Definir los tipos de dato

Definir la asignación a variables

Definir la reasignación a variables (en especial por los arreglos)

Usar keyword para indicar que es una palabra reservada

Literal es para valores explicitos. Por ejemplo la definición de un entero

Operador es para el símbolo de ese operador

# Símbolos (números, operadores, etc cosas sueltas para luego llamarlas)

<left\_parenthesis> ::= “є”

<right\_parenthesis> ::= “э”

<plus\_operator> ::= “+”

<minus\_operator> ::= “-“

<multiplication\_operator> ::= “\*”

<division\_operator> ::= “/”

<int\_division\_operator> ::= “//”

<modulo\_operator> ::= “%”

<power\_operator> ::= “^”

<increment\_operator> ::= "++"

<decrement\_operator> ::= "--"

<digit1to9\_literal> ::= [1-9]

<decimal\_digit\_literal> ::= [0-9]

<zero\_literal> ::= 0

<bool\_literal> ::= “true” | “false”

<dot\_literal> :: “.”

<letter\_or\_underscore> ::= [a-zA-Z\_]

<identifier\_char> ::= [a-zA-Z0-9\_]

<int\_keyword> ::= “int”

<float\_keyword> ::= “float”

<bool\_keyword> ::= “bool”

<char\_keyword> ::= “char”

<string\_keyword> ::= “string”

<let\_keyword> ::= “let”

<numeric\_type> ::= <int\_keyword> | <float\_keyword>

<text\_type> ::= <char\_keyword> | <string\_keyword>

# Tipos de datos literales

<int\_literal> ::= <minus\_operator>? <digit1to9\_literal> <decimal\_digit\_literal>\*

<int\_literal> ::= <zero\_literal>

<float\_literal> ::= <zero\_literal> <dot\_literal> <zero\_literal>

<float\_literal> ::= <minus\_operator>? <zero\_literal> <dot\_literal> <decimal\_digit\_literal>\* <digit1to9\_literal>+

<float\_literal> ::= <minus\_operator>? <digit1to9\_literal> <decimal\_digit\_literal>\* <dot\_literal> (<decimal\_digit\_literal>\* <digit1to9\_literal>+ | <zero\_literal>)

<char\_literal> ::= ‘\.’\ NO ESTOY SEGURO

<string\_literal> ::= “\.\*”\ NO ESTOY SEGURO

# Identificador

<identifier> ::= <letter\_or\_underscore> <identifier\_char>\*

# Operaciones aritméticas

<unary\_negative> ::= <minus\_operator> (<int\_literal> | <float\_literal>)

<postfix\_expression> ::= < identifier > (<increment\_operator> | <decrement\_operator>)

<arithmetic\_expression> ::= <arithmetic\_expression> (<plus\_operator> | <minus\_operator>) <term> | <term>

<term> ::= <term> (<multiplication\_operator> | <division\_operator> | <int\_division\_operator> | <modulo\_operator>) <power> | <power>

<power> ::= <factor> <power\_operator> <power>| <factor>

<factor> ::= <left\_parenthesis> <arithmetic\_expression> <right\_parenthesis> | <arithmetic\_operands>

<arithmetic\_operands> ::= <int\_literal> | <float\_literal> | <identifier> | <unary\_negative> | <postfix\_expression>

# Notas temporales

Hacer que la recursión solo venga de un lado



Poner primero los de más baja precedencia para que su árbol de recursión se expanda primas

Lo de las operaciones aritméticas fue tomado de https://athena.ecs.csus.edu/~gordonvs/135/resources/04bnfParseTrees.pdf