Processamento de Linguagens (3º ano de Curso)

Trabalho Prático 1

Relatório de Desenvolvimento

Grupo 43

Gonçalo Esteves	João Araújo	Rui Oliveira
(A85731)	(A84306)	(A83610)

5 de Abril de 2020

Resumo

No dia a dia deparamo-nos com variados projetos de software, sendo que é normal a existência de soluções envolvendo várias pastas e ficheiros. Este relatório visa então a documentação do nosso trabalho que se baseia na implementação de um programa que cria as diretorias e ficheiros de determinado projeto. Paralelamente, tem-se como objetivo demonstrar a capacidade de definição de expressões regulares e desenvolvimento de processadores de linguagens no contexto da Unidade Curricular.

Conteúdo

Capítulo 1

Introdução

Integrado na cadeira de *Processamento de Linguagens*, foi desenvolvido o primeiro trabalho prático que requeria a escolha de um dos enunciados fornecidos pela equipa docente, sendo que aquele que selecionamos foi o Template Multi-file por se tratar de um tema que poderia ser transportado para algo com que lidamos no nosso quotidiano. Para a realização deste projeto, exploramos a utilização das ferramentas do Flex.

Capítulo 2

Análise e Especificação

2.1 Descrição informal do problema

O enunciado escolhido para este projeto foi o primeiro, Template Multi-file, , que se encontra dividido na análise de três componentes:

1. Metadados (autor e email);

2. Tree (estrutura de directorias e ficheiros a criar);

3. Template de cada ficheiro.

Para tal, inicialmente é necessário armazenar os metadados do template. Posteriormente, definir e especificar padrões relativamente à tree/estrutura do projeto e, consequentemente, definir as ações semânticas de acordo com os padrões respetivos, de modo a saber os paths dos ficheiros e diretorias. Após analisar a tree, voltar a fazer o mesmo, no entanto, com o objetivo de recolher a informação no que diz respeito ao conteúdo dos ficheiros do projeto.

2.2 Especificação dos Requisitos

A realização deste trabalho prático tem os seguintes requisitos:

1. Especificar os padrões de frases que quer encontrar no texto-fonte, através de ERs;

2. Identificar as acções semânticas a realizar como reacção ao reconhecimento de cada um desses padrões;

3. Identificar as Estruturas de Dados globais que possamos eventualmente precisar para armazenar temporariamente a informação que se vai extraindo do texto-fonte ou que se vai construindo à medida que o processamento avança;

4. Desenvolver um Filtro de Texto para fazer o reconhecimento dos padrões identificados e proceder à transformação pretendida, com recurso ao Gerador Flex.

Capítulo 3

Concepção/desenho da Resolução

3.1 Estruturas de Dados

No que diz respeito às estruturas de dados, de modo a armazenar a informação relativamante às diretorias e aos ficheiros a criar, foram utilizados dois arrays dinâmicos, sendo que um guarda os nomes das directorias enquanto que outro os paths para os ficheiros. Além disso, usamos outras duas strings para armazenar, respetivamente, o email e o autor do projeto.

3.2 Algoritmos

As decisões tomadas na escolha dos filtros a utilizar teve um papel fundamental, uma vez que o analisador léxico da aplicação é o responsável por identificar os tokens principais da linguagem. Para isso foram definidos vários contextos, e expressões regulares para os capturar.

Para filtrar os metadados, o raciocínio aplicado foi o seguinte: quando encontramos ^===\ meta\n, é aplicada a ação de iniciar o estado Meta e dentro deste são aplicadas as expressões que encontram e guardam o email e o autor na estrutura, sendo que no final das mesmas é executado BEGIN INITIAL.

Para a filtragem da estrutura, o raciocínio inicial é semelhante ao anterior: ao encontrar ^===\ tree\n é aplicada a ação de início do estado Tree, sendo que termina quando encontra um = no início de uma linha. Até encontrar este caracter, são aplicadas expressões que distinguem e guardam a informação na estrutura verificando se estamos a tratar de um ficheiro ou uma diretoria, e a profundidade do mesmo, através do número de hífens que o antecedem. Além disso, quando o número de hífens da diretoria ou ficheiro que estamos a analisar for superior à profundidade da árvore de diretorias no momento, a mesma é rejeitada, enquanto que quando o contrário acontece, regredimos de profundidade até à equivalente ao número de hífens.

