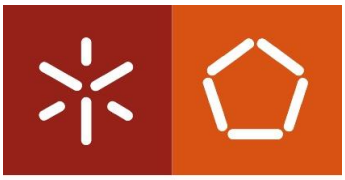

Universidade do Minho

Departamento de Informática

Transformador Publico2NetLang

TP1 - Exercício 4 - Grupo 7

Joao Teixeira (A85504)
Jose Filipe Ferreira (A83683)
Miguel Solino (A86435)

5 de Abril de 2020

Resumo

O objetivo deste projeto e processar um ficheiro HTML para produzir um JSON organizado, utilizando *FLex* para gerar um parser eficiente utilizando expressoes regulares.

Conteúdo

Capítulo 1

Introdução

Este projeto tem como objetivo o processamento de um ficheiro *HTML*, contendo os comentários de uma notícia, de forma extrair todos os dados relevantes e transforma-los num formato *JSON*.

Para o processamento deste ficheiro, foi escrito um filtro de texto utilizando o gerador *FLex*, e a linguagem de programação C.

Em primeiro lugar, iremos analizar o problema, determinar que funcionalidades implementar e identificar os desafios que serão necessários ultrapassar para implementar essas funcionalidades.

Em seguida, iremos analizar a nossa solução, a estrutura do *parser*, como foram escolhidos os subcontextos e que expressoes regulares foram escolhidas para os construir, e a estrutura do projeto onde iremos falar das estruturas de dados utilizadas.

Para concluir, iremos apresentar um pequeno manual de utilização do programa.

Capítulo 2

Problema

O programa tem que cumprir os seguintes requisitos:

- Converter o ficheiro de input para *UTF8*;
- Identificar no ficheiro de *input* os diversos comentários;
- Separar os diferentes elementos dos comentários;
- Garantir que as respostas a um comentário são guardadas de forma aninhada;
- Contar o número de respostas a cada comentário;

O ficheiro de *input* apresentou varíos problemas que precisa de ser tratados para conseguir cumprir os requisitos propostos.

O primeiro sendo a codificação do ficheiro, estando codificado em *latin1*, e sendo preciso converte-lo para *utf8*, de forma a que o output do programa não apareça com caracteres ilegiveis.

O segundo foi o facto de alguns dos comentários apresentados conterem mais do que uma linha, invalidando assim o *JSON* de output se não forem tratados.

Capítulo 3

Solução

3.1 Contextos de Processamento

Para processar o ficheiro com os comentários foram implementados 7 subcontextos: *DEFAULT*, *THREAD*, *CONTENT*, *ID*, *USER*, *DATE* e *REPLY*.

3.1.1 DEFAULT

Toda a zona correspondente ao contexto DEFAULT é ignorada, pois é a zona que contem a informação irrelevante ao programa.

3.1.2 THREAD

No THREAD é onde se encontra a informação relevante sobre os comentários, e onde estão contidos todos os subcontextos descritos abaixo.

3.1.3 CONTENT

Neste contexto é onde o conteudo de cada resposta é tratado e formatado, removendo linhas em branco, e escapando os caracteres *'\n'*, de forma a assegurar que o output do programa é válido.

3.1.4 ID

Aqui é onde é identificado o ID de cada resposta, ignorando todo o resto presente neste contexto.

3.1.5 USER

À semelhança do contexto acima descrito, o USER é onde são identificados os detalhes do utilizador criador do respetivo comentário.

3.1.6 DATE

No contexto DATE, é feito o processamento tanto da data do comentário, como a *timestamp* do mesmo.

3.1.7 REPLY

No contexto REPLY, estamos dentro das respostas de um comentário. Aqui, como o processamento é recursivo, voltamos a ter os contextos acima descritos, pois respostas são comentários normais.

3.2 Arquitetura do Projeto

Graças à simplicidade do projeto, foi-nos possível resolver o problema num só módulo, não comprometendo a legibilidade da solução. Neste módulo é tratado o input, selecionada a informação relevante presente e formatada para ir de encontro ao formato pretendido.

Quando a estruturas de dados, apenas mantemos em memória uma *stack* de contadores, para ser possível armazenar o número de respostas de um comentário, respeitando a recursividade do processamento no campo das respostas. Para a restante informação não nos foi necessário armazenar nada em memória, dando *flush* desta mal é processada.

