

# 03. Stack, Queue, Deque

Div. 3 알고리즘 스터디 / 임지환



# #2805 나무 자르기

- 적당히 높이 설정을 한 후 잘려진 부분을 챙긴다.
- 잘려진 부분의 합이 최소 M미터는 되야 한다.
- 위의 조건을 지키며 최대한 덜 잘라야 한다.



# #2805 나무 자르기

가장 길이가 긴 나무의 높이부터 시작해서 1씩 낮춰가며 시도해본다?



# #2805 나무 자르기

가장 길이가 긴 나무의 높이부터 시작해서 1씩 낮춰가며 시도해본다?

- 나무의 개수가 최대 $10^6$개, 주어지는 나무의 최대 높이는 $10^9$
- 한번의 시도마다 최대 $10^6$ 번의 연산이 주어지고,
- 이걸 1000번만 반복해도 $10^9$ 번의 연산
- 1000번보다 더 해야 할걸요?



# #2805 나무 자르기

- 이분탐색 이용

  절단기의 높이를 높게 설정하면 덜 가져가고, 낮게 설정하면 더 가져가게 되므로

  절단기의 설정 가능 높이 : 나무의 길이, 즉 $0 \leq h \leq 10^9$

  탐색 범위가 주어졌으므로 이분탐색 실행시 시간복잡도 : $O(\log n)$

$$\log 10^9 \approx \log(2^{10})^3 = \log 2^{30} \approx 30$$



```
#include <stdio.h>
typedef long long lint;
int arr[1000000];
int main() {
    int n, m;
    scanf("%d%d", &n, &m);
    for (int i = 0; i < n; i++) scanf("%d", arr + i);

    int Min = 0, Max = 1000000000;
    while (Min + 1 < Max) {        //break when Min +1 == Max or over
        int mid = (Min + Max) / 2;
        lint sum = 0;
        for (int i = 0; i<n; i++)
            if(arr[i] > mid)
                sum += arr[i]-mid;
        if (sum >= m) Min = mid;
        else Max = mid;
    }
    printf("%d\n", Max-1);
}
```



# #5052 전화번호 목록

"한 번호가 다른 번호의 접두어인 경우가 없어야 한다."



# #5052 전화번호 목록

Ex)

911

97625999

91125426

에서 “911”은 “91125426”의 접두사이므로 일관성이 없다.



# #5052 전화번호 목록

1. 주어진 번호들을 사전순 정렬
2. 현재 보고 있는 번호와 바로 이전에 있는 번호를 비교
    1) 일관성이 있을 경우, 이전에 있는 번호가 현재 번호의 접두사가 될 수 없음
    2) 일관성이 없을 경우 이전에 있는 번호가 현재 번호의 접두사가 됨



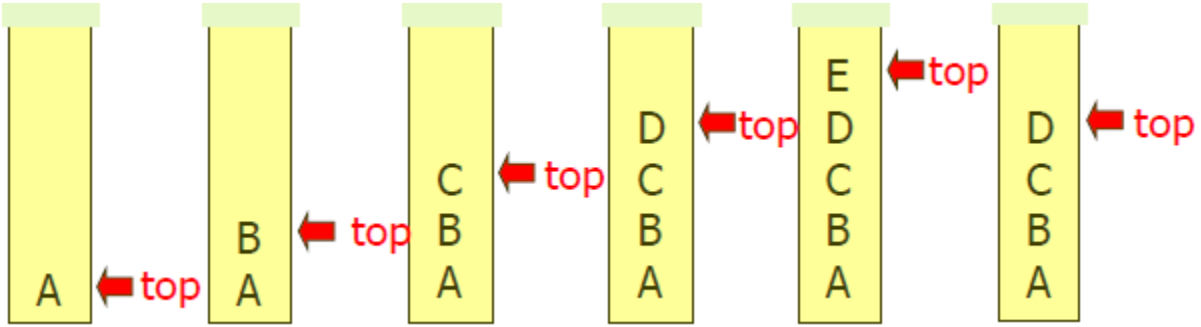
```
bool is_consistent() {
    for (int i = 1; i < n; i++) {
        bool flag = false;
        int Size = str[i - 1].size();
        for (int j = 0; j < Size; j++) {
            if (str[i][j] != str[i - 1][j])
                flag = true, break;
        }
        if (!flag) return false;
    }
    return true;
}
```

```
int main() {
    for (scanf("%d", &tc); tc--;) {
        input();
        sort(str, str + n);
        if (is_consistent())
            cout<<"YES\n";
        else cout << "NO\n";
    }
    return 0;
}
```

# Stack

- 마지막에 들어간 원소가 가장 먼저 나오는 자료구조
- LIFO list = stack

  Last-In First-Out





# 스택 연산

- top( ) : 맨 위의 값에 접근
- push( ) : 맨 위에 새로운 값 추가
- pop( ) : 맨 위의 값 제거



# STL을 통한 스택 사용법

```
stack <int> st;          // 스택 자료구조 선언
int n = st.top();        // top에 위치한 값 return
st.push(1);              ┐
st.push(n);              ┘  push 연산
st.pop();                // pop 연산
st.empty();              // 비었을 경우 true, 아닐 경우 false
```

# STL을 통한 스택 사용법

```
stack <int> st;          // 스택 자료구조 선언
int n = st.top();        // top에 위치한 값 return
st.push(1);
st.push(n);
st.pop();                // pop 연산
st.empty();              // 비었을 경우 true, 아닐 경우 false
```

push 연산

Error occurs when empty!



# #9012 괄호

- 올바르게 짝 지어진 괄호 문자열인지 판별

## 입력

입력 데이터는 표준 입력을 사용한다. 입력은 T개의 테스트 데이터로 주어진다. 입력의 첫 번째 줄에는 입력 데이터의 수를 나타내는 정수 T가 주어진다. 각 테스트 데이터의 첫째 줄에는 괄호 문자열이 한 줄에 주어진다. 하나의 괄호 문자열의 길이는 2 이상 50 이하이다.

## 출력

출력은 표준 출력을 사용한다. 만일 입력 괄호 문자열이 올바른 괄호 문자열(VPS)이면 “YES”, 아니면 “NO”를 한 줄에 하나씩 차례대로 출력해야 한다.



# #9012 괄호

- 예시 입력

(())())

(((()())()

(()())((()))

((()()(()))(((())))()

()()()()(())()())()

((()(()))()(



# #9012 괄호

1. 그냥 개수만 똑같아서는 되지 않는다. ( ex : ")(" )
2. 대응하는 왼쪽 괄호가 있으면 오른쪽 괄호를 넣어도 문제가 없다.

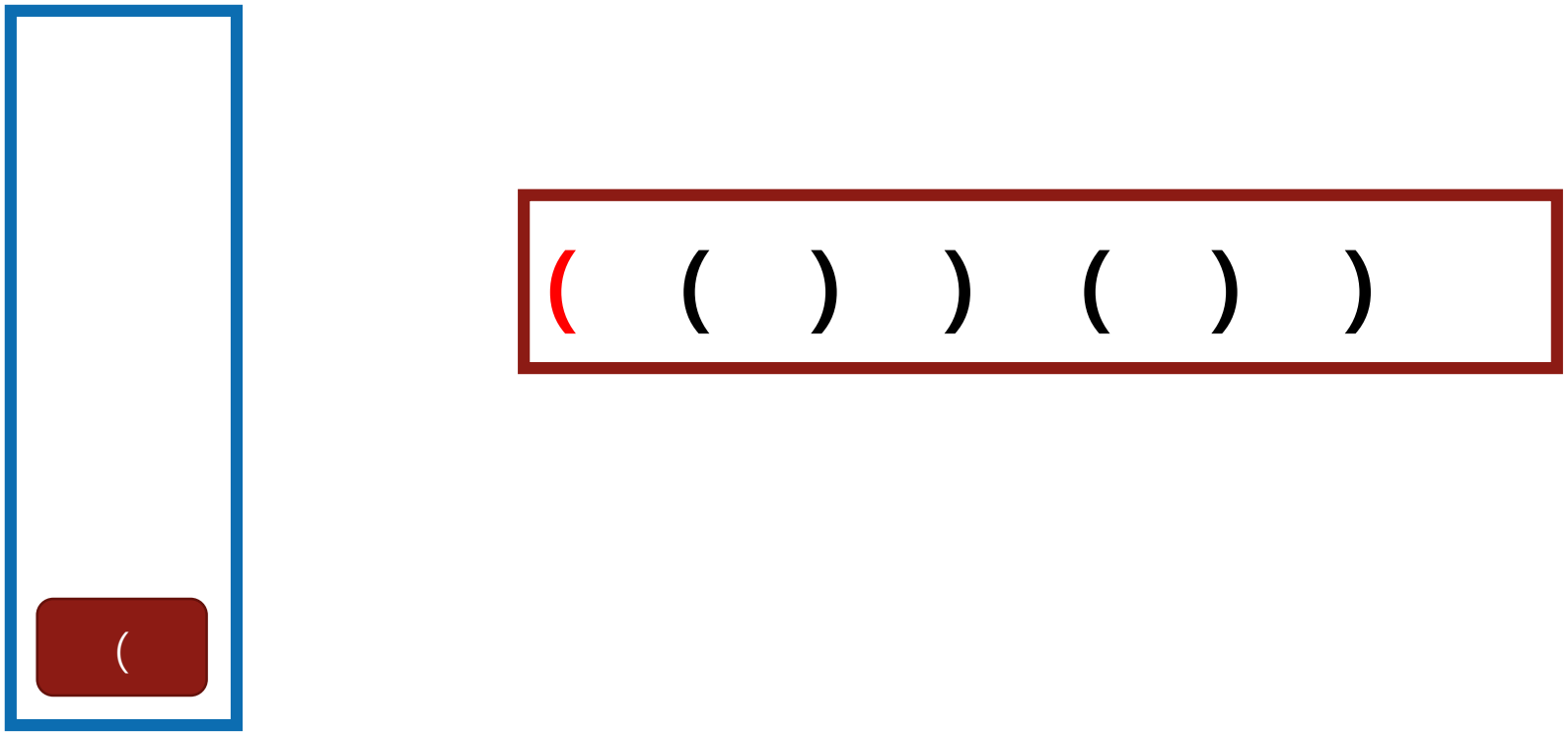


# #9012 괄호

1. 그냥 개수만 똑같아서는 되지 않는다. ( ex : ")(" )
2. 대응하는 왼쪽 괄호가 있으면 오른쪽 괄호를 넣어도 문제가 없다.
3. **쌍이 맞은 괄호가 있으면 지워도 된다!**

