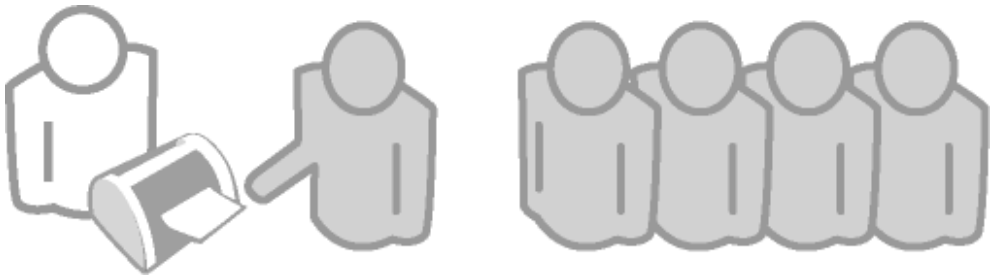
목차

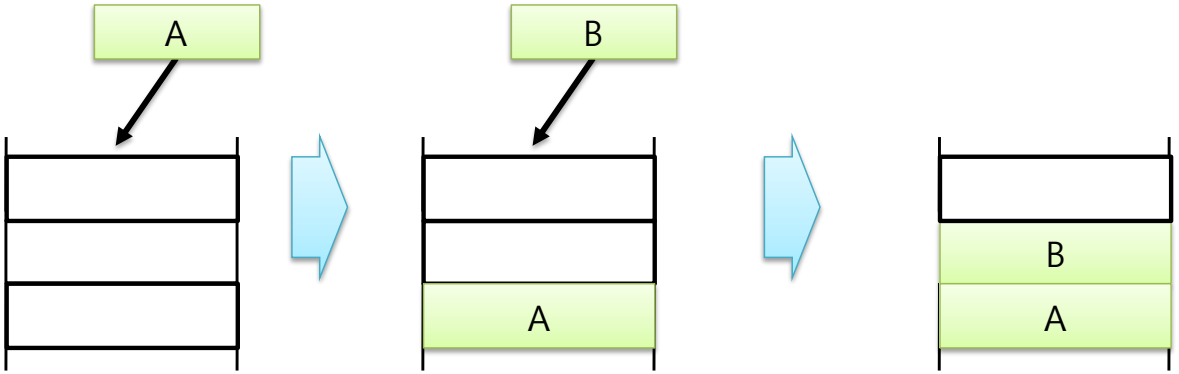
1. 큐의 개념 (1/3)

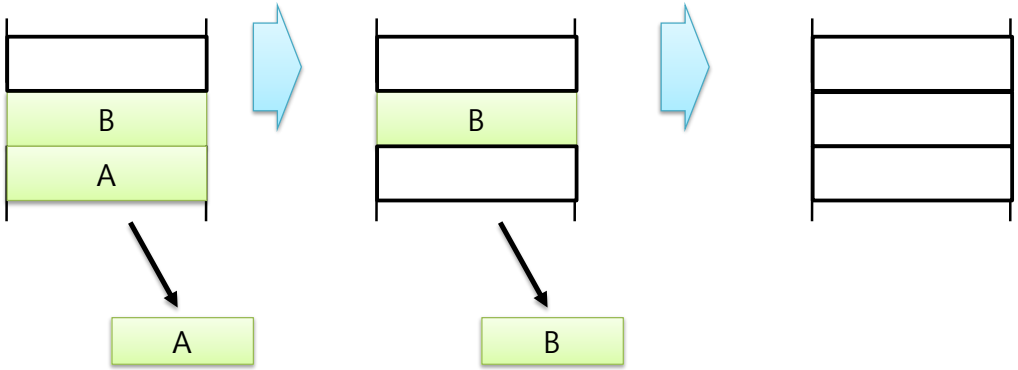
- 큐(Queue)
  - 일반적인 의미
    - 대기열
  - 자료구조에서의 의미
    - FIFO
  - FIFO(First-In-First-Out)
    - 선입선출 (先入先出)

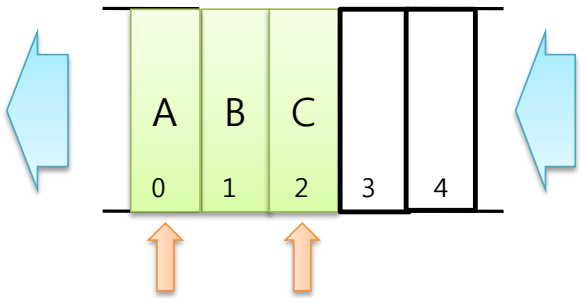
  - 사용 예
    - 선입선출의 특성이 있는 시스템
    - 다양한 알고리즘



1. 큐의 개념 (2/3)



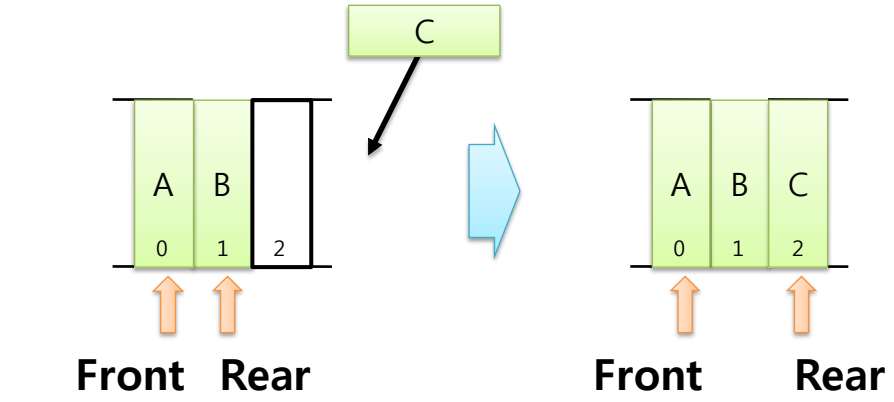


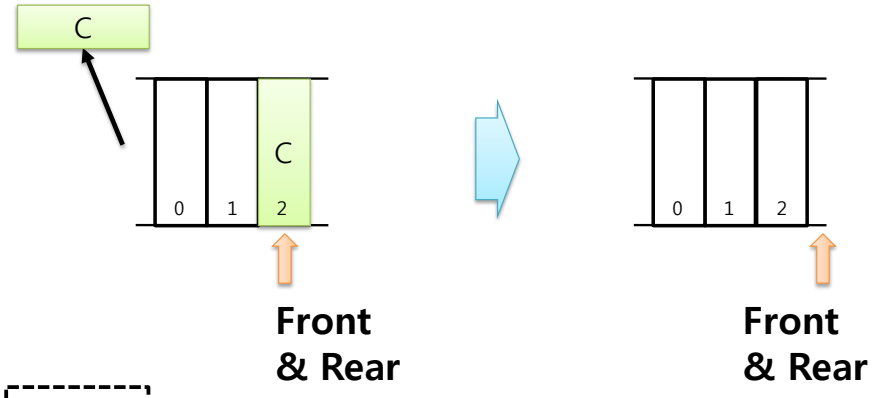


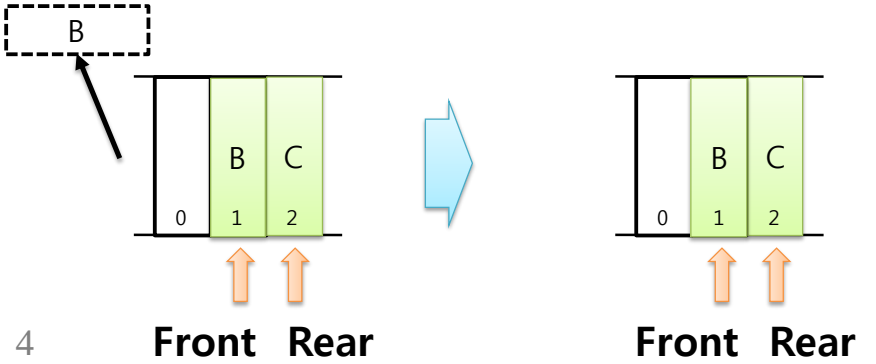
큐의 제일 앞(front, **프런트**)
큐의 제일 끝(rear, **리어**)

1. 큐의 개념 (3/3)

- 3가지 연산
  - 인큐(Enqueue, En-Queue)
    - 넘침(Overflow) 현상
  - 디큐(Dequeue, De-Queue)
    - 부족(Underflow) 현상
  - 피크(Peek)







