CHAPTER 12

스택 구현 및 응용

학습 목표

- 스택에 대해 이해한다.
- 스택의 특징과 연산 방법을 알아본다.
- 순차 자료구조와 연결자료구조를 이용해 스택을 구현해 본다.
- 스택을 응용하는 방법을 알아본다.

스택의 이해 : 스택의 개념과 구조

- 스택(stack)
  - 접시를 쌓듯이 자료를 차곡차곡 쌓아 올린 형태의 자료구조

그림 5-1 스택의 개념 예

스택의 이해 : 스택의 개념과 구조

- 스택(stack)
  - 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(**L**ast-**I**n)한 원소는 맨 위에 쌓여 있다가 가장 먼저 삭제(**F**irst-**O**ut)됨 ☞ **후입선출 구조** (**LIFO**, Last-In-First-Out)

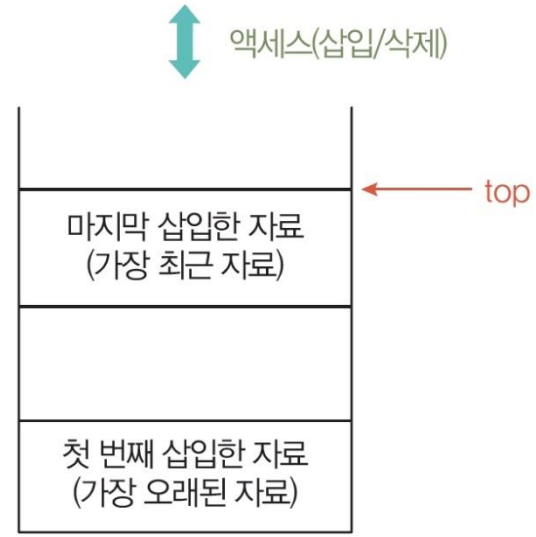


그림 5-2 스택의 구조

스택의 이해 : 스택의 개념과 구조

- 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.

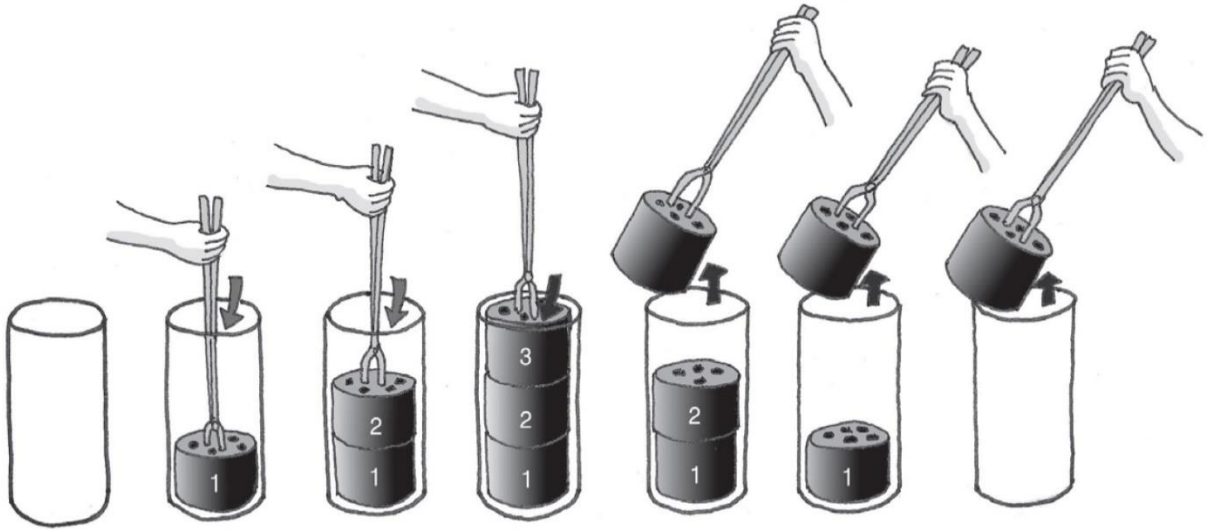


그림 5-3 스택의 LIFO 구조 예 : 연탄 아궁이

스택의 이해 : 스택의 개념과 구조

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

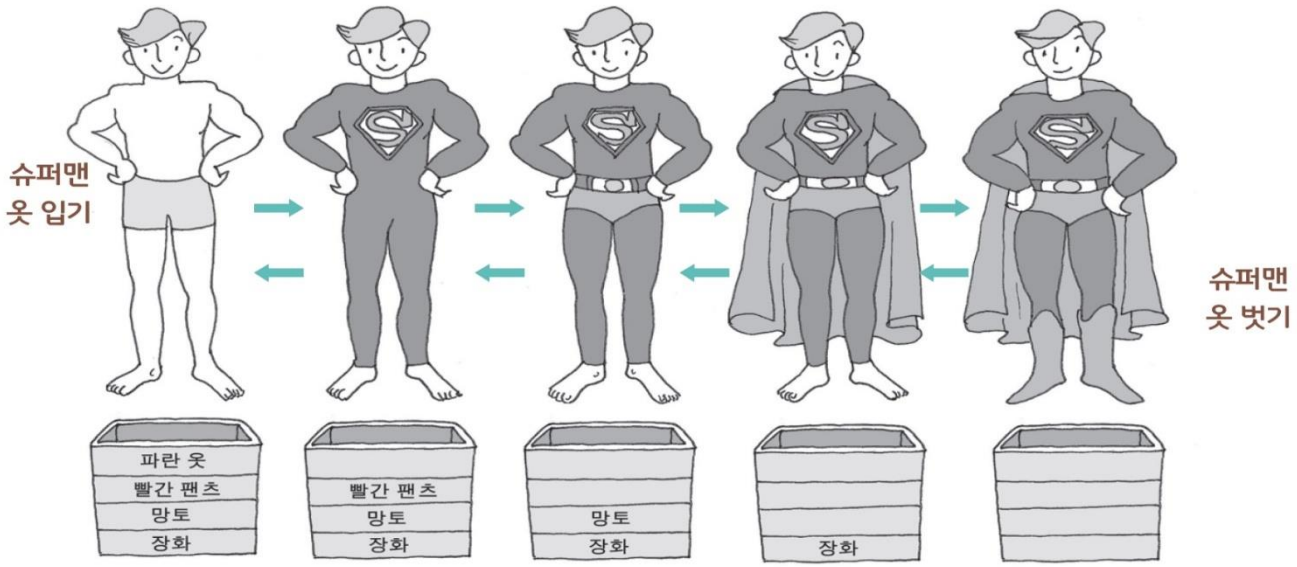


그림 5-4 스택의 LIFO 구조 예 : 슈퍼맨 옷 입기

스택의 이해 : 스택의 개념과 구조

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

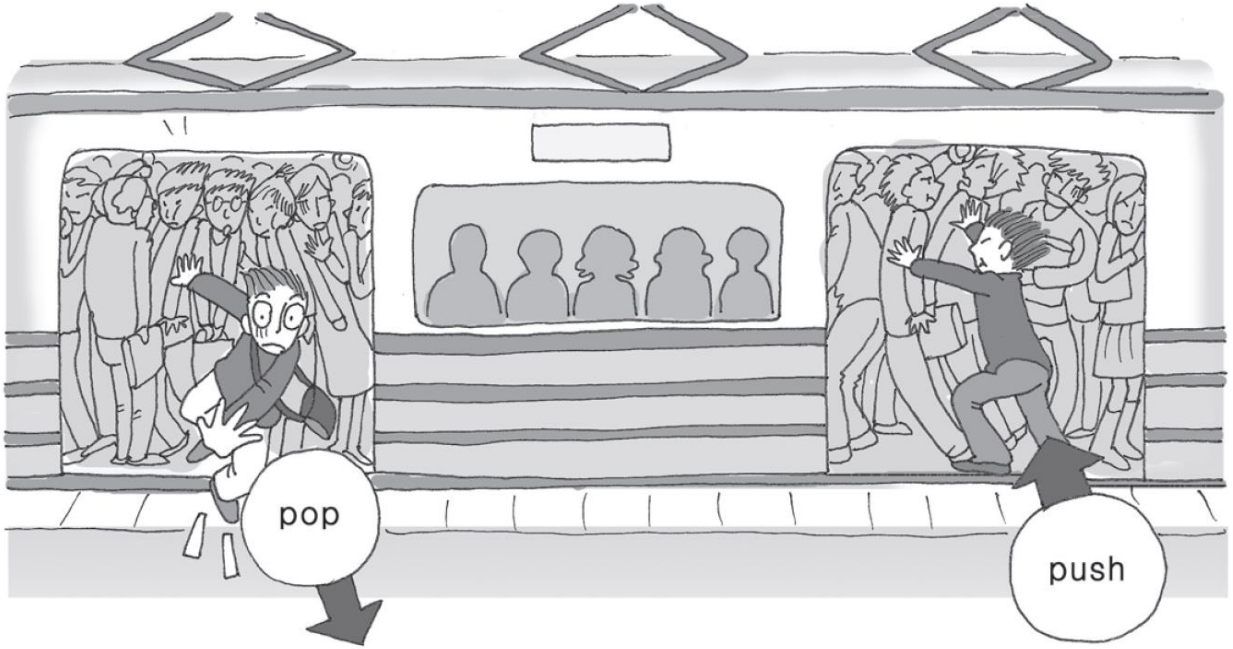


그림 5-5 만원 지하철에서의 pop과 push

스택의 이해 : 스택의 개념과 구조

- 스택에서의 원소 삽입/삭제 과정
  - [그림 5-6]공백 스택에 원소 A, B, C를 순서대로 삽입하고 한번 삭제하는 연산과정 동안의 스택 변화

