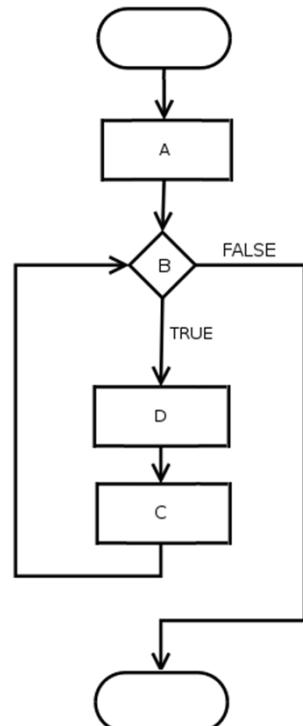


# 자료구조 및 알고리즘

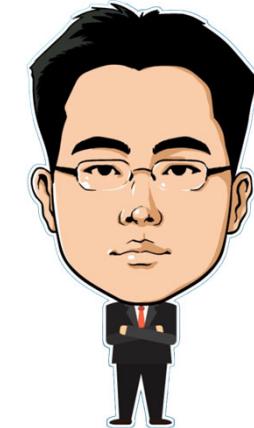
```
for(A;B;C)  
D;
```



큐  
(Queue)

Seo, Doo-Ok

Clickseo.com  
[clickseo@gmail.com](mailto:clickseo@gmail.com)



# 목 차



백문이불여일타(百聞而不如一打)

- 큐의 이해

- 큐 구현



# 큐의 이해



## ● 큐의 이해

백문이불여일타(百聞而不如一打)

- 선형 큐, 원형 큐, 연결 큐
- Python 내장 클래스: list
- C++ STL: <queue> 클래스
- Deque(Double-ended Queue)
- 큐의 구현: 순차.연결 자료구조

## ● 큐 구현

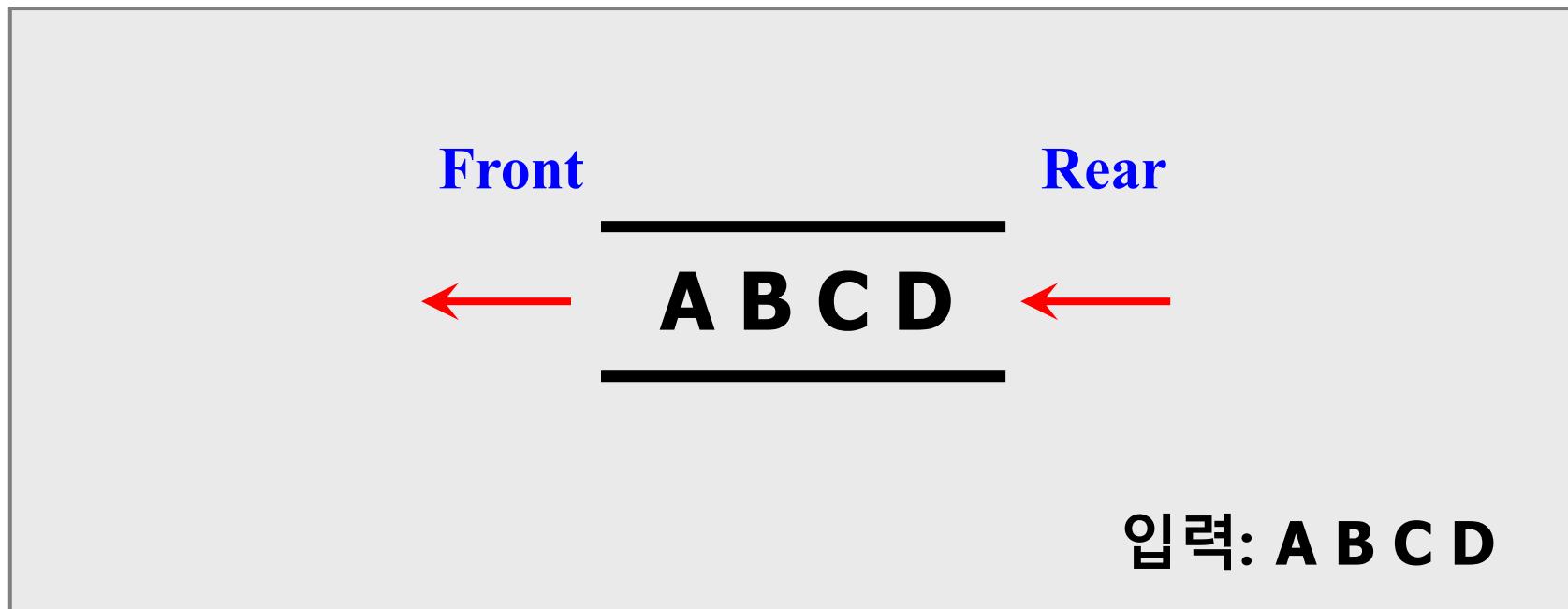


# 큐의 이해 (1/3)

- 큐(Queue)

- 선입선출(FIFO, First-In First-Out)

- 리스트의 한쪽 끝에서 삽입 작업이 이루어지고 반대쪽 끝에서 삭제 작업이 이루어져서 삽입된 순서대로 삭제되는 구조



# 큐의 이해 (2/3)

---

## ● 다양한 큐 활용

### ○ 회문(回文, Palindromes)

- 앞에서 읽거나 뒤에서 읽으나 똑같은 단어나 문장
- 예: 똑바로 읽어도 거꾸로 읽어도 '기러기', '토마토', '우영우' 등

### ○ 운영체제의 작업 큐

- 프로세스 스케줄링 큐(Process Scheduling Queue)
- 프린터 버퍼 큐(Printer Buffer Queue)

### ○ 시뮬레이션에서의 큐잉 시스템

- 모의실험
- 큐잉 이론(Queueing Theory)
- 이벤트 발생시기
  - 시간 구동 시뮬레이션(Time-Driven Simulation)
  - 사건 구동 시뮬레이션(Event-Driven Simulation)

# 큐의 이해 (3/3)

---

## ● 운영체제(OS, Operating System)

### ○ 프로세스 스케줄링(Process Scheduling) 알고리즘별 분류

- **FIFO**(First In First Out): **FCFS**(First Come First Service):
- **SJF**(Shortest Job First) **스케줄링**
- **우선순위**(Priority) **스케줄링**
- **기한부**(Deadline) **스케줄링**
- **라운드 로빈**(Round Robin) **스케줄링**
- **SRT**(Short Remaining Time) **스케줄링**
- **HRN**(Highest Response ratio Next) **스케줄링**
- **다단계 큐**(MLQ, Multi Level Queue) **스케줄링**
- **다단계 피드백 큐**(MFQ, Multilevel Feedback Queue) **스케줄링**
- ...

### ○ 프린터 버퍼 큐(Printer Buffer Queue)



# 큐의 이해

선형 큐, 원형 큐, 연결 큐

front      rear  
-1           -1

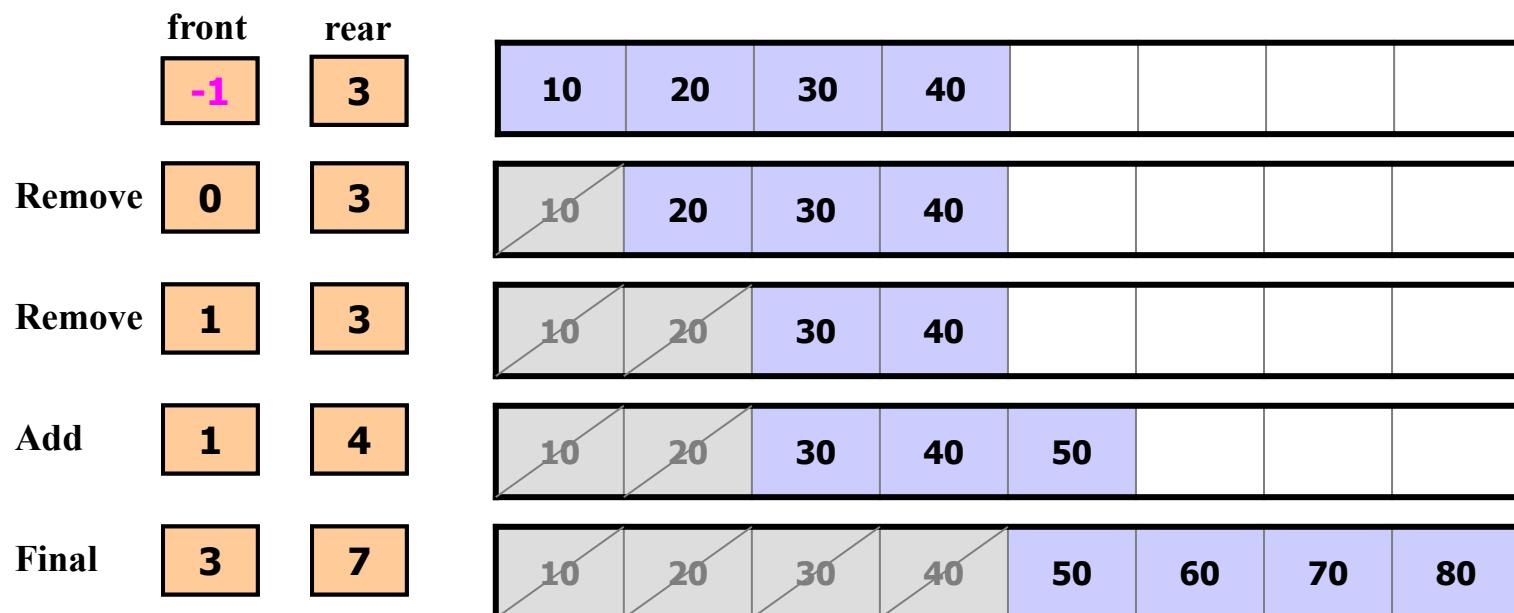


# 선형 큐 (1/3)

## ● 선형 큐(Linear Queue)

### ○ 큐의 상태

- 초기 상태:  $\text{front} = \text{rear} = -1$
- 공백 상태:  $\text{front} == \text{rear}$
- 포화 상태:  $\text{rear} == n - 1$  ( $n$  : 배열의 크기,  $n - 1$  : 배열의 마지막 인덱스)

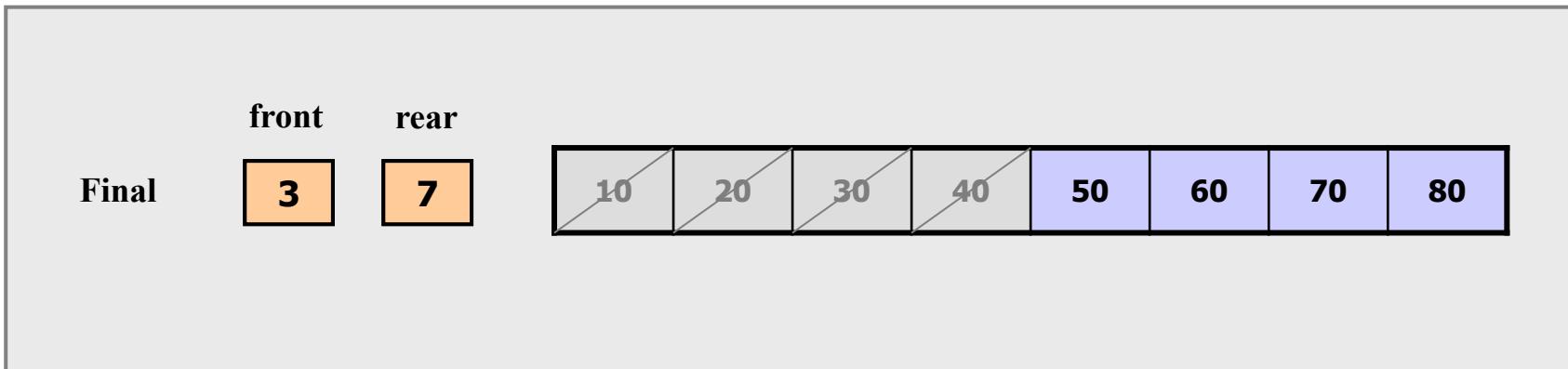


# 선형 큐 (2/3)

## ● 선형 큐

### ○ 선형 큐의 잘못된 포화 상태 인식

- 큐에서 삽입과 삭제를 반복하면서 아래와 같은 상태일 경우, 앞부분에 빈자리가 있지만 rear = n - 1 상태이므로 포화 상태로 인식하고 더 이상의 삽입을 수행하지 않는다.



