컴파일러 1차 과제

교수님 성함 : 류기열 교수님
학과 : 소프트웨어학과
학번 : 201720723
이름 : 박수린
제출일 : 2019/04/01

1. 서론

1.1. 개요

이번 과제는 본 과제를 위해 정의된 Mini-C언어의 어휘를 분석하는 lexical analyzer을 flex을 이용하여 작성하는 것이었다. 과제 문서에 설명된 Mini-C 언어의 어휘 기준에 따라 keywords, integer, double, string, operators, comments, 기타 특수 문자들을 각각 분석하고 토큰 리스트, 심볼 테이블, 그리고 스트링 테이블을 출력하였다.

1.2. 구현하지 못한 부분

이번 과제에서 구현하지 못한 부분은 심볼테이블과 스트링 테이블을 만들 때 원래는 char형 이중 포인터를 이용하여 문자열 배열을 구현하려고 했으나, flex 오류가 발생하여 char* strings[100]의 형태로 구현하게 되었다. 그렇기 때문에 저장할 수 있는 string과 symbol의 개수가 정해져 있게 되는데, 다음 과제에서는 linked list을 사용하여 이 부분을 해결할 예정이다.

2. 문제 분석

2.1. token의 종류

2.1.1. Keywords

int, double, str, if, while, return (문자열 그 자체이다.)

2.1.2. ID

영문 대소문자, 숫자, _로 이루어져 있으나 시작은 대소문자와 _만 가능하고 _로만으로는 이루어질 수 없다. 또한 16글자까지 저장하여 구분한다.

2.1.3. INTEGER

십진수만을 사용하기 때문에 0으로 시작하는 8진수 입력법에 대해서는 고려하지 않았다. 최대 하위 10자리만 저장한다.

2.1.4. DOUBLE

C언어의 표기법을 따르기 때문에 .4 -.4 등의 입력 또한 고려하였다. 또한 exponential 입력도 고려하였다.

2.1.5. STRING

C언어의 표기법을 따르기 때문에 “ ”사이에 있는 문자들이 string table에 저장되도록 하였다.