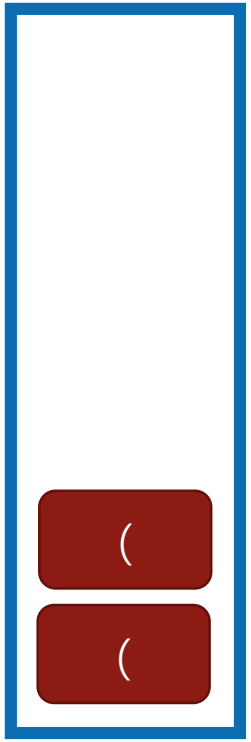
# #9012 괄호

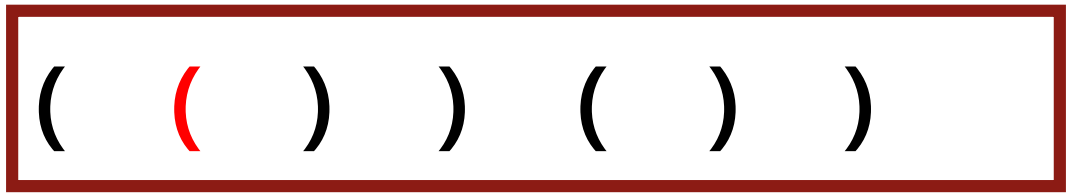
ex 1. “( ( ) ) ( ) )”



# #9012 괄호

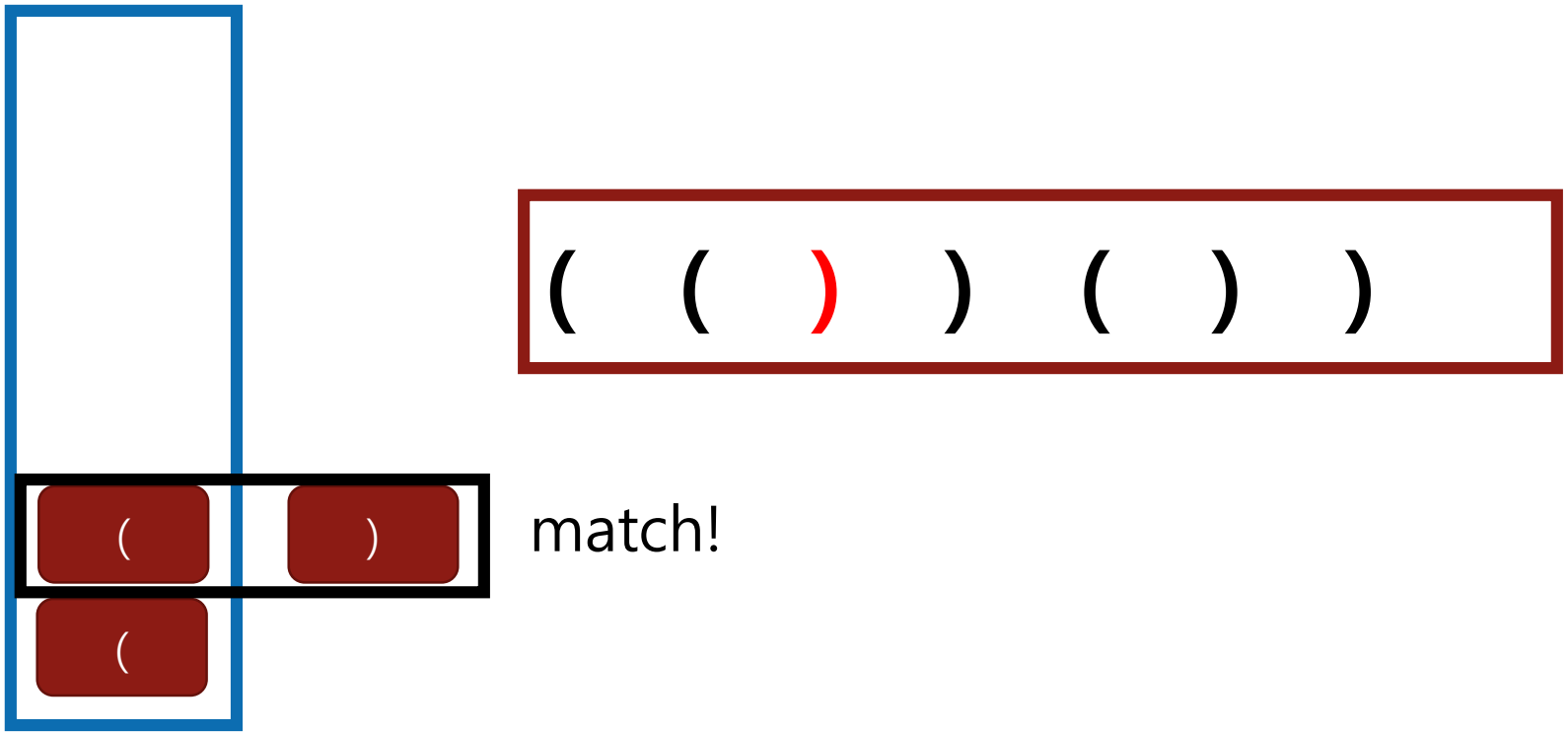
ex 1. “( ( ) ) ( ) )”



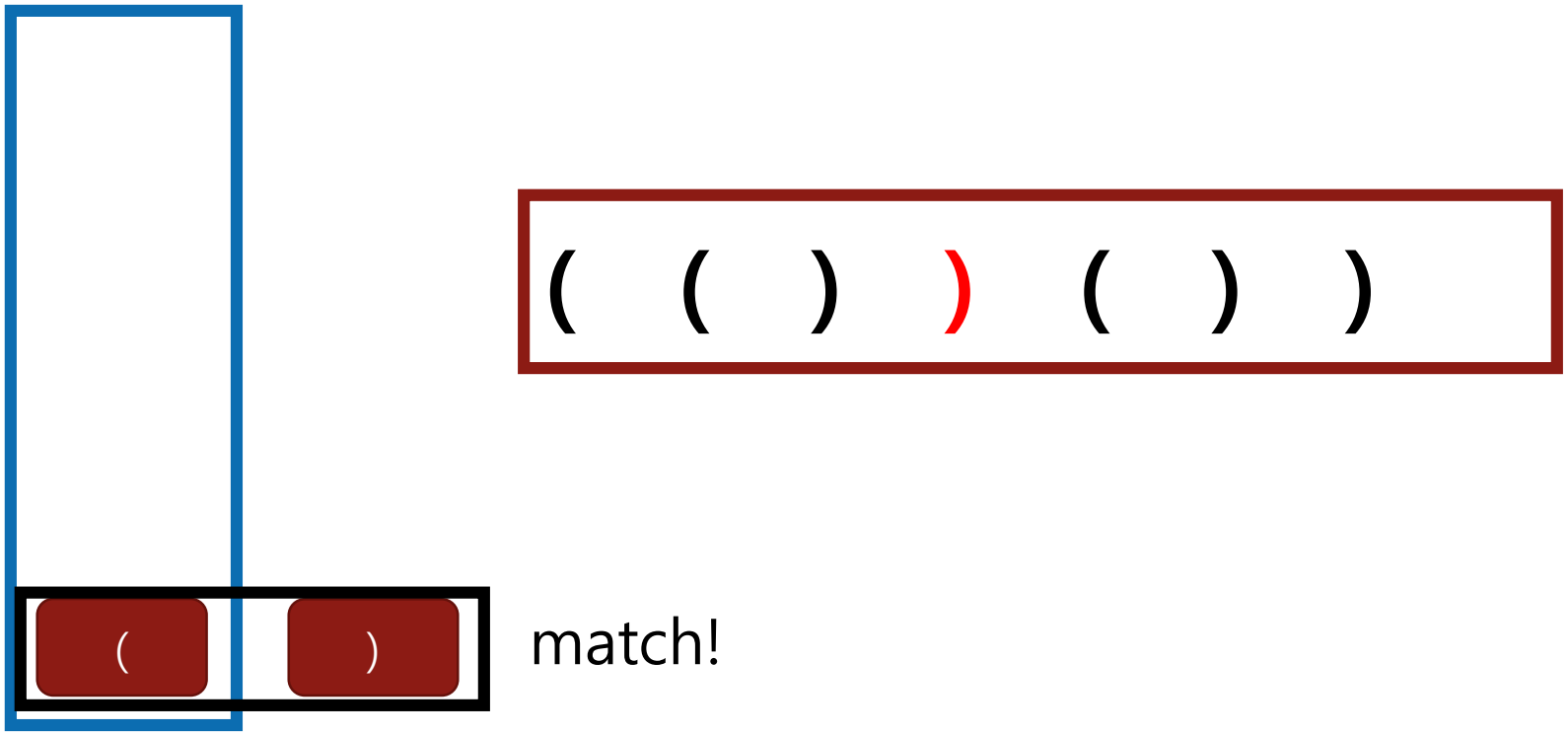
# #9012 괄호

ex 1. “( ( ) ) ( ) )”



