Homework: TypeLang

Learning Objectives:

1. Understanding and implementing typing rules

Instructions:

- Total points 53 pt
- Early deadline: Nov 14 (Wed) 2018 at 6:00 PM; Regular deadline: Nov 16 (Fri) 2018 at 6:00 PM (or till TAs start grading the homework)
- Download hw8code.zip from Canvas. Interpreter for Typelang is significantly different compared to previous interpreters:
  - Env in Typelang is generic compared to previous interpreters.
  - Two new files Checker.java and Type.java have been added
  - Type.java defines all the valid types of Typelang.
  - Checker.java defines type checking semantics of all expressions.
  - Typelang.g has changed to add type information in expressions. Please review the changes in file to understand the syntax.
  - Finally Interpreter.java has been changed to add type checking phase before evaluation of Typelang programs.
- Set up the programming project following the instructions in the tutorial from hw2 (similar steps)
- Extend the Typelang interpreter for Q1 - Q6.
- How to submit:
  - Please submit your solutions in one zip file with all the source code files (just zip the complete project's folder).
  - Write your solutions to question 7 in a HW8.scm file and store it under your code directory.
  - Submit the zip file to Canvas under Assignments, Homework 8.

Questions:

1. (10 pt) Implement the type rules for the memory related expressions:
   (a) (5 pt) DerefExp: Let a deref expression be (deref e1), where e1 is an expression.
   - if e1's type is ErrorT then (deref e1)'s type should be ErrorT

- if e1's type is RefT then (deref e1)'s type should RefT.nestType(). Note that nestType() is method in RefT class.
- otherwise, (deref e1)'s type is ErrorT with message "The dereference expression expect a reference type " + "found " + e1's type + " in " + expression.

Note that you have to add e1's and e2's type and expression in the error message. Examples:
$ (deref (ref : num 45))
45
// no explicit error cases
$ (deref 45)
Type error: The dereference expression expects a reference type, found num in (deref (num 45))

(b) (5 pt) AssignExp: Let a set expression be (set! e1 e2), where e1 and e2 are expressions.

- if e1's type is ErrorT then (set! e1 e2)'s type should be ErrorT
- if e1's type is RefT and nestedType of e1 is T then
  - if e2's type is ErrorT then (set! e1 e2)'s type should be ErrorT
  - if e2's type is typeEqual To T then (set! e1 e2)'s type should be e2's type.
  - otherwise (set! e1 e2)'s type is ErrorT with message "The inner type of the reference type is " + nestedType T + " the rhs type is " + e2's type + " in " + expression
- otherwise (set! e1 e2)'s type is ErrorT with message "The lhs of the assignment expression expect a reference type found " + e1's type + " in "+ expression.

Note that you have to add e1's and e2's type and expression in the error message. Examples:
$ (set! (ref : num 0) #t)
Type error: The inner type of the reference type is number the rhs type is bool in (set! (ref 0) #t)
$ (set! (ref : bool #t) (list : num 1 2 3 4 5 6))
Type error: The inner type of the reference type is bool the rhs type is List<number> in (set! (ref #t) (list 1 2 3 4 5 6))

2. (10 pt) Implement the type checking rules for list expressions

(a) (5 pt) CarExp: Let a car expression be (car e1), where e1 is an expression.

- if e1's type is ErrorT then (car e1)'s type should be ErrorT
- if e1's type is PairT then (car e1)'s type should be the type of the first element of the pair
- otherwise, (car e1)'s type is ErrorT with message "The car expect an expression of type Pair, found"+ e1's type+ "in" + expression

Note that you have to add e1's type and expression in the error message. See some examples below.
$ (car 2)
Type error: The car expect an expression of type Pair, found num in (car 2)
$ (car (car 2))
Type error: The car expect an expression of type Pair, found num in (car 2)

(b) (5 pt) ListExp: Let a list expression be (list : T e1 e2 e3 ... en), where T is type of list and e1, e2, e3 ... en are expressions

- if type of any expression ei, where ei is an expression of element in list at position i, is ErrorT then type of (list : T e1 e2 e3 ... en) is ErrorT.
- if type of any expression ei, where ei is an expression of an element of list, is not T then type of (list : T e1 e2 e3 ... en) is ErrorT with message "The " + index + " expression should have type " + T + " found " + Type of ei + " in " + "expression". where index is the position of expression in list's expression list.
- else type of (list : T e1 e2 e3 ... en) is ListT.

Note that you have to add ei's type and expression in the error message. Some examples appear below.
$ (list : bool 1 2 3 4 5 6 7)
Type error: The 0 expression should have type bool, found number in (list 1 2 3 4 5 6 7)
$ (list : num 1 2 3 4 5 #t 6 7 8)
Type error: The 5 expression should have type number, found bool in (list 1 2 3 4 5 #t 6 7 8)

3. (5 pt) Implement typing rules for CompoundArithExp expressions.

