제 5 강의 . 스택과 큐의 응용

학습 목차

1. 후위표기법
2. 스택을 이용한 후위표기법 변환
3. 스택을 이용한 후위표기법의 계산

1. 후위 표기법

(정의) 후위표기법(postfix notation) : 후위표기법은 연산자를 피연산자의 뒤에 놓는 방법이다. 스택의 응용의 예이며 수식의 계산은 계산기에서나 컴퓨터 프로그래밍을 할 때 자주 나타난다.

x = a/b-c+d*e-a*c

다음의 수식을 사람이 계산한다고 할 때 계산하는 과정을 살펴보자.

(중간결과는 temp에 기록한다)

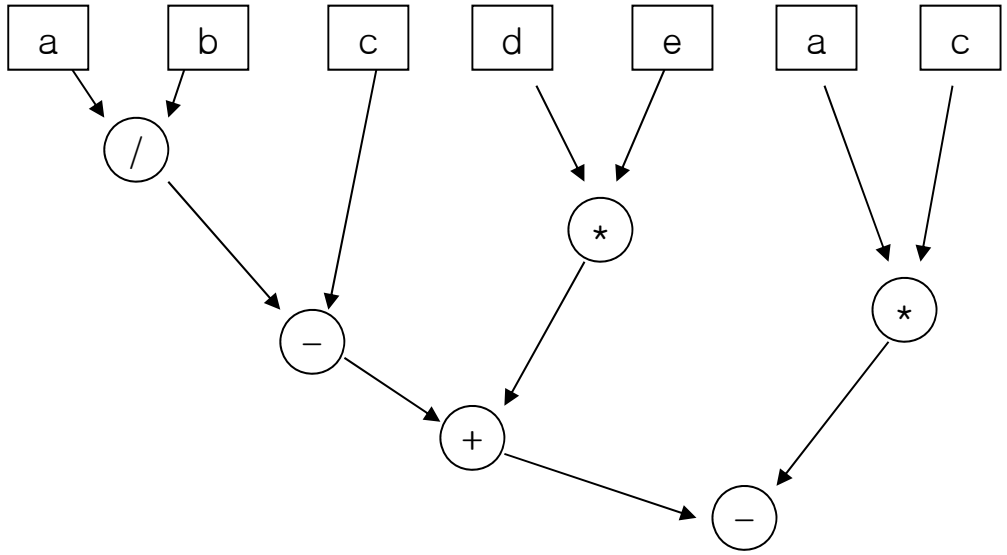
x = (a/b)-c+d*e-a*c
= (temp1)-c+d*e-a*c
= (temp1)-c+(temp2)-a*c
= (temp1)-c+(temp2)-(temp3)
= (temp4)+(temp2)-(temp3)
= (temp5)-(temp3)
= (temp6)

위처럼 계산하는 이유는 *, / 연산자가 +.- 연산자보다 먼저 계산해야 하고 또 연산자 *, / 나 +,- 가 동시에 있으면 왼쪽에 있는 식부터 계산해야 하는 것을 알고 있기 때문이다

사람은 아래식을 마음속에 다음과 같이 계산을 하게 된다.

안쪽 괄호부터 계산을 해 나간다.

x = ((((a/b)-c)+(d*e))-(a*c))



그런데 컴퓨터에 어떻게 이러한 사실을 알려서 계산을 하게 될까요?

1) 수식 계산 – 사람과 컴퓨터

- **사람**

1) 연산자의 우선 순위를 정한다. 우선순위가 같으면 왼쪽부터인지 오른쪽부터인지 정한다.

(/,*.+,- 연산자는 왼쪽부터이지만, 지수 연산자는 오른쪽 부터이다.)

2) 복잡하면 괄호를 사용하여 계산하며 중간 결과를 저장한다.

(((A/(B**C))+(D*E))-(A*C))

- **컴퓨터로 수식의 계산**

사람이 하는 방법대로 계산할 수도 있지만 연구 결과 중간 과정을 줄이고 한번에 왼쪽에서 오른쪽으로 계산할 수 있는 방법을 개발하였다.