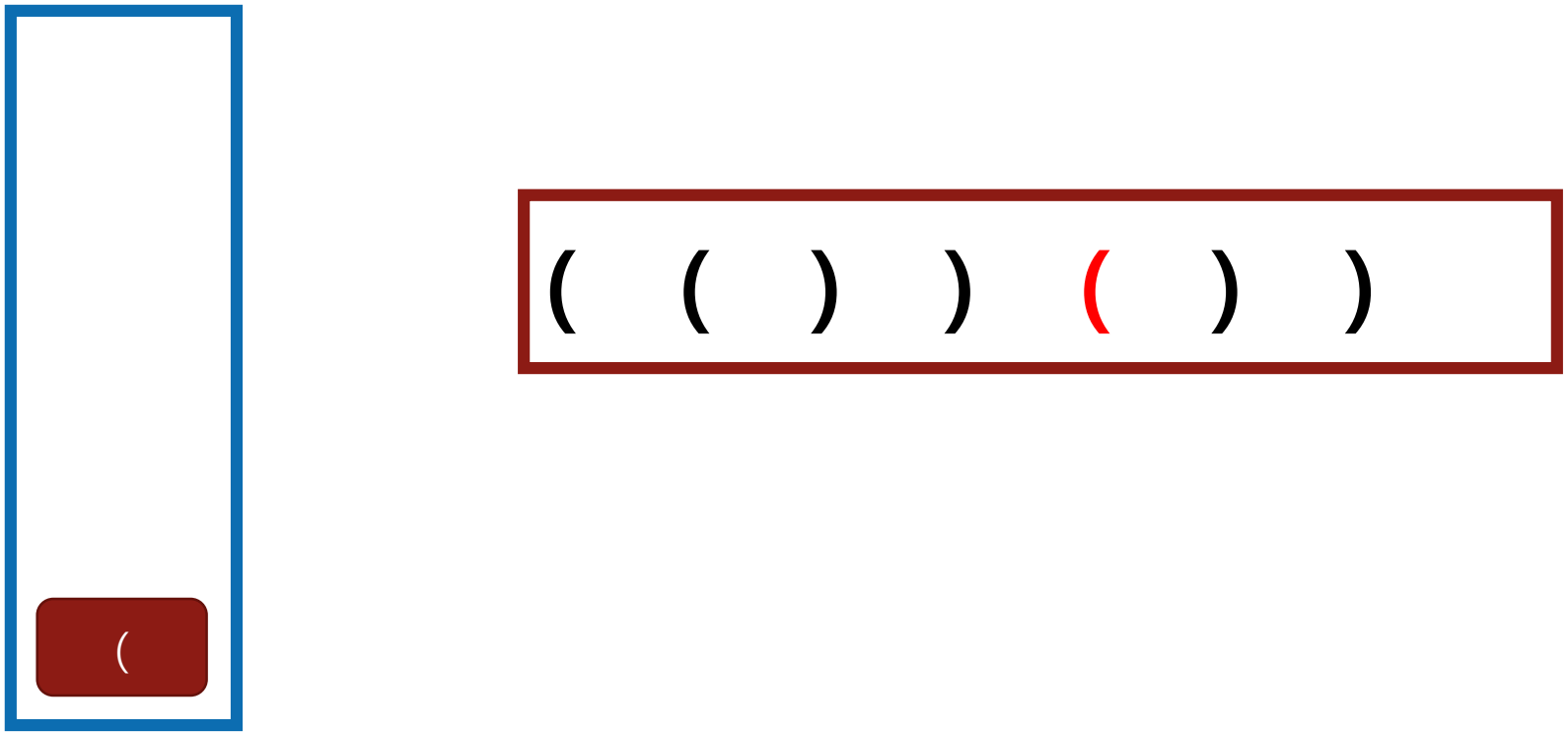
# #9012 괄호

ex 1. “( ( ) ) ( ) )”



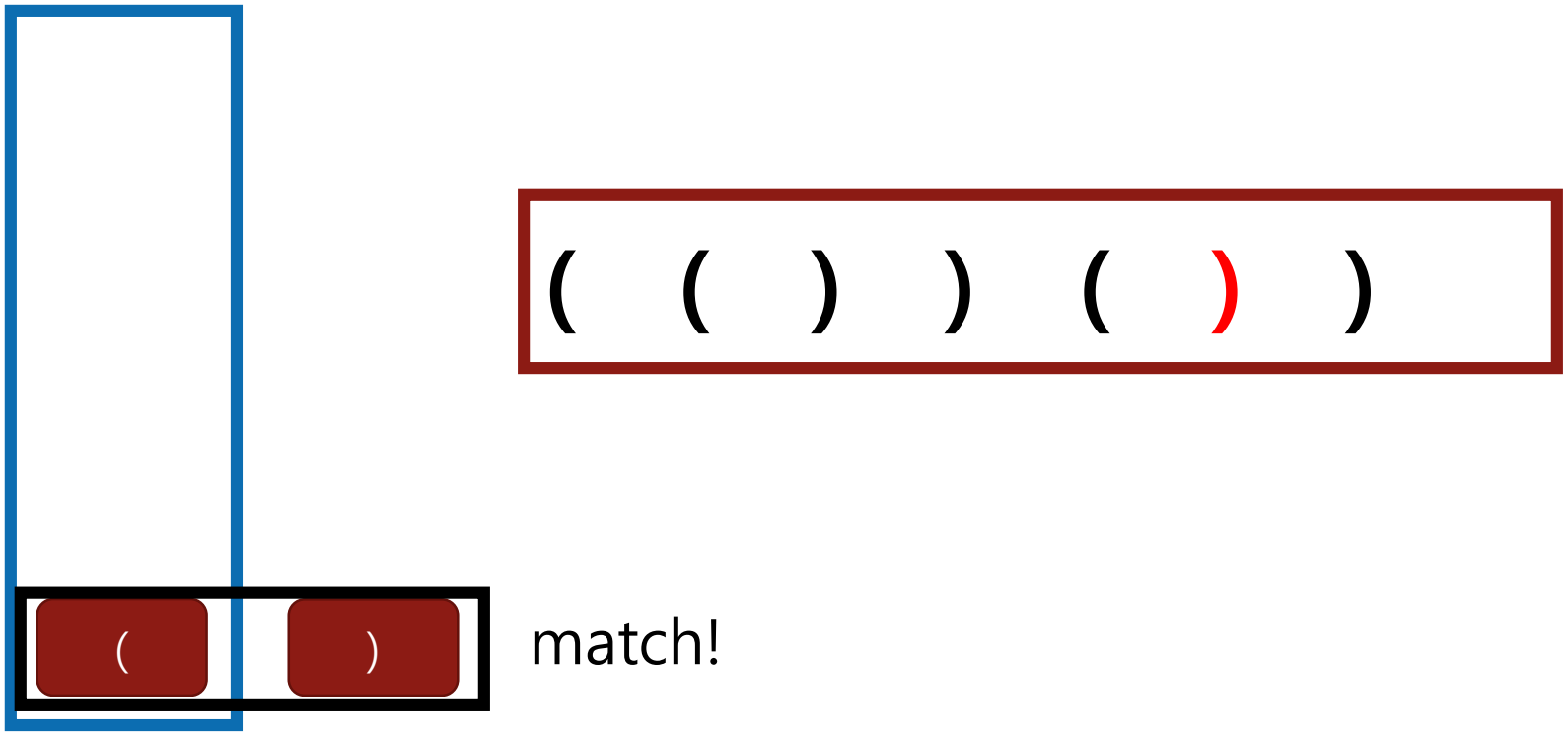
# #9012 괄호

ex 1. “( ( ) ) ( ) )”



