CS 511: Homework Assignment 3
Due: November 4, 11:55pm

1 Assignment Objectives

Get acquainted with the notion of

- sequential aspects of Erlang

2 Assignment Policies

Collaboration Policy. This homework may be done in individually or in pairs. Use of the Internet is allowed, but should not include searching for existing solutions.

Under absolutely no circumstances code can be exchanged between students. Excerpts of code presented in class can be used.

Assignments from previous offerings of the course must not be re-used. Violations will be penalized appropriately.

Late Policy. Late submissions are allowed with a penalty of 2 points per hour past the deadline.

3 Assignment

The aim of this assignment is to write an interpreter for a simple functional language called SFL.

4 SFL

4.1 Syntax

A program in SFL is called an *expression* and is defined by the following grammar:

```
<Exp> := <Num>
      |  <Id>
      |  -(<Exp>,<Exp>)
      |  +(<Exp>,<Exp>)
      |  zero?(<Exp>)
      |  if <Exp> then <Exp> else <Exp>
      |  let identifier = <Exp> in <Exp>
      |  proc (<Id>) <Exp>
      |  <Exp>(<Exp>)
      |  (<Exp>)
```

You will be supplied with a parser for SFL. As an example, the result of parsing the string `"let y=3 in +(2,y)"` is

```
{ok,{letExp,{id,1,y},
            {numExp,{num,1,3}},
            {plusExp,{numExp,{num,1,2}},{idExp,{id,1,y}}}}}
```

For the possible values that you may get from the parser, please inspect `parser.yrl`.

Note: In order to generate the lexer and the parser you must run these lines (ignore the shift/reduce and reduce/reduce conflicts).

```
32> leex:file(lexer).
{ok,"./lexer.erl"}
33> c(lexer).
{ok,lexer}
34> yecc:file(parser).
parser.yrl: Warning: conflicts: 3 shift/reduce, 0 reduce/reduce
{ok,"parser.erl"}
35> c(parser).
{ok,parser}
```

4.2 Semantics

An expression can return three possible values: a number, a boolean or a closure. Here are some examples of expressions in SFL, collected in a module called `tests`. The `runStr/1` function parses and evaluates an expression. It will be defined in another module (`interp.erl`).

```
-module(tests).
-export([start/0]).

start() ->
    lists:map(fun interp:runStr/1,examples()).

examples() ->
    [ex1(), ex2(), ex3(), ex4(), ex5(), ex6(), ex7(), ex8(), ex9()].

ex1() ->
    "let x=1 in let x=3 in +(x,7)".

ex2() ->
    "+(2,3)".

ex3() ->
    "proc (x) +(x,3)".

ex4() ->
    "let y=3 in proc (x) +(x,y)".

ex5() ->
```

```
    "let y=3 in +(2,y)".

ex6() ->
    "let y=proc(x) +(x,1) in y(5)".

ex7() ->
    "let x=1 in let y=proc(z) +(z,x) in y(6)".

ex8() ->
    "zero?(7)".

ex9() ->
     "let x=1 in let f=proc (y) +(y,x)  in let x=2 in f(3) ".
```

When we run these examples we get the following output:

```
16> c(tests).
{ok,tests}
17> tests:start().
[{num,10},
 {num,5},
 {proc,x,
       {plusExp,{idExp,{id,1,x}},{numExp,{num,1,3}}},
       {dict,0,16,16,8,80,48,
             {[],[],[],[],[],[],[],[],[],[],[],[],[],[],...},
             {{[],[],[],[],[],[],[],[],[],[],[],[],...}}}},
 {proc,x,
       {plusExp,{idExp,{id,1,x}},{idExp,{id,1,y}}},
       {dict,1,16,16,8,80,48,
             {[],[],[],[],[],[],[],[],[],[],[],[],[],...},
             {{[],[],[],[],[],[],[],[],[],[[...]],[],...}}}},
 {num,5},
 {num,6},
 {num,7},
 {bool,false},
 {num,4}]
```

You can also parse and evaluate a program from a file using

```
-spec runFile(string()) -> valType().
```

5 The Interpreter

Your task is to build an interpreter for SFL. It should conform to the following type specification:

```
-spec valueOf(expType(),envType()) -> valType().
```

where these types are defined as follows (`types.hrl`):

```
-type envType() :: dict:dict(atom(),valType()).
-type expType() :: tuple().

-type numValType() :: { num, integer() }.
-type boolValType() :: { bool, boolean() }.
-type procValType() :: { proc, atom(), expType(), envType()}.
-type valType() :: numValType() | boolValType() | procValType().
```

5.1 Summary

Modules to complete:

- `interp.erl`: Implement `valueOf`.
- `env.erl`: Implement the following operations (that just constitute wrappers for the corresponding operations in `dict`, which you should look up).

```erlang
-module(env).
-compile(export_all).
-include("types.hrl").


-spec new()-> envType().
new() ->
    %% define

-spec add(envType(),atom(),valType())-> envType().
add(Env,Key,Value) ->
    %% define

-spec lookup(envType(),atom())-> valType().
lookup(Env,Key) ->
    %% define
```

You must make sure that your code passes the Dialyzer analysis (you may ignore the "Unknown functions" warning).

```
$ dialyzer interp.erl
  Checking whether the PLT /Users/ebonelli/.dialyzer_plt is up-to-date... yes
  Proceeding with analysis...
Unknown functions:
  env:add/3
  env:lookup/2
  env:new/0
  lexer:string/1
  parser:parse/1
 done in 0m1.77s
done (passed successfully)
```

6 Submission Instructions

Submit a zip file named `Assignment3_<Surname>.zip` (where `<Surname>` should be replaced by your surname) through Canvas containing all the files in the stub (which should have been completed).