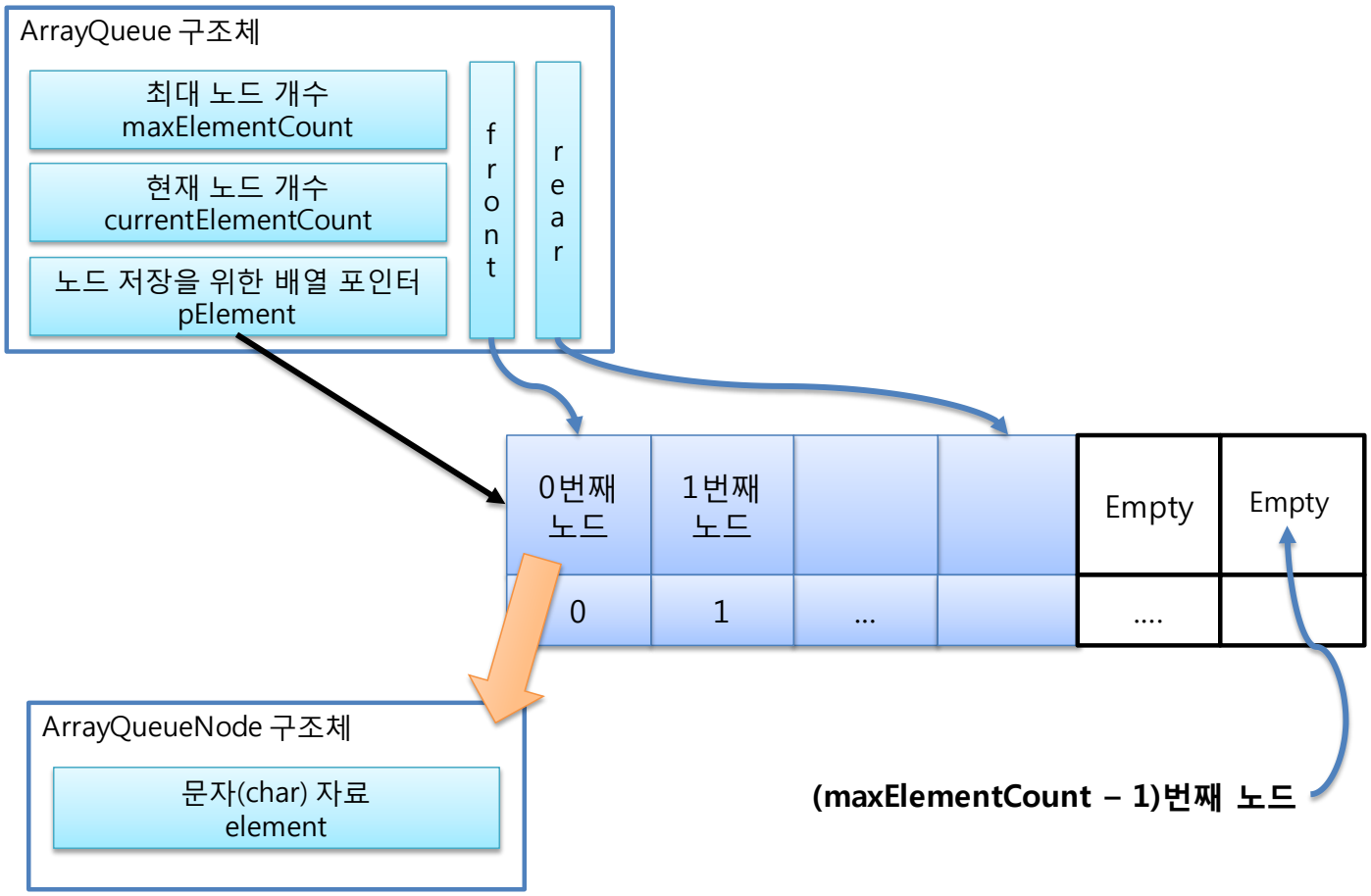
2. 큐 추상 자료형

- 큐 생성
  - 큐의 크기 n

- 인큐(Enqueue)
  - 원소 추가 가능 여부 판단
- 디큐(Dequeue)
  - 공백 큐인지 판단
- 피크(Peek)

- 큐 삭제

3. 배열로 구현한 선형 큐 (2/6)

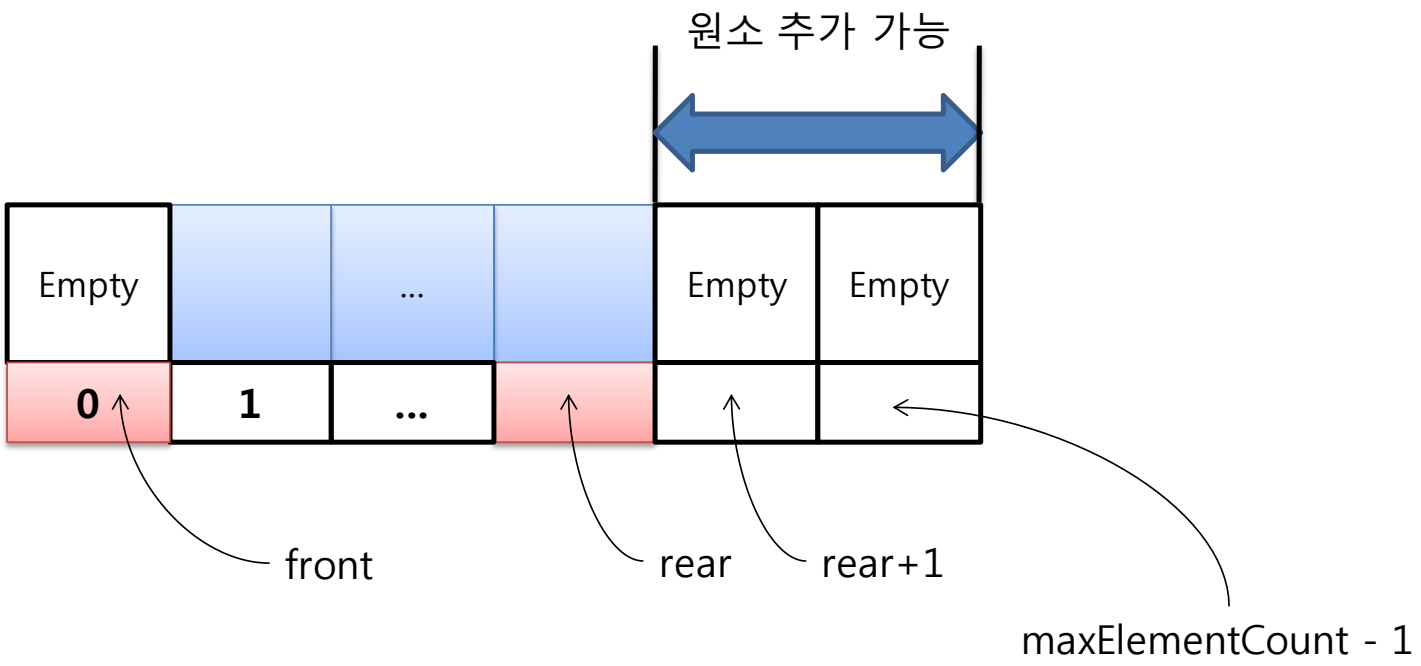


3. 배열로 구현한 선형 큐 (3/6)

- 큐의 생성
- 인큐
- 디큐과 피크
- 기타
- 예제 프로그램

3. 배열로 구현한 선형 큐 (4/6)

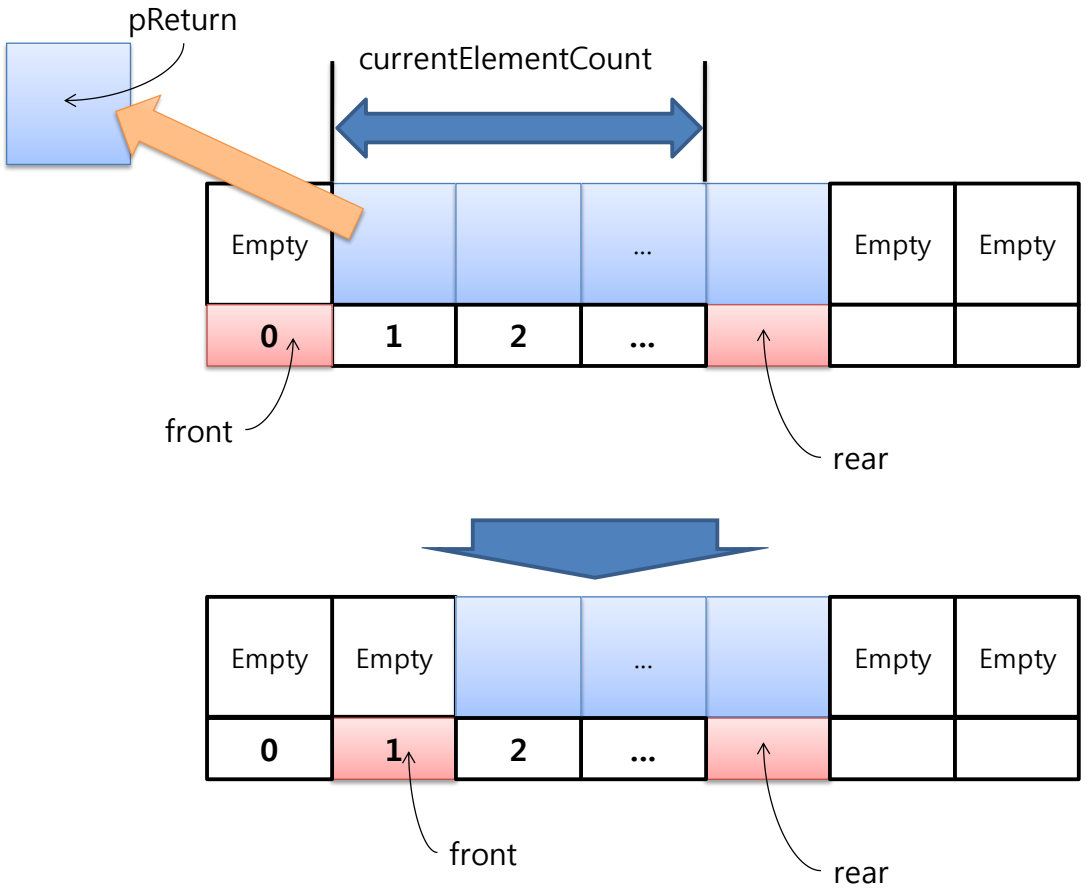
- 인큐



pQueue->rear == pQueue->maxElementCount - 1

3. 배열로 구현한 선형 큐 (5/6)

