2021 여름학기 동국대학교 SW역량강화캠프

4일차. 자료구조 1(스택, 큐)

● 자료구조(Data Structure)

▶ 데이터를 효율적으로 접근하고 수정하기 위한 데이터의 저장 방식

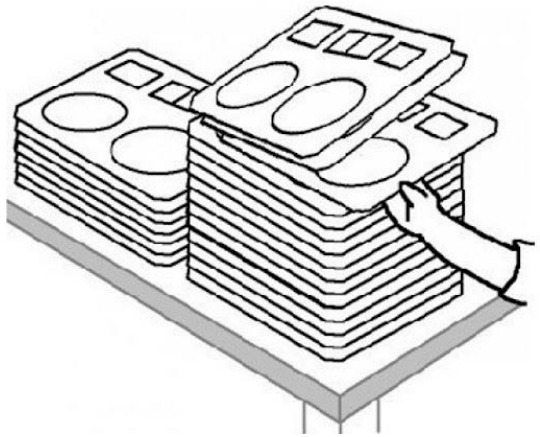
▶ 배열, 리스트, 스택, 큐, 덱, 힙, 트리맵 등등

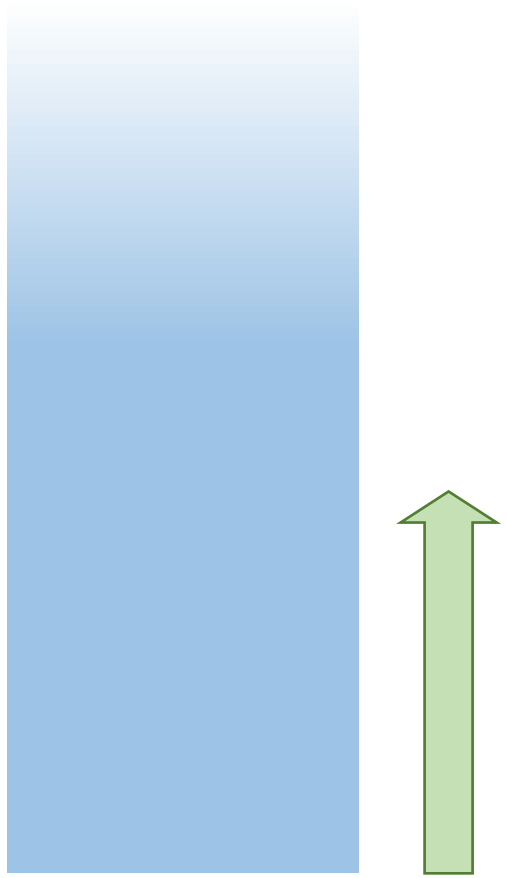
● 스택(Stack)

▶ 접시 쌓기

▶ 1번 → 2번 → 3번 순서로 쌓았다면, 꺼낼 때에는 3번 → 2번 → 1번

▶ FILO(First In Last Out) , LIFO(Last In First Out)





● 스택 (101)

문제

스택을 배운 서영이는 직접 스택 프로그램을 구현하고 싶어졌다.

서영이가 설계한 프로그램의 명령어들은 아래와 같다.

push x : 정수 x를 출력하고, 그 수를 스택에 넣는다. (x는 int형 범위 이내의 수이다.)

pop : 스택에서 가장 위에 있는 정수를 빼고, 그 수를 출력한다. 만약 스택에 들어있는 정수가 없는 경우에는 -1을 출력한다.

size: 스택에 들어있는 정수의 개수를 출력한다.

empty: 스택이 비어있으면 1, 아니면 0을 출력한다.

top: 스택의 가장 위에 있는 정수를 출력한다. 만약 스택에 들어있는 정수가 없는 경우에는 -1을 출력한다.

end : 프로그램을 종료한다.

서영이를 도와 위의 명령어를 수행하는 프로그램을 작성해보자. (유효한 명령어만 주어진다고 한다.)