Let a CompoundArithExp be (ArithExp e1 e2 e3 ... en), where e1, e2, e3... en are expressions.

- if type of any expression ei, where ei is an expression of element in list at position i, is ErrorT then type of (list : T e1 e2 e3 ... en) is ErrorT.
- if type of any expression ei, where ei is an expression of element in list at position i, is not NumT then type of (list : T e1 e2 e3 ... en) is ErrorT with message: "expected num found " + ei's type + " in " + expression
- else type of (ArithExp e1 e2 e3 ... en) is NumT.

Note that you have to add ei's type and expression in the error message. Some examples appear below.
$ (+ #t 6)
Type error: expected num found bool in (+ #t 6)
$ (+ 5 6 7 #t 56)
Type error: expected num found bool in (+ 5 6 7 #t 56)
$ (* 45.0 #t)
Type error: expected num found bool in (* 45.0 #t)
$ (/ (list : num 3 4 5 6 7) 45)
Type error: expected num found List<number> in (/ (list 3 4 5 6 7) 45)

4. (5 pt) Implement the type rules for comparison expressions:

BinaryComparator: Let a BinaryComparator be (binary operator e1 e2), where e1 and e2 are expressions.

- if e1's type is ErrorT then (binary operator e1 e2)'s type should be ErrorT
- if e2's type is ErrorT then (binary operator e1 e2)'s type should be ErrorT

- if e1's type is not NumT then (binary operator e1 e2)'s type should be ErrorT with message : "The first argument of a binary expression should be num Type, found " + e1's type + " in " + expression.
- if e2's type is not NumT then (binary operator e1 e2)'s type should be ErrorT with message : "The second argument of a binary expression should be num Type, found " + e2's type + " in " + expression.
- otherwise (binary operator e1 e2)'s type should be BoolT.

Note that you have to add e1's and e2's type and expression in the error message. Some examples appear below.
$ (< #t #t)
Type error: The first argument of a binary expression should be num Type, found bool in (< #t #t)
$ (> (list: num 45 45 56 56 67) 67)
Type error: The first argument of a binary expression should be num Type, found List<number> in (> (list 45 45 56 56 67) 67)

5. (5 pt) Implement the type checking rules for conditions expressions.

   IfExp: Let a IfExp be (if cond then else), where cond, then, else are expressions.

   - if cond's type is ErrorT then (if cond then else)'s type should be ErrorT
   - if cond's type is not BoolT then (if cond then else)'s type should be ErrorT with message: "The condition should have boolean type, found " +cond's type+ " in " + expression
   - if then's type is ErrorT then (if cond then else)'s type should be ErrorT
   - if else's type is ErrorT then (if cond then else)'s type should be ErrorT
   - if then's type and else's type are typeEqual then (if cond then else)'s type should be then's type.
   - else (if cond then else)'s type should be ErrorT with message: "The then and else expressions should have the same " + "type, then has type " + then's type + " else has type " +else's type+ " in " + expression.

   Note that you have to add cond's, then's and else's type and expression in the error message. Some examples appear below.
   $ (if 5 56 67)
   Type error: The condition should have boolean type, found number in (if 5 56 67)
   $ (if #t #t 56)
   Type error: The then and else expressions should have the same type, then has type bool else has type number in (if #t #t 56)

6. (10 pt) Implement the type checking rules for function calls.

   CallExp: Let a call expression be (ef e1 ... en) with type:

   - if the type of ef is ErrorT, return ErrorT
   - if the type of ef is not FuncT, the type of the call expression is ErrorT, reporting the message "Expect a function type in the call expression, found "+ef's type+"in "+ expression

- if any one of e1, e2, ...en has ErrorT, the call expression has ErrorT
- given that ef has FuncT (T1 ... Tn)->Tb, if the actual parameter ei does not have a type Ti, the call expression has ErrorT, reporting the message "The expected type of the " + i + "th actual parameter is " + Ti + ", found " + ei's type +"in "+expression
- otherwise, the type of call expression is Tb

Some examples appear below.

```
$(define add: (num num num -> num) (lambda (x: num y: num z: num) (+ x (+
    y z))))
```

$ (add 5 56 #t)
Type error: The expected type of the third actual parameter is number, found bool in (add 5 56 #t)
$ (3 4)
Type error: Expect a function type in the call expression, found number in (3 4)

7. (8 pt) For all the above typing rules (total 8 of them) you implement, write a typelang program for each type rule to test and demonstrate your type check implementation. (You can use typelang.g in hw8code.zip as a reference for the syntax of TypeLang). For each expression, put in comments which type rules the expression is exercising. For example:
$ (list: num 45 45 56 56 67) // test correct types for list expressions
$ (* 45.0 #t) // test incorrect types for compound arithmetic expressions