컴파일러 hw1 – 어휘 분석

- 소프트웨어학과 201520882학번 천윤서

1. 서론

- 개요

본 과제는 flex를 활용해서 Mini-C라는 정의된 언어의 어휘를 분석하는 과제이다. 분석된 어휘는 name과 value를 가진 token으로 출력되며, 어휘의 id와 string은 각각 symbol table과 string table이라는 장소에 저장되어진다. 이를 위해 Regular Expression에 관한 지식을 완벽히 습득해야 하는 과제이다.

- 구현된 부분

과제에서 요구하는 어휘의 모든 요구사항을 충족하며, 모든 분석된 어휘를 바탕으로 모든 Token으로 변환하는 것을 구현하였다. Token은 별도의 장소에 저장되지 않고, 그때 그때 반환되며, 이 프로그램은 변환된 토큰, symbol table, string table을 stdin에 출력하도록 만들어져 있다. ID는 symbol table에 저장되며, 반환되는 토큰의 name은 ID, value는 해당 symbol table의 Index를 지니며, string 역시 string table이라는 곳에 저장되어 name은 STRING, value는 string table의 Index를 지닌다.

- 구현되지 않은 부분

과제에서 요구하는 부분은 모두 구현하였다.

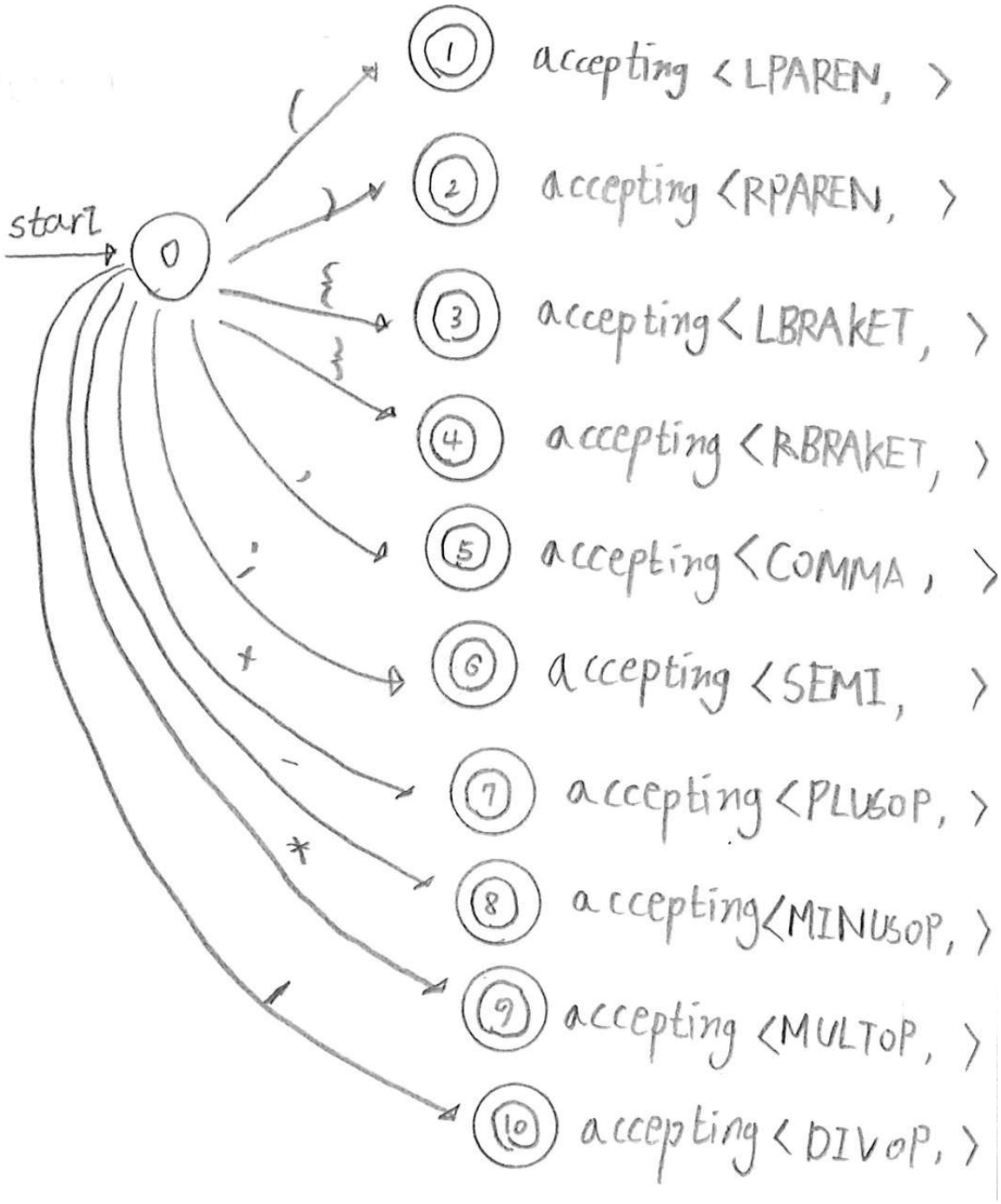
2. 문제 분석

- token의 종류

토큰	값	패턴
ID	Symbol table의 포인터	1. 영문자 대소문자, 숫자, underscore(_)로만 이루어짐. 2. 첫 글자는 반드시 대소문자, _만 가능. 3. '_'문자만으로 이루어질 수는 없음. 4. 길이는 제한이 없으나 실제 구분은 첫 16자로 함.
LPAREN		(
RPAREN		)
LBRAKET		{
RBRAKET		}
COMMA		,
SEMI		;
INT		int
DOUBLE		double
STR		str
PRINT		print
RETURN		return
IF		if
ELSE		else
WHILE		while
PLUSOP		+
MINUSOP		-
MULTOP		*
DIVOP		/
ASIGNOP		=
RELOP	LT, LE, GT, GE, EQ, NE	<, >, <=, >, >=, ==, !=
INTEGER	정수	1. 기본적으로 C언어의 int 표기법을 따름(십진수만 사용) 2. 자리 수는 제한 없으나 값을 저장 시 최대 10자리만 저장.
DOUBLE	실수	C언어의 Double Constant 표기법을 따름.
STRING	스트링 테이블의 인덱스	기본적으로 C언어의 표기법을 따름.

