21001715 컴파일러
Programming Assignment #1
Due: 2020년 4월 7일 23:59PM

Flex 를 이용하여 C 소스코드를 위한 Lexical Analyzer 만들기

1. Token

토큰이란, 소스코드에서 의미를 가진 최소 길이의 문자열들이다. 예를 들어, C 언어에서 다음과 같은 코드

```
char ch='A';
```

가 있다면, 이는 5개의 토큰 **`char`**, **`ch`**, **`=`**, **`'A'`**, **`;`** 으로 나누어질 수 있다. 본 프로젝트에서는 C 소스코드를 Identifier, Keyword, 문자상수, 문자열상수, 정수상수, 주석, 연산자, punctuation symbol들로 분해하고, 그 결과를 화면에 출력하는 프로그램을 작성한다.

(1) Identifier

Identifier란, 변수 이름이나 함수 이름을 나타내는 문자열로, 다음 조건을 만족한다.
- 대문자, 소문자, 또는 밑줄 _ 로 시작 가능
- 두 번째 문자부터는 대문자, 소문자, 밑줄 _, 숫자 모두 가능
- Keyword는 제외.

(2) Keyword

본 프로젝트에서 구분하는 keyword의 종류는 다음과 같이 13종류가 있다.

`break`	**`case`**	**`char`**	**`default`**	**`else`**	**`float`**	**`for`**
`if`	**`int`**	**`return`**	**`switch`**	**`void`**	**`while`**	

(3) Character constant

본 프로젝트에서 분리해야 할 문자 상수는 다음과 같다.

`'A'`	**`'B'`**	**`'C'`**	**`'D'`**	**`'E'`**	**`'F'`**	**`'G'`**	**`'H'`**	**`'I'`**	**`'J'`**	**`'K'`**	**`'L'`**	**`'M'`**
`'N'`	**`'O'`**	**`'P'`**	**`'Q'`**	**`'R'`**	**`'S'`**	**`'T'`**	**`'U'`**	**`'V'`**	**`'W'`**	**`'X'`**	**`'Y'`**	**`'Z'`**
`'a'`	**`'b'`**	**`'c'`**	**`'d'`**	**`'e'`**	**`'f'`**	**`'g'`**	**`'h'`**	**`'i'`**	**`'j'`**	**`'k'`**	**`'l'`**	**`'m'`**
`'n'`	**`'o'`**	**`'p'`**	**`'q'`**	**`'r'`**	**`'s'`**	**`'t'`**	**`'u'`**	**`'v'`**	**`'w'`**	**`'x'`**	**`'y'`**	**`'z'`**
`'0'`	**`'1'`**	**`'2'`**	**`'3'`**	**`'4'`**	**`'5'`**	**`'6'`**	**`'7'`**	**`'8'`**	**`'9'`**			
`'\''`	**`'\"'`**	**`'\?'`**	**`'\\'`**	**`'\a'`**	**`'\b'`**	**`'\f'`**	**`'\n'`**	**`'\r'`**	**`'\t'`**	**`'\v'`**	**`'\0'`**	
`'!'`	**`'#'`**	**`'%'`**	**`'^'`**	**`'&'`**	**`'*'`**	**`'_'`**	**`'('`**	**`')'`**	**`'-'`**	**`'+'`**	**`'='`**	**`'~'`**
`'['`	**`']'`**	**`'|'`**	**`';'`**	**`':'`**	**`'{'`**	**`'}'`**	**`','`**	**`'.'`**	**`'<'`**	**`'>'`**	**`'/'`**	

(4) String constant

문자 상수는 두 개의 " 와 그 사이에 있는 character 또는 escape character들의 열을 뜻한다. 예를 들어, **`"She says, \"Hello world!\"\n"`** 는 하나의 문자열이다.

(5) Integer constant

C 언어에서 정수 상수는 10진수, 8진수, 16진수로 표현 가능하다.
- 10진수: `0` 단독으로 존재하거나, `0`이 아닌 숫자로 시작하고, 그 이후는 `0`을 포함한 모든 숫자들이 등장할 수 있다.
- 8진수: `0`으로 시작하여, 그 다음부터는 `0`부터 `7`까지의 숫자들이 반복된다. (예를 들어 `0123`은 8진수 표현이며, 10진수로 83이다.)
- 16진수: `0x`나 `0X`로 시작하여, 그 다음부터는 숫자들이나 `A`부터 `F`까지, `a`부터 `f`까지의 알파벳들이 등장할 수 있다. 대소문자는 혼용 가능하다. (예를 들어 `0x1Af`는 16진수 표현이며, 10진수로 431이다.)

