Project Milestone - 1
Compiler and Virtual Machine for a Programming Language(MADS)

Team 12 :-
Team Members:
1] Ashwin Srinivasan
2] Deepti Paul
3] Madhu Madhavan
4] Srinivasan Sundar

Our Programming Language name: *MADS*
Github link: https://github.com/Srini2305/SER502-Spring2020-Team-12

GRAMMAR:

Program:
Program := start '{' StatementList '}' end

StatementList:
StatementList ::= Statement StatementList | Statement

Statement:
Statement ::= Declaration ';'
| Expr ';'
| PrintStatement ';'
| TernaryOperator ';'
| LoopStatement
| UnaryExpr

Declaration:
Declaration ::= DataType Identifier
| InitializeStatement

InitializeStatement ::= DataType Identifier '=' DataTypeValue
| Identifier '=' DataTypeValue

Arithmetic Expression:

```
Expr ::= Expr '+' Expr
        | Expr '-' Expr
        | Expr '*' Expr
        | Expr '/' Expr
        | Expr '%' Expr
        | (Expr)
        | Identifier
        | Digit
```

Unary Expression:

```
UnaryExpr ::= '++' Identifier | '--' Identifier
```

Conditional Statement:

```
ConditionStmt ::= ConditionStmt LogicalOperator ConditionStmt
                | '!' ConditionStmt
                | RelationalExpr
                | LogicalExpr
                | Bool
```

Relational Expression:

```
RelationalExpr ::= Expr RelationalOperator Expr
                | Identifier
                | Bool
```

Logical Expression:

```
LogicalExpr ::= RelationalExpr LogicalOperator RelationalExpr
                | RelationalExpr LogicalOperator LogicalExpr
                | '!' RelationalExpr
                | '!' LogicalExpr
```

Ternary Operator:

```
TernaryOperator ::= ConditionStmt '?' TernaryStatement ':' TernaryStatement
```

```
TernaryStatement ::= printStatement
                    | InitializeStatement
                    | BoolExpr
```

Loop Statement:
LoopStatement ::= IfLoop
| WhileLoop
| ForLoop

IfLoop ::= if ‘(‘ ConditionStmt ‘)’ ‘{‘ Statement ‘}’ else ‘{‘ Statement ‘}’

WhileLoop ::= while ‘(‘ ConditionStmt ‘)’ ‘{‘ Statement ‘}’

ForLoop ::= for Identifier in range ‘(‘ NumberValue ‘,’ NumberValue ‘)’ ‘{‘ Statement ‘}’
| for ‘(‘ InitializeStatement ‘;’ ConditionStmt ‘;’ Expr ‘)’ ‘{‘ Statement ‘}’
| for ‘(‘ InitializeStatement ‘;’ ConditionStmt ‘;’ UnaryExpr ‘)’ ‘{‘ Statement ‘}’

PrintStatement ::= print ‘(’ Identifier ‘)’
| print ‘(’ String ‘)’

DataType:
DataType ::= int | float | string | bool

DataTypeValue:
DataTypeValue ::= Integer | Float | String | Bool

Operator:
LogicalOperator ::= ‘&’ | ‘| |’
RelationalOperator ::= ‘>’ | ‘<’ | ‘>=’ | ‘<=’ | ‘==’ | ‘!=’

String ::= ‘“’ [a-zA-Z0-9]* ‘”’

Digit ::= [0-9]+
Integer ::= ‘-’ Digit | Digit
Float ::= Digit | Digit ‘.’ Digit | ‘-’ Digit | ‘-’ Digit ‘.’ Digit
Bool ::= true | false

Identifier ::= [a-z][a-zA-Z0-9]*

NumberValue ::= Identifier | Digit

Keyword ::= start | end | int | float | string | bool | for | while | if | else | true | false | print | and | or | not

MADS Language Explanation:

- *Program begins with the 'start' keyword and ends with the 'end' keyword.*
- *Block of code is represented using open '{' and close '}' curly braces*
- *Each Statement has a line terminator - Semicolon ';'*
- *Datatypes like bool, int, float, string are supported in this language.*
- *Variable names can contain lowercase, uppercase alphabet and numbers. Variable names should start with lowercase alphabet.*
- *Relational operations (> , < , >= , <= , == , !=) are supported in this language.*
- *Logical Operators like and (&), or (| |), not (!) are supported in this language.*
- *Decision statements like if-else statement and ternary operator ({condition} ? {true_statement} : {false_statement}) are supported in this language.*
- *Loop statements like while, for statements are supported in this language.*
- *For loop statement has two variants:*
    - *for i in range(0,5){ }*
    - *for (int i = 0 ; i < 5 ; i++)*
- *Basic arithmetic expression (+, - , *, /, %) are supported in this language.*
- *Print statement is used to print the value of a variable and string values.*
- *Data types don't have default values. Error is printed if a variable is used without initializing.*

Design and Tools planned to be used for the project:

- ANTLR:

"ANTLR (ANother Tool for Language Recognition) is a powerful parser generator for reading, processing, executing, or translating structured text or binary files."
Grammar is defined using .g4 file in ANTLR.
ANTLR is used for token generation, building lexer and parser.
ANTLR helps to generate the parse tree. From a grammar, ANTLR generates a parser that can build and walk parse trees.

- Data Structures:

We have planned on using Stack for looping and decision constructs, and HashMap. HashMap can act as a lookup table during runtime. In addition, according to the features to be added we are planning to use stack for functions. Thus, temporarily storing the variables created within the function after a function call. The other data structure HashMap is used for implementing key, value pair logic to store data.

Steps involved in Language Design:

- Lexical Analyzer:

We planned to use ANTLR, which reads the input and divides it into tokens based on the grammar defined by us.

- Parser:

The parser takes a stream of tokens from the Lexer as the input and generates the parse tree. ANTLR generates a parser that can build and walk parse trees. The generated parse tree is provided as input to intermediate code generation.

- Intermediate Code:

We planned to have intermediate code generation written in Java. Intermediate code is generated by a listener class written in java along with listener classes generated by ANTLR.

- Runtime:

We planned on using python for handling the runtime environment. Runtime reads the intermediate code, identifies the control statements and stores the address of the control statement in the lookup hashmap. It then stores the declared variable and its values in the lookup environment. It performs execution line by line and updates the variable values in the hashmap. The order of execution depends on the conditional statements.

SYSTEM DESIGN