후위 표기식 1. (infix → postfix).

— 문제 · infix가 주어질때 (괄호 포함) ⟶ postfix 출력.

— 접근

[0] [1] [2] [3] [4]

1 + 2 * 3 ⟶ 1 2 3 * +

1 + 2 + 3 ⟶ 1 2 + 3 +.

↑

[3]에 오는 연산자의 우선순위에 따라 결과가 다름을 알수있다.

* [1] 연산자는 [3] 연산자에 의해 pop순서가 결정된다.

([3]보다 우선순위 크거나 같을 때만 먼저 pop된다.)

① 숫자를 만나면 결과 S에 추가한다.

② 연산자를 만나면 해당 연산자는 (뒤에 오는 피연산자보다 뒤에 오는것이 postfix이므로) 무조건 스택으로 들어갈 것이다. (아직)

③ 해당 연산자를 넣기 전에 스택 내에 그보다 우선순위가 더 높은 연산자들이 있다면 (같거나) 해당 연산자보다 앞에 와야한다는 뜻.

⇒ 해당 연산자 보다 우선순위 더 높거나 같은 애들 다 빼서 추가함.

but, 괄호는 어떻게 처리?

④ '('를 우선순위 가장 낮게하면 스택에서 가장 늦게 빠질것.

⇒ ')'를 만나야만 빠질 수 있다.

'('를 새로운 식의 시작이라고하면 각 괄호를 부분식으로 분리할수 있다.

1 + 2 * (3 + 4) 에서

↑

얘를 벽처럼 두면 ')'를 만나기 전까지는 해당 부분식이 끝나지 않게 막을 수 있다. (가장 우선순위 낮으므로 맨 마지막에 나옴)

⇒ '('만났을 때는 부분 식의 시작이므로 강제로 스택에 추가한다.

- 구현.

· Infix의 앞뒤에 '(', ')'를 추가하고 시작하면 후처리 없이 반복문 돌릴 수 있다. (스택 비는지 체크도 필요X).

· 각 단계의 연산자 ('(', '+', '-', '*', '/')의 우선순위를 반환하는 간단한 함수를 만들어두면 좋다. (')'는 어차피 all pop).

```
int pr(chard c) {
    if(c=='(') return 0;
    else if(c=='+'||c=='-') return 1;
    else return 2;
}
```

· 숫자 만나면 바로 print.

· '(' 만나면 바로 push.

· ')' 만나면 '(' 만날 때 까지 연산자 All pop. (print)

· 다른 연산자는 연산자 우선순위 그보다 크거나 같은 것들 All pop.

```
while(pr(s.top())>=pr(c)){
    cout<<s.top();
    S.pop();
}
S.push(c); // 해당 연산자는 다시 넣어줌.
```

* 사실 스택은 empty 체크가 상당히 중요하다.
but, 앞뒤로 '(', ')' 추가했기 때문에 실수를 막는 효과도 있다.

후위 표기식 2 (postfix → 결과) Easy.
계산.

- 구현

1 2 + 3 4 * -

↳ 숫자 만나면 스택에 넣고, 모든 연산자는 이항 연산자이므로
연산자 만나면 두개 꺼내서 (먼저 b, a 순서로 나옴)
(a □ b)를 다시 push해주면 끝. → 마지막에 top() 이 결과.