Universidade do Minho

Mestrado Integrado em Engenharia Informática

Processamento de Linguagens

Trabalho Prático nº 2 (FLex)

A75377 Bruno Magalhães
A79021 Diogo Silva

6 de Maio de 2018

Resumo

O seguinte documento relata a resolução do segundo trabalho prático da unidade curricular de Processamento de Linguagens da Universidade do Minho. Irá ser exposto o contexto do exercício e das decisões tomadas para a sua solução e implementação.

Conteúdo

1 Introdução

Este exercício pretende a familiarização com o Flex. O Flex é uma ferramenta que gera processadores de texto capazes de encontrar padrões lexicais em ficheiros de texto.

Para identificar qual dos problemas deveria ser resolvido pelo grupo de trabalho, foi utilizada a fórmula mostrada no exercício anterior. Essa fórmula consiste em calcular o *mod* do menor número de aluno do grupo por 5 e somar 1. Tal como no exercício anterior, o menor número de aluno do grupo é 75377 e, aplicando a fórmula, o resultado obtido é 3. Assim sendo, o problema que servirá de nosso caso de estudo é o apresentado na secção 2.3 do enunciado fornecido pelos docentes.

1.1 Identificação do caso de estudo

O diverso número de marcas (*tags*) que têm de ser inseridas durante o processo de desenvolvimento de um documento HTML faz com que este se possa tornar fastidioso. Para acelerar este processo, é possível recorrer a editores inteligentes que gerem a inserção de marcas de forma automática. Um outro método muito conhecido, é a utilização de uma linguagem de abreviaturas que facilitem a escrita do documento para, mais tarde, recorrer ao pré-processamento deste para substituir as abreviaturas pelas marcas corretas. Os sistemas que utilizam este último método são os chamados de sistemas **Wiki**.

1.2 Objetivo

O objetivo deste exercício é desenvolver uma linguagem de anotação Wiki e o respetivo pré-processador de texto, em Flex, que seja capaz de gerar documentos HTML a partir dessa mesma linguagem. A linguagem de anotação deve conter abreviaturas que facilitem a escrita de formatação tal como:

- Negrito;
- Itálico;
- Sublinhado;
- Vários níveis de títulos;
- Listas de itens ordenados e não ordenados;
- Termos e descrições;

Para além dos itens listados em cima, a linguagem pode conter tipos de abreviaturas com funcionalidades adicionais.

1.3 Estrutura do Documento

Na próxima secção é apresentada uma solução aos casos definidos, isto é, a anotação da linguagem definida por nós assim como a explicação da implementação do processador textual. Progressivamente, será construído um ficheiro de rascunho que será utilizado como input para o processador final. Durante a construção desse ficheiro é explicado como o Flex trata o input e como ele produz o resultado final.

Na secção seguinte encontram-se as conclusões obtidas com a resolução deste exercício.

Por fim, encontram-se os apêndices com o código fonte elaborado para ambos os casos assim como as referências bibliográficas.

2 Implementação de soluções

O primeiro passo foi definir a linguagem *Mark-up* que será reconhecida pelo pré-processador. Serviu de inspiração a linguagem *Wikitext*, utilizada pelo software *MediaWiki*.

Mark-up	**Utilidade**
... Título	Dá início ao documento com o título definido
\-texto-\	Negrito
\~texto~\	Itálico
_texto_\	Sublinhado
\[1-6]- texto	Inserir um título de secção
\+ texto	Dar início a um parágrafo
“ texto ”	Inserir uma citação.
#	Lista ordenada
*	Lista não ordenada
;	Inserir termo de uma lista de descrições
:	Inserir descrições de uma lista de descrições
\“url”	Insere uma imagem

Tabela 1: Definição da linguagem Wiki.

O Flex é capaz de encontrar padrões textuais. Para tal, deve-se descrever um conjunto de regras que são compostas por padrões e por ações. Sempre que o processador gerado pelo Flex identifique um dos padrões designados, vai desencadear uma ou mais ações. Se encontrar mais do que uma correspondência, considera aquela que inclua mais texto correspondido. É também possível definir estados que permitem que um conjunto de regras só seja ativado quando o processador se encontrar nesse estado. Estes estados podem ser inclusivos ou exclusivos. Para os primeiros, o processador tenta corresponder com as regras definidas para o estado mas também com todas as outras. No caso dos estados exclusivos, o processador apenas tenta encontrar correspondência com as regras referentes ao estado. Com recurso a estas funcionalidades, implementamos o pré-processador.

