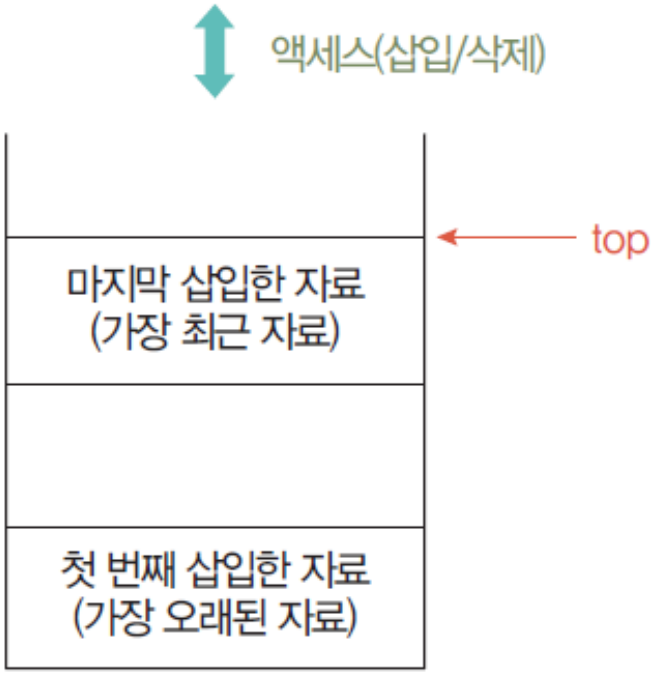
스택

❖ 스택(stack)

- 스택에 저장된 원소는 top으로 정한 곳에서만 접근 가능
- top의 위치에서만 원소를 삽입하므로, 먼저 삽입한 원소는 밑에 쌓이고, 나중에 삽입한 원소는 위에 쌓이는 구조
- 마지막에 삽입(Last-In)한 원소는 맨 위에 쌓여 있다가 가장 먼저 삭제(First-Out)됨
  - ☞ 후입선출 구조 (LIFO, Last-In-First-Out)



스택

❖ 스택의 연산

- 스택에서의 삽입 연산 : push
- 스택에서의 삭제 연산 : pop

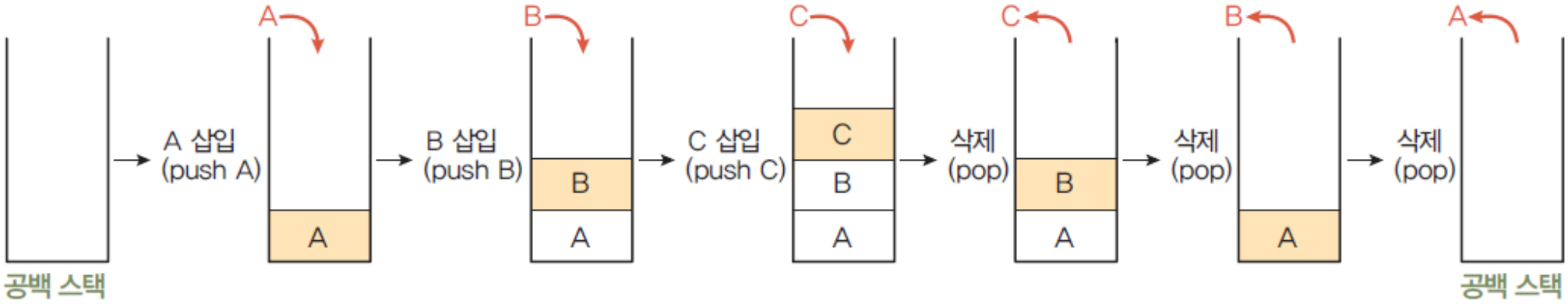


그림 5-6 스택의 데이터 삽입(push)과 삭제(pop) 과정

❖ 스택의 push() 알고리즘

① top ← top+1;

- 스택 S에서 top이 마지막 자료를 가리키고 있으므로 그 위에 자료를 삽입하려면 먼저 top의 위치를 하나 증가
- 만약 이때 top의 위치가 스택의 크기(stack_SIZE)보다 크다면 오버플로우(overflow)상태가 되므로 삽입 연산을 수행하지 못하고 연산 종료

② S(top) ← x;

- 오버플로우 상태가 아니라면 스택의 top이 가리키는 위치에 x 삽입

❖ 스택의 pop() 알고리즘

① `return S(top);`

- 공백 스택이 아니면 top에 있는 원소를 삭제하고 반환

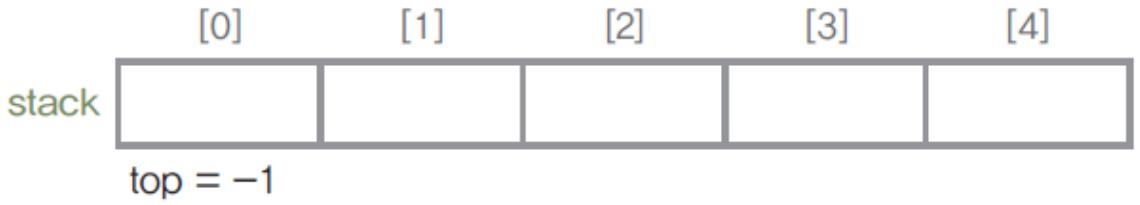
② `top ← top-1;`

- 스택의 마지막 원소가 삭제되면 그 아래 원소, 즉 스택에 남아 있는 원소 중에서 가장 위에 있는 원소가 top이 되어야 하므로 top 위치를 하나 감소

