COSE213: Data Structure

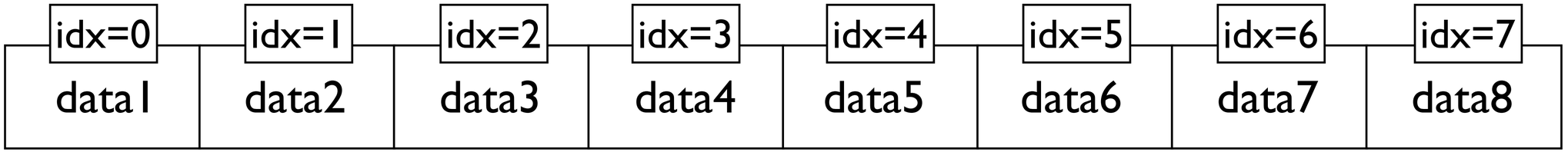
Lecture 3 - 스택(Stacks)

Minseok Jeon

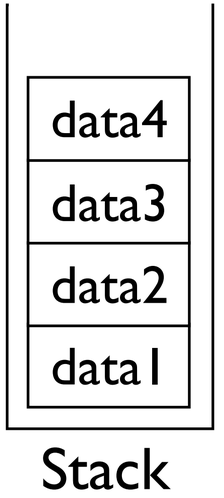
2024 Fall

리뷰: 배열 (Array)

- 배열(Array) 자료구조: 동일한 데이터 타입을 가진 값들을 연속된 공간에 저장하는 자료구조.

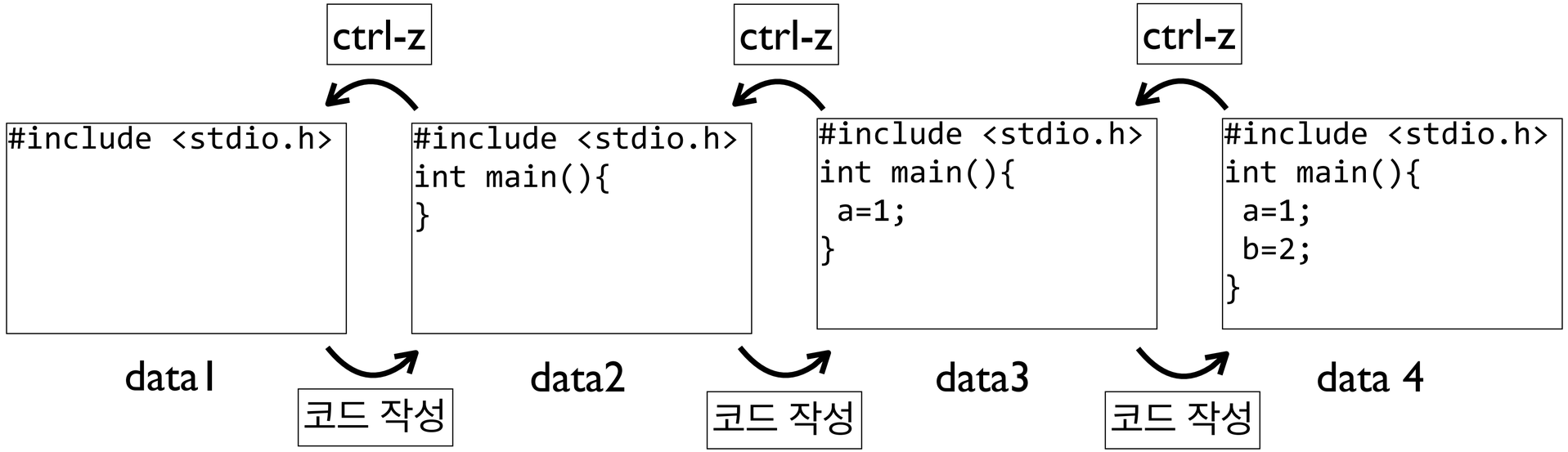


- 배열의 추상 자료형 (Abstract Data Type):
  - `create(type,size)` : 주어진 타입(`type`)과 길이(`size`)를 가지는 배열을 생성
  - `read(arr,index)` : 배열(arr)에서 주어진 인덱스(index)에 해당하는 자료를 반환
  - `update(arr,index,value)` : 배열(arr)에서 주어진 인덱스(index) 위치에 새로운 데이터(value)를 저장
- 배열은 다른 자료구조(e.g., 스택)들을 구현하는데 사용될 수 있음



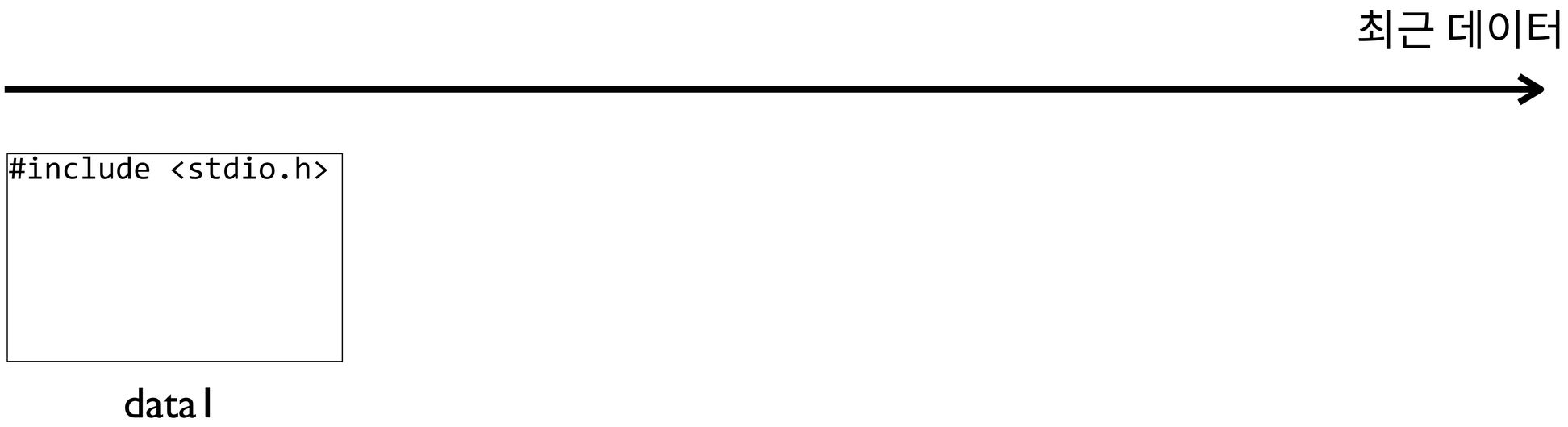
문제: 되돌리기 (Ctrl-z)

- 문제: 되돌리기 (Ctrl-z) 구현을 위한 코드 작성 데이터 저장



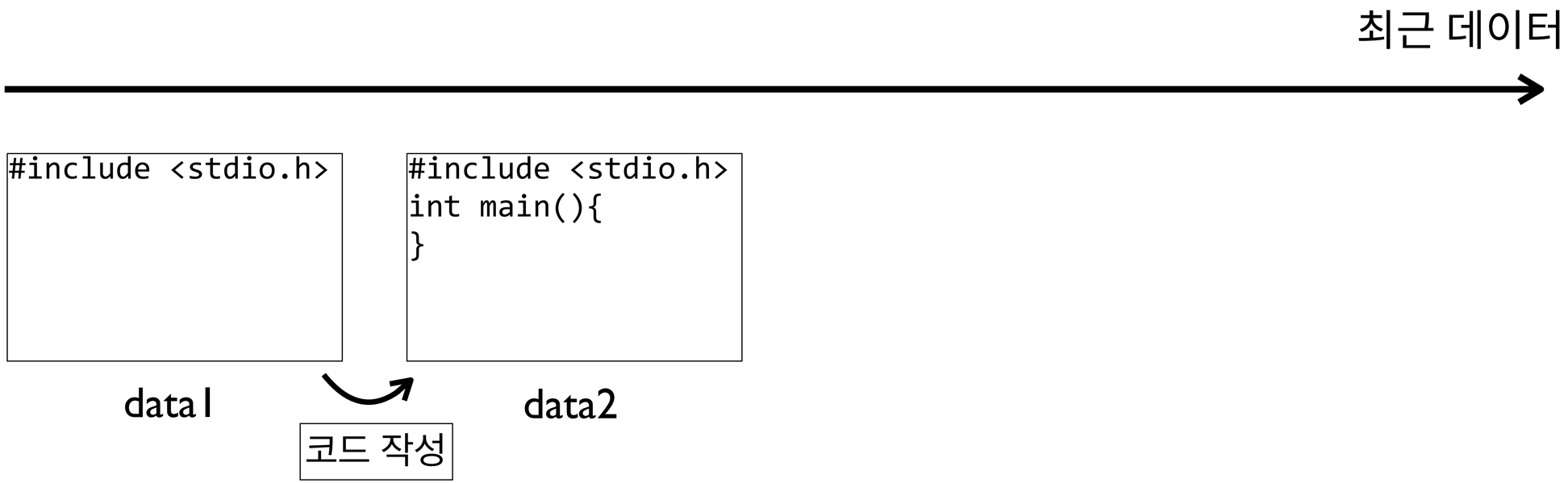
문제: 되돌리기 (Ctrl-z)