2.1 Início de documento

Todo o documento está inserido dentro de uma tag *html* e possuí um cabeçalho e um corpo. É dentro do cabeçalho que são definidos parâmetros como, por exemplo, o *design* ou o título. O nosso pré-processador não abrange funcionalidades de formatação do documento mas é possível definir um título juntamente com a abreviatura de início de documento. No Flex, a regra que trata esta abreviatura tenta encontrar um padrão que, no início da linha, contenha exatamente três pontos seguidos de um espaço e texto alfa-numérico combinado com alguns símbolos. A regra encontra-se definida da seguinte forma:

```
1  TEXT      [a-zA-Z0-9 #@?\~_.-]
2
3  ^"..."\ {TEXT}*?                    { if(title_done == 0){
4                                          strcpy(title, yytext);
5                                          printHeader(title);
6                                          title_done = 1;
7                                        } else
8                                            ECHO;
9                                      }
```

A variável **title_done** existe para ser possível saber se o título já foi imprimido e evitar ser imprimido várias vezes durante o documento. A função **printHeader** é responsável por imprimir o título e todas as outras marcas necessárias.

2.2 Títulos de secção

Os títulos de secção podem conter vários tamanhos. Em HTML, definimos esses títulos com as marcas h1, h2, h3, etc. Para a implementação desta abreviatura, já foi necessário recorrer a um estado exclusivo. Assim que o processador encontra o padrão definido, imprime a marca de abertura do título desejado e entra num estado a que chamamos de **HEADER**. Nesse estado, a única regra existente tem a finalidade de saber quando terminar o título. Definimos que isso acontece quando o *scanner* apanha um *new line*. Assim que encontra este padrão, imprime a marca de término do título. Dentro do estado, para os restantes *tokens*, o Flex aplica a regra *default* que é apenas imprimir esse mesmo *token*. Este procedimento está implementado nas seguintes regras.

```
1  \\[1-6]-\                          { open_header = yytext[1] - '0';
2                                       fprintf(yyout,"<h%d>", open_header);
3                                       BEGIN HEADER;
4                                     }
5
6  <HEADER>{
7    \n                               { fprintf(yyout, "</h%d>\n",
     ↪  open_header);
8                                       open_header=0;
9                                       BEGIN INITIAL;
10                                    }
11 }
```

Para este estado, estão também definidas regras que permitem a formatação de texto (negrito, itálico e sublinhado). Estas regras serão explicadas mais à frente.

2.3 Parágrafos

Os parágrafos encontram-se delimitados pelas marcas "p" em HTML. Para a nossa linguagem, estes começam quando é feita correspondência com o padrão \+ seguido de um espaço no início da linha. A implementação no processador é idêntica à dos títulos.

```
1  \\+\                               {fprintf(yyout, "<p>");BEGIN PAR;}
2  <PAR>{
3    \n                               {fprintf(yyout, "</p>\n"); BEGIN
     ↪  INITIAL;}
4  }
```

2.4 Citações

O processador pode encontrar citações estando no estado inicial ou a percorrer um parágrafo. Neste último caso, encontramos um problema ao terminar a citação, pois teríamos de voltar ao estado de parágrafo e não ao inicial. Para resolver esse problema, recorremos à pilha de estados onde, no caso de ser encontrada a abreviatura que delimitava o início de uma citação, é alterado o estado do processador através da função **yy_push_state(QUOTE)** que insere no topo da pilha o estado **QUOTE**. Já nesse estado, o processador só reconhece o padrão que delimita o fim da citação e, quando esse é correspondido, retira o estado do topo da pilha com a função **yy_pop_state()** e volta ao estado em que se encontrava.

```
<INITIAL,PAR>"``"                {fprintf(yyout, "\"");yy_push_state(QUOTE);}
<QUOTE,PAR>{
  "''"                           {fprintf(yyout, "\"");yy_pop_state();}
}
```

