Forest++

Sarah Ling, Simon Jenkins

0.1 Minimally Working Interpreter

a) AST representing of Data:
Currently, our minimally working *Forest++* has datatypes: Season, number of Trees, Tree Type, and Size of Tree. These are all the expected datatypes that will also appear in our final edition of *Forest++*

b) AST representation of Combined forms:
Currently *Forest++* has Grove, Forest, and Landscape implemented. Grove is a combination of Number of Trees, Tree Type (kind), and scale size of the Tree. Forest is a list of Groves. Landscape is the combination of the Season and Forest. This is how the final AST in *Forest++* was imagined to be. The AST file in our minimally working version is complete and a sizeable chunk towards our final goal.

c) Parser recognizability of the AST
Our parser can parse and allocated to all the datatypes and combined forms as should be. The parser allocates the season by noticing the ':' and understands that the values separated by ',' are part of a list, with strings of the type of tree and "size" characterizing each specific grove. As for unfinished parts of our parser, our model cannot parse undesignated size values and unspecified randomized values. For example:

```
fall: birch size, pine size, oak size
```

cannot be run quite yet by our program since there is no value in front of size, and there is no specified value before "birch", "pine", or "oak". In order to run it you would have to say something like:

```
fall: 15 birch 1 size, random pine 2 size, 3-8 oak 1.5 size
```

d) Evaluator Class
Our evaluator can produce what is parsed as a blank file with SVG trees spread upon the file. Some of the tree files were taken from open-source materials online, others were made. The one aspect that we have yet to implement is the season option. Ideally, you can designate "fall" or "spring" which will change the overall color scheme of the trees, but currently, they are defaulted to "fall" and cannot be changed until further implementation from the Evaluator class. Since we could not figure out how to change the SVG of the individual tree file per season, we changed the background color to denote some effect of season.

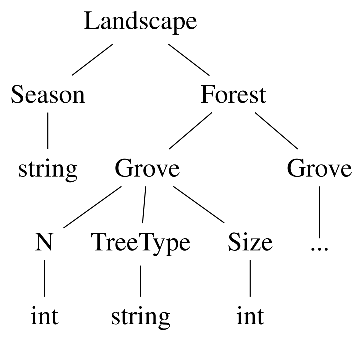
0.2 Minimal Formal Grammar

```
<landscape> ::= <season>:␣<forest>
    | <forest>

<forest> ::= <grove>,␣<forest>
    | <grove>
<season> ::= spring
    | fall
<grove> ::= <n> <treetype> <size> size
    | <n> <treetype>

<n> ::= <int>
    | random
    | <int> - <int>
<int> ::= 1 - 20

<size> ::= <s>
```

```
    | random
<s> ::= <fp>
    | random
<fp> ::= R in {0.25, 2}

<type> ::= maple
    | oak
    | birch
    | pine
```

How our AST is set up, there is no particular right associativity or left associativity, meaning there is only one possible AST that can be created from an input. We refined some of the numbers so that n can only range from 1 - 20, since more trees would be too much as of now, and that the random scale factor for a tree is from 0.25 to 2, since 5x a tree was too large.

This tree shows how the types within the AST of the language fit together:



0.3 Minimal Semantics

Associativity will be ignored, since there can only be one AST tree made for each input.

Syntax : Abstract Syntax : Type : Meaning
Tree : Maple or Oak or Pine or Birch : 'a: Tree How we will distinguish which SVG file to access
Season : Fall or Spring : 'b: Season : How we will know what color scheme to use
Grove : num: int; kind: Tree; size: float : int → Tree → float : How we will know how many and what size trees we will put on the page of a certain type
Forest : list grove : list$< Grove >$: How we will keep the groves together
Landscape : season: Season; forest: Forest : Season → Forest : How we combine the colors with the SVG file choices
. : . : . : .
Season: : Season ':␣' : string : The colon string indicates that what is left of the colon is the season, and what is right of the colon is the beginning of the Forest
Grove, Grove, Grove, etc : Grove',␣': string : The commas inside the Forest denote where one grove starts and another one ends