

## 2021 spring PL project (OurScheme) - Project 1

Due : 6/27(€) 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  $f, \dots, \dagger, \ddagger, \sim, \% , \S , < , \mathbb{E}$

// not numbers, strings, #t or nil, and do not contain •Ž•%•%'•#t' nil" "••—

// '(', ')', single-quote, double-quote, semi-colon and '('•')'•™š•>™š•œš•žŸ;

// white-spaces ;

// Symbols are case-sensitive Šš œj €£

// (i.e., uppercase and lowercase are different); ¢¥j £•¢£•| \$''

Note :

(separators)

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

©"% ' "a " « ¬ - ® ° ± Š² œ ± ³

(a) one or more white-spaces ´µ ¶ μžŸ

(b) '(' (note : '(' is a token by itself) · ¸ Š¹ ° Žtoken

(c) ')' (note : ')' is a token by itself) » ¸ Š¹ ° Žtoken

(d) the single-quote character (') (note : it is a token by itself) ~™š¹ ° Žtoken

(e) the double-quote character (") (note : it starts a STRING) > "ŠŽ%0' ¼½  
 (f) ';' (note : it starts a line-comment) æšŽ¾¿

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' (' Å, 'a#f') is a SYMBOL. #####

Note :

(float)  
 ÅÃÄ±^´µÄÄ•Æ" ÇÈŽ±^¢•ÉæÊË•Ì Í Î ð³3.000•-17.200•0.125•-0.500  
 Ĩ Ç ÐÑÒ printf("%.3f", ...)  
 Ĩ Java ÐÑÒ String.format("%.3f", ...)  
 ÓÔÕÊµ•%0Ì

Note :

('.' '#')  
 ' ' †®ŒÍ Ê×Ø  
 1. ÇÛÜŽ FLOAT ... ´ÉæÌ  
 2. ÛÜŽ SYMBOL... ´ÉæÌ  
 3. ÛÜÄÇ°ÝÞ²ÆßŒÍ DOTÌ  
 '# ' †®ŒÍ à×Ø³  
 1. ÇÛÜŽ NILª' TŞ... ´Éæ" ÛÜŽ SYMBOL ... ´ÉæÌ  
 2. ÛÜÄ°#t²' °#f²°ÝÞ²Æ" ÇßŽ NILª' TŞ... ´ÉæÌ

Note :

(stringá----âã%Š)  
 OurScheme...stringÜC/Java...printf()...escape(ää)...æç"  
 èÛéé'n', '\", \'t', 'n'è'\Œì µcaseÌ  
 í î '\%i' ðñ...%i' •Ž'n', '"', 't', ' ' '\ " ò(ó´µ)\%i' ô õ  
 âãö÷(øÛŽ´µùú...'\%i')Ì

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(üüæý) 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. !!!!  
 !!!! 'py S 0 " ' !!!!  
 !!!! -- ... 1 AA  
 !!!! - 'μ S 0 1 !!!!

Note :

ö³ ™š... S-expression(S 0 ) >>'...<< |ê >>(quote ...)<<

a. Ĩ C Đ" 1... the basic program building block Ž ' Ĩ  
 Ĩ OurScheme Đ" 1...the basic program building block Ž ' μ S 0 S-expŠĭ

b. S-exp †@Žatom•list' dotted pairĭ

c. atom†@Ž:

- 1.integer¤ĭ ĭ 123Š
- 2.float¤ĭ ĭ "12.34 ' 12. ' .34Š
- 3.string¤ĭ ĭ "°Hi, there!²Š
- 4.symbol¤ĭ ĭ abcŠ

d. symbols

examples : Abc, abc, aBc, a-B!c?, !?!, t, nil

```
// žŸ• Š¤ž! %Š²Š " # "æ±Š"¤$%&' (°separators²...) ÷Š
// ĭ ¢£• | " ĭ aB•AB•Ab•ab*Ž• |...Ššĭ
// +μŠš†, bound(-)). †, not bound(•-))ô S-expĭ
examples:
Á/0 symbol abc -)ô S-exp
>>(abc "Hi there" (5 3))<<,
†@12/ôö3Ž the "value" of abc is abc ...°4² Ž
>>(abc "Hi there" (5 3))<<.
°-)²...0ü5¤°4²...0üšŽ6(789ĭ
```

```
// t, nil
// t, nil ŽàμAA) ÷...: ; 4¤t <ô°=²" nil <ô°>²Š
// t¤' #tŠ•nil¤' #f ' >>)<<Š•Žššĭ
// t == #t --- meaning "true"
// nil == #f == () --- meaning "false"
// OurScheme ?¿
1.'t' • '#t' " èÇÛ†^ ----- '#t'
2.'nil'•'#f'•'()' " èÇÛ†^ ---'nil'
```

