Significance

```
(Tripp) Milton Lamb
10/26/2025
CS-524 Fall
```

1. Overview

Significance is a toy programming language created for CS-524. The interpreter (an executable of the same name) is written in Rust. The interpreter is capable of REPL and full file parsing.

Simply invoking the executable in a terminal will start the REPL.

```
Significance.exe
```

> Note: the REPL still has some bugs, and is not fully ready for use

By passing a file path as an argument to the executable, the file parsing will initiate.

```
Significance.exe <filename>
```

> Currently, in order to support the second delivery, running Significance on a file will also produce an 'ast.json' file in the current working directly. This JSON file is a rendering of the Abstract Syntax Tree generated by parsing the filename argument. This feature will be removed or made optional via command line optional argument in the final version.

2. Language Basics

Most of the available features of Significance follow expected behavior. The major differences are the reduction of feature size and the inclusion of uncertainty as part of the native behavior of the language.

2.1 Native Type

There is only a single native type in significance; the `real` type. The `real` type can behave very similarly to a standard double precision number; however, there is optional uncertainty behavior. When the uncertainty behavior is not assigned it will still be propagated, but the propagated value is simply `0.0`.

> As an example the two below methods are equivalent.
>
> ```
> x := 5.2
> ```
>
> and
>
> ```
> x := 5.2 +/- 0.0
> ```
>
> The following still yields a base double precision number of `5.2`, but also includes an uncertainty value
>
> ```
> x := 5.2 +/- 0.02
> ```

`real` types are immutable. Once they are assigned a value they cannot be reassigned.

> Note: There is technically a single reassignment from the default `0.0 +/- 0.0` upon declaration, but this will likely change before the final release.

2.2 Statements

There are three kinds of statements in Significance.

2.2.1 Declaration

The declaration statement is required in order to use a variable. A variable is declared by the pattern:

```
{<id> : <type>}
```

or concretely

```
{x : real}
```

There is only one implemented type in Significance so `<type>` can currently only be replaced by `real`. Once a variable has been declared it cannot be redeclared.

2.2.2 Assignment

The second kind of statement available in Significance is the assignment statement which follows the pattern:

```
<id> := <expression>
```

The expression usage should follow expectation. An expression can be a literal (`2.1 +/- 0.015`); an identifier (`x`); an expression following a unary operator (`-x`); two expressions connected with a binary operator (`x * 4.2 +/- 0.04`); or an internal function call (`sin(x)`).

2.2.3 Expression Statement

The third kind of statement available in Significance is the print expression statement. If an expression exists without an assignment, the result of the expression will be printed to console.

```
<expression>
```

> An example:
>
> ```
> {x :real}
> x = 5.0
> x # will print '5.0' to console
> ```

2.3 Comments

Significance also supports single line comments. Anything following a pound symbol `#` but before a newline will be considered a comment, and will have no effect on the program execution.

3. Operators

3.1 Binary Operators

All operators propagate the uncertainty of the operands. Addition and Subtraction propagate uncertainty in quadrature:

$$\text{Given } a \pm \delta_a \text{ and } b \pm \delta_b$$
$$\text{Addition: } z = a + b, \quad \delta_z = \sqrt{(\delta_a)^2 + (\delta_b)^2}$$
$$\text{Subtraction: } z = a - b, \quad \delta_z = \sqrt{(\delta_a)^2 + (\delta_b)^2}$$

while multiplication and division propagate relative uncertainty:

$$\text{Given } a \pm \delta_a \text{ and } b \pm \delta_b$$
$$\text{Multiplication: } z = a \times b, \quad \frac{\delta_z}{|z|} = \sqrt{\left(\frac{\delta_a}{|a|}\right)^2 + \left(\frac{\delta_b}{|b|}\right)^2}$$
$$\text{Division: } z = \frac{a}{b}, \quad \frac{\delta_z}{|z|} = \sqrt{\left(\frac{\delta_a}{|a|}\right)^2 + \left(\frac{\delta_b}{|b|}\right)^2}$$

Uncertainty propagation with modulus, power, and root have not been implemented yet, and so set the uncertainty to zero rather than give incorrect uncertainty.

