

데이터구조와 알고리즘

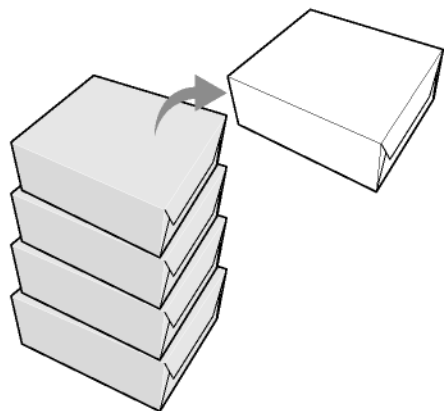
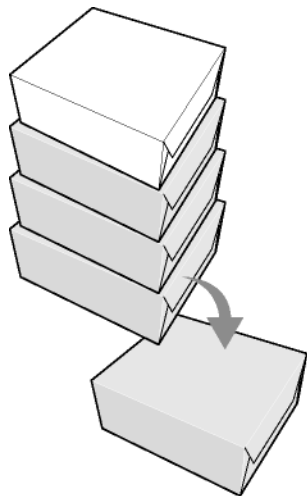
Stack

남춘성

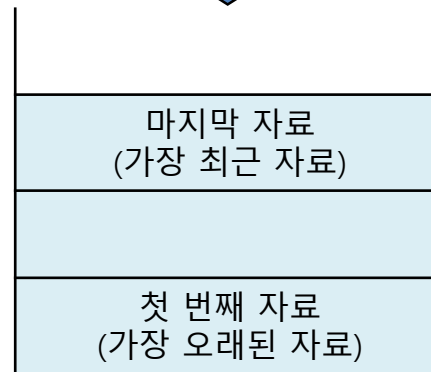
스택(stack)

- 스택 (Stack)

- 접시를 쌓듯이 자료를 차곡차곡 쌓아 올린 형태의 자료구조
- 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)
- **후입선출 구조 (LIFO, Last-In-First-Out)**

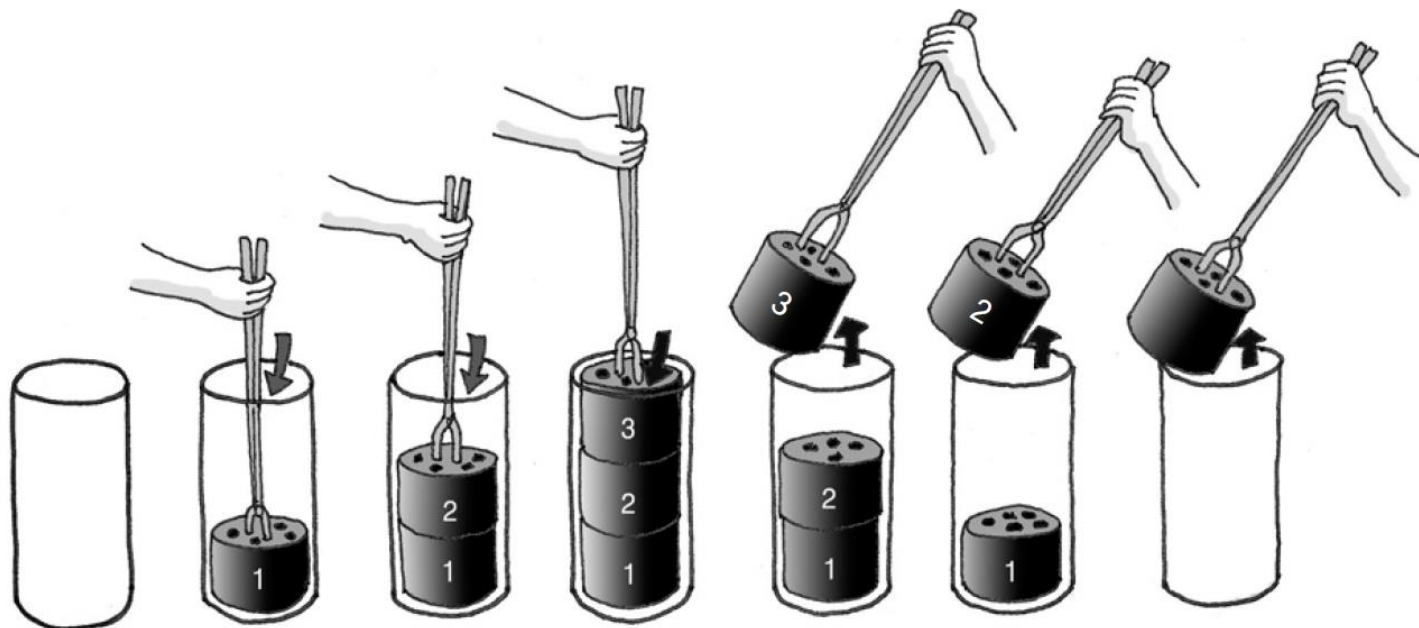


액세스(삽입/삭제)



스택(stack)

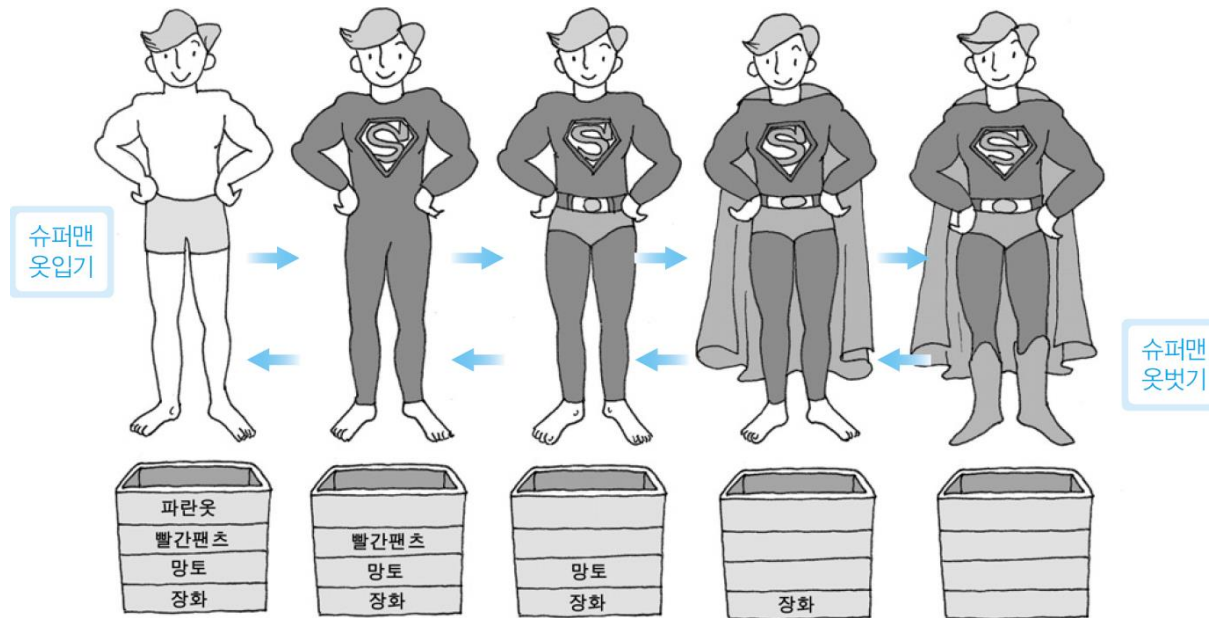
- 후입선출 구조의 예1 : 연탄 아궁이
 - 연탄을 하나씩 쌓으면서 아궁이에 넣으므로 마지막에 넣은 3번 연탄이 가장 위에 쌓여 있다.
 - 연탄을 아궁이에서 꺼낼 때에는 위에서부터 하나씩 꺼내야 하므로 마지막에 넣은 3번 연탄을 가장 먼저 꺼내게 된다.



