Definitions

Symbols: s are alphanumeric strings that begin with a letter.

Terms: t are either a symbol or a pair.

Pairs: p are entered by using the syntax (t1 . t2) where t1 and t2 are both terms.

Lists are either the symbol nil, which designates an empty list, or they are constructed from one or more pairs.  
  
 In the case of a pair, the first term in the pair is referred to as the head of the list represented by that pair and the sub-list contained in the second term is referred to as the tail of that list. The head element is some term and may therefore itself represent either a symbol or a list.  
  
A proper list is also referred to as a nil-terminated list. When the second element of the final pair is terminated by some symbol other than nil, the sequence of pairs is referred to as a “dotted list”.

Printed Representation of a Pair

To write any sequence of pairs (whether they may turn out to represent a proper, nil-terminated list or a dotted list) one begins by writing an open parenthesis and by establishing a reference to the “remaining sub-list”.  
  
If the remaining sub-list is a pair, the head of that sub-list (which may itself represent a list) is written in its entirety, followed by a space.  
  
Then the remaining sub-list is replaced by the second element of the pair; and the process of writing that sub-list is repeated. If the second element was the symbol nil, a close parenthesis is written and writing of the list is complete.  
  
In the case of a dotted list, the tail of the sub-list will be some non-nil symbol. In this case, a dot is written followed by a space followed by the name of the non-nil symbol followed by the final close parenthesis. If effect, that final pair will be written out in using a syntax similar to that for entering a pair. The only difference is that the first element of the pair will be preceded by all of the other elements of the dotted list.

Abstraction

**Note:** The syntax [x]t is shorthand for lambda (x) t

Abstraction of a symbol from a term, produces a term:

[x]x => I

[x]y => K y

[x](t1 t2) => S [x]t1 [x]t2

Reduction

Reduction of terms:

S t1 t2 t3 => t1 t3 (t2 t3)

K t1 t2 => t1

I t => t

Y h => h (Y h)

Function Definition

Expressions: e can be any term, or abstraction

(Recursive) Functions: f are defined in terms of expressions.

def f = e => f = Y [f]e