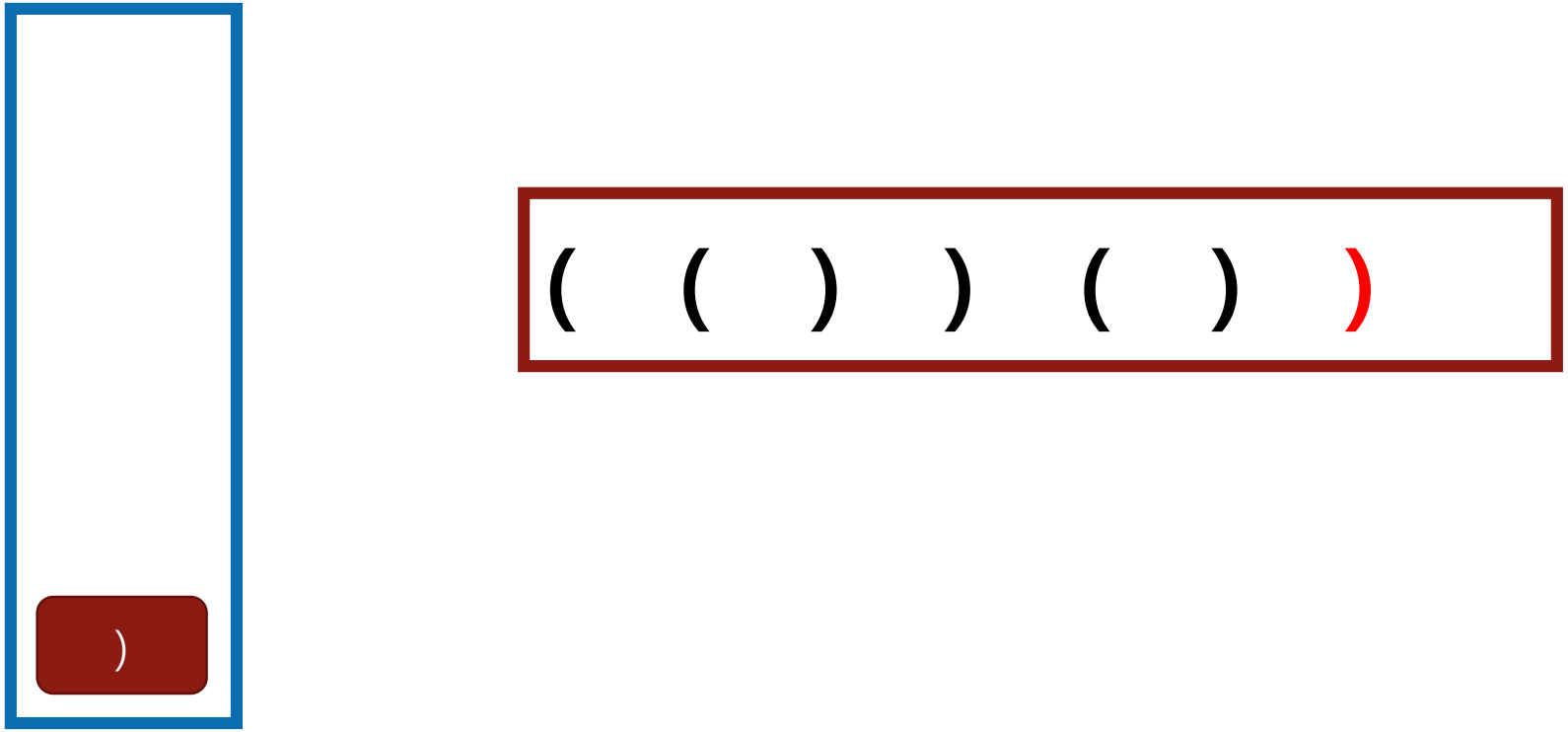
# #9012 괄호

ex 1. “( ( ) ) ( ) )”



# #9012 괄호

ex 1. “( ( ) ) ( ) )”



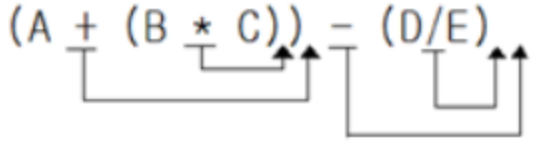


```cpp
#include <stack>
#include <string>
#include <iostream>
using namespace std;

int main(){
    int T;
    for(scanf("%d", &T); T--;){
        string str;
        stack<char> st;
        cin>>str;
        for(int i = 0; i<str.size(); i++){
            if(!st.empty() && st.top() == '(' && str[i] == ')')
                st.pop();
            else st.push(str[i]);
        }
        if(st.empty()) cout<<"YES\n";
        else cout<<"NO\n";
    }
    return 0;
}
```

# #1918 후위표기식(postfix notation)

- 수식에서 연산자(operator)가 피연사자(operand)의 앞에 위치하게끔 하는 표기법
- 왼쪽부터 차례로 읽어가며 계산할 수 없었던 중위 표기식(infix notation)과 달리, 이를 가능하게 해줌.



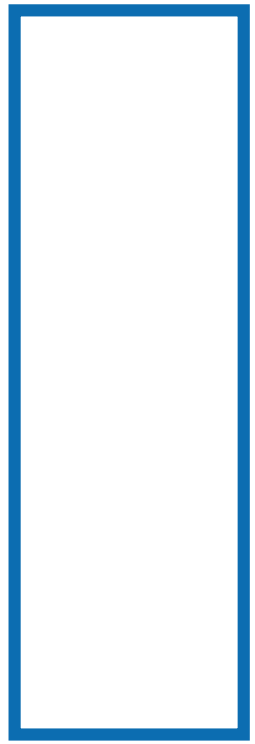

결과 : ABC*+DE/-



# Infix expression to postfix expression

A * ( B + C ) / D

stack



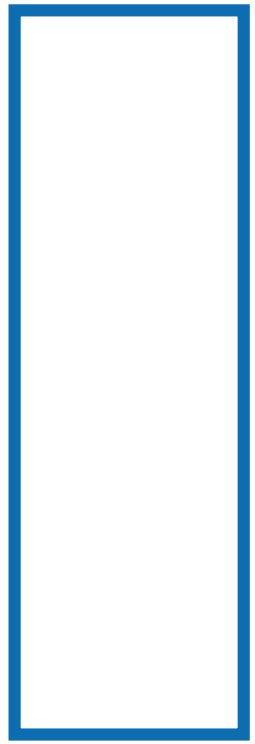
Result :

# Infix expression to postfix expression

A * ( B + C ) / D

1. operand가 나오면 그대로 출력

stack



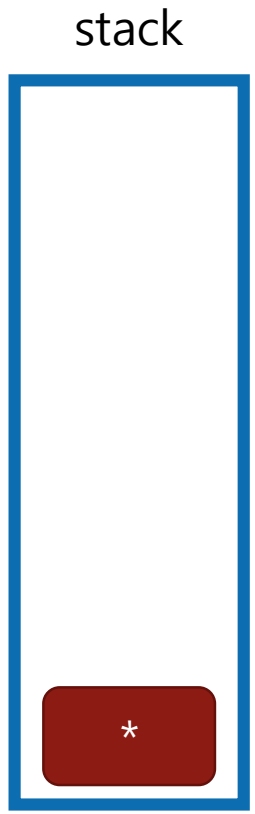
Result : A

# Infix expression to postfix expression

A * ( B + C ) / D

1. operand가 나오면 그대로 출력
2. stack에 operator이 없는 경우 그대로 insert
   아닌 경우 top과 비교

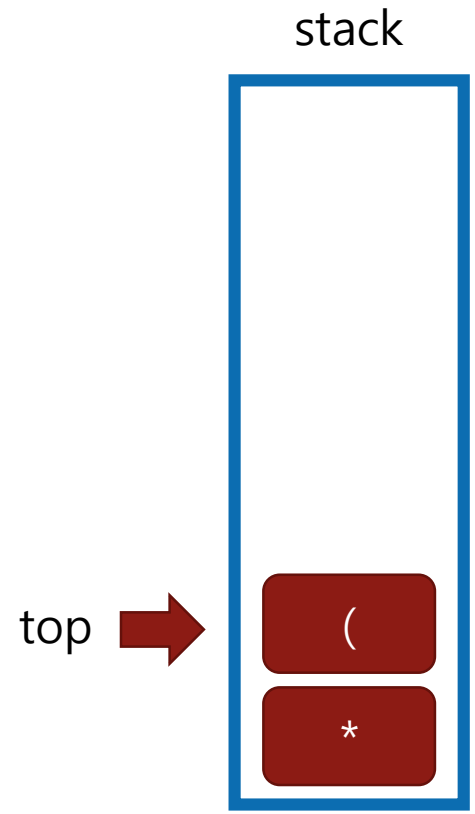
Result : A



# Infix expression to postfix expression

A * ( B + C ) / D

1. operand가 나오면 그대로 출력
2. stack에 operator이 없는 경우 그대로 insert
   아닌 경우 top과 비교
3. 왼쪽 괄호가 들어오면 그대로 insert

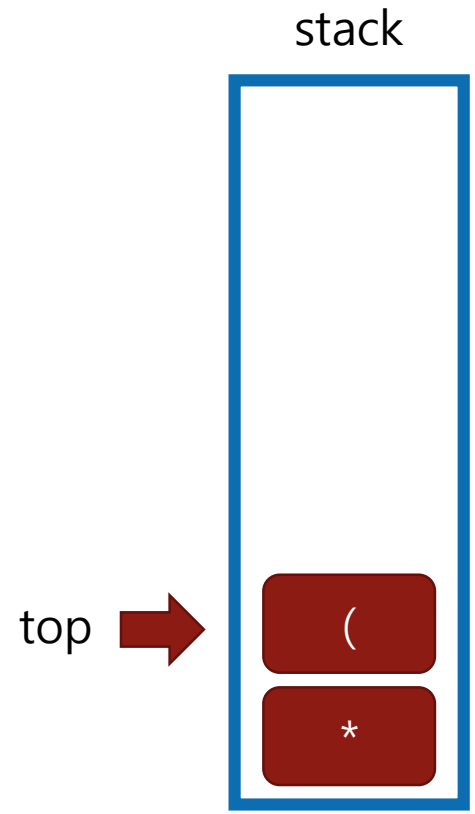



Result : **A**

# Infix expression to postfix expression

A * ( B + C ) / D

1. operand가 나오면 그대로 출력
2. stack에 operator이 없는 경우 그대로 insert
   아닌 경우 top과 비교
3. 왼쪽 괄호가 들어오면 그대로 insert