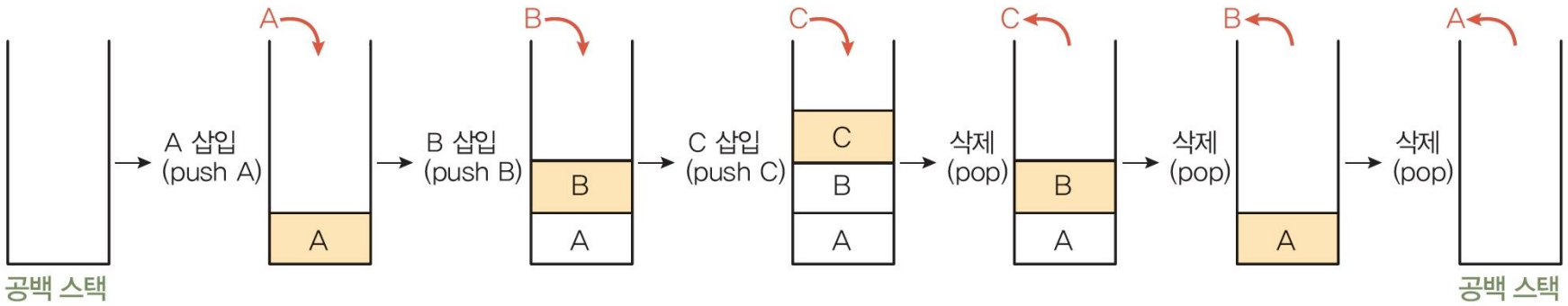


그림 5-6 스택의 데이터 삽입(push)과 삭제(pop) 과정

스택의 이해 : 스택의 추상 자료형

ADT 5-1 스택의 추상 자료형

```
ADT Stack
데이터 : 0개 이상의 원소를 가진 유한 순서 리스트
연산 :
   S ∈ Stack; item ∈ Element;

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

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

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

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

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

스택의 이해 : 스택의 추상 자료형

- 스택의 push() 알고리즘

① top ← 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) ← x;

- 오버플로우 상태가 아니라면 스택의 top이 가리키는 위치에 x 삽입

알고리즘 5-1 스택의 원소 삽입

```
push(S, x)
    ❶ top ← top + 1;
      if (top > stack_SIZE) then overflow;
      else
        ❷ S(top) ← x;
end push()
```

스택의 이해 : 스택의 추상 자료형

- 스택의 pop() 알고리즘

① return S(top);

- 공백 스택이 아니면 top에 있는 원소를 삭제하고 반환

② top ← top-1;

- 스택의 마지막 원소가 삭제되면 그 아래 원소, 즉 스택에 남아 있는 원소 중에서 가장 위에 있는 원소가 top이 되어야 하므로 top 위치를 하나 감소

알고리즘 5-2 스택의 원소 삭제

```
pop(S)
    if (isStackEmpty(S)) then underflow;
    else {
        ❶ return S(top);
        ❷ top ← top - 1;
    }
end pop()
```

스택의 구현: 순차 자료구조를 이용한 스택 구현

- 순차 자료구조를 이용한 스택의 구현
  - 순차 자료구조인 1차원 배열을 이용하여 구현
    - 스택의 크기 : 배열의 크기
    - 스택에 저장된 원소의 순서 : 배열 원소의 인덱스
      - 인덱스 0번 : 스택의 첫번째 원소
      - 인덱스 n-1번 : 스택의 n번째 원소
    - 변수 top : 스택에 저장된 마지막 원소에 대한 인덱스 저장
      - 공백 상태 : top = -1 (초기값)
      - 포화 상태 : top = n-1

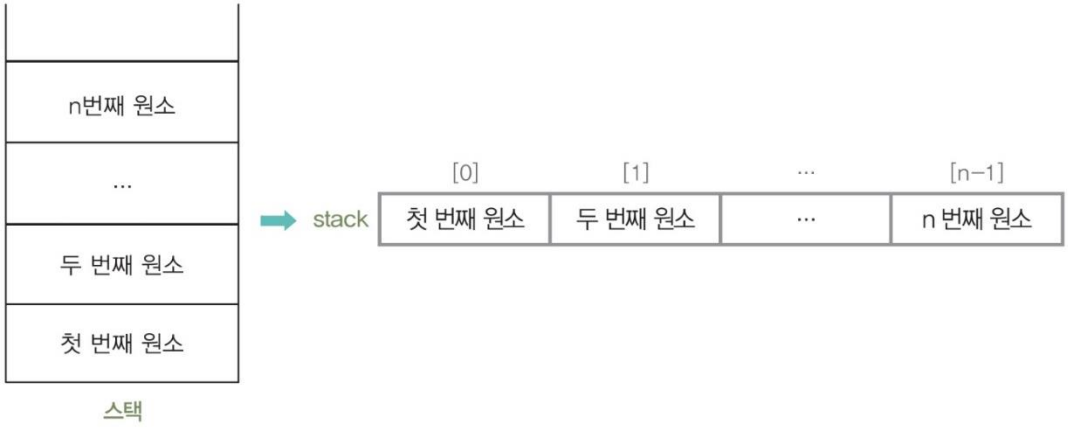
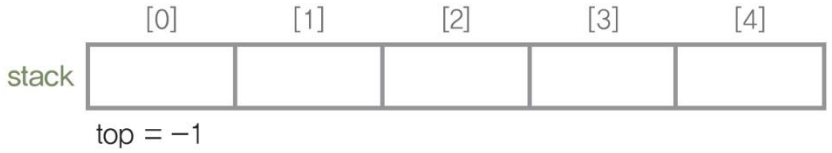


그림 5-7 1차원 배열을 이용한 순차 스택

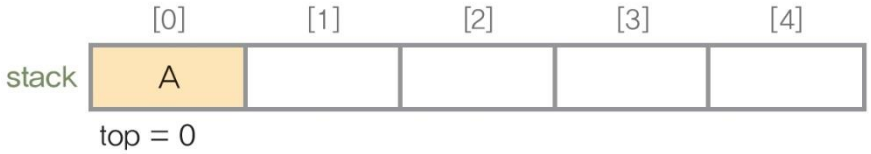
스택의 구현: 순차 자료구조를 이용한 스택 구현

- 크기가 5인 스택을 생성하여 원소 A, B, C를 순서대로 삽입한 후에 원소 하나를 삭제하는 과정

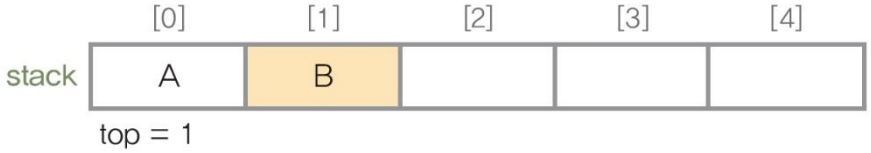
1 공백 스택 생성 : createStack(stack, 5);



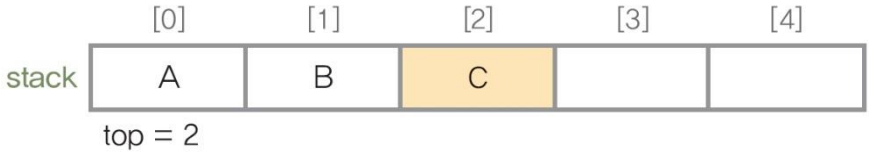
2 원소 A 삽입 : push(stack, A);



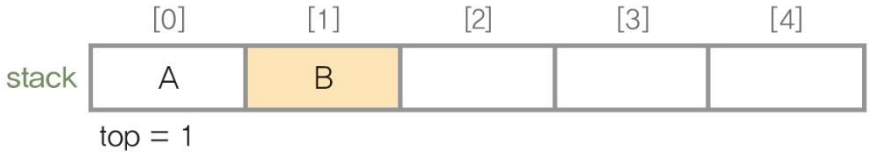
3 원소 B 삽입 : push(stack, B);



4 원소 C 삽입 : push(stack, C);



5 원소 삭제 : pop(stack);



스택의 구현: 순차 자료구조를 이용한 스택 구현

- 순차 자료구조를 이용해 순차 스택 구현하기 :

- 실행 결과