▶ Stack 자료구조는 java.util.Stack 에 정의되어 있습니다.

```
import java.util.Stack;
```

▶ Stack 자료구조는 다음과 같이 선언할 수 있습니다.

```
Stack<Integer> stack = new Stack<Integer>();
// stack은 이렇게 선언합니다.
// 참조형 변수로만 선언 가능합니다 int(X) char(X) Integer(O) Character(O)
// Stack<object> stack_name = new Stack<object>();
```

핵심 코드

▶ Stack에 들어있는 원소의 수는 size 메소드를 통해 알 수 있습니다.

```
if(str.equals("size")) {
    int x = stack.size(); // 스택에 쌓여있는 원소의 개수를 x에 저장합니다.
    sb.append(x+"\n");
}
```

▶ Stack이 비어 있는 지의 여부는 empty 메소드를 통해 알 수 있습니다.

```
if(str.equals("empty")) {
    Boolean isEmpty = stack.empty(); // isEmpty에 스택이 비어있는지 여부를 저장합니다.
    if(isEmpty) sb.append("1\n");
    else sb.append("0\n");
}
```

핵심 코드

▶ Stack에 값을 쌓는 것은 push 메소드를 이용하면 됩니다. (add 함수도 동일합니다)

```
if(str.equals("push")) {
    int x = Integer.parseInt(st.nextToken());
    stack.push(x); // stack에 x를 쌓습니다.
    sb.append(x+"\n");
}
```

▶ Stack에서 가장 위에 쌓여 있는 값은 peek 메소드를 이용하여 접근할 수 있습니다.

```
if(str.equals("top")) {
    if(stack.empty()) { // 문제 조건에 따라 스택이 비어있다면 -1을 출력합니다.
        sb.append("-1\n");
        continue;
    }
    int x = stack.peek(); // 스택의 맨 위에 쌓인 값을 x에 저장합니다.
    sb.append(x+"\n");
}
```

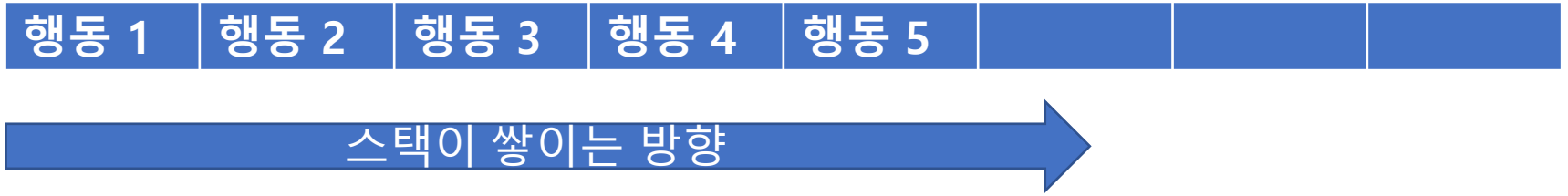
핵심 코드

▶ Stack에서 값을 빼내는 것은 pop 메소드로 할 수 있습니다.

```
if (str.equals("pop")) {
    if (stack.empty()) { // 문제 조건에 따라 스택이 비어있다면 -1을 출력합니다.
        sb.append("-1\n");
        continue;
    }
    int x = stack.pop(); // 스택의 맨 위에 쌓인 값을 x에 저장하고 스택에서 제거합니다.
    /*
     * 또는 다음과 같이 쓸 수도 있습니다.
     * int x = stack.peek(); // 스택의 맨 위에 쌓인 값을 x에 저장합니다.
     * stack.pop(); // 스택의 맨 위에 쌓인 값을 제거합니다.
     */
    sb.append(x + "\n");
}
```

● 스택의 활용

▶ 구현 문제에서는 계산식 파싱, undo/redo 구현에 주로 쓰임



▶ 문자열/수열에서 값을 하나씩 수열에 넣어가면서 원하는 값만을 남길 때 사용

▶ 필요에 따라 최근 쌓인 여러 개의 값을 보기 위해 배열로 스택을 구현해야 할 때도 있다.