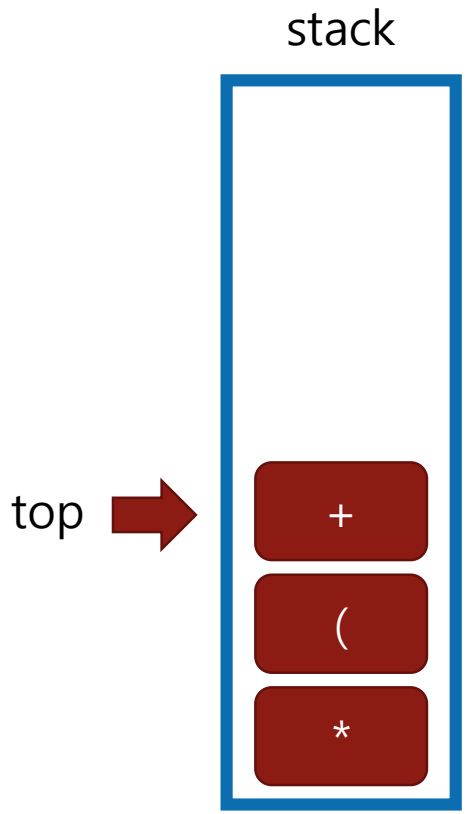



Result : **A B**

# Infix expression to postfix expression

A * ( B + C ) / D

1. operand가 나오면 그대로 출력
2. stack에 operator이 없는 경우 그대로 insert
   아닌 경우 top과 비교
3. 왼쪽 괄호가 들어오면 그대로 insert

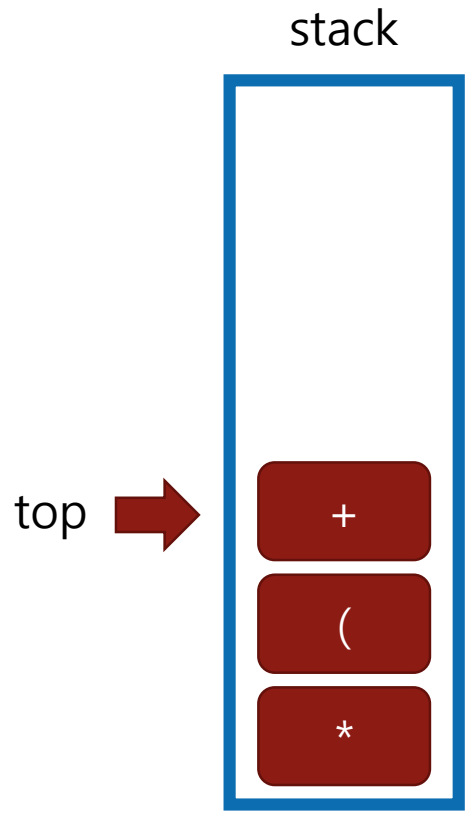



Result : **A B**

# Infix expression to postfix expression

A * ( B + C ) / D

1. operand가 나오면 그대로 출력
2. stack에 operator이 없는 경우 그대로 insert
   아닌 경우 top과 비교
3. 왼쪽 괄호가 들어오면 그대로 insert



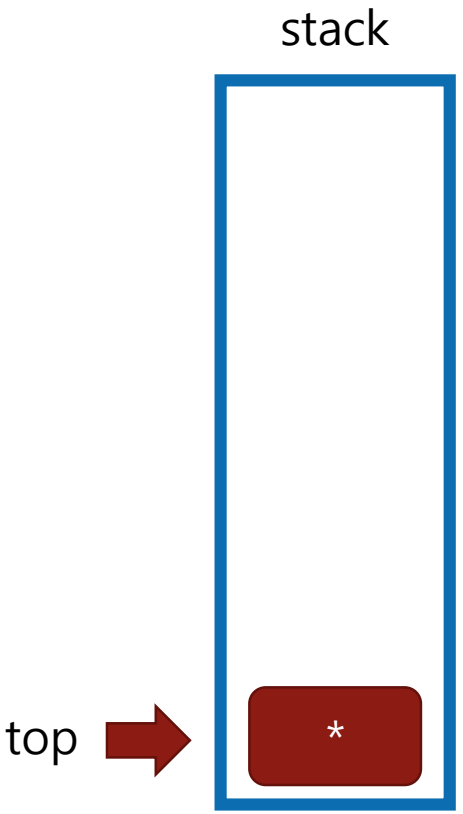

Result : **A B C**

# Infix expression to postfix expression

A * ( B + C ) / D

1. operand가 나오면 그대로 출력
2. stack에 operator이 없는 경우 그대로 insert
   아닌 경우 top과 비교
3. 왼쪽 괄호가 들어오면 그대로 insert
4. 오른쪽 괄호가 오면 stack에서 왼쪽괄호가 나올 때까지 pop

Result : **A B C +**

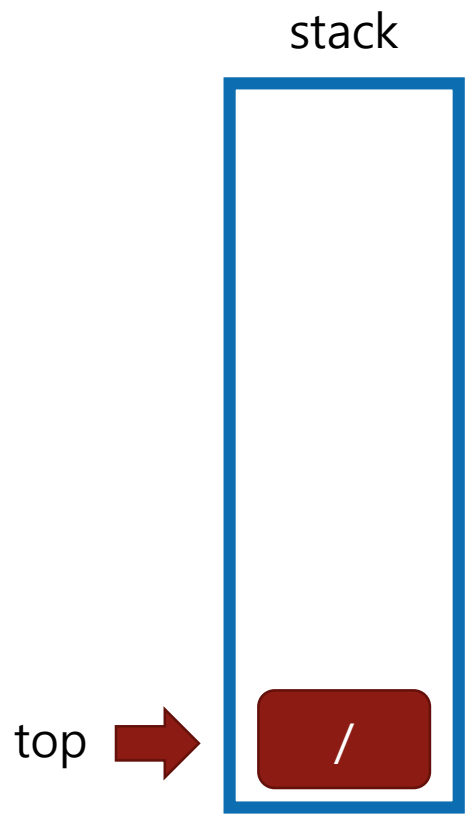


# Infix expression to postfix expression

A * ( B + C ) / D

1. operand가 나오면 그대로 출력
2. stack에 operator이 없는 경우 그대로 insert
   아닌 경우 top과 비교
3. 왼쪽 괄호가 들어오면 그대로 insert
4. 오른쪽 괄호가 오면 stack에서 왼쪽괄호가 나올 때까지 pop

Result : **A B C + ***

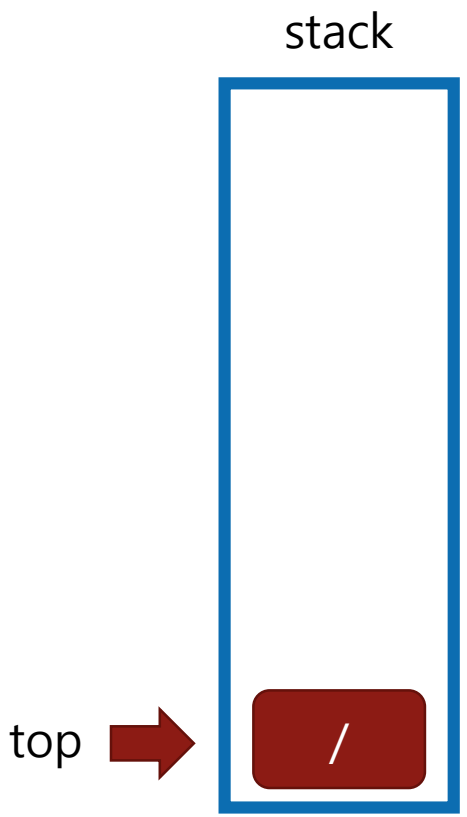


# Infix expression to postfix expression

A * ( B + C ) / D

1. operand가 나오면 그대로 출력
2. stack에 operator이 없는 경우 그대로 insert
   아닌 경우 top과 비교
3. 왼쪽 괄호가 들어오면 그대로 insert
4. 오른쪽 괄호가 오면 stack에서 왼쪽괄호가 나올 때까지 pop

Result : **A B C + * D**



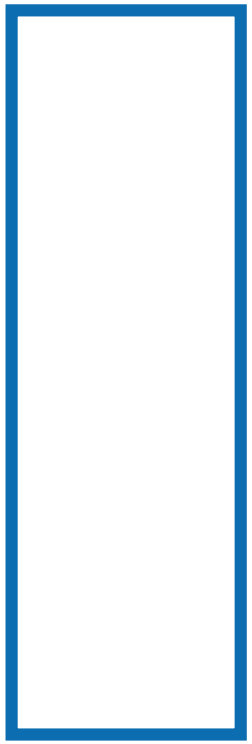
# Infix expression to postfix expression

