João Batista Romão Damasceno

Manual de Uso SwiftLexical

Manual de uso do programaa SwiftLexical para o sistema Operacional Windows desenvolvido para a cadeira de compiladores do IFCE campus Maracanaú.

Fortaleza – fevereiro – 2021

1. Linguagem Swift

Swift é o resultado das pesquisas mais recentes em linguagens de programação, combinadas com décadas de experiência na construção de plataformas Apple. Os parâmetros nomeados são expressos em uma sintaxe limpa que torna as APIs no Swift ainda mais fáceis de ler e manter. Melhor ainda, você nem precisa digitar ponto-e-vírgula.

Os tipos inferidos tornam o código mais limpo e menos sujeito a erros, enquanto os módulos eliminam cabeçalhos e fornecem namespaces. Para melhor oferecer suporte a idiomas e emojis internacionais, as Strings são corretas para Unicode e usam uma codificação baseada em UTF-8 para otimizar o desempenho para uma ampla variedade de casos de uso. A memória é gerenciada automaticamente usando uma contagem de referência rígida e determinística, mantendo o uso de memória no mínimo sem a sobrecarga da coleta de lixo.

2. Vantagens da Linguagem

- Linguagem moderna
- Projetada para segurança
- Rápida e poderosa
- Boa para ser a primeira linguagem de programação aprendida
- Compatibilidade de fonte e binária
- Código open source
- Fácil gerenciamento de pacotes
- Compatibilidade com Objetive-C

3. Tabelas de Tokens

Nas tabelas abaixo é possivel vê as palavras aceitas e seus tokens. O token e o valor na coluna da esquerda e o valor que ele procura no aarquivo o da coluna direita.

3.1 Operadores Aritmeticos

Token	Valor
SOMA	+
SUBTRAÇÃO	-
MULTIPLICAÇÃO	*
DIVISÃO	/
MODULO	%

3.2 Operadores Comparação

Token	Valor
IGUALDADE	==
DESIGUALDADE	!=
IDENTIDADE	===
NAOIDENTIDADE	!===
MENOR	<

MENORIGUAL	<=
MAIOR	>
MAIORIGUAL	>=

3.3 Operadores Lógicos

Token	Valor
ANDBOOL	&&
ORBOOL	||
NOT	!

3.4 Operadores bit a bit

Token	Valor
BITAND	&
BITOR	|
BITXOR	^
BITCOMP	~
SHIFTL	<<
SHIFTR	>>

3.5 Operadores de atribuição

Token	Valor
ATRIBUICAO	=
PLUSEQ	+=
SUBEQ	-=
MULTEQ	*=
DIVEQ	/=
MODEQ	%=
SHIFTLEQ	<<=
SHIFTREQ	>>=
BITANDEQ	&=
OREQ	|=
RETORNOFUNCAO	->

3.6 Operadores de Intervalo

Token	Valor
INTERVALOFECHADO	...
INTERVALOSEMIABERTO	..<

3.7 Operadores de Comuns

Token	Valor
COLCHETEABERTO	[
COLCHETEFECHADO	]

CHAVEABERTA	{
CHAVEFECHADA	}
PARENTESEABERTO	(
PARENTESEFECHADO	)
PONTOVIRG	;
PONTO	.
VIRGULA	,
INTERROG	?
DOISPONTOS	:

3.8 Estruturas Condicionais

Token	Valor
IF	If
IFAVAILABLE	if #available
ELSE	else
ELSEIF	else if
SWITCH	switch

3.9 Estruturas de repetição

Token	Valor
FOR	for
REPEATWHILE	repeat {
WHILE	while

3.10 Sub rotinas

Token	Valor
PUBLICFUNC	public func
OPENFUNC	open func
INTERNALFUNC	internal func
FILEPRIVATEFUNC	fileprivate func
PRIVATEFUNC	private func
STATICFUNC	static func

3.11 Palavras reservadas

Token	Valor
ANY	any
ASSOCIATEDTYPE	associatedtype
BREAK	break
CASE	case
CATCH	catch
CLASS	class
CONTINUE	continue
DEFAULT	default

