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 procesador usando o programa **FLex**, que faz a leitura das linhas de um ficheiro passado para input e gera conteúdo para uma linguagem específica, sendo **HTML** a especificada no enunciado do trabalho prático.

Com o apoio de vários tipos de estruturas de dados em C, é então armazenada e depois usada toda a informação necessária para a geração do documento final na linguagem indicada.

Dos exemplos juntos em anexo com o enunciado, faz-se uso do ficheiro em formato XML *exemplo-Enamex.xml* para demonstração no capítulo seguinte do relatório.

É depois pedido formar uma linguagem própria de anotação que faz uma abreviação da formatação conhecida do HTML para então facilitar a escrita por vezes cansativa das etiquetas. Um exemplo em formato *TXT* não anexado com o enunciado é então criado com base no que foi pedido, usado como input.

Um pré-processador para essa linguagem é criado para fazer o reconhecimento dos padrões da própria anotação e gerar depois marcas conhecidas de que criam um documento.

O desenvolvimento do processador é explicado no 2º capítulo deste relatório.

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 usaremos a mesma tag com uma barra antes da palavra </**ENAMEX**>. Portanto primeiro faz-se o processamento dos mesmos.
Antes do processamento, cria-se o ficheiro HTML onde irá conter toda a informação necessária. Para isso gera-se etiquetas HTML para inicializar corretamente o documento, junto com o título que será lido na parte do separador de um browser.

Para fazer a leitura de todos os nomes no documento Enamex, temos que criar padrões que combinem 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]);}
```

Ficheiro: EnameXPro1.l

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 percorre outras duas vezes o array com mais dois ciclos *for*, resultando num tempo de execução menos eficaz.

Primeiro, a função verifica se o array de strings está vazio. Os seguintes processos usarão dois contadores que vão ser incrementados à medida que encontra nomes inteiros no array.

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

Ficheiro: EnameXPro1.l

Os contadores anteriores são então usados nestes ciclos, que tem o objetivo de ordenar os nomes completos, usando o algoritmo simples bubble sort, comparando cada string no array e usando uma variável temporária.

```
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          }
```

Ficheiro: EnameXPro1.l

Os contadores são mais uma vez usados para verificar se existem nomes completos repetidos, comparando se duas strings são iguais.

```
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          }
```

Ficheiro: EnameXPro1.l

A função addNomes() adiciona então os nomes processados e ordenados para o ficheiro através do tipo C FILE.

Após serem todos adicionados, faz-se então o fecho do documento, gerando as tags necessárias para concluír o código HTML.

2.3 Países e cidades

Pretende-se então agora fazer o processamento de todos os países e cidades, usando mais uma vez como input o ficheiro *exemplo-Enamex.xml*. Na parte deste ficheiro (EnameXPro2.l), teremos dois arrays de strings que irão servir para armazenar países e cidades então processadas pelas linhas acima referidas.

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]);}
87  .                         ;
```

Ficheiro: EnameXPro2.l

Chapter 3

WIKI

3.1 Pré-processador HTML

Para a segunda parte do trabalho, é pedido um ficheiro em formato lex que faça o pré-processamento de abreviaturas de tags conhecidas de HTML que serão depois usados na geração de um novo ficheiro.

Uma função em C *inicializarHTML()* insere as tags *<html>* e *<head>* para dar início ao documento e dar também inicio ao título da página. Outras funções seguintes tratam de fechar estas etiquetas, que se vai falar mais tarde.

Para complementar a etiqueta *<head>* ao bocado referida, a função *addTitle()* adiciona o título que será mostrado no separador do browser.

A seguir está o padrão usado para processamento do título. um caractér especial *";"* dá o inicio ao título e outro conclui a linha.

```
126  TITLE [;][a-zA-Z0-9]+[;]
```

Ficheiro: Wiki.l

A indentação é a definição mais simples, usando só o padrão [:], em que o caractér ":" gera no documento final a string *"   "*, sendo * * um espaço em *HTML*. Se de seguida o padrão encontrar caractéres semelhantes, é então repetida a indentação as vezes que for necessária.

```
39   void addSpace (){
40           FILE *f;
41           f = fopen("wikipage.html","a");
42           if (f==NULL) exit(EXIT_FAILURE);
43           fputs("    ",f);
44           fclose(f);
45   }
...
127  INDENT          [:]
...
```

Ficheiro: Wiki.l

Os padrões negrito (´), itálico (/) e sublinhado (_) são muito semelhantes, sendo diferentes na maneira como o primeiro e último caractér são combinados. As etiquetas usadas são correspondentemente *<b>*, *<i>* e *<u>*.

```
128  BOLD            ['][ ]?[a-zA-Z0-9]+[ ]?)+[']
129  ITALIC          [/][ ]?[a-zA-Z0-9]+[ ]?)+[/]
130  SUBL            [_][ ]?[a-zA-Z0-9]+[ ]?)+[_]
```

Ficheiro: Wiki.l

é também usado um padrão para combinar newlines (\n) usados no ficheiro input e também o padrão [-+] para gerar um novo parágrafo. Ambos usam as etiquetas *
*.

```
131  PARAGRAPH       [\-\+]
132  LINEBREAK       [\n]
```

Ficheiro: Wiki.l

Listas simples e listas núméradas são partilhadas com o mesmo padrão para identificação do início e no fim das mesmas, gerando então as etiquetas HTML *
* e *
*.

```
133  BEGINULST       (\=>)
134  ENDULIST        (<\=)
135  UNORDLIST       [\*][ ]?[a-zA-Z0-9]+[ ]?)+[\*]
136  NUMERLIST       [\#][ ]?[a-zA-Z0-9]+[ ]?)+[\#]
```

Ficheiro: Wiki.l

A lista será núméradа se o padrão combinado tiver o caractér cardinal (#) no início e no fim da linha lida. A variável de tipo inteiro *numerlist* é então usada para gerar a numeração de cada item limitado com caractéres cardinais.

```
108         fputs("<li>",f);
109         fprintf(f,"%d",numerlist);
110         fputs(". ",f);
111         fputs(text,f);
112         fputs("</li>\n",f);
113         fclose(f);
114         numerlist++;
```

Ficheiro: Wiki.l

No fim do ficheiro input, a string *"</body></html>"* é então imprimida no documento HTML, finalizando o processamento do mesmo.

Chapter 4

Conclusão

Após dias de desenvolvimento, há que realçar as principais dificuldades como 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.

Houve também vários casos de falhas de pattern-matching de etiquetas no formato XML, fazendo mensagens de erro estranhas e elevando ainda mais a dificuldade para contornar essa situação. Após várias correcções e testes ao trabalho, corrigiu-se esse precalço.

Um dos fatores que pode levar a estas dificuldades é não estar totalmente familiarizado com a ferramenta FLex, tendo a sua própria gramática um pouco complicada.

No entanto, com o desenvolvimento destes projetos, haverá também sempre a expansão do nosso conhecimento e interesse em Expressões Regulares e na forma como ela é usada nos dias de hoje.