



실전 알고리즘 0x03강 스택, 큐, 덱

BaaaaaaaaaaaaaaaaarkingDog

목차



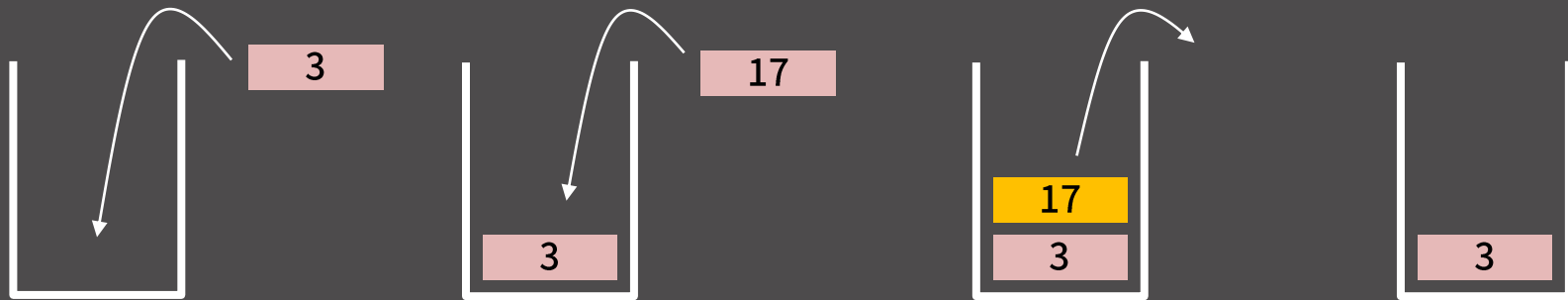
0x00 스택(Stack)


0x01 큐(Queue)

0x02 덱(Deque)

0x03 문제 소개

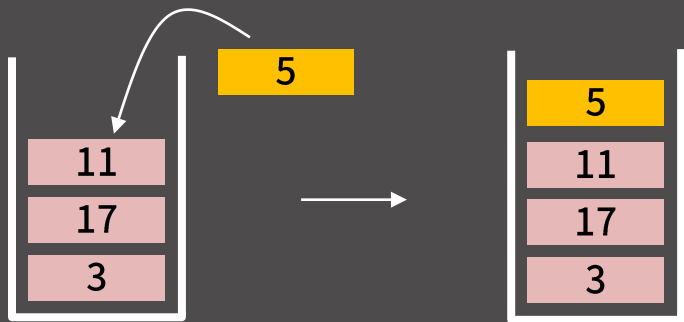
0x00 스택(Stack) - 정의



- 스택을 들어본 적 있나요? 스택의 예시 : 나서스 스택 
- 스택은 **한쪽 끝에서만 원소를 넣거나 뺄 수 있는 자료구조**입니다. 프링글스 통을 생각하면 이해가 쉬울 것입니다. 먼저 들어간 원소가 제일 나중에 나온다는 의미로 FILO(First in Last Out) 자료구조라고 부르기도 합니다.
- 큐, 덱도 스택과 마찬가지로 특정 위치에서만 원소를 넣거나 뺄 수 있고 이러한 자료구조들을 Restricted Structure라고 부릅니다.

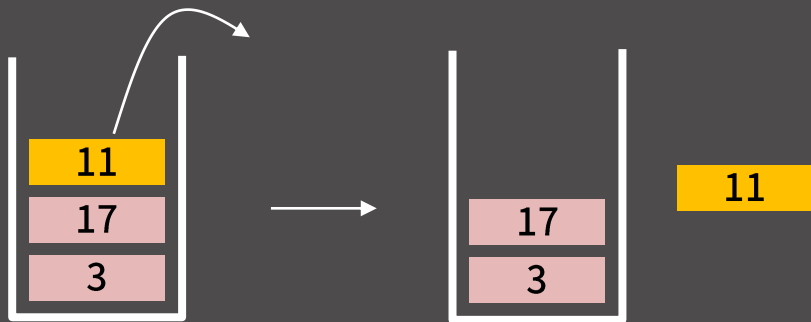
0x00 스택(Stack) - 연산의 시간복잡도

원소를 추가



- 배열의 끝에 원소를 추가하는 것과 비슷하게 $O(1)$ 에 추가 가능

원소를 제거



- 배열의 끝에 원소를 제거하는 것과 비슷하게 $O(1)$ 에 제거 가능

0x00 스택(Stack) – 연산의 시간복잡도

제일 꼭대기의 원소를 확인



- 스택의 끝이 어딘지 알고 있으므로 **$O(1)$** 에 확인 가능

- 원소를 추가 = $O(1)$
- 원소를 제거 = $O(1)$
- 제일 꼭대기의 원소를 확인 = $O(1)$

0x00 스택(Stack) – 쓰임새



- 학부 과정에서 반드시 짚고 넘어가는 스택의 대표적인 응용 사례는 **수식의 괄호 쌍**, **전위/중위/후위 표기법**, **DFS**, **Flood Fill** 이 있습니다. 각 사례는 이 강의 안에 담기엔 양이 굉장히 많기 때문에 여기서 다루지 않고 각각에 대한 독립적인 강의를 따로 만들 것입니다. 특히 DFS는 코딩테스트 단골 문제이기 때문에 반드시 익숙해져야 합니다.
- 문자열 뒤집기도 스택을 이용해 풀 수 있는 문제입니다.(모든 문자를 스택에 넣은 후 꺼내면 알아서 뒤집어짐) 그런데 스택으로 문자열을 뒤집는 것 보다 그냥 배열에서 for문을 한 번 역으로 돌리는게 훨씬 간편합니다.