Capítulo 4

Manual de Utilização

Este programa é de utilização simples, recebendo como argumento o ficheiro HTML a ser processado, podendo ser chamado desta forma: `./html2json path_to_input_file`.

Depois de processado, o JSON resultante irá ser retornado pelo *stdout*. Como o JSON não é formatado de uma forma muito legível, o programa pode ser encadeado com ferramentas processamento de JSON, como por exemplo, o *jq*, para facilitar a leitura do mesmo. Isto pode ser feito com o comando exemplo `./html2json path_to_input_file | jq`. O resultado da formatação deste será mais uma vez retornado pelo *stdout*.

Capítulo 5

Conclusão

Para conluir, fazemos um balanço bastante positivo do projeto, conseguindo atingir todos os requisitos definidos. A utilização de expressoes regulares para tratamento de strings mostrou ser algo extremamente poderoso, permitindo simplificar tarefas, que de outra forma seriam muito mais trabalhosas e demoradas.

Como trabalho futuro, gostaríamos de trabalhar mais a formatação do output para facilitar a leitura do mesmo.

Apêndice A

Flex

```
%%
<DEFAULT,THREAD>\<li\ class=\"comment\"\  {BEGIN THREAD; printf(",{"); rsc[cursor-1]++;}
<REPLY>\<li\ class=\"comment\"\           {BEGIN THREAD; rsc[cursor-1]++; }
<*>\<li\ class=\"comment\"\               {BEGIN THREAD; printf("{"); rsc[cursor-1]++;}
<THREAD>\<\/ol\>                          {char *bool = rsc[cursor] ? "true" : "false"; \
                                           char *reply = rsc[cursor] ? " " : "\"replies\": [], "; \
                                           printf(",%s\"hasReplies\": %s, \"numberOfReplies\": %d, \"likes\": 0}]"
\
                                           rsc[cursor--] = 0;}
<THREAD>\<\/form\>\r\n\<\/li\>\r\n        {char *bool = rsc[cursor] ? "true" : "false"; \
                                           char *reply = rsc[cursor] ? " " : "\"replies\": [], "; \
                                           printf(",%s\"hasReplies\": %s, \"numberOfReplies\": %d, \"likes\": 0}",
\
                                           rsc[cursor]=0; \
                                           BEGIN DEFAULT;}

<THREAD>data-comment-id=                  BEGIN ID;
<ID>[^\>]+                                printf("\"id\": %s,", yytext);
<ID>\>                                    BEGIN THREAD;

<THREAD>\<p\>[\ \r\n]*                    {BEGIN CONTENT; printf("\"commentText\": \"");}
<CONTENT>[^\"\r\n]                        ECHO;
<CONTENT>\"                               printf("\'");
<CONTENT>\r\n/\r\n                        {;}
<CONTENT>\r                               {;}
<CONTENT>\r\n\ *\<\/p\>\r\n               {BEGIN THREAD; printf("\"");}
<CONTENT>\n                               {printf("\\n");}

<THREAD>\<a\ href[^\>]+\>                 BEGIN USER;
<USER>[^\<]+/\                            printf("\"user\": \"%s\",", yytext);
<USER>\<\/a\>\r\n                         BEGIN THREAD;

<THREAD>datetime=\"                       BEGIN DATE;
<DATE>[0-9\-]+[^T]                        printf("\"date\": \"%s\",", yytext);
<DATE>[0-9:.]+[^\"]                       printf("\"timestamp\": \"%s\",", yytext);
<DATE>\>\r\n                              BEGIN THREAD;

<THREAD>\<ol\ class[^\>]*\>\r\n           {BEGIN REPLY; printf(",\"replies\": [{"); cursor++;}
<*>\<\/li\>\r\n/\<li                      {char *bool = rsc[cursor] ? "true" : "false"; \
                                           char *reply = rsc[cursor] ? " " : "\"replies\": [], "; \
                                           printf(",%s\"hasReplies\": %s, \"numberOfReplies\": %d, \"likes\": 0}",


<*>.|\n                                    {;}
%%
```