Processamento de Linguagens
(3º ano de Curso)

Trabalho Prático 1

Relatório de Desenvolvimento

Etienne Costa	Pedro Costa	Rui Azevedo
(a76089)	(A85700)	(a80789)

5 de Abril de 2020

Resumo

O presente relatório é referente ao primeiro trabalho prático da cadeira *Processamento de Linguagens* e tem como objectivo demonstrar o processo de desenvolvimento de um filtro de texto através da ferramenta *Flex*.

Com este trabalho pretende-se criar um programa capaz de aceitar um nome de um projecto e um ficheiro de descrição de um *template* multi-file e gerar as respectivas directorias e ficheiros, formando assim um projecto inicial.

Conteúdo

Capítulo 1

Introdução

A criação da estrutura de um projecto, bem como os documentos que o compõem, tem uma importância gigante no desenvolvimento de cada sistema. Se a cada vez que se inicia um projecto fosse necessário criar de raiz todas as directorias e ficheiros base que o compõem, estaria-se de facto a desperdiçar tempo e provavelmente dinheiro num contexto empresarial.

Dado isto, o projecto escolhido para desenvolver foi o trabalho relativo à criação de um *template* inicial de um projecto, dada a sua utilidade académica e empresarial.

Para este fim, foi criado um filtro de texto através da ferramenta *Flex*, capaz de ler um ficheiro com um *template* de um projecto *multi-file*. Este ficheiro, que servirá de *input* à aplicação, tem um formato específico, que irá ser apresentado posteriormente.

Após o desenvolvimento desta aplicação, o resultado pretendido é a criação de um sistema capaz de gerar, após a leitura do ficheiro *template*, todas as directorias, sub-directorias e ficheiros pretendidos.

O documento apresentado irá apresentar a análise e especificação do problema, a concepção do sistema onde se irá demonstrar o processo e desenvolvimento da aplicação, e, por fim, a codificação e testes da mesma.

Capítulo 2

Análise e Especificação

2.1 Descrição informal do problema

Pretende-se criar um programa *"mkfromtemplate"* cujos argumentos são o nome do projecto e um ficheiro descritivo de um *template multi-file*. Este ficheiro tem referências para o nome dos ficheiros que devem ser criados bem como o seu conteúdo inicial. Para álem disso, contém a árvore de directorias que devem ser criadas a quando da execução do programa desenvolvido.

O resultado final será a criação de um projecto com todo o conteúdo que está contido no ficheiro *template* de *input*

2.2 Especificação do Requisitos

Para que esta programa faça o pretendido, é imperativo que exista um ficheiro *template* que irá ser lido pela aplicação. Este ficheiro tem uma estrutura bem definida, apresentada de seguida.

Listing 2.1: Template

```
1 == meta
2
3 email: author@mail.pt
4 author : author name
5
6 == tree
7
8 {%name%}/
9 -{%name%}.fl
10 -doc/
11 --{%name%}.md
12 -exemplo/
13 -Makefile
14
15 == FileName
16
17 FileName content
```

2.2.1 Dados

Na *listagem 2.1*, pode-se ver a estrutura que o ficheiro *template* deve ter. De seguida, é apresentada a descrição de cada das etiquetas definidas.

- ***meta*** : referente aos dados pessoais do autor, *i.e*, contém o seu nome e o seu email. Esta informação não é exaustiva podendo o utilizador acrescentar mais informação.
- ***tree*** : salienta como a árvore de directorias deve ser criada bem como em que cada pasta devem ser guardados os ficheiros.
- ***FileName*** : cria um ficheiro com o nome *FileName*. A baixo desta etiqueta, o utilizador deve escrever o conteúdo com que o ficheiro deve ser inicializado. O utilizador poderá definir o número de ficheiros que bem entender.

Capítulo 3

Concepção / Desenho da resolução

Para a desenvolvimento do sistema foram desenvolvidos cinco ficheiros que compõem o filtro usado para extrair a informação do ficheiro de entrada e criar o resultado final pretendido:

- **Makefile**: cria o executável da aplicação
- **mkfromtemplate.f**: filtros de texto escritos em *flex*
- **template_flex**: ficheiro de teste

3.1 Estruturas de dados

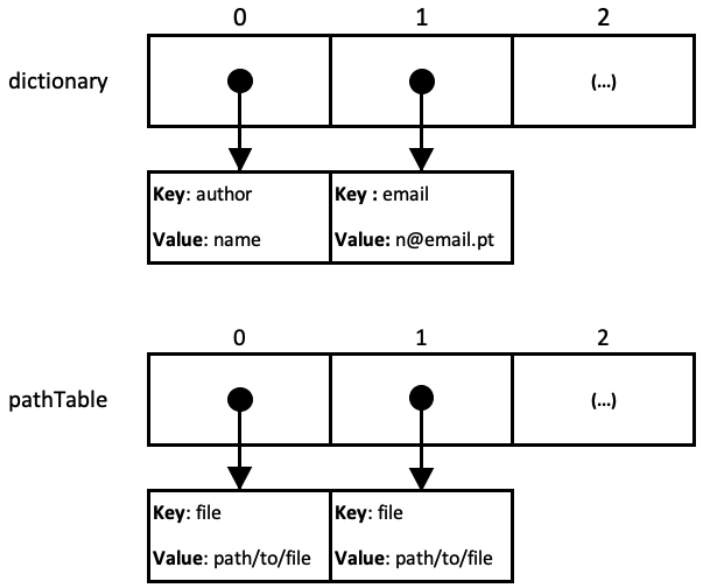

Figura 3.1: Tabelas de *hash*

Para além disso, foram usadas estruturas de dados, consideradas adequadas, para guardar a informação desejada. Estas estruturas de dados pertencem à biblioteca *glib.h*, que contém um conjunto de estruturas de dados já pré-definidas bem como funções para trabalhar sobre elas.