- 문제: 되돌리기 (Ctrl-z) 구현을 위한 코드 작성 데이터 저장
  - 특징 1: 데이터간 (시간) 순서가 있음



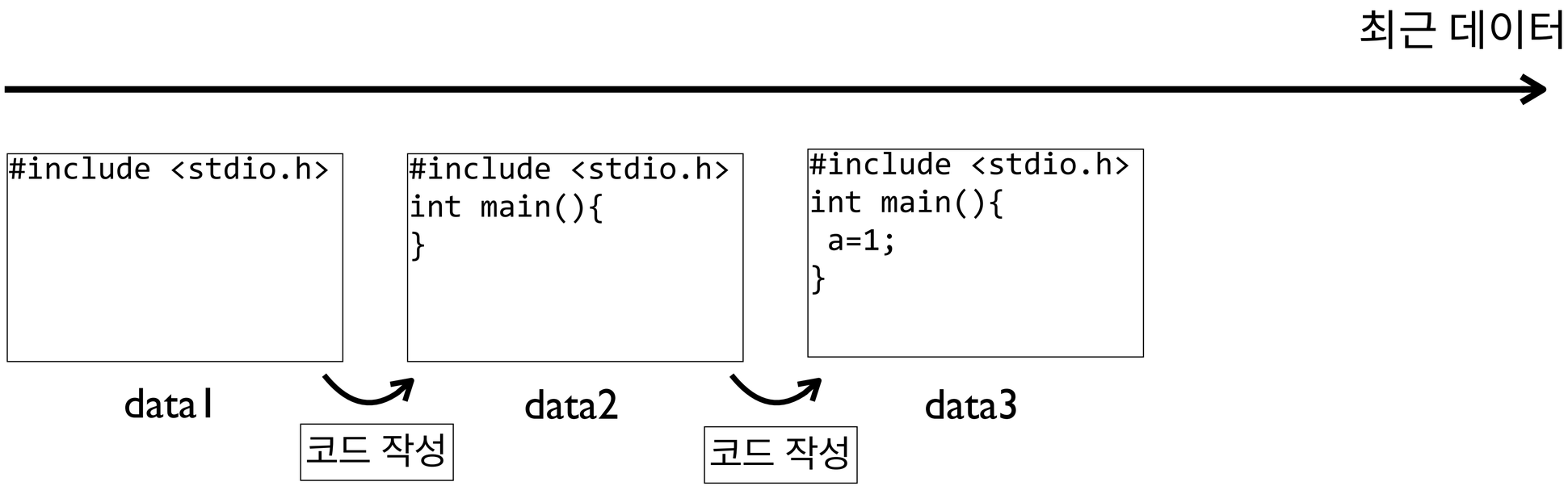
문제: 되돌리기 (Ctrl-z)

- 문제: 되돌리기 (Ctrl-z) 구현을 위한 코드 작성 데이터 저장
  - 특징 1: 데이터간 (시간) 순서가 있음



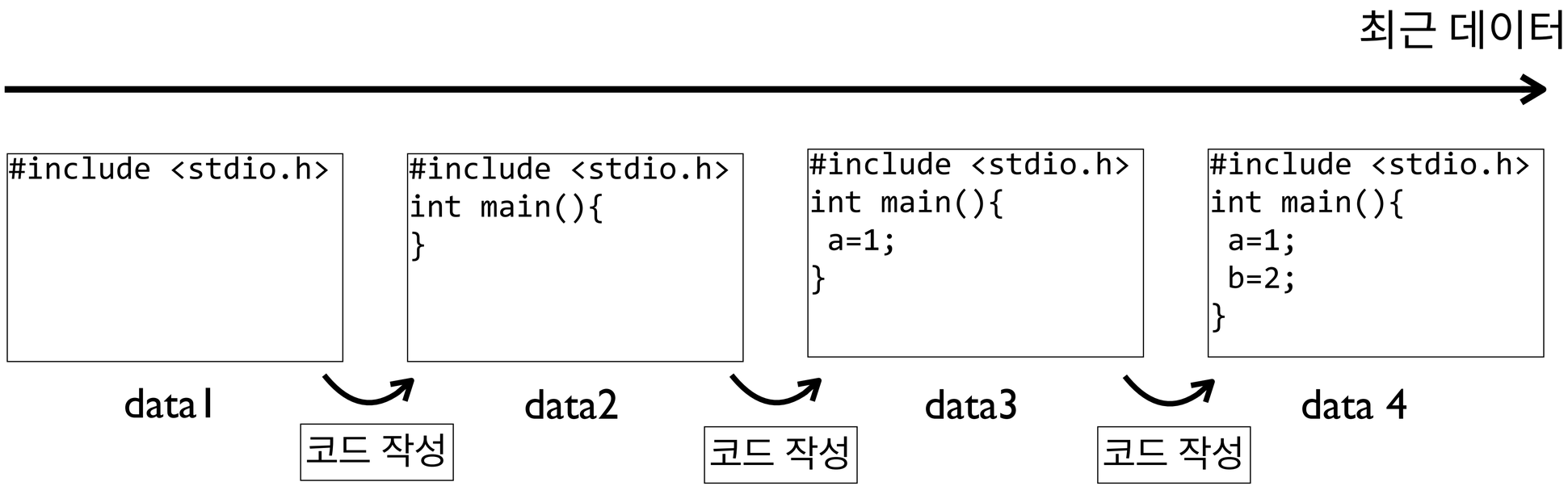
문제: 되돌리기 (Ctrl-z)

- 문제: 되돌리기 (Ctrl-z) 구현을 위한 코드 작성 데이터 저장
  - 특징 1: 데이터간 (시간) 순서가 있음



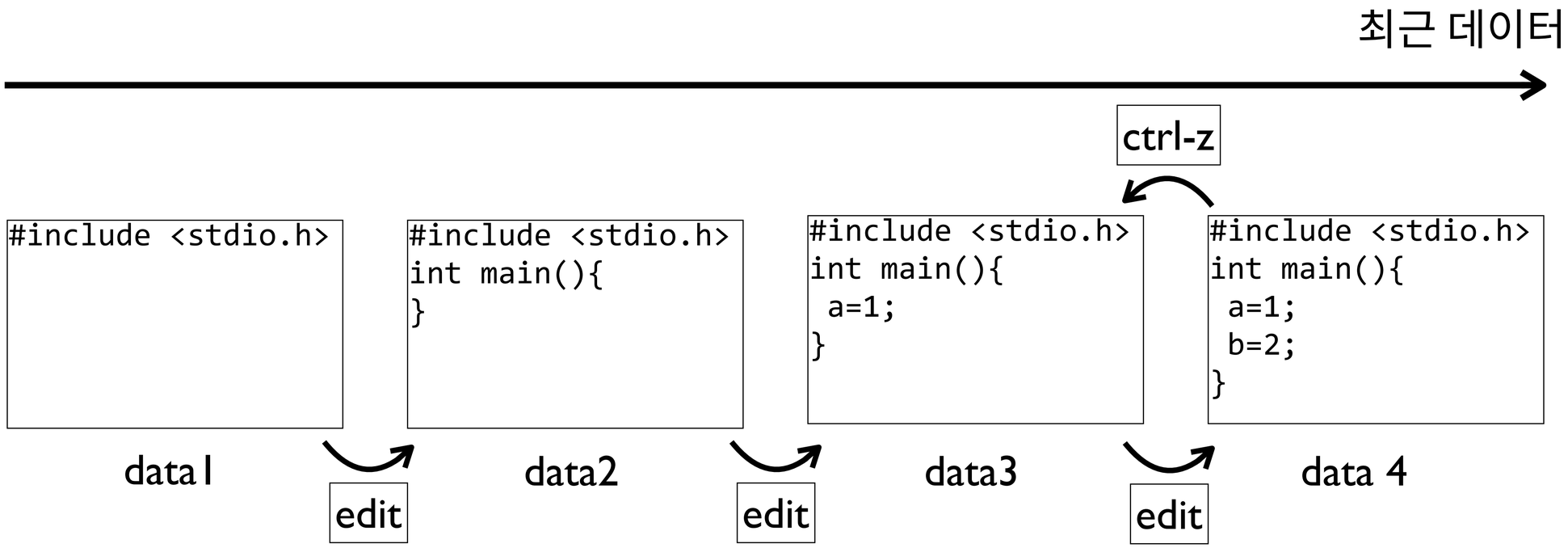
문제: 되돌리기 (Ctrl-z)