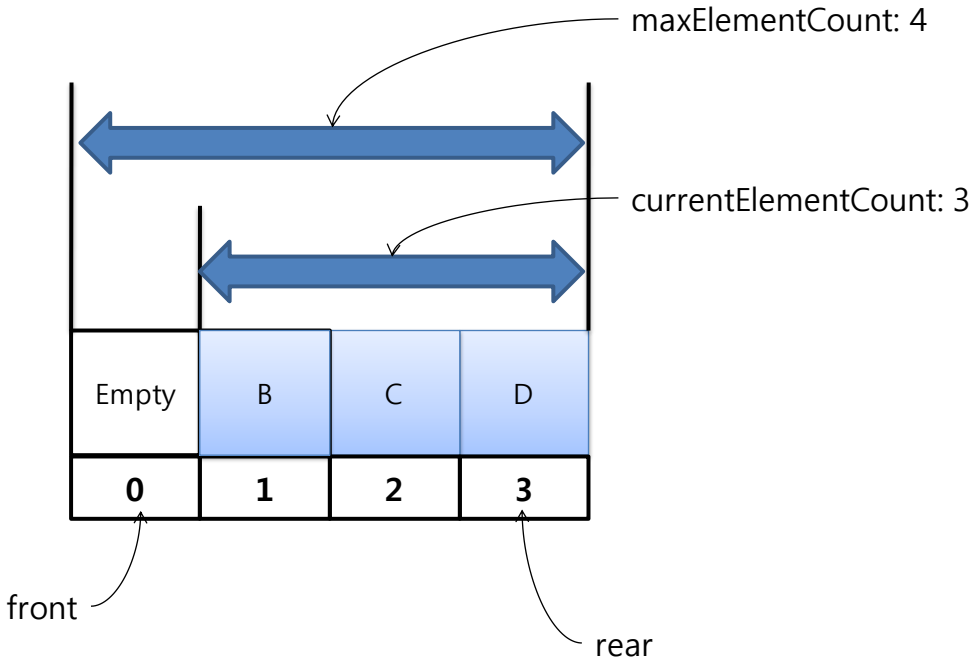
- 디큐와 피크



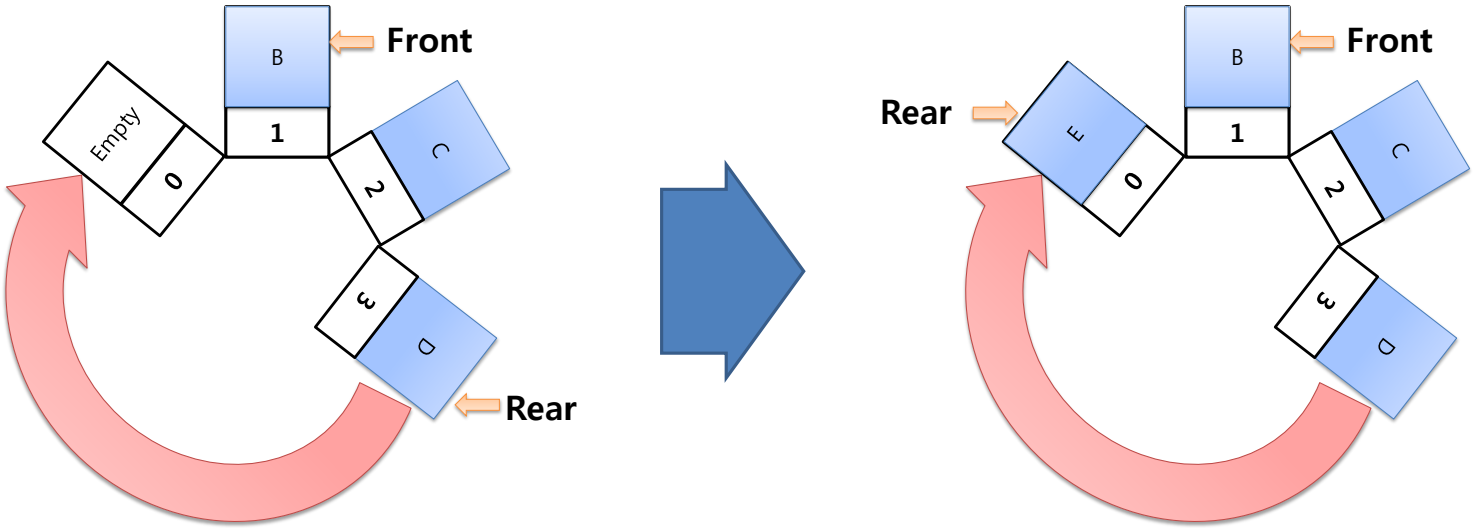
pQueue->currentElementCount == 0

3. 배열로 구현한 선형 큐 (6/6)

- 예제 프로그램
  - 시나리오
    - 인큐: A B C D
    - 디큐 1회
    - 피크 1회
    - 인큐 E

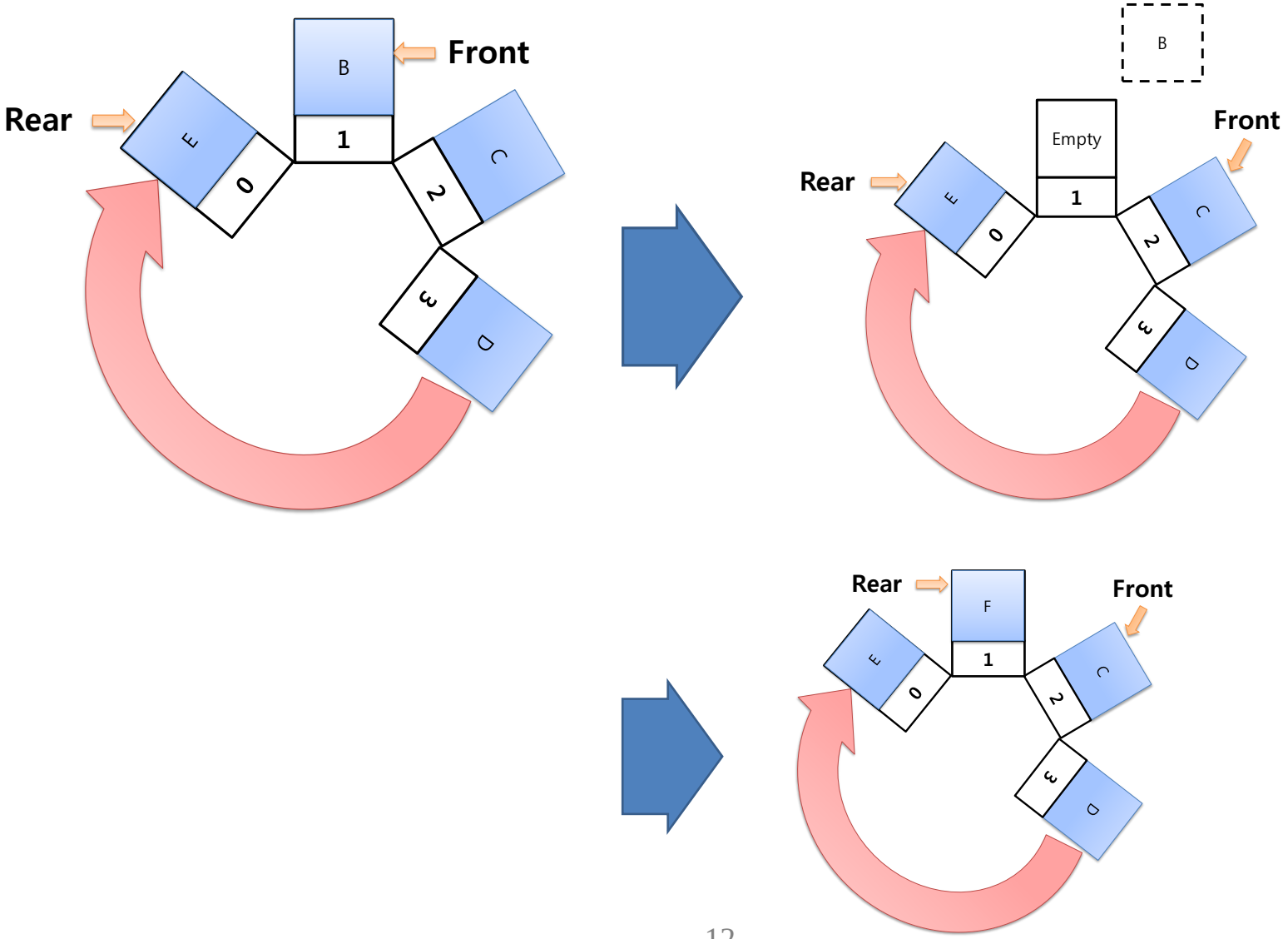


4. 배열로 구현한 원형 큐 (1/5)



rear = (rear + 1) % maxElementCount

4. 배열로 구현한 원형 큐 (2/5)



