**부록: 참고 자료 (Final Chapter)**

**/\*\*\* 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 val;

} TokenType;

precedence getPrecedence(char \* str);

void printToken(int flag, TokenType token);// flag = 1/0: operand value or symbol

int unFSM(TokenType \*token, char \*infix);

0

1

2

3

S (Separator, others)

영문자

숫자

영문자

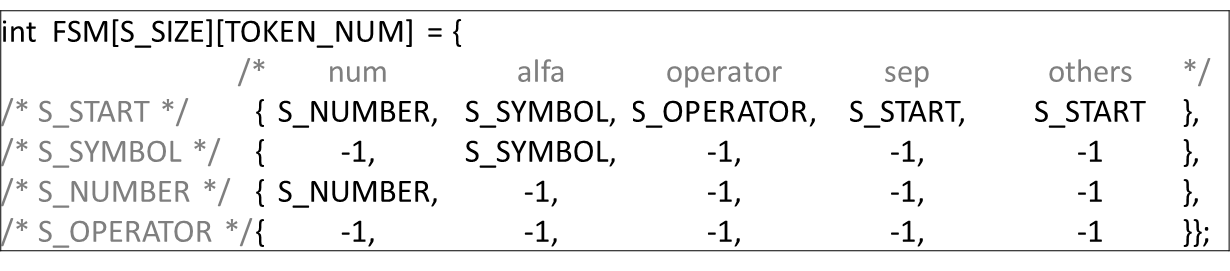
opr

숫자

S

S

S



**/\*\*\* symbolTable.h \*\*\*/**

void PrintSymbols (void);

int FindSymbol (char name[]);

int GetSymbolValue(char name[]);

int FillSymbolTable (char \*fname);

**/\*\*\* stack.h \*\*\*/**

#include <ctype.h>

#include "scanner.h"

void push(TokenType

TokenType \* pop();

TokenType stackTop();/\* element at stack top \*/

**/\*\*\* FinalChapter.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;**

**}**

**}**

**/\*\*\* 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 \*/**

**}**

**/\*\*\* 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);**

**}**