Processamento de Linguagens - MiEI (3^{o} ano)

Um Processador de Inglês Corrente, um Processador de DTDs e um Processador de *Named Entities* em Flex

Relatório de Desenvolvimento

Ana Isabel Castro (a55522) | Joana Miguel (a57127) | Lúcia Abreu (a71634)

23 de Abril de 2017

Resumo

Este documento consiste no relatório de desenvolvimento da segunda parte do primeiro trabalho prático de Processamento de Linguagens: desenvolvimento de um Processador de Inglês Corrente, um Processador de DTDs e um Processador de *Named Entities* em `FLex`.

Estes Processadores de Linguagens Regulares filtram e transformam textos, utilizando o `FLex` para gerar filtros de texto em `C`.

O Processador de Inglês Corrente analisa um texto escrito em inglês corrente e: expande as contrações que encontra e gera um lista ordenada em `HTML` de todos os verbos encontrados.

O Processador de DTDs permite analisar um ficheiro DTD que define o modelo de um ficheiro de informação em XML. Através deste processor consegue-se extraír todos os elementos definidos, os respetivos atributos e as depêndencias associdas a cada elemento.

O Processador de *Named Entities*, como próprio nome o indica, analisa ficheiros num dialecto `XML` denominado `ENAMEX` de forma a identificar com as etiquetas `ENAMEX` e `TIMEX` as Entidades referidas um texto através de nomes proprios (do tipo: Pessoa, País, Cidade) ou Datas, extraindo essa informação e reproduzindo-a numa página `HTML`.

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

A **segunda parte** do primeiro trabalho da unidade curricular de Processamento de Linguagens do 3º ano do Mestrado Integrado em Engenharia Informática consiste no desenvolvimento e implementação de Processadores de Linguagens Regulares, que filtrem ou transformem textos, utilizando para isso o `FLex` para gerar filtros de texto em `C`.

Os principais objetivos com este trabalho prático são:

- aumentar a experiência de uso do ambiente `Linux` e de algumas ferramentas de apoio à programação;
- aumentar a capacidade de escrever Expressões Regulares (ER) para descrição de padrões de frases;
- desenvolver, a partir de Expressões Regulares (ERs), sistemática e automaticamente Processadores de Linguagens Regulares, que filtrem ou transformem textos com base no conceito de regras de produção Condição → Ação;
- utilizar o `FLex` para gerar filtros de texto em `C`.

Para tal, foram abordados três problemas/enunciados:

- `Processador de Inglês Corrente`
- `Processador de DTDs`
- `Processador de Named Entities`

Estrutura do Relatório

O Capítulo 2 explica a arquitetura da solução do projeto.

Os Capítulos 3, 4 e 5 referem-se ao **Processador de Inglês Corrente**, **Processador de DTDs** e **Processador de Named Entities**, respetivamente. Estes capítulos estão divididos em seis subcapítulos:

- **Enunciado**
- **Descrição do Problema**
- **Desenho da solução**
- **Especificação**
- **Implementação**
- **Testes e Resultados**

O Capítulo 6 refere-se à **Conclusão** na qual se sumariza brevemente o que se concretizou com este trabalho.

Em **Apêndice** seguem os filtros de texto implementados e os ficheiros fonte para testes, para todos os problemas resolvidos.

Capítulo 2

Arquitetura da solução

Os Processadores desenvolvidos neste projeto foram implementados em `FLex`. A partir de um ficheiro `FLex`, com uma especificação válida, o sistema `FLex` produz um programa em `C`. Por defeito, este programa tem o nome `lex.yy.c`, que pode ser alterado utilizando a opção do sistema `FLex`: `flex -o ficheiro.yy.c ficheiro.l`.

Assim, aplica-se o seguinte comando:

```
flex ficheiro.fl
```

O ficheiro `lex.yy.c` foi gerado e encontra-se no sistema de ficheiros. Após produzir o programa `C`, é possível produzir um programa executável compilando esse programa:

```
gcc -c lex.yy.c
```

O programa gerado - `a.out` - recebe o texto fonte do *standard input*. Assim, é utilizado o seguinte comando para executar o programa:

```
a.out < teste.txt
```

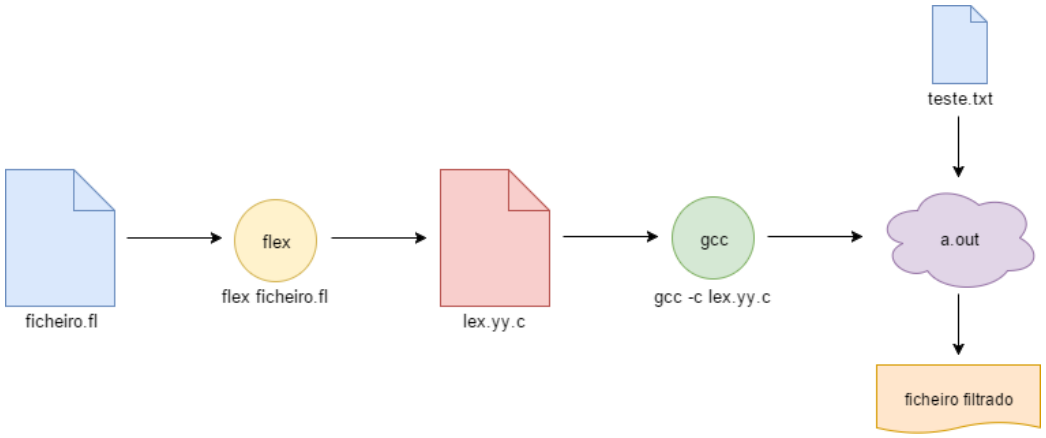

Figura 2.1: Esquema representativo da arquitetura geral de um filtro em FLex

Capítulo 3

Processador de Inglês Corrente

3.1 Enunciado

Com um processador de inglês corrente pretende-se que o mesmo leia um texto escrito em inglês corrente e que contenha contrações do tipo:

- *I'm*
- *I'll*
- *We're*
- *Can't*
- *It's*

Entre outras, que serão observadas mais adiante.
Pretende-se que após a leitura do mesmo, o processador:

- Reproduza o texto de entrada na saída expandido as contrações que encontrar nesse texto.
- Gere em `HTML` um lista ordenada de todos os verbos que encontrar no texto, no infinitivo não flexionado. Para tal é preciso ter-se em atenção que:
  - Este tipo de forma verbal em inglês é precedida pela palavra *to*.
  - Essa palavra é retirada quando o verbo é precedido de um *modal verb*: *can, could, may, might, must, shall, will, should, would*.
  - Na forma interrogativa o verbo no infinitivo é precedido por *do, does, did*.

Pretende-se o uso de variados ficheiros fonte para testes e resultados.

3.2 Descrição do problema

3.2.1 Expandir as Contrações

O inglês corrente é uma língua que usa muitas contrações - *contracted forms* - que basicamente são uma forma mais fácil de escrever certas palavras conectando-as, e que facilitam principalmente o diálogo. São frequentemente usadas num discurso informal, e por isso, corrente.

As contrações envolvem a elisão de uma vogal, e a inserção de um apóstrofo no seu lugar, havendo por vezes também outras mudanças. Muitas das contrações envolvem a negação de verbos auxiliares. Por exemplo:

It's é a contração de *It is*
Won't é a contração de *Will not*

Na língua inglesa são utilizadas 141 contrações, ainda que algumas são mais frequentemente usadas que outras.

É necessário então entender que tipo de contrações existem para que mais tarde se possa especificar um padrão através de uma expressão regular:

- **Negações**: palavra seguida de *not*

  Exemplos:

  - *Are not* → *Aren't*
  - *Could not* → *Couldn't*
  - *Will not* → *Won't*

- **Verbo *to have***: palavra seguida de *have*

  Exemplos:

  - *Could have* → *Could've*
  - *Must have* → *Must've*
  - *They have* → *They've*

- **Verbo modal *would***: palavra seguida de *would*

  Exemplos:

  - *He would* → *He'd*
  - *She would* → *She'd*
  - *We would* → *We'd*

- **Verbo modal *will***: palavra seguida de *will*

  Exemplos:

  - *He will* → *He'll*
  - *How will* → *How'll*
  - *What will* → *What'll*

- **Conjugações do verbo *to be***: palavra seguida de *am*, *are* ou *is*

  Exemplos:

  - *I am* → *I'm*
  - *There are* → *There're*
  - *When is* → *When's*

- **Casos particulares**

  Exemplos:

  - *Shall not* → *Shan't*
  - *Will not* → *Won't*
  - *Going to* → *Gonna*
  - *Got to* → *Gotta*
  - *You all* → *Y'all*

Existe outro caso particular - o *ain't* - que, por ser um substituto para um grande número de contrações (*am not, is not, are not, has not, have not*) e a sua expansão depender da pessoa do discurso e do tempo verbal, a sua contração não será expandida.

3.2.2 Unidades de Medida

Como problema adicional, achou-se útil expandir as abreviaturas ou símbolos das unidades de medida para as suas respetivas denominações.

Como se trata de um processador de inglês corrente, decidiu-se apenas abordar as unidades de medida mais conhecidas e usadas.

Assim sendo, abordou-se as unidades de **comprimento**, **massa** e **tempo** do Sistema Internacional de unidades (SI). Decidiu-se colocar de parte as unidades de corrente elétrica (ampère), temperatura (kelvin), quantidade de matéria (mol) e intensidade luminosa (candela) do SI por não serem tão frequentemente usadas no quotidiano, sendo por isso mais raras no inglês informal e corrente abordado neste projeto.

Foram também abordadas **outras unidades de medida** que não fazem parte do Sistema Internacional de unidades, mas sim do Sistema Imperial, mais conhecido por *unidade inglesa*. Unidades que, como o próprio nome indica, são frequentemente usadas em países de língua inglesa.

Para cada grandeza, como existem vários múltiplos e submúltiplos, foram escolhidos mais uma vez apenas aqueles mais frequentemente usados no inglês corrente.

Unidades de Comprimento

- **Metre (m)**
  Submúltiplos:
  - *Decimetre (dm)*
  - *Centimetre (cm)*
  - *Millimetre (mm)*
  - *Nanometre (nm)*

Unidades de Massa

- **Kilogram (kg)**
  Submúltiplos:
  - *Gram (g)*
  - *Milligram (mg)*

Unidades de Tempo

- **Second (s) or (sec)**
  Submúltiplos:
  - *Millisecond (ms)*
  - *Nanosecond (ns)*
  - *Hour (h)* $\rightarrow$ *3600 seconds*
  - *Minutes (min)* $\rightarrow$ *60 seconds*

Outras unidades de medida

- *Miles (ml)*
- *Inches (in)*
- *Feet (ft)*
- *Yard (yd)*
- *Gallons (gal)*
- *Horsepower (hp)*
- *Grain (gr)*
- *Ounces (oz)*

- *Pounds (lb)*
- *Teaspoons (tsp)*
- *Tablespoons (tbsp)*

3.2.3 Listagem de Verbos

Pretende-se gerar uma lista ordenada de todos os verbos (no infinitivo não flexionado) que se encontram num texto. É importante então saber como os verbos ocorrem no texto:

- Este tipo de forma verbal (infinitivo não flexionado) em inglês é precedida pela palavra *to*.

  Exemplos:

  - *to be*
  - *to eat*
  - *to sleep*

- A palavra *to* é retirada quando o verbo é precedido de um *modal verb*: *can, could, may, might, must, shall, will, should, would*.

  Exemplos:

  - *I **can take** you home.*
  - *We **could go** on a trip.*
  - *The students **must behave** as I say.*

  Porém, no caso dos *modal verbs* no modo interrogativo, existe um pronome pessoal entre o *modal verb* e o verbo que queremos listar.

  Exemplos:

  - *May **I** use your umbrella?*
  - *Shall **we** go for a drink after work?*
  - *Will **I** wait a lonely lifetime?*

- Na forma interrogativa o verbo no infinitivo é precedido por *do, does, did*.

  Exemplos:

  - *Do you have your homework ready?*
  - *What material does she like?*
  - *Did you take your vitamin this morning?*

  Porém, como é possível observar no primeiro exemplo, entre a palavra *do* e o verbo existe sempre um pronome pessoal, neste caso o pronome *you*.

- Adicionalmente abordou-se a forma negativa, em que a palavra *not* fica entre o *modal verb* e o verbo que queremos listar.

  Exemplos:

  - *Sorry, I can **not** understand what you are saying.*
  - *You must **not** be from around here.*
  - *I shall **not** walk alone.*

3.3 Desenho da solução

3.3.1 Arquitetura

Os processadores desenvolvidos foram implementados em `FLex`, e dos três problemas abordados, tomou-se a decisão de implementar dois processadores separados:

- **Processador de Inglês** → `processador_ingles.l`
  - Expansão das Contrações
  - Unidades de Medida
- **Processador de Verbos** → `processador_verbos.l`
  - Listagem dos Verbos

Que geram um ficheiro em `HTML` (Processador de Verbos) ou imprimem o resultado para o *standard output* ou para um ficheiro `.txt` (Processador de Ingles).

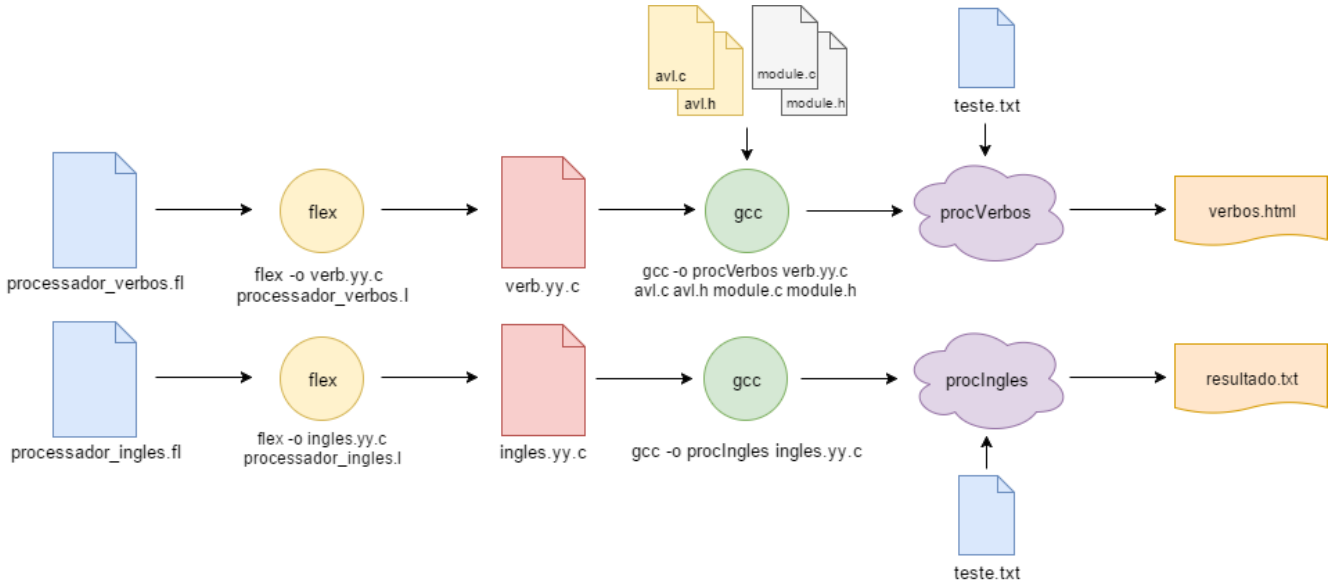

Figura 3.1: Esquema representativo da arquitetura do Processador de Inglês Corrente em FLex

3.4 Especificação

3.4.1 Especificar os padrões das Expressões Regulares

Esta etapa consistiu em especificar os padrões através de expressões regulares que permitam capturar os campos pretendidos no texto-fonte.

Expandir as Contrações

No ínicio da especificação deste problema da expansão das contrações, observou-se que existem dois tipos de pelicas:

- `Aren’t`
- `Aren't`

Por esse motivo, as expressões regulares vão ter este pormenor em conta, fazendo *match* com contrações que tenham qualquer uma das pelicas existentes.

Apresentam-se então as especificações das expressões regulares da expansão das contrações:

- **Negações**
  - `n('|')t` → faz *match* com a parte da palavra que corresponde à negação.
    Exemplo: Aren’t.
- **Verbo *to have***
  - `('|')ve` → faz *match* com a parte da palavra que corresponde ao *have*.
    Exemplo: Could’ve.
- **Verbo modal *would***
  - `('|')d` → faz *match* com a parte da palavra que corresponde ao *would*.
    Exemplo: She’d.
- **Verbo modal *will***
  - `('|')ll` → faz *match* com a parte da palavra que corresponde ao *will*.
    Exemplo: We’ll.
- **Conjugações do verbo *to be***
  - `('|')s` → faz *match* com a parte da palavra que corresponde ao *is*.
    Exemplo: It’s.
  - `('|')m` → faz *match* com a parte da palavra que corresponde ao *am*.
    Exemplo: I’m.
  - `('|')re` → faz *match* com a parte da palavra que corresponde ao *are*.
    Exemplo: You’re.

- **Casos particulares**
  - `shan('|')t` → faz *match* com a palavra *shan't* (que corresponde a *shall not*).
  - `won('|')t` → faz *match* com a palavra *won't* (que corresponde a *will not*).
  - `gonna` → faz *match* com a palavra *gonna* (que corresponde a *going to*).
  - `gotta` → faz *match* com a palavra *gotta* (que corresponde a *got to*).
  - `y('|')ll` → faz *match* com a palavra *y'all* (que corresponde ao *you all*).
  - `ain('|')t` → faz *match* com a palavra *ain't*, que não será expandida.

Unidades de Medida

Como o objetivo é encontrar as abreviaturas de unidades de medidas, e como sabemos que estas são precedidas de algarismos, temos a seguinte expressão regular:

- `[0-9]+\ *` → faz *match* com uma sequência de algarismos, seguida de zero ou mais espaços.

Unidades de Comprimento:

- `m` → faz *match* com a abreviatura `m` (*metre*).
- `dm` → faz *match* com a abreviatura `dm` (*decimetre*).
- `cm` → faz *match* com a abreviatura `cm` (*centimetre*).
- `mm` → faz *match* com a abreviatura `mm` (*millimetre*).
- `nm` → faz *match* com a abreviatura `nm` (*nanometre*).

Unidades de Massa:

- `g` → faz *match* com a abreviatura `g` (*gram*).
- `mg` → faz *match* com a abreviatura `mg` (*milligram*).
- `kg` → faz *match* com a abreviatura `kg` (*kilogram*).

Unidades de Tempo:

- `(s|sec)` → faz *match* com a abreviatura `s` ou `sec` (*seconds*).
- `ms` → faz *match* com a abreviatura `ms` (*milliseconds*).
- `ns` → faz *match* com a abreviatura `ns` (*nanoseconds*).
- `min` → faz *match* com a abreviatura `min` (*minutes*).
- `h` → faz *match* com a abreviatura `h` (*hours*).

Outras unidades de medida:

- `ml` → faz *match* com a abreviatura `ml` (*miles*).
- `(in|\")` → faz *match* com a abreviatura `in` ou o símbolo `"` (*inches*).
- `(ft|\')` → faz *match* com a abreviatura `ft` ou o símbolo `'` (*feet*).
- `yd` → faz *match* com a abreviatura `yd` (*yards*).
- `ac` → faz *match* com a abreviatura `ac` (*acres*)..
- `gal` → faz *match* com a abreviatura `gal` (*gallons*).
- `hp` → faz *match* com a abreviatura `hp` (*horsepower*).
- `gr` → faz *match* com a abreviatura `gr` (*grain*).
- `oz` → faz *match* com a abreviatura `oz` (*ounces*).
- `lb` → faz *match* com a abreviatura `lb` (*pounds*).
- `mph` → faz *match* com a abreviatura `mph` (*miles per hour*).
- `tsp` → faz *match* com a abreviatura `tsp` (*teaspoons*).
- `tbsp` → faz *match* com a abreviatura `tbsp` (*tablespoons*).

Listagem de Verbos

Como se pretende obter uma lista com os verbos no infinitivo não flexionado, que ocorrem nos moldes explicados no **subcapítulo 3.2.3**, as expressões regulares utilizadas têm como objetivo encontrar esses verbos nas suas diversas ocorrências.

Para simplificar as expressões regulares, definiu-se três *name definitions*:

- `modalVerb` `?i:(can|could|may|might|must|shall|will|should|would)`

  Que define qualquer um dos *modal verbs* como sendo um `modalVerb`, sendo que a expressão é *case insensitive*.

- `pronoun` `?i:(i|you|he|she|it|we|you|they)`

  Que define qualquer um dos pronomes pessoais como sendo um `pronoun`, sendo que a expressão é também *case insensitive*.

- `question` `?i:("do"|does|did)`

  Que define qualquer uma das conjugações do verbo *to do* (*do, does, did*) como sendo uma `question`, sendo que a expressão é *case insensitive*.

Apresentam-se então as especificações das expressões regulares:

- `\ to` → permite detetar apenas as palavras *to* que precedem os verbos no infinitivo não flexionado, garantido que é precedida de um espaço para que não sejam apanhadas as palavras *to* que façam parte de outras palavras.
- `{modalVerb}` → permite detetar as palavras que são verbos modais (*can, could, may, might, must, shall, will, should, would*).
- `{question}\ {pronoun}` → permite detetar os pronomes pessoais que são precedidos das palavras *do, does, did.*
- `\ {pronoun}` → permite detetar os pronomes pessoais que são precedidos de um espaço.
- `\ "not"` → permite detetar as palavras *not* precedidas de um espaço.
- `\ [a-zA-Z]+` → permite detetar uma sequência de um ou mais caracteres de A a Z (case insensitive), precedida de um espaço.
- `.|\n` → permite detetar qualquer caractere exceto `\n` ou o caractere `\n`.

3.4.2 Identificar as Acções Semânticas

Para cada uma das Expressões Regulares Especificadas, definiram-se as seguintes ações semânticas:

Expandir as Contrações

- **Negações**
  - `n('|')t` → quando faz *match* com a palavra `n't`, imprime `" not"` no lugar dela.
    Exemplo: Aren't → Are not.
- **Verbo *to have***
  - `('|')ve` → quando faz *match* com a palavra `'ve`, imprime `" have"` no lugar dela.
    Exemplo: Could've → Could have.
- **Verbo modal *would***
  - `('|')d` → quando faz *match* com a palavra `'d`, imprime `" would"` no lugar dela.
    Exemplo: She'd → She would.
- **Verbo modal *will***
  - `('|')ll` → quando faz *match* com a palavra `'ll`, imprime `" will"` no lugar dela.
    Exemplo: We'll → We will.

- **Conjugações do verbo *to be***
  - `('|')s` → quando faz *match* com a palavra `'s`, imprime `" is"` no lugar dela.
    Exemplo: It's → It is.
  - `('|')m` → quando faz *match* com a palavra `'m`, imprime `" am"` no lugar dela.
    Exemplo: I'm → I am.
  - `('|')re` → quando faz *match* com a palavra `'re`, imprime `" are"` no lugar dela.
    Exemplo: You're → You are.
- **Casos particulares**
  - `shan('|')t` → quando faz *match* com a palavra `shan't`, imprime `"shall not"`.
  - `won('|')t` → quando faz *match* com a palavra `won't`, imprime `"will not"`.
  - `gonna` → quando faz *match* com a palavra `gonna`, imprime `"going to"`.
  - `gotta` → quando faz *match* com a palavra `gotta`, imprime `"got to"`.
  - `y('|')ll` → quando faz *match* com a palavra `y'all`, imprime `"you all"`.
  - `ain('|')t` → quando faz *match* com a palavra `ain't` não realiza nenhuma ação semântica, pelos motivos explicados no **subcapítulo 3.2.1**.

