CS4215 Programming Language Implementation

Lab task for Week 6
Type System and Preprocessor for simPL

1. Download a folder named lab06.zip from IVLE workbin. The folder contains the following:
   (a) `sPL.ml` which contains simPL language.
   (b) `sPLc.ml` which contains a core internal language.
   (c) `sPL_type.ml` which contains partial code on type checking and preprocessing.
   (d) `s*.spl` which contains some test examples. A summary of the test files is provided in `info_test.txt`.
   (e) `./bincomp6.sh` which is shell script with a sequence of compilation commands produce an executable, called `splt`.
   (f) `test6.sh` which is a script to test your `splt` against given test examples and output the results in `out.sp*`.
   (g) `diff6.sh` which is a script to compare your outcome against our expected answers in `test/ref6.out.sp*`
2. Deadline for Lab06 is 9March (Fri) 6pm.
3. You are to do your coding only in `sPL_type.ml`.
   Complete the code for `type_infer` method at those places marked by `failwith "TO BE IMPLEMENTED"`. You may check your solution against our expected answers using script `test6.sh`, then `diff6.sh`. Please remember to fill an inferred type for function used in each application.
4. The pre-processor into core language is meant to translate away partial applications and let constructs by a method, called `trans_exp`. We have provided the codes to eliminate partial applications. Complete the preprocessing code for removing let construct at those places marked by `failwith "TO BE IMPLEMENTED"`.
5. **BONUS 10% :** The current simPL language requires type annotations for functions and let constructs, but not applications. We can minimise programmer effort by making some of these type annotations optional. The places (or features) where type annotations are currently expected include: (i) body of `let` (ii) local definitions of `let` (iii) function definition

(iv) recursive function definition We suggest you make the types at (i) and (ii) optional, since these are more easily achieved. You must do the following for these two features of the `let` construct.

- change the corresponding type of each feature to option type in `sPL.ml` (but keep the types required in `sPLc.ml`)
- change parser to make the type declaration optional for these features
- change `type_infer` to infer types when these are not given
- you must remember to add the inferred type to the core sPLc expression output by `type_infer`.

Those of you who wish to understand how type inference can be built in a more systematic way may wish to study the Hindley-Milner type inference system, and perhaps rewrite the current type inference in that elegant framework! A good starting point for this adventure is the following.

`http://steshaw.org/hm/`