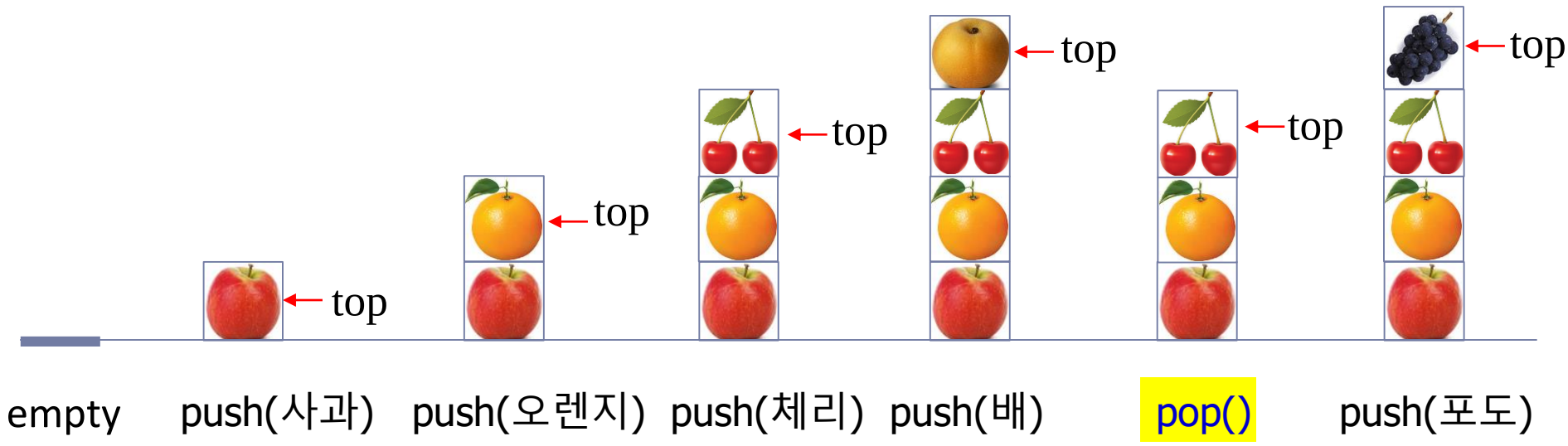
제3장 스택과 큐

스택, 큐, 계산기

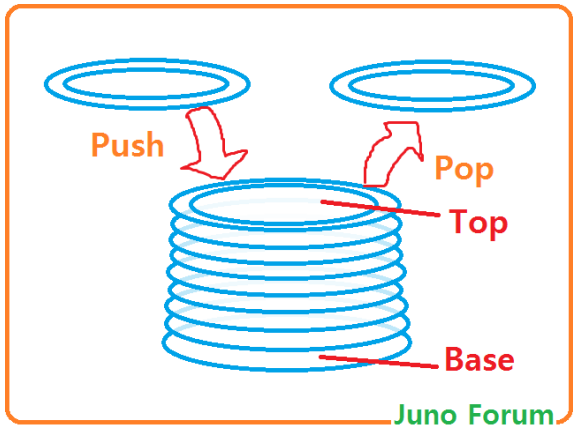
3.1 스택

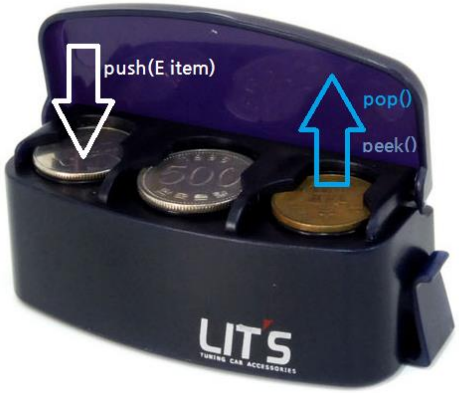
- 한 쪽 끝에서만 item(항목)을 삭제하거나 새로운 item을 저장하는 자료 구조
- 새 item을 저장하는 연산: push
- Top item을 삭제하는 연산: pop
- 후입 선출(Last-In First-Out, LIFO) 원칙하에 item의 삽입과 삭제 수행



스택의 응용

- 프로그램 실행 시 함수 구동을 위한 시스템 스택
- 미로 찾기
- 컴파일러의 괄호 짝 맞추기
- 회문(Palindrome) 검사하기
- 문서 편집기의 undo
- 후위표기법(Postfix Notation) 수식 계산하기
- 중위표기법(Infix Notation) 수식의 후위표기법 변환
- 트리의 방문(4장)
- 그래프의 깊이우선탐색(9장)