0x00 스택(Stack) – STL Stack

- 없는 것 빼고 다 있는 STL에 Stack도 있습니다.
- 레퍼런스 사이트 : <http://www.cplusplus.com/reference/stack/stack/>

```
stack<int> S;
S.push(10); // 10
S.push(20); // 10 20
S.push(30); // 10 20 30
cout << S.size() << '\n'; // 3 출력
if(S.empty()) cout << "S is empty\n";
else cout << "S is not empty\n"; // not empty 출력
S.pop(); // 10 20
cout << S.top() << '\n'; // 20
cout << S.top() << '\n'; // 20
S.pop(); // 10
S.pop(); // empty
if(S.empty()) cout << "S is empty\n"; // empty 출력
else cout << "S is not empty\n";
```

0x00 스택(Stack) – 배열을 이용한 구현



- 스택은 배열로 정말 간단하게 구현할 수 있는 자료구조입니다.
- 연결 리스트를 이용해서 구현할 수도 있지만 코딩테스트에서 쓰기엔 배열을 이용한 구현이 더 편합니다.
- 배열을 이용한 구현이 STL Stack보다 조금 더 빠르다는 장점이 있고, 구현이 간편한 편이긴 하지만 그래도 STL을 쓸 수 있는 환경이면 그냥 STL을 쓰는 것을 추천드립니다.

0x00 스택(Stack) – 배열을 이용한 구현



변수 설명

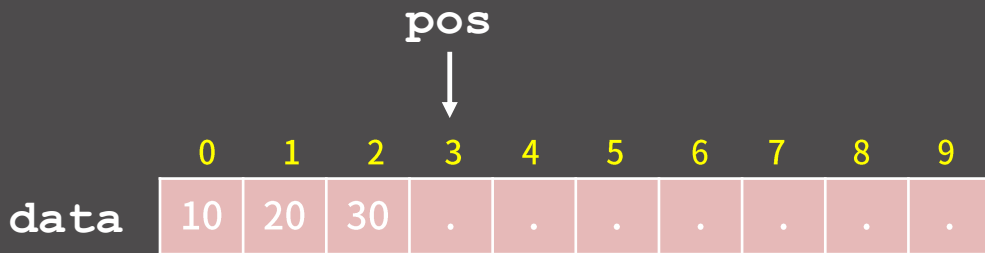
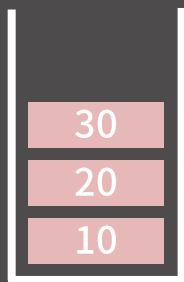
- `dat[i]`는 `i`번째 원소의 값, `pos`는 스택 내에 새로운 원소가 들어갈 위치이고, 이 값은 스택 내의 원소의 갯수와 동일합니다.
- `pos`의 초기값은 0입니다.

```
const int MX = 10000007;  
int dat[MX];  
int pos; // 알아서 0으로 초기화 됨
```

0x00 스택(Stack) - 배열을 이용한 구현

예시

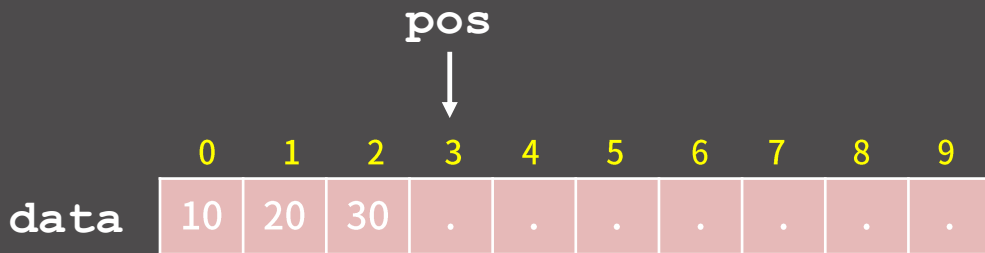
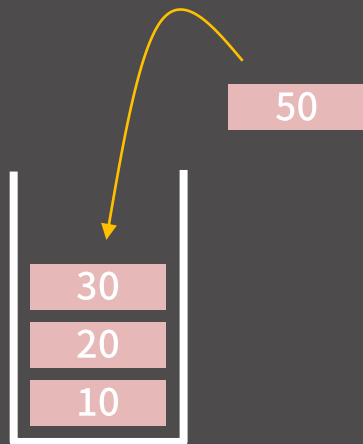
- 10, 20, 30이 스택에 들어있습니다.



0x00 스택(Stack) - 배열을 이용한 구현

삽입

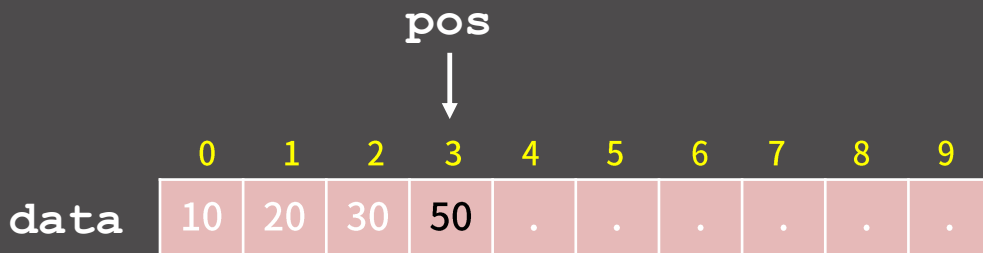
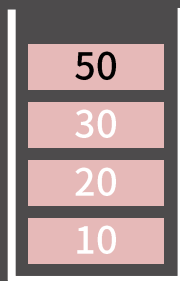
- 50을 스택에 넣는 과정을 step by step으로 이해해봅시다.



0x00 스택(Stack) - 배열을 이용한 구현

삽입

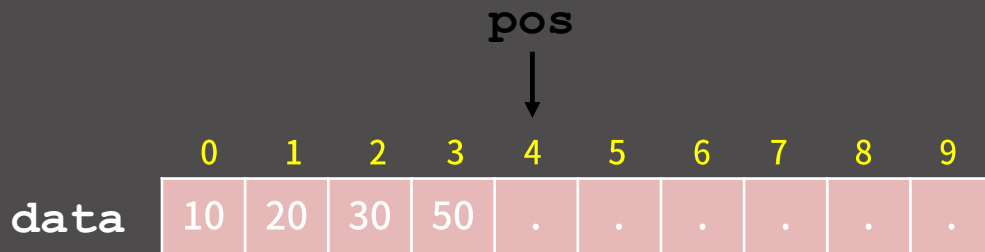
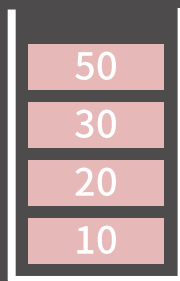
- 1. pos가 가리키는 곳에 새로 삽입할 원소를 씁니다.



