Assignment 4

CS 4400 / CS 5400 Programming Languages

General

Due: Friday, November 22, 11:59pm.

Instructions:

1. Submissions are handled via the Khoury Handin server: https://handins.ccs.neu.edu/
2. Download the assignment pack from https://vesely.io/teaching/CS4400f19/m/cw/04/Assignment4.zip.
3. You can choose to complete this assignment as a pair. If you work as a pair, submit as a pair. Completing an assignment with a partner but submitting individually is considered cheating.
4. Complete the information in `Assignment4.hs` and submit *all* `.hs` files included in the original pack.
5. You are free to add top-level definitions, but add them to the bottom of each file, below the separator line.
6. Each top-level function must include a type signature, followed by one or more defining equations.
7. Make sure your file loads into GHCi or can be compiled by GHC without any errors.
8. Use the provided `SimpleTests` module to write your own tests in the `tests` function. If appropriate, feel free to use QuickCheck for tests.
9. If some parts of the assignment are unclear, ask. Please do not wait until just before the submission deadline.

Grade: To calculate your grade, we will take the following into account:

a) Quality of submission: Does your code compile without errors? Did you follow the above steps?
b) Correctness: How well does it implement the specification?
c) QA: How well did you test your code?

MiniImp

In the lecture, we have looked at a simple imperative language with assignment, while loops and printing values to an output stream. In this assignment, we will extend this language with more forms of looping, reading in values from an input stream, and arrays.

The Haskell module `MiniImp` contains an initial implementation of an interpreter. Expressions are evaluated relative to the current store, which is used to look up the meaning of variables

```
evalExpr :: Store Value -> Expr -> Maybe Value
```

The execution of statements is implemented by the function

```
execStmt :: (Stmt, Store Value, In) -> Maybe (Store Value, In, Out)
```

which takes an input *configuration* – a tuple containing the statement to be executed, the current state of the store, and the input stream. It returns an output configuration, containing the (potentially updated) store, the input stream (with values possibly removed), and the output stream.

The input and output streams are represented as lists of integers. For input, the head of the list is the next value to be read, new input is added at the end of the list. The output stream grows left to right: the head is the oldest value printed by the program.

Exercises

Exercise 1 Complete the basic implementation of MiniImp. This includes the commands `Assign`, `Seq`, `While`, `If`, and `Print`, and expressions

Exercise 2 Implement the semantics of `DoWhile`: `DoWhile b c` first evaluates its body `b` and then continues repeatedly executing `b` while the condition `c` is true. Unlike with `While`, the body `b` is executed at least once.

Exercise 3 Implement `For`, a for-loop construct. `For x e1 e2 b` evaluates the expression `e1` to a value `v1` and assigns it to the variable `x`. Then it evaluates `e2` to `v2`. If the value stored in `x` is *greater than* `v2` the for-loop is done. If the value in `x` is *less than or equal to* `v2`, the body, `b`, is performed. Then `x` is incremented by 1 and the for-loop is run again.

Exercise 4 Implement `Read`, which reads in a value from the input stream and saves it in the given variable.

Exercise 5 Implement arrays. Array indices start from zero. We will represent arrays as another type of value. Choose an appropriate representation of the array value and implement the semantics of:

- the statement `NewArray x e1 e2`, which allocates a new array in the variable `x`. The size is given by the value of `e1` and all fields are to be initialized to the value of `e2`.
- the statement `Set x e1 e2`, which updates the array stored in `x` by storing the value of `e2` at the index resulting from evaluating `e1`.
- the expression `Get x e`, which returns the value stored in the array `x` at the index given by the value of `e`.
- the statement `ForEach x a b` which iterates over the array `a`. In each iteration, the array element is assigned to `x` and the body `b` is executed.

For example, evaluating the program

```
Seq (NewArray "array" (Val (Num 5)) (Val (Num 1)))
    (ForEach "x" "array" (Print (Var "x")))
```

should result in the output stream:

```
[Num 1, Num 1, Num 1, Num 1, Num 1]
```

For `Set` and `Get`, if the index is out of bounds, the execution should fail.

Exercise 6 Write a MiniImp program which reads 5 numbers from the input stream, stores them in an array of an appropriate size. Then it reads one more number and prints each element of the array multiplied by that number.

For example, if the program is executed with input stream

```
[1, 2, 3, 4, 5, 2]
```

then the output stream should be:

```
[Num 2, Num 4, Num 6, Num 8, Num 10]
```

Exercise 7 (required for CS5400, optional for CS4400) Write a program which keeps reading numbers from the input stream while they are greater than 0, and prints their running sum after each number is read.

For example, executing the program with the input stream

```
[10, 20, 30, 40, 50, 60, 70, 0]
```

should result in the following output stream

```
[Num 10, Num 30, Num 60, Num 100, Num 150, Num 210, Num 280]
```

or

```
[Num 10, Num 30, Num 60, Num 100, Num 150, Num 210, Num 280, Num 280]
```