실습 과제 (Final)

(/* 보너스? 4: 0; */)

[Q] 중위/후위표기수식 변환과 계산

중위 표기법 수식을 입력 받아, 토큰으로 분리하고, 후기 표기 수식으로 변환하고, 변수는 값이 저장되어 있는 심볼테이블을 참고하여, 수식을 계산한 값을 출력하는 프로그램을 작성한다. (※ 참조: Project 24 & 25 중위/후기표기법, Project 26. 유한상태기)

1) 변수들의 이름과 값이 들어 있는 파일, '**inputSymbol.txt**'를 읽어 Symbol Table을 만들고 변수 별 값을 출력한다. <1점>
2) 파일, '**inputInfix.txt**'에는 중위표기수식이 한 줄에 하나씩 적혀 있다. 모든 수식을 한 줄씩 문자열로 가져와 아래와 같이 처리한다..
3) 중위표기수식 문자열을 토큰으로 분리하고, 토큰 별, 토큰의 문자(열)과 토큰의 우선순위의 인덱스를 출력한다. 피연산자인 경우 값도 같이 출력한다. <1점>
4) 토큰 형태로 구성된 중위표기수식을 출력하고, 이를 후기표기수식으로 변환하여 출력한다. 각 토큰의 심볼의 표시 문자(열)은 중괄호로 묶어 출력하고, 후기표시수식 출력 시 변수는 그 값을 출력한다. <1점>
5) 변환된 후기표기수식을 계산하여 그 결과 값을 출력한다. <1점>

※ 입/출력 실행 예시 참조

✧ 제한사항

- 중위표기수식은 피연산자, 연산자, 왼쪽 소괄호, 오른쪽 소괄호로 구성된다.
- 중위표기수식은 구두점, Semicolon('**:**') 또는 Newline('**₩n**') 문자로 끝난다.
- 피연산자는, 자연수 숫자(numeric)와 변수(영어 대소문자만으로 구성되는 symbol)로 구성되고, 연산자로는 2항산술연산자('**+**', '**-**', '*****', '**/**', '**%**'), 구두점('**;**')과 소괄호('**(**', '**)**'), 그리고 토큰 분리문자(' ', '**₩t**', '**₩n**') 등을 사용한다.
- 연산자 우선순위와 결합법칙
  - ✓ 연산자 중, 곱하기(*), 나누기(/), 나머지(%) 등은 같은 우선 순위를 가지며,, 2항연산자 더하기(+)와 빼기(-)도 같은 우선순위를 가진다. 모든 연산자는 왼쪽에서 오른쪽으로 결합법칙이 적용된다.. (1항 연산자, +와 -는 사용하지 않는다.)
  - ✓ 왼쪽 소괄호는 처음엔 (※스택에 들어 오기 전) 모든 연산자보다 높은 우선순위를 가지지만, 짝이 되는 오른쪽 소괄호를 만날 때까지는 (※스택 안에서) 가장 낮은 우선순위를 가진다.
  - ✓ 곱하기(*), 나누기(/), 나머지(%) 등은 더하기(+)와 빼기(-)보다 높은 우선 순위를 가진다.

✧ Q2-입/출력 실행 예시

```
*** Symbols
--------------------
         I      1
        II      2
       III      3
        IV      4
         V      5
        IX      9
         X     10
        XL     40
         L     50
        XC     90
         C    100
        CD    400
         D    500
        CM    900
         M   1000
--------------------
```

inputSymbols.txt

```
I 1
II 2
III 3
IV 4
V 5
IX 9
X 10
XL 40
L 50
XC 90
C 100
CD 400
D 500
CM 900
M 1000
```

inputInfix.txt

```
(D - CD) * X + (XC + V) / (L - IX * V) + M
2 + 3 * 4 - 12 / 6;
12 + III * (20 - 14) - CM/L
V+X+L+C+I+V+I+C+L+X+V;
CM-6*L*II+IX-7%V+3*X
```