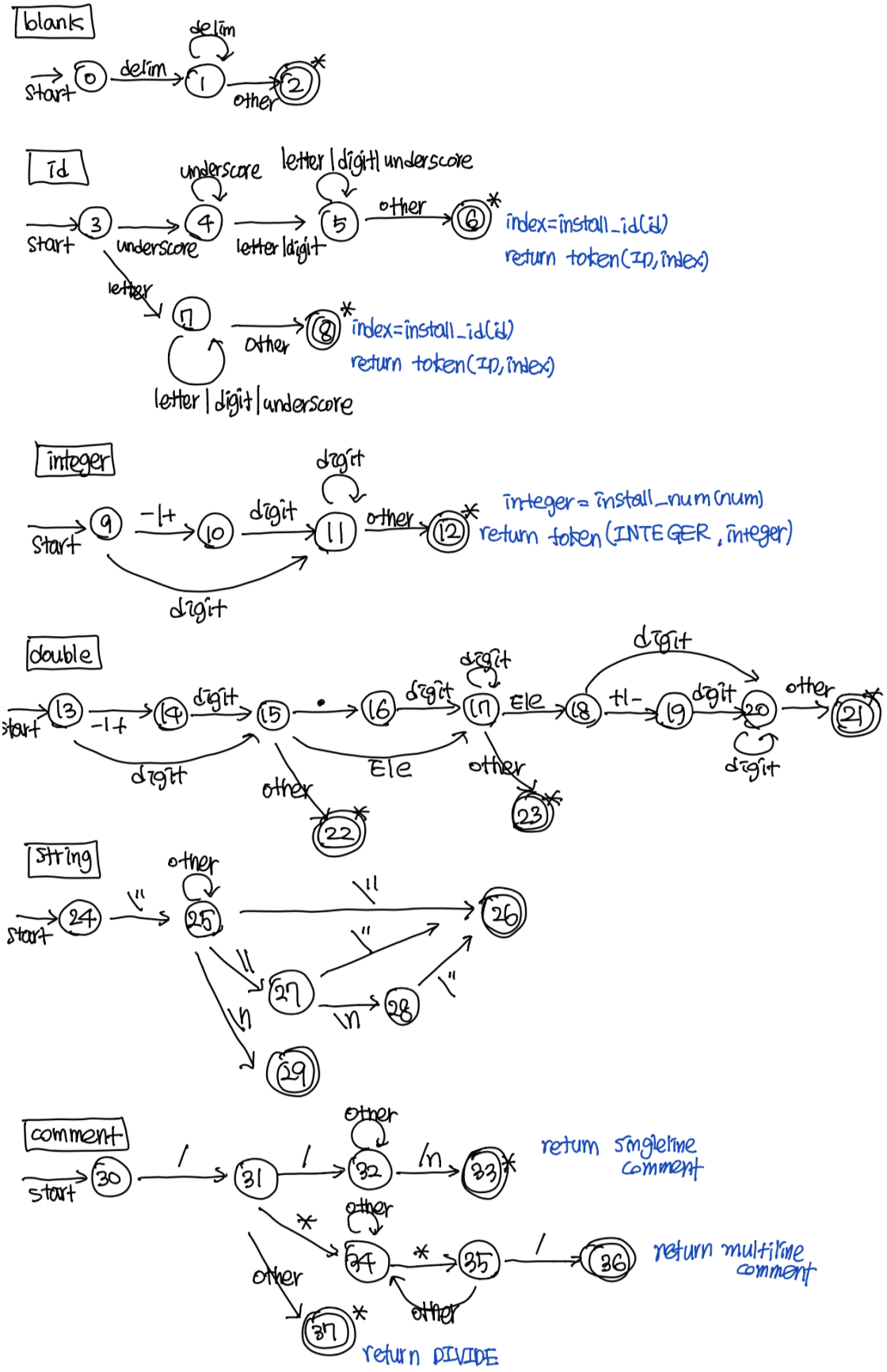
2.1.6. OPERATORS

+, -, *, /, =, >, >=, <, <=, ==, != (문자열 그 자체이다.) 1-2가 입력되면 1과 -2가 정수로 분석될 수 있도록 하였고 1 - 2 로 입력되면 1(integer)과 -(sub) 2(integer)로 각각 인식될 수 있도록 하였다.

2.1.7. SPECIAL

과제 문서에 명시되어있는 “,();{}을 각각 문자 그 자체로 인식한다.

2.2. transition diagram



2.3. 각 토큰을 위한 regular expression

blank [\t\n]+

id [a-zA-Z_]([0-9a-zA-Z_]?)*

integer [-+]?[0-9]+

double [-+]?(([0-9]+)|([0-9]*\.[0-9]+))([eE][-+]?[0-9]+)?

string "\""([^\n\"\\]*(\\[.\n])*)*"\""

comments (\/\/([^(\n)])*)|(\/*([^(*\/)])**\/)

3. 설계

- yyin을 이용하여 파일을 받아올 수 있다.

```
printf("==========TOKEN LIST==========\n");
printf("TOKEN\t\t%s\n","LEXEME");
while((tok = yylex())!=0){
        if(tok == INTEGER)
        {
                int len = strlen(yylval);
                if(len > 10)    yylval = yylval + (len/10-1) + (len%10);
                if(yylval[0] == '0')
                        printf("%s === ERROR! can't be started with 0!\n", yylval);
                else
                        printf("<INTEGER, %s>\t%s\n",yylval,yylval);
        }
        if(tok == DOUBLE) printf("<DOUBLE, %s>\t%s\n",yylval,yylval);
        if(tok == COMMENT) printf("<COMMENT>\t%s\n",yylval);
        if(tok == STRING)
        {
                printf("<STRING, %d>\t%s\n",compare_str(yylval),yylval);
        }
        if(tok == ID)
        {
                int id = compare_sy(yylval);
                printf("<ID, %d>\t\t%s\n",id,symbols[id-1]);
        }
}
```

- while((tok=yylex())!=0)을 이용해서 계속해서 토큰을 받아온다.
- token이 INTEGER일 경우에는 길이가 10보다 크다면 하위 10글자만 출력한다.
- INTEGER가 0으로 시작할 경우에는 error로 출력한다.
- string과 ID는 int compare_str(char* str)과 int compare_sy(char* str) 함수를 이용하여 이미 symbol table 과 string table에 같은 값을 가지고 있는지 살펴보고 같은 값이 있으면 그 문자열의 index를 없으면 문자열을 새로 동적할당하여 table에 추가해준 후 새로운 index를 반환해준다.

```
{id} {yylval = yytext; if(check_(yylval)==1) return ID; if(check_(yylval)==0) printf("%s === ERROR! only underscore!\n",yylval);}
```

- ID는 모두 _로 이루어질 수 없기 때문에 int check_(char* id) 함수를 이용하여 flex의 rules section에서 ID로 인식할 수 없도록 한다. (declare section에 함수를 선언해두었다.)

