Programming Languages — Homework 2
A JavaScript expression interpreter

Due: Wednesday, 6 March 2013, 23:55

In this assignment you will implement an interpreter for a subset of JavaScript. For this subset, your interpreter should have the same behavior as the V8, the JavaScript engine used in Chrome and in Node.js.

Feel free to use G+ to ask questions, to discuss corner cases, or to post interesting test cases.

This assignment will be graded out of 10 points.

The language

Your interpreter should implement the following features of JavaScript:

1. The `boolean` constants `true` and `false`
2. The `number` constants
3. The **`undefined`** constant
4. Unary operations: `!` (logical negation) and `-` (unary minus)
5. Binary operations:
   (a) Arithmetic operations: `+`, `-`, `*`, `/`, and `%`
   (b) Equality operations: `==`, `!=`, `===` (strict equality), `!==` (strict disequality)
   (c) Relational operations: `>`, `<`, `>=`, and `<=`.
   (d) Logical operations: `&&`, and `||`
6. Ternary conditional expressions (e.g., `true ? 5 : 0`)

The JavaScript (ECMAScript 5.1) specification can be found at:

```
http://www.ecma-international.org/ecma-262/5.1/
```

You should use this document in order to fully understand the semantics of JavaScript expressions.

For every expression in the subset of JavaScript listed above, the output of your interpreter should match the output of V8. Small differences due to rounding of numbers are allowed.

The V8 implemention may, or may not, agree with the JavaScript specification. If there is a difference, the behavior should match V8, not the specification.

You can download Node.js here `http://www.nodejs.org`. You can find the JavaScript Console in Chrome in the "Developer" submenu of the "View" menu, or in the "Tools" submenu of the "wrench" menu in the toolbar.

Be careful! JavaScript operations have quite a few corner cases. Pay particular attention to the behavior of the operations on the "wrong" type (e.g., `!9` or `false+1`). Also, be warned that some Haskell operators behave slightly differently from the corresponding JavaScript operators.

Template

You should implement your interpreter by modifying the template `hw2.hs` provided on the course web site. The template provides a JavaScript parser an abstract syntax tree type and some utility functions. You need to implement the `step` and `value` functions. Do not change anything else.

Note that the provided parser does not accept the full set of JavaScript expressions enumerated above. For instance, it does not handle numbers of the form "`1.0e-6`". We will not test your interpreter on expressions the parser cannot already handle.

The interpreter accepts one command-line argument, a file name. The interpreter will read the file and evaluate the JavaScript therein. If no argument is provided, the interpreter will provide a REPL, similar to the Node.js REPL (but obviously supporting only the subset of JavaScript outlined above).

You should be able to run the interpreter as follows:

```
runhaskell hw2.hs file.js
```

To compile and run the template, you need to install the `System.Console.Haskeline` library. You can install packages using `cabal`, the Haskell package manager. To install the above library, you can run the following in your Unix shell:

```
cabal install haskeline
```

Submission

1. Complete the survey linked from the course web page after completing this assignment.
2. Submit your code on Moodle by 23:55 on Wednesday, 7 March 2013. Include your name in the file.
3. With your code, submit a set of test cases as a single file, one expression per line.