ASSIGNMENT #1

SPL PARSER AND LEXER

Lopez | Casey

Created a parser and lexer for a language that supports λ-calculus. The language is based on the grammar defined assignment 1. Also provided facilities for displaying parse and lex information to the screen.

EVALUATION

The program meets the goals of the assignment. The output is given in list form with no special formatting. The code base for the lexer and parser are very modular and are designed so that the grammar and symbol table can be extended at a later time. There is currently limited support for error handling and it does not use the mechanisms provided in Prelude.

PROJECT CONTRIBUTIONS

The resulting project comes from two sets of files. Originally, each team member worked separately to experiment with different approaches to the problem and to gain the same level of understanding of the Haskell language and functional programming. Joint effort was made on the lexer part of the project with each team member rewriting and refactoring parts of the lexer. Lopez's work efforts focused working on I/O and pretty printing while Casey's efforts focused on parsing.

FILE BREAKDOWN

Filename	**Description**
main.hs	Initializes program and handles all I/O
CmdOpts.hs	Facilities for handling and checking command-line arguments
Parser.hs	SPL Parser module, contains all parsing code and data types for AST data structures
Tokenizer.hs	SPL Lexer module, contains all lexer code
PrettyPrint.hs	All Formatting facilities and support for printing most data types found in this project
ListAux.hs	Used to split lists in an unusual manner, needed to print

SPL Revised Grammer

```
program  ::=  Empty
            | statements

statements  ::=  statement
               | statement statements

statement  ::=  expression ;

expression  ::=  term
               | Let Id Eq expression
               | Lamda Id Dot expression

term  ::=  factor
         | factor + term
         | factor - term

factor   ::=  application
            | application * factor
            | application / factor

application ::=  primary
               | primary application

primary  ::=  Identifier
            | Natural
            | ( complex )

complex ::=  expression
           | expression ; complex
```