SER 502 Project Milestone - 1
Compiler and Interpreter for a Programming language

Team :
Jigisha Deven Gadhia - 1221069187
Sandhya Tadi - 1219346947
Venkata Naga Sonia Kalidindi- 1219398622
Akhila Sai Mandava - 1220311417

Name of the Programming Language: JASS
Github Link: https://github.com/akhilamandava/SER502-Spring2021-Team21

Introduction: The project includes the design of a lexical analyzer, parser and interpreter. This document is a clear description of the idea along with the grammar of the language. It gives a detailed explanation of the tools required for the project and the knowledge acquisition required for implementation. This document is a combination of language description, prerequisites and techniques used for the implementation, and functionalities of the language.

Overview: The compilation process is a sequential process that contains various phases. Each phase takes an input from its previous phase. Initially, a source code (here JASS source code) is passed through a lexical analyzer. This phase is a text scanner that generates the source code to a set of lexemes in the form of tokens. The lexer is to be implemented in Python. Now these tokens are given as an input to the next phase called Syntax Analysis. Here an abstract syntax tree is generated using the Top-down approach. The parser is implemented using Prolog. This phase also checks if the expressions are syntactically correct. We are checking this using Definite Clause Grammar. The next stage is semantic analysis, here it checks if the generated parse tree is following the rules using DCG. Finally, the interpreter executes these instructions to generate an output. The interpreter is to be implemented in Prolog.

Language Design:
This programming language supports the following :

1. Primitive Types:
    a. Numeric Data Types: int, float
    b. Boolean Data Type: bool
    c. Character Data Type: string, char
2. Operators:
    a. Arithmetic Operators: +, -, *, /, %
    b. Boolean Operators: and, or, not, ==, >>, <<, =>, =<
    c. String Operators: strlen(), strconcat()
    d. Assignment Operator: =

3. Conditional Constructs:
    a. Ternary Operator: '?:'
    b. If-then-else
    c. Else-if
4. Looping Structures:
    a. Traditional for loop ("For i in range(n1,n2)" where n1,n2 are integers, the statement means for(i=n1;i<n2;i++))
    b. Traditional while loop
5. Commands:
    a. Print Statement to display the output.
    b. Function Calls
    c. Assignment (i=String, x=10)

Languages and Techniques used:

- **Language used:**
    1. SWI- Prolog for Parser and Interpreter.
- **Parsing Technique:**
    1. Parsing is based on the Top-down approach using DCG, implemented in Prolog.
- **Data Structures:**
    1. Parse: Syntax-Tree
    2. Interpreter: Lookup Table

Grammar:

Program ::= { Block } FunctionDefinition | { Block }
Block ::= Declaration ; Command
FunctionDefinition ::= define String (Parameter) { Block Return }

Declaration ::= Declaration ; Declaration | Initialize | int Identifier | char Identifier |
string Identifier | float Identifier | bool Identifier
Command ::= Command Command | Identifier = Expression ; | Identifier = String ; | If | While |
For | Print ;|UnaryOp ; | StringFunction | FunctionCall; | Identifier = FunctionCall ;

Parameter ::= Parameter , Parameter | Declaration
Initialize ::= int Identifier = Integer | char Identifier = Character | string Identifier = String |
float Identifier = Float | bool Identifier = Boolean | Declaration | Null

FunctionCall ::= String (NewIdentifier)
NewIdentifier ::= Identifier | NewIdentifier , Identifier

StringFunction ::= StringLength
StringLength ::= strlen (Identifier) | strlen (String)

Ternary ::= (Boolean) ? { Expression } : { Expression }

If ::= if (Boolean) { Block } else { Block } | if (Boolean) { Block } Elseif
Elseif ::= elseif (Boolean) { Block } else { Block } | elseif (Boolean) { Block } Elseif | EMPTY

For ::=for (Initialize ; Boolean ; UnaryOp) { Block } | for Identifier in range (Integer , Integer) { Block }
While ::=while (Boolean) { Block }

UnaryOp ::= Identifier++ | Identifier--

Expression ::= Expression + Expression | Expression - Expression | Expression * Expression | Expression / Expression | Expression % Expression | (Expression) | Integer | Character | Float | String | Ternary | Identifier | Identifier = Expression

Boolean ::= true | false | Expression == Expression | Expression >> Expression | Expression << Expression | Expression <= Expression | Expression >= Expression | not Boolean | Expression and Expression | Expression or Expression | Expression | Boolean and Boolean | Boolean or Boolean

Identifier ::= Alphabets | Identifier Identifier
Print ::= print (Statement)
Statement ::= Statement Statement | Identifier | “ String ” | EMPTY
String ::= Character | Character String | EMPTY
Character ::= Alphabets | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ! | @ | # | $ | % | ^ | & | * | (|) | _ | + | ; | : | ” | ’ | \ | , | . | < | > | ? | / | ~ | ` | { | } | [|] | -
Alphabets ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | x | y | z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | X | Y | Z

Float ::= Digits Point Digits | Integer
Integer ::= Digits
Digits ::= Digit | Digit Digits
Digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
Point ::= .
Return ::= return Expression ; | return ;