Table 1: Recursively-parsable LL(1) grammar for arithmetic expressions

				Predict
(1)	$<E>$	$\rightarrow$	$<T><T_T>$	(, +, -, `var`, `num`, `func`
(2)	$<T_T>$	$\rightarrow$	$+<T><T_T>$	+
(3)	$<T_T>$	$\rightarrow$	$-<T><T_T>$	−
(4)	$<T_T>$	$\rightarrow$	ϵ	), EOF
(5)	$<T>$	$\rightarrow$	$<F><F_T>$	(, +, -, `var`, `num`, `func`
(6)	$<F>$	$\rightarrow$	$<N>$	+, -, `num`
(7)	$<F>$	$\rightarrow$	$(<E>)$	(
(8)	$<F>$	$\rightarrow$	`var`	`var`
(9)	$<F>$	$\rightarrow$	`func`$<F>$	`func`
(10)	$<F_T>$	$\rightarrow$	$*<F><F_T>$	*
(11)	$<F_T>$	$\rightarrow$	$/<F><F_T>$	/
(12)	$<F_T>$	$\rightarrow$	$\hat{}<F><F_T>$	^
(13)	$<F_T>$	$\rightarrow$	`func` $<F><F_T>$	`func`
(13)	$<F_T>$	$\rightarrow$	ϵ	+, -,), EOF
(14)	$<N>$	$\rightarrow$	`num`	`num`
(15)	$<N>$	$\rightarrow$	+`num`	+
(16)	$<N>$	$\rightarrow$	−`num`	-

- `num` denotes anything returned as a number from the scanner, e.g. 10.203, .105, 11., 212, 10.1E-15
- `func` includes any of $\sin, \cos, \tan, \cot, \csc, \sec, \lg, \ln, \log, \exp$
- When applying rule (13), implicitly insert a multiplication operator * before `func`.