

UNIVERSIDADE DO MINHO

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA  
LICENCIATURA EM ENGENHARIA INFORMÁTICA

Processamento de linguagens - Trabalho Prático #2  
Ano Letivo 2021/2022  
Grupo 6

Gonçalo Braz (a93178)      Simão Cunha (a93262)

10 de junho de 2022

## 1 Introdução

Este relatório surge no âmbito do segundo trabalho prático da UC de Processamento de Linguagens que consistiu no desenvolvimento de ferramentas de avaliação gramatical de texto de modo a compilar o mesmo e produzir os resultados pretendidos no enunciado em questão. Para isto, disponibilizamos da linguagem de programação `Python` e das bibliotecas `ply` para a construção dos analisadores léxico e sintático.

A equipa docente disponibilizou a escolha de um de 4 enunciados. Após uma ponderação sobre a escolha, o nosso grupo decidiu seguir com a resolução do primeiro enunciado: **Linguagem de templates** (inspirada nos `templates Pandoc`).

### 1.1 Problema

O enunciado escolhido, **Linguagem de templates** (inspirada nos `templates Pandoc`), consiste na construção de uma gramática e de um *lexer* para fazer a análise de um dado *template pandoc* fornecido ao programa e, através de um dicionário também ele fornecido ao programa, gerar um texto final.

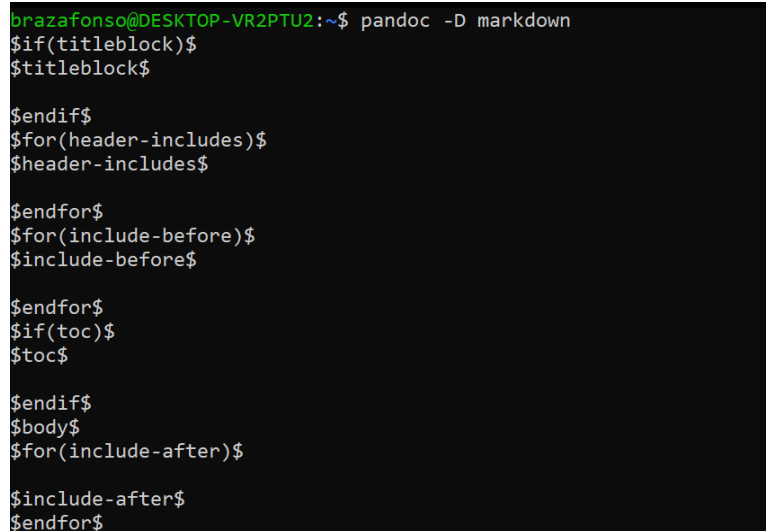
Existem 3 hipóteses principais de funcionamento do *parser* que temos de construir:

- Dado um *template* T1 gerar uma função em *Python* `expand_T1` que, recebendo um dicionário, dê um "texto final";
- Escrever um programa que dado um *template* T1 e um dicionário produza o "texto final";
- Escrever um programa que dado um ficheiro *template* (T1) e um ficheiro **YAML** (dicionário) produza o "texto final".

## 1.2 Estudo do problema

De forma a podermos começar a abordagem ao problema, devemos primeiramente familiarizarmos-nos com o assunto que iremos abordar, que neste caso são os *templates* Pandoc. Informação sobre estes pode ser encontrada em 1.

Analisemos um exemplo de um *template* de Pandoc simples, o *template* para ficheiros `markdown`:



```
brazafonso@DESKTOP-VR2PTU2:~$ pandoc -D markdown
$if(titleblock)$
$titleblock$

$endif$
$for(header-includes)$
$header-includes$

$endfor$
$for(include-before)$
$include-before$

$endfor$
$if(toc)$
$toc$

$endif$
$body$
$for(include-after)$
$include-after$
$endfor$
```

Figura 1: *Template* Pandoc de `markdown`

Observando este exemplo, relativamente simples, conseguimos identificar algumas particularidades desta linguagem:

- Variáveis, delimitadas por `$`;
- Condicionais e *loops* `if` e `for`;
- As variáveis nas condições não têm delimitadores.