0x00 스택(Stack) - 배열을 이용한 구현

삽입

- 2. pos를 1 증가시킵니다.



0x00 스택(Stack) - 배열을 이용한 구현

삽입

- 증감 연산자를 이용해 한 줄에 처리할 수도 있습니다.

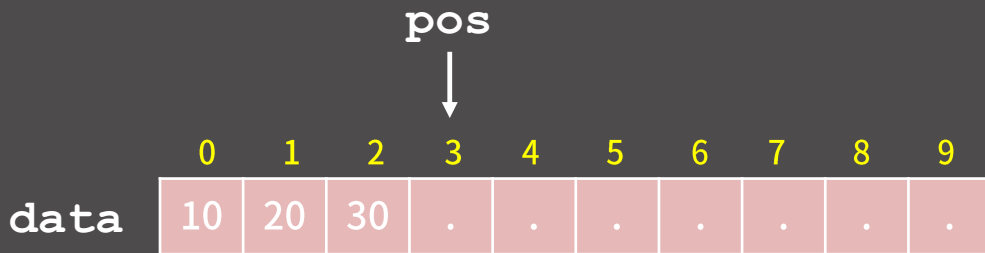
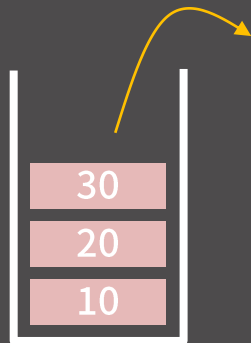
```
void push(int val){  
    dat[pos] = val; // step 1  
    pos++; // step 2  
}
```

```
void push(int val){  
    dat[pos++] = val; // step 1, 2  
}
```

0x00 스택(Stack) - 배열을 이용한 구현

삭제

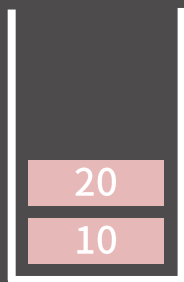
- 30을 빼내기 위해서는 `pos`를 1 감소시키기만 하면 됩니다.



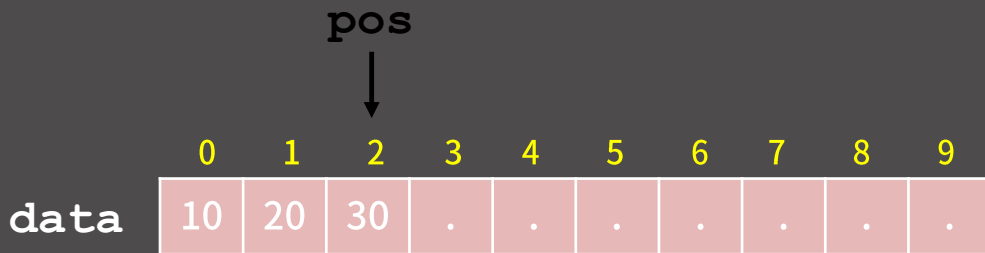
0x00 스택(Stack) - 배열을 이용한 구현

삭제

- 2번지의 값은 관념적으로 더 이상 Stack에 속해있지 않는 값이기 때문에 굳이 30을 지우지 않아도 됩니다.
- pos가 0일 경우에는 pop 함수를 실행하지 않아야 합니다.



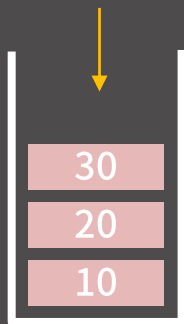
```
void pop() {  
    pos--;  
}
```



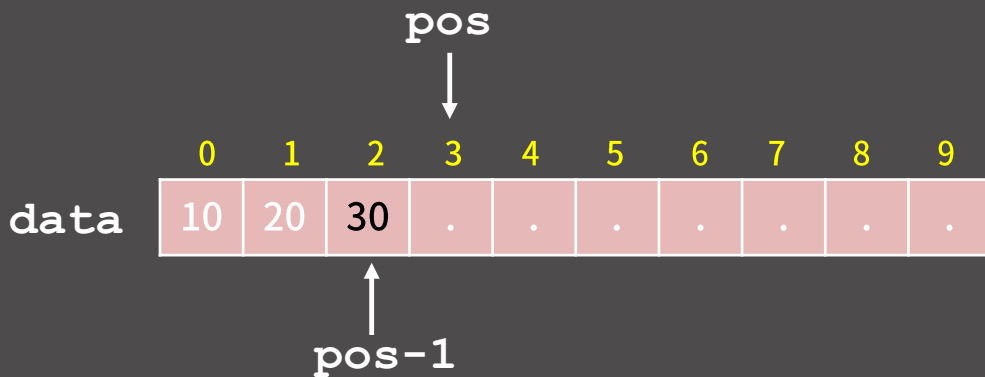
0x00 스택(Stack) - 배열을 이용한 구현

제일 꼭대기의 원소 확인

- `data[pos-1]` 을 반환하면 됩니다.
- `pos`가 0일 경우에는 `top` 함수를 실행하지 않아야 합니다.



```
int top() {  
    return dat[pos-1];  
}
```