[그림 6-3] 스택 자료구조의 예 : 연탄아궁이

스택(stack)

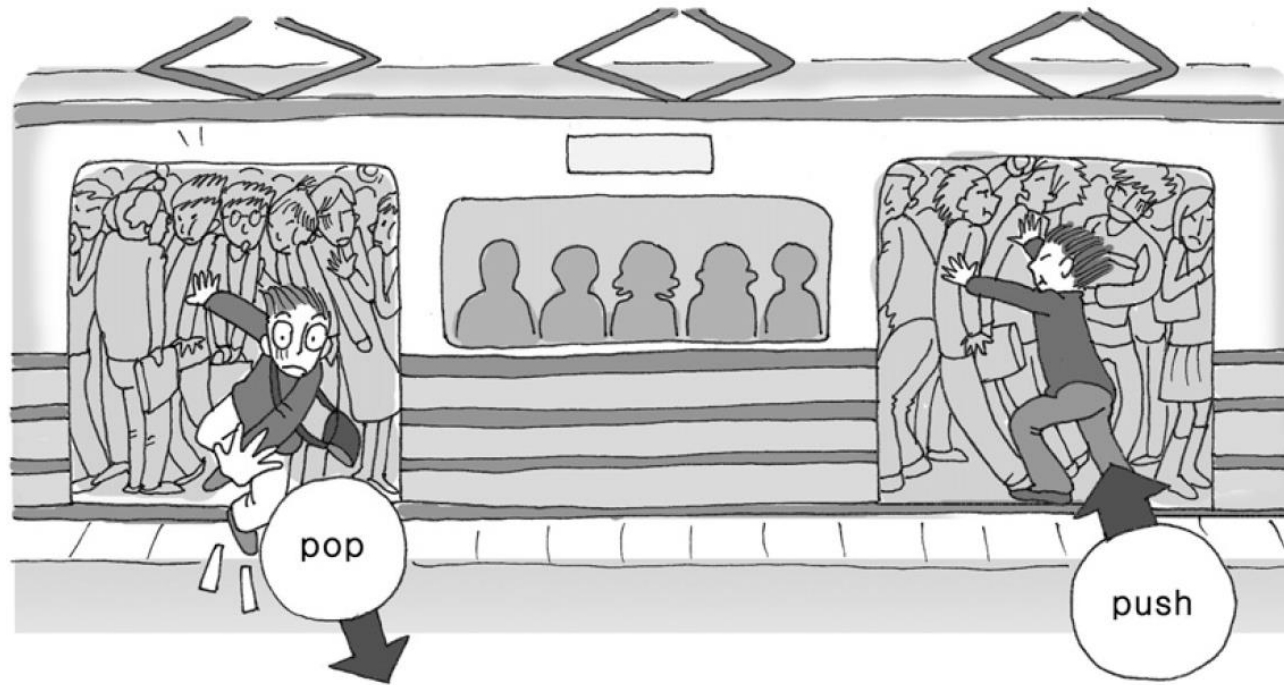
- 후입선출 구조의 예2 : 슈퍼맨의 옷 갈아입기
 - 슈퍼맨이 옷을 벗는 순서
 - ①장화 → ②망토 → ③빨간팬츠 → ④파란옷
 - 슈퍼맨이 옷을 입는 순서
 - ④파란옷 → ③빨간팬츠 → ②망토 → ①장화



[그림 6-4] 스택 자료구조의 예: 슈퍼맨 옷 입기

스택(stack)

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

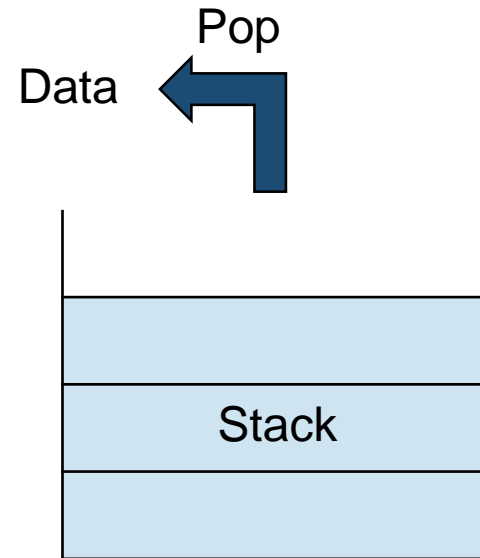
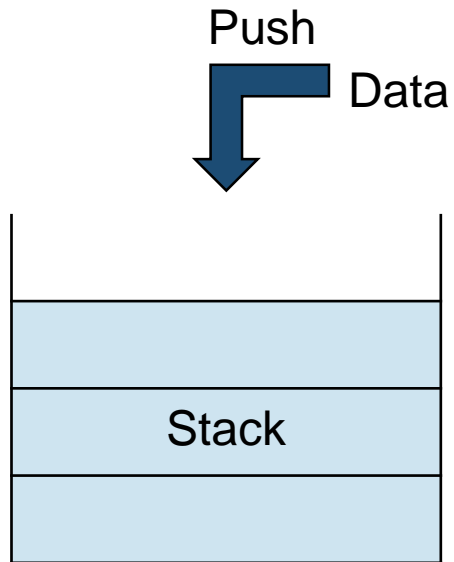


[그림 6-5] 만원 전철에서의 pop과 push

스택(stack)

- 스택(Stack)의 연산

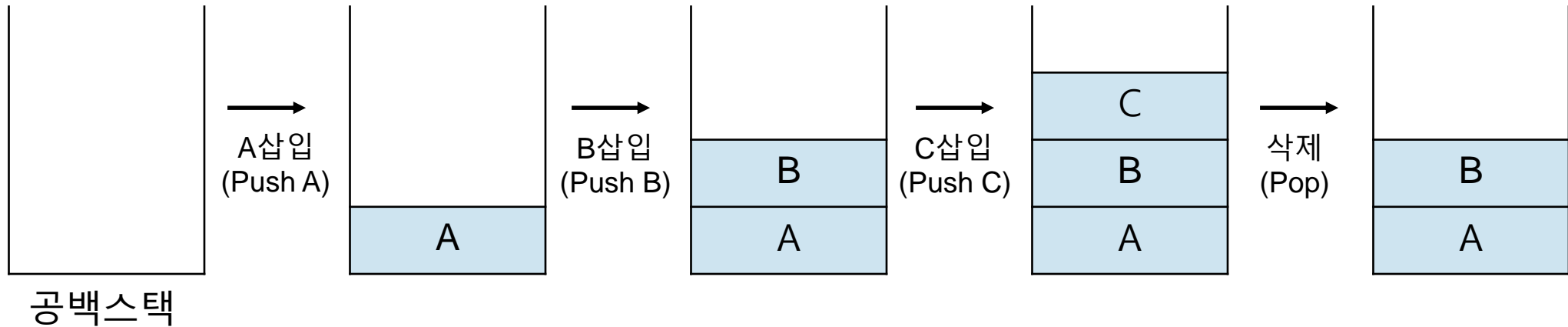
- 스택(Stack)에서의 삽입 연산 : push
- 스택(Stack)에서의 삭제 연산 : pop
- 스택(Stack)에서의 피크 연산 : peak



스택(stack)

- 스택에서의 원소 삽입/삭제 과정

- 공백 스택에 원소 A, B, C를 순서대로 삽입하고 한번 삭제하는 연산과정 동안의 스택 변화



스택(stack)

ADT Stack

데이터 : 0개 이상의 원소를 가진 유한 순서 리스트

연산 :

$S \in \text{Stack}; \text{item} \in \text{Element};$

createStack(S) ::= create an empty stack S; // 공백 스택 S를 생성하는 연산

push(S, item) ::= insert item onto the top of Stack S; // 스택 S의 top에 item(원소)을 삽입하는 연산

isEmpty(S) ::= **if** (S is empty) **then return true** // 스택 S가 공백인지 아닌지를 확인하는 연산
else return false;

pop(S) ::= **if** (isEmpty(S)) **then return** error // 스택 S의 top에 있는 item(원소)을 스택 S에서 삭제하고 반환하는 연산
else { delete and return the top item of Stack S};

delete(S) ::= **if** (isEmpty(S)) **then return** error // 스택 S의 top에 있는 item(원소)을 삭제하는 연산
else delete the top item of Stack S;

peek(S) ::= **if** (isEmpty(S)) **then return** error // 스택 S의 top에 있는 item(원소)을 반환하는 연산
else return (the top item of the Stack S);

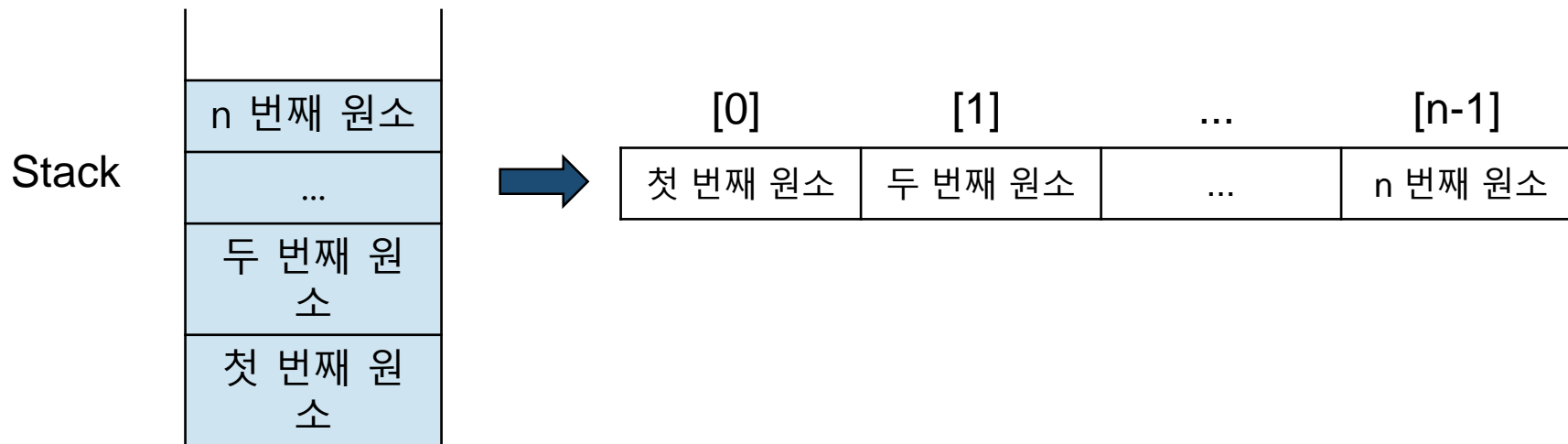
End Stack

스택(stack)