Analisemos agora com maior detalhe duas das expressões regulares relativas ao estado Tree e uma do estado inicial:

1. **<Tree>^[-]+[ ]*{FILE}\/\n :** é verificado, primeiramente, se a frase é iniciada por 1 ou mais hífens e se a procedem 0 ou mais espaços. Após isso, verificamos se procede um nome válido de uma diretoria com um \/\n no fim da linha. Caso a expressão regular seja verificada, começa-se por verificar se é necessário alocar espaço para a inserção da diretoria detetada no array das mesmas e guarda-se o número dos hífens que se encontram no início da linha. Caso este seja superior aa 0 o processo continua, senão fica por aqui. Guarda-se também o número de espaços que antecedem a diretoria e obtemos o path da mesma através da função *criarPath*. Tendo o path, utiliza-se a função *criarDiretorua* de modo a criar a diretoria e adicionamos a mesma ao array que armazena as diretorias;

2. **<Tree>^[-]+[ ]*\{%name%\}\.{EXT}\n :** o início desta expressão é igual à anterior. Após isso, deteta {%name%} que corresponde ao nome do projeto e, seguidamente, o tipo de file com um \n no fim da linha. Caso a ER seja verificada, é verificado se é necessário alocar espaço para a inserção do ficheiro detetado no array dos mesmos e conta-se o número de hífens que se encontram no início da linha, sendo que assim como a ER anterior, o processo só continua caso seja superior a 0. Após isto, guarda-se o número de espaços que antecedem {%name%} e obtém-se o path respetivo através da função *criarPath*, recorrendo ao nome do projeto que se encontra guardado na variável global *nome*. De seguida, o path e o tipo de file a criar são concatenados e é invocada a função *criarFicheiro*. Por fim, adicionamos o path, já concatenado, ao array que guarda os paths dos ficheiros.

Após o estado Tree terminar, pois encontrou um =, filtramos as frases **==\ \{%name%}\.{EXT}\n** e **==\ ({FILE}\/)*{FILE}(\.{EXT})?\n** que servem para informar que a partir desse ponto até ser detetado o próximo = o que está contido nesse intervalo é o conteúdo do ficheiro que a expressão regular deteta. Estas duas expressões diferem não só no tipo de ficheiro encontrado, ou seja, se este se encontra dependente do nome do projeto ou não, mas também na forma como podem ser identificados. Deste modo, torna-se possível criar ficheiros (não dependentes do nome do projeto) com o mesmo nome e extensão desde que em pastas diferentes, sendo assim possível estes serem identificados pelo path total ou parcial.

Para filtragem destes dois tipos de frases, o raciocínio é, após estas serem detetadas, verificar em que indíce do array de ficheiros da nossa estrutura de dados se encontra o nome do ficheiro detetado e, após isso, abrir o descritor de ficheiro do mesmo e iniciar o estado File, responsável por aplicar as expressões que escrevem no ficheiro a partir do descritor referido.

Capítulo 4

Codificação e Testes

4.1 Alternativas, Decisões e Problemas de Implementação

A primeira decisão com que nos deparamos foi a formulação de algumas expressões regulares que achamos que teriam maior importância ao longo do trabalho e optamos então por definir três variáveis, EMAIL, FILE e EXT.

A primeira, caracteriza como deve ser constituído um endereço de email, exigindo que este cumpra uma forma que lhe é comum: primeiramente, é constituído por uma expressão, na qual não poderão aparecer alguns caracteres (como o @, por exemplo). Esta expressão é seguida de um @, que consequentemente será seguido de uma ou mais expressões terminadas com um ponto (gmail. ou di.uminho.), aparecendo por fim a extensão do endereço (com/pt/...).

A segunda, define como deverão ser formados os nomes dos ficheiros ou diretorias. Nestes não deverão aparecer certos caracteres que o próprio sistema operativo não permite na criação de diretorias ou ficheiros (os caracteres excluidos pela expressão regular).