● 방금 건 취소(2782)

문제

헬로알고 홍선기 선생님은 윌리와 함께 월간평가 문제 제작을 위해 테스트 케이스를 만들고 있다. 선생님이 숫자를 부르면 윌리가 이것을 타이핑 하는 형태이다. 하지만 선기쌤은 변덕이 많기 때문에 가끔씩 0을 외쳐서 방금 부른 숫자를 취소하라고 시킨다. 윌리는 이렇게 모든 수를 받아 적은 후 그 합을 알고 싶어 한다. 윌리를 도와주자!

입력

첫 번째 줄에 정수 K가 주어진다. (1 ≤ K ≤ 100,000)

이후 K개의 줄에 정수가 1개씩 주어진다. 정수는 0에서 1,000,000 사이의 값을 가지며, 정수가 "0" 일 경우에는 가장 최근에 쓴 수를 지우고, 아닐 경우 해당 수를 쓴다.

정수가 "0"일 경우에 지울 수 있는 수가 있음을 보장할 수 있다.

●스택을 이용한 풀이 (undo 구현)

▶ 수를 더할 때마다, 더하는 수를 stack에 push한다.

▶ 만약 0이 나온다면, 가장 최근의 수를 빼는 것이므로 stack에서 pop하고 답에서 뺀다.

입력	1	3	5	4	0	0	7	0	0	6

스택				

핵심 코드

```
int ans = 0;
while(K-- > 0){
    int x = Integer.parseInt(br.readLine()); // 입력 받은 수 x
    if(x != 0) { // x가 0이 아니라면, stack에 x를 넣고, 답에 x를 더한다.
        stack.push(x);
        ans += x;
    }
    else { // 0이 입력으로 들어오면, stack에서 수를 빼내고, 빼낸 수만큼 답에서 빼준다.
        int y = stack.pop();
        ans -= y;
    }
}
```

● 괄호 (102)

문제

경섭이와 예인이는 괄호를 무척 사랑한다.

그러나 모든 괄호를 사랑하는 예인이와 달리 경섭이는 정상적인 괄호만을 좋아한다.

정상적인 괄호는 다음과 같은 성질을 만족할 때, 비로소 정상적인 괄호라고 한다.

한쌍의 괄호 () 는 정상적인 괄호이다.

X 가 정상적인 괄호라면, (X) 도 정상적인 괄호이다.

X 와 Y 가 정상적인 괄호면, XY 도 정상적인 괄호이다.

예인이와 경섭이는 서로 앙숙인데, 특히 경섭이는 예인이가 만든 비정상적인 괄호(정상적인 괄호를 제외한 모든 괄호를 비정상적인 괄호라 한다.)를 보면 매우 화를 낸다고 한다. 경섭이와 예인이의 친구 서영이는 경섭이가 화를 내지 않도록 사전에 비정상적인 괄호를 없애려고 한다.

서영이를 도와 예인이가 괄호를 그렸을 때, 이 괄호가 정상적인 괄호인지 알려주자.

● 스택을 이용한 풀이

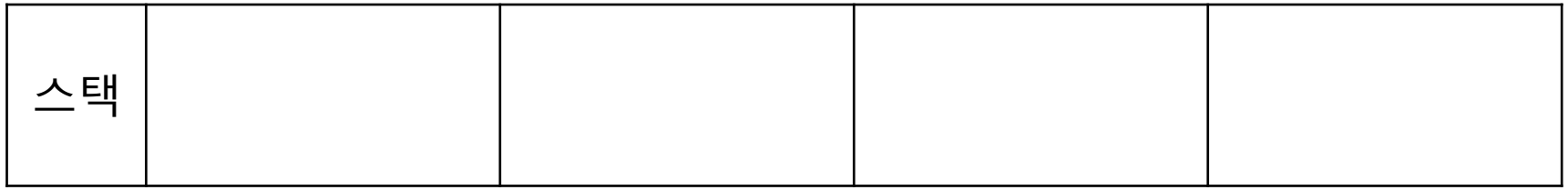
▶ 정상적인 괄호문자열은 다음과 같은 특징을 가진다.

