Universidade Federal de Alagoas
Instituto de Computação
Compiladores 2021.1

Especificação dos tokens da linguagem neo-gorm

Eduardo Brasil Araujo , Lael de Lima Santa Rosa

Maceió - Alagoas
Junho de 2022

Sumário

1. Linguagem de implementação **3**

2. Enumeração dos tokens **3**

3. Expressões regulares dos lexemas **4**

3.1 Representação da ocorrência de símbolos usando expressões regulares **4**

1. Linguagem de implementação

A linguagem que implementa a *neo-gorm* é C++20.

2. Enumeração dos *tokens*

```
enum TokenCategory
{
    END_OF_FILE,
    IDENTIFIER,
    INT32,
    UINT32,
    INT64,
    UINT64,
    FLOAT32,
    FLOAT64,
    CHAR,
    STRING,

    BOOLEAN,
    INT_LITERAL,
    FLOAT_LITERAL,
    CHAR_LITERAL,
    STRING_LITERAL,
    TRUE_CODE,
    FALSE_CODE,
    NEGATION_OP,
    CAST_OP,
    ASSIGN_OP,
    CONCAT_OP,
    LENGTH_OP,
    PLUS_OP,
    MINUS_OP,
    MULT_OP,
    DIV_OP,
    MOD_OP,
    AND_OP,
    OR_OP,
    EQ_OP,
    NTEQ_OP,
    GT_OP,
    LT_OP,
    GTEQ_OP,
    LTEQ_OP,
```

```
    MAIN_CODE,
    PRINTF_CODE,
    READ_CODE,
    RETURN_CODE,
    FN_CODE,
    IF_CODE,
    ELSE_CODE,
    WHILE_CODE,
    FOR_CODE,
    LEFT_BRACKET,
    LEFT_SQRBRACKET,
    LEFT_PAREN,
    RIGHT_BRACKET,
    RIGHT_SQRBRACKET,
    RIGHT_PAREN,
    COMMA,
    COLON,
    SEMICOLON,
    QUOTE,
    ERR_NUMBER_LITERAL,
    ERR_SYMBOL,
    ERR_CHAR,
};
```

3. Expressões regulares dos lexemas

3.1 Definindo nomes auxiliares

- ➔ letter [a-z][A-Z]
- ➔ digit [0-9]

3.2 Tabela de correspondência

ID	NOME	EXPRESSÃO REGULAR
0	EOF	<<EOF>>
1	IDENTIFIER	[{letter}][{letter}{digit}_]*
2	INT32	(i32)[^a-zA-Z0-9_]
3	UINT32	(u32)[^a-zA-Z0-9_]
4	INT64	(i64)[^a-zA-Z0-9_]

5	UINT64	(u64)[^a-zA-Z0-9_]
6	FLOAT32	(f32)[^a-zA-Z0-9_]
7	FLOAT64	(f64)[^a-zA-Z0-9_]
8	CHAR	(char)[^a-zA-Z0-9_]
9	STRING	(str)[^a-zA-Z0-9_]
10	BOOLEAN	(bool)[^a-zA-Z0-9_]
11	INT_LITERAL	(-?{digit}+)[^a-zA-Z\._]
12	FLOAT_LITERAL	(-?{digit}+\.{digit}+)[^a-zA-Z\._]
13	CHAR_LITERAL	\'[^’]\'
14	STRING_LITERAL	\"[^”]*\"
15	TRUE_CODE	true[^a-zA-Z0-9]
16	FALSE_CODE	false[^a-zA-Z0-9]
17	NEGATION_OP	!
18	CAST_OP	(as)[ ]
19	ASSIGN_OP	=
20	CONCAT_OP	\.\.
21	LENGTH_OP	\^
22	PLUS_OP	\+
23	MINUS_OP	-
24	MULT_OP	*
25	DIV_OP	\/
26	MOD_OP	\%
27	AND_OP	&&
28	OR_OP	||
29	EQ_OP	==
30	NTEQ_OP	!=
31	GT_OP	>
32	LT_OP	<
33	GTEQ_OP	>=

34	LTEQ_OP	<=
35	MAIN_CODE	(main)[^a-zA-Z0-9_]
36	READ_CODE	(read)[^a-zA-Z0-9_]
37	PRINTF_CODE	(printf)[^a-zA-Z0-9_]
38	RETURN_CODE	(return)[^a-zA-Z0-9_]
39	FN_CODE	(fn)[^a-zA-Z0-9_]
40	IF_CODE	(if)[^a-zA-Z0-9_]
41	ELSE_CODE	(else)[^a-zA-Z0-9_]
42	WHILE_CODE	(while)[^a-zA-Z0-9_]
43	FOR_CODE	(for)[^a-zA-Z0-9_]
44	LEFT_BRACKET	\{
45	LEFT_SQRBRACKET	\[
46	LEFT_PAREN	\(
47	RIGHT_BRACKET	\}
48	RIGHT_SQRBRACKET	\]
49	RIGH_PAREN	\)
50	COMMA	,
51	COLON	:
52	SEMICOLON	;
53	ERR_NUMBER_LITERAL	[0-9.][0-9\\.a-zA-Z]*
54	ERR_SYMBOL	[^0-9a-zA-Z\{\[\(\}\]\),:;\.!/*\+-]
55	ERR_CHAR	\'[^']*\'