The ability of my scanner

我的 scanner 主要的功能就是分析出 P language 中的各種 token，並針對不同類型的 token 進行分類。按照 spec，token 主要分為兩種類型，一類會被 scanner 傳遞給 parser，另一類則會被 scanner 直接丟棄而不會再傳遞給 parser。

1. 會被傳到 parser 的 token 類型

Delimiters

lextemplate.l 當中寫法如下：

```
","    {tokenChar(',');}  ← 會在 terminal 上印出<,>
";"    {tokenChar(';');}
":"    {tokenChar(':');}
"("    {tokenChar('(');}
")"    {tokenChar(')');}
"["    {tokenChar('[');}
 "]"    {tokenChar(']');}
```

comma	,
semicolon	;
colon	:
parentheses	( )
square brackets	[ ]

Arithmetic, Relational, and Logical Operators

lextemplate.l 當中寫法如下：

```
"+"    {token(+);}  ← 會在 terminal 上印出<+>
"-"    {token(-);}
"*"    {token(*);}
"/"    {token(/);}
"mod"    {token(mod);}
":="    {token(:=);}
"<"    {token(<);}
"<="    {token(<=);}
"<>"    {token(<>);}
">="    {token(>=);}
">"    {token(>);}
"="    {token(=);}
"and"    {token(and);}
"or"    {token(or);}
 "not"    {token(not);}
```

addition	+
subtraction	-
multiplication	*
division	/ mod
assignment	:=
relational	< <= <> >= > =
logical	and or not

Keywords

lextemplate.l 當中寫法如下(僅舉五個例子)：

```
"array"       {token(KWarray);}  ← 會在 terminal 上印出<KWarray>
"begin"       {token(KWbegin);}
"boolean"     {token(KWboolean);}
"def"         {token(KWdef);}
"do"          {token(KWdo);}
```

Identifiers

lextemplate.l 當中寫法如下：

```
digit    [0-9]  ← 先行定義 digit 與 letter 的含意
letter [a-zA-Z_]
%%
{letter}({letter}|{digit})*    {tokenString(id,yytext);}  ← 能夠將 ident,Ident,IDENT 視為不同的 id,在 terminal 上印出<id: a>
%%
```

Integer Constants

lextemplate.l 當中寫法如下：

```
-?0|[1-9][0-9]*    {tokenString(integer,yytext);}  ← decimal,在 terminal 上印出<integer: 1>
0[1-7][0-7]*      {tokenString(oct_integer,yytext);}  ← octal,在 terminal 上印出<oct_integer: 01>
```

Floating-Point Constants

lextemplate.l 當中寫法如下：

```
-?{digit}+\.{digit}+    {tokenString(float,yytext);}  ← float,在 terminal 上印出<float: 1.23>
```

Scientific Notations

lextemplate.l 當中寫法如下：

```
-?{digit}+(\.{digit}+)?[Ee][-+]?{digit}+ {tokenString(scientific,yytext);}  ← 科學記號 ,在 terminal 上印出<scientific: 1.23e-4>
```

String Constants

lextemplate.l 當中寫法如下：

```
%s str  ← 建立一個叫做 str 的 state
%%
"\""    {LIST;BEGIN str;}
<str>.*\"      {
                  pch=strtok(yytext,"\"");  ← 去除字串最後面的 ”
                  tokenString(string,pch);  ← 在 terminal 上印出<string: hello world>
                  strcat(buf, "\"");  ← 重新將 ” 連接到字串的最後面
                  BEGIN 0;
               }
%%
```

2. 不會被傳到 parser 的 token 類型

Whitespace

lextemplate.l 當中寫法如下：

```
[ \t]     {LIST;}  ← space and tab
```

```
\n          {           ← newline
                LIST; ← 即為 strcat(buf, yytext);
                if (Opt_S)
                   printf("%d: %s", linenum, buf); ← source program listing
                linenum++;
                buf[0] = '\0'; ← 清空 buffer
            }
```

Comments

lextemplate.l 當中寫法如下：

```
%s com1 com2 ← 建立 com1 及 com2 兩個 states
%%
   /*   C-style   */
"/*" {LIST;BEGIN com2;} ← 看見 /* 進入 com2 state
<com2>.*\*\/ {          ← 在 com2 state 遇到任意字元並以 */ 作結，返回 INITIAL state
                  LIST;
                  BEGIN 0;
                 }
<com2>.*     {          ← 在 com2 state 遇到任意字元，繼續留在 com2 state
                  LIST;
                  BEGIN com2;
                 }
<com2>\n {               ← 在 com2 state 遇到 \n，將同一行的內容都印出來
                LIST;
                if (Opt_S)
                   printf("%d: %s", linenum, buf);
                linenum++;
                buf[0] = '\0';
                BEGIN com2;
            }
   /*   C++-style   */
"//" {LIST;BEGIN com1;} ← 看見 // 進入 com1 state
<com1>.* {LIST;BEGIN com1;} ← 在 com1 state 遇到任意字元，繼續留在 com1 state
<com1>\n {                  ← 在 com1 state 遇到 \n，將同一行的內容都印出來，並返回 INITIAL state
                LIST;
                if (Opt_S)
                   printf("%d: %s", linenum, buf);
                linenum++;
                buf[0] = '\0';
                BEGIN 0;
```

```
            }
%%
```

PseudoComments

lextemplate.l 當中寫法如下：

%s com3 ← 建立 **com3 state**

%%

"//&S-" {LIST;Opt_S = 0;BEGIN com3;} ← 看見 **//&S-** 進入 **com3 state, Opt_S** 改為 **0,**不印 **source program**

"//&S+" {LIST;Opt_S = 1;BEGIN com3;} ← 看見 **//&S+** 進入 **com3 state, Opt_S** 改為 **1,**印出 **source program**

"//&T-" {LIST;Opt_T = 0;BEGIN com3;} ← 看見 **//&T-** 進入 **com3 state, Opt_T** 改為 **0,**不印 **token**

"//&T+" {LIST;Opt_T = 1;BEGIN com3;} ← 看見 **//&T+** 進入 **com3 state, Opt_T** 改為 **1,**印出 **token**

<com3>.* {LIST;BEGIN com3;} ← 在 **com3 state** 遇到任意字元，繼續留在 **com3 state**

<com3>\n { ← 在 **com3 state** 遇到 **\n**，將同一行的內容都印出來，並返回 **INITIAL state**

```
            LIST;
            if (Opt_S)
               printf("%d: %s", linenum, buf);
            linenum++;
            buf[0] = '\0';
            BEGIN 0;
          }
%%
```

The platform to run my scanner

我使用 VMware Workstation，並在其上裝設 ubuntu14.04LTS。所以只要在 terminal 輸入必要的三個指令，就能使我的 scanner 在 Linux 作業系統的環境下執行。

How to run my scanner

方法一：

1. 開啟 terminal，依序輸入以下三個指令：

```
% lex lextemplate.l
(亦可輸入：% flex lextemplate.l)
% gcc -o scanner lex.yy.c -lfl
% ./scanner [input file]
```

方法二：

1. 製作一個 Makefile 檔案，內容如下：

```
scanner: lex.yy.c
        gcc -o scanner lex.yy.c -lfl
lex.yy.c: lextemplate.l
```

lex lextemplate.l

2. 開啟 terminal，依序輸入以下兩個指令：

   % make -f Makefile

   % ./scanner [input file]