

## 1. Print Source Code

這是最基本的功能，將讀進來的 Source Code 一行一行的印出來，首先宣告以下的變數：

```
3  %{
4      #include <stdio.h>
5      #include <string.h>
6      #include <math.h>
7
8      int numL = 1;
9      char buf[1000];
10     int bufIndex = 0;
11
12     int tokenOn = 1;
13     int sourceOn = 1;
14  %}
```

numL 為目前的行數，buf 則是存放在該行的 code 等待印出，最後一個 bufIndex 則是記錄著目前這行 code 的長度。在我所做的 scanner 中，每個 match 到 pattern 的字串都會被存在 buffer 之中，以下提供一個例子：

```
57  "/*"    {
58          BEGIN(COMMENT);
59
60          strcpy(&(buf[bufIndex]), yytext);
61          bufIndex += yyleng;
62      }
```

這邊可以看到，只要 match 到 "/\*" 這個 pattern，該字串就會被放到 buf 之中，而 bufIndex 也會更新，而其他的 pattern 也會有一樣的動作。最後當遇到換行的時候，會將該行的 code 印出：

```

189  \n  {
190
191      /* Check pragma */
192      if(sourceOn == 1)  printf("%d:%s\n",numL,buf);
193
194      numL++; /* Enter newline */
195
196      buf[0] = 0; /* Clean Buffer */
197      bufIndex = 0;
198
199  }

```

將該行 code 印出以後，會增加 numL，代表進入下一行，並且將 buf[0] 設為 null character，以防被其他 pattern 印出這行的 code，並重設 bufIndex 為 0。

## 2. Pragma Directives

為了控制是否印出 source code 或 token，首先宣告兩個變數：

```

3  %{
4      #include <stdio.h>
5      #include <string.h>
6      #include <math.h>
7
8      int numL = 1;
9      char buf[1000];
10     int bufIndex = 0;
11
12     int tokenOn = 1;
13     int sourceOn = 1;
14  %}

```

tokenOn 等於 1 時代表要印出 token，sourceOn 等於 1 時代表要印出 source code，而只要這兩個變數不是 1，則代表不需要印

出，因此在宣告時，預設都為 1 那就是預設 token 和 source code 都是要印出來的。而接下來在每一個 pattern 中，都會檢查 tokenOn 或 sourceOn 的值，來決定是不是要印出資訊，以下提供一個檢查 tokenOn 例子：

```
94 {KEY} {
95     /* Check pragma */
96     if(tokenOn == 1)    printf("#key:%s\n",yytext);
```

以及換行要印出 source code 時，檢查 sourceOn：

```
189 \n {
190
191     /* Check pragma */
192     if(sourceOn == 1)    printf("%d:%s\n",numL,buf);
```

那麼如何在 input 中識別出 pragma directives 呢？

首先我們先定義一些 pattern：

```
43 SPACE    [ \f\t\r\v]+
44
45 SOURCE_ON  #{SPACE}?"pragma"{SPACE}"source"{SPACE}"on"
46 SOURCE_OFF #{SPACE}?"pragma"{SPACE}"source"{SPACE}"off"
47 TOKEN_ON   #{SPACE}?"pragma"{SPACE}"token"{SPACE}"on"
48 TOKEN_OFF  #{SPACE}?"pragma"{SPACE}"token"{SPACE}"off"
```

這樣一來，當我們在 input 中遇到符合這些 pattern 的 code 時：

```
52 {SOURCE_ON}    { sourceOn = 1; strcpy(&(buf[bufIndex]), yytext); bufIndex += yyleng;}
53 {SOURCE_OFF}   { sourceOn = 0; strcpy(&(buf[bufIndex]), yytext); bufIndex += yyleng;}
54 {TOKEN_ON}     { tokenOn = 1;  strcpy(&(buf[bufIndex]), yytext); bufIndex += yyleng;}
55 {TOKEN_OFF}    { tokenOn = 0;  strcpy(&(buf[bufIndex]), yytext); bufIndex += yyleng;}
```

我們就能更改 sourceOn 和 tokenOn 的值，以達到控制的目的。

### 3. Comment

遇到 comment 時，只要直接印出該行的 source code 就好，但因為有其他的 pattern 可能會不小心 match 到，因此我們需要使用 exclusive start condition，首先定義：

```
16 %x COMMENT
```