- 자료구조를 이용한 스택의 구현

- 자료구조인 배열을 이용하여 구현

- 스택의 크기 : 배열의 크기
- 스택에 저장된 원소의 순서 : 배열 원소의 인덱스
 - ✓ 인덱스 0번 : 스택의 첫번째 원소
 - ✓ 인덱스 $n-1$ 번 : 스택의 n 번째 원소
- 변수 top : 스택에 저장된 마지막 원소에 대한 인덱스 저장
 - ✓ 공백 상태 : $top = -1$ (초기값)
 - ✓ 포화 상태 : $top = n-1$



스택(stack)

- 스택의 push 알고리즘

① $top \leftarrow top+1;$

- 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 $S(top) \leftarrow x;$

- 오버플로우 상태가 아니라면 스택의 top이 가리키는 위치에 x 삽입

```
push(S, x)
  top  $\leftarrow$  top+1;           // ①
  if (top > stack_SIZE) then
    overflow;
  else
    S(top)  $\leftarrow$  x;         // ②
end push()
```

스택(stack)

- 스택의 pop 알고리즘

① return S(top);

– 스택이 공백 스택이 아니라면 top이 가리키는 원소를 먼저 반환

② top \leftarrow top-1;

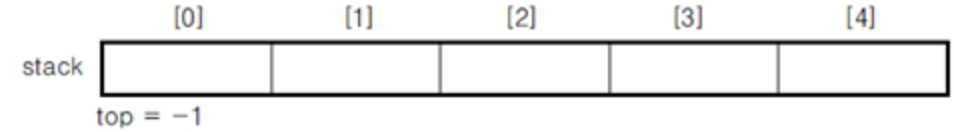
– 스택의 top 원소를 반환하였으므로 top의 위치는 그 아래의 원소로 변경하기 위해 top의 위치를 하나 감소

```
pop(S)
  if (top = 0) then underflow;
  else {
    return S(top);           // ①
    top  $\leftarrow$  top-1;       // ②
  }
end pop()
```

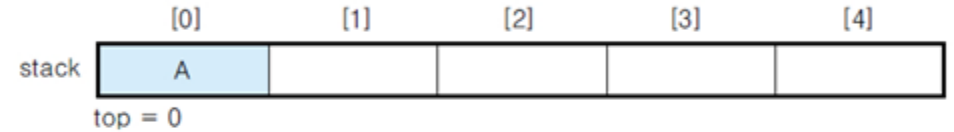
스택(stack)

- 크기가 5인 1차원 배열의 스택에서 Push/Pop의 수행과정

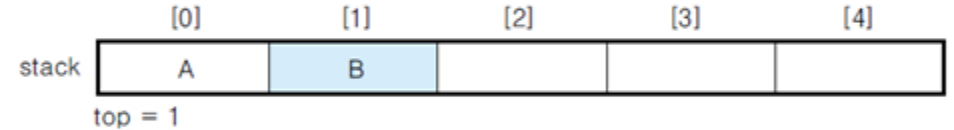
① 공백 스택 생성 : `createStack(S, 5)` ;



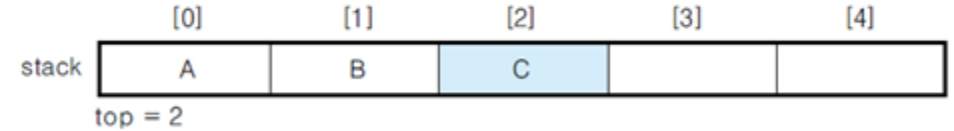
② 원소 A 삽입 : `push(S, A)`;



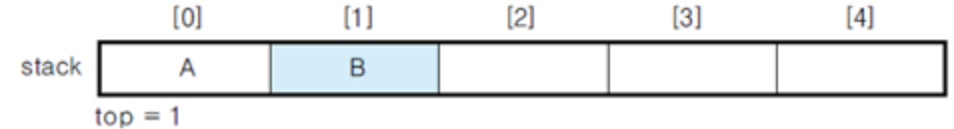
③ 원소 B 삽입 : `push(S, B)`;



④ 원소 C 삽입 : `push(S, C)`;

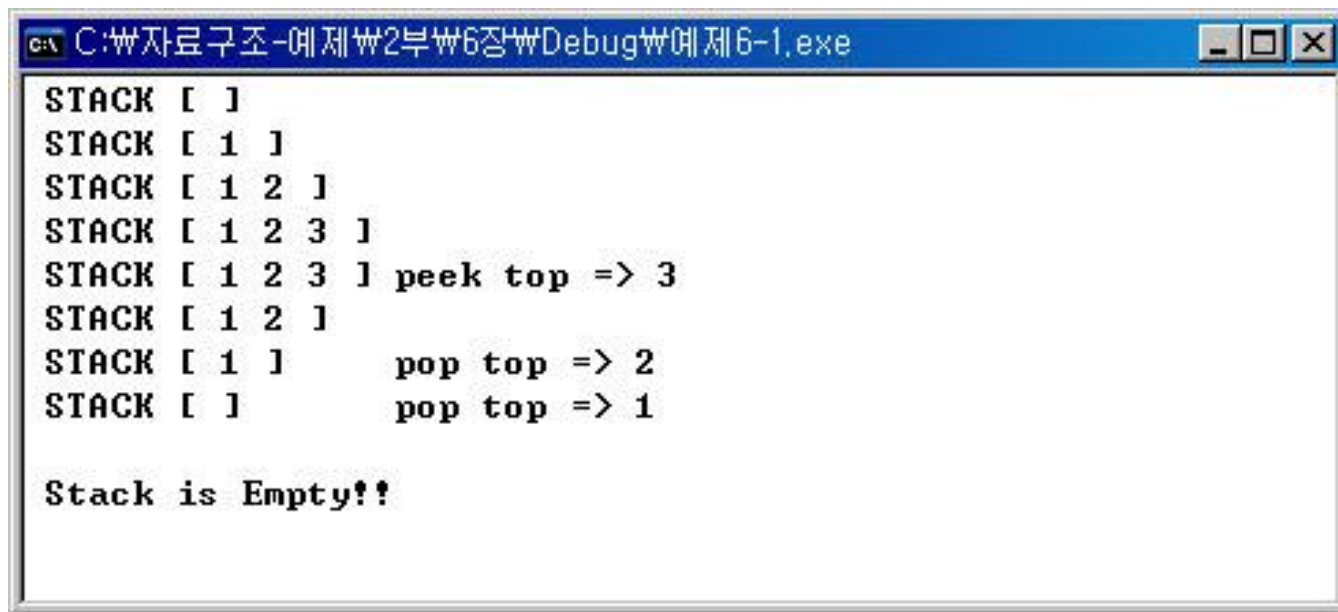


⑤ 원소 삭제 : `pop(S)`;



스택(stack)

- 실행 결과>



A screenshot of a Windows command prompt window. The title bar shows the file path: C:\자료구조-예제\2부\6장\Debug\예제6-1.exe. The window contains the following text:

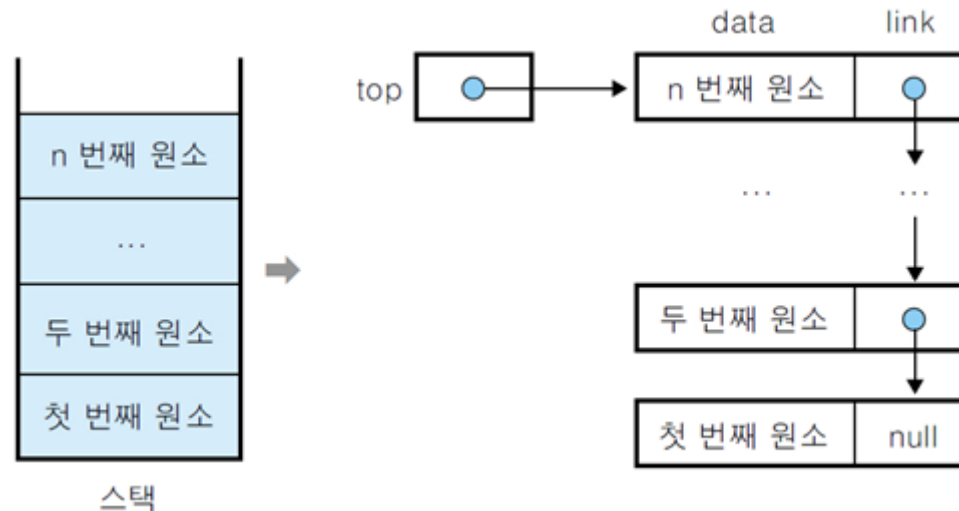
```
STACK [ ]  
STACK [ 1 ]  
STACK [ 1 2 ]  
STACK [ 1 2 3 ]  
STACK [ 1 2 3 ] peek top => 3  
STACK [ 1 2 ]  
STACK [ 1 ]      pop top => 2  
STACK [ ]        pop top => 1  
  
Stack is Empty!!
```

스택(stack)

- 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 그대로 가지고 있다.