2.5 Formatação do texto

Para implementar a formatação de texto, utilizamos um processo muito simples. Visto que as abreviaturas definidas utilizam marcas que delimitam o início e fim da formatação, apenas substituímos essas abreviaturas pelas marcas finais.

```
<INITIAL,PAR,ULIST,OLIST,HEADER>"\\-"        {fprintf(yyout,"<b>");}
<INITIAL,PAR,ULIST,OLIST,HEADER>"-\\"        {fprintf(yyout,"</b>");}

<INITIAL,PAR,ULIST,OLIST,HEADER>"\\~"        {fprintf(yyout,"<i>");}
<INITIAL,PAR,ULIST,OLIST,HEADER>"~\\"        {fprintf(yyout,"</i>");}

<INITIAL,PAR,ULIST,OLIST,HEADER>"\\_"        {fprintf(yyout,"<u>");}
<INITIAL,PAR,ULIST,OLIST,HEADER>"_\\"        {fprintf(yyout,"</u>");}
```

Como é possível observar, as regras de formatação podem ser correspondidas em quase todos os estados do processador. Poderíamos ter utilizado estados para implementar estas funcionalidades, mas achamos que esse método seria desnecessário uma vez que dentro do estado apenas teríamos a regra a definir o término da formatação.

2.6 Lista ordenada

Tal como na linguagem *Wikitext*, as listas ordenadas são representadas com o símbolo "#" seguido de um espaço. Estando no estado inicial, a primeira abreviatura a aparecer indica ao processador que deve iniciar uma listagem, ou seja, imprime as marcas de início de lista ordenada e imprime ainda o primeiro item. As seguintes abreviaturas marcam apenas itens. Este processo é obtido recorrendo a um estado **OLIST** exclusivo, o qual o processador entra quando encontra uma abreviatura de lista. De forma a inserir as marcas de fim de lista, o processador deve encontrar uma linha em branco, ou seja, dois caracteres *new line*, estando já no estado da lista.

```
#\                                { fprintf(yyout,"<ol>\n\t<li>");
                                    ol_count++;
                                    BEGIN OLIST;
                                  }
<OLIST>{
  #\                              { if(ol_count == 2)
                                    fprintf(yyout,"</ol>\n\t<li>");
                                  else
                                    fprintf(yyout, "\t<li>");
                                  }
  ##\                             { if(ol_count == 1){
                                    fprintf(yyout,"\t<ol
                                     ↪  type=\"a\">\n\t\t<li>");
                                    ol_count++;
                                  }else if(ol_count == 3){
                                    fprintf(yyout,
                                     ↪  "\t\t</ol>\n\t\t<li>");
```

```
                                        ol_count--;
                                      }else
                                        fprintf(yyout, "\t\t<li>");
                                    }
  ###\                              { if(ol_count == 2){
                                        fprintf(yyout,"\t\t<ol
                                         ↪  type=\"i\">\n\t\t\t<li>");
                                        ol_count++;
                                      }else
                                        fprintf(yyout, "\t\t\t<li>");
                                    }
  \n                                { fprintf(yyout, "</li>\n");}
  \n\n                              { fprintf(yyout, "</li>\n");
                                      for(int i = 0; i < ol_count; i++)
                                        fprintf(yyout,"</ol>\n");
                                      ol_count--; BEGIN INITIAL;
                                    }
}
```

É possível utilizar listas aninhadas. A flag **ol_count** permite isso, condicionando o fluxo de forma a que o processador identifique se é a primeira vez que encontra uma abreviatura ou não. Por exemplo, se tivermos 3 itens numa lista de nível 1 e logo a seguir inserirmos um item de nivel 2, a flag vai identificar que o processador vinha do nível 1 e insere novas marcas de abertura de uma lista ordenada.

```
# item 1, nivel 1
# item 2, nivel 1
# item 3, nivel 1
## item 4, nivel 2
### item 5, nivel 3
# item 6, nivel 1
```

A lista apresentada em cima, é representada, em HTML, da seguinte forma:

```
<ol>
        <li>item 1, nivel 1</li>
        <li>item 2, nivel 1</li>
        <li>item 3, nivel 1</li>
        <ol type="a">
                <li>item 4, nivel 2</li>
                <ol type="i">
                        <li>item 5, nivel 3</li>
                </ol>
        </ol>
        <li>item 6, nivel 1</li>
</ol>
```

Os itens da lista podem conter formatação textual.

2.7 Listas não ordenadas

As listas não ordenadas são abreviadas com o símbolo "*". Estas funcionam exatamente da mesma forma que as listas não ordenadas, pelo que a sua implementação é idêntica.