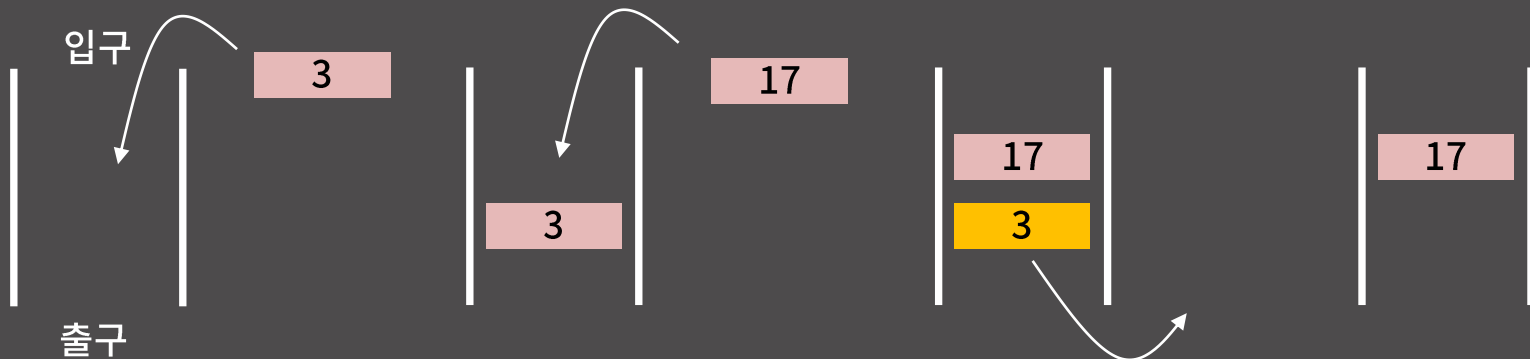


0x00 스택(Stack) – 예제 1



- BOJ 10828번 : 스택(icpc.me/10828)
- 스택을 충실하게 구현하면 됩니다.
- 정답코드 1(STL Stack) : <http://boj.kr/9c0fc55ca24448269df936143f69199e>
- 정답코드 2(배열을 이용한 구현) : <http://boj.kr/8a8ee971d1444e9d81c842bddb6e2fdb>

0x01 큐(Queue) - 정의



- 큐를 들어본 적 있나요?
- 큐는 한쪽 끝에서 원소를 넣고 반대쪽 끝에서 원소를 뺄 수 있는 자료구조입니다. 공항에서 줄을 서는 과정을 생각해보면 이해가 갈 것입니다.
- 먼저 들어간 원소가 제일 먼저 나온다는 의미로 FIFO(First In First Out) 자료구조라고 부르기도 합니다.

큐의 예시 : 롤 큐 안잡힘ㅠㅠ



0x01 큐(Queue) – 연산의 시간복잡도



- 큐도 스택과 마찬가지로 원소의 추가/제거, 제일 앞/제일 뒤에 위치한 원소의 확인이 모두 $O(1)$ 입니다.
- 원소를 추가 = $O(1)$
- 원소를 제거 = $O(1)$
- 제일 앞/제일 뒤에 위치한 원소를 확인 = $O(1)$

0x01 큐(Queue) – 쓰임새



- 큐는 **BFS**, **Flood Fill** 에서 사용됩니다. 스택과 비슷하게 이 강의 안에 담기엔 양이 굉장히 많기 때문에 여기서 다루지 않고 각각에 대한 독립적인 강의를 따로 만들 것입니다.
- BFS, Flood Fill 이외에는 딱히 코딩테스트에서 나올 부분이 보이지 않습니다.

0x01 큐(Queue) – STL Queue

- 없는 것 빼고 다 있는 STL에 Queue도 있습니다.
- 레퍼런스 사이트 : <http://www.cplusplus.com/reference/queue/queue/>

```
int main(void) {
    queue<int> Q;
    Q.push(10); // 10
    Q.push(20); // 10 20
    Q.push(30); // 10 20 30
    cout << Q.size() << '\n'; // 3
    if(Q.empty()) cout << "empty\n";
    else cout << "not empty\n"; // not empty가 출력됨
    Q.pop(); // 20 30
    Q.push(40); // 20 30 40
    cout << Q.front() << '\n'; // 20
    cout << Q.back() << '\n'; // 40
    Q.pop(); // 30 40
}
```

0x01 큐(Queue) – 배열을 이용한 구현



- 큐 또한 배열로 정말 간단하게 구현할 수 있는 자료구조입니다.
- 연결 리스트를 이용해서 구현할 수도 있지만 코딩테스트에서 쓰기엔 배열을 이용한 구현이 더 편합니다.
- Stack과 비슷하게 배열을 이용한 구현이 STL Queue보다 조금 더 빠르다는 장점이 있고, 구현이 간편한 편이긴 하지만 그래도 STL을 쓸 수 있는 환경이면 그냥 STL을 쓰는 것을 추천드립니다.
- 정석적인 구현은 메모리누수를 막을 수 있는 원형 큐(Circular Queue)이지만 야매 Linked List와 비슷하게 어차피 알고리즘 문제의 환경에서는 메모리누수를 걱정해야할 정도로 삽입이 일어나지 않으므로 그냥 선형 큐로 구현했습니다. 다만 나중에 면접 대비를 위해서는 원형 큐를 따로 공부하셔야 합니다.