1) 수식을 후위 표기법(postfix notation)으로 바꾼다.

2) 후위식을 계산한다. (후위표기법으로 표기한 식을 후위식이라 하자)

☄ 연산자의 우선 순위(precedence) 표 보기 – C 언어

연산자	우선순위	결합성(associativity)
() [] -> .	17	left-to-right
-- ++	16	left-to-right
-- ++ ! ~ - + & * sizeof	15	right-to-left
(type)	14	right-to-left
* / %	13	left-to-right
+ -	12	left-to-right
<< >>	11	left-to-right
> >= < <=	10	left-to-right
== !=	9	left-to-right
&	8	left-to-right
^	7	left-to-right
|	6	left-to-right
&&	5	left-to-right
||	4	left-to-right
?:	3	right-to-left
= += -= /= *= %= <<= >>= &= ^= |=	2	right-to-left
,	1	left-to-right

수식 계산 – 컴퓨터

(1) 수식을 후위 표기법(postfix notation)으로 바꾼다.

사람이 사용하는 수식의 표기 방법은 중위(infix) 표기법이라고 한다.

중위표기법은 연산자(operator)를 피연산자(operand) 가운데 놓는 방법이다.

3 * 5 => (피연산자) (연산자) (피연산자)

후위표기법은 연산자를 피연산자 뒤에 놓고 표기하는 방법이다.

3 5 * => (피연산자) (피연산자) (연산자)

후위표기법으로 놓으면 괄호 없이도 계산 순서가 일정하다.

예 1) 3 * 5 + 4

중위 표기법 -> 3 * 5 + 4

후위 표기법 -> 3 5 * 4 +

예 2) 3 * (5 + 4)

중위 표기법 -> 3 * (5 + 4) => 괄호가 필요하다.(5+4를 먼저 계산하려면)

후위 표기법 -> 3 5 4 + * => 괄호가 필요 없다.

* 괄호가 필요 없으면 계산 순서를 생각할 필요가 없어서 편하다.

(2) 후위식을 계산한다.

후위식은 중위식과 계산 방법이 다르다. 그렇지만 중위식처럼 식의 왼쪽, 오른쪽 우선순위에 따라 옮겨 다니지 않고 왼쪽부터 기계적으로 계산을 하면 된다.

예를 들어 3 * 5 + 4 = 3 5 * 4 + 식은

1단계) 3 5 * 4 +

2단계) 15 4 + => 3 5 * 를 계산하여 저장한다.

3단계) 19 => 15 4 + 를 계산한다.

다른 예로 3 * (5 + 4) = 3 5 4 + *

1단계) 3 5 4 + *

2단계) 3 9 * => 5 4 + 를 계산한다.

3단계) 27 => 3 9 * 를 계산한다.

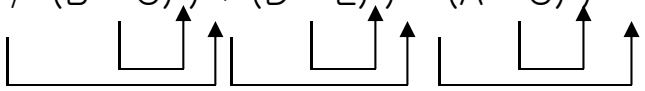
계산하는 방법은 연산자가 나올 때까지 읽어서 연산자의 앞에 있는 피연산자 두개를 이용하여 계산하고 그 자리에 저장한다.

2) 중위표기를 후위표기로 바꾸기

- 중위식을 후위식으로 쉽게 바꾸는 방법

  중위식에 괄호를 친 다음 연산자를 괄호 뒤로 옮긴 후 괄호를 지운다.