Embora a linguagem de *templates* Pandoc apresente muitas mais particularidades do que as observadas neste exemplo, servimo-nos dele como base para a gramática, posteriormente expandindo-a para complementar as outras funcionalidades dos *templates*, tais como:

- *nesting* de `if` e `for`;
- condicionais `elif` e `else`;
- escrita alternativa de delimitadores;
- comentários;
- texto constante, i.e., texto fixo que se aplica diretamente no texto final, a não ser que esteja dentro de uma condicional com condição falsa;
- *partials*;
- *pipes*.



Os diferentes tuplos que podem ser resultados do *parser* são:

- (VAR,"valor da variavel")
- (PARTIAL,"valor do partial")
- (PIPE,"valor do pipe")
- (WORD,"valor do texto fixo")
- (CONST,"valor do texto fixo")
- (IF"id","condição do if")
- (FOR"id","condição do for")
- (ELIF,"condição do elif")
- (ELSE)
- (ENDIF"id")
- (ENDFOR"id")

Os comentários, por natureza, são ignorados pela sintaxe, não sendo passados para o compilador.

No caso de *loops* e condicionais, os seus iniciadores e terminadores (exemplo: *if* e *endif*), necessitam de um valor de identificação único de forma a que o compilador saiba as instruções (tuplos) que pertencem a estes blocos específicos.

O resultado final do *parser* será uma *string* com um conjunto de tuplos todos seguidos. Estes serão separados utilizando a expressão regular:

$$r'\((\backslash w+)\),([^\()]+|"([^\"]*)")\)?\)'$$

Assim isolados, são transformados numa lista de tuplos.

## 2.2 Analisador léxico

Com a gramática construída, passamos à construção do *lexer*. Este é construído utilizando a biblioteca *lex*.

Os *tokens* e literais definidos são:

```
tokens = ["VARIABLE","COMMENTBEGIN","COMMENTEND","COMMENT","PIPE",
          "PARTIAL","WORD","CONST","IF","ENDIF","ELSEIF","ELSE",
          "FOR","ENDFOR","ENDCONDITION"]
literals = ["$", "(", ")", "{", "}", "-", ".", "=", ":"]
```

Figura 4: *Tokens e literais*

O analisador que construímos tem 4 estados:

1. INITIAL

Estado inicial, que reconhece quase todos os *tokens* da gramática.

2. condition (inclusivo)

Início de condição ou *loop* (*if,elseif,for*), na forma básica (*\$if()*).

Este estado é necessário de forma a procurar o término de condição ou *loop* básico, *)\$*.

### 3. `conditionAlt` (inclusivo)

Estado similar ao anterior, utilizado para a escrita alternativa, por exemplo, `${if, com término }`.

### 4. `comment` (exclusivo)

Estado utilizado para reconhecer comentários, iniciado com o token `COMMENTBEGIN`, `$--` e término o final da linha.

Como tudo dentro desta linha é irrelevante, este estado reconhece qualquer caracter, 1 a 1 com o mesmo *token*, `COMMENT`.

## Exemplo de *tokens*

- **VARIABLE:** `$variavel$` ou `${variavel}` ou `$objeto.atributo$` (WORD com delimitadores)
- **CONST:** qualquer char (singular) exceto `$`. Para aceitar `$` no *template*, reconhece-se `$$`
- **WORD:** qualquer palavra começada com uma letra e seguida por qualquer símbolo alfanumérico, `.`, `-` ou `_`. O ponto serve como separador entre um objeto e o seu parâmetro
- **PARTIAL:** `$partial()` ou `${partial()}` ou `$variavel:partial()`
- **PIPE:** `$variavel/pipe$` ou `${variavel/pipe}` ou `$variavel/pipe1/pipe2$`
- **IF:** `$if(` ou `${if(`
- **ENDIF:** `$endif$` ou `${endif}`
- **ELSEIF:** `$elseif(` ou `${elseif(`
- **ELSE:** `$else$` ou `${else}`
- **FOR:** `$for(` ou `${for(`
- **ENDFOR:** `$endfor$` ou `${endfor}`
- **ENDCONDITION:** `)$` ou `)}`
- **COMMENTBEGIN:** `$--`
- **COMMENT:** qualquer char (singular) exceto *newline*
- **COMMENTEND:** *newline* ou fim de linha

## 2.3 Implementações

Nesta secção iremos explicar a lógica e funcionamento do programa, desde os argumentos recebidos e permitidos até à função que compila os resultados do *parser* sintático, que produz o texto final.

