Dust

Text, letter

Description automatically generated31 de marzo del 2022

A01252554 Santiago Díaz Guevara

# Propósito y objetivo del proyecto

Desarrollar el compilador para un lenguaje de programación que permita usar vectores para realizar análisis estadísticos significativos, incluyendo algunos métodos estadísticos básicos y sus gráficas.

El lenguaje por desarrollar tiene el nombre de Dust, por estar diseñado en base a la sintaxis de Rust, pero con un conjunto de características muy reducido.

# Requerimientos del lenguaje

## Elementos básicos

Palabras reservadas: as, break, continue, else, false, fn, if, let, loop, return, static, true, while, bool, i32, f64, char

Literales: , ) ( : ; ] [ -> } { = - ! + \* / % == != > < <= >= || &&

|  |  |
| --- | --- |
| Token | Expresión Regular |
| IDENTIFIER | [A-Za-z\_][0-9A-Za-z\_]\* |
| CHAR\_LITERAL | ‘[^’\\\n\t\r]’ |
| INTEGER\_LITERAL | [0-9][0-9]\* |
| FLOAT\_LITERAL | [0-9][0-9]\*\.[0-9][0-9]\* |
| BOOL\_LITERAL | true | false |

## Diagramas de sintaxis

Precedence (from higher to lower)

IDENTIFIER

Function Call, array indexing

Unary Operators (-, !)

as

\* / %

+ -

==, !=, <, >, <=, >=

&&

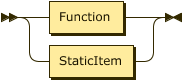
||

return, break

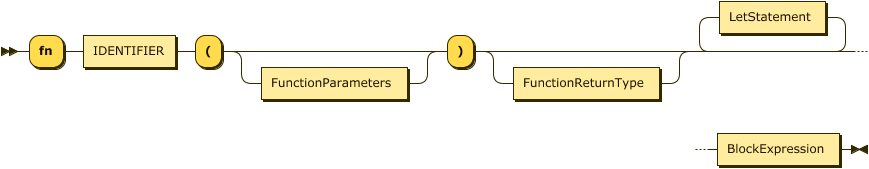
Crate (program):



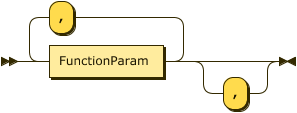
Item:



Function:



FunctionParameters:



FunctionParam:



FunctionReturnType:



StaticItem:



Statements:



Statement:



ExpressionStatement:



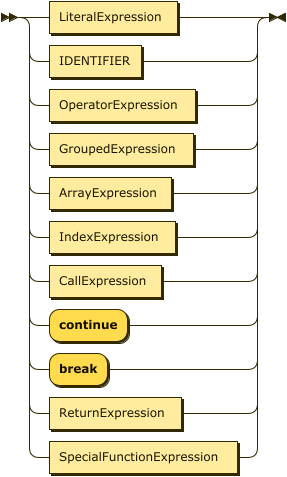
LetStatement:



Expression:



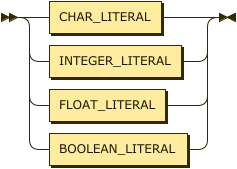
ExpressionWithoutBlock:



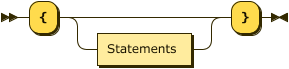
ExpressionWithBlock:



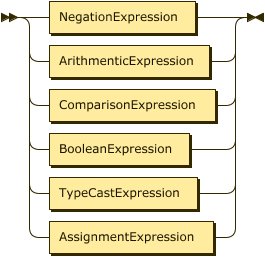
LiteralExpression:



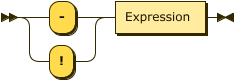
BlockExpression:



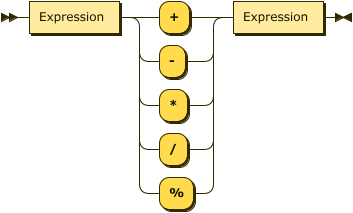
OperatorExpression:



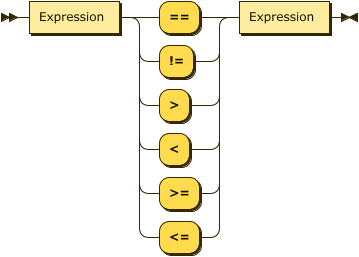
NegationExpression:



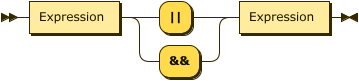
ArithmeticExpression:



ComparisonExpression:



BooleanExpression:



TypeCastExpression:



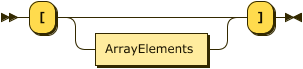
AssignmentExpression:



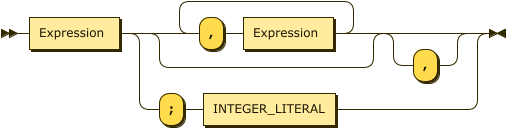
GroupedExpression:



ArrayExpression:



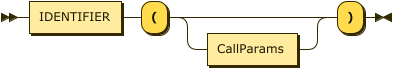
ArrayElements:



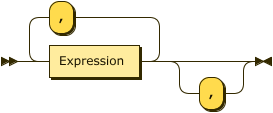
IndexExpression:



