2017 spring PL project (OurScheme) - Project 1

Due : 6/25(日) midnight (23:59)

```
// You are to implement something like the following

// 'expr' is a pointer that points to a linked list data structure;
// The linked list data structure results from reading in
// the user's input.

Print 'Welcome to OurScheme!'
Print '\n'
Print '> '

repeat

   ReadSExp(expr);

   PrintSExp(expr); // You must "pretty print" this data structure.

   Print '> '

until (OR (user entered '(exit)')
          (END-OF-FILE encountered)
      )

if ( END-OF-FILE encountered ) // and NOT 「user entered '(exit)'」
   Print 'ERROR (no more input) : END-OF-FILE encountered'

Print '\n'
Print 'Thanks for using OurScheme!' // Doesn't matter whether there is an
                                    // '\n' at the end
```

2. Syntax of OurScheme

terminal (token) :

```
LEFT-PAREN          // '('
RIGHT-PAREN         // ')'
INT                 // e.g., '123', '+123', '-123'
STRING              // "string's (example)." (strings do not extend across lines)
DOT                 // '.'
FLOAT               // '123.567', '123.', '.567', '+123.4', '-.123'
NIL                 // 'nil' or '#f', but not 'NIL' nor 'nIL'
```

```
T                   // 't' or '#t', but not 'T' nor '#T'
QUOTE                // '
SYMBOL               // a consecutive sequence of printable characters that are
                    // not numbers, strings, #t or nil, and do not contain
                    // '(', ')', single-quote, double-quote, semi-colon and
                    // white-spaces ;
                    // Symbols are case-sensitive
                    // (i.e., uppercase and lowercase are different);
```

Note :

With the exception of strings, token are separated by the following "separators" :

(a) one or more white-spaces
(b) '(' (note : '(' is a token by itself)
(c) ')' (note : ')' is a token by itself)
(d) the single-quote character (') (note : it is a token by itself)
(e) the double-quote character (") (note : it starts a STRING)
(f) ';' (note : it starts a line-comment)

Examples :

FLOAT : '3.25', '.25', '+.25', '-.25', '+3.'
INT : '3', '+3', '-3'
'3.25a' is a SYMBOL.
'a.b' is a SYMBOL.
'#f' is NIL
'#fa' (alternatively, 'a#f') is a SYMBOL.

Note :

When the system prints a float, it always print the fraction part in
three digits. Examples : 3.000, -17.200, 0.125, -0.500

Use printf("%.3f", ...) in C or String.format("%.3f", ...) in Java
to get these three digits.

Note :

'.' can mean several things :
it is either part of a FLOAT or part of a SYMBOL or a DOT.

It means a DOT only when it "stands alone".

'#' can also mean two things :
it is either part of NIL (or T) or part of a SYMBOL.

It is part of NIL (or T) only when it is '#t' or '#f' that "stand alone".

Note :

OurScheme 的 string 有 C/Java 的 printf()的 escape 的概念，但只限於'\n', '\"', '\t', '\n'與'\\'這五個 case。

如果'\'字元之後的字元不是'n', '"', 't', 或'\'，此(第一個)'\'字元就無特殊意義(而只是一個普通的'\'字元)。

Examples of acceptable (= legal) strings in OurScheme :

"There is an ENTER HERE>>\nSee?!"
"Use '\"' to start and close a string."
"OurScheme allows the use of '\\n', '\\t' and '\\\"' in a string."
"Please enter YES\NO below this line >\n"
"You need to handle >>\\<<"
"You also need to handle >>\"<<"
"When you print '\a', you should get two chars: a backslash and an 'a'"

Syntax of OurScheme :

<S-exp> ::= <ATOM>
| LEFT-PAREN <S-exp> { <S-exp> } [DOT <S-exp>] RIGHT-PAREN
| QUOTE <S-exp>

<ATOM> ::= SYMBOL | INT | FLOAT | STRING
| NIL | T | LEFT-PAREN RIGHT-PAREN

!!!!! Once the attempt to read in an S-expression fails, the line !!!!!
!!!!! containing the error-token is ignored. The system starts !!!!!
!!!!! to read in an S-expression from the next input line. !!!!!