Unidades de Medida

Com o objetivo de se encontrar todas as abreviaturas de unidades de medida, foi necessário recorrer ao uso de *start conditions*. As ***start conditions*** são declaradas na primeira secção do ficheiro de input - a secção de *definitions* - recorrendo a linhas não identadas que começam com `%x` seguidas de uma lista de nomes.

A *start condition* é ativada usando a ação `BEGIN`. Até que o próximo `BEGIN` seja executado, as regras da *start condition* atual estão ativas e as regras das restantes inativas.

Neste problema é definida uma *start condition*:

- `<UNIDADE>` → Sempre que é detetada uma sequência de algarismos seguida de zero ou mais espaços, é ativada esta *start condition*. Depois de ativada, e se for detectada uma das abreviaturas de unidades de medida, imprime a unidade de medida por extenso.

Unidades de Comprimento:

- `<UNIDADE>m` → quando faz *match* com a abreviatura `m`, imprime `"metres"`.
- `<UNIDADE>dm` → quando faz *match* com a abreviatura `dm`, imprime `"decimetres"`.
- `<UNIDADE>cm` → quando faz *match* com a abreviatura `cm`, imprime `"centimetres"`.
- `<UNIDADE>mm` → quando faz *match* com a abreviatura `mm`, imprime `"millimetres"`.
- `<UNIDADE>nm` → quando faz *match* com a abreviatura `nm`, imprime `"nanometres"`.

Unidades de Massa:

- `<UNIDADE>g` → quando faz *match* com a abreviatura `g`, imprime `"grams"`.
- `<UNIDADE>mg` → quando faz *match* com a abreviatura `mg`, imprime `"milligrams"`.
- `<UNIDADE>kg` → quando faz *match* com a abreviatura `kg`, imprime `"kilograms"`.

Unidades de Tempo:

- `<UNIDADE>(s|sec)` → quando faz *match* com a abreviatura `s` ou `sec`, imprime `"seconds"`.
- `<UNIDADE>ms` → quando faz *match* com a abreviatura `ms`, imprime `"milliseconds"`.
- `<UNIDADE>ns` → quando faz *match* com a abreviatura `ns`, imprime `"nanoseconds"`.
- `<UNIDADE>min` → quando faz *match* com a abreviatura `min`, imprime `"minutes"`.
- `<UNIDADE>h` → quando faz *match* com a abreviatura `h`, imprime `"hours"`.

Outras unidades de medida:

- `<UNIDADE>ml` → quando faz *match* com a abreviatura `ml`, imprime `"miles"`.
- `<UNIDADE>(in|\")` → quando faz *match* com a abreviatura `in` ou símbolo `"`, imprime `"inches"`.
- `<UNIDADE>(ft|\')` → quando faz *match* com a abreviatura `ft` ou o símbolo `'`, imprime `"feet"`.
- `<UNIDADE>yd` → quando faz *match* com a abreviatura `yd`, imprime `"yards"`.
- `<UNIDADE>ac` → quando faz *match* com a abreviatura `ac`, imprime `"acres"`.
- `<UNIDADE>gal` → quando faz *match* com a abreviatura `gal`, imprime `"gallons"`.
- `<UNIDADE>hp` → quando faz *match* com a abreviatura `hp`, imprime `"horsepower"`.
- `<UNIDADE>gr` → quando faz *match* com a abreviatura `gr`, imprime `"grains"`.
- `<UNIDADE>oz` → quando faz *match* com a abreviatura `oz`, imprime `"ounces"`.
- `<UNIDADE>lb` → quando faz *match* com a abreviatura `lb`, imprime `"pounds"`.
- `<UNIDADE>mph` → quando faz *match* com a abreviatura `mph`, imprime `"miles per hour"`.
- `<UNIDADE>tsp` → quando faz *match* com a abreviatura `tsp`, imprime `"teaspoons"`.
- `<UNIDADE>tbsp` → quando faz *match* com a abreviatura `tbsp`, imprime `"tablespoons"`.

Listagem de Verbos

Com o objetivo de se gerar uma lista ordenada de todos os verbos - no infinitivo não flexionado - encontrados num texto, foi necessário recorrer ao uso de *start conditions*.

Neste problema são definidas duas *start conditions*:

- `<FRASE>` → Sempre que é detetado um *modal verb*, é ativada a *start condition* `<FRASE>`, sendo que depois de ser ativada, as regras desta *start condition* irão verificar se:
  - Existe um `pronoun` (pronome pessoal) após o *modal verb*. Se tal se verificar, ativa a *start condition* `<GUARDAR>`.
  - Se encontra na forma negativa, em que a palavra `not` fica após o *modal verb*. Se tal se verificar, ativa a *start condition* `<GUARDAR>`.
  - Se nenhuma das hipóteses anteriores se verificar, encontrou-se o verbo pretendido. É então inserido na estrutura de dados, e ativada a *start condition* `<INITIAL>`.
- `<GUARDAR>` → Esta *start condition* é ativada sempre que:
  - É detetada a palavra *to*.
  - É detetada uma `question` seguida de um `pronoun`. Ex: *Did she*.
  - Dentro da *start condition* `<FRASE>`:
    - É detetado um `pronoun` (pronome pessoal).
    - É detetada a palavra `not` (forma negativa).

    Quando esta *start condition* é ativada, sabe-se que foi encontrado o verbo pretendido. É então inserido na estrutura de dados, e ativada a *start condition* `<INITIAL>`.

Apresentam-se então as ações semânticas:

- `{modalVerb}` → ativa a *start condition* `<FRASE>`.
- `\ to` → ativa a *start condition* `<GUARDAR>`.
- `{question}\ {pronoun}` → ativa a *start condition* `<GUARDAR>`.
- `<FRASE>\ {pronoun}` → ativa a *start condition* `<GUARDAR>`.
- `<FRASE>\ "not"` → ativa a *start condition* `<GUARDAR>`.
- `<FRASE>\ [a-zA-Z]+` → como encontrou o verbo pretendido:
  - guarda-o na variável `verbo`
  - coloca um `\0` no final do verbo
  - coloca o verbo todo a minúsculas
  - insere-o na estrutura de dados, árvore balanceada, `lista`
  - ativa a *start condition* `<INITIAL>`.
- `<GUARDAR>\ [a-zA-Z]+` → como encontrou o verbo pretendido:
  - guarda-o na variável `verbo`
  - coloca um `\0` no final do verbo
  - coloca o verbo todo a minúsculas
  - insere-o na estrutura de dados, árvore balanceada, `lista`
  - ativa a *start condition* `<INITIAL>`.
- `.|\n` → a ação semântica não faz nada.

3.4.3 Identificar as Estruturas de Dados globais

Dos três problemas abordados, só foi necessária a utilização de Estruturas de Dados globais no problema da **Listagem de Verbos**.

Listagem de Verbos

Visto que neste problema o objetivo é encontrar verbos num texto fonte, e apresentar a lista dos verbos encontrados por ordem alfabética, foi necessário recorrer-se a uma estrutura de dados global:

- `AVL lista`

Em que cada nodo da árvore é uma *struct* que guarda o **nome** do verbo e respetivo número de **ocorrências** à medida que concorda com o padrão, permitindo guardar os verbos de todo o ficheiro fonte.

```
typedef struct listaVerbos{
    char *nome;
    int ocorrencias;
} *VERBO;
```

Quando há um *match*, antes de se inserir na `AVL`, é verificado se o verbo já se encontra na estrutura de dados. Se já existe, incrementa-se as ocorrências na respetiva *struct* do verbo, se não existe, o verbo é inserido na `AVL` com ocorrência `1`. Tal encontra-se representado na **Figura 3.1**.

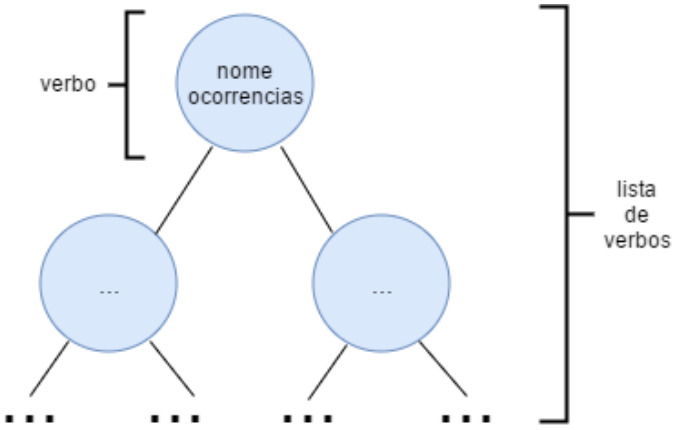

Figura 3.2: Esquema representativo da AVL que armazena os verbos

3.5 Implementação

Dos três problemas abordados, tomou-se a decisão de implementar dois processadores separados:

- **Processador de Inglês** → `processador_ingles.l`

  Onde se implementam os problemas:

  - Expansão das Contrações
  - Unidades de Medida

- **Processador de Verbos** → `processador_verbos.l`

  Onde se implementa o problema:

  - Listagem dos Verbos

Processador de Inglês

A implementação deste processador não exigiu qualquer complexidade, todas as ações semânticas são *printfs* para o *standard output*. Este processador não possui quaisquer módulos adicionais ou implementação em `C`.

Processador de Verbos

Visto ser necessário encontrar verbos num texto fonte, e guardá-los por ordem alfabética num ficheiro `HTML`, este processador exigiu alguma complexidade:
Para a implementação deste problema, foram necessários:

- Uma biblioteca para manipulação `AVL`, sendo necessários:
  - `avl.c`
  - `avl.h`
- O filtro de texto, em `FLex`:
  - `processador_verbos.l`
- A restante implementação do processador, em `C`, sendo necessários:
  - `module.c`
  - `module.h`

O `module.c` possui as seguintes funções auxiliares em `C` que complementam o processamento do ficheiro fonte:

- Antes de um verbo ser guardado, é colocado todo em minusculas:
  - `char* minusculas( char *t );`
- E só depois inserido na estrutura de dados:
  - `void insereVerbo( char *v , AVL lista );`
    Primeiro, é verificado se o verbo já existe na estrutura. Se existir, é incrementado o respetivo número de ocorrências na sua *struct*:
    - `void incrementaOcorrencias(VERBO verb, int num);`

    Se não existir, é criada a respetiva *struct* e inserido na estrutura:
    - `VERBO newVerb(char *nome, int ocorrencias);`
- Para gerar o ficheiro `HTML` a seguinte função é chamada na *main*:
  - `void geraListaVerbos(AVL avl);`

  Esta função abre um ficheiro para escrita, e vai imprimindo as linhas de código `HTML` correspondentes à *head*, *title*, *body*, etc. Para cada linha da tabela um ciclo imprime o verbo e a respetiva ocorrência.

3.6 Testes e Resultados

Para cada problema abordado, realizaram-se os seguintes testes e respetivos resultados:

3.6.1 Expandir as Contrações

Resultado do teste com o ficheiro em **apêndice B.1.1**.

```
Ca not we or will not we create jobs?
We have met before! No, we have not. I would have remembered.
Obviously you did not, because we have.
I think they are very nice boys. It was not me!
I shall not dignify that with an answer.
If it was not for love we would have faded away.
Why would you let me race with your car?
What are you up to? I am doing the dishes!
I am not going to tell you anything. It is going to rain.
All you all need to know about cars.
You have already been there. We will be coming back.
He must not have realised we would be here.
They would have not been so big! What are you up to?
You would have thought he would have learned that by now.
There would have been a time when I would have had his back.
She would have made me pay again.
You are not who I thought you were. He would not that!
I would not have done that if I were you! I ought not fight.
Joe could not keep up with the other guys.
We shall not get away before evening.
```

3.6.2 Unidades de Medida

Resultado do teste com o ficheiro em **apêndice B.1.2**.

```
The road was originally about 9.0metres wide.
If your step length is 70 centimetres and your pedometer has recorded 5000
secondsteps, the distance calculation is 0.7metres x 5000 secondsteps = 350
We can replace 628 nanometres with 6.28 x 10-7 metres.
I got a 9 millimetres. Ready to go off any minute so you feel it.
She saved 20 kilograms of potatoes a year for each son.
The pill has been identified as zolpidem 10 milligrams.
Please consider that a single can of soft drink contains 35 to 40 grams of
The last 10 seconds of his life.
Assume each frame is equal to 16.6666 milliseconds.
Just 1hours and 30minutes to go people!
System update in 43 nanoseconds.
```

```
I am going to be 500 miles away from you.
The length is a very portable 11.8 inches.
The finished surface (8-16inches) is exceedingly smooth.
The Port has a depth of 12.19 metres, corresponding to a draught of 40 feet
My grandmother is only 5feet tall.
Turn left and after 100 yards turn right at the roundabout into Paisley Roa
All 128 acres are covered with seaside paspalum.
These fish need at least 1 teaspoons of salt per 5 gallons of water to be h
The gigantic Queen Mary 2 cruisecondse liner is propelled by four engines,
As at 31 Decemetresber 2007 thourse ECB held 18,091,733 ounces of fine gold
The vehicle can carry more than 300pounds of cargo.
Why did the wind reach 36.5miles per hour?
Combine a can of condensed milk, 4 tablespoons of powdered chocolate and 3
```

3.6.3 Listagem de Verbos

Resultado do teste com o ficheiro em **apêndice B.1.3**.

Lista de Verbos e Suas Ocorrencias

Verbos	Ocorrencias
arrive	1
be	4
behave	1
cause	1
cry	1
go	3
leave	1
ride	1
see	1
speak	1
suffice	1
take	1
understand	1
use	2
wait	1

Letra da música *Abba - The Winner Takes It All*

Resultado do teste com a letra da música *Abba - The Winner Takes It All.*

Lista de Verbos e Suas Ocorrencias

Verbos	Ocorrencias
be	1
complain	1
decide	1
fall	3
feel	1
kiss	2
know	1
more	1
play	1
say	2
shake	1
throw	1

Capítulo 4

Processador de DTDs

4.1 Enunciado

Um *Document Type Definition* (DTD) é um ficheiro de notação própria que define a estrutura, os elementos e os atributos permitidos de um documento XML.

Neste trabalho pretende-se escrever um processador que receba um DTD e gere a lista (ordenada alfabeticamente) de todas os elementos definidos. Além disso, para cada elemento obter:

- os seus atributos
- as suas dependências

Pretende-se, também, usar a linguagem `Dotty` e o processador `dot` de forma a criar um grafo com todos os elementos de que depende um determinado elemento.

Para tal, começa-se por definir a estrutura de um documento `DTD`.

4.2 Descrição do problema

Começa-se por compreender de que forma se estrutura um ficheiro DTD.

Conhecer e perceber a estrutura deste tipo de ficheiros é essencial para estabelecer corretamente o filtro requerido.

4.2.1 Estrutura de um DTD

Um DTD é um "Document Type Definition".

Através de um DTD é possível estabelecer um padrão a nível estrutural para troca de dados em XML.

Um DTD define a estrutura, os componentes permitidos de um documento XML e, por isso, é denominado XML DTD.

O XML DTD pode ser especificado dentro ou fora do documento e a sua sintaxe básica consiste:

```
<!DOCTYPE elemento DTD identificador
[
   declaração1
   declaração2
   ........
]>
```

Sendo que:

- `<!DOCTYPE` → início do DTD
- `elemento` → indica ao *parser* onde deve começar a analisar
- `DTD identificador` → define o tipo de documento
- `parêntesis retos [ ]` → delimitam a lista de declarações

Portanto, o DTD consiste numa série de declarações. As declaraçãoes são compostas por componentes do XML.

Assim, cada uma das declarações pode ser:

- **ELEMENT** → Elemento

  O elemento XML pode ser considerado como um bloco de construção de um documento XML. Podem armazenar texto, outros elementos, atributos, definir objetos mídia ou consistir numa combinação de todos.

- **ATTLIST** → Lista de Atributos

  Um atributo está sempre associado a um elemento. Um elemento tem uma lista de atributos (que pode ser vazia). Os atributos fornecem mais informações sobre o elemento XML ou podem definir numa propriedade do elemento.

- **ENTITY** → Entidade

  As entidades são caracteres especiais reservados em XML. Consistem num mecanismo de substituição simples que se aplica no contexto de marcação de acordo com esse documento. Podem ser inseridos no texto dos elementos.

Num ficheiro DTD, cada um dos componentes deve ser especificado seguindo uma determinada estrutura:

- **Elemento**

```
<!ELEMENT element-name category>
ou
<!ELEMENT element-name (element-content)>
```

sendo que:

- `ELEMENT` → indica que se está a definir um elemento
- `element-name` → indica o nome do elemento
- `category` → define o tipo do elemento, ou seja, o que o elemento pode conter. A categoria poderá ser `EMPTY` ou `ANY` como se pode consultar na Tabela 4.1.

Tabela 4.1: Categorias possiveis definir para um elemento

	Descrição	**Exemplo**
EMPTY	Elemento vazio	<!ELEMENT element-name EMPTY>
ANY	Elemento pode conter qualquer tipo de dados	<!ELEMENT element-name ANY>

- `(element-content)` → lista de dependências, elementos "filhos". Contem a lista dos elementos que dependem desse elemento.

Na Tabela 4.2 encontram-se indicados os operadores auxilares para definir as dependências de um elemento.

Tabela 4.2: Operadores auxiliares para definir as dependências de um elemento

	Descrição	**Exemplo**
child1, child2...	, define as várias dependências do elemento (outros elementos) o elemento depende de child1 e de child2	<!ELEMENT element-name (child1,child2,...)>
child1 — child2	— define que o elemento depende de um elemento ou de child1 ou de child2	<!ELEMENT element-name (child, (child1 — child2))>
child-name+	+ define que a dependência pode ocorrer 1 ou mais vezes	<!ELEMENT element-name (child+)>
child-name*	* define que a dependência pode ocorrer 0 ou mais vezes	<!ELEMENT element-name (child*)>
child-name?	? define que a dependência pode ocorrer 0 ou uma vez	<!ELEMENT element-name (child?)>

Além disso, na lista de dependências, juntamente com as dependências, pode ter a indicação `#PCDATA` que representa que o elemento tem dados de caráter que devem ser analisados gramaticalmente. Sem esta indicação o elemento não é analisado.

Por exemplo:

```
<!ELEMENT name (#PCDATA)>
```

- **Atributo**

```
<!ATTLIST element-name attribute-name attribute-type attribute-value>
```

sendo que:

- `ATTLIST` → indica que se está a definir um atributo

- `element-name` → especifica o nome do elemento a que se refere o atributo
- `attribute-name` → indica o nome do atributo
- `attribute-type` → define o tipo do atributo

Tabela 4.3: Possíveis tipos de um atributo

Tipo do atributo	Exemplo
CDATA	O valor do atributo é um conjunto de caracteres
(en1—en2—..)	O valor do atributo é um dos valores da lista enumerada
ID	O valor do atributo é um identificador único
IDREF	O valor do atributo é um identificador de outro elemento
IDREFS	O valor do atributo é uma lista de outros identificadores
NMTOKEN	O valor do atributo é um nome válido XML
NMTOKENS	O valor do atributo é uma lista de nomes válidos do XML
ENTITY	O valor do atributo define uma entidade
ENTITIES	O valor do atributo define uma lista de entidades
NOTATION	O valor do atributo é um nome de uma notação

- `attribute-value` → estabelece um valor fixo

Tabela 4.4: Possíveis valores de um atributo

Tipo do atributo	Descrição	
value	Valor por defeito do atributo	`<!ATTLIST element-name attribute-name CDATA "0»`
#REQUIRED	O atributo é requerido obrigatório	`<!ATTLIST element-name attribute-name attribute-type #REQUIRED>`
#IMPLIED	O atributo é opcional	`<!ATTLIST element-name attribute-name attribute-type #IMPLIED>`
#FIXED value	O valor do atributo é fixo	`<!ATTLIST element-name attribute-name attribute-type #FIXED "value»`

- **Entidade**

```
<!ENTITY entity-name "entity-value">
```

sendo que:

- `ENTITY` → indica que se está a definir uma entidade
- `entity-name` → indica o nome da entidade
- `entity-value` → armazena o valor da entidade

4.2.2 Considerações iniciais

O trabalho consiste em escrever um processador:

- **Recebe:** um DTD
- **Gera:** a lista (ordenada alfabeticamente) de todos os elementos definidos, dos seus atributos e das suas dependências (outros elementos).

Após analisar a estrutura geral do DTD, detalhada anteriormente, estabeleceram-se algumas considerações iniciais para desenvolver o processador requerido.

- **Lista de nomes dos Elementos**

  Sabe-se que os elementos podem ser encontrados através das seguintes declarações:

  ```
  <!ELEMENT element-name category>
  <!ELEMENT element-name (element-content)>
  ```

  Assim, através de `element-name` é possível obter o nome dos elementos.

- **Dependências de cada elemento**

  Através das declarações que definem um elemento:

  ```
  <!ELEMENT element-name category>
  ```

  – Um elemento definido através desta declaração não tem dependências e, portanto, a lista de dependências é vazia.

  ```
  <!ELEMENT element-name (element-content)>
  ```

  – A lista de dependências pode ser obtida através do conteúdo definido em (`element-content`).
  – As dependências podem estar separados por operadores auxiliares.
  – Os operadores auxiliares devem ser descartados.
  – A lista poderá conter a etiqueta `#PCDATA`, que deve ser descartada.

- **Atributos de cada elemento**

  ```
  <!ATTLIST element-name attribute-name attribute-type attribute-value>
  ```

  – Na declaração de cada atributo pode-se obter o nome do elemento através de `element-name`.

4.3 Desenho da solução

O processador de DTD foi desenhado e implementado em `FLex`. Para suporte à estrutura de dados que vai sendo preenchida no processamento é utilizada uma biblioteca de estruturas: glib. Este processador tem dois modos de funcionamento:

- Gerar uma lista com os elementos, as suas dependências (outros elementos) e os seus atributos
- Grafo de dependências de elementos

Estas duas funcionalidades são atingidas com diferentes *flags* de chamada no programa.

O *output* dos comandos é para o *standart output* no caso da lista de elementos, e para o ficheiro passado por parâmetro no caso do grafo.