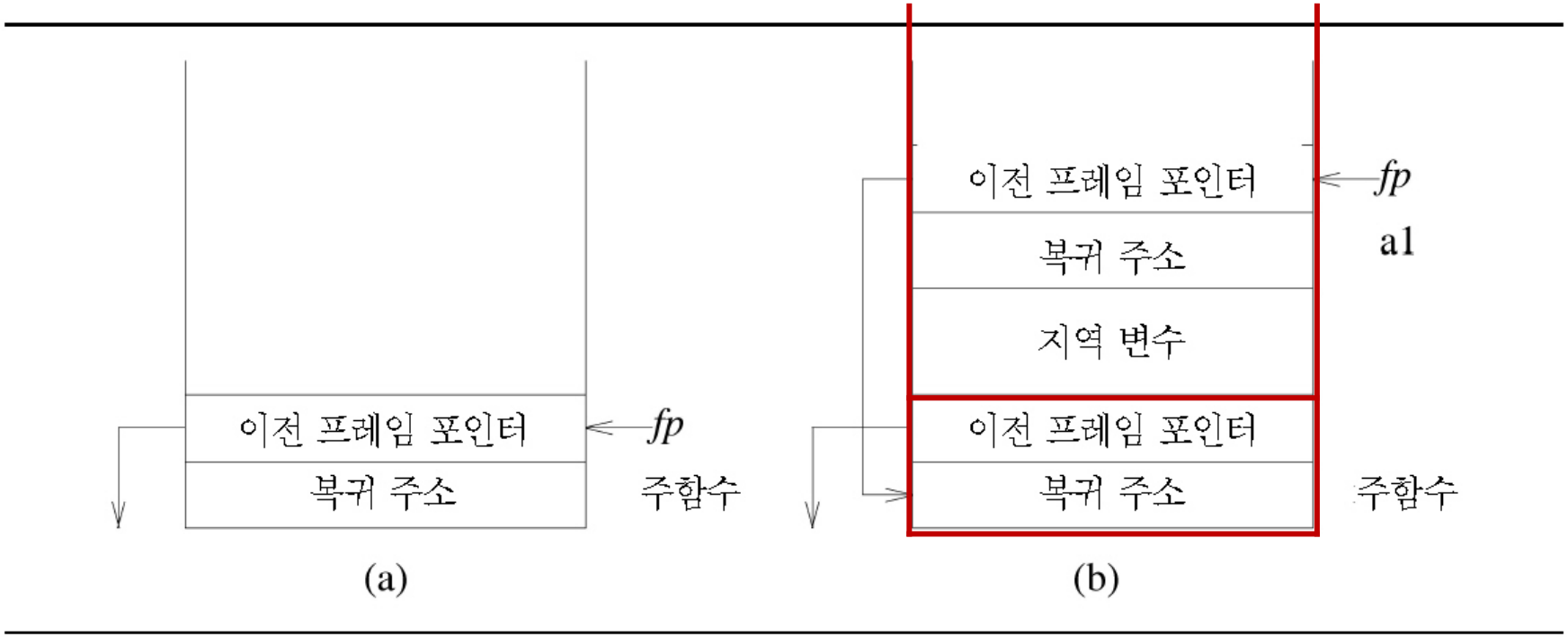


스택 응용 - 시스템 스택

- **시스템 스택**
  - 프로그램 실행시 함수 호출을 처리
  - 함수 호출시 활성 레코드 (activation record) 또는
    스택 프레임(stack frame) 구조 생성하여 시스템 스택의 top에 둠
    - 이전의 스택 프레임에 대한 포인터
    - 복귀 주소
    - 지역 변수
    - 매개 변수
  - 함수가 자기자신을 호출하는 순환 호출도 마찬가지로 처리
    →순환 호출시마다 새로운 스택 프레임 생성
    →최악의 경우 가용 메모리 전부 소모