- 문제: 되돌리기 (Ctrl-z) 구현을 위한 코드 작성 데이터 저장
  - 특징 1: 데이터간 (시간) 순서가 있음



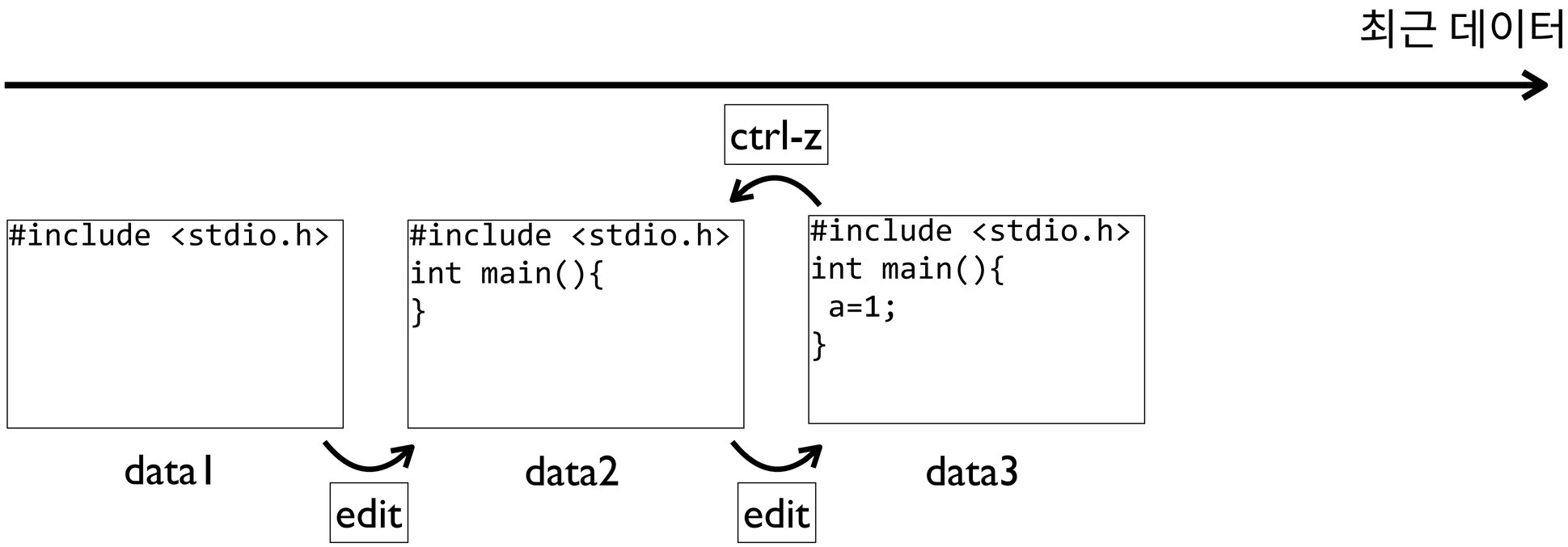
문제: 되돌리기 (Ctrl-z)

- 문제: 되돌리기 (Ctrl-z) 구현을 위한 코드 작성 데이터 저장
  - 특징 1: 데이터간 (시간) 순서가 있음
  - 특징 2: 가장 최근(Last) 데이터에 접근해야함



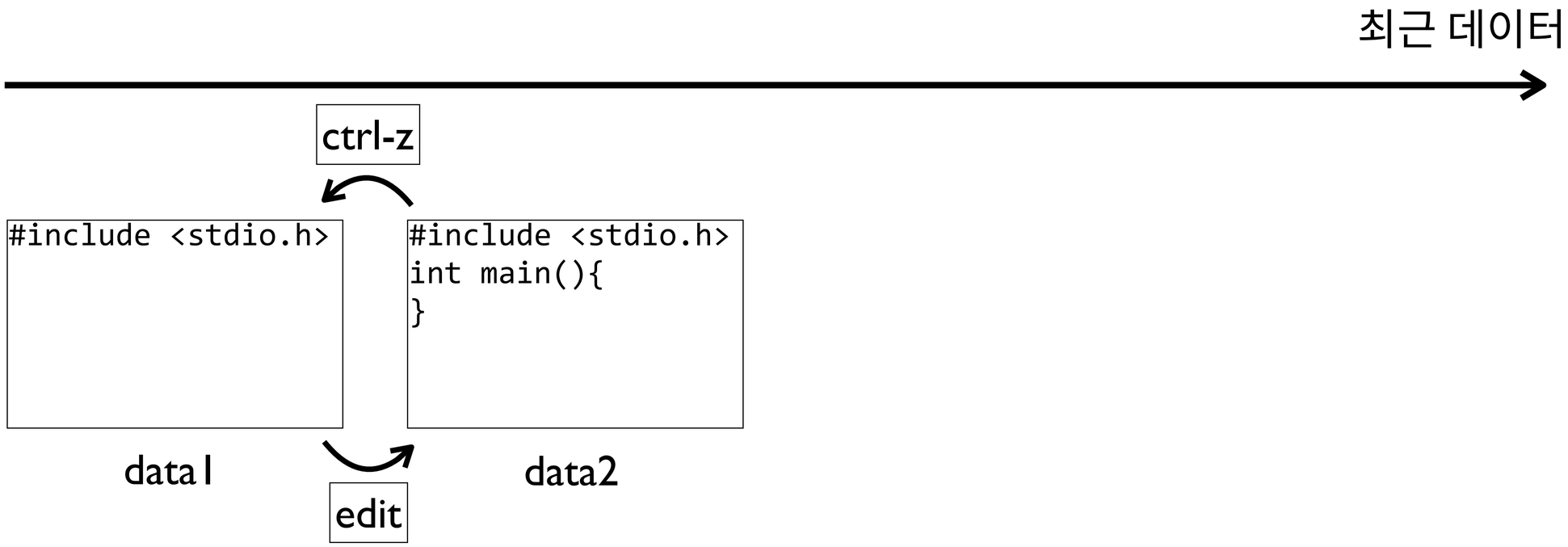
문제: 되돌리기 (Ctrl-z)

- 문제: 되돌리기 (Ctrl-z) 구현을 위한 코드 작성 데이터 저장
  - 특징 1: 데이터간 (시간) 순서가 있음
  - 특징 2: 가장 최근(Last) 데이터에 접근해야함



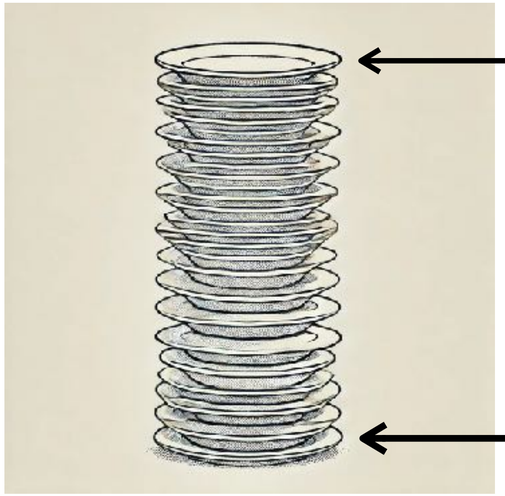
문제: 되돌리기 (Ctrl-z)

- 문제: 되돌리기 (Ctrl-z) 구현을 위한 코드 작성 데이터 저장
  - 특징 1: 데이터간 (시간) 순서가 있음
  - 특징 2: 가장 최근(Last) 데이터에 접근해야함



문제: 되돌리기 (Ctrl-z)

- 문제: 되돌리기 (Ctrl-z) 구현을 위한 코드 작성 데이터 저장
  - 특징 1: 데이터간 (시간) 순서가 있음
  - 특징 2: 가장 최근(Last) 데이터에 접근해야함
- 필요한 자료구조의 형태:



가장 최근에 저장된 데이터

- 새로운 (최신) 자료 저장 시 가장 위에 자료를 올림
- 가장 최신 자료만을 조회 및 삭제할 수 있음

가장 오래전에 저장된 데이터

해결책: 스택 (Stack)