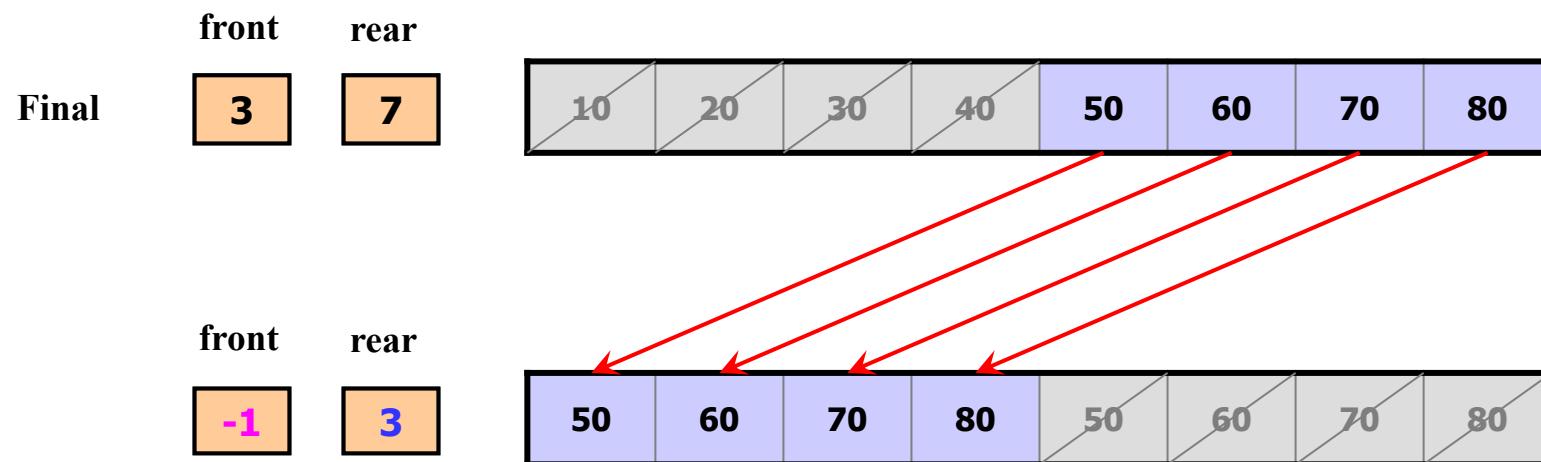
$\text{rear} + 1 == \text{maxQueueSIZE}$

# 선형 큐 (3/3)

## ● 선형 큐

### ○ 선형 큐의 잘못된 포화상태 인식의 해결 방법 #1

- 저장된 원소들을 배열의 앞부분으로 이동
  - 순차자료에서의 이동 작업은 연산이 복잡하여 효율성이 떨어짐

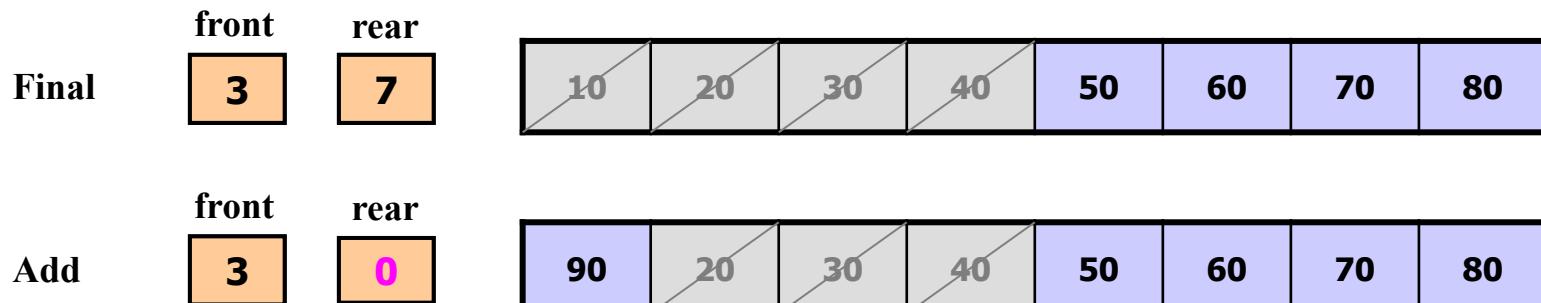
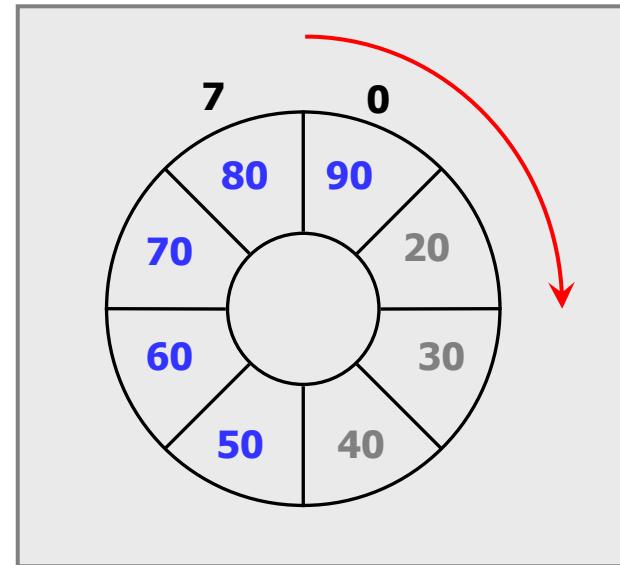


# 원형 큐 (1/2)

## ● 원형 큐(Circular Queue)

- 선형 큐의 잘못된 포화상태 인식의 해결 방법 #2

( $\text{rear} + 1$ ) %  $\text{maxQueueSIZE}$

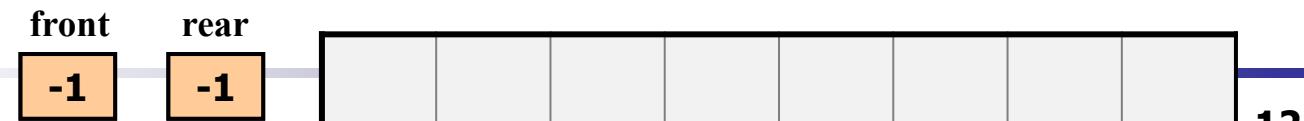
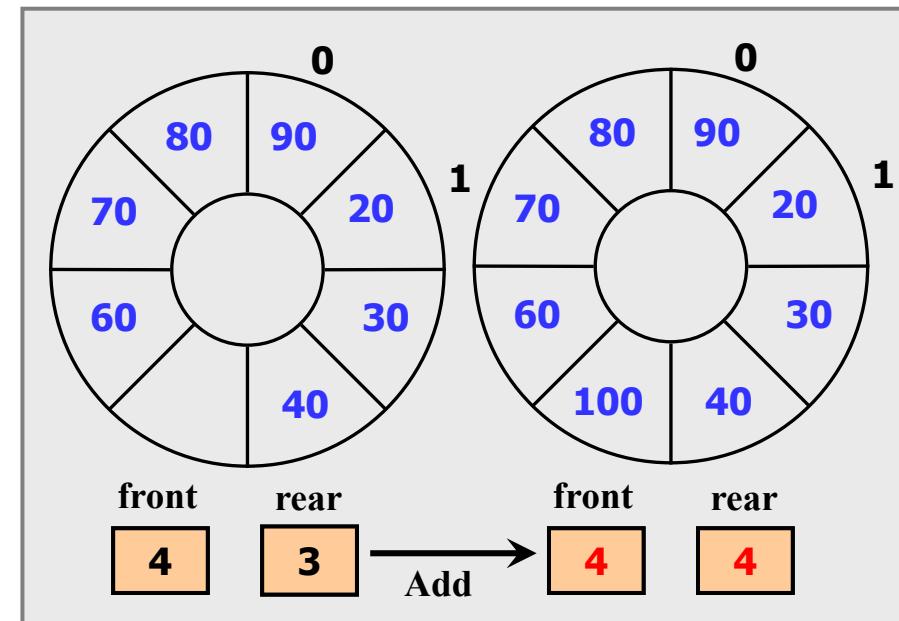
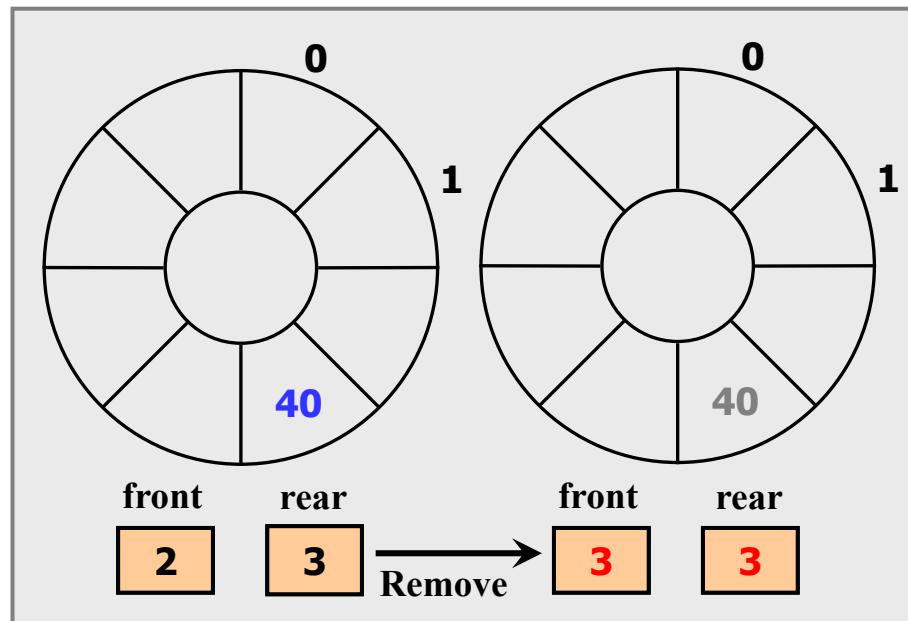


# 원형 큐 (2/2)

## ● 원형 큐: 문제점

### ○ 문제점

- 빈 큐와 꽉 찬 큐의 판정불가: `front == rear`
- 별도의 Count 변수 유지



# 연결 큐

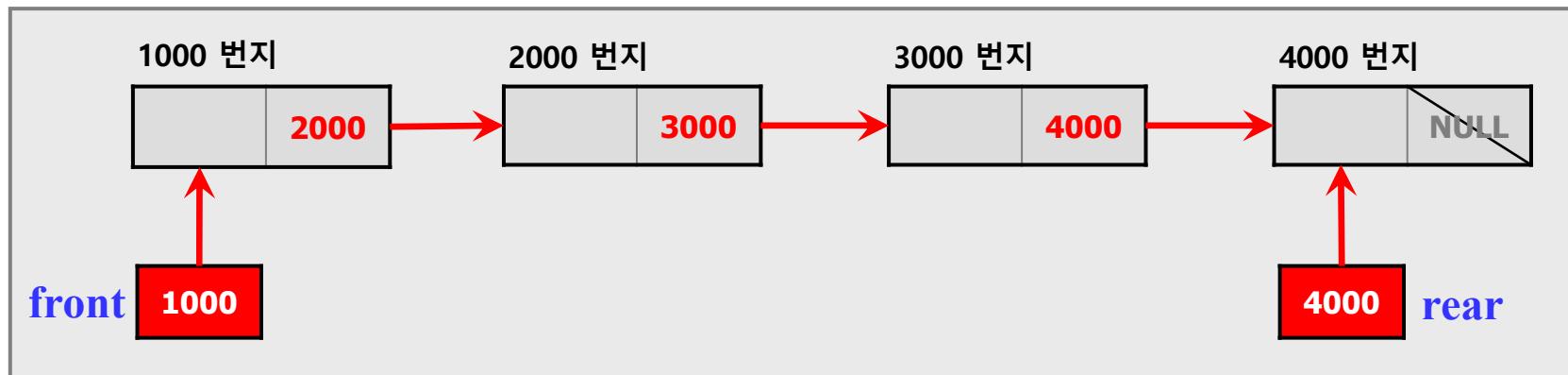
## ● 연결 큐(Linked Queue)

### ○ 순차 자료구조로 구현한 큐의 문제점

- 크기가 제한되어 큐의 길이를 마음대로 사용할 수 없다.
- 원소가 없을 때에도 항상 그 크기를 유지하고 있어야 하므로 메모리 낭비

### ○ 연결 큐의 구조

- 데이터 필드와 링크 필드를 가진 노드로 구성
  - front** : 첫 번째 노드를 가리키는 포인터
  - rear** : 마지막 노드를 가리키는 포인터
  - 초기 상태(공백 큐): front와 rear 모두 널(NULL) 포인터로 설정





## 큐의 이해

**Python 내장 클래스: list**

**C++ STL: queue 클래스**



# Python 내장 클래스: list

예제 6-1: 큐의 이해 -- Python 내장 클래스(list class)

| Python

```
q = []

while True:
    num = int(input('임의의 정수 입력(종료: 0): '))
    if num == 0:
        break
    q.append(num)

print(f'queue size      : {len(q)} ')
print(f'queue is empty: {len(q) == 0} ')

while q:
    # print(f'front element: {q.pop(0)} ')
    print(f'front element: {q[0]} ')
    q.pop(0)
```

```
IDLE Shell 3.11.2
File Edit Shell Debug Options Window 
Python 3.11.2 (tags/v3.11.2:878ead1, Type "help", "copyright", "credits"
>>> ===== RESTART: C:\Users\
임의의 정수 입력(종료: 0): 10
임의의 정수 입력(종료: 0): 20
임의의 정수 입력(종료: 0): 30
임의의 정수 입력(종료: 0): 40
임의의 정수 입력(종료: 0): 50
임의의 정수 입력(종료: 0): 0
queue size : 5
queue is empty: False
front element: 10
front element: 20
front element: 30
front element: 40
front element: 50
>>>
```

# C++ STL: queue 클래스 (1/4)

## ● 컨테이너 라이브러리(Containers Library)

### ○ 컨테이너 어댑터(Container Adaptor)

- <stack> : 스택(Stack) 구조, LIFO(Last in First out) 데이터 구조
- <queue> : 큐(Queue) 구조, FIFO(First in First out) 데이터 구조
- <priority\_queue> : 우선순위 큐(Priority queue)
  
