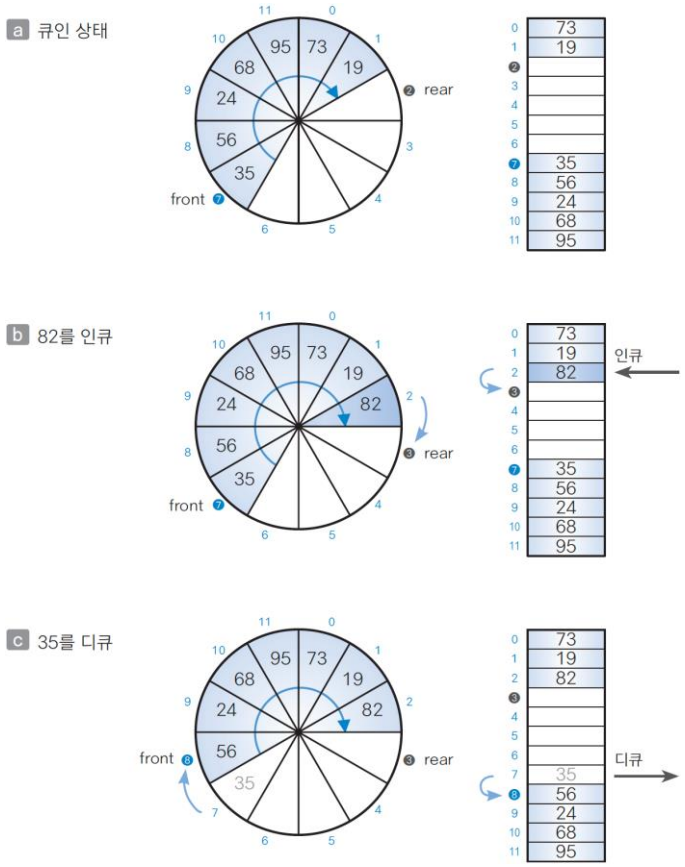




04-2 큐란?

링 버퍼로 큐 구현하기 – (2)

- **링 버퍼**ring buffer



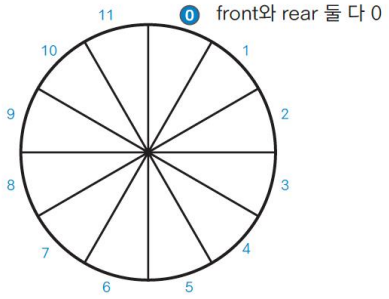
- a : 7개의 데이터 35, 56, 24, 68, 95, 73, 19가 늘어선 순서대로 que[7], que[8], …, que[11], que[0], que[1]에 저장됨. front 값은 7, rear 값은 2
- b : a 에서 82를 인큐한 다음의 상태
  맨 끝의 다음에 위치한 que[rear], 즉 que[2]에 82를 저장하고 rear값을 1 증가시켜 3으로 만듦
- c : b 에서 35를 디큐한 다음의 상태
  맨 앞 원소인 que[front], 즉 que[7]의 값인 35를 꺼내고 front값을 1 증가시켜 8로 만듦

04-2 큐란?

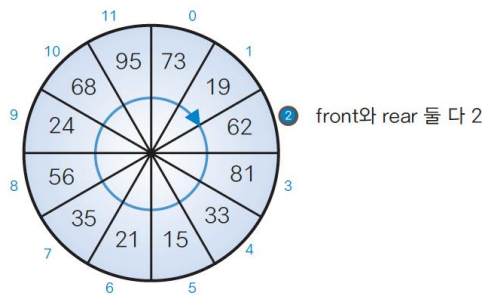
링 버퍼로 큐 구현하기 – (3)

- **실습 4-3 [A]**
  - **Empty**: 예외 처리 클래스
    - 비어 있는 큐에 deque( ), peek( ) 함수를 호출할 때 내보내는 예외 처리
  - **Full**: 예외 처리 클래스
    - 가득 차 있는 큐에 enque( ) 함수를 호출할 때 내보내는 예외 처리

a 비어 있는 큐(no = 0)



b 가득 차 있는 큐(no = 12)



Do it! 실습 4-3 [A]

• 완성 파일 chap04/fixed_queue.py

```
01: # 고정 길이 큐 클래스 FixedQueue 구현하기
02:
03: from typing import Any
04:
05: class FixedQueue:
06:
07:     class Empty(Exception):
08:         """비어 있는 FixedQueue에서 디큐 또는 피크할 때 내보내는 예외 처리"""
09:         pass
10:
11:     class Full(Exception):
12:         """가득 차 있는 FixedQueue에서 인큐할 때 내보내는 예외 처리"""
13:         pass
```

04-2 큐란?

