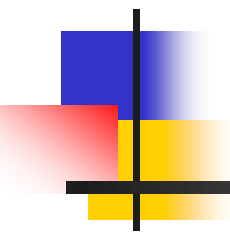


# C Programming (CSE2035)

## (Chap12. Queues)



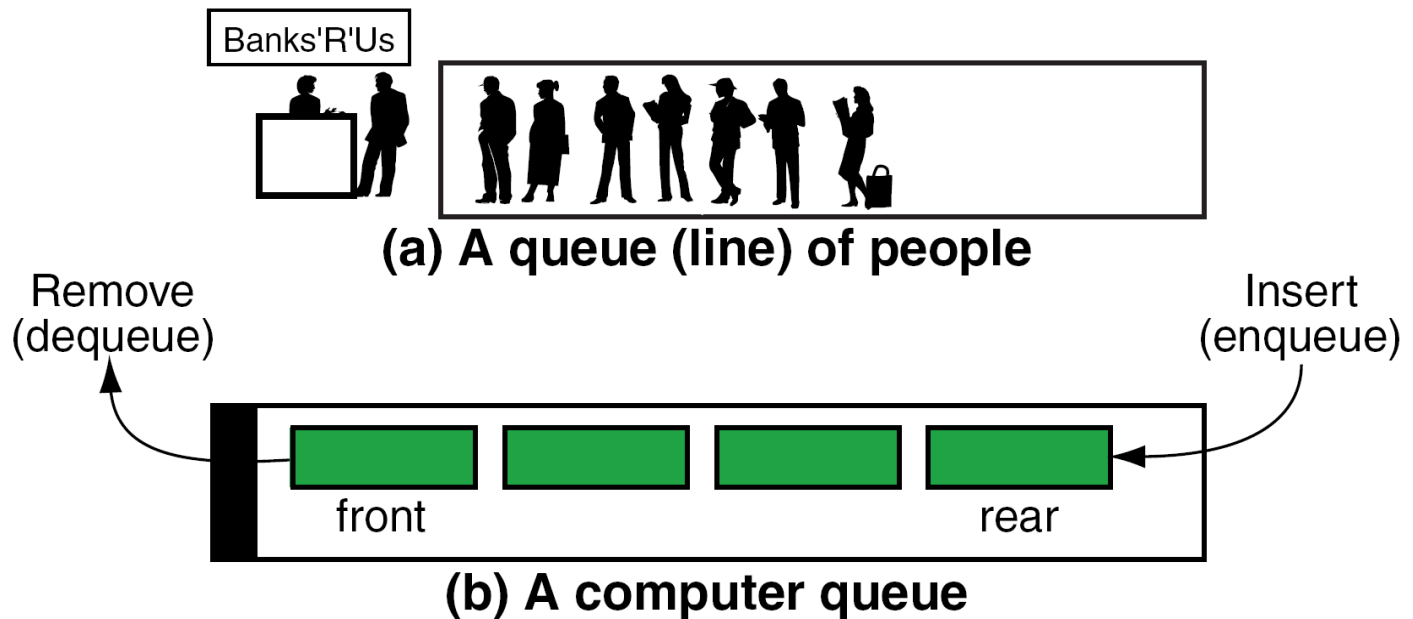
---

**Sungwon Jung, Ph.D.**

**Bigdata Processing & DB LAB**  
**Dept. of Computer Science and Engineering**  
**Sogang University**  
**Seoul, Korea**  
**Tel: +82-2-705-8930**  
**Email : [jungsung@sogang.ac.kr](mailto:jungsung@sogang.ac.kr)**

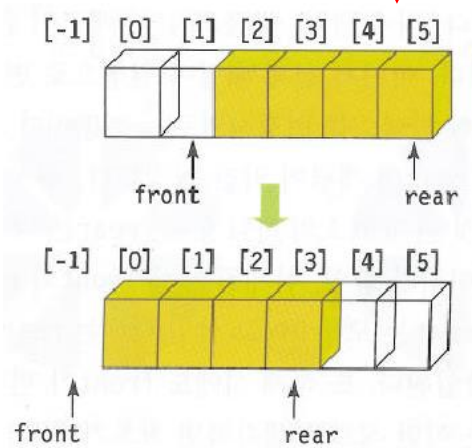
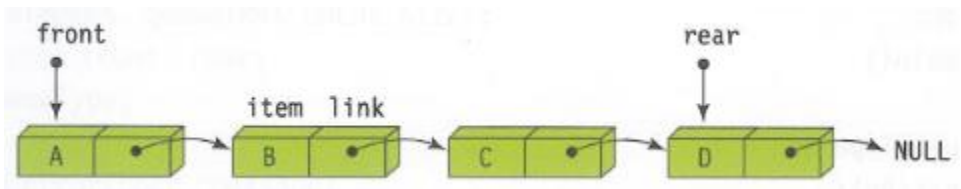
# Queues