- <flat\_set> (since C++23)
- <flat\_map> (since C++23)
- <flat\_multiset> (since C++23)
- <flat\_multimap> (since C++23)



# C++ STL: queue 클래스 (2/4)

## ● queue 클래스

- 큐, FIFO(First in First out)

```
// C++ STL : <queue>
#include <queue>
using namespace std;

queue<DataType> queueName // 빈 큐 생성

void push(const value_type &val); // 데이터 추가
void pop(); // 데이터 삭제
value_type &front(); // 첫 번째 원소 반환
value_type &back(); // 마지막 원소 반환
bool empty() const; // 빈 큐 여부
size_type size() const; // 큐의 크기
```

# C++ STL: queue 클래스 (3/4)

예제 6-2: 큐의 이해 -- C++ STL(queue class)

| C++

```
#include <iostream>
#include <queue>
using namespace std;

int main(void)
{
    int num;
    queue<int> q;

    while (true) {
        cout << "임의의 정수 입력(종료: 0): ";
        cin >> num;
        if (num == 0)
            break;
        q.push(num);
    }

    cout << endl;
    cout << "queue size : " << q.size() << endl;
    cout << "queue is empty: " << q.empty() << endl;

    while (!q.empty()) {
        cout << "front element : " << q.front() << endl;
        q.pop();
    }
    return 0;
}
```

```
Microsoft Visual Studio 디버그 x + ▾
임의의 정수 입력(종료: 0): 10
임의의 정수 입력(종료: 0): 20
임의의 정수 입력(종료: 0): 30
임의의 정수 입력(종료: 0): 40
임의의 정수 입력(종료: 0): 50
임의의 정수 입력(종료: 0): 0
queue size : 5
queue is empty: 0
front element : 10
front element : 20
front element : 30
front element : 40
front element : 50
C:\Users\click\OneDrive\문서\cppClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...
```

# C++ STL: queue 클래스 (4/4)

예제 6-3: 큐의 이해 -- C++ STL(priority\_queue class)

| C++

```
#include <iostream>
#include <queue>
using namespace std;

int main(void)
{
    int num;
    priority_queue<int> pQ;

    while (true) {
        cout << "임의의 정수 입력(종료: 0): ";
        cin >> num;
        if (num == 0)
            break;
        pQ.push(num);
    }

    cout << endl;
    cout << "queue size : " << pQ.size() << endl;
    cout << "queue is empty: " << pQ.empty() << endl;

    while (!pQ.empty()) {
        cout << pQ.top() << " ";
        pQ.pop();
    }
    return 0;
}
```

```
Microsoft Visual Studio 디버그 x + ▾
임의의 정수 입력(종료: 0): 30
임의의 정수 입력(종료: 0): 10
임의의 정수 입력(종료: 0): 50
임의의 정수 입력(종료: 0): 20
임의의 정수 입력(종료: 0): 40
임의의 정수 입력(종료: 0): 0
queue size : 5
queue is empty: 0
50 40 30 20 10
C:\Users\click\OneDrive\문서\cppClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...
```



## 큐의 이해

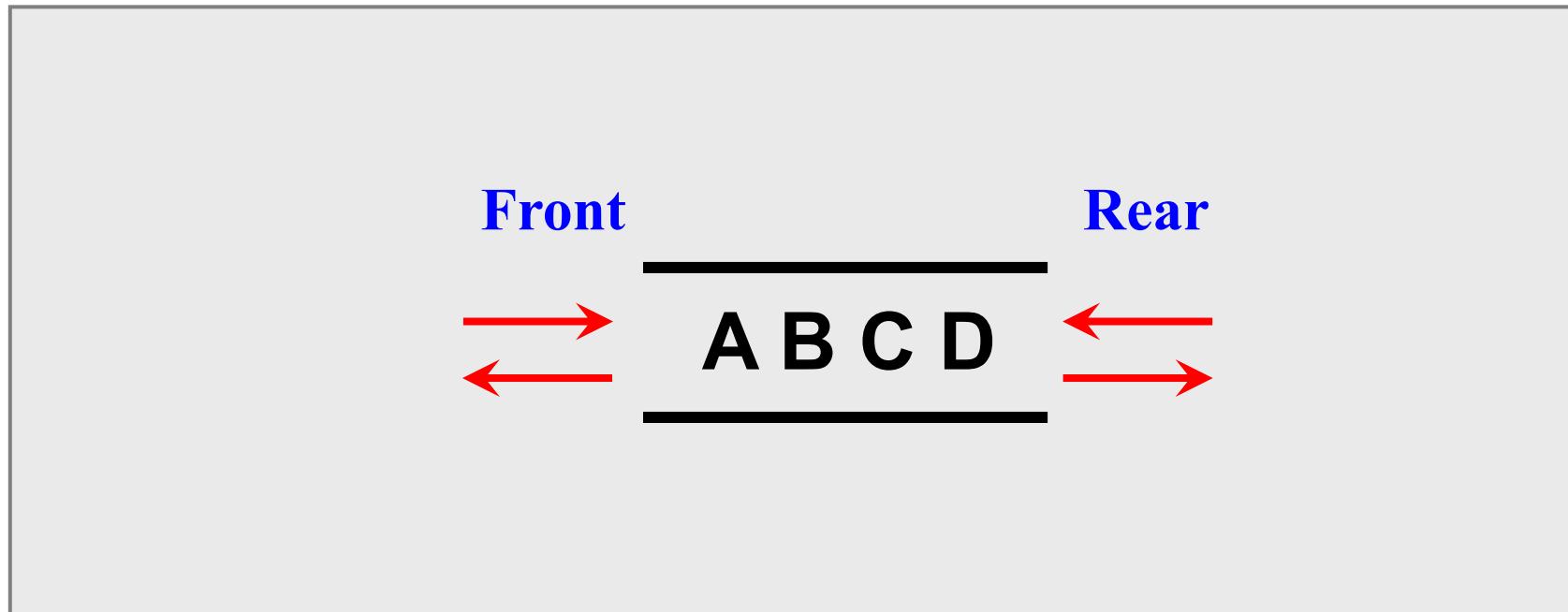
**Deque(Double-ended Queue)**



# Deque (1/2)

- **Deque(Double-ended Queue)**

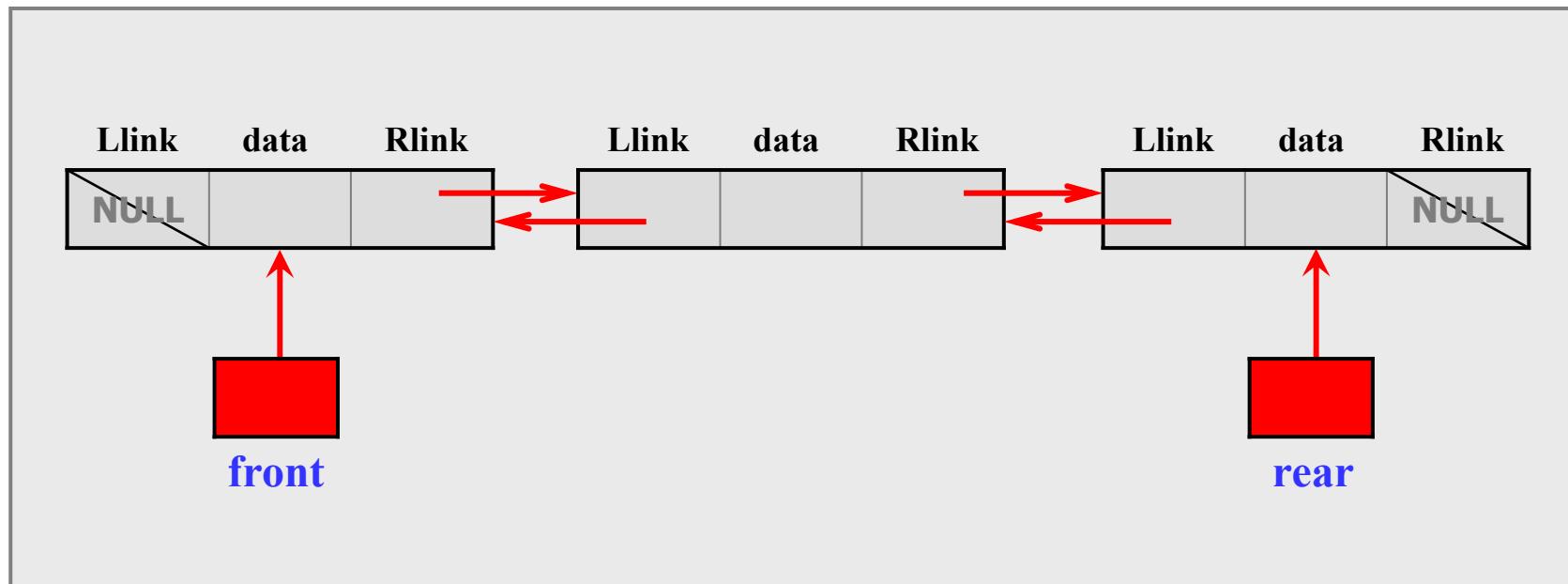
- 큐의 양쪽 끝에서 삽입과 삭제가 모두 발생할 수 있는 큐
  - 스택과 큐의 성질을 모두 가지고 있는 자료구조



# Deque (2/2)

- **Deque:** 구현

- 이중 연결 리스트 구조를 이용한 Deque의 구현

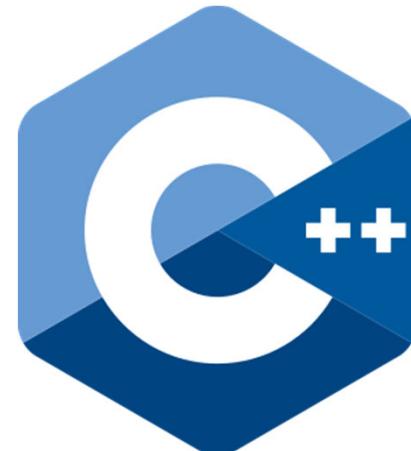


# C++ STL: deque 클래스 (1/4)

## ● 컨테이너 라이브러리(Containers Library)

### ○ 순차 컨테이너(Sequence Containers)

- <array> : 정적 연속 배열(static contiguous array) (since C++11)
- <vector> : 동적 연속 배열(dynamic contiguous array)
- <**deque**> : 덱(double-ended queue)
- <forward\_list> : 단일 연결 리스트(singly-linked list) (since C++11)
- <list> : 이중 연결 리스트(doubly-linked list)



# C++ STL: deque 클래스 (2/4)

## ● deque 클래스

### ○ 벡터와 비슷하지만 양쪽으로 요소를 추가할 수 있다.

- 앞과 뒤에 요소를 빠르게 추가하고 빠르게 제거할 수 있다.
  - 표준적으로 힙 메모리에 블록 단위로 메모리를 할당하게 구현된다.
  - **capacity** 함수와 **reserve** 멤버 함수가 없다.
- 벡터와 다르게 앞으로도 추가를 할 수 있게 앞쪽으로도 여유 메모리를 할당하기 때문에 메모리를 많이 차지한다.



그림 19-8 시퀀스 컨테이너 덱

- 참고로 **deque** 클래스는 **queue** 클래스의 베이스 클래스가 된다.

[ 이미지 출처: Behrouz A. Forouzan 외 1인, "포르잔 C++ 에센셜"(3판), 한빛아카데미, 2020 ]

# C++ STL: deque 클래스 (3/4)

예제 6-4: deque 클래스 -- 항목의 순서 회전

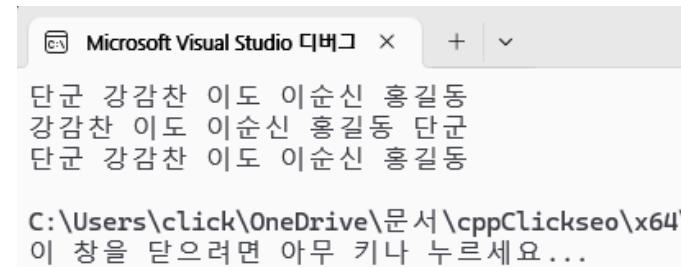
(1/2)

```
#include <iostream>
#include <iomanip>
#include <string>
#include <deque>
using namespace std;

void print(deque<string> deq);

int main(void)
{
    // deque 객체 생성
    deque<string> deQ(5);

    string str[5] = { "단군", "강감찬", "이도", "이순신", "홍길동" };
    for (int i = 0; i < 5; ++i) {
        deQ[i] = str[i];
    }
    print(deQ);
```

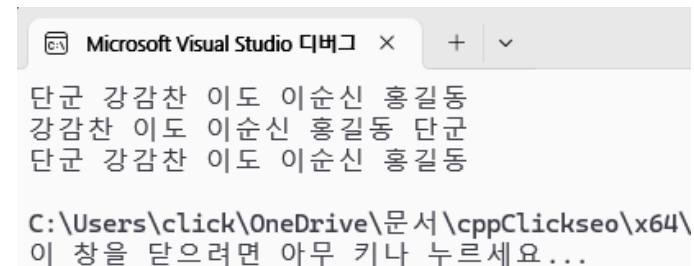


# C++ STL: deque 클래스 (4/4)

## 예제 6-4: deque 클래스 -- 항목의 순서 회전

(2/2)

```
// deque 시계 방향으로 한 요소 회전한다.  
deQ.push_back(deQ.front());  
deQ.pop_front();  
print(deQ);  
  
// deque 반대 방향으로 한 요소 회전한다.  
deQ.push_front(deQ.back());  
deQ.pop_back();  
print(deQ);  
  
return 0;  
}  
  
void print(deque <string> deq) {  
    for (int i = 0; i < deQ.size(); ++i) {  
        cout << deQ.at(i) << " ";  
    }  
    cout << endl;  
}
```