### 2.3.1 Inicialização

De forma a permitir diversos modos de utilização, o programa que codificamos aceita entre 1 a 3 argumentos, tendo claro, caminhos diferentes para produzir o resultado dependendo destes mas, deste modo, somos capazes de permitir um qualquer dos três modos de utilização, em termos de argumentos, explícitos na secção 1.1.

O único argumento obrigatório é o *template*, que será sempre o primeiro argumento.

Se o programa receber 1 argumento, irá utilizar um dicionário *default* e escreverá o *output* (texto final) no **standard output**.

Se receber 2 argumentos, primeiro verifica-se se o segundo argumento é um ficheiro. Caso seja, faz-se uma verificação se representa um dicionário, que pode ser do tipo **YAML** ou **Python**. Neste caso, o ficheiro é convertido num dicionário e este será utilizado para produzir o texto final, escrito no terminal. Caso contrário, considera-se que este argumento é o ficheiro de *output*.

Se não for um ficheiro, faz-se uma verificação se é um dicionário do tipo **Python** e, caso não seja, o programa terminará, avisando que recebeu um dicionário inválido.

Se receber 3 argumentos, o segundo argumento é assumido como sendo o dicionário e o terceiro como sendo o ficheiro de *output*.

### 2.3.2 Análise do *template*

Caso o programa chegue a esta fase, passará o *template*, o dicionário `Python` e o ficheiro de *output* para a função `expand_T1` (hipótese 1). Esta função pode ser chamada recursivamente ou então de forma isolada, como no caso dos *partials*.

Esta função começa por verificar se a validade do *template*. Em primeiro lugar, verifica se este ficheiro que existe. Caso exista, verificamos a sua sintaxe, utilizando o *lexer* e a gramática referidos anteriormente.

Se for encontrado algum erro, a função parará a sua execução. Contrariamente, irá guardar o resultado do analisador sintático, a lista de tuplos referida na secção 2.1, e irá interpretar estes tuplos, como sendo instruções, na principal função de compilação `compile_template`.

### 2.3.3 Geração do texto final

`compile_template` é a principal função de tratamento da lista de instruções.

As instruções reconhecidas são as descritas em 2.1.

A função pode receber 6 argumentos, sendo 3 deles opcionais.

```
def compile_template(tuple_stack, dictionary, file, condition = True, type = 0, j=0):
```

Figura 5: Definição de `compile_template`

As variáveis `condition`, `type` e `j` são utilizadas no caso de chamadas recursivas, como será explicado mais abaixo.

A função percorre todas as instruções da lista de instruções, reagindo de forma diferente para cada:

- **CONS**: instrução mais simples, é diretamente escrita no ficheiro de *output*.
- **VAR**: verifica se o dicionário contém a variável. Em caso positivo, escreve-a. A variável pode variar dependendo do contexto, por exemplo, no caso da variável ser uma lista e ao mesmo tempo ser a condição deste, a variável será, na realidade, o valor na lista no índice da iteração atual.

No caso da variável `it` dentro de um `if`, `for` ou `elseif`, esta é considerada como sendo igual à variável de condição.

- **PARTIAL**: chama a função `compile_partial` que irá separar o *partial* da variável se houver. Se houver variável, o *partial* apenas é aplicado se esta for um dicionário. Se não houver variável é aplicado o *partial*. Claro está, o *partial*, sendo um *template*, é aplicado através de uma chamada da função `expand_T1`.
- **PIPE**: chama a função `compile_pipe` que irá separar a variável dos vários métodos nele aplicado, verificando primeiro a validade de todos os métodos, se eles pertencem aos métodos reconhecidos, e depois se a variável existe. Apenas no caso destes dois serem válidos é que os métodos serão aplicados à variável, cada um com as suas verificações isoladas.

Os métodos disponíveis são:

– <code>pairs</code>	– <code>reverse</code>	– <code>allbutlast</code>	– <code>roman</code>
– <code>uppercase</code>	– <code>first</code>	– <code>chomp</code>	
– <code>lowercase</code>	– <code>last</code>	– <code>nowrap</code>	
– <code>length</code>	– <code>rest</code>	– <code>alpha</code>	

Todos os métodos tentam seguir as definições usadas pelos *templates* `Pandoc` (1).