- Queue는 뒤에서 새로운 데이터가 추가되고 앞에서 데이터가 하나씩 삭제되는 자료 구조를 가지고 있다.
- FIFO(first in-first out): 먼저 들어온 데이터가 먼저 나가는 구조



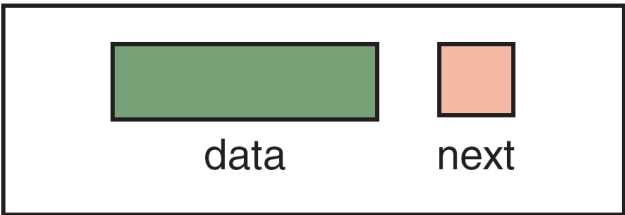
# Linked Queue

- Array를 이용하여 구현한 queue
  - 장점: 가장 간단하게 구현할 수 있는 방법은 배열을 이용한 방법.
  - 단점: 삽입과 관련된 변수 rear와 삭제와 관련된 변수 front 값이 증가만 하기 때문에 언젠가는 배열의 끝에 도달하게 되고 배열의 앞부분이 비어 있더라도 사용하지 못함.  
즉, 주기적으로 모든 요소들은 왼쪽으로 이동시켜야 함.
- Linked list를 이용하여 구현한 queue
  - 장점: 크기가 제한되지 않고, 삽입과 삭제에 용이하다.
  - 단점: 구현이 복잡하고 link 필드로 인해 메모리 공간을 더 많이 사용.