스택(stack)

- 연결 자료구조를 이용한 스택의 구현
 - 단순 연결 리스트를 이용하여 구현
 - 스택의 원소 : 단순 연결 리스트의 노드
 - 스택 원소의 순서 : 노드의 링크 포인터로 연결
 - push : 리스트의 마지막에 노드 삽입
 - pop : 리스트의 마지막 노드 삭제
 - 변수 top : 단순 연결 리스트의 마지막 노드를 가리키는 포인터 변수
 - 초기 상태 : top = null



스택(stack)

- 단순 연결 리스트의 스택에서 [그림6-6]의 연산 수행과정

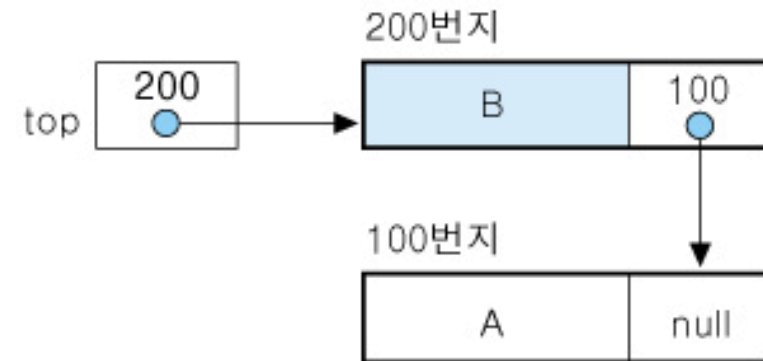
① 공백 스택 생성 : `createStack(S);`



② 원소 A 삽입 : `push(S, A);`

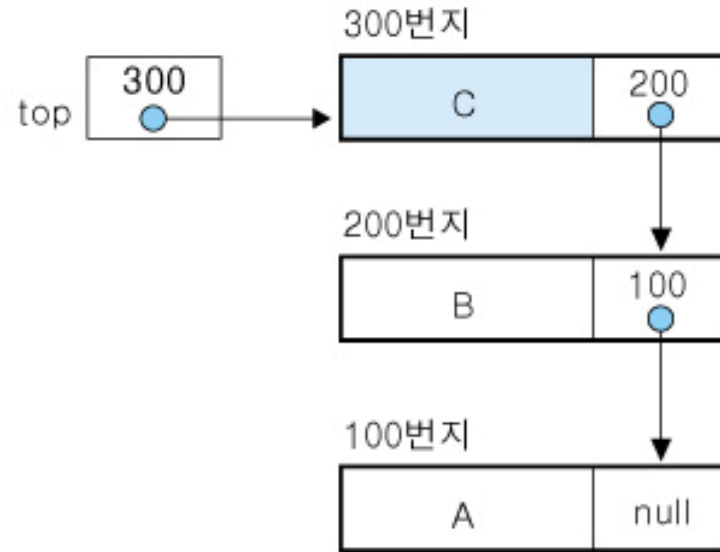


③ 원소 B 삽입 : `push(S, B);`

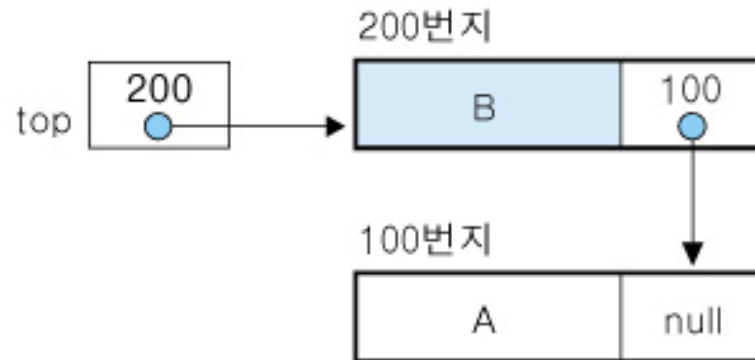


스택(stack)

