BACHARELADO EM

CIÊNCIA DA COMPUTAÇÃO

JOÃO PEDRO SOARES DA FRANCA

LÉO MARTELLI

MARISTELA AYUMI RICARDO

WELINTON FERNANDO

CSS#

COMPILADORES

Presidente Epitácio – SP

2025

SUMÁRIO

1. INTRODUÇÃO

O projeto tem como objetivo o desenvolvimento de um analisador sintático para a linguagem CSS#, uma pseudo-linguagem criada com base em elementos estruturais do CSS tradicional, mas expandida com recursos de controle e atribuição semelhantes aos encontrados em linguagens de programação.

A linguagem CSS# define variáveis, expressões aritméticas e lógicas, atribuições e estruturas condicionais, permitindo a criação de códigos mais dinâmicos. Sua sintaxe inclui definições de variáveis no formato <var>:: valor, utilização de operadores (+, -, *, /, >, <, ==, !=, >=, <=), e comandos de controle como %if, %elseif, %else, %endif e %switch.

Além disso, a linguagem adota estruturas semelhantes ao CSS para definição de propriedades gráficas, abrangendo atributos como position, top, left, color, font-size, margin, padding, width, height, entre outros.

2. ESTRUTURAS DA LINGUAGEM

A linguagem **CSS#** é composta por estruturas que combinam conceitos de marcação e programação, permitindo tanto a definição de propriedades gráficas quanto o uso de expressões e controle de fluxo.

A seguir são apresentadas suas principais estruturas, conforme a especificação do projeto.

2.1. Definição de Variáveis

A definição de variáveis ocorre por meio da estrutura “variavel: valor”.

O nome da variável deve:

- Começar com uma letra (maiúscula ou minúscula),
- não conter caracteres especiais (exceto ‘_’).

O valor pode ser:

- Um número (ex.: 10),
- uma palavra (ex.: azul),
- ou conter um # para representar valores hexadecimais (ex.: #FF00FF).

2.2. Expressões

As expressões podem ser formadas por: números, variáveis, operações aritméticas (+, -, *, /) e operações lógicas (>, <, ==, !=, >=, <=). Exemplo: “x: 10 + 5” e “cond: x > 0”.

2.3. Atribuições

As atribuições seguem o formato: “variável: expressão”. Exemplo: “largura: 100 + 20”

2.4. Estruturas Condicionais

As estruturas condicionais são representadas por blocos delimitados por marcadores percentuais. A forma geral é:

%if <condição>

<código>

%elseif <condição>

<código>

%else

<código>

%endif

2.5. Estrutura de Seleção (Switch)

A estrutura %switch permite múltiplos casos com a opção de um bloco padrão (default):

%switch <expressão>

%case valor: <código>;

%case valor: <código>;

default: <código>;

%endswitch

2.6. Estruturas de Código e Atributos

O código de estilo é estruturado em blocos semelhantes ao CSS, utilizando tags e atributos. O formato geral é:

<tag_geral> {

<atributo>

}

- <tag_geral> pode conter prefixos como . (classe) ou # (id), e aceitar múltiplas tags separadas por vírgula.
- <atributo> representa pares nome–valor com propriedades de estilo.

2.7. Atributos Suportados

A linguagem define as seguintes propriedades: position, top, left, z-index, background-color, background-image, background-repeat, color, font-size, font-family, text-align, text-decoration, justify-content, margin, padding, border, width, height, display, cursor. Exemplo:

```
.divExemplo {
    color: #FF00FF;
    width: 200px;
    margin: 10px;
}
```

3. ANALISADOR LÉXICO

O analisador léxico da linguagem CSS# é responsável por identificar os tokens que compõem os elementos da linguagem. Esses tokens são definidos a partir de expressões regulares que representam números, letras, palavras, operadores e estruturas especiais.

A seguir estão os principais tokens e suas respectivas expressões regulares conforme a definição formal em EBNF.

Token	Descrição
<digito>	Dígito numérico de 0 a 9
<letra>	Letras e sublinhado
<palavra>	Sequência de letras e dígitos iniciando com letra
<var>	Declaração de variável
<expressao>	Combinação de valores, palavras, expressões lógicas ou aritméticas
<expressaoLogica>	Comparações lógicas
<expressaoAritmetica>	Operações aritméticas
<atribuicao>	Atribuição de valor a uma variável
<if>	Estrutura condicional
<switch>	Estrutura de múltipla escolha
<case_stmt>	Caso dentro de um switch
<codigo>	Bloco de código com tags e atributos
<tag_geral>	Identificador de seletor
<tag>	Elemento estrutural
<atributo>	Propriedade de estilo
<unidade_medida>	Unidade de medida de valores numéricos
<cor>	Representação de cor
<rgb_value>	Componente de cor RGB
<alpha_value>	Valor de transparência
<hex>	Valor hexadecimal de cor
<hex_value>	Dígitos hexadecimais

Esses tokens formam a base da análise léxica e são utilizados pelo analisador sintático na identificação das regras da gramática EBNF que descrevem a estrutura formal da linguagem CSS#.

