자료구조 Data Structure | 조행래

스택과 큐

스택과 큐의 응용(2): 수식 계산

학습 목표

- 수식의 표현 방법을 이해한다.
- 스택을 이용하여 후위 표기식을 계산하는 알고리즘을 구현할 수 있다.
- 스택을 이용하여 중위 표기식에서 후위 표기식으로 변환하는 알고리즘을 구현할 수 있다.

1. 수식의 정의

- **수식이란?**
  - 연산자와 피연산자, 괄호 등으로 구성된 문자열
  - 예:
    - x = a / b – c + d * e – a * c
    - x = ((a / (b – c + d)) * (e – a) * c

- **연산자의 우선 순위(Precedence)**
  - 서로 다른 연산자의 연산 순서를 정의
  - 예: c + d * e

- **연산자의 결합성(Associativity)**
  - 동일한 우선순위를 갖는 연산자들간의 실행 순서
  - 예: b – c + d, a = b = c

2. 수식의 표현 방법

- **중위 표기법과 후위 표기법**
  - 중위 표기법(Infix notation)
    - 피연산자들 사이에 연산자가 위치
    - 괄호를 포함할 수 있고, 계산 과정이 복잡
  - 후위 표기법(Postfix notation)
    - 피연산자들 다음에 연산자가 위치
    - 괄호가 필요없고, 한번의 스캔으로 수식 계산 가능

중위 표기	후위 표기
2 + 3	2 3 +
a * b + 5	a b * 5 +
a + b * 5	a b 5 * +
(1 + 2) * 7	1 2 + 7 *
((a / (b – c + d)) * (e – a) * c	a b c – d + / e a - * c *

3. 후위 표기식의 계산

- **스택을 이용**
  - 피연산자는 스택에 저장
  - 연산자의 경우, 스택에서 피연산자 pop & 결과를 push
- **예: 6 2 / 3 - 4 2 * +**

Token	Stack[0]	Stack[1]	Stack[2]	Top
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

후위 표기식의 계산 알고리즘

```
// 토큰: enum { lparen, rparen, plus, minus, times, divide, mod, eos, operand }
token = get_token(&symbol,  &n);      // 수식에서 토큰을 하나씩 검사
while (token  !=  eos)  {
    if (token  == operand)                // 피연산자를 만나면 스택에 저장
        push(symbol – '0');
    else  {     // 연산자를 만나면 스택에서 피연산자 2개를 제거한 후 결과 저장
        op2  =  pop();     // 피연산자 제거
        op1  =  pop();     // 피연산자 제거
        switch ( token )    {
            case  plus     :   push(op1 + op2);    break;     // 결과 저장
            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();        //   return  result
```

후위 표기식의 계산 알고리즘 (계속)

```
precedence get_token (char *symbol, int * n)
{   //  수식 문자열에서 다음 문자를 검사하여 해당 token을 반환

    *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;   // 피연산자
    }
}
```

4. 중위 표기에서 후위 표기로의 변환

- **방법 1: 괄호를 이용하여 변환**
  - 예: a / b – c + d * e – a * c
    - ((((a / b) – c) + (d * e)) – (a * c))
    - a b / c – d e * + a c * –
  - 두 단계 처리로 인해 비효율적임

- **방법 2: Stack을 이용하여 변환**
  - 연산자들의 우선순위를 이용하여 변환
  - 우선순위(top) < 우선순위(incoming): 입력 연산자를 스택에 저장
  - 우선순위(top) > 우선순위(incoming): 스택 top에 있는 연산자를 출력
  - 우선순위(top) = 우선순위(incoming): 결합성에 따라 처리
  - 괄호가 있는 수식 처리에 주의

수식 변환의 예(1)

Token	Stack [0]	[1]	[2]	Top	Output
a				-1	a
+	+			0	a
b	+			0	ab
*	+	*		1	ab
c	+	*		1	abc
eos				-1	abc*+

Token	Stack [0]	[1]	[2]	Top	Output
a				-1	a
*	*			0	a
b	*			0	ab
+	+			1	ab*
c	+			1	ab*c
eos				-1	ab*c+

수식 변환의 예(2)

Token	Stack [0]	[1]	[2]	Top	Output
a				-1	a
*	*			0	a
(	*	(		1	a
b	*	(		1	ab
+	*	(	+	2	ab
c	*	(	+	2	abc
)	*			0	abc +
*	*			0	abc + *
d	*			0	abc + * d
eos	*			0	abc + * d *

입력 연산자 (: 우선 순위가 가장 높다.
스택 top에서의 (: 우선 순위가 가장 낮다.
입력 연산자) : (를 만날 때까지 스택에 있는 연산자들을 pop

5. 수식 변환 알고리즘

- **연산자를 위한 스택 필요**
- **괄호 연산자를 처리하기 위한 두 종류의 우선순위**
  - int isp[]: stack에 저장된 연산자의 우선순위
  - int icp[]: 입력 연산자의 우선순위

```
typedef enum { lparen, rparen, plus, minus, times,
               divide, mod, eos, operand } precedence;
precedence   stack[ MAX_STACK_SIZE ];
/* isp와 icp 배열 – 연산자들의 순서
   lparen, rparen, plus, minus, times, divide, mod, eos */
int    isp[ ] = { 0, 19, 12, 12, 13, 13, 13, 0 };
int    icp[ ] = { 20, 19, 12, 12, 13, 13, 13, 0 };
```

수식 변환 알고리즘

```
void postfix ( void )
{  // 수식 변환 프로그램. (infix → postfix)
   // expr[](infix 저장)과 stack[], 그리고 top은 전역변수

   char symbol;
   precedence token;
   int n = 0;
   top = -1;

   push(eos);          // eos를 스택에 삽입
   for (token = get_token(&symbol, &n); token != eos;
                  token = get_token(&symbol, &n)) {
        if (token == operand)      printf("%c", symbol);
```

수식 변환 알고리즘 (계속)

```
    else if (token == rparen) {
    // 왼쪽 괄호가 나올 때까지 stack pop
        while (stack[top] != lparen)
            print_token(pop());
        pop();  // 왼쪽 괄호 제거
    }
    else { // 우선순위가 높은 연산자 pop. associativity?
        while (isp[stack[top]] >= icp[token])
            print_token(pop());
        push(token);
    }
}
while ( (token = pop() ) != eos)
    print_token(token);    // stack의 모든 연산자 출력
printf("₩n");
```

요약 정리

- 수식의 표현 방법: 중위 표기와 후위 표기
- 후위 표기식의 계산 알고리즘을 설명
- 중위 표기식에서 후위 표기식으로 변환하는 알고리즘을 설명