```
** 순차 스택 연산 **

STACK [ ]
STACK [ 1 ]
STACK [ 1 2 ]
STACK [ 1 2 3 ]
STACK [ 1 2 3 ]  peek => 3
STACK [ 1 2 ]    pop  => 3
STACK [ 1 ]      pop  => 2
STACK [ ]        pop  => 1
```

스택의 구현: 순차 자료구조를 이용한 스택 구현

- 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 가짐

스택의 구현:연결 자료구조를 이용한 스택의 구현

- 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

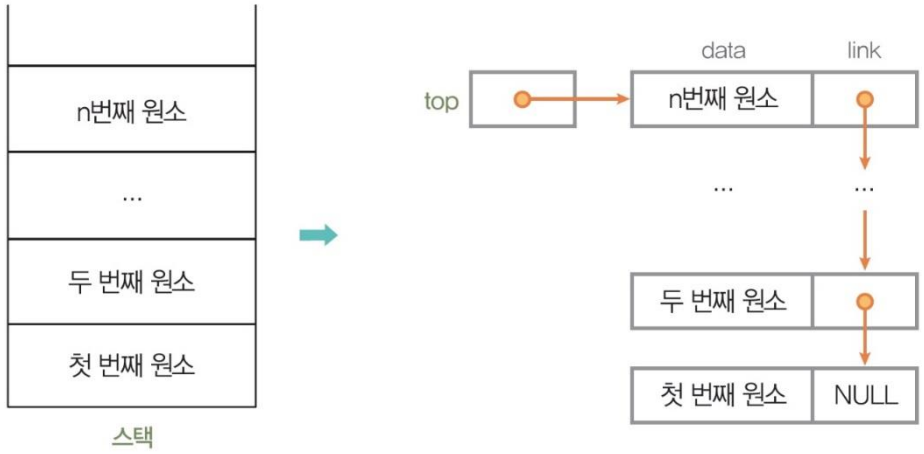


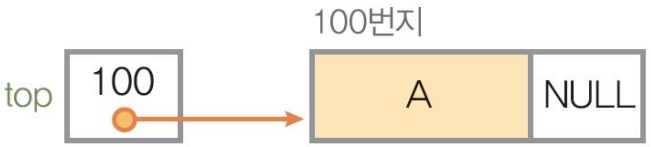
그림 5-8 단순 연결 리스트를 이용한 연결 스택

스택의 구현:연결 자료구조를 이용한 스택의 구현

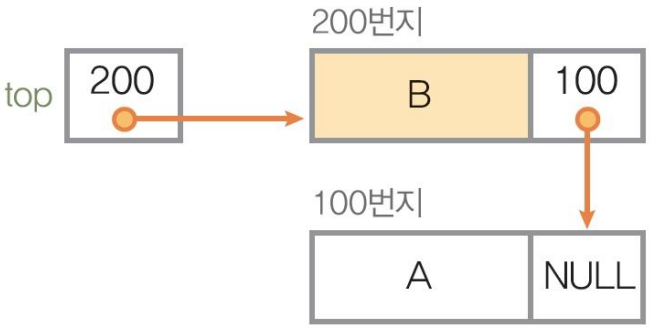
- 원소 A, B, C를 순서대로 삽입하면서 스택을 생성한 후에 원소 하나를 삭제하는 과정

1 공백 스택 생성 : createStack(stack);

2 원소 A 삽입 : push(stack, A);

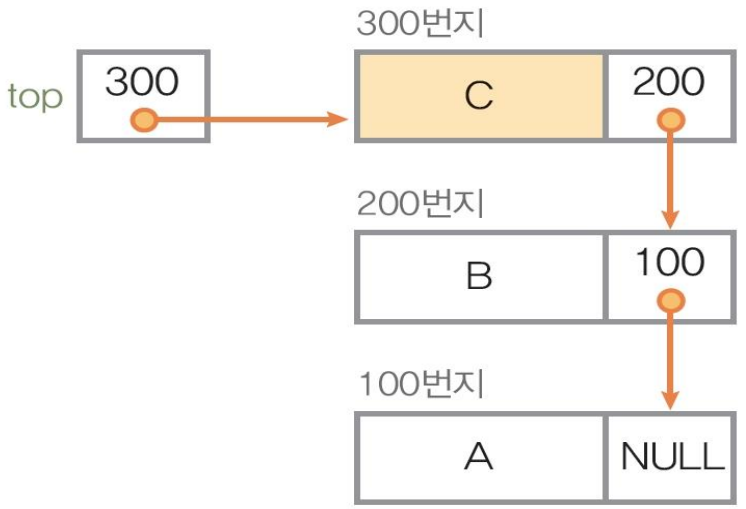


3 원소 B 삽입 : push(stack, B);

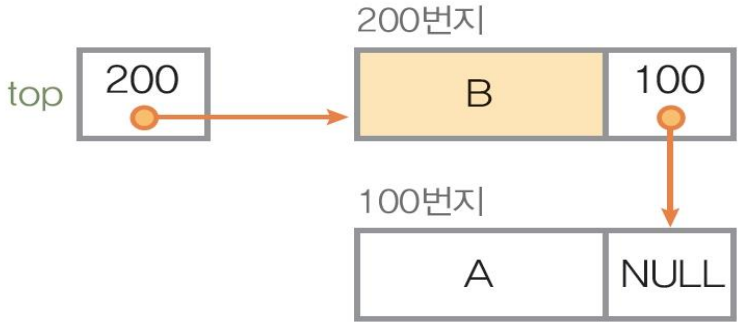


스택의 구현:연결 자료구조를 이용한 스택의 구현

4 원소 C 삽입 : push(stack, C);



5 원소 삭제 : pop(stack);



스택의 구현:연결 자료구조를 이용한 스택의 구현

- 연결 자료구조를 이용해 연결 스택 구현하기 :

- 실행 결과

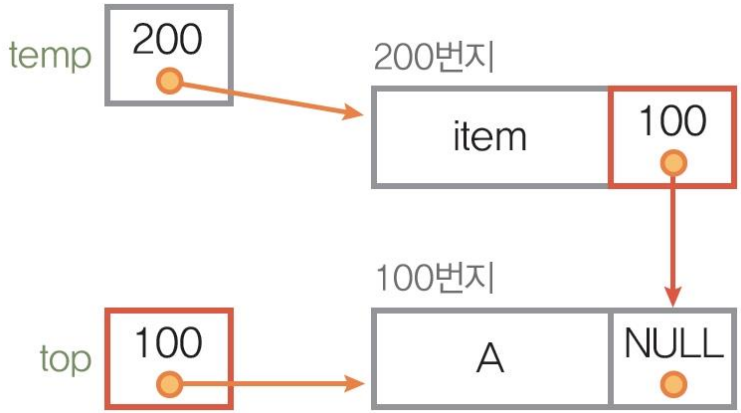
```
** 연결 스택 연산 **