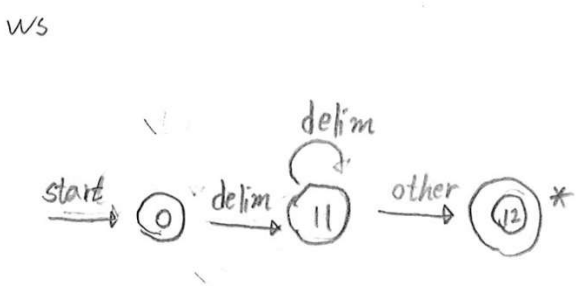
- transition diagram

1. LPAREN, RPAREN, LBRAKET, RBRAKET, COMMA, SEMI, PLUSOP, MINUOP, MULTOP, DIVOP



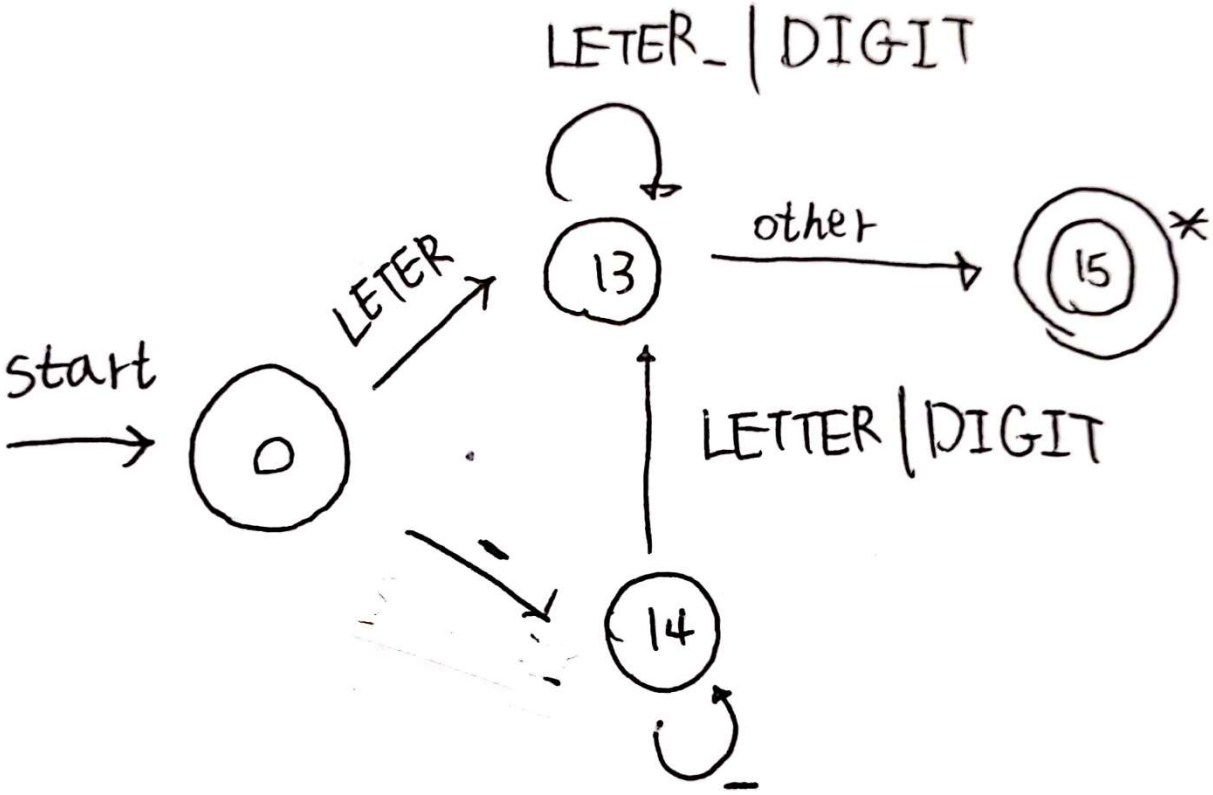
- 이 Case들은 그냥 순서대로 글자가 오면 처리하면 되고, Regular Expression도 간략히 표현된다.

2. WhiteSpace



- Delim(₩t₩n₩)이 한 번 이상 올 경우를 전부 포함한다. 다른 글자가 오면 Retraction한다.

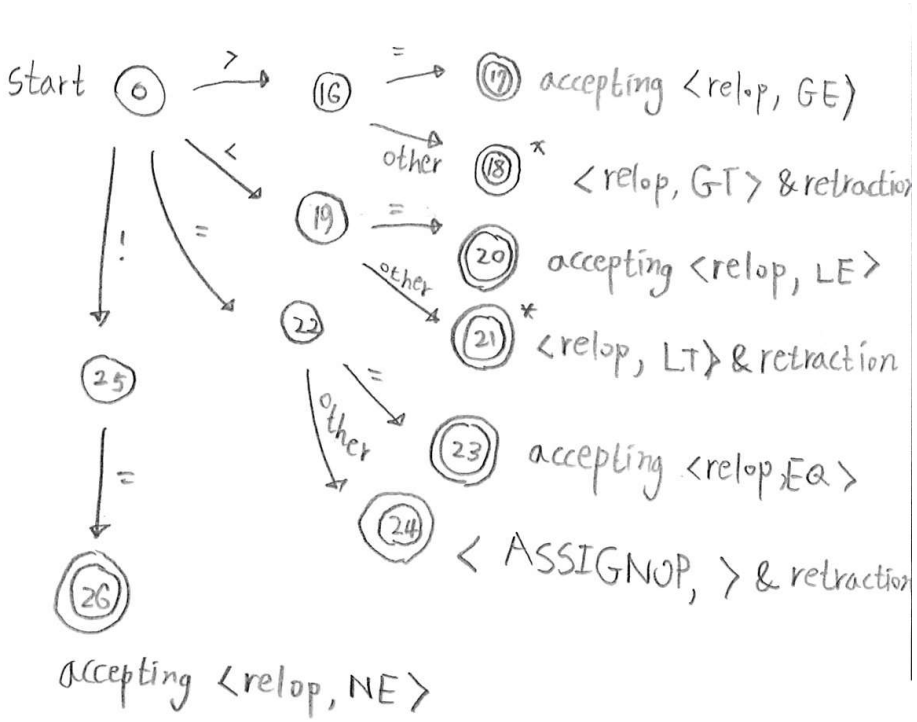
3.ID



- letter_ : 영문자와 _를 포함한 집합. letter : 영문자를 포함한 집합 digit : 숫자를 포함한 집합.

