Recursive Descent Parser

Design and implement a recursive descent parser for Pascal Junior. The parser should use your existing lexical analyzer to get the next token, and your existing expression parser to recognize expressions. Thus, in the recursive descent parser, "expression" should be regarded as a non-terminal symbol with a predefined procedure, express, that recognizes it.

The calling sequence is:

```
Recursive Descent Parser
      ↓
      ↓
Express
      ↓
Lexical Analyzer
```

Document your parser with both grammars. Use a format similar to that given in class (or in the text). You may use "{ ... }" to denote repetition, and "[ ... ]" to denote optional parts. Your test files, rd*.inp, will be placed on the class web site approximately 1 week before the assignment is due. The compiler directive to turn on/off the printing of flushed tokens for the Recursive Descent parser will be "r+/-".

So that I can be assured that your recursive descent parser is "recursing" as it should, each time a procedure representing a non-terminal is entered and exited you will print out a message stating that the specific procedure has been entered (or exited). Furthermore, indentation of the messages must reflect levels of call nesting. That is, every time you enter a called procedure, indent your messages 3 more spaces, when you exit a procedure, un-indent. Entry and exit messages for each call of a procedure will then be spaced the same. Use the compiler option "i+/-" to turn this Tracing on and off.