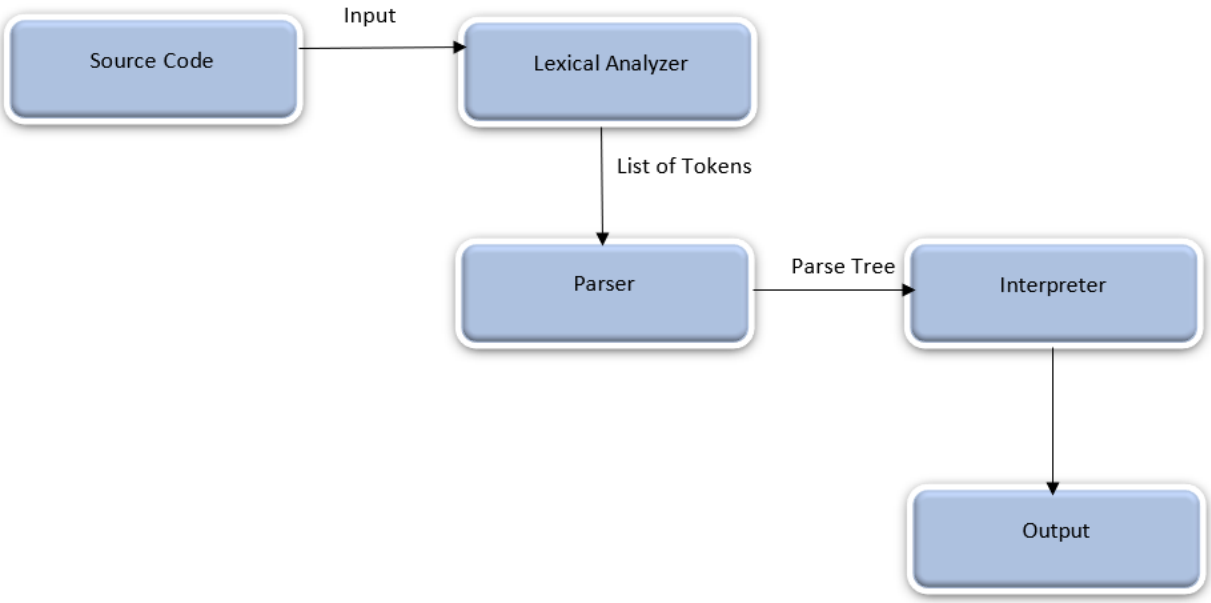
SER502-Spring2023-Team22

#MileStone-1
Team: Chedvihas Punnam, Gnanadeep Settykara, Teja Lam, Naga Vamsi Bhargava Reddy Seelam, Vikranth Reddy Tripuram

Language Name: Rev
Language-Extension: .rev

Design-Flow:



Source Code: The source-code file for the Rev programming language has .rev extension to it and it contains the actual program that would be executed.

Lexical Analyzer: The lexical analyzer or lexer is a vital part of the compilation process, responsible for recognising and breaking down the source-code into meaningful tokens. The tokens are produced as per the grammar rules set for the Rev language. Prolog is used to generate the list of tokens and are stored using list data-structure.

Parser: The Parser receives the token list after it has been produced. The parser examines each token one at a time according to the grammar rules and uses Prolog to construct a Parse tree. This stage is performed once all tokens have been parsed. The grammatical predicates are prepared in accordance with the DCG. (Definite clause grammar). Top-down parsing technique is used and the parse tree that is formed is then transmitted to the interpreter.

Interpreter: The interpreter examines the parse tree by traversing each node in the parse tree. Each node in the list would be analyzed using the semantic rules & modified accordingly. Prolog would be used to implement the interpreter.

Data-Structure: List, Design-language: Prolog, Grammar: DCG,

Data Types: Rev language supports 3 primitive data types. They are:

1. int : permits all integer numbers
2. varchar: allows string-value assignment
3. bool: allows true and false values

Declaration & Assignment: Various supported data-types can be used to initialize variables by following the syntax. Assignment-operator can be used for assigning values to variables. Examples:

1. int g; (here, variable ‘g’ of “int” type is declared)
2. Int t=100; (here variable ‘t’ is assigned with an integer value of “100”)

Operators:

1. The Rev language uses arithmetic operators ‘+’, ‘-’, ‘*’, & ‘/’ to express addition, subtraction, multiplication, & division.
2. In the Rev language, Boolean operators 'and,' 'or,' and 'not' are used to validate loops and conditional expressions.
3. In the Rev language, comparison operations like greater than (>), greater than or equal to (>=), equals to (==), less than (<) , less than or equal to (<=) , not equals (!=) can be used.