2.8 Listas de descrições

As listas de descrições permitem associar a um determinado termo uma ou mais descrições. Para isso, e tal como nas listas implementadas anteriormente, o primeiro símbolo indica ao processador que este deve inserir as marcas de início de uma lista de descrição assim como o termo correspondente. Para além disso, o processador entra num estado chamado **DICT** onde apenas reconhece o símbolo que representa as descrições, uma regra que descarta as quebras de linha e outra que identifica quando terminar a lista.

```
;{TEXT}+                          { fprintf(yyout,
↪  "<dl>\n\t<dt><b>%s</b></dt>\n", yytext+2);
                                    BEGIN DICT;
                                  }
<DICT>{
  :{TEXT}+                        {fprintf(yyout, "\t<dd>%s</dd>\n", yytext+2);}
  \n                              /*Discarda os line breaks*/
  \n\n                            {fprintf(yyout, "</dl>\n"); BEGIN INITIAL;}
}
```

2.9 Imagens

A inserção de imagens é obtida através da abreviatura \"url", onde *url* é a localização da imagem. Esta abreviatura torna a implementação mais fácil, sendo apenas necessário inserir, na marca *html*, o *url*.

```
\\\"[^\"]*\"                      {fprintf(yyout, "<img src=%s>\n", yytext+1);}
```

2.10 Fim de documento

Assim que o processador encontre o fim do ficheiro de input, deve imprimir as marcas de fim de corpo e fim de documento. Para o caso em que isso acontece estando o processador no estado inicial, basta imprimir essas marcas e dar o processo por terminado. No entanto, no caso em que o processador se encontre num dos estados de uma lista, este deve inserir também as respetivas marcas de fim de lista. Para resolver esse problema, foram definidas regras para os vários casos em que o processador se encontra no fim do ficheiro.

```
<OLIST><<EOF>>                    { for(int i = 0; i < ol_count; i++)
                                      fprintf(yyout,"</ol>\n");
                                    ol_count--; BEGIN INITIAL;
                                    fprintf(yyout, "\t</body>\n</html>");
                                    yyterminate();
                                  }
<ULIST><<EOF>>                    { for(int i = 0; i < ul_count; i++)
                                      fprintf(yyout, "</ul>\n");
                                    ul_count=0; BEGIN INITIAL;
                                    fprintf(yyout, "\t</body>\n</html>");
                                    yyterminate();
                                  }
<DICT><<EOF>>                     { fprintf(yyout, "</dl>\n");
                                    fprintf(yyout, "\t</body>\n</html>");
                                    yyterminate();
```

```
16                                          }
17  <<EOF>>                                 {fprintf(yyout,
    ↪  "\t</body>\n</html>");yyterminate();}
```

3 Conclusões

A ferramenta Flex permitiu que o desenvolvimento de um documento em HTML se tornasse mais fácil e menos fastidioso. Com anotação da linguagem mais simples e recorrendo ao pré-processamento é possível gerar o mesmo documento caso fosse desenvolvido em HTML.

