Danilo Hidalgo
Christian Trejo
CSC372

Project 2 - Part 1: The Grammar

Layout of our grammar in this document:

- Digit, Integer
- String, String Literal
- Variable Names
- Variable Assignment and Reassignment
- Basic Integer Expressions
- Booleans
- Boolean Expressions
- Comparisons
- Conditionals
- Loops
- Printing to Output
- Arguments
- Arrays
- Statements (general)
- Comments

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

String Literals:
```
<string> ::= <char><string>  | <char>
<string_literal> ::= “<string>”
<char> ::= <charLetter> | <digit> | @|#|$|%|^|&|*|(|)|-|=|[|]|{|}|\\|||'|;|:|<|>|,|.|?|/|`|~|"
```

Variable Names:
```
<charLetter> ::= a | b | … | y | z | A |  … | Z
<charNum>   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<var>  ::= <charLetter><end> | <charLetter> | _<charLetter> | _<end>
<end> ::= <charLetter><end> | <charNum><end> | <empty> | <end>
```

Variable Assignment:

```
<vars_assignment>  ::= let <var> = <value>
<var_reassignment> ::= <var> = <value>
<value> ::=  <var> | <integer> | <string_literal> | <ray> | <bool> | <or_expr> | <ray_index>
```

Basic integer expressions:

Note: Precedence: (lowest) + -, mod * /, ( ) (highest)

```
<expr>     ::= <expr> + <mmd_expr> | <expr> - <mmd_expr> | <mmd_expr>
<mmd_expr> ::= <mmd_expr>*<root> | <mmd_expr>/<root> |
               <mmd_expr> mod <root> | <root>
<root>     ::= <integer> | <var> | (<expr>) | -<expr>
```

Booleans:

```
<bool> ::= T | F
```

Boolean Expressions:

Note: Precedence: (lowest) or, and, not (highest)

```
<or_expr>    ::= <or_expr> or <and_expr> | <and_expr>
<and_expr> ::= <and_expr> and <not_expr> | <not_expr>
<not_expr>  ::= not <bool_root> | <bool_root> | <comparison>
<bool_root> ::= <bool> | (<or_expr>)
```

Comparison Expressions:

```
<comparison> ::= <or_expr> != <or_expr> | <or_expr> == <or_expr> |
                 <or_expr> < <or_expr> | <or_expr> <= <or_expr> |
                 <or_expr> > <or_expr> | <or_expr> >= <or_expr>
```

Conditionals:

```
<conditional>   ::= <if_statement> |
                    <if_statement>
                    <elf_statement>
                    <else_statement>
<if_statement>  ::= if <bool_expr>:
                        <statement>
<elf_statement> ::= elf <bool_expr>:
                        <statement>
                    <elf_statement>
                      | <empty>
<else_statement> ::= else:
                        <statement>
```

Loops:

```
<loops> ::=     for <var> in <range>[<integer>]:
                        <statement>  |
                loop <or_expr>:
                        <statement>
<range> ::= <integer> .. <integer>
```

Note: In for-loop, <integer> is an optional increment and defaults = 1.
Note: In for-loop, <range> is the range of values to iterate over, inclusive of starting value but exclusive of the ending value.

Printing to Output:

```
out(<print_argument>)                Note: No newline
outln(<print_argument>)              Note: Printed with trailing \n
<print_argument> ::= <digit> | <integer> | <string_literal> | <var>
```

Arguments:

argos

Arrays:

```
<ray>      ::= [<int_list>]  |  [<string_list>]  |  [<bool_list>] | b{<expr>} | i{<expr>} | s{<expr>}
<int_list> ::= <integer>,<int_list>  |  <var>,<int_list>  |  <integer>  |  <var>
<string_list> ::= <string_literal>,<string_list>  |  <var>,<string_list>  |  <string_literal>  |  <var>
<bool_list> ::= <bool>,<bool_list>  |  <var>,<bool_list>  |  <bool> | <var>
```

Note: <var> must be of the same type as the array

Accessing arrays:
<ray_index> ::= <var>[<expr>]
<ray_index_assign> ::= <ray_index_access> = <value>

Statements:
<statement> ::= <var_assignment> <statement> | <var_reassignment> <statement> | <loops>
<statement> | <conditionals> <statement> | hallpass <statement> | hallpass

Note: *hallpass* - keyword to represent an empty block

Comments:
<comment> ::= ? <string>