4. 배열로 구현한 원형 큐 (4/5)

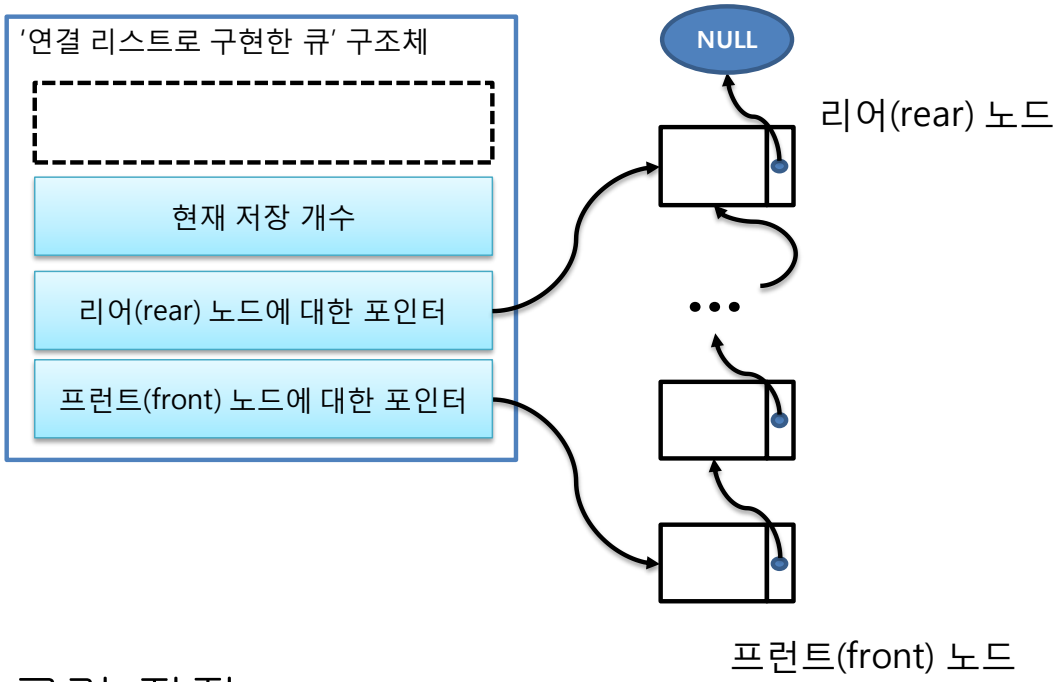
- 큐의 생성
- 인큐
- 디큐과 피크
- 기타
- 예제 프로그램

4. 배열로 구현한 원형 큐 (5/5)

- 예제 프로그램
  - 시나리오
    - 인큐: A B C D
    - 디큐 1회
    - 피크 1회
    - 인큐 E
  - 그 외
    - 함수 displayArrayQueue()

5. 연결 리스트로 구현한 큐 (1/7)

- 연결 리스트(Linked List)로 구현한 큐

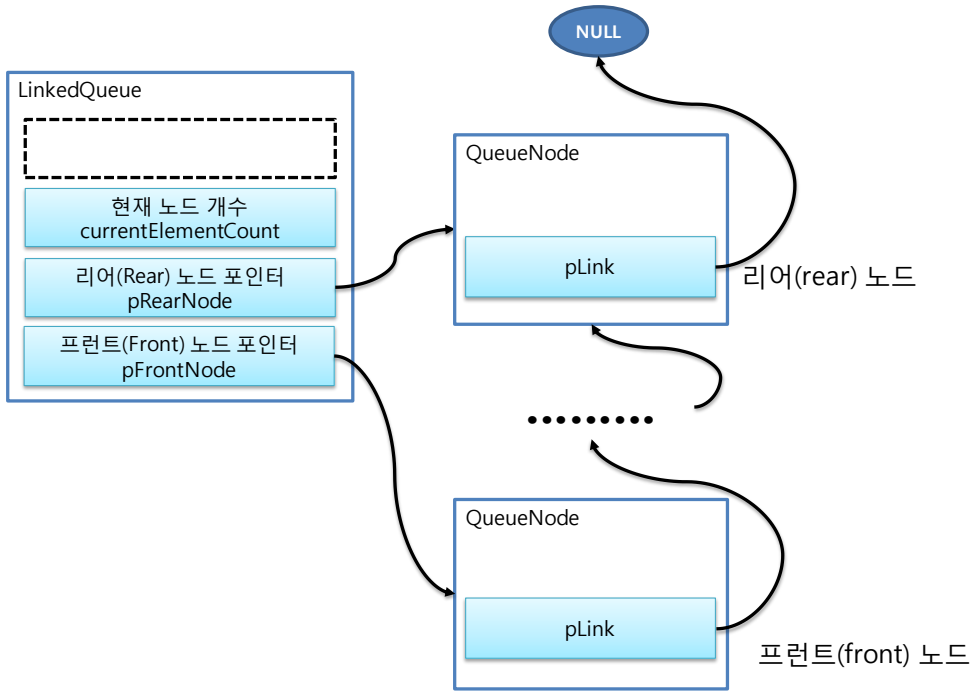


- 차이점
  - 큐의 크기 지정

5. 연결 리스트로 구현한 큐 (3/7)

- 구조체

- 소스 파일
  - linkedqueue.h

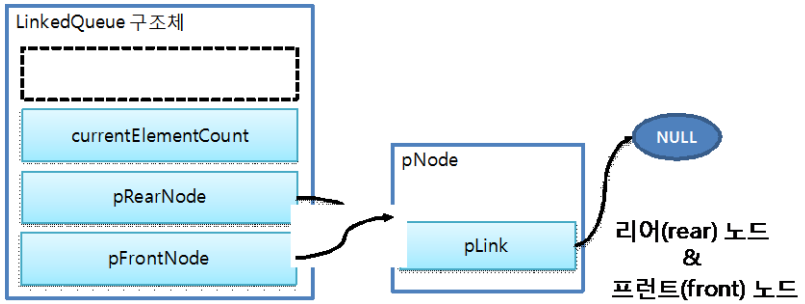


5. 연결 리스트로 구현한 큐 (4/7)

- 큐의 생성
- 인큐
- 디큐과 피크
- 기타
- 예제 프로그램

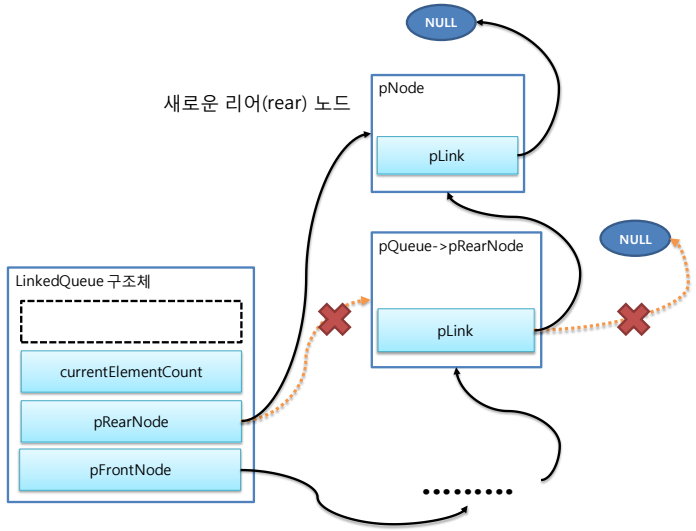
5. 연결 리스트로 구현한 큐 (5/7)

- 인큐
  - 공백 상태일 때




```
pQueue->pFrontNode = pNode;
pQueue->pRearNode = pNode;
```