4. 입력 데이터

4.1. 정상적인 프로그램 (ex1.txt)

```
int double str if while return          //keyword들이 모두 포함되도록 하였다
_345 345abc
//underscore로 시작하는 ID (숫자로 시작되면 ID가 아니고 345(정수) abc(ID) 따로 인식되도록 하였다.
abc345                                  //abc345 는 소문자로 시작하므로 ID이다.
345 1-2 1 - 2 1111222233334444          //10자리 이상의 정수는 하위 10글자만 출력한다.
-0.45 3.45E2
```

```
"abcd"
( ;
"abcd" "foeooe"                    // 같은 문자열은 같은 index를 가진다.
f {
//ab   cd                          // 한줄 단위 comment
/*ab"c"d*/                         // 여러 줄 단위 comment (안의 " "는 문자열로 인식되지 않는다)
"abcd"";                    // " " 안은 문자열로 인식되고 이어서 나오는 " 한글자는 quote로 인식
10 <= 2 f = 2; if (a==b);
skfjaleijlse3433356789             // 16글자가 넘어가는 ID 표현
skfjaleijlse3433355677             // 앞의 16글자로 비교한다. 위의 ID 와 같음
_____333333333333333333333333
_____3333333333333333
_____________1"           //" " 사이에 줄 바꿈이 있으면 각자 quote로 인식한다.
_____________2
//16글자가 넘지만 underscore만으로 이루어지지 않았으면 상위 16개의 _만 ID로 저장된다.
/*fff
 dddd*/                            //   /**/ comment는 여러 줄도 인식해야한다
 "abcde\                            // " "안에 \가 오면 여러 줄을 인식할 수 있어야 한다.
 ksdfjle"
```

4.2. 오류가 포함된 프로그램 (ex2.txt)

```
123_d                        //123(정수)와 _d(ID)는 각각 인식되도록 한다.
_$%2200022                   //인식할 수 없는 문자는 에러를 출력한다.
00012                        //0으로 시작하는 십진수는 에러를 출력한다,
//* fjfd
/* gjkgk
"k"
*
*
kd
// ab
cd //
f = 12;
_________________            //underscore로만 이루어진 아이디는 에러를 출력한다,
____
"abcd\                       // "안에서 줄바꿈을 하려면 \n전에 바로 \가 와야한다."
dklfd
"                            //그렇지 않은 경우에는 앞부터 모두 다른 토큰으로 인식한다.
```

5. 결과 데이터

5.1. 토큰 리스트

5.1.1. 성공 프로그램 (ex1.txt)

```
==========TOKEN LIST==========
TOKEN           LEXEME
<INT, >         int
<DOUBLEKEY, >   double
<STR, >         str
<IF, >          if
<WHILE, >       while
<RETURN, >      return
<ID, 1>         inter
<ID, 2>         _345
<INTEGER, 345>  345
<ID, 3>         abc
<ID, 4>         abc345
<INTEGER, 345>  345
<INTEGER, 1>    1
<INTEGER, -2>   -2
<INTEGER, 1>    1
<SUB, >         -
<INTEGER, 2>    2
<INTEGER, 2233334444>   2233334444
<DOUBLE, -0.45> -0.45
<DOUBLE, 3.45E2>        3.45E2
<STRING, 1>     "abcd"
<LPAREN, >      (
<SEMICOLON, >   ;
<STRING, 1>     "abcd"
<STRING, 2>     "foeooe"
<ID, 5>         f
<LEFT, >        {
<COMMENT>       //ab  cd
<COMMENT>       /*ab"c"d*/
<STRING, 1>     "abcd"
<QUOTE, >       "
<SEMICOLON, >   ;
<INTEGER, 10>   10
<LEQ, >         <=
<INTEGER, 2>    2
<ID, 5>         f
<TO, >          =
<INTEGER, 2>    2
<SEMICOLON, >   ;
<IF, >          if
<LPAREN, >      (
<ID, 6>         a
<EQ, >          ==
<ID, 7>         b
<RPAREN, >      )
<SEMICOLON, >   ;
<ID, 8>         skfjaleijlse3433
<ID, 8>         skfjaleijlse3433
<ID, 9>         ______3333333333
<ID, 9>         ______3333333333
<ID, 10>                ________________
<QUOTE, >       "
<ID, 10>                ________________
<COMMENT>       /*fff
dddd*/
<STRING, 3>     "abcde\
ksdfjle"
```