④ 원소 C 삽입 : `push(S, C);`



⑤ 원소 삭제 : `pop(S);`



스택(stack) - 응용 1

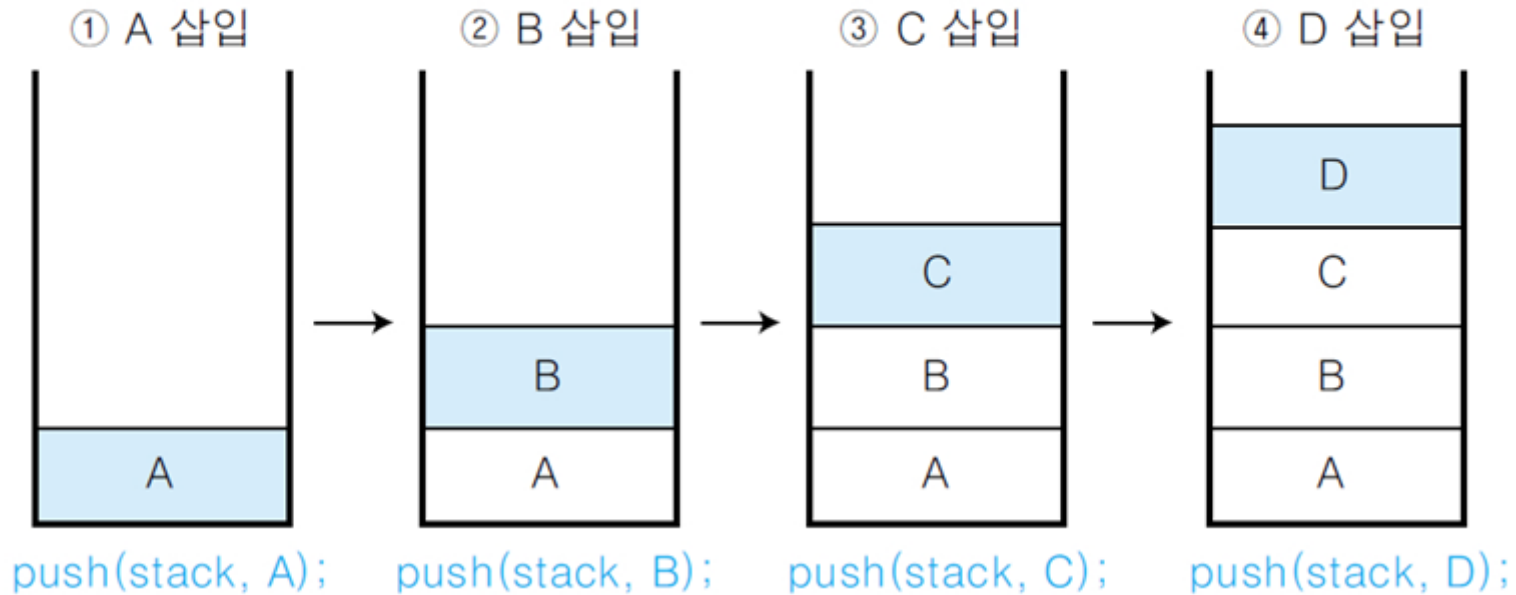
- 역순 문자열 만들기

- 스택의 후입선출(LIFO) 성질을 이용

- ① 문자열을 순서대로 스택에 push 하기

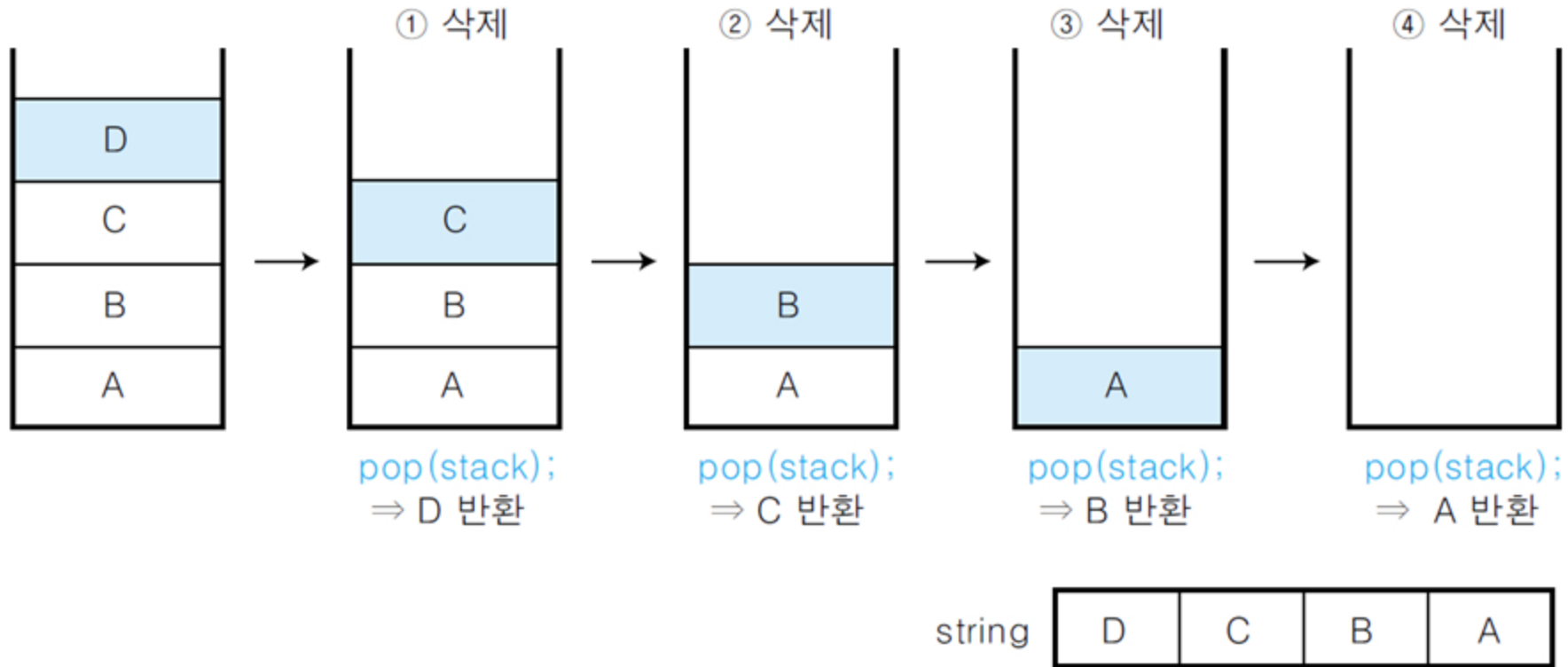
string

A	B	C	D
---	---	---	---



스택(stack) - 응용 1

② 스택을 pop하여 문자열로 저장하기



스택(stack) – 응용 2

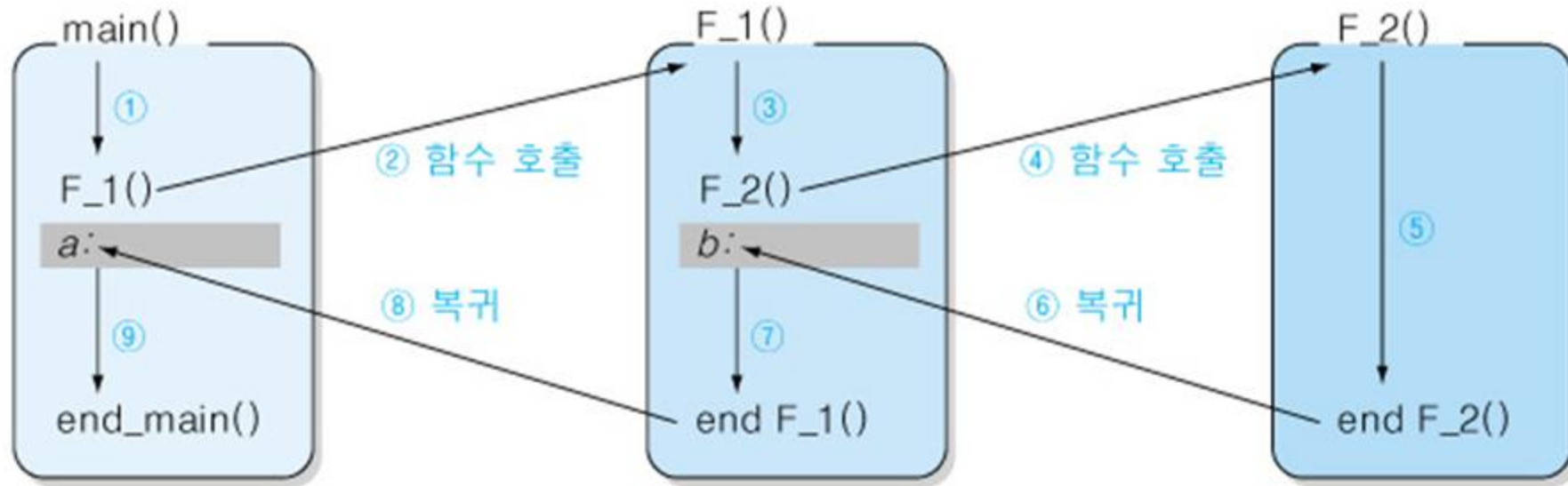
- 시스템 스택

- 프로그램에서의 호출과 복귀에 따른 수행 순서를 관리

- 가장 마지막에 호출된 함수가 가장 먼저 실행을 완료하고 복귀하는 후입선출 구조이므로, 후입선출 구조의 스택을 이용하여 수행순서 관리
 - 함수 호출이 발생하면 호출한 함수 수행에 필요한 지역변수, 매개변수 및 수행 후 복귀할 주소 등의 정보를 스택 프레임(stack frame)에 저장하여 시스템 스택에 삽입
 - 함수의 실행이 끝나면 시스템 스택의 top 원소(스택 프레임)를 삭제(pop)하면서 프레임에 저장되어있던 복귀주소를 확인하고 복귀
 - 함수 호출과 복귀에 따라 이 과정을 반복하여 전체 프로그램 수행이 종료되면 시스템 스택은 공백스택이 된다.

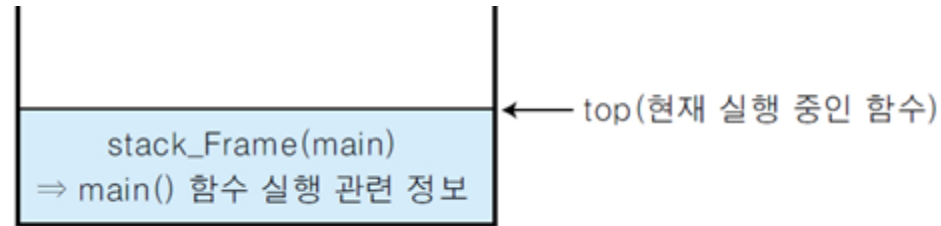
스택(stack) – 응용 2

- 함수 호출과 복귀에 따른 전체 프로그램의 수행 순서



스택(stack) – 응용 2

- ① 프로그램이 실행을 시작하여 main() 함수가 실행되면서 main() 함수에 관련된 정보를 스택 프레임에 저장하여 시스템 스택에 삽입



- ② main() 함수 실행 중에 F_1() 함수 호출을 만나면 함수 호출과 복귀에 필요한 정보를 스택 프레임에 저장하여 시스템 스택에 삽입하고, 호출된 함수인 F_1() 함수로 이동
- 이때 스택 프레임에는 호출된 함수의 수행이 끝나고 main() 함수로 복귀할 주소 a를 저장

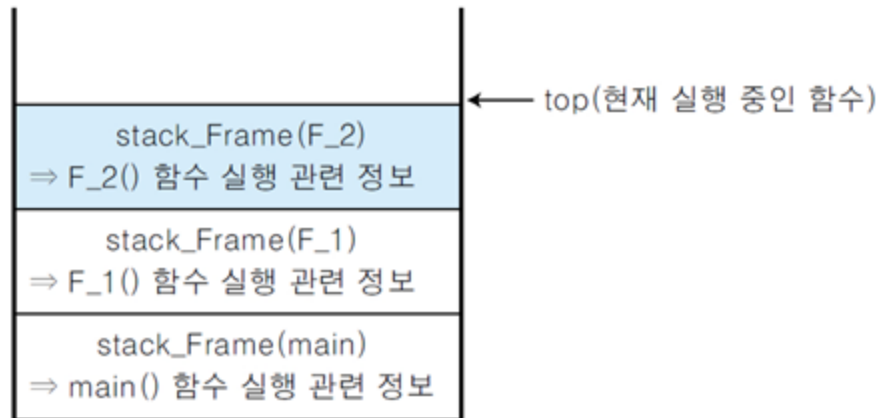
```
push(System_stack, stack_Frame(F_1));
```



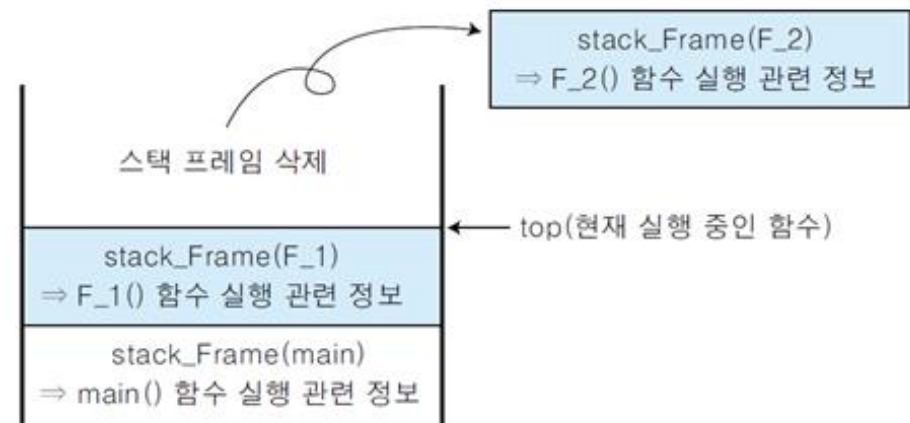
스택(stack) – 응용 2

- ③ 호출된 함수 F_1() 함수를 실행
- ④ F_1() 함수 실행 중에 F_2() 함수 호출을 만나면 다시 함수 호출과 복귀에 필요한 정보를 스택 프레임에 저장하여 시스템 스택에 삽입하고, 호출된 함수인 F_2() 함수를 실행
 - 스택 프레임에는 F_1() 함수로 복귀할 주소 b를 저장
- ⑤ 호출된 함수 F_2() 함수를 실행 완료
- ⑥ 시스템 스택의 top에 있는 스택 프레임을 pop하여 정보를 확인하고, F_1() 함수의 b 주소로 복귀