Por fim, a terceira variável indica-nos o formato de uma extensão de um ficheiro. Esta poderá ser constituída por qualquer caractér exceto os indicados, podendo ou não haver mais do que uma extensão (simplesmente .txt ou então .svg.png) .

```
EMAIL       [^ @\/|?<>*:="\n\t\r]+@([a-z]+\.)+[a-z]+
EXT         ([^.\n\t\r ]\.)*([^.\n\t\r ])+
FILE        [^ \/|?<>*:="\n\t\r]+
```

Figura 4.1: ER definidas como variáveis

Posteriormente, no momento em que estivemos a avaliar a melhor solução de modo a armazenar dados, chegamos à conlusão que o melhor seria a utilização de arrays dinâmicos, uma vez que como não sabemos quantos ficheiros/diretorias o programa irá ler quando for executado, comparativamente a, por exemplo, um array estático, torna-se uma melhor opção. Portanto, de modo a otimizar os mesmos, optamos por implementá-los de forma a que o seu tamanho vai duplicando sempre que atinge a sua capacidade máxima. No entanto, estes duas estruturas não funcionam de forma igual visto que, por um lado, o de diretorias funciona como stack onde apenas se vai armazenando as que vamos entrando e à medida que saímos de uma determinada diretoria, esta é apagada da memória, e, por outro lado, o de ficheiros não tem essa especificidade. Esta diferença surge da necessidade de controlarmos melhor as diretorias que se encontram "abertas"à medida que a profundidade da estrutura de diretorias/ficheiros cresce.

4.2 Modo de Utilização

Tirando proveito do uso de uma makefile, o nosso programa torna-se de fácil criação e execução. Primeiramente, e dentro da pasta da diretoria, é necessário correr o comando *make*, por forma a criar o executável do programa. De seguida, basta apenas correr *./mkfromtemplate <NomeDoProjeto> <TemplateDoProjeto>* e o programa encarregar-se-á de criar uma pasta seguindo as especificações dadas.

4.3 Testes realizados e Resultados

De modo a demonstrar os pontos abordados ao longo do relatório, apresentamos a seguir o teste realizado relativamente a um Template Multi-file, neste caso o fornecido pela equipa docente, e o respetivo resultado obtido:

```
ruizinho@rodrigo:~/Desktop/Universidade/3º ano/2º semestre/PL/TP1$ ls -l
total 24
-rw-rw-r-- 1 ruizinho ruizinho  283 Apr  5 04:23 Makefile
-rw-rw-r-- 1 ruizinho ruizinho 1367 Apr  4 02:20 mk.c
-rw-rw-r-- 1 ruizinho ruizinho 8131 Apr  5 04:19 mkfromtemplate.l
-rw-rw-r-- 1 ruizinho ruizinho  210 Apr  4 02:20 mk.h
-rw-rw-r-- 1 ruizinho ruizinho  611 Apr  4 02:23 template
ruizinho@rodrigo:~/Desktop/Universidade/3º ano/2º semestre/PL/TP1$ make
flex mkfromtemplate.l
gcc -c lex.yy.c
gcc -Wall -c mk.c
gcc -o mkfromtemplate mk.o lex.yy.o -ll
ruizinho@rodrigo:~/Desktop/Universidade/3º ano/2º semestre/PL/TP1$ ./mkfromtemplate projeto template
ruizinho@rodrigo:~/Desktop/Universidade/3º ano/2º semestre/PL/TP1$ tree projeto
projeto
├── doc
│   └── projeto.md
├── exemplo
├── Makefile
├── projeto.fl
└── README

2 directories, 4 files
```

Figura 4.2: Terminal após a execução do programa

Capítulo 5

Conclusão

Terminado o desenvolvimento deste trabalho prático, acreditamos ter conseguido cumprir com os objetivos estabelecidos, uma vez que criamos um programa que gera as diretorias e ficheiros de determinado projeto, através do nome e ficheiro descrição de um template multi-file do mesmo. Este projeto permitiu uma consolidação da matéria lecionada, sendo que constatamos a evidente versatilidade e potencialidade que o Flex oferece, revelando uma enorme eficácia na filtração e tratamento de informação.