Milestone 1 - Spring 2023 Group 3

Members:
Akshata Bharadwaj - akodagnu
Rhishabh Hattarki - rhattark
Anmol More - amore9
Akansh Kumar - akuma352
Manoj Dara - mdara1

GitHub link: https://github.com/rhattark/SER502-Spring2023-Team3

Name - Hamlet Code

Design

Design Tools : DCG, Prolog, Python (for tokenization), bash/executable to run the program
Data Structures: List

Flow :

1. Bash code that takes program file as cli argument (eg. hamlet program.hamlet)
2. Internally it will run the following programs
    a. Python code to read the program file and generate tokens
    b. Prolog code that takes these tokens to create parse tree
    c. Prolog code that takes the parse tree to evaluate the output

Inspiration :

The aim to bring the worlds of computer programming and software development closer together inspired the creation of a programming language like HamletCode. It is an attempt to combine the creativity of language with the logic of code in order to make programming more fun and interesting to a broader audience. It may also be used to study and teach programming principles in an interesting and enjoyable way.

Data types	1. Boolean - a. bool i. aye(True) ii. nay(False) 2. String - a. verse 3. Numeric - a. numeral

Assignment operator	be (=)
Ternary operator	<condition>?<statement>:<statement>
if-then-else	If it be <condition> thee shall <scene_declaration>, otherwise <scene_declaration>
for	forsooth <assignment | variable_declaration> . <expression> . <assignment> doth <scene_declaration> .
while	whilst it be <expression> doth <scene_declaration> .
range-for	forsooth, let <identifier> be <number> to <number> by step size of <number> . doth <scene_declaration> .
print	sayeth <identifier | string | boolean>.
addition	+
subtraction	-
multiplication	*
division	/
not	naught
and	and
or	or
Less than / equal to	< / <=
Greater than / equal to	> / >=
equality	==

Grammar

program ::= act_declaration

act_declaration ::= "act" scene_declaration "end act"

scene_declaration ::= "scene" statement_list "end scene"

statement_list ::= statement | statement statement_list | scene_declaration

statement ::= variable_declaration
| assignment
| print
| conditional
| ternary
| traditional-for
| traditional-while
| range-for

variable_declaration ::= datatype identifier " be " expression "."

assignment ::= identifier " be " expression "."

datatype ::= “numeral” | “verse” | “bool”

expression ::= arithmetic_expression | boolean_expression

arithmetic_expression ::= arithmetic_term
| arithmetic_expression add_sub_operator arithmetic_term

arithmetic_term ::= arithmetic_factor
| arithmetic_term mul_div_operator arithmetic_factor

arithmetic_factor ::= identifier
| number
| "(" arithmetic_expression ")"

boolean_expression ::= arithmetic_expression comparison_operator arithmetic_expression
| bool_literal
| "(" boolean_expression ")"

bool_literal ::= "aye" | "nay"

print ::= "sayeth" (identifier | string | bool_literal) "."

conditional ::= "if it be" boolean_expression "then thee shall" scene_declaration "otherwise"
scene_declaration "."

ternary ::= boolean_expression "?" statement_list ":" statement_list "."

traditional-for ::= “forsooth” (assignment | variable_declaration) “.” boolean_expression “.” assignment “doth” scene_declaration “.”

traditional-while ::= "whilst it be" boolean_expression "doth" scene_declaration "."

range-for ::= "forsooth, let" identifier "be" number "to" number "by step size of" number ". doth" scene_declaration "."

identifier ::= letter | letter identifier

number ::= digit | digit number

string ::= “ letters_or_digit “

letters_or_digit ::= letter | digit | letter letters_or_digit | digit letters_or_digit

add_sub_operator ::= "+" | "-"
mul_div_operator ::= "*" | "/"
comparison_operator ::= "<" | ">" | "<=" | ">=" | "=="

letter ::= a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "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" | "W" | "X" | "Y" | "Z"

digit ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

Example Program

```
act
  scene
    let numeral a be 10.
    let verse b be "Hello, world!".
    let bool c be a > 5.
    sayeth a.
    sayeth b.
    sayeth c.
    if it be c then thee shall
      sayeth "The condition is true.".
    otherwise
      sayeth "The condition is false.".
```

```
  end scene
  scene
    forsooth, let i be 1 to 5 by step size of 1. doth
      sayeth i.
  end scene
  scene
    whilst it be i <= 5 doth
      sayeth i.
      let i be i + 1.
  end scene
end act
```