STACK [ ]
STACK [ 1 ]
STACK [ 2 1 ]
STACK [ 3 2 1 ]
STACK [ 3 2 1 ]      peek => 3
STACK [ 2 1 ]        pop  => 3
STACK [ 1 ]          pop  => 2
STACK [ ]            pop  => 1
```

스택의 구현:연결 자료구조를 이용한 스택의 구현

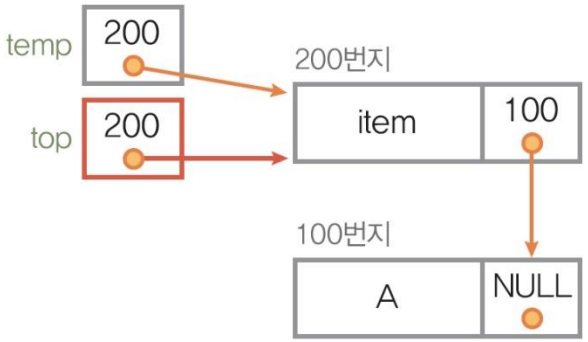
- 연결 스택에서 삽입 연산을 수행하는 과정
  - 원소 item을 저장할 노드에 대한 메모리 할당, 포인터 temp 설정
  - 삽입할 노드의 데이터 필드에 원소 item을 저장

  - 삽입할 노드의 링크 필드에 포인터 top의 값 저장하면, 새로 삽입한 노드가 현재 스택의 마지막 노드(현재 top이 가리키는 노드)로 연결

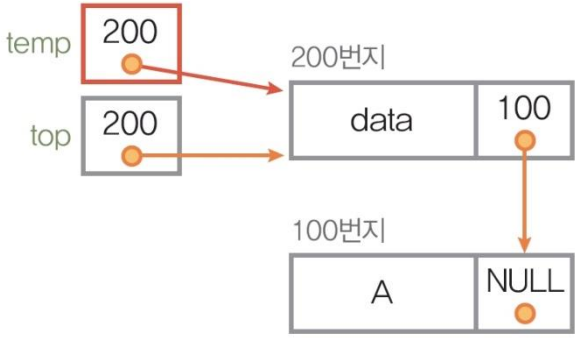


스택의 구현:연결 자료구조를 이용한 스택의 구현

- 포인터 temp의 값(삽입 노드의 주소)을 포인터 top에 설정, 새로 삽입한 노드가 스택의 top 노드가 되도록 조정

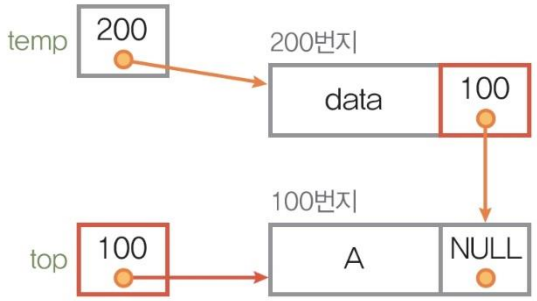


- 연결 스택에서 삭제하는 연산을 수행하는 과정
  - 포인터 temp를 top 노드에 설정하여 삭제할 노드를 가리킴



스택의 구현:연결 자료구조를 이용한 스택의 구현

- 스택의 마지막 노드의 데이터 필드값을 변수 item에 저장
- 포인터 top의 위치를 현재 마지막 노드의 아래 노드로 이동



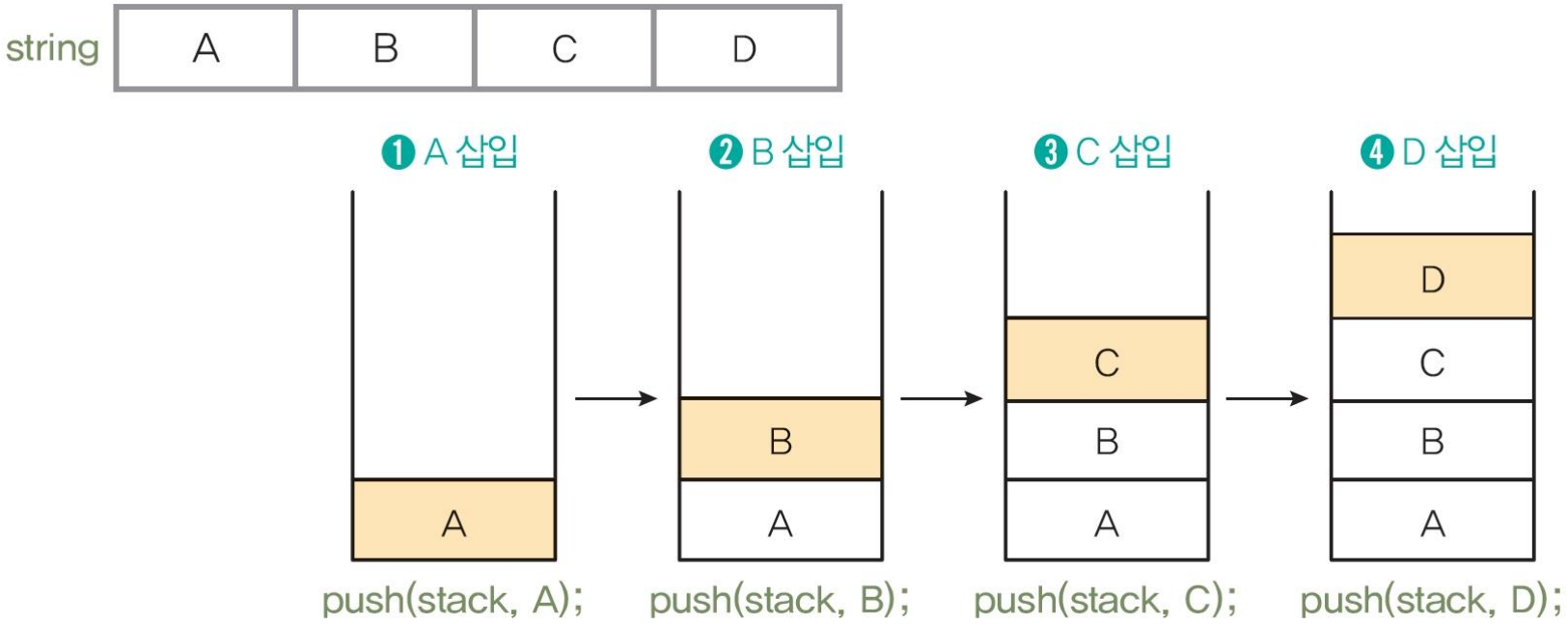
- 포인터 temp가 가리키는 노드를 메모리 해제
- 변수 item의 값, 즉 스택의 top이었던 노드의 데이터를 반환
- 연결 스택에 노드가 있는 동안, top 노드부터 링크 필드를 따라

아래 노드로 이동하면서 데이터 필드값을 출력. top 노드부터 출력하므로 출력 화면에서 왼쪽이 스택의 마지막 원소인 top이 됨

스택의 응용 : 스택을 이용한 역순 문자열

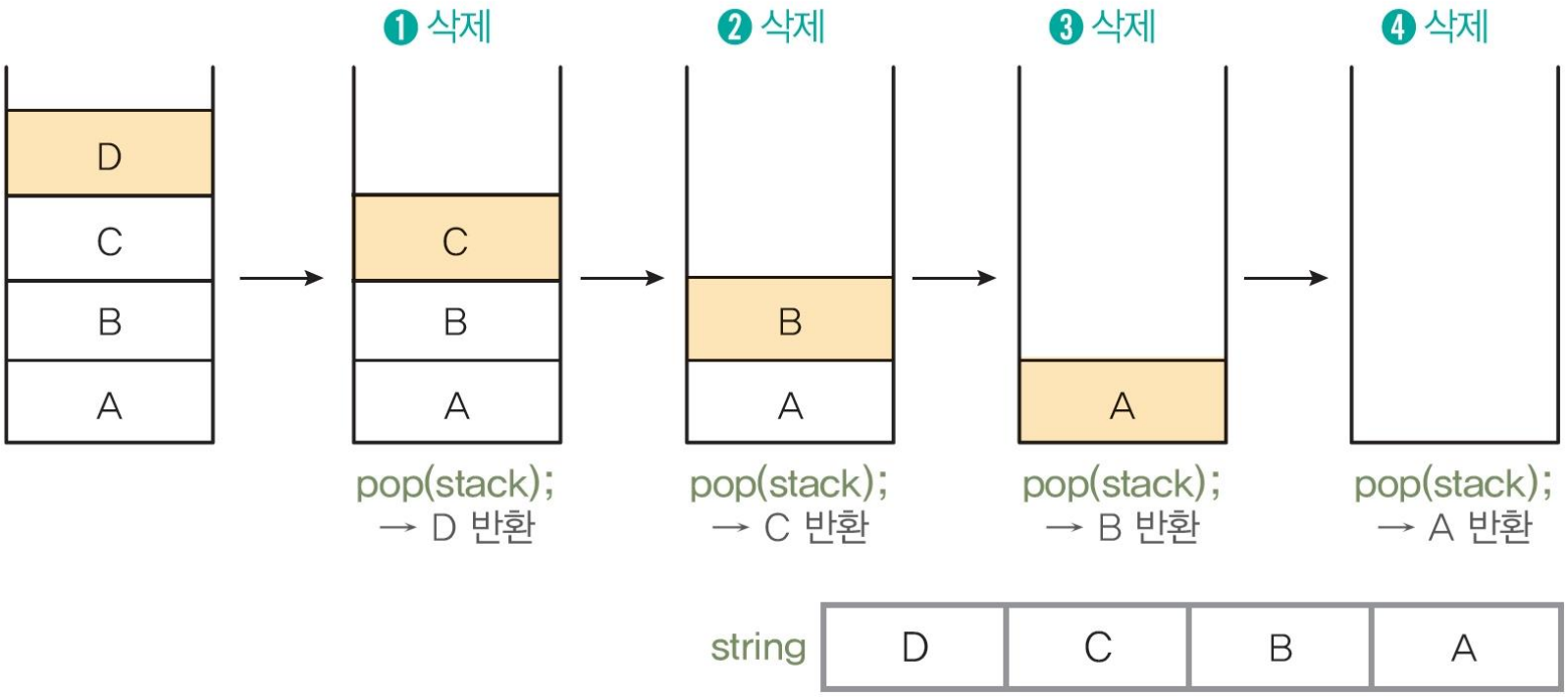
- 역순 문자열 만들기
  - 스택의 후입선출(LIFO) 성질을 이용

① 문자열을 순서대로 스택에 삽입



스택의 응용 : 스택을 이용한 역순 문자열

② 스택에서 삭제하여 문자열을 만들기



스택의 응용 : 시스템 스택

- 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 됨

스택의 응용 : 시스템 스택

- 함수 호출과 복귀에 따른 전체 프로그램의 수행 순서

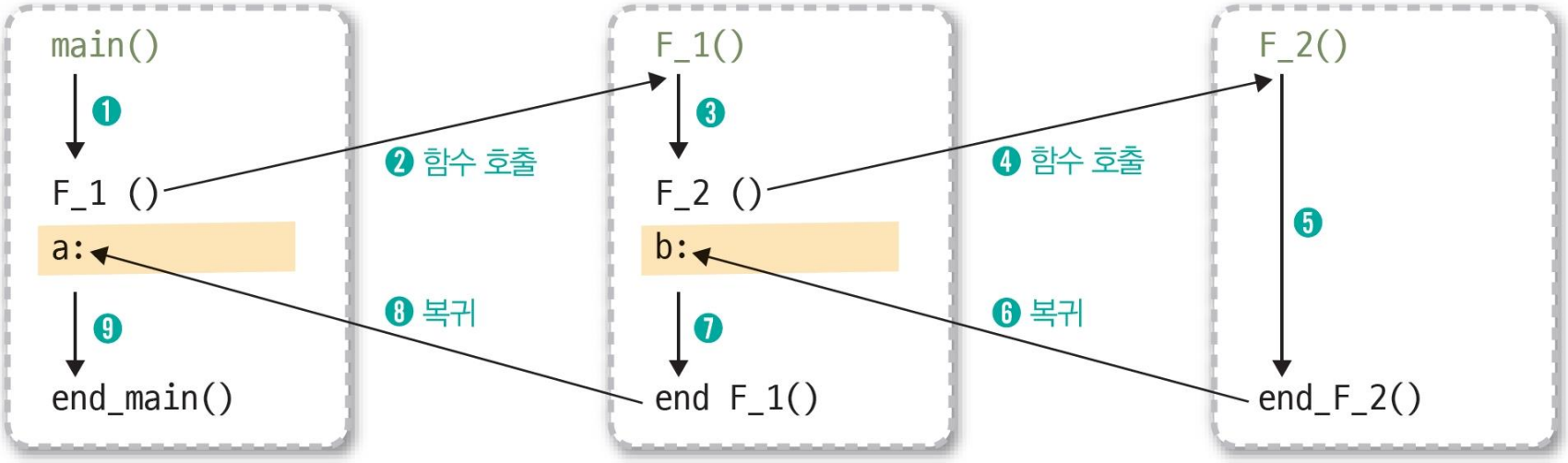
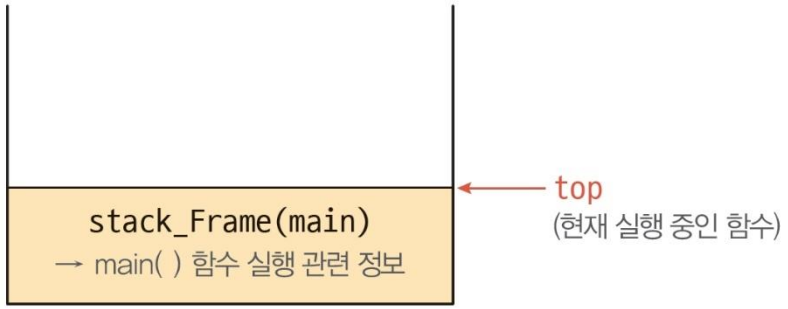


그림 5-9 함수 호출과 복귀에 따른 전체 프로그램 수행 순서

스택의 응용 : 시스템 스택

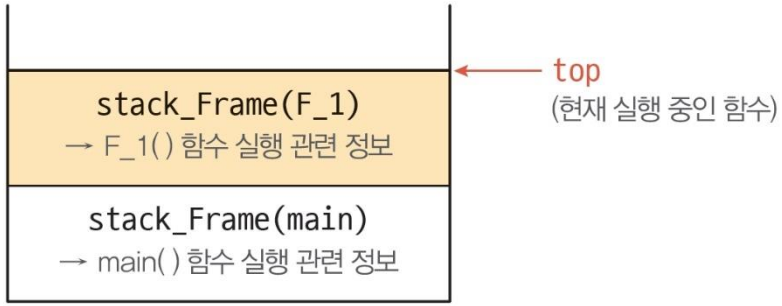
❶ main( )함수 실행 시작 : 시작하면 main( )함수가 호출되어 실행, main( ) 함수 시작과 관련된 정보를 스택 프레임에 저장, 시스템 스택에 삽입

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



❷ F_1( ) 함수 호출 : main() 함수 실행 중 F_1() 함수 호출을 만나면 함수 호출과 복귀에 필요한 정보를 스택 프레임에 저장, 시스템 스택에 삽입, 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));
```



스택의 응용 : 시스템 스택

❸ 호출된 함수 F_1() 함수 실행

❹ F_2( ) 함수 호출 : 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

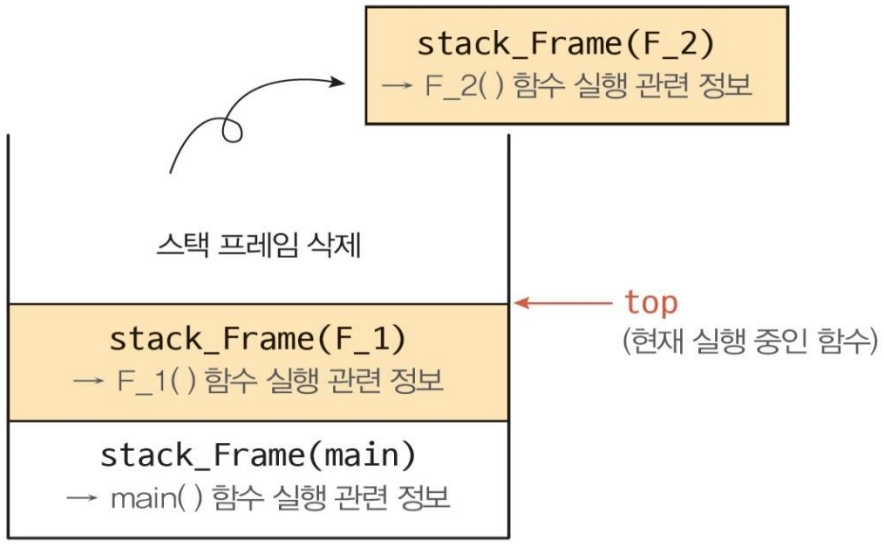
```
push(System_stack, stack_Frame(F_2));
```