`push(System_stack, stack_Frame(F_2));`



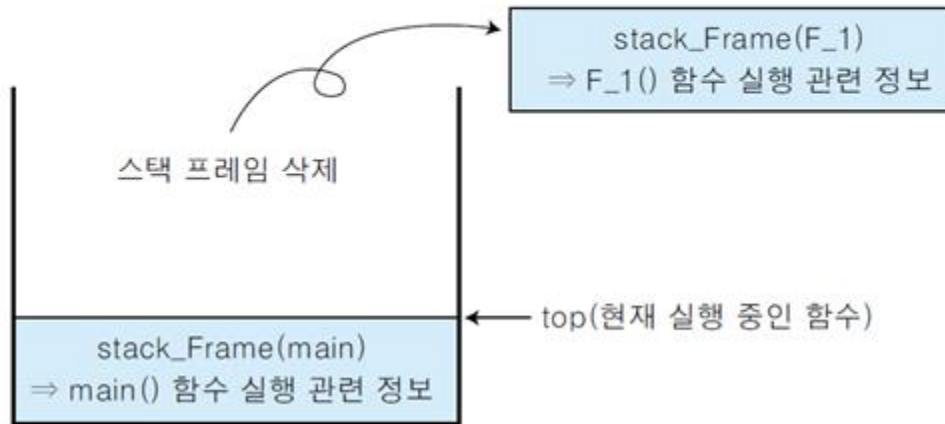
`pop(System_stack);`



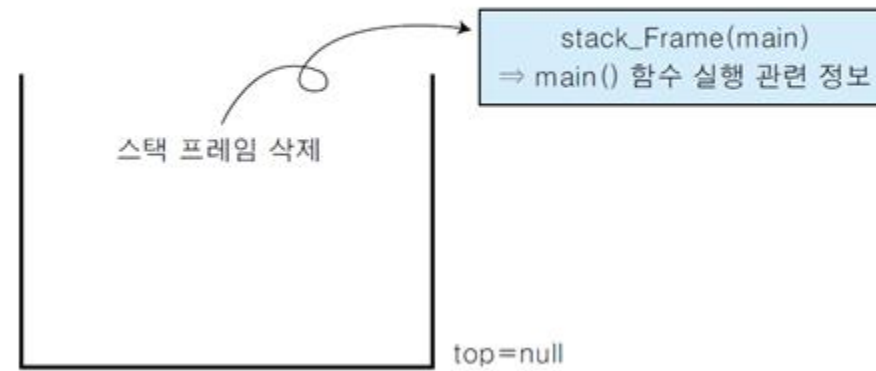
스택(stack) - 응용 2

- ⑦ 복귀된 함수 F_1() 함수의 b주소 이후 부분을 실행
- ⑧ F_1()함수 실행이 완료되면, 시스템 스택의 top에 있는 스택 프레임을 pop하여 정보를 확인하고 main() 함수의 a주소로 복귀
- ⑨ 복귀된 main() 함수의 a주소 이후 부분에 대한 실행이 완료되면 시스템 스택의 top에 있는 스택 프레임을 pop하는데, 시스템 스택이 공백이 되었으므로 전체 프로그램 실행 종료

pop(System_stack);



pop(System_stack);



스택(stack) – 응용 3

- 수식의 괄호 검사

- 수식에 포함되어있는 괄호는 가장 마지막에 열린 괄호를 가장 먼저 닫아 주어야 하는 후입선출 구조로 구성되어있으므로, 후입선출 구조의 스택을 이용하여 괄호를 검사한다.
- 수식을 왼쪽에서 오른쪽으로 하나씩 읽으면서 괄호 검사

- ① 왼쪽 괄호를 만나면 스택에 **push**
- ② 오른쪽 괄호를 만나면 스택을 **pop**하여 마지막에 저장한 괄호와 같은 종류 인지를 확인
 - 같은 종류의 괄호가 아닌 경우 괄호의 짝이 잘못 사용된 수식임.

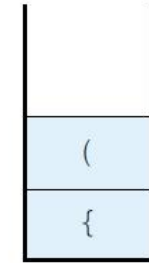
- 수식에 대한 검사가 모두 끝났을 때 스택은 공백 스택이 됨
 - 수식이 끝났어도 스택이 공백이 되지 않으면 괄호의 개수가 틀린 수식임.

스택(stack) - 응용 3

$\{(A+B)-3\} * 5 + [\{\cos(x+y)+7\}-1] * 4$

$\{ (A + B) - 3 \} * 5 + [\{ \cos(x + y) + 7 \} - 1] * 4$

① push(stack, {)
② push(stack, ()



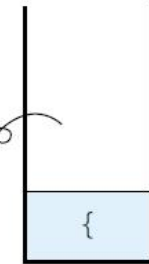
스택 stack

$\{ (A + B) - 3 \} * 5 + [\{ \cos(x + y) + 7 \} - 1] * 4$

③ pop(stack)

같은 종류인지 확인

삭제된 원소 : (



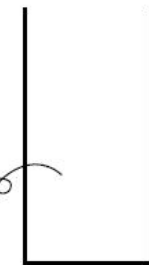
스택 stack

$\{ (A + B) - 3 \} * 5 + [\{ \cos(x + y) + 7 \} - 1] * 4$

④ pop(stack)

같은 종류인지 확인

삭제된 원소 : {

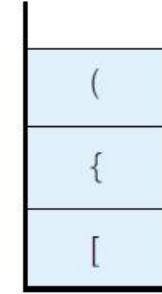


스택 stack

스택(stack) - 응용 3

{ (A + B) - 3 } * 5 + [{cos(x + y) + 7} - 1] * 4

⑦ push(stack, ()
⑥ push(stack, {)
⑤ push(stack, [)



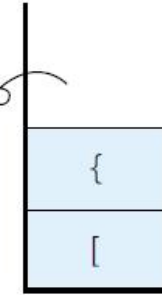
스택 stack

{ (A + B) - 3 } * 5 + [{cos(x + y) + 7} - 1] * 4

⑧ pop(stack)

삭제된 원소 : (

같은 종류인지 확인

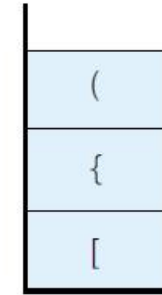


스택 stack

스택(stack) – 응용 3

{ (A + B) - 3 } * 5 + [{ cos(x + y) + 7 } - 1] * 4

⑦ push(stack, ()
⑥ push(stack, {)
⑤ push(stack, [)



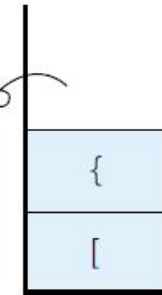
스택 stack

{ (A + B) - 3 } * 5 + [{ cos(x + y) + 7 } - 1] * 4

⑧ pop(stack)

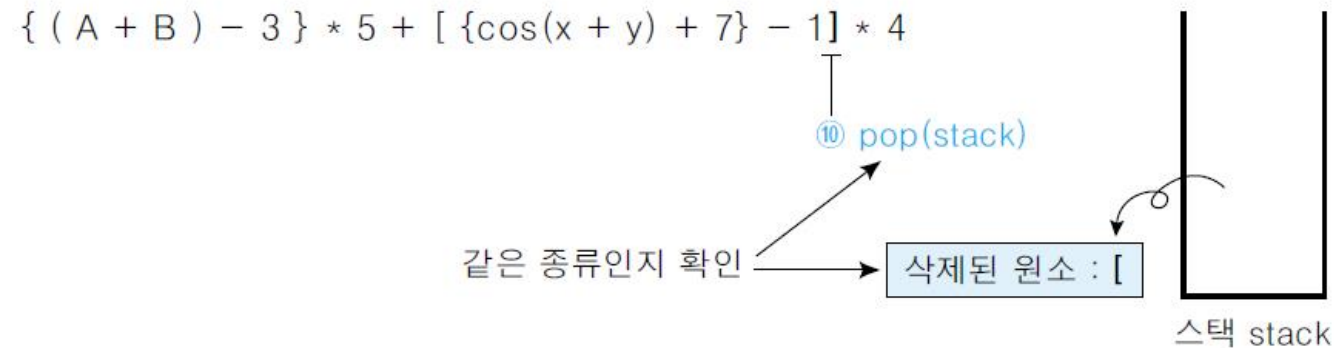
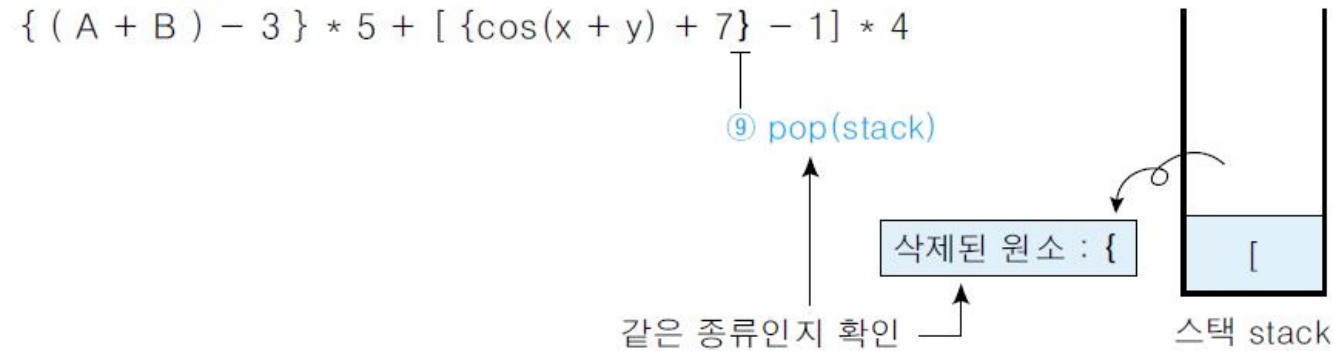
삭제된 원소 : (