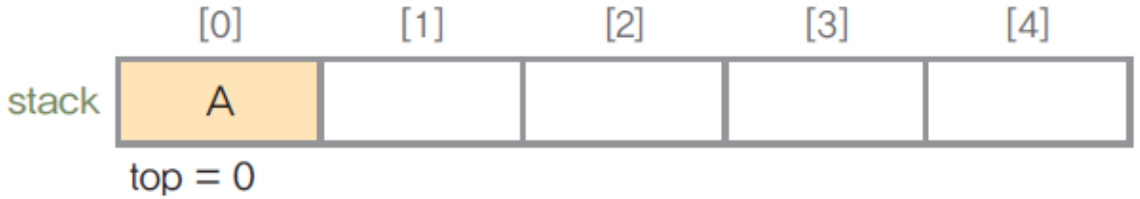
❖ 순차 자료구조를 이용한 스택의 구현

- ㅇ 순차 자료구조인 1차원 배열을 이용하여 구현
  - 스택의 크기 : 배열의 크기
  - 스택에 저장된 원소의 순서 : 배열 원소의 인덱스
    - 인덱스 0번 : 스택의 첫번째 원소
    - 인덱스 n-1번 : 스택의 n번째 원소
  - 변수 top : 스택에 저장된 마지막 원소에 대한 인덱스 저장
    - 공백 상태 : top = -1 (초기값)
    - 포화 상태 : top = n-1

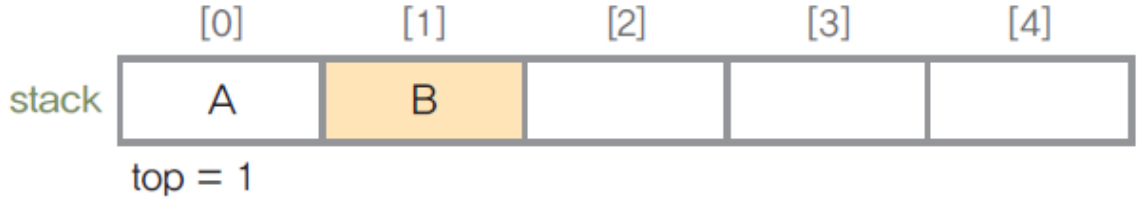
1 공백 스택 생성 : createStack(S, 5);



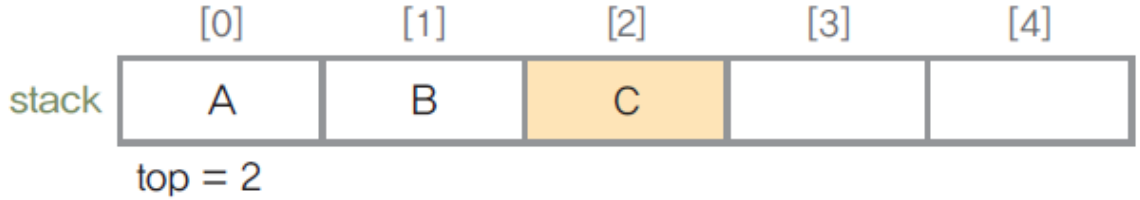
2 원소 A 삽입 : push(S, A);



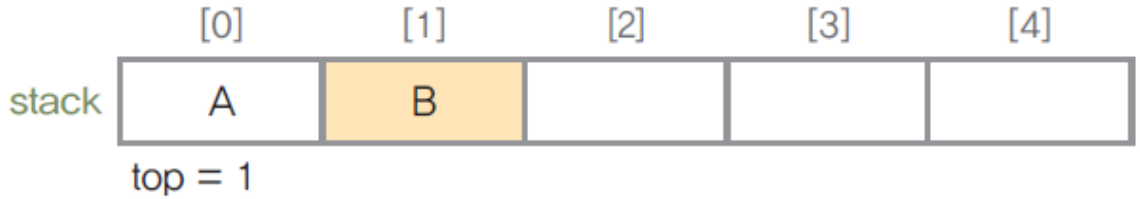
3 원소 B 삽입 : push(S, B);



4 원소 C 삽입 : push(S, C);



5 원소 삭제 : pop(S);



❖ **순차 자료구조로 구현한 스택의 장점**

- 순차 자료구조인 1차원 배열을 사용하여 쉽게 구현

❖ **순차 자료구조로 구현한 스택의 단점**

- 물리적으로 크기가 고정된 배열을 사용하므로 스택의 크기 변경 어려움
- 순차 자료구조의 단점을 가짐

```
public int search(int[] arr, int target) {
    int first = 0;
    int last = arr.length;
    int mid;

    while (first <= last) {  // 검색 종료 조건
        mid = (first + last) / 2;
        if (target == arr[mid]) {  // 찾음
            return mid;            // 검색된 인덱스 리턴
        } else {
            if (target < arr[mid]) {
                last = mid - 1;
            } else {
                first = mid + 1;
            }
        }
    }
    return -1;  // 검색 실패, target이 없음
}
```