# 큐의 이해

큐 구현: 순차 자료구조

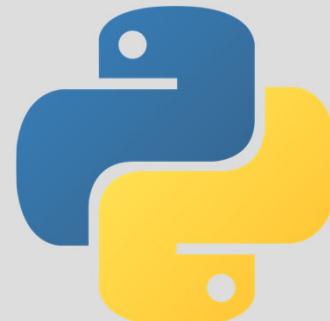
front      rear  
-1           -1



# 큐 구현: 순차 자료구조 (1/3)

## ● 큐 구현: 순차 자료구조

```
class ArrayQueue:  
    def __init__(self):  
        self.__queue = []  
  
    def __del__(self):  
  
    def empty(self) -> bool:  
    def size(self) -> int:  
    def push(self, num) -> None:  
    def pop(self) -> None:  
    def front(self):  
    def back(self):  
    def printQueue(self) -> None:
```



# 큐 구현: 순차 자료구조 (2/3)

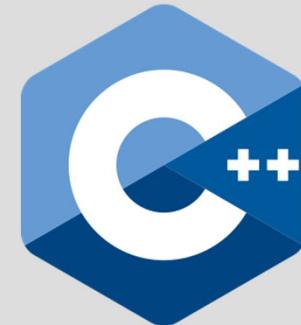
## ● 큐 구현: 순차 자료구조

```
// #pragma once
#ifndef _ArrayQueue_H_
#define _ArrayQueue_H_

// 클래스 설계: ArrayQueue
template <typename T>
class ArrayQueue {
public:
    ArrayQueue(int size = 10);                                // 빈 큐 생성
    ~ArrayQueue(void);                                         // 큐 삭제
    bool empty(void) const;                                    // 빈 큐 여부
    bool full(void) const;                                     // 포화 상태 여부
    int size(void) const;                                      // 원소 개수
    void push(const T &data);                                 // 데이터 삽입
    void pop(void);                                           // 데이터 삭제
    T front(void) const;                                       // 첫번째 노드의 데이터 반환
    T back(void) const;                                        // 마지막 노드의 데이터 반환
    void printQueue(void) const;                               // 전체 데이터 출력

private:
    int front_;
    int rear_;
    int maxSize_;
    T *queue_;
};

#endif
```



# 큐 구현: 순차 자료구조 (3/3)

## ● 큐 구현: 순차 자료구조

```
#define queueMAXSIZE 100
typedef int element;

typedef struct _ArrayQueue {
    element queue[queueMAXSIZE];
    int front;
    int rear;
} ArrayQueue;

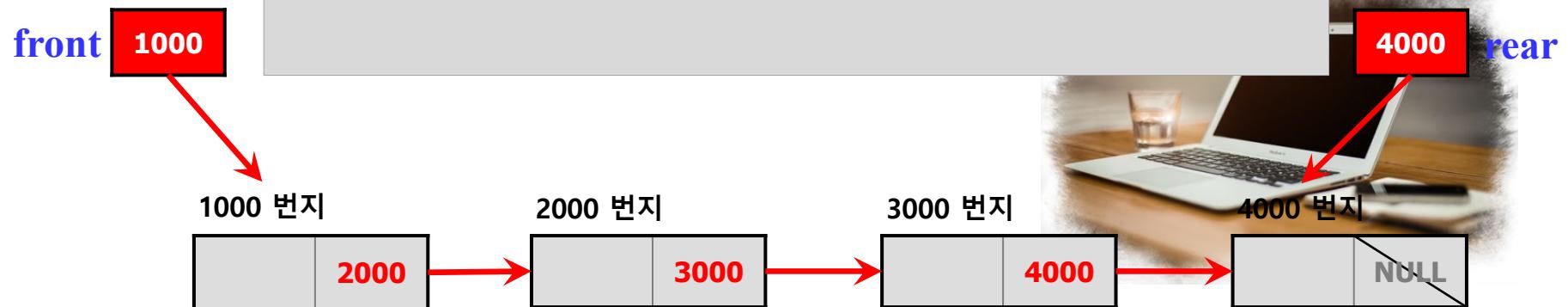
ArrayQueue *queueCreate(void);
void queueDestroy(ArrayQueue *Queue);
void enQueue(ArrayQueue *Queue, element item);
element deQueue(ArrayQueue *Queue);
element peek(ArrayQueue *Queue);
int isEmpty(ArrayQueue *Queue);
int isFull(ArrayQueue *Queue);
void queuePrint(ArrayQueue *Queue);
```





# 큐의 이해

## 큐 구현: 연결 자료구조



# 큐 구현: 연결 자료 구조 (1/3)

## ● 큐 구현: 연결 자료구조

```
# 클래스 설계: LinkedQueue
```

```
class LinkedQueue:
```

```
    class SNode:
```

```
        def __init__(self, data, link=None):
```

```
            self.data = data
```

```
            self.link = link
```

```
# 빈 큐 생성
```

```
def __init__(self):
```

```
    self.__front = None
```

```
    self.__rear = None
```

```
    self.__count = 0
```

```
def __del__(self):
```

```
def empty(self) -> bool:
```

```
def size(self) -> int:
```

```
def push(self, data) -> None:
```

```
def pop(self) -> None :
```

```
def front(self):
```

```
def back(self):
```

```
def printQueue(self) -> None :
```

```
# 큐 삭제
```

```
# 빈 큐 여부
```

```
# 큐의 원소 개수
```

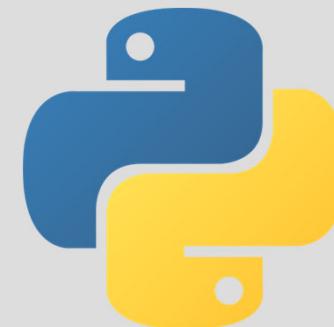
```
# 데이터 삽입
```

```
# 데이터 삭제
```

```
# 큐에서 첫 번째 데이터 반환
```

```
# 큐에서 맨 마지막 데이터 반환
```

```
# 큐의 전체 데이터 출력
```



# 큐 구현: 연결 자료구조 (2/3)

## ● 큐 구현: 연결 자료구조

```
// LinkedQueue.cpp
#include "LinkedNode(template).cpp"

// #pragma once
#ifndef _LinkedQueue_Template_H_
#define _LinkedQueue_Template_H_

// 클래스 설계: LinkedQueue
template <typename T>
class LinkedQueue {
    LinkedQueue();           // 빈 큐 생성
    ~LinkedQueue();          // 큐 삭제
    bool   empty(void) const; // 빈 큐 여부
    int    size(void) const;  // 원소 개수
    void   push(const T &data); // 데이터 삽입
    void   pop(void);        // 데이터 삭제
    T     front(void) const; // 첫번째 노드의 데이터 반환
    T     back(void) const;  // 마지막 노드의 데이터 반환
    void   printQueue(void) const; // 큐의 전체 데이터 출력

private:
    SNode<T>      *front_;
    SNode<T>      *rear_;
    int            count_;
};

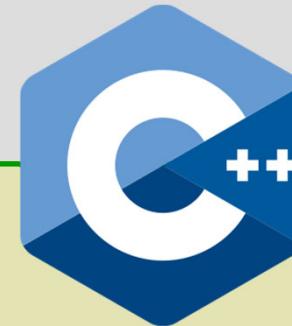
#endif
```

```
// LinkedNode(template).cpp
// #pragma once
#ifndef _SNode_Template_H_
#define _SNode_Template_H_

template <typename T> class LinkedQueue;

// 클래스 설계: SNode(data, link)
template <typename T>
class SNode {
public:
    SNode(const T &data);
private:
    T      data;
    SNode<T> *link;
};

template <typename T> friend class LinkedQueue;
```



# 큐 구현: 연결 자료 구조 (3/3)

## ● 큐 구현: 연결 자료구조

```
// #pragma once
#include "LinkedSNode.h"           // SNode
typedef int element;

// 큐 생성: LinkedQueue
#ifndef _LinkdedQueue_H_
#define _LinkdedQueue H

typedef struct _LinkedQueue {
    SNode *front;
    SNode *rear;
    int count;
} LinkedQueue;
#endif

// 큐 구현(C): 큐 생성 및 활용
LinkedQueue* queueCreate(void);
void queueDestroy(LinkedQueue *Queue);
void enqueue(LinkedQueue *Queue, element data);
void dequeue(LinkedQueue *Queue);
element front(LinkedQueue *Queue);
element back(LinkedQueue *Queue);
_Bool queueEmpty(LinkedQueue *Queue);
int queueSize(LinkedQueue *Queue);
void printQueue(LinkedQueue *Queue);
```



# 큐 구현



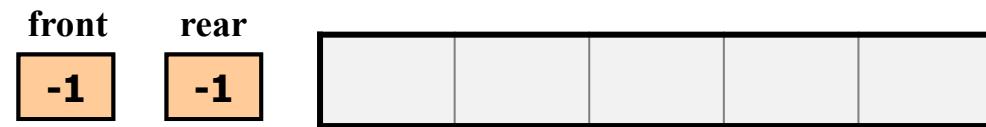
백문이불여일타(百聞而不如一打)

- 큐의 이해

- 큐 구현

- 순차 자료 구조

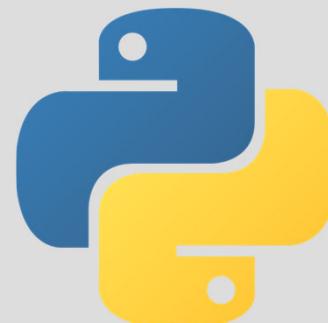
- 연결 자료 구조



# 큐 구현(Python): 순차 자료 구조 (1/7)

## ● 큐 구현: 순차 자료구조

```
class ArrayQueue:  
    def __init__(self):  
        self.__queue = []  
  
    def __del__(self):  
  
    def empty(self) -> bool:  
    def size(self) -> int:  
    def push(self, num) -> None:  
    def pop(self) -> None:  
    def front(self):  
    def back(self):  
    def printQueue(self) -> None:
```



# 큐 구현(Python): 순차 자료 구조 (2/7)

## ● 큐 구현: 순차 자료구조

### ○ 프로그램 실행 결과는 다음과 같다.

```
### 큐 구현: 1차원 배열 ###
1) 데이터 삽입: push
2) 데이터 삭제: pop
3) 전체 출력
4) 프로그램 종료
```

메뉴 선택: 3

Queue [ 1 2 3 4 5 ]

```
### 큐 구현: 1차원 배열 ###
1) 데이터 삽입: push
2) 데이터 삭제: pop
3) 전체 출력
4) 프로그램 종료
```

메뉴 선택: 2

삭제 된 데이터: 1

front  
-1

rear  
-1



The screenshot shows the Python IDLE Shell interface. The code runs a queue implementation. It starts with menu options, then performs multiple insertions ('push') followed by a deletion ('pop'), and finally prints the entire queue. A green box highlights the menu selection loop and the final print statement.

```
*IDLE Shell 3.13.2*
File Edit Shell Debug Options Window Help
Python 3.13.2 (tags/v3.13.2:4f8bb39, Feb 4, 2024, 15:17:05)
Type "help", "copyright", "credits" or "license" for more information
>>>
=====
RESTART: C:\Users\click\Documents\queue.py

### 큐 구현: 1차원 배열 ###
1) 데이터 삽입: push
2) 데이터 삭제: pop
3) 전체 출력
4) 프로그램 종료

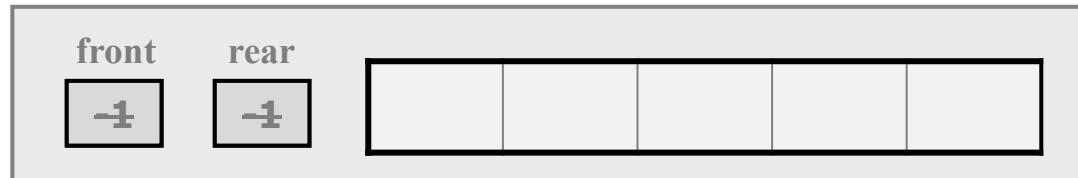
메뉴 선택: 1
삽입 할 데이터 입력(종료: 0): 1
삽입 할 데이터 입력(종료: 0): 2
삽입 할 데이터 입력(종료: 0): 3
삽입 할 데이터 입력(종료: 0): 4
삽입 할 데이터 입력(종료: 0): 5
삽입 할 데이터 입력(종료: 0): 0
```

# 큐 구현(Python): 순차 자료구조 (3/7)

예제 6-5: 큐 -- 순차 자료구조

ArrayQueue.py (1/5)

```
class ArrayQueue:  
    # 빈 큐 생성  
    def __init__(self):  
        self.__queue = []  
  
    # 큐 삭제  
    def __del__(self):  
        self.__queue.clear()  
  
    # 빈 큐 여부  
    def empty(self) -> bool:  
        if not self.__queue:  
            return True  
        return False  
  
    # 큐의 원소 개수  
    def size(self) -> int:  
        return len(self.__queue)
```



# 큐 구현(Python): 순차 자료구조 (4/7)

예제 6-5: 큐 -- 순차 자료구조

ArrayQueue.py

(2/5)