Note : a quoted S-expression >>'...<< is the same as >>(quote ...)<<

a. In C, the basic program building block is a statement.
In OurScheme, the basic program building block is
an S-expression (S-exp, for short).

b. An S-exp is either an atom, a list, or a dotted pair.

c. An atom is either an integer (e.g., 123), a float
(e.g., 12.34 or 12. or .34), a string (e.g., "Hi, there!"),
or a symbol (e.g., abc).

d. Abc, abc, aBc, a-B!c?, !??, t, nil are examples of symbols

```
// Blanks and line-returns ("white-space characters") are
// considered delimiters (please see the above definition of
// "separators")

// Upper case and lower case are different, e.g., aB, AB, Ab,
// ab are all different symbols.

// Each symbol may or may not be bound to an S-exp.

// When I say that a symbol abc is bound to the S-exp
// >>(abc "Hi there" (5 3))<<,
// you could take what I mean to be that the "value" of abc
// is >>(abc "Hi there" (5 3))<<.

// "Binding" (rather than "value") is a better way of saying
// what the situation really is.

// t, nil are two system-defined constant values
//      (t for "true" and nil for "false")
// t (or #t) and nil (or #f or >>()<<) are not symbols.

// t is also written as #t, meaning "true"
// nil is also written as () or #f, meaning "false"
// In other word,
//      these two are the same : t      #t
//      these three are the same : nil      #f      ()

// OurScheme understands both 't' and '#t', but it only prints '#t'
// OurScheme understands all these three : 'nil', '#f', '()',
//      but it only prints 'nil'.

// Side remark :
//      (True) Scheme uses #t, #f and ()
```

// "Other Lisps" use t, nil and ()
// (Please see the short article: ThreeLisps.txt)

e. An 「S-exp sequence」 is of the form

S1 S2 S3 ... Sn

where each Si is an S-exp.

// e.g., (1) 1 (1 . 1)

// e.g., 1 2 (3 4 (5))

// Each of the above S-exp sequence contains three S-exp

f. A dotted pair is of the form

(SS1 . S2)

where S2 is an S-exp, whereas SS1 is an 「S-exp sequence」.

// Note that there is a dot between SS1 and S2,

// with one or more spaces in between

// e.g., (1 . 2)

// e.g., (1 2 3 4 . 5)

// e.g., (1 2 3 4 . ())

// e.g., (1 . (2 . (3 . abc)))

// e.g., (1 2 3 . abc)

// e.g., ((1) (2 (3)) . (abc))

// e.g., ((1) (2 (3)) . (nil))

// e.g., ((1) (2 (3)) . nil)

g. The following notations of dotted pairs are equivalent.

(S1 S2 S3 S4 . S5)

(S1 . (S2 . (S3 . (S4 . S5))))

h. Comment :

What we refer to as a "dotted pair" is different from what other professionals refer to as a "dotted pair".

What other professionals mean by a dotted pair is just (S1 . S2), where S1 and S2 are S-exp.

i. A list is of the form

(SS1)

where SS1 is an 「S-exp sequence」.

// Note : () is known as "the empty list"

// For historical reasons, () is defined to be the same

// as nil or #f, meaning "false"

j. A list (S1 S2 ... Sn) is actually a short-handed
notation for the following dotted pair
(S1 . (S2 . (...(Sn . nil)))...)))
In other words, a list is actually a special kind of
dotted pair.

Another way of writing the list (S1 S2 ... Sn) is
(S1 S2 ... Sn . nil)

// In other word, there are three (seven?) ways for writing
// the same list.
// (S1 S2 S3 S4 S5)
// (S1 . (S2 . (S3 . (S4 . (S5 . nil)))))
// (S1 . (S2 . (S3 . (S4 . (S5 . #f)))))
// (S1 . (S2 . (S3 . (S4 . (S5 . ())))))
// (S1 S2 S3 S4 S5 . nil)
// (S1 S2 S3 S4 S5 . #f)
// (S1 S2 S3 S4 S5 . ())

k. When the system prints out a dotted pair, it
always tries to print it in list-like format.

For example, if the dotted pair is
(1 . (2 . (3 . (4 . 5))))
Then the system prints it as
(1 2 3 4 . 5)

But if the dotted pair is
(1 . (2 . (3 . (4 . nil))))
The system does not print it as
(1 2 3 4 . nil)
Instead, the system prints it as
(1 2 3 4)

l. Line comments

A line comment begins with ';' until the end-of-line.
Note that ';' is a separator.
Therefore, for example, 'ab;b' stands for a symbol 'ab' followed
by a line comment ';b'.)

3. Here are some examples of what your program should
do for project 1. (The following assumes that your program
runs interactively.)