스택 응용 - 시스템 스택

- 주 함수가 함수 a1을 호출하는 예



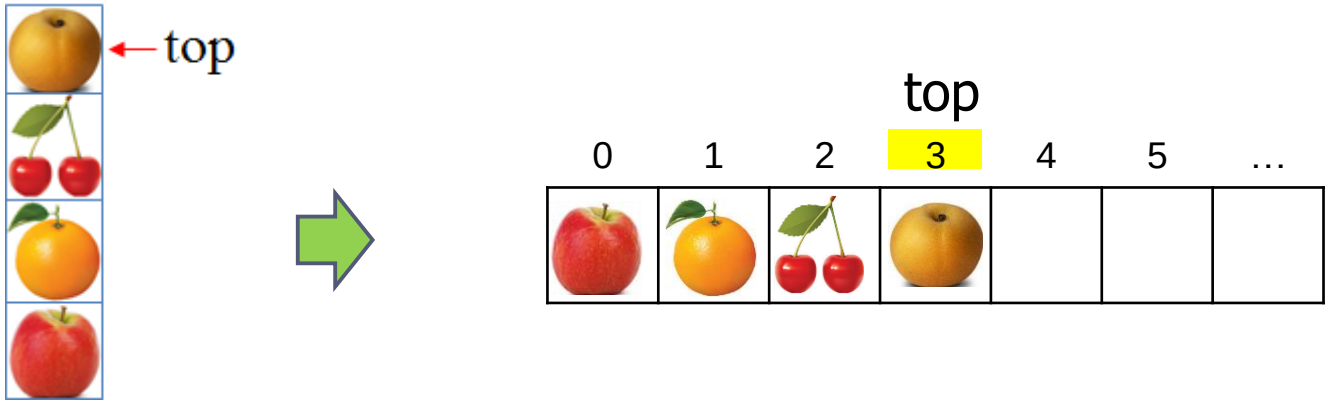
함수 호출 뒤의 시스템 스택

```
 1 #include    <iostream>
 2
 3 using namespace std ;
 4
 5 int function2(int val)
 6 {
 7     return val*val ;
 8 }
 9
10 int function1(int a)
11 {
12    a = function2(a) ;
13
14    return a + 1 ;
15 }
16
17 int main()
18 {
19     function1(3) ;
20
21     return 0 ;
22 }
```

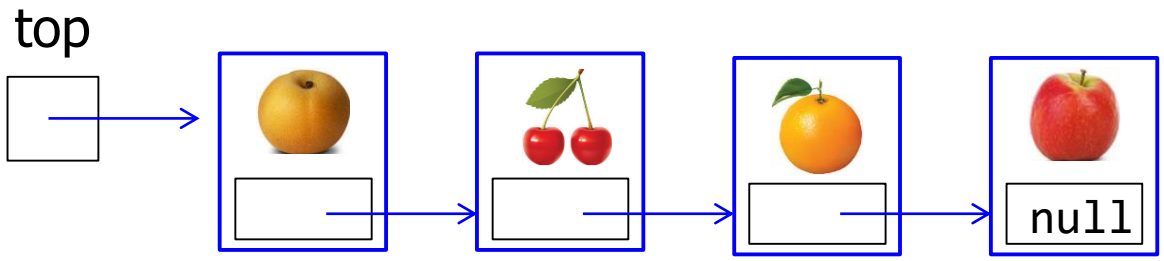
```
#include<iostream>

using namespace std ;

int factorial(int n)
{
if (n <= 1) return 1 ;
else return n * factorial(n-1) ;
}

int main()
{
cout << factorial(4) ;

return 0 ;
}
```

스택의 구현

- 배열로 구현된 스택



- 단순연결리스트로 구현된 스택



배열로 구현한 ArrayStack 클래스

```
01 import java.util.EmptyStackException;
02 public class ArrayStack<E> {
03     private E   s[];       // 스택을 위한 배열
04     private int top;       // 스택의 top 항목의 배열 원소 인덱스
05     public ArrayStack() { // 스택 생성자
06         s = (E[]) new Object[1];  // 초기에 크기가 1인 배열 생성
07         top = -1;
08     }
09     public int     size()    { return top+1;}        // 스택에 있는 항목의 수를  리턴
10     public boolean isEmpty() { return (top == -1);} // 스택이 empty이면 true 리턴

       // peek(), push(), pop(), resize() 메소드 선언
   }
```

- Line 01: java.util 라이브러리에 선언된 EmptyStackException 클래스를 이용하여 underflow 발생 시 프로그램 종료
- Line 05~08: ArrayStack 클래스의 생성자
  - - 크기가 1인 배열 s와 top = -1을 가진 객체 생성
- Line 09: 스택에 있는 item의 수 리턴
- Line 10: 스택이 empty인지를 검사하는 메소드