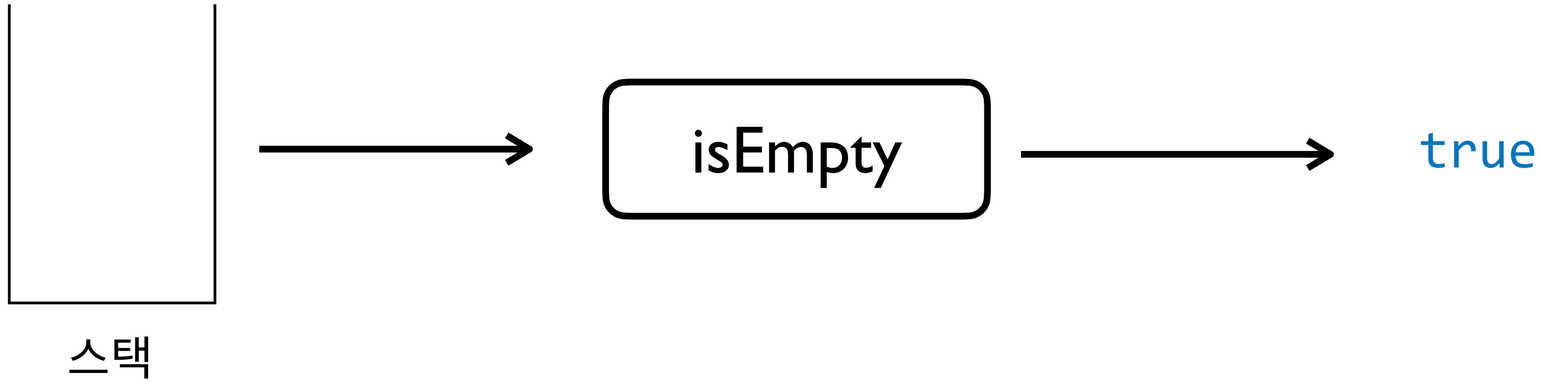
- 스택(Stack): 후입선출(LIFO: Last In, First Out) 원칙을 따르는 자료구조
- 스택 자료구조는 다음의 기능들을 제공함 (스택의 추상 자료형):
  - create() : 비어있는 스택을 생성 후 반환
  - `isEmpty(s)` : 스택 s가 비어있는지 확인함
  - `isFull(s)` : 스택 s가 꽉 차있는지 확인함
  - `push(s,x)` : 스택 s의 가장 위에 주어진 새로운 데이터 x를 추가
  - `pop(s)` : 스택 s의 가장 위에 있는 데이터를 삭제하고 반환
  - `peek(s)` : 스택 s의 가장 위 데이터를 제거하지 않고 반환

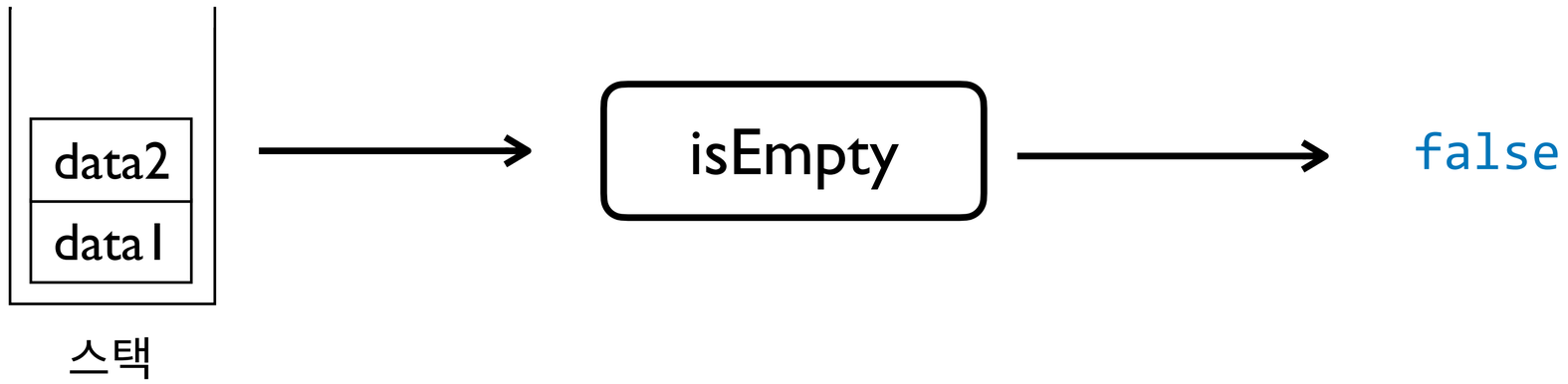
ADT of Stack in Ocaml: https://ocaml.org/manual/5.2/api/Stack.html

ADT of Stack in Java: https://docs.oracle.com/javase/8/docs/api/java/util/Stack.html

isEmpty

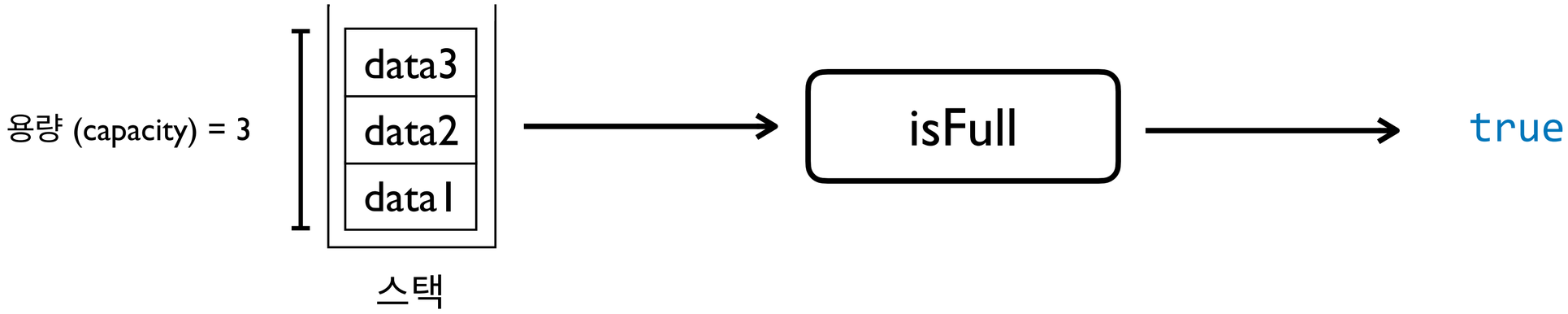
- `isEmpty` : 스택이 비어있으면 `true`를 아니면 `false`를 반환

