Parsing

Programming Language

Purpose Parse an unambiguous, infix-notation language.

Exercises

This homework concerns the language we saw in class that unambiguously deals with summation and products:

```
<AE>   ::= <PROD> + <AE>         (1)
         | <PROD>                (2)

<PROD> ::= <ATOM> * <PROD>       (3)
         | <ATOM>                (4)

<ATOM> ::= <num>                 (5)
         | { <AE> }              (6)
```

Exercise 1 Show the following strings are an AE with a step by step derivation. Use a block comment, and label each expansion with the rule it follows:

```
"5"
"{5}"
"{2 + 5}"
"{{{2 + 5}} * {5 * {{3 + 6}}}}"
```

Exercise 2 Define an abstract syntax tree structure using `define-type` which captures this language. Note that you should have one type per non-terminal!

Exercise 3 Design your `parse-ae` function which takes in a `Sexpr` and outputs an `AE`.

Exercise 4 Design your `parse` function which composes `parse-ae` and `string->sexpr`. Test this function on the above strings.

Exercise 5 Explain the relationship between how many constructors are in the output and how many steps there were to derive the string from <AE>.

Exercise 6 Some of parentheses in the final string given above seem excessive; they are not. Identify a pair of seemingly-excessive parentheses and explain why they are not.

Exercise 7 Replace the right-most sum with a product in the final string. Now are any of the parenthesis excessive? If so, why?

Exercise 8 Leave a comment describing how you feel about programming with an unambiguous, infix-notation language.