1. 문자열에 등장하는 '(' 와 ')'의 개수가 같다.
2. 문자열의 모든 ')'는 보다 앞에 등장하는 '('와 일대일 매칭된다.

● 스택을 이용한 풀이

▶ ‘)’가 입력으로 들어올 때마다, 앞에 매칭될 수 있는 ‘(‘가 있는지 확인하고, 있다면 매칭

▶ ‘(‘ 가 들어온다면 스택에 ‘(‘를 넣고, ‘)’가 들어온다면 스택에 ‘(‘가 있는지 확인하고 있다면 매칭시킨다.



핵심 코드

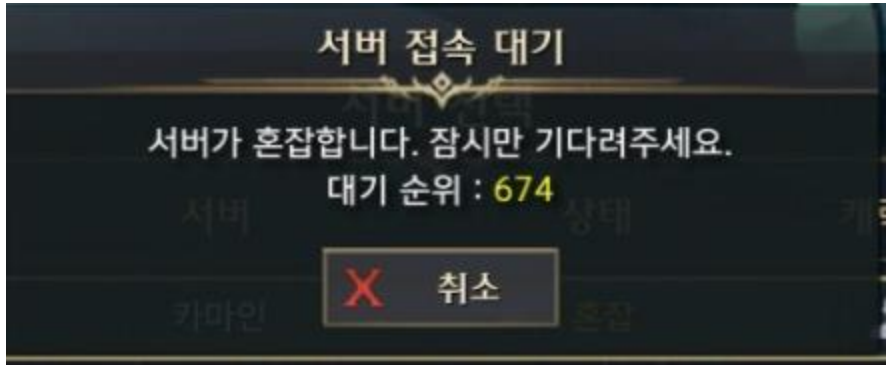
```
Stack<Character> stack = new Stack<Character>(); // 스택은 매칭시킬 수 있는 여는 괄호를 담고 있다.
Boolean pos = true;
for (int i = 0; i < str.length; i++) {
    if (str[i] == '(') { // 여는 괄호는 스택에 push한다.
        stack.push('(');
    }
    else { // 닫는 괄호는 매칭시킬 수 있는 여는 괄호가 있다면, 매칭시킨다.
        if (stack.empty()) { // 만약 매칭시킬 수 있는 여는 괄호가 없다면, 정상적인 괄호문자열이 될 수 없다.
            pos = false;
            break;
        }
        else
            stack.pop(); // 매칭이 완료된 여는 괄호는 stack에서 제거한다.
    }
}

if (pos == true && stack.empty()) {
    System.out.println(":)");
}
```