DEFER	defer
DEINIT	deinit
DO	do
ENUM	enum
EXTENSION	extension
FALLTHROUGH	fallthrough
FALSE	false
FILEPRIVATE	fileprivate
FUNC	func
GUARD	guard
IMPORT	import
IN	in
INIT	init
INOUT	inout
INTERNAL	internal
LET	let
NIL	nil
OPEN	open
OPERATOR	operator
PRINT	print
PRIVATE	private
PROTOCOL	protocol
PUBLIC	public
RETHROWS	rethrows
RETURN	return
SELF	self
STATIC	static
STRUCT	struct
SUPER	super
TRUE	true
TYPEALIAS	typealias
VAR	var
WHERE	where

3.12 Tipos comuns

Token	Valor
INT	Int
FLOAT	Float
DOUBLE	Double
STRING	String
CHAR	Char
BOOL	Bool

3.13 Identificadores para casos genericos

Token	Valor exemplo
IDENTIFICADOR	sum
VALOROPCIONAL	a? ou a!
INTEIRO	1,2,3
DECIMAL	1.5

CARACTERE	CARACTERE
ARRAY	[a,b]
MATRIX	[[a,b],[c,d]]
WhiteSpace	\r|\n|\r\n\t\f

4. Analisador Lexico

```
package codigo;
import static codigo.Tokens.*;
%%
%public
%class Analisador
%unicode
%line
%column
%type Tokens
%{
	StringBuffer string = new StringBuffer();
%}
LineTerminator = \r|\n|\r\n
InputCharacter = [^\r\n]
WhiteSpace     = {LineTerminator} | [ \t\f]
Alfa = [a-zA-Z]
AlfaNumerico = [a-zA-Z0-9]
Caractere = '[a-zA-Z]'
Identificador = ({Alfa}|_)({AlfaNumerico}|_)*
ValorOpcional = ({Alfa}|_)({AlfaNumerico}|_)*("!"|"?")?
Digito = [0-9]
Inteiro = {Digito}+("_"{Digito}+)*
Decimal = {Digito}+"."{Digito}+|"."{Digito}+
DecOrInt = {Decimal}|{Inteiro}
Array = \[(([a-zA-Z0-9]+\,[a-zA-Z0-9]+))*\]|\[{Identificador}*\]
```

```
Matrix = \[?({Array}+\,{Array}+)\]|\[{Array}*\]

RepeatWhile = repeat(" ")*\{

%state STRING

%state CHAR

%

<YYINITIAL> {

/* SIMBOLOS */

/* Operadores Aritméticos */

"+"                                                    { return new
Tokens("SOMA", yytext(), yyline, yycolumn, "Operador de soma"); }

"-"                                                    { return new
Tokens("SUBTRACAO", yytext(), yyline, yycolumn, "Operador de
subtração"); }

"*"                                                    { return new
Tokens("MULTIPLICACAO", yytext(), yyline, yycolumn, "Operador de
multiplicação"); }

"/"                                                    { return new
Tokens("DIVISAO", yytext(), yyline, yycolumn, "Operador de divisão"); }

"%"                                                    { return new
Tokens("MODULO", yytext(), yyline, yycolumn, "Operador de modulo,
retorna o resto de ums divisão"); }

/* Operadores Comparação */

"=="                                                   { return new
Tokens("IGUALDADE", yytext(), yyline, yycolumn, "Verifica se dois
operandos são iguais: (a == a) retorna true, (a == b) retorna false"); }

"!="                                                   { return new
Tokens("DESIGUALDADE", yytext(), yyline, yycolumn, "Verifica se dois
operandos são diferentes: (a == a) retorna false, (a == b) retorna true"); }

"==="                                                  { return new
Tokens("IDENTIDADE", yytext(), yyline, yycolumn, "Verifica se duas
instâncias de uma classe apontam para a mesma memoria:(classA == classA)
retorna true, (classA == classB) retorna false");}

"!=="                                                  { return new
Tokens("NAOIDENTIDADE", yytext(), yyline, yycolumn, "Verifica se duas
instâncias de uma classe não apontam para a mesma memoria:(classA ==
classA) retorna false, (classA == classB) retorna true");}
```