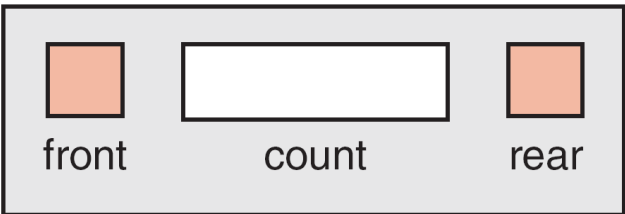


# Linked Queue

- Linked list를 이용한 queue의 자료구조



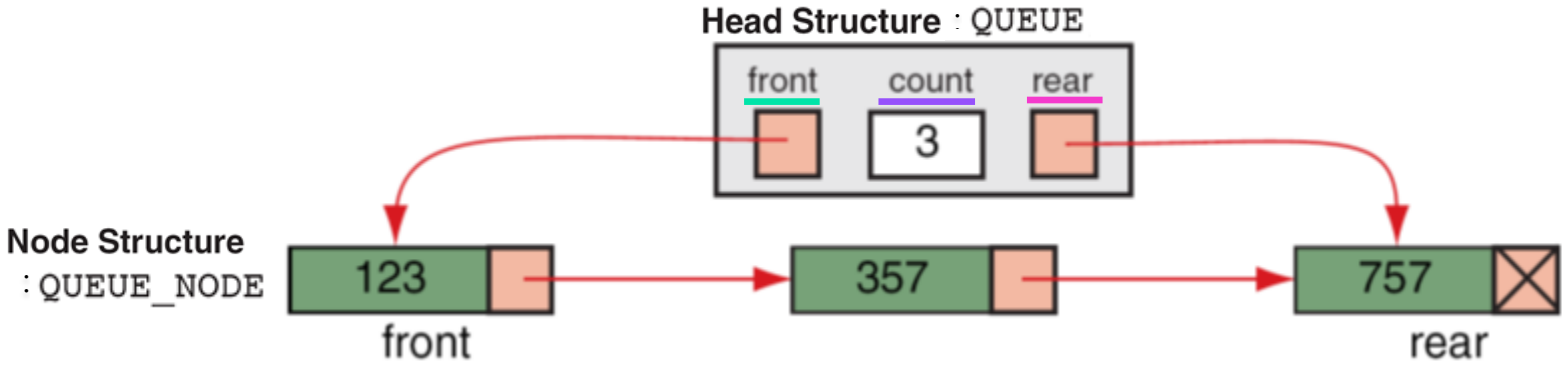
Node Structure



Head Structure

```
typedef struct node
{
    int data;
    struct node* next;
} QUEUE_NODE;

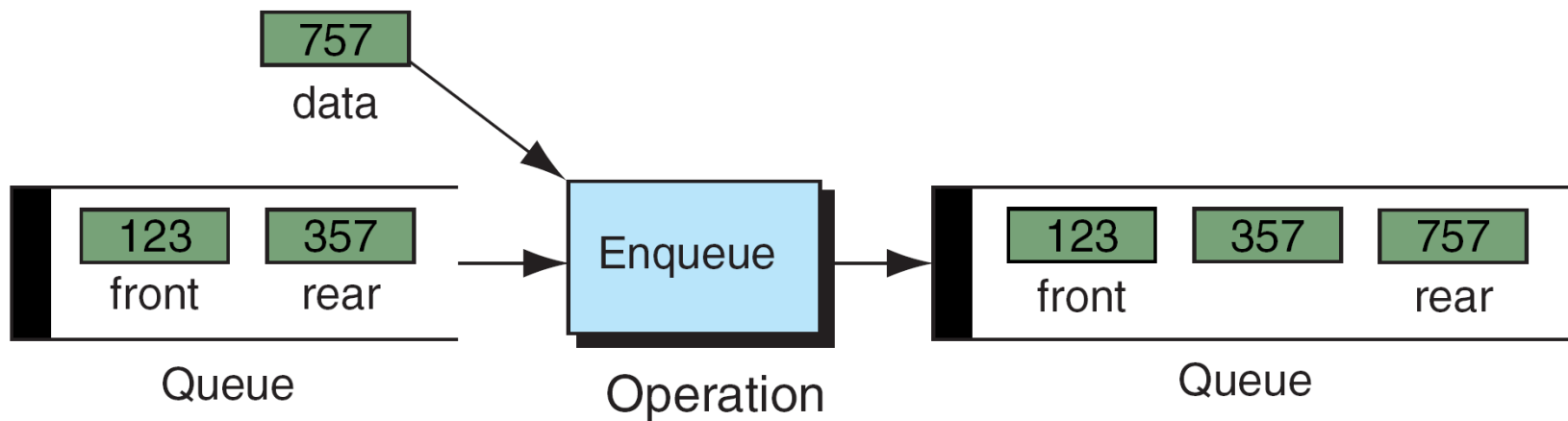
typedef struct
{
    QUEUE_NODE* front;
    int count;
    QUEUE_NODE* rear;
} QUEUE;
```



# Enqueue

## ■ Queue의 Enqueue연산

- 새로운 node를 생성한 후, data 값을 저장한다.
- Linked list의 끝에 새로운 node를 추가한다.

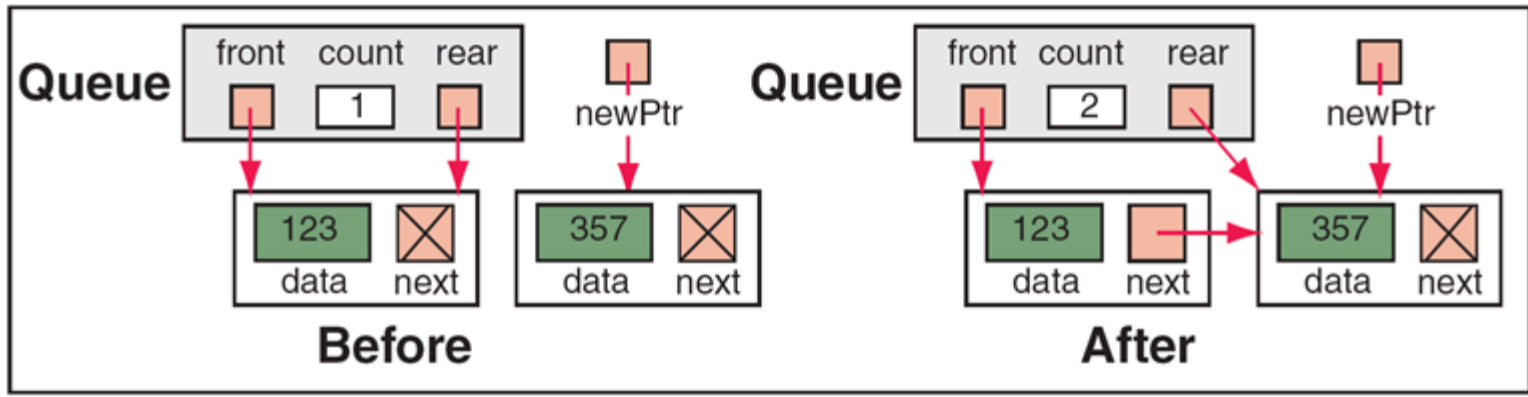


# Enqueue

- Queue가 NULL인 경우의 enqueue연산



- Queue에 기존의 node가 존재하는 경우의 enqueue연산



# Enqueue

```
bool enqueue (QUEUE* queue, int dataIn)
{
    // Local Declarations
    QUEUE_NODE* newPtr;

    // Statements
    if (!(newPtr = malloc(sizeof(QUEUE_NODE))))
        return false;

    newPtr->data = dataIn;
    newPtr->next = NULL;
    if (queue->count == 0)
        // Inserting into null queue
        queue->front = newPtr;
    else
        queue->rear->next = newPtr;
    (queue->count)++;
    queue->rear = newPtr;
    return true;
} // enqueue
```

## ■ bool enqueue(QUEUE\* queue, int dataIn)

- Node를 queue에 삽입하는 함수

Queue가 NULL인 경우

Queue에 기존의 node가 존재하는 경우