Topic: Compiler for R in C using Bottom-up Approach (LALR) with Constructs: FOR, WHILE, IF and IF-ELSE statements

Team Members:

Kirthika Gurumurthy	PES1201700230	
Richa	PES1201700688	**6'H'**
Akanksha	PES1201701799	

Grammar:

Data types:

<identifier> → < “.” ><letter><identifier> | (<letter> | <digit>| “_” | “.”) <identifier>) | λ
<numeric> → <integer> | <double>
<list> → <string_list> | <double_list> | <integer_list>
<integer> → (“+” | “-”) <digit> ‘L’
<double> → (“+” | “-”) <digit> “.” <digit> | (“+” | “-”) <digit> “.”
<string> → “ ‘ “ ([^\\] | <escapesequence>) <string> “ ‘ “ | ‘ “ ‘ ([^\\] |
<escapesequence>) <string>) ‘ “ ‘ | λ
<escapesequence> → [\\](.)
<string_list> → “c(” <string_item> “)”
<string_item> → <string> | <string_item> “,” <string_item> | <double_item> | <list>
<integer_list> → “c(” <integer_item> “)”
<integer_item> → <integer> | <integer_item> “,” <integer_item>
<double_list> → “c(” <double_item> “)”
<double_item> → <double> | <double_item> “,” <double_item> | <integer_item>
<letter> → <lower> | <upper>
<lower> → “a” | “b” | “c” | “d” |.....| “y” | “z”
<upper> → “A” | “B” | “C” | “D” |.....| “Y” | “Z”
<digit> → [0-9] <digit> | [0-9]

Constructs:

<constructs> → <for_loop> | <while_loop> | <if_statement>
<for_loop> → “for(“<identifier> “in” <iterable> “) {“ <statements> “}”
<while_loop> → “while” <condition> “{” <statements> “}”
<if_statement> → “if” <condition> “{“ <statements> “}”

<iterable> → <range> | <list>
<range> → <integer> “:” <integer> | “seq(“<integer> “,” <integer> ”)” | “seq(“<integer> “,”
<integer> “,” <numeric> ”)”
<condition> → <and_cond> | <or_cond>
<statements> → <assign_expr> <statements> | <print_stat> <statements> |
<arithmetic_expr> <statements> | λ

<comp_ops> → "<" | ">" | "==" | ">=" | "<=" | "!="
<u_op> → “+=” | “-=” | “*=” | “*=” | “/=”
<and_cond> → <expr> <comp_ops> <expr> | (“&&” (<and_cond>|<or_cond>))

<or_cond> → <expr> <comp_ops> <expr> | (“||” (<and_cond>|<or_cond>))
<expr> → <identifier> | <character> | <numeric> |<list>
<assign_ops> → “=” | “<-”
<assign_expr> → <identifier> (<assign_ops> | <u_op>) <expr>
<print_stat> → “print(” <identifier> “)”
<import_stmt> → “import ::from(“ <library> “,” <func> “)” | <func> “<-” <library> “::” <func>
<arithmetic_ops1> → “+” | “-”
<arithmetic_ops2> → “*” | “/”
<arithmetic_expr> → <id2> | <id1> <arithmetic_ops1> <id2>
<id2> → <id3> | <id2> <arithmetic_ops2> <id3>
<id3> → <numeric> | <identifier>
<library> → “dplyr” | “ggplot2” | “knitr” | “lubridate” | ...
<func> → <<< function of <library> >>> | <func> “,” <func>

Tokens:

1. **Keyword**
   <keyword, lexeme, line#>

2. **Identifier**
   <id, lexeme, value, type, line#>

3. **Operators**
   <op, lexeme, type = [<comp_ops>, <u_op>, <arithmetic>, <assign_ops>, <arithmetic_ops1>, <arithmetic_ops2>], line#>

4. **Punctuators**
   <punc, lexeme, type = [COLON, ESCAPE CHARACTERS, PARENTHESES, QUOTES], line#>

5. **Literal**
   <literal, lexeme, type, line#>