Project
SNU 4910.210, Programming Principles Fall 2020
Chung-Kil Hur
due: 12/19(Sat) 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 f (`A^*`)` E`)`	def
	|	`(val` x E`)`	val
E	::=	n	integer
	|	x	name
	|	`nil`	list nil
	|	`(cons` E E`)`	pair constructor
	|	`(fst` E`)`	the first component of a product type value
	|	`(snd` E`)`	the second component of a product type value
	|	`(inl` E`)`	the left tagged value of sum type
	|	`(inr` E`)`	the right tagged value of sum type
	|	`(nil?` E`)`	is nil
	|	`(int?` E`)`	is int
	|	`(prod?` E`)`	is product type
	|	`(sum?` E`)`	is sum type
	|	`(let (`B^*`)` E`)`	name binding of def/val
	|	`(app` E E^*`)`	function call
	|	`(match E ((x) E) ((x) E))`	pattern matching for inl and inr values
	|	`(+` E E`)`	integer addition
	|	`(-` E E`)`	integer subtraction
	|	`(*` E E`)`	integer multiplication
	|	`(/` E E`)`	integer division
	|	`(%` E E`)`	integer remainder
	|	`(=` 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 (by-name 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 de
    ned 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.
- `(inl` v`)` and `(inr` v`)` are sum type values.
- `nil` and `(cons` v_1 v_2`)` are product type values.
- `(match` E_1 `((`x_1`)` E_2`) ((`x_2`)` E_3`))` first evaluates E_1 into value v. If v is `(inl` v_1`)`, it evaluates E_2 with binding $x_1 := v_1$ to get the final value. If v is `(inr` v_2`)`, it evaluates E_3 with binding $x_2 := v_2$ to get the final value.
- `true` and `false` are encoded as `inr 0` and `inl 0`, respectively.
- `(nil? E)` first evaluates `E` into value `v`. If `v` is `nil`, it returns `true`.
- `(int? E)` first evaluates `E` into value `v`. If `v` is `integer`, it returns `true`. Otherwise, it returns `false`.
- `(prod?` first evaluates `E` into value `v`. If `v` is product type value, it returns `true`. Otherwise, it returns `false`.
- `(sum?` first evaluates `E` into value `v`. If `v` is sum type value, 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 (15 Points) Optimize `interp` to handle tail recursive input programs, such as the example code shown below.

```
(let ((def f (x sum) (match (> x 0) ((_) sum) ((_) (app f (- x 1)
(+ x sum))))))) (app f 10 0))
```

Problem 3 (15 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) Implement the function to find the nth prime number in the language defined above.
Hint: See `nthPrime` function in the lecture note.