4. EBNF

A gramática da linguagem CSS# é definida em EBNF (Extended Backus–Naur Form), descrevendo de maneira formal as regras sintáticas que compõem o analisador.

As produções abaixo especificam as estruturas básicas da linguagem, seguidas por exemplos que ilustram seu uso prático.

4.1 Regras da Linguagem (EBNF)

<digito> ::= ("0" | "1" | ... | "9")

<letra> ::= ("a" | ... | "z" | "A" | ... | "Z" | "_")

<palavra> ::= <letra> {<letra> | <digito>}

<string> ::= “ ‘ “ <palavra> “ ‘ “

<var> ::= ">>" <palavra> ":" (<cor> | <digito> {<digito>} <unidade_medida> | <string>)

<expressao> ::= (<digito> | " ' " <palavra> " ' " | <expressaoLogica> | <expressaoAritmetica>)

<expressaoLogica> ::= <expressao> (">" | "<" | "==" | "!=" | ">=" | "<=") <expressao>

<expressaoAritmetica> ::= <expressao> ("+" | "-" | "*" | "/") <expressao>

<atribuicao> ::= <palavra> ":" <expressao>

<if> ::= "%if" <if_stmt> "%endif"

<if_stmt> ::= <expressao> <codigo> [("%elseif" <if_stmt> | "%else") <codigo>]

<switch> ::= "%switch" <expressao> <case_stmt> ["default:" <codigo>] "%endswitch"

<case_stmt> ::= "%case" <var> ":" <codigo> [<case_stmt>]

<codigo> ::= <tag_geral> "{" <atributo> "}"

<tag_geral> ::= ["." | "#"] <tag> { [","] ["." | "#"] <tag> }

<tag> ::= ("div" | "h1" | "body" | ...)

<atributo> ::= (

("top" | "left" | "width" | "height" | "font-size" | "margin" | "padding") ":" <digito> {<digito>} <unidade_medida>

| "color" ":" <cor>

)

<unidade_medida> ::= ("px" | "vw" | "vh" | "%" | "pt" | "rem" | "em")

<cor> ::= (

"rgb(" <rgb_value> "," <rgb_value> "," <rgb_value> ")"

| "rgba(" <rgb_value> "," <rgb_value> "," <rgb_value> "," <alpha_value> ")"

| <hex>

)

<alpha_value> ::= ("0.0" | … | "1.0")

<rgb_value> ::= ("0" | "1" | ... | "255")

<hex> ::= "#" <hex_value>

<hex_value> ::= (("A" | … | "F") | <digito>) (repetido 6 vezes)

4.2 Exemplos Práticos

- Definição de variável: >>corPrincipal: #FF00FF
- Expressão e Atribuição: largura: 200 + 50
- Estrutura Condicional:

```
%if largura > 100
    .divPrincipal { color: #00FF00 }
%else
    .divPrincipal { color: #FF0000 }
%endif
```

- Estrutura Switch:

```
%switch tema
    %case >>escuro: body { background-color: #000000 }
    %case >>claro: body { background-color: #FFFFFF }
    default: body { background-color: #CCCCCC }
%endswitch
```

- Bloco de código padrão com atributos:

```
#container {
    width: 500px
    height: 300px
    color: rgb(255, 255, 255)
}
```

5. CONJUNTO DE FIRST E FOLLOW

Os conjuntos FIRST e FOLLOW da linguagem CSS# definem, respectivamente, os símbolos que podem iniciar e suceder uma determinada produção. Eles são fundamentais para a construção da tabela preditiva do analisador sintático LL(1).

5.1 Conjuntos FIRST

Não-Terminal	FIRST
<digito>	{ "0", "1", …, "9" }
<letra>	{ "a", …, "z", "A", …, "Z", "_" }
<palavra>	{ "a", …, "z", "A", …, "Z", "_" }
<var>	{ ">>" }
<expressao>	{ "0", …, "9", "a", …, "z", "A", …, "Z", "_" }
<expressaoLogica>	{ "0", …, "9", "a", …, "z", "A", …, "Z", "_" }
<expressaoAritmetica>	{ "0", …, "9", "a", …, "z", "A", …, "Z", "_" }
<atribuicao>	{ "a", …, "z", "A", …, "Z", "_" }
<if>	{ "%if" }
<if_stmt>	{ "0", …, "9", "a", …, "z", "A", …, "Z", "_" }
<switch>	{ "%switch" }
<case_stmt>	{ "%case" }
<tag_geral>	{ ".", "#" }
<codigo>	{ ".", "#" }
<tag>	{ "div", "h1", "body", … }
<atributo>	{ "top", "left", "width", "height", "font-size", "margin", "padding", "color" }
<unidade_medida>	{ "px", "vw", "vh", "%", "pt", "rem", "em" }
<cor>	{ "rgb(", "rgba(", "#" }
<alpha_value>	{ "0.0", …, "1.0" }
<rgb_value>	{ "0", …, "255" }
<hex>	{ "#" }
<hex_value>	{ "A", …, "F", "0", …, "9" }