0x01 큐(Queue) - 배열을 이용한 구현



변수 설명

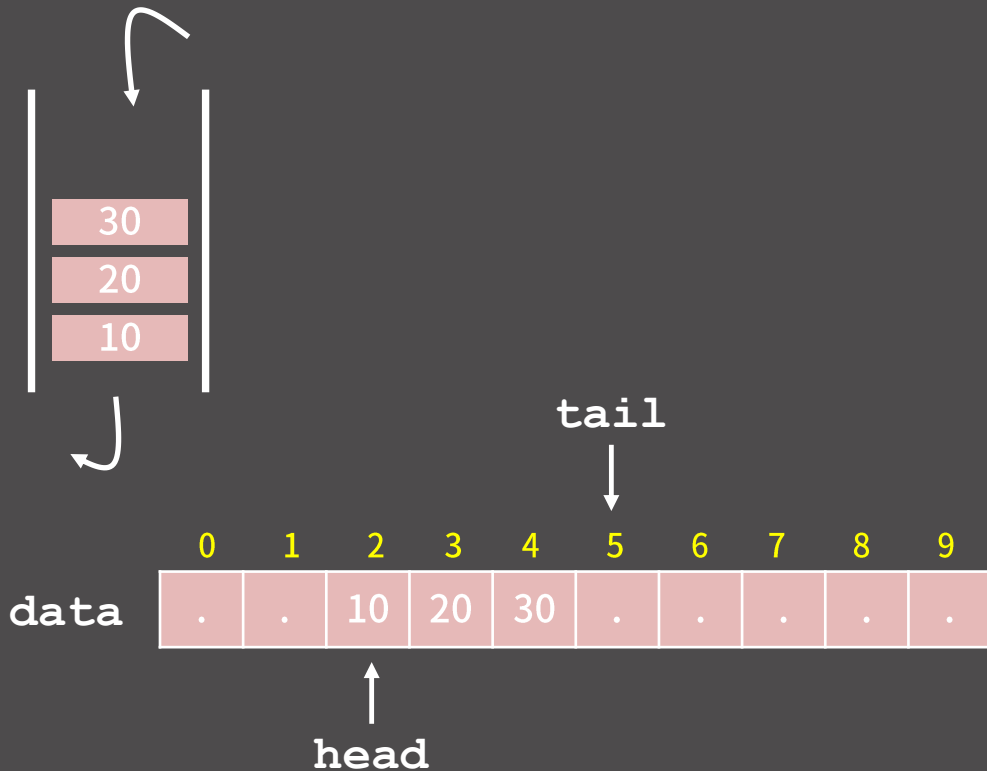
- `dat[i]`는 i 번지 원소의 값, `head`는 제거될 원소의 위치, `tail`은 큐 내에서 삽입할 위치를 의미합니다.
- `head`, `tail`의 초기값은 0이며 `tail-head` 는 큐의 길이입니다.

```
const int MX = 10000007;  
int dat[MX];  
int head, tail;
```


0x01 큐(Queue) - 배열을 이용한 구현

예시

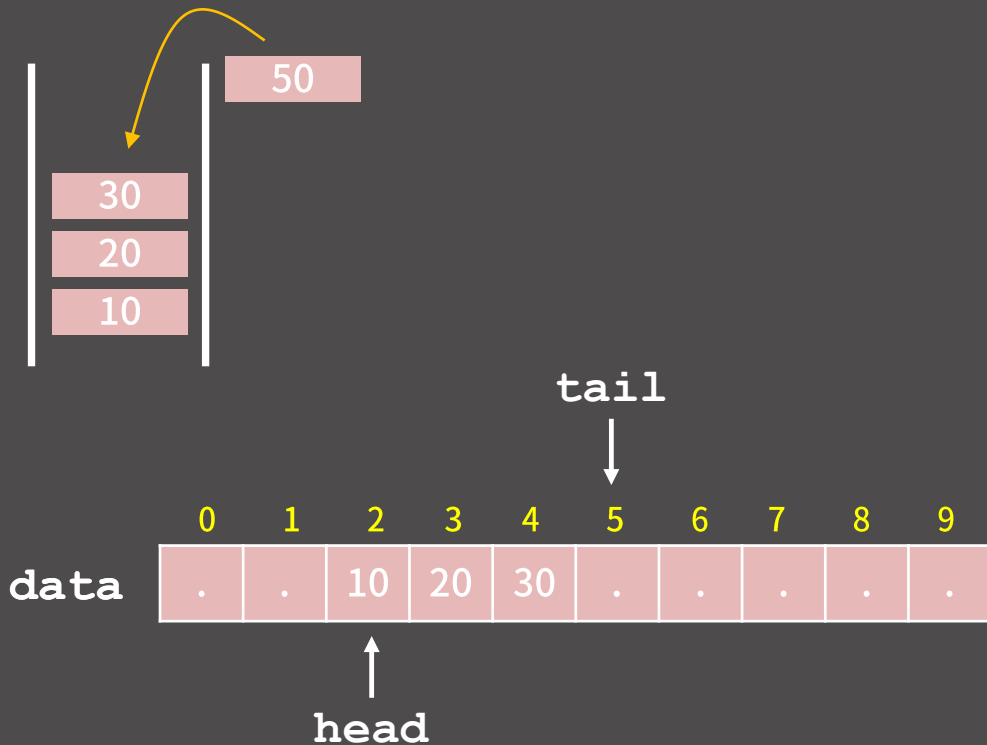
- 10, 20, 30이 큐에 들어있습니다.



0x01 큐(Queue) - 배열을 이용한 구현

삽입

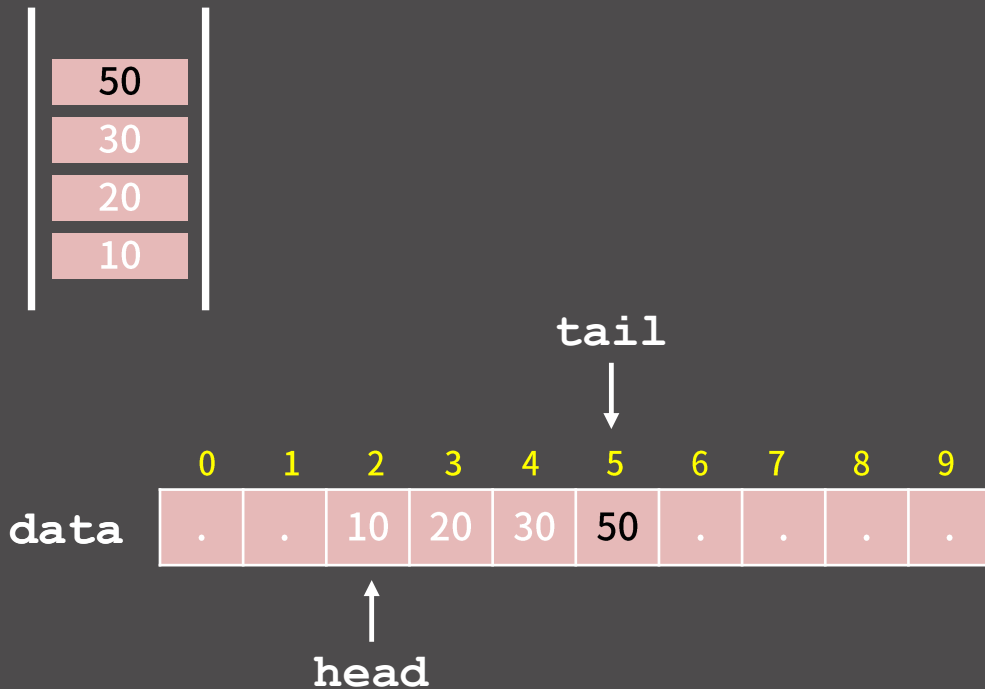
- 50을 큐에 넣는 과정을 step by step으로 이해해봅시다.