예) (((A / (B ^ C)) + (D * E)) - (A * C))



=> A B C ^ / D E * + A C * -

중위표기(infix)	후위표기(postfix)
2 + 3 * 4	2 3 4 * +
a * b + 5	a b * 5 +
(1 + 2) * 7	1 2 + 7 *
a*b/c	a b * c /
(a / (b – c + d)) * (e - a) * c	a b c – d + / e a - * c *
a / b – c + d * e – a * c	a b / c – d e * + a c * -

Q/A

1. 중위식을 후위식으로 바꾸어라.
   (1) A / B * (C + D) + E
   (2) (((A /B) + C) – (D * E))

2. 후위식을 중위식으로 바꾸어라.
   (1) B C – D * E +
   (2) A B / C D + * E +

2. 스택을 이용한 후위표기법 변환

중위표기를 후위표기로 바꾸는 알고리즘은 다음과 같다. 수식을 읽어 연산자와 피연산자에 따라 다음과 같이 처리한다.

(1) 피연산자는 출력한다.

(2) 연산자는 앞 연산자(스택의 맨 위)를 살펴서 출력하거나 대기한다(스택에 넣는다, 대기 된 자료들은 나중에 대기 된 연산자가 먼저 나온다, LIFO, 스택을 이용)

(3) 연산자의 대기(스택에 push)여부는 연산자간의 우선순위에 따른다.

예) 후위표기 변환 예 (1)

수식 " a+b*c "를 후위식(postfix)으로 변환

-> " abc*+ "

입력 단계별 , 스택, 출력 결과

a+b*c

토큰(입력)	스택의 모양	top 변수의 값	출력
a		-1	a

⇩

토큰	스택의 모양	top 변수의 값	출력
+	+	0	a

⇩

토큰	스택의 모양	top 변수의 값	출력
b	+	0	a b

⇩

토큰	스택의 모양	top 변수의 값	출력
*	+ *	1	a b

⇩

토큰	스택의 모양	top 변수의 값	출력
c	+ *	1	a b c

⇩

토큰	스택의 모양	top 변수의 값	출력
eos(끝)		-1	a b c * +

예) 후위표기 변환 예 (2)

괄호(parenthesis)가 있는 경우

중위표기법에 괄호가 있는 경우는 변환과정을 복잡하게 한다.

-후위표기법으로 표기하면 괄호가 없어진다.

왼쪽괄호 - 무조건 스택에 넣는다.

오른쪽 괄호의 처리 - 왼쪽 괄호가 나올 때까지 스택에서 pop 한다.

예) 수식 " a*(b+c)*d " -> 후위표기법으로 변환후 " abc+*d* "