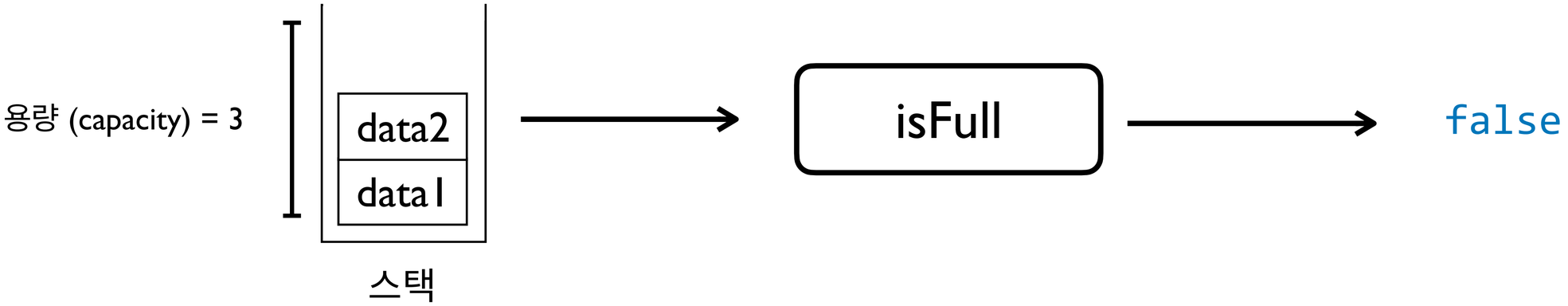




isFull

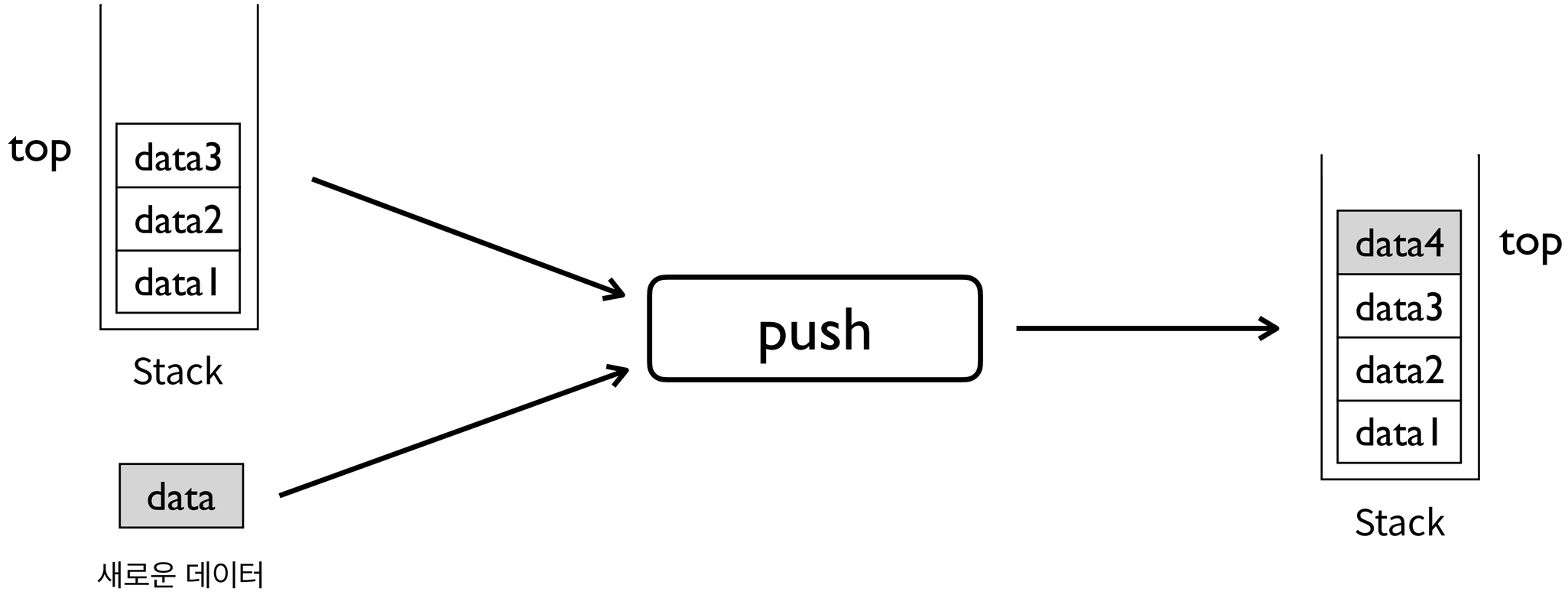
- `isFull` : 스택이 가득 차 있으면 `true`를 아니면 `false`를 반환





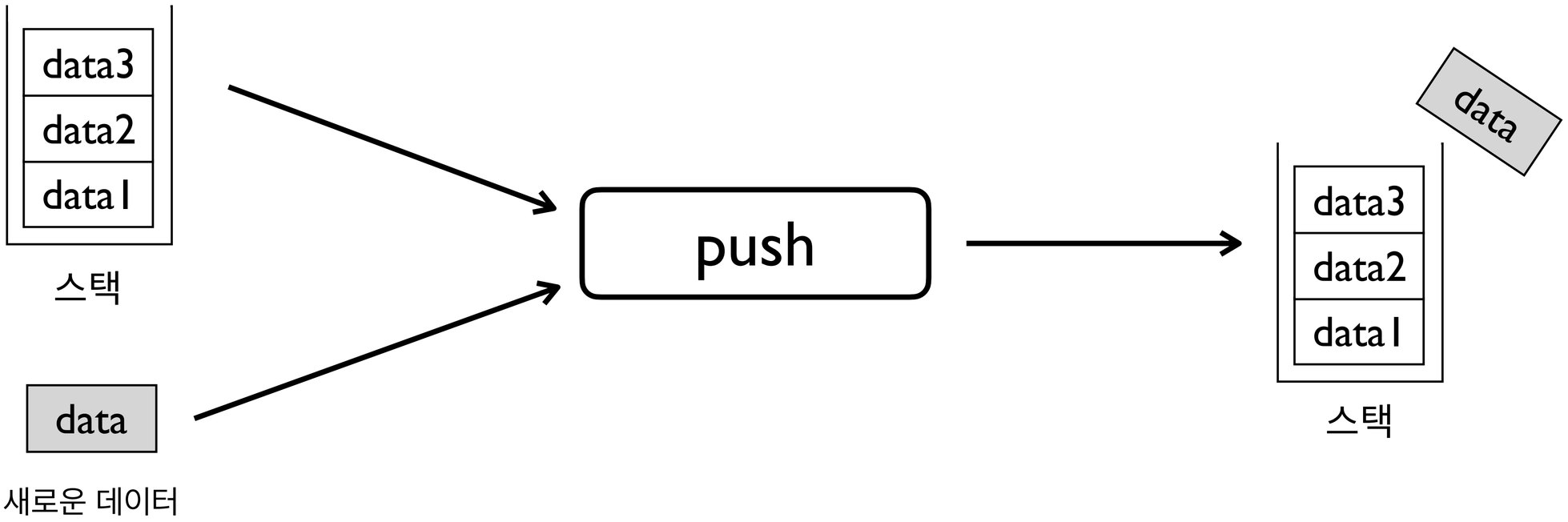
push

- push : 스택(stack)의 가장 위(top)에 주어진 새로운 데이터를 추가
  - 추가된 데이터가 스택의 가장 위에 위치하게 됨



push

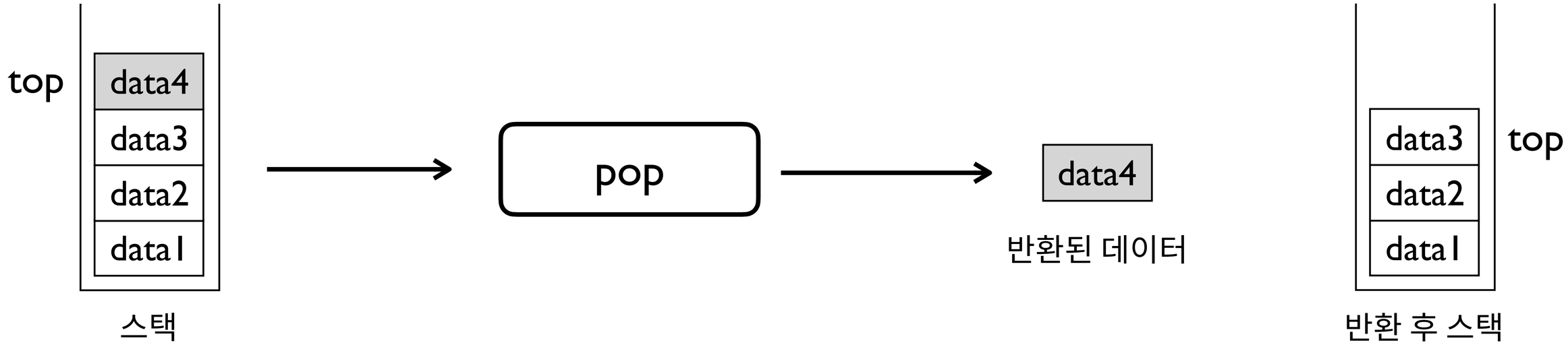
- push : 스택(stack)의 가장 위(top)에 주어진 새로운 데이터를 추가
  - 추가된 데이터가 스택의 가장 위에 위치하게 됨
  - 추가할 공간이 없을 때 데이터를 추가할 경우 `overflow` 발생