"<" { return new Tokens("MENOR", yytext(), yyline, yycolumn, "Retorna true se o operando da esquerda for menor que o da direita: a<b"); }

"<=" { return new Tokens("MENORIGUAL", yytext(), yyline, yycolumn, "Retorna true se o operando da esquerda for menor ou igual que o da direita: a<=b"); }

">" { return new Tokens("MAIOR", yytext(), yyline, yycolumn, "Retorna true se o operando da esquerda for maior que o da direita: b>a"); }

">=" { return new Tokens("MAIORIGUAL", yytext(), yyline, yycolumn, "Retorna true se o operando da esquerda for maior ou igual que o da direita: b>a"); }

/* Operadores Lógicos */

"&&" { return new Tokens("ANDBOOL", yytext(), yyline, yycolumn, "Operador lógico AND: (a==true && b==true) retorna true "); }

"||" { return new Tokens("ORBOOL", yytext(), yyline, yycolumn, "Operador lógico, retorna true se pelomenos um dos operandos forem true"); }

"!" { return new Tokens("NOT", yytext(), yyline, yycolumn, "Inverte o estado da lógica do operando: !true é equivalente a false"); }

/* Operadores bit a bit */

"&" { return new Tokens("BITAND", yytext(), yyline, yycolumn, "Operador AND binário: 1010 & 0011 = 0010"); }

"|" { return new Tokens("BITOR", yytext(), yyline, yycolumn, "Operador OR binário: 1010 | 0011 = 1011"); }

"^" { return new Tokens("BITXOR", yytext(), yyline, yycolumn, "Operador binário XOR: 1010 ^ 0011 = 1001"); }

"~" { return new Tokens("BITCOMP", yytext(), yyline, yycolumn, "Operador de complemento binário: ~10 retorna -11"); }