e. 'μ•S-exp <£, ...@ Ž: S1 S2 S3 ... Sn" AĐ+μ Si \*Ž ' μ S-expĭ  
 // Ĩ ĭ " (1) 1 (1 . 1)  
 // Ĩ ĭ " 1 2 (3 4 (5))  
 // ' B...+μS-exp<£\*—ÊμS-exp

f. (DOT)A dotted pair ...@ Ž:  
 (SS1 . S2)

AĐ S2 Ž ' μ S-exp" ø SS1 Ž ' μ•S-exp sequence(<£), ĭ

```
// 0 SS1 • S2 0CÜ ´µÄ"
// 0CÜ ´µ´ ¶µžŸ
// Î í " (1 . 2)
// Î í " (1 2 3 4 . 5)
// Î í " (1 2 3 4 . 0)
// Î í " (1 . (2 . (3 . abc)))
// Î í " (1 2 3 .abc)
// Î í " ((1) (2 (3)) . (abc))
// Î í " ((1) (2 (3)) . (nil))
// Î í " ((1) (2 (3)) . nil)
```

g. B...(DOT)A dotted pairŠšŽequivalent( D...)ĭ  
 (S1 S2 S3 S4 . S5)= (S1 . (S2 . (S3 . (S4 . S5))))

i. A list...@ :  
 (SS1)  
 A0 SS1 Ž ´µ•S-exp <E, ĭ  
 // 0 : 0 #°žŸ0²  
 // -êEFGH" 0 ĭ #ë nil ' #fJ | ö÷" ö3Ž\*false²

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 . 0 ) ) ) ) ) )
// (S1 S2 S3 S4 S5 . nil)
// (S1 S2 S3 S4 S5 . #f)
// (S1 S2 S3 S4 S5 . 0)
```

j. A list (S1 S2 ... Sn) KL' Ž®~ dotted pair... fŠš  
 = (S1 . (S2 . (... (Sn . nil))))  
 = (S1 S2 ... Sn . 0)

example:  
 MNĖü"®~0#J | list:  
 (S1 S2 S3 S4 S5)  
 = (S1 . (S2 . (S3 . (S4 . (S5 . nil)))))  
 = (S1.(S2.(S3.(S4.(S5.#f)))))  
 = (S1 . (S2 . (S3 . (S4 . (S5 . 0 ) ) ) ) ) )  
 = (S1 S2 S3 S4 S5 . 0)  
 = (S1 S2 S3 S4 S5 . #f)  
 = (S1 S2 S3 S4 S5 . 0)

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. ÁÃÄ^Pa dotted pairÆ" ÇÈŽÿ ®QRĖÖ...ÿ ‡^ÇĬ

1Î í " í î dotted pairŽ

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

SñÃÃ A‡^#

(1 2 3 4 . 5)

2Î í " dotted pairŽ

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

ÃÃ•T A‡^#

(1 2 3 4 . 5)

JU" ÃÃ A‡^#

(1 2 3 4)

l. Line comments(´ ...¾¿)

´ ...¾¿® ' ; ¼½ VÔ WĬ (' ; Ž ´µæ±Š)

Î í " 'ab;b' <ÖŠš 'ab' ñX´ ...¾¿';b'Ĭ §

=====  
Project 1†, TPY...error - ÈZ?

È[Ü\µ†, TPY...error³

ERROR (unexpected token) : atom or '(' expected when token at Line X Column Y is >>...<< ( •[Ė] ^token...ó ´µchar)

ERROR (unexpected token) : ')' expected when token at Line X Column Y is >>...<< ( •[Ė] ^token...ó ´µchar)

ERROR (no closing quote) : END-OF-LINE encountered at Line X Column Y ( •[Ė] ^token...str...´ ´µĖ)

ERROR (no more input) : END-OF-FILE encountered ( Ô\_ñ`%ï a"èb` Ô(exit)" Tcd...Error)

\$%e®~fĬ ³ // ¾³interactive I/O öüÖÔ 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 ***** ( •[E] ^token...str... ^´μ[E)
```

```
> (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!

