Assignment #6: Interpreter Phase III

Due: March 5, 2010 11:59pm

Overview

For this assignment you will extend your interpreter to support anonymous functions and closures. This requires a modification to the parser and a rearchitecture of the implementation of function invocations. You can start with either the solution to assignment #4 or the solution to assignment #5.

The $D^\flat$ Language

The following is the only modification to $D^\flat$'s syntax that is necessary to support this feature (though additional modifications could certainly be made to simplify the syntax).

factor → (**expression**) | `id` {(**arguments**)}$_{opt}$ | `number` | `true` | `false` | `unit`
| `fn` (**parameters**) **declarations compound_statement**

Part 1: Parsing

Modify the parser to support the use of anonymous functions. This will also require a modification of the abstract syntax.

Part 2: Function Invocation with Closures

Extend the interpreter to support function invocation using static scoping as implemented by closures. This differs from the previous assignment in that your state will no longer model a stack of activation records. Moreover, there is no need to model the global variables separately (they will simply be at the end of the environment chain).

Semantic Note: The write statements should print only `<fun>` when given an expression that results in a function.

Part 3: Dynamic Typing

Dynamically verify that the new expressions (i.e., function expressions) do not violate the type system. Specifically, anonymous functions can only be assigned to variables (those variables can then, in turn, be used in an invocation). Any invocation of such a function must check for a number of actual arguments equal to the number of formal parameters.

Notes

- Strive for simplicity in your programming. Do not try to be fancy. For example, the syntax graphs translate directly into code. Learn and use that technique.
- Test data with "correct" output will also be given. Your output can differ in minor ways (e.g., syntax error message format) from this "correct" output yet still be correct; it is up to you to verify your code's correctness.
- Your program will be exercised by some shell scripts, which will be provided. These scripts depend on the exit status of your program. If your program detects a "serious" error, your program should use "OS.Process.exit" to terminate.
- Grading will be divided as follows.

Part	Percentage
1	20
2	70
3	10

- Get started **now** to avoid the last minute rush.