In the previous assignment, you implemented a lexical analyzer, or scanner, to tokenize program statements. Your scanner accepted statements and returned a string of tokens corresponding to the statement. For example, your scanner accepts the statement ***num1 = num2/4 + num3/21*** and returns the following token string: ***identifier*****:*num1*** ***assignment identifier*****:*num2*** ***operator*****:*/*** ***number*****:*4*** ***operator*****:*+*** ***identifier*****:*num3*** ***operator*****:*/*** ***number***:21.

In this assignment, you will be writing a **recognizer**, i.e. a computer program that recognizes token lists that can be generated by a given grammar, and therefore constitute a valid statement in the language defined by that grammar. Thus, your program accepts a string of tokens and categorizes it as either a valid statement, or an invalid statement. The grammar that you will be using for this purpose is shown below.

$statement \rightarrow assign\_stmt \,|\, expression$
$assign\_stmt \rightarrow identifier\ assignment\ expression$
$expression \rightarrow (identifier \,|\, number)\ operator\ expression \,|\, (identifier \,|\, number)$

To perform this task, your program implements the following algorithm.

1. Takes the token string and removes the details, like identifier name etc. (highlighted in magenta above), retaining only the tokens
2. Assume that **t1,t2,t3,t4,...tN** represents the string of tokens being processed.
3. Starting from the right end of the string of tokens, your program attempts to find the longest token substring that matches the **right hand side** of some production in the grammar.
   (a) If it finds a match, it replaces that substring of tokens with the **left hand side** of the corresponding production.
   (b) If no match is found, it attempts to the do the same with the next smallest substring starting from the right end.
4. This process is repeated till the token string has been reduced to the non-terminal ***statement***, or no further reduction is possible. The first case corresponds to a valid statement, and the second one corresponds to an invalid statement.

**Example**: **a = b** tokenizes as ***identifier assignment identifier***

1. Check to see if the RHS of any rule matches ***identifier assignment identifier.*** It does not.
2. Forget the left most token ***identifier***, and check to see if the RHS of any rule matches ***assignment identifier.*** It does not.
3. Forget the left most token, ***assignment***, and check to see if the RHS of any rule matches ***identifier.*** Rule 3, **expression -> identifier** matches.
4. Replace **identifier** (RHS) with **expression** (LHS), and repeat the process for ***identifier assignment expression***
5. RHS of rule 2 matches ***identifier assignment expression***
6. Replace with LHS, ***assign_stmt***, and repeat the process.
7. RHS of rule 1 matches ***assign_stmt***
8. Replace with LHS, **statement.** Success!

Test your program on each statement in the test file provided. **When executed, your program must read each statement in the file, and tokenize it using the *tokenize* function you wrote for assignment 1. If the tokenization has no error,** the string of tokens is forwarded to the **recognizer which classifies each statement as either a valid, or an invalid, statement. Submit both your program and the output of your program for each statement in the file.**