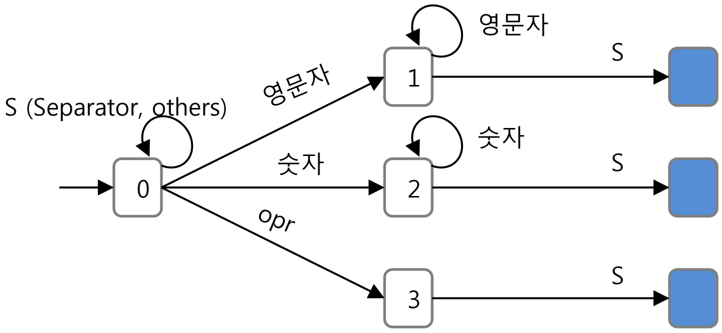
```
*** Tokens in  (D - CD) * X + (XC + V) / (L - IX * V) + M
-----------------------------------------------
Token :   (              precedence =   0
Token :   D =   500      precedence =  -1
Token :   -              precedence =   3
Token :  CD =   400      precedence =  -1
Token :   )              precedence =   1
Token :   *              precedence =   4
Token :   X =    10      precedence =  -1
Token :   +              precedence =   2
Token :   (              precedence =   0
Token :  XC =    90      precedence =  -1
Token :   +              precedence =   2
Token :   V =     5      precedence =  -1
Token :   )              precedence =   1
Token :   /              precedence =   5
Token :   (              precedence =   0
Token :   L =    50      precedence =  -1
Token :   -              precedence =   3
Token :  IX =     9      precedence =  -1
Token :   *              precedence =   4
Token :   V =     5      precedence =  -1
Token :   )              precedence =   1
Token :   +              precedence =   2
Token :   M =  1000      precedence =  -1
---------------------------------------
* Infix expression:
[(] [D] [-] [CD] [)] [*] [X] [+] [(] [XC] [+] [V] [)] [/] [(] [L] [-] [IX] [*] [V] [)] [+] [M]
---------------------------------------
* Postfix expression:
[500] [400] [-] [10] [*] [90] [5] [+] [50] [9] [5] [*] [-] [/] [+] [1000] [+]
* Evaluated => 2019
-----------------------------------------------
```

```
*** Tokens in  2 + 3 * 4 - 12 / 6;
-------------------------------------------------
Token :   2 =     2     precedence =  -1
Token :   +             precedence =   2
Token :   3 =     3     precedence =  -1
Token :   *             precedence =   4
Token :   4 =     4     precedence =  -1
Token :   -             precedence =   3
Token :  12 =    12     precedence =  -1
Token :   /             precedence =   5
Token :   6 =     6     precedence =  -1
Token :   ;             precedence =   7
-----------------------------------------
* Infix expression:
[2] [+] [3] [*] [4] [-] [12] [/] [6] [;]
-----------------------------------------
* Postfix expression:
[2] [3] [4] [*] [+] [12] [6] [/] [-]
* Evaluated => 12
-------------------------------------------------
*** Tokens in  12 + III * (20 - 14) - CM/L
-------------------------------------------------
Token :  12 =    12     precedence =  -1
Token :   +             precedence =   2
Token : III =     3     precedence =  -1
Token :   *             precedence =   4
Token :   (             precedence =   0
Token :  20 =    20     precedence =  -1
Token :   -             precedence =   3
Token :  14 =    14     precedence =  -1
Token :   )             precedence =   1
Token :   -             precedence =   3
Token :  CM =   900     precedence =  -1
Token :   /             precedence =   5
Token :   L =    50     precedence =  -1
-----------------------------------------
* Infix expression:
[12] [+] [III] [*] [(] [20] [-] [14] [)] [-] [CM] [/] [L]
-----------------------------------------
* Postfix expression:
[12] [3] [20] [14] [-] [*] [+] [900] [50] [/] [-]
* Evaluated => 12
-------------------------------------------------
```

```
*** Tokens in  V+X+L+C+I+V+I+C+L+X+V;
-------------------------------------------------
Token :   V =     5     precedence =  -1
Token :   +             precedence =   2
Token :   X =    10     precedence =  -1
Token :   +             precedence =   2
Token :   L =    50     precedence =  -1
Token :   +             precedence =   2
Token :   C =   100     precedence =  -1
Token :   +             precedence =   2
Token :   I =     1     precedence =  -1
Token :   +             precedence =   2
Token :   V =     5     precedence =  -1
Token :   +             precedence =   2
Token :   I =     1     precedence =  -1
Token :   +             precedence =   2
Token :   C =   100     precedence =  -1
Token :   +             precedence =   2
Token :   L =    50     precedence =  -1
Token :   +             precedence =   2
Token :   X =    10     precedence =  -1
Token :   +             precedence =   2
Token :   V =     5     precedence =  -1
Token :   ;             precedence =   7
-----------------------------------------
* Infix expression:
[V] [+] [X] [+] [L] [+] [C] [+] [I] [+] [V] [+] [I] [+] [C] [+] [L] [+] [X] [+] [V] [;]
-----------------------------------------
* Postfix expression:
[5] [10] [+] [50] [+] [100] [+] [1] [+] [5] [+] [1] [+] [100] [+] [50] [+] [10] [+] [5] [+]
* Evaluated => 337
-------------------------------------------------
*** Tokens in  CM-6*L*II+IX-7%V+3*X
-------------------------------------------------
Token :  CM =   900     precedence =  -1
Token :   -             precedence =   3
Token :   6 =     6     precedence =  -1
Token :   *             precedence =   4
Token :   L =    50     precedence =  -1
Token :   *             precedence =   4
Token :  II =     2     precedence =  -1
Token :   +             precedence =   2
Token :  IX =     9     precedence =  -1
Token :   -             precedence =   3
Token :   7 =     7     precedence =  -1
Token :   %             precedence =   6
Token :   V =     5     precedence =  -1
Token :   +             precedence =   2
Token :   3 =     3     precedence =  -1
Token :   *             precedence =   4
Token :   X =    10     precedence =  -1
-----------------------------------------
* Infix expression:
[CM] [-] [6] [*] [L] [*] [II] [+] [IX] [-] [7] [%] [V] [+] [3] [*] [X]
-----------------------------------------
* Postfix expression:
[900] [6] [50] [*] [2] [*] [-] [9] [+] [7] [5] [%] [-] [3] [10] [*] [+]
* Evaluated => 337
-------------------------------------------------
```

