Lexical elements:	The Jack language includes five types of terminal elements (tokens):
keyword:	'**class**' | '**constructor**' | '**function**' | '**method**' | '**field**' | '**static**' | '**var**' | '**int**' | '**char**' | '**boolean**' | '**void**' | '**true**' | '**false**' | '**null**' | '**this**' | '**let**' | '**do**' | '**if**' | '**else**' | '**while**' | '**return**'
symbol:	'**{**' | '**}**' | '**(**' | '**)**' | '**[**' | '**]**' | '**.**' | '**,**' | '**;**' | '**+**' | '**-**' | '*****' | '**/**' | '**&**' | '**|**' | '**<**' | '**>**' | '**=**' | '~'
integerConstant:	A decimal number in the range 0 .. 32767.
StringConstant	'**"**' A sequence of Unicode characters not including double quote or newline '**"**'
identifier:	A sequence of letters, digits, and underscore ('_') not starting with a digit.
Program structure:	A Jack program is a collection of classes, each appearing in a separate file. The compilation unit is a class. A class is a sequence of tokens structured according to the following context free syntax:
class:	'**class**' className '**{**' classVarDec* subroutineDec* '**}**'
classVarDec:	('**static**' | '**field**') type varName ('**,**' varName)* '**;**'
type:	'**int**' | '**char**' | '**boolean**' | className
subroutineDec:	('**constructor**' | '**function**' | '**method**') ('**void**' | type) subroutineName '**(**' parameterList '**)**' subroutineBody
parameterList:	((type varName) ('**,**' type varName)*)?
subroutineBody:	'**{**' varDec* statements '**}**'
varDec:	'**var**' type varName ('**,**' varName)* '**;**'
className:	identifier
subroutineName:	identifier
varName:	identifier
Statements:	
statements:	statement*
statement:	letStatement | ifStatement | whileStatement | doStatement | returnStatement
letStatement:	'**let**' varName ('**[**' expression '**]**')? '**=**' expression '**;**'
ifStatement:	'**if**' '**(**' expression '**)**' '**{**' statements '**}**' ('**else**' '**{**' statements '**}**')?
whileStatement:	'**while**' '**(**' expression '**)**' '**{**' statements '**}**'
doStatement:	'**do**' subroutineCall '**;**'
ReturnStatement	'**return**' expression? '**;**'
Expressions:	
expression:	term (op term)*
term:	integerConstant | stringConstant | keywordConstant | varName | varName '**[**' expression '**]**' | subroutineCall | '**(**' expression '**)**' | unaryOp term
subroutineCall:	subroutineName '**(**' expressionList '**)**' | (className | varName) '**.**' subroutineName '**(**' expressionList '**)**'
expressionList:	(expression ('**,**' expression)*)?
op:	'**+**' | '**-**' | '*****' | '**/**' | '**&**' | '**|**' | '**<**' | '**>**' | '**=**'
unaryOp:	'**-**' | '**~**'
KeywordConstant:	'**true**' | '**false**' | '**null**' | '**this**'