Ternary operator: Ternary operator (?:) is supported by the rev language, which is equivalent to if-then else operation like for example (x==1)?(y=7):(z=3) , here if x==1 is true, then y=7 is executed, else z=3

Conditional stmnts:

1. if-then: A classic 'if' loop is used to check for a statement, and if it is true, the 'then' block is run.

2. if-then-else: When the 'if' condition is met, the 'then' statement are performed. Otherwise, 'otherwise' expressions are carried out.

Loops:

1. for: for statement works as classic for loop with 3 conditional parameters namely initialization, end condition and iterative condition.

syntax: for(<Initialization> ; <End condition> ; <Iterative statement>)

Eg: for(i=0;i<=100;i++)

```
{
    //statements
}
```

2. foreach: foreach acts as looping statement for lists or in a range

syntax:foreach(int i in list)

```
{
    // statements
}
```

In the above syntax, list is a integer list

3.while: in while we give a condition and if it is true, all code inside it will execute.

Syntax: while(<condition>)

```
{
    // statements
}
```

Print Statement: print is used to display the statements on the console that are given inside it.

Syntax:

print “<statements>”

Comments: In Rev programming language, we use ! symbol to denote the statements as comments which will not be executed.
Syntax: !<statements>

Rev Language Grammar:

prg is program.
blk is block.
stmnt is statement.

INIT → 'begin'.

EXIT → 'end'.

COMMENT → '!'

INT → /^[0-9]+$/

FLOAT→ /([0-9]*[.])?[0-9]+/

BOOL → /'True' | 'False'/

VARCHAR → /\"[\x00-\x7F]*\"/

CHAR→ /'[\x00-\x7F]'/

DATATYPE → 'int' | 'varchar'| 'bool' | 'float'

IDNTFR → /^[a-zA-Z_$][a-zA-Z_$0-9]*$/.

ADD → '+'

SUB → '-'

AND → 'and'

NOT → 'not'

OR → 'or'

DIV → '/'

MUL → '*'

ASSGN → '='

PRINT → 'print'

IF → 'if'

ELSE → 'else'

ELIF → ‘elseif’

CMP → ‘<’ | ‘<=’ | ‘>’ | ‘>=’ | ‘!=’ |‘==’

CNDOPR → and | or | not

IN → ‘in’

WHILE → ‘while’

RNG → ‘range’

FOR → ‘for’

OPENBRC → ‘{‘

CLOSEBRC → ‘}’

OPENPRNTHS → ‘(‘

CLOSEPRNTHS → ‘)’

DLMTR → ‘;’

STRTQT → ‘”’

ENDQT → ‘”’

COMMA → ‘,’

prg ::= INIT stmnt DLMTR END.

stmnt ::= stmntop DLMTR stmnt | stmnt

stmntop ::= declare | assign | if-else | print | for | while

declare ::= DATATYPE IDNTFR | DATATYPE IDNTFR ASSGN data

data ::= FLOAT | CHAR | BOOL | VARCHAR| INT

comment ::= COMMENT VARCHAR

assign ::= IDNTFR ASSGN expressn

expressn ::= trm | trm ADD expressn | trm SUB expressn
trm ::= fctr | fctr MUL trm | fctr DIV trm
fctr ::= data | IDNTFR | OPENPRNTHS expressn CLOSEPRNTHS

if-else ::= IF OPENPRNTHS condition CLOSEPRNTHS OPENBRC stmnt DLMTR CLOSEBRC | IF OPENPRNTHS condition CLOSEPRNTHS OPENBRC stmnt DLMTR CLOSEBRC DLMTR ELSE OPENPRNTHS stmnt DLMTR CLOSEPRNTHS | IF OPENPRNTHS condition CLOSEPRNTHS OPENBRC stmnt DLMTR CLOSEBRC DLMTR, elif DLMTR ELSE OPENBRC stmnt DLMTR CLOSEBRC

elif ::= elif1 | elif DLMTR elif1
elif1 ::= ELSEIF OPENPRNTHS condition CLOSEPRNTHS OPENBRC stmnt DLMTR CLOSEBRC

print ::= PRINT IDNTFR | PRINT STRTQT STRING ENDQT

while ::= WHILE OPENPRNTHS condition CLOSEPRNTHS OPENBRC stmnt CLOSEBRC

for ::= FOR forRng OPENBRC stmnt CLOSEBRC
forRng ::= IDNTFR IN RANGE OPENPRNTHS expressn COMMA expressn CLOSEPRNTHS | OPENPRNTHS IDNTFR ASSGN expressn DLMTR IDNTFR CMP expressn DLMTR CLOSEPRNTHS | OPENPRNTHS IDNTFR ASSGN expressn DLMTR IDNTFR CMP expressn DLMTR expressn CLOSEPRNTHS

condition ::= BOOL | IDNTFR CMP expressn | IDNTFR CMP expressn CNDOPR condition