배열로 구현한 ArrayStack 클래스

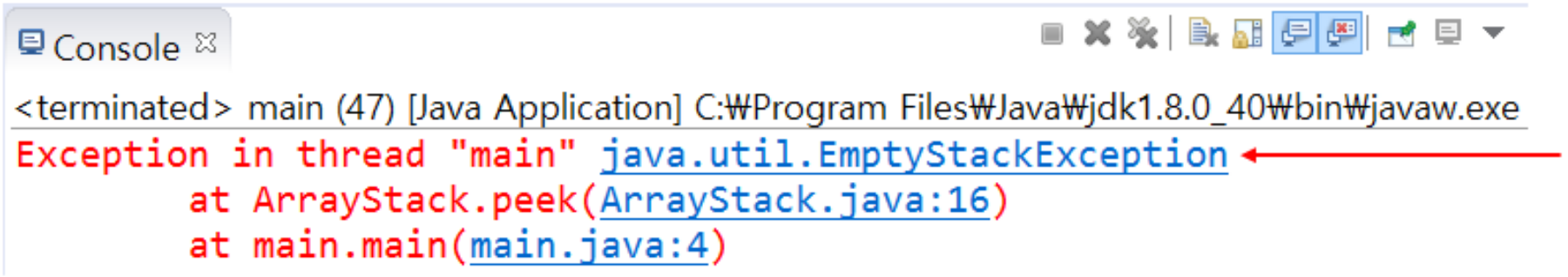
```
01 public E peek() {  // 스택 top 항목의 내용만을 리턴
02     if (isEmpty()) throw new EmptyStackException(); // underflow 시 프로그램 정지
03     return s[top];
04 }
```

- **peek() 메소드: 스택의 top에 있는 item을 리턴**
  - 만일 스택이 empty일 때에는 EmptyStackException을 발생시켜 예외 발생 에러 메시지 출력 후 프로그램 종료

배열로 구현한 ArrayStack 클래스

- underflow 발생 시 프로그램 종료

```
01 public class main {
02     public static void main(String[] args) {
03         ArrayStack<String> stack = new ArrayStack<String>();
04         stack.peek();
05         stack.push("apple");
```



배열로 구현한 ArrayStack 클래스

```
01  public void push(E newItem) { // push 연산
02      if (size() == s.length)
03          resize(2*s.length);    // 스택을 2배의 크기로 확장
04      s[++top] = newItem;        // 새 항목을 push
05  }
```

- **push() 메소드: 새 item을 스택에 삽입**
  - overflow가 발생하면, 2.1절에서 선언된 resize() 메소드를 호출하여 배열의 크기를 2배로 확장
  - Line 04: top을 1 증가시킨 후에 newItem을 s[top]에 저장

배열로 구현한 ArrayStack 클래스

```
01 public E pop() {  // pop 연산
02     if (isEmpty()) throw new EmptyStackException(); // underflow시 프로그램 정지
03     E item = s[top];
04     s[top--] = null;          // null로 초기화
05     if (size() > 0 && size()==s.length/4)
06         resize(s.length/2);  // 스택을 1/2 크기로 축소
07     return item;
08 }
```

- **pop() 메소드: 스택 top item을 삭제한 후 리턴**
  - Line 04: s[top]을 null로 만들어서 s[top]이 참조하던 객체를 가비지 컬렉션 처리
  - s[top]을 null로 만든 이후에는 top을 1 감소
  - Line 05~06: 스택의 item 수가 배열 s의 1/4만 차지하면, 메모리 낭비를 줄이기 위해 resize()를 호출하여 배열 s의 크기를 1/2로 축소
    - resize() 실행 이후에 배열 s의 1/2은 item들이 차지하고 나머지 1/2은 비어있게 됨

```java
public class main {
    public static void main(String[] args) {
        ArrayStack<String> stack = new ArrayStack<String>();

        stack.push("apple");
        stack.push("orange");
        stack.push("cherry");
        System.out.println(stack.peek());
        stack.push("pear");
        stack.print();
        stack.pop();
        System.out.println(stack.peek());
        stack.push("grape");
        stack.print();
    }
}
```

Problems @ Javadoc Console

<terminated> ArrayStack (1) [Java Application] C:₩Program Files₩Java₩jdk1.8.0_40₩bin₩javaw.exe

```
cherry
apple    orange  cherry  pear   <-- top
cherry
apple    orange  cherry  grape  <-- top
```

```
template <classT>
Class Stack {
private:
  T *stack;                // array for stack elements
  int top;    // array position of top element
  int capacity;    // capacity of stack array
public :
  ….
}

template <class T>
Stack<T>::Stack (int stackCapacity) : capacity (stackCapacity)
{
  if (capacity < 1) throw "Stack capacity must be > 0";
  stack = new T[capacity];
  top = -1;
}

template <class T>
inline bool  Stack<T>::isEmpty() const { return top == -1;}
```

```
template <class T>
inline T&  Stack<T>::top() const
{
  if (IsEmpty()) throw "Stack is empty";
  return stack[top];
}

template <class T>
void Stack<T>::push (const T& x)
{// Add x to the stack.
  if (top ==capacity-1)
  {
    resize(2 * capacity);
    capacity*=2;
  }
  stack[++top] = x;
}

template <class T>
void Stack<T>::pop()
{// Delete top element from the stack.
  if (IsEmpty()) throw "Stack is empty. Cannot delete.";
  stack[top--].~T();   // destructor for T
}
```