Welcome to OurScheme!

```
> (1 . (2 . (3 . 4)))
( 1
  2
  3
  .
  4
)

> (1 . (2 . (3 . nil)))
( 1
  2
  3
)

> (1 . (2 . (3 . ())))
( 1
  2
  3
)

> (1 . (2 . (3 . #f)))
( 1
  2
  3
)

> 13
13

> 13.
13.000

> +3
3

> +3.
```

```
3.000

> -3
-3

> -3.
-3.000

> a
a

> t
#t

> #t
#t

> nil
nil

> ()
nil

> #f
nil

> (t () . (1 2 3))
( #t
    nil
    1
    2
    3
)

> (t . nil . (1 2 3))
ERROR (unexpected token) : ')' expected when token at Line 1 Column 10 is >>.<<

> "There is an ENTER HERE>>\nSee?!"
"There is an ENTER HERE>>
See?!"

> "Use '\"' to start and close a string."
```

```
"Use '"' to start and close a string."

> "OurScheme allows the use of '\\n', '\\t' and    '\\\"' in a string."
"OurScheme allows the use of '\n', '\t' and    '\"' in a string."

> "Please enter YES\NO below this line >\n"
"Please enter YES\NO below this line >
"

> "You need to handle >>\\<<"      "You also need to handle >>\"<<"
"You need to handle >>\<<"

> "You also need to handle >>"<<"

> ((1 2 3) . (4 . (5 . nil)))
( ( 1
      2
      3
   )
   4
   5
)

> ((1 2 3) . (4 . (5 . ())))
( ( 1
      2
      3
   )
   4
   5
)

> (12.5            . (4 . 5))
( 12.500
   4
   .
   5
)

> (10 12.())      ; same as : ( 10 12. () )
( 10
   12.000
```

```
    nil
)

> (10 ().125)      ; same as : ( 10 () .125 )
( 10
    nil
    0.125
)

> ( 1 2.5)
( 1
    2.500
)

> ( 1 2.a)
( 1
    2.a
)

> (1 2.25.5.a)
( 1
    2.25.5.a
)

> (12 (        . 3))
ERROR (unexpected token) : atom or '(' expected when token at Line 1 Column 10 is >>.<<

> "Hi"
"Hi"

> "(1 . 2      . 3)"
"(1 . 2      . 3)"

> (((1 . 2)
        . ((3 4)
              .
             (5 . 6)
           )
      )
      . (7 . 8)
    )
( ( ( 1
```

```
          .
          2
       )
       ( 3
          4
       )
       5
       .
       6
    )
    7
    .
    8
)

> ())
nil

> ERROR (unexpected token) : atom or '(' expected when token at Line 1 Column 1 is >>)<<

> (Hi there ! How are you ?)
( Hi
    there
    !
    How
    are
    you
    ?
)

> (Hi there! How are you?)
( Hi
    there!
    How
    are
    you?
)

> (Hi! (How about using . (Lisp (instead of . C?))))
( Hi!
    ( How
       about
```

```
        using
        Lisp
        ( instead
            of
            .
            C?
        )
    )
)

> (Hi there) (How are you)
( Hi
    there
)

> ( How
    are
    you
)

> (Hi
                .
                (there    .(          ; note that there may be no space between
                                      ; '.' and '('
                How is it going?))
                )
( Hi
    there
    How
    is
    it
    going?
)

> ; Note : We have just introduced the use of comments.
    ; ';' starts a comment until the end of line.
    ; A comment is something that ReadSExp() should skip when
    ; reading in an S-expression.

(1 2 3) )
( 1
    2
```

```
   3
)

> ERROR (unexpected token) : atom or '(' expected when token at Line 1 Column 2 is >>)<<

> (1 2
     3) (4 5
( 1
   2
   3
)

>             6)
( 4
   5
   6
)

>               '(Hi
                  .
                  (there    .(         ; note that there may be no space between
                                          ; '.' and '('
                  How is it going?))
                  )
( quote
   ( Hi
      there
      How
      is
      it
      going?
   )
)

> '(1 2 3) )
( quote
   ( 1
      2
      3
   )
)
```

>   ERROR (unexpected token) : atom or '(' expected when token at Line 1 Column 2 is >>)<<

> '(1 2 3) .25
(quote
  (1
    2
    3
)
)

> 0.250

>   (
    exit   ; as of now, your system only understands 'exit' ;

)   ; and the program terminates when it sees '(exit)'

Thanks for using OurScheme!