5.1.2. 에러 프로그램 (ex2.txt)

```
(bagsulin-ui-MacBook-Pro-6:hw1_201720723 surin$ ./hw1 ex2.txt
==========TOKEN LIST==========
TOKEN           LEXEME
<INTEGER, 123>  123
<ID, 1>         _d
_ === ERROR! only underscore!
$ ===Error!
% ===Error!
<INTEGER, 2200022>      2200022
00012 === ERROR! can't be started with 0!
<COMMENT>       //* fjfd
<DIV, >         /
<MUL, >         *
<ID, 2>         gjkgk
<STRING, 1>     "k"
<MUL, >         *
<MUL, >         *
<ID, 3>         kd
<COMMENT>       // ab
<ID, 4>         cd
<COMMENT>       //
<ID, 5>         f
<TO, >          =
<INTEGER, 12>   12
<SEMICOLON, >   ;
___________________ === ERROR! only underscore!
_____ === ERROR! only underscore!
<QUOTE, >       "
<ID, 6>         abcd
\ ===Error!
<ID, 7>         dklfd
<QUOTE, >       "
```

5.2. 심볼 테이블

5.2.1. 성공 프로그램 (ex1.txt)

```
==========SYMBOL TABLE==========
index = 1       symbols = inter
index = 2       symbols = _345
index = 3       symbols = abc
index = 4       symbols = abc345
index = 5       symbols = f
index = 6       symbols = a
index = 7       symbols = b
index = 8       symbols = skfjaleijlse3433
index = 9       symbols = ______3333333333
index = 10      symbols = _________________
```

5.2.2. 에러 프로그램 (ex2.txt)

```
==========SYMBOL TABLE==========
index = 1       symbols = _d
index = 2       symbols = gjkgk
index = 3       symbols = kd
index = 4       symbols = cd
index = 5       symbols = f
index = 6       symbols = abcd
index = 7       symbols = dklfd
```

5.3. 스트링 테이블

5.3.1. 성공 프로그램 (ex1.txt)

```
==========STRING TABLE==========
index = 1       symbols = "abcd"
index = 2       symbols = "foeooe"
index = 3       symbols = "abcde\
ksdfjle"
```

5.3.2. 에러 프로그램 (ex2.txt)

```
==========STRING TABLE==========
index = 1       symbols = "k"
```