(6) Real constant

C 언어에서 실수 상수는 다음과 같은 세 가지 형태로 표현될 수 있다
`[숫자들].[숫자들]`
`[숫자들].[숫자들][e 또는 E][숫자들]`
`[숫자들].[숫자들][e 또는 E][+ 또는 -][숫자들]`

예를 들어, `0.0`, `3.14`, `54.12e9`, `391.132E-21` 이 실수상수이다.

(7) Comment

C 언어에서 주석은 두 가지로, `//`로 시작되는 한 줄짜리 주석과 `/*`로 시작하여 `*/`로 끝나는 한 줄 이상의 주석이 있다.

(8) Operator

본 프로젝트에서는 다음과 같은 23개의 연산자들을 읽어들일 수 있어야 한다.

`+`	`-`	`*`	`/`	`%`	`!`	`=`	`<`	`>`	`+=`	`-=`	`*=`
`/=`	`%=`	`==`	`!=`	`<=`	`>=`	`&&`	`||`	`++`	`--`	`,`	

(9) Punctuation symbol

본 프로젝트에서는 다음과 같은 8개의 puctuation symbol들을 읽어들일 수 있어야 한다.

`(`	`)`	`{`	`}`
`[`	`]`	`;`	`:`

(10) Error

위의 어떤 규칙도 적용할 수 없는 경우에는 에러를 발생시킨다. 예를 들어 `a = '4;` 라는 코드가 있으면 적용할 수 없기 때문에 에러를 발생시키고 종료시킨다. 만일 `a = 4k;` 라고 하면 에러를 발생시키지 않는데, 그 이유는 `a`, `=`, `4`, `k`, `;` 로 토큰 분해가 가능하기 때문이다. 이 때는 다음에 하게 될 문법 검사에서 에러를 발생시킬 것이다.

2. Your mission

Flex를 이용하여 1에서 소개했던 토큰들을 분리할 수 있는 lexical analyzer를 만드는 것이다. 출력은 다음과 같은 형식을 따른다:

1) 토큰이 정수 상수, 문자 상수일 때:

우선, 토큰 값을 **yylval.iVal** 이라는 변수에 넣고, 다음 명령을 이용하여 출력한다:

```
printf("%-15s:%19s,%3d,%3d,%5d\n", yytext, tokenType[2 또는 4], line, pos,
                                   yylval.iVal );
```

2) 토큰이 실수 상수일 때:

우선, 토큰 값을 **yylval.rVal** 이라는 변수에 넣고, 다음 명령을 이용하여 출력한다:

```
printf("%-15s:%19s,%3d,%3d,%5.2f\n", yytext, tokenType[5], line, pos,
                                   yylval.rVal );
```

3) 그 외의 토큰인 경우:

다음 명령을 이용하여 출력한다:

```
printf("%-15s:%19s,%3d,%3d\n", yytext, tokenType[다른 숫자], line, pos);
```

4) 에러인 경우:

다음 명령을 이용하여 출력한다 (. . . 에는 여러분이 적절한 변수들을 넣을 것):

```
printf("Error occurred in Line %d, Position %d\n", ...);
```

Line 또는 변수 **line**은 소스코드의 위에서부터 몇 번째 줄에 있는지에 대한 값이다. 맨 위를 1번째라 가정한다.

Position 또는 변수 **pos**는 소스코드의 왼쪽에서부터 몇 번째 칸에 있는지에 대한 값이다. 맨 왼쪽을 1번째라 하며, 탭 하나는 1 칸이라 가정한다.