4 Apêndices

4.1 Código fonte do pré-processador para HTML

```
%option noyywrap stack
%{
#include <stdio.h>
#include <string.h>

void printHeader(char*);

int open_header = 0;
int ol_count = 0;
int ul_count = 0;
char title[256];
int title_done = 0;
%}

TEXT      [a-zA-Z0-9 #@?\~_.-]

%x QUOTE PAR HEADER OLIST ULIST DICT
%%

^"..."\ {TEXT}*?                    { if(title_done == 0){
                                        strcpy(title, yytext);
                                        printHeader(title);
                                        title_done = 1;
                                      } else
                                          ECHO;
                                    }
\\\"[^\"]*\"                        {fprintf(yyout, "<img src=%s>\n",
 ↪  yytext+1);}
.                                   ECHO;
\n                                  /*Descarta os new lines*/

<INITIAL,PAR>"``"                   {fprintf(yyout,
 ↪  "\"");yy_push_state(QUOTE);}
<QUOTE,PAR>{
  "''"                              {fprintf(yyout, "\"");yy_pop_state();}
}

\\\+\                               {fprintf(yyout, "<p>");BEGIN PAR;}
<PAR>{
  \n                                {fprintf(yyout, "</p>\n"); BEGIN
   ↪  INITIAL;}
}

\\[1-6]-\                           { open_header = yytext[1] - '0';
                                      fprintf(yyout,"<h%d>", open_header);
                                      BEGIN HEADER;
                                    }

<HEADER>{
```

```
47    \n                                    { fprintf(yyout, "</h%d>\n",
      ↪  open_header);
48                                          open_header=0;
49                                          BEGIN INITIAL;
50                                        }
51  }
52
53  <INITIAL,PAR,ULIST,OLIST,HEADER>"\\-"       {fprintf(yyout,"<b>");}
54  <INITIAL,PAR,ULIST,OLIST,HEADER>"-\\"       {fprintf(yyout,"</b>");}
55
56  <INITIAL,PAR,ULIST,OLIST,HEADER>"\\~"       {fprintf(yyout,"<i>");}
57  <INITIAL,PAR,ULIST,OLIST,HEADER>"~\\"       {fprintf(yyout,"</i>");}
58
59  <INITIAL,PAR,ULIST,OLIST,HEADER>"\\_"       {fprintf(yyout,"<u>");}
60  <INITIAL,PAR,ULIST,OLIST,HEADER>"_\\"       {fprintf(yyout,"</u>");}
61
62  #\                                    { fprintf(yyout,"<ol>\n\t<li>");
63                                          ol_count++;
64                                          BEGIN OLIST;
65                                        }
66  <OLIST>{
67    #\                                  { if(ol_count == 2)
68                                            fprintf(yyout,"</ol>\n\t<li>");
69                                          else if(ol_count == 3){
70                                            fprintf(yyout,
                                            ↪  "\t\t</ol>\n\t</ol>\n\t<li>");
71                                            ol_count -= 2;
72                                          }else
73                                            fprintf(yyout, "\t<li>");
74                                        }
75    ##\                                 { if(ol_count == 1){
76                                            fprintf(yyout,"\t<ol
                                            ↪  type=\"a\">\n\t\t<li>");
77                                            ol_count++;
78                                          }else if(ol_count == 3){
79                                            fprintf(yyout,
                                            ↪  "\t\t</ol>\n\t\t<li>");
80                                            ol_count--;
81                                          }else
82                                            fprintf(yyout, "\t\t<li>");
83                                        }
84    ###\                                { if(ol_count == 2){
85                                            fprintf(yyout,"\t\t<ol
                                            ↪  type=\"i\">\n\t\t\t<li>");
86                                            ol_count++;
87                                          }else
88                                            fprintf(yyout, "\t\t\t<li>");
89                                        }
90    \n                                  { fprintf(yyout, "</li>\n");}
91    \n\n                                { fprintf(yyout, "</li>\n");
92                                          for(int i = 0; i < ol_count; i++)
93                                            fprintf(yyout,"</ol>\n");
94                                          ol_count--; BEGIN INITIAL;
```

```
95                                      }
96  }
97
98  \*\                                 { fprintf(yyout,"<ul>\n\t<li>");
99                                        BEGIN ULIST;
100                                       ul_count++;
101                                     }
102 <ULIST>{
103   \*\                               { if(ul_count == 2){
104                                         fprintf(yyout, "\t</ul>\n\t<li>");
105                                         ul_count--;
106                                       }else
107                                         fprintf(yyout, "\t<li>");
108                                     }
109   \*\*\                             { if(ul_count == 1){
110                                         fprintf(yyout, "\t<ul>\n\t\t<li>");
111                                         ul_count++;
112                                     }else if(ul_count == 3){
113                                       fprintf(yyout, "\t\t</ul>\n\t\t<li>");
114                                       ul_count--;
115                                     }else
116                                       fprintf(yyout, "\t\t<li>");
117                                     }
118   \*\*\*\                           { if(ul_count == 2){
119                                         fprintf(yyout,
                                          ↪  "\t\t<ul>\n\t\t\t<li>");
120                                         ul_count++;
121                                       }else
122                                         fprintf(yyout, "\t\t\t<li>");
123                                     }
124   \n                                { fprintf(yyout, "</li>\n");}
125   \n\n                              { fprintf(yyout, "</li>\n");
126                                       for(int i = 0; i < ul_count; i++)
127                                         fprintf(yyout, "</ul>\n");
128                                       ul_count=0; BEGIN INITIAL;
129                                     }
130 }
131
132 ;{TEXT}+                            { fprintf(yyout,
    ↪  "<dl>\n\t<dt><b>%s</b></dt>\n", yytext+2);
133                                       BEGIN DICT;
134                                     }
135 <DICT>{
136   :{TEXT}+                          {fprintf(yyout, "\t<dd>%s</dd>\n",
    ↪  yytext+2);}
137   \n                                /*Descarta os line breaks*/
138   \n\n                              {fprintf(yyout, "</dl>\n"); BEGIN
    ↪  INITIAL;}
139 }
140
141 <OLIST><<EOF>>                      { for(int i = 0; i < ol_count; i++)
142                                         fprintf(yyout,"</ol>\n");
143                                       ol_count--; BEGIN INITIAL;
144                                       fprintf(yyout, "\t</body>\n</html>");
```

```
                                        yyterminate();
                                      }
<ULIST><<EOF>>                        { for(int i = 0; i < ul_count; i++)
                                          fprintf(yyout, "</ul>\n");
                                        ul_count=0; BEGIN INITIAL;
                                        fprintf(yyout, "\t</body>\n</html>");
                                        yyterminate();
                                      }
<DICT><<EOF>>                         { fprintf(yyout, "</dl>\n");
                                        fprintf(yyout, "\t</body>\n</html>");
                                        yyterminate();
                                      }
<<EOF>>                               {fprintf(yyout,
 ↪  "\t</body>\n</html>");yyterminate();}

%%

void printHeader(char* title){
  fprintf(yyout, "<!DOCTYPE html>\n<html>\n\t<head>\n\t\t<meta
   ↪  charset=\"utf-8\">\n\t\t<title>%s</title>\n\t</head>\n<body>\n",
   ↪  title+4);
}

int main( int argc, char** argv){
  yyin = fopen(argv[1],"r");
  yyout = fopen(argv[2],"w");
  yylex();
}
```

