Universidade Federal de Alagoas
Instituto de Computação
Compiladores 2021.1

Gramática EBNF da Linguagem neo-gorm

Eduardo Brasil Araujo, Lael de Lima Santa Rosa

Maceió - Alagoas
Junho de 2022

Sumário

Especificação EBNF

```
  Program         =  { Function } MainFunction

  Function        =  "fn" [ Type ] Identifier "(" Parameters ")" "{"
Declarations Statements "}"

  Parameters      =  [ Parameter { "," Parameter } ]

  Parameter       =  Type Identifier

  MainFunction    =  "fn" "i32" "main" "(" ")" "{" Declarations
Statements "}"

  Declarations    =  { Declaration }
  Declaration     =  Type  Identifier [ "[" Integer "]" ]
                     { "," Identifier [ "[" Integer "]" ] } [ "="
Expression ] ";"

  Type            =  "i32" | "u32" | "i64" | "u64" | "bool" | "f32" |
"f64" | "char" | "str"

  Statements      =  { Statement }
  Statement       =  ";" | Block | Assignment | IfStatement |
WhileStatement | ForStatement | CallStatement | ReturnStatement

  Block           =  "{" Statements "}"

  Assignment      =  Identifier [ "[" Expression "]" ] "=" Expression
";"

  IfStatement     =  "if" "(" Expression ")" Statement [ "else"
Statement ]

  WhileStatement  =  "while" "(" Expression ")" Statement

  ForStatement    =  "for" "(" Type Identifier ":" Integer "," Integer
[ "," Integer ] ")" Statement

  CallStatement   =  Call";"

  ReturnStatement =  "return" Expression";"

  Call            =  Identifier "(" Arguments ")"

  Arguments       =  [ Expression {"," Expression} ]

  Expression      =  Conjunction { "||" Conjunction }

  Conjunction     =  Equality { "&&" Equality }
  Equality        =  Relation [ EquOp Relation ]
  EquOp           =  "==" | "!="

  Relation        =  Addition [ RelOp Addition ]
  RelOp           =  "<" | "<=" | ">" | ">="

  Addition        =  Term { AddOp Term }
  AddOp           =  "+" | "-"

  Term            =  Factor { MulOp Factor }
  MulOp           =  "*" | "/" | "%"

  Factor          =  [ UnaryOp ] Primary
  UnaryOp         =  "-" | "!"

  Primary         =  Identifier [ "[" Expression "]" ] [ ".."
Identifier ]
                  |  Identifier"^"
                  |  Literal
                  |  "(" Expression ")"
                  |  Expression "as" Type
                  |  Call

  Identifier      =  Letter { Letter | Digit | "_" }
  Letter          =  "a"-"z""A"-"Z"
  Digit           =  "0"-"9"
  Literal         =  Integer | Boolean | Float | Char
  Integer         =  Digit { Digit }
  Boolean         =  "true" | "false"
  Float           =  Integer"."Integer
  Char            =  "'"ASCIIChar"'"
  ASCIIChar       =  0x00-0xff
```