CS4215 Programming Language Implementation

Lab task for Week 07
Interpreters for simPL

1. Download a folder named lab07.zip from IVLE workbin. The folder contains the following:
   (a) `sPL.ml` which contains the AST of simPL language.
   (b) `sPLc.ml` which contains the AST of a core internal language for simPL.
   (c) `sp*.spl` which contains some test examples. A summary of the test files is provided in `info_test.txt`.
   (d) an incomplete version of `sPL_type.ml`. Please replace it with a copy of the type inference system you have developed in lab06. In case you did not complete this, a copy of our solution will be posted after the deadline of lab06 has expired.
   (e) Use `make lab07` to produce an executable, called `spli`.
   (f) `test7.sh` which is a script to test your `spli` against given test examples and output the results in `out.sp*`.
   (g) `diff7.sh` which is a script to compare your outcome against our expected answers in `test/ref7.out.s*`
2. Deadline for Lab07 is 9March (Sat) 4pm.
3. Your task is to implement an interpreter for simPL that works entirely by substitution. The initial code for this interpreter is a method in the `sPL_inter1.ml` module.

   ```
   eval (e:sPL_expr) : sPL_expr
   ```

   This interpreter will make heavy use of the following substitution operation in `sPL_inter.ml`:

   ```
   apply_subs (ss:(id*sPL_expr)list)
         (e:sPL_expr) : sPL_expr
   ```

   You must implement this substitution to avoid both **name clash** and **name capture**. Name clash is avoided by removing substitution that clash with bound variables. Name capture is avoided by renaming bound variables to use fresh names. Fresh identifier names can be generated by a `names` generator that we have provided in `Debug.Basic` module (see `debug.ml`). You should design a couple of your own test harness when completing this substitution method.

4. After you have completed the code for `apply_subs`, you must implement some missing code for the substitution-based interpreter, called `eval`, in `sPL_inter1.ml`. As usual, complete the code at those places marked by `failwith "TO BE IMPLEMENTED"`. You may check your solution against our expected answers by using script `test7.sh`, and then `diff7.sh`.

5. **BONUS 10% :** *(You are strongly encourage to attempt this part, as it uses concepts on closures that are helpful for understanding how first-class functions can be implemented.)* The interpreter you have just designed is based on substitution which can be expensive for large programs. Furthermore, it is only applicable to functional languages and may not work well for imperative-style languages. For this extra assignment, you are to implement an interpreter that uses an environment of values for its variables. This interpreter will need to build closures to represent function objects. A closure is essentially a function together with an environment for its free variables. A representation of it can be found in `sPL_inter2.ml` as an instance of type `sPL_value`. An environment of values for its free variables can be captured by `(sPL_value Environ.et)`. However, we have used instead `((sPL_value ref) Environ.et)`, so as to support circular closure (that points to itself) The new interpreter now have the following interface:

   ```
   evaluate (env:(sPL_value ref) Environ.et)
                             (e:sPL_expr) : sPL_value
   ```

   Complete the code, and test it using the scripts `test7a.sh` and `diff7.sh`. Your code is correct if the output only differ by the way it represents functions, namely closures in this version of the interpreter.