"<<" { return new Tokens("SHIFTL", yytext(), yyline, yycolumn, "Left Shift, faz o deslocamento binário a esquerda, ocorrendo uma multiplicação: 10<<1 == 20"); }

```
">>"                    { return new Tokens("SHIFTR", yytext(), yyline, yycolumn, "Left Right, faz o deslocamento binário a direita, ocorrendo uam divisão: 10>>1 == 5"); }


/* Operadores de atribuição */

"="                     { return new Tokens("ATRIBUICAO", yytext(), yyline, yycolumn, "Atribuição de valores"); }

"+="                    { return new Tokens("PLUSEQ", yytext(), yyline, yycolumn, "(a += b) equivalente a (a = a + b)"); }

"-="                    { return new Tokens("SUBEQ", yytext(), yyline, yycolumn, "(a -= b) equivalente a (a = a - b)"); }

"*="                    { return new Tokens("MULTEQ", yytext(), yyline, yycolumn, "(a *= b) equivalente a (a = a * b)"); }

"/="                    { return new Tokens("DIVEQ", yytext(), yyline, yycolumn, "(a /= b) equivalente a (a = a / b)"); }

"%="                    { return new Tokens("MODEQ", yytext(), yyline, yycolumn, "(a %= b) equivalente a (a = a % b)"); }

"<<="                   { return new Tokens("SHIFTLEQ", yytext(), yyline, yycolumn, "Left Shift com atribuição: A <<= B é equivalente a A = A << B"); }

">>="                   { return new Tokens("SHIFTREQ", yytext(), yyline, yycolumn, "Right Shift com atribuição: A >>= B é equivalente a A = A >> B"); }

"&="                    { return new Tokens("BITANDEQ", yytext(), yyline, yycolumn, "Operador AND binário com atribuição, sendo a=10 e b=3, a recebe o valor 2: a &= b, a == 2 "); }

"|="                    { return new Tokens("OREQ", yytext(), yyline, yycolumn, "Retorna e atribui true se pelomenos um dos operandos for true: A |= B é equivalente a A = A | B" ); }

"->"                    { return new Tokens("RETORNOFUNCAO", yytext(), yyline, yycolumn, "Precede o tipo de retorno de uma função func a () -> String" ); }


/* Operadores de intervalo */
```

```
"..."                                   { return new
Tokens("INTERVALOFECHADO", yytext(), yyline, yycolumn, "Retorna um
intervalo de valor fechado por exemplo 1..3 retorna os valores 1,2,3" ); }

"..<"                                   { return new
Tokens("INTERVALOSEMIABERTO", yytext(), yyline, yycolumn, "Retorna
um intervalo de valor semiaberto, pois não retorna o ultimo por exemplo 1..3
retorna os valores 1,2" ); }


/* Operadores Comuns */

"["                                     { return new
Tokens("COLCHETEABERTO", yytext(), yyline, yycolumn, "Usado em
Arrays: ary = [1, \"two\", 3.0]."); }

"]"                                     { return new
Tokens("COLCHETEFECHADO", yytext(), yyline, yycolumn, "Usado em
Arrays: ary = [1, \"two\", 3.0]."); }

"{"                                     { return new
Tokens("CHAVEABERTA", yytext(), yyline, yycolumn, "Usado para delimitar
Hash: h = { 7 => 35, \"c\" => 2, \"a\" => 1 }"); }

"}"                                     { return new
Tokens("CHAVEFECHADA", yytext(), yyline, yycolumn, "Usado para
delimitar Hash: h = { 7 => 35, \"c\" => 2, \"a\" => 1 }"); }

"("                                     { return new
Tokens("PARENTESEABERTO", yytext(), yyline, yycolumn, "Usado para
limitar um conjundo de dados: \"(1 + 2)*5\" ou \"for a in (1..6)\""); }

")"                                     { return new
Tokens("PARENTESEFECHADO", yytext(), yyline, yycolumn, "Usado para
limitar um conjundo de dados: \"(1 + 2)*5\" ou \"for a in (1..6)\""); }

";"                                     { return new
Tokens("PONTOVIRG", yytext(), yyline, yycolumn, "Delimitador de
instrução, pode ser omitido."); }

"."                                     { return new
Tokens("PONTO", yytext(), yyline, yycolumn, "Usado para trabalhar com
objetos: \"arr = Array.append(3)\" ou \"arr.remove(3)\"."); }

","                                     { return new
Tokens("VIRGULA", yytext(), yyline, yycolumn, "Usado para separar
identificadores e dados: \"ary = [1, \"two\", 3.0]\" ou \"Array.new(3, true)\"."); }

"?"                                     { return new
Tokens("INTERROG", yytext(), yyline, yycolumn, "Aplicado noperador
ternário ou valor opcional"); }
```

":" { return new Tokens("DOISPONTOS", yytext(), yyline, yycolumn,"Aplicado no operador ternário"); }

/* Estruturas condicionais */

"if" { return new Tokens("IF", yytext(), yyline, yycolumn, "Expressãos usadas para execução condicional. Os valores false e nil são falsos, e tudo o mais é verdade."); }

"if #available" { return new Tokens("IFAVAILABLE", yytext(), yyline, yycolumn, "Expressãos usadas para verificar versão do dispositivo."); }

"else" { return new Tokens("ELSE", yytext(), yyline, yycolumn, "Representa a execução de um comando quando a condição não é validada"); }

"else if" { return new Tokens("ELSEIF", yytext(), yyline, yycolumn, "Representa a execução de um comando quando a condição não é validada junto com outro condicional IF"); }

"switch" { return new Tokens("SWITCH", yytext(), yyline, yycolumn, "Considera um valor e o compara com vários padrões de correspondência possíveis"); }

/* Estruturas de repetição */

"for" { return new Tokens("FOR", yytext(), yyline, yycolumn, "Palavra para estrutura de repetição, Executa o corpo para cada elemento no resultado da expressão."); }

{RepeatWhile} { return new Tokens("REPEATWHILE", yytext(), yyline, yycolumn, "Define uma estrutura de repetição sem condição"); }

"while" { return new Tokens("WHILE", yytext(), yyline, yycolumn, "Executa o corpo enquanto a expressão de condição retorna verdadeira."); }