pop

- pop : 스택(Stack) 의 가장 위(top)에 있는 요소를 삭제하고 반환

  - pop이 실행되기 전 위에서 두번째 데이터가 pop이 실행된 후 가장 위(top)에 위치하게 됨

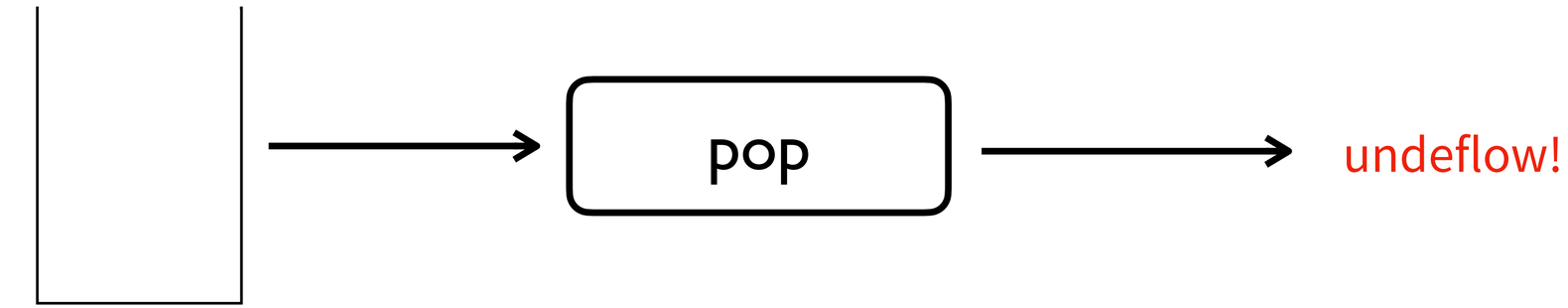


pop

- pop : 스택(Stack) 의 가장 위(top)에 있는 요소를 삭제하고 반환

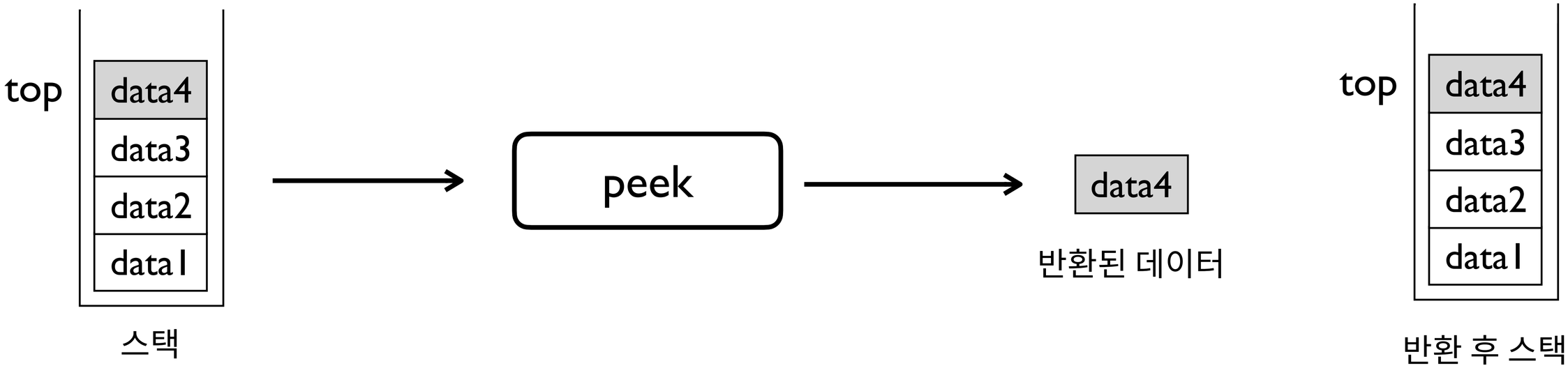
  - pop이 실행되기 전 위에서 두번째 데이터가 pop이 실행된 후 가장 위(top)에 위치하게 됨

  - 비어있는 스택에서 pop을 실행 할 경우 underflow 발생

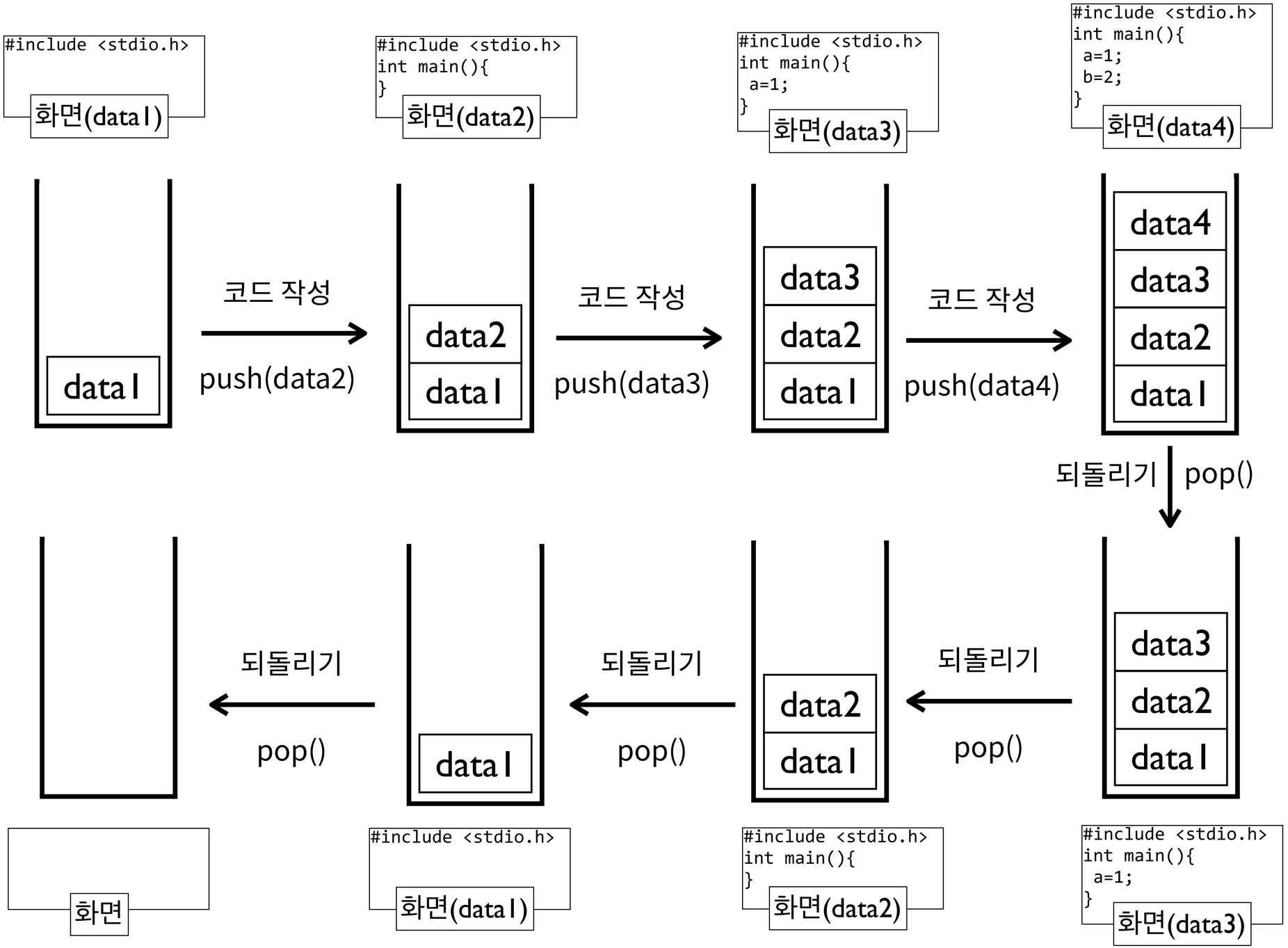


peek

- peek : 스택의 맨 위 항목을 제거하지 않고 반환
  - Peek 실행 전후로 스택의 상태는 변하지 않음



- peek대신 top으로 표기하기도 함:
  ADT of Stack in Ocaml: https://ocaml.org/manual/5.2/api/Stack.html


#include <stdio.h>
화면(data1)
#include <stdio.h>
int main(){
}
화면(data2)
#include <stdio.h>
int main(){
a=1;
}
화면(data3)
#include <stdio.h>
int main(){
a=1;
b=2;
}
화면(data4)
data1
코드 작성
push(data2)
data2
data1
코드 작성
push(data3)
data3
data2
data1
코드 작성
push(data4)
data4
data3
data2
data1
되돌리기
pop()
data3
data2
data1
되돌리기
pop()
data2
data1
되돌리기
pop()
data1
되돌리기
pop()
화면
#include <stdio.h>
화면(data1)
#include <stdio.h>
int main(){
}
화면(data2)
#include <stdio.h>
int main(){
a=1;
}
화면(data3)

스택의 응용: 괄호 검사

- 스택을 이용하여 주어진 코드에서 소괄호 (), 중괄호 {}, 대괄호 [ ]의 짝이 맞는지 검사할 수 있음.
  - 괄호의 검사조건은 다음의 세가지임

    조건 1 : 왼 괄호와 오른 괄호의 개수가 같아야 함.

    조건 2 : 같은 타입의 괄호에서 왼 괄호는 오른 괄호보다 먼저 나와야 함.

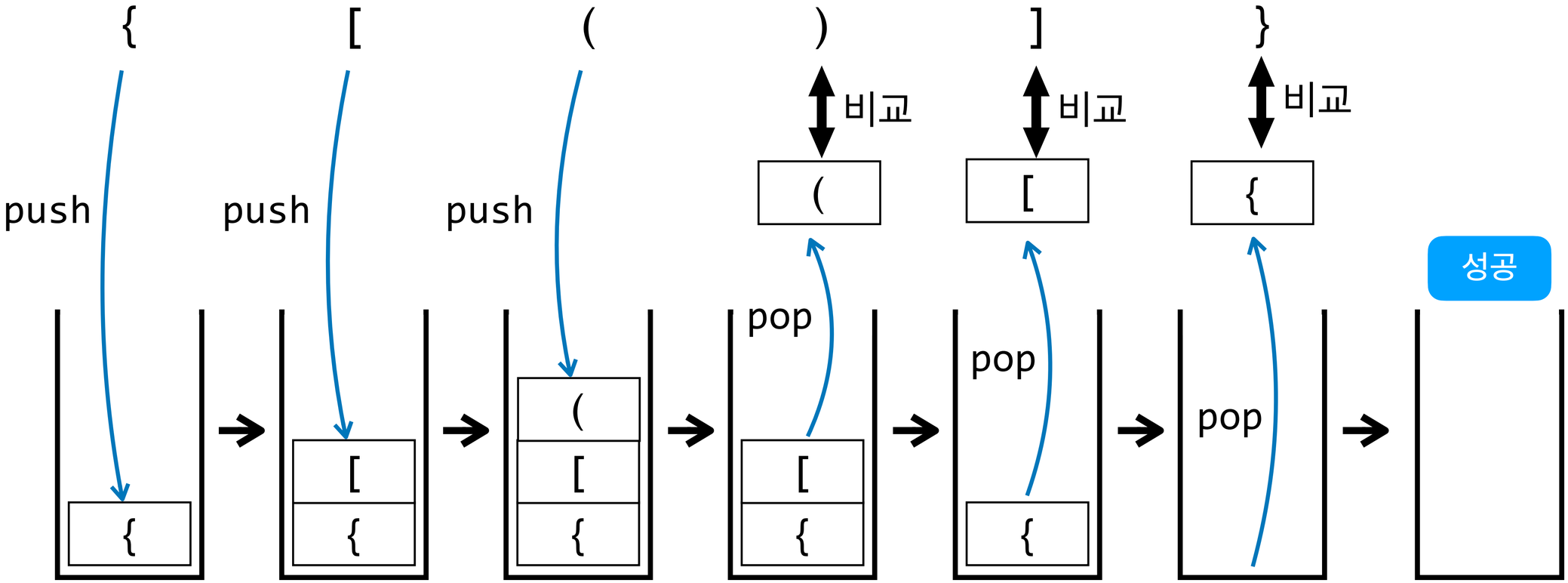
    조건 3 : 서로 다른 타입의 왼 괄호와 오른 괄호 쌍은 서로 교차하면 안됨.

