컴파일러: Project 1,lexical analyzer (Due: 4/4 23:59)

이 프로젝트의 목적은 lex utility를 이용하여 C^{+-}(C의 feature를 가감한; subc라 부른다)에 대한 lexical analyzer를 만드는 데 있다. lex input인 subc.l을 만들고, lex를 이용하여 lexical analyzer를 만든 뒤, subc로 작성된 program에서 모든 token들을 찾아내서 output으로 print 한다.

Lexical Structure of subc

- **Definitions**

letter	A-Z, a-z, _
digit	0-9
white space	‘ ‘, \t, \n,
integer-constant	$-?\{digit\}^+$
float-constant	$-?\{digit\}^+ . \{digit\}^* ([eE][+-]? \{digit\}^+)?$

- **Operators**

  ( ), [ ], { }, ->, ., ,, .., \, !, ~, ++, --, *, /, %, +, -,
  <<, >>, <, <=, >, >=, ==, !=, &, ^, |, &&, ||, ?, :, ;,
  =, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=

- **Comments**

  /* ... */ (Nesting을 허용하므로 /*Hong/*lastname*/Kil-Dong*/ is OK)

- **Keywords**

‘auto’,‘break’,‘case’,‘char’,‘continue’,‘default’,‘do’,‘double’,‘else’,
‘extern’,‘float’,‘for’,‘goto’,‘if’,‘int’,‘long’,‘register’,‘return’,‘short’
‘sizeof’,‘static’,‘struct’,‘switch’,‘typedef’,‘union’,‘unsigned’,‘void’,‘while’

A Few Notes

1. Identifier and Key words

- Identifier와 Keyword에 대하여 다른 pattern(regular expression)을 사용하여 recognize 한다.
- Identifier 보다 Keyword pattern이 더 우선하도록 keyword pattern 을 identifier pattern 보다 먼저 (위에) 정의한다. [Precedence Rule]
- Lex 에서 제공하는 yytext 사용하여 각 토큰에 해당하는 lexeme 출력한다.

2. Comments C의 comment는 nesting이 허락되지 않으나 이번 숙제에는 nesting이 가능한 comment를 recognize 해야 한다. lex의 start condition을 이용하여 normal mode와 comment mode를 구별하고 normal mode로 lex가 시작된다. Input 중에 “/*” 를 읽는 순간 comment mode가 시작되면서 comment mode에서는 세가지 동작이 이루어진다: (1) 또 다른 “/*”를 읽게 되면 depth count를 증가시키고 (2) “*/” 를 읽게 되면 depth count를 감소시키며 만약 depth count가 0이 되면 normal mode가 시작된다. (3) 그 외의 모든 character들은 무시된다. Lex가 처음 시작될 때 normal mode에서 시작되기 위해 main()을 lex input의 procedure section에 넣고 main()에서 start mode를 set 한다. (main()에는 yylex() call도 있어야 함)

3. DOTDOT and Float Const 우리의 float const의 정의는 C의 정의와 약간 다르다. (.325를 허락 안함) 1..2는 “float (1.)” “.” “integer” 로 이해되면 안되고 “integer” “..” “integer”로 해석되어 range를 나타낼 수 있어야 한다. 이를 위해 lex의 lookahead operator(/) 를 이용한다.

Your Lex Program

주어진 input subc 프로그램에서 위의 모든 token들을 recognize하여 standard out에 다음과 같이 print한다. “token type”, “lexeme”

1. Makefile template

```
subc:  lex.yy.o
       gcc –o subc lex.yy.o –ll
lex.yy.o:  lex.yy.c
       gcc –c –g lex.yy.c
lex.yy.c:  subc.l
       lex subc.l
```

2. subc.l template

```
%{
#include “subc.h”
int commentdepth = 0; /* depth of comment nesting */
%}
...
%%
..
%%
main()
{
...
yylex();
...
}
```