# 데이터 삽입: 맨 마지막으로 새로운 데이터 추가

```
def push(self, num) -> None:  
    self.__queue.append(num)
```

# 데이터 삭제: 첫 번째 데이터 삭제

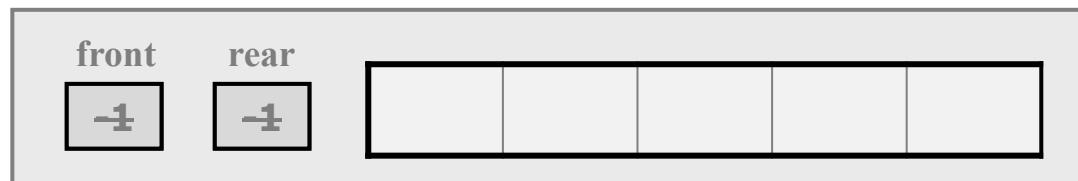
```
def pop(self) -> None:  
    if self.empty():  
        return  
    self.__queue.pop(0)
```

# 큐에서 첫 번째 데이터 확인

```
def front(self):  
    if not self.__queue:  
        return None  
    return self.__queue[0]
```

# 큐에서 맨 마지막 데이터 확인

```
def back(self):  
    if not self.__queue:  
        return None  
    return self.__queue[-1]
```



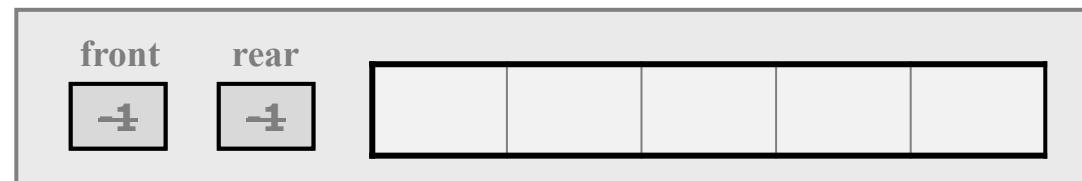
# 큐 구현(Python): 순차 자료 구조 (5/7)

예제 6-5: 큐 -- 순차 자료구조

ArrayQueue.py (3/5)

```
# 큐의 전체 원소 출력
def printQueue(self) -> None:
    if self.empty():
        print('입력된 데이터가 없습니다!!!!')
        return

    print('\n Queue [ ', end = ' ')
    for i in range(len(self.__queue)):
        print(self.__queue[i], end = ' ')
    print(' ]\n')
```



# 큐 구현(Python): 순차 자료 구조 (6/7)

예제 6-5: 큐 -- 순차 자료구조

ArrayQueue.py (4/5)

```
if __name__ == '__main__':
    import os          # system
    import sys         # exit

    q = ArrayQueue()
    while (True):
        os.system('cls')
        print('\n ### 큐 구현: 1차원 배열 ###')
        print('1) 데이터 삽입: push')
        print('2) 데이터 삭제: pop')
        print('3) 전체 출력')
        print('4) 프로그램 종료\n')
        print('메뉴 선택: ', end=' ')
        choice = int(input())
```

The screenshot shows a code editor window with the file '06\_ArrayQueue.py' open. The code defines a class 'ArrayQueue' with methods for push, pop, and printing the queue. Below the code editor is a terminal window showing the program's execution. The terminal output includes menu options, user inputs for pushing values 1 through 5, and a final message asking if the user wants to continue.

```
06_ArrayQueue.py X
02.(DS)_자료구조및알고리즘 > 04.자료구조및알고리즘 > 02.0

12     class ArrayQueue:
13         # 빈 큐 생성
14         def __init__(self):
15             self.__queue = []
16

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

### 큐 구현: 1차원 배열 ###
1) 데이터 삽입: push
2) 데이터 삭제: pop
3) 전체 출력
4) 프로그램 종료

메뉴 선택: 1
삽입 할 데이터 입력(종료: 0): 1
삽입 할 데이터 입력(종료: 0): 2
삽입 할 데이터 입력(종료: 0): 3
삽입 할 데이터 입력(종료: 0): 4
삽입 할 데이터 입력(종료: 0): 5
삽입 할 데이터 입력(종료: 0): 0
계속하려면 아무 키나 누르십시오 . . .
```

# 큐 구현(Python): 순차 자료 구조 (7/7)

예제 6-5: 큐 -- 순차 자료구조

ArrayQueue.py (5/5)

```
match choice:
    case 1:
        while True:
            num = int(input('삽입 할 데이터 입력(종료: 0): '))
            if num == 0:
                break
            q.push(num)
    case 2:
        if not q.empty():
            print(f'\n삭제 된 데이터: {q.front()}')
            q.pop()
    case 3:
        q.printQueue()
    case 4:
        sys.exit("Wn프로그램 종료!!!")
    case _:
        print('\n잘못 선택 하셨습니다. \n')
os.system('pause')

# del s
# s.__del__
```

```
06_ArrayQueue.py X
02.(DS)_자료구조및알고리즘 > 04.자료구조및알고리즘 > 02.0
12     class ArrayQueue:
13         # 빈 큐 생성
14         def __init__(self):
15             self.__queue = []
16
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
### 큐 구현: 1차원 배열 ####
1) 데이터 삽입: push
2) 데이터 삭제: pop
3) 전체 출력
4) 프로그램 종료
메뉴 선택: 1
삽입 할 데이터 입력(종료: 0): 1
삽입 할 데이터 입력(종료: 0): 2
삽입 할 데이터 입력(종료: 0): 3
삽입 할 데이터 입력(종료: 0): 4
삽입 할 데이터 입력(종료: 0): 5
삽입 할 데이터 입력(종료: 0): 0
계속하려면 아무 키나 누르십시오 . . . []
```

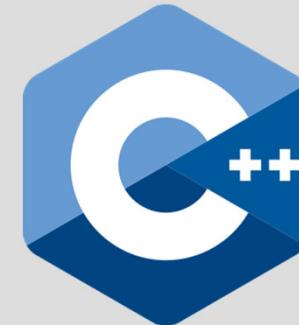
# 큐 구현(C++): 순차 자료구조 (1/10)

## ● 큐 구현: 순차 자료구조

```
// #pragma once
#ifndef _ArrayQueue_H_
#define _ArrayQueue_H_

// 클래스 설계: ArrayQueue
template <typename T>
class ArrayQueue {
public:
    ArrayQueue(int size = 10);                                // 빈 큐 생성
    ~ArrayQueue(void);                                         // 큐 삭제
    bool empty(void) const;                                    // 빈 큐 여부
    bool full(void) const;                                     // 포화 상태 여부
    int size(void) const;                                      // 원소 개수
    void push(const T &data);                                 // 데이터 삽입
    void pop(void);                                           // 데이터 삭제
    T front(void) const;                                       // 첫번째 노드의 데이터 반환
    T back(void) const;                                        // 마지막 노드의 데이터 반환
    void printQueue(void) const;                               // 전체 데이터 출력

private:
    int front_;                                              front
    int rear_;                                                 rear
    int maxSize_;
    T *queue_;                                                 [ ]
```



# 큐 구현(C++): 순차 자료 구조 (2/10)

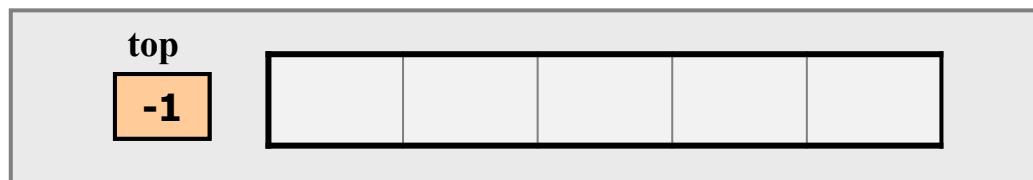
## ● 큐 구현: 순차 자료구조

- 프로그램 실행 결과는 다음과 같다.

```
## 스택 구현: 1차원 배열 ##

1) 데이터 삽입: push
2) 데이터 삭제: pop
3) 데이터 전체 출력
4) 프로그램 종료

메뉴 선택: 1
데이터 입력(종료: 0): 1
데이터 입력(종료: 0): 2
데이터 입력(종료: 0): 3
데이터 입력(종료: 0): 4
데이터 입력(종료: 0): 5
데이터 입력(종료: 0): 0
계속하려면 아무 키나 누르십시오 . . .
```



```
## 스택 구현: 1차원 배열 ##

1) 데이터 삽입: push
2) 데이터 삭제: pop
3) 데이터 전체 출력
4) 프로그램 종료

메뉴 선택: 3
##### 입력된 데이터 #####
STACK [ 5 4 3 2 1 ]
계속하려면 아무 키나 누르십시오 . . .
```

```
## 스택 구현: 1차원 배열 ##

1) 데이터 삽입: push
2) 데이터 삭제: pop
3) 데이터 전체 출력
4) 프로그램 종료

메뉴 선택: 2
삭제 된 데이터: 5
계속하려면 아무 키나 누르십시오 . . .
```

# 큐 구현(C++): 순차 자료구조 (3/10)

예제 6-6: 큐 -- 순차 자료구조

ArrayQueue(demo).cpp (1/2)

```
#include <iostream>
#include "ArrayQueue.cpp"           // ArrayQueue
// #include <conio.h>              // _getch, _getche
using namespace std;

int main(void)
{
    int num, choice;
    ArrayQueue<int> q = ArrayQueue<int>();

    while (true) {
        system("cls");
        cout << "\n\t### 큐 구현: 1차원 배열 ### \n\n" << endl;
        cout << "1) 데이터 삽입: push" << endl;
        cout << "2) 데이터 삭제: pop" << endl;
        cout << "3) 전체 출력" << endl;
        cout << "4) 프로그램 종료 \n" << endl;
        cout << "메뉴 선택: ";
        cin >> choice;
```

# 큐 구현(C++): 순차 자료 구조 (4/10)

예제 6-6: 큐 -- 순차 자료구조

ArrayQueue(demo).cpp (2/2)

```
switch (choice) {
    case 1:
        while (true) {
            cout << "데이터 입력(종료: 0): ";
            cin >> num;
            if (num == 0)
                break;
            q.push(num);
        }
        break;
    case 2: cout << "삭제 된 데이터: " << q.front() << endl;
        q.pop();
        break;
    case 3: q.printQueue();
        break;
    case 4: cout << "프로그램 종료!!!" << endl;
        exit(0); // return 0;
    default: cout << "잘못 선택 하셨습니다!!!" << endl;
}
// print("계속하려면 아무 키나 누르십시오...") ;
// _getch();
system("pause");
}
// s.~ArrayQueue();
return 0;
}
```

# 큐 구현(C++): 순차 자료구조 (5/10)

예제 6-6: 큐 -- 순차 자료구조

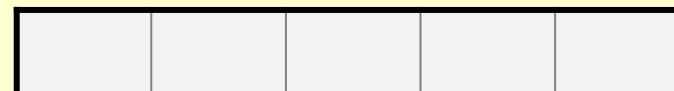
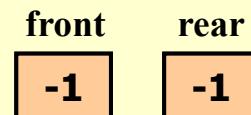
ArrayQueue.cpp (1/6)

```
#include <iostream>
using namespace std;

// #pragma once
#ifndef __ArrayQueue_H__
#define __ArrayQueue_H__

// 클래스 설계: ArrayQueue
template <typename T>
class ArrayQueue {
public:
    ArrayQueue(int size = 10); // 빈 큐 생성
    ~ArrayQueue(void); // 큐 삭제
    bool empty(void) const; // 큐 여부
    bool full(void) const; // 큐 상태 여부
    int size(void) const; // 원소 개수
    void push(const T &data); // 데이터 삽입
    void pop(void); // 데이터 삭제
    T front(void) const; // 첫번째 데이터 반환
    T rear(void) const; // 마지막 데이터 반환
    void printQueue(void) const; // 전체 데이터 출력

private:
    int front_; // front_
    int rear_; // rear_
    int maxSize_; // maxSize_
    T *queue_; // queue_
};
```



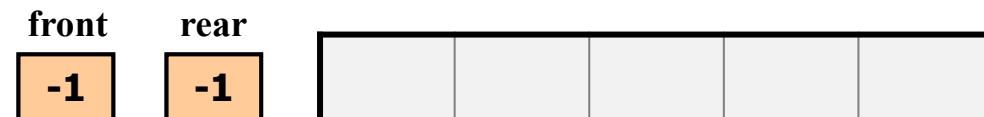
# 큐 구현(C++): 순차 자료구조 (6/10)

예제 6-6: 큐 -- 순차 자료구조

ArrayQueue.cpp (2/6)

```
// 빈 큐 생성
template <typename T>
ArrayQueue<T>::ArrayQueue(int size)
    : front_(-1), rear_(-1), maxSize_(size) {
    queue_ = new T[maxSize_];
}
```

```
// 큐 삭제
template <typename T>
ArrayQueue<T>::~ArrayQueue(void) {
    delete[] queue_;
}
```



