

자료구조: 2022년 1학기 [강의]

스택을 이용한 수식 계산

강지훈

jhkang@cnu.ac.kr

충남대학교 컴퓨터융합학부



© J.-H. Kang, CNU

수식의 표기법



□ 수식의 표기법

- 연산자가 연산값 **사**이에
- 괄호가 필요

Infix : 2 + 5

Prefix : + 2 5

Postfix : 2 5 +

- 연산자가 연산값 **앞**에
- 괄호가 불필요

- 연산자가 연산값 **뒤**에
- 괄호가 불필요
- 컴파일러가 사용

□ 괄호가 필요한 수식: Infix

● **Infix** : $3 * (7 - 5)$

● **Postfix** : $3\ 7\ 5\ -\ *$

● **Prefix** : $*\ 3\ -\ 7\ 5$

$$\begin{aligned} & 3 * (7 - 5) \\ = & 3 * 2 \\ = & 6 \end{aligned}$$



□ 괄호가 필요 없는 수식: Postfix

● Infix : $3 * (7 - 5)$

● **Postfix** : $3\ 7\ 5\ -\ *$

● Prefix : $*\ 3\ -\ 7\ 5$

$3\ 7\ 5\ -\ *$
 $=\ 3\ 2\ *$
 $=\ 6$



□ 괄호가 필요 없는 수식: Prefix

● Infix : $3 * (7 - 5)$

● Postfix : $3 \ 7 \ 5 \ - \ *$

● Prefix : $* \ 3 \ - \ 7 \ 5$

$* \ 3 \ - \ 7 \ 5$
 $= \ * \ 3 \ 2$
 $= \ 6$



□ Infix 와 Postfix

■ 예제

Infix	Postfix
$2+3*4$	234^*+
$a*b+5$	ab^*5+
$(1+2)*7$	$12+7^*$
$(a/(b-c+d))*(e-a)*c$	$abc-d+ / ea-*c^*$
$a/b-c+d*e-a*c$	$ab/c-de^*+ac^*-$



후위 표기식의 계산



□ 후위 표기식의 계산

■ 스택을 사용

- 쉽고 효율적

■ 방법

- [Push] 연산자가 나타날 때까지 스택에 연산값을 삽입
- 연산자가 나타나면,
 - ◆ [Pop] 스택에서 연산자에 필요한 수 만큼 연산값을 삭제
 - ◆ 삭제한 연산값을 사용하여 연산자를 실행
 - ◆ [Push] 연산 결과를 스택에 삽입

□ 후위 표기식의 계산 알고리즘

```

Stack<Number> numberStack = new Stack() ;
Token token ;
Number operand1, operand2, result ;
token = postfix.nextToken() ;
while ( token.type() != TOKEN.EndOfExpression) {
    if ( token.type() == TOKEN.Operand ) {
        numberStack.push(token.value()) ;
    }
    else { // 연산자가 2 개의 연산값을 필요로 한다고 가정했음
        operand1 = numberStack.pop() ;
        operand2 = numberStack.pop() ;
        result = evalOperator (token.operator(), operand1, operand2) ;
        numberStack.push(result) ;
    }
    token = postfix.nextToken() ;
}
Number finalResult = numberStack .pop() ;

```

□ 예제

Infix: 6 / 2 - 3 + 4 * 2 \$

→ Postfix: 6 2 / 3 - 4 2 * + \$

Token	Stack				Top
	[0]	[1]	[2]	[3]	
					-1
6	6				0
2	6	2			1
/	3				0
3	3	3			1
-	0				0
4	0	4			1
2	0	4	2		2
*	0	8			1
+	8				0
\$					-1

Infix 를 Postfix 로 바꾸기



□ 왜 바꾸는가?

■ 사람은 infix 가 익숙하다

- 우리는 초등학교 시절부터 infix 를 사용했다.

■ 컴퓨터는 postfix 가 효율적이다

- 괄호가 필요 없다.
- 스택을 사용하여 간편하게 계산할 수 있다.

■ 그렇다면?

- 사람은 익숙한 infix 로 수식을 표현하고, 그 대신 계산의 사전 단계로 컴퓨터가 infix 를 postfix 로 변환한다.

□ Infix 를 Postfix로 변환하는 3 단계 방법

$a / b - c + d * e - a * c$



(1) 수식에 괄호를 완벽하게 친다

$(((a / b) - c) + (d * e)) - (a * c)$



(2) 연산자를 해당 오른쪽 괄호 위치로 옮긴다

$(((a b /) c -) (d e *) +) (a c *) -$



(3) 모든 괄호를 삭제한다.

$a b / c - d e * + a c * -$



정리하면

$a b / c - d e * + a c * -$

□ 스택을 이용한 수식 변환

- 수식을 왼쪽에서 오른쪽으로 스캔
- 연산값(operand)은 나타날 때마다 후위 표기식으로 출력
 - 연산값들 끼리의 순서는 바뀌지 않는다
- 연산자(operator)는 스택에 삽입
 - 삽입 전 할일: 현재 스택 안에 있는 연산자 중에서, 삽입되는 연산자보다 우선순위가 낮은 연산자들은 차례로 스택에서 빼내어, 후위 표기식으로 출력

□ 스택을 이용한 수식 변환

■ 남은 해결할 점은?

- 괄호는 어떻게?
- 연산자의 결합법칙은?

