Rage

Nick Leluan & Taylor Willittes

Intro to Rage

- Rage is a spoken english-like language
  - More english-like than Python
- Name came from our frustration of using our own language
- Our language includes integers, string literals, arrays, and type inference

Syntax Choices - Variable Assignments

- Variable assignments:
  - <var_assgn> ::= var <var_name> = < i_expr>
    - Ex: var example = 2
    - Ex: var x = example (this will use type inference, so x is of type int)
  - <bool_var_assgn> :: = var <var_name> = <boolean>
    - Ex: var example2 = true
  - Variable names can be anything except for keywords.
    - Keywords include add, mult, div, sub, from, to, equal, etc.

Syntax Choices - Arithmetic Expressions

- Arithmetic Expressions
  - Chose to use add/sub/mult/div/mod to make the language more english like
  - Grammar:
  - <i_expr> ::= <i_expr> add <mult_expr> | <i_expr> sub <mult_expr> | <mult_expr>
  - <mult_expr> ::= <mult_expr> mult <neg_expr> | <mult_expr> div <neg_expr> | <mult_expr> mod <neg_expr> | <neg_expr>
  - Ex: var x = 3 add 3.5

Type Inference

- Rage uses type inference on variable assignments
  - Ex:

```
24
25 var x = 3.14
26 var y = x
27
28 Would translate to::
29
30 double x = 3.14
31 double y = x
32
```

- Type inference is also used for arithmetic expressions
  - Ex:

```
21
22 var x = 3 add 3.5
23 var y = x
24
25 Would translate to::
26
27 double x = 3 + 3.5
28 double y = x
29
```

Syntax Choices - Conditionals

- Conditionals
  - <conditionals> ::= if <comp> then <statements> end if | or if <comp> then <statements> end if | or <statement> end if
    - Ex: if x < y then z = z + 1 end if
    - The or keyword acts as the else keyword in Java

Syntax Choices - Loops

- Two types of loops: from loop and for each loop
- From loop acts as a for loop in java
  - <from_loop> ::= from <var_assgn> to <comp> increment by <integers> <statements> end | from <var_assgn> to <comp> decrement by <integers> <statements> end
  - Examples will be provided in Demo 1

Loops Example

- Example of a from loop with if statements:

```
5
6 from var i = 0 to m increment by 1
7     if i mod x equals 0 then
8         count = count + 1
9     end if
10    if i mod y equals 0 then
11        count = count + 1
12    end if
13    var num = x mult y
14    if i mod num equals 0 then
15        count = count - 1
16    end if
17 end from
```

Syntax Choices - Miscellaneous

- Printing to output
  - <print_to_output> ::= output(<root>) | outputs(<root>)
  - Output will print with no new line and outputs will print with a new line
- Command Line Arguments
  - <CLA> ::= cmd(<int>) | cmd(<int>) <CLA>
  - Ex: cmd(0) will translate to args[0] in java
- String Literals
  - <str_literals> ::= “<chars>”
- Comments
  - <comments> ::= start comment | end comment | *<chars>
  - Using the “*” character will allow for in-line comments

Tutorial Part 1

- https://youtu.be/_Jz8UZ2_VN0

Syntax Choices - Loops

- For each loop
  - <for_each> ::= for each <var_name> in <statements> do <statements> end for each
  - Chose this syntax to try to mimic how it would be spoken in English
  - An example will be shown in the Demo 2!

Syntax Choices - Arrays

- Can only hold ints
- You can assign or reassign a variable to an array
- Grammar:
  - <arrays> ::= var <var_name> = array(<integers> <empty> | <integers>, <integers>)
  - Ex: var x = array(1, 2, 3)
  - This will translate into an int array in Java

Tutorial Part 2

- https://youtu.be/GrCXaT7tkjk

Translator Problems

- Type inference
  - Difficult to translate the expression variable.
  - Solution was to use a hash map that stored the variable name as a String and the type as a String.
- In-line comments
  - Pattern matching the “*” character was challenging.
  - To solve this we had to check for the “*” character in every line.
- Overall it was challenging to figure out how to write the translator efficiently so we did not have to rewrite code.

Additional Language Choices

- If we were to continue to work on this language we would love to add more data structures. Such as hash maps, linked lists, and maybe trees.
- Adding methods to use on string literals similar to the methods that Java has.
- Also adding more methods to use on arrays.
- We would also like to improve upon what we completed and add more arithmetic expressions (square root/exponents).
- Changing keywords to translate to any language.
- Adding a way to create functions and return values from functions.