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

Team Members:

Kirthika Gurumurthy Richa Akanksha PES1201700230 PES1201700688 PES1201701799

6'H'

Grammar:

```
Data types:
```

```
<identifier> \rightarrow < "." ><letter><identifier> | ( <letter> | <digit>| " " | "." ) <identifier>) | \lambda
<numeric> → <integer> | <double>
< → < string list> | < double list> | < integer list>
<integer> \rightarrow ("+" | "-" ) <digit> 'L'
<double> → ( "+" | "-" ) <digit> "." <digit> | ( "+" | "-" ) <digit> "."
<string> \rightarrow " ' " ( [^\\] | <escapesequence>) <string> " ' " | ' " ' ( [^\\] |
<escapesequence>) <string> ) ' " ' | λ
\langle escapesequence \rangle \rightarrow [\](.)
<string list> → "c(" <string item> ")"
<string_item> → <string> | <string_item> "," <string_item> | <double_item> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 
<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> \rightarrow "a" | "b" | "c" | "d" |.....| "y" | "z"
upper> → "A" | "B" | "C" | "D" |.....| "Y" | "Z"
\langle digit \rangle \rightarrow [0-9] \langle digit \rangle | [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> | terable>
<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> \rightarrow "<" | ">" | "==" | ">=" | "<=" | "!="</pre>
<u op> \rightarrow "+=" | "-=" | "*=" | "*=" | "/="
```

<and cond $> \rightarrow <$ expr> <comp ops> <expr> | ("&&" (<and cond> | <or cond>))

```
 \begin{aligned} &< \text{or\_cond>} \rightarrow &< \text{expr>} < \text{comp\_ops>} < \text{expr>} \mid (\text{``||''} (\text{<and\_cond>|<or\_cond>)}) \\ &< \text{expr>} \rightarrow &< \text{identifier>} \mid &< \text{character>} \mid &< \text{numeric>} \mid &< \text{list>} \\ &< \text{assign\_ops>} \rightarrow &= \text{``|''<-''} \\ &< \text{assign\_expr>} \rightarrow &< \text{identifier>} (\text{``cassign\_ops>} \mid &< \text{u\_op>}) &< \text{expr>} \\ &< \text{print\_stat>} \rightarrow & \text{``print(''} &< \text{identifier>} &'')'' \\ &< \text{import\_stmt>} \rightarrow & \text{``import} :: \text{``from(''} &< \text{library>} &'', \text{``func>} &'')'' \mid &< \text{func>} \\ &< \text{arithmetic\_ops1>} \rightarrow &+ \text{``+''} \mid &- \text{``-} \\ &< \text{arithmetic\_ops2>} \rightarrow &+ \text{``*i'} \mid &'' \mid &< \text{arithmetic\_ops1>} &< \text{id2>} \\ &< \text{id2>} \rightarrow &< \text{id3>} \mid &< \text{id2>} &< \text{arithmetic\_ops2>} &< \text{id3>} \\ &< \text{id3>} \rightarrow &< \text{numeric>} \mid &< \text{identifier>} \\ &< \text{library>} \rightarrow &+ \text{``dplyr''} \mid & \text{``ggplot2''} \mid & \text{``knitr''} \mid & \text{``lubridate''} \mid \dots \\ &< \text{``func>} \rightarrow &<< \text{``function of } &< \text{``j''} &< \text{``func>} \end{aligned}
```

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#>