Assignment #5: Interpreter Phase II

Due: February 26, 2010 11:59pm

Overview

For this assignment, you will extend your interpreter to support functions, function invocations, and return statements. You may modify either your solution to Phase I or the given solution to Phase I.

Educational Goals

- Parameter Passing
- Static Scope
- Dynamic Semantics (Evaluation)

Part 1: Evaluation of Functions without `return`

Extend your interpreter to support function invocations. For this assignment, evaluation of a function is performed in much the same way as in most imperative languages. Specifically, evaluation of a function must support static scoping.

When invoking a function, you will need to access the code in the body of the function. You will likely find it easiest to modify your state to support binding function names to the associated abstract syntax tree for the function. Then, when invoking the function, you can retrieve the code from the state by name.

Furthermore, you will likely want to modify your state abstraction to support some notion of a global environment and a stack of activation records.

Since the **return** statement is not required for this part, we will adopt the admittedly bizarre semantics that a function without a return will result in the `unit` value (i.e., once control reaches the end of a function, the function "returns" `unit`).

Finally, all arguments are passed by value and an invocation must pass a number of arguments equal to the number expected by the function.

Part 2: Evaluation of Functions with `return`

Complete the implementation of the language by providing support for the `return` statement. When executed, the evaluation of the function terminates with the value of the specified expression.

Notes

- Strive for simplicity in your programming.
- Test data with "correct" output will be given. Your output can differ in minor ways from this "correct" output yet still be correct; it is up to you to verify your code's correctness. The output from the D$^\flat$ programs (through `print` statements) **must** match the corresponding "correct" output exactly.
- 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 OS.Process.failure".
- You must follow the steps described above in developing your program. Grading will be divided as follows.

Part	Percentage
1	80
2	20