링 버퍼로 큐 구현하기 – (4)

- **실습 4-3 [A]**
  - **_init_( )**: 초기화하는 함수
    - 큐 배열을 생성하는 등의 준비 작업
    - 설정하는 변수 5가지

- **que**: 큐의 배열(밀어 넣는 데이터를 저장하는 list형 배열)
- **capacity**: 큐의 최대 크기를 나타내는 int형 정수(que의 원소 수)
- **front, rear**: 맨 앞의 원소, 맨 끝의 원소를 나타내는 인덱스. rear는 다음에 인큐할 때 데이터를 저장하는 원소의 인덱스.
- **no**: 큐에 쌓여 있는 데이터 개수를 나타내는 int형 정수. 변수 front와 rear의 값이 같을 경우 큐가 비어 있는지 또는 가득 차 있는지 구별하기 위해 필요함. 큐가 비어 있는 경우에는 no가 0이 되고, 가득 차 있는 경우에는 capacity와 같은 값이 됨

```
15:     def __init__(self, capacity: int) -> None:
16:         """큐 초기화"""
17:         self.no = 0                     # 현재 데이터 개수
18:         self.front = 0                  # 맨 앞 원소 커서
19:         self.rear = 0                   # 맨 끝 원소 커서
20:         self.capacity = capacity        # 큐의 크기
21:         self.que = [None] * capacity    # 큐의 본체
22:
23:     def __len__(self) -> int:
24:         """큐에 있는 모든 데이터 개수를 반환"""
25:         return self.no
26:
27:     def is_empty(self) -> bool:
28:         """큐가 비어 있는지 판단"""
29:         return self.no <= 0
30:
31:     def is_full(self) -> bool:
32:         """큐가 가득 차 있는지 판단"""
33:         return self.no >= self.capacity
```

04-2 큐란?

링 버퍼로 큐 구현하기 – (5)

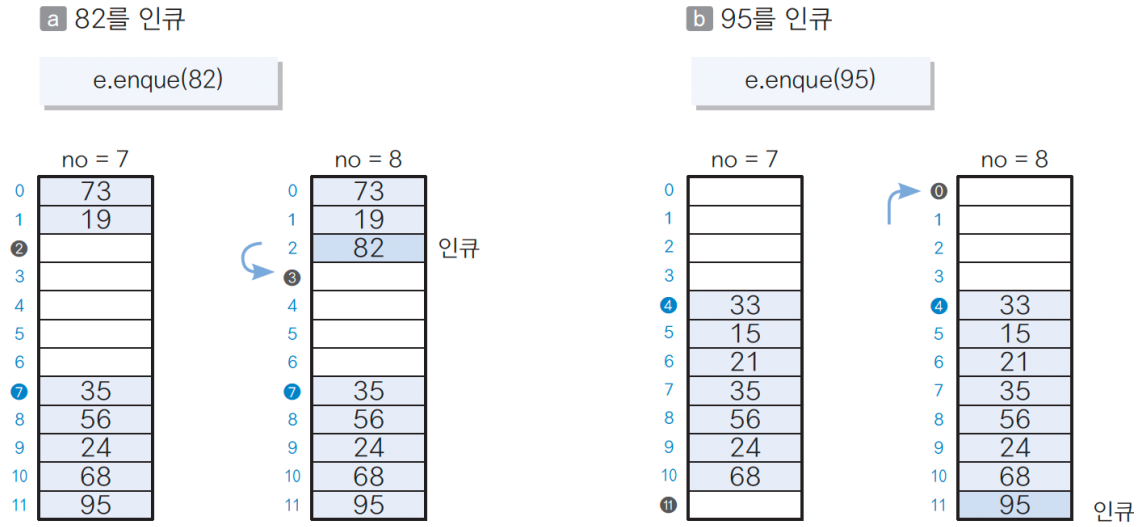
- **실습 4-3 [A]**
  - **__len__( )**: 추가한 데이터 개수를 알아내는 함수
    - 큐에 추가한 데이터 개수 반환
      (no의 값은 그대로 반환)
  - **is_empty( )**: 큐가 비어 있는지를 판단하는 함수
    - 큐가 비어있는지 판단
    - 비어 있으면 True, 그렇지 않으면 False 반환
  - **is_full( )**: 큐가 가득 차 있는지를 판단하는 함수
    - 큐가 가득 차 있어서 더 이상 데이터를 추가할 수 없는 상태인지 검사
    - 가득 차 있으면 True, 그렇지 않으면 False를 반환

```
15:     def __init__(self, capacity: int) -> None:
16:         """큐 초기화"""
17:         self.no = 0                     # 현재 데이터 개수
18:         self.front = 0                  # 맨 앞 원소 커서
19:         self.rear = 0                   # 맨 끝 원소 커서
20:         self.capacity = capacity        # 큐의 크기
21:         self.que = [None] * capacity    # 큐의 본체
22:
23:     def __len__(self) -> int:
24:         """큐에 있는 모든 데이터 개수를 반환"""
25:         return self.no
26:
27:     def is_empty(self) -> bool:
28:         """큐가 비어 있는지 판단"""
29:         return self.no <= 0
30:
31:     def is_full(self) -> bool:
32:         """큐가 가득 차 있는지 판단"""
33:         return self.no >= self.capacity
```

