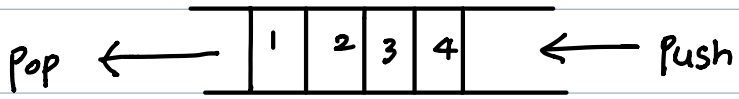
선형자료구조 - ② 큐 (Queue)



먼저들어간 원소가

먼저 나온다. First in first out

더 이상 뺄 것이 없는데, 더 빼라고 한다면?

Queue Underflow

반대 상황은 Queue Overflow

정리) Queue 는 First in First out,

연산	에러
• Queue.push(X)	• Queue Overflow
• Queue.pop()	• Queue Underflow

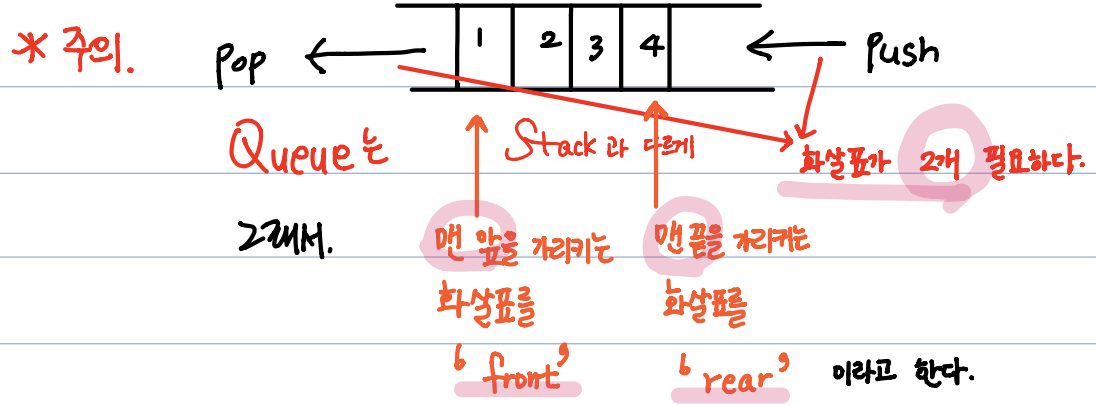
Queue 구현)

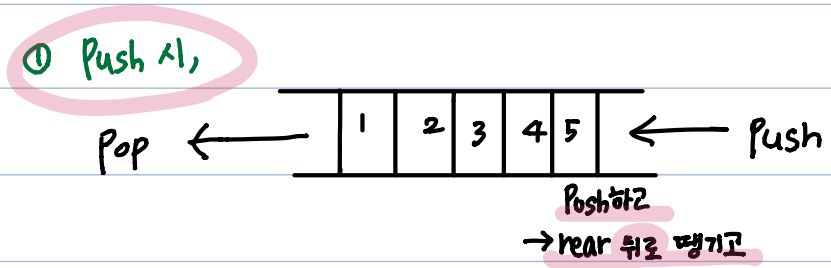
[문제] 큐 구현하기

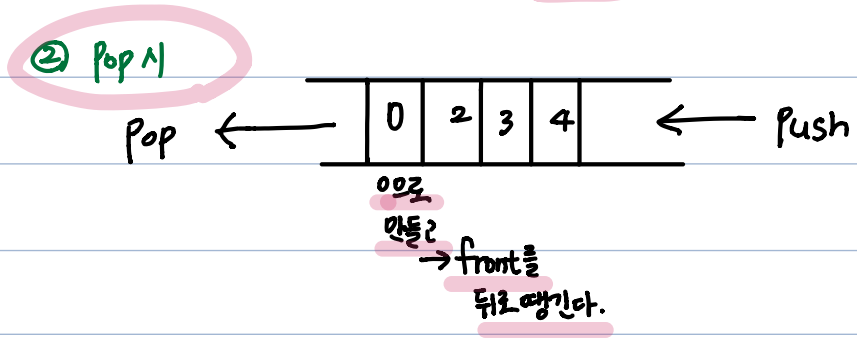
* 다음과 같은 일을 하는 큐를 구현하라

Q를 큐라 하자.

Q.create(x) : Q의 크기를 x로 지정한다.
Q.push(x) : Q에 x를 삽입한다.
Q.pop() : Q에서 원소 하나를 제거한다.
Q.front() : Q의 가장 앞에 있는 원소를 반환한다.
Q.size() : Q내에 존재하는 원소의 개수를 반환한다.