/* Sub rotinas */

"public func" { return new Tokens("PUBLICFUNC", yytext(), yyline, yycolumn, "Permitem que a função seja usada em qualquer arquivo de origem de seu módulo de definição e também em um arquivo de origem de outro módulo que importa o módulo de definição."); }

"open func" { return new Tokens("OPENFUNC", yytext(), yyline, yycolumn, "Permitem que a função seja usada em qualquer arquivo de

origem de seu módulo de definição e também em um arquivo de origem de outro módulo que importa o módulo de definição."); }

"internal func" { return new Tokens("INTERNALFUNC", yytext(), yyline, yycolumn, "Permite que função sejam usadas em qualquer arquivo de origem de seu módulo de definição, mas não em qualquer arquivo de origem fora desse módulo."); }

"fileprivate func" { return new Tokens("FILEPRIVATEFUNC", yytext(), yyline, yycolumn, "Permite que função sejam usadas em qualquer arquivo de origem de seu módulo de definição, mas não em qualquer arquivo de origem fora desse módulo."); }

"private func" { return new Tokens("PRIVATEFUNC", yytext(), yyline, yycolumn, "Restringe o uso da função à declaração anexa e às extensões dessa declaração que estão no mesmo arquivo."); }

"static func" { return new Tokens("STATICFUNC", yytext(), yyline, yycolumn, "Define a função que são chamados na porpria class. Também usado para definir membros estáticos."); }

/* Palavras reservadas */

"break" { return new Tokens("BREAK", yytext(), yyline, yycolumn, "Sai do loop mais interno. O break não sai da expressão case"); }

"case" { return new Tokens("CASE", yytext(), yyline, yycolumn, "As case expressões também são para execução condicional. sendo que suas comparações equivale ao mesmo que ==="); }

"class" { return new Tokens("CLASS", yytext(), yyline, yycolumn, "Define uma nova classe."); }

"do" { return new Tokens("DO", yytext(), yyline, yycolumn, "Usado em tratamento de erros"); }

"false" { return new Tokens("FALSE", yytext(), yyline, yycolumn, "A única instância da classe FalseClass (representa falso)"); }

"true" { return new Tokens("TRUE", yytext(), yyline, yycolumn, "A única instância da classe TrueClass (valor verdadeiro típico)"); }

"in" { return new Tokens("IN", yytext(), yyline, yycolumn, "Define um contado para uma estrutura de Repetição For"); }

"nil" { return new Tokens("NIL", yytext(), yyline, yycolumn, "É equivalente a Nulo.a única instância da Classe NilClass (representa falso)"); }

"return" { return new Tokens("RETURN", yytext(), yyline, yycolumn, "Sai do método com o valor de retorno."); }

"self" { return new Tokens("SELF", yytext(), yyline, yycolumn, "O receptor do método atual"); }

"super" { return new Tokens("SUPER", yytext(), yyline, yycolumn, "Chama o método que substitui o método atual"); }

"associatedtype" { return new Tokens("ASSOCIATEDTYPE", yytext(), yyline, yycolumn, "Dá um nome de espaço reservado para um tipo que é usado como parte de um protocolo"); }

"protocol" { return new Tokens("PROTOCOL", yytext(), yyline, yycolumn, "Equivalente ao conceito de interface em outras linguagens"); }

"init" { return new Tokens("INIT", yytext(), yyline, yycolumn, "São chamados para criar uma nova instância de um tipo específico"); }

"deinit" { return new Tokens("DEINIT", yytext(), yyline, yycolumn, "Chamado imediatamente antes de uma instância de classe ser desalocada"); }

"enum" { return new Tokens("ENUM", yytext(), yyline, yycolumn, "Define um tipo comum para um grupo de valores relacionados e permite que você trabalhe com esses valores de uma forma de segurança de tipo em seu código"); }

"extension" { return new Tokens("EXTENSION", yytext(), yyline, yycolumn, "Permite adicionar uma nova funcionalidade a uma classe, estrutura, enumeração ou tipo de protocolo existente"); }

"fileprivate" { return new Tokens("FILEPRIVATE", yytext(), yyline, yycolumn, "Uma construção de controle de acesso que restringe o escopo apenas ao arquivo de origem de definição"); }

