CS 496: Homework 4
Due: 7 March, 11:55pm

1 Assignment Policies

Collaboration Policy. Homework will be done individually: each student must hand in their own answers. It is acceptable for students to collaborate in understanding the material but not in solving the problems or programming. 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.

2 Assignment

This assignment consists in extending REC to allow for mutually recursive function definitions. The resulting language will be called REC-M. It modifies the concrete syntax for `letrec` as follows. The production

<Expression> ::= `letrec` <Identifier>(<Identifier>) = <Expression> `in` <Expression>

in REC is replaced with:

<Expression> ::= `letrec` { <Identifier>(<Identifier>) = <Expression>}$^+$ `in` <Expression>

in REC-M. The expression { <Identifier>(<Identifier>) = <Expression>}$^+$ above means that there may be 1 or more declarations. Here is an example of a valid program in REC-M:

```
letrec
  even(x) = if zero?(x)
              then 1
              else (odd (x - 1))
  odd(x)  = if zero?(x)
              then 0
              else (even (x - 1))
in (odd 99)
```

Evaluating that expression should produce the result `NumVal 1`, meaning that 99 is indeed odd. If we replace 99 in the code above with 98 and evaluate the resulting expression, this time we should get `NumVal 0` as a result. This is correct since 98 is not an odd number.

Note that the above expression is not syntactically valid in REC. To see this, try running it in the interpreter for REC.

3 Implementing REC-M

To facilitate the process of implementing REC-M a stub has been provided for you in Canvas. This stub has been obtained by taking the interpreter for REC and applying some changes. Here is a summary of the changes:

1. The `parser.mly` file has been updated so that the parser is capable of parsing expressions such as

```
letrec
  even(x) = if zero?(x)
              then 1
              else (odd (x - 1))
  odd(x)  = if zero?(x)
              then 0
              else (even (x - 1))
in (odd 99)
```

   Here is the result of parsing it:

```
Letrec
 ([Dec ("even", "x",
    ITE (IsZero (Var "x"), Int 1, App (Var "odd", Sub (Var "x", Int 1))));
   Dec ("odd", "x",
    ITE (IsZero (Var "x"), Int 0, App (Var "even", Sub (Var "x", Int 1))))],
 App (Var "odd", Int 99))
```

   Note that `Letrec` now has two arguments, `Letrec of (dec list)*expr`, where `dec` is a new type with the constructor `Dec of (string*string*expr)`. Each `dec` in the first parameter of `Letrec` represents one of the function declarations in the first part of the `letrec` expression.

2. The `environment` datatype has been updated by creating a new sum type with the syntax

```
type env =
  | EmptyEnv
  | LetEnv of (string*exp_val*env)
  | LetrecEnv of ((Ast.dec list)*env)
```

   Instead of maintaining the environment as a list and calling functions to add to it when needed, you will construct a new env type with the necessary constructor, where the final parameter is the environment that you are extending.

You will have to update

1. `lookup` in the file `ds.ml` by implementing the `find_dec` function. It currently reads as follows:

```
(* TODO Return the declaration specified by id, or None if it is not found
   Look at the matches in the LetrecEnv case of lookup for hints on what you
   should be returning *)
let rec find_dec (decs:(Ast.dec list)) (id:string):(Ast.dec option) =
  match decs with
  | [] -> None
  | Ast.Dec(name, var, body)::_ when name = id -> failwith "Implement me"
  | Ast.Dec(name, var, body)::rest -> failwith "Implement me"
```

This function utilizes OCaml's `option` type which has the syntax

```
type 'a option = Some of 'a | None
```

This can be seen as an alternative to error handling with "null" checks used in Java. You can return `None` if you didn't find what you were looking for or `Some x` if you did. The result can be unpacked by the caller, as seen in the following excerpt from `lookup`

```
...
| LetrecEnv (decs, saved_env)    ->
  let dec = find_dec decs id in
  match dec with
  | None -> lookup saved_env id
  | Some (Ast.Dec(name, var, body)) -> ProcVal(var, body, env)
```

2. `eval` in the file `interp.ml`. The case that must be updated is this one:

```
...
| Letrec(decs, e2) ->
  (* TODO evaluate e2 with a new LetrecEnv *)
  failwith "Implement me"
```

4 Submission instructions

Submit a file named `HW4_<SURNAME>.zip` through Canvas. Include only the supporting files uploaded into Canvas but where `interp.ml` and `ds.ml` have been completed, as described in this document. Please write your name in the source code using comments.