❺ 호출된 함수 F_2() 함수 실행

스택의 응용 : 시스템 스택

❻ F_2( ) 함수 실행 종료, F_1( ) 함수로 복귀 : F_2( ) 함수 실행이 끝나면 F_2( ) 함수를 호출했던 이전 위치로 복귀하여 이전 함수 F_1( )의 작업을 계속 해야 함. 시스템 스택의 top에 있는 스택 프레임을 pop하여 정보를 확인하고 복귀 및 작업 전환 실행함

```
pop(System_stack);
```

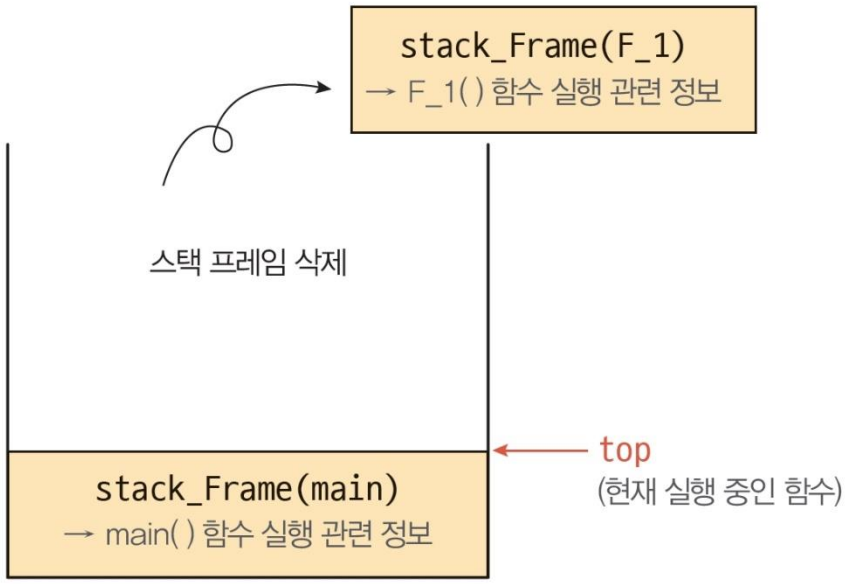


❼ F_1( ) 함수로 복귀하여 F_1( ) 함수의 나머지 부분 실행

스택의 응용 : 시스템 스택

❽ F_1( ) 함수 실행 종료, main( ) 함수로 복귀 : 스택의 top에 있는 스택 프레임을 pop하여 정보를 확인하고 복귀 및 작업 전환을 실행

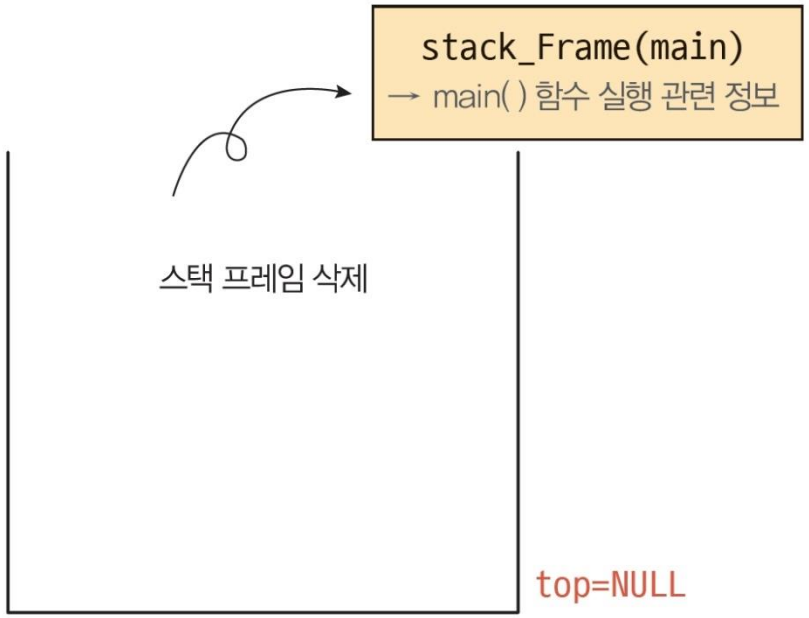
```
pop(System_stack);
```



스택의 응용 : 시스템 스택

❾ main( ) 함수 실행 완료(전체 프로그램 실행 완료) : 정상적인 함수 호출과 복귀가 모두 완료되었으므로 시스템 스택은 공백이 됨

```
pop(System_stack);
```



스택의 응용 : 스택을 이용한 수식의 괄호 검사

- 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 pop하여 마지막에 저장한 괄호와 같은 종류인지를 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.

스택의 응용 : 스택을 이용한 수식의 괄호 검사

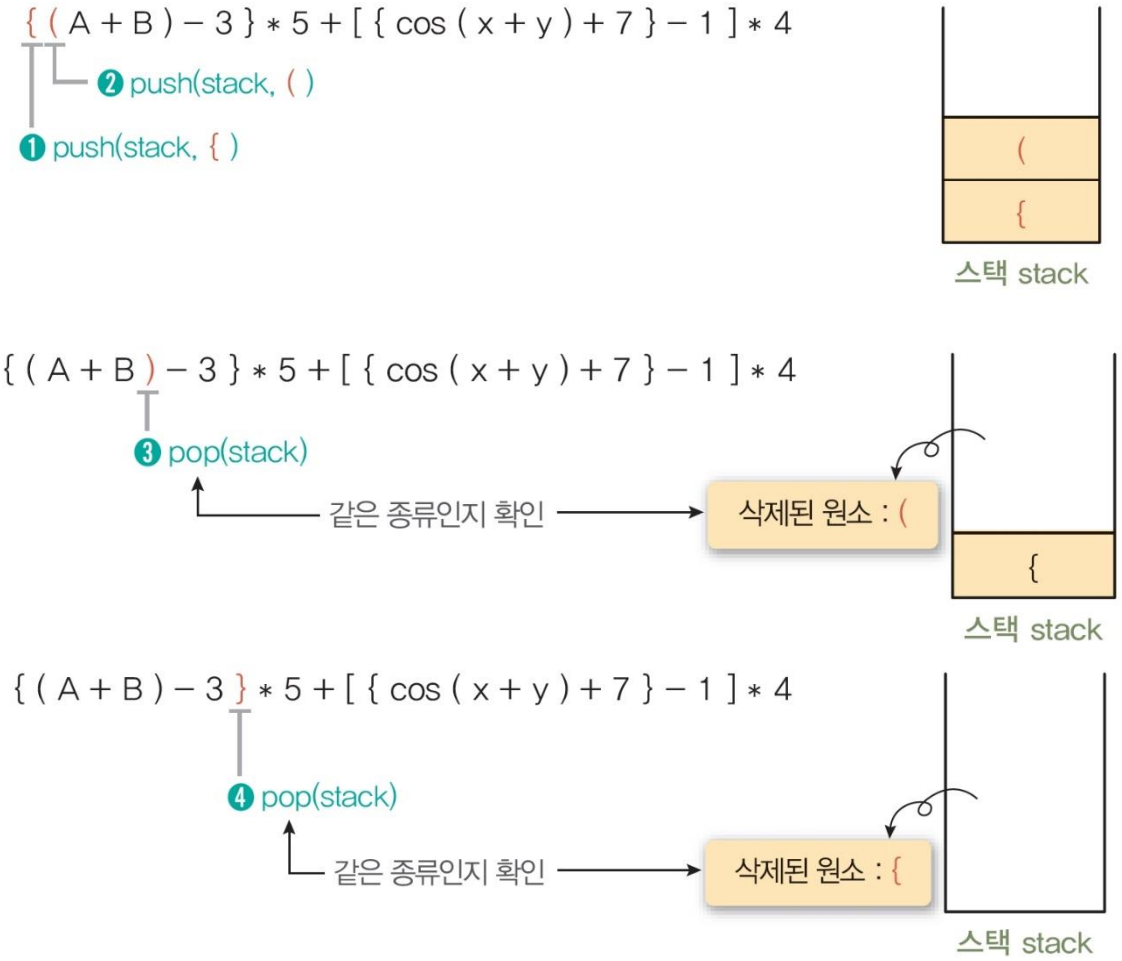
알고리즘 5-3 수식의 괄호 쌍 검사

```
testPair()
   exp ← Expression;
   Stack ← null;
   while (true) do {
      symbol ← getSymbol(exp);
      case {
         symbol == "(" or "[" or "{" :
                    push(Stack, symbol);
         symbol == ")" :
                    open_pair ← pop(Stack);
                    if (open_pair ≠ "(") then return false;
         symbol == "]" :
                     open_pair ← pop(Stack);
                     if (open_pair ≠ "[") then return false;
         symbol == "}" :
                    open_pair ← pop(Stack);
                    if (open_pair ≠"{") then return false;
         symbol == null :
                    if (isStackEmpty(Stack)) then return true;
                    else return false;
         else :
      }
   }
end testPair()
```