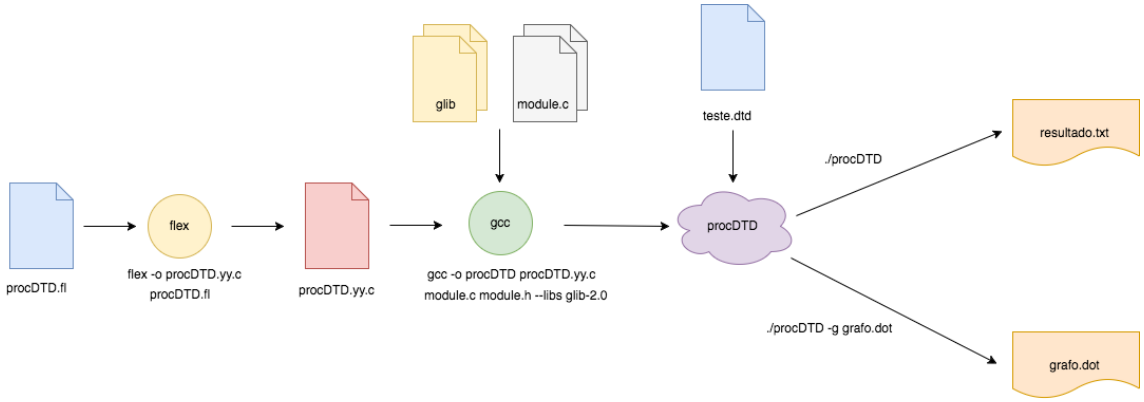

Figura 4.1: Esquema representativo da arquitectura do Processador de DTDs em FLex

4.4 Especificação

4.4.1 Especificar os padrões das Expressões Regulares

Os DTD podem ser constituídos por diferentes componentes: Elemento, Lista de Atributos e Entidade. Assim, para cada um destes componentes, definiu-se um conjunto de expressões regulares para os conseguir identificar e caracterizar.

Elemento

- **Começo de elemento**
  - `\<!ELEMENT` → Expressão regular que faz *match* ao texto exacto pelo qual um elemento começa.
- **Fim de um elemento**

- `\*?\>` → A expressão regular serve para identificar o fim de um elemento. Este elemento pode ter no fim um asterisco, que identifica que as dependências do elemento se repetem várias vezes. Para finalizar o elemento existe ainda o caracter `>`.

- **Nome do elemento**
  - `[^ E\(\n]+` → O nome de um elemento é algo que não contenha espaços e que não seja uma abertura de parênteses (que identifica o começo de dependências do elemento). Esta expressão regular captura exactamente essa identificação.

- **Início da definição de dependências**
  - `\(` → A expressão regular identifica o começo de definição de dependências do elemento.

- **Término da definição de dependências**
  - `\)` → A expressão regular serve para identificar o caracter de fecho de parênteses que indica que a definição de dependência terminou.

- **Elemento vazio**
  - `EMPTY` → A expressão regular tenta fazer *match* ao texto `EMPTY` que caracteriza um elemento vazio.

- **Texto analisável de um elemento**
  - `\#PCDATA` → Um elemento pode conter texto que será analisado, esta possibilidade é identificada pelo texto `#PCDATA` que esta expressão regular identifica.

- **Nome de dependência de um elemento**
  - `[^, \#\|\(\)\*\?]+` → Para capturar uma dependência de um elemento, quer-se o texto que não contenha parênteses, que não seja `#PCDATA`, que não contenha `,` (que separam as dependências) e que não tenham os operadores de expressão regular para identificar multiplicidade (`*` e `+`) e possibilidade de existência (`?`) das dependências. A expressão regular representa exactamente estas condições.

Lista de atributos

- **Começo de um ATTList**
  - `\<!ATTLIST` → Expressão regular que faz *match* ao texto do começo de um `ATTList`.

- **Fim de um ATTList**
  - `\>` → A expressão regular identifica o caracter exacto do fim da definição de um ATTList (`>`).

- **Elemento ao qual pertence a lista de atributos**

  - `[^ \n]+` → Expressão regular que identifica o elemento ao qual diz respeito o `ATTList`. Não pode conter espaços nem o caractér fim de linha (`\n`)

- **Identificação que seguirá uma definição de um atributo**

  - `[^\>\na-zA-Z]` → A expressão regular captura caracteres de forma a identificar que se segue a definição de um atributo. Esta expressão regular caso faça *match* permite que seja identificado um conjunto de caracteres antes do nome do atributo, e assim se tenha a certeza que de seguida virá a definição de um atributo.

- **Identificação do nome e tipo de um atributo**

  - `[^ \t]+` → A expressão regular permite fazer *match* ao nome e ao tipo de um atributo. Tanto o nome como o tipo são um conjunto de caracteres sem espaços nem *tabs*.

- **Identificação de um valor fixo de um atributo**

  - `\#FIXED" "[^ \>]+` → A expressão regular tenta fazer *match* com qualquer expressão com a sintaxe de um valor fixo de um atributo: `#FIXED valor`.

- **Identificação do valor de obrigatório de um atributo**

  - `\#REQUIRED` → De forma análoga à anterior, a expressão regular tenta fazer *match* com o valor de obrigatório do atributo, que neste caso é `#REQUIRED`.

- **Identificação do valor de opcional de um atributo**

  - `\#IMPLIED` → O valor do atributo também pode ser opcional - `#IMPLIED`, assim, a expressão regular tenta fazer *match* com este valor do atributo.

- **Identificação de um valor por defeito de um atributo**

  - `\"[^ \"]+\"` → Para além dos três valores de atributos já identificados anteriormente com diferentes expressões regulares, o atributo pode ainda ter um valor por defeito identificado por `"valor"`. Assim, a expressão regular tentará capturar todos os caracteres compreendidos entre duas aspas (`"`).

Entidade

- **Começo de um Entity**

  - `\<!ENTITY` → Expressão regular que faz *match* ao texto do começo de uma `Entity`.

- **Fim de um Entity**

  - `\>` → A expressão regular identifica o caracter exacto do fim da definição de um Entity (`>`).

Outros

- **Identificação de outros caracteres**
  - `(.|\n)` → Expressão regular utilizada para identificar todos os caracteres ou `\n` que não sejam capturados em todas as outras expressões regulares.

4.4.2 Definição das *start condition*

A identificação de cada um dos constituintes de um DTD pode-se tornar uma tarefa complicada se o contexto de identificação não for isolado. Assim, para auxiliar a identificação de cada um destes constituintes, foi utilizado o mecanismo de *start conditions* disponível no **FLex**.

Este mecanismo de *start conditions* consegue criar contextos separados o que permite isolar as expressões regulares que são aplicadas em diferentes *start conditions*. Partindo da estrutura de cada um dos constituintes de um DTD, identificaram-se as seguintes *start conditions*.

Os estados identificados foram os seguintes:

- `ELEMENT` - Identifica o contexto de identificação de um elemento
  - `ELEMDEP` - Contexto de identificação de uma ou mais dependências
- `ATTLIST` - Identifica o contexto de identificação de um conjunto de atributos
  - `ATTRIBUTE` - Contexto de identificação de um atributo
    - `ATTNAME` - Identificação do nome do atributo
    - `ATTTYPE` - Identificação do tipo do atributo
    - `ATTVALUE` - Identificação do valor do atributo
- `ENTITY` - Identifica o contexto de identificação de uma entidade
- `INITIAL` - *Start condition* inicial do **FLex**.

Após identificar as diferentes *start conditions* foi necessário perceber como seria feita a transição entre elas. A Figura 4.2 tem a representação das diferentes *start conditions* (nodos) e os padrões que levam à sua transição para outra *start condition* (ligações). De seguida, para cada constituinte de um DTD é feita uma análise às suas *start conditions*.

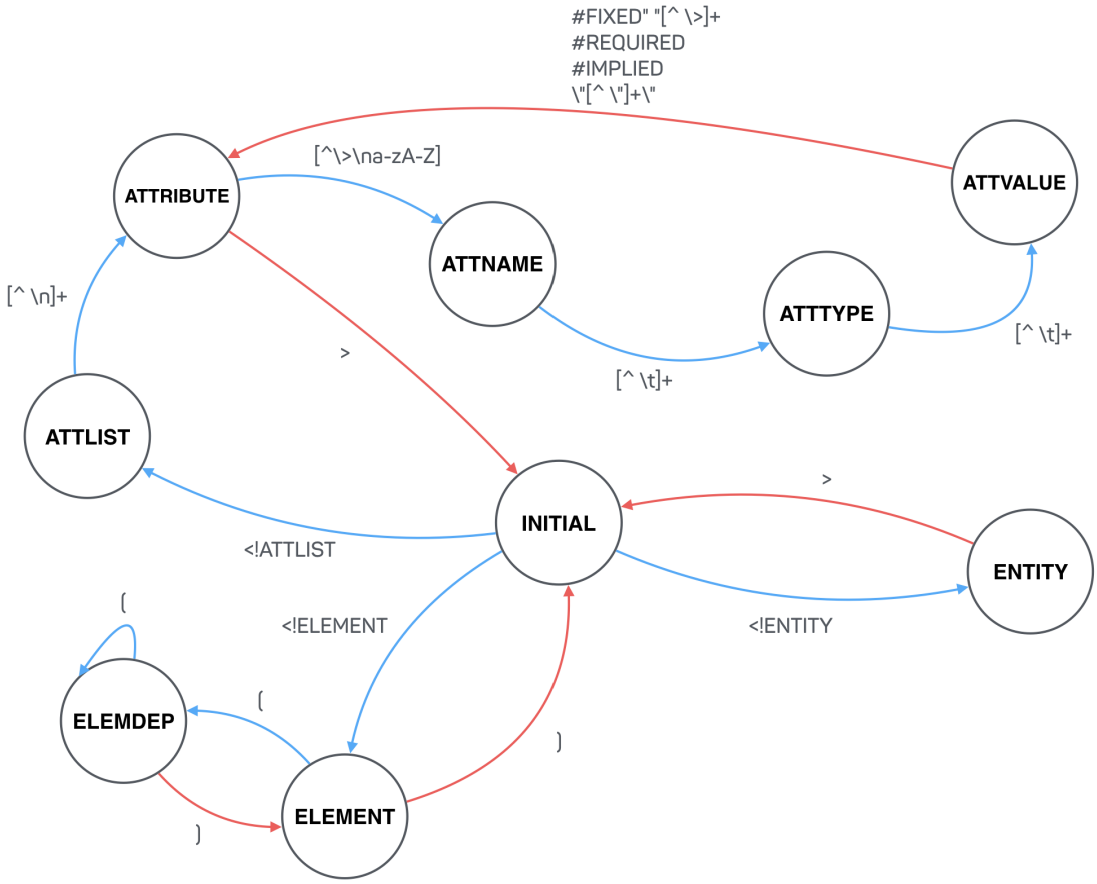

Figura 4.2: Diferentes *start conditions* e transições entre elas. A azul as transições de progresso, a vermelho as de regresso.

ELEMENT

A transição da *start condition* `INITIAL` para `ELEMENT` é feita quando é encontrado um começo do elemento Element.

Na definição de um element, de seguida podem aparecer dependências ou definição de element vazio. Caso seja encontrado um `(`, detecta-se que seguirá um conjunto de dependências e assim existe a transição para a *start condition* `ELEMDEP`.

Durante a *start condition* `ELEMDEP` outros conjuntos de dependências poderão surgir com o aparecimento do caracter `(` o que levará à passagem recursiva para `ELEMDEP`. Quando o caracter de fim de parênteses, `)`, é encontrado é feita a transição para a *start condition* anterior.

Como as dependências podem ser constituídas por conjuntos aninhados de conjuntos de dependências, houve a necessidade de em vez de definir o estado seguinte a transitar quando terminada a leitura de dependências, se recuperasse a *start condition* anterior. Assim, tirou-se partida do mecanismo de *Stack* que o FLex disponibiliza para ajuda ás *start conditions* que resolve exactamente estas situações.

Para o seguinte conjunto de dependências, a *Stack* de *start conditions* a serem recuperadas no final da identificação de dependências encontra-se ilustrada na Figura 4.3:

```
(a, b, (c, d, (e, f, g), h))
```

Figura 4.3: *Stack* de *start conditions* ao longo do processamento de dependências de um elemento

Este mecanismo permite que a transição e recuperação de *start conditions* fique assegurada. No fim de lidas todas as dependências, regressa-se à *start condition* `INITIAL`.

ATTLIST

A *start condition* `ATTLIST` é ativada a partir da `INITIAL` quando o ATTList é identificado.

O ATTList é constituído pelo nome do elemento ao qual a lista de atributos diz respeito e por uma lista de atributos. Assim, após a leitura de algo que não seja o nome do elemento, a passagem é feita para a *start condition* `ATTRIBUTE`. Após o contexto estar restringido ao atributo, sabe-se que existem três informações a serem lidas:

- Nome do atributo
- Tipo do atributo
- Valor do atributo.

Para cada uma destas informações, foi definida uma *start condition* para melhor isolar o seu contexto de identificação:

- `ATTNAME`
- `ATTTYPE`
- `ATTVALUE`.

Assim, quando se está num atributo, transita-se para a *start condition* `ATTNAME` para identificar o nome do atributo. Da `ATTNAME` passa-se para a `ATTTYPE` para identificar o tipo do atributo. Por fim, após identificar o tipo, é identificado o valor. Sendo que este valor pode ser uma de quatro coisas:

- Atributo com valor fixo
- Atributo de carácter obrigatório
- Atributo de carácter opcional
- Atributo com valor por defeito.

No fim de identificar o valor do atributo, uma vez que podem existir vários atributos, a transição é feita para a *start condition* `ATTRIBUTE` para continuar a identificação de outros atributos. No fim de todos os atributos serem lidos, é regressa-se à *start condition* `INITIAL`.

ENTITY

A transição para a *start condition* `ENTITY` é feita através da identificação do constituinte do DTO, Entity. Quando for encontrado o fim do entity, existe a transição para a *start condition* `INITIAL`.

4.4.3 Acções Semânticas

Após a identificação e explicação das expressões regulares utilizadas bem como as *start condition* utilizadas para isolar o contexto de identificação de cada um dos componentes, seguem as ações semânticas associadas a cada uma das expressões regulares.

Elemento

- **Começo de elemento**
  - `\<!ELEMENT` → É inicializada a *start condition* `ELEMENT`.
- **Fim de um elemento**
  - `<ELEMENT>\*?\>` → regressa-se a `INITIAL`.
- **Nome do elemento**
  - `<ELEMENT>[^ E\(\n]+` → É criada uma nova informação sobre o nome do elemento extraída e essa informação é inserida na estrutura global.
- **Início da definição de dependências**
  - `<ELEMENT>\(` → É feita a transição para a *start condition* `ELEMDEP` e é guardado a *start condition* atual na *Stack* para ser recuperada mais tarde.
- **Término da definição de dependências**
  - `<ELEMDEP>\)` → É recuperada a start condition anterior.
- **Elemento vazio**

- `<ELEMENT>EMPTY` → Não existe processamento da informação extraída, esta é apenas consumida.

- **Texto analisável de um elemento**

  - `<ELEMDEP>\#PCDATA` → Não existe processamento da informação extraída, esta é apenas consumida.

- **Nome de dependência de um elemento**

  - `<ELEMDEP>[^, \#\|\(\)\*\?]+` → O nome da dependência capturado, é guardada na estrutura das dependências do elemento a ser processado.

Lista de atributos

- **Começo de um ATTList**

  - `\<!ATTLIST` → É inicializada a *start condition* `ATTLIST`.

- **Fim de um ATTList**

  - `<ATTRIBUTE>\>` → É feita a transição de regresso para a *start condition* `INITIAL`.

- **Elemento ao qual pertence a lista de atributos**

  - `<ATTLIST>[^ \n]+` → É recuperado da estrutura global o elemento à qual pertence a definição da lista de atributos. Assim, permitir-se-á a alteração e adição de atributos a esse elemento. É feita a transição para a *start condition* `ATTRIBUTE` para proceder à identificação da lista de atributos.

- **Identificação que seguirá uma definição de um atributo**

  - `<ATTRIBUTE>[^\>\na-zA-Z]` → Existe a transição para a *start condition* `ATTNAME`.

- **Identificação do nome de um atributo**

  - `<ATTNAME>[^ \t]+` → Extraído o nome do atributo, este é adicionado à lista de atributos do elemento actual ao qual pertence a lista de atributos a ser processada. É ainda feita a transição para a *start condition* `ATTTYPE`.

- **Identificação do tipo de um atributo**

  - `<ATTTYPE>[^ \t]+` → Não existe processamento da informação extraída, esta é apenas consumida. É feita a transição para a *start condition* `ATTVALUE`.

- **Identificação de um valor fixo de um atributo**

  - `<ATTVALUE>\#FIXED" "[^ \>]+` → Não existe processamento da informação extraída, esta é apenas consumida. Para continuar a identificação de outros atributos, é feita a transição para a *start condition* `ATTRIBUTE`.

- **Identificação do valor de obrigatório de um atributo**
  - `<ATTVALUE>\#REQUIRED` → De forma análoga à acção anterior, não existe processamento da informação extraída, esta é apenas consumida. Para continuar a identificação de outros atributos, é feita a transição para a *start condition* `ATTRIBUTE`.
- **Identificação do valor de opcional de um atributo**
  - `<ATTVALUE>\#IMPLIED` → Não existe processamento da informação extraída, esta é apenas consumida. Para continuar a identificação de outros atributos, é feita a transição para a *start condition* `ATTRIBUTE`.
- **Identificação de um valor por defeito de um atributo**
  - `<ATTVALUE>\"[^ \"]+\"` → De forma análoga à acção anterior, não existe processamento da informação extraída, esta é apenas consumida. Para continuar a identificação de outros atributos, é feita a transição para a *start condition* `ATTRIBUTE`.

Entidade

- **Começo de um Entidade**
  - `\<!ENTITY` → É inicializada a *start condition* `ENTITY`.
- **Fim de uma Entidade**
  - `<ENTITY>\>` → É feita a transição de regresso para a *start condition* `INITIAL`.

Outros

- **Identificação de outros caracteres**
  - `<*>(.|\n)` → A informação é apenas consumida, não existe outra ação semântica associada.

4.4.4 Identificação das Estruturas de Dados globais

As estruturas de dados onde a informação é guardada enquanto o input está a ser processado, foram escolhidas de acordo com as necessidades do programa desenvolvido. Para as estruturas, utilizou-se a biblioteca `glib`, open source, com diversas estruturas de dados prontas a usar.

É um requisito listar alfabeticamente os elementos do DTD passado por *input*. A estrutura de dados escolhida foi uma árvore binária (`GTree` na biblioteca `glib`), que tem a ordenação como uma das suas características, bem como eficiência na procura de repetidos para não provocar incoerências na estrutura de dados.

No processamento do *input*, a informação dos diferentes elementos é extraída e adicionada à árvore binária. A informação guardada para cada elemento é:

- Nome elemento

- Lista de atributos
- Lista de dependências.

Assim, foi criada uma estrutura de dados para poder suportar esta informação do elemento:

```
typedef struct sElemento {
    char * nome;
    GSequence * atributos;
    GSequence * dependencias;
} Elemento;
```

Tanto as listas de atributos como a lista de dependências, foram representadas com *GSequence* que é uma estrutura de dados de listas dinâmicas.

A árvore binária que vai sendo construída encontra-se ordenada pelo nome do elemento. A Figura 4.4 ilustra a estrutura escolhida (árvore), bem como cada nodo (informação de elemento).

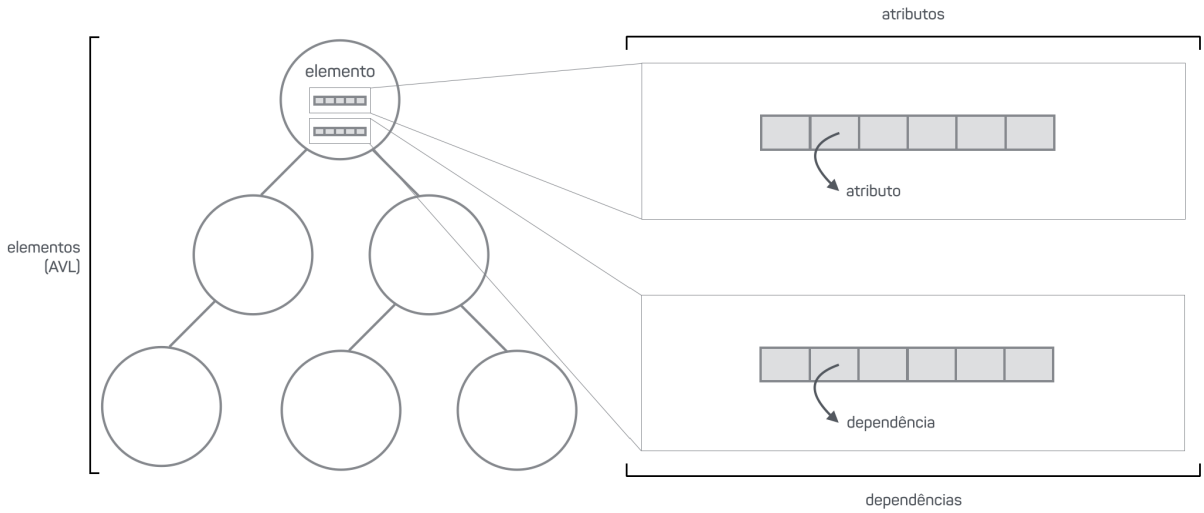

Figura 4.4: Estrutura para guardar a informação dos DTD processados

4.5 Implementação

A implementação foi feita em FLex com código em `C`. Para melhor separar os conteúdos desenvolvidos para resolver estes problemas, foram separados em dois tipos de ficheiros:

- Ficheiro com a extensão `.fl` onde está o código de FLex que permite o processamento e identificação de termos de input
- Ficheiros com extensão `.c` onde fica o código `C` que dá suporte ao processamento de ficheiros e disponibiliza as estruturas para guardar a informação processada.

A implementação dos problemas enunciados é composta por duas partes:

- Listagem dos elementos, das suas dependências e atributos
- Geração de um grafo de dependências dos elementos.

Como ambos os problemas partilham o mesmo domínio, a implementação ficou nos mesmos ficheiros, apenas sendo feita a diferenciação nas funções chamadas.

Listagem dos elementos, as suas dependências e atributos

Para a listagem dos elementos, as suas dependências e atributos, é feita uma navegação sobre a estrutura de dados global (árvore - *GTree*) que contém a informação de cada um dos elementos. Como a árvore se encontra ordenada alfabeticamente, a iteração será feita de acordo com essa ordenação. Assim, em cada nodo da árvore, é imprimido o nome da informação e:

- navega-se na lista dos atributos (*GSequence*) e imprime-se cada um dos atributos entre parênteses
- navega-se na lista das dependências (*GSequence*) e imprime-se cada uma dessas dependências.

Geração de um grafo de dependências dos elementos.

A geração do grafo de dependências é feita no formato `.dot` que é o formato suportado por ferramentas como o *GraphViz*. O que este grafo representa é a dependência dos elementos de outros elementos. Para gerar este grafo de dependências é necessário listar as várias dependências de um elemento. Por exemplo, se um elemento `A` tiver as seguintes dependências:

- B
- C
- D,

o ficheiro a gerar terá o seguinte formato:

```
digraph nome {
    A -> B;
    A -> C;
    A -> D;
}
```

A geração de tal informação é possível através de uma travessia na estrutura global de informação de elementos e visitando de seguida cada uma das suas dependências.

Foi desenvolvida a hipótese de gerar um grafo de dependências para um elemento em específico, bem como para todos os elementos.

Para dar uma ideia do grau de dependências de cada um dos elementos, os nodos do grafo vão sendo coloridos com cores na seguinte gama:

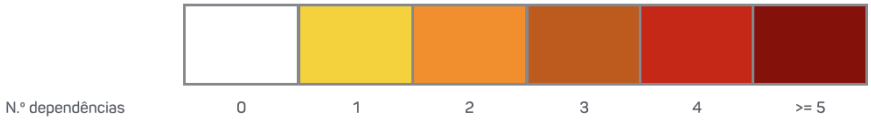

Figura 4.5: Gama de cores consoante número de dependências

4.6 Testes e Resultados

Para mostrar as diferentes capacidades do processador de DTDs, correram-se três exemplos com os dois modos de funcionamento do processador.