Java의 return 있는 pop은
top과 pop을 연속하여 호출하면
구현 가능

단순연결리스트로 구현한 ListStack 클래스

```
01 import java.util.EmptyStackException;
02 public class ListStack <E> {
03     private Node<E> top;    // 스택 top 항목을 가진 Node를 가리키기 위해
04     private int size;       // 스택의 항목 수
05     public ListStack() {    // 스택 생성자
06         top = null;
07         size = 0;
08     }
09     public int     size()    { return size;}       // 스택의 항목의 수를 리턴
10     public boolean isEmpty() { return size == 0;} // 스택이 empty이면 true 리턴

       //peek(), push(), pop() 메소드 선언
   }
```

- Node 클래스: 2.2절의 Node 클래스와 동일
- Line 01: 자바 util 라이브러리에 선언된 EmptyStackException 클래스이고, underflow 발생 시 프로그램 정지
- Line 05~08: ListStack 객체를 생성하기 위한 생성자, 객체는 스택 top item을 가진 Node 레퍼런스와 스택의 item 수를 저장하는 size를 가짐
- Line 09~10: 각각 스택의 item 수를 리턴, 스택이 empty이면 true를 리턴하는 메소드

단순연결리스트로 구현한 ListStack 클래스

```
01 public E peek() {  // 스택 top 항목만을 리턴
02     if (isEmpty()) throw new EmptyStackException(); // underflow 시 프로그램 정지
03     return top.getItem();
04 }
```

- **peek() 메소드: 스택의 top item을 리턴**
  - 스택이 empty인 경우, underflow 가 발생한 것이므로 프로그램 종료

```
01 public void push(E newItem){  // 스택 push 연산
02     Node newNode = new Node(newItem, top);  // 리스트 앞부분에 삽입
03     top = newNode;                          // top이 새 노드 가리킴
04     size++;                                 // 스택 항목 수 1 증가
05 }
```

- **push() 메소드: 스택에 새 item을 push하는 메소드**
  - Line 02: Node 객체를 생성하여 newItem을 newNode에 저장하고, top이 가진 레퍼런스를next에 복사
  - 이후 top이 새 Node를 가리키도록
    - 즉, 새 노드를 항상 연결리스트의 가장 앞에 삽입
  - Line 04: 스택의 item 수인 size 1 증가