# 큐 구현(C++): 순차 자료구조 (7/10)

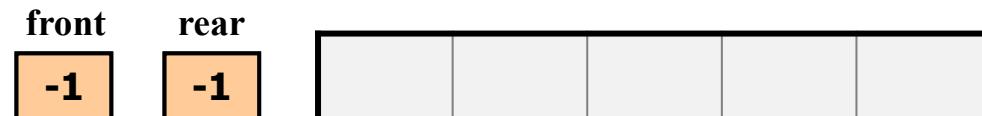
예제 6-6: 큐 -- 순차 자료구조

ArrayQueue.cpp (3/6)

```
// 빈 큐 여부
template <typename T>
bool ArrayQueue<T>::empty(void) const {
    return front_ == rear_;
}

// 포화 상태 여부
template <typename T>
bool ArrayQueue<T>::full(void) const {
    return rear_ == maxSize_ - 1;
}

// 원소 개수
template <typename T>
int ArrayQueue<T>::size(void) const {
    return rear_ - front_ + 1;
}
```



# 큐 구현(C++): 순차 자료구조 (8/10)

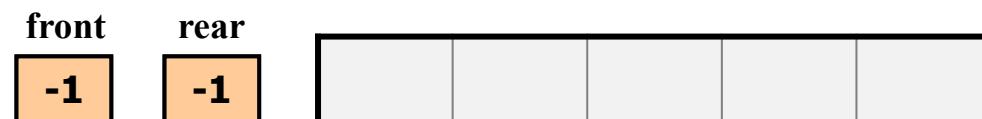
예제 6-6: 큐 -- 순차 자료구조

ArrayQueue.cpp (4/6)

```
// 데이터 삽입: 큐에 새로운 데이터 추가
template <typename T>
void ArrayQueue<T>::push(const T &data) {
    if (full())
        return;
    queue_[++rear_] = data;
}
```

```
// 데이터 삭제: 큐에서 맨 위의 데이터 삭제
```

```
template <typename T>
void ArrayQueue<T>::pop(void) {
    if (empty())
        return;
    ++front_;
```



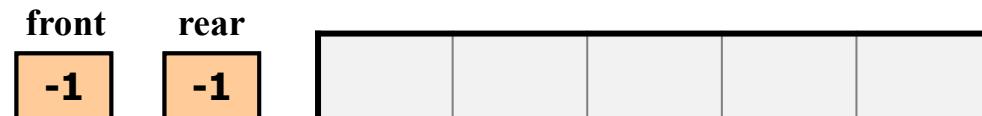
# 큐 구현(C++): 순차 자료구조 (9/10)

예제 6-6: 큐 -- 순차 자료구조

ArrayQueue.cpp (5/6)

```
// 첫번째 데이터 반환
template <typename T>
T ArrayQueue<T>::front(void) const {
    if (empty()) {
        return T();
    }
    return queue_[front_ + 1];
}

// 마지막 데이터 반환
template <typename T>
T ArrayQueue<T>::back(void) const {
    if (empty()) {
        return T();
    }
    return queue_[rear_];
}
```



# 큐 구현(C++): 순차 자료구조 (10/10)

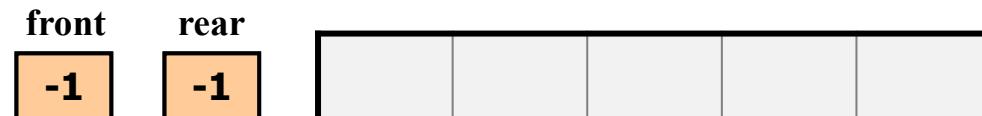
예제 6-6: 큐 -- 순차 자료구조

ArrayQueue.cpp (6/6)

```
// 전체 데이터 출력
template <typename T>
void ArrayQueue<T>::printQueue(void) const {
    if (empty()) {
        cout << "입력된 데이터가 없습니다!!!!" << endl;
        return;
    }

    cout << "\n\t##### 입력된 데이터 #####\n" << endl;
    cout << "QUEUE [ ";
    for (int i = front_ + 1; i <= rear_; ++i) {
        cout.width(3);
        cout << queue_[i];
    }
    cout << " ]" << endl;
}

#endif
```



# 큐 구현(C): 순차 자료 구조 (1/5)

## ● 큐 구현: 순차 자료구조

```
#define queueMAXSIZE 100
typedef int element;

typedef struct _ArrayQueue {
    element queue[queueMAXSIZE];
    int front;
    int rear;
} ArrayQueue;
```

```
ArrayQueue *queueCreate(void);
void queueDestroy(ArrayQueue *Queue);
void enQueue(ArrayQueue *Queue, element item);
element deQueue(ArrayQueue *Queue);
element peek(ArrayQueue *Queue);
int isEmpty(ArrayQueue *Queue);
int isFull(ArrayQueue *Queue);
void queuePrint(ArrayQueue *Queue);
```



# 큐 구현(C): 순차 자료 구조 (2/5)

예제 6-7: 큐 구현 -- 순차 자료 구조

(1/4)

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>           // bool, true, false
#define queueMAXSIZE 100

typedef int element;
typedef struct _ArrayQueue {
    element      queue[queueMAXSIZE];
    int         front;
    int         rear;
} ArrayQueue;

ArrayQueue *queueCreate(void);
void        queueDestroy(ArrayQueue*);
void        enqueue(ArrayQueue *Q, element item);
element    dequeue(ArrayQueue *Q);
element    peek(ArrayQueue *Q);
int         isEmpty(ArrayQueue *Q);
int         isFull(ArrayQueue *Q);
void        queuePrint(ArrayQueue *Q);
```

# 큐 구현(C): 순차 자료 구조 (3/5)

예제 6-7: 큐 구현 -- 순차 자료 구조

(2/4)

```
int main(void)
{
    int num, data;
    arrayQueue *q = queueCreate();

    while (true) {
        system("cls");
        printf("\n ### 큐 구현: 1차원 배열 ### \n\n");
        printf("1) 데이터 삽입(enQueue) : \n");
        printf("2) 데이터 삭제(deQueue) : \n");
        printf("3) 전체 출력\n");
        printf("4) 프로그램 종료 \n\n");
        printf("메뉴 선택: ");
        scanf_s("%d", &num);
        // scanf("%d", &num);

        switch (num) {
            case 1: printf("\n 삽입 할 데이터 입력 : ");
                      scanf_s("%d", &data); // scanf("%d", &data);
                      enqueue(q, data);
                      break;
            case 2: printf("삭제 된 데이터 : %3d \n", dequeue(q));
                      break;
            case 3: queuePrint(q);
                      break;
            case 4: printf("프로그램 종료... \n");
                      return 0;
            default: printf("잘못 선택 하셨습니다. \n");
        }
        system("pause");
    }
    queueDestroy(q);
    return 0;
}
```

# 큐 구현(C): 순차 자료 구조 (4/5)

예제 6-7: 큐 구현 -- 순차 자료 구조

(3/4)

```
arrayQueue *queueCreate(void) {
    arrayQueue *Q;
    Q = (arrayQueue*)malloc(sizeof(arrayQueue));
    if (Q == NULL) {
        printf("Queue 생성 실패!!!\n");
        return NULL;
    }
    Q->front = -1;
    Q->rear = -1;
    return Q;
}

void queueDestroy(arrayQueue *Q) {
    free(Q);
}

void enqueue(arrayQueue *Q, element item) {
    if (Q->rear + 1 >= queueMAXSIZE)
        return;
    Q->queue[++Q->rear] = item;
}

element dequeue(arrayQueue *Q) {
    if (Q->front == Q->rear)
        return -1;
    return Q->queue[++Q->front];
}
```

# 큐 구현(C): 순차 자료 구조 (5/5)

예제 6-7: 큐 구현 -- 순차 자료 구조

(4/4)

```
element      peek(arrayQueue *Q) {
    if (Q->front == -1)
        return -1;

    return Q->queue[Q->front + 1];
}

int         isEmpty(arrayQueue *Q) {
    if (Q->front == Q->rear)
        return 1;
    return 0;
}

int         isFull(arrayQueue *Q) {
    if (Q->rear == queueMAXSIZE - 1)
        return 1;
    return 0;
}

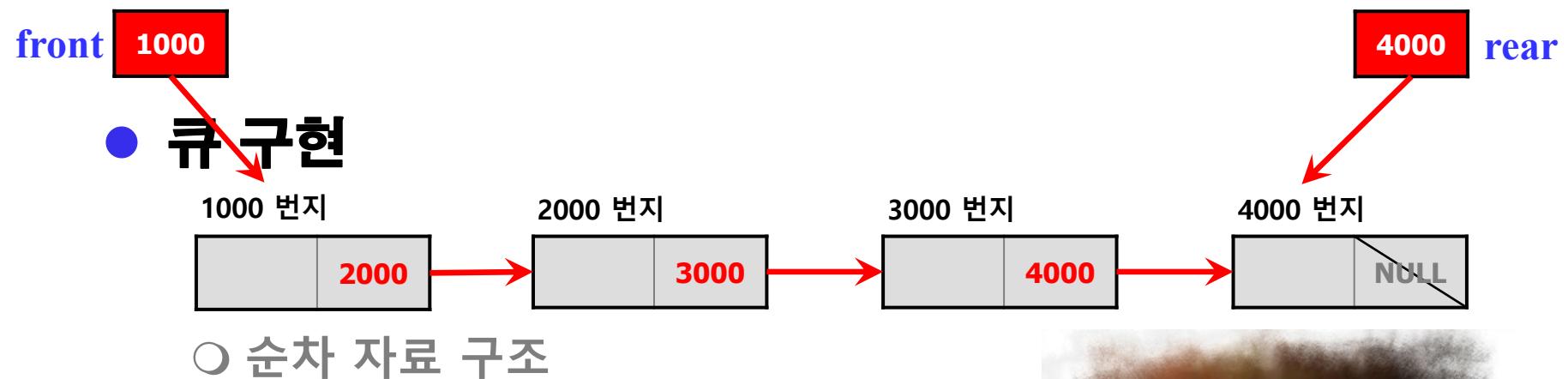
void        queuePrint(arrayQueue *Q) {
printf("\n Queue : [");
for (int i = Q->front + 1; i <= Q->rear; i++)
    printf("%3d", Q->queue[i]);
printf(" ] \n");
}
```

# 큐 구현



백문이불여일타(百聞而不如一打)

## ● 큐의 이해



## ○ 연결 자료 구조

# 큐 구현(Python): 연결 자료 구조 (1/5)

## ● 큐 구현: 연결 자료구조

```
# 클래스 설계: LinkedQueue
```

```
class LinkedQueue:
```

```
    class SNode:
```

```
        def __init__(self, data, link=None):
            self.data = data
            self.link = link
```

```
# 빈 큐 생성
```

```
def __init__(self):
    self.__front = None
    self.__rear = None
    self.__count = 0
```

```
def __del__:
```

```
def empty(self) -> bool:
```

```
def size(self) -> int:
```

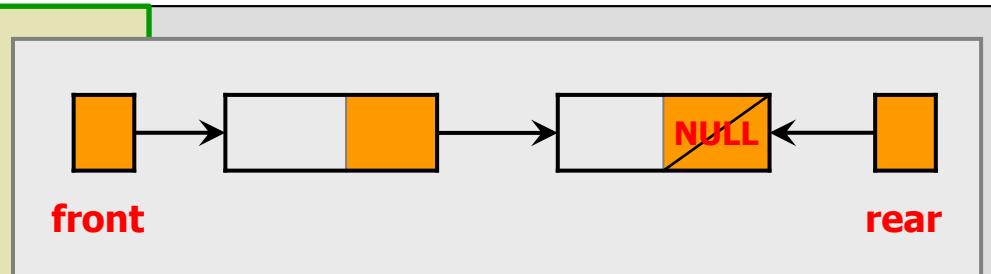
```
def push(self, data) -> None:
```

```
def pop(self) -> None:
```

```
def front(self):
```

```
def back(self):
```

```
def printQueue(self):
```



# 큐 삭제

# 빈 큐 여부

# 큐의 원소 개수

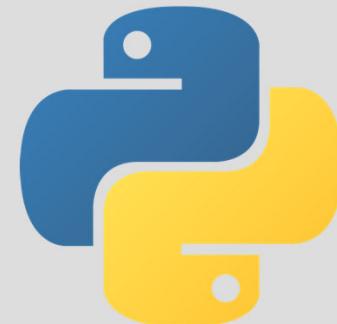
# 데이터 삽입

# 데이터 삭제

# 큐에서 첫 번째 데이터 반환

# 큐에서 맨 마지막 데이터 반환

# 큐의 전체 데이터 출력



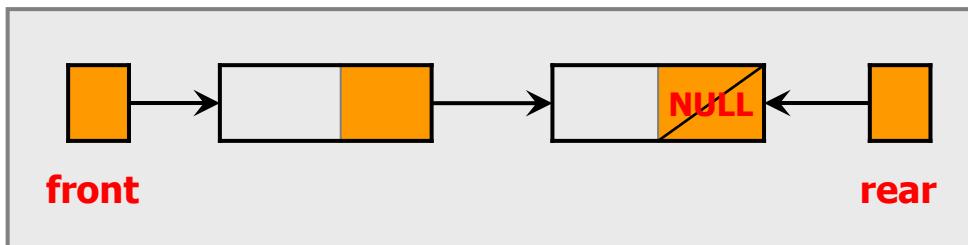
# 큐 구현(Python): 연결 자료 구조 (2/5)

## ● 큐 구현: 연결 자료구조

- 프로그램 실행 결과는 다음과 같다.