a*(b+c)*d

토큰	스택의 모양	top 변수의 값	출력
a		-1	a

⇩

토큰	스택의 모양	top 변수의 값	출력
*	*	0	a

⇩

토큰	스택의 모양	top 변수의 값	출력
(	* (	1	a

⇩

a*(b+c)*d

토큰	스택의 모양	top 변수의 값	출력
b	* (	1	a b

⇩

토큰	스택의 모양	top 변수의 값	출력
+	* (+	2	a b

⇩

토큰	스택의 모양	top 변수의 값	출력
c	* (+	2	a b c

⇩

토큰	스택의 모양	top 변수의 값	출력
)	*	0	a b c +

⇩

토큰	스택의 모양	top 변수의 값	출력
*	*	0	a b c + *

⇩

토큰	스택의 모양	top 변수의 값	출력
d	*	0	a b c + * d

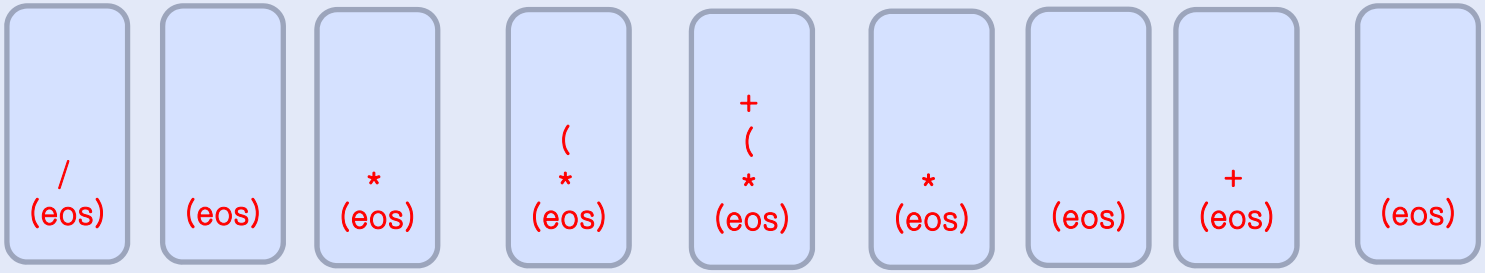
⇩

토큰	스택의 모양	top 변수의 값	출력
eos		-1	a b c + * d *

Q/A

1. 다음식을 후위식으로 바꿀때 스택의 변화를 적어라.

(1) A / B * (C + D) + E



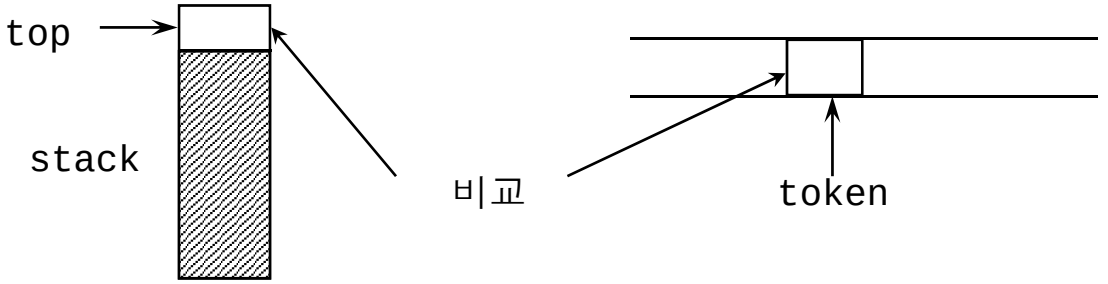
(2) 3 + 2 * 4

(3) ((A /B) + C) – (D * E)

(알고리즘) – 연산자 입력시 스택 처리

•연산자가 입력되었을 때 우선순위 비교를 통한 연산자의 push(), pop() 결정

연산자 우선순위를 2가지로 만든다
– in-stack precedence(isp), incoming precedence(icp)



"in-stack precedence"
스택에 있는 연산자의 우선순위

"in-comming precedence"
입력될 때 연산자의 우선순위

If (isp[stack[top]] < icp[token])
/* 입력된 연산자가 스택의 맨 위 연산자보다 우선순위가 클 경우) */
-> push()

If (isp[stack[top]] ≥ icp[token])
/* 입력된 연산자가 스택의 맨 위 연산자보다 작거나 같을 경우) */
-> pop() until (isp[stack[top]] < icp[token])

프로그램을 위한 선언 부분

선언 부

```
/* 수식 계산을 위한 프로그램 선언 부분 */
#define MAX_STACK_SIZE 100
/* maximum stack size */
#define MAX_EXPR_SIZE 100
/* max size of expression */

/* 열거형 타입 precedence 선언 */
typedef enum {lparen, rparen, plus, minus,
times, divide, mode, eos, operand
} precedence;

precedence stack[MAX_STACK_SIZE];/* 스택선언 */
char expr[MAX_EXPR_SIZE]; /* 입력 문자열 */

static int isp[] = {0,19,12,12,13,13,13,0};
static int icp[] = {20,19,12,12,13,13,13,0};
```

get_token 함수

```
/* 입력에서 토큰을 받아들이는 프로그램 */
precedence get_token(char *symbol, int *n) {
    *symbol = expr[(*n)++];
    switch(*symbol) {
    case '(': return lparen;
    case ')': return rparen;
    case '+': return plus;
    case '-': return minus;
    case '/': return divide;
    case '*': return times;
    case '%': return mod;
    case ' ': return eos;
    default: return operand;
    }
}
```

postfix 함수

```
/* 중위표기를 후위표기로 바꾸는 프로그램 */
void postfix(void) {
    char symbol;
    precedence token;
    int n = 0;
    int top = 0;
    stack[0] = eos;
    for(token = get_token(&symbol, &n); token != eos;
        token = get_token(&symbol, &n)) {
        if(token == operand) printf("%c", symbol);
        else if(token == rparen) {
            while(stack[top] != lparen)
                print_token(pop());
            pop();
        }
        else {
            while(isp[stack[top]] >= icp[token])
                print_token(pop());
            push(token);
        }
    }
    while((token = pop()) != eos)
        print_token(token);
    printf("\n");
}
```