0x01 큐(Queue) - 배열을 이용한 구현

삽입

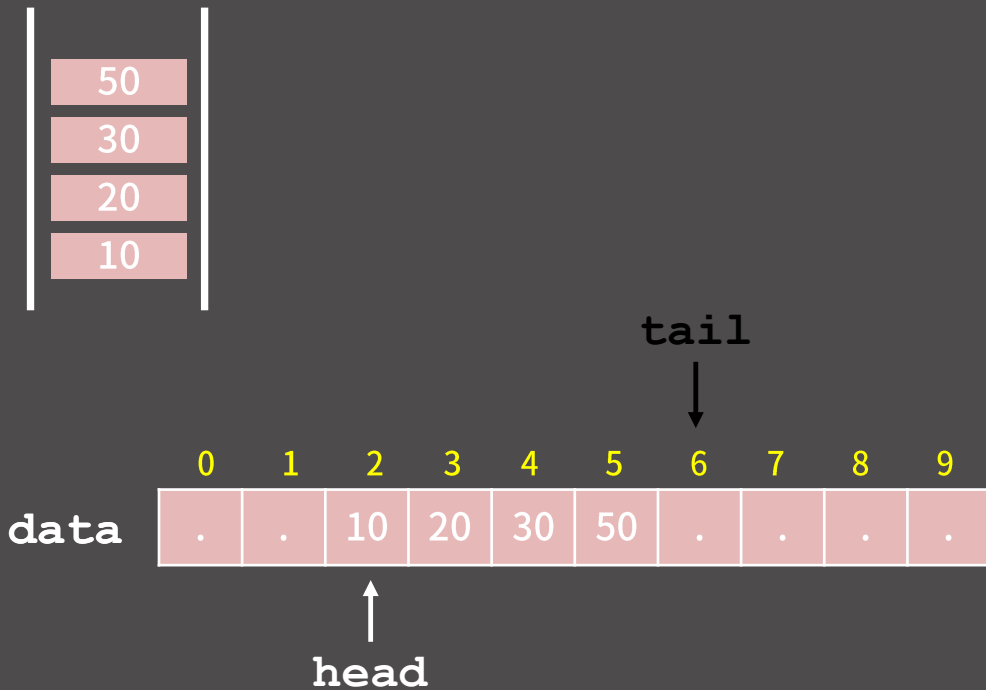
- 1. `tail`이 가리키는 곳에 새로 삽입할 원소를 씁니다.



0x01 큐(Queue) - 배열을 이용한 구현

삽입

- 2. `tail`을 1 증가시킵니다.



0x01 큐(Queue) - 배열을 이용한 구현

삽입

- 증감 연산자를 이용해 한 줄에 처리할 수도 있습니다.

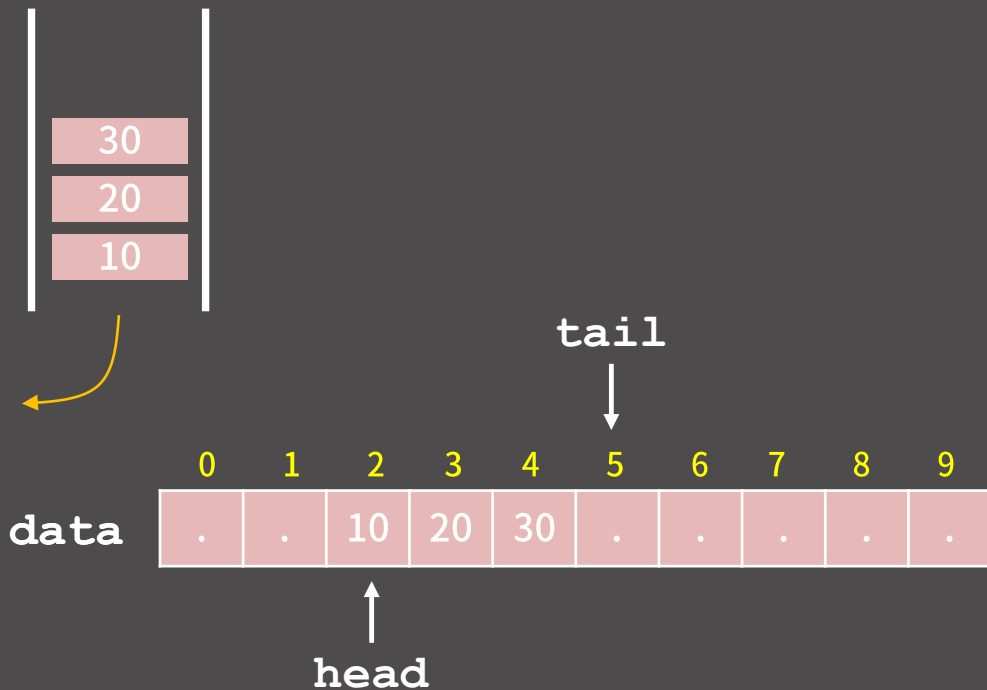
```
void push(int val){  
    dat[pos] = val; // step 1  
    pos++; // step 2  
}
```

```
void push(int val){  
    dat[pos++] = val; // step 1, 2  
}
```