有了 start condition 以後，當我們遇到 **comment** 的開頭 **"/\*\*"** 時，便進入了 COMMENT 的 condition 之中：

```
57  "/**"    {
58          BEGIN(COMMENT);
59
60          strcpy(&(buf[bufIndex]), yytext);
61          bufIndex += yyleng;
62      }
```

在 condition 之中遇到除了換行以外的字元，我們就將它放到 buf 中等待印出：

```
81  <COMMENT>. {
82      /* If use .* then it will eat the ending of comment */
83
84      strcpy(&(buf[bufIndex]), yytext);
85      bufIndex += yyleng;
86  }
```

在 condition 之中遇到換行字元，將該行的 comment 印出，並增加記錄行數的 numL，重設 buf 和 bufIndex，繼續讀取 comment：

```
70  <COMMENT>\n {
71      /* Comment newline print it out */
72
73      /* Check pragma */
74      if(source0n == 1) printf("%d:%s\n", numL, buf);
75
76      numL++; /* Enter newline */
77
78      buf[0] = 0; /* Clean Buffer */
79      bufIndex = 0;
80  }
```

最後當遇到 **comment** 結尾 **"\*/"** 時，我們就退出 condition，讓 scanner 去 match 其他的 pattern：

```

63  <COMMENT>"*/" {
64      /* End of Comment */
65      BEGIN(INITIAL);
66
67      strcpy(&(buf[bufIndex]), yytext);
68      bufIndex += yyleng;
69  }

```

最後還有另外一種單行的 **comment**：

```

88  "//".* {
89      /* Single line comment , simply copy to buf, and wait for \n to handle*/
90      strcpy(&(buf[bufIndex]), yytext);
91      bufIndex += yyleng;
92  }

```

由“//”後面緊跟著除了換行以外的字元，將這些字串放入 **buf** 中等待遇到換行字元，遇到以後會直接印出。

## 4. Keyword

首先，我們先把所有的 **keyword** 收集起來定義到兩個別稱：

```

KEY    "void"|"int"|"double"|"bool"|"char"|"null"|"for"|"while"|"do"|"if"|"else"|"switch"|"return"
|"break"|"continue"|"const"|"true"|"false"|"struct"|"case"|"default"

```

以及所有的 **stdio.h** 裡的 **function**：

```

STDIO  "remove"|"rename"|"tmpfile"|"tmpnam"|"fclose"|"fflush"|"fopen"|"freopen"|"setbuf"|"setvbuf"
|"fprintf"|"fscanf"|"printf"|"scanf"|"sprintf"|"sscanf"|"vfprintf"|"vprintf"|"vsprintf"|"fgetc"|"fgets"
|"fputc"|"fputs"|"getc"|"getchar"|"gets"|"putc"|"putchar"|"puts"|"ungetc"|"fread"|"fwrite"|"fgetpos"
|"fseek"|"fsetpos"|"ftell"|"rewind"|"clearerr"|"feof"|"ferror"|"perror"

```

如此一來只要 input **match** 到 **KEY** 和 **STDIO** 的字串，我們就可以直接輸出 **token**，並把字串放入 **buf** 裡：

```
94 {KEY} {
95     /* Check pragma */
96     if(tokenOn == 1)    printf("#key:%s\n",yytext);
97
98     strcpy(&(buf[bufIndex]), yytext);
99     bufIndex += yyleng;
100 }
102 {STDIO} {
103     /* Check pragma */
104     if(tokenOn == 1)    printf("#key:%s\n",yytext);
105
106     strcpy(&(buf[bufIndex]), yytext);
107     bufIndex += yyleng;
108 }
```

## 5. Identifiers

根據 C99 Standard 中的定義：

**identifier:**

*identifier-nondigit*

*identifier identifier-nondigit*

*identifier digit*

**identifier-nondigit:**

*nondigit*

*universal-character-name*

*other implementation-defined characters*

**nondigit:** one of

<u> </u>	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

由以上式子可以推出，**Identifier** 是 **nondigit** 開頭，後面可以再接上 **nondigit** 或 **digit**，因此我們直接定義 ID：

**ID**            **[a-zA-Z\_]+[a-zA-Z\_0-9]\***

並在遇到 ID 時，將 token 印出：

```
176 {ID} {
177     /* Check pragma */
178     if(token0n == 1)    printf("#id:%s\n",yytext);
179
180     strcpy(&(buf[bufIndex]), yytext);
181     bufIndex += yyleng;
182 }
```