"func" { return new Tokens("FUNC", yytext(), yyline, yycolumn, "Pedaços de código autocontidos que executam uma tarefa específica"); }

"import" { return new Tokens("IMPORT", yytext(), yyline, yycolumn, "Expõe uma estrutura ou aplicativo que é construído e enviado como uma única unidade no binário fornecido"); }

"inout" { return new Tokens("INOUT", yytext(), yyline, yycolumn, "Um valor que é passado para uma função e modificado por ela, e é passado de volta para fora da função para substituir o valor original"); }

"internal" { return new Tokens("INTERNAL", yytext(), yyline, yycolumn, "Permite que as entidades sejam usadas em qualquer arquivo

de origem de seu módulo de definição, mas não em qualquer arquivo de origem fora desse módulo"); }

"let" { return new Tokens("LET", yytext(), yyline, yycolumn, "Define uma variável como imutável"); }

"var" { return new Tokens("VAR", yytext(), yyline, yycolumn, "Define uma variável como variável"); }

"open" { return new Tokens("OPEN", yytext(), yyline, yycolumn, "Uma construção de controle de acesso que permite que os objetos sejam acessíveis e subclassíveis fora de seu módulo de definiçãoa"); }

"operator" { return new Tokens("OPERATOR", yytext(), yyline, yycolumn, "Um símbolo ou frase especial que você usa para verificar, alterar ou combinar valores"); }

"private" { return new Tokens("PRIVATE", yytext(), yyline, yycolumn, "Uma construção de controle de acesso que permite que as entidades tenham como escopo sua declaração de definição"); }

"public" { return new Tokens("PUBLIC", yytext(), yyline, yycolumn, "Permitem que as entidades sejam usadas em qualquer arquivo de origem de seu módulo de definição e também em um arquivo de origem de outro módulo que importa o módulo de definição."); }

"static" { return new Tokens("STATIC", yytext(), yyline, yycolumn, "Define métodos que são chamados no próprio tipo. Também usado para definir membros estáticos."); }

"struct" { return new Tokens("STRUCT", yytext(), yyline, yycolumn, "Uma construção flexível de uso geral que se torna os blocos de construção do código do seu programa e também pode fornecer inicializadores inteligentes de membros"); }

"typealias" { return new Tokens("TYPEALIAS", yytext(), yyline, yycolumn, "Introduz um alias nomeado de um tipo existente em seu programa"); }

"continue" { return new Tokens("CONTINUE", yytext(), yyline, yycolumn, "Termina a execução do programa da iteração atual de uma instrução de loop, mas não interrompe a execução da instrução de loop."); }

"default" { return new Tokens("DEFAULT", yytext(), yyline, yycolumn, "Usado para cobrir quaisquer valores que não sejam explicitamente endereçados em um caso"); }

"defer" { return new Tokens("DEFER", yytext(), yyline, yycolumn, "Usado para executar o código antes de transferir o controle do programa para fora do escopo em que aparece"); }

"fallthrough" { return new Tokens("FALLTHROUGH", yytext(), yyline, yycolumn, "Permite explicitamente que a execução continue de um caso para o outro em uma switch instrução"); }

"guard" { return new Tokens("GUARD", yytext(), yyline, yycolumn, "Usado para transferir o controle do programa para fora de um escopo se uma ou mais condições não forem atendidas, ao mesmo tempo que desembrulhará quaisquer valores opcionais fornecidos"); }

"repeat" { return new Tokens("REPEAT", yytext(), yyline, yycolumn, "Executa uma única passagem pelo bloco do loop primeiro, antes de considerar a condição do loop"); }

"where" { return new Tokens("WHERE", yytext(), yyline, yycolumn, "Requer que um tipo associado esteja em conformidade com um determinado protocolo ou que determinados parâmetros de tipo e tipos associados sejam os mesmos"); }

"Any" { return new Tokens("ANY", yytext(), yyline, yycolumn, "Pode ser usado para representar uma instância de qualquer tipo, incluindo tipos de função"); }