코드	괄호 검사
{ A [(i + I)] = 0;}	
if((i==0) && (j==0)	
A[(I+I]) = 0;	

스택의 응용: 괄호 검사

- 스택을 이용하여 주어진 코드에서 소괄호 (), 중괄호 {}, 대괄호 [ ]의 짝이 맞는지 검사할 수 있음.

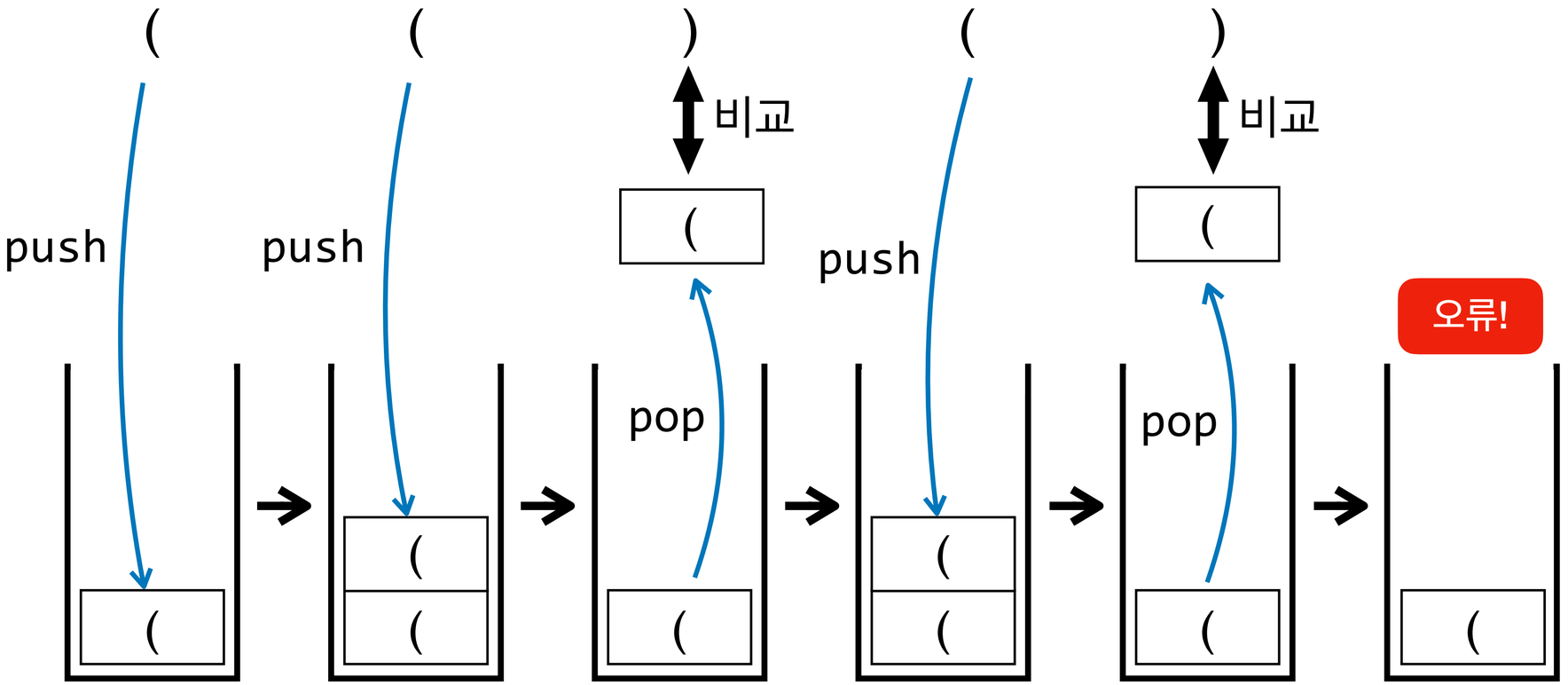
코드 : { A [(i + I)] = 0;}



스택의 응용: 괄호 검사

- 스택을 이용하여 주어진 코드에서 소괄호 (), 중괄호 {}, 대괄호 [ ]의 짝이 맞는지 검사할 수 있음.

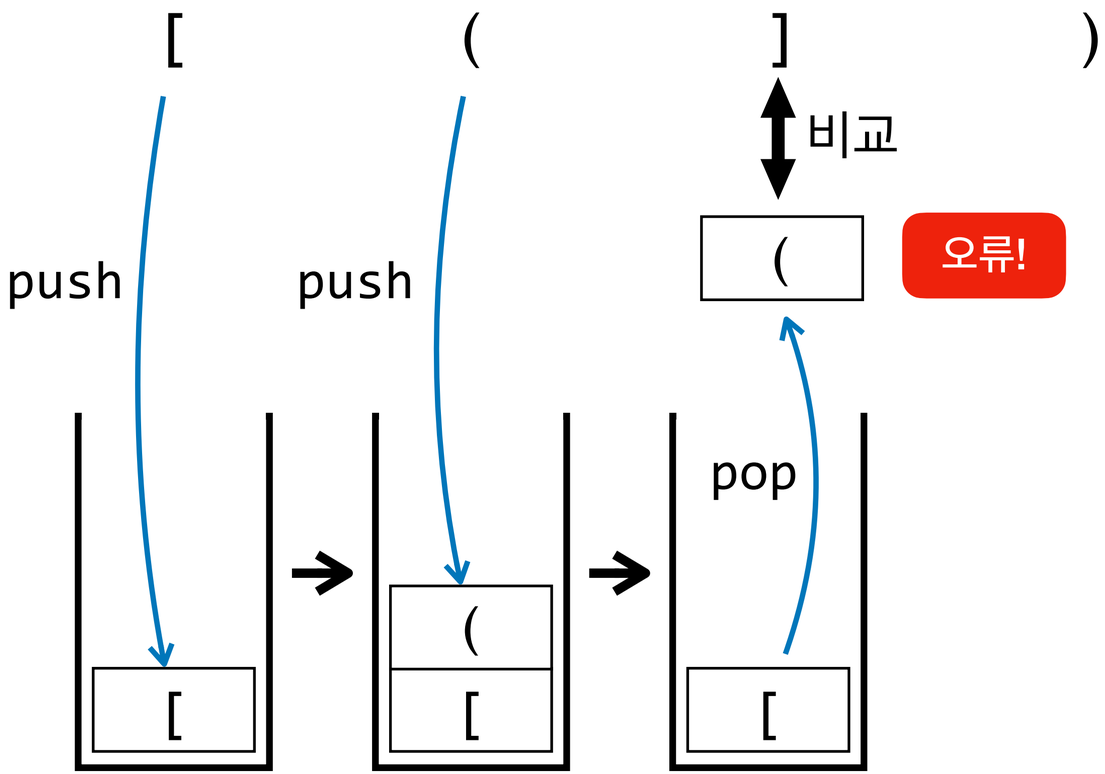
코드 : if((i==0) && (j==0)



스택의 응용: 괄호 검사

- 스택을 이용하여 주어진 코드에서 소괄호 (), 중괄호 {}, 대괄호 [ ]의 짝이 맞는지 검사할 수 있음.

