

**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> → < "." > <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#>`