4.6.1 Ficheiro sinopse.dtd e resultados

O ficheiro de *input* foi o ficheiro no **apêndice B.2.1** e contém a definição de um DTD de sinópse, que define o modelo de descrição de uma sinópse em XML. Este ficheiro contém 21 elementos.

Listagem de elementos, as suas dependências e os seus atributos

De seguida segue a listagem produzida como *output* pelo processador.

```
Elementos:

:: Elemento a ::
        1 Atributo(s): href
        0 Dependencias
:: Elemento abstract ::
        0 Atributos
        2 Dependencia(s): image, p
:: Elemento author ::
        0 Atributos
        0 Dependencias
:: Elemento b ::
        0 Atributos
        0 Dependencias
:: Elemento date ::
        0 Atributos
        0 Dependencias
:: Elemento dd ::
        0 Atributos
        1 Dependencia(s): ul
:: Elemento dl ::
        0 Atributos
        2 Dependencia(s): dd, dt
:: Elemento dt ::
```

```
        0 Atributos
        1 Dependencia(s): b
:: Elemento image ::
        1 Atributo(s): href
        0 Dependencias
:: Elemento li ::
        0 Atributos
        0 Dependencias
:: Elemento meta ::
        0 Atributos
        3 Dependencia(s): author, date, obs
:: Elemento obs ::
        0 Atributos
        0 Dependencias
:: Elemento ol ::
        0 Atributos
        1 Dependencia(s): li
:: Elemento p ::
        0 Atributos
        2 Dependencia(s): a, b
:: Elemento project ::
        0 Atributos
        0 Dependencias
:: Elemento ref ::
        1 Atributo(s): type
        0 Dependencias
:: Elemento section ::
        3 Atributo(s): encterm, title, url
        9 Dependencia(s): author, date, dl, image, meta, ol, p, section, ul
:: Elemento sinopse ::
        0 Atributos
        5 Dependencia(s): author, date, project, text, title
:: Elemento text ::
        0 Atributos
        2 Dependencia(s): abstract, section
:: Elemento title ::
        0 Atributos
        0 Dependencias
:: Elemento ul ::
        0 Atributos
        1 Dependencia(s): li
```

Grafo global de dependências

Segue o grafo global gerado pelo programa.

```
digraph dependencies {
        "a" [style=filled, fillcolor=white];
        abstract -> image;
        abstract -> p;
        "abstract" [style=filled, fillcolor=orange];
        "author" [style=filled, fillcolor=white];
        "b" [style=filled, fillcolor=white];
        "date" [style=filled, fillcolor=white];
        dd -> ul;
        "dd" [style=filled, fillcolor=yellow];
        dl -> dd;
        dl -> dt;
        "dl" [style=filled, fillcolor=orange];
        dt -> b;
        "dt" [style=filled, fillcolor=yellow];
        "image" [style=filled, fillcolor=white];
        "li" [style=filled, fillcolor=white];
        meta -> author;
        meta -> date;
        meta -> obs;
        "meta" [style=filled, fillcolor=orangered];
        "obs" [style=filled, fillcolor=white];
        ol -> li;
        "ol" [style=filled, fillcolor=yellow];
        p -> a;
        p -> b;
        "p" [style=filled, fillcolor=orange];
        "project" [style=filled, fillcolor=white];
        "ref" [style=filled, fillcolor=white];
        section -> author;
        section -> date;
        section -> dl;
        section -> image;
        section -> meta;
        section -> ol;
        section -> p;
        section -> section;
        section -> ul;
        "section" [style=filled, fillcolor=red4];
        sinopse -> author;
        sinopse -> date;
        sinopse -> project;
```

```
        sinopse -> text;
        sinopse -> title;
        "sinopse" [style=filled, fillcolor=red4];
        text -> abstract;
        text -> section;
        "text" [style=filled, fillcolor=orange];
        "title" [style=filled, fillcolor=white];
        ul -> li;
        "ul" [style=filled, fillcolor=yellow];
}
```

Este grafo depois de compilado por um programa que suporte `.dot` como é o caso do `GraphViz` produz o grafo que se encontra na Figura 4.6.

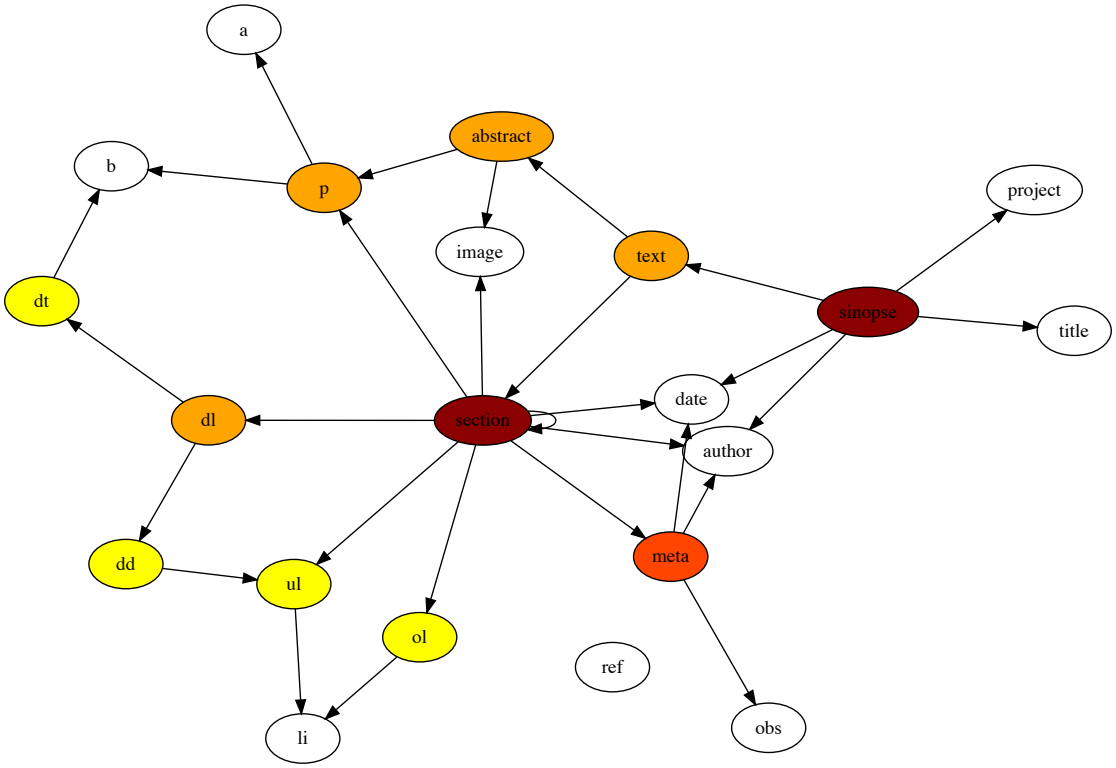

Figura 4.6: Grafo gerado de dependências globais do DTD sinopse com cores consoante o número de dependências

4.6.2 Ficheiro bi.dtd e resultados

O ficheiro de *input* foi o ficheiro no **apêndice B.2.2** e contém a definição de um DTD de um bi, que define o modelo de descrição de um BI (Bilhete identidade) em XML. Este ficheiro contém 17 elementos.

Listagem de elementos, as suas dependências e os seus atributos

De seguida segue a listagem produzida como *output* pelo processador.

```
Elementos:

:: Elemento autor ::
        0 Atributos
        0 Dependencias
:: Elemento bi ::
        0 Atributos
        15 Dependencia(s): autor, biografia, data, depoente, editor, entrev
:: Elemento biografia ::
        0 Atributos
        0 Dependencias
:: Elemento data ::
        3 Atributo(s): ano, dia, mes
        0 Dependencias
:: Elemento depoente ::
        1 Atributo(s): estadocivil
        0 Dependencias
:: Elemento editor ::
        0 Atributos
        0 Dependencias
:: Elemento entrevistador ::
        0 Atributos
        0 Dependencias
:: Elemento escolaridade ::
        0 Atributos
        0 Dependencias
:: Elemento foto ::
        1 Atributo(s): ficheiro
        0 Dependencias
:: Elemento morada ::
        0 Atributos
        0 Dependencias
:: Elemento nascimento ::
        4 Atributo(s): ano, dia, mes, onde
        0 Dependencias
:: Elemento notas ::
        0 Atributos
        0 Dependencias
:: Elemento profissao ::
        0 Atributos
        0 Dependencias
:: Elemento projecto ::
```

```
        0 Atributos
        0 Dependencias
:: Elemento rel ::
        1 Atributo(s): tipo
        0 Dependencias
:: Elemento titulo ::
        0 Atributos
        0 Dependencias
:: Elemento transcritor ::
        0 Atributos
        0 Dependencias
```

Grafo global de dependências

Segue o grafo global gerado pelo programa.

```
digraph dependencies {
        "autor" [style=filled, fillcolor=white];
        bi -> autor;
        bi -> biografia;
        bi -> data;
        bi -> depoente;
        bi -> editor;
        bi -> entrevistador;
        bi -> foto;
        bi -> morada;
        bi -> nascimento;
        bi -> notas;
        bi -> profissao;
        bi -> projecto;
        bi -> rel;
        bi -> titulo;
        bi -> transcritor;
        "bi" [style=filled, fillcolor=red4];
        "biografia" [style=filled, fillcolor=white];
        "data" [style=filled, fillcolor=white];
        "depoente" [style=filled, fillcolor=white];
        "editor" [style=filled, fillcolor=white];
        "entrevistador" [style=filled, fillcolor=white];
        "escolaridade" [style=filled, fillcolor=white];
        "foto" [style=filled, fillcolor=white];
        "morada" [style=filled, fillcolor=white];
        "nascimento" [style=filled, fillcolor=white];
        "notas" [style=filled, fillcolor=white];
        "profissao" [style=filled, fillcolor=white];
```

```
        "projecto" [style=filled, fillcolor=white];
        "rel" [style=filled, fillcolor=white];
        "titulo" [style=filled, fillcolor=white];
        "transcritor" [style=filled, fillcolor=white];
}
```

Este grafo depois de compilado por um programa que suporte `.dot` como é o caso do `GraphViz` produz o grafo que se encontra na Figura 4.8.

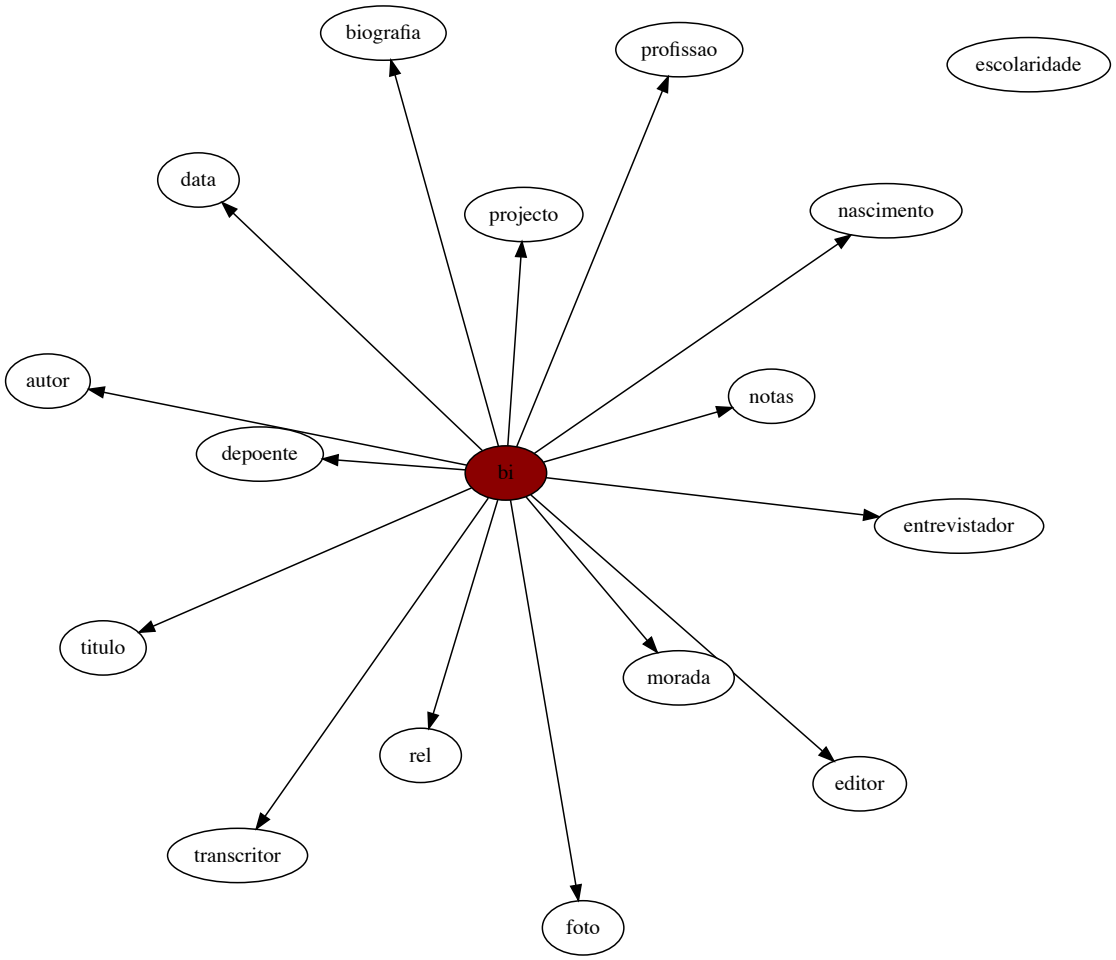

Figura 4.7: Grafo gerado de dependências globais do DTD BI com cores consoante o número de dependências

4.6.3 Ficheiro fotos.dtd e resultados

O ficheiro de *input* foi o ficheiro no **apêndice B.2.3** e contém a definição de um DTD de um album de fotos, que define o modelo de descrição de um album de fotos em XML. Este ficheiro contém 8 elementos.

Listagem de elementos, as suas dependências e os seus atributos

De seguida segue a listagem produzida como *output* pelo processador.

```
Elementos:

:: Elemento autor ::
        1 Atributo(s): tipo
        0 Dependencias
:: Elemento facto ::
        0 Atributos
        0 Dependencias
:: Elemento foto ::
        1 Atributo(s): ficheiro
        6 Dependencia(s): autor, facto, legenda, onde, quando, quem
:: Elemento fotos ::
        0 Atributos
        2 Dependencia(s): autor, foto
:: Elemento legenda ::
        0 Atributos
        0 Dependencias
:: Elemento onde ::
        0 Atributos
        0 Dependencias
:: Elemento quando ::
        1 Atributo(s): data
        0 Dependencias
:: Elemento quem ::
        0 Atributos
        0 Dependencias
```

Grafo global de dependências

Segue o grafo global gerado pelo programa.

```
digraph dependencies {
        "autor" [style=filled, fillcolor=white];
        "facto" [style=filled, fillcolor=white];
        foto -> autor;
        foto -> facto;
        foto -> legenda;
        foto -> onde;
        foto -> quando;
        foto -> quem;
        "foto" [style=filled, fillcolor=red4];
        fotos -> autor;
```

```
        fotos -> foto;
        "fotos" [style=filled, fillcolor=orange];
        "legenda" [style=filled, fillcolor=white];
        "onde" [style=filled, fillcolor=white];
        "quando" [style=filled, fillcolor=white];
        "quem" [style=filled, fillcolor=white];
}
```

Este grafo depois de compilado por um programa que suporte `.dot` como é o caso do `GraphViz` produz o grafo que se encontra na Figura 4.8.

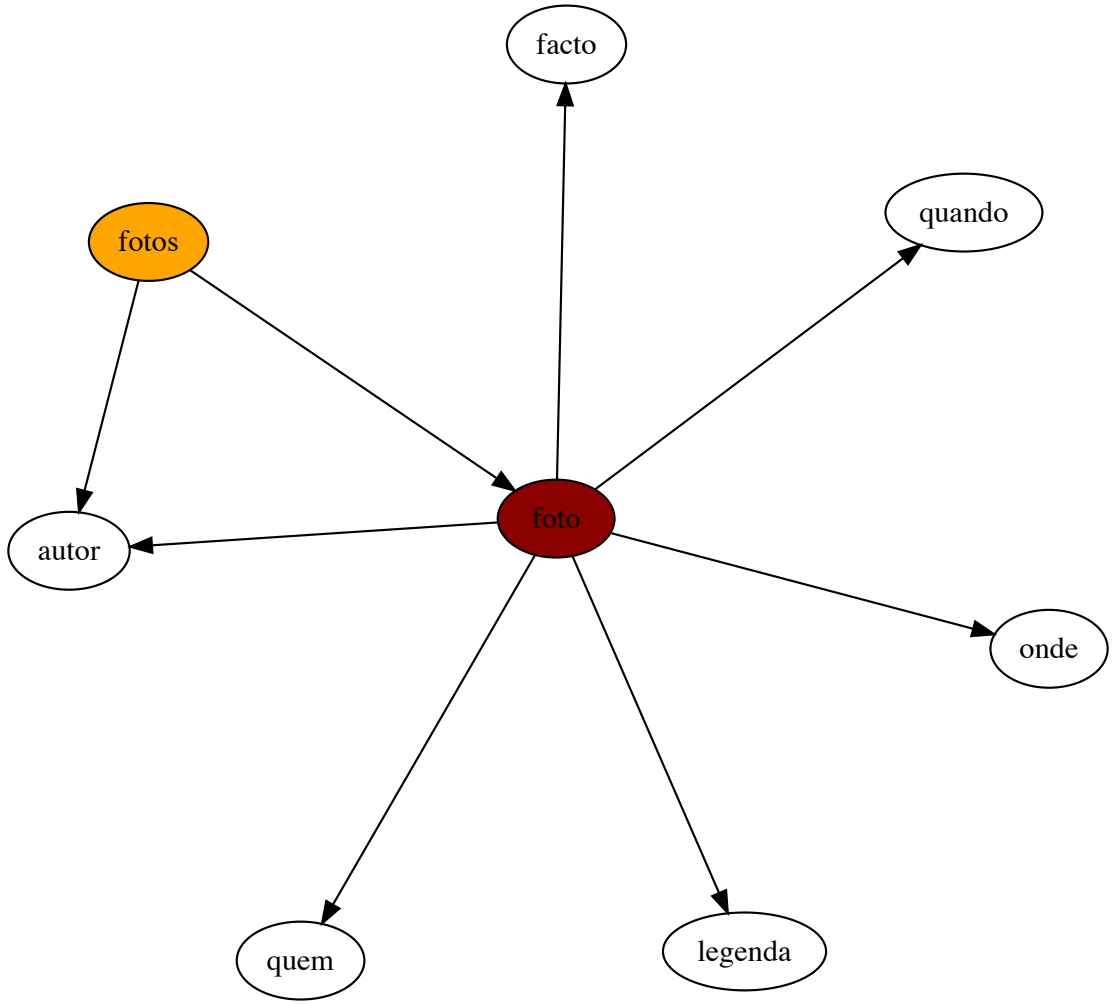

Figura 4.8: Grafo gerado de dependências globais do DTD fotos com cores consoante o número de dependências

Capítulo 5

Processador de *Named Entities*

5.1 Enunciado

Pretende-se analisar e processar um documento anotado num dialeto XML denominado ENAMEX, o qual permite identificar através das etiquetas ENAMEX e TIMEX as entidades referenciadas num texto através de nomes próprios, (tipo: Pessoa, País, Cidade) ou Datas.

Segue abaixo um pequeno exemplo de um texto anotado no dialeto XML - ENAMEX.