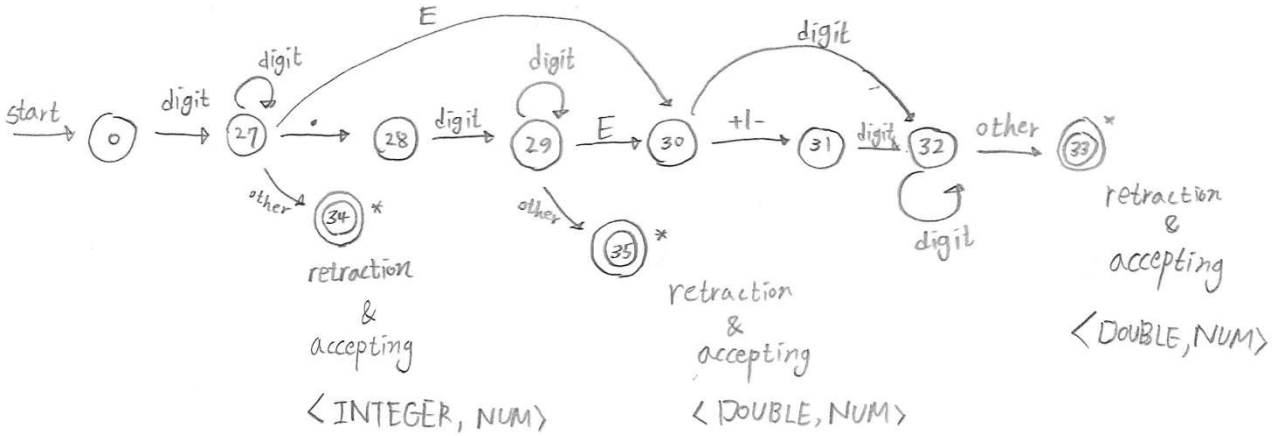
- 시작은 LETTER나 _(UNDERBAR)로 시작한다. 언더 바로 시작하면 반드시 숫자나 문자가 한 번 이상 나와야 하고, 그 이후에는 LETTER_와 DIGIT을 반복하다가 다른 글자가 나오면 Retraction한다.

4. GE, GT, LE, LT, EQ, ASSIGNOP, NE



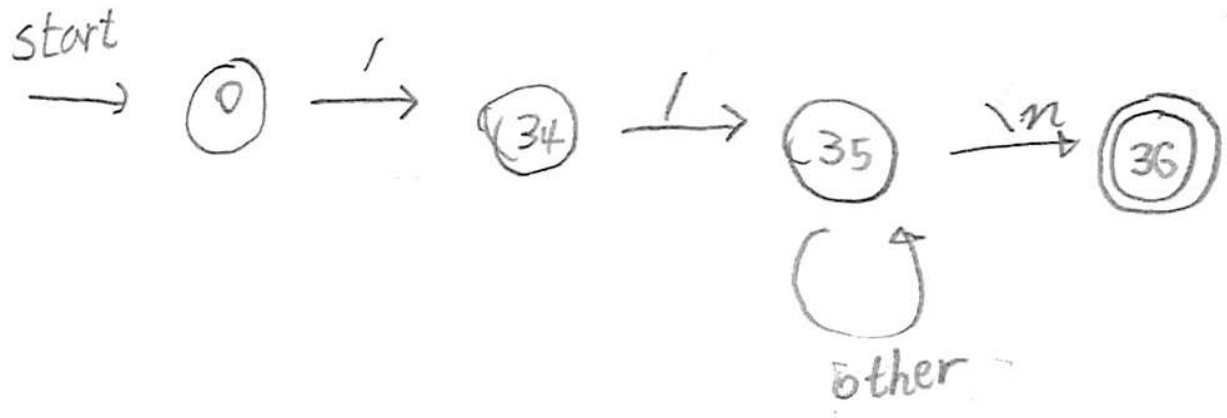
- 역시 Regular Expression 이 거의 문자 그대로 나온다. 다만 Retraction 이 아닌 것과 Retraction 인 것들이 섞여있다.

5. INTEGER, DOUBLE



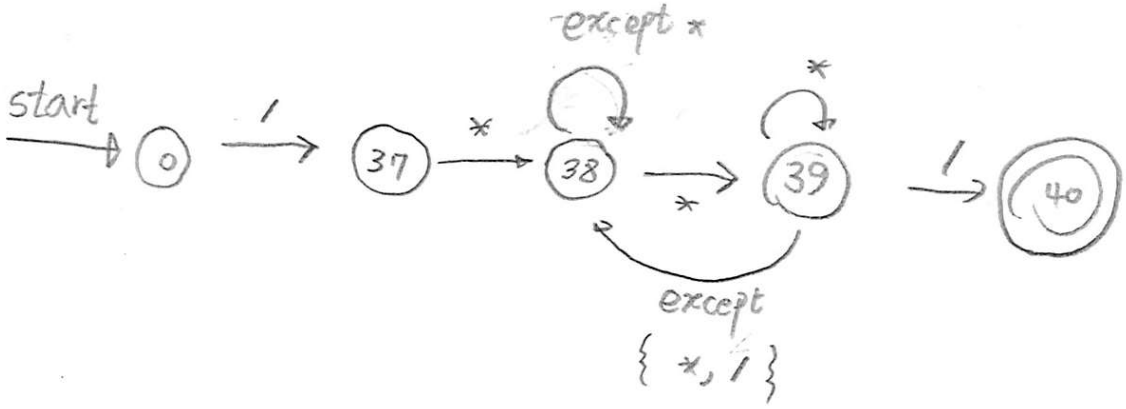
- 시작 후 숫자를 받으며, 숫자를 반복하다가 다른 것이 오면 INTEGER 이다. 숫자 뒤에 .이나 E 가 오면 DOUBLE 로 취급되며, 그 후 숫자를 반복하다 끝날 수도 있고, E 이후에 기호가 들어가거나 아니면 바로 숫자가 반복된 뒤 다른 것이 나오면 끝난다.

