HW 4
50pts
Posted Friday, October 17
Due Monday, October 27 at midnight

An Interpreter for PLAN

The goal is to write an interpreter for a simple functional language called PLAN. This is yet another recursive descent parser with associated interpretation (i.e., attribute grammar), but this time it should be written in Scheme. A PLAN program is a list, as defined by the following grammar:

P	$\rightarrow$	`(` `prog` E `)`	PLAN program
E	$\rightarrow$	I	identifier expression
E	$\rightarrow$	C	constant expression
E	$\rightarrow$	`(` `myskip` E `)`	skip expression
E	$\rightarrow$	`(` `myadd` E E `)`	addition expression
E	$\rightarrow$	`(` `mymul` E E `)`	multiplication expression
E	$\rightarrow$	`(` `myneg` E `)`	negation expression
E	$\rightarrow$	`(` `mylet` I E E `)`	let expression
I	$\rightarrow$	`a` | `b` | ... | `z`	
C	$\rightarrow$	`integer`	

Here are five valid PLAN programs:

(1) `(prog 5)`
(2) `(prog (myadd (myadd 7 (myskip (mymul 4 5))) (mymul 2 5)))`
(3) `(prog (mylet z (myadd 4 5) (mymul z 2)))`
(4) `(prog (mylet a 66 (myadd (mylet b (mymul 2 4) (myadd 2 b)) (mymul 2 a))))`
(5) `(prog (mylet x 66 (myadd (mylet x (mymul 2 4) (myadd 2 x)) (mymul 2 x))))`

Each PLAN program and expression evaluates to an integer value. The semantics of a program is defined as follows:

- The entire program `(` `prog` E `)` evaluates to whatever E evaluates to.
- `(` `myskip` E `)` evaluates to the integer value 0, regardless of what the subexpression E looks like.
- `(` `myadd` E E `)` evaluates to the sum of whatever values the two subexpressions evaluate to.
- `(` `mymul` E E `)` evaluates to the product of whatever values the two subexpressions evaluate to.
- `(` `myneg` E `)` evaluates to $X * (-1)$, where X is the integer value that the subexpression evaluates to.

- `( mylet` I E_1 E_2 `)` has the following semantics. First, subexpression E_1 is evaluated. The resulting integer value is "bound" to the identifier I. Then the second subexpression, E_2 is evaluated. The result of that evaluation serves as the value of the entire `mylet` expression. The binding between the identifier I and the integer value is *active only* during evaluation of E_2.
- I evaluates to the value that the identifier is bound to by a surrounding `mylet` expression. If there are multiple bindings for the identifier, the latest such binding is used.
- C evaluates to the value of the integer constant.

Based on these rules, the five programs from above are evaluated as follows:

- Program (1) evaluates to 5
- Program (2) evaluates to 17
- Program (3) evaluates to 18
- Program (4) evaluates to 142
- Program (5) evaluates to 142

Write a Scheme function `myinterpreter` that takes as input a list of PLAN programs and produces a list of the corresponding values. For example,

```
> ( myinterpreter
     '( (prog 5)
        (prog (mylet z (myadd 4 5) (mymul z 2)))
      )
   )
```

Should produce the list

```
(5 18)
```

Please read this carefully before starting to code:

- Submit your intepreter code in file `plan.rkt`. A minimal starter code can be found `plan.rkt`. Don't forget that your implementation should conform to the R5RS language.
- Assume that the list given to the interpreter is not empty and contains only valid PLAN programs. A program is valid in the sense that (i) it is in the language outlined above and (ii) when identifier evaluation happens an enclosing binding exists for that identifier. The input does not contain programs such as `(prog a)` or `(prog (mylet a 5 (myadd b 10)))` and you do not need to worry about error handling.
- Type predicates `integer?` and `symbol?` will be useful. `integer?` checks if the argument is an integer, and `symbol?` checks if the argument is a symbol such as `a`, `b`, etc.
- For the bindings, consider using a list where each element of the list is one specific binding. A binding is really just a pair of an identifier and an integer value.
- The only built-in Scheme functions and forms you are allowed to use are `equal?, car, cdr, cons, cond, if, +, *, null?, symbol?`, and `integer?`.
- Comment your code using the contract style outlined `Contracts.htm`. 40 points are autograded on Submitty and 10 points are awarded for code quality are comments.