□ 예: 괄호가 없는 경우

Infix: 6 / 2 - 3 + 4 * 2 \$

→ Postfix: 6 2 / 3 - 4 2 * + \$

토큰	스택				Top	Postfix로 출력
	[0]	[1]	[2]	[3]		
					-1	
6					-1	6
/	/				0	6
2	/				0	6 2
-	-				1	6 2 /
3	-				1	6 2 / 3
+	+				1	6 2 / 3 -
4	+				1	6 2 / 3 - 4
*	+	*			2	6 2 / 3 - 4
2	+	*			2	6 2 / 3 - 4 2
\$					-1	6 2 / 3 - 4 2 * +

□ 괄호가 있는 수식의 변환

■ 요령

- 여는 괄호를 만나면 대응되는 닫는 괄호를 만날 때까지 그 괄호 안의 수식만 별도로 변환
- 토큰으로 여는 괄호를 만나면 무조건 스택에 삽입
 - 괄호 안의 수식을 처리하는 동안은, 잠정적으로 스택에 들어가 있는 여는 괄호가 스택의 바닥 (bottom)
- 닫는 괄호는 수식의 끝을 나타내는 토큰으로 간주
 - 닫는 괄호를 만나면 스택에서 여는 괄호를 만날 때까지 모든 연산자를 계속 빼내어 출력

예: 괄호가 있는 경우

Infix: $a * (b + c) * d \$$

→ Postfix: $a \ b \ c \ + \ * \ d \ * \ \$$

토큰	스택				Top	Postfix로 출력
	[0]	[1]	[2]	[3]		
					-1	
a					-1	a
*	*				0	a
(*	(1	a
b	*	(1	a b
+	*	(+		2	a b
c	*	(+		2	a b c
)	*				0	a b c +
*	*				0	a b c + *
d	*				0	a b c + * d
\$					-1	a b c + * d *

□ 괄호가 있는 수식의 변환

- 연산자 우선순위를 사용
 - 괄호도 연산자로 보고 역할에 맞는 우선순위를 갖게 한다.
- 왼쪽 괄호 '(' 를 만나면:
 - '(' 는 무조건 스택의 top에 삽입
 - 그러므로, 입력 토큰으로서의 '(' 의 우선순위는 **가장 높은** 순위를 갖게 설정한다
- 괄호 사이에 있는 연산자들을 만나면:
 - 이 연산자들은 '(' 위에 쌓여야 한다.
 - 그러므로, '(' 는 스택에 들어 있는 동안 **가장 낮은** 우선순위를 가져야 한다.
- ')' 를 만나면:
 - '(' 이후에 쌓인 모든 연산자를 삭제해야 한다.
 - ')' 는 연산자의 우선순위가 필요하지 않다.
 - ◆ 형식적으로 가장 높은 우선순위를 부여한다.

연산자	스택 안 우선순위	입력 토큰 우선순위
(0	20
)	19	19
+	12	12
-	12	12
*	13	13
/	13	13
%	13	13
\$	0	0

□ 결합 법칙 (Associativity)

- 괄호를 완벽하게 치면 결합법칙은 불필요
- 대부분의 연산자는 **왼쪽우선 (left-to-right)** 결합법칙을 가지고 있다.
 - $2+3+4 \Rightarrow ((2+3)+4)$
 - $9-4-3 \Rightarrow ((9-4)-3)$
 - $36/6/2 \Rightarrow ((36/6)/2)$

□ 오른쪽 우선 결합법칙

■ 그러나, **오른쪽 우선(right-to-left) 결합법칙**을 가지고 있는 연산자도 있다.

- (C 언어의) '=' (assignment) 연산자:
 - ◆ $a=b=c=0;$ ➡ $(a=(b=(c=0)))$;
- (Fortran 언어의) '**' (승) 연산자: $(a**x \rightarrow a^x)$
 - ◆ $2**3**2 \rightarrow (2**(3**2))$
 - ◆ $(2**(3**2))$ 는 512, 그러나 $((2**3)**2)$ 는 64

■ Infix 를 postfix 로

- 오른쪽 우선 : $2 \wedge 3 \wedge 2 \rightarrow 2 \ 3 \ 2 \wedge \wedge$
 - ◆ (^ 는 ** 대신 사용했음)
- 왼쪽 우선: $2 + 3 + 2 \rightarrow 2 \ 3 + 2 +$

□ 예: 오른쪽 우선 결합법칙

Infix: 2 ^ 3 ^ 2 \$

→ Postfix: 2 3 2 ^ ^ \$

토큰	스택			Top	Postfix로 출력
	[0]	[1]	[2]		
				-1	
2				-1	2
^	^			0	2
3	^			1	2 3
^	^	^		1	2 3
2	^	^		2	2 3 2
\$				-1	2 3 2 ^ ^

■ 중요한 관찰!

- '^'는 먼저 삽입된 '^' 과 동일한 연산자이지만, '^' 는 스택에 삽입되어야 한다.
- 그러므로, 입력 토큰으로서 연산자 '^'의 우선순위는 스택 안 (in-stack)에 있는 연산자 '^'의 우선순위보다 높아야 한다.

□ 예: 우선순위 테이블

- 오른쪽우선 (Right-to-Left) 결합법칙의 연산자:
입력 토큰 우선순위는 **스택 안** 우선순위보다 **높아야** 한다.
- 왼쪽우선 (Left-to-Right) 결합법칙의 연산자:
두 우선순위 값이 **같아야** 한다.

Operator	스택 안 우선순위	입력 토큰 우선순위
(0	20
)	19	19
+	12	12
-	12	12
*	13	13
/	13	13
%	13	13
^	16	17
\$	0	0

□ 복잡도 분석

- 주어진 Infix 를 Postfix 로 바꾸어 계산
- n : 수식에서 토큰의 개수
 - infixToPostfix: $O(n)$
 - evalPostfix: $O(n)$



End of “Expression Calculation”