"rethrows" { return new Tokens("RETHROWS", yytext(), yyline, yycolumn, " Indica que a função gera um erro apenas se um de seus parâmetros de função gerar um erro"); }

"print" { return new Tokens("PRINT", yytext(), yyline, yycolumn, "Função utilizada para imprimir coisas no console da IDE exemplo print(2)"); }

"catch" { return new Tokens("CATCH", yytext(), yyline, yycolumn, "Usado em tratamento de erros"); }

/* Tipos Comuns */

"Int" { return new Tokens("Int", yytext(), yyline, yycolumn, "Define uma variavel ou retorno de uma função como Tipo Inteiro como 0,1,2,3 ..."); }

"Float" { return new Tokens("FLOAT", yytext(), yyline, yycolumn, "Define uma variavel ou retorno de uma função como Tipo decimal como 0.5, 1.325"); }

"Double" { return new Tokens("DOUBLE", yytext(), yyline, yycolumn, "Define uma variavel ou retorno de uma função como Tipo decimal com dobro de precisão de um float 0.58545854, 1.325"); }

"String" { return new Tokens("STRING", yytext(), yyline, yycolumn, "Define uma variavel ou retorno de uma função como Tipo String nome casa"); }

"Char" { return new Tokens("CHAR", yytext(), yyline, yycolumn, "Define uma variavel ou retorno de uma função como Tipo Caractere por exemplo a letra a, b ou c"); }

"Bool" { return new Tokens("BOOL", yytext(), yyline, yycolumn, "Define uma variavel ou retorno de uma função como Tipo Booleano podendo ser True ou False "); }

}

<YYINITIAL> {

/* IDENTIFICADORES */

{Identificador} { return new Tokens("IDENTIFICADOR", yytext(), yyline, yycolumn, "Identificador de métodos, variáveis, constantes, etc"); }

{ValorOpcional} { return new Tokens("VALOROPCIONAL", yytext(), yyline, yycolumn, "Identificador de métodos que utilizam ? para que pode ser null ou ! por que não e null"); }

/* NÚMEROS */

{DecOrInt}("e"|"E")("+"|"-")?{DecOrInt} { return new Tokens("NOTACAO_CIENTIFICA", yytext(), yyline, yycolumn, "Escrita de notação científica"); }

{Inteiro} { return new Tokens("INTEIRO", yytext(), yyline, yycolumn, "Valor de número inteiro."); }

{Decimal} { return new Tokens("DECIMAL", yytext(), yyline, yycolumn, "Valor de número decimal"); }

{Caractere} { return new Tokens("CARACTERE", yytext(), yyline, yycolumn, "Um caractere como as letras a,b,c"); }

{Array} { return new Tokens("ARRAY", yytext(), yyline, yycolumn, "Array de valores"); }

{Matrix} { return new Tokens("MATRIX", yytext(), yyline, yycolumn, "Matrix de valores"); }

\" {yybegin(STRING); string.setLength(0); }

/* COMENTÁRIO */

```
"/*"(.*{LineTerminator}*)*"*/" | "//"+.+ | "//" | "///" { /* ignore */ }

/* espaço em branco */

{WhiteSpace}                    { /* ignore */ }

}

/* ESTÁDO PARA TRATAR STRINGS */

<STRING> {

\"                                              {

                                                  yybegin(YYINITIAL);

                    return new Tokens

                ("STRING", string.toString(), yyline, yycolumn, "String,
sequência de caracteres.");

                                                }

/* CONDIÇÃO PARA TRATAR O CHAR */

/* [^\n\r]                                      { string.append( yytext() );
yybegin(CHAR); } */


[^\n\r\"\\]+                    { string.append( yytext() ); }

\\\\n                                   { string.append("\\n"); }

\\\\t                                   { string.append("\\t"); }

\\\\r                                   { string.append("\\r"); }

\\t                                             { string.append("\t"); }

\\n                                             { string.append("\n"); }

\\r                                             { string.append("\r"); }

\\\"                                    { string.append('\"'); }

\\\\                                    { string.append("\\"); }

}


/* Exceção caso entre caractére inválido */

[^]                                             {

                                                  throw new
RuntimeException("Caractere inválido " + yytext() + " na linha " + (yyline+1)
+ ", coluna " +(yycolumn+1));
```