6. SINGLINE COMMENT



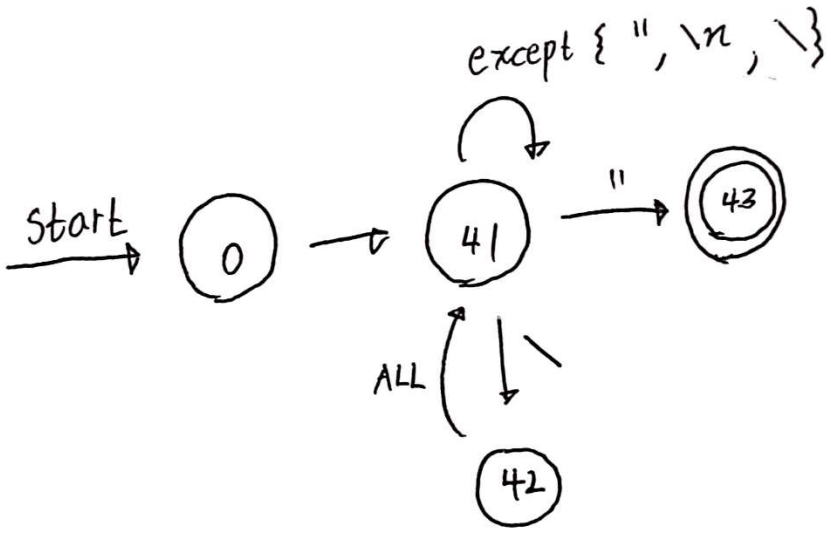
- //가 나온 뒤 new line 을 제외한 모든 글자를 반복해서 받을 수 있다.

7. MULTILINE COMMENT



- /*가 나온 뒤 *를 제외한 글자를 반복할 수 있다. *가 하나 이상 나온다면 곧바로 /가 다시 나오면 끝나며 *나 /를 제외한 글자가 나오면 다시 *를 제외한 글자를 무한정 반복할 수 있다.

8. STRING



- 따옴표를 받으면 41 번 State 로 가서 시작하며, 따옴표, New line 이나 Back Slash 를 제외한 것들을 반복한다. Back Slash 를 받으면 그 뒤에는 아무거나 올 수 있으며(C 언어의 String 규칙에 따라서 New Line 도 반드시 백슬래시 뒤에만 올 수 있고, 나머지가 오면 에러를 출력하거나 경고를 띄운다.) 또 다시 이 규칙을 반복하다가 따옴표가 나오면 string 이 닫히면서 끝난다.

- 각 토큰을 위한 regular expression

토큰	예시	REGULAR EXPRESSION	비고
<LPAREN,>	(	\(	
<RPAREN,>	)	\)	
<LBRAKET,>	{	\{	
<RBRAKET,>	}	\}	
<COMMA,>	,	,	
<SEMI,>	;	;	
<PLUSOP,>	+	\+	
<MINUSOP,>	-	-	
<MUILTOP,>	*	*	
<DIVOP,>	/	/	
WHITESPACE		(\ |\t|\n)+	나오면 무시하도록 동작.
SINGLECOMMENT	//	\/\/.*	실제로는 같은 COMMENT 토큰으로 처리됨.
MULTICOMMENT	/**/	\/*([^*]|*+[^*\/])**+\/	
<RELOP, GE>	>=	>=	
<RELOP, GT>	>	>	
<RELOP, LE>	<=	<=	
<RELOP, LT>	<	<	
<RELOP, EQ>	==	==	
<RELOP, NE>	!=	!=	
<ASSIGNOP,>	=	=	
<ID, INDEX>	a	(LE|(_+([a-zA-Z]|[0-9])))([a-zA-Z_]|[0-9])*	1. LE_: [a-zA-Z_] 2. LE: [a-zA-Z] 3. DG: [0-9]
<INTEGER, NUM>	123	DG+	
<DOUBLE, NUM>	31E-2	DG+(((\.DG+E)|E)((\+|-)?DG+)|\.DG+)	
<STRING, INDEX>	"a"	\"([^\"\n]|\\(.|\n))*\"	C언의 규칙에 따름
<INT,>	int	1. KEYWORD는 ID의 특별한 경우로 볼 수 있다. 2. Regular Expression 대신 ID로 먼저 처리하고 Key Word 테이블을 따로 둬서 처리. (컴파일러 강의노트 2번(2.lex)의 필기 내용을 참조함.)	
<DOUBLE,>	double		
<STR,>	str		
<PRINT,>	print		
<RETURN,>	return		
<IF,>	if		
<ELSE,>	else		
<WHILE,>	while		

3. 설계

- 구조체/주요 변수

이름	변수	설명
token	char name[10] char value[25]	반환되는 토큰을 저장하는 구조체이다.
keyword_table	char keyword[15] char token[15]	keyword 의 정보를 담고 있는 구조체이다. 이를 이용해서 ID 에서 keyword 를 걸러낸다.
char **symbol_table/char **string_table		심볼 테이블과 스트링 테이블.
int symbolIndex / int stringIndex		현재 테이블의 맨 끝을 가리키는 변수이다.
int symbol_table_size / int string_table_size		테이블의 크기를 나타낸다. 동적으로 2 배씩 증가한다.
keyword_table keyword[KEY_NUM]		실제 keyword 가 담겨있는 키워드 테이블이다.
int lineCount		에러가 일어난 위치를 알려주기 위해 줄을 세는 변수.

