CSE 3302 Assignment 1

Due: 9 Feb 2010, 11am

There are two problems on this assignment, worth 15 points each. The second problem involves a bit of Scala programming—it would be wise to start early to ensure you can run the Scala compiler. For Problem 1, please hand in a hard copy at the beginning of class on 9 Feb, or email it to me before the deadline. Type or write neatly. For Problem 2, email your version of Assign.scala to nystrom@uta.edu before the deadline.

1 Errors

1. [8 pts] Choose any compiled programming language (e.g., Java, C, Scala). Give two examples each of the following errors caught during different stages of compilation or execution:
   (a) A lexical error caught by the scanner.
   (b) A syntax error caught by the parser.
   (c) A semantic error caught by the type checker.
   (d) A run-time error.

2. [3 pts] Why might it be infeasible—or indeed, impossible—for the run-time errors you listed in (d) above to be caught at compile time?

3. [4 pts] Languages can be either statically or dynamically typed. In statically typed languages, type errors (e.g., assigning a string into an integer variable, adding a boolean to a float) are caught at compile time; in dynamically typed languages, type errors are caught at run-time. What are some advantages and disadvantages of each approach.

2 While loops

The file http://ranger.uta.edu/~nystrom/courses/3302/hw1/Assign.scala contains a simple interpreter written in Scala for a language with arithmetic and boolean expressions, assignment, and if. In this problem, you will extend the interpreter to support some new expressions and statements.

You can download the Scala tools from http://www.scala-lang.org/downloads. See the slides for Lecture 3 for other scala resources, or browse scala-lang.org. To compile the interpreter, do:

$ scalac Assign.scala

($ is the command-line prompt and should not be typed in.)

To run the interpreter on expression exp, do:

$ scala Assign exp

This should print the result of evaluating exp. For example:

$ scala Assign "1+2"
parsed = Bin(+,Num(1),Num(2))
final store = Map()
final value = Num(3)
$$\text{scala Assign } "x := 1; x+2"$$

parsed = Seq(Assign(x,Num(1)),Bin(+,Var(x),Num(2)))
final store = Map(x -> Num(1))
final value = Num(3)

The language that will be interpreted has the following grammar, written in EBNF notation:

```
Seq ::= Stm ( ";" Stm )*
Stm ::= ID "=" Exp
   | "if" Exp "then" Stm else Stm
   | "while" Exp "do" Stm
   | Exp
Exp ::= Rel
Rel ::= Plus ( ";" Plus )*    
   | Plus ( ">" Plus )*    
   | Plus ( ">=" Plus )*    
   | Plus ( "==" Plus )*    
   | Plus ( "!=" Plus )*    
Plus ::= Mult ( "+" Mult )*    
   | Mult ( "+=" Mult )*    
   | Mult ( "+|" Mult )*    
Mult ::= Factor ( "+" Factor )*    
   | Factor ( "+=" Factor )*    
   | Factor ( "+|" Factor )*    
Factor ::= ID
   | NUM
   | "true"
   | "false"
   | "(" Seq ")"
```

1. [10 pts] Add support for short-circuiting || and && expressions, similar to those found in Java or C. The parser has already been extended. You need only add a case to the eval method to handle these constructs. Hint: translate these expressions into an if.

2. [5 pts] Add support for while statements to the interpreter. The parser has already been extended. You need only add a case to the eval method to handle while. A while statement should evaluate its condition, and if true, evaluate its body, possibly modifying the store, and then repeat. If the condition evaluates to false, the while statement should evaluate to 0.