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
<ENAMEX> Vivia </ENAMEX> este herói no <ENAMEX TYPE="CITY"> Rio de
Janeiro </ENAMEX>, <ENAMEX TYPE="COUNTRY">Brasil</ENAMEX> e era
professor não se sabe de que doutrinas no ano de
<TIMEX TYPE="DATE"> 1710 </TIMEX>, quando os franceses
comandados por <ENAMEX TYPE="PERSON"> Duclerc </ENAMEX>, atacaram a cidade.
O governador portou-se mal, e permanecia numa deplorável inação, quando
<ENAMEX TYPE="PERSON"> Bento do Amaral </ENAMEX> à frente dos seus estudantes
e de paisanos armados, saiu a tomar o passo aos invasores, repelindo-os energicamente,
e dando lugar a que o ataque se malograsse e os agressores ficassem prisioneiros.
<ENAMEX> Não </ENAMEX> tardou <ENAMEX TYPE="PERSON"> Dugua-Trouin </ENAMEX> a vir tomar
a desforra.
</document>
```

Além de implementar um processador que analise um ficheiro ENAMEX, também é pedido que se produza uma página HTML com as Pessoas, Países e Cidades referidas. A fim de determinar o período de tempo no qual decorre a narrativa, há que indicar o menor ano citado, assim como o mais actual. Irá-se recorrer ao Google Maps de modo a localizar as cidades referidas no texto num mapa mundial.

5.2 Descrição do problema

Os documentos anotados num dialeto XML - ENAMEX permitem a identificação das entidades referenciadas no texto pelas etiquetas ENAMEX e TIMEX. No XML disponibilizado para teste pode-se

Tabela 5.1: Etiquetas ficheiro XML e o tipo de informação que referenciam

Tipo de Informação	Etiqueta
Nome de Pessoas	<ENAMEX TYPE=”PERSON”>... </ENAMEX>
	<ENAMEX TYPE=”NAME>... </ENAMEX>
	<ENAMEX TYPE=”NAME CERTEZA=??>... </ENAMEX>
Países	<ENAMEX TYPE=”COUNTRY”>... </ENAMEX>
Cidades	<ENAMEX TYPE=”CITY”>... </ENAMEX>
	<ENAMEX TYPE=”LOCATION”>... </ENAMEX>
Datas	<TIMEX TYPE=”DATE”>... </TIMEX>

encontrar as seguintes etiquetas `ENAMEX`, que permitem referenciar nomes próprios do tipo: Pessoa, País, Cidade, assim como a etiqueta `TIMEX` que permite referenciar datas.

Na Tabela 5.1 identifica-se as etiquetas presentes no texto e a que tipo de informação se referem, isto é, se identificam o nome de uma pessoa, uma data, uma cidade, etc.

Como exemplo foi disponibilizado o ficheiro exemplo-Enamex1.xml. Este tem como cabeçalho :

```
<?xml version="1.0" encoding="UTF-8"?>
```

Informando que a versão do `XML` utilizada é a versão 1.0, e que a codificação utilizada é `UTF-8`.

O ficheiro `XML` é composto por um conjunto de documentos, que contêm as etiquetas `ENAMEX` e `TIMEX` referidas acima:

```
<document>
...
</document>
```

Pretende-se desenvolver um processador `flex` que análise um ficheiro `ENAMEX` de modo a:

- Produzir um índice `HTML` com as Pessoas, Países e Cidades referidas no ficheiro;
- Determinar o período em que ocorre a narrativa, identificando o menor e maior ano;
- Utilizar o `Google Maps` de modo a localizar as cidades referidas no ficheiro num mapa global.

Adicionalmente decidiu-se imprimir para o `stdout` a listagem do nome das pessoas, países, cidades e anos referenciados no texto, e para cada um destes itens, o número de vezes que é referenciado no texto. Também é imprimido o número global de nomes referenciados no texto por tipo (pessoas, cidades e países), assim como número de anos.

O cálculo do número de ocorrências de nome de pessoas, países, cidades e anos poderá ajudar a inferir sobre a temática do texto, qual a personagem/pessoa mais relevante na história, e por conseguinte o local, cidade e ano. Também ajudará a inferir a complexidade da história, por exemplo, quanto mais personagens na história mais complexa e rebuscada esta é.

5.3 Desenho da solução

Para a concretização das tarefas enunciadas na Descrição do Problema, que inclui funcionalidades adicionais ao requerido no enunciado, criou-se um ficheiro `Flex` com o nome `namedEntities.l`. Este ficheiro foi escrito de acordo com a sintaxe do `Flex`, recorrendo a expressões regulares para capturar e extrair a informação desejada, inserindo esta informação em estruturas de dados adequadas, de maneira a que não haja repetidos e que esteja ordenada. Posteriormente este ficheiro será passado como parâmetro ao programa `Flex` , que gerará um analisador léxico escrito na linguagem `C`, sendo depois compilado com um compilador da linguagem `C`, gerando o programa executável `namedEntites`.

O programa gerado (`namedEntites`), irá receber como input um ficheiro de texto `ENAMEX` em `XML`, gerando um ficheiro `HTML` com a listagem ordenada do nome das pessoas, países, cidades, o menor e maior ano, assim como a localização num mapa dos países e cidades mencionados no texto.

Como tal, começou-se por elaborar um esboço da página a gerar, de forma a representá-la esquematicamente. Na Figura 5.1 encontra-se representado o esquema projetado da página `HTML` a ser criada.

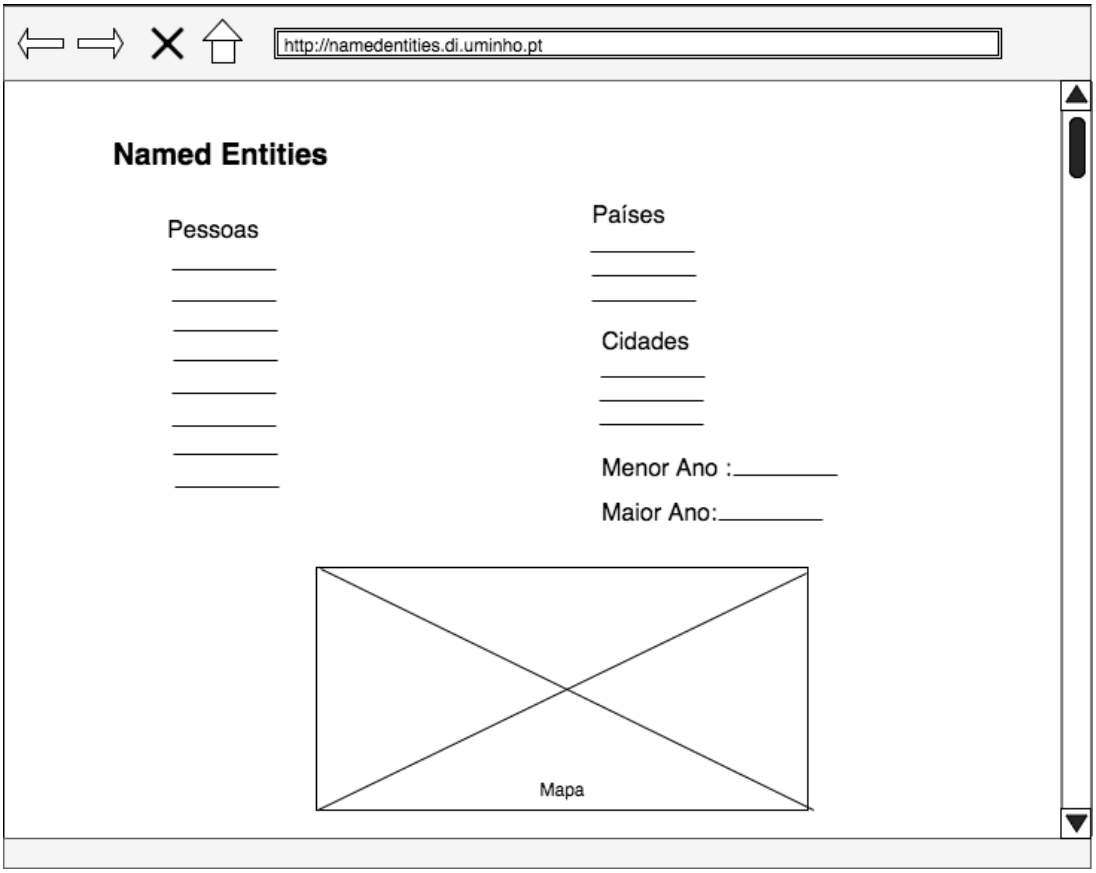

Figura 5.1: Esquema da página HTML

Paralelamente também se imprime para a consola a listagem ordenada do nome das pessoas, países, cidades e anos referidos no texto. Também se irá imprimir o número de ocorrências no texto de cada um dos itens, assim como o número total de cada tipo.

5.3.1 Arquitetura

Na Figura 5.2 pode-se visualizar de forma esquemática quais os ficheiros, e programas necessários, assim como os ficheiros gerados na resolução deste problema.

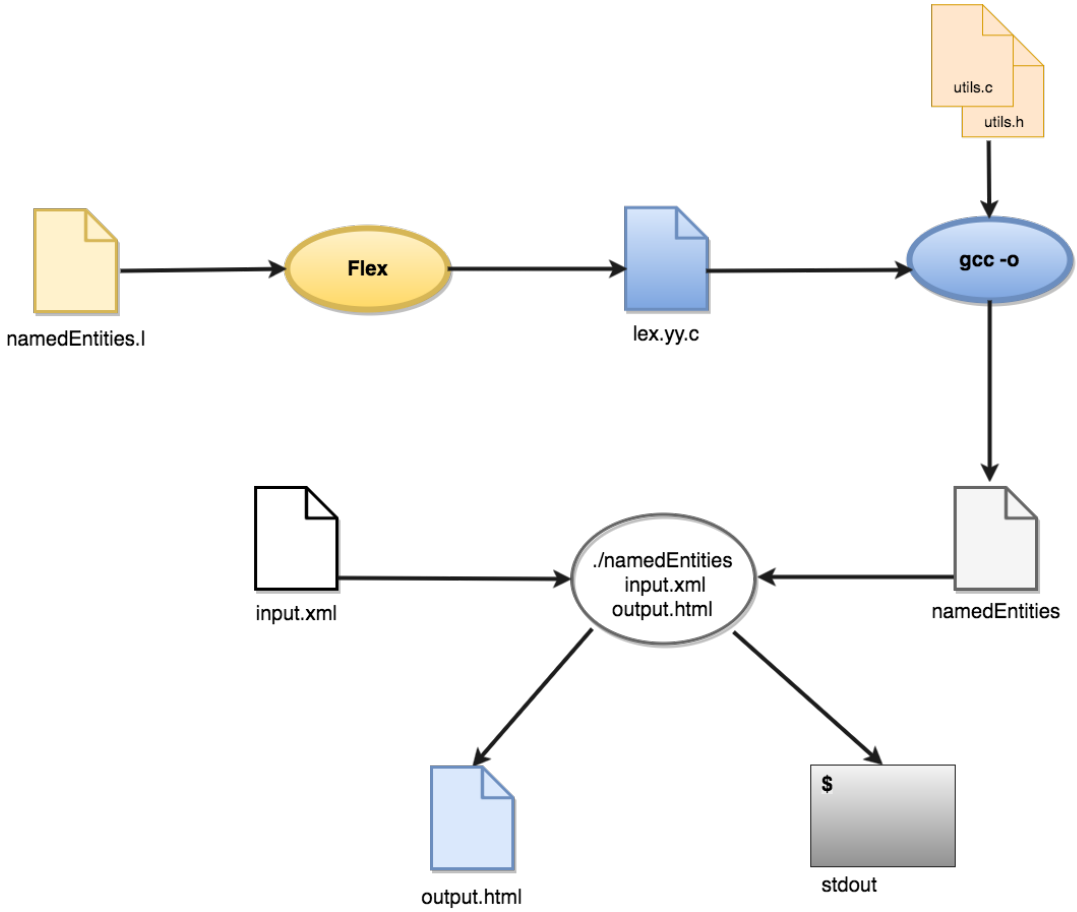

Figura 5.2: Arquitectura do Processador de *Named Entities*

5.4 Especificação

Para resolver o problema proposto, teve-se que extrair a informação necessária do ficheiro de texto fonte, sendo necessário executar diversos passos, descritos de seguida.

5.4.1 Especificar os padrões das Expressões Regulares

A etapa seguinte consistiu na especificação dos padrões através de expressões regulares que permitam capturar os campos pretendidos no texto-fonte.

De forma a facilitar a reutilização e simplificar a leitura das expressões regulares, estas foram definidas com um nome.

- `[ \t\r\n]` → Definida com o nome `whitespace`, faz match com um espaço, ou uma tab ou uma mudança de linha.
- `\<ENAMEX{whitespace}+TYPE=` → Definida com o nome `inicioTagEnamex`, permite fazer match com todas as etiquetas que começam com `<ENAMEX TYPE=`.
- `\<\/ENAMEX\>` → Definida com o nome `fimTagEnamex`, delimita o fim da tag que tem no seu conteúdo o nome de pessoas, cidades e países.
- `[^<]*` → Definida com o nome `conteudoTag`, permite fazer match com tudo até ao fim da tag actual. Uma limitação desta expressão regular é o de não permitir o símbolo `<` no texto que está dentro das tags. (Nota: procurou-se uma solução para este problema, pelo uso de um lookahead positivo: `[.\n]*(?=\<\/ENAMEX\>)`, mas o `flex` não suporta esta operação).
- `\<TIMEX{whitespace}+TYPE=` → Definida com o nome `inicioTagTimex`, permite fazer match com todas as etiquetas que começam com `<TIMEX TYPE=`.
- `\<\/TIMEX\>` → Definida com o nome `fimTagTimex`, delimita o fim de uma tag de tempo.
- `[A-Za-z]+` → Definida com o nome `palavra`, faz match com uma palavra normal. Uma palavra é uma sequência de letras de A a Z, minúscula ou maiúscula, que pode ocorrer uma ou mais vezes.
- `[0-9]` → Definida com o nome `digito`, permite fazer match com um dígito de 0 a 9.
- `{digito}{3,4}` → Definida com o nome `ano`, permite fazer match com um número de 3 ou 4 dígitos. Não se suporta anos com menos dígitos porque seria complicado distinguir anos de dias e meses.
- `{digito}{1,2}` → Definida com o nome `dia`, permite fazer match com qualquer número com 1 ou 2 dígitos.
- `{digito}{1,2}` → Definida com o nome `mesNumerico`, faz match com meses definidos de forma númerica.
- `{palavra}` → Definida com o nome `mesExtenso`, faz match com meses definidos por extenso.
- `{dia}[- \/]{mesNumerico}[- \/]{ano}` → Definida com o nome `data1`, faz match com a seguinte representação das datas `dia/mês/ano` ou `dia-mês-ano`.
- `{dia}{whitespace}+de{whitespace}+{mesExtenso}{whitespace}+de{whitespace}+{ano}` → Definida com o nome `data2`, faz match com a seguinte representação de datas, por exemplo `12 de Setembro de 1990`. O dia é seguido de um `de`, depois o mês por extenso, de novo um `de`, e por fim um ano. Entre cada elemento pode há um ou mais `whitespace`.
- `<!--` → Definida com o nome `inicioComentario`, faz match com o início de comentários em documentos em `XML`.
- `-->` → Definida com o nome `fimComentario`, faz match com o delimitador de fim de comentários em documentos em `XML`.

5.4.2 Identificar Estados e Acções Semânticas

Para facilitar a definição de regras no `Flex`, definiu-se 5 estados:

- `NOME`
- `CIDADE`
- `PAIS`
- `ANO`
- `COMENTARIO`

NOME

O estado `NOME` ficará activo quando encontrar as tags `ENAMEX` referentes ao tipo Pessoa. Este estado torna-se activo aquando do início das seguintes tags:

```
{inicioTagEnamex}\"PERSON\"\>␣{
␣␣␣␣BEGIN␣NOME;
}

{inicioTagEnamex}\"NAME\"\>␣{
␣␣␣␣BEGIN␣NOME;
}

{inicioTagEnamex}\"NAME\"{whitespace}+CERTEZA=\"\?\?\"\>␣{
␣␣␣␣BEGIN␣NOME;
}
```

Depois de activado o estado `NOME`, é extraído todo o texto até ao fim da tag. Para cada texto extraído executam-se as seguintes ações semânticas:

- Retirar espaços/mudanças de linha/tabs do conteúdo;
- Inserir o conteúdo já processado numa estrutura de dados adequada.

Quando se encontra o fim da tag, o estado `NOME` fica inactivo e activa-se o estado `INITIAL`.

CIDADE

Os estado `CIDADE` ficará activo quando encontrar as tags `ENAMEX` referentes ao tipo Cidade. Este estado torna-se activo aquando do início das seguintes tags:

```
{inicioTagEnamex}\"CITY\"\>␣{
␣␣␣␣BEGIN␣CIDADE;
}
```

```
{inicioTagEnamex}\"LOCATION\"\>␣{
␣␣␣␣BEGIN␣CIDADE;
}
```

Depois de activado o estado `CIDADE`, é extraído todo o texto até ao fim da tag. Para cada texto extraído executam-se as seguintes ações semânticas:

- Retirar espaços/mudanças de linha/tabs do conteúdo;
- Inserir o conteúdo já processado numa estrutura de dados adequada.

Quando se encontra o fim da tag, o estado `CIDADE` fica inactivo e activa-se o estado `INITIAL`.

NOTA: Decidiu-se que uma `LOCATION` iria corresponder a uma cidade, logo ao encontrar-se uma tag com este tipo o estado `CIDADE` ficará activo.

PAIS

Os estado PAIS ficará activo quando encontrar as tags `ENAMEX` referentes ao tipo País. Este estado torna-se activo aquando do início da seguinte tag:

```
{inicioTagEnamex}\"COUNTRY\"\>␣{
␣␣␣␣BEGIN␣PAIS;
}
```

Depois de activado o estado `PAIS`, é extraído todo o texto até ao fim da tag. Para cada texto extraído executam-se as seguintes ações semânticas:

- Retirar espaços/mudanças de linha/tabs do conteúdo;
- Inserir o conteúdo já processado numa estrutura de dados adequada.

Quando se encontra o fim da tag, o estado `PAIS` fica inactivo e activa-se o estado `INITIAL`.

ANO

O estado `ANO` é um pouco mais complexo, pois apenas se quer extrair datas num formato que contenham um ano. Para iniciar o estado, apenas se tenta fazer match com início de uma tag `TIMEX`:

```
{inicioTagTimex}\"DATE\"\>␣{
␣␣␣␣BEGIN␣ANO;
}
```

Depois de activado o estado `ANO`, é extraído todo o texto até ao fim da tag, que faça match com as seguintes expressões regulares:

```
<ANO>{whitespace}*{data1}{whitespace}*
<ANO>{whitespace}*{data2}{whitespace}*
<ANO>{whitespace}*{ano}{whitespace}*
```

Para cada texto extraído executam-se as seguintes ações semânticas:

- Retirar espaços/mudanças de linha/tabs do conteúdo;
- Extrair o ano;
- Inserir o conteúdo já processado numa estrutura de dados adequada.

Para extrair o ano, teve-se que tratar cada tipo de data. Para tal, partiu-se a string correspondente à data pelo separador / ou - ou espaço, de modo a separar os campos individuais da data. Para datas do tipo data1, o ano é o 3º campo; para datas do tipo data2 é o 5º campo; e para datas do tipo ano, é o 1º campo.

Quando se encontra o fim da tag, o estado `ANO` fica inactivo e activa-se o estado `INITIAL`.

COMENTARIO

De modo a identificar e ignorar os possíveis comentários de um ficheiro `XML`, definiu-se o estado `COMENTARIO`. Quando o `flex` encontra a expressão regular `inicioComentario`, inicía-se o estado `COMENTARIO`.

```
{inicioComentario} {
    BEGIN COMENTARIO;
}
```

Neste estado, tudo é ignorado até haver um match com a expressão regular `fimComentario`, que mudará o estado para `INITIAL`.

```
<COMENTARIO>{fimComentario} {
    BEGIN INITIAL;
}
```

Na Figura 5.3 encontram-se representados numa máquina de estados os cinco estados acima descritos.

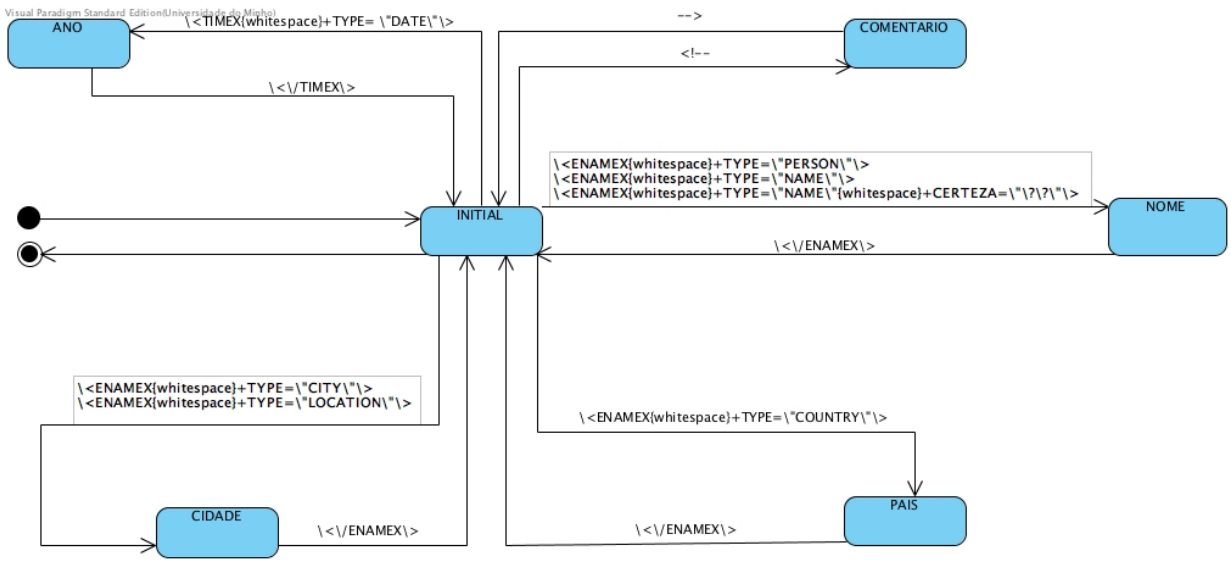

Figura 5.3: Máquina de Estados do processador *Named Entities*

5.4.3 Identificar as Estruturas de Dados globais

De modo a guardar a informação relevante extraída do ficheiro fonte, teve que se recorrer a estruturas de dados em memória de modo a ir armazenado essa informação à medida que se faz a leitura do ficheiro. Posteriormente, essas estruturas serão processadas de modo a imprimir para um página `HTML` e para o `stdout` a informação relevante.

O objectivo deste trabalho visava a recolha dos nomes das pessoas referenciadas no texto, das cidades, paises dos anos. Uma vez que não se pretende ter nomes , cidades, países e anos repetidos optou-se por armazenar cada tipo de informação em uma árvore binária balanceada (`GTree` da biblioteca `GLib`), onde cada nodo da árvore é identificado por uma chave e contém um valor. Como os nomes, cidades, países e anos são as chaves, garante-se que não há repetidos. Adicionalmente, aproveitou-se o valor associado a uma chave para contar quantas vezes essa chave apareceu no texto.

A Figura 5.4 representa de forma esquemática e simples as estruturas de dados escolhidas para armazenar a informação capturada do texto-fonte.

Para além disso, com esta solução os dados estão automaticamente ordenados. Isto é de especial utilidade para obter o menor e maior ano, que correspondem ao elemento mais à esquerda e mais à direita da árvore, respetivamente.

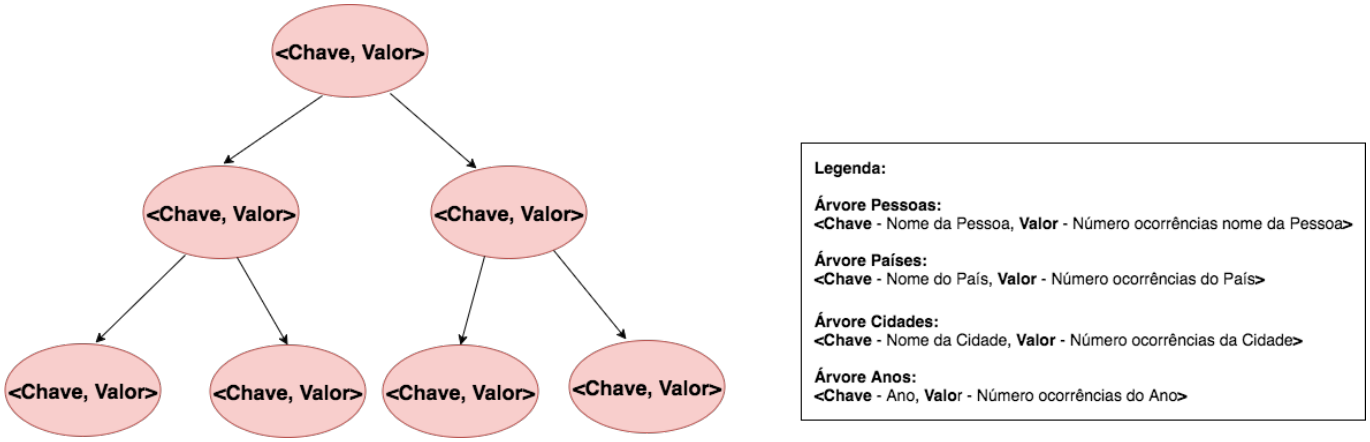

Figura 5.4: Esquema da estrutura de dados- Árvore Binária Balanceada

5.5 Implementação

De modo a executar as ações semânticas, gerar a página `HTML` e imprimir os resultados para a consola, teve-se que implementar algumas funções auxiliares em `C`.

retiraWhitespace

```
char* retiraWhitespace(char* s, int tam);
```

Função que retira espaços, mudanças de linha e tabs que estejam a mais de modo a normalizar a string recebida.

extraiAno

```
char* extraiAno(char* s, int pos);
```

Função que parte a string da data recebida pelos caracteres `/` e `-` e devolve o elemento na posição especificada como argumento.

insereArvore

```
void insereArvore(GTree* a, char* s);
```

Função que insere a string na árvore, incrementado o contador dessa string se ela já existir.

comparaAnos

```
int comparaAnos(char* ano1, char* ano2);
```

Função que converte as strings dos anos de uma árvore, para ordenar os anos de forma numérica.

ficheiroHtml