4.2 Exemplo de input para o pré-processador

```
... Porsche 911 GT3.
\1- Porsche 911 GT3 - Automotive Perfection.
\2- Story
\+ The \-Porsche 911 GT3-\ is a high performance version of the Porsche 911
 ↪  sports car \_primarily intended for racing_\. It is a line of
 ↪  high-performance models, which began with the 1973 911 Carrera RS. The
 ↪  GT3, introduced in 1999, is named after the \~Fédération Internationale de
 ↪  l'Automobile~\ (FIA) Group GT3 class, in which it was ``designed to
 ↪  compete''.

; Porsche GT3 Overview
: Made by Porsche
: Production from 1999 to present
: Assembly at Stuttgart-Zuffenhausen in Germany
: Sports car
: 2-door coupe
: RR layout
: 3.6 4.0 L Flat-6 Engine
: 6-speed manual or 7-speed dual-clutch Transmission

\2- The Porsche GT3 road cars:
```

```
17
18  # 996 GT3
19  ## 996.1 GT3
20  ## 996.2 GT3
21  ## 996 GT3 RS
22  # 997 GT3
23  ## 997.1 GT3
24  ## 997.2 GT3
25  ## 997 GT3 RS
26  ### 997 GT3 RS 4.0
27  # 991 GT3
28  ## 991.1 GT3 RS
29  ## 991.2 GT3
30  ### 991.2 GT3 RS
31
32  \2- The Porsche GT3 racing cars:
33
34  * 1998 996 GT3 Cup
35  ** 1999 996 GT3 R
36  ** 2000 996 GT3 Cup[42]
37  *** 2001 996 GT3 RS
38  * 2004 996 GT3 RSR { with sequential gear box[43]
39  ** 2005 997 GT3 Cup[44]
40  *** 2006 997 GT3 RSR[45]
41  * 2008 997 GT3 Cup S[46]
42  ** 2010 997 GT3 Cup[47]
43  *** 2010 997 GT3 R[48] { homologation includes 2013 bodywork update[49]
44  * 2011 997 GT3 Cup
45  ** 2012 997 GT3 RSR[50]
46  *** 2013 991 GT3 Cup[51]
47  * 2013 991 RSR
48  ** 2014 991 GT America
49  *** 2016 991 GT3 R
50  * 2017 991 RSR[52]
51  ** 2017 991 GT3 Cup[53]
52
53  \3- The 2019 Porsche GT3 RS
54  \"https://hips.hearstapps.com/amv-prod-cad-assets.s3.amazonaws.com/images/18q1/
     ↪  699329/2019-porsche-911-gt3-rs-debuts-looks-bad-ass-because-it-is-news-car-
     ↪  and-driver-photo-702863-s-original.jpg?crop=1xw:1xh;center,center&resize=900
     ↪  :*"
```

Referências

[1] Vern Paxson, *Flex - a scanner generator*. `ftp://ftp.gnu.org/old-gnu/Manuals/flex-2.5.4/html_mono/flex.html` [acedido a 06/05/2018]