- 주요 함수/모듈

함수/모듈	설명
int countNewLine(char *input)	₩n 이 나온 횟수를 세는 함수이다. 에러를 반환할 때 위치도 함께 알려주기 위해서 사용한다.
void initTable()	심볼 테이블을 동적으로 선언하는 함수.
int isKeyword(char* input)	ID 로 판별된 토큰이 키워드에 속하는지 키워드 테이블을 검사하여 그 인덱스를 반환해주는 함수. 키워드가 아니면 -1 을 반환한다.
char* putSymbolTable(char* input)	해당하는 input 이 심볼 테이블에 있는지 검사하고 없으면 추가하고 해당 심볼의 인덱스를 반환하는 함수. 또한, 넣으면서 테이블을 동적으로 조절하는 기능도 수행.
char* putStringTable(char* input)	해당하는 input 이 스트링 테이블에 있는지 검사하고 없으면 추가하며, 해당 string 의 인덱스를 반환하는 함수. 또한, 넣으면서 테이블을 동적으로 조절하는 기능도 수행.
char* cutDouble(char* input)	input 으로 들어온 double 을 C 언어 double 의 유효숫자 만큼 잘라내는 함수. 오버 플로우를 감지하지는 않는데, 오버 플로우는 연산으로도 발생하는 만큼, 어휘 분석단계에서 처리할 일이 아니라고 판단했다.
int isDecimalNum(char *input)	input 으로 들어온 숫자가 8 진수가 아닌 10 진수인지 여부를 판단하는 함수이다. 10 진수가 맞으면 1 을, 아니면 0 을 반환한다.
char* processMultilineString(char *input)	string 에 ₩가 들어와 줄변환이 일어났는지 감지하고 이를 C 스타일로 처리해주는 함수이다. 양 끝의 따옴표를 잘라내는 역할 또한 수행한다.
token getToken(int type)	Regular Expression 으로 분석된 어휘들을 받아서 심볼 테이블에 넣는 등, 각 어휘에 맞는 기능을 수행 후, 토큰을 반환하는 함수.
void printToken(token t)	input 토큰을 문자열 형태로 stdin 에 출력하는 함수.
void printSymbolTable()	심볼 테이블 전체를 stdin 에 출력하고 관련된 모든 메모리를 해제하는 함수
void printStringTable()	스트링 테이블 전체를 stdin 에 출력하고 관련된 모든 메모리를 해제하는 함수.
Main 함수	초기화 후 파일을 열고, 토큰을 받아서 getToken 함수를 실행하는 역할만 수행.

4. 입력 데이터

- ex1.txt(정상 파일)

```
int main()
{
	int cost_1 = 12345678901234567890123456789O, int cost_2 = 15345;
	int costResult;
	costResult = cost_1 + cost_2;
	costResult = cost_1 - 3.141592;
	costResult = cost_2/3.14E13;
	costResult = cost_2 * 364E-10;

	//this is only for test ₩o/₩o/
	//finish homework! ₩o/₩o/₩o/ :)

	str veryveryveryveryveryverylongID____ = "such a small thing₩n₩t";
	double hyperLongDouble = 1234567890.12345678901234567890123456789OE+100;
	print("%s₩"",str);

	double alpha = 3.1415926535;
	double beta   = 3.141592653537E+10;
	int _6, __a, b_, _c_;

	print("multi line string₩
			two line₩n");

	int ____________________________________a;
	int ____________________________________b;


	while(1)
	{
		if(alpha > beta) print("₩"alpha win₩"");
		else if(alpha >= beta) print("₩"alpha win₩"");
		else if(alpha < beta) print("beta win₩n");
		else if(alpha <= beta) print("beta win₩n");
		else if(alpha == beta) print("ambigous₩"₩"");
		else if(alpha != beta) print("ambigous₩"₩"");
	}
```

```
        return 0;
        /*-------------------------------------------------
    finish - 2019/03/30
        made by - Yunseo Chun
        student code - 201520882
        /////*****
        -------------------------------------------------*/

}
```

-ex2.txt(비정상 파일)

```
int main()
{
        int a = 00123; //Unacceptable integer
        $%!; //Unacceptable character
        double __ = 3..145; //Unacceptable id and double
        print("Hello World!₩n); //Unacceptable String
}
```

5. 결과 데이터

- 정상 파일(콘솔에서 "./program ex1.txt > result.txt" 입력 후 result.txt의 내용)

TOKEN LEXEME
<INT, > int
<ID, 1> main
<LPAREN, > (
<RPAREN, >)
<LBRAKET, > {
<INT, > int
<ID, 2> cost_1
<ASSIGNOP, > =
<INTEGER, 5678901234567890> 12345678901234567890123456789O
<COMMA, > ,
<INT, > int
<ID, 3> cost_2
<ASSIGNOP, > =
<INTEGER, 15345> 15345
<SEMI, > ;
<INT, > int
<ID, 4> costResult
<SEMI, > ;
<ID, 4> costResult
<ASSIGNOP, > =
<ID, 2> cost_1
<PLUSOP, > +
<ID, 3> cost_2
<SEMI, > ;
<ID, 4> costResult
<ASSIGNOP, > =
<ID, 2> cost_1
<MINUSOP, > -
<DOUBLE, 3.141592> 3.141592
<SEMI, > ;
<ID, 4> costResult
<ASSIGNOP, > =
<ID, 3> cost_2
<DIVOP, > /
<DOUBLE, 3.14E13> 3.14E13
<SEMI, > ;