0x01 큐(Queue) - 배열을 이용한 구현

삭제

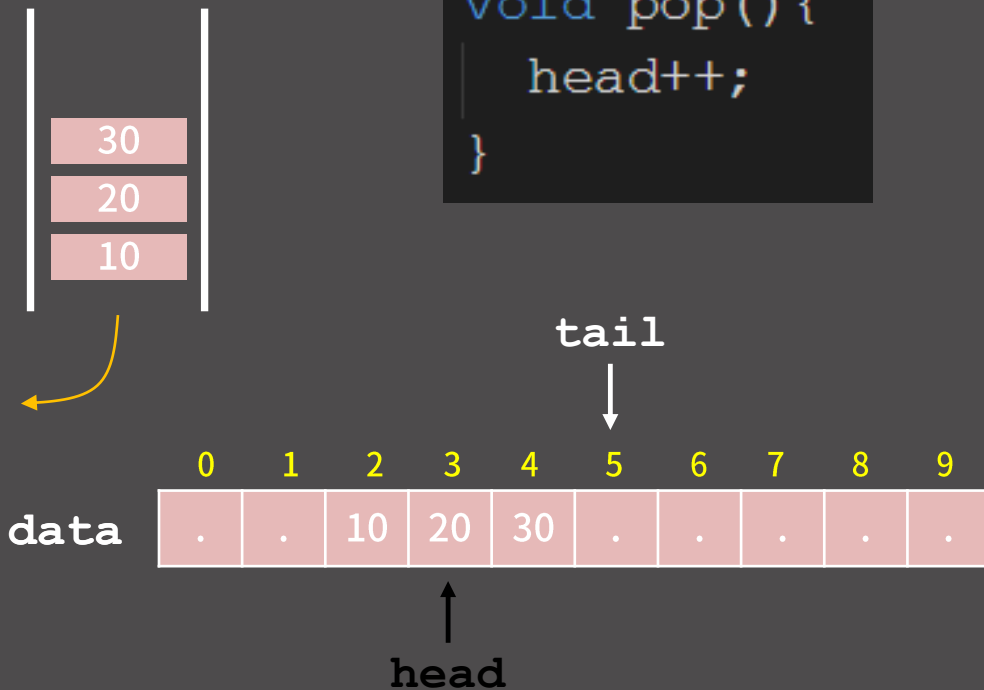
- 10을 빼내기 위해서는 `head`를 1 증가시키기만 하면 됩니다.
- `head = tail`일 때(=큐가 비어있을 때)에는 해당 연산을 수행하지 않도록 해야합니다.



0x01 큐(Queue) - 배열을 이용한 구현

삭제

- 2번지에 적힌 값은 앞으로 영영 참고될 일이 없으니 굳이 지우지 않아도 괜찮습니다.
- `head = tail` 일 때(=큐가 비어있을 때)에는 해당 연산을 수행하지 않도록 해야합니다.



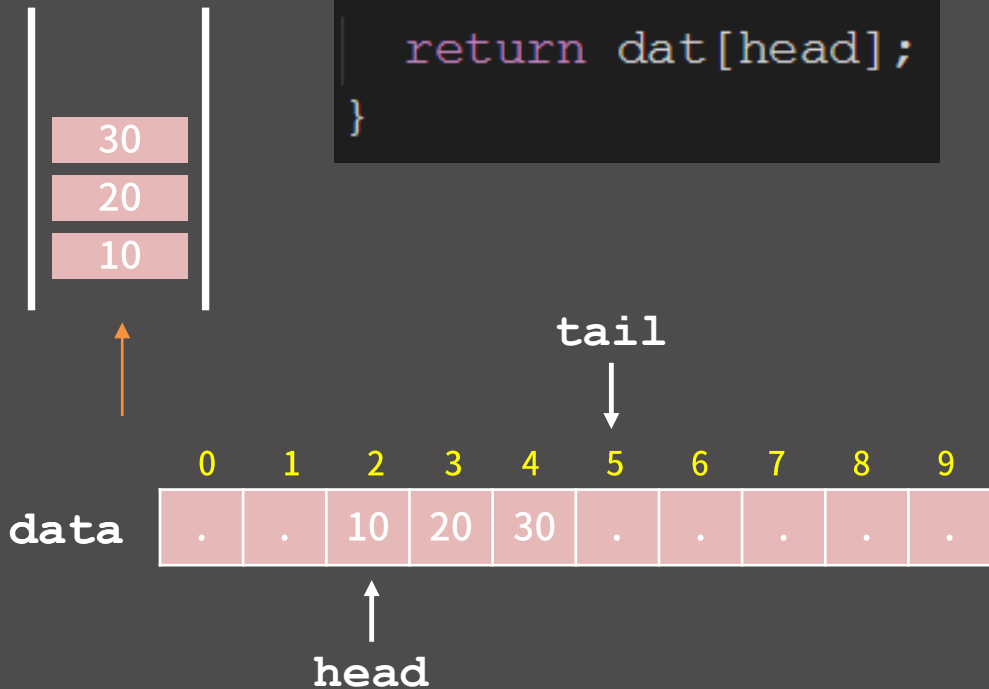
0x01 큐(Queue) - 배열을 이용한 구현



제일 앞에 있는 원소 확인

- `data[head]`를 반환합니다.
- `head = tail`일 때(=큐가 비어있을 때)에는 해당 연산을 수행하지 않도록 해야합니다.