A * ( B + C ) / D

1. operand가 나오면 그대로 출력
2. stack에 operator이 없는 경우 그대로 insert
   아닌 경우 top과 비교
3. 왼쪽 괄호가 들어오면 그대로 insert
4. 오른쪽 괄호가 오면 stack에서 왼쪽괄호가 나올 때까지 pop

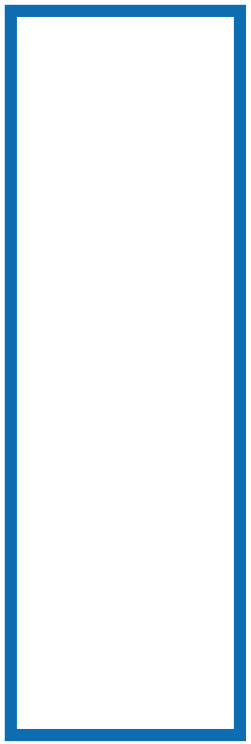
Result : **A B C + * D /**

stack



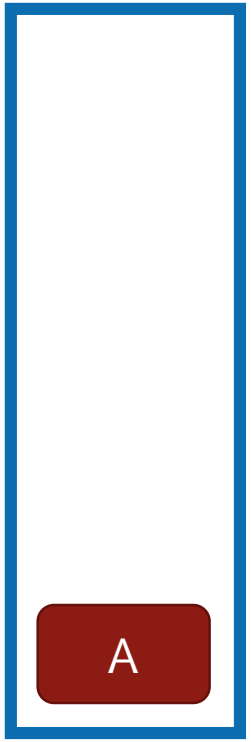
# 후위표기식을 통한 연산 과정

ex)

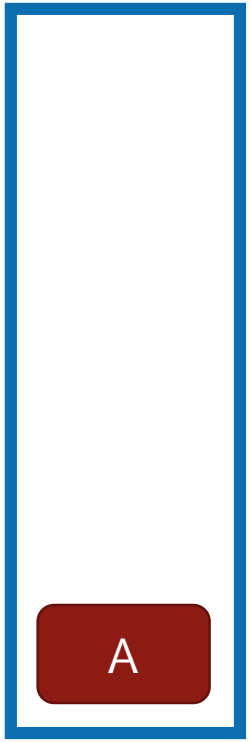
# 후위표기식을 통한 연산 과정

ex)

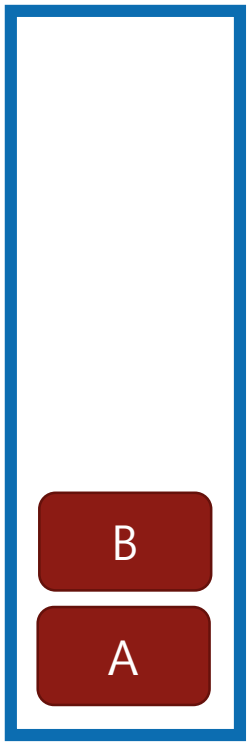
# 후위표기식을 통한 연산 과정

ex)

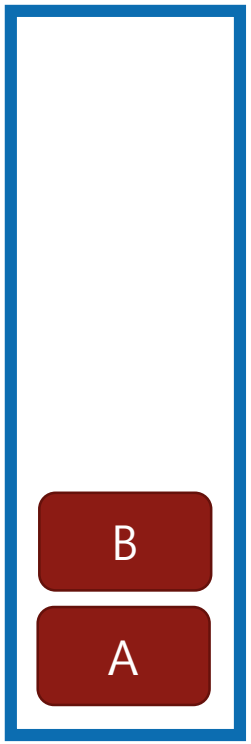
# 후위표기식을 통한 연산 과정

ex)

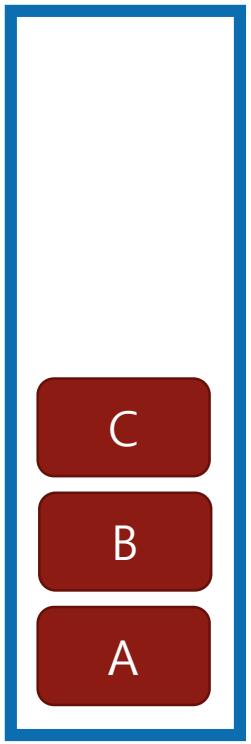
# 후위표기식을 통한 연산 과정

ex)

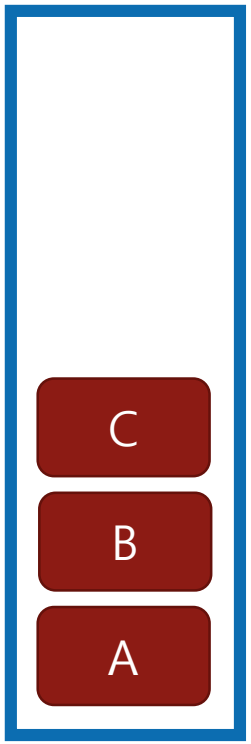
# 후위표기식을 통한 연산 과정

ex)

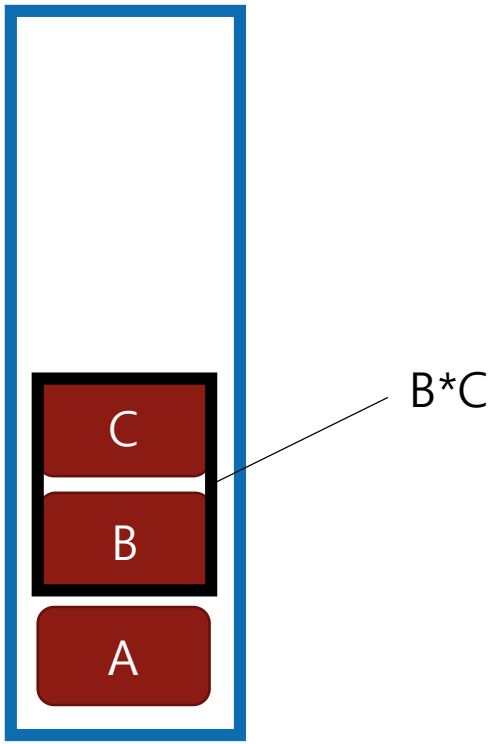
# 후위표기식을 통한 연산 과정

ex)

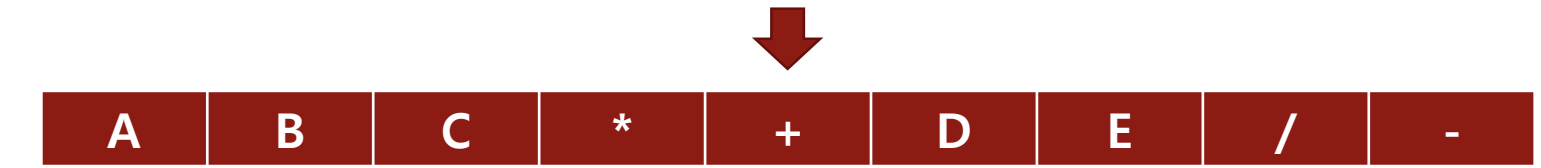
# 후위표기식을 통한 연산 과정

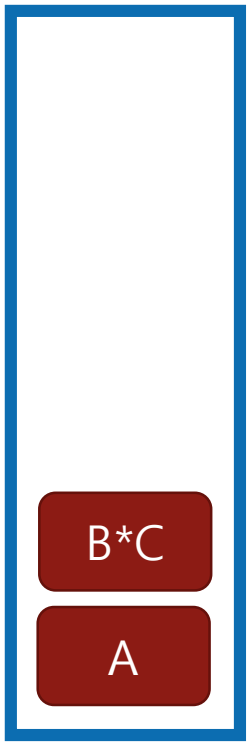
ex)

# 후위표기식을 통한 연산 과정

ex)

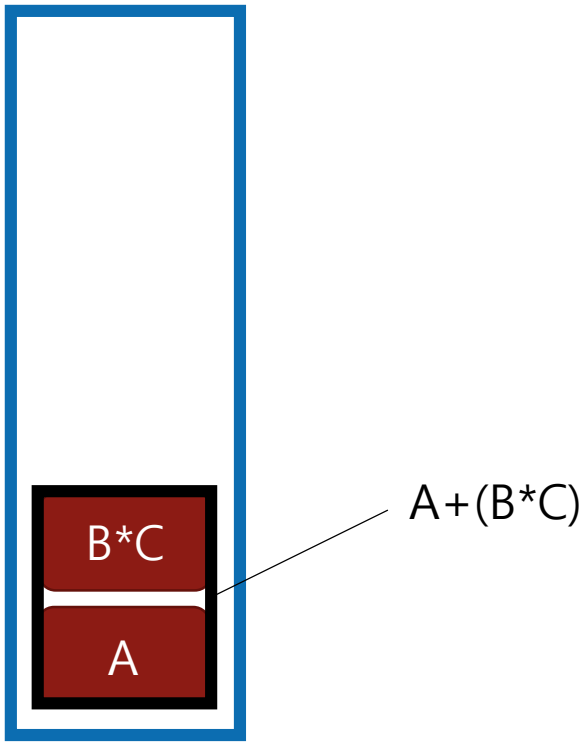


# 후위표기식을 통한 연산 과정

ex)