<ID, 4>	costResult
<ASSIGNOP, >	=
<ID, 3>	cost_2
<MULTOP, >	*
<DOUBLE, 364E-10>	364E-10
<SEMI, >	;
<COMMENTS, >	//this is only for test ₩o/₩o/
<COMMENTS, >	//finish homework! ₩o/₩o/₩o/ :)
<STR, >	str
<ID, 5>	veryveryveryveryveryverylongID____
<ASSIGNOP, >	=
<STRING, 1>	"such a small thing₩n₩t"
<SEMI, >	;
<DOUBLE, >	double
<ID, 6>	hyperLongDouble
<ASSIGNOP, >	=
<DOUBLE, 1234567890.1234567E+100>
	1234567890.12345678901234567890123456789E+100
<SEMI, >	;
<PRINT, >	print
<LPAREN, >	(
<STRING, 2>	"%s₩""
<COMMA, >	,
<STR, >	str
<RPAREN, >	)
<SEMI, >	;
<DOUBLE, >	double
<ID, 7>	alpha
<ASSIGNOP, >	=
<DOUBLE, 3.1415926535>	3.1415926535
<SEMI, >	;
<DOUBLE, >	double
<ID, 8>	beta
<ASSIGNOP, >	=
<DOUBLE, 3.141592653537E+10E+100>	3.141592653537E+10
<SEMI, >	;
<INT, >	int
<ID, 9>	_6
<COMMA, >	,
<ID, 10>	__a

<COMMA, >		,
<ID, 11>	b_
<COMMA, >		,
<ID, 12>	_c_
<SEMI, >		;
<PRINT, >		print
<LPAREN, >		(
<STRING, 3>		"multi line string₩
		two line₩n"
<RPAREN, >		)
<SEMI, >		;
<INT, >		int
<ID, 13>	_____a
<SEMI, >		;
<INT, >		int
<ID, 13>	_____b
<SEMI, >		;
<WHILE, >		while
<LPAREN, >		(
<INTEGER, 1>		1
<RPAREN, >		)
<LBRAKET, >		{
<IF, >		if
<LPAREN, >		(
<ID, 7>		alpha
<RELOP, GT>		>
<ID, 8>		beta
<RPAREN, >		)
<PRINT, >		print
<LPAREN, >		(
<STRING, 4>		"₩"alpha win₩""
<RPAREN, >		)
<SEMI, >		;
<ELSE, >		else
<IF, >		if
<LPAREN, >		(
<ID, 7>		alpha
<RELOP, GE>		>=
<ID, 8>		beta
<RPAREN, >		)

<PRINT, >	print
<LPAREN, >	(
<STRING, 4>	"₩"alpha win₩""
<RPAREN, >	)
<SEMI, >	;
<ELSE, >	else
<IF, >	if
<LPAREN, >	(
<ID, 7>	alpha
<RELOP, LT>	<
<ID, 8>	beta
<RPAREN, >	)
<PRINT, >	print
<LPAREN, >	(
<STRING, 5>	"beta win₩n"
<RPAREN, >	)
<SEMI, >	;
<ELSE, >	else
<IF, >	if
<LPAREN, >	(
<ID, 7>	alpha
<RELOP, LE>	<=
<ID, 8>	beta
<RPAREN, >	)
<PRINT, >	print
<LPAREN, >	(
<STRING, 5>	"beta win₩n"
<RPAREN, >	)
<SEMI, >	;
<ELSE, >	else
<IF, >	if
<LPAREN, >	(
<ID, 7>	alpha
<RELOP, EQ>	==
<ID, 8>	beta
<RPAREN, >	)
<PRINT, >	print
<LPAREN, >	(
<STRING, 6>	"ambigous₩"₩""
<RPAREN, >	)

<SEMI, >		;
<ELSE, >		else
<IF, >		if
<LPAREN, >		(
<ID, 7>		alpha
<RELOP, NE>		!=
<ID, 8>		beta
<RPAREN, >		)
<PRINT, >		print
<LPAREN, >		(
<STRING, 6>		"ambigous₩"₩""
<RPAREN, >		)
<SEMI, >		;
<RBRAKET, >		}
<RETURN, >		return
<INTEGER, 0>		0
<SEMI, >		;
<RBRAKET, >		}

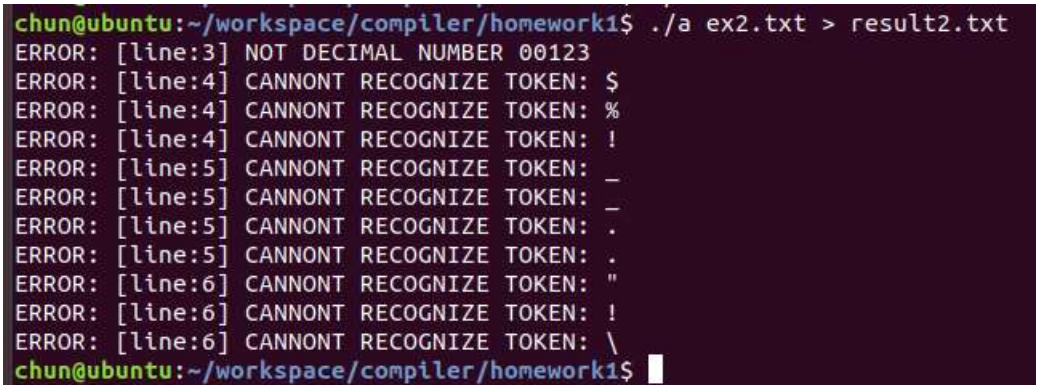
----------SYMBOL-TABLE----------

index	symbols
1	"main"
2	"cost_1"
3	"cost_2"
4	"costResult"
5	"veryveryveryvery"
6	"hyperLongDouble"
7	"alpha"
8	"beta"
9	"_6"
10	"__a"
11	"b_"
12	"_c_"
13	"_____"

----------STRING-TABLE----------

index	strings	
1	such a small thing₩n₩t	
2	%s₩"	
3	multi line string	two line₩n
4	₩"alpha win₩"	
5	beta win₩n	
6	ambigous₩"₩"	

- 비정상 파일



비정상 문자에 관한 에러와, "_"만으로 이루어진 ID, .이 두 번 들어온 DOUBLE, 0이 앞에 온 INTEGER, 닫는 따옴표가 없는 string에 관해서 stderr에 에러를 출력한다.

