2021 spring PL project (OurScheme) - Project 1

Due : 6/27(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 連續的可打印字符序列
                  // not numbers, strings, #t or nil, and do not contain  不是數字、字串、#t 或 nil，並且不包含
                  // '(', ')', single-quote, double-quote, semi-colon and   '(' ' )'、單引號、雙引號、分號 和 空格 ;
                  // white-spaces ;
                  // Symbols are case-sensitive  符號區分大小寫
                  // (i.e., uppercase and lowercase are different);（即大寫和小寫不同）;
```

Note :

(separators)

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

除"字串"外，令牌由以下“分隔符”分隔:

(a) one or more white-spaces 一個或多個空格

(b) '(' (note : '(' is a token by itself) 左括號本身是token

(c) ')' (note : ')' is a token by itself) 右括號本身是token

(d) the single-quote character (') (note : it is a token by itself) 單引號本身是token

(e) the double-quote character (") (note : it starts a STRING) 雙引號是字串開頭
(f) ';' (note : it starts a line-comment) 分號是註解

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' (或者, 'a#f') is a SYMBOL. ##################

Note :
(float)
當系統打印一個浮點數時，它總是打印小數部分三位數。示例：3.000、-17.200、0.125、-0.500
在 C 中使用 printf("%.3f", ...)
在 Java 中使用 String.format("%.3f", ...)
得到這三個數字。

Note :
('.'、'#')

'.'可以表示三件事
1.它要么是 FLOAT 的一部分。
2.要么是 SYMBOL的一部分。
3.只有當它“獨立”時才表示 DOT。

'#' 可以表示兩件事:
1.它要么是 NIL（或 T）的一部分，要么是 SYMBOL 的一部分。
2.只有當“#t”或“#f”“獨立”時，它才是 NIL（或 T）的一部分。

Note :
(string內----特殊字符)
OurScheme的string有C/Java的printf()的escape(逃脫)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'"

Note :

Syntax(文法分析) 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

Note :

!!!!! 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.   !!!!!
!!!!! 一旦嘗試讀取 S 表達式失敗，一行 !!!!!
!!!!! 包含錯誤標記的將被忽略。系統啟動！！！！
!!!!! 從下一個輸入行讀入 S 表達式。 !!!!!

Note :
注意：帶引號的 S-expression(S 表達式) >>'...<< 等同於 >>(quote ...)<<

a. 在 C 中，基本的 the basic program building block 是一條語句。
在 OurScheme 中，基本的the basic program building block是一個 S 表達式（簡稱 S-exp）。

b. S-exp 可以是atom、list或dotted pair。

c. atom可以是:
□□1.integer（例如 123）
□□2.float（例如，12.34 或 12. 或 .34）
□□3.string（例如，“Hi, there!”）
4.symbol（例如 abc）
□□
d. symbols
□□examples : Abc, abc, aBc, a-B!c?, !??, t, nil

□ // 空格和換行符（“空白字符”）被視為 "分隔符"（請參閱上述“separators”的定義）
// 大小寫不同，如aB、AB、Ab、ab都是不同的符號。
// 每個符號可能bound(綁定)也可能 not bound(不綁定)到 S-exp。
□□ examples:□□
□□□當我說 symbol abc 綁定到 S-exp
□□□□>>(abc "Hi there" (5 3))<<,
□□□可以看成我得意思是 the "value" of abc is abc 的“值” 是
□□□□>>(abc "Hi there" (5 3))<<.
□□□“綁定”的說法比（“值”的說法）是描述更精確。
□
□ // t, nil
□ // t, nil 是兩個系統定義的常量值（t 代表“真”，nil 代表“假”）
// t（或#t）和nil（或#f 或>>()<<）不是符號。□
// t == #t --- meaning "true"
// nil == #f == () --- meaning "false"
// OurScheme 理解
□□□□1.'t' 和 '#t'，但它只打印 ------- '#t'
□□□□2.'nil'、'#f'、'()'，但它只打印 ---'nil'

e. 一個「S-exp 序列」的形式是: S1 S2 S3 ... Sn，其中每個 Si 都是一個 S-exp。
// 例如，(1) 1 (1 . 1)
// 例如，1 2 (3 4 (5))
// 上面的每個S-exp序列都包含三個S-exp

f. (DOT)A dotted pair 的形式是:
□ (SS1 . S2)
□
其中 S2 是一個 S-exp，而 SS1 是一個「S-exp sequence(序列)」。

// 注意 SS1 和 S2 之間有一個點，
// 中間有一個或多個空格
// 例如，(1 . 2)
// 例如，(1 2 3 4 . 5)
// 例如，(1 2 3 4 . ())
// 例如，(1 . (2 . (3 . abc)))
// 例如，(1 2 3 .abc)
// 例如，((1) (2 (3)) . (abc))
// 例如，((1) (2 (3)) . (nil))
// 例如，((1) (2 (3)) . nil)

g. 下面的(DOT)A dotted pair符號是equivalent(等價的)。
(S1 S2 S3 S4 . S5)= (S1 . (S2 . (S3 . (S4 . S5))))

i. A list的形式:
(SS1)
其中 SS1 是一個「S-exp 序列」。
// 注意：() 被稱為“空列表”
// 由於歷史原因，() 被作為與 nil 或 #f相同意義，意思是“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 . ())

j. A list (S1 S2 ... Sn) 實際上是以下dotted pair的簡寫符號
= (S1 . (S2 . (...(Sn . nil)))...)))
= (S1 S2 ... Sn . 無)

example:
七種寫法，以下皆為相同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 . 無)
= (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)

k. 當系統印出a dotted pair時，它總是嘗試以類似列表的格式打印它。

1 例如，如果dotted pair是
(1 . (2 . (3 . (4 . 5))))
□□ 然後系統將其打印為
□□□ (1 2 3 4 . 5)

2 例如，dotted pair是
(1 . (2 . (3 . (4 . nil))))
□□ 系統不會將其打印為
□□□ (1 2 3 4 . 無)
□□ 相反，系統將其打印為
□□□ (1 2 3 4)
□□

l. Line comments(一行的註解)
□ 一行的註解以 ';' 開頭 直到行尾。(';' 是一個分隔符)
□ 例如，'ab;b' 代表符號 'ab' 後跟一行的註解';b'。）

==========

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 >>...<< (行、列紀錄token的第一個char)
ERROR (unexpected token) : ')' expected when token at Line X Column Y is >>...<< (行、列紀錄token的第一個char)
ERROR (no closing quote) : END-OF-LINE encountered at Line X Column Y (行、列紀錄token的str的下一個列)
ERROR (no more input) : END-OF-FILE encountered (讀到最後沒字元了，但還沒讀到(exit)，會發生的Error)

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 ******(行、列紀錄token的str的下一個列)

> (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!
```