```
// ===
```

```
¾³í î interactive I/OTgÔ EOF error "hi j òTŽí ^
```

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

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

```
=====
```

```
3. ®~Žk...l m project 1nI ... ^ofÎÎ ĩ ¢®~>pk...l <®interactively(q )r ĩ §
```

Welcome to OurScheme!

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

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

```
> (1 . (2 . (3 . 0)))
(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 >>.<<
```

```
*Tst^P^u \"---> "".....  
> "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 . 0)))  
( (1  
2  
3  
)  
4  
5  
)

> (12.5 . (4 . 5)) \*vtokenTv2 12.5 // 4|.|5  
( 12.500  
4  
.  
5  
)

> (10 12.0) ; same as : ( 10 12. 0 )  
( 10  
12.000  
nil  
)

> (10 0.125) ; same as : ( 10 0 .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) ***** . ſšwB...Ā•Ô^Px
. ((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) ) *****0Ž? 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 >><< *****0Ž?

```

```

> '(1 2 3) .25 *****0Ž? 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/OÛy

zÜE...Ž´µinteractive systemĳ í ĳ { : (ĳ DOS" ĳ)} "Arĳ XPetite  
 Chez Scheme...rĳ ĳ ~•ĳ Aĳ í ĳ (note that for the first project, you only  
 need to read in an S-expression and then print out an S-expression)<sup>3</sup>

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  
 )

\*\*\*\*\*€•, f„...?

> ; 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!

// =====

èPALŽÒ ´µinput†x‡z...ÂÃ"ø•z...l ...outputT"~"Ô
´µoutput†%ł Š®" ÁPAL‡ z...l Æ" here is what really
happens :

// inputs ~ ´ ¼<
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(É' ´ •

// outputs ~ ´ ¼<
Welcome to OurScheme!

> a

```

> 3

> ( cons  
3  
5  
)

> (( Hi  
"I"  
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[É' ´ •"Ø••—,´ ´ ...trailing white space(s)

¾³

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.

-êŽo••GH" PAL õüĭ k... PÐ' Ó°\_ñ...Ž! ²ĭ  
Hò"ĭ PAL Òx5' z... P...° " " •²Ð" . `Û°\_ñ...ŽŸ²ĭ

// =====

Rules for printing an S-expression s [ ^P%0' ...— ]

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

// 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

// 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

```

// =====

(•š› Û)

PAL...rI ŽŒ³

Ç•compilez...I "íî compile`žŸ"Çôð ĩ † †x† z...I Ĩ

Ç•†ó´µ† •Ç". ôŽrunz...I •"®ó´µ† †I #input•|Æ.

£z...I ...output"ˆ"Œ´µoutput†ĭ

z...I ¨¥runĭ(\$`z )" PALô5uz...output†ë• ""•†, "

©Ä...áªÜ«¬´Ä•| " PALôŒð#•" " 0"z" a"(-1•-1-®"

ŒŽó´µ† •Ç)ĭ

©Ä...áª|°J| " PALôŒ´µ† †±x´²(íî³ˆÜ† †aô

"µ¶"u")ĭ

• 0"Ü´µ† †•PALôTrunz...I ´²(íîz\*„... )ĭ

íîĭrunz...I ...(Œf¶²...)„I ðŒÜ"¹Rô°»¼" cdexception"

PALôŒð#•" " ú•zĭ

// =====

gTestNum ( ' uTestNum ) :

#a½¼zdebug" ŠÜ...† •Ç...´¼<\*Üµinteger (it has no preceding

white-spaces, and is immediately followed by a LINE-ENTER character)"

ðinteger<Œž"ÄwŒµ† •ÇŽó´µ† •Ç"ĭ zÄÄŒ´µglobal'

file-scope...Ä•xÄŒµintegerĭ ðintegeruêz...debugÄI T~Ü½¼ĭ

(I ´¼<ô Ç(ðtest numberÆÇÄĭ)" SñÈÉÜ(if you want)"Ê„"