6. 소스 코드

```
%{
#define LPAREN 1
#define RPAREN 2
#define LBRAKET 3
#define RBRAKET 4
#define COMMA 5
#define SEMI 6
#define PLUSOP 7
#define MINUSOP 8
#define MULTOP 9
#define DIVOP 10
#define WHITESPACE 11
#define SINGLECOMMENT 12
#define MULTICOMMENT 13
#define GE 14
#define GT 15
#define LE 16
#define LT 17
#define EQ 18
#define NE 19
#define ASSIGNOP 20
#define ID 21
#define INTEGER 22
#define DOUBLE 23
#define STRING 24
#define UNKNOWN 25
#define KEY_NUM 8

int symbolIndex = 1, stringIndex = 1;
int symbol_table_size = 4; int string_table_size = 4;
int lineCount = 1;

char **symbol_table;
char **string_table;

typedef struct _token{
        char name[10];
        char value[25];
```

```
} token;

typedef struct _keyword_table {
        char keyword[15];
        char token[15];
} keyword_table;

keyword_table keyword[KEY_NUM] = {{"int", "INT"}, {"double", "DOUBLE"}, {"str", "STR"}, {"print", "PRINT"},
{"return", "RETURN"}, {"if", "IF"}, {"else", "ELSE"}, {"while", "WHILE"}};
%}

LPAREN ₩(
RPAREN ₩)
LBRAKET ₩{
RBRAKET ₩}
COMMA ,
SEMI ;
PLUSOP ₩+
MINUSOP -
MULTOP ₩*
DIVOP ₩/
WHITESPACE (₩ |₩t|₩n)+
SINGLECOMMENT ₩/₩/.*
MULTICOMMENT ₩/₩*([^₩*]|(₩*+[^₩*₩/]))*₩*+₩/
GE >=
GT >
LE <=
LT <
EQ ==
NE !=
ASSIGNOP =
ID ([a-zA-Z]|(_+([a-zA-Z]|[0-9])))([a-zA-Z_]|[0-9])*
INTEGER [0-9]+
DOUBLE [0-9]+(((₩.[0-9]+E)|E)((₩+|-)?[0-9]+)|₩.[0-9]+)
STRING ₩"([^₩n₩"]|(₩₩(.|₩n)))*₩"

%%
{LPAREN} return LPAREN;
{RPAREN} return RPAREN;
{LBRAKET} return LBRAKET;
```

```
{RBRAKET} return RBRAKET;
{COMMA} return COMMA;
{SEMI} return SEMI;
{PLUSOP} return PLUSOP;
{MINUSOP} return MINUSOP;
{MULTOP} return MULTOP;
{DIVOP} return DIVOP;
{WHITESPACE} return WHITESPACE;
{SINGLECOMMENT} {return SINGLECOMMENT;}
{MULTICOMMENT} return MULTICOMMENT;
{GE} return GE;
{GT} return GT;
{LE} return LE;
{LT} return LT;
{EQ} return EQ;
{NE} return NE;
{ASSIGNOP} return ASSIGNOP;
{ID} return ID;
{INTEGER} return INTEGER;
{DOUBLE} return DOUBLE;
{STRING} return STRING;
. return UNKNOWN;
%%

int countNewLine(char *input)
{
	int i, count;
	for(i=0; i<yyleng; i++)
	{
		if(input[i] == '₩n') count++;
	}
	return count;
}

void initTable()
{
	symbol_table = (char**)malloc(sizeof(char**)*symbol_table_size);
	string_table = (char**)malloc(sizeof(char**)*string_table_size);

	int i;
```

```
	for(i = 1; i<symbol_table_size; i++) symbol_table[i] = (char*)calloc(17,sizeof(char));
}

int isKeyword(char *input)
{
	int i;
	for(i=0; i < KEY_NUM; i++)
	{
		if(strcmp(input, keyword[i].keyword) == 0)
		{
			return i;
		}
	}
	return -1;
}

char* putSymbolTable(char* input)
{
	int i;
	char *str = (char*)malloc(sizeof(char)*25);
	char *idBuffer = (char*)malloc(sizeof(char)*17);

	if(symbolIndex >=  symbol_table_size)
	{
		int before_size = symbol_table_size;
		symbol_table_size *= 2;
		symbol_table = (char**)realloc(symbol_table, sizeof(char*)*symbol_table_size);

		for(i = before_size; i<symbol_table_size; i++)
		{
			symbol_table[i] = (char*)calloc(17,sizeof(char));
		}
	}

	strncpy(idBuffer, input, 16);
	idBuffer[16] = '₩0';

	for(i=1; i<symbolIndex; i++)
	{
```

```
                if(strcmp(idBuffer, symbol_table[i])==0)
                {
                        sprintf(str, "%d", i);
                        free(idBuffer);
                        return str;
                }
        }

        strcpy(symbol_table[symbolIndex], idBuffer);
        free(idBuffer);
        symbolIndex++;

        sprintf(str, "%d", symbolIndex-1);
        return str;
}

char* putStringTable(char *input)
{
        int i;
        char *str = (char*)malloc(sizeof(char)*25);
        for(i=1; i<stringIndex; i++)
        {
                if(strcmp(input, string_table[i]) == 0)
                {
                        sprintf(str, "%d", i);
                        return str;
                }
        }

        if(stringIndex >= string_table_size)
        {
                int before_size = string_table_size;
                string_table_size *= 2;
                string_table = (char**)realloc(string_table, sizeof(char*)*string_table_size);
        }

        string_table[stringIndex] = (char*)calloc(yyleng+1, sizeof(char));
        strcpy(string_table[stringIndex], input);
        stringIndex++;
        sprintf(str, "%d", stringIndex-1);
```

```
	return str;
}

char* cutDouble(char* input)
{
	int inputIndex = 0;
	int strIndex = 0;
	int sig = 0;
	int exp = 0;
	char* str = (char*)malloc(sizeof(char)*25);
	while((inputIndex < yyleng)&&(input[inputIndex] != 'E'))
	{
		if((input[inputIndex] >= '0')&&(input[inputIndex] <= '9')) sig++;


		if(sig <= 17) str[strIndex++] = input[inputIndex++];
		else inputIndex++;
	}

	str[strIndex++] = input[inputIndex++]; //put E
	while(inputIndex < yyleng && strIndex < 25)
	{
		str[strIndex++] = input[inputIndex++];
	}

	str[24] = '₩0';
	return str;
}

int isDecimalNum(char *input)
{
	if(input[0] == '0' && strlen(input) > 2) return 0;
	else return 1;
}

char*   processMultilineString(char* input)
{
	char* output; char* ptr;
	output = (char*)calloc(yyleng+1, sizeof(char));
	int inputIndex = 1, outputIndex =0;