04-2 큐란?

링 버퍼로 큐 구현하기 – (6)

- **실습 4-3 [B]**
  - **enque( )**: 큐에 데이터를 인큐하는 함수
    - 큐가 가득 차서 인큐할 수 없는 경우 예외 처리인 FixedQueue.Full을 내보냄



Do it! 실습 4-3 [B]

• 완성 파일 chap04/fixed_queue.py

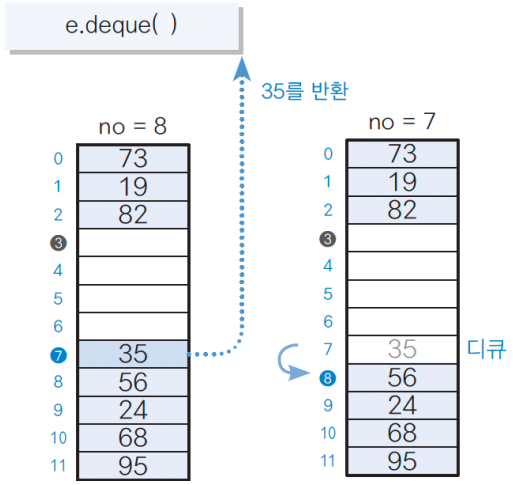
```
36:     def enque(self, x: Any) -> None:
37:         """데이터 x를 인큐"""
38:         if self.is_full():
39:             raise FixedQueue.Full  # 큐가 가득 차 있는 경우 예외 처리 발생
40:         self.que[self.rear] = x
41:         self.rear += 1
42:         self.no += 1
43:         if self.rear == self.capacity:
44:             self.rear = 0
```

04-2 큐란?

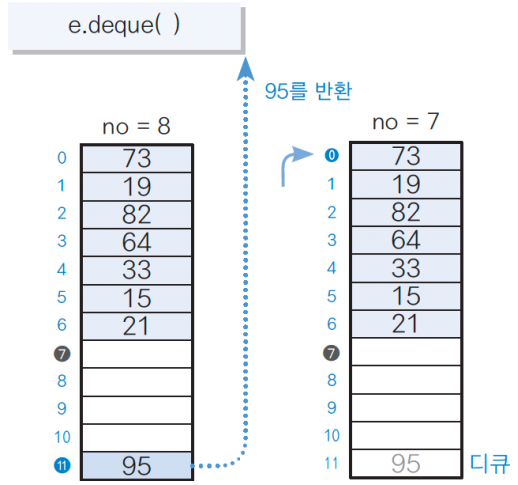
링 버퍼로 큐 구현하기 – (7)

- **실습 4-3 [C]**
  - **deque( )**: 큐의 맨 앞부터 데이터를 디큐하여 값을 반환하는 함수
    - 큐가 비어 있어 디큐할 수 없는 경우
      예외 처리인 FixedQueue.Empty를 내보냄

a 디큐



b 디큐



Do it! 실습 4-3 [C]

• 완성 파일 chap04/fixed_queue.py

```
47:     def deque(self) -> Any:
48:         """데이터를 디큐"""
49:         if self.is_empty():
50:             raise FixedQueue.Empty  # 큐가 비어 있는 경우 예외 처리 발생
51:         x = self.que[self.front]
52:         self.front += 1
53:         self.no -= 1
54:         if self.front == self.capacity:
55:             self.front = 0
56:         return x
```

04-2 큐란?