<실시 예>

hello.c 파일의 내용이 다음과 같다고 하자:

```
int main(void){
      int _a1;
      float _b1;
      _a1 = '\n'+01024;
      _b1 = 40.12359E+2;
      printf("Hello World!\n");
      return 0;
}
```

Lexical analyzer 파일 이름이 **token**일 때, 실행 결과는 다음과 같다.

```
$ ./token hello.c
int             :             Keyword,  1,  1
main            :          Identifier,  1,  5
(               : Punctuation symbol,  1,  9
void            :             Keyword,  1, 10
)               : Punctuation symbol,  1, 14
{               : Punctuation symbol,  1, 15
```

```
int             :            Keyword,  2,  2
_a1             :         Identifier,  2,  6
;               : Punctuation symbol,  2,  9
float           :            Keyword,  3,  2
_b1             :         Identifier,  3,  8
;               : Punctuation symbol,  3, 11
_a1             :         Identifier,  4,  2
=               :           Operator,  4,  6
'\n'            :          Character,  4,  8,   10
+               :           Operator,  4, 12
01024           :            Integer,  4, 13,  532
;               : Punctuation symbol,  4, 18
_b1             :         Identifier,  5,  2
=               :           Operator,  5,  6
40.12359E+2     :        Real number,  5,  8,4012.36
;               : Punctuation symbol,  5, 19
printf          :         Identifier,  6,  2
(               : Punctuation symbol,  6,  8
"Hello World!\n":             String,  6,  9
)               : Punctuation symbol,  6, 25
;               : Punctuation symbol,  6, 26
return          :            Keyword,  7,  2
0               :            Integer,  7,  9,    0
;               : Punctuation symbol,  7, 10
}               : Punctuation symbol,  8,  1
```

만일 위 소스 코드에서 `'\n'` 대신 `'\n` 이라고 쓰면 다음과 같이 출력될 것이다:

```
int             :            Keyword,  1,  1
main            :         Identifier,  1,  5
(               : Punctuation symbol,  1,  9
void            :            Keyword,  1, 10
)               : Punctuation symbol,  1, 14
{               : Punctuation symbol,  1, 15
int             :            Keyword,  2,  2
_a1             :         Identifier,  2,  6
;               : Punctuation symbol,  2,  9
float           :            Keyword,  3,  2
_b1             :         Identifier,  3,  8
;               : Punctuation symbol,  3, 11
_a1             :         Identifier,  4,  2
=               :           Operator,  4,  6
Error occurred in Line 4, Position 8
```

3. 파일 설명

`foo.l` : 작성해야 할 파일 이름. 무조건 **`foo.l`** 파일을 만들어서 이 파일에 소스코드를 작성해야 한다.

`Makefile` : **`make`** 명령을 위한 매크로 파일. 절대로 수정하거나 지우지 말 것!
원래, **`foo.l`** 파일로 lexical analyzer를 만드려면 **`flex foo.l`** 과 **`gcc -o token lex.yy.c -ll`** 을 연속으로 실행해야 하는데, **`Makefile`** 파일이 있으면 이것 대신에 **`make`**라는 명령 한 번으로 위의 두 명령이 순차적으로 실행되게 할 수 있다.
소스코드에 아무 문제가 없으면 **`lex.yy.c`**, **`lex.yy.o`**, **`token`** 파일들이 차례로 생성된다. 이들을 지우고 싶으면 **`make clean`** 명령을 내려주면 된다.

`hello.c` : **`foo.l`** 파일을 완성하고 난 뒤 테스트해보기 위한 파일. 위에 써 있는 실행결과가 나오면 어느정도 성공한 것이라 보면 된다. 하지만, 실제 채점에서는 더 많은 C 소스코드로 테스트 해 보므로, **`hello.c`** 파일을 시험해 봤을 때 위의 결과가 나왔다고 안심하지 말고, 여러 가지 소스코드를 만들어서 추가적으로 테스트를 수행해 보는 것을 추천한다.

4. 제출

- 4월 7일 화요일 23:59까지. 시계는 제출용 서버의 시계를 따른다.

- 여러분의 계정에 **`PA1`** 디렉토리가 있고, 이 안에 **`foo.l`**, **`Makefile`**, **`hello.c`** 파일들이 있을 것이다. 이 중 **`foo.l`** 파일의 내용을 채우면 된다.

- 만일 위의 파일들 중 어느 하나를 지웠으면 담당 교수에게 문의할 것.

- 하루씩 Delay 될 때마다 점수의 20%를 감점한다. (예를 들어, 이틀 Delay하고 8점을 받았으면 4.8점 획득)

- Copy는 해당 PA 0점. 본인 소스코드는 반드시 본인이 모두 타이핑 하여 만들어야 함. 오해를 방지하기 위하여 둘 이상이 같이 의논하여 짰더라도 최대한 달라보이게 짤 것! (변수 바꾸기, 주석 더 넣기, 들여쓰기 바꾸기 등 만으로는 안 됨)