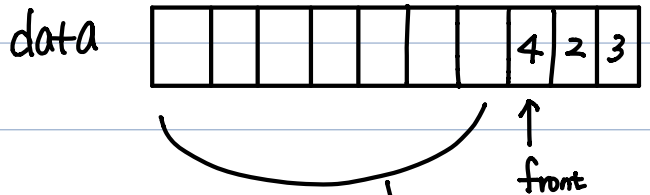



```c
#include <stdio.h>

struct Queue{
  // int   0 1 2 3 4 5 6 7
  //data   1 2 3 4 5 5 6
  /*이때 pop하면 하나를 없애고 뒤의 것을 앞으로 다 당겨와야 하는데
  그러면 시간이 너무 많이 걸리니까
  그냥 pop한 것을 0으로만 바꿔주고 뒤의 것은 그대로 남기는 것으로
  일단 구현한다.
  */

  int data[100];
  int f, r;
  int capacity;

  void create(int y){
    capacity = y;
    f = 0;
    r = 0;
    //왜냐면 아직 아무것도 push되어 있지 않으니까.
  }
```

```
22
23    void push(int y){
24      if(r-f>=capacity)//r-f = 원소의 개수.
25        printf("Queue overflow\n");
26      else{
27        data[r++] = y;
28      }
29    }
30
31    void pop(){
32      if(r-f<=0)//즉, 가지고 있는 원소가 없는데 빼라고 하면
33        printf("Queue underflow\n");
34      else{
35        data[f]=0;
36        f++;
37      }
38    }
39
40    int front(){//큐의 가장 앞에있는 (가장 처음에 들어온 원소를) 반환환
41      if(r-f<=0)
42        return -1; //즉, 자기고 있는 원소가 없으면,
43        //반환할 것이 없다는 표시로 -1을 반환.
44      else{
45        return data[f];
46      }
47    }
48
49    int size(){
50      return r-f;
51
52    }
53
54  };
55
56  int main() {
57
58    //큐 구현하기
59    Queue q1;
60    q1.create(3);
61
62    q1.push(1);
63    q1.push(2);
64    q1.push(3);
65    q1.push(4);//overflow
66    q1.push(5);//overflow
67
68    printf("%d\n", q1.front());//1
69
70    q1.pop();//1제거
71    q1.pop();//2제거
72
73    printf("%d\n", q1.front());//3
74    printf("%d\n", q1.size());//1
75
76    q1.pop();
77    q1.pop();//underflow
78
79    printf("%d\n", q1.size());//0
80
81    return 0;
82  }
```

Queue 구현의 문제점) 공간활용을 제대로 못하고 있다.

capacity = 4



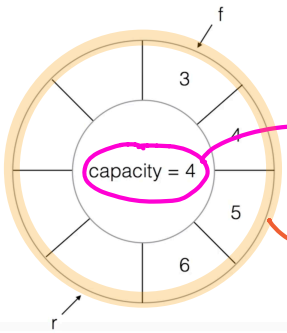
rear이 계속 증가만 하기 때문에

즉, data의 끝에서는

앞에 이만큼의 공간이 남아있음에도 불구하고

더이상 진행되지 못하는 상황이 생긴다.

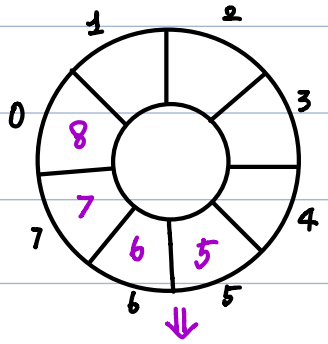
→ 해결책. 6 원형큐 (circular Queue)

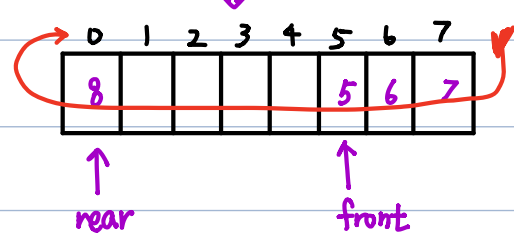


capacity : 이만큼 최대 넣겠다고 정한 용량

Maxnum : 큐길이유.

원형큐의 구현)





벗어나면 앞으로 오더라

원형Queue 사용시 고려사항 ①)

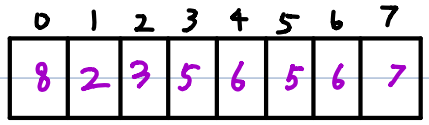
이렇게 rear이 다시 앞으로 온다.

그러나 여기선 (rear - front =) size를 이런식으로 구할수 없다.

→ 해결책 : Push 할 때마다 별도의 변수를 증가 시킨다.

Pop 할 때마다 별도의 변수를 감소 시킨다.

원형Queue 사용시 고려사항 ②)



) front와 rear이 같을 때는 2가지 상황이 가능

1. 꽉 차있거나

2. 비어있거나

→ 해결책:
이를 통해 판단한다

: 좋은 코딩습관들

: 원형 Queue 라서 달라진 점.

```c
#include <stdio.h>

const int Max_num = 10;//배열공간의 최대값은 앞으로 이렇게 const int문법을 사용하는것이 좋은 코딩이다.

struct Queue{
  int data[Max_num];
  int f, r;
  int capacity;

  int numElement; //push, pop할 때마다 변경시켜주는 별도의 변수

  void create(int y){
    capacity = y;
    f=0; r=0;
    numElement = 0;
  }

  void push(int y){
    if(numElement>=capacity)
      printf("Queue overflow\n");//즉 내가 가지고 있는 원소의 개수(numElement)가
      //capacity를 넘으면 Queue overflow이다.

    else{//그게 아니면

      data[r]=y;
      r++
      /*if(r>=Max_num)//data의 공간을 넘으면 다시 앞으로 와야함.
      r=0;*/
      //위의 코딩보다
      r = (r+1) % Max_num;//으로 하는 것이 좋은 코딩습관이다.

      numElement++;
    }
  }
```

Q. 근데 왜 capacity로 나눠야 해?

```
35
36    void pop(){
37      if(numElement<=0)
38        printf("Queue underflow\n");
39      else{
40
41        data[f]=0;
42
43        f = (f+1) % Max_num;
44
45        if(f>=Max_num)
46          f=0;
47
48        numElement--;
49      }
50    }
51
52    int front(){
53      if(numElement<=0)
54        return -1;
55      else{
56        return data[f];
57      }
58    }
59
60    void size(){
61      return numElement;
62    }
63
64  };
65
```

MAX_num으로 나누죠?
→ 배열크기를 넘어야 원형의 시작점으로 감.

이렇게!!

) 이거 대신