링 버퍼로 큐 구현하기 – (8)

- **실습 4-3 [D]**
  - **peek( )**: 데이터를 들여다보는 함수
    - 맨 앞 데이터, 즉 다음 디큐에서 꺼낼 데이터
    - 큐가 비어있을 때는 예외 처리 FixedQueue.Empty
  - **find( )**: 검색하는 함수
    - 큐의 배열에서 value와 같은 데이터가 포함되어 있는 위치를 알아냄
    - 큐의 맨 앞 원소(front)에서 맨 끝 쪽으로 선형 검색을 수행
    - 스캔할 때 주목하는 인덱스 idx를 구하는 식
      (i + front) % capacity

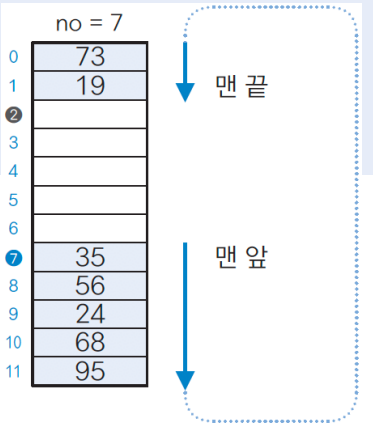
i	0 ⇨ 1 ⇨ 2 ⇨ 3 ⇨ 4 ⇨ 5 ⇨ 6
idx	7 ⇨ 8 ⇨ 9 ⇨ 10 ⇨ 11 ⇨ 0 ⇨ 1

    - 검색에 성공하면 찾은 원소의 인덱스를 반환하고, 실패하면 -1을 반환

Do it! 실습 4-3 [D]

• 완성 파일 chap04/fixed_queue.py

```
59:     def peek(self) -> Any:
60:         """큐에서 데이터를 피크(맨 앞 데이터를 들여다봄)"""
61:         if self.is_empty():
62:             raise FixedQueue.Empty        # 큐가 비어 있는 경우 예외 처리를 발생
63:         return self.que[self.front]
64:
65:     def find(self, value: Any) -> Any:
66:         """큐에서 value를 찾아 인덱스를 반환(없으면 -1을 반환)"""
67:         for i in range(self.no):
68:             idx = (i + self.front) % self.capacity
69:             if self.que[idx] == value:    # 검색 성공
70:                 return idx
71:         return -1                         # 검색 실패
```



04-2 큐란?

링 버퍼로 큐 구현하기 – (9)

- **실습 4-3 [D]**
  - **count( )**: 데이터 개수를 세는 함수
    - 큐에 있는 데이터(value)의 개수를 구하여 반환
  - **__contains__( )**: 데이터가 포함되어 있는지 판단하는 함수
    - 큐에 데이터(value)가 들어 있는지를 판단
    - 들어 있으면 True, 그렇지 않으면 False 반환
    - 내부의 count( ) 함수를 호출하여 구현
  - **clear( )**: 큐의 전체 원소를 삭제하는 함수
    - 현재 큐에 들어 있는 모든 데이터를 삭제
  - **dump( )**: 큐의 전체 데이터를 출력하는 함수
    - 큐에 들어 있는 모든 데이터를 맨 앞부터 맨 끝 쪽으로 순서대로 출력
    - 큐가 비어 있으면 '큐가 비어 있습니다.'를 출력

```
73:     def count(self, value: Any) -> bool:
74:         """큐에 있는 value의 개수를 반환"""
75:         c = 0
76:         for i in range(self.no):          # 큐 데이터를 선형 검색
77:             idx = (i + self.front) % self.capacity
78:             if self.que[idx] == value:    # 검색 성공
79:                 c += 1                    # 들어 있음
80:         return c
81:
82:     def __contains__(self, value: Any) -> bool:
83:         """큐에 value가 있는지 판단"""
84:         return self.count(value)
85:
86:     def clear(self) -> None:
87:         """큐의 모든 데이터를 비움"""
88:         self.no = self.front = self.rear = 0
89:
90:     def dump(self) -> None:
91:         """모든 데이터를 맨 앞부터 맨 끝 순으로 출력"""
92:         if self.is_empty():               # 큐가 비어 있음
93:             print('큐가 비었습니다.')
94:         else:
95:             for i in range(self.no):
96:                 print(self.que[(i + self.front) % self.capacity], end=' ')
97:             print()
```

04-2 큐란?

링 버퍼로 큐 프로그램 만들기

- **실습 4-4**
  - 큐 FixedQueue 클래스를 실제 사용하는 프로그램
  - **chap04/fixed_queue_test.py 참조** (p.178)