코드 : A[(i+1]) = 0;



스택의 응용: 괄호 검사

```
procedure isValidParentheses(⟨t1,t2,…,tn⟩)
  s ← createStack()
  for i=1 to n do
    if ti = '(' or ti = '{' or ti = '[' then
      push(s, ti)
    end if
    if ti = ')' or ti = '}' or ti = ']' then
      x ← pop(s)
      if isMatching(x, ti) = false then
        return false
      end if
    end if
  return isEmpty(s)
end procedure
```

```
procedure isMatching(a, b)
  if a = '(' and b = ')' then
    return true
  elif a = '{' and b = '}' then
    return true
  elif a = '[' and b = ']' then
    return true
  else
    return false
  end if
end procedure
```

스택의 응용: 후위 표기 수식의 계산

- 중위 표기 수식: 연산자를 피연산자 사이에 표기하는 방법

  A+B, 5+A*B

- 후위 표기 수식: 연산자를 피연산자 뒤에 표기하는 방법

  A B +, 5 A B * +

  - 후위 표현식을 사용하는 이유

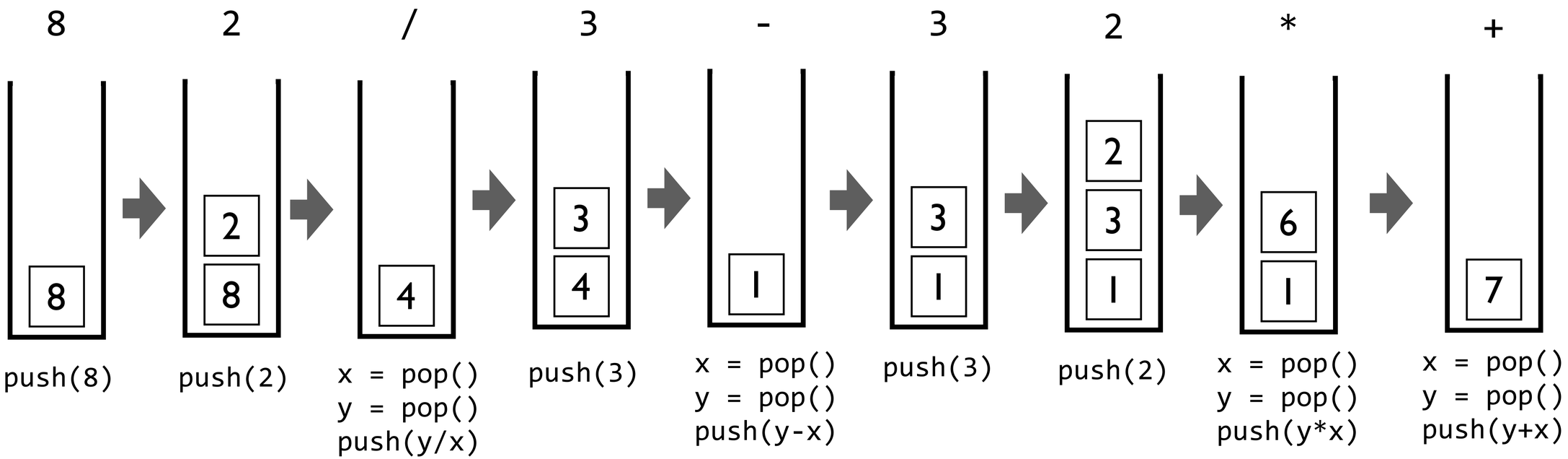
    - 괄호를 사용하지 않고도 계산해야할 순서를 명확하게 알 수 있음

    - 연산자의 우선순위를 생각할 필요 없음

    - 수식을 읽으면서 바로 계산할 수 있음

스택의 응용: 후위 표기 수식의 계산

- 후위 표현식: `8 2 / 3 - 3 2 * +`



스택의 응용: 후위 표기 수식의 계산

```
procedure evaluatePostfix(⟨t1,t2,…,tn⟩)
  s ← createStack()
  for i=1 to n do
  if isOperand(ti) then
    push(s, ti)
  else
    x ← pop(s)
    y ← pop(s)
    r ← eval(ti, x, y)
    push(r)
  end if
  return pop(s)
end procedure
```

isOperand(p): if p is an operand, return true else (i.e., operation) return false

eval(o, x, y): perform the operation o on y and x as follows:

return y + x *if o = '+':*

return y - x *if o = '-':*

*return y * x* *if o = '*':*

…

배열로 스택 (Stack) 구현하기

- 스택 (Stack.h)의 추상 자료형
  - `Stack* create()` : 비어있는 배열 스택을 생성 후 반환
  - `void push(Stack* s, int item)`: 스택의 맨 위에 주어진 새로운 정수 데이터를 추가
  - `int pop(Stack* s)` : 스택의 가장 위에 있는 데이터를 삭제하고 반환
  - `int peek(Stack* s)` : 스택의 맨 위에 있는 데이터을 제거하지 않고 반환
  - `bool isEmpty(Stack* s)` : 스택가 비어있으면 `true`를 아니면 `false`를 반환
  - `bool isFull(Stack* s)` : 스택이 가득 차 있으면 `true`를 아니면 `false`를 반환
  - `void destroy(Stack* s)` : 스택이 사용하고 있는 메모리를 해제함

Example

```c
#include "Stack.h"
#include <stdio.h>

int main() {
    Stack* s = create();

    printf("Stack is empty:  %d\n", isEmpty(s));

    push(s, 10);
    printf("Top element is %d\n", peek(s));

    push(s, 20);
    push(s, 30);

    int data = pop(s);

    printf("Popped element is %d\n", data);

    printf("Now, top element is %d\n", peek(s));
    destroy(s);
    return 0;
}
```

배열로 스택 (Stack) 구현하기

- 스택(Stack)은 다음과 같은 정보를 가지는 자료구조임

```c
#define MAX 100  // capacity

typedef struct {
    int *items;
    int top;
    int capacity;
} Stack;
```

- create : 비어있는 스택을 생성 후 반환

```
procedure create():
  stack ← allocateStack()
  stack.items ← allocateArray()
  stack.top ← -1
 Stack.capacity ← maxCapacity()
  return stack
end procedure
```

슈도코드

```c
Stack* create() {



}
```

C 언어 구현 (ToDo)

배열 스택 (Array Stack)

- `isEmpty` : 스택이 비어있으면 `true`를 아니면 `false`를 반환

```
procedure isEmpty(stack)
    if stack.top = -1 then
        return true
    else
        return false
    end if
end procedure
```

```
bool isEmpty(Stack* s) {



}
```

- `isFull` : 스택이 가득 차 있으면 `true`를 아니면 `false`를 반환

```
procedure isFull(stack)
    if stack.top = stack.capacity - 1 then
        return true
    else
        return false
    end if
end procedure
```

```
bool isFull(Stack* s) {



}
```

배열 스택 (Array Stack)

- push : 스택의 맨 위에 주어진 새로운 데이터를 추가

```
procedure push(stack, item)
  if isFull(stack) then
    print("Cannot push. Stack is full.")
    return error()                          ▷ failed
  else
    stack.top ← stack.top + 1
    stack.items[stack.top] ← item
    return                                  ▷ stack is updated
  end if
end procedure
```

```
  void push(Stack* s, int item) {




  }
```

배열 스택 (Array Stack)

- pop : 스택의 맨 위에 있는 데이터를 삭제하고 반환

```
procedure pop(stack)
    if isEmpty(stack) then
        print (“Cannot pop. Stack is empty.”)
        return error()                          ▷ failed
    else
        data ← stack.items[stack.top]
        stack.top ← stack.top - 1
        return data              ▷ stack is also updated
  end if
end procedure
```

```
int pop(Stack* s) {









}
```

배열 스택 (Array Stack)

- peek : 스택의 맨 위 데이터를 제거하지 않고 반환

```
procedure peek(stack)
    if isEmpty(stack) then
        print (“Cannot peek. Stack is empty”)
        return error()                          ▷ failed
    else
        return stack.items[stack.top]
  end if
end procedure
```

```c
int peek(Stack* s) {/*todo*/









}
```

배열 스택 (Array Stack)

- free : 스택이 사용하고 있는 메모리를 해제함

```
procedure destroy(stack)
  free(stack.items)
  free(stack)
end procedure
```

```
void destroy(Stack* s) {




}
```

마무리 (Wrap-up)

- 문제:
  - 종종 **가장 최근 (Last) 데이터**만을 접근, 추가, 삭제 해야하는 상황이 있음 (e.g., 에디터에서 되돌리기, 웹 브라우저 뒤로가기).
- 해결책:
  - 스택(Stack): 후입선출(LIFO: Last In, First Out) 원칙을 따르는 자료구조
  - 스택 자료구조는 다음의 기능들을 제공함 (스택의 추상 자료형):
    - create() : 비어있는 스택을 생성 후 반환
    - `push(s,x)` : 스택 s의 맨 위에 주어진 새로운 요소 x를 추가
    - `pop(s)` : 스택 s의 맨 위에 있는 요소를 삭제하고 반환
    - `peek(s)` : 스택 s의 맨 위 항목을 제거하지 않고 반환
    - ...