// ==================== Project 1 I/O requirement ===========

Project 1 的 I/O 要求

你要寫的是一個 interactive system。如果正常(在 DOS 視窗)跑，其運作跟 Petite Chez Scheme 的運作很像。其例如下(note that for the first project, you only need to read in an S-expression and then print out an S-expression)：

  Welcome to OurScheme!

  > a   ; a line-comment starts with a ';', and continues until end-of-line
  a

  > 3   ; your system should be able to skip all line-comments
  3

  > 3.5
  3.500 ; always print 3 digits behind '.' for reals

  > +3
  3

```
> +3.25
3.250

> 1.55555 ; Use printf( "%.3f", ...) in C or String.format( "%.3f", ...) in Java
1.556

> (cons 3 5)    ; once the system prints the output, it prints a blank line
( cons
   3
   5
)

>                      ; the system first prints '> ', and then starts to get
                       ; the user's input until either an unexpected character
     (       (         ; is encountered or the user has entered an S-expression
                       ;
Hi "!" How       ; note that the principle of "longest match preferred"
                       ; should be honored ; e.g., if the user enters 'How',
. "are you?"    ; you should get 'How' and not (just) 'H' or 'Ho' ;

          )     "Fine.    Thank you."

  )       ( 3 .      ; if, on the same line that the S-expression ends, the
( ( Hi
       "!"
       How
       .
       "are you?"
   )
   "Fine.    Thank you."
)

>                      ; user also starts another input, then the
                       ; system also starts processing the second input,
            .          ; but will print the output for the first input first
ERROR (unexpected token) : atom or '(' expected when token at Line 4 Column 8 is >>.<<

> ( 1 2 ) ( 3 4 ) 5
( 1
   2
)
```

```
> ( 3
   4
)

> 5

> ; the above is an example of how the system handles "multiple
   ; input on the same line"
   ; The point : the user may have already started entering input
   ;                    BEFORE the system prints '> '

   (exit       ; this is the way to get out of user-system dialog ;
                ; below, there is a LINE-ENTER preceding 'Thanks' and
 )              ; two LINE-ENTER following '!'

Thanks for using OurScheme!
```

// ==========

但 PAL 是用一個 input 檔來測你的系統，而且你的程式的 output 會"導"到一個 output 檔去。所以，當 PAL 測試你的程式時，here is what really happens :

```
// input 自下一行開始
1
a     ; a line-comment starts with a ';', and continues until end-of-line
3     ; your system should be able to skip all line-comments
(cons 3 5)    ; once it prints the output, it prints a blank line
                ; the system first prints '> ', and then starts to get
                  ; the user's input until either an unexpected character
   (      (       ; is encountered or the user has entered an S-expression
                  ;
Hi "!" How      ; note that the principle of "longest match preferred"
                  ; should be honored ; e.g., if the user enters 'How',
. "are you?"    ; you should get 'How' and not (just) 'H' or 'Ho' ;

        )    "Fine.    Thank you."

 )      ( 3 .     ; if, on the same line that the S-expression ends, the
                ; user also starts another input, then the
                  ; system also starts processing the second input,
          .       ; but will print the output for the first input first
```

```
( 1 2 ) ( 3 4 ) 5
; the above is an example of how the system handles "multiple
   ; input on the same line"
   ; The point : the user may have already started entering input
   ;                        BEFORE the system prints '> '

   (exit         ; this is the way to get out of user-system dialog ;
                   ; below, there is a LINE-ENTER preceding 'Thanks' and
  )                ; two LINE-ENTER following '!'
// input 至上一行止



// output 自下一行開始
Welcome to OurScheme!

> a

> 3

> ( cons
   3
   5
)

> ( ( Hi
       "!"
       How
       .
       "are you?"
   )
   "Fine.    Thank you."
)

> ERROR (unexpected token) : atom or '(' expected when token at Line 4 Column 8 is >>.<<

> ( 1
   2
)

> ( 3
   4
```

)

> 5

>
Thanks for using OurScheme!
// output 至上一行止，而且不包括上一行的 trailing white space(s)

註：
For some unknown reason, PAL cannot get the "final white spaces" in your output.

Therefore, in the "standard answer" that PAL uses to compare your output with, there are no "final white spaces" either.

