Trabalho Prático Nº2 - Compiladores

Marco Sousa[1,2[62608]], José Malheiro[1,2[93271]], and Miguel Fernandes[1,2[94269]]

[1] Universidade do Minho
[2] Licenciatura em Engenharia Informática, Braga, Portugal

Abstract. **Keywords:** Expressões regulares · Python · Templates · Gramática Independente de Contexto · Compiladores.

1 Introdução

1.1 Contextualização

O presente relatório foi escrito no âmbito da Unidade Curricular (UC) de Processamento de Linguagens (PL), apresentando como objetivo a descrição da solução desenvolvida para o segundo trabalho prático, assim como as decisões tomadas para a sua conceção.

A solução é relativa ao primeiro enunciado: **Linguagens de Templates** (inspirada nos *templates Pandoc*).

Neste sentido, é proposto o desenvolvimento de um compilador de *templates*, de modo a criar ficheiros *markup*, a partir do processamento de *templates* definidos. Para tal usou-se a linguagem *YAML* para guardar toda a informação a ser usada.

Assim, engloba o desenvolvimento do **analisador léxico** e o **parser**, bem como a gramática e a estrutura da árvore sintática criadas para poder gerar o texto final.

1.2 Breve Descrição do Trabalho Proposto

O enunciado escolhido, **Linguagem de templates** (inspirada nos *templates Pandoc*), propõe a criação de um compilador de *templates*. Deste modo, utiliza-se:

1. um *Template*.
2. um Dicionário.

. . . para construir o texto final, um **ficheiro *markup***, com todos os dados submetidos pelo utilizador.

Os *templates* a usar devem incluir um conjunto de regras, para permitir substituir os dados submetidos pelo utilizador nos parâmetros desejados. Neste sentido, propõe-se o desenvolvimento da **gramática** que cada *template* deve seguir, independemente da linguagem que está a utilizar.

Para a criação do **compilador**, segue-se a criação do:

- Analisador Léxico.
- *Parser*.

. . . que permite capturar os campos de **metadados** dos *templates*, sem afetar o restante texto, para posteriormente os poder tratar, gerando o texto final. Para tal deve-se recorrer ao módulo *PLY* do *Python*, nomeadamente o gerador de analisador léxicos - *Lex* - e o gerador de compiladores (com base em gramáticas tradutoras) - *Yacc*.

O **dicionário** corresponde aos dados a serem inseridos no *template*. No projeto a realizar utilizar-se-á ficheiros *YAML*, usufruindo da organização estrutural da linguagem.

2 HandleDoc

- Gramática abstrata
  - máquina de estados
  - estratégia utilizada – arvore de parsing concreta – classes – hierarquia – descentralização das responsabilidades – diagrama de classes
  - Funcionalidades – if – elseif – else
  – for
  – pipes Tal como no *Pandoc*, foram criados *Pipes* que transformam o valor de uma variável ou de um *Partial*.

Os *Pipes* são especificados com o uso de uma barra (/), depois do nome da variável ou do *partial*. Por exemplo:

```
$author/uppercase$

$for(authors/reverse)$
    $it.name$
$endfor$

$subtemplate()/lowercase$
```

Os *Pipes* também podem ser encadeados:

```
$author/uppercase/reverse$
```

Quanto aos *Pipes* criados, foram feitos todos os que o *Pandoc* tem implementados até ao momento da realização do presente relatório, mais concretamente:

– *pairs:* Converte um dicionário ou uma lista para um dicionário de listas. Caso o valor seja uma lista, a chave corresponde ao índice do array, começando em 1.
– *uppercase:* Converte um texto todo para letras maiúsculas.
– *lowercase:* Converte um texto todo para letras minúsculas.

– subtemplating
– range
– it
– sep
CONSTRUIR MANUAL!!

3 Conclusões

4 Anexos