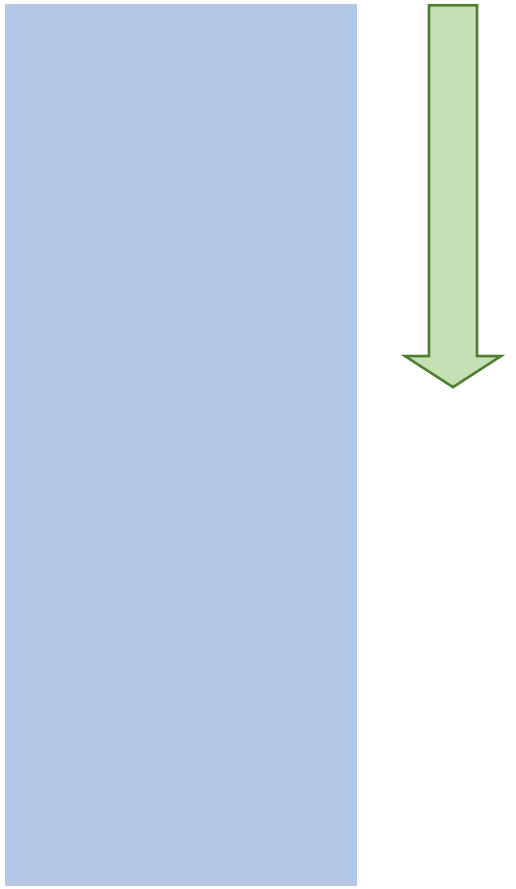
● 큐

▶ 대기열

▶ 1번 → 2번 → 3번 순서로 넣었다면, 꺼낼 때에도 1번 → 2번 → 3번

▶ FIFO(First In First Out)





● 직접 만든 큐 (121)

문제

준함이는 STL에서 지원하는 큐 기능을 믿지 않는다.
그러나 원하는 프로그램을 짜기 위해 큐 기능이 어떻게든 필요했던 준함이는 당신에게 큐 기능을 구현해 줄 것을 요청했다.
준함이가 원하는 큐 기능은 다음 명령들을 처리하는 것이다.

```
push X: 정수 X를 큐에 넣는다.
pop: 큐에서 가장 앞에 있는 수를 빼고, 그 수를 출력한다. 만약, 큐에 들어있는 정수가 없는 경우에는 -1을 출력한다.
size: 큐에 들어있는 정수의 개수를 출력한다.
empty: 큐가 비어있으면 1을, 아니면 0을 출력한다.
front: 큐의 가장 앞에 있는 정수를 출력한다. 만약 큐에 들어있는 정수가 없는 경우에는 -1을 출력한다.
back: 큐의 가장 뒤에 있는 정수를 출력한다. 만약 큐에 들어있는 정수가 없는 경우에는 -1을 출력한다.
```

준함이를 위해 큐 기능을 구현해주자

▶ Queue 자료구조는 java.util.Queue 에 정의되어 있습니다.
하지만 Queue는 LinkedList로 구현되어 있기에, LinkedList도 import 해주어야 합니다.

```
import java.util.Queue;
import java.util.LinkedList;
```

▶ Queue 자료구조는 다음과 같이 선언할 수 있습니다.

```
Queue<Integer> queue = new LinkedList<>();
// Queue는 LinkedList로 구현되어있습니다.
// 참조형 변수로만 선언 가능합니다.
// Queue<Object> queue_name = new LinkedList<>();
```

핵심 코드

▶ queue에 들어있는 원소의 수는 size 메소드를 통해 알 수 있습니다.

```
if(str.equals("size")) {
    int x = queue.size(); // queue에 들어있는 원소의 개수를 x에 저장합니다.
    sb.append(x+"\n");
}
```

▶ queue가 비어 있는 지의 여부는 isEmpty 메소드를 통해 알 수 있습니다.

```
if(str.equals("empty")) {
    Boolean isEmpty = queue.isEmpty(); // isEmpty에 큐가 비어있는지 여부를 저장합니다.
    if(isEmpty) sb.append("1\n");
    else sb.append("0\n");
}
```

핵심 코드

▶ Queue에 값을 넣는 것은 offer 메소드를 이용하면 됩니다. (add 함수도 동일합니다)

```
if(str.equals("push")) {
    int x = Integer.parseInt(st.nextToken());
    queue.offer(x); // queue에 x를 넣습니다.
    back = x;
}
```

▶ Queue에서 가장 먼저 들어간 값은 peek 메소드를 이용하여 접근할 수 있습니다.

```
if(str.equals("front")) {
    if(queue.isEmpty()) { // 문제 조건에 따라 큐가 비어있다면 -1을 출력합니다.
        sb.append("-1\n");
        continue;
    }
    int x = queue.peek(); // 큐에 가장 먼저 들어간 값을 x에 저장합니다.
    sb.append(x+"\n");
}
```