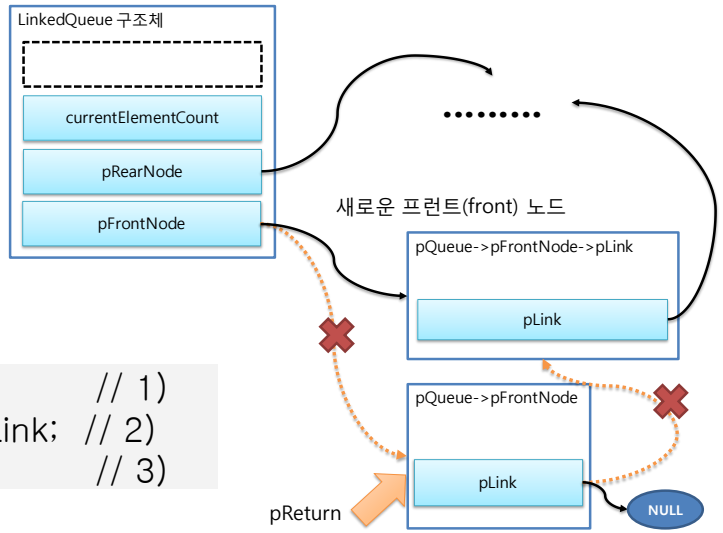
  - 공백 상태가 아닐 때




```
pQueue->pRearNode->pLink = pNode; // 1)
pQueue->pRearNode = pNode;        // 2)
```

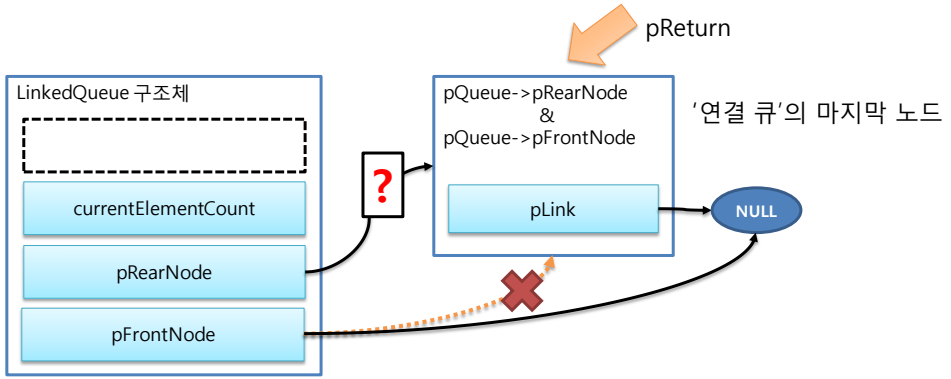
5. 연결 리스트로 구현한 큐 (6/7)

- 디큐
  - 노드가 2개 이상인 큐 (일반적인 경우)




```
pReturn = pQueue->pFrontNode;                    // 1)
pQueue->pFrontNode = pQueue->pFrontNode->pLink;  // 2)
pReturn->pLink = NULL;                           // 3)
```

  - 노드가 1개인 큐



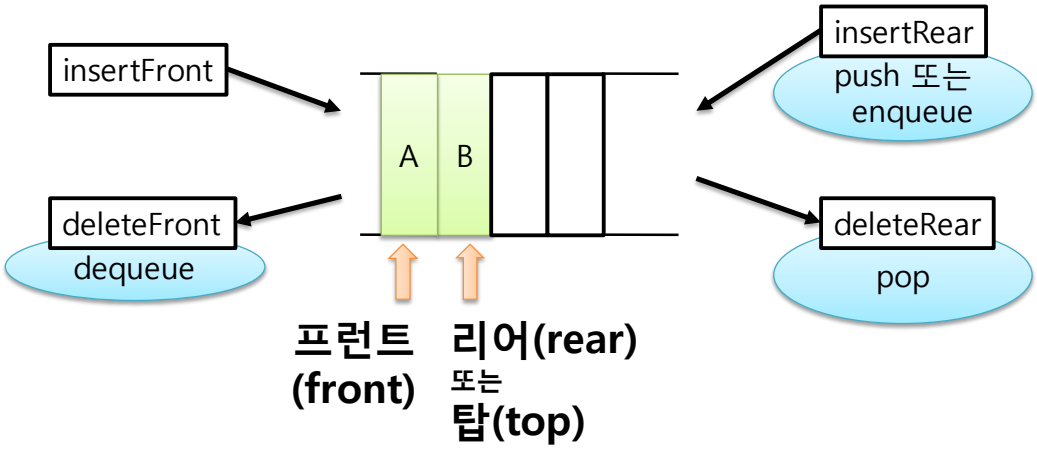

```
pQueue->pRearNode = NULL;
```

5. 연결 리스트로 구현한 큐 (7/7)

- 기타 연산들
  - 큐 제거
    - deleteLinkedList()

6. 연결 리스트로 구현한 덱 (1/9)

- 덱(Deque): Double-Ended Queue



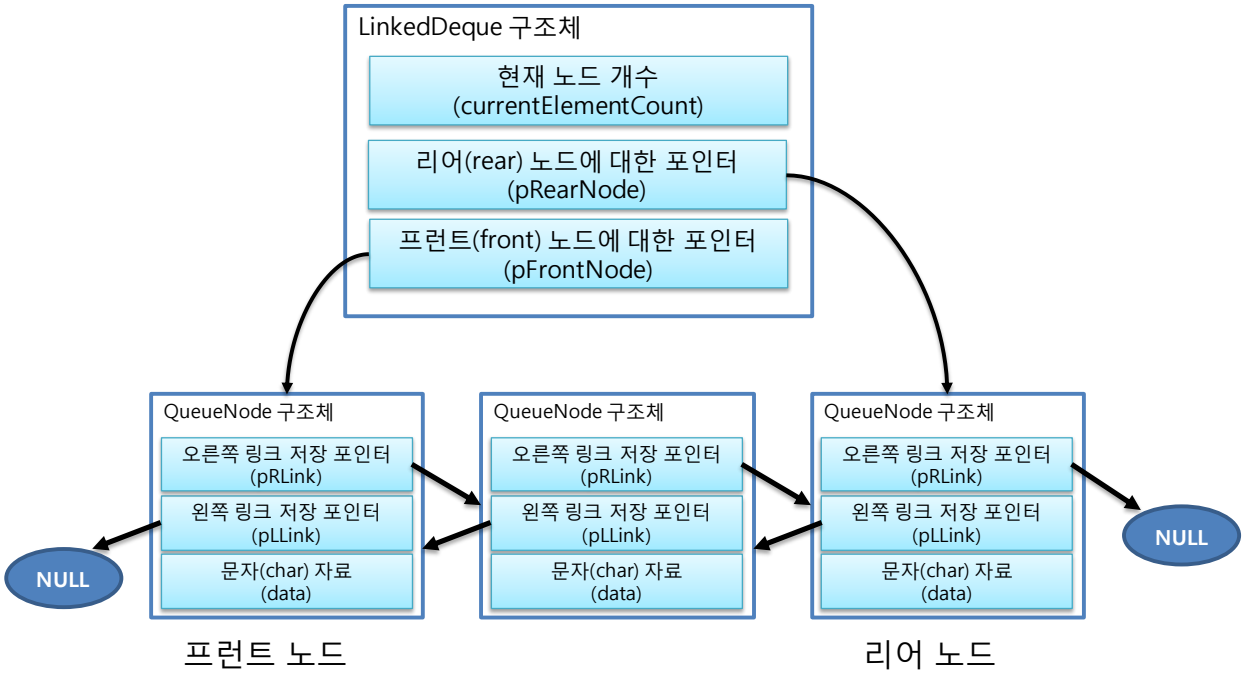
연산		덱	스택	큐
앞, 프런트(front)	추가	insertFront	없음	없음
	반환	deleteFront	없음	Dequeue
뒤, 리어(rear)	추가	insertRear	Push	Enqueue
	반환	deleteRear	Pop	없음

5. 연결 리스트로 구현한 덱 (2/9)

- 덱 추상 자료형
  - 덱 생성 / 덱 삭제
  - 앞 추가 insertFront()
  - 뒤 추가 insertRear()
  - 앞 제거 deleteFront()
  - 뒤 제거 deleteRear()
  - 앞 반환 peekFront()
  - 뒤 반환 peekRear()

5. 연결 리스트로 구현한 덱 (3/9)

- 구조체



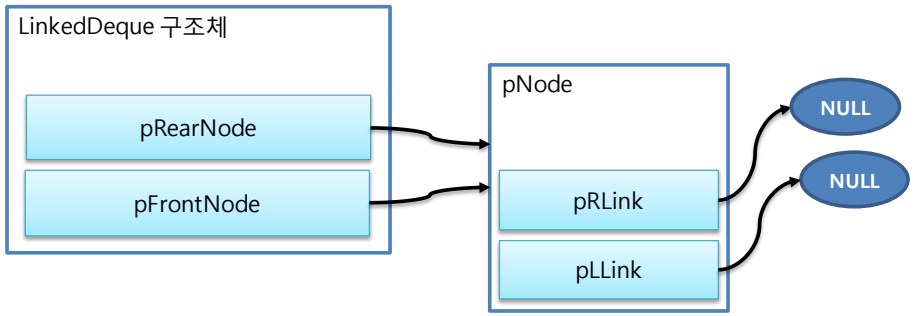
- 소스 파일
  - linkeddeque.h

5. 연결 리스트로 구현한 덱 (4/9)