```

```
	while(inputIndex <   yyleng)
	{
		if(input[inputIndex] == '₩₩' && input[inputIndex+1] == '₩n')
		{
			inputIndex += 2;
			lineCount++;
		}
		else
		{
			output[outputIndex++] = input[inputIndex++];
		}
	}

	output[strlen(output)-1] = '₩0';
	return output;

}

token getToken(int type)
{
	int i, tmp;
	token t;
	strcpy(t.name, "");
	strcpy(t.value, "");
	char *str; char* buf;

	switch(type)
	{
		case LPAREN:
			strcpy(t.name, "LPAREN");
			break;
		case RPAREN:
			strcpy(t.name, "RPAREN");
			break;
		case LBRAKET:
			strcpy(t.name, "LBRAKET");
			break;
		case RBRAKET:
			strcpy(t.name, "RBRAKET");
```

```
            break;
case COMMA:
            strcpy(t.name, "COMMA");
            break;
case SEMI:
            strcpy(t.name, "SEMI");
            break;
case PLUSOP:
            strcpy(t.name, "PLUSOP");
            break;
case MINUSOP:
            strcpy(t.name, "MINUSOP");
            break;
case DIVOP:
            strcpy(t.name, "DIVOP");
            break;
case MULTOP:
            strcpy(t.name, "MULTOP");
            break;
case GE:
            strcpy(t.name, "RELOP");
            strcpy(t.value, "GE");
            break;
case GT:
            strcpy(t.name, "RELOP");
            strcpy(t.value, "GT");
            break;
case LE:
            strcpy(t.name, "RELOP");
            strcpy(t.value, "LE");
            break;
case LT:
            strcpy(t.name, "RELOP");
            strcpy(t.value, "LT");
            break;
case EQ:
            strcpy(t.name, "RELOP");
            strcpy(t.value, "EQ");
            break;
case NE:
```

```
		strcpy(t.name, "RELOP");
		strcpy(t.value, "NE");
		break;
	case ASSIGNOP:
		strcpy(t.name, "ASSIGNOP");
		break;
	case ID:
		if(isKeyword(yytext)==-1)
		{
			strcpy(t.name, "ID");

			str = putSymbolTable(yytext);
			strcpy(t.value, str);
			free(str);
		}
		else
		{
			strcpy(t.name, keyword[isKeyword(yytext)].token);
		}
		break;
	case INTEGER:
		strcpy(t.name, "INTEGER");
		if( yyleng >= 16)
		{
			for(i=16; i >= 0; i--)
			{
				t.value[i] = yytext[yyleng-(16-i)];
			}
		}
		else strcpy(t.value, yytext);
		break;
	case DOUBLE:
		strcpy(t.name, "DOUBLE");
		str = cutDouble(yytext);
		strcpy(t.value, str);
		free(str);
		break;
	case STRING:
		strcpy(t.name, "STRING");
		buf = processMultilineString(yytext);
```

```
                        str = putStringTable(buf);
                        strcpy(t.value, str);
                        free(buf);
                        free(str);
                        break;
                case SINGLECOMMENT:
                        strcpy(t.name, "COMMENTS");
                        break;
                case MULTICOMMENT:
                        strcpy(t.name, "COMMENTS");
                        break;
                default:
                        break;
        }

        return t;
}

void printToken(token t)
{
        printf("<%s, %s>", t.name, t.value);
        printf("₩t₩t%s₩n", yytext);
}

void printSymbolTable()
{
        int i;
        char teststr[256];
        printf("index₩t₩tsymbols₩n");
        for(i=1; i<symbolIndex; i++)
        {
                printf("%d₩t₩t₩"%s₩"₩n", i, symbol_table[i]);
        }

        for(i=1; i<symbol_table_size; i++) free(symbol_table[i]);

        free(symbol_table);
}

void printStringTable()
```

```
{
	int i;
	printf("index₩t₩tstrings₩n");

	for(i=1; i<stringIndex; i++)
	{
		printf("%d₩t₩t%s₩n", i, string_table[i]);
	}

	for(i=1; i<stringIndex; i++) free(string_table[i]);
	free(string_table);
}

int main(int argc, char* argv[])
{
	initTable();
	if(argc > 1)
	{
		FILE* fp;
		fp = fopen(argv[1], "r");
		if(!fp)
		{
			fprintf(stderr, "ERROR: CANNOT OPEN FILE₩n");
			exit(1);
		}
		yyin = fp;
	}

	int val; token tok;
	printf("TOKEN₩t₩tLEXEME₩n");

	while((val=yylex())!= 0)
	{
		if(val==WHITESPACE)
		{
			lineCount += countNewLine(yytext);
		}
		else if(val == MULTICOMMENT)
		{
			lineCount += countNewLine(yytext);
```

```
                }
                else if(val == UNKNOWN)
                {
                        fprintf(stderr, "ERROR: [line:%d] CANNONT RECOGNIZE TOKEN: %s₩n",lineCount,
yytext);
                }
                else if(val == INTEGER)
                {
                        if(isDecimalNum(yytext))
                        {
                                tok = getToken(val);
                                printToken(tok);
                        }
                        else
                        {
                                fprintf(stderr, "ERROR: [line:%d] NOT DECIMAL NUMBER %s₩n",
lineCount, yytext);
                        }
                }
                else
                {

                        tok = getToken(val);
                        printToken(tok);
                }
        }
        printf("₩n₩n-------------------------SYMBOL-TABLE--------------------------₩n₩n");
        printSymbolTable();
        printf("₩n₩n-------------------------STRING-TABLE--------------------------₩n₩n");
        printStringTable();

        return 0;
}
```