```
void ficheiroHtml(char* nomeFicheiro, GTree* pessoas, GTree* cidades,
GTree* paises, GTree* datas);
```

Função que recebe as 4 árvores do trabalho e um ficheiro `HTML` com o nome também recebido como parâmetro. A página gerada do `HTML` usam sempre os mesmos ficheiros externos de `javascript` e de `CSS`, para obter o mapa do Google Maps e embelezar a página, respetivamente.

imprimirTreeConsola

```
void imprimirTreeConsola(GTree* t, char* titulo);
```

Esta função recebe uma árvore e imprime na consola o título e todos os nós da árvore, especificando quantas vezes aparece cada chave. Também imprime o total de nós da árvore. Esta função é chamada no programa para cada árvore, para imprimir esta informação na consola.

5.6 Testes e Resultados

O programa pode ser testado com a seguinte execução de comandos:

```
$ make
$ ./namedEntities input.xml output.html
```

NOTA: As páginas `HTML` obtidas encontra-se idêntica ao rascunho definido inicialmente como layout da página.

5.6.1 Ficheiro Fonte exemplo-Enamex1.xml e Resultados

No ficheiro `HTML` encontra-se a informação extraída do ficheiro fonte, `exemplo-Enamex1.xml` com recurso ao `flex`. A página produzida após a execução dos comandos acima definidos com o ficheiro `exemplo-Enamex1.xml` pode ser consultada na Figura 5.5.

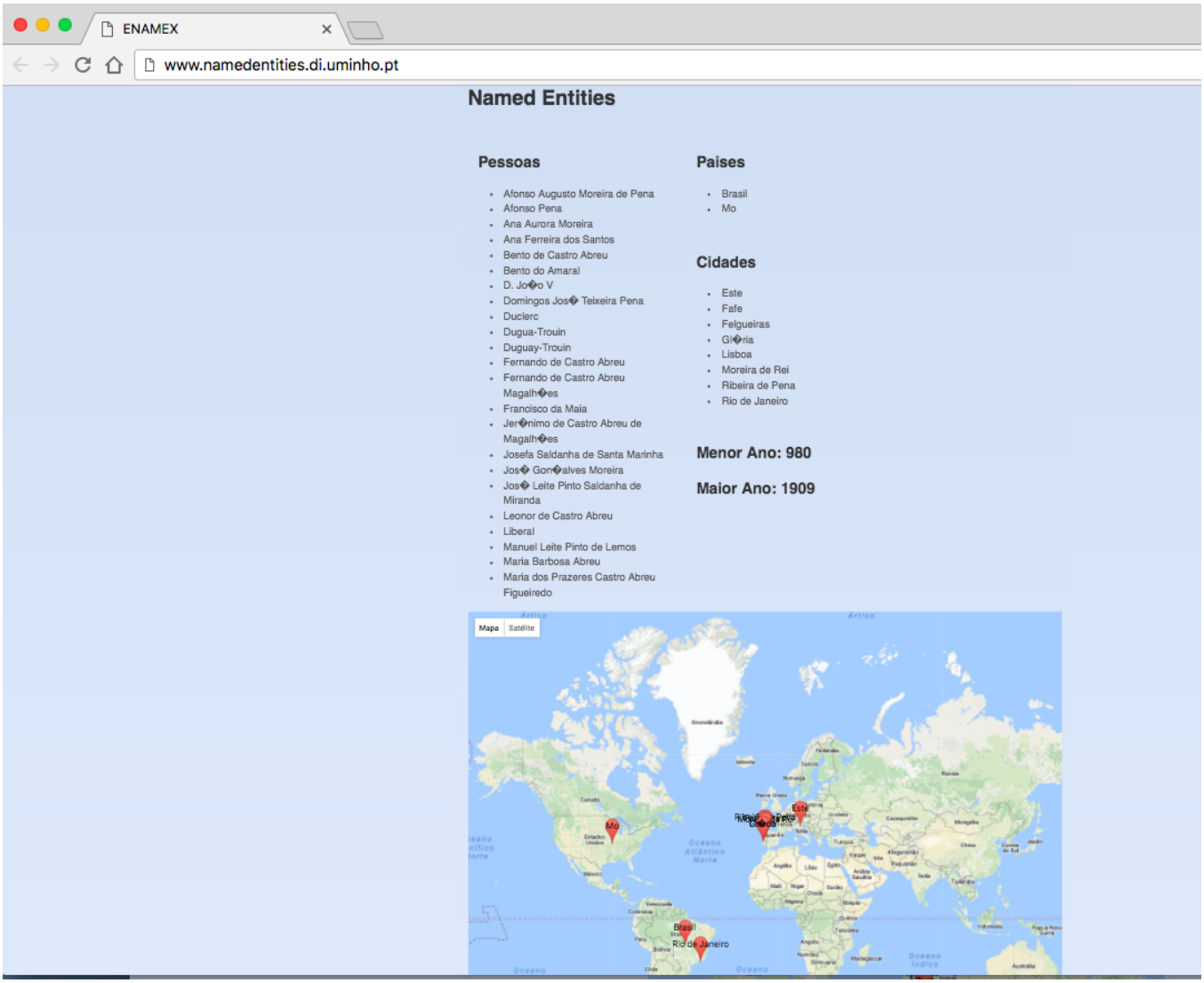

Figura 5.5: Página HTML gerada a partir do ficheiro fonte exemplo-Enamex1.xml

De seguida apresenta-se o resultado obtido no `stdout`.

```
                          Nome Pessoa | Ocorrências
-----------------------------------------------------------------------
          Afonso Augusto Moreira de Pena | 1
                       Afonso Pena | 1
                 Ana Aurora Moreira | 1
             Ana Ferreira dos Santos | 1
               Bento de Castro Abreu | 1
                     Bento do Amaral | 1
                           D. João V | 1
        Domingos José Teixeira Pena | 1
```

Duclerc	1
Dugua-Trouin	1
Duguay-Trouin	1
Fernando de Castro Abreu	1
Fernando de Castro Abreu Magalhães	1
Francisco da Maia	2
Jerónimo de Castro Abreu de Magalhães	1
Josefa Saldanha de Santa Marinha	1
José Gonçalves Moreira	1
José Leite Pinto Saldanha de Miranda	2
Leonor de Castro Abreu	1
Liberal	1
Manuel Leite Pinto de Lemos	1
Maria Barbosa Abreu	1
Maria dos Prazeres Castro Abreu Figueiredo	1
Total Único	23

Cidade	Ocorrências
Este	2
Fafe	3
Felgueiras	1
Glória	1
Lisboa	1
Moreira de Rei	1
Ribeira de Pena	1
Rio de Janeiro	4
Total Único	8

País	Ocorrências
Brasil	3
Mo	2
Total Único	2

Ano	Ocorrências
980	1
1710	1

1711	1
1800	1
1827	1
1848	1
1869	1
1882	1
1883	1
1885	1
1892	1
1895	2
1899	1
1903	1
1906	1
1909	2
Total Único	16

5.6.2 Ficheiro Fonte exemplo-Enamex2.xml e Resultados

Página HTML obtida, ver Figura 5.6, na qual se encontra a informação extraída do ficheiro fonte, `exemplo-Enamex2.xml` com recurso ao `flex`.

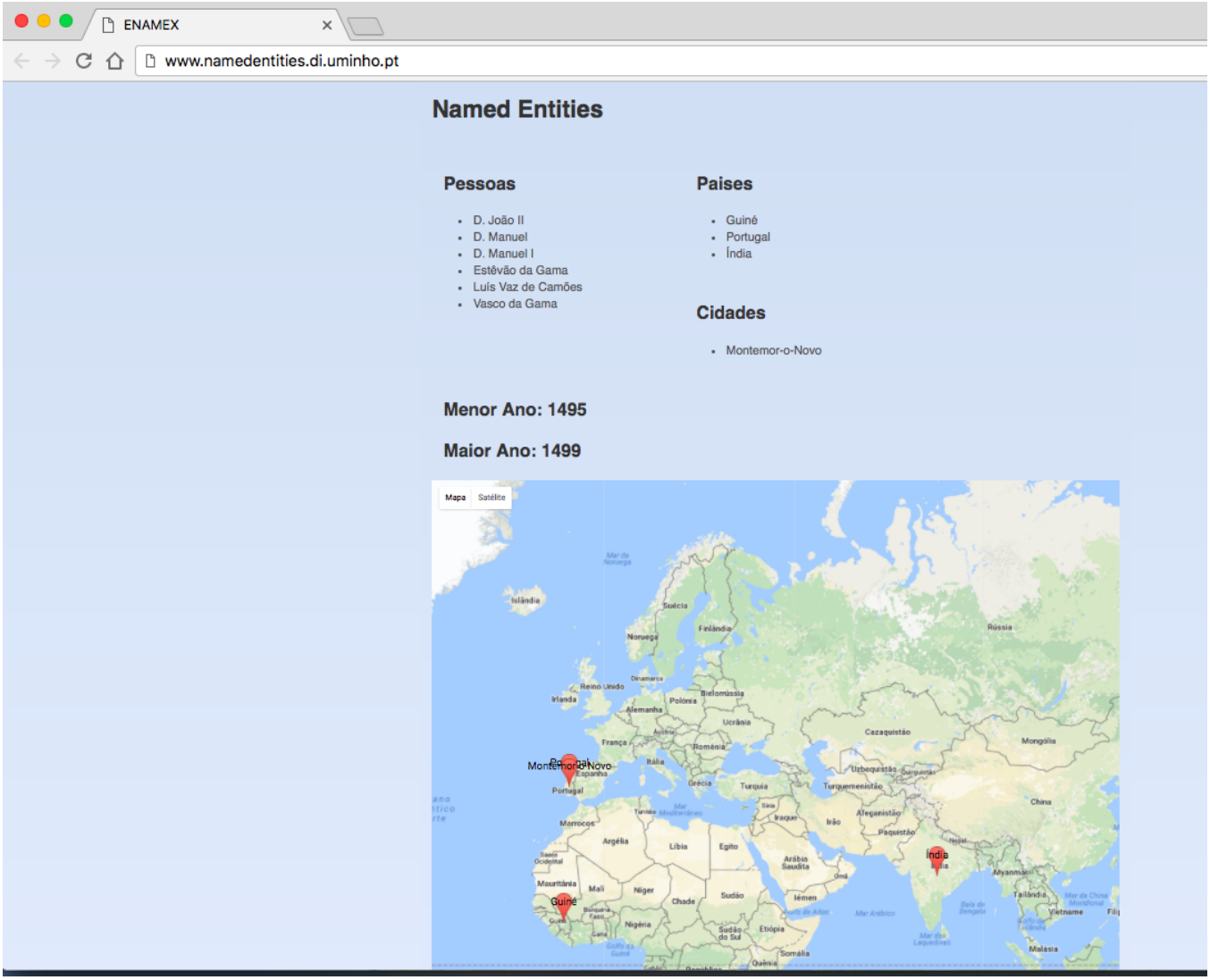

Figura 5.6: Página HTML gerada a partir do ficheiro fonte exemplo-Enamex2.xml

De seguida apresenta-se o resultado obtido no `stdout`.

```
                    Nome Pessoa | Ocorrências
-----------------------------------------------------------------------
                      D. João II | 5
                       D. Manuel | 1
                     D. Manuel I | 2
                  Estêvão da Gama | 1
```

```
                    Luís Vaz de Camões | 1
                         Vasco da Gama | 3
-----------------------------------------------------------------
                          Total Único | 6


                              Cidade | Ocorrências
-----------------------------------------------------------------
                     Montemor-o-Novo | 1
-----------------------------------------------------------------
                          Total Único | 1


                               País | Ocorrências
-----------------------------------------------------------------
                              Guiné | 1
                            Portugal | 2
                              Índia | 3
-----------------------------------------------------------------
                         Total Único | 3


                                Ano | Ocorrências
-----------------------------------------------------------------
                               1495 | 1
                               1497 | 2
                               1499 | 1
-----------------------------------------------------------------
                          Total Único | 3
```

5.6.3 Ficheiro Fonte exemplo-Enamex3.xml e Resultados

Página HTML obtida, ver Figura 5.7, na qual se encontra a informação extraída do ficheiro fonte, `exemplo-Enamex3.xml` com recurso ao `flex`.

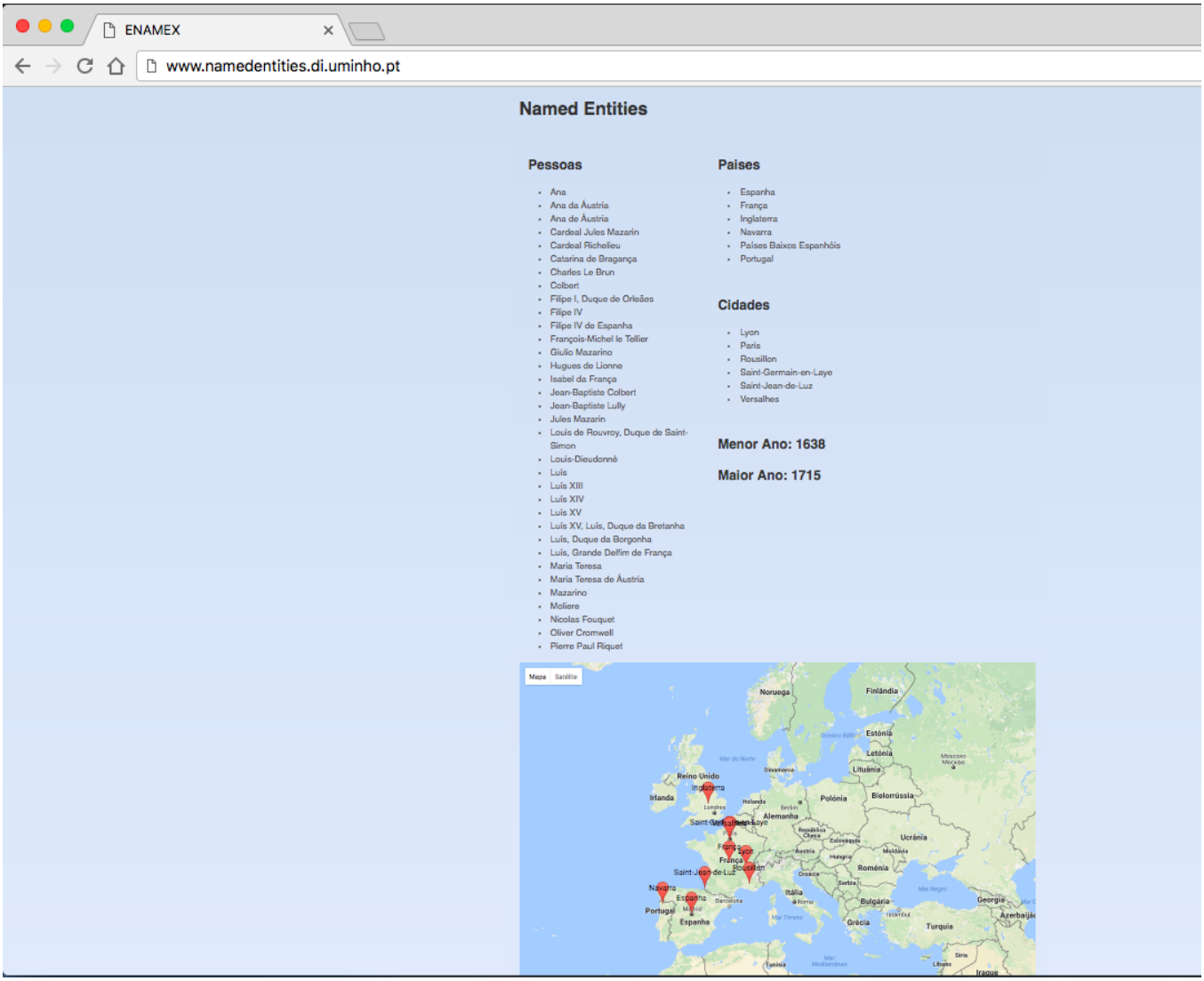

Figura 5.7: Página HTML gerada a partir do ficheiro fonte exemplo-Enamex3.xml

De seguida apresenta-se o resultado obtido no `stdout`.

```
                                Nome Pessoa | Ocorrências
------------------------------------------------------------------------
                                        Ana | 1
                                Ana da Áustria | 2
                                Ana de Áustria | 2
                            Cardeal Jules Mazarin | 1
                                Cardeal Richelieu | 1
                            Catarina de Bragança | 1
                                 Charles Le Brun | 1
                                        Colbert | 2
```

Filipe I, Duque de Orleães	1
Filipe IV	1
Filipe IV de Espanha	1
François-Michel le Tellier	1
Giulio Mazarino	1
Hugues de Lionne	1
Isabel da França	1
Jean-Baptiste Colbert	2
Jean-Baptiste Lully	1
Jules Mazarin	2
Louis de Rouvroy, Duque de Saint-Simon	1
Louis-Dieudonné	1
Luís	6
Luís XIII	5
Luís XIV	7
Luís XV	1
Luís XV, Luís, Duque da Bretanha	1
Luís, Duque da Borgonha	1
Luís, Grande Delfim de França	1
Maria Teresa	1
Maria Teresa de Áustria	1
Mazarino	3
Moliere	1
Nicolas Fouquet	1
Oliver Cromwell	1
Pierre Paul Riquet	1
Total Único	34

Cidade	Ocorrências
Lyon	1
Paris	1
Rousillon	1
Saint-Germain-en-Laye	1
Saint-Jean-de-Luz	1
Versalhes	2
Total Único	6

País	Ocorrências
Espanha	5

França	14
Inglaterra	1
Navarra	1
Países Baixos Espanhóis	1
Portugal	1
Total Único	6

Ano	Ocorrências
1638	2
1643	2
1648	1
1651	1
1658	1
1660	1
1661	2
1665	1
1681	1
1715	1
Total Único	10

Capítulo 6

Conclusão

A elaboração do trabalho prático sobre `Flex` permitiu consolidar e alargar os conhecimentos lecionados nas aulas, uma vez que os problemas abordados têm uma maior dimensão.

Para além disto, a realização deste trabalho possibilitou:

- aumentar a experiência de uso do ambiente `Linux` e de algumas ferramentas de apoio à programação;
- aumentar a capacidade de escrever Expressões Regulares (ER) para descrição de padrões de frases;
- desenvolver, a partir de Expressões Regulares (ERs), sistemática e automaticamente Processadores de Linguagens Regulares, que filtrem ou transformem textos com base no conceito de regras de produção Condição → Ação;
- utilizar o `FLex` para gerar filtros de texto em `C`.

Neste trabalho realizaram-se três enunciados: Processador de Inglês Corrente, Processador de DTDs e Processador de *Named Entities*. Estes problemas consistiram em implementar um processador em `Flex`, de modo a extrair informação ou processar os ficheiros fonte.

Foram desenvolvidos dois Processadores de Inglês Corrente:

- **Processador de Inglês**
  - Que expande contrações como *She'll* ou *We're*
  - Que escreve as abreviaturas de unidades de medida por extenso (m → *metre*)
- **Processador de Verbos**
  - Que lista todos os verbos no infinitivo não flexionado que se encontram no ficheiro fonte

Este tipo de processadores tornam-se numa ferramenta bastante útil para o quotidiano:

- Permite expandir as contrações em inglês, tornando-o o discurso mais formal e cuidado, e mais compreensível. Filtra até as contrações mais difíceis: *She'dn't've → she would not have.*

- Permite escrever por extenso os símbolos e abreviaturas das unidades de medidas mais usadas, tornando o texto mais compreensivo e literário.
- Permite listar todos os verbos no infinitivo não flexionado que existem num texto e respetivas ocorrências, tornando possível de uma forma imediata e clara observar quais os verbos que existem e quais os mais frequentes. É muito útil para perceber qual o tipo de discurso do texto, se possui muitos verbos ou poucos, qual a sua ocorrência, se são verbos mais violentos ou mais alegres, mais simples ou complicados.

Nas áreas científicas utiliza-se com frequência XML para representar todo o tipo de informação. Os DTDs permitem documentar a estrutura da informação que pode ser inserida nos ficheiros XML. Contudo, estes ficheiros, apesar de úteis, são de difícil interpretação. Por isso, o processor de DTD desenvolvido permite perceber facilmente como pode ser estruturado um documento cumprimindo o que se encontra definido no DTD.

Através do processor DTD, tendo a definição do que é um DTD consegue-se:

- perceber facilmente quais são os elementos que se podem utilizar
- as dependências de cada elemento
- os atributos de cada elemento

Além disso, representou-se a informação dos elementos num grafo onde é perceptível os elementos e as suas dependências. A parte da coloração dada nos grafos permite identificar com facilidade quais os elementos que têm maior número de dependências. Apesar de muita de informação do DTD não ter sido utilizada, o processador já se encontra pronto para permitir a sua extração e utilzação em versões futuras.

Foi desenvolvido um processador `Flex` para extrair informação de ficheiros num dialecto `XML`, denominado `ENAMEX`, no qual se fez uso de expressões regulares, assim como de estados, permitindo assim cobrir os diferentes funcionalidades do `Flex`. Para além disso, teve-se que gerar uma página `HTML` na qual se apresentava um Mapa com a localização dos Cidades e Países referenciados no ficheiro fonte.

A realização do enunciado processador de *Named Entities* revelou-se um desafio bastante interessante, uma vez que permitiu alargar conhecimentos para além do `Flex`. Este processador pode ser bastante útil na determinação de uma forma rápida e eficaz de, por exemplo, qual a personagem mais relevante numa história, ou da Cidade, País, ou Ano. Isto porque calcula o número de ocorrências de cada item de um dado tipo.

O foco central da resolução deste trabalho prático foi a consolidação dos conhecimentos adquiridos sobre `Flex`, logo conclui-se que os objectivos foram plenamente cumpridos com a resolução de três enunciados dos seis propostos.

Bibliografia

[1] Eastwood, J. 2002. *Oxford Guide to English Grammar* Oxford University Press

[2] Ramalho, J. C. 2001. *Lex: Um Gerador de analisadores léxicos para linguagens regulares* Universidade do Minho

[3] Lesk, M. E. and Schmidt, E. 1975. *Lex - A Lexical Analyzer Generator*

[4] 1995. *Flex Documentation, version 2.5*

[5] *GLib Reference Manual* Open Source Software Foundation

Appendices

Apêndice A

Filtros de Texto

A.1 Processador de Inglês Corrente

A.1.1 Processador de Inglês

```
%{

#include <stdio.h>
#include <string.h>

%}

%x UNIDADE

%%

ain('|')t               { ECHO; }
gonna                   { printf( "going␣to" ); }
gotta                   { printf( "got␣to" ); }
shan('|')t              { printf( "shall␣not" ); }
won('|')t               { printf( "will␣not" ); }
y('|')all               { printf( "you␣all" ); }


n('|')t         { printf( "␣not" ); }
('|')ve         { printf( "␣have" ); }
('|')d          { printf( "␣would" ); }
('|')ll         { printf( "␣will" ); }
('|')s          { printf( "␣is" ); }
('|')m          { printf( "␣am" ); }
('|')re         { printf( "␣are" ); }
```

```
[0-9]+\ *                       { ECHO;
                                  BEGIN UNIDADE; }

<UNIDADE>m                      { printf( "metres" );
                                  BEGIN INITIAL; }

<UNIDADE>dm                     { printf( "decimetres" );
                                  BEGIN INITIAL; }

<UNIDADE>cm                     { printf( "centimetres" );
                                  BEGIN INITIAL; }

