Project
SNU 4910.210, Fall 2019
Chung-Kil Hur
due: 12/20(Fri.) 23:59

Problem 1 (50 Points) In Scala, implement an interpreter `interp` for the programming language E given below.

$$\texttt{interp} : E \to V$$

A	::=	x	call by value
B	::=	`(def` (A^*) E`)`	def
	|	`(val` x E`)`	val
E	::=	n	Integer
	|	x	name
	|	`true`	true
	|	`false`	false
	|	`nil`	list nil
	|	`(if` E E E`)`	conditional
	|	`(cons` E E`)`	pair construction
	|	`(fst` E`)`	the first component of a pair
	|	`(snd` E`)`	the second component of a pair
	|	`(nil?` E`)`	is nil
	|	`(app` E E^*`)`	function call
	|	`(let` (B^*) E`)`	name binding to def/val/lazy val
	|	`(+` E E`)`	integer addition
	|	`(-` E E`)`	integer subtraction
	|	`(*` E E`)`	integer multiplication
	|	`(=` E E`)`	integer equality
	|	`(<` E E`)`	integer less-than
	|	`(>` E E`)`	integer greater-than

- For ill-typed inputs, you can return arbitrary values, or raise exceptions.
- X^* denotes that X can appear 0 or more times.
- `let` clauses create a new scope like a 'block' in Scala. Name bindings `def`, and `val` work the similar way as in Scala.
  - `(def f (`A^*`) E)` assigns name f to expression E with arguments A^*. Examples include `(def f (a b) (+ a b))` and `(def g () 3)`.
  - `(val x E)` assigns name x to the value obtained by evaluating E.
  - We do not allow the same name to be defined twice in the frame.
  - You do not have to consider forward reference in `val`. For example, `(val x (cons 1 x))`.
  - Hint: Implement environment with mutable data structure for `lazyness`.
- `Enviornment` is collection of `Frames`. `Frame` is created when a new scope is created.
- `(nil? E)` first evaluates E into value v. If v is nil, it returns `true`. Otherwise, it returns `false`.
- For additional information, post questions on the GitHub course webpage.
- examples in src/test/scala/TestMain.scala.

Problem 2 (10 Points) Optimize `interp` to handle tail recursive input programs, such as the example code shown below.
(Hint: Use Scala's tail recursion.)

Problem 3 (20 Points) Add lazy evaluation to `interp` by implementing `by-name` and `lazy-val` following.

A	::=	$\cdots$	
	|	(`by-name x`)	call by name
B	::=	$\cdots$	
	|	(`lazy-val` x E)	lazy val

- Name bindings `lazy-val` work the similar way as in Scala.
  - `(lazy-val x E)` assigns name x to the value obtained by evaluating E lazily.
  - Hint: Implement environment with mutable data structure for `lazyness`.

Problem 4 (20 Points) Add record to `interp` by implementing `rmk` and `rfd` following.

E	::=	$\cdots$	
	|	(`rmk` B^*)	Record constructor
	|	(`rfd` E x)	Record field access

- `rmk` and `rfd` implement record types.
  - (`rmk` B^*) constructs a record value.
  - `(rfd E x)` projects out the field x of the record value obtained by evaluating E.