Sean Poston

CS390 HW1

1. The <*expr*> statement of <*expr*> *'-'* <*expr*> could cause some ambiguities.
   Take for example the implementation of this language as A[1] := 5 – 3 – 1.
   When done properly (from left to right) this would yield the result of 1. This would be the parser creating the two expressions of ((5- 3) – 1), to put it in easy terms. However, we could also get the number 3 if it's calculated as (5 – (3 – 1)).

   To correct the grammar, I believe simply changing it to:
   <expr> { '-' <expr>}
   would easily remedy the situation.

4. I changed my while loop to be:

   <while> ::= *'while' <bool-expr> <stmnt-list> 'end'*

I found that this would keep it "backwards compatible" for the "rich body of existing code," but would allow for more functionality within the loop.

I thought of it as changing a Java function that required an Integer to a general function that could accept multiple types; you wouldn't have to change the main function arguments that call it, but it could be used for more applications.

5. I chose to add a whole new type of <*bool-lit*> to be able to just call things true/false. This way, now we can assign an array with integers and Booleans, and it will also carry over to the Boolean operations where they can be used in something like (*while* A[1] = true *output* A[2]). This is functionally not much different from the 0/false and every other integer/true setup before, but it will be more readable.

6. I don't know what would've been missed by using EBNF. It truly seems that at this basic level, there's not much that can't be implemented using EBNF over BNF.