Part 3 Summary

Due November 21st at 10pm

*** indicates an extra credit function

IMPLEMENTATION: lambda

(lambda? exp) tester : returns #t if exp is a lambda expression; #f otherwise

ABSTRACTION: closure

(make-closure lambda-exp env) constructor : creates abstraction storing lambda exp and its env

(closure? exp) tester : returns #t if exp is a closure; #f otherwise

(procedure-parameters closure) selector : returns list of parameters of closure

(procedure-body closure) selector : returns list of expressions in closure

(procedure-env closure) selector : returns environment in which closure was defined

(apply-closure closure vals) controller : apply closure to evaluated arguments vals

IMPLEMENTATION: let

(let? exp) tester : returns #t if exp is a let expression; #f otherwise

(let->lambda exp) returns lambda expression equivalent of let expression

(eval-let exp env) controller : evaluates let expression

IMPLEMENTATION: map

***(map? exp)** tester : returns #t if exp is a map expression; #f otherwise

***(eval-map exp env)** controller : evaluates map expression

IMPLEMENTATION: filter

***(filter? exp)** tester : returns #t if exp is a filter expression; #f otherwise

***(eval-filter exp env)** controller : evaluates filter expression

IMPLEMENTATION: fold

***(fold? exp)** tester : returns #t if exp is a fold expression; #f otherwise

***(eval-fold exp env)** controller : evaluates fold expression