부록: 참고 자료

```
/*** scanner.h  ***/
#define  CHAR_SPACE     ' '
#define  CHAR_TAB       '\t'
#define  CHAR_NEWLINE   '\n'

#define  SYMADD          '+'
#define  SYMSUB          '-'
#define  SYMMUL          '*'
#define  SYMDIV          '/'
#define  SYMMOD          '%'
#define  SYMLPA          '('
#define  SYMRPA          ')'
#define  SYMEOS          ';'

#define  S_SIZE      4
#define  S_START     0
#define  S_SYMBOL    1
#define  S_NUMBER    2
#define  S_OPERATOR  3

#define  TOKEN_NUM     5
#define  NUMERIC       0
#define  ALFA          1
#define  OPERATOR      2
#define  SEPARATOR     3
#define  OTHERS        4

typedef enum {
        lparen, rparen, plus, minus, times, divide, mod, eos, operand = -1
} precedence;

typedef struct {
        char sym[10];
        precedence pre;
        int type;
        int val;
} TokenType;

precedence       getPrecedence(char * str);
void             printToken( int flag, TokenType token);
int              unFSM(TokenType *token, char *infix);
```




```
int FSM[S_SIZE][TOKEN_NUM] = {
                /*    num          alfa        operator       sep         others    */
/* S_START */    { S_NUMBER,  S_SYMBOL,  S_OPERATOR,  S_START,    S_START   },
/* S_SYMBOL */   {     -1,     S_SYMBOL,       -1,         -1,          -1     },
/* S_NUMBER */   { S_NUMBER,       -1,          -1,         -1,          -1     },
/* S_OPERATOR */{     -1,          -1,          -1,         -1,          -1     }};
```

```
/*** stack.h  ***/

#include  <ctype.h>
#include "scanner.h"
void push(TokenType
TokenType  * pop();
TokenType stackTop();/* element at stack top */
```

```
/*** symbolTable.h  ***/

void     PrintSymbols (void);
int      FindSymbol (char name[]);
int      GetSymbolValue(char name[]);
int      FillSymbolTable (char *fname);
```

```
/*** FinalQ2.cpp (main)  ***/

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>
#include "scanner.h"
#include "symbolTable.h"

char *symbolFileName = "inputSymbols.txt";
char *infixFileName = "inputInfix.txt";

int in2postfix(TokenType * infix, TokenType * postfix, int length);
int eval(TokenType * postfix, int length);

nt main() {
        char infixString[80] = { 0 };
        TokenType infixToken[80] = { 0 }; int infixSize = 0;
        TokenType postfixToken[80] = { 0 }; int postfixSize = 0;

        /* Symbol Table [5] */
        if (FillSymbolTable(symbolFileName) > 0) PrintSymbols();
        else {
                printf("Error: File %s not found!!!₩n", symbolFileName);
                return - 1;
        }

        /* Open to get Infix Expression String */
        FILE *spInfix = fopen(infixFileName, "r");
        if (spInfix == NULL) {
                printf("Error: File %s not found!!!₩n", infixFileName);
                return -1;
        }
        while (fgets(infixString, 79, spInfix)) { //Infix String
                /* Infix String to Tokens [5] */
                if (strlen(infixString) <= 1) break;
                strtok(infixString, "₩n");         //remove neline if exists
                //printf("* Input infix String: %s₩n", infixString);
                infixSize = runFSM(infixToken, infixString);

                printf("* Infix expression: ₩n");
                for (int i = 0; i < infixSize; i++) printToken(0, infixToken[i]);
                printf("₩n");
                printf("----------------------------------------₩n");

                /* Infix Tokens to Postfix Tokens [5] */
                postfixSize = in2postfix(infixToken, postfixToken, infixSize);
                printf("* Postfix expression:₩n");
                for (int i = 0; i < postfixSize; i++) printToken(1, postfixToken[i]);
                printf("₩n");

                /* Evaluate Postfix Expression [5] */
                printf("* Evaluated => %d₩n", eval(postfixToken, postfixSize));
                printf("------------------------------------------------₩n");
        }
        return 0;
}
                                                }
```

