Compiler Design HW1 Report

102062209邱政凱

我的Scanner具備的功能基本上是依據HW的SPEC後再依據C99的SPEC在Constant的判斷上多增加了一些規則。基本上整體的運作邏輯是：吃到token ->確認token on/off, on則輸出token -> 塞到buffer -> 遇到換行字符確認source on/off, 若on則輸出buffer並清空。在Error的部分則是參考了gcc在-std=99的設定下會把那些部分判斷成"lexical error"(而非semantic error)，以確保不會處理到應該要是在parser階段才處理的部分，也不會漏處理應該要在Lexical analysis階段就判斷的error。

Error的輸出格式部分，看助教的範例好像助教的code是用 “/”運算子去看某些字元後面不能接哪些字元。而我認為輸出的error訊息要盡量完整，因此我是盡可能把error的token完整的吃進來用yytext輸出。(i.e. 01234我會輸出01234而非0，report最後會有整理我的error輸出格式)

額外的部分我主要支援了一些integer和floating point的suffix (123.123f、123L)和string constant的prefix (L”123”)，在double的部分支援123. 或 .123這些支援在C99中的語法。並在Pragma宣告的部分完整參照C99的規則。

* Non-existent Character

當輸入出現並沒有在c99中支援的字符時，我會直接輸出該字符作為Error訊息。

* Comment：吃到“//”後便進入LINE Comment state，再吃到換行字元前的所有東西都直接放入line buffer而不輸出token。吃到”/\*”則進入COMMENT state，在遇到另一個”\*/”前把所有吃到的東西都放到line buffer(不過遇到\n要換行)。
* Pragma : 基本上我用了比較全面的規則來處理pragma的部分



Pragma依照C99的STANDARD必須要:

在同一行的前面不能有空白(含comment)以外的字元、每個單字之間至少都要有空白(含comment)間隔、Pragma完整敘述後不能再加除了comment以外的其他敘述。因此我的scanner在pragma敘述不符合以上三種狀況，或是打了不存在的pragma時會輸出error。

例如：

123 #pragma source on 、 #pragma source once 、 # pragma source on x = 123 我都會輸出成error。

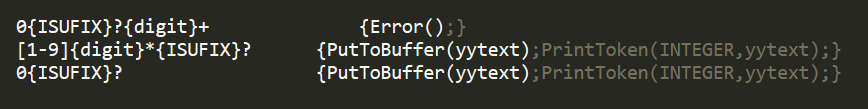
而我的判斷正確的pragma方式就是# 、 pragma 、 source/token 、on/off 全部都必須同時出現，且它們之間必須都間隔一個以上的空白或/\*\*/comment。

* Identifier:

C99中定義的identifier的合法字元包刮 0-9 a-z A-Z和\_ (底線)。

而一個合法的identifier只要第一個字元不是0-9，其他各種各樣的組合都是合法的。 E.g. \_\_123 、 a12\_\_。所以我處理identifier的方式基本上就是[a-zA-Z\_][0-9a-zA-Z\_]\*

* KEY : 基本上C99定義的關鍵字我就是直接把助教的投影片上列舉的KEY打上去並直接輸出成KEY的Token。Stdio.h的部分則是另外去找stdio.h把裡面的library function照著字典序排列輸入。
* Integer constant

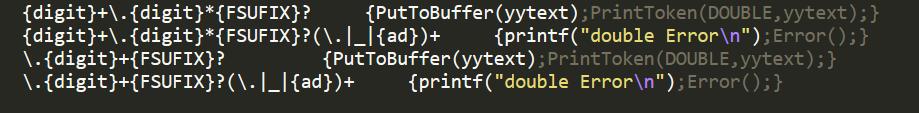


Integer constant的部分，我支援了C99的integer suffix(123U,123LL)來描述unsigned或long的狀況。在error的部分我會在第一個數字是0而有其他數字接在後面的狀況發生。

合法:

1. 0
2. {Non-zero digit} {digit}

* Double constant:



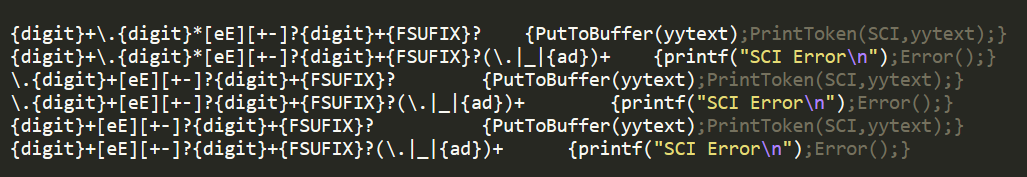
Double constant的部分我除了支援基本的 123.123這種形式外還支援了 .123和 123. 這些省略0的表示方式。並支援floating point constant的suffix (f、F、l 、L) 。另一點比較值得說明的就是，在C99 standard中，double的整數部分是可以有0當作prefix的，例如: 00123.456在C99的standard中是合法的，因此如果判斷是double，我就不會針對開頭是0這件事輸出error。