Symbol	Name	Description
`+`	Addition	Adds two numbers, propagates uncertainty in quadrature
`-`	Subtraction	Subtracts two numbers, propagates uncertainty in quadrature
`*`	Multiplication	Multiplies two numbers, propagates relative uncertainty
`/`	Division	Divides two numbers, propagates relative uncertainty
`%`	Modulus	Returns remainder of division,
`**`	Power	Raises left operand to the power of right operand
`//`	Root	Takes the nth root (left // right = left^(1/right))
:=	immutable assignment	Assigns the right operand (expression) to the left operand (identifier) immutably

3.2 Unary Operators

Both unary operators are used. The unary plus operator doesn't actually do anything, but it is allowed for clarity.

> Note: that the plus in the scientific notation form isn't actually a unary operator, it is part of the format of the exponent definition.

Symbol	Name	Description
`+`	Unary Plus	Returns the value unchanged
`-`	Unary Minus	Negates the value

3.3 Precedence Rules

The precedence rules should be as expected. The precedence rules of Significance follow the majority of other languages and mathematics in general.

Symbol	Name	Precedence Level	Associativity
`()`	Parentheses	1 (highest)	N/A
`+`, `-`	Unary Plus/Minus	2	Right
`**`, `//`	Power, Root	3	Right
`*`, `/`, `%`	Multiplication, Division, Modulus	4	Left
`+`, `-`	Addition, Subtraction	5	Left
`:=`	Immutable Assignment*	6 (lowest)	Right

*Immutable assignment isn't technically associative or make use of a precedence level in Significance since it isn't part of an expression, only an assignment statement; however, it is still useful to be mentioned.

4. Functions

4.1 Built-in Functions

Function	Description
`sin(x)`	Returns the sine of x [radians]
`cos(x)`	Returns the cosine of x [radians]
`sqrt(x)`	*Returns the square root of x

*`sqrt(x)` is unnecessary in Significance since `x // 2.0` is equivalent, but it was a requirement of the language so it has been included.

4.2 User Defined Functions

User defined functions do not exist in this language.

5. Example Programs

5.1 First Example

```
#This is a comment
#Below is an example program in significance

{x : real} # this is the `x` variable
{y : real} # this represents a change in `x`
{z : real} # `z` is the next iteration of `x`
{w : real} # `w` is the magnitude of `x` and `z`

x := 12.3 +/- 0.5    # assign 12.3 with uncertainty 0.5 to `x`
y := 2.6             # assign 2.6 with uncertainty 0.0 to `y`
z := x + y           # assign the sum of `x` and `y` to `z`
z                    # print `z` to console
w := x*x + z**2      # assign the sum of the square of `x`
                     #     and the square of `z` to `w`
w
```

5.2 Second Example

```
sin(2.1 +/- 0.1) # print the result of taking the sine of the
                 #     literal `2.1 +/- 0.1`
```

6. Third Delivery Implementation Plan

This section addresses the concerns listed in the delivery 2 requirements, and makes note of the features still required to be implemented before the final delivery of the language.

6.1 REPL

REPL is not fully implemented. REPL will run, but the symbol table and the run var table is not currently being updated properly. There is a REPL mode of the Significance parser. In this mode a full program lifetime parser, analyzer, and executor will be created which store the symbol and run time var table for the entire time of REPL interaction.

The file parsing uses `AstParser.parse_program()`, `SemanticAnalyzer.analyze_program()`, `Executor.execute_program()` calls while the REPL uses `AstParser.parse_statement()`, `SemanticAnalyzer.analyze_statement()`, `Executor.execute_statement()` calls. The `*_statement()` call are used by the `*_program()` calls which leads to nearly full overlap between the file parser and the REPL parser.

6.2 Error Reporting

Syntax and semantic error reporting are implemented, but haven't been fully tested. The formatting needs improvement as well. Errors are simply print to the console after input is given to the Significance executable. Currently some errors are not being properly handled and will crash the Significance parser.

6.3 Expression Evaluation

Expression evaluation is already fully implemented. Operations and precedence follow the grammar from the previous delivery and the descriptions listed above. Internally there is an Executor class which takes in the decorated AST tree as an argument. It then creates an internal run time variable table based on the symbol table and then modifies the run time variable table dynamically based on assignment statements in the program.

6.4 Grammar

The grammar intention has not changed since inception, but it will be double checked and included in the final delivery.