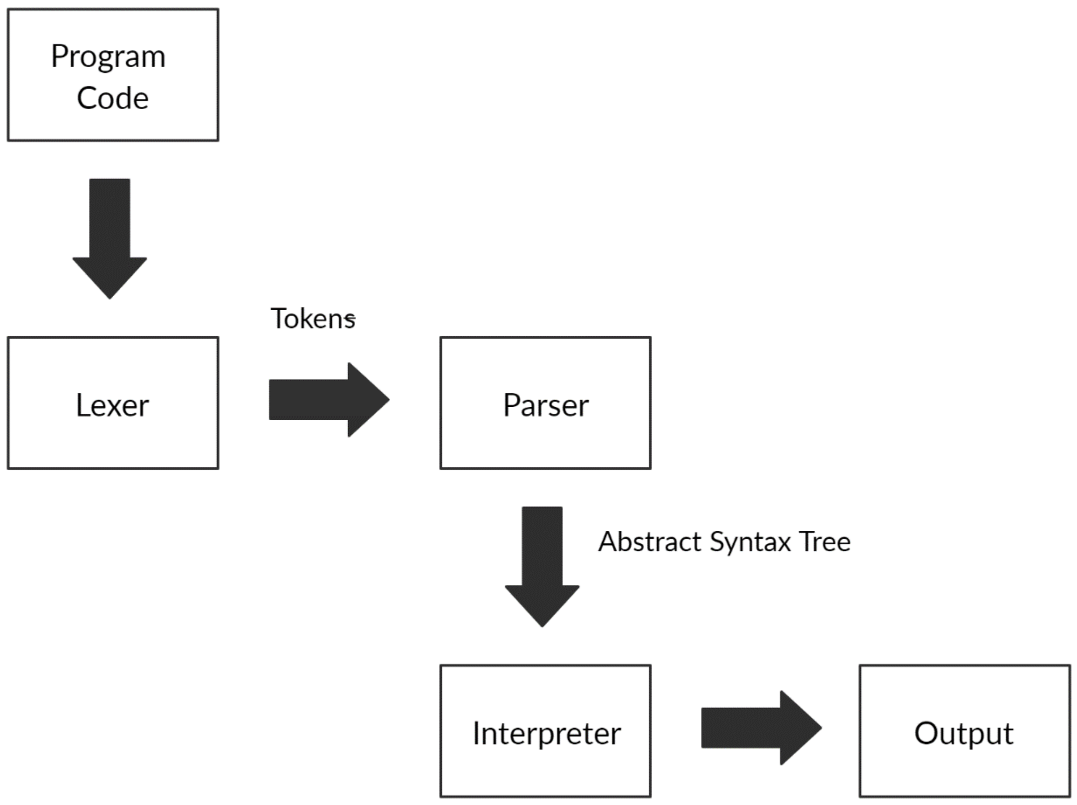
SER 502 Project: Compiler and Virtual Machine for a Programming Language

Team 11 – Milestone I:

GitHub Repository URL: https://github.com/AmudhanManisekaran/SER502-Spring2020-Team11

Language Name: CodeZilla

Language Design :



We have decided to implement the lexical analyzer in Python, Parser, and Interpreter using Definite Clause Grammar (DCG) in Prolog, a logic programming language.

Lexer (Lexical Analyser): Python program will be written to read the user's code. This code will be converted to a set of tokens, which will be stored in a token file.

Parser: The tokens from the Lexer will be passed through the Parser containing Prolog rules. These rules will generate the equivalent Abstract syntax trees. These syntax trees will act as the intermediate code.

Interpreter: The interpreter will evaluate the parse tree obtained from the previous step using Syntax-directed semantics. At each node in the parse tree, the environment will be updated.

The Parser and Interpreter rules will be written in **SWI-Prolog**.

The **data types** supported by CodeZilla are:

- Integer
- Float
- Double
- String
- Boolean

The **operators** supported by CodeZilla are:

- Addition
- Subtraction
- Multiplication
- Division
- Comparison (<, >, <=, >=, =:=)
- AND, OR, NOT for Boolean expressions.
- Ternary ($)
- Assignment (:=)

The **conditionals** supported by CodeZilla are:

- Standard ‘if’
- If then else

The **loops** supported by CodeZilla are:

- While
- For

The **terminators** and **delimiters** used are:

- Start
- End
- Semi-colon (;)
- Whitespace (//)

CodeZilla allows the user to print the numbers, boolean values and strings using **show** keyword. User input can be obtained by using the **read** keyword.

Grammar Rules:

PROGRAM ::= SPACE Start; BLOCK SPACE End;

BLOCK ::= DECLARELIST; CONDITIONLIST;

DECLARELIST ::= DECLARE; DECLARELIST | DECLARE

DECLARE ::= num SPACE ID | var SPACE ID | str SPACE ID

CONDITIONLIST ::= CONDITION; CONDITIONLIST | CONDITION

CONDITION ::= ASSIGN | IF | TERNARY | LOOPS | SHOW SPACE DATA SPACE| READ SPACE ID | BLOCK

Conditional Operations:

ASSIGN ::= ID := EXP | ID := STRING

IF ::= if SPACE BOOL SPACE then SPACE CONDITION SPACE endif | if SPACE BOOL SPACE then SPACE CONDITION SPACE else SPACE CONDITION SPACE endif

TERNARY ::= ID := BOOL SPACE $ SPACE EXP / EXP

LOOPS ::= while SPACE BOOL SPACE do SPACE CONDITION SPACE endwhile | for SPACE (ID := VALUE ; BOOL_CONDITION ; EXP)| for SPACE ID SPACE inrange(VALUE, VALUE) SPACE do SPACE CONDITION

Boolean Expressions:

BOOL ::= true | false | BOOL SPACE and SPACE BOOL | BOOL SPACE or SPACE BOOL | not SPACE BOOL | BOOL_CONDITION

BOOL_CONDITION ::= EXP SPACE COMPARE SPACE EXP

COMPARE ::= < | > | <= | >= | =:=

Arithmetic Expressions:

```
EXP        ::= EXP + TERM | EXP – TERM | TERM
TERM       ::= TERM / FACTOR | TERM * FACTOR | FACTOR
FACTOR     ::= VALUE | (EXP)
VALUE      ::= ID | NUMBER
```

Identifiers:

```
ID         ::= ^[a-z]+ [A-Z]? [a-z]* [0-9]*$
```

Data types:

```
NUMBER     ::= INT | FLOAT | DOUBLE
INT        ::= ^[0-9]+$
FLOAT      ::= ^[0-9]+ \ . [0-9]{1,7} $
DOUBLE     ::= ^[0-9]+ \ . [0-9]{1,15}$
STRING     ::=  << (.)*? >>
```

Keywords:

```
DATA             ::= STRING | ID
SPACE            ::= //
```

Sample Program:

```
Start;

num variableOne;

num variable2;

num loopcount;

str statement1;

loopcount:= 0;

statement1 := <<hello world>>;

read variableOne;

if variableOne<10 then variable2:=15; endif;

show variable2;

variable3 := variableOne + ( variable2 / 5);

show variable3;

for (x:=0 ; x<=6 ; x=x+2) do show statement1;

End;
```

OUTPUT:

```
Enter variableOne: 5

15 8 hello world hello world hello world
```