<UNIDADE>mm                     { printf( "millimetres" );
                                  BEGIN INITIAL; }

<UNIDADE>nm                     { printf( "nanometres" );
                                  BEGIN INITIAL; }

<UNIDADE>g                      { printf( "grams" );
                                  BEGIN INITIAL; }

<UNIDADE>mg                     { printf( "milligrams" );
                                  BEGIN INITIAL; }

<UNIDADE>kg                     { printf( "kilograms" );
                                  BEGIN INITIAL; }

<UNIDADE>(s|sec)                { printf( "seconds" );
                                  BEGIN INITIAL; }

<UNIDADE>ms                     { printf( "milliseconds" );
                                  BEGIN INITIAL; }

<UNIDADE>ns                     { printf( "nanoseconds" );
                                  BEGIN INITIAL; }

<UNIDADE>min                    { printf( "minutes" );
                                  BEGIN INITIAL; }

<UNIDADE>h                      { printf( "hours" );
                                  BEGIN INITIAL; }

<UNIDADE>ml                     { printf( "miles" );
                                  BEGIN INITIAL; }
```

```
<UNIDADE>(in|\'')       { printf( "inches" );
                                        BEGIN INITIAL; }

<UNIDADE>(ft|\`)        { printf( "feet" );
                                        BEGIN INITIAL; }

<UNIDADE>yd             { printf( "yards" );
                                        BEGIN INITIAL; }

<UNIDADE>ac             { printf( "acres" );
                                        BEGIN INITIAL; }

<UNIDADE>gal            { printf( "gallons" );
                                        BEGIN INITIAL; }

<UNIDADE>hp             { printf( "horsepower" );
                                        BEGIN INITIAL; }

<UNIDADE>gr             { printf( "grains" );
                                        BEGIN INITIAL; }

<UNIDADE>oz             { printf( "ounces" );
                                        BEGIN INITIAL; }

<UNIDADE>lb             { printf( "pounds" );
                                        BEGIN INITIAL; }

<UNIDADE>mph            { printf( "miles␣per␣hour" );
                                        BEGIN INITIAL; }

<UNIDADE>tsp            { printf( "teaspoons" );
                                        BEGIN INITIAL; }

<UNIDADE>tbsp           { printf( "tablespoons" );
                                        BEGIN INITIAL; }

.|\n                    { ECHO; }

%%

int yywrap(){

        return 1;
}
```

```
int main(){

        yylex();
        return 0;

}
```

A.1.2 Processador de Verbos

```
%{

        #include <stdio.h>
        #include <string.h>
        #include <ctype.h>
        #include "avl.h"
        #include "module.h"

        AVL lista;
        char* verbo;

%}

modalVerb       ?i:(can|could|may|might|must|shall|will|should|would)
pronoun         ?i:(i|you|he|she|it|we|you|they)
question        ?i:("do"|does|did)

%x FRASE
%x GUARDAR

%%

{modalVerb}                     {       BEGIN FRASE; }

\ to                            {       BEGIN GUARDAR; }

{question}\ {pronoun}           {       BEGIN GUARDAR; }

<FRASE>\ {pronoun}              {       BEGIN GUARDAR; }

<FRASE>\ "not"                  {       BEGIN GUARDAR; }

<FRASE>\ [a-zA-Z]+              {       verbo = strdup(yytext);
                                        verbo[strlen(verbo)] = '\0';
```

```
                                        minusculas(verbo);
                                        insereVerbo(verbo, lista);

                                        BEGIN INITIAL; }

<GUARDAR>\ [a-zA-Z]+            {       verbo = strdup(yytext);
                                        verbo[strlen(verbo)] = '\0';
                                        minusculas(verbo);
                                        insereVerbo(verbo, lista);

                                        BEGIN INITIAL; }

.|\n                                    {;}

%%

int yywrap(){

    return 1;
}

int main(int argc, char **argv){

        lista = avl_create(comparaVerbos, NULL, NULL);

        if(argc>=2){
                yyin=fopen(argv[1], "r");
        }
        yylex();

        geraListaVerbos(lista);

        return 0;
}
```

A.2 Processador de DTDs

```
%{
        #include "module.h"

        GTree * elementos;
        Elemento * elementoActual;
%}

%x ELEMENT ELEMDEP ATTLIST ATTRIBUTE ATTNAME ATTTYPE ATTVALUE ENTITY
%option stack

%%

elementos = g_tree_new_full(compararChave, NULL, freeChave, freeElemento);
elementoActual = NULL;

\<!ELEMENT                  { BEGIN ELEMENT; }
\<!ATTLIST                  { BEGIN ATTLIST; }
\<!ENTITY                   { BEGIN ENTITY; }
<ATTLIST>[^ \n]+            { BEGIN ATTRIBUTE;
                              elementoActual = (Elemento*) g_tree_lookup( elementos, yytext );
                            }
<ATTRIBUTE>\>               { BEGIN INITIAL; }
<ATTRIBUTE>[^\>\na-zA-Z]    { BEGIN ATTNAME; }
<ATTNAME>[^ \t]+            { insereAtributo(elementoActual, yytext);
                              BEGIN ATTTYPE;
                            }
<ATTTYPE>[^ \t]+            { BEGIN ATTVALUE; }
<ATTVALUE>\#FIXED" "[^ \>]+ { BEGIN ATTRIBUTE; }
<ATTVALUE>\#REQUIRED        { BEGIN ATTRIBUTE; }
<ATTVALUE>\#IMPLIED         { BEGIN ATTRIBUTE; }
<ATTVALUE>\"[^ \"]+\"       { BEGIN ATTRIBUTE; }
<ELEMENT>\*?\>              { BEGIN INITIAL; }
<ELEMENT>EMPTY              { }
<ELEMENT>[^ E\(\n]+         { elementoActual = insereElemento(elementos, yytext); }
<ELEMENT>\(                 { yy_push_state(ELEMDEP); }
<ELEMDEP>\#PCDATA           { }
<ELEMDEP>[^, \#\|\(\)\*\?]+ { insereDependencia(elementoActual, yytext); }
<ELEMDEP>\(                 { yy_push_state(ELEMDEP); }
<ELEMDEP>\)                 { yy_pop_state(); }
<ENTITY>\>                  { BEGIN INITIAL; }
<*>(.|\n)                   { }
%%
```

```
int yywrap(){
        return 1;
}

int main(int argc, char * argv[]){
        int hflag = 0;
        int gflag = 0;
        int index;
        int c;

        while ((c = getopt (argc, argv, "gh")) != -1) {
                switch (c){
                        case 'g':
                        gflag = 1;
                        break;
                        case 'h':
                        hflag = 1;
                        break;
                        case '?':
                        if (isprint (optopt))
                        printf("Opcao desconhecida `-%c' %s -h para ver ajuda.\n",
            optopt, argv[0]);
                        else
                        printf("Caractere de opcao desconhecido `\\x%x' %s -h para ver ajuda.\n"
            optopt, argv[0]);
                        return 1;
                        default:
                        abort();
                }
        }

        index = optind;

        if (hflag == 1){
                printf("\n%s [-h] [-g ficheiro [elemento]]\n\n", argv[0]);
                printf("\t-h\t\t\t\t\tPara ver a ajuda\n");
                printf("\t-g\tficheiro.dot [elemento]\t\tGera o grafo de dependencias do
        ficheiro DTD de input no output ficheiro.g.");
                printf("\n\t\t\t\t\t\tCaso o elemento seja definido, apenas o
        grafo de dependencias do elemento sera gerado.\n");
                printf("\n\nex: %s -g grafoDependencias.dot fotos\n\n", argv[0]);
        }
        else {
                yylex();

                if (gflag) {
                        int numberOfArguments = argc-index;
```

```
                    if (numberOfArguments == 1) { gerarDependenciasGlobais(elementos,
        argv[index]); }
                    else if (numberOfArguments == 2) { gerarDependencias(elementos,
        argv[index], argv[index+1]); }
                    else { printf("Numero de argumentos invalidos"); }
                }
                else {
                    printf("\nElementos:\n\n");
                    g_tree_foreach( elementos, printElemento, NULL );
                    g_tree_destroy( elementos );
                }
        }

        return 0;
}
```

A.3 Processador de *Named Entities*

```
%option noyywrap
%{
    #include <glib.h>
    #include <stdio.h>
    #include "utils.h"

    GTree* pessoas;
    GTree* cidades;
    GTree* paises;
    GTree* anos;
    char* aux;

%}

whitespace [ \t\r\n]
inicioComentario <!--
fimComentario -->
inicioTagEnamex \<ENAMEX{whitespace}+TYPE=
fimTagEnamex \<\/ENAMEX\>
conteudoTag [^<]*
inicioTagTimex \<TIMEX{whitespace}+TYPE=
fimTagTimex \<\/TIMEX\>
palavra [A-Za-z]+
digito [0-9]
ano {digito}{3,4}
dia {digito}{1,2}
mesNumerico {digito}{1,2}
```

```
mesExtenso {palavra}
data1 {dia}[- \/]{mesNumerico}[- \/]{ano}
data2 {dia}{whitespace}+de{whitespace}+{mesExtenso}{whitespace}+de{whitespace}+{ano}

%x COMENTARIO NOME CIDADE PAIS ANO

%%

<COMENTARIO>{fimComentario} {
    BEGIN INITIAL;
}

<COMENTARIO>.|\n {;
}



<NOME>{fimTagEnamex} {
    BEGIN INITIAL;
}

<NOME>{conteudoTag} {
    aux = retiraWhitespace(yytext, yyleng);
    insereArvore(pessoas, aux);
}



<CIDADE>{fimTagEnamex} {
    BEGIN INITIAL;
}

<CIDADE>{conteudoTag} {
    aux = retiraWhitespace(yytext, yyleng);
    insereArvore(cidades, aux);
}



<PAIS>{fimTagEnamex} {
    BEGIN INITIAL;
}

<PAIS>{conteudoTag} {
    aux = retiraWhitespace(yytext, yyleng);
    insereArvore(paises, aux);
}
```

```
<ANO>{fimTagTimex} {
    BEGIN INITIAL;
}


<ANO>{whitespace}*{data1}{whitespace}* {
    aux = retiraWhitespace(yytext, yyleng);
    aux = extraiAno(aux, 3);
    insereArvore(anos, aux);
}

<ANO>{whitespace}*{data2}{whitespace}* {
    aux = retiraWhitespace(yytext, yyleng);
    aux = extraiAno(aux, 5);
    insereArvore(anos, aux);
}

<ANO>{whitespace}*{ano}{whitespace}* {
    aux = retiraWhitespace(yytext, yyleng);
    aux = extraiAno(aux, 1);
    insereArvore(anos, aux);
}

<ANO>.|\n {;
}



{inicioComentario} {
    BEGIN COMENTARIO;
}

{inicioTagEnamex}\"PERSON\"\> {
    BEGIN NOME;
}

{inicioTagEnamex}\"NAME\"\> {
    BEGIN NOME;
}

{inicioTagEnamex}\"NAME\"{whitespace}+CERTEZA=\"\?\?\"\> {
    BEGIN NOME;
}

{inicioTagEnamex}\"CITY\"\> {
    BEGIN CIDADE;
```

```
}

{inicioTagEnamex}\"LOCATION\"\> {
    BEGIN CIDADE;
}

{inicioTagEnamex}\"COUNTRY\"\> {
    BEGIN PAIS;
}

\<ENAMEX\>{conteudoTag}{fimTagEnamex} {;
}

{inicioTagTimex}\"DATE\"\> {
    BEGIN ANO;
}


.|\n {;
}

%%

int main(int argc, char** argv) {
    if(argc != 3) {
        printf("ERRO: argumentos inválidos ou insuficientes!\n");
        printf("USE: ./namedEntities input.xml output.html\n");
        exit(1);
    }

    yyin = fopen(argv[1], "r");

    pessoas = g_tree_new((GCompareFunc) strcmp);
    cidades = g_tree_new((GCompareFunc) strcmp);
    paises = g_tree_new((GCompareFunc) strcmp);
    anos = g_tree_new((GCompareFunc) comparaAnos);

    yylex();

    ficheiroHtml(argv[2], pessoas, cidades, paises, anos);

    imprimirTreeConsola(pessoas, "Nome Pessoa");
    imprimirTreeConsola(cidades, "Cidade");
    imprimirTreeConsola(paises, "País");
    imprimirTreeConsola(anos, "Ano");

    return 0;
}
```

Apêndice B

Ficheiros Fonte

B.1 Processador de Inglês Corrente

B.1.1 Expandir as Contrações

```
Can't we or won't we create jobs?
We've met before! No, we haven't. I'd have remembered.
Obviously you didn't, because we have.
I think they're very nice boys. It wasn't me!
I shan't dignify that with an answer.
If it wasn't for love we'd have faded away.
Why'd you let me race with your car?
What're you up to? I'm doing the dishes!
I'm not gonna tell you anything. It's gonna rain.
All y'all need to know about cars.
You've already been there. We'll be coming back.
He mustn't've realised we'd be here.
They’d’ven’t been so big! What're you up to?
You'd've thought he would've learned that by now.
There'd've been a time when I'd have had his back.
She'd've made me pay again.
You'ren't who I thought you were. He'dn't that!
I'dn't've done that if I were you! I oughtn't fight.
Joe couldn't keep up with the other guys.
We shan't get away before evening.
```

B.1.2 Unidades de Medida

```
The road was originally about 9.0m wide.
If your step length is 70 cm and your pedometer has recorded 5000 steps, the distance calc
We can replace 628 nm with 6.28 × 10-7 m.
I got a 9 mm. Ready to go off any minute so you feel it.
```

She saved 20 kg of potatoes a year for each son.
The pill has been identified as zolpidem 10 mg.
Please consider that a single can of soft drink contains 35 to 40 g of sugar.
The last 10 sec of his life.
Assume each frame is equal to 16.6666 ms.
Just 1h and 30min to go people!
System update in 43 ns.
I'm gonna be 500 ml away from you.
The length is a very portable 11.8 in.
The finished surface (8-16") is exceedingly smooth.
The Port has a depth of 12.19 m, corresponding to a draught of 40 ft.
My grandmother is only 5' tall.
Turn left and after 100 yd turn right at the roundabout into Paisley Road West.
All 128 ac are covered with seaside paspalum.
These fish need at least 1 tsp of salt per 5 gal of water to be healthy.
The gigantic Queen Mary 2 cruise liner is propelled by four engines, which deliver a total
As at 31 December 2007 the ECB held 18,091,733 oz of fine gold.
The vehicle can carry more than 300lb of cargo.
Why did the wind reach 36.5mph?
Combine a can of condensed milk, 4 tbsp of powdered chocolate and 3 tsp of butter in a pan

B.1.3 Listagem de Verbos

We can take a trip to the hood.
We could go on a trip.
May I use your umbrella?
He may be in the library.
Sorry, I can not understand what you are saying.
The students must behave as I say.
She must be very busy, since she has three children.
Shall we go for a drink after work?
Can I leave now?
Could I cry now?
It is late, you should go home.
She can arrive after dinner.
She must be at the beauty salon.
You should see a dentist.
I can speak English very well.
I could ride a bike when I was a child.
Could I use your bathroom?
Extreme rain could cause the river to flood the city.
Something's telling me it might be you.
Will I wait a lonely lifetime?
I thought that knowledge alone would suffice me.
Did you take your vitamin this morning?

```
Do you have your homework ready?
What material does she like?
You must not be from around here.
I shall not walk alone.
```

B.2 Processador de DTDs

B.2.1 sinopse.dtd

```
<!ELEMENT sinopse (project, title?, author?, date?, text)>
<!ELEMENT text (abstract, section*)>
<!ELEMENT abstract (p | image)*>
<!ELEMENT section (author?,date?,(meta | p | image | ul | ol |dl |section)*)>
<!ATTLIST section
        title CDATA #REQUIRED
        url CDATA #IMPLIED
        encterm CDATA #IMPLIED
>
<!ELEMENT dl (dt,dd)*>
<!ELEMENT dt (#PCDATA|b)*>
<!ELEMENT dd (#PCDATA|ul)*>

<!ELEMENT meta (author | date | obs)*>
<!ELEMENT date (#PCDATA)>
<!ELEMENT obs (#PCDATA)>
<!ELEMENT li (#PCDATA)>
<!ELEMENT ul (li)*>
<!ELEMENT ol (li)*>
<!ELEMENT image (#PCDATA)>
<!ATTLIST image
        href CDATA #REQUIRED
>
<!ELEMENT project (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT p (#PCDATA|a|b)*>
<!ELEMENT ref (#PCDATA)>
<!ATTLIST ref
        type CDATA #IMPLIED
>
<!ELEMENT a (#PCDATA)>
<!ATTLIST a href CDATA #REQUIRED>
<!ELEMENT b (#PCDATA)>
```

B.2.2 bi.dtd

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- edited with XML Spy v4.1 U (http://www.xmlspy.com) by siglab (University of Minho)
 -->
<!ELEMENT bi (projecto, ((depoente, entrevistador?, transcritor?, editor?) | autor),
 foto?, titulo?, biografia?, data?, profissao?, nascimento?, morada?, notas?,rel*)>
<!ELEMENT foto (#PCDATA)>
<!ATTLIST foto
        ficheiro CDATA #REQUIRED
>
<!ELEMENT projecto (#PCDATA)>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT notas (#PCDATA)>
<!ELEMENT depoente (#PCDATA)>
<!ELEMENT entrevistador (#PCDATA)>
<!ELEMENT editor (#PCDATA)>
<!ELEMENT transcritor (#PCDATA)>
<!-- NOTA: biografia deve ler lida como 'resumo' no caso de histórias -->
<!ELEMENT biografia (#PCDATA)>
<!ELEMENT autor (#PCDATA)>
<!ELEMENT data EMPTY>
<!ATTLIST data
        dia CDATA #IMPLIED
        mes CDATA #REQUIRED
        ano CDATA #REQUIRED
>
<!ELEMENT profissao (#PCDATA)>
<!ELEMENT nascimento EMPTY>
<!ATTLIST nascimento
        dia CDATA #IMPLIED
        mes CDATA #IMPLIED
        ano CDATA #REQUIRED
        onde CDATA #IMPLIED
>
<!ELEMENT morada (#PCDATA)>
<!ELEMENT escolaridade (#PCDATA)>
<!ATTLIST depoente
        estadocivil (solteiro | casado | viúvo | divorciado | junto) #IMPLIED
>
<!ELEMENT rel (#PCDATA)>
<!ATTLIST rel tipo CDATA #REQUIRED>
```

B.2.3 fotos.dtd

```
<!ELEMENT fotos (autor*,foto*)>
<!ELEMENT foto (quem, onde?, quando?, facto?, legenda?,autor?)>
        <!ATTLIST foto ficheiro CDATA #REQUIRED>
<!ELEMENT quem (#PCDATA)>
<!ELEMENT quando (#PCDATA)>
        <!ATTLIST quando data CDATA #IMPLIED>
<!ELEMENT onde (#PCDATA)>
<!ELEMENT facto (#PCDATA)>
<!ELEMENT legenda (#PCDATA)>
<!ELEMENT autor (#PCDATA)>
<!ATTLIST autor tipo CDATA #IMPLIED>
```

B.3 Processador de Name Entities

B.3.1 Exemplo-Enamex1.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
<ENAMEX> Vivia </ENAMEX> este herói no <ENAMEX TYPE="CITY"> Rio de
Janeiro </ENAMEX>, <ENAMEX TYPE="COUNTRY">Brasil</ENAMEX> e era
professor não se sabe de que doutrinas no ano de
<TIMEX TYPE="DATE"> 1710 </TIMEX>, quando os franceses comandados por
<ENAMEX TYPE="PERSON"> Duclerc </ENAMEX>, atacaram a cidade.

O governador portou-se mal, e permanecia numa deplorável inacção, quando
<ENAMEX TYPE="PERSON"> Bento do Amaral </ENAMEX> à frente dos seus
estudantes e de paisanos armados, saiu a tomar o passo aos invasores,
repelindo-os energicamente, e dando lugar a que o ataque se malograsse
e os agressores ficassem prisioneiros.

<ENAMEX> Não </ENAMEX> tardou <ENAMEX TYPE="PERSON"> Dugua-Trouin </ENAMEX>
a vir tomar a desforra no ano imediato e desta vez as forças eram
mais numerosas, o chefe mais ágil e mais enérgico, ao passo que o
governador português era ainda o mesmo.
<ENAMEX> Em </ENAMEX> <TIMEX TYPE="DATE">Setembro </TIMEX> de
<TIMEX TYPE="DATE"> 1711 </TIMEX> <ENAMEX TYPE="PERSON">
Duguay-Trouin </ENAMEX>
tomou o <ENAMEX TYPE="CITY">Rio de Janeiro</ENAMEX> sem resistência.

<ENAMEX> Quem </ENAMEX> salvou ainda dessa vez a honra da bandeira,
foi <ENAMEX TYPE="PERSON"> Francisco da Maia</ENAMEX>,
que à frente dos seus cinquenta estudantes combateu desesperadamente
contra os franceses nas proximidades do outeiro
da <ENAMEX TYPE="CITY"> Glória </ENAMEX> no dia <TIMEX TYPE="DATE"> 22
de Setembro de 980</TIMEX>, e ali morreu combatendo como um verdadeiro herói.

<ENAMEX> Os </ENAMEX> próprios inimigos lhe prestaram essa homenagem,
e <ENAMEX TYPE="PERSON"> D. João V</ENAMEX>, por um público diploma
dirigido à família de <ENAMEX TYPE="PERSON">Francisco da Maia</ENAMEX>,
louvou os serviços e a intrepidez do heróico professor.
</document>
--------------------------------------------------------------------------------
<document>
 <ENAMEX TYPE="NAME" CERTEZA="??"> Afonso Pena </ENAMEX>
(<ENAMEX> Presidente da República do Brasil </ENAMEX>)
```

```
<ENAMEX TYPE="NAME" CERTEZA="??"> Afonso Augusto Moreira de Pena </ENAMEX>, nascido em
<TIMEX TYPE="DATE"> 30 de
<ENAMEX>Novembro </ENAMEX> de 1848 </TIMEX>,  no <ENAMEX> Estado de Minas Gerais </ENAMEX> ?
<ENAMEX TYPE="CITY"> Rio de Janeiro </ENAMEX>, licenciado em <ENAMEX> Ciências Jurídico </ENAMEX> ?
<ENAMEX> Sociais </ENAMEX>.
<ENAMEX> Exerceu </ENAMEX> a  advocacia e enveredou pela política no partido
<ENAMEX TYPE="NAME" CERTEZA="??"> Liberal </ENAMEX>.
<ENAMEX> Consta </ENAMEX> do seu currículo: <ENAMEX> Ministro da Guerra </ENAMEX>
(<TIMEX TYPE="DATE"> 1882 </TIMEX>),
<ENAMEX> Ministro da Agricultura </ENAMEX> (<TIMEX TYPE="DATE"> 1883 </TIMEX>), <ENAMEX> Ministro do
 Interior </ENAMEX> e
da <ENAMEX> Justiça </ENAMEX> (<TIMEX TYPE="DATE"> 1885 </TIMEX>), afastou-se temporariamente com o
advento da
<ENAMEX> Republica </ENAMEX> regressando como <ENAMEX> Presidente do Estado de Minas Gerais </ENAMEX>
(<TIMEX TYPE="DATE"> 1892 </TIMEX>), <ENAMEX> Senador </ENAMEX> estadual (<TIMEX TYPE="DATE"> 1899
</TIMEX>),
<ENAMEX> Vice </ENAMEX> ? <ENAMEX> Presidente </ENAMEX> (<TIMEX TYPE="DATE"> 1903 </TIMEX>) e
<ENAMEX> Presidente da República do Brasil </ENAMEX> (<TIMEX TYPE="DATE"> 1906 </TIMEX>) até á sua
morte em
<TIMEX TYPE="DATE"> 1909 </TIMEX>.
<ENAMEX TYPE="LOCATION"> Este </ENAMEX> ilustre ≪brasileiro≫ era filho de
 <ENAMEX TYPE="NAME" CERTEZA="??"> Domingos José
Teixeira Pena </ENAMEX>, natural de <ENAMEX TYPE="LOCATION"> Ribeira de Pena </ENAMEX> e de D.
<ENAMEX TYPE="NAME"> Ana Aurora Moreira </ENAMEX>, filha de <ENAMEX TYPE="NAME" CERTEZA="??"> José
Gonçalves Moreira </ENAMEX>
natural de <ENAMEX TYPE="LOCATION"> Moreira de Rei </ENAMEX>, concelho de <ENAMEX TYPE="LOCATION">
Fafe </ENAMEX> e de
D. <ENAMEX TYPE="NAME"> Ana Ferreira dos Santos </ENAMEX>, natural da então <ENAMEX> Vila de Santa
Bárbara </ENAMEX>,
<ENAMEX> Província de Minas Gerais </ENAMEX>.