같은 종류인지 확인



스택 stack

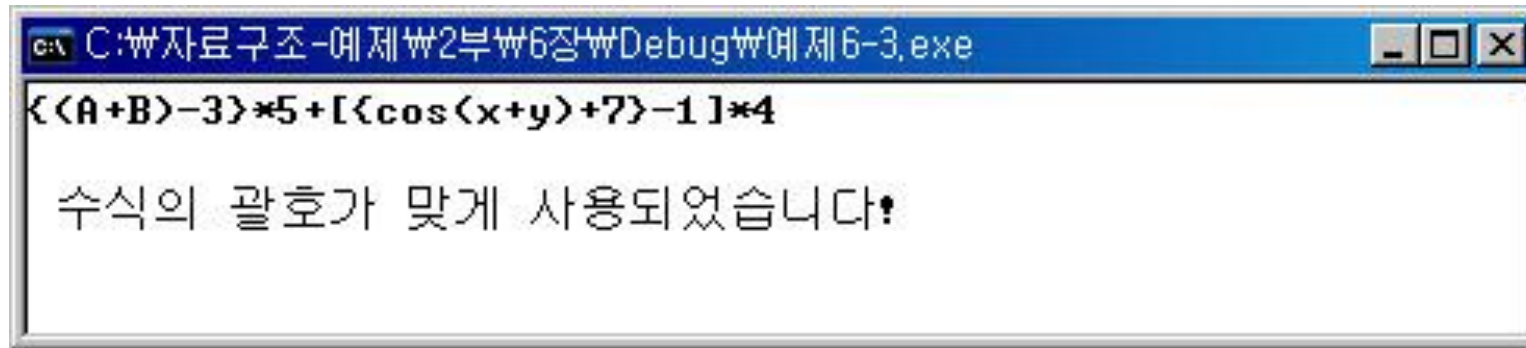
스택(stack) – 응용 3



[그림 6-20] 수식 처리에 따른 스택 사용 과정

스택(stack) – 응용 3

- 실습 1



스택(stack) – 응용 4

- 수식의 표기법
 - 전위표기법(prefix notation)
 - 연산자를 피연산자를 앞에 표기하는 방법
 - 예) +AB
 - 중위표기법(infix notation)
 - 연산자를 피연산자의 가운데 표기하는 방법
 - 예) A+B
 - 후위표기법(postfix notation)
 - 연산자를 피연산자 뒤에 표기하는 방법
 - 예) AB+

스택(stack) - 응용 4

- 중위표기식의 전위표기식 변환 방법

- ① 수식의 각 연산자에 대해서 우선순위에 따라 괄호를 사용하여 다시 표현한다.
- ② 각 연산자를 그에 대응하는 왼쪽괄호의 앞으로 이동시킨다.
- ③ 괄호를 제거한다.

• 예) $A*B-C/D$

• 1단계: $((A*B) - (C/D))$

• 2단계:

$\Rightarrow -(*(A\ B) \ / (C\ D))$

• 3단계: $-*AB/CD$

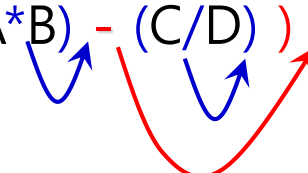
스택(stack) – 응용 4

– 중위표기식의 후위표기식 변환 방법

- ① 수식의 각 연산자에 대해서 우선순위에 따라 괄호를 사용하여 다시 표현한다.
- ② 각 연산자를 그에 대응하는 오른쪽괄호의 뒤로 이동시킨다.
- ③ 괄호를 제거한다.

• 예) $A*B-C/D$

• 1단계: $((A*B)-(C/D))$



• 2단계:

$\Rightarrow ((A\ B)^*(C\ D)/)-$

• 3단계: $AB*CD/-$

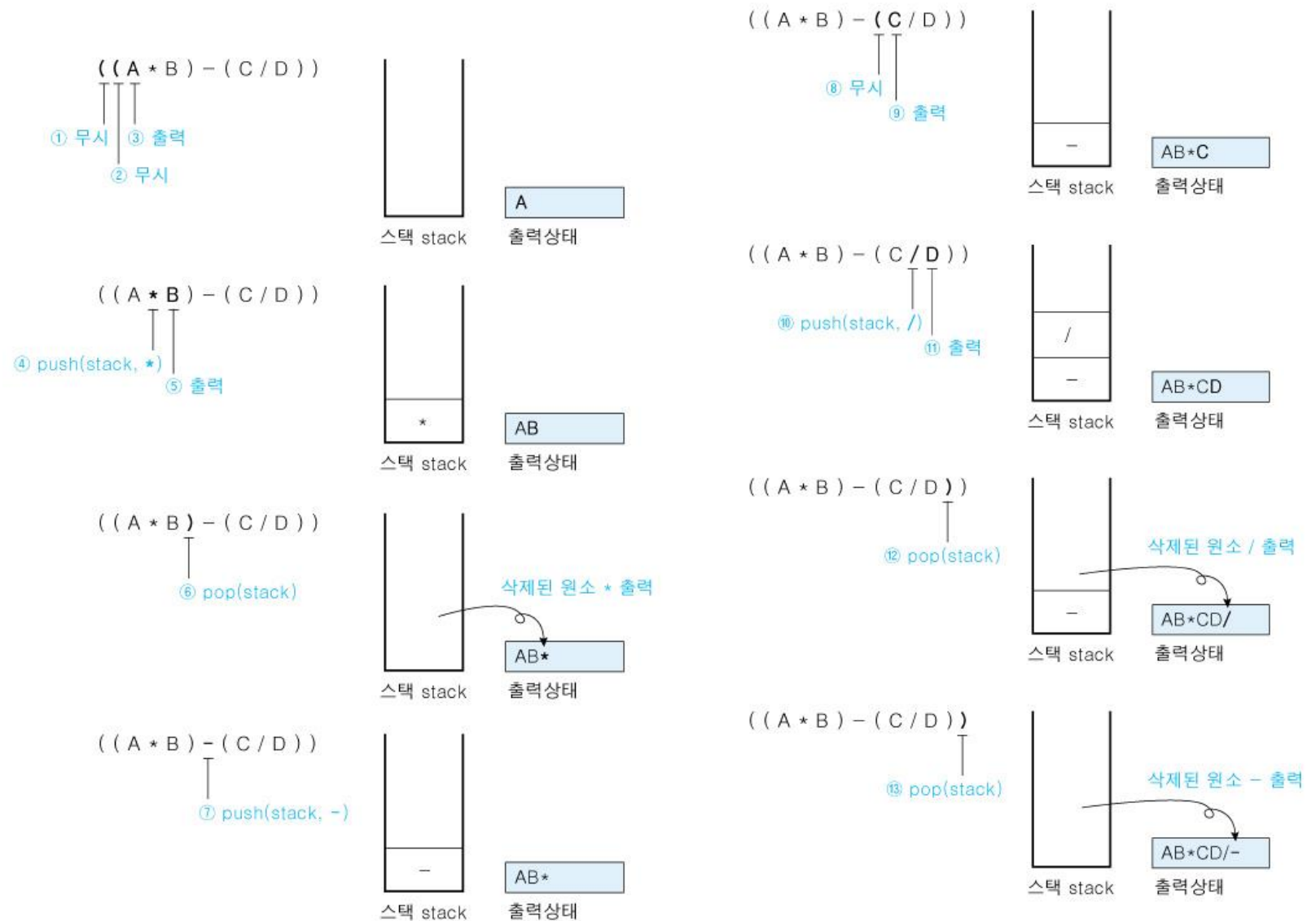
스택(stack) – 응용 4

- 스택을 사용하여 입력된 중위표기식을 후위표기식으로 변환
– 변환 방법

- (1) 왼쪽괄호를 만나면 무시하고 다음 문자를 읽는다.
- (2) **피연산자**를 만나면 **출력**한다.
- (3) **연산자**를 만나면 스택에 **push**한다.
- (4) **오른쪽괄호**를 만나면 스택을 **pop**하여 **출력**한다.
- (5) 수식이 끝나면, 스택이 공백이 될 때까지 **pop**하여 **출력**한다.