CallExpression:



CallParams:



LoopExpression:



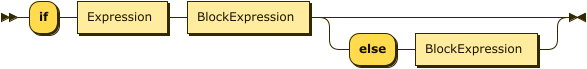
InfiniteLoopExpression:



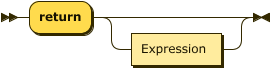
PredicateLoopExpression:



IfExpression:



ReturnExpression:



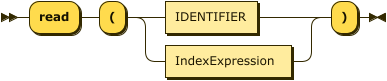
SpecialFunctionExpression:



IoExpression:



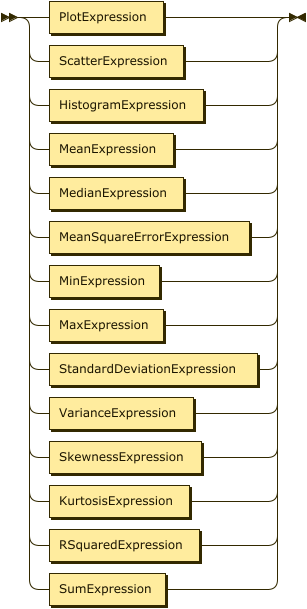
ReadExpression:



WriteExpression:



StatisticExpression:



PlotExpression:



ScatterExpression:



HistogramExpression:



MeanExpression:



MedianExpression:



MeanSquareErrorExpression:



MinExpression:



MaxExpression:



StandardDeviationExpression:



VarianceExpression:



SkewnessExpression:



KurtosisExpression:



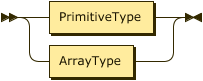
RSquaredExpression:



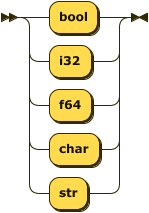
SumExpression:



Type:



PrimitiveType:



ArrayType:



## Características semánticas

Semantic Table

Todas las celdas vacías dan error, el diseño del lenguaje obliga que haya *casts* explícitos para realizar operadores binarias entre diferentes tipos. Los casts se realizan a través del operador ‘as’ o funciones de mapeo definidas por el usuario.

Operaciones binarias

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| left | right | \* | / | % | + | - | == | != | < | > | <= | >= | && | || | as |
| bool | bool |  |  |  |  |  | bool | bool |  |  |  |  | bool | bool | bool |
| bool | i32 |  |  |  |  |  |  |  |  |  |  |  |  |  | i32 |
| bool | f64 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| bool | char |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| i32 | bool |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| i32 | i32 | i32 | i32 | i32 | i32 | i32 | bool | bool | bool | bool | bool | bool |  |  | i32 |
| i32 | f64 |  |  |  |  |  |  |  |  |  |  |  |  |  | f64 |
| i32 | char |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| f64 | bool |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| f64 | i32 |  |  |  |  |  |  |  |  |  |  |  |  |  | i32 |
| f64 | f64 | f64 | f64 |  | f64 | f64 | bool | bool | bool | bool | bool | bool |  |  | f64 |
| f64 | char |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| char | bool |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| char | i32 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| char | f64 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| char | char |  |  |  |  |  | bool | bool | bool | bool | bool | bool |  |  | char |

Operaciones unarias

|  |  |  |
| --- | --- | --- |
| operando | - | ! |
| bool |  | bool |
| i32 | i32 |  |
| f64 | f64 |  |
| char |  |  |

## Descripción de funciones especiales

read(s: [char])

Lee de stdin y coloca los caracteres en el arreglo pasado.

write(s: [char])

Escribe el arreglo de caracteres pasado a stdout.

plot(x: [f64], y: [f64])

Plotea ‘y’ contra ‘x’ con líneas.

scatter(x: [f64], y: [f64])

Muestra un scatter plot de ‘y’ contra ‘x‘.

histogram(x: [f64])

Plotea un histograma.

mean(x: [f64]) -> f64

Calcula la media de un arreglo de números.

median(x: [f64]) -> f64

Calcula la mediana de un arreglo de números.

mean\_square\_error(x: [f64], y: [f64]) -> f64

Calcula el promedio del error cuadrado entre los valores de ‘x’ y ‘y’.

min(x: [f64]) -> f64

Retorna el valor menor del arreglo.

max(x: [f64]) -> f64

Retorna el valor máximo del arreglo.

standard\_deviation(x: [f64]) -> f64

Retorna la desviación estándar de los valores del arreglo.

variance(x: [f64]) -> f64

Retorna la varianza de los valores del arreglo.

skewness(x: [f64]) -> f64

Retorna la curtosis de los valores del arreglo.

r\_squared(x: [f64], y: [f64]) -> f64

Retorna el valor r2 de los arreglos ‘x’ y ‘y’.

## Tipos de datos

Bool, i32, f64, char.

Arreglos de tipo primitivo de hasta dos dimensiones.

# Lenguaje y sistema operativo usado para desarrollo

Para desarrollar el compilador, se usa como lenguaje base Python, y el sistema operativo Windows.

# Bibliografía

The Rust Reference. <https://doc.rust-lang.org/stable/reference/>