- 덱의 생성
- 앞 추가
- 뒤 추가
- 앞 제거/앞 반환
- 뒤 제거/뒤 반환
- 기타
- 예제 프로그램

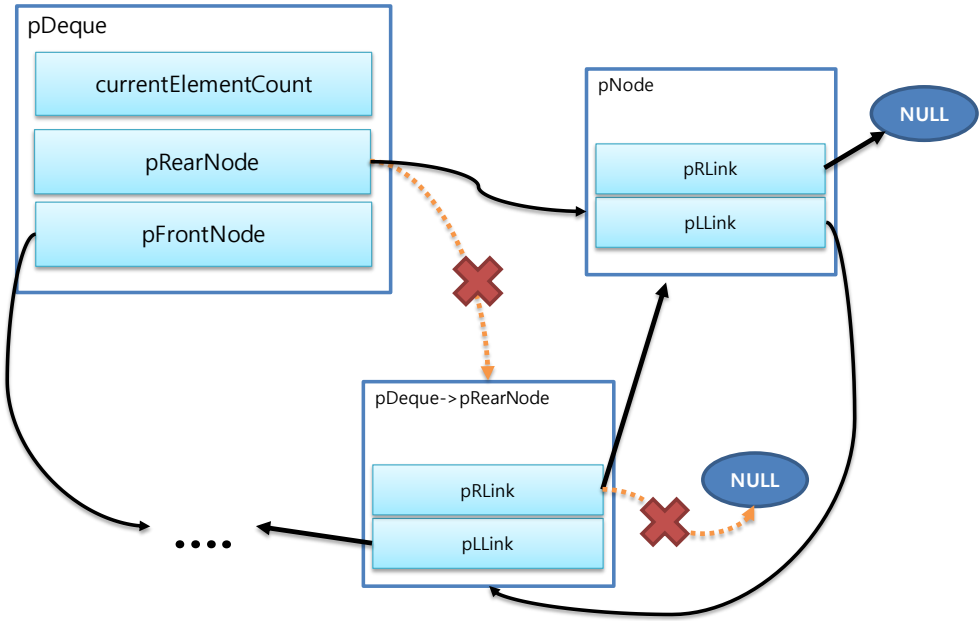
5. 연결 리스트로 구현한 덱 (5/9)

- 뒤 추가
  - 공백 상태일 때




```
pDeque->pFrontNode = pNode;
pDeque->pRearNode = pNode;
```

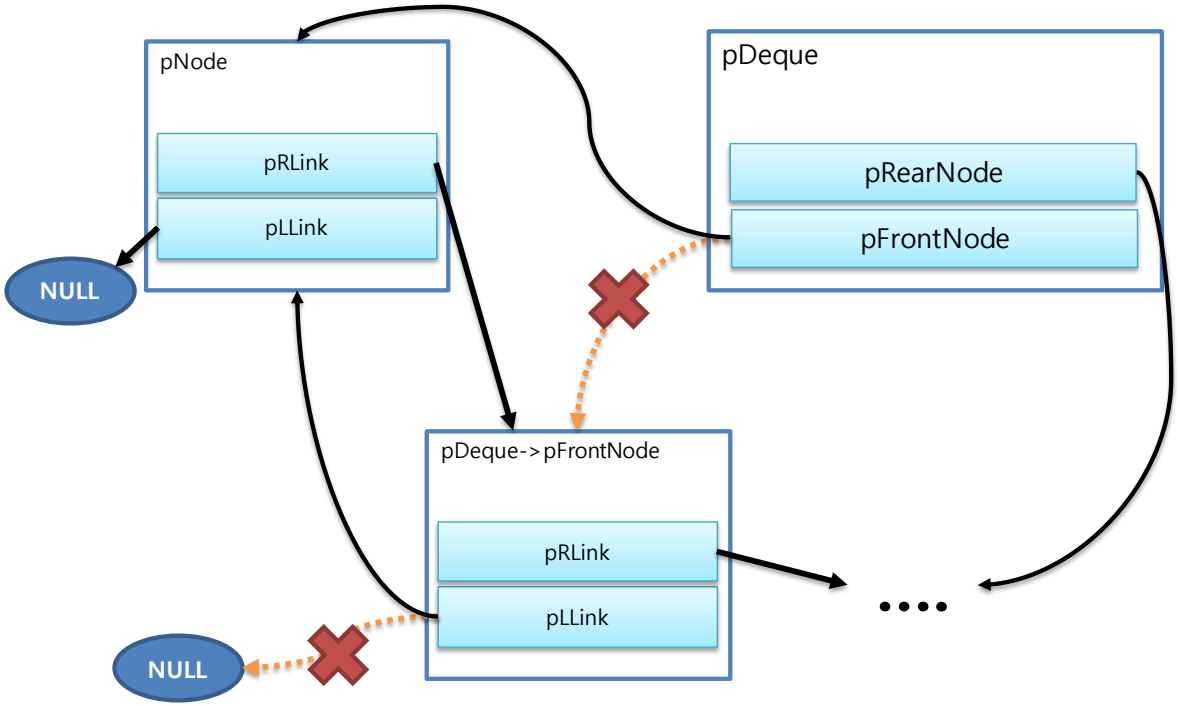
공백 상태가 아닐 때




```
pDeque->pRearNode->pRLink = pNode;
pNode->pLLink = pDeque->pRearNode;
pDeque->pRearNode = pNode;
```

5. 연결 리스트로 구현한 덱 (6/9)

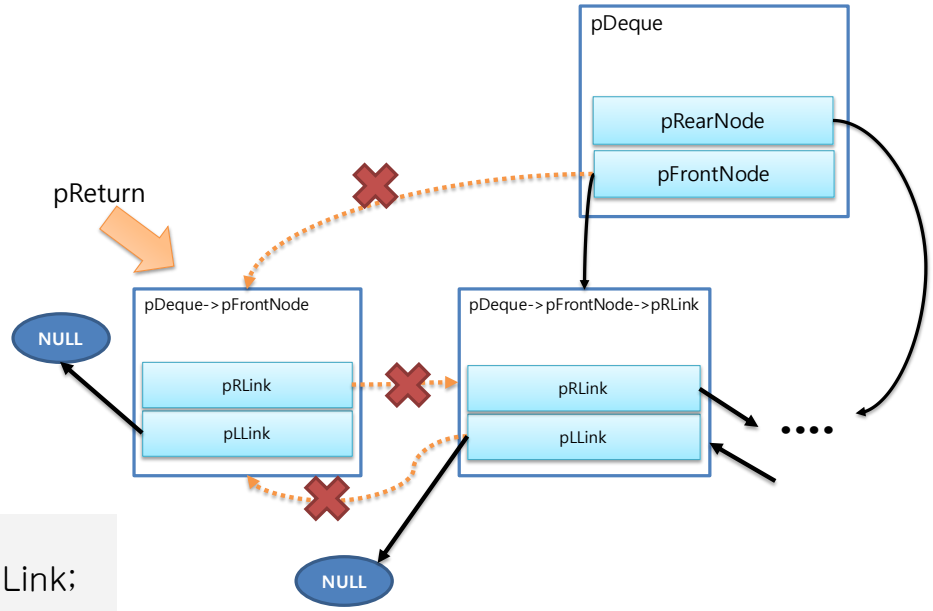
- 앞 추가
  - 공백 상태일 때
  - 공백 상태가 아닐 때




```
pDeque->pFrontNode->pLLink = pNode;
pNode->pRLink = pDeque->pFrontNode;
pDeque->pFrontNode = pNode;
```

5. 연결 리스트로 구현한 덱 (7/9)

- 앞 제거/반환

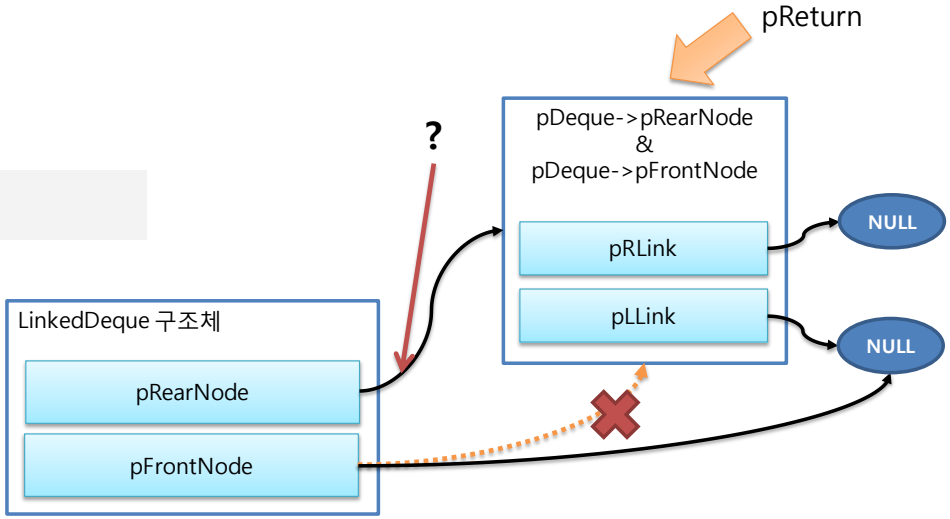


  - 공통

```
pReturn = pDeque->pFrontNode;
pDeque->pFrontNode = pDeque->pFrontNode->pRLink;
pReturn->pRLink = NULL;
```