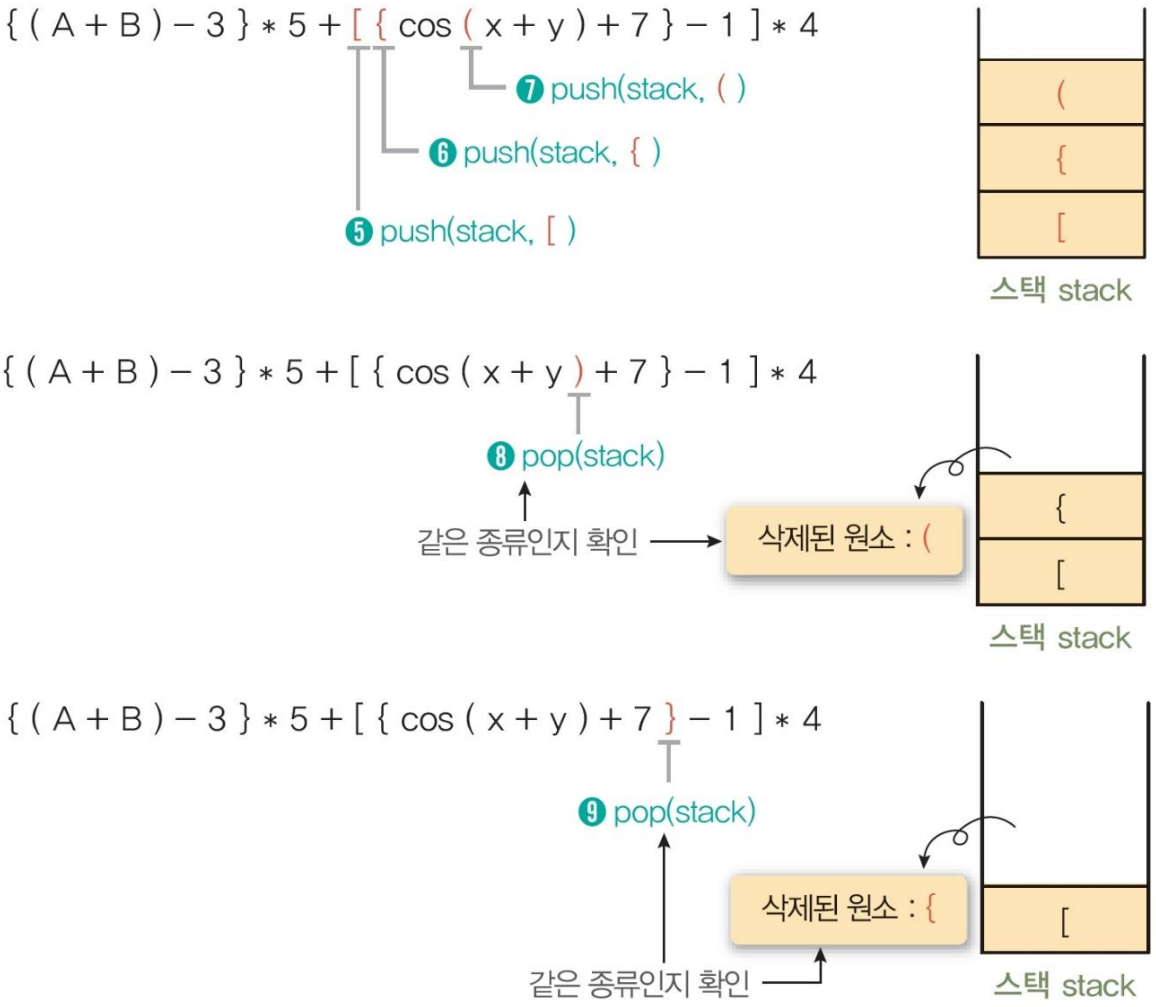
스택의 응용 : 스택을 이용한 수식의 괄호 검사

- 예

{(A+B)−3}*5+[{cos(x+y)+7}−1]*4



스택의 응용 : 스택을 이용한 수식의 괄호 검사



스택의 응용 : 스택을 이용한 수식의 괄호 검사

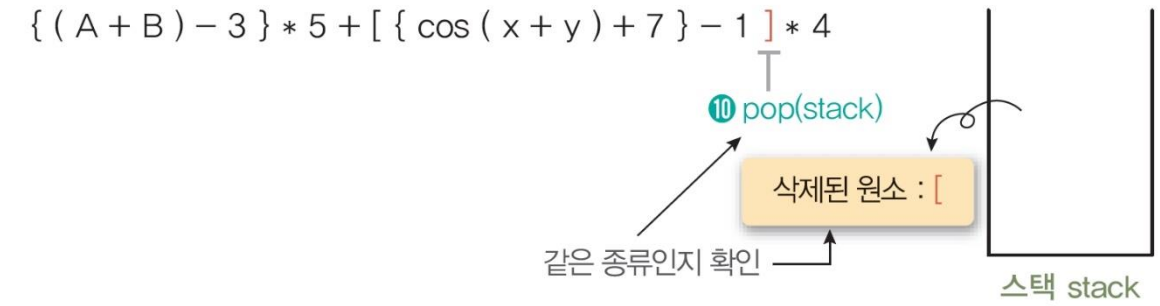
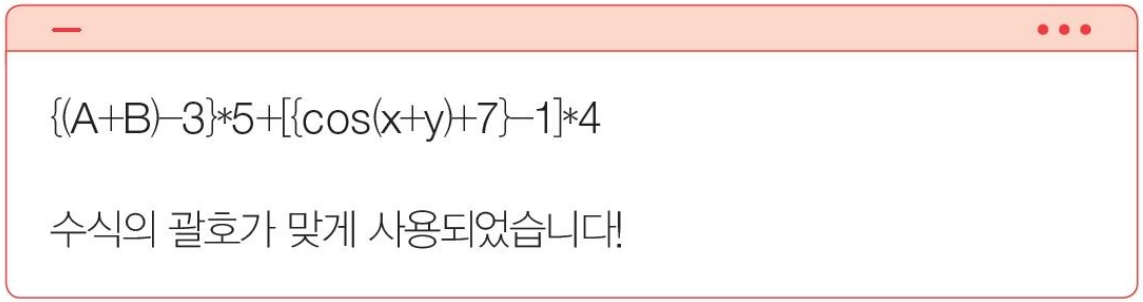


그림 5-10 수식 처리에 따른 스택 사용 과정

스택의 응용 : 스택을 이용한 수식의 괄호 검사

- 스택을 이용해 수식의 괄호쌍 검사하기 :

- 실행 결과



스택의 응용:스택을 이용한 수식의 후위 표기법 변환

- 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+

스택의 응용:스택을 이용한 수식의 후위 표기법 변환

- 중위표기식의 전위표기식 변환 방법
  - ❶ 수식의 각 연산자에 대해 우선순위에 따라 괄호를 사용해 다시 표현한다.
  - ❷ 각 연산자를 그에 대응하는 왼쪽 괄호의 앞으로 이동시킨다.

    -(*(A B) /(C D))
  - ❸ 괄호를 제거한다.

스택의 응용:스택을 이용한 수식의 후위 표기법 변환

- 중위표기식의 후위표기식 변환 방법
  - ❶ 수식의 각 연산자에 대해 우선순위에 따라 괄호를 사용해 다시 표현한다.
  - ❷ 각 연산자를 그에 대응하는 오른쪽 괄호의 뒤로 이동시킨다.
  - ❸ 괄호를 제거한다.