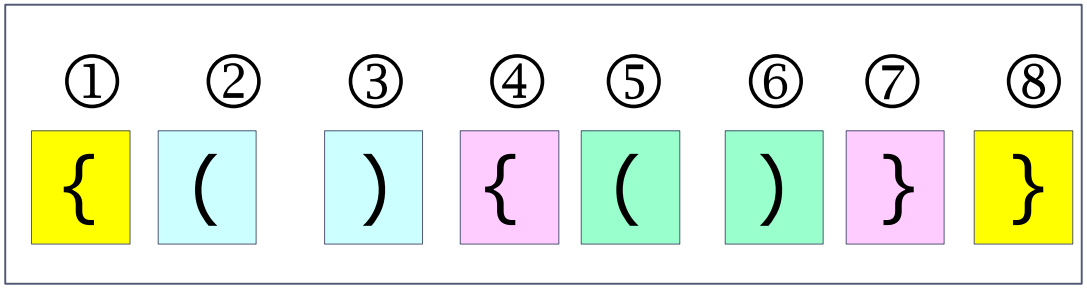
단순연결리스트로 구현한 ListStack 클래스

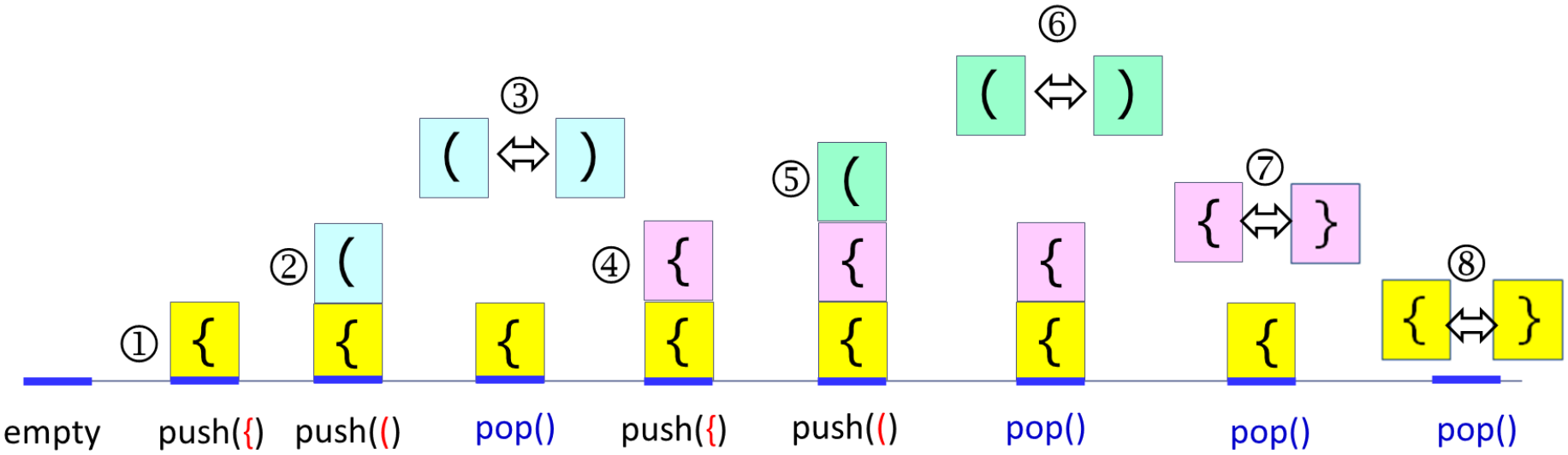
```
01 public E pop() {    // 스택 pop연산
02     if (isEmpty()) throw new EmptyStackException(); // underflow 시 프로그램 정지
03     E topItem = top.getItem();    // 스택 top 항목을 topItem에 저장
04     top = top.getNext();          // top이 top 바로 아래 항목을 가리킴
05     size--;                       // 스택 항목 수를 1 감소
06     return topItem;
07 }
```

- **pop() 메소드: 스택이 empty가 아닐 때, top이 가리키는 노드의 item을 topItem에 저장한 뒤 line 06에서 이를 리턴**
  - Line 04: top을 top이 참조하던 노드를 가리키게
  - Line 05: size 1 감소

스택 응용 예제 - 괄호쌍 맞추기

[예제 1]