- **IF:** começa por calcular todas as instruções pertencentes a este bloco (**elseif** e **else** inclusive), de modo que se o bloco não tenha qualquer condição válida, se consiga saltar para a primeira instrução após o bloco.

Com o bloco calculado, este é passado para compilação na função `compile_if` que faz as verificações das condicionais, procurando as instruções que devem ser aplicadas no texto final.

Assim que uma condição válida é encontrada, é de novo calculado o bloco de instruções para esta condicional e passado para a função `compile_template`, que trata da lista como se fosse um *template* próprio, ou seja, poderá dentro de um **if**, haver qualquer tipo de instrução, seja novos **if**, **for**, entre outros.

- **FOR:** de forma similar ao **if**, no caso do **for**, também é calculado o seu bloco de instruções e só depois verificado na função `compile_for` se estas se aplicam. No caso específico do **for**, a variável de condição determina o número de iterações, ou seja, de chamadas de `compile_template` que serão feitas, dependendo se a condição é uma lista ou uma variável simples.

#### 2.3.4 Outros

Notável foi ainda o ficheiro `utilities.py` que escrevemos, onde temos muitas das funções auxiliares mais importantes para o programa, como é o caso das funções de procura no dicionário, que permitem a procura de dicionários dentro de dicionários ou da devolução do item específico da iteração atual.



### 3 Exemplos de utilização

Nesta secção, faremos algumas demonstrações da utilização e resultados do programa.

Como mencionado anteriormente, o programa tem duas assunções *default*: o dicionário e o ficheiro de escrita, que é o terminal.

O dicionário *default* é o seguinte:

```
dictionary = {
  'lang' : 'en',
  'dir' : 'exemplo/exemplo',
  'author-meta' : ['autor-1','autor-2'],
  'author' : 'myself',
  'date-meta' : 'today',
  'keywords' : 'html_template',
  'sep' : ',',
  'title-prefix' : 'exemplo',
  'pagetitle' : 'pagina fixe',
  'quotes' : 'quote',
  'highlighting-css' : 'h-css',
  'css' : 'css',
  'math' : '1+1=2',
  'header-includes' : ['header 1','header 2'],
  'include-before' : ['include 1'],
  'title' : 'template',
  'idprefix' : 'prefixo',
  'subtitle' : 'subtitulo',
  'date' : 'today-date',
  'toc' : 'TOC',
  'body' : 'corpo do ficheiro',
  'include-after' : 'include after',
  'teste' : {
    'a' : [1,3,5,6],
    'b' : 'text',
    'c' : [
      {
        'another_one' : 'text1'
      },
      {
        'another_one' : 'text2'
      }
    ]
  }
}
```

Figura 6: Dicionário *default*

O *template* de teste que criámos é o seguinte:

```
$if(x)$  
$titleblock$  
$elseif(b)$  
$else$  
$if(teste)$  
$it$  
$endif$  
$toc$  
$it$  
$endif$  
$for(header-includes)$  
$header-includes$  
  
$endfor$  
${for(include-before)}  
$include-before$  
$endfor$  
$if(toc)$  
$table-of-contents$  
  
$endif$  
$body$  
$for(include-after)$  
  
$include-after$  
$endfor$  
  
$teste.a$  
$teste.a/reverse$  
$for(teste.c)$  
$teste.c.another_one$  
$teste.c.another_one/uppercase$  
$endfor$
```

Figura 7: Template teste

Este *template* apresenta quase todas as funcionalidades que disponibilizamos, `if` com `elseif` e `else` e `if` aninhado, `for`, acesso a dicionários dentro de dicionários e uso de `pipes`.