5.2 Conjuntos FOLLOW

Não-Terminal	FOLLOW
<digito>	{ "px", "vw", "vh", "%", "pt", "rem", "em", "a", …, "z", "A", …, "Z", "_", "$" }
<letra>	{ "0", "1", …, "9", "$" }
<palavra>	{ ":", " ' " }
<var>	{ ":" }
<expressao>	{ ">", "<", "==", "!=", ">=", "<=", "+", "-", "*", "/", ".", "#", "%case", "$" }
<expressaoLogica>	{ "$" }
<expressaoAritmetica>	{ "$" }
<atribuicao>	{ ∅ }
<if>	{ ∅ }
<if_stmt>	{ "%endif", "$" }
<switch>	{ ∅ }
<case_stmt>	{ "default:", "%endswitch", "$" }
<codigo>	{ "%elseif", "%endswitch", "%case", "$" }
<tag_geral>	{ "{" }
<tag>	{ ",", "$" }
<atributo>	{ "}" }
<unidade_medida>	{ ";" }
<cor>	{ ";" }
<rgb_value>	{ ",", ")" }
<alpha_value>	{ ")" }
<hex>	{ "$" }

<hex_value>	{ "$" }

6. TABELA PREDITIVA

	digito	letra	>>	%if	%switch	%case	.	#	color	px	rgb(	rgba(	$
<digito>	→ ("0" | "1" | ... | "9")												
<letra>		→ ("a" | ... | "z" | "A" | ... | "Z" | "_")											
<palavra>		→ <letra> { <letra> | <digito> }											
<var>			→ ">>" <palavra> ":" ["#"] (<letra> | <digito>){<letra> | <digito>}										
<expressao>	→ (<digito> | ' <palavra> ' | <expressaoLogica> | <expressaoAritmetica>)	→ (<digito> | ' <palavra> ' | <expressaoLogica> | <expressaoAritmetica>)						→ (<digito> | ' <palavra> ' | <expressaoLogica> | <expressaoAritmetica>)					
<expressaoLogica>	→ <expressao> (">" | "<" | "==" | "!=" | ">=" | "<=") <expressao>	→ <expressao> (">" | "<" | "==" | "!=" | ">=" | "<=") <expressao>											
<expressaoAritimetica>	→ <expressao> ("+" | "-" | "*" | "/") <expressao>	→ <expressao> ("+" | "-" | "*" | "/") <expressao>											
<atribuicao>		→ <palavra> ":" <expressao>											
<if>				→ "%if" <if_stmt> "%endif"									
<if_stmt>	→ <expressao> <codigo> [("%elseif" <if_stmt> | "%else") <codigo>]	→ <expressao> <codigo> [("%elseif" <if_stmt> | "%else") <codigo>]											
<switch>					→ "%switch" <expressao> <case_stmt> ["default:" <codigo>] "%endswitch"								
<case_stmt>						→ "%case" <var> ":" <codigo> [<case_stmt>]							
<codigo>							→ <tag_geral> "{" <atributo> "}"	→ <tag_geral> "{" <atributo> "}"					
<tag_geral>							→ ["." | "#"] <tag> { [","] ["." | "#"] <tag> }	→ ["." | "#"] <tag> { [","] ["." | "#"] <tag> }					
<tag>		→ ("div" | "h1" | "body" | ...)											
<atributo>									→ "color" ":" <cor>	→ ("top" | "left" | "width" | "height" | "font-size" | "margin" | "padding") ":" <digito>{<digito>}<unidade_medida>			
<unidade_medida>										→ ("px" | "vw" | "vh" | "%" | "pt" | "rem" | "em")			
<cor>								→ <hex>			→ "rgb(" <rgb_value> "," <rgb_value> "," <rgb_value> ")"	→ "rgba(" <rgb_value> "," <rgb_value> "," <rgb_value> "," <alpha_value> ")"	
<alpha_value>													→ ("0.0" | ... | "1.0")
<rgb_value>	→ ("0" | "1" | ... | "255")												
<hex>								→ "#" <hex_value>					
<hex_value>								→ (("A" | ... | "F") | <digito>) ×6					