  - 노드가 2개 이상인 덱 (일반적인 경우)

```
pDeque->pFrontNode->pLLink = NULL;
```



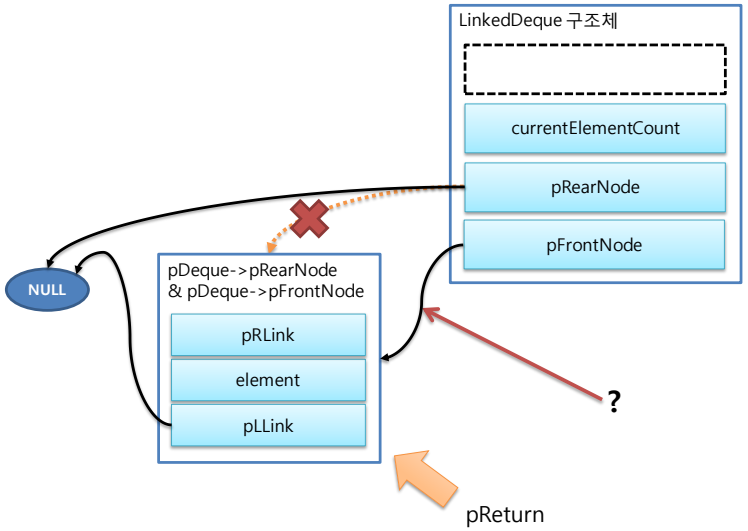
  - 노드가 1개인 큐

```
pDeque->pRearNode = NULL;
```

5. 연결 리스트로 구현한 덱 (8/9)

- 뒤 제거/반환
  - 공통

```
pReturn = pDeque->pRearNode;
pDeque->pRearNode = pDeque->pRearNode->pLLink;
pReturn->pLLink = NULL;
```

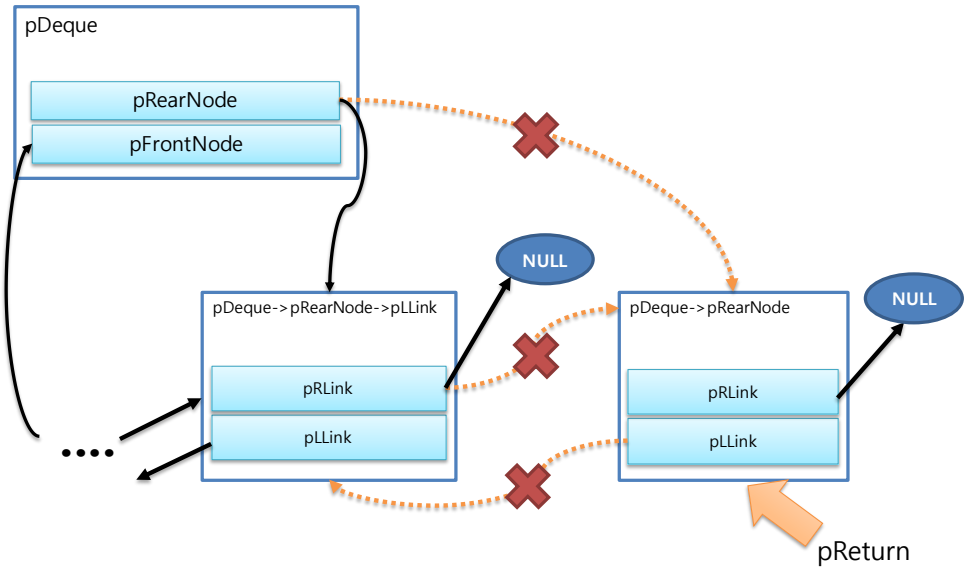


  - 노드가 2개 이상인 덱 (일반적인 경우)

```
pDeque->pRearNode->pRLink = NULL;
```

  - 노드가 1개인 큐

```
pDeque->pFrontNode = NULL;
```



5. 연결 리스트로 구현한 덱 (9/9)

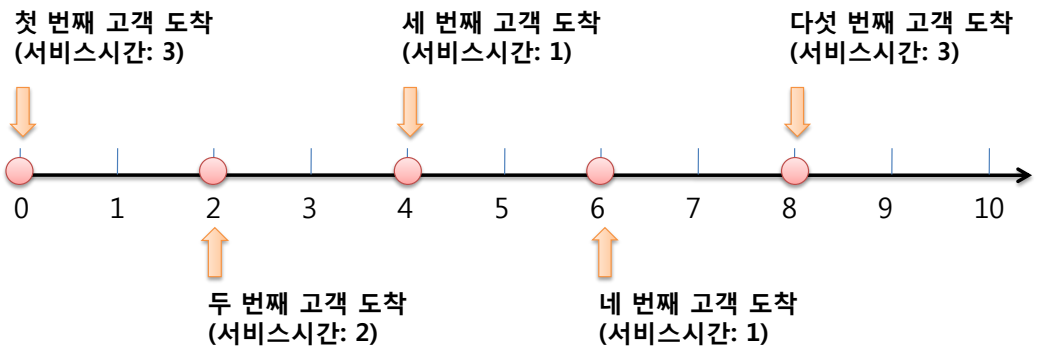
- 기타 연산들
  - 래퍼(Wrapper) 함수들
- 예제 프로그램
  - 시나리오
    - 앞 추가: A B
    - 뒤 추가: C D
    - 뒤 제거 1회
    - 앞 제거 1회

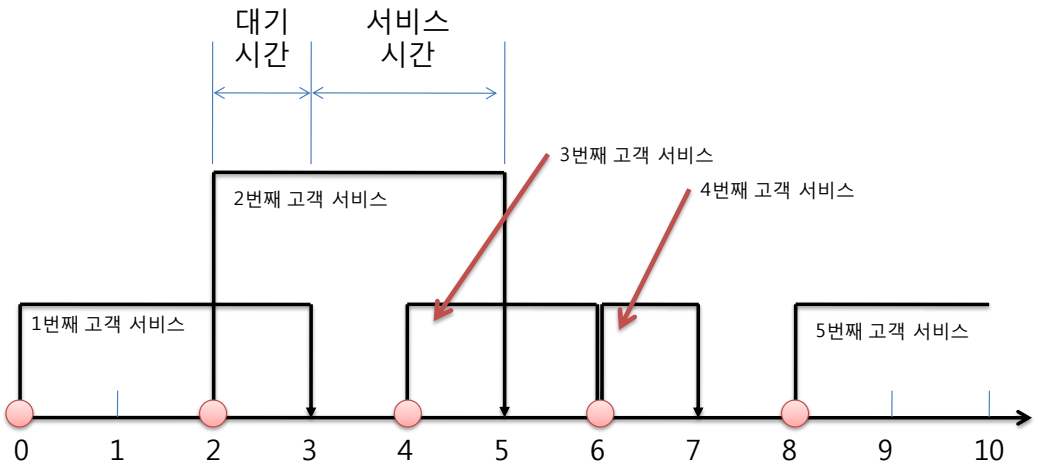
7. 큐 응용: 시뮬레이션 (1/8)

- 은행업무 시뮬레이션

7. 큐 응용: 시뮬레이션 (2/8)

- 시뮬레이션 알고리즘 (1)