 </document>
--------------------------------------------------------------------------------
<document>
<ENAMEX TYPE="NAME" CERTEZA="??"> Bento de Castro Abreu </ENAMEX>,
emigrou para o <ENAMEX TYPE="CITY"> Rio de Janeiro </ENAMEX> em <TIMEX TYPE="DATE"> 1/2/1895
</TIMEX>,
com 26 anos de idade, é referido no seu passaporte como proprietário, sobrinho de outro ≪<ENAMEX>
Brasileiro </ENAMEX>≫
<ENAMEX TYPE="NAME" CERTEZA="??"> Fernando de Castro Abreu </ENAMEX> e <ENAMEX> Magalhães </ENAMEX>,
que apoiou financeiramente a construção da casa do <ENAMEX> Santo Novo </ENAMEX>.

<ENAMEX> Era </ENAMEX> filho de <ENAMEX TYPE="NAME" CERTEZA="??"> José Leite Pinto Saldanha de
Miranda </ENAMEX>,
<TIMEX TYPE="DATE"> 25/1/1827 </TIMEX>, e de <ENAMEX TYPE="NAME" CERTEZA="??"> Maria dos Prazeres
Castro Abreu Figueiredo </ENAMEX>.

<ENAMEX> Neto de António Leite Pinto Saldanha de Miranda </ENAMEX>, da <ENAMEX> Casa de Ambrões
</ENAMEX>,
freguesia de <ENAMEX> São Jorge da Várzea </ENAMEX>, <ENAMEX TYPE="LOCATION"> Felgueiras </ENAMEX> e
 de
<ENAMEX TYPE="NAME" CERTEZA="??"> Leonor de Castro Abreu </ENAMEX> e <ENAMEX> Magalhães </ENAMEX>
(nasc. <TIMEX TYPE="DATE"> 1800 </TIMEX>), da <ENAMEX> Casa do Santo </ENAMEX>-
<ENAMEX TYPE="LOCATION"> Fafe </ENAMEX>.

<ENAMEX> Bisneto </ENAMEX> paterno de <ENAMEX TYPE="NAME" CERTEZA="??"> Manuel Leite Pinto de Lemos
</ENAMEX> e de
<ENAMEX TYPE="NAME" CERTEZA="??"> Josefa Saldanha de Santa Marinha </ENAMEX> e materno de
<ENAMEX TYPE="NAME" CERTEZA="??"> Jerónimo de Castro Abreu de Magalhães </ENAMEX> e de
<ENAMEX TYPE="NAME" CERTEZA="??"> Maria Barbosa Abreu </ENAMEX> e <ENAMEX> Bacelar de São Nicolau
de Basto </ENAMEX>.

<!--
<ENAMEX TYPE="PERSON"> Maria Antonieta </ENAMEX> nasceu a <TIMEX TYPE="DATE"> 21 de Julho de
500 </TIMEX>.
-->
```

```
<ENAMEX TYPE="NAME" CERTEZA="??"> José Leite Pinto Saldanha de Miranda </ENAMEX>, vendo que a
<ENAMEX> Casa do Santo </ENAMEX>
se tornava pequena, mandou construir a <ENAMEX> Casa do Santo Novo </ENAMEX>,
concluída em <TIMEX TYPE="DATE"> 1869 </TIMEX>, situada na rua <ENAMEX> Major Miguel Ferreira
</ENAMEX>,
actualmente propriedade da <ENAMEX> Câmara Municipal de Fafe </ENAMEX>.
<ENAMEX> Fazia </ENAMEX> parte da propriedade o <ENAMEX> Solar do Santo Velho </ENAMEX> com
<ENAMEX> Brasão do Século </ENAMEX> XIX e a quinta anexa, expropriada para expansão da cidade.

A <ENAMEX> Casa do Santo Novo </ENAMEX> apresenta características arquitectónicas do tipo
neoclássico,
estando a sua fachada muito distanciada do alinhamento da rua.
<ENAMEX> Ao </ENAMEX> distanciar-se da rua, parece reconfirmar um estatuto social de família
aristocrática,
ao contrário do que acontece com as restantes casas de ?<ENAMEX> Brasileiros </ENAMEX>? de
<ENAMEX TYPE="LOCATION"> Fafe </ENAMEX>.

<TIMEX TYPE="DATE"> 18/3/1909 </TIMEX>  -  chegou  do  <ENAMEX TYPE="COUNTRY"> Brasil </ENAMEX>
para onde estava
há  anos,  o sr (...), da casa do <ENAMEX> Santo </ENAMEX>, desta vila.

CASTRO ABREU E LEITE, BENTO DE
         <TIMEX TYPE="DATE"> 14/2/1895 </TIMEX>  partiu para os E.U.do <ENAMEX TYPE="COUNTRY">
         Brasil </ENAMEX>,
embarcando em <ENAMEX TYPE="LOCATION"> Lisboa </ENAMEX> no
vapor que daquela cidade saiu, o <ENAMEX> Ex </ENAMEX>.
<ENAMEX TYPE="COUNTRY"> Mo </ENAMEX>.sr(...).
<ENAMEX TYPE="LOCATION">Este </ENAMEX> cavalheiro pertence a nobre família da ilustre
casa do <ENAMEX> Santo </ENAMEX>,(...), dirigiu-se para a casa
do seu tio o <ENAMEX> Ex </ENAMEX>.<ENAMEX TYPE="COUNTRY"> Mo</ENAMEX>. sr.
<ENAMEX TYPE="NAME" CERTEZA="??"> Fernando de Castro Abreu Magalhães </ENAMEX>.
</document>
```

B.3.2 Exemplo-Enamex2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
<!-- Descoberta do caminho marítimo para a Índia -->
A descoberta do caminho marítimo para a <ENAMEX TYPE="COUNTRY"> Índia </ENAMEX> é a designação comum
 para a primeira viagem realizada directamente da Europa para a <ENAMEX TYPE="COUNTRY"> Índia
</ENAMEX>, pelo oceano Atlântico, feita sob o comando do navegador português <ENAMEX TYPE="PERSON">
Vasco da Gama </ENAMEX> durante o reinado do rei <ENAMEX TYPE="PERSON"> D. Manuel I </ENAMEX> , em
<TIMEX TYPE="DATE"> 1497 </TIMEX> - <TIMEX TYPE="DATE"> 1499 </TIMEX>. Uma das mais notáveis viagens
 da era dos Descobrimentos, consolidou a presença marítima e o domínio das rotas comerciais pelos
 portugueses.

O projeto para o caminho marítimo para a <ENAMEX TYPE="COUNTRY"> Índia </ENAMEX> foi delineado pelo
rei português <ENAMEX TYPE="NAME"> D. João II </ENAMEX> como medida de redução dos custos nas trocas
 comerciais com a Ásia e tentativa de monopolizar o comércio das especiarias. A juntar à cada vez
 mais sólida presença marítima portuguesa, <ENAMEX TYPE="NAME"> D. João II </ENAMEX> almejava o
 domínio das rotas comerciais e expansão do reino de <ENAMEX TYPE="COUNTRY"> Portugal </ENAMEX> que
 já se transformava em Império. Porém, o empreendimento não seria realizado durante o seu reinado.
 Seria o seu sucessor, <ENAMEX TYPE="NAME"> D. Manuel I </ENAMEX> que iria designar
 <ENAMEX TYPE="NAME" CERTEZA="??"> Vasco da Gama </ENAMEX> para esta expedição, embora mantendo o
 plano original.

Porém, este empreendimento não era bem visto pelas altas classes. Nas Cortes de <ENAMEX TYPE="CITY">
 Montemor-o-Novo </ENAMEX> de <TIMEX TYPE="DATE"> 1495 </TIMEX> era bem patente a opinião contrária
 quanto à viagem que <ENAMEX TYPE="NAME"> D. João II </ENAMEX> tão esforçadamente havia preparado.
 Contentavam-se com o comércio da <ENAMEX TYPE="COUNTRY"> Guiné </ENAMEX> e do Norte de África e
 temia-se pela manutenção dos eventuais territórios além-mar, pelo custo implicado na expedição e
 manutenção das rotas marítimas que daí adviessem. Esta posição é personificada na personagem do
 Velho do Restelo que aparece, n'Os Lusíadas de <ENAMEX TYPE="PERSON"> Luís Vaz de Camões </ENAMEX>
 , a opor-se ao embarque da armada.

O rei <ENAMEX TYPE="PERSON"> D. Manuel </ENAMEX> não era dessa opinião. Mantendo o plano de
```

```
<ENAMEX TYPE="PERSON"> D. João II </ENAMEX>, mandou aparelhar as naus e escolheu
<ENAMEX TYPE="PERSON"> Vasco da Gama </ENAMEX>, cavaleiro da sua casa, para capitão desta armada.
Curiosamente, segundo o plano original, <ENAMEX TYPE="NAME" CERTEZA="??"> D. João II </ENAMEX>
teria designado seu pai, <ENAMEX TYPE="PERSON"> Estêvão da Gama </ENAMEX>, para chefiar a armada;
mas a esta altura tinham ambos já falecido.

A <TIMEX TYPE="DATE"> 8 de Julho de 1497 </TIMEX>, iniciava-se a expedição semi-planetária que
terminaria dois anos depois com a entrada da nau Bérrio rio Tejo adentro, trazendo a boa-nova que
elevaria <ENAMEX TYPE="COUNTRY"> Portugal </ENAMEX> a uma posição de prestígio marítimo.
</document>
```

B.3.3 Exemplo-Enamex3.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
<!-- Luís XIV de França -->
<ENAMEX TYPE="PERSON"> Luís XIV </ENAMEX> (<ENAMEX TYPE="CITY"> Saint-Germain-en-Laye </ENAMEX>,
<TIMEX TYPE="DATE"> 5 de setembro de 1638 </TIMEX> { <ENAMEX TYPE="CITY"> Versalhes </ENAMEX>,
<TIMEX TYPE="DATE"> 1 de setembro de 1715 </TIMEX>), apelidado de "o Grande" e "Rei Sol", foi o Rei
da <ENAMEX TYPE="COUNTRY"> França </ENAMEX> e <ENAMEX TYPE="COUNTRY"> Navarra </ENAMEX> de
 <TIMEX TYPE="DATE"> 1643 </TIMEX> até à sua morte. Seu reinado de 72 anos é um dos mais longos na
  história europeia. Foi um dos líderes da crescente centralização de poder na era do absolutismo
  europeu.

Era filho do rei <ENAMEX TYPE="PERSON"> Luís XIII </ENAMEX> e de sua esposa <ENAMEX TYPE="PERSON">
Ana da Áustria </ENAMEX>. Seu pai morreu em <TIMEX TYPE="DATE"> 1643 </TIMEX>, quando
<ENAMEX TYPE="PERSON"> Luís </ENAMEX> tinha apenas quatro anos de idade, tendo sua mãe se instaurando
 regente em seu nome. Seu reinado pessoal começou em <TIMEX TYPE="DATE"> 1661 </TIMEX>, após a morte
  do seu principal ministro, o cardeal italiano <ENAMEX TYPE="PERSON"> Jules Mazarin </ENAMEX>.
  <ENAMEX TYPE="PERSON"> Luís </ENAMEX> apoiava o conceito do direito divino dos reis, continuando
  a política de seus predecessores de criar um governo centralizado a partir da capital. Procurou
  eliminar os últimos vestígios de feudalismo que ainda existiam em algumas partes da
  <ENAMEX TYPE="COUNTRY"> França </ENAMEX> e pacificar a aristocracia, oferecendo a muitos membros
da nobreza a oportunidade de morar no seu luxuoso Palácio de Versalhes. Por esses meios
, <ENAMEX TYPE="PERSON"> Luís </ENAMEX> se tornou um dos monarcas franceses mais poderosos da
história e consolidou o sistema da monarquia absoluta que perdurou na <ENAMEX TYPE="COUNTRY">
 França </ENAMEX> até à Revolução Francesa.

Seu reinado viu a <ENAMEX TYPE="COUNTRY"> França </ENAMEX> chegar à liderança das potências
europeias, e lutar em três guerras diferentes: a Guerra Franco-Holandesa, a Guerra dos Nove Anos
e a Guerra da Sucessão Espanhola. Ocorreram ainda os conflitos menores da Guerra de Devolução e
Guerra das Reuniões. <ENAMEX TYPE="PERSON"> Luís </ENAMEX> acabou morrendo alguns dias antes de
completar 77 anos, sendo sucedido por seu bisneto de cinco anos de idade <ENAMEX TYPE="PERSON">
Luís XV </ENAMEX>. Todos os outros herdeiros tinham morrido antes dele: seu filho
<ENAMEX TYPE="PERSON"> Luís, Grande Delfim de França </ENAMEX>, o filho mais velho deste
<ENAMEX TYPE="PERSON"> Luís, Duque da Borgonha </ENAMEX>, e o irmão mais novo de
<ENAMEX TYPE="PERSON"> Luís XV, Luís, Duque da Bretanha </ENAMEX>.
</document>

<document>
<!-- A sua Vida----------------------------------------------------------------------
----------------------------------------------------------------------->
Nasceu em <TIMEX TYPE="DATE"> 1638 </TIMEX>, tendo como seus pais <ENAMEX TYPE="NAME"> Luís XIII
 </ENAMEX> e <ENAMEX TYPE="NAME"> Ana de Áustria </ENAMEX>, que já estavam casados há vinte e
 três anos. Por isso alguns historiadores acreditam que ele não era filho biológico de
 <ENAMEX TYPE="NAME"> Luís XIII </ENAMEX>. Foi batizado <ENAMEX TYPE="NAME"> Louis-Dieudonné
</ENAMEX> ("Luís, o presente de Deus") e recebeu além do tradicional título de Delfim o de Premier
Fils de France ("Primogênito da França").

<ENAMEX TYPE="NAME" CERTEZA="??"> Luís XIII </ENAMEX> e <ENAMEX TYPE="NAME" CERTEZA="??"> Ana da
Áustria </ENAMEX> tiveram um segundo filho, <ENAMEX TYPE="NAME"> Filipe I, Duque de Orleães
 </ENAMEX>. O rei não confiava em sua mulher e procurou evitar que ela ganhasse influência sobre
 o país. Porém, após sua morte em 1643, <ENAMEX TYPE="NAME"> Ana </ENAMEX>tornou-se regente. Ela
 confiou todos os poderes do Estado ao cardeal italiano <ENAMEX TYPE="NAME" CERTEZA="??"> Giulio
 Mazarino </ENAMEX>, que era odiado pela maioria dos círculos políticos franceses.
```

```
Ao mesmo tempo que a Guerra dos Trinta Anos acabava em <TIMEX TYPE="DATE"> 1648 </TIMEX>, uma
guerra civil francesa conhecida como Fronda começou. O <ENAMEX TYPE="NAME" CERTEZA="??"> cardeal
 Mazarino </ENAMEX> deu continuidade à centralização do poder iniciada pelo seu antecessor, o
<ENAMEX TYPE="NAME" CERTEZA="??"> Cardeal Richelieu </ENAMEX>. Tentou aumentar o poder da Coroa
às custas da nobreza e impôs uma taxa aos membros do Parlamento, na época composto na maior
parte pelo alto clero e nobreza.

O Parlamento não só se recusou a pagar como anulou todos os éditos financeiros anteriores
promulgados por <ENAMEX TYPE="NAME"> Mazarino </ENAMEX>, que por conta disso mandou prendê-los,
 o que fez <ENAMEX TYPE="CITY"> Paris </ENAMEX> ser tomada por revoltas. <ENAMEX TYPE="NAME">
 Luís XIV </ENAMEX> e a corte tiveram que deixar a cidade.

Quando o tumulto começou a passar foi assinada a Paz de Vestfália, que restaurou o controle da
Coroa sobre o Exército Francês, e que foi sucedida pela Paz de Rueil, que encerrou os conflitos
temporariamente.
</document>

<document>

<!-------------------------------------------------Começo do seu reinado -------
---------------------------------------------------------->
O período de regência exercido pela mãe de <ENAMEX TYPE="PERSON"> Luís </ENAMEX> terminou
oficialmente em <TIMEX TYPE="DATE"> 1651 </TIMEX>, quando ele tinha 13 anos. <ENAMEX TYPE="PERSON">
 Luís </ENAMEX> assumiu o trono, mas <ENAMEX TYPE="PERSON"> Mazarino </ENAMEX> continuou a controlar
  os assuntos de Estado até à sua morte em <TIMEX TYPE="DATE"> 1661 </TIMEX>. Quando isso aconteceu,
os outros membros do governo esperavam que fosse substituído por <ENAMEX TYPE="NAME"> Nicolas
Fouquet </ENAMEX>, o superintendente de finanças. Ele não só não assumiu como foi preso por má
administração do Tesouro francês. O rei anunciou que assumiria ele próprio o governo do reino.
O seu conselho, o conseil d'en haut, contava com nomes de prestígio como <ENAMEX TYPE="NAME">
Jean-Baptiste Colbert </ENAMEX>, <ENAMEX TYPE="NAME"> Hugues de Lionne </ENAMEX> e
<ENAMEX TYPE="NAME"> François-Michel le Tellier </ENAMEX>. Nenhum destes pertencia a alta
aristocracia, o que levou o grande memorialista do fim do reinado, o Duque e Par do Reino
<ENAMEX TYPE="NAME"> Louis de Rouvroy, Duque de Saint-Simon </ENAMEX> a chamar o governo de
"Reino da pequena burguesia".

O Tesouro estava perto da falência quando <ENAMEX TYPE="NAME" CERTEZA="??"> Luís XIV </ENAMEX>
assumiu o poder. As coisas não melhoraram já que ele gastava dinheiro extravagantemente, despendendo
 vastas somas de dinheiro financiando a Corte Real. Parte desse dinheiro ele gastou como patrono
 das artes, financiando nomes como <ENAMEX TYPE="NAME" CERTEZA="??"> Moliere </ENAMEX>,
 <ENAMEX TYPE="NAME" CERTEZA="??"> Charles Le Brun </ENAMEX> e <ENAMEX TYPE="NAME" CERTEZA="??">
 Jean-Baptiste Lully </ENAMEX>. Também gastou muito em melhorias no antigo Palácio do Louvre, que
 acabou por abandonar em favor da nova fundação de <ENAMEX TYPE="CITY"> Versalhes </ENAMEX>,
 construído sobre um antigo pavilhão de caça de <ENAMEX TYPE="NAME" CERTEZA="??"> Luís XIII </ENAMEX>.
Em <TIMEX TYPE="DATE"> 1665 </TIMEX>, <ENAMEX TYPE="NAME" CERTEZA="??"> Luís XIV </ENAMEX> nomeou
<ENAMEX TYPE="NAME" CERTEZA="??"> Jean-Baptiste Colbert </ENAMEX> para a chefia da Controladoria
Geral. <ENAMEX TYPE="NAME" CERTEZA="??"> Colbert </ENAMEX> reduziu o défice da <ENAMEX TYPE="COUNTRY">
França </ENAMEX> através de uma reforma fiscal, que tornou os impostos mais eficientes.


Seu plano incluía os aides e douanes (ambos taxas comerciais), a gabelle (imposto sobre o sal) e
o taille (imposto sobre as terras). Por outro lado, não aboliu a isenção fiscal de que se valia o
clero e a nobreza. O método de coleta de impostos também foi melhorado.

<ENAMEX TYPE="PERSON"> Colbert </ENAMEX> fez planos de longo prazo para o desenvolvimento da
<ENAMEX TYPE="COUNTRY"> França </ENAMEX> através do comércio. A sua administração criou novas
indústrias e encorajou os fabricantes e inventores a produzir. Também modernizou a Marinha, as
estradas e os aquedutos. Ele é considerado um dos pais da escola de pensamento conhecida como
mercantilismo, sendo que na <ENAMEX TYPE="COUNTRY"> França </ENAMEX> "Colbertismo" é um sinônimo
 de mercantilismo.

<ENAMEX TYPE="PERSON"> Luís XIV </ENAMEX> ordenou a construção do complexo conhecido como Hôtel des
Invalides (Palácio dos Inválidos) para servir de moradia a militares que o serviram lealmente em
combate, mas que foram dispensados por motivo de ferimento de guerra ou idade avançada e que até
 então tinham como alternativas apenas a mendicância e o banditismo.

No seu reinado foi construído o Canal do Midi, que uniu o Mediterrâneo e o Atlântico e foi muito
importante para o desenvolvimento econômico da <ENAMEX TYPE="COUNTRY"> França </ENAMEX> e da Europa,
 sendo considerado fundamental para a Revolução Industrial. O Canal tem 240 km e foi projetado por
```

```
  <ENAMEX TYPE="PERSON"> Pierre Paul Riquet </ENAMEX>, sendo inaugurado em <TIMEX TYPE="DATE"> 1681
   </TIMEX>.
</document>

<document>
<!------------- Casamento ------------------------>
Enquanto a guerra com a <ENAMEX TYPE="COUNTRY"> Espanha </ENAMEX> continuava, os franceses receberam
 apoio militar da <ENAMEX TYPE="COUNTRY"> Inglaterra </ENAMEX> , dirigida por <ENAMEX TYPE="PERSON">
 Oliver Cromwell </ENAMEX>. A aliança Anglo-francesa venceu a guerra em <TIMEX TYPE="DATE"> 1658
</TIMEX> na Batalha das Dunas. O resultado foi o Tratado dos Pirenéus, que fixou a fronteira entre
<ENAMEX TYPE="COUNTRY"> Espanha </ENAMEX> e <ENAMEX TYPE="COUNTRY"> França </ENAMEX>. A
<ENAMEX TYPE="COUNTRY"> Espanha </ENAMEX> cedeu várias províncias e cidades à <ENAMEX TYPE="COUNTRY">
França </ENAMEX> nos <ENAMEX TYPