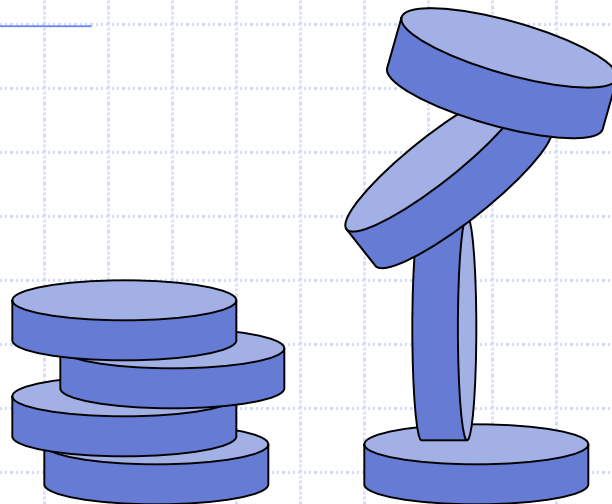


스택

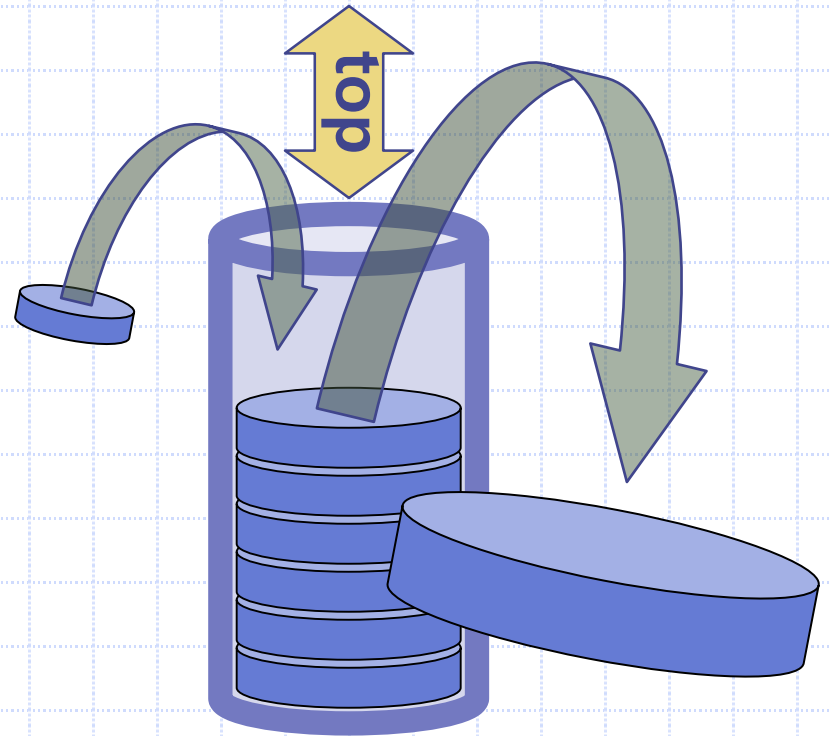


Outline

- ◆ 6.1 스택 ADT
- ◆ 6.2 스택 ADT 메소드
- ◆ 6.3 스택 응용
- ◆ 6.4 스택 ADT 구현
- ◆ 6.5 응용문제

스택 ADT

- ◆ **스택** ADT는 임의의 개체를 저장
- ◆ 삽입과 삭제는 **후입선출**(Last-In First-Out, LIFO) 순서를 따른다
- ◆ 삽입과 삭제는 스택의 **top**이라 불리는 위치에서 수행





스택 ADT 메소드

◆ 주요 스택 메소드

- **push**(e): 원소를 삽입
- **element pop**(): 가장 최근에 삽입된 원소를 삭제하여 반환

◆ 보조 스택 메소드

- **element top**(): 가장 최근에 삽입된 원소를 (삭제하지 않고) 반환
- **integer size**(): 저장된 원소의 수를 반환
- **boolean isEmpty**(): 아무 원소도 저장되어 있지 않고 비어 있는지 여부를 반환

◆ 예외

- **iterator elements**(): 스택 원소 전체를 반환
- **emptyStackException**(): 비어 있는 스택에서 삭제나 **top**을 시도할 경우 발령
- **fullStackException**(): 만원 스택에서 삽입을 시도할 경우 발령

스택 응용

◆ 직접 응용

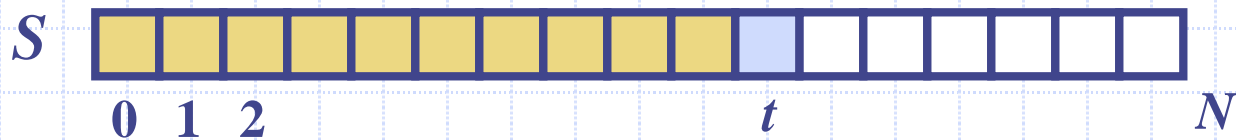
- 웹브라우저에서 방문한 웹페이지들의 기록
- 문서편집기에서 되돌리기 기록
- 윈도 운영체제에서 겹쳐진 윈도우들
- C++ 실행환경 또는 자바가상기계에서 메소드의 연쇄적인 호출
- 재귀의 구현

◆ 간접 응용

- 알고리즘 수행을 위한 보조 데이터구조
- 다른 데이터구조를 구성하는 요소

배열에 기초한 스택

- ◆ 크기 N 의 배열을 사용
- ◆ 원소들을 배열의 왼쪽에서 오른쪽으로 추가
- ◆ 변수 t 를 사용하여 **top** 원소의 첨자를 관리



초기화

◆ 초기에는 스택에 아무 원소도 없다

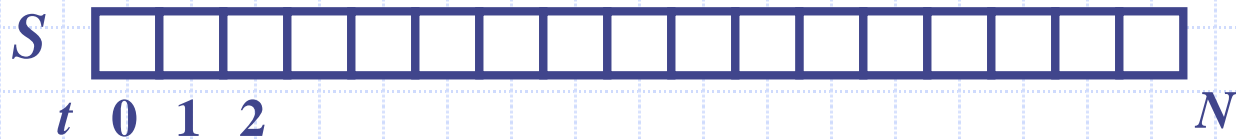
Alg *initStack()*

input stack S , size N , top t

output an empty stack S

1. $t \leftarrow -1$

2. **return**

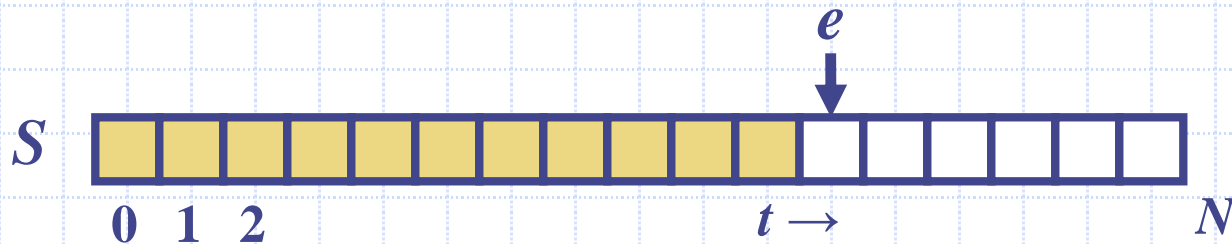


삽입

- ◆ 스택이 만원인 경우, **push** 작업은 **fullStackException**을 발령
 - 배열에 기초한 구현의 한계
 - 구현상의 오류일 뿐 **스택** ADT 취급 상 논리적 오류는 아님

Alg *push*(*e*)
input stack *S*, size *N*, top *t*,
element *e*
output none

1. if ($t = N - 1$)
 fullStackException()
2. $t \leftarrow t + 1$
3. $S[t] \leftarrow e$
4. return



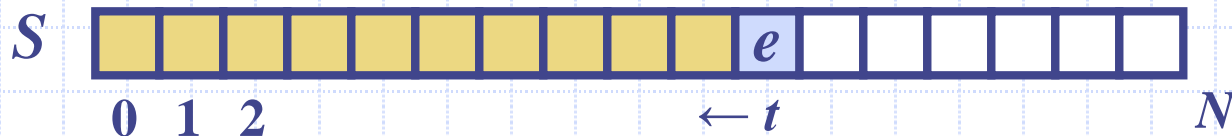
삭제

- ◆ 스택이 빈 경우, **pop** 작업은 **emptyStackException**을 발령
 - **스택** ADT 취급 상 **논리적 오류**

Alg *pop()*

input stack S , size N , top t
output element

1. **if** (*isEmpty()*)
 emptyStackException()
2. $t \leftarrow t - 1$
3. **return** $S[t + 1]$



보조 메소드

- ◆ **boolean isEmpty()**: 스택이 비어 있는지 여부를 반환
- ◆ **integer size()**: 스택에 저장된 원소의 수를 반환
- ◆ **element top()**: 가장 최근에 삽입된 원소를 삭제하지 않고 반환

Alg *isEmpty()*

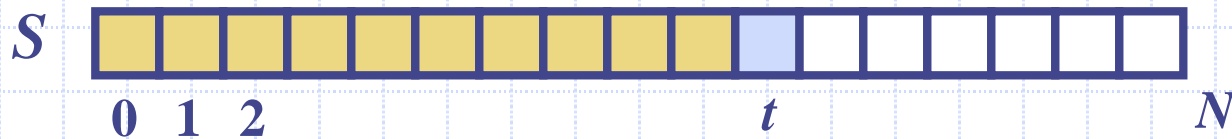
1. **return $t = -1$**

Alg *size()*

1. **return $t + 1$**

Alg *top()*

1. **if (*isEmpty()*)**
 emptyStackException()
2. **return $S[t]$**



성과와 제약

◆ 성능

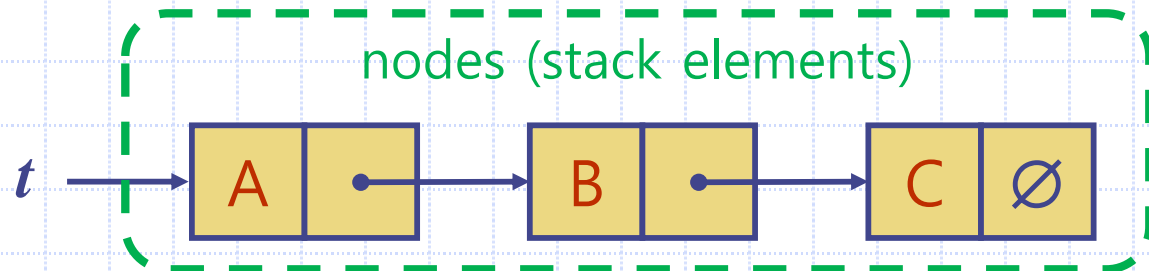
- 스택의 원소 개수를 n 이라 하면,
- 기억장소 사용: $O(n)$
- 각 작업의 실행시간: $O(1)$

◆ 제약

- 스택의 최대 크기를 예측할 수 있어야 하며 이 값은 실행중 변경할 수 없다 (예외: 동적 할당)
- 만원인 스택에 새로운 원소를 **push** 시도할 경우 구현상의 오류를 일으킨다

연결리스트에 기초한 스택

- ◆ 단일연결리스트를 사용하여 스택 구현 가능
 - 삽입과 삭제가 특정위치에서만 수행되므로, **헤더노드**는 불필요
- ◆ top 원소를 연결리스트의 첫 노드에 저장하고 이곳을 t 로 가리키게 한다
- ◆ 기억장소 사용: $O(n)$
- ◆ **스택** ADT의 각 작업: $O(1)$



초기화

◆ 초기에는 아무 노드도 없다

$t \longrightarrow \emptyset$

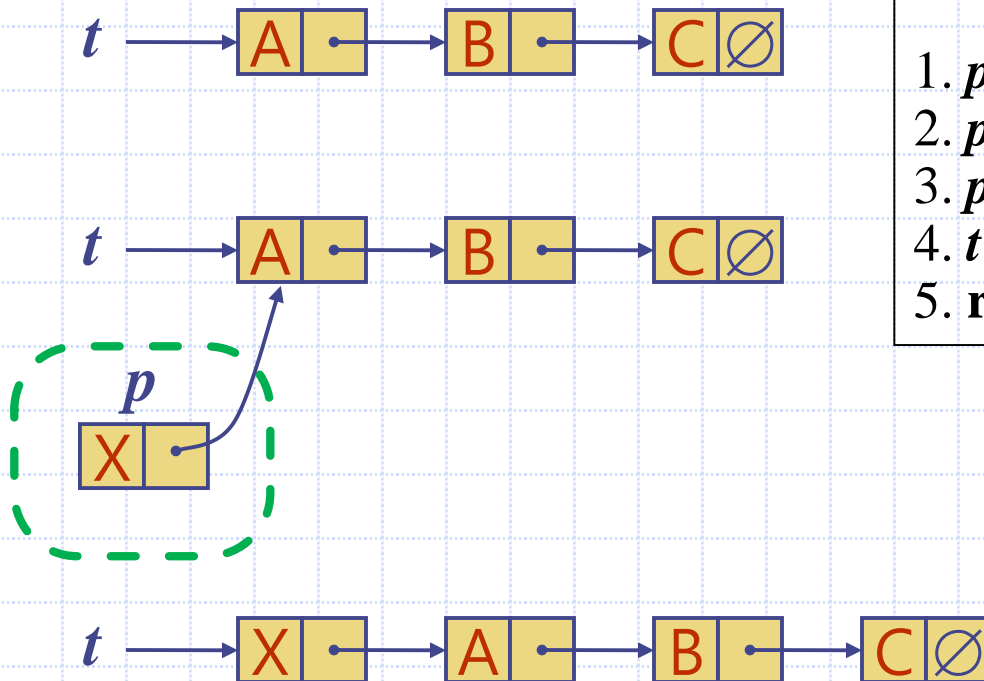
Alg *initStack()*

input top t

output an empty stack with top t

1. $t \leftarrow \emptyset$
2. **return**

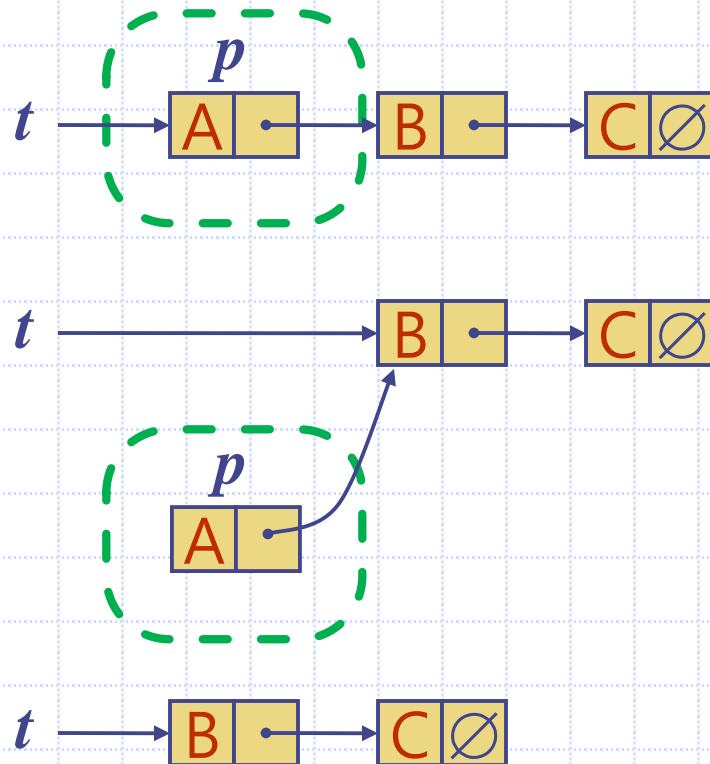
삽입



Alg *push*(e)
input top t , element e
output none

1. $p \leftarrow \text{getnode}()$
2. $p.\text{elem} \leftarrow e$
3. $p.\text{next} \leftarrow t$
4. $t \leftarrow p$
5. **return**

삭제



Alg **isEmpty()**
input top t
output boolean

1. return $t = \emptyset$

Alg **pop()**
input top t
output element

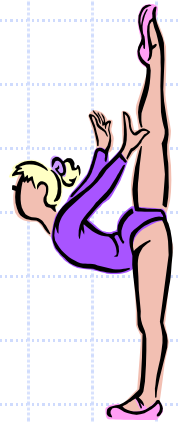
1. if (**isEmpty()**)
 emptyStackException()
2. $e \leftarrow t.\text{elem}$
3. $p \leftarrow t$
4. $t \leftarrow t.\text{next}$
5. **putnode**(p)
6. return e

응용문제

- ◆ 데이터구조 설계 문제들을 제시하고, 이를 해결하기 위한 알고리즘의 주요 또는 보조 데이터구조로서 **스택**을 어떻게 활용할지 공부
- ◆ 설계 문제
 - 심볼 균형
 - 기간
 - 후위수식
 - 다중스택



응용문제: 심볼 균형



- ◆ 컴파일러가 소스프로그램의 문법(syntax) 오류를 검사할 때, 심볼(symbol)들의 균형도 검사
- ◆ 균형 예:
 - (...)
 - ...{...(...{...}...)}...[...]
- ◆ 불균형 예:
 - [...(...)]...
 - (...{...[...]}...
- ◆ 문제: 심볼 균형 여부를 결정하는 utility 알고리즘 **isBalanced**를 작성하라

```
int f(A[...], ...)
/* ..... */
{
    if (...) {
        ... ;
        while (...) {
            A[...];
            ff(...);
        }
    }
    ... ;
}
```

해결: 스택을 사용

- ◆ end-of-file까지 심볼을 읽어들이면서 다음을 반복:
 - 열림(open) 심볼이면, 스택에 **push**
 - 닫힘(close) 심볼이면, 스택을 **pop**하여 서로 짝이 맞는지 확인 - 안 맞으면 **불균형**으로 결정
 - 둘 다 아니면, 통과
- ◆ 반복 종료 후 스택이 비어 있으면 **균형**으로 결정
- ◆ 이 알고리즘의 스택은 **일반**(generic) 스택 - 즉, 배열이나 연결리스트 등의 방식으로 구현을 특정하지 않는다

Alg **isBalanced()**

input stream of symbols

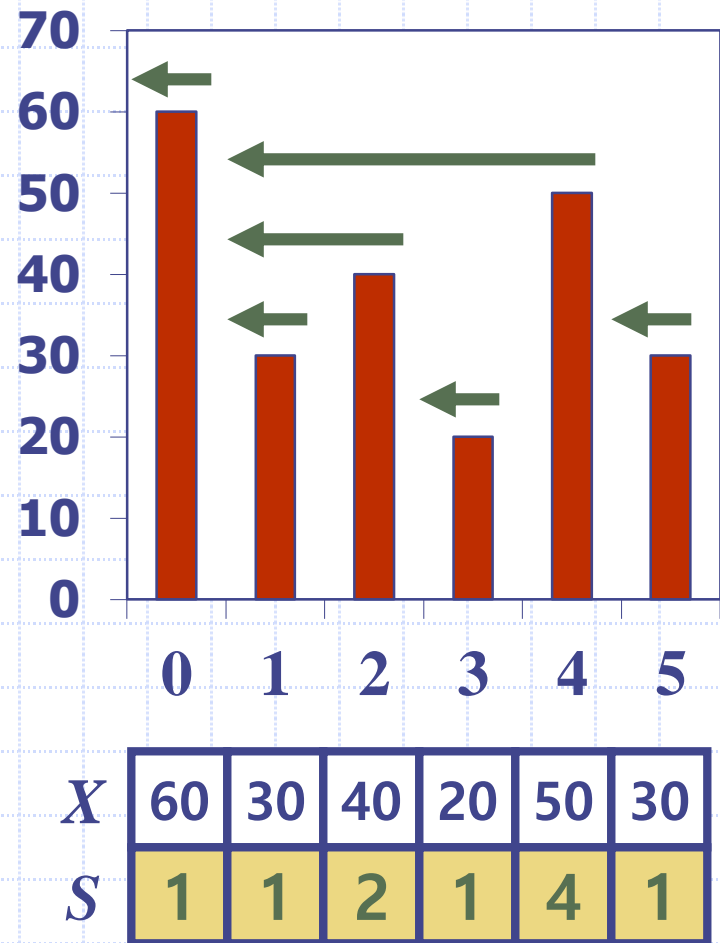
output boolean

1. $S \leftarrow \text{empty stack}$ {initStack}
2. **while** (!endOfFile())
 - $s \leftarrow \text{getSymbol}()$
 - if** (isOpenSymbol(s))
 - $S.\text{push}(s)$
 - elseif** (isCloseSymbol(s))
 - if** (S.isEmpty())
 - return False**
 - $t \leftarrow S.\text{pop}()$
 - if** (!isCounterpart(s, t))
 - return False**
3. **return** S.isEmpty()



응용문제: 기간

- ◆ 정의: 배열 X 에 대해, $X[i]$ 의 기간(span) $S[i]$ 란 $X[i]$ 바로 앞의 $X[j] \leq X[i]$ 인 연속적인 $X[j]$ 원소들의 최대 개수다
- ◆ 응용: 재무 분석 등 - 예를 들어,
 - 52주차 주가 최고점
 - 7일차 혈당 최고수치
- ◆ 문제: 주어진 배열로부터 기간배열을 구하라
- ◆ 해결
 - A. 정의를 적용
 - B. 스택을 사용



해결 A: 정의를 적용

- ◆ 기간의 정의를 적용함으로써 기간 값들을 2차 시간(quadratic time)에 계산할 수 있다

Alg *spans*(X, S, n)

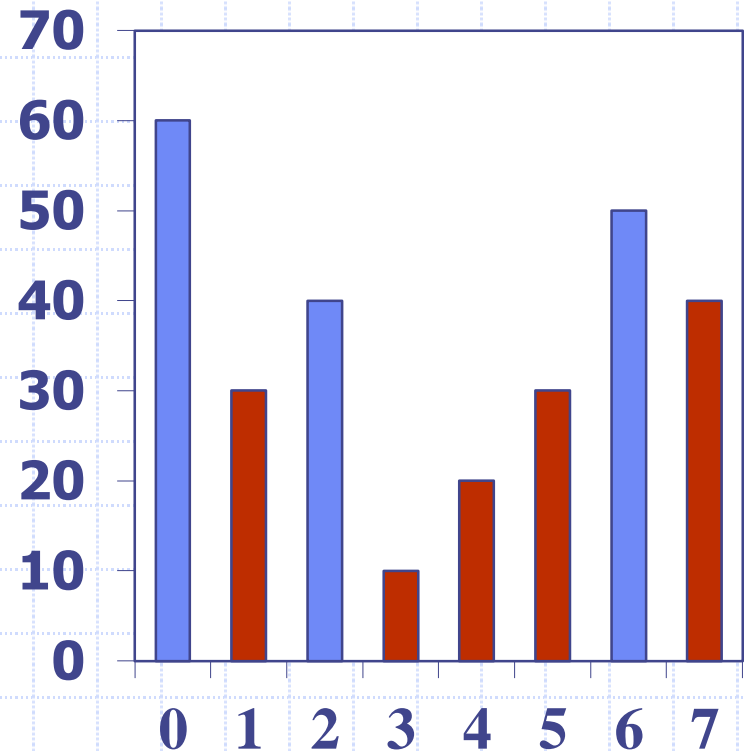
input array X, S of n integers

output array S of spans of X

1. for $i \leftarrow 0$ to $n - 1$	$\{ n \}$
$s \leftarrow 1$	$\{ n \}$
while $((s \leq i) \ \& \ (X[i - s] \leq X[i]))$	$\{ 1 + 2 + \dots + (n - 1) \}$
$s \leftarrow s + 1$	$\{ 1 + 2 + \dots + (n - 1) \}$
$S[i] \leftarrow s$	$\{ n \}$
2. return	$\{ 1 \}$

해결 B: 스택을 사용

- ◆ “뒤돌아 볼 때” 장벽으로 보이는 막대의 첨자를 스택에 저장
- ◆ 배열을 왼쪽에서 오른쪽으로 스캔
 - i 가 현재 첨자라 하면
 - $X[j] > X[i]$ 인 첨자 j 를 찾을 때까지 첨자들을 스택으로부터 pop
 - $S[i] \leftarrow i - j$ 로 치환
 - i 를 스택에 push



기간 계산 (선형)

- ◆ 각 배열첨자는:
 - 스택에 정확히 1회 push 되고
 - 스택으로부터 최대 1회 pop 된다
- ◆ while 문은 최대 n 회 실행한다
- ◆ 실행시간: $O(n)$
- ◆ **spans**의 S 는 일반 스택

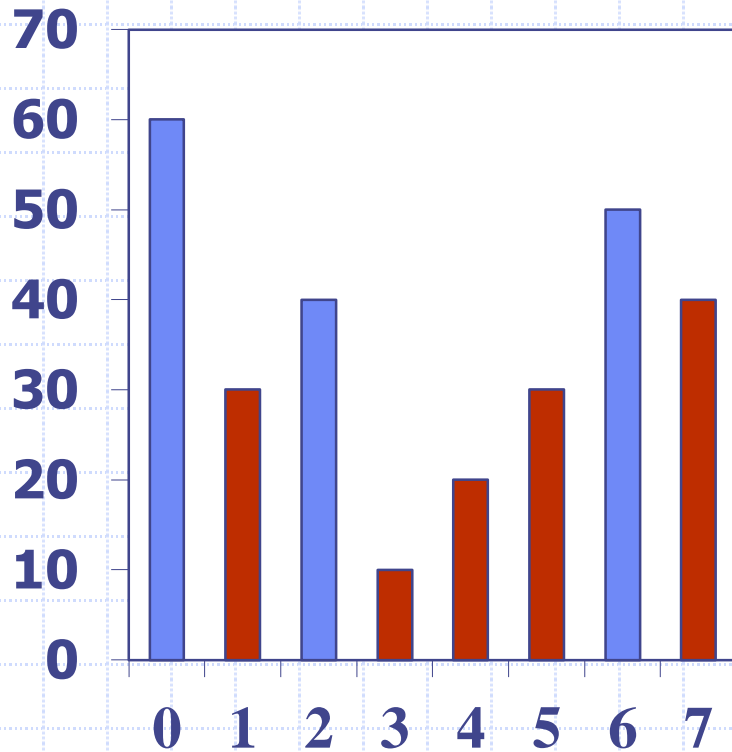
Alg **spans**(X, S, n)

input array X, S of n integers

output array S of spans of X

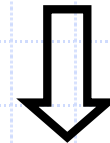
```
1.  $A \leftarrow \text{empty stack}$  { 1 }
2. for  $i \leftarrow 0$  to  $n - 1$  { n }
    while ( $\neg A.\text{isEmpty}()$  &
           ( $X[A.\text{top}()] \leq X[i]$ )) { n }
         $A.\text{pop}()$  { n }
    if ( $A.\text{isEmpty}()$ ) { n }
         $S[i] \leftarrow i + 1$  { n }
    else
         $S[i] \leftarrow i - A.\text{top}()$  { n }
         $A.\text{push}(i)$  { n }
3. while ( $\neg A.\text{isEmpty}()$ ) { n }
     $A.\text{pop}()$  { n }
4. return { 1 }
```

기간 계산 수행 예



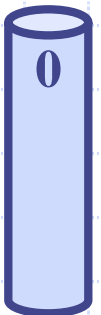
(a) for 문에 진입하기 전

X	60	30	40	10	20	30	50	40
S								



(b) $i = 0$ 실행 직후

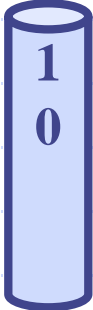
X	60	30	40	10	20	30	50	40
S	1							



기간 계산 수행 예 (conti.)

(c) $i = 1$ 실행 직후

X	60	30	40	10	20	30	50	40
S	1	1						



(e) $i = 3$ 실행 직후

X	60	30	40	10	20	30	50	40
S	1	1	2	1				



(d) $i = 2$ 실행 직후

X	60	30	40	10	20	30	50	40
S	1	1	2					



(f) $i = 4$ 실행 직후

X	60	30	40	10	20	30	50	40
S	1	1	2	1	2			



기간 계산 수행 예 (conti.)

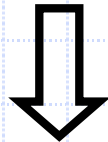
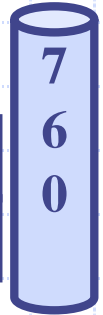
(g) $i = 5$ 실행 직후

X	60	30	40	10	20	30	50	40
S	1	1	2	1	2	3		



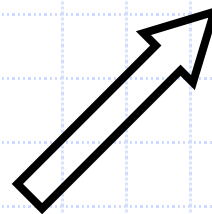
(i) $i = 7$ 실행 직후

X	60	30	40	10	20	30	50	40
S	1	1	2	1	2	3	6	1



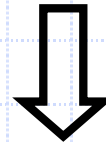
(h) $i = 6$ 실행 직후

X	60	30	40	10	20	30	50	40
S	1	1	2	1	2	3	6	



(j) 완료

X	60	30	40	10	20	30	50	40
S	1	1	2	1	2	3	6	1



응용문제: 후위수식



- ◆ 아래와 같은 쇼핑 내역을 계산하고 싶다(단위: 원)

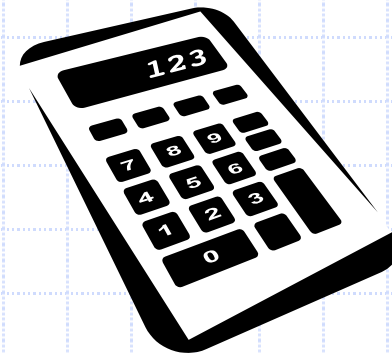
{5, 5, 5, 2, 6, 3, 6, 3}

- ◆ 중위수식(infix expression)

- $5 \times 3 + 2 + (6 + 3) \times 2$
- 암묵적 우선순위(precedence)
- 우선순위는 괄호에 의해 무시됨

- ◆ 후위수식(postfix expression)

- $5\ 3 \times 2 + 6\ 3 + 2 \times +$
- 우선순위 없음
- 괄호 없음



응용문제: 후위수식 (conti.)

◆ 중위수식: 피연산자 연산자 피연산자

- $(A + B) \times C - (D \times E)$
- 우선순위는 괄호에 의해 무시됨
- 예: 수학

◆ 후위수식: 피연산자... 연산자

- $A B + C \times D E \times -$
- 역폴란드식(reverse Polish) 표기라고도 불림
- 예: 국어

◆ 전위수식: 연산자 피연산자...

- $- \times + A B C \times D E$
- 폴란드식(Polish) 표기라고도 불림
- 예: 영어

응용문제: 후위수식 (conti.)

- ◆ 후위수식은 수식의 평가가 단순직선적이므로 컴퓨터에 의한 처리가 용이
- ◆ 처리의 두 단계
 1. 수식 전환(convert): 소스프로그램의 중위수식을 모두 후위 수식으로 전환
 2. 수식 평가(evaluate): 후위수식들을 평가

응용문제: 후위수식 (conti.)

◆ 문제: 두 단계의 알고리즘을 각각 작성하라

- `convert()`: 중위수식을 후위수식으로 변환
- `evaluate()`: 후위수식을 평가

◆ 전제: 입력은 표준입력을 통한 중위수식으로써 표준 우선순위의 `+`, `-`, `x`, `/`, `(`, `)` 연산자들로 표현됨

◆ 사용 가능

- 표준입력 함수
 - ◆ `symbol getSymbol()`: 입력으로부터 한 개의 심볼을 읽어들이 반환
 - ◆ `boolean endOfFile()`: end-of-file 도달 여부를 반환
- Utility 함수
 - ◆ `boolean isOperand(s)`: 심볼 `s`가 피연산자인지 여부를 반환

해결: 스택을 사용

- ◆ 두 알고리즘 모두에서 스택을 사용
 - `convert`에서는, 입력 수식의 연산자들을 스택에 저장
 - `evaluate`에서는, 입력 수식의 피연산자들을 스택에 저장
- ◆ 전제: 각 스택을 일반 스택으로 설계

해결: 수식 전환

P : array[0..4] of integers = (0, 1, 1, 2, 2)
{Precedence information of operators
needs to be defined externally as:

	(+	-	x	/
P	0	1	1	2	2

}

Alg **convert()**

input stream of legal infix expression

output stream of postfix expression

1. $S \leftarrow$ empty stack {stores operators}

2. **while** (!endOfFile())

$s \leftarrow$ getSymbol()

if (isOperand(s))

$write(s)$

elseif (s = '(')

$S.push(s)$

elseif (s = ')')

while ($S.top() \neq '('$)

$write(S.pop())$

$S.pop()$

else {s is an operator}

while (! $S.isEmpty()$ &

($P[s] \leq P[S.top()]$))

$write(S.pop())$

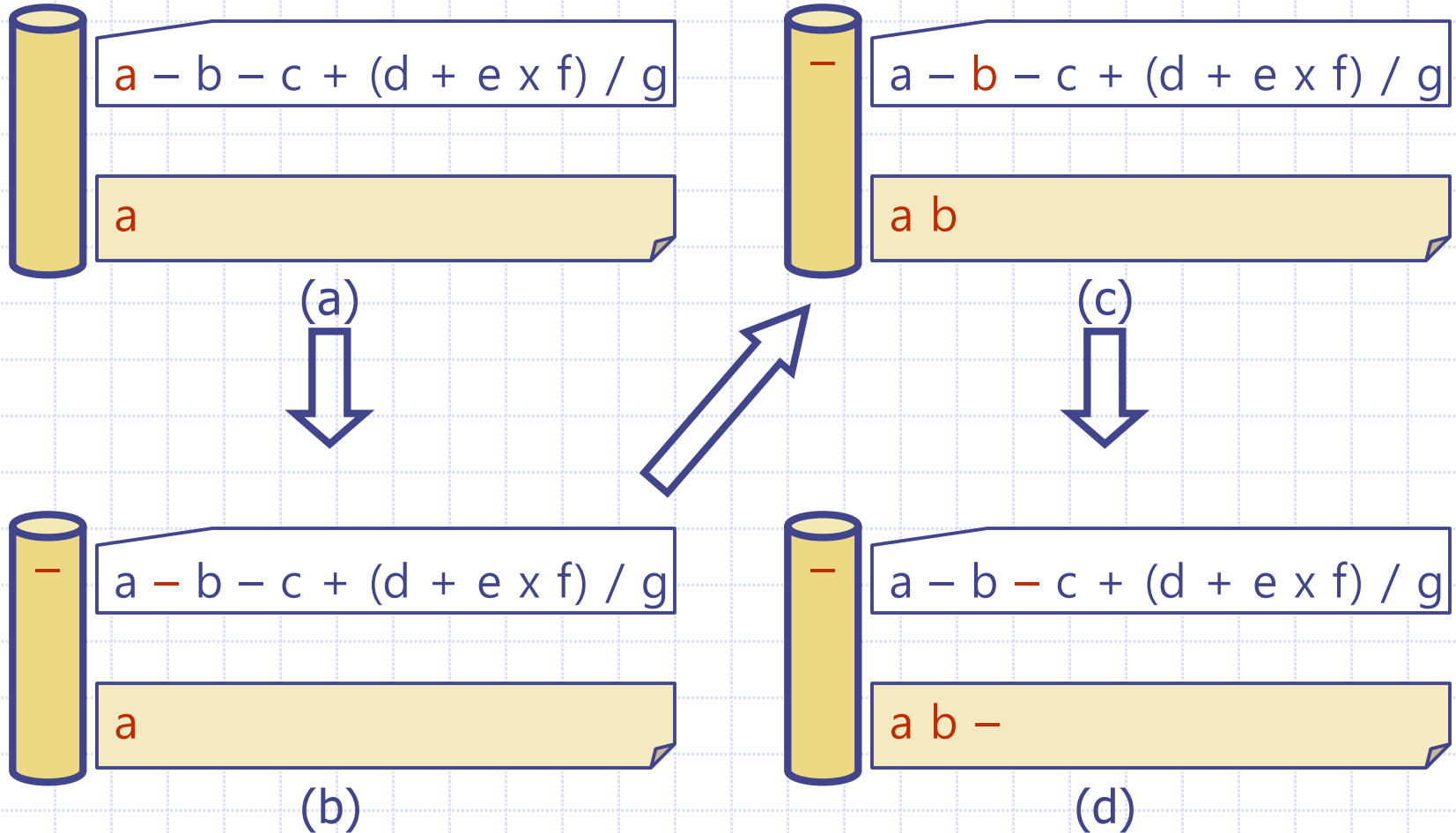
$S.push(s)$

3. **while** (! $S.isEmpty()$)

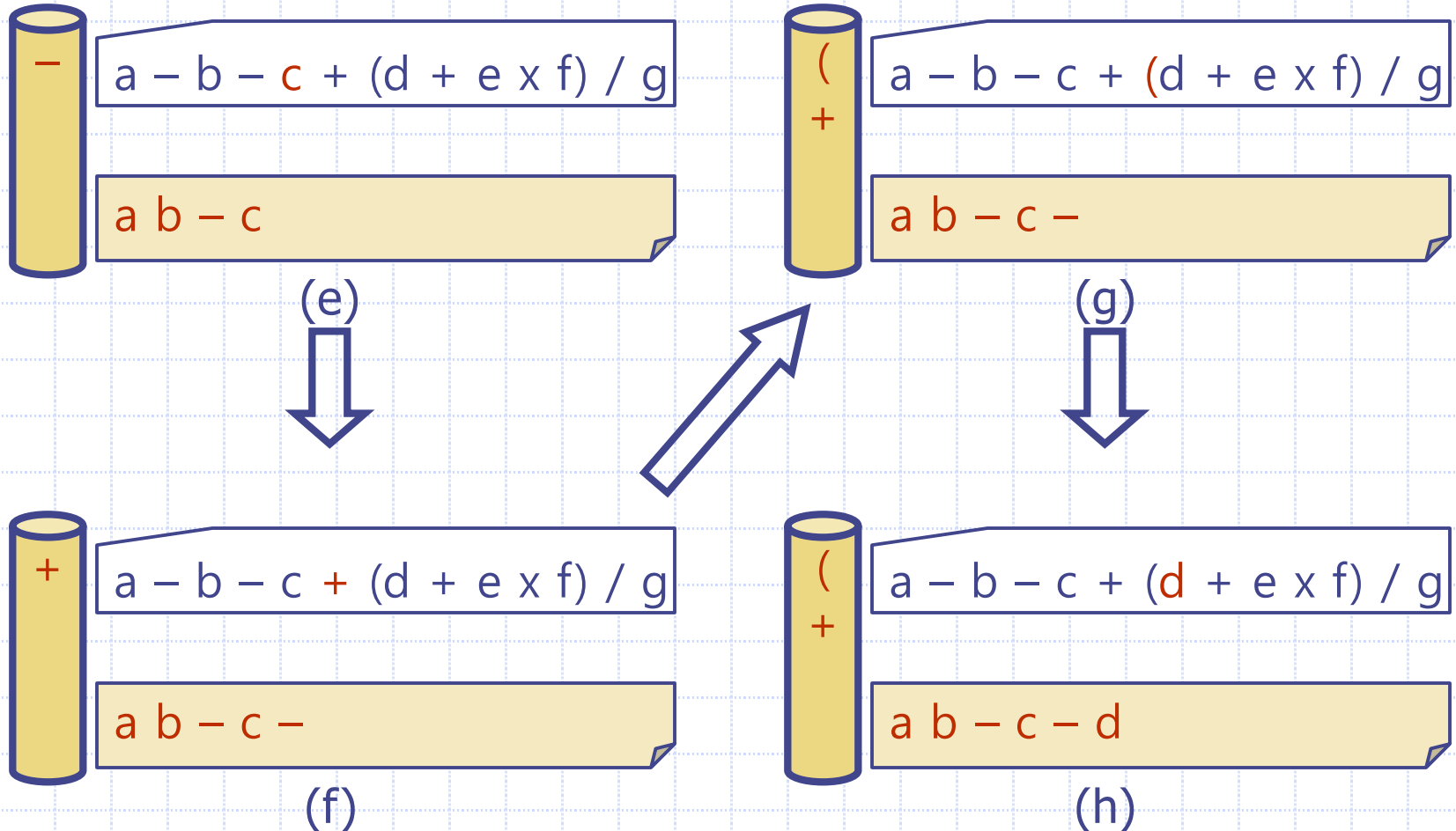
$write(S.pop())$

4. **return**

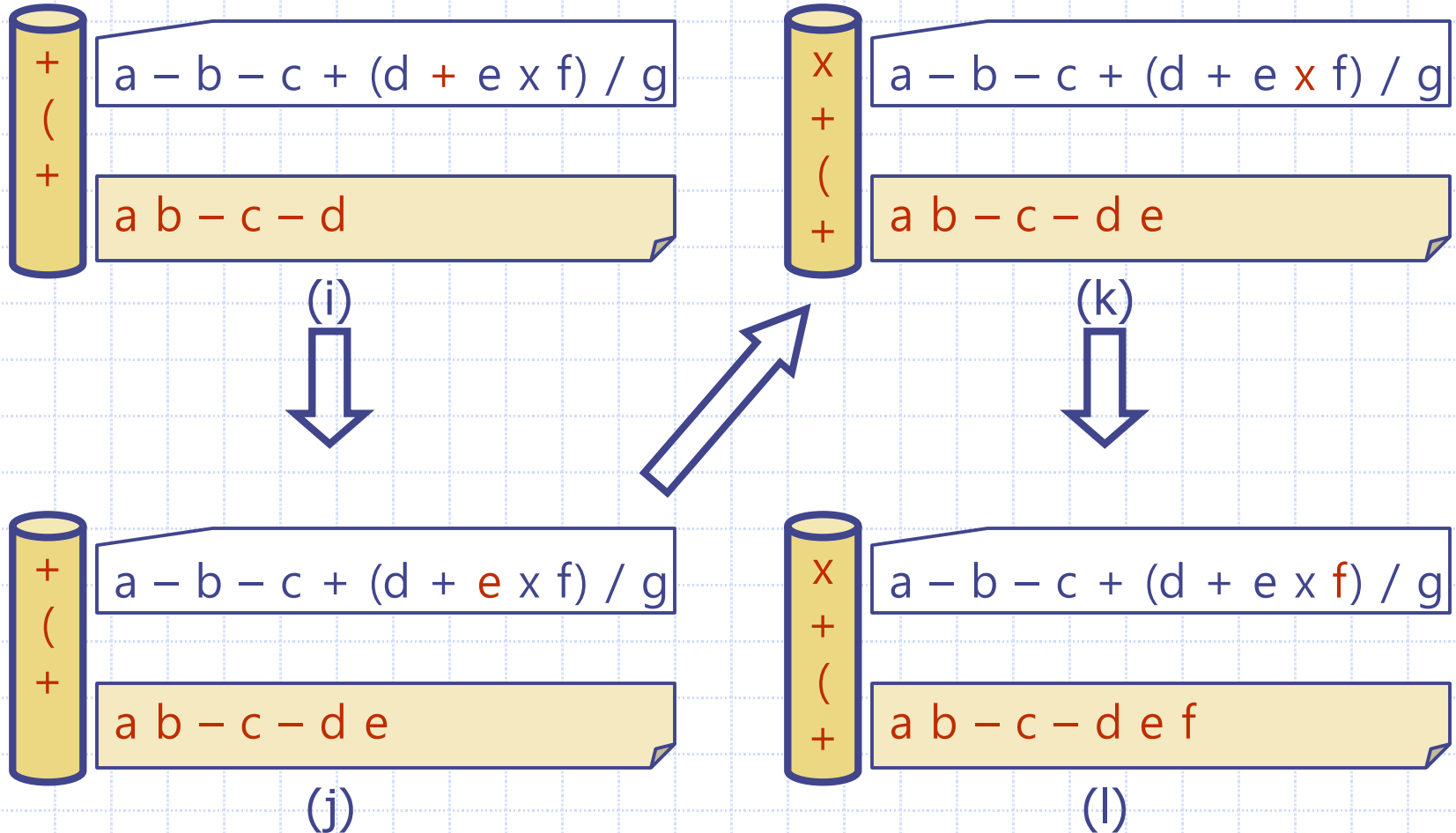
해결: 수식 전환 예



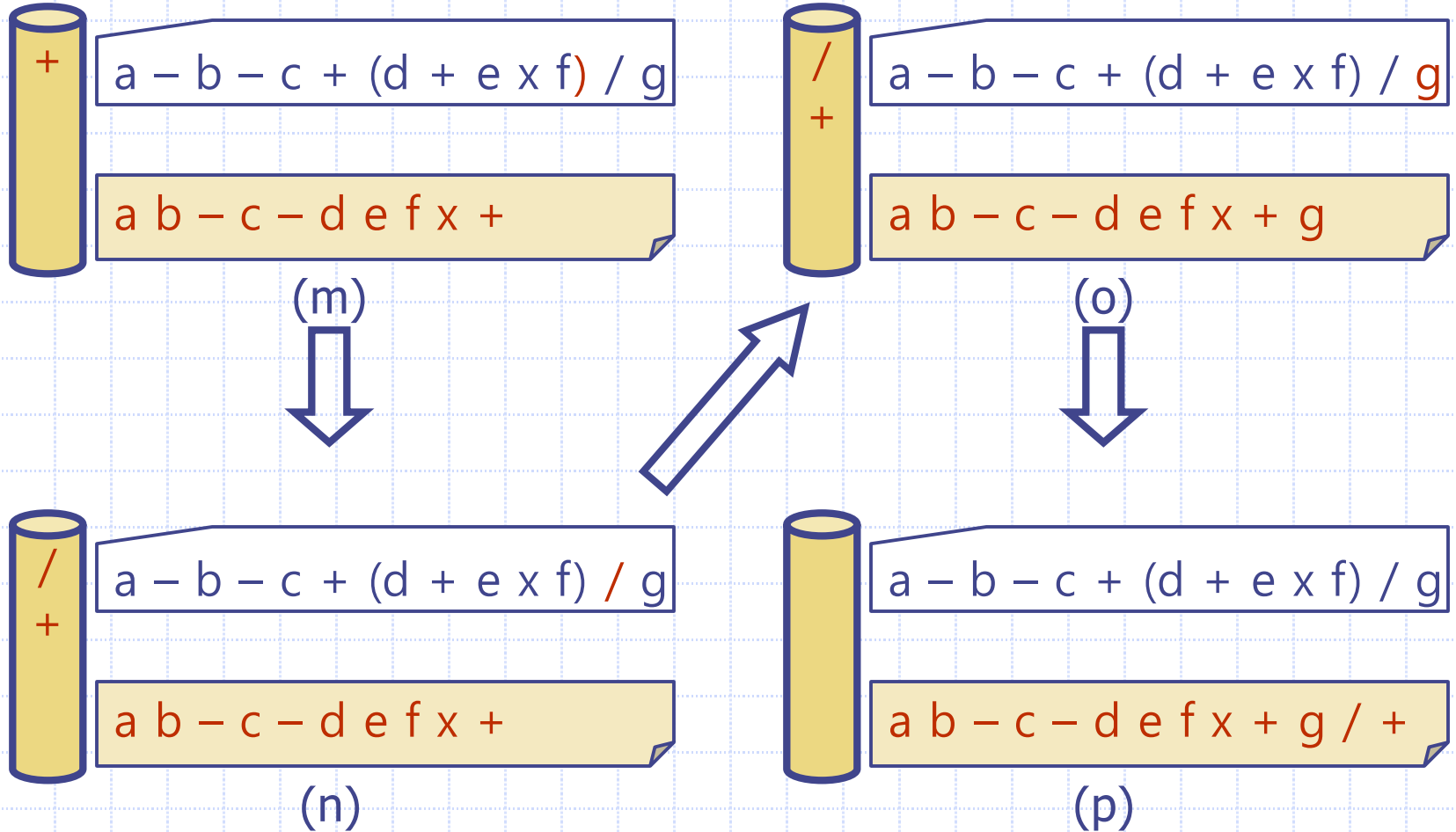
해결: 수식 전환 예 (conti.)



해결: 수식 전환 예 (conti.)



해결: 수식 전환 예 (conti.)



해결: 수식 평가

Alg *evaluate()*

input stream of legal postfix exp

output number

1. $S \leftarrow \text{empty stack}$ {stores operands}
2. **while** (*!endOfFile()*)
 $s \leftarrow \text{getSymbol}()$
 if (*isOperand(s)*)
 $S.\text{push}(s)$
 else { s is an operator}
 $a \leftarrow S.\text{pop}()$
 $b \leftarrow S.\text{pop}()$
 $S.\text{push}(\text{doOperator}(s, b, a))$
3. **write**($S.\text{pop}()$)
4. **return**

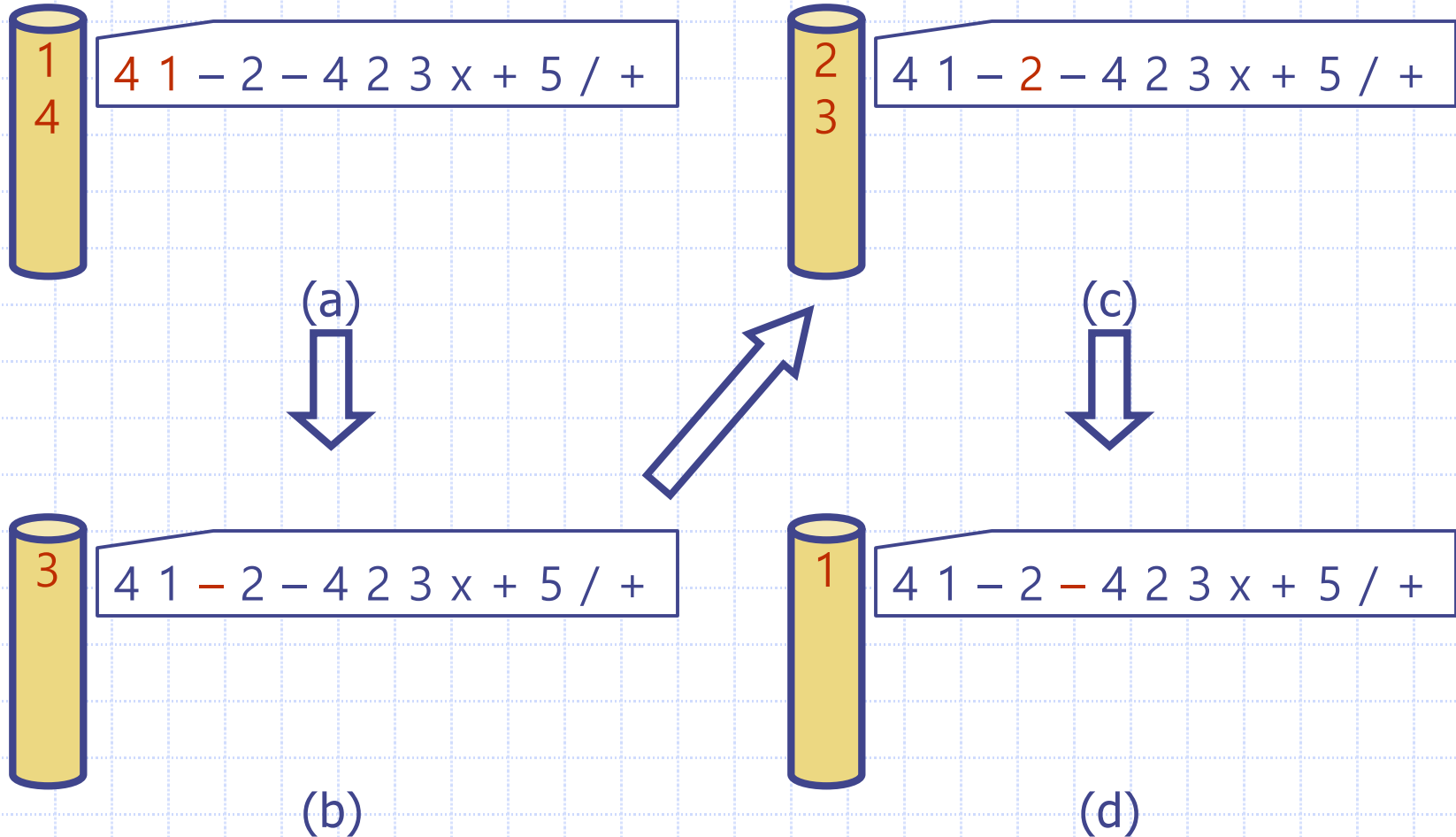
Alg *doOperator(op, x, y)*

input operator op , operand x, y

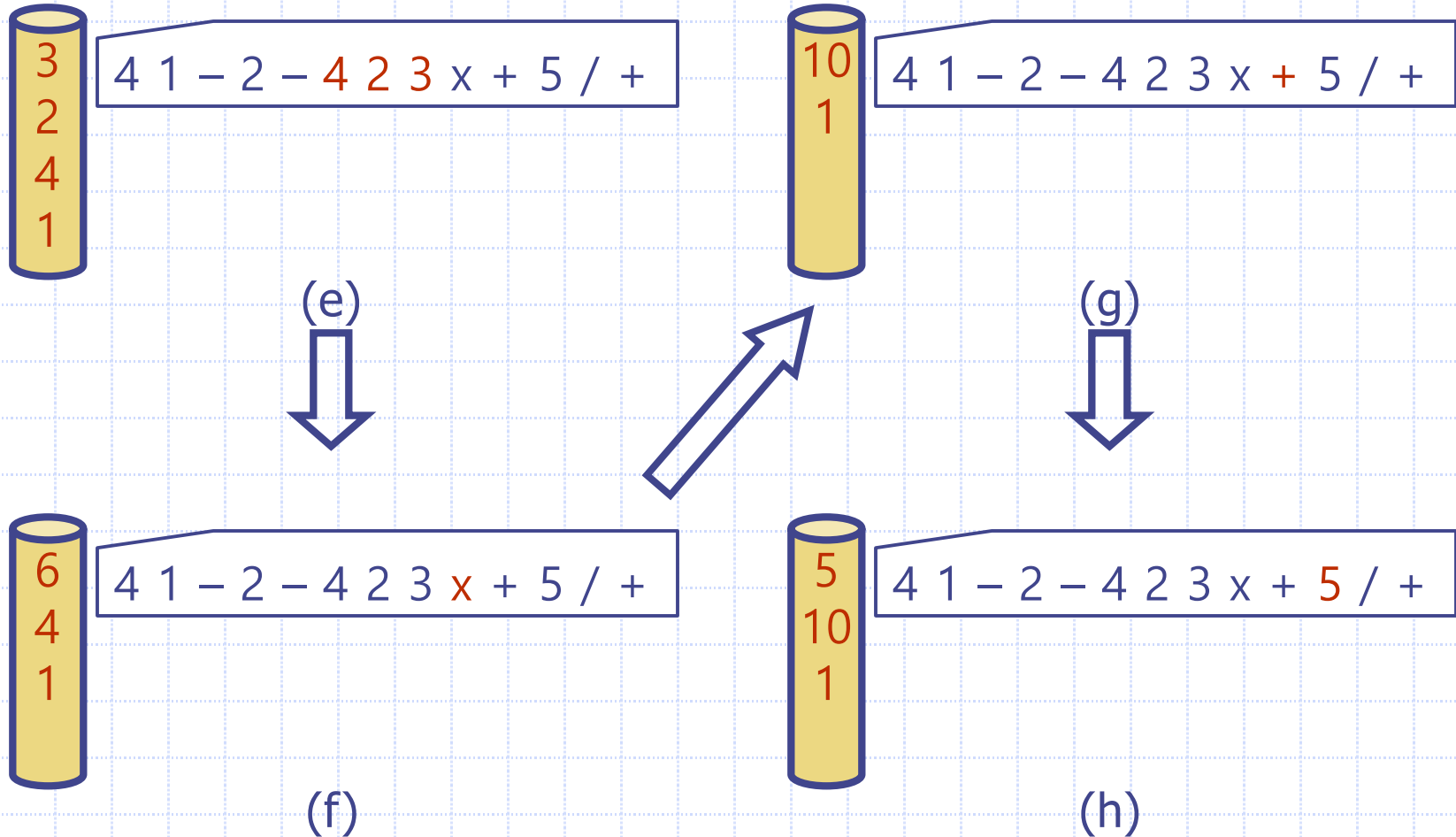
output result of applying y to x

1. **switch** (op)
 $'+' : v \leftarrow x + y$
 $'-' : v \leftarrow x - y$
 $'\times' : v \leftarrow x \times y$
 $'/' : v \leftarrow x / y$
2. **return** v

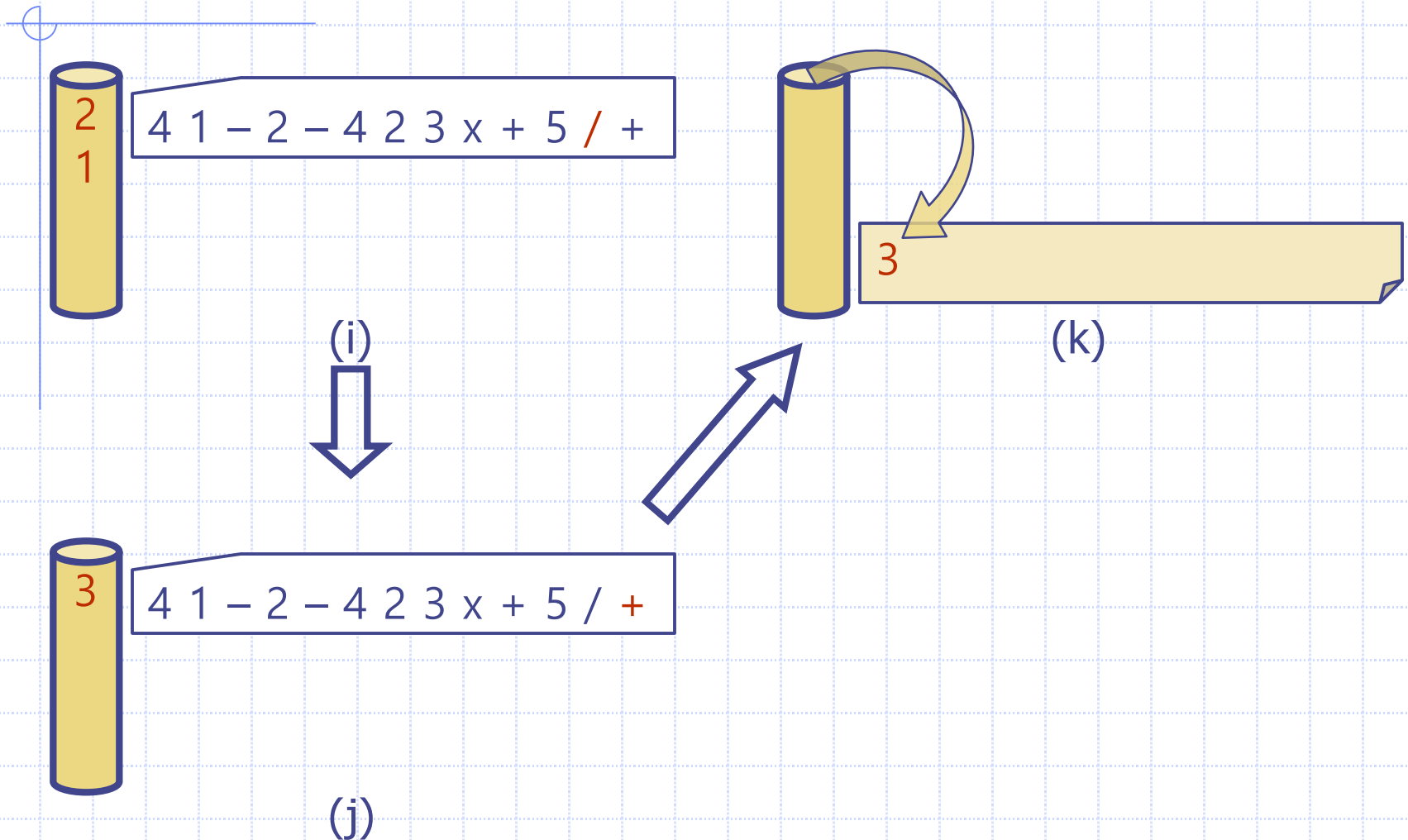
해결: 수식 평가 수행 예

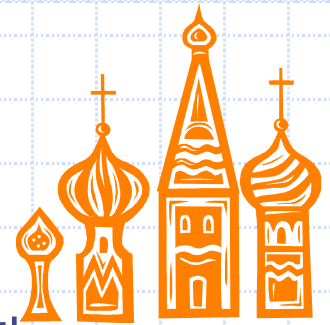


해결: 수식 평가 수행 예 (conti.)



해결: 수식 평가 수행 예 (conti.)





응용문제: 다중스택

◆ 문제: $n > 2$ 개의 스택, S_0, S_1, \dots, S_{n-1} 을 구현할 데이터구조를 설계하고 아래의 관련 메소드들을 작성하라

- integer **size**(i): S_i ($0 \leq i \leq n-1$)의 크기를 반환
- boolean **isEmpty**(i): S_i 가 비어 있는지 여부를 반환
- boolean **isFull**(i): S_i 가 만원인지 여부를 반환
- element **top**(i): S_i 의 top 원소를 반환
- **initMultiStack**(): n 개의 스택 S_0, S_1, \dots, S_{n-1} 을 초기화
- **push**(i, e): S_i 에 원소 e 를 삽입
- element **pop**(i): S_i 의 top 원소를 삭제하여 반환

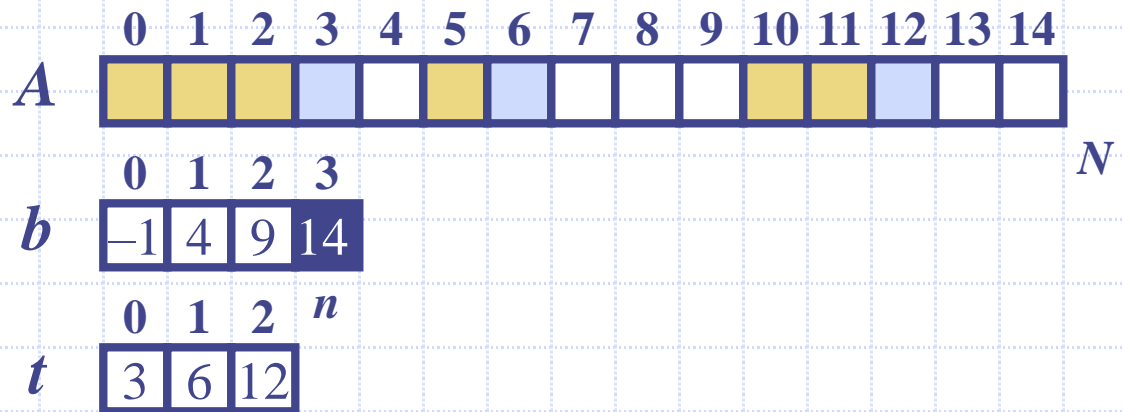
◆ 해결

- A. 1D 배열 사용
- B. 연결리스트의 배열 사용

해결 A: 1D 배열 사용

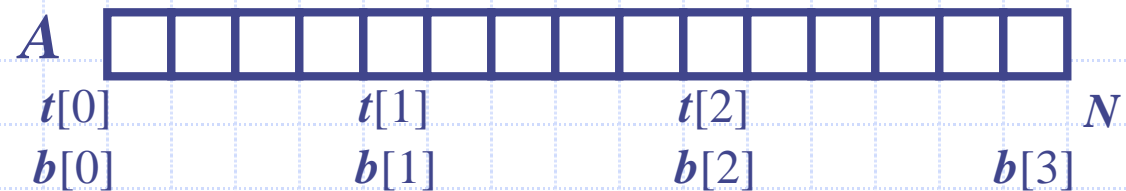
- ◆ 크기 N 인 1D 배열 A 를 분할하여 각 스택에 동일한 기억장소를 할당하고 각 스택이 오른쪽 방향으로 성장하도록 정의
- ◆ S_0, S_1, \dots, S_{n-1} 스택들의 base 와 top들을 각각 크기 $n + 1$ 인 배열 b 와 크기 n 인 배열 t 에 저장 - 즉,
 - $b[i]$ 는 S_i ($0 \leq i \leq n$)의 base 값을 저장
 - $t[i]$ 는 S_i ($0 \leq i \leq n - 1$)의 top 값을 저장
- ◆ N 은 n 의 정수 배수 값을 선택

예: $N = 15, n = 3$

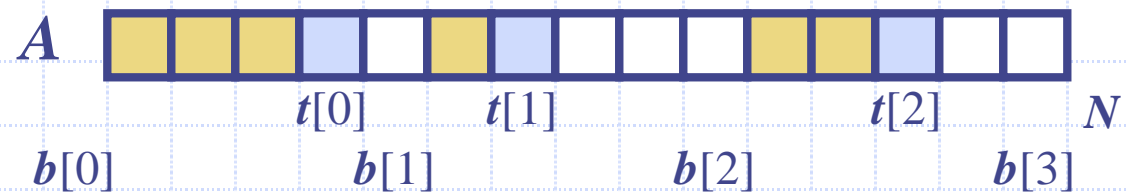


해결: 작동 원리

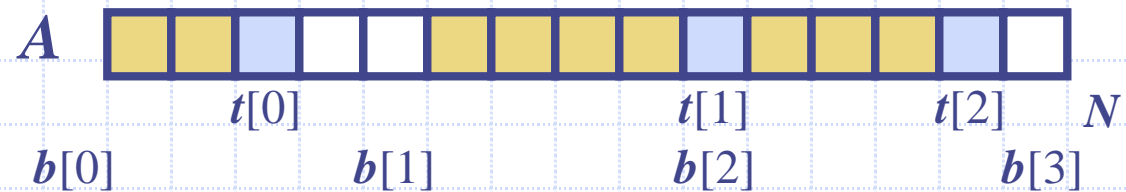
(a) 초기



(b) 평소



(c) 만원



$\text{push}(1, X) \Rightarrow \text{fullStackException}$

해결: 메소드

Alg *initMultiStack()*

input array A , t , b , size N ,
integer n

output multistack

1. $stacksize \leftarrow N / n$
2. for $i \leftarrow 0$ to $n - 1$
 $b[i] \leftarrow stacksize \times i - 1$
 $t[i] \leftarrow b[i]$
3. $b[n] \leftarrow N - 1$
4. return

Alg *size(i)*

1. return $t[i] - b[i]$

Alg *isEmpty(i)*

1. return $t[i] = b[i]$

Alg *isFull(i)*

1. return $t[i] = b[i + 1]$

Alg *top(i)*

1. if (*isEmpty(i)*)
 emptyStackException()
2. return $A[t[i]]$

해결: 메소드 (conti.)

Alg *push*(*i*, *e*)

input array *A*, *t*, *b*, stack *i*,
element *e*

output none

1. if (*isFull*(*i*))
 fullStackException(*i*)
2. $t[i] \leftarrow t[i] + 1$
3. $A[t[i]] \leftarrow e$
4. return

Alg *pop*(*i*)

input array *A*, *t*, *b*, stack *i*
output element

1. if (*isEmpty*(*i*))
 emptyStackException()
2. $t[i] \leftarrow t[i] - 1$
3. return $A[t[i] + 1]$

해결: fullStackException이 취할 수 있는 세 가지 전략

◆ 고정 베이스(fixed bases)

- 각 스택의 베이스 위치는 불변이므로 삽입처리 불가
- 장점: 단순
- 단점: 지역의 overflow는 전역적인 정지를 초래

◆ 부동 베이스(floating bases)

- 다른 스택으로부터 미사용 셀을 탈취함 – 이때 다른 스택들 중에는 가장 가까운 왼쪽 이웃을, 없으면 가장 가까운 오른쪽의 이웃을 선택
- 장점: 기억장소 활용 최대화
- 단점: overflow가 빈번한 경우 속도 저하

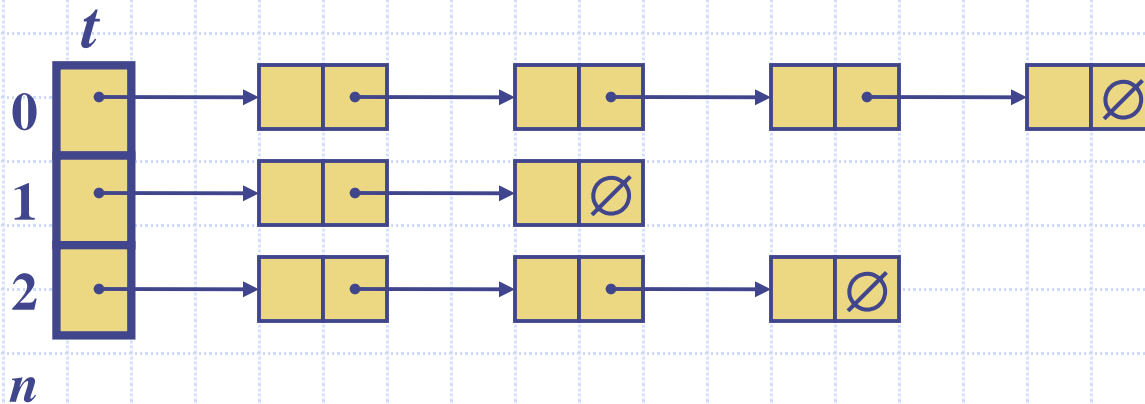
◆ 부동 베이스 + 기억장소 재할당(reallocation)

- 남은 기억장소 총량을 각 스택의 성장폭에 비례하여 재할당
- 장점: 기억장소 활용 최대화 및 overflow 재발 최소화
- 단점: 알고리즘 복잡하며, 미사용 기억장소 개수가 작으면 무의미

해결 B: 연결리스트의 배열 사용

- ◆ 각 스택에 대해 연결리스트를 사용하며, 추가로 각 스택의 top을 가리키는 포인터 배열 한 개 사용

예: $n = 3$



해결: 메소드

Alg *initMultiStack()*

input array t , integer n

output multistack

1. **for** $i \leftarrow 0$ **to** $n - 1$
 $t[i] \leftarrow \emptyset$
2. **return**

Alg *isEmpty(i)*

input array t , stack i

output boolean

1. **return** $t[i] = \emptyset$

Alg *top(i)*

input array t , stack i

output element

1. **if** ($isEmpty(i)$)
 $emptyStackException()$
2. **return** $t[i].elem$

해결: 메소드 (conti.)

Alg *push*(*i*, *e*)

input array *t*, stack *i*, element *e*

output none

1. $q \leftarrow \text{getnode}()$
2. $q.\text{elem} \leftarrow e$
3. $q.\text{next} \leftarrow t[i]$
4. $t[i] \leftarrow q$
5. **return**

Alg *pop*(*i*)

input array *t*, stack *i*

output element

1. **if** (*isEmpty*(*i*))
 emptyStackException()
2. $e \leftarrow t[i].\text{elem}$
3. $q \leftarrow t[i]$
4. $t[i] \leftarrow t[i].\text{next}$
5. *putnode*(*q*)
6. **return** *e*