

## CAS 743 — Functional Programming

16 February 2006

### Imperative Programming Language Interpreter

#### 4.1

Use the interaction framework from Assignment 1 to produce `ghci`-like interaction, for example:

```
let k = 3 * 4
> k = 12
let n = 3 * (k + 2)
> n = 42
n - k
> 30
n + m
> error: Variable "m" undefined!
```

#### Hints:

- Determine the **abstract** input grammar for this system (as a datatype) — you will at least need two non-terminals.
  - Determine the **concrete** input grammar for this system.
  - Define a parsing function for these inputs using *ReadPrec* or *Parsec* (both in *Text . ParserCombinators*).
  - Determine an appropriate state datatype for this interaction.
  - You will need an evaluation function for expressions with variables; this will of course need a variable assignment.
  - Define the *Transition* function for this interaction.
  - Define a *main* computation (using *runprocess* from Assignment 1) and compile your program.
- Test both the interpreted and the compiled versions.

## 4.2

Modify your solution to 4.1 to produce an interpreter for a simple imperative programming language, including at least assignments, conditionals, while loops, and print statements.

Enable the following interactions:

- Loading programs, with “:load *progfile*”. (Function and procedure definitions and calls are not yet necessary.)
- Printing the loaded program to the screen, with “:list”.
- Inspecting and changing the current memory state (as in 4.1).
- Clearing the current memory state with “:clear”.
- Running the loaded program from the current memory state, with “:run”.
- **Optional:** Enable a debugging or tracing mode of your own design.

## 4.3

- (a) Propose and explain a new state type that would enable you to add a “read(*var*)” statement to the language, the execution of which would assign to *var* a value read from the keyboard.
- (b) **Optional:** Implement “read(*var*)”.