```
//2020118008 박보경
//심화컴퓨터학부

//01
//2020118008 박보경
//본인은 이 소스파일을 다른 사람의 소스를 복사하지 않고 직접 작성하였습니다.

//후위 표기법으로 표현된 하나의 수식을 파일로 입력받아 그 계산 결과를 화면에 출력하는 프로그램
//후위 표기법 계산

#define _CRT_SECURE_NO_WARNINGS
#define MAX_STACK_SIZE 100
#define MAX_EXPR_SIZE 80

//열거형 자료형 ( ) + - * / % endofstack 피연산자
typedef enum {lparen, rparen, plus, minus, times, divide, mod, eos, operand } precedence;

int stack[MAX_STACK_SIZE];
char expr[MAX_EXPR_SIZE];
int top = -1;

void push(int item);
int pop();
void stackFull();
void stackEmpty();
int eval();
precedence getToken(char* symbol, int* n);

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
	FILE* fp = fopen("input.txt", "r");

	//파일에 있는 데이터를 80만큼 expr에 받음
	fgets(expr, 80, fp);
	int result = eval();

	printf("postfix expression : %s \n", expr); //
```

```
        printf("the evaluation value: %d \n", result);

        return 0;
}


void push(int item)
{
        if (top >= MAX_STACK_SIZE - 1)
                stackFull();

        stack[++top] = item;
}

int pop()
{
        if (top == -1)
                stackEmpty();

        return stack[top--];
}

void stackFull()
{
        fprintf(stderr, "stack is full, cannot add element \n");
        exit(EXIT_FAILURE);
}

void stackEmpty()
{
        fprintf(stderr, "stack is empty, cannot delete element \n");
        exit(EXIT_FAILURE);
}


//후위표기법을 계산하는 함수 (스택을 이용)
int eval()
{
        precedence token;
        char symbol;
        int op1, op2;
        int n = 0;
```

```
        //token에 연산자와 피연산자를 받음
        token = getToken(&symbol, &n);

        //end of stack이 아닐 때
        //피연산자일때는 아스키코드를 활용해서 int형으로 바꿔서 push
        //연산자일때는 피연산자 2개를 pop해서 연산
        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);
                        }
                }
                token = getToken(&symbol, &n);
        }

        return pop();
}

//연산자와 피연산자 받기
precedence getToken(char* symbol, int* n)
{
        *symbol = expr[(*n)++];
        switch (*symbol) {
        case '+': return plus;
        case '-': return minus;
        case '*': return times;
```

```
        case '/': return divide;
        case '%': return mod;
        case '\0': return eos;
        default: return operand;
        }
}

//02
//2020118008 박보경
//본인은 이 소스파일을 다른 사람의 소스를 복사하지 않고 직접 작성하였습니다.

//중위표기법으로 표현된 하나의 수식을 파일로 입력받아 후위표기법으로 변환하여 화면 및 파일에 동시에 출력하는 프로그램
//중위->후위

#define _CRT_SECURE_NO_WARNINGS
#define MAX_STACK_SIZE 100
#define MAX_EXPR_SIZE 80
#define MAX_STRING_SIZE 80

//열거형 자료형
typedef enum { lparen, rparen, plus, minus, times, divide, mod, eos, operand }
precedence;

precedence stack[MAX_STACK_SIZE];
char expr[MAX_EXPR_SIZE];
char post[MAX_STRING_SIZE];
int top = -1;
int index = 0;

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

void push(int item);
precedence pop();
void stackFull();
void stackEmpty();
void printToken(precedence token);
void postfix(void);

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
int main()
{
	FILE* fp = fopen("input.txt", "r");
	fgets(expr, 80, fp);
	printf("<<<<<<<infix to postfix>>>>>>>>>>>\n");
	printf("infix expression		:%s \n", expr);


	printf("postfix expression		:");
	postfix();

	FILE* fb = fopen("output.txt", "w");
	fputs(post, fb);


	return 0;
}


void push(int item)
{
	if (top >= MAX_STACK_SIZE - 1)
		stackFull();

	stack[++top] = item;
}

precedence pop()
{
	if (top == -1)
		stackEmpty();

	return stack[top--];
}


void stackFull()
{
	fprintf(stderr, "stack is full, cannot add element \n");
	exit(EXIT_FAILURE);
}
```

```
void stackEmpty()
{
        fprintf(stderr, "stack is empty, cannot delete element \n");
        exit(EXIT_FAILURE);
}

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 times;
        case '/': return divide;
        case '%': return mod;
        case '\0': return eos;
        default: return operand;
        }
}

void printToken(precedence token)
{
        char item;

        switch (token) {
        case plus:
                item = '+';
                break;
        case minus:
                item = '-';
                break;
        case times:
                item = '*';
                break;
        case divide:
                item = '/';
                break;
        case mod:
                item = '%';
        }
```

```
        printf("%c", item);
        post[index++] = item;

        return;
}

//중위->후위
void postfix()
{
        char symbol;
        precedence token;
        int n = 0;
        top = 0;
        stack[0] = eos;

        for (token = getToken(&symbol, &n); token != eos; token = getToken(&symbol,
&n)) {
                if (token == operand) {
                        printf("%c", symbol);
                        post[index++] = symbol;
                }
                else if (token == rparen) {
                        while (stack[top] != lparen)
                                printToken(pop());
                        pop();
                }
                else {
                        while (isp[stack[top]] >= icp[token])
                                printToken(pop());
                        push(token);
                }
        }

        while ((token = pop()) != eos)
                printToken(token);
        printf("\n");
}
```