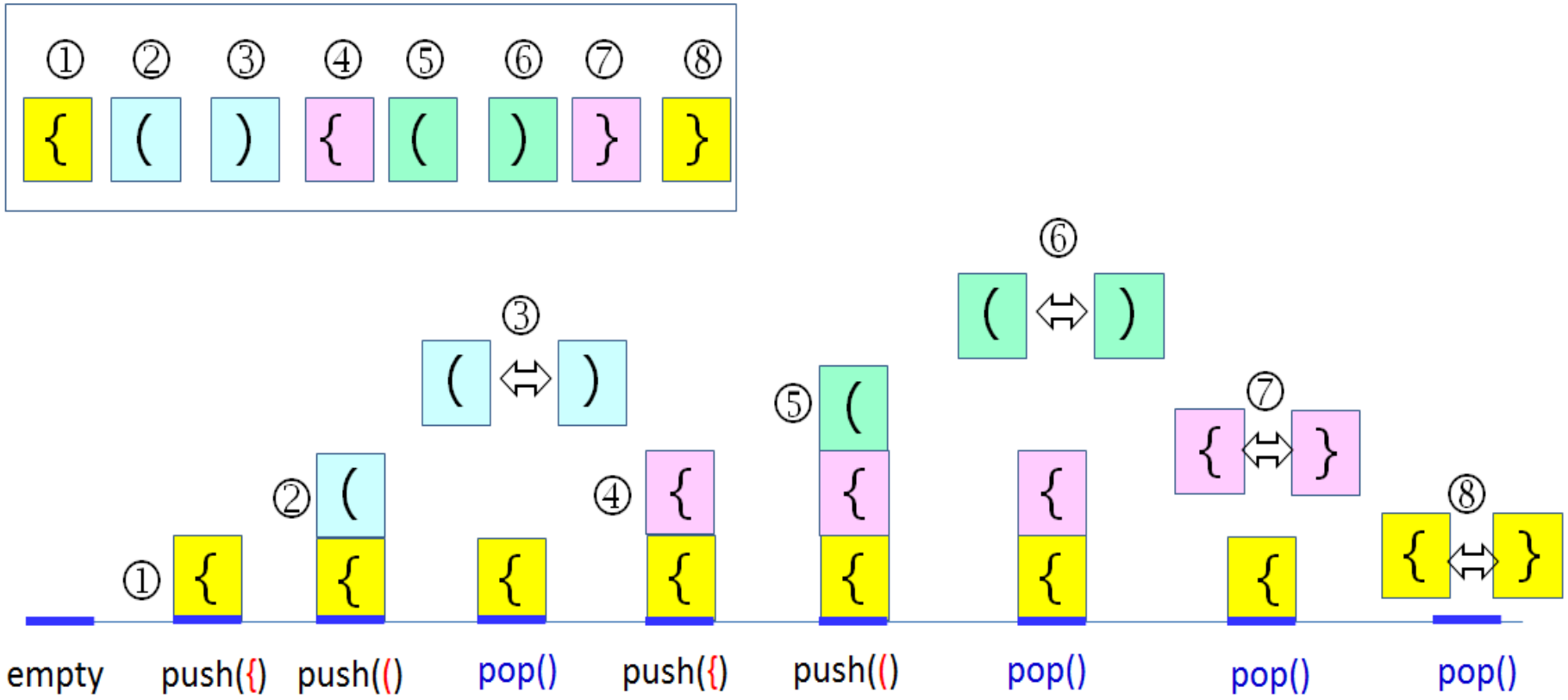


스택 응용 - 컴파일러의 괄호 짝 맞추기

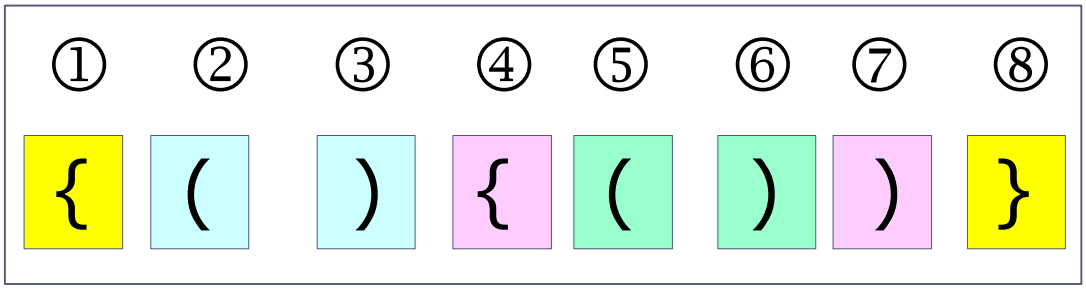
[핵심 아이디어] 왼쪽 괄호는 스택에 push,
오른쪽 괄호를 읽으면 pop 수행

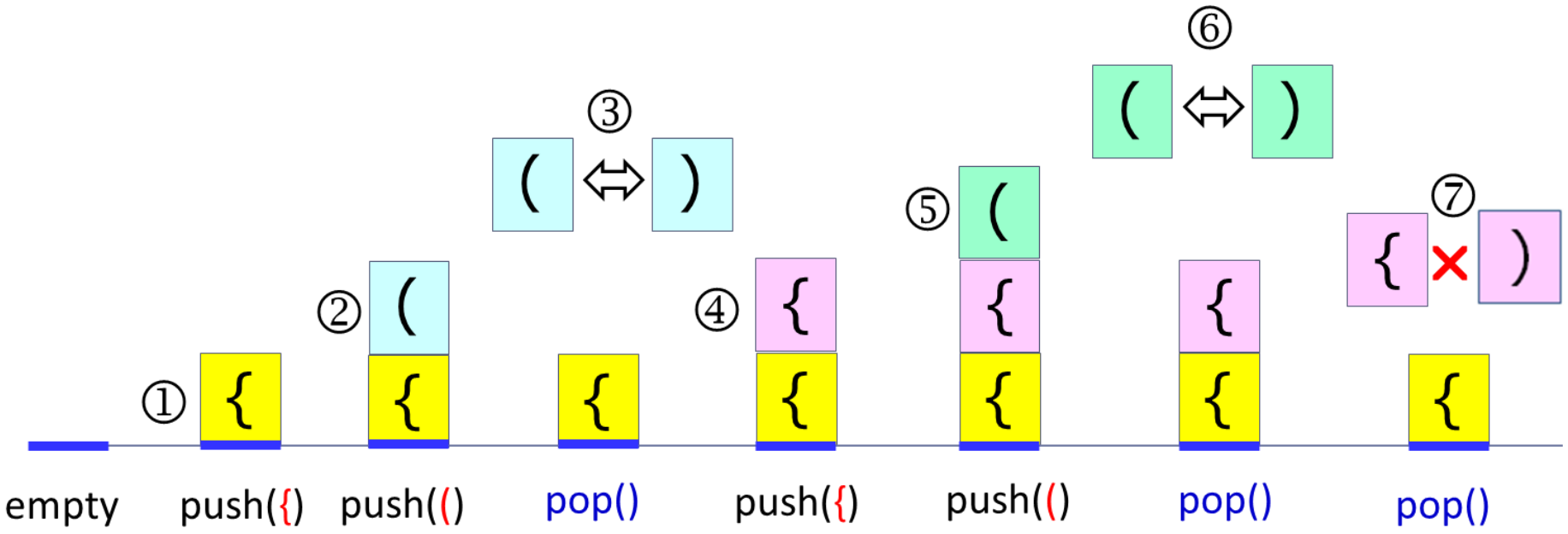
- pop된 왼쪽 괄호와 바로 읽었던 오른쪽 괄호가 다른 종류이면 에러 처리, 같은 종류이면 다음 괄호를 읽음
- 모든 괄호를 읽은 뒤 에러가 없고 스택이 empty이면, 괄호들이 정상적으로 사용된 것
- 만일 모든 괄호를 처리한 후 스택이 empty가 아니면 짝이 맞지 않는 괄호가 스택에 남은 것이므로 에러 처리