스택의 응용:스택을 이용한 수식의 후위 표기법 변환

- 컴퓨터 내부에서 스택을 사용해 중위 표기법을 후위 표기법으로 바꾸는 방법

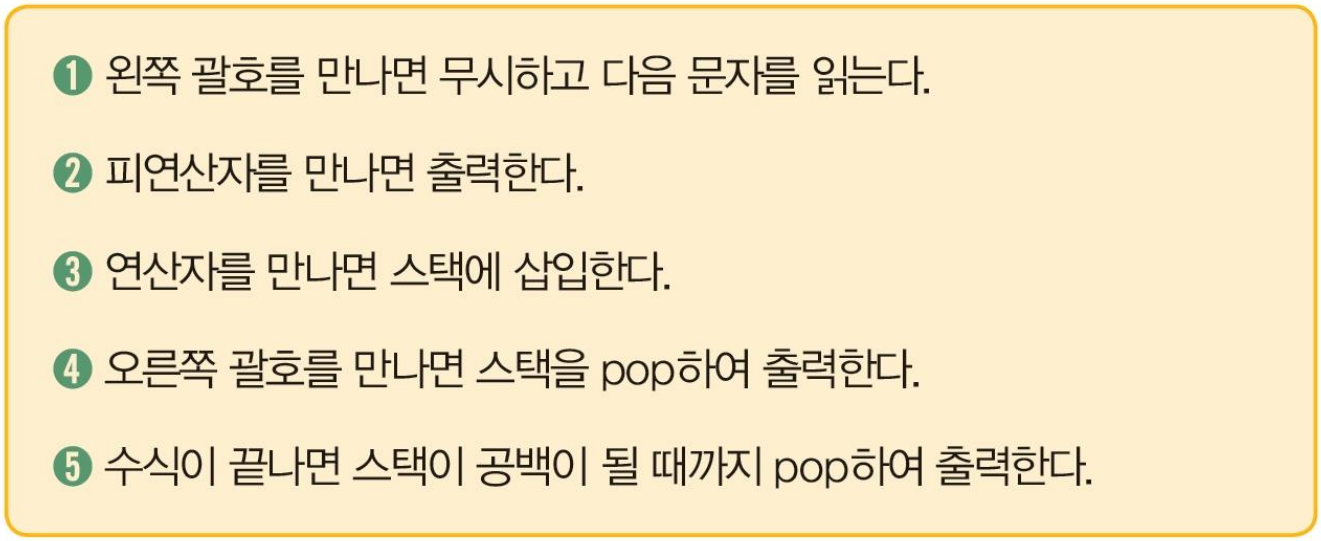


그림 5-13 컴퓨터 내부에서 중위 표기법을 후위 표기법으로 바꾸는 방법

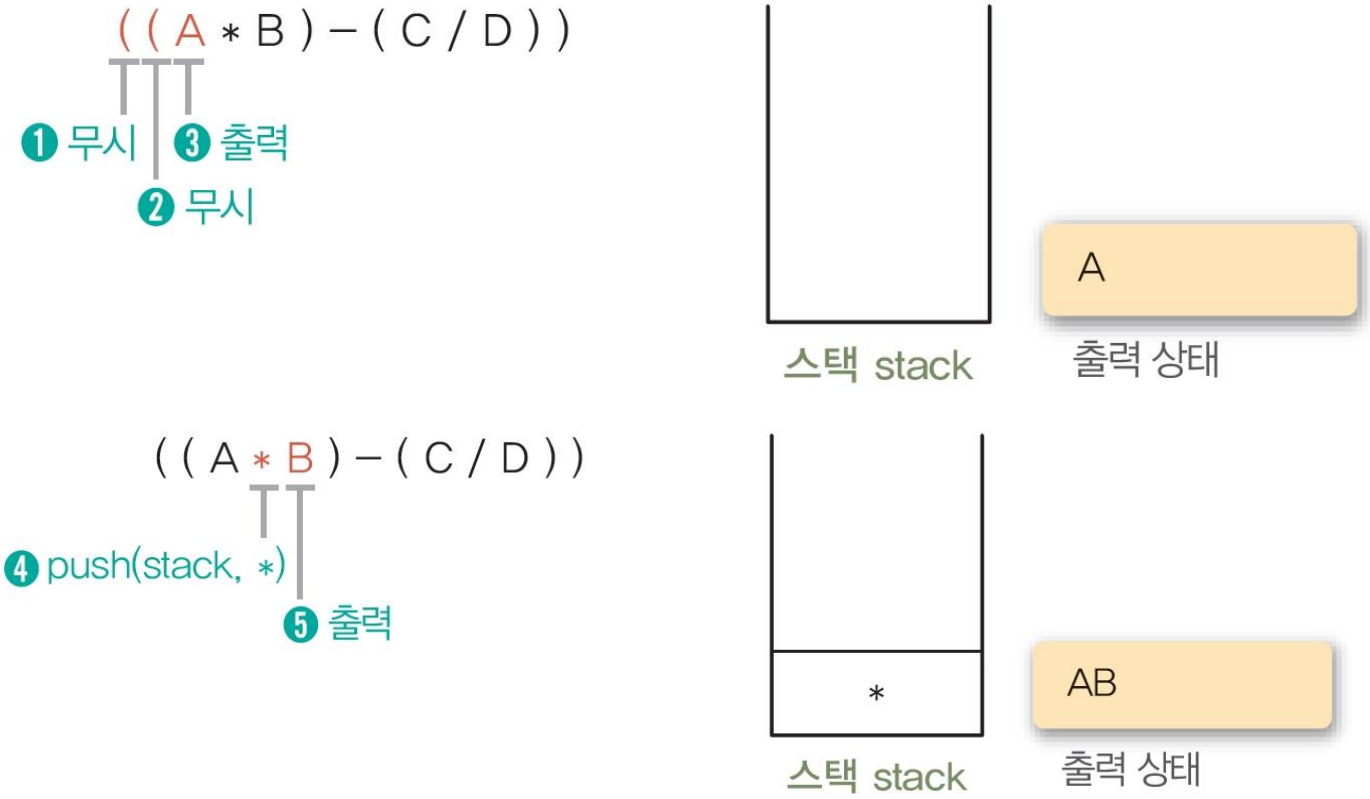
스택의 응용:스택을 이용한 수식의 후위 표기법 변환

알고리즘 5-4 후위 표기법으로 변환

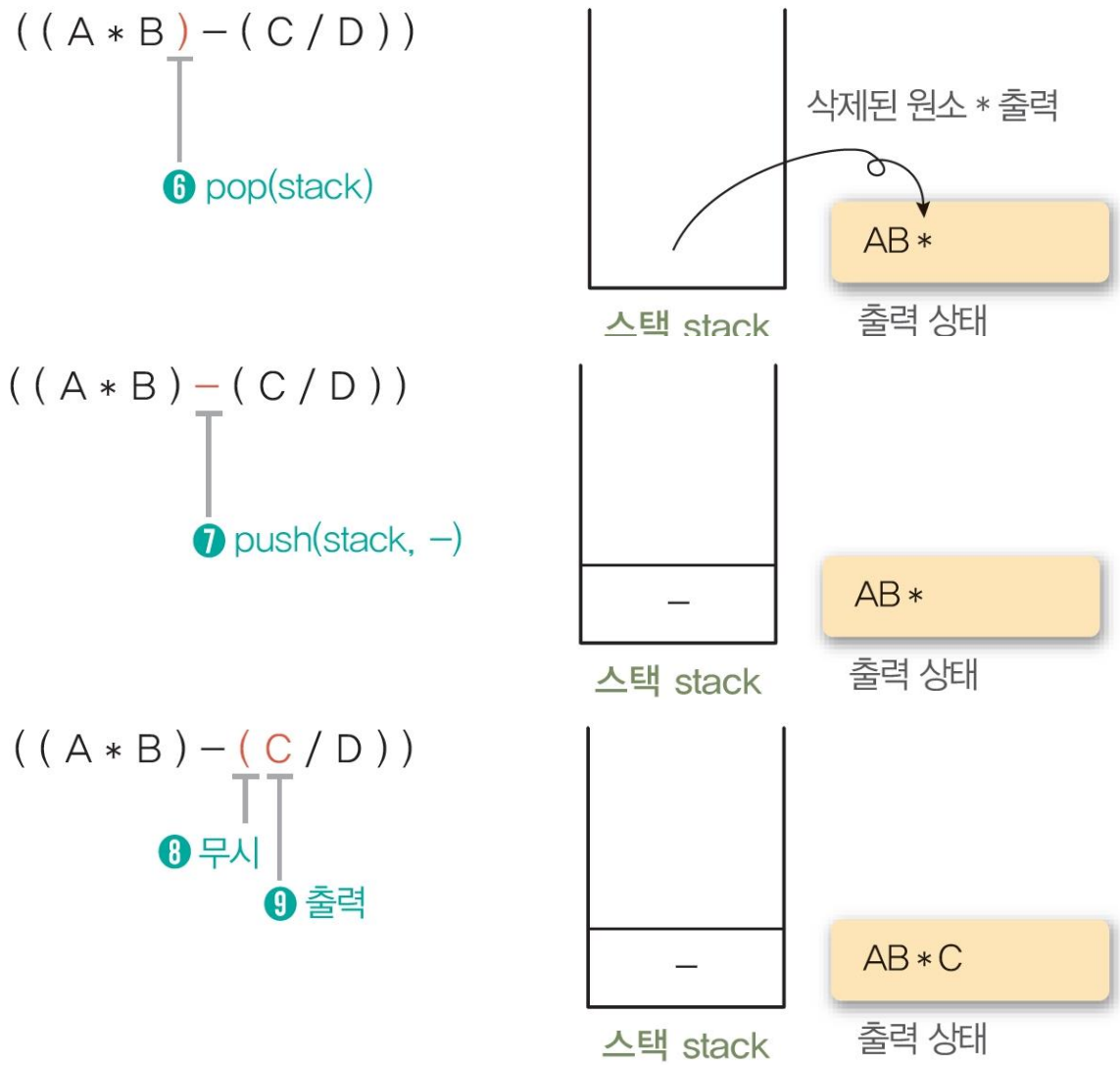
```
infix_to_postfix(exp)
   while (true) do {
      symbol ← getSymbol(exp);
      case {
         symbol == operand : // 피연산자 처리: 출력
                    print(symbol);
         symbol == operator :   // 연산자 처리: 스택에 push
                    push(stack, symbol);
         symbol == ")" :      // 오른쪽 괄호 처리: 스택을 pop하여 출력
                    print(pop(stack));
         symbol == null :    // 수식의 끝 처리:
                    while (top > -1) do // 스택이 공백이 될 때까지 pop하여 출력
                       print(pop(stack));
         else :
      }
   }
end infix_to_postfix()
```

스택의 응용:스택을 이용한 수식의 후위 표기법 변환

- 스택을 이용하여 수식 A*B-C/D를 후위 표기법으로 변환



스택의 응용:스택을 이용한 수식의 후위 표기법 변환



스택의 응용:스택을 이용한 수식의 후위 표기법 변환

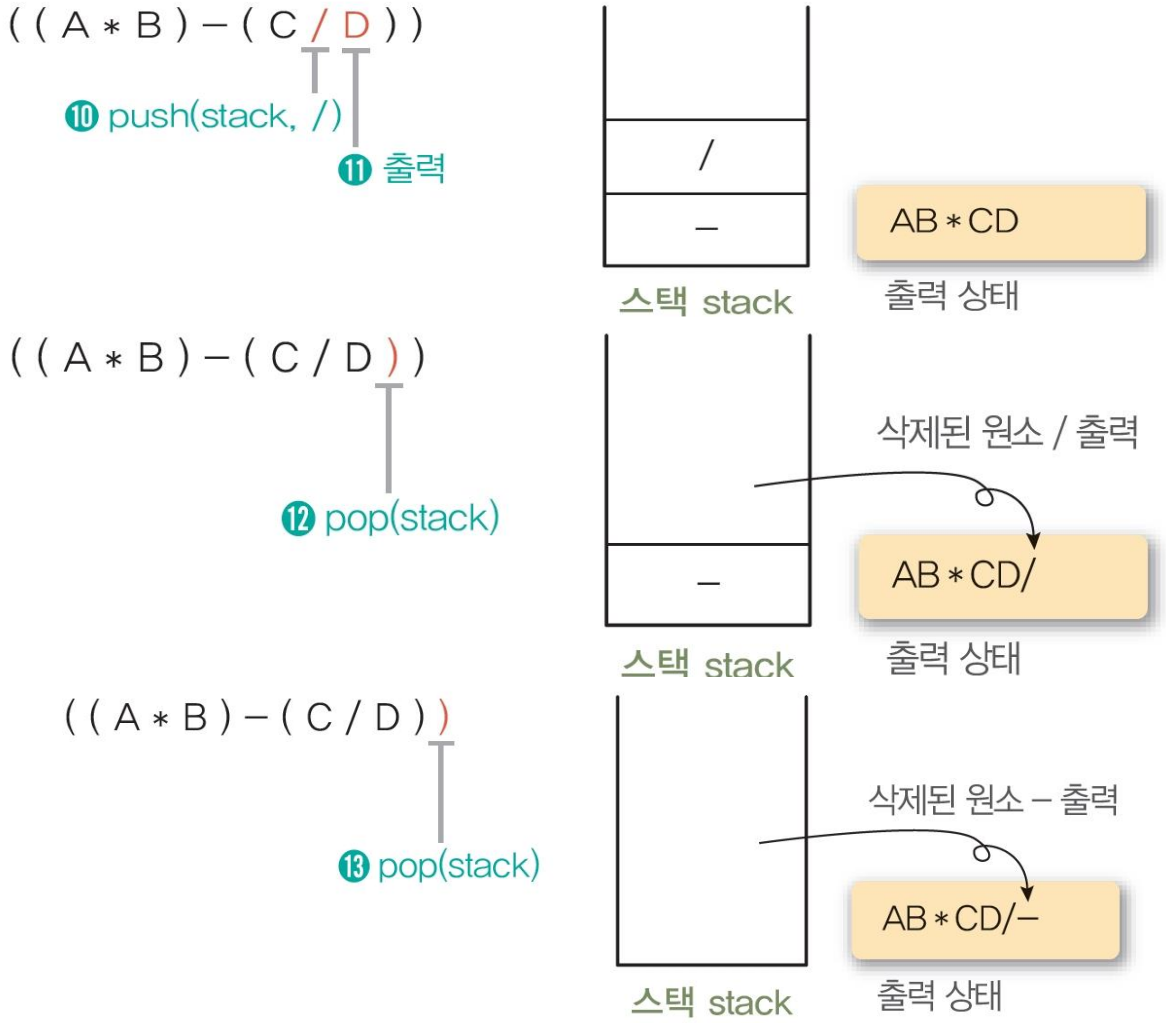


그림 5-14 스택을 사용해 수식 A*B-C/D를 후위 표기법으로 바꾸는 과정 예

스택의 응용:스택을 이용한 후위 표기법 수식의 연산

- 스택을 이용한 후위 표기법 수식의 연산
  - 스택을 사용해 후위 표기법 수식을 계산하는 방법

❶ 피연산자를 만나면 스택에 push한다.

