MILESTONE 1

Team Members:

Akansha Reddy Anthireddygari | Girija Rani Nimmagadda | SaiRachana Paladugu | Sasikanth Potluri

Name of the Program: *ARGS*

Extension : .args

GitHub Link: https://github.com/Girija2905/SER502-Spring2023-Team11

Tools Used:

- Lexer : Python
- Parser : Python
- Interpreter : Python

Introduction:

Args is a programming language that has been intentionally created to be easily understandable and usable by people who have little or no experience in programming.

Language Tools:

To achieve a program output, the initial step involves reading the input file via a lexical analyzer, which converts the file's characters into tokens, stored in a token list. Following this, a parser verifies if the source code complies with the syntax rules and produces a parse tree. An interpreter processes the parse tree using syntax-based interpretation. The parser creates an intermediate code or parse tree file with a unique extension. These operations are designed in Prolog, utilizing list data structures.

Language Constraints and its operators:

1. Starting the program-> command “Start program”

```
Start program{
        !!block!!
}
```

2. **Block ->** should contain declarations and expressions.

3. **Primitive Types ->** num for integer, str for string and bit for boolean

   Declaration of primitive types:

   num x =:= 4

   str y =:= “this is a string”

   bit z =:= true

   bit w =:= false

4. **Assignment operator** -> “=:=”
5. **Comments** -> !!.....!!

   Syntax:

   !!This is a comment !!

6. **Print statement** -> “show” for print

   Syntax:

   show(x)

7. **Conditional constructs** ->
   - ***Ternary operator:***

     Syntax:

     x>y ?? show(x) :: show(y)

   - ***If then else:*** if for if; or_if for elif; if_not for else

     Syntax:

     ```
     if(condition){
         show(“if condition”)
     }or_if(condition){
         show(“or_if condition”)
     }if_not{
         show(“if_not condition”)
     }
     ```

8. **Looping Structures** ->
    - ***while loop:***

        Syntax:

        ```
        when(condition){
            !!expressions!!
        }
        ```

    - ***for loop****:* "for var in scope(start,end,increment)" -> here var is variable; scope is used instead of range; start is

        Syntax:

        ```
        for var in scope(0,10,1){
            show(var)
        }
        ```

        !!this will loop through 0 to 10 by incrementing var by 1 and display it!!

Operators:

1. ***Arithmetic Operators:***

    + for add operator

    ~ for subtract operator

    >* for multiplication operator

    >/ for division operator

    $ for modulus operator

    ^* for exponentiation operator

2. ***Comparison Operators:***

    =::= for equality operator

    > for greater than

    < for less than

    >=:= for greater than or equal

    <=:= for less than or equal

    !=:= for not equal to

3. ***Logical Operators**:*

    && for AND

    || for OR

    ## for NOT

Grammar

<program> ::= Start Program{<block>}

<block> ::= <declaration> | <expression>

<primitive_type> ::= num | str | bit

<declaration> ::= <primitive_type> <identifier>

<declaration> ::= <primitive_type> <identifier> =:= <value>

<identifier> ::= [a-zA-Z][a-zA-Z0-9]*

<value> ::= <num_value> | <str_value> | <bit_value>

<str_value> ::= [a-zA-Z0-9]*

<num_value> ::= <digit> | ~<digit>

<digit> ::= <digit><digit>

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

<bit_value> ::= true | false

<comparison_operator> ::= > | < | >=:= | <=:= | !=:= | =::=

<arithmetic_operator> ::= + | ~ | >* | >/ | $| ^*

<logical_operator> ::= && | || | ##

<print_statement> ::= show(<....>)

<comment> ::= !!....!!

<expression> ::= <identifier> =:= <arithmetic_expression>

<expression> ::= <identifier> =:= <ternary_operator>

<expression> ::= <identifier> =:= <logical_expression>

<expression> ::= <arithmetic_expression> | <logical_expression>

<expression> ::= <conditional_expression> | <looping_expression>

<arithmetic_expression> ::= <identifier> <arithmetic_operator> <expression>

<arithmetic_expression> ::= <identifier> <arithmetic_operator> <identifier>

<condition> ::= <comparison_expression> { <logical_operator> <comparison_expression> }

<comparison_expression> ::= <identifier> <comparison_operator> <expression>

<comparison_expression> ::= <identifier> <comparison_operator> <identifier>

<conditional_expression> ::= <ternary_operator> | <if_then_else>

<ternary_operator> ::= <condition> ?? <expression> :: <expression>

<if_then_else> ::= <if_statement> { <or_if_statement> } [<if_not_statement>]

<if_statement> ::= if(<condition>){ <block> }

<or_if_statement> ::= or_if(<condition>){<block> }

<if_not_statement> ::= if_not{<block> }

<looping_expression> ::= <while_loop> | <for_loop>

<while_loop> ::= when(<condition>){<block> }

<for_loop> ::= for <identifier> in scope(<start> ,<end> ,<increment>){ <block> }

<start> ::= <identifier> | <value>

<end> ::= <identifier> | <value>

<increment> ::= <num_value>