合法:

1. {digit sequence}(opt) . {digit sequence}{floating point suffix}(opt)
2. {digit sequence} . (nothing) {floating point suffix}(opt)

而Error則會在double的敘述後還有接 “\.”、[:alpha:]、 “\_”等等字元的狀況下發生。一個token連接這些字符通常lexical analyzer會判斷為同一個token的延伸，而非不同的token。這部分我會這樣判是因為我觀察gcc的compiler，像是123.123.123、 123.123abc、 123.123\_x\_ 等等的狀況下，gcc會輸出 “too many floating points” 、 “inappropriate suffix”等等的敘述，因此可以推斷應該是lexical error，而非文法階段的error，所以我才把這些規則納入。

* Scientific Notation



科學記號的部分我主要也是參考C99的standard。基本上c99定義scientific notation就是 {合法的FP constant}[eE][+-]?{digit}+ 這樣的形式，只是suffix必須放到最後面。

合法:

(1): {digit sequence}(opt) . {digit sequence} [eE] [+-] {floating point suffix}(opt)

(2): {digit sequence} . [eE] [+-] {digit sequence}{floating point suffix}(opt)

(3) {digit sequence}[Ee] [+-] {digit sequence}{floating point suffix}(opt)

Error的部分規則類似double，就是在合法的scientific notation後馬上接“\.”、[:alpha:]、 “\_”等等字元的狀況下發生。至於 123e (後面沒接數字)

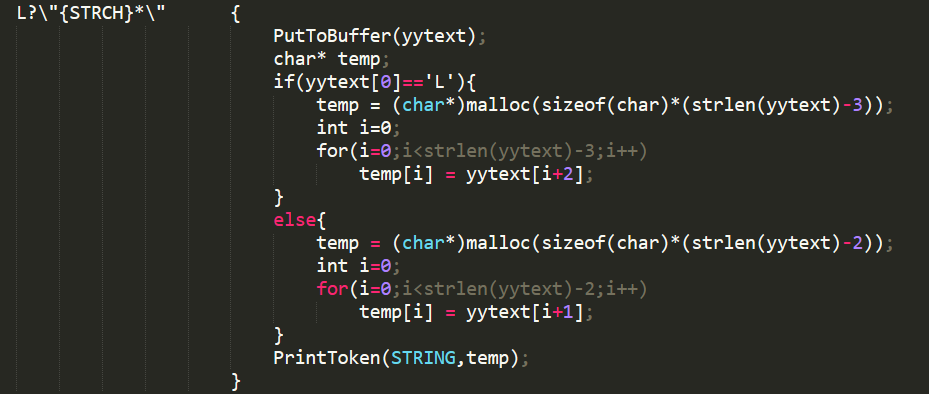
、123e\*123(非+-的記號)等等的狀況，因為 “123e” 這樣的規則本身就是個非法的integer constant(或非法的identifier)，因此在我的scanner中這兩種狀況在他被認定成是scientific notation的error之前它就會被別的地方判掉了

* Character constant

字元常數的部分依照C99的定義，不能是以下三種形式 ‘\’、 ‘\n’、 ‘ \’’ 也就是不能有單引號、換行字符、或是單斜線的出現。

* String Consant

String constant比較特別的地方是在token的輸出需要把雙引號拔掉，在print source的部分則不需要，因此用以下這段code，先把yytext直接塞進buffer，再把雙引號拔掉後print token出來。



例外根據c99，跟character constant有點像，只是單引號是被允許的，而雙引號是不被允許的，因此match的rule吃入兩個雙引號中夾著的所有排除‘\’、 ‘\n’、 ‘ \”’的字元。例如 “123\q123” (不存在的跳脫字元)、 “123 <EOF> (未匹配的雙引號) 等等我都會輸出成ERROR。

Error Handling:

以下列舉我額外處理的error以及範例的input和對應輸出的error：

Input output

0{digit}+{ISUFIX}? 01234 01234

{digit}+({alpha}|\_)({alpha}|{digit}|\_)\* 123abc 123abc

123e+ 123e

123e\*321 123e

{digit}+\.{digit}\*{FSUFIX}?(\.|\_|{ad})+ 123.123abc 123.123abc

345.abc 345.abc

123.123.123 123.123.123

123..123 123..123

\.{digit}+{FSUFIX}?(\.|\_|{ad})+ .123abc .123abc

{digit}+[eE][+-]?{digit}+{FSUFIX}?(\.|\_|{ad})+ 1E+3abc 1E+3abc

\.{digit}+[eE][+-]?{digit}+{FSUFIX}?(\.|\_|{ad})+ .1E+3abc .1E+3abc

{digit}+\.{digit}\*[eE][+-]?{digit}+{FSUFIX}?(\.|\_|{ad})+ 2.2E+3abc 2.2E+3abc

^{BLNAK}\*#(\_|{alpha}|{digit})\* #pragma source once #pragma source once

#include <stdio.h> #include