❷ 연산자를 만나면 필요한 만큼의 피연산자를 스택에서 pop하여 연산하고, 연산 결과를 다시 스택에 push한다.

❸ 수식이 끝나면 마지막으로 스택을 pop하여 출력한다.

그림 5-15 스택을 사용해 후위 표기법 수식을 계산하는 방법

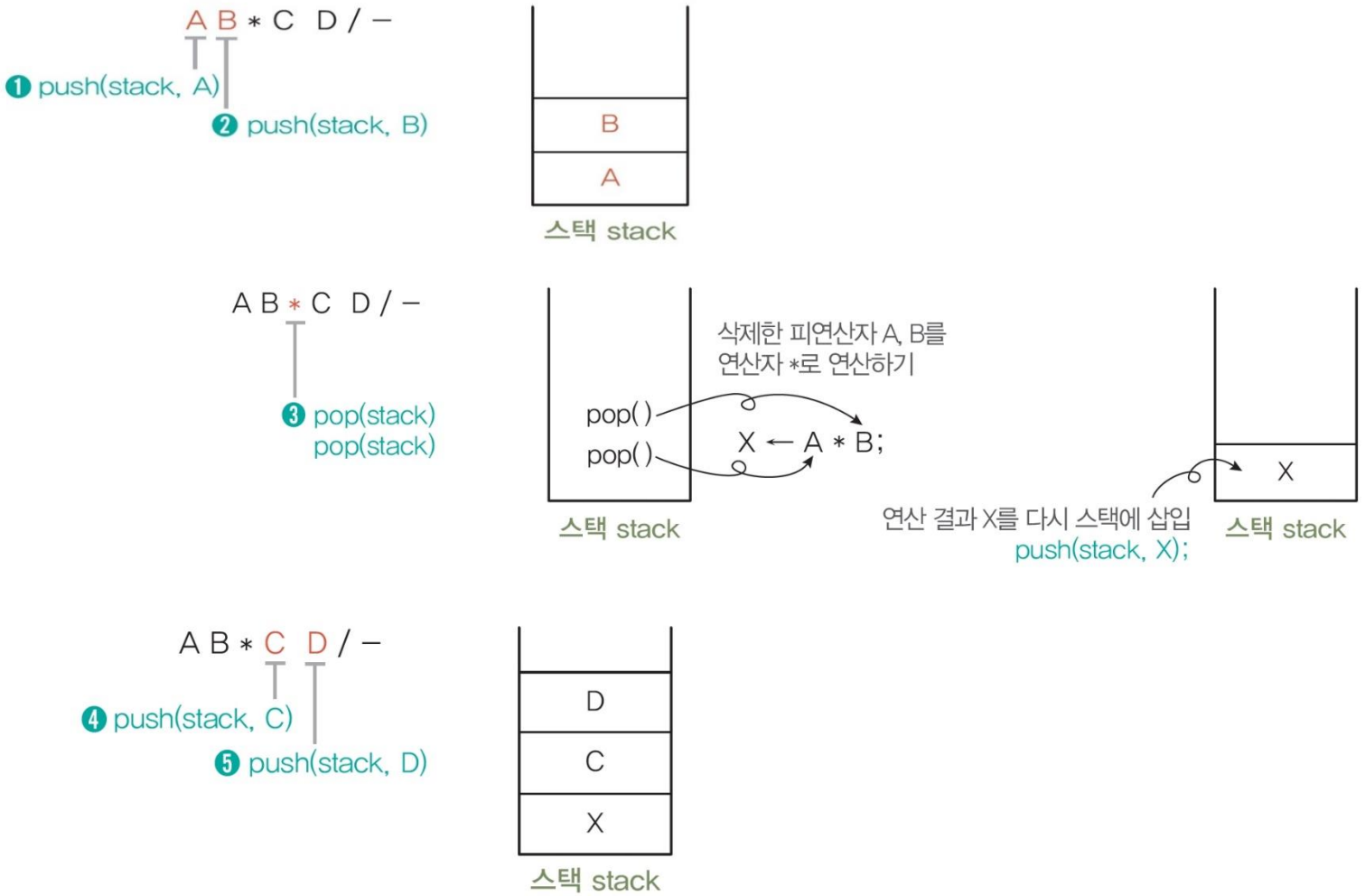
스택의 응용:스택을 이용한 후위 표기법 수식의 연산

알고리즘 5-5 후위 표기법으로 수식 연산

```
evalPostfix(exp)
    while (true) do {
        symbol ← getSymbol(exp);
        case {
            symbol == operand : // 피연산자 처리
                        push(Stack, symbol);
            symbol == operator :   // 연산자 처리
                        opr2 ← pop(Stack);
                        opr1 ← pop(Stack);
                        // 스택에서 꺼낸 피연산자들을 연산자로 연산
                        result ← opr1 op(symbol) opr2;
                        push(Stack, result);
            symbol == null :    // 후위 수식의 끝
                        print(pop(Stack));
        }
    }
end evalPostfix()
```

스택의 응용:스택을 이용한 후위 표기법 수식의 연산

- 위의 알고리즘으로 수식 AB*CD/-를 연산



스택의 응용:스택을 이용한 후위 표기법 수식의 연산

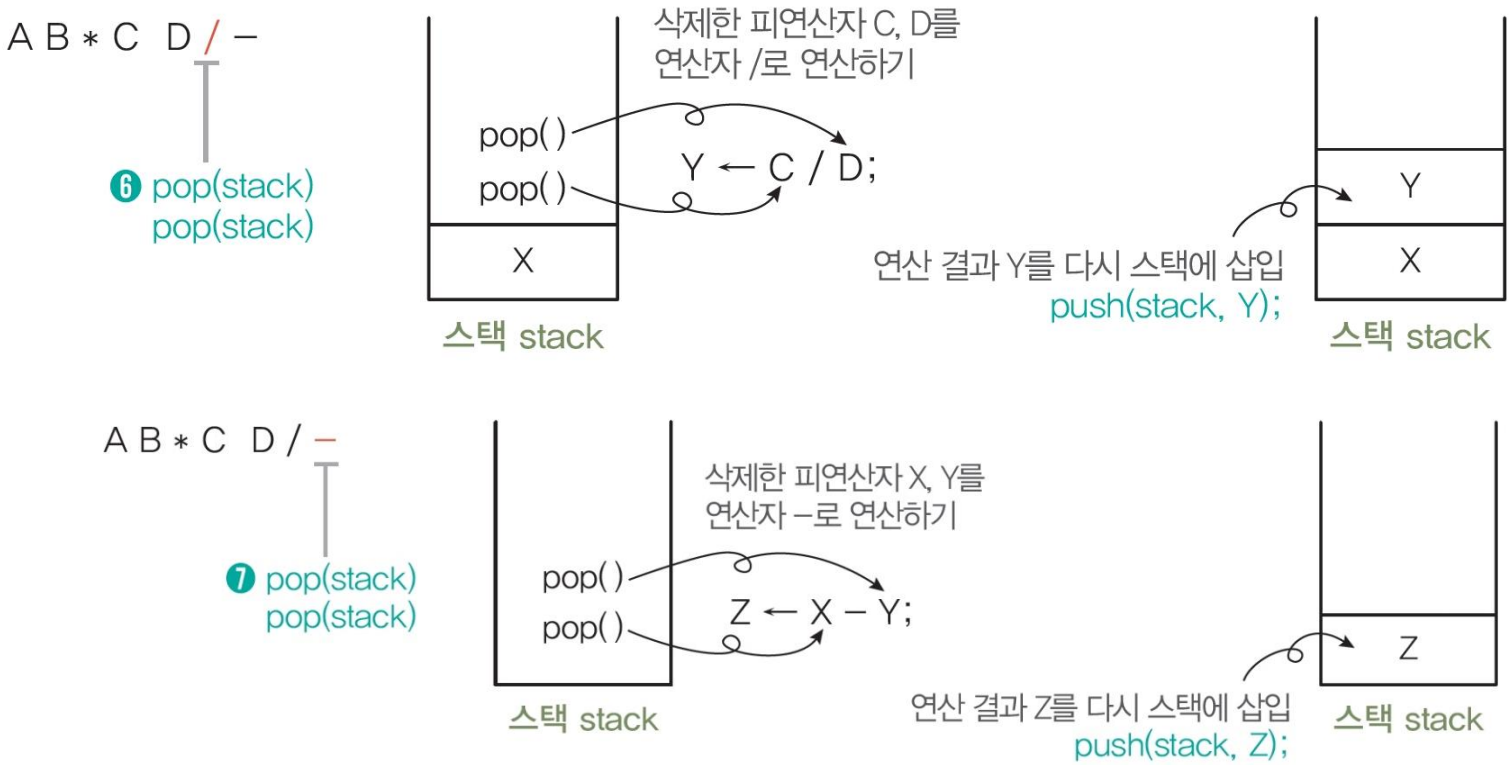


그림 5-16 스택을 사용한 후위 표기법 수식의 연산 과정

스택의 응용:스택을 이용한 후위 표기법 수식의 연산

- 수식을 후위 표기법으로 연산하기 :

- 실행 결과

```
후위 표기식 : 35*62/-

연산 결과 => 12
```

질문 및 정리