Añ...LINE-ENTER" ±¼<z...«¬" { : processing"ĭ )

// =====

OurScheme...I/O–Ē³

ĭ ¹...I/O–Ē\$–®' ...I/Ofĭ ĭ ^

ĭ ¹' " (syntax) error...cd•ĭ ê(OurScheme)ÄÄ Œ"•ÄPY...%ĭ ""

ÄŒĭ ĭcdÆ" (OurScheme)ÄÄÄÄ•Đò%i ĭ...lineëcolumn"

ø^Pí fĭĐŠĭ ...

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



èÜ´µÎÎ5' âã(ØK' Û¶¶N" /ÑÛeÒÕN)" ÓôŽstring

b`"close"ô³PYLINE-ENTERÎ Î³ // >p'(Žcolumn 1  
// ó´ [18%ĩ (•ÔLINE-ENTER)  
( cons "Hi" "How  
are you" )

òÆÛ^...Ž

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

Î ´p̄cdunexpected character " Â ô | ° skip"

ÂÂTEÑÒÀ...´´ " # (ÂÂ^P)'> 'ðñ...inputÎ

Î ^S-expression...–ĚĚê¹û\_ñÎ

Î ´p³processÔÑÒÀŠenter...'(exit)'" ðñÎ input†Đ bÛ«¬

ÑÒÀinput" ÂÂ\*•Œ. •Â?TÎ

Î -êŽÒinput†-input•ø•Žinteractive I/O" Š®Û†, cd•ÑÒÀ

b`input | ´µS-expressionô`inputa, ...ÎÎ "

òÆÂÂÂÂÛprintí ¯...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)...outputÁÂŽ, fŒ

Answer :

Project 1 :

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

Project 2 :

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

$x^3$

Project 1  $\hat{U} \otimes x(\hat{U}PDS) \bullet \bullet \text{evaluate} \hat{U}(\hat{E}DS)^{\wedge} P\%$   
Š@Ó : (quote (1 2 3))l

Project 2  $\hat{Z} \otimes x(\hat{U}PDS) \bullet \text{evaluate}(\hat{o}DS) \bullet \pm \text{evaluate} \dots \text{result}$   
( $\hat{U}\hat{U}\hat{Z}\mu DS)^{\wedge} P\%$  Š@Ó : (1 2 3))l

( '(quote (1 2 3))'Ÿ@evaluateŠÓ...hî Ž'(1 2 3)')

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

l ™(p...ßðàá³

> \$žâj ~~

> 1.Á '> . 'Æ" TŽERRORă???

> ŽŽERROR.msg³TŽcolumn: 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.\$žabc"abcT^P, fä???

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

â yabuki í ³âj Óabc'abc™Žsymbolă?bŽTÜAæhî ? 03/09 22:27

answer (to yabuki's question) :

> abc'abc  
abc

> ( quote  
abc  
)

>

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

l ™(p...ßðàá³

> ŌžŸçè~ée3•ÓA¿l

>

> ÜŸöÁÄŽ\_ñ´µ†®m ¨´ ^P§...Tokenñ Š•ëËì <í

> fîí´

> ()

> ^Ōî öï Ž\_ñ...token“ Š®Û^Pí line 1

> žŸxa³

> "\_ñ´µ‰´ "\n

> \n

> \n

> "P Ä

> #, fŌð^line 3Ō£\_ñ´µ‰´ tokenñòñÄÄí´

> \nóóóóóóóóóóóóóóóó

> \n                      Ō

> \n                      ö

> "P Ä                      Ó÷

> -----

> "\_ñ´µ‰´ "óóóóóó\nóó

> \n                      Ō

> \n                      ö

> "P Ä                      ÷

> øaž! #, f. Žline 3Ō

" 3

í legal inputðñ(í´ )ÜPY•´´µinput S-expression...Üù‰í´ , "

ß£Ō´ ™l Ž´´input...ó´ l legal inputðñ(í´ )ÜPY

•space´ tab´ ¼¿, " ŌŌ´ ô•™l Ž´´input...ó´ l