Scheme PROJECT
198:314 Principles of Programming Languages
Spring 1999
due: April 29, 1999, at midnight

Project Description
In this project, you are asked to write a compiler for the expression language introduced in homework 4. In order to make parsing simpler, the syntax has been slightly modified as follows:

\[
\begin{align*}
\text{<program>} &::= (\text{<stmt_list>}) \\
\text{<stmt_list>} &::= (\text{<stmt>})\text{<stmt_list>} | \text{<stmt>} \\
\text{<stmt>} &::= \text{id} := \text{<expr>} | \\
& \quad \text{id} := \text{intnum} | \text{id} := \text{floatnum} | \\
& \quad \text{print} (\text{id}) \\
\text{<expr>} &::= \text{id} + \text{id} | \text{id} * \text{id} | \text{id} / \text{id}
\end{align*}
\]

\text{id} is an alpha-numerical name with low-case letters (e.g.: \text{a1, bb}), \text{intnum} is an integer number (e.g.: 4, 55), and \text{floatnum} is a floating point number (e.g.: 3.14, 44.01).

The parser has been implemented and is provided to you. For example, the program

\[
(\text{d} := 2) \\
(\text{e} := 3) \\
(\text{a} := \text{d} + \text{e}) \\
(\text{b} := 2.5) \\
(\text{c} := \text{b} * \text{a}) \\
(\text{print} (\text{c}))
\]

has the following syntax tree represented as a Scheme list:

\[
'(\text{assign d (intnum 2)}) \\
(\text{assign e (intnum 3)}) \\
(\text{assign a (expr d + e)}) \\
(\text{assign b (floatnum 2.5)}) \\
(\text{assign c (expr b * a)}) \\
(\text{print c})
\]

Handout
The directory \text{~uli/cs314/project3} on the undergraduate UNIX cluster contains the initial code for your compiler in file “\text{compiler.ss}”. The function that you will have to write is \text{gen-code}, which takes a syntax tree as input. You will have to implement type checking and code generation for the type system and architecture described in our 4th homework. A sample solution of the homework is available on our web page.
File “program.ss” contains two test programs. Of course, your compiler is expected to work on all syntactically correct programs, not only the two provided test cases. You may want to add new test cases in “program.ss” to test your compiler.

Use the Scheme interpreter scm for your project. Once you call scm, you can load your compiler using the command (load ’‘compiler.ss’’). This will also load all programs defined in “program.ss”. To run the provided compiler on input program prog1, just say (compile prog1). This function call will return a number which you can ignore. The output of your compiler will be written to file “a.out”.

START EARLY!
Good luck.