Para guardar a informação relativa aos meta-dados do ficheiro, *i.e*, autor, email, etc., e para a árbore de diretorias, foram usadas duas tabelas de *hash*, designadas, respectivamente, *dictionary* e *pathTable*. A figura 3.1 ilustra o conteúdo das tabelas.

3.2 Estrutura do processador de texto

Para o processador de texto *flex*, para além das estruturas de dados referidas na secção anterior e das expressões regulares, foram definidos quatro estados, para além do estado pré-definido inicial, para que a informação seja processada mais facilmente:

- **META**: estado correspondente aos meta-dados.
- **TREE**: estado correspondente à árvore de directorias.
- **FILESTATE**: estado correspondente a um ficheiro .
- **INFILE**: estado que trabalha um ficheiro.

Pode-se então criar um autómato correspondente à transição de estados. O estado 0 é referente ao estado inicial, os restantes estados são, respectivamente, os enumerados em cima.

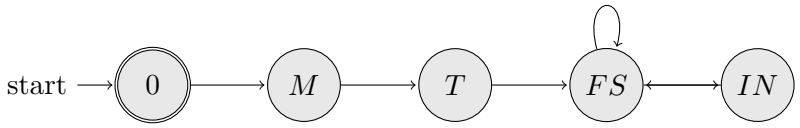

Figura 3.2: Autómato de transição de estados

Capítulo 4

Codificação e Testes

4.1 Filtro de texto *flex*

Numa primeira fase, o filtro de texto tenta encontrar os meta-dados. Caso esta *tag* exista, entra no estado ***META***. Dentro deste estado, é feito *parsing* aos meta-dados e estes mesmos são adicionados à tabela de *hash*.

Após o estado ***META***, o filtro entra no estado ***TREE***, onde se vai analisar o sistema de ficheiros em texto e recriá-lo ao mesmo tempo. Para esse efeito, temos 2 grupos de ER's a atuar dentro deste estado. Um deles é capaz de ler diretorias e recorre ao comando ***mkdir -p*** para as criar. O outro grupo trata apenas de criar ficheiros. Para esse efeito, primeiro lê o nome do ficheiro a criar até apanhar um ponto (.) ou um `\n`. Caso seja um `\n` procede directamente a criar o ficheiro, caso contrário temos uma outra ER capaz de ler extensões e fazer a junção da extensão ao nome do ficheiro apanhado, fazendo depois uso do comando ***touch*** para criar um ficheiro vazio. Todo este processo é controlado por uma variável global que acumula a nossa diretoria atual.

Após processada a árvore de directorias, ficámos com uma tabela *hasht* capaz de associar nomes de ficheiros ao seu caminho absoluto. Partimos então para o processamento dos *templates* de cada ficheiro. Quando apanhamos o nome do ficheiro que queremos inserir informação entramos no modo ***FILESTATE***. Através do mesmo mecanismo de apanhar nomes e extensões, quando obtemos o nome do ficheiro onde queremos escrever abrimos um descritor para o mesmo e entramos no estado ***INFILE*** onde simplesmente escrevemos tudo o que é lido no ficheiro para o correspondente.

4.2 Testes realizados e Resultados

```
=== meta

email: jj@di.uminho.pt
author: J.João

# "name" é dado por argumento de linha de comando (argv[1])

=== tree

{%name%}/
- {%name%}.fl
- doc/
-- {%name%}.md
-- {%name%}
- exemplo/
- README
- Makefile

=== Makefile

{%name%}: {%name%}.fl
    flex {%name%}.fl
    cc -o {%name%} lex.yy.c

install: {%name%}
    cp {%name%} /usr/local/bin/

=== {%name%}

teste

=== {%name%}.md
# NAME

{%name%} - o nosso fabuloso filtro ...FIXME

## Synopsis

   {%name%} file*

## Description

## See also

## Author
```

Figura 4.1: Ficheiro de entrada

```
Comments and bug reports to {%author%}, {%email%}.

=== {%name%}.fl
%option noyywrap yylineno
%%

%%
int main(){
  yylex();
  return 0;
}
=== README

FIXME: descrição sumária do filtro
```

Figura 4.2: Ficheiro de entrada (continuação)

```
teste/
├── Makefile
├── README
├── doc
│   ├── teste
│   └── teste.md
├── exemplo
└── teste.fl

2 directories, 5 files
```

Figura 4.3: Vista da árvore de directorias

Capítulo 5

Conclusão

O presente relatório demonstrou todo o processo necessário para construir esta aplicação de criação de projectos.

A ferramenta *Flex* foi bastante útil no desenvolvimento deste projecto pois fornece uma maneira de processar texto, através de expressões regulares, muito intuitiva e eficiente. Para além disso, a biblioteca de estruturas de dados genéricas *glib* foi também bastante útil, poupando tempo de desenvolvimento e fornecendo as estruturas necessárias, neste caso, um módulo de tabelas de *hash*, eficiente.

Por fim, conclui-se que o objectivo do trabalho foi atingido. O programa consegue ler eficientemente o ficheiro de entrada e consegue reproduzir todas ficheiros referenciados no mesmo. Para além disso, o trabalho desenvolvido é uma ferramenta bastante útil para usar no futuro dada a sua facilidade de organizar um projecto.