스택(stack) - 응용 4

- 예) $((A*B)-(C/D))$



[그림 6-20] 스택을 사용한 후위 표기식 변환 과정

스택(stack) – 응용 5

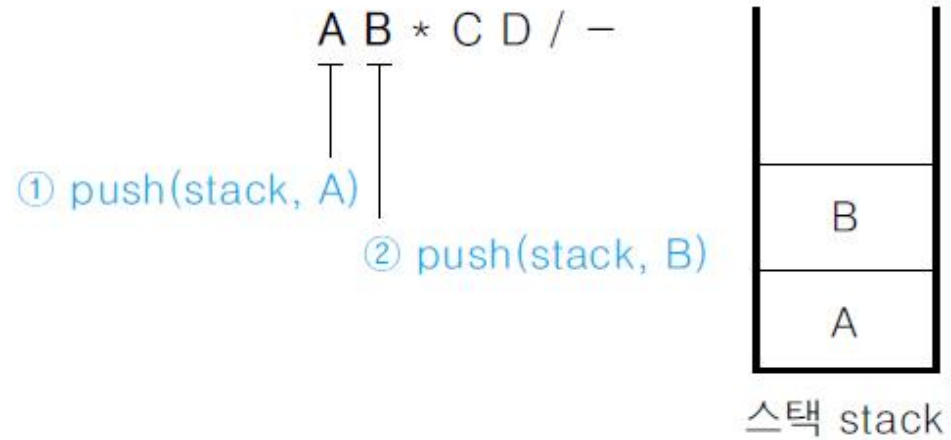
- 스택을 사용하여 후위표기식을 연산
 - 연산 방법

- (1) 피연산자를 만나면 스택에 **push** 한다.
- (2) 연산자를 만나면 필요한 만큼의 피연산자를 스택에서 **pop**하여 연산하고, 연산결과를 다시 스택에 **push** 한다.
- (3) 수식이 끝나면, 마지막으로 스택을 **pop**하여 출력한다.

- 수식이 끝나고 스택에 마지막으로 남아있는 원소는 전체 수식의 연산결과 값이 된다.

스택(stack) – 응용 5

- 예) $AB*CD/-$

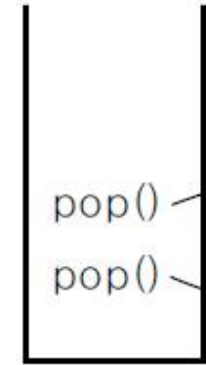


스택(stack) – 응용 5

- 예) $AB*CD/-$

A B * C D / -

③ pop(stack)
pop(stack)



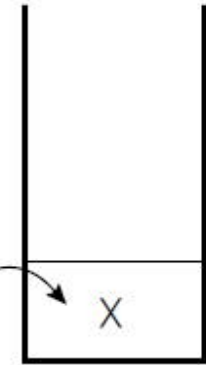
스택 stack

삭제한 피연산자 A, B를
연산자 *로 연산하기

$X \leftarrow A * B;$

연산 결과 X를 다시 스택에 삽입

push(stack, X);

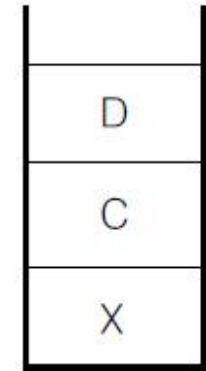


스택 stack

A B * C D / -

④ push(stack, C)

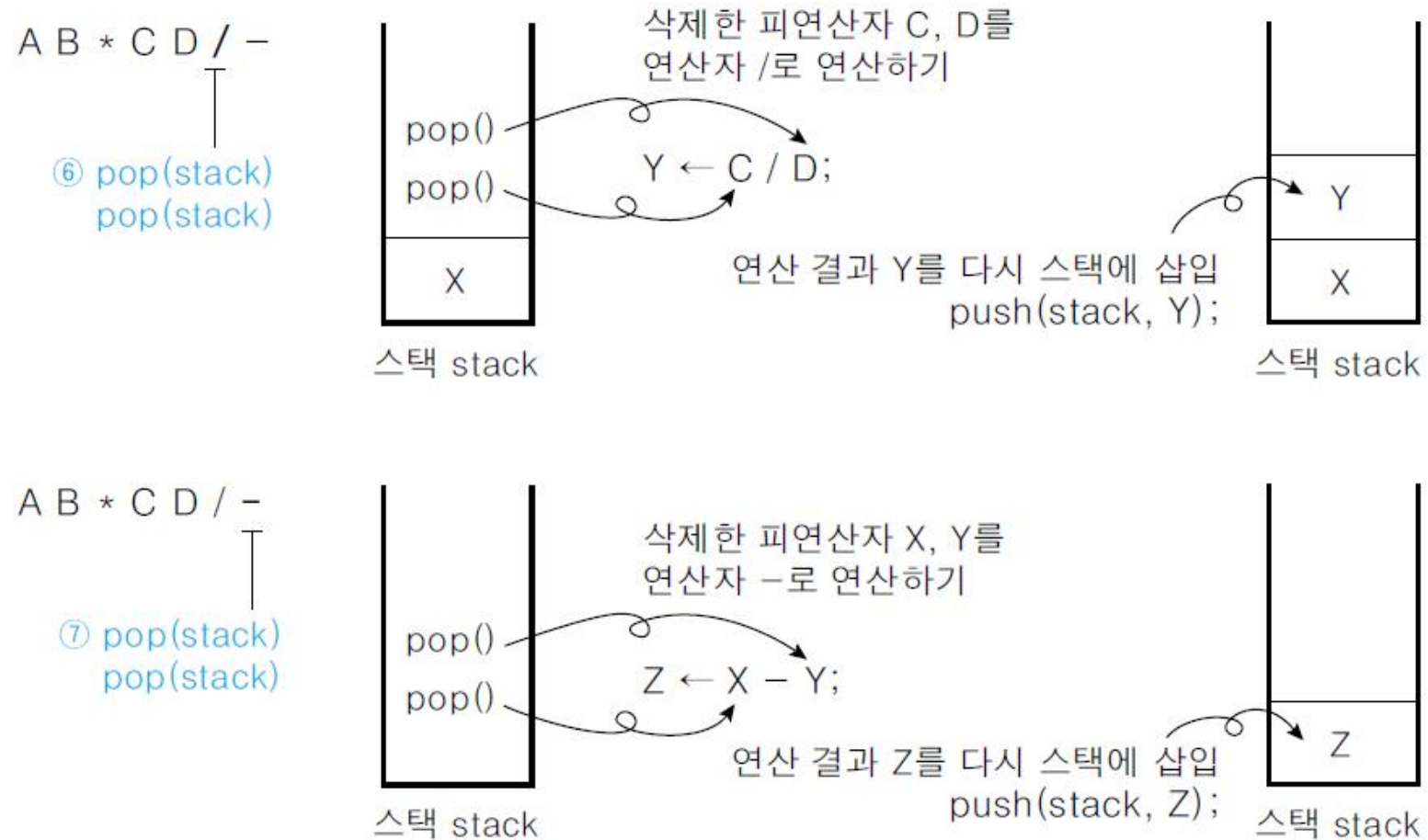
⑤ push(stack, D)



스택 stack

스택(stack) – 응용 5

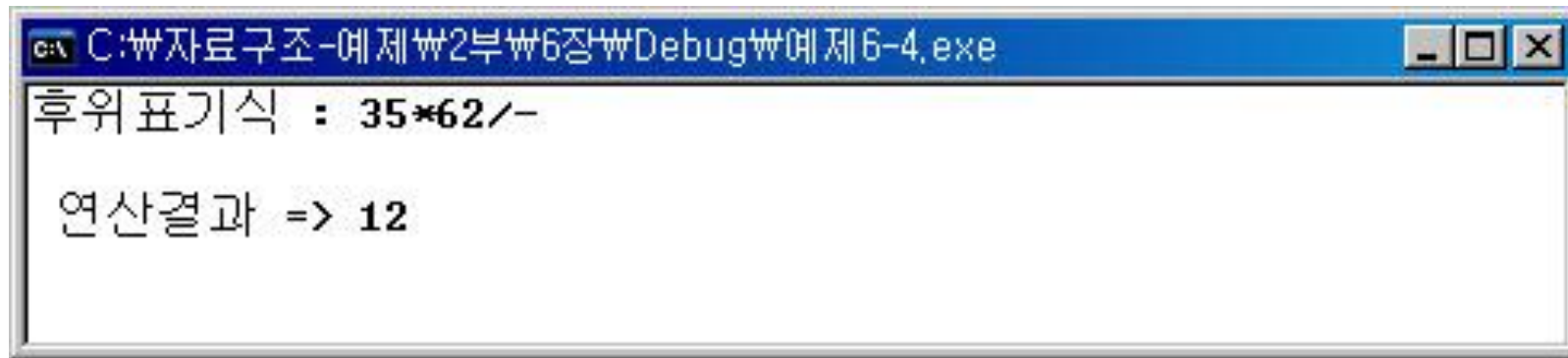
- 예) $AB*CD/-$



[그림 6-22] 스택을 사용한 후위 표기법 연산 과정

스택(stack) – 응용 5

- 실행 결과 >



```
C:\자료구조-예제\2부\6장\Debug\예제6-4.exe
후위 표기식 : 35*62/-
연산결과 => 12
```