## 6. Operator

一樣的，我們將**所有 operator** 收集起來，放到 **OP** 這個別稱中：

**OP**            **"+"|"-"|"\*"|"/"|"%"|"++"|"--"|"<"|<="|>"|>="|"=="|"!="|"="|"&&"|"|"|"!"|"&"**

而 input 進來如果 **match** 到 **OP**，就將 token 印出：

```
140 {OP} {
141     /* Check pragma */
142     if(token0n == 1)    printf("#op:%s\n",yytext);
143
144     strcpy(&(buf[bufIndex]), yytext);
145     bufIndex += yyleng;
146 }
```

## 7. Punctuation

如同 Operator，**收集所有的符號**，放到 **PUNC** 這個別稱中：

**PUNC**            **":"|";"|"", "|".|"["|"]"|"("|")"|"{"|"}"**

而 input 進來如果 **match** 到 **PUNC**，就將 token 印出：

```

132 {PUNC} {
133     /* Check pragma */
134     if(tokenOn == 1)    printf("#punc:%s\n",yytext);
135
136     strcpy(&(buf[bufIndex]), yytext);
137     bufIndex += yyleng;
138 }

```

## 8. Integer Constant

根據 C99 Standard 中的定義：

*integer-constant:*

*decimal-constant integer-suffix<sub>opt</sub>*

*octal-constant integer-suffix<sub>opt</sub>*

*hexadecimal-constant integer-suffix<sub>opt</sub>*

由於這次要求實作十進位，因此後面兩個不需考慮，而 **decimal-constant** 的定義如下：

*decimal-constant:*

*nonzero-digit*

*decimal-constant digit*

*nonzero-digit:* one of

**1 2 3 4 5 6 7 8 9**

而 **integer-suffix** 的定義如下：



*integer-suffix:*

*unsigned-suffix long-suffix<sub>opt</sub>*

*unsigned-suffix long-long-suffix*

*long-suffix unsigned-suffix<sub>opt</sub>*

*long-long-suffix unsigned-suffix<sub>opt</sub>*

*unsigned-suffix: one of*

**u U**

*long-suffix: one of*

**l L**

*long-long-suffix: one of*

**ll LL**

這樣我們可以總結出，**整數常數**是由**非 0** 的數字開頭，後面可以接上數字，最後接上一個 **optional** 的 **suffix** 因此我們可以定義：

```
INT      ({DIGIT} | [1-9]{DIGIT}*){IS}?
```

```
L        (l|L|ll|LL)
```

```
U        (u|U)
```

```
IS       ({U}|{L}|{L}{U}|{U}{L})
```

IS 就是所有可能的 integer-suffix，DIGIT 則是 0~9 的數字，比較特別的是，在 **C99** 中，**0** 是屬於八進位，但由於我觀察 testcase 中，0 也算是這次的範圍，因此**特別把 0 包括進去**。

而 input 進來如果 **match** 到 **INT**，就將 **token** 印出：

```

108 {INT} {
109     /* Check pragma */
110     if(token0n == 1)    printf("#integer:%s\n",yytext);
111
112     strcpy(&(buf[bufIndex]), yytext);
113     bufIndex += yyleng;
114 }

```

## 9. Floating Point Constant

在 C99 Standard 中，Scientific notation 是歸類在 Floating point constant 之中的，但我打算將 scientific notation 放在下一個再來介紹，這邊只說明非 Scientific notation 的 Floating point constant，首先在 Standard 中：

*decimal-floating-constant:*

*fractional-constant* *exponent-part*<sub>opt</sub> *floating-suffix*<sub>opt</sub>  
*digit-sequence* *exponent-part* *floating-suffix*<sub>opt</sub>

去除掉 Scientific notation，我們可以得到 Floating point constant：

*fractional-constant* + *floating-suffix*(opt)

我們再往 fractional-constant 和 suffix 的定義去看

*fractional-constant:*

*digit-sequence*<sub>opt</sub> . *digit-sequence*  
*digit-sequence* .

*floating-suffix:* one of

**f l F L**

digit-sequence 是由數個 0~9 的數字所組成的數字串，照著以上這些定義，我們可以看出以下的 pattern：

```

18  DIGIT      [0-9]
19  DIGIT_SEQ  {DIGIT}+
20  FRAC       ({DIGIT_SEQ}?"."{DIGIT_SEQ})|({DIGIT_SEQ}"."")
21  FS         (f|F|l|L)
22
23  DOUBLE     {FRAC}{FS}?