핵심 코드

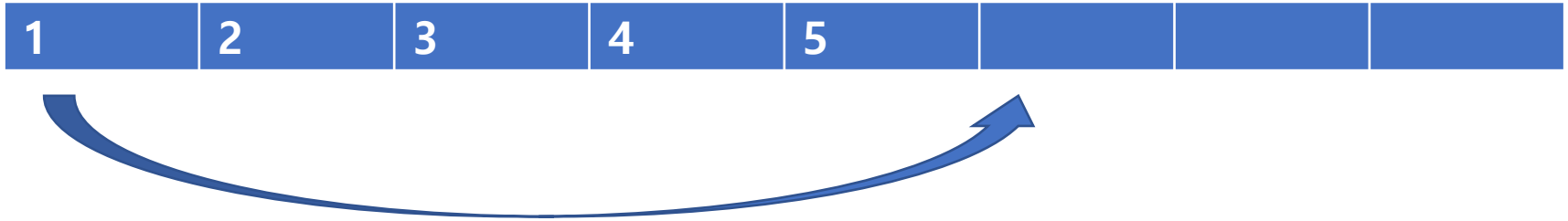
▶ Queue에서 값을 빼내는 것은 poll/remove 메소드로 할 수 있습니다.

```
if (str.equals("pop")) {
    if (queue.isEmpty()) { // 문제 조건에 따라 큐가 비어있다면 -1을 출력합니다.
        sb.append("-1\n");
        continue;
    }
    int x = queue.poll(); // 큐에 가장 먼저 들어간 값을 x에 저장하고, 큐에서 제거합니다.
    /*
     * 또는 다음과 같이 쓸 수도 있습니다
     * int x = queue.peek(); // 큐에 가장 먼저 들어간 값을 x에 저장합니다.
     * queue.remove(); // 큐에 가장 먼저 들어간 값을 큐에서 제거합니다. queue.poll(); 을 사용해도 됩니다.
     */
    sb.append(x + "\n");
}
```

▶ Java에서는 queue에서 back의 기능을 하는 메소드가 구현되어 있지 않습니다.

● 큐의 활용

▶ 구현 문제에서는 대기열, 맨 뒤로 보내기 연산 등을 구현할 때 쓰임



▶ 큐는 보통 구현에 활용되기보다는 BFS에서 사용된다.

● 조세퍼스 문제 (5)

문제

조세퍼스 문제는 다음과 같다.

1번부터 N번까지 N명의 사람이 원을 이루면서 앉아있고, 양의 정수 M(≤ N)이 주어진다. 이제 순서대로 M번째 사람을 제거한다. 한 사람이 제거되면 남은 사람들로 이루어진 원을 따라 이 과정을 계속해 나간다. 이 과정은 N명의 사람이 모두 제거될 때까지 계속된다. 원에서 사람들이 제거되는 순서를 (N, M)-조세퍼스 순열이라고 한다. 예를 들어 (7, 3)-조세퍼스 순열은 3, 6, 2, 7, 5, 1, 4 이다.

N과 M이 주어지면 (N,M)-조세퍼스 순열을 구하는 프로그램을 작성하시오.

● 큐를 이용한 풀이

▶ M번째 수를 꺼내는 작업은 다음과 같습니다.

1. 맨 앞 수를 맨 뒤로 보내는 작업을 M-1번 반복합니다.
2. 맨 앞 수를 큐에서 제거하고 출력합니다.

1	2	3	4	5	6	7

4	5	6	7	1	2

7	1	2	4	5

핵심 코드

```
for (int i = 1; i <= N; i++) { // 큐에 1~N의 수를 차례로 넣는다.
    queue.offer(i);
}

while (!queue.isEmpty()) { // 큐가 빌 때까지 반복한다.
    for (int i = 1; i < M; i++) { // M-1개는 맨 뒤로 보내고
        int x = queue.poll();
        queue.offer(x);
    }
    int x = queue.poll(); // M번째는 빼내서 출력한다.
    sb.append(x + " ");
}
```