// ==

PAL 的運作是這樣：

它先 compile 你的程式，如果 compile 沒問題，它就用若干測試檔來測試你的程式。

它先測第一個測試數據，也就是 run 你的程式、並以第一個測試檔作為 input、同時也

把你的程式的 output"導"到一個 output 檔。

若你的程式順利 run 完(祝福你！)，PAL 就比對你的 output 檔與「標準答案檔」，

若二者的內容有任何一點不同，PAL 就到此為止，並說"你答錯了"(給看不給看則端視

這是第幾個測試數據)。

若二者的內容完全相同，PAL 就用下一個測試檔再來一次(如果已經沒有測試檔了就

"恭喜答對")。

換句話說，有幾個測試檔、PAL 就會 run 你的程式幾次(如果你都過的話)。

如果在 run 你的程式的(這麼多次的)過程之中有"疑似無窮迴圈"或發生 exception，

PAL 就到此為止，並通知你。

// ==

gTestNum (或 uTestNum) :

為了幫助你 debug，所有的測試數據的一開始都有個 integer (it has no preceding white-spaces, and is immediately followed by a LINE-ENTER character)，此 integer 代表著"目前這個測試數據是第幾個測試數據"。你應該用一個 global 或 file-scope 的變數來存這個 integer。此 integer 對於你的 debug 工作將會很有幫助。(程式一開始就讀它(此 test number 保證存在)，然後想辦法(if you want)"跳過"其後的 LINE-ENTER，再開始你的任何"正常 processing"。)

// ==========

OurScheme 的 I/O 規矩：

※ 基本的 I/O 規矩請由以上的 I/O 範例推導

※ 基本上，(syntax) error 的發生、在於(OurScheme)系統讀到"不該出現的字元"，當這狀況發生時，(OurScheme)系統應該知道此字元的 line 與 column，從而印出如範例中所示的

```
ERROR (unexpected token at line 4, column 8) : .
```

但有一個狀況比較特殊(事實上有多種，我們只考慮這種)，那就是 string 還沒"close"就已出現 LINE-ENTER。例： // 假設'('是 column 1
// 第一行共 18 字元(不計 LINE-ENTER)

```
( cons "Hi" "How
       are you" )
```

此時要印的是

```
ERROR (no closing quote) : END-OF-LINE encountered at line 1, column 19
```

※ 一旦發生 unexpected character 錯誤，該行就完全 skip，

系統會把使用者的下一行視為(系統印出)'> '之後的 input。

※ 印 S-expression 的規矩列於本文最後。

※ 一旦已 process 到使用者所 enter 的'(exit)'，之後在 input 檔中若還有任何

使用者 input，系統都不須也不該理會。

※ 由於是用 input 檔給 input、而不是 interactive I/O，所以有可能發生「使用者

還沒 input 完一個 S-expression 就沒 input 了」的狀況，

此時系統應該要 print 如下的 message :

ERROR (no more input) : END-OF-FILE encountered

Thanks for using OurScheme!

```
// ================================================

Rules for printing an S-expression s

if s is an atom
   then print s with no leading white space and with one trailing '\n'
          note : For 'nil', '()' and '#f', always print 'nil'.
          note : For '#t' and 't', always print '#t'.

else { // s is of the form : '(' s1 s2 ... sn [ '.' snn ] ')'

   let M be the number of characters that are already
         printed on the current line

   print '(', print one space, print s1
   print M+2 spaces, print s2
   ...
   print M+2 spaces, print sn
   if there are '.' and snn following sn
      print M+2 spaces, print '.', print '\n'
      print M+2 spaces, print snn
   print M spaces, print ')', print '\n'

} // else s is of the form : '(' s1 s2 ... sn [ '.' snn ] ')'
```

Example :

(((1 . 2) (3 4) 5 . 6) 7 . 8)

should be printed as // output 之中的各行的前兩個 space 不算

```
// output starts from the next line
( ( ( 1
          .
          2
      )
      ( 3
          4
      )
      5
      .
      6
   )
   7
   .
   8
)
// output terminates here, and does not include this line
// all lines in the output have no trailing spaces or tabs
```

Example :

(((1 . "ICE CYCU") (THIS is (41 42 . 43)) Chung . Yuan) 7 . 8)

should be printed as // output 之中的各行的前兩個 space 不算

```
// output starts from the next line
( ( ( 1
          .
          "ICE CYCU"
      )
      ( THIS
          is
          ( 41
              42
```

```
            .
            43
          )
        )
        Chung
        .
        Yuan
      )
      7
      .
      8
  )
  // output terminates here, and does not include this line
  // all lines in the output have no trailing spaces or tabs
```

```
// ================================================
```