```

FRAC 就是 fractional-constant，FS 就是 floating-suffix。

而 input 進來如果 **match** 到 **DOUBLE**，就將 **token** 印出：

```

117  {DOUBLE} {
118      /* Check pragma */
119      if(token0n == 1)    printf("#double:%s\n",yytext);
120
121      strcpy(&(buf[bufIndex]), yytext);
122      bufIndex += yyleng;
123  }

```

## 10. Scientific notation

延續剛剛提到的 C99 Standard，扣除掉已經處理的部份，我們可以知道 Scientific notation 的定義如下：

**fractional-constant + exponent part + floating-suffix(opt)**

或是

**digit-sequence + exponent part + floating-suffix(opt)**

而 exponent part 的定義：

***exponent-part:***

***e sign<sub>opt</sub> digit-sequence***

***E sign<sub>opt</sub> digit-sequence***

sign 就是+和-，有了這些資訊，我們就可以定義出：

```

E      [Ee] [+−]?{DIGIT_SEQ}
SCI    ({FRAC}|{DIGIT_SEQ}){E}{FS}?

```

E 就是 exponent part 。

而 input 進來如果 **match** 到 **SCI**，就將 **token** 印出：

```
125 {SCI} {
126     /* Check pragma */
127     if(token0n == 1)    printf("#sci:%s\n",yytext);
128
129     strcpy(&(buf[bufIndex]), yytext);
130     bufIndex += yyleng;
131 }
```

## 11. String & Char Constant

String 的定義如下：

**STR**      `"(\\(n|t) | [^"\\\n])*\"`

String 是由兩個雙引號包起來的字串，而雙引號中間的多個字元可以是非`'`，非雙引號，非換行字元的其他所有字元，由於這次也支援控制字元`\n`和`\t`，因此中間也可以是`"\n"`或`"\t"`，在這邊的`\n`和`\t`是兩個字元(`\`和 `n`)，而不是單個字元(換行字元`\n`)。

Char 的定義如下：

**CHAR**      `'([^\n] | \\(n|t))'`

Char 是由兩個單引號包起來的一個字元，這個字元可以是非`'`，非雙引號，非換行字元的其他所有字元，由於這次也支援控制字元`\n`和`\t`，因此中間也可以是`"\n"`或`"\t"`，在這邊的`\n`和`\t`是兩個字元(`\`和 `n`)，而不是單個字元(換行字元`\n`)。

定義完以上的 pattern 後，input 進來如果 **match** 到 **STR** 或 **CHAR**，就將 **token** 印出：

```

160 {STR} {
161     /* Extract String */
162     char str[1000];
163     strcpy(str,yytext+1); /* The first " is gone */
164     str[yyleng-2] = 0; /* The last " is gone */
165
166     /* Error Detect? */
167
168     /* Check pragma */
169     if(tokenOn == 1)    printf("#string:%s\n",str);
170
171     strcpy(&(buf[bufIndex]), yytext);
172     bufIndex += yyleng;
173 }

```

```

149 {CHAR} {
150
151     /* Error Detect? */
152
153     /* Check pragma */
154     if(tokenOn == 1)    printf("#char:%s\n",yytext);
155
156     strcpy(&(buf[bufIndex]), yytext);
157     bufIndex += yyleng;
158 }

```

## 12. Error Handling

我的作法是將所有合法的 pattern 都完整定義出來，如果 scanner 讀到的 input 沒辦法 match 到上面講的所有 pattern(前面 1~11)，那麼就是不合法的 input，因此我將最後一個 rule(放在 Rule section 最下面)訂為：

```

200 . {
201     fprintf(stderr, "Error at line %d: %s\n", numL, yytext); exit(1);
202
203     strcpy(&(buf[bufIndex]), yytext);
204     bufIndex += yyleng;
205 }

```

“.”代表著除了換行字元以外的所有字元，因此如果沒有 match 到前面的 rule，就會執行 fprintf 印出錯誤資訊，並執行 exit(1)，

scanner 就會結束。

### 13. Space

由於前面沒有定義到 Space 的 pattern，因此我們必須在明確訂出 Space 的規則，否則 scanner 會將 Space 視為不合法的 input，而進行 error handling，因此我們定義：

```
SPACE    [ \f\t\r\v]+
```

Space 只有可能由這些字元所構成，而 input match 到 SPACE 時，我們就將它放入 buf，等待印出。

```
183 {SPACE} {  
184     strcpy(&(buf[bufIndex]), yytext);  
185     bufIndex += yyleng;  
186 }
```