CS320 Programming Languages
Homework #3: MRFWAE

Due: 27 March 2019

The goal of Homework #3 is to implement the interpreter of MRFWAE, which is the extended version of FWAE with Multi-arugments and Records.

1 Functions with Multiple Arguments

Implement MRFWAE to support any number of arguments to a function (including zero), and any number of arguments (including zero) in a function application:

```
<MRFWAE>  ::= <number>
            | {+ <MRFWAE> <MRFWAE>}
            | {- <MRFWAE> <MRFWAE>}
            | {with {<id> <MRFWAE>} <MRFWAE>}
            | <id>
            | {<MRFWAE> <MRFWAE>*}
            | {fun {<id>*} <MRFWAE>}
```

For parsing, any alphanumeric string other than `with`, or `fun` can be an identifier. At run-time, a new error is now possible: function application with the wrong number of arguments. Your interpreter should detect the mismatch and report an error that includes the words `"wrong arity"`:

```
test(run("{{fun {x y} {+ x y}} 1 2}"), "3")
test(run("{{fun {} {+ 3 4}}}"), "7")
testExc(run("{{fun {x y} {+ x y}} 1}"), "wrong arity")
```

2 Adding Records

Extend your interpreter to support the construction of records with named fields, and to support field selection from a record:

```
<MRFWAE> ::= ...
           | {record {<id> <MRFWAE>}*}
           | {access <MRFWAE> <id>}
```

Adding records means that the language now has three kinds of values: numbers, functions, and records. At run-time, an error may occur because a record is misused as a number, a number is supplied to access, or a record supplied to access does not have the named field. Your error message for the last case should include the words `"no such field"`, otherwise you can make up your own error messages (or just let primitive error checking handle problems, such as trying to add a record to a number).

```
test(run("{access {record {x 1} {y 2}} x}"), "1")
testExc(run("{access {record {x 1} {y 2}} z}"), "no such field")
testExc(run("{record {x {access {record {y 1}} z}}}"), "no such field")
```

Since records are now values, the result of the interpreter can be a record, not only a number or a function. For homework purposes, we don't want to nail down the representation of a record result, because there are many choices. The examples below therefore use `run`, which is a wrapper on your interpreter that just returns a number string if it produces a number, and it returns the string `"function"` if it produces a function value, but it returns the string `"record"` if it produces a record value:

```
test(run("42"), "42")
test(run("{fun {x} x}"), "function")
test(run("{record {x 1}}"), "record")
```