CS 496: Homework Assignment 5
Due: 7 April, 11:55pm

1 Assignment Policies

Collaboration Policy. Homework will be done individually (unless indicated otherwise by the instructor): 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

The aim of this assignment is to extend the interpreter for EXPLICIT-REFS in two different ways.

1. The first extension consists in allowing for procedures to declare multiple parameters. Consequently, the language should also allow a call to a procedure to receive multiple arguments.
2. The second extension consists in adding a `for` construct.

The resulting language will be called EXPLICIT-REFS-Plus.

3 Extension 1: Functions with Multiple Parameters

Detailed indications on each of the modifications will be provided below. Before doing so however, we give an example of a program written in EXPLICIT-REFS-Plus:

```
let a = newref(3)
in let f = proc(x,y) { setref(x, deref(x) - y) }
   in begin
```

```
      (f a 2);
      deref(a)
    end
```

Here `f` is defined to be a function that takes two parameters `x` and `y` which are separated by commas.

This extension to EXPLICIT-REFS is not strictly necessary since we can already write the following code which although equivalent in its behavior, is less clear.

```
let a = newref(3)
in let f = proc(x) { proc(y) { setref(x, deref(x) - y) }}
   in begin
      ((f a) 2);
      deref(a)
    end
```

3.1 Modifying the Language Grammar

We accommodate for our multiple parameters we must modify our AST located in `ast.ml` as follows:

- Proc

  ```
  type expr =
    ...
    | Proc of string*expr
  ```

  Has been replaced with the following code:

  ```
  type expr =
    ...
    | Proc of (string list)*expr
  ```

- App

  ```
  type expr =
    ...
    | App of expr*expr
  ```

  Has been replaced with the following code:

  ```
  type expr =
    ...
    | App of expr*(expr list)
  ```

- Letrec

```
type expr =
  ...
  | Letrec of string*string*expr*expr
```

Has been replaced with the following code:

```
type expr =
  ...
  | Letrec of string*(string list)*expr*expr
```

We must then modify our expression values located in `ds.ml` to reflect the changes in our new AST:

- ProcVal

```
type exp_val =
  ...
  | ProcVal of string*Ast.expr*env
```

Has been replaced with the following code:

```
type exp_val =
  ...
  | ProcVal of (string list)*Ast.expr*env
```

- ExtendEnvRec

```
type exp_val =
  ...
  | ExtendEnvRec of string*string*Ast.expr*env
```

Has been replaced with the following code:

```
type exp_val =
  ...
  | ExtendEnvRec of string*(string list)*Ast.expr*env
```

These changes have all been performed for you in the stub but it is critical that you understand the reason for each change.

3.2 Modifying the Interpreter

You may now address the final step, namely updating the interpreter itself. Update the following two cases of the `eval_expr` function in `interp.ml` so that it is capable of handling the language extension:

```
let rec eval_expr (en:env) (e:expr):exp_val =
  ...
  | App(e1, e2) -> failwith "implement me!"
  | Proc(x, e) -> (* only change x to xs *)
  | Letrec(id, param, body, en) -> (* only change param to params *)
```

4 Extension 2: For Loops

Here is an example that uses the `for` loop extension.

```
let x = newref(0)
in begin
     for i=1 to 10 (
      setref(x, deref(x) - (-1))
     );
     deref(x)
   end
```

The result of running this program should be `NumVal 10`. The following code should evaluate to `NumVal 55`:

```
let x = newref(0)
in begin
     for i=1 to 10 (
      setref(x, deref(x) - (0 - i))
     );
     deref(x)
   end
```

Here are some variants:

```
let x = newref(0)
in begin
     for i=1 to 1 (
      setref(x, deref(x) - (-1))
     );
     deref(x)
   end
```

This returns `NumVal 1` because it performs one iteration. However, the following program:

```
let x = newref(0)
in begin
     for i=10 to 1 (
      setref(x, deref(x) - (-1))
     );
     deref(x)
   end
```

should return `NumVal 0` since it performs no iterations.

Some more examples:

```
let l = 1 in
let u = 10 in
let x = newref(0)
in begin
     for i=l to u (
      setref(x, deref(x) - (-1))
     );
     deref(x)
   end
```

should return `NumVal 10` and

```
let l = 1 in
let u = 10 in
let x = newref(0)
in begin
     for i=l to u - 1 (
      setref(x, deref(x) - (-1))
     );
     deref(x)
   end
```

should return `NumVal 9`.

4.1 Modifying the Language Grammar

We must simply update our AST in `ast.ml` to reflect the newly added For Loop expression.

```
type expr =
  ...
  | For of string*expr*expr*expr
```

4.2 Modifying the Interpreter

Add a new case in the definition of `eval_expr` to handle the for loop. Your solution **must** use the `for` construct available in OCaml for implementing the behavior of this extension.

5 Submission instructions

Submit a file named `HW5_<SURNAME>.zip` through Canvas. It should include all the files required to run the interpreter. Please include your name and pledge in the `interp.ml` source file.