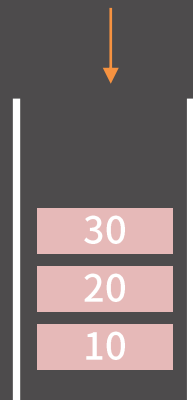
```
int front() {  
    return dat[head];  
}
```



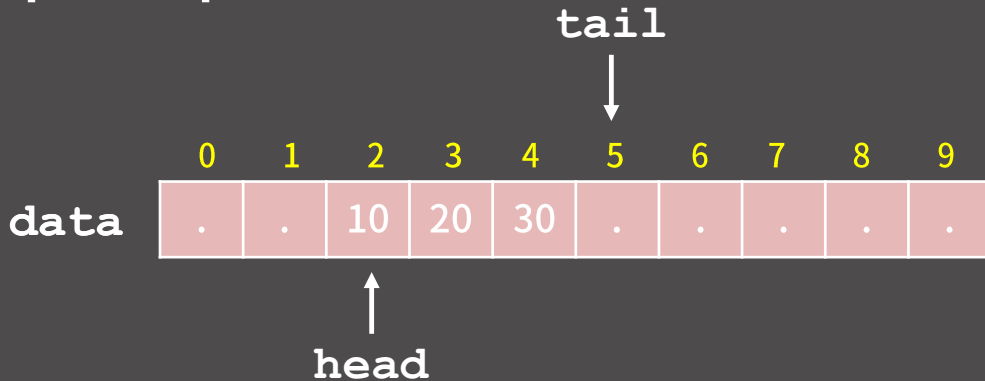
0x01 큐(Queue) - 배열을 이용한 구현

제일 뒤에 있는 원소 확인

- `data[tail-1]`를 반환합니다.
- `head = tail`일 때(=큐가 비어있을 때)에는 해당 연산을 수행하지 않도록 해야합니다.



```
int back() {  
    return dat[tail-1];  
}
```

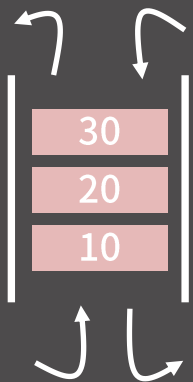


0x01 큐(Queue) – 예제 1



- BOJ 10845번 : 큐(icpc.me/10845)
- 큐를 충실하게 구현하면 됩니다.
- 정답코드 1(STL Queue) : <http://boj.kr/c4808270817b438abbc3646261c2c072>
- 정답코드 2(배열을 이용한 구현) : <http://boj.kr/fb23fbb9146543889dd7387b801ab585>

0x02 덱(Deque) - 정의



- 덱을 들어본 적 있나요? (하스스톤 덱에서의 덱(Deck)과는 다릅니다.)
- 덱은 양쪽 끝에서 원소를 넣고 뺄 수 있는 자료구조입니다. 양 쪽이 전부 뚫린 셔틀콕 원통형 배드를 생각해보면 이해가 갈 것입니다.

0x02 덱(Deque) – 연산의 시간복잡도



- 덱도 큐,스택과 마찬가지로 앞/뒤에서 원소의 추가/제거, 제일 앞/제일 뒤에 위치한 원소의 확인이 모두 $O(1)$ 입니다.
- 원소를 추가 = $O(1)$
- 원소를 제거 = $O(1)$
- 제일 앞/제일 뒤에 위치한 원소를 확인 = $O(1)$

0x02 덱(Deque) – 쓰임새



- 잘 생각해보면 덱이 스택의 성질과 큐의 성질을 모두 가지고 있습니다.
- Linked List와 유사하게 덱을 사용해야하는 문제는 보통 양쪽에서 삽입/삭제가 일어나는 상황을 시뮬레이션하는 느낌의 문제입니다. 딱 보면 덱을 사용하는 문제라는 것을 알 수 있을 것입니다.

0x02 덱(Deque) – STL Deque

- 없는 것 빼고 다 있는 STL에 Deque도 있습니다.
- 레퍼런스 사이트 : <http://www.cplusplus.com/reference/deque/deque/>

```
deque<int> DQ;
DQ.push_back(10); // 10
DQ.push_back(40); // 10 40
DQ.push_front(20); // 20 10 40
cout << DQ.size() << '\n'; // 3
cout << DQ.back() << '\n'; // 40
if(DQ.empty()) cout << "empty\n";
else cout << "not empty\n"; // not empty 출력
DQ.pop_front(); // 10 40
DQ.push_back(20); // 10 40 20
DQ.pop_back(); // 10 40
cout << DQ.front() << '\n'; // 10
cout << DQ[0] << ' ' << DQ[1] << '\n'; // 10 20
```

0x02 덱(Deque) – STL Deque



- 덱 또한 배열로 정말 간단하게 구현할 수 있는 자료구조입니다.
- 연결 리스트를 이용해서 구현할 수도 있지만 코딩테스트에서 쓰기엔 배열을 이용한 구현이 더 편합니다.
- 마찬가지로 STL을 쓸 수 있는 환경이면 그냥 STL을 쓰는 것을 추천드립니다.
- 정석적인 구현은 메모리누수를 막을 수 있는 원형 덱(Circular Deque)이지만 야매 Linked List와 비슷하게 어차피 알고리즘 문제의 환경에서는 메모리누수를 걱정해야할 정도로 삽입이 일어나지 않으므로 그냥 선형 덱으로 구현했습니다.
- 코드의 구성이 스택, 큐와 거의 흡사하기 때문에 각 함수에 대한 자세한 설명은 넘어가겠습니다. 대신 주석을 상세하게 달아둘테니 예제의 정답 코드를 확인해보세요.

0x02 덱(Deque) – 예제 1



- BOJ 10866번 : 덱(icpc.me/10866)
- 덱을 충실하게 구현하면 됩니다.
- 정답코드 1(STL Deque) : <http://boj.kr/d5da87fb4fd649beac722bf0b9707c1c>
- 정답코드 2(배열을 이용한 구현) : <http://boj.kr/783bf26fa3504b5f95d8461f141a3823>

0x03 문제 소개



문제 번호	알고리즘 분류	발상 난이도	구현 난이도
1874	Stack	3/10	2/10
1021	Deque	2/10	2/10
5430	Deque	3/10	3/10
9012	Stack	3/10	1/10

- 전반적으로 발상이 조금 어렵습니다. 고민을 충분히 해보시고, 실마리가 잡히지 않는다면 풀이를 참고하되 꼭 직접 구현해보세요.

강의 정리

- 스택, 큐, 덱의 정의와 구현 방법에 대해 알았습니다.
- 수식의 괄호 쌍, 전위/중위/후위 표기법, BFS/DFS, Flood Fill등의 메이저한 각 자료구조의 응용은 앞으로 있을 강의에서 짚고 넘어갈 예정입니다.