6. 소스 프로그램 (hw1.l)

```
%{
#define ID 1
#define INTEGER 2
#define DOUBLE 3
#define STRING 4
#define COMMENT 5
extern char* yylval;
int check_(char* id);
%}
blank [ \t\n]+
id [a-zA-Z_]([0-9a-zA-Z_]?)*
integer [-+]?[0-9]+
double [-+]?(([0-9]+)|([0-9]*\.[0-9]+))([eE][-+]?[0-9]+)?
string "\""([^\n\"\\]*(\\[.\n])*)*"\""
comments (\/\/([^(\n)])*)|(\/\*([^(\*\/)])*\*\/)
%%
{blank} ;
"int" {printf("<INT, >\t\t%s\n","int");}
"double" {printf("<DOUBLEKEY, >\t%s\n","double");}
"str" {printf("<STR, >\t\t%s\n","str");}
"if" {printf("<IF, >\t\t%s\n","if");}
"while" {printf("<WHILE, >\t%s\n","while");}
"return" {printf("<RETURN, >\t%s\n","return");}
{integer} {yylval = yytext; return INTEGER;}
{double} {yylval = yytext; return DOUBLE;}
{comments} {yylval = yytext; return COMMENT;}
{string} {yylval = yytext; return STRING;}
{id} {yylval = yytext; if(check_(yylval)==1) return ID; if(check_(yylval)==0) printf("%s === ERROR!
only underscore!\n",yylval);}
\" {printf("<QUOTE, >\t%s\n","\"");}
\, {printf("<COMMA, >\t%s\n",",");}
\( {printf("<LPAREN, >\t%s\n","(");}
\) {printf("<RPAREN, >\t%s\n",")");}
\; {printf("<SEMICOLON, >\t%s\n",";");}
\{ {printf("<LEFT, >\t%s\n","{");}
\} {printf("<RIGHT, >\t%s\n","}");}
\+ {printf("<ADD, >\t\t+\n");}
\- {printf("<SUB, >\t\t-\n");}
\* {printf("<MUL, >\t\t*\n");}
\/ {printf("<DIV, >\t\t/\n");}
\= {printf("<TO, >\t\t=\n");}
\> {printf("<GT, >\t\t>\n");}
```

```
">=" {printf("<GEQ, >\t\t>=\n");}
\< {printf("<LT, >\t\t<\n");}
"<=" {printf("<LEQ, >\t\t<=\n");}
"==" {printf("<EQ, >\t\t==\n");}
"!=" {printf("<NEQ, >\t\t!=\n");}
. {ECHO; printf(" ===Error!\n");}
%%
char* yylval;
#include <stdio.h>
#include <string.h>
extern FILE* yyin;
extern int yylex();

int index_str = 0;
int index_sy = 0;
char* strings[100];
char* symbols[100];

int check_(char* id)
{
	int len = strlen(id);
	int i;
	for(int i = 0; i<len; i++)
	{
		if(id[i] != '_')
			return 1;
	}
	return 0;
}

int compare_str(char* str)
{
	int i;
	for(i = 0; i<index_str; i++)
	{
		if(strcmp(strings[i], str) == 0)
			return i+1;
	}
	strings[index_str] = malloc(sizeof(char)*(strlen(str)+1));
	strcpy(strings[index_str], yylval);
	return ++index_str;
}
```

```
int compare_sy(char* str)
{
	int i;
	for(i = 0; i<index_sy; i++)
	{
		if(strncmp(symbols[i], str, 16) == 0)
			return i+1;
	}
	symbols[index_sy] = malloc(sizeof(char)*17);
	strncpy(symbols[index_sy], yylval, 16);
	return ++index_sy;
}

int main(int argc, char* argv[])
{
	if(argc>1)
	{
		FILE* file;
		file = fopen(argv[1], "r");
		if(!file)
		{
			fprintf(stderr, "could not open %s!\n",argv[1]);
			exit(1);
		}
		yyin = file;
	}
	int tok;
	int i;
	printf("==========TOKEN LIST==========\n");
	printf("TOKEN\t\t%s\n","LEXEME");
	while((tok = yylex())!=0){
		if(tok == INTEGER)
		{
			int len = strlen(yylval);
			if(len > 10)		yylval = yylval + (len/10-1) + (len%10);
			if(yylval[0] == '0')
				printf("%s === ERROR! can't be started with 0!\n", yylval);
			else
				printf("<INTEGER, %s>\t%s\n",yylval,yylval);
		}
		if(tok == DOUBLE) printf("<DOUBLE, %s>\t%s\n",yylval,yylval);
		if(tok == COMMENT) printf("<COMMENT>\t%s\n",yylval);
		if(tok == STRING)
```

```
                {
                        printf("<STRING, %d>\t%s\n",compare_str(yylval),yylval);
                }
                if(tok == ID)
                {
                        int id = compare_sy(yylval);
                        printf("<ID, %d>\t\t%s\n",id,symbols[id-1]);
                }
        }
        printf("==========SYMBOL TABLE==========\n");
        for(i = 0; i<index_sy; i++)
                printf("index = %d \t symbols = %s\n",i+1,symbols[i]);
        printf("==========STRING TABLE==========\n");
        for(i = 0; i<index_str; i++)
                printf("index = %d \t symbols = %s\n",i+1,strings[i]);
        return 0;
}
```