```
### 큐 구현: 단순연결리스트 ###
1) 데이터 삽입: push
2) 데이터 삭제: pop
3) 전체 출력
4) 프로그램 종료

메뉴 선택: 3
##### 입력된 데이터 #####
QUEUE [ 1 2 3 4 5 ]
```



```
*IDLE Shell 3.11.2*
File Edit Shell Debug Options Window Help
Python 3.11.2 (tags/v3.11.2:878ead1, Feb
Type "help", "copyright", "credits" or "
>>> ===== RESTART: C:\Users\Click

### 큐 구현: 단순연결리스트 ###
1) 데이터 삽입: push
2) 데이터 삭제: pop
3) 전체 출력
4) 프로그램 종료

메뉴 선택: 1
삽입할 데이터 입력(종료: 0): 1
삽입할 데이터 입력(종료: 0): 2
삽입할 데이터 입력(종료: 0): 3
삽입할 데이터 입력(종료: 0): 4
삽입할 데이터 입력(종료: 0): 5
삽입할 데이터 입력(종료: 0): 0

C:\WINDOWS\system: X + v - □ ×
계속 하려면 아무 키나 누르십시오 . . .
```

```
### 큐 구현: 단순연결리스트 ###
1) 데이터 삽입: push
2) 데이터 삭제: pop
3) 전체 출력
4) 프로그램 종료

메뉴 선택: 2
삭제 된 데이터: 1
```

# 큐 구현(Python): 연결 자료 구조 (3/5)

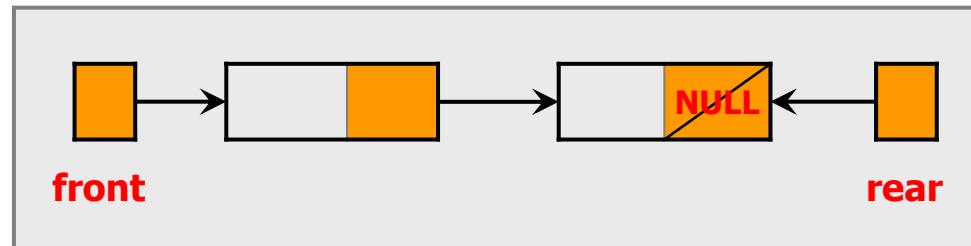
예제 6-8: 큐 -- 연결 자료구조

LinkedQueue.py (1/3)

```
# 클래스 설계: LinkedQueue
class LinkedQueue:
    class SNode:
        def __init__(self, data, link=None):
            self.data = data
            self.link = link

    # 빈 큐 생성
    def __init__(self):
        self.__front = None
        self.__rear = None
        self.__count = 0

    def __del__(self):
    def empty(self) -> bool:
    def size(self) -> int:
    def push(self, data) -> None:
    def pop(self) -> None:
    def front(self):
    def back(self):
    def printQueue(self):
```



```
# 큐 삭제
# 빈 큐 여부
# 큐의 원소 개수
# 데이터 삽입
# 데이터 삭제
# 큐에서 첫 번째 데이터 반환
# 큐에서 마지막 데이터 반환
# 큐의 전체 데이터 출력
```

## 큐 구현(Python): 연결 자료 구조 (4/5)

### 예제 6-8: 큐 -- 연결 자료구조

LinkedQueue.py (2/3)

```
if __name__ == '__main__':
    import os           # system
    import sys          # exit

    # "SNode" is not defined
    # tNode = SNode(10, None)
    q = LinkedQueue()

    while True:
        os.system('cls')
        print('\n ### 큐 구현: 단순연결리스트 ###')
        print('1) 데이터 삽입: push')
        print('2) 데이터 삭제: pop')
        print('3) 전체 출력')
        print('4) 프로그램 종료\n')
        print('메뉴 선택: ', end=' ')
        choice = int(input())
        if choice == 1:
            data = int(input('데이터를 입력하세요: '))
            q.push(data)
            print(f'{data}가 큐에 추가되었습니다.')
        elif choice == 2:
            data = q.pop()
            if data is None:
                print('큐가 비었습니다.')
            else:
                print(f'{data}가 큐에서 제거되었습니다.')
        elif choice == 3:
            print(q)
        elif choice == 4:
            print('프로그램을 종료합니다.')
            break
        else:
            print('잘못된 선택입니다. 다시 시도해주세요.')
```

```
 알고리즘 > 04. 자료 구조 & 알고리즘 > 02.(예제)_연습문제 > 01.(초판)_202409  
 12  # LinkedQueue class: SNode, front, rear, count  
 13  class LinkedQueue:  
 14      class SNode:  
 15          def __init__(self, data, link=None):  
 16              self.data = data  
 17              self.link = link  
  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS  
  
### 큐 구현: 단순연결리스트 ###  
1) 데이터 삽입: push  
2) 데이터 삭제: pop  
3) 전체 출력  
4) 프로그램 종료  
  
메뉴 선택: 1  
삽입 할 데이터 입력(종료: 0): 1  
삽입 할 데이터 입력(종료: 0): 2  
삽입 할 데이터 입력(종료: 0): 3  
삽입 할 데이터 입력(종료: 0): 4  
삽입 할 데이터 입력(종료: 0): 5  
삽입 할 데이터 입력(종료: 0): 0  
계속하려면 아무 키나 누르십시오 . . . □
```

# 큐 구현(Python): 연결 자료 구조 (5/5)

예제 6-8: 큐 -- 연결 자료구조

LinkedQueue.py (3/3)

```
match choice:
    case 1:
        while True:
            num = int(input('삽입 할 데이터 입력(종료: 0): '))
            if num == 0:
                break
            q.push(num)
    case 2:
        print(f'\n삭제 된 데이터: {q.front()}')
        q.pop()
    case 3:
        q.printQueue()
    case 4:
        sys.exit("프로그램 종료!!!")
    case _: print('\n잘못 선택 하셨습니다. \n')
os.system('pause')

# del s
# s.__del__
```

06\_LinkedQueue.py x  
알고리즘 > 04.자료구조&알고리즘 > 02.(예제)\_연습문제 > 01.(초판)\_202409

```
12 # LinkedQueue class: SNode, front, rear, count
13 class LinkedQueue:
14     class SNode:
15         def __init__(self, data, link=None):
16             self.data = data
17             self.link = link
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

### 큐 구현: 단순연결리스트 ###
1) 데이터 삽입: push
2) 데이터 삭제: pop
3) 전체 출력
4) 프로그램 종료

메뉴 선택: 1  
삽입 할 데이터 입력(종료: 0): 1  
삽입 할 데이터 입력(종료: 0): 2  
삽입 할 데이터 입력(종료: 0): 3  
삽입 할 데이터 입력(종료: 0): 4  
삽입 할 데이터 입력(종료: 0): 5  
삽입 할 데이터 입력(종료: 0): 0  
계속하려면 아무 키나 누르십시오 . . . []

# 큐 구현(C++): 연결 자료 구조 (1/7)

## ● 큐 구현: 연결 자료구조

```
// LinkedQueue.cpp
#include "LinkedNode(template).cpp"

// #pragma once
#ifndef _LinkedQueue_Template_H_
#define _LinkedQueue_Template_H_

// 클래스 설계: LinkedQueue
template <typename T>
class LinkedQueue {
    LinkedQueue();           // 빈 큐 생성
    ~LinkedQueue();          // 큐 삭제
    bool    empty(void) const; // 빈 큐 여부
    int     size(void) const; // 원소 개수
    void    push(const T &data); // 데이터 삽입
    void    pop(void);        // 데이터 삭제
    T      front(void) const; // 첫번째 노드의 데이터 반환
    T      back(void) const; // 마지막 노드의 데이터 반환
    void   printQueue(void) const; // 큐의 전체 데이터 출력

private:
    SNode<T>      *front_;
    SNode<T>      *rear_;
    int            count_;
};

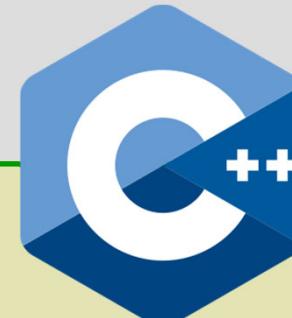
#endif
```

```
// LinkedNode(template).cpp
// #pragma once
#ifndef _SNode_Template_H_
#define _SNode_Template_H_

template <typename T> class LinkedQueue;

// 클래스 설계: SNode(data, Llink, Rlink)
template <typename T>
class SNode {
public:
    SNode(const T &data);
private:
    T      data;
    SNode<T> *link;
};

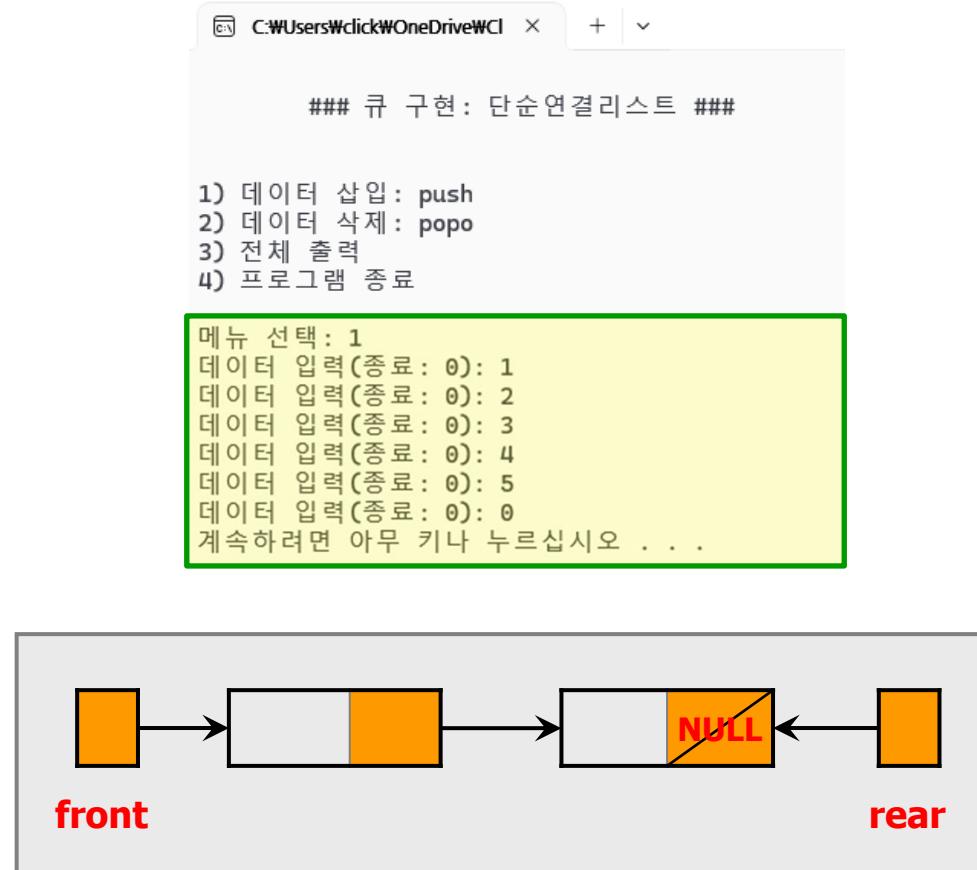
template <typename T> friend class LinkedQueue;
```



# 큐 구현(C++): 연결 자료 구조 (2/7)

## ● 큐 구현: 연결 자료구조

### ○ 프로그램 실행 결과는 다음과 같다.



- 1) 데이터 삽입: push
- 2) 데이터 삭제: popo
- 3) 전체 출력
- 4) 프로그램 종료

```
메뉴 선택: 1  
데이터 입력(종료: 0): 1  
데이터 입력(종료: 0): 2  
데이터 입력(종료: 0): 3  
데이터 입력(종료: 0): 4  
데이터 입력(종료: 0): 5  
데이터 입력(종료: 0): 0  
계속하려면 아무 키나 누르십시오 . . .
```

```
C:\Users\click\OneDrive\Cl
```

### 큐 구현: 단순연결리스트 ###

- 1) 데이터 삽입: push
- 2) 데이터 삭제: popo
- 3) 전체 출력
- 4) 프로그램 종료

메뉴 선택: 3

##### 입력된 데이터 #####

QUEUE [ 1 2 3 4 5 ]  
계속하려면 아무 키나 누르십시오 . . .

```
C:\Users\click\OneDrive\Cl
```

### 큐 구현: 단순연결리스트 ###

- 1) 데이터 삽입: push
- 2) 데이터 삭제: popo
- 3) 전체 출력
- 4) 프로그램 종료

메뉴 선택: 2  
삭제 된 데이터: 1  
계속하려면 아무 키나 누르십시오 . . .

