Rudimentary Interpreter CS345H Spring 2011 Due 9:30am Sep 15, 2011

You must do this assignment solo. The goal is to become familiar with Racket and the interpretive approach to analyzing the semantics of programming languages.

Rudimentary Interpreter

Write a parser and interpreter for the WAE language we discussed in class. **The textbook can be of great assistance in this part of the assignment;** they provide the beginnings of a parser, an abstract syntax datatype and an interpreter. Remember, the WAE language has numbers, two arithmetic operators (`+`, `-`), identifiers and `with` expressions. Of course, to handle identifiers and `with` expressions you'll have to implement substitution. (You do not have to implement caching substitution, but you are free to do so.) Finally, both the concrete syntax and abstract syntax are specified by the WAE language's BNF (which can be found in the textbook).

Once you've **written and tested** the parser and interpreter for WAE, extend the language with these features:

Binary arithmetic operators

> In place of having separate rules for `+` and `-`, define a single syntactic rule for all binary arithmetic operators. Parse these into a `binop` datatype variant. Define a table that maps operator names (symbols) to actual functions (Racket procedures) that perform the corresponding operation. Having a single rule like this, accompanied by a table, makes your language easier to extend: once you have modified your parser and interpreter once to support binary operators, you won't need to touch either one to add any number of new ones. To demonstrate this, define multiplication and division (using `*` and `/` to represent them in the language's concrete syntax).

Multi-armed `with`

> Each identifier bound by the `with` expression is bound only in its body. There will be zero or more identifiers bound by each `with` expression. If there are multiple bindings of the same identifier in a single `with` expression's bindings list, your interpreter should halt with an error message. An example:
>
> ```
> {with {{x 2}
>        {y 3}}
>   {with {{z {+ x y}}}
>     {+ x z}}}
> ```
>
> will evaluate to `7`, while
>
> ```
> {with {{x 2}
>        {x 3}}
>         {+ x 2}}
> ```
>
> will halt with an error message.

The syntax of the WAE language with these additional features can be captured using EBNF notation:

```
<WAE> ::= <num>
    | {<op> <WAE> <WAE>}
    | {with {{<id> <WAE>}*} <WAE>}
    | <id>
<op> ::= + | - | * | /
where an <id> is not +, -, *, /, or with.
```

The textbook introduced you to BNF. An extension of this notation, called EBNF (Extended Backus-Naur Form), provides three additional operators:

- ? means that one or more symbols to its left can appear zero or one times.
- * means that one or more symbols to its left can be repeated zero or more times.
- + means that one or more symbols to its left can appear one or more times.

You should turn in a single Racket program containing all of the code needed to run your parser and interpreter. **Implement the function** `parse`, which consumes an expression in the language's concrete syntax and returns the abstract syntax representation of that expression. Also **implement the function** `interp`, which consumes an abstract syntax expression (as returned by the `parse` function) and returns a Scheme number.

Support Code

Your code **must** adhere to the following templates, without any changes:

```
(define-type Binding
  [binding (name symbol?) (named-expr WAE?)])

(define-type WAE
  [num (n number?)]
  [binop (op procedure?) (lhs WAE?) (rhs WAE?)]
  [with (lob (listof Binding?)) (body WAE?)]
  [id (name symbol?)])

;; parse : s-exp -> WAE
;; Consumes an s-expression and generates the corresponding WAE
(define (parse sexp)
  (...))

;; interp : WAE -> number
;; Consumes a WAE representation of an expression and computes
;;   the corresponding numerical result
(define (interp expr)
  (...))
```

Turnin

Use the CS (unix) “turnin” program to submit your work:

1) turnin --submit alexloh cs345-hw1
2) make the submissions as a file named "hw1-{student id}.ss"

[Thanks to the textbook author for this assignment]