Challenge

Spare Time Teaching

March 23, 2014

You may not add parameters or change the output.

Introduction

As we are studying languages we need to understand compilers and interpreters. The first phase of both is parsing, often followed by some sort of static analysis, such as checking that all variables are bound. Then as a final phase comes code generation.

We have very high expectations of our compilers such as; it should be fast, it should give meaningful errors, and it should be correct.

Problem

This time we ask you to write a compiler for a small (artificial) language called Block. The language looks as follows:

```
[ decl x
; [ use x
  ; use w
  ; decl y ]
; decl x
; use y ]
```

That is, it consists of blocks, use and decl-statements. The semantic rules are: You may use variables that are declared in any block, you may not declare variables with the same name more once in a block. Therefore the example program contains two errors this means that the output of your compiler should be:

```
[ decl x
; [ use x
  ; use w -- undefined variable w
  ; decl y ]
; decl x -- x is already defined
; use y ]
```

If we remove those two lines we should get:

```
enter 1
init 0, 1
enter 1
load 0, 1
init 0, 2
leave
load 0, 2
leave
```

The syntax is, '`enter` *size*' opens a new heap-frame consisting of *size* cells, '`load` *cell*`,` *frame*' fetches a reference to *cell* in *frame*, and '`init` *cell*`,` *frame*' initializes the content of *cell* in *frame*. A load should refer to the latest defined or the last in the program.

The challenge is to write a one-pass compiler, for BLOCK, thus it should do both parsing, static analysis, and code generation in a single pass, giving output as illustrated above. The challenge should be solved in one ocaml-yacc file (and an ocaml-lex file).

Note: You do not need to write to a file.