# 큐 구현(C++): 연결 자료구조 (3/7)

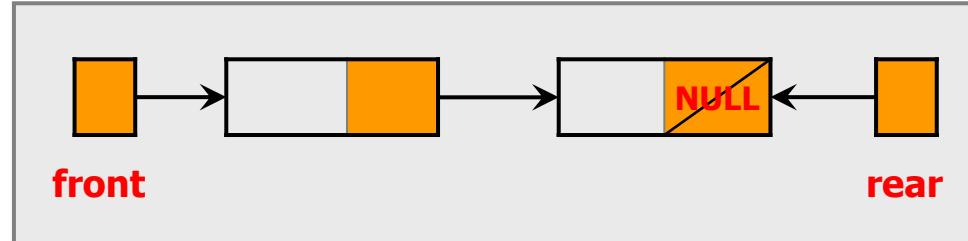
예제 6-9: 큐 -- 연결 자료구조

LinkedQueue(demo).cpp (1/2)

```
#include <iostream>
#include "LinkedQueue.cpp"
// #include <conio.h>
using namespace std;

int main(void)
{
    int num, choice;
    LinkedQueue<int> q = LinkedQueue<int>();

    while (true) {
        system("cls");
        cout << "\n\t### 큐 구현: 단순연결리스트 ### \n\n" << endl;
        cout << "1) 데이터 삽입: push" << endl;
        cout << "2) 데이터 삭제: popo" << endl;
        cout << "3) 전체 출력" << endl;
        cout << "4) 프로그램 종료 \n" << endl;
        cout << "메뉴 선택: ";
        cin >> choice;
```



# 큐 구현(C++): 연결 자료구조 (4/7)

예제 6-9: 큐 -- 연결 자료구조

LinkedQueue(demo).cpp (2/2)

```
switch (choice) {
    case 1:
        while (true) {
            cout << "데이터 입력(종료: 0): ";
            cin >> num;
            if (num == 0)
                break;
            q.push(num);
        }
        break;
    case 2: cout << "삭제 된 데이터: " << q.front() << endl;
        q.pop();
        break;
    case 3: q.printQueue();
        break;
    case 4: cout << "프로그램 종료!!!!" << endl;
        exit(0); // return 0;
    default: cout << "잘못 선택 하셨습니다!!!!" << endl;
}
// print("계속하려면 아무 키나 누르십시오...") ;
// _getch();
system("pause");
}
// q.~LinkedQueue();
return 0;
}
```

# 큐 구현(C++): 연결 자료구조 (5/7)

예제 6-9: 큐 -- 연결 자료구조

LinkedNode(template).cpp (1/2)

```
// #pragma once

#ifndef __SNode_Template_H__
#define __SNode_Template_H__


template <typename T> class LinkedQueue;

// 클래스 설계: SNode
template <typename T>
class SNode {
public:
    SNode(const T &data);
private:
    T           data_;
    SNode<T>   *link_;
    template <typename T> friend class LinkedQueue;
};


```

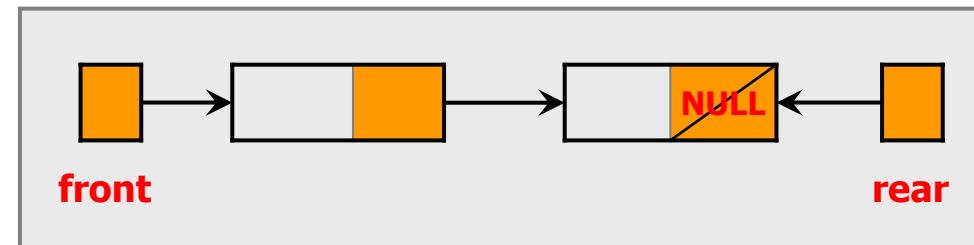
# 큐 구현(C++): 연결 자료구조 (6/7)

예제 6-9: 큐 -- 연결 자료구조

LinkedNode(template).cpp (2/2)

```
// SNode class: 노드 생성 및 조작 함수
template <typename T>
SNode<T>::SNode(const T &data) :
    data_(data), link_(nullptr) {}

#endif
```



# 큐 구현(C++): 연결 자료구조 (7/7)

예제 6-9: 큐 -- 연결 자료구조

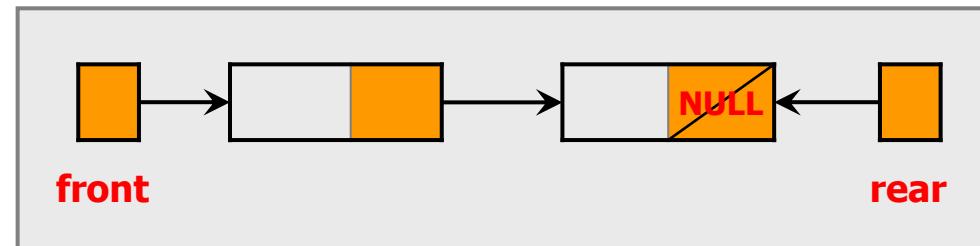
LinkedQueue.cpp

```
#include <iostream>
#include "LinkedNode(template).cpp"
using namespace std;

// #pragma once
#ifndef __LinkedQueue_Template_H
#define __LinkedQueue_Template_H

// 클래스 설계: LinkedQueue
template <typename T>
class LinkedQueue {
public:
    LinkedQueue();
    ~LinkedQueue();
    bool empty(void) const;
    int size(void) const;
    void push(const T &data);
    void pop(void);
    T front(void) const;
    T back(void) const;
    void printQueue(void) const;
private:
    SNode<T> *front_;
    SNode<T> *rear_;
    int count_;
};
```

// SNode<T>



// 빈 큐 생성  
// 빈 큐 삭제  
// 원소 여부  
// 원소 개수  
// 데이터 삽입  
// 데이터 삭제  
// 첫번째 노드의 데이터 반환  
// 마지막 노드의 데이터 반환  
// 전체 데이터 출력

#endif

# 큐 구현(C): 연결 자료 구조 (1/6)

## ● 큐 구현: 연결 자료구조

```
// #pragma once
#include "LinkedSNode.h"           // SNode
typedef int element;

// 큐 생성: LinkedQueue
#ifndef _LinkdedQueue_H_
#define _LinkdedQueue H

typedef struct _LinkedQueue {
    SNode *front;
    SNode *rear;
    int count;
} LinkedQueue;
#endif

// 큐 구현(C): 큐 생성 및 활용
LinkedQueue *queueCreate(void);
void queueDestroy(LinkedQueue *Queue);
void enqueue(LinkedQueue *Queue, element data);
void dequeue(LinkedQueue *Queue);
element front(LinkedQueue *Queue);
element back(LinkedQueue *Queue);
_Bool queueEmpty(LinkedQueue *Queue);
int queueSize(LinkedQueue *Queue);
void printQueue(LinkedQueue *Queue);
```



# 큐 구현(C): 연결 자료 구조 (2/6)

예제 6-10: 큐 구현 -- 연결 자료 구조

LinkedQueue(demo).c

```
#include <stdio.h>
#include <stdlib.h>           // system
#include <stdbool.h>          // bool, true, false
#include "LinkedQueue.h"        // LinkedQueue
// #include "LinkedSNode.h"      // SNode

int main(void)
{
    int num, data;
    LinkedQueue *Q = queueCreate();
    while(true) {
        system("cls");
        printf("\n ##### 큐 구현: 단순 연결 리스트 ##### \n\n");
        printf("1) 데이터 삽입(enQueue) \n");
        printf("2) 데이터 삭제(deQueue) \n");
        printf("3) 전체 출력 \n");
        printf("4) 프로그램 종료 \n\n");
        printf("메뉴 선택: ");
        scanf("%d", &num); // scanf("%d", &num);
        switch(num) {
            case 1: printf("\n 삽입 할 데이터 입력 : ");
                scanf("%d", &data); // scanf("%d", &data);
                enQueue(Q, data); break;
            case 2: printf("삭제 된 데이터 : %3d \n", front(Q));
                deQueue(Q); break;
            case 3: printQueue(Q); break;
            case 4: printf("프로그램 종료... \n");
                return 0;
            default: printf("잘못 선택 하셨습니다. \n");
        }
        system("pause");
    }
    queueDestroy(Q);
    return 0;
}
```

### 큐 구현: 단순 연결 리스트 ###  
1) 데이터 삽입(enQueue)  
2) 데이터 삭제(deQueue)  
3) 전체 출력  
4) 프로그램 종료  
메뉴 선택:

# 큐 구현(C): 연결 자료 구조 (3/6)

예제 6-10: 큐 구현 -- 연결 자료 구조

LinkedQueue.h

```
// #pragma once
#include "LinkedSNode.h"      // SNode
typedef      int      element;

// 큐 생성: LinkedQueue
#ifndef __LinkdedQueue_H__
#define __LinkdedQueue_H__
typedef struct _LinkedQueue {
    SNode *front;
    SNode *rear;
    int count;
} LinkedQueue;
#endif

// 큐 구현(C) : 큐 생성 및 활용
LinkedQueue *queueCreate(void);
void queueDestroy(LinkedQueue *Queue);
void enQueue(LinkedQueue *Queue, element data);
void deQueue(LinkedQueue *Queue);
element front(LinkedQueue *Queue);
element back(LinkedQueue *Queue);
Bool queueEmpty(LinkedQueue *Queue);
int queueSize(LinkedQueue *Queue);
void printQueue(LinkedQueue *Queue);
```

# 큐 구현(C): 연결 자료 구조 (4/6)

예제 6-10: 큐 구현 -- 연결 자료 구조

LinkedQueue.c (1/3)

```
#include <stdio.h>
#include <stdlib.h>           // malloc, free
#include "LinkedQueue.h"        // LinkedQueue, SNode
// #include "LinkedSNode.h"      // SNode

// queueCreate : 빈 큐 생성
LinkedQueue* queueCreate(void) {
    LinkedQueue *Q = (LinkedQueue*)malloc(sizeof(LinkedQueue));
    if(Q == NULL) {
        printf("스택 생성 실패!!! \n");
        return NULL;
    }
    Q->front = NULL;
    Q->rear = NULL;
    Q->count = 0;
    return Q;
}

// queueDestroy : 큐 삭제 -- 모든 노드 삭제
void queueDestroy(LinkedQueue *Q) {
    SNode *temp = Q->front;
    while(temp) {
        Q->front = temp->link;
        free(temp);
        temp = Q->front;
    }
    free(Q);
}
```

# 큐 구현(C): 연결 자료 구조 (5/6)

예제 6-10: 큐 구현 -- 연결 자료 구조

LinkedQueue.c (2/3)

```
// enqueue : 큐에서 데이터 삽입
void enqueue(LinkedQueue *Q, element data) {
    SNode *newSNode = makeSNode(data);
    if (Q->front == NULL) {
        Q->front = newSNode;
        Q->rear = newSNode;
    }
    else{
        Q->rear->link = newSNode;
        Q->rear = newSNode;
    }
    Q->count++;
}

// dequeue : 큐에서 데이터 삭제
void dequeue(LinkedQueue *Q) {
    if (queueEmpty(Q)) return;
    SNode *temp = Q->front;
    Q->front = temp->link;
    if (Q->front == NULL)
        Q->rear = NULL;
    free(temp);
    Q->count--;
}
```

# 큐 구현(C): 연결 자료 구조 (6/6)

예제 6-10: 큐 구현 -- 연결 자료 구조

LinkedQueue.c (3/3)

```
// front : 큐에서 첫 번째 원소 확인
element front(LinkedQueue* Q) {
    if(queueEmpty(Q)) return EOF;
    return Q->front->data;
}

// back : 큐에서 맨 마지막 원소
element back(LinkedQueue* Q) {
    if(queueEmpty(Q)) return EOF;
    return Q->rear->data;
}

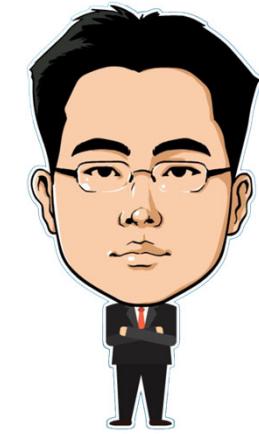
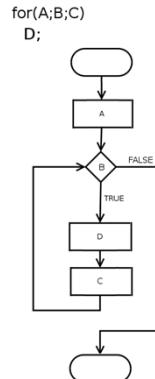
// queueEmpty : 큐의 공백 상태 여부 판단
_Bool queueEmpty(LinkedQueue* Q) {
    return Q->front == NULL;
}

// queueSize: 큐의 크기
int queueSize(LinkedQueue* Q) {
    return Q->count;
}

// printQueue : 큐의 전체 원소 출력
void printQueue(LinkedQueue* Q) {
    SNode* temp = Q->front;
    printf("\n Queue [");
    while(temp) {
        printf("%3d", temp->data);
        temp = temp->link;
    }
    printf(" ]\n");
}
```

# 참고문헌

- [1] "이것이 자료구조+알고리즘이다: with C 언어", 박상현, 한빛미디어, 2022.
- [2] "C++로 구현하는 자료구조와 알고리즘(2판)", Michael T. Goodrich, 김유성 외 2인 번역, 한빛아카데미, 2020.
- [3] "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 문병로, 한빛아카데미, 2022.
- [4] 문병로, "IT CookBook, 쉽게 배우는 알고리즘: 관계 중심의 사고법"(3판), 개정판, 한빛아카데미, 2024.
- [5] "코딩 테스트를 위한 자료 구조와 알고리즘 with C++", John Carey 외 2인, 황선규 역, 길벗, 2020.
- [6] "이것이 취업을 위한 코딩 테스트다 with 파이썬", 나동빈, 한빛미디어, 2020.
- [7] "SW Expert Academy", SAMSUNG, 2025 of viewing the site, <https://swexpertacademy.com/>.
- [8] "BAEKJOON", (BOJ) BaekJoon Online Judge, 2025 of viewing the site, <https://www.acmicpc.net/>.
- [9] "programmers", grepp, 2025 of viewing the site, <https://programmers.co.kr/>.
- [10] "goormlevel", goorm, 2025 of viewing the site, <https://level.goorm.io/>



이 강의자료는 저작권법에 따라 보호받는 저작물이므로 무단 전제와 무단 복제를 금지하며,  
내용의 전부 또는 일부를 이용하려면 반드시 저작권자의 서면 동의를 받아야 합니다.

Copyright © Clickseo.com. All rights reserved.