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”

<wall\_comment> ::= “|”

<left\_exclamation> ::= “!”

<right\_exclamation> ::= “¡”

<left\_block> ::= “¿”

<right\_block> ::= “?”

<plus\_operator> ::= “+”

<minus\_operator> ::= “-“

<multiplication\_operator> ::= “\*”

<division\_operator> ::= “/”

<int\_division\_operator> ::= “//”

<modulo\_operator> ::= “%”

<power\_operator> ::= “^”

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

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

<assignment\_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”

<input\_keyword> ::= “input”

<output\_keyword> ::= “output”

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

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

<greater\_operator> ::= “>”

<greater\_equal\_operator> ::= “>=”

<less\_equal\_operator> ::= “<=”

<equal\_operator> ::= “==”

<not\_equal\_operator> ::= “!=”

<delimiter> ::= “$”

<line\_break> ::= “\n”

# 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>

# Operaciones lógicas

Esto vease como: 5 > 3, x == 4

<relational\_expression> ::= <arithmetic\_expression> <relational\_operator> <arithmetic\_expression>

<relational\_operator> ::= <greater\_operator> | <less\_operator> | <less\_equal\_operator> | <greater\_equal\_operator> | <equal\_operator> | <not\_equal\_operator>

<logical\_operator> ::= "~" | "@"

Esto permite cosas como (5+2 @ 3 ) @ False o también 5 > 2

<condition> ::= <contion\_simple> ( <logical\_operator> <condition\_simple> )\*

<condition\_simple> ::= <relational\_expression> | <bool\_literal> | <identifier> | <left\_ parenthesis > <condition> <right \_ parenthesis >

<not\_logical\_operator> ::= ”Σ”

<not\_condicion> ::= <not\_logical\_operator> <condicion\_simple>

Declaración de variables

Pero ya seria algo como Let int x = 5$

<declaration> ::= <let\_keyword> <type> <identifier> <assignment\_operator> <literal> <delimiter>

<type> ::= <int\_keyword> | <float\_keyword> | <bool\_keyword> | <char\_keyword> | <string\_keyword>

/// AQUI TAL VEZ QUITAMOS EL BOOL

<literal> ::= <int\_literal> | <float\_literal> | <bool\_literal> | <char\_literal> | <string\_literal>

REASIGNACIÓN aqui no contemplamos x += 2 o y \*= 2 preguntar al profe sino algo así (suponga que ya declaro x) x = 2$

<Reassignment> ::= <identifier> <assignment\_operator> <literal> <delimiter>

Comentarios

// Tomando a dot\_literal como un punto que en REGEX es cualquier cosa menos salto de linea

<simple\_comment> ::= <wall\_comment> <dot\_literal>\*

<multiple\_comment> ::= <left\_exclamation> ( <dot\_literal> | <line\_break> )\* <right\_exclamation>

CREACION IF o intento - ME FALTAN COSAS POR DEFINIR

<controlIf> ::= "IF" <left\_parenthesis> <condition> <right\_parenthesis> ( <block> | <statements>)\*

<block> ::= <left\_block> <statements>\* <right\_block>

<statements> ::= <declaration> | <reassignment> | <controlIf> |

# 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