3. 스택을 이용한 후위표기식 계산

후위표기(postfix)식의 계산

- 괄호(parenthesis)가 필요 없다
- 연산자 우선순위(precedence)가 필요 없다

프로그램 복잡도

- 시간 : O(r)
  r : 수식의 기호의 개수
- 기억공간 : S(n) = n
  n : 연산자의 수

예) 6 2 / 3 - 4 2 * +

토큰	스택의 모양	top 변수의 값
6	6	0

토큰	스택의 모양	top 변수의 값
2	6 2	1

토큰	스택의 모양	top 변수의 값
/	3	0

토큰	스택의 모양	top 변수의 값
3	3 3	1

예) 6 2 / 3 - 4 2 * +

토큰	스택의 모양	top 변수의 값
–	0	0

토큰	스택의 모양	top 변수의 값
4	0 4	1

토큰	스택의 모양	top 변수의 값
2	0 4 2	2

토큰	스택의 모양	top 변수의 값
*	0 8	1

토큰	스택의 모양	top 변수의 값
+	8	0

Q/A

1. 후위식을 계산할 때 스택의 변화를 적어라.

(1) 1 2 + 7 *

(2) 3 3 / 4 - 5 6 * + 3 4 * -

eval() 함수

get_token() 함수

/* 수식에서 토큰을 한 개씩 읽는 프로그램 */

```
eval()
{
   if the token is operand
         convert to number and push to the stack
   otherwise
         1) pop two operands from the stack
         2) perform the specified operation
         3) push the result back on the stack
   }
```

eval 함수

```
/* 후위표기식을 계산하는 프로그램 */
int eval(void)
{
     precedence token;
     char symbol;
     int op1, op2;
     int n = 0;
     int top = -1;
     token = get_token(&symbol, &n);
     while (token != eos) {
          if(token == operand)
               push(symbol-'0');
          else {
               op2 = pop();
               op1 = pop();
               switch(token) {
               case plus: push(op1 + op2);
                                   break;
(계속 …)
```

eval 함수

```
                    case minus: push(op1 - op2);
                                break;
                    case times: push(op1 * op2);
                                break;
                    case divide: push(op1 / op2);
                                break;
                    case mod: push(op1 % op2);
                    }
                }
                token = get_token(&symbol, &n);
            }
            return pop();
        }
```

Review

⦿ 스택은 리스트의 특별한 형태로 삽입과 삭제가 한쪽 끝에서만 일어난다. 마치 문이 한 개인 관광버스에 손님들을 타고 내릴 때, 먼저 탄 사람이 맨 나중에 내리는 것과 같은 구조이다(First In Last Out). 다시 말하면 맨 나중에 탄 사람이 맨 먼저 내리는 구조이다(Last in First Out, LIFO).

⦿ 큐 또한 리스트의 특별한 형태로 삽입과 삭제가 한쪽 끝에서만 일어난다. 삽입과 삭제는 서로 반대쪽에서 일어나는 것이 스택과 다르다. 문이 두개인 마을버스의 타는 곳과 내리는 곳이 따로 있어서 먼저 탄 사람이 먼저 내리는 구조이다(First In First Out, FIFO).

⦿ 컴퓨터에서 수식의 계산은 (1) 수식을 후위표기식으로 바꾸고, (2) 후위표기식을 계산하는 두 과정으로 이루어진다. 두 과정 모두 스택 자료구조를 필요로 한다.