Trabalho Prático nº1
FLex

Sérgio Oliveira
a62134

10 de Outubro de 2019

Contents

Chapter 1

Introdução

Este relatório tem como objetivo mostrar o desenvolvimento de um programa escrito em formato Lex, que faz o processamento de linhas de um ficheiro passado para input, filtrando os conteúdos para as variáveis corretas.
Dos exemplos juntos em anexo com o enunciado do trabalho prático, fazemos uso do ficheiro em formato *XML* **exemplo-Enamex** para demonstração no 1º capítulo do relatório.
Para o 2º capítulo explica-se como um pré-processador HTML é desenvolvido. Um exemplo em formato *TXT* não anexado com o enunciado é usado no processamento de tags personalizadas e resumidas para gerar marcas conhecidas de **HTML** que criam um documento para facilitar a edição desse mesmo.

Chapter 2

Enamex

2.1 Nomes

A funcionalidade básica desta parte do trabalho está em identificar padrões da linguagem *Enamex* para processamento de palavras-chave, assim como nomes completos de pessoas, locais e cidades.

Depois de processado, é então gerada uma página HTML com toda a informação extraída do exemplo *Enamex*, filtrando a informação que nos é pedida.
Como no exemplo em anexo junto com o trabalho prático, existem diversos caractéres especiais vindos do tipo de linguagem *XML*, que geralmente começam sempre com o tag <**ENAMEX**>. Para conseguirmos finalizar essa parte do codigo temos de usar a mesma tag com uma barra antes da palavra /**ENAMEX**. Portanto primeiro temos de fazer o processamento dos mesmos. Para fazer a leitura de todos os nomes no documento Enamex, temos que fazer parsing das linhas que contenham as tags **ENAMEX TYPE**, **PERSON** e **NAME**. De seguida temos de procurar os nomes dentro das tags, fazendo um grupo de captura de todos os nomes que contenham caractéres específicos como se mostra no código extraído do trabalho.

```
92  PERSON      [<]ENAMEX[ ]TYPE[=]["]PERSON["][>]
93  NAME        [<]ENAMEX[ ]TYPE[=]["]NAME["][>]
94  PROCNOME    ([.]?[ ]?[\-\.ãa-zA-Z]+[.]?[ ]?)+
 . . .
97  {PERSON}{PROCNOME}[<]     {yytext[yyleng-1]='\0'; addNome(&yytext[22]);}
98  {PERSON}[ ]{PROCNOME}[<]     {yytext[yyleng-1]='\0'; addNome(&yytext[23]);}
99  {NAME}{PROCNOME}[<]       {yytext[yyleng-1]='\0'; addNome(&yytext[20]);}
100 {NAME}[ ]{PROCNOME}[<]       {yytext[yyleng-1]='\0'; addNome(&yytext[21]);}
```

Todos os nomes reconhecidos irão então ser passados como argumento para uma função em C *addNome()*, que guardará todo o processamento para um array de strings.

2.2 Ordenação

Outro objetivo indicado nas alíneas do enunciado seria ordenar todos os nomes por ordem alfabética. Depois de processar todos os nomes do exemplo e antes de criarmos o documento HTML, temos a função em C *ordenar()*.

A função, para além do algoritmo normal de sort de strings através de dois ciclos *for*, elimina também nomes completos que estão repetidos no nosso array de nomes. A desvantagem deste processo escolhido é que, ao verificar os nomes repetidos, também iremos percorrer duas vezes o array com mais dois ciclos *for*.

Primeiro, a função verifica o array, usando dois contadores:

```
71  while (nome[i]!=NULL && i<50) {count++;i++;}

72  for (i=0;i<count;i++){
73             for (j=i;j<count;j++){
74                if (strcmp(nome[i],nome[j])>0){
75                    strcpy (temp,nome[i]);
76                    strcpy (nome[i],nome[j]);
77                    strcpy (nome[j],temp);
78                }
79             }
80          }

81  for (i=0;i<count;i++){
82             for (j=i;j<count;j++){
83                if (strcmp(nome[i],nome[j])==0){
84                    strcpy (temp,nome[i]);
85                }
86             }
87          }
```

2.3 Países e cidades

Também temos de fazer a leitura de todos os países e cidades no nosso ficheiro exemplo. O processo é parecido com o processamento de nomes. Usámos o mesmo grupo de captura para processar os nomes que estão dentro de *tags*. A diferença está em procurarmos as etiquetas **COUNTRY** e **CITY** em vez das tags **PERSON** e **NAME**.

```
71  CITY      [<]ENAMEX[ ]TYPE[=]["]CITY["][>]
72  COUNTRY         [<]ENAMEX[ ]TYPE[=]["]COUNTRY["][>]
73  PROC   ([a-zA-Z\-\.ãõâáóéçD\.\n][ ]?)*
. . .

76  {CITY}{PROC}[<]     {yytext[yyleng-1]='\0'; addCidade(&yytext[20]);}
77  {COUNTRY}[ ]{PROC}[<]     {yytext[yyleng-1]='\0'; addPais(&yytext[23]);}
```

Na parte em linguagem C deste ficheiro Lex, teremos dois arrays de strings que irão servir para armazenar países e cidades então processadas pelas linhas acima referidas.

Chapter 3

WIKI

3.1 De FLex para HTML

3.2 Problemas

Chapter 4

Conclusão

Após dias de desenvolvimento, há que realçar as principais dificuldades. O facto de não poder usar os conjuntos de letras, dígitos e caractéres especiais (nomeadamente \d, \s e \w), devido a prováveis limitações de libraries e visto que dava problemas na compilação em FLex e gcc, foi posta de parte a introdução desses caracteres.

Após várias correcções e testes ao trabalho, corrigiu-se esse precalço.

Com o desenvolvimento destes projetos, haverá também sempre a expansão do nosso conhecimento e interesse em sistemas UNIX e UNIX-like, assim como em Expressões Regulares.