SER502 Project - Milestone 1

Team 4

Dev Jani (djani4)

Kalyani Joshi (kjoshi29)

Kirtan Soni (kgsoni)

Lakshmi Kruthi Hosamane Keshava Raman (lhosaman)

Submitted to : Professor Ajay Bansal

SER502 Emerging Languages and Programming Paradigms

1 Project Description - About Our Language

Our team is working on developing a programming language named ”DEVLang”. The goal of our project is to create a programming language whose syntax derives of existing programming languages like Java and C. The GitHub repository to our project could be found at https://github.com/KirtanSoni/SER502-DEVLang-Team4

2 Design

This section discusses the different statements which our programming language DEVLang includes using which our Grammar was created.

- Just like other languages would have ”begin - end” structure, our language has the ”dev - lang” which serves the same purpose. Next, there are statements which could consist of conditional statements, loops, assignment, print and null statements.
- Our language has loops named ”loop”, ”loopwith” which corresponds to the traditional for-loop and while-loop respectively. Lastly, we create a loop named ”looprange” which functions just like the statement ”for i in range(2,5)” would.
- Additionally, we have single line statements which perform the same function as an if-statement, print statement, declaration and assigning of variable.
- DEVLang has dataypes, integer, character array (charr) and boolean. Along with that DEVLang supports:-
  - Arithmetic operators like addition (’+’), subtraction (’-’), multiplication (’*’) and division (’́).
  - Relational operators like less than (’¡’), greater than (’¿’), equal to (’==’).
  - Logic operators like

  Lastly, DEVLang has the provision of using ternary operators as well.
- The rules of the parsing technique of our programming language are written in DCG.

3 Grammar

```
procedure --> ['dev'], statement_pipeline, ['lang'].


statement_pipeline --> statement, comma, statement_pipeline | statement.
```

```
block --> ['{'], statement_pipeline, ['}'].

statement --> conditional_statement| loops| assignment_statement| print_statements| null_statements.

conditional_statement --> if_statement, block, ['otherwise'], block.
conditional_statement --> ['?'], conditional_statement, [':'],
statement_pipeline, [':'], statement_pipeline.

if_statement --> ['if'], ['('], conditional_logic, [')'].

conditional_logic --> integer_comparison | logical_comparison | bool.

integer_comparison --> ['integer'], int, comparison_operator, ['integer'], int.
comparison_operator --> ['>'] | ['<'] | ['=='].

logical_comparison --> ['and'], ['('], conditional_logic, comma, conditional_logic, [')'].
logical_comparison --> ['or'], ['('], conditional_logic, comma, conditional_logic, [')'].
logical_comparison --> ['not'], ['('], conditional_logic, comma.

loops --> loop_part, block | loopwith_part, block | looprange_part, block.

loop_part --> ['loop'], ['('], conditional_logic, [')'].
loopwith_part --> ['loopwith'], ['('], assignment_statement, [':'], conditional_logic, [')'].
looprange_part --> ['looprange'], ['('], assignment_statement, ['integer'], int,
['integer'], int, [')'].

assignment_statement --> variable, assignment_operator, literal | variable, assignment_operator,
expression.

variable --> ['let'], var| literal| expression.

print_statements --> ['tout'], ['('], data_type, [')']| ['tout'], ['('], literal, [')'].
```

```
null_statements --> [' '].


literal --> data_type.

bool --> ['bool(true)'] | ['bool(false)'].

int --> ['int('], numbers , [')'] | expression.

var --> character.

data_type --> bool | int | charr | variable.

charr --> ['charr('], character, [')'].


character--> lowercase_char | uppercase_char| numbers| character.

expression_part --> ['integer'], int | variable.


expression --> ['['], expression_part, operator, expression_part, [']'].


operator --> ['+'] | ['-'] | ['*'] | ['/'].


assignment_operator --> ['='].

comma --> [','].


lowercase_char --> ['a'] | ['b'] | ['c'] | ['d'] | ['e'] | ['f'] | ['g'] | ['h'] | ['i'] | ['j'] | ['k'

uppercase_char --> ['A'] | ['B'] | ['C'] | ['D'] | ['E'] | ['F'] | ['G'] | ['H'] | ['I'] | ['J'] | ['K'

numbers --> ['0'] | ['1'] | ['2'] | ['3'] | ['4'] | ['5'] | ['6'] | ['7'] | ['8'] | ['9'].
```

4 Data Structures

During the parsing process, the parser will generate an Abstract Syntax Tree (AST) which would depict the syntactic structure of our programming language, DEVLang. This AST will comprise of nodes corresponding to various language constructs like statements, declarations etc.