/*** POSTFIX_INFIX.docx (Stack 예제) ***/

```
#define MAX_STACK_SIZE 100  /* maximum stack size */
#define MAX_EXPR_SIZE 100   /* max size of expression */
typedef enum {lparen, rparen, plus, minus, times, divide,
                  mod, eos, operand } precedence;
int stack[MAX_STACK_SIZE];  /* global stack */
char expr[MAX_EXPR_SIZE];   /* input string */
precedence stack[MAX_STACK_SIZE];
/* isp and icp arrays - index is value of precedence
lparen, rparen, plus, minus, times, divide, mod, eos */
/* isp: in stack precedence, icp: incoming precedence */
static int isp[] =  {  0, 19, 12, 12, 13, 13, 13, 0 };
static int icp[] =  { 20, 19, 12, 12, 13, 13, 13, 0 };

precedence getToken (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;
   }
}
```

```
/*** Infix to Postfix Implementation ***/

void postfix(void) {
  char symbol; precedence token; int n = 0;
  stack[0] = eos;
  for (token=getToken(&symbol,&n); token!=eos;
            token=getToken(&symbol,&n)){
    if (token == operand) printf("%c", symbol);
    else if (token == rparen){
       while (stack[top] != lparen)
          printToken(pop());
       pop(); /* discard the left parenthesis */
    } else {
       while (isp[stack[top]] >= icp[token])
          printToken(pop());
       push(token);
    }
  }
  while ((token=pop()) != eos) printToken(token);
}
```

```
/*** Evaluating of Postfix Expression ***/

int eval(void){
   precedence token;
   char symbol;  int op1,op2;
   int n = 0; top = -1;
   token = getToken(&symbol, &n);
   while (token != eos) {
      if (token==operand) push(symbol-' 0' );
      else {
         op2=pop(); op1=pop();
         switch(token) {
            case plus:  push(op1+op2); break;
            case minus: push(op1-op2); break;
            case times: push(op1*op2); break;
            case divide: push(op1/op2); break;
            case mod: push(op1%op2);
         } /* switch */
      } /* else */
         token = getToken(&symbol, &n);
   } /* while */
   return pop(); /* return result */
}
```