LaRK

INTRODUCTION

Need of programming languages :

As the technology is evolved we will be facing more problems. Existing languages may be out of context for that problem. For example, to have complete fault tolerant system for telecom networks Ericsson company has spent 5 years researching for a language which is suitable for telecom networks and ended up creating a new programming language called Erlang, but now not just Telecom companies but like Google, Facebook, whatsApp and Github uses Erlang. And for data science we have R language and Julia. And now recently Mozilla has released a language called Rust to be as fast as C and make programming less sucked. Always there is a requirement for new inventions and that's why people come up with new languages.

ABSTRACT

This language is named as LaRK and is proposed by,

RITESH KUMAR (2015UCP1727) & ***PAWAN KUMAR*** (2015UCP1622)

Supported feature :

- **Data types :** int, float, char.
- **Operators :** Paranthesis[( )], Postfix increment decrement [++/--], Prefix increment decrement [++/--], Exponent [^], Multiplication [*], Division [/], Modulus [%], Addition [+], Subtraction [-], Operational assignment [^=, *=, /=, %=, +=, -=].
- **Operations :** Assignment, Algebraic operation.
- **Precedence Table :**

Precedence	***Operators***	***Associativity***
01.	( ), ++/-- (Postfix)	**L** to **R**
02.	++/-- (Prefix)	**R** to **L**
03.	^	**R** to **L**
04.	*, /, %	**L** to **R**
05.	+, -	**L** to **R**
06.	^=, *=, /=, %=, +=, -=	**R** to **L**

GRAMMAR

```
P  : : = Begin <return_type> kick_start ( ) { <stmt_list> return <int>;} End


<stmt_list>      : : =  <stmt>; | <stmt> ;  <stmt_list>
<stmt>           : : =  <varb> = <expr> | <varb> <op> = <expr> | <declaration>|<primary>


<declaration>    : : =  <key> <varb_list> | <initial2>
<varb_list>      : : =  <varb> , <varb_list> | <varb>


<initial2>       : : =  <key> <initial1>
<initial1>       : : =  <initial> , <initial1> | <initial>
<initial>        : : = <varb> = <initial> | <varb> = <const>


<expr>            : : = <term> <op1> <expr> | <term>
<term>            : : = <factor> <op2> <term> | <factor>
<factor>          : : = <primary> ^ <factor> | <primary> | <expr> | (<factor>)
<primary>         : : = ++ <data> | -- <data> | <data>
<data>            : : =  <type> ++ | <type> -- | <type>
<type>            : : =  (<type>) | <s_int> | <s_float> | <varb>


<s_int>          : : =  <op1> <int> | <int>
<int>            : : =  <digit> <int> | <digit>


<s_float>        : : = <op1> <float> | <float>
<float>          : : =  <int> . <int>


<varb>           : : =  <alpha> | <alpha> <alpha_num>
```

```
<alpha_num>   : : = <alpha> | <digit>

<const>       : : = <s_float> | <s_int>

<alpha>        : :  =

              a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|_|
              A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

<digit>       : : = 0|1|2|3|4|5|6|7|8|9

<key>         : : =  int | float | char

<return_type>  : : = int | void

<op>          : : = <op1> | <op2> | ^

<op1>         : : =  + | -

<op2>         : : = * | / | %
```