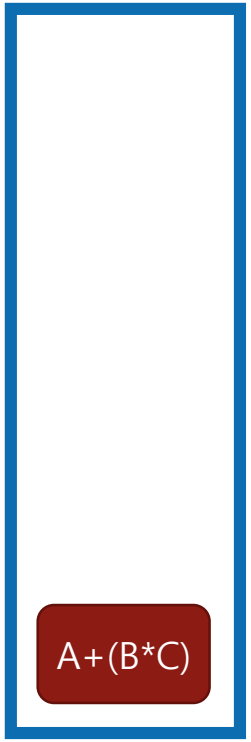
| A | B | C | * | + | D | E | / | - |
|---|---|---|---|---|---|---|---|---|



# 후위표기식을 통한 연산 과정

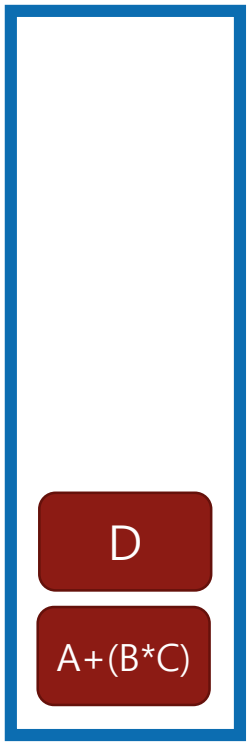
ex)

| A | B | C | * | + | D | E | / | - |
|---|---|---|---|---|---|---|---|---|



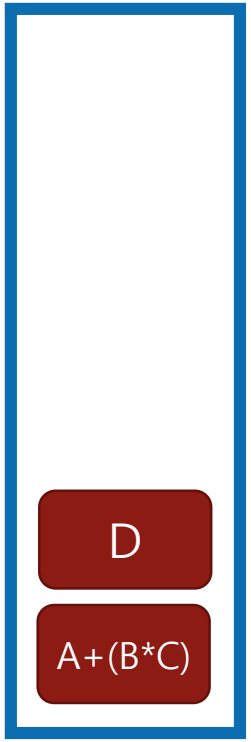
# 후위표기식을 통한 연산 과정

ex)

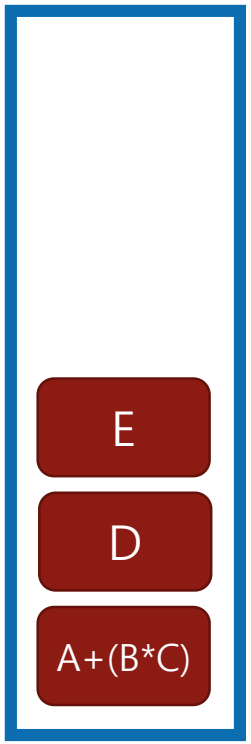
# 후위표기식을 통한 연산 과정

ex)

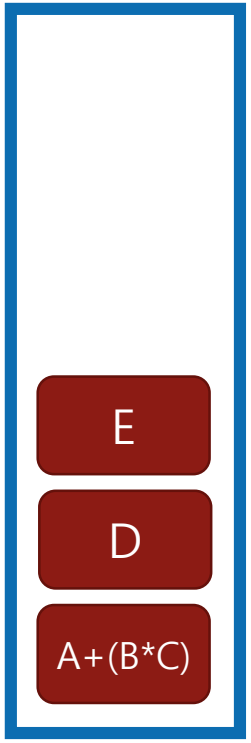
# 후위표기식을 통한 연산 과정

ex)

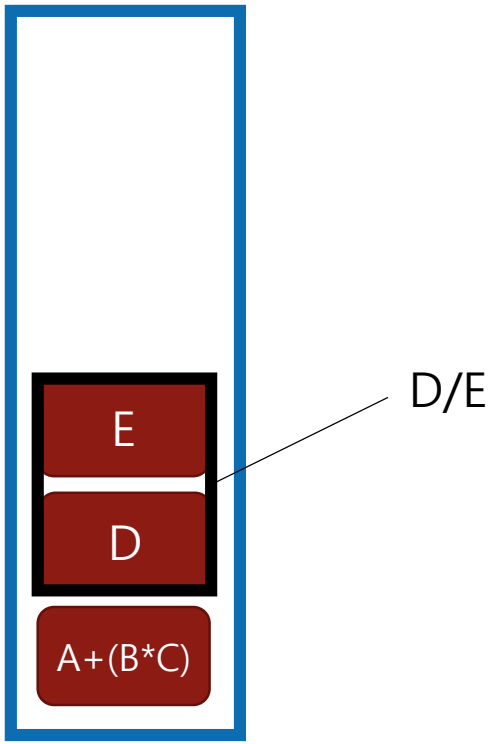
# 후위표기식을 통한 연산 과정

ex)

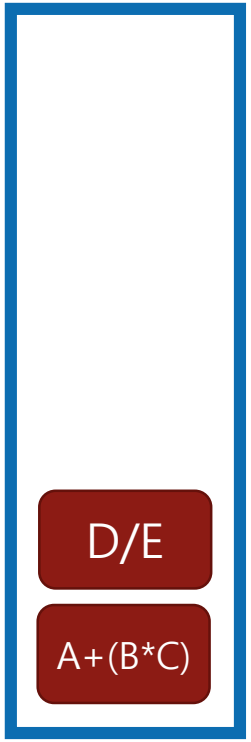
# 후위표기식을 통한 연산 과정

ex)

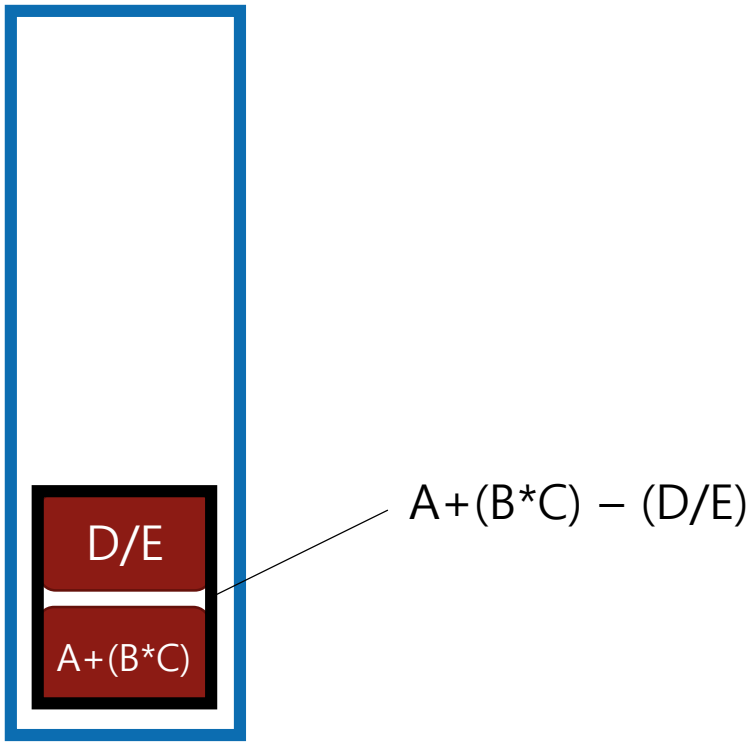
# 후위표기식을 통한 연산 과정

ex)

# 후위표기식을 통한 연산 과정

ex)

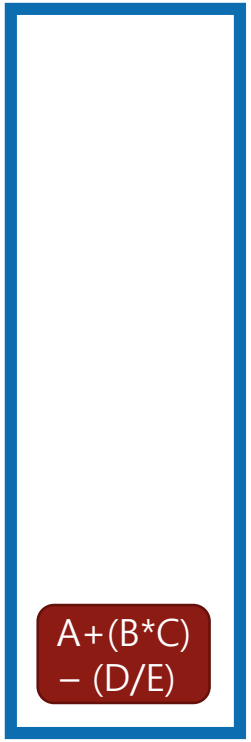




A+(B*C) – (D/E)

# 후위표기식을 통한 연산 과정

ex)

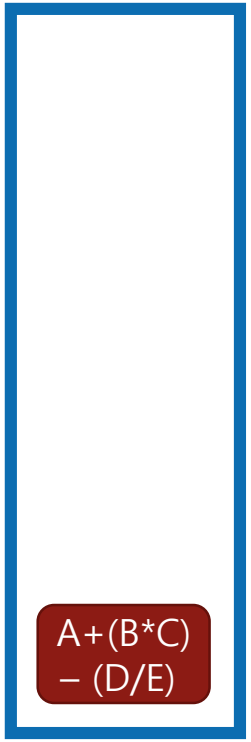
| A | B | C | * | + | D | E | / | - |
|---|---|---|---|---|---|---|---|---|