}

5. Exemplos

5.1 Exemplo 1

```
public class Math {
      init() {}
      static func sum(a: Int, b: String) {
       return a + b
      }
}
```

5.1 Exemplo 2

```
import Foundation

struct Rule {
    var variable: LanguageElements
    var rules: [[LanguageElements]]
}
extension Rule: Hashable {
    public func hash(into hasher: inout Hasher) {
        hasher.combine(ObjectIdentifier(Rule.self).hashValue)
    }
    static func == (lhs: Rule, rhs: Rule) -> Bool {
        return lhs.variable == rhs.variable && lhs.rules == rhs.rules
    }
}
```

5.1 Exemplo 3

```
//
//  File.swift
//  DAWG_Teoria_da_computacao
//
//  Created by Joao Batista on 04/09/20.
//  Copyright © 2020 Joao Batista. All rights reserved.
//
import Foundation


class File {
```

```
    static let path = "/Users/ItSelf/Documents/Faculdade/Teoria da Computação/DAWG_Teoria_comp/DAWG_Teoria_da_computacao/DAWG_Teoria_da_computacao/Resource/"

    static func readCSVFile(withName fileName: String) -> String {

        //        Bundle.main.path(forResource: fileName, ofType: csv, inDirectory: Files.xcassets")

        let filepath = path+fileName+".csv"


        do {

            var contents = try String(contentsOfFile: filepath, encoding: .utf8)

            contents = cleanRows(file: contents)

            return contents

        } catch {

            print("File Read Error for file \(filepath)")

            return "Arquivo não encontrado"

        }

    }

    static func readFile(withName fileName: String, andFileType fileType: String) -> String {

        let filepath = path+fileName+"."+fileType

        //        guard let path = Bundle.main.path(forResource: fileName, ofType: fileType) else {return ""}

        do {

            let text = try String(contentsOfFile: filepath, encoding: .utf8)

            return text

        } catch {

            print("Arquivo não lozalizado")

            return ""

        }

    }

    static func cleanRows(file: String) -> String {

        var cleanFile = file
```

```
        cleanFile = cleanFile.replacingOccurrences(of: "\r", with: "\n")
        cleanFile = cleanFile.replacingOccurrences(of: "\n\n", with: "\n")
        return cleanFile
    }
    static func createArray(fromCSV csv: String) -> [[String]] {
        var result: [[String]] = []
        let rows = csv.components(separatedBy: "\n")
        for row in rows {
            let columns = row.components(separatedBy: ";")
            result.append(columns)
        }
        return result
    }

    static func getSPlusSMinus(fromFileString file: String) -> (sPlus: Set<String>,
sMinus: Set<String>) {
        var result: (sPlus: Set<String>, sMinus: Set<String>) = ([], [])
        let formatedString = file.replacingOccurrences(of: "\n", with: "")
        let strings = formatedString.components(separatedBy: "\r")

        // Separar entre S+ e S-
        var sPlus = strings.filter { self.isSPlus(theString: $0) == true }
        var sMinus = strings.filter { self.isSPlus(theString: $0) == false}

        // Remove strings desnecessarias
        sPlus = sPlus.map { $0.replacingOccurrences(of: "\t", with: "")}.map {
$0.replacingOccurrences(of: "+", with: "")}.filter {$0 != ""}
        sMinus = sMinus.map { $0.replacingOccurrences(of: "\t", with: "0")}.filter {$0 !=
""}
        result.sPlus = Set(sPlus)
        result.sMinus = Set(sMinus)
        return result
```

```
    }
    private static func isSPlus(theString string: String) -> Bool {
        if string.last == "+" {
            return true
        } else {
            return false
        }
    }
}
```

6 Alerta

Para que o programa funcione corretamente é necessario que o arquivo de entrada do analisador seja na extesão Swift. Demais linguagens podem funcionar, entretanto não de mandeira correta devido aos tokes especificos de cada linguagem.