Project 1 可能會出現的 error - 總整理

總共有四個可能會出現的 error：

```
  ERROR (unexpected token) : atom or '(' expected when token at Line X Column Y is >>...<<
  ERROR (unexpected token) : ')' expected when token at Line X Column Y is >>...<<
  ERROR (no closing quote) : END-OF-LINE encountered at Line X Column Y
  ERROR (no more input) : END-OF-FILE encountered
```

請參考以下範例： // 註：interactive I/O 無法得到 EOF error

```
Welcome to OurScheme!

> (1 2 . ; this is a comment
) ; comment again
ERROR (unexpected token) : atom or '(' expected when token at Line 2 Column 1 is >>)<<

> .
ERROR (unexpected token) : atom or '(' expected when token at Line 1 Column 1 is >>.<<

>

    . 34 56
ERROR (unexpected token) : atom or '(' expected when token at Line 3 Column 4 is >>.<<
```

```
> (1 2 . ;
34 56 ) ; See?
ERROR (unexpected token) : ')' expected when token at Line 2 Column 4 is >>56<<

> ( 1 2 (3
4
      )
    .     "Hi, CYCU-ICE
ERROR (no closing quote) : END-OF-LINE encountered at Line 4 Column 21

> (23 56 "How do you do?
ERROR (no closing quote) : END-OF-LINE encountered at Line 1 Column 23

> "
ERROR (no closing quote) : END-OF-LINE encountered at Line 1 Column 2

> (exit 0)
( exit
   0
)

> (exit)

Thanks for using OurScheme!

// ===
```

註：如果 interactive I/O 會碰到 EOF error ，結束方式就會是如下

```
> (exit 0)
( exit
   0
)

> ERROR (no more input) : END-OF-FILE encountered
Thanks for using OurScheme!
```

// ==

Q and A (modified to fit the current version of Project 1)

// ================== Q and A No. 1 ======================

Question : '(1 2 3)的 output 應該是什麼？

Answer :

Project 1 :

```
> '(1 2 3)
( quote
   ( 1
      2
      3
   )
)
```

Project 2 :

```
> '(1 2 3)
( 1
   2
   3
)
```

解釋：

Project 1 只讀進來(建出 DS)、不 evaluate 就直接(把 DS)印出去，
所以得 : (quote (1 2 3))。

Project 2 是讀進來(建出 DS)、evaluate(此 DS)、再把 evaluate 的 result
(依舊是個 DS)印出去，所以得 : (1 2 3)。

(將'(quote (1 2 3))'予以 evaluate 所得的結果是'(1 2 3)')

// ================== Q and A No. 2 ======================

※ 引述《...》之銘言：

> 請問老大~~

> 1.當輸入'> .'時，會是 ERROR 嗎???
> 若是是 ERROR.msg：會是 column: 1 or 2 ???

It is an error. (Let us suppose that it is '> .$', where '$' is
LINE-ENTER char.)

ERROR (unexpected token) : atom or '(' expected when token at Line 1 Column 1 is >>.<<

> 2.請問 abc"abc 會印出什麼呢???

The first token is 'abc'. The second token is a string that starts
with : "abc

Therefore, // for project 1

> abc"abc
abc

> ERROR (no closing quote) : END-OF-LINE encountered at line 1, column 5

→ yabuki 推：老大那 abc'abc 算是 symbol 嗎?還是會有其他結果? 03/09 22:27

answer (to yabuki's question) :

> abc'abc
abc

> (quote
  abc
)

>

// ================== Q and A No. 3 ======================

※ 引述《...》之銘言：

> 這問題困擾很久百思不得其解。

>
> 依題意應該是最後一個可以執行（或印出）的 Token 後將行號和欄位初始化
> 範例如下

> ())
> ^這玩意兒是最後的 token，所以要印出在 line 1 錯

> 問題來了：
> "最後一個字串"\n
> \n
> \n
> "出錯點

> 為什麼這裡印 line 3？把最後一個字串 token 拿掉後應該如下
> \n 1
> \n 2
> \n 3
> "出錯點 第4行
> ----------

> "最後一個字串" \n 1
> \n 2
> \n 3
> "出錯點 4

> 加了空白為什麼也是 line 3？

答：

在 legal input 之後(同一行)要出現「下一個 input S-expression 的有效字元」，才把這一行算作是下一 input 的第一行。 若 legal input 之後(同一行)只出現「space 或 tab 或註解」，那這一行就不算作是下一 input 的第一行。