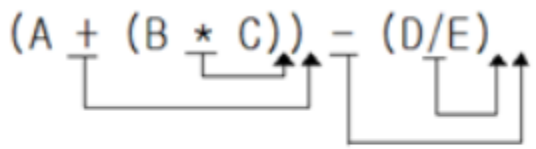


# 후위표기식을 통한 연산 과정

ex)

| A | B | C | * | + | D | E | / | - |
|---|---|---|---|---|---|---|---|---|



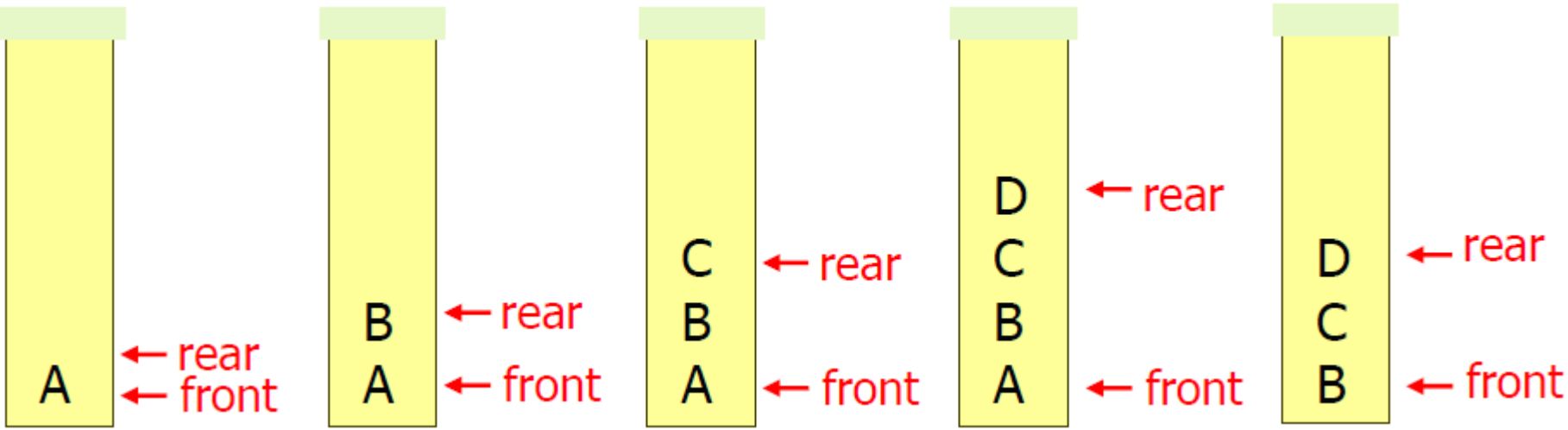




결과 : ABC*+DE/-

# Queue

- 먼저 들어간 원소가 가장 먼저 나오는 자료구조
- FIFO list = queue

  First-In First-Out





# 큐 연산

- front( ) : 맨 앞의 값에 접근
- back( ) : 맨 뒤의 값에 접근
- push( ) : 맨 뒤에 새로운 값 추가
- pop( ) : 맨 앞의 값 제거



# STL을 통한 큐 사용법

```
queue <int> q;          // 정수형 큐 자료구조 선언
int n = q.front();      // front에 위치한 값 return
q.push(1);              ┐
q.push(n);              ┘ push 연산
q.pop();                // back에서 pop 연산
q.empty();              // 비었을 경우 true, 아닐 경우 false
```



# Deque

- Double-Ended queue
- 양방향 삽입 및 삭제가 가능!



# STL을 통한 덱 사용법

```
deque <int> dq;

int n = dq.front();

dq.push_back(1);

dq.push_front(n);

dq.pop_back(); dq.pop_front();

dq.empty();
```



# #2164 카드2

## 문제

N장의 카드가 있다. 각각의 카드는 차례로 1부터 N까지의 번호가 붙어 있으며, 1번 카드가 제일 위에, N번 카드가 제일 아래인 상태로 순서대로 카드가 놓여 있다.

이제 다음과 같은 동작을 카드가 한 장 남을 때까지 반복하게 된다. 우선, 제일 위에 있는 카드를 바닥에 버린다. 그 다음, 제일 위에 있는 카드를 제일 아래에 있는 카드 밑으로 옮긴다.

예를 들어 N=4인 경우를 생각해 보자. 카드는 제일 위에서부터 1234 의 순서로 놓여있다. 1을 버리면 234가 남는다. 여기서 2를 제일 아래로 옮기면 342가 된다. 3을 버리면 42가 되고, 4를 밑으로 옮기면 24가 된다. 마지막으로 2를 버리고 나면, 남는 카드는 4가 된다.

N이 주어졌을 때, 제일 마지막에 남게 되는 카드를 구하는 프로그램을 작성하시오.

## 입력

첫째 줄에 정수 N(1≤N≤500,000)이 주어진다.

## 출력

첫째 줄에 남게 되는 카드의 번호를 출력한다.



# #2164 카드2

## 문제

N장의 카드가 있다. 각각의 카드는 차례로 1부터 N까지의 번호가 붙어 있으며, 1번 카드가 제일 위에, N번 카드가 제일 아래인 상태로 순서대로 카드가 놓여 있다.

이제 다음과 같은 동작을 카드가 한 장 남을 때까지 반복하게 된다. 우선, 제일 위에 있는 카드를 바닥에 버린다. 그 다음, 제일 위에 있는 카드를 제일 아래에 있는 카드 밑으로 옮긴다.

예를 들어 N=4인 경우를 생각해 보자. 카드는 제일 위에서부터 1234 의 순서로 놓여있다. 1을 버리면 234가 남는다. 여기서 2를 제일 아래로 옮기면 342가 된다. 3을 버리면 42가 되고, 4를 밑으로 옮기면 24가 된다. 마지막으로 2를 버리고 나면, 남는 카드는 4가 된다.

N이 주어졌을 때, 제일 마지막에 남게 되는 카드를 구하는 프로그램을 작성하시오.

## 입력

첫째 줄에 정수 N(1≤N≤500,000)이 주어진다.

## 출력

첫째 줄에 남게 되는 카드의 번호를 출력한다.

# #2164 카드2

## 문제

N장의 카드가 있다. 각각의 카드는 차례로 1부터 N까지의 번호가 붙어 있으며, 1번 카드가 제일 위에, N번 카드가 제일 아래인 상태로 순서대로 카드가 놓여 있다.

이제 다음과 같은 동작을 카드가 한 장 남을 때까지 반복하게 된다. 우선, 제일 위에 있는 카드를 바닥에 버린다. 그 다음, 제일 위에 있는 카드를 제일 아래에 있는 카드 밑으로 옮긴다.

예를 들어 N=4인 경우를 생각해 보자. 카드는 제일 위에서부터 1234 의 순서로 놓여있다. 1을 버리면 234가 남는다. 여기서 2를 제일 아래로 옮기면 342가 된다. 3을 버리면 42가 되고, 4를 밑으로 옮기면 24가 된다. 마지막으로 2를 버리고 나면, 남는 카드는 4가 된다.

N이 주어졌을 때, 제일 마지막에 남게 되는 카드를 구하는 프로그램을 작성하시오.

## 입력

첫째 줄에 정수 N(1≤N≤500,000)이 주어진다.

## 출력

첫째 줄에 남게 되는 카드의 번호를 출력한다.

```cpp
#include <stdio.h>
#include <queue>
using namespace std;

int main() {
    int n;
    scanf("%d", &n);
    queue <int> q;
    for (int i = 1; i <= n; i++)
        q.push(i);
    while(q.size() != 1) {
        q.pop();
        int top = q.front();
        q.pop();
        q.push(top);
    }
    printf("%d\n", q.front());
    return 0;
}
```



# #2164 카드2

## 문제

N장의 카드가 있다. 각각의 카드는 차례로 1부터 N까지의 번호가 붙어 있으며, 1번 카드가 제일 위에, N번 카드가 제일 아래인 상태로 순서대로 카드가 놓여 있다.

이제 다음과 같은 동작을 카드가 한 장 남을 때까지 반복하게 된다. 우선, 제일 위에 있는 카드를 바닥에 버린다. 그 다음, 제일 위에 있는 카드를 제일 아래에 있는 카드 밑으로 옮긴다.

예를 들어 N=4인 경우를 생각해 보자. 카드는 제일 위에서부터 1234 의 순서로 놓여있다. 1을 버리면 234가 남는다. 여기서 2를 제일 아래로 옮기면 342가 된다. 3을 버리면 42가 되고, 4를 밑으로 옮기면 24가 된다. 마지막으로 2를 버리고 나면, 남는 카드는 4가 된다.

N이 주어졌을 때, 제일 마지막에 남게 되는 카드를 구하는 프로그램을 작성하시오.

## 입력

첫째 줄에 정수 N(1≤N≤500,000)이 주어진다.

## 출력

첫째 줄에 남게 되는 카드의 번호를 출력한다.

```cpp
#include <stdio.h>
#include <deque>
using namespace std;

int main() {
    int n;
    scanf("%d", &n);
    deque <int> q;
    for (int i = 1; i <= n; i++)
        q.push_back(i);
    while(q.size() != 1) {
        q.pop_front();
        int top = q.front();
        q.pop_front();
        q.push_back(top);
    }
    printf("%d\n", q.front());
    return 0;
}
```



# Problem set

- 10828 스택
- 9012 괄호
- 1918 후위표기식
- 1935 후위표기식2
- 2841 외계인의 기타 연주
- 1725 히스토그램
- 10845 큐
- 2164 카드2
- 1966 프린터 큐
- 10866 덱
- 5430 AC