[예 제]



[예 제]





회문 검사하기

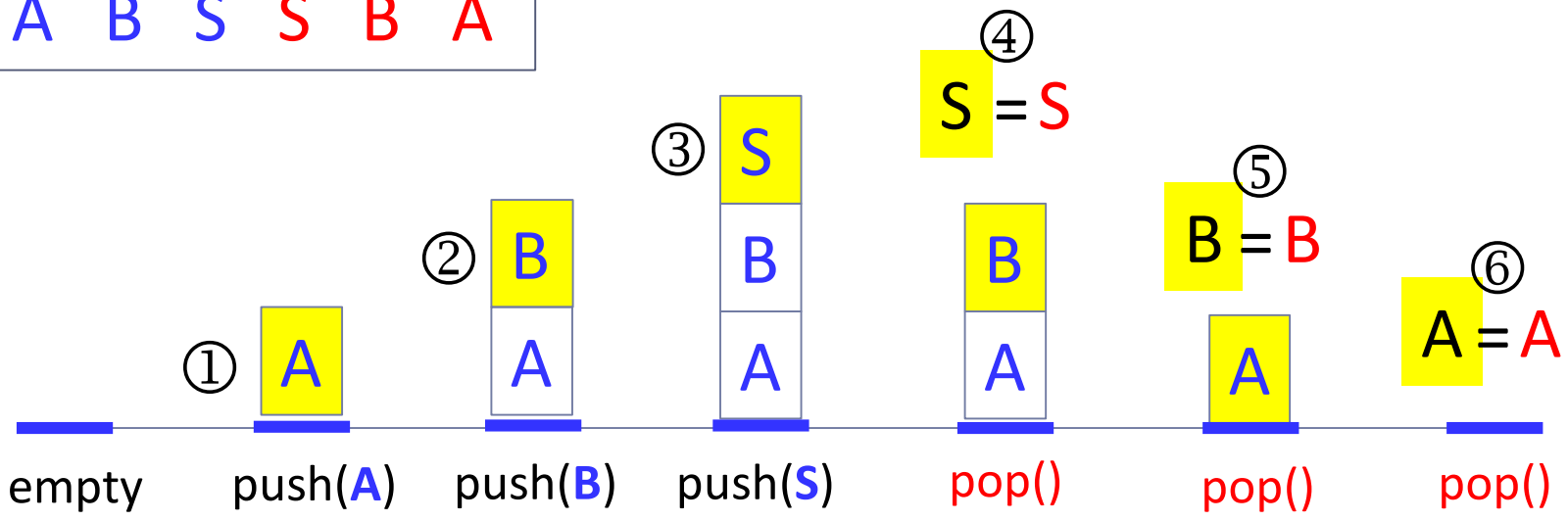
- [핵심 아이디어] 전반부의 문자들을 스택에 push한 후, 후반부의 각 문자를 차례로 pop한 문자와 비교

- 회문(Palindrome): 앞으로부터 읽으나 뒤로부터 읽으나 동일한 스트링
- 회문 검사하기는 주어진 스트링의 앞부분 반을 차례대로 읽어 스택에 push한 후, 문자열의 길이가 짝수이면 뒷부분의 문자 1 개를 읽을 때마다 pop하여 읽어 들인 문자와 pop된 문자를 비교하는 과정을 반복 수행
- 만약 마지막 비교까지 두 문자가 동일하고 스택이 empty가 되면, 입력 문자열은 회문

- 문자열의 길이가 홀수인 경우, 주어진 스트링의 앞부분 반을 차례로 읽어 스택에 push한 후, 중간 문자를 읽고 버린다. 이후 짝수 경우와 동일하게 비교 수행

[예제]

①	②	③	④	⑤	⑥
A	B	S	S	B	A



[예제]

①	②	③	④	⑤	⑥	⑦
R	A	C	E	C	A	R