==========

3. 以下是您的程式執行project 1操作的一些範例。（以下假設您的程序以interactively(互動式)運行。）

```
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 need to handle >>\<<"

> "You also 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)) *□token□□□ 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) ************************** 左括號前面的點不用印出來
    . ((3 4)
       .
       (5 . 6)
      )
   )
   . (7 . 8)
  )
( ( ( 1
      .
      2
    )
    ( 3
      4
    )
    5
    .
    6
  )
  7
  .
  8
)

> ())
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) ) ****************??  S-exp: 1.(1 2 3)    2.)
( 1
  2
  3
)

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

> (1 2
   3)
( 1
```

```
  2
  3
)

>  (4 5  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) )  # S-exp: 1.'(1 2 3)    2.)
(
  ( 1
    2
    3
  )
)

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

> '(1 2 3) .25 **********□□?  S-exp: 1.'(1 2 3)    2. .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 □I/O□□
```

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
 )
 ********************************************************************************好亂什麼東西?
 >            ; 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

 > ; 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.
  由於某些未知原因，PAL 無法在您的輸出中獲得“最後的空白”。
  因此，在 PAL 用來比較你的輸出的“標準答案”中，也沒有“最後的空格”。

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

Rules for printing an S-expression s [印出字串的規則]

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
```

// ===

(不太重要)

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!
```

// ==

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。