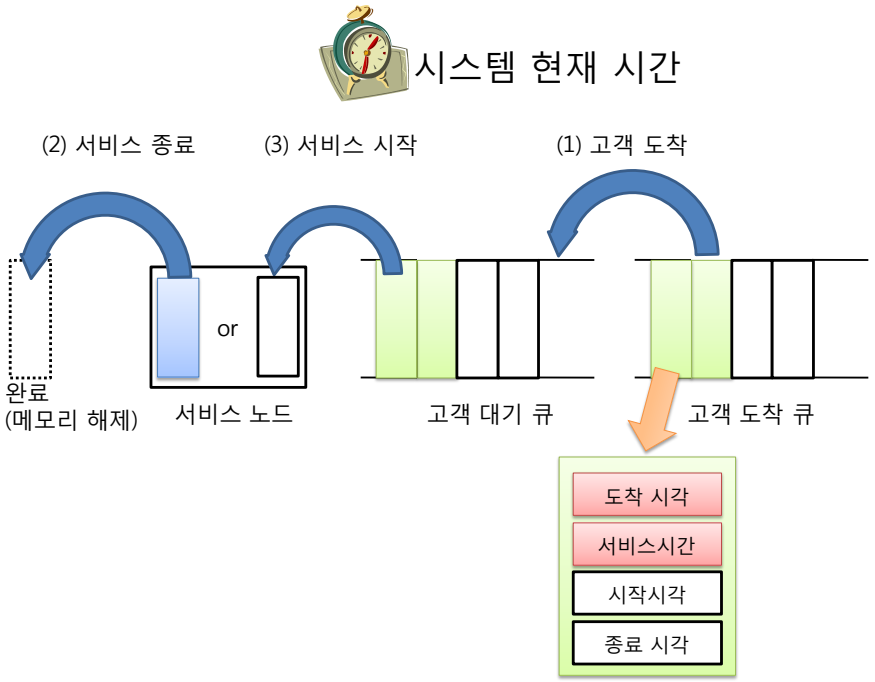


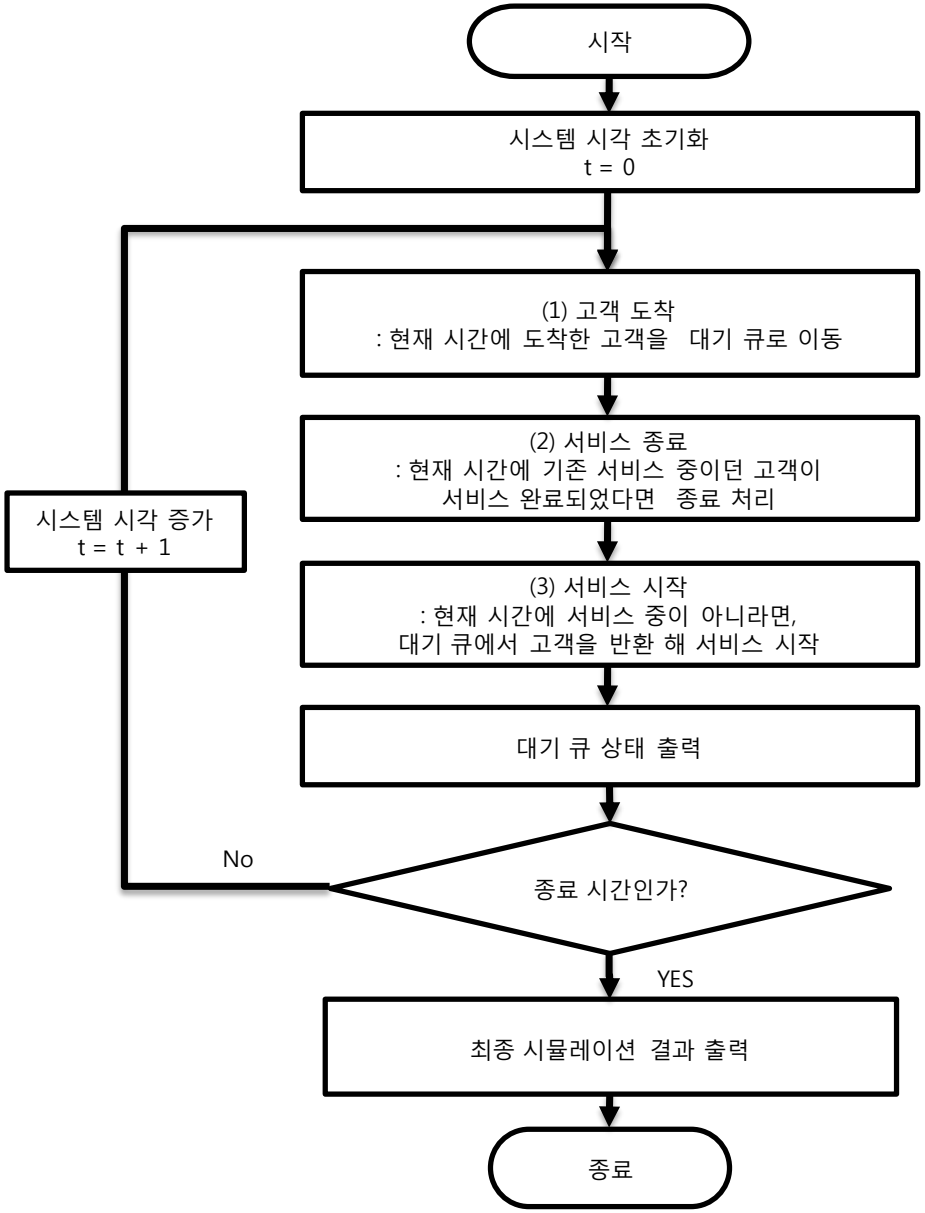


: 새로운 고객 도착

7. 큐 응용: 시뮬레이션 (3/8)

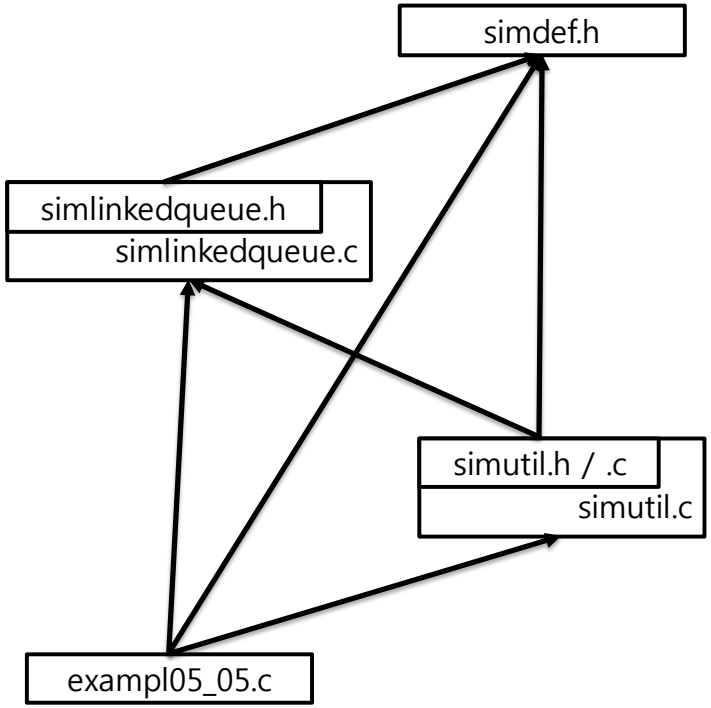
- 시뮬레이션 알고리즘 (2)




시작
시스템 시각 초기화
t = 0
(1) 고객 도착
: 현재 시간에 도착한 고객을 대기 큐로 이동
(2) 서비스 종료
: 현재 시간에 기존 서비스 중이던 고객이
서비스 완료되었다면 종료 처리
(3) 서비스 시작
: 현재 시간에 서비스 중이 아니라면,
대기 큐에서 고객을 반환 해 서비스 시작
대기 큐 상태 출력
종료 시간인가?
No
YES
시스템 시각 증가
t = t + 1
최종 시뮬레이션 결과 출력
종료

7. 큐 응용: 시뮬레이션 (5/8)

파일 이름
05_05.vcproj
simdef.h
simlinkedqueue.h
simlinkedqueue.c
simutil.h
simutil.c
exampl05_05.c



5. **연결 리스트로 구현한 큐**

7. 큐 응용: 시뮬레이션 (6/8)

- 문제 모델링

```
04:     typedef enum SimStatusType { arrival, start, end } SimStatus;
05:
06:     typedef struct SimCustomerType
07:     {
08:             SimStatus status;
09:             int arrivalTime;    // 도착 시각.
10:             int serviceTime;    // 서비스 시간.
11:             int startTime;      // 시작 시각.
12:             int endTime;        // 종료 시각: 시작 시각 + 서비스 시간.
13:     } SimCustomer;
```

simdef.h

7. 큐 응용: 시뮬레이션 (8/8)

- 시뮬레이션의 구현

```
05: void insertCutomer(int arrivalTime, int processTime, LinkedQueue *pQueue);
08: void processArrival(int currentTime, LinkedQueue *pArrivalQueue, LinkedQueue *pWaitQueue);
11: QueueNode* processServiceNodeStart(int currentTime, LinkedQueue *pWaitQueue);
14: QueueNode* processServiceNodeEnd(int currentTime, QueueNode *pServiceNode,
15:                                  int *pServiceUserCount, int *pTotalWaitTime);
18: void printSimCustomer(int currentTime, SimCustomer customer);
21: void printWaitQueueStatus(int currentTime, LinkedQueue *pWaitQueue);
24: void printReport(LinkedQueue *pWaitQueue,
25:                                      int serviceUserCount,
26:                                      int totalWaitTime);
```

이번 장에서는

- 큐의 개념
- 큐 추상 자료형
- 배열로 구현한 선형 큐
- 배열로 구현한 원형 큐
- 연결 리스트로 구현한 큐
- 연결 리스트로 구현한 덱
- 큐의 응용: 시뮬레이션