### 3.1 Teste - *template* de teste

Chamando o programa apenas com o *template*, temos:

```
i\3º ano\Processamento de linguagens\p1_projeto\TP2\main.py" .\templates_teste\template_teste.txt
{'a': [1, 3, 5, 6], 'b': 'text', 'c': [{'another_one': 'text1'}, {'another_one': 'text2'}]}
TOC

header 1

header 2

include 1

corpo do ficheiro

include after

1 3 5 6
6 5 3 1
text1
TEXT1
text2
TEXT2
PS C:\Users\brazz\OneDrive\Ambiente de Trabalho\uni\3º ano\Processamento de linguagens\p1_projeto\TP2> █
```

Figura 8: *Output*

### 3.2 Teste - *template* de HTML Pandoc

Num outro teste, de forma a testar o programa com os *templates* que o inspiraram, utilizamos o *template* para ficheiros HTML do Pandoc.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" ${if(lang)} lang="${lang}" xml:lang="${lang}" $endif ${if(dir)} dir="${dir}" $endif>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <meta http-equiv="Content-Style-Type" content="text/css" />
  <meta name="generator" content="pandoc" />
  $for(author-meta)$
  <meta name="author" content="$author-meta" />
  $endfor$
  ${if(date-meta)}
  <meta name="date" content="${date-meta}" />
  $endif$
  $if(keywords)$
  <meta name="keywords" content="$for(keywords)$${keywords}$sep$, $endfor$" />
  $endif$
  <title>${if(title-prefix)}${title-prefix} - $endif${pagetitle}</title>
  <style type="text/css">code{white-space: pre;}</style>
  $if(quotes)$
  <style type="text/css">q { quotes: "“" "”" "‘" "’"; }</style>
  $endif$
  $if(highlighting-css)$
  <style type="text/css">
$highlighting-css$
  </style>
  $endif$
  $for(css)$
  <link rel="stylesheet" href="$css" type="text/css" />
  $endfor$
  $if(math)$
  $math$
  $endif$
  $for(header-includes)$
  $header-includes$
  $endfor$
</head>
<body>
  $for(include-before)$
  $include-before$
  $endfor$
  $if(title)$
  <div id="$idprefix$header">
    <h1 class="title">${title}</h1>
    $if(subtitle)$
    <h1 class="subtitle">${subtitle}</h1>
    $endif$
  $for(author)$
  <h2 class="author">${author}</h2>
  $endfor$
  $if(date)$
  <h3 class="date">${date}</h3>
  $endif$
  </div>
  $endif$
  $if(toc)$
  <div id="$idprefix$TOC">
    $toc$
  </div>
  $endif$
  $body$
  $for(include-after)$
  $include-after$
  $endfor$
</body>
</html>
```

Figura 9: *Template* HTML Pandoc

Como *output* obtemos:

```
eto\TP2\main.py" .\templates_teste\template_html.txt
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en" dir="exemplo/exemplo">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <meta http-equiv="Content-Style-Type" content="text/css" />
  <meta name="generator" content="pandoc" />
  <meta name="author" content="autor-1" />
  <meta name="author" content="autor-2" />
  <meta name="date" content="today" />
  <meta name="keywords" content="html_template,, " />
  <title>exemplo - pagina fixe</title>
  <style type="text/css">code{white-space: pre;}</style>
  <style type="text/css">q { quotes: "“" "”" "‘" "’"; }</style>
  <style type="text/css">
h-css
  </style>
  <link rel="stylesheet" href="css" type="text/css" />
  1+1=2
  header 1
  header 2
</head>
<body>
include 1
<div id="prefixoheader">
<h1 class="title">template</h1>
<h1 class="subtitle">subtitulo</h1>
<h2 class="author">myself</h2>
<h3 class="date">today-date</h3>
</div>
<div id="prefixoTOC">
TOC
</div>
corpo do ficheiro
include after
</body>
</html>
PS C:\Users\braza\OneDrive\Ambiente de Trabalho\uni\3º ano\Processamento de linguagens\pl_projeto\TP2> □
```

Figura 10: *Output*

## 4 Conclusão

Com a realização deste trabalho consideramos ter conseguido ser bastante sucedidos na construção de um leitor de *templates* Pandoc e, conseqüente, a construção de *outputs* com os resultados esperados.

Aprofundamos ainda a nossa experiência na construção de gramáticas de linguagem, através do uso de expressões regulares e do estudo das bibliotecas do *lex* e *yacc* do *ply*, assim como o conhecimento da linguagem *python*.

O programa que produzimos foi além de capaz de, produzir resultados satisfatórios, permitir diversos modos de utilização. Embora tenhamos feito uma boa emulação das regras dos *template* de Pandoc, como trabalho futuro ainda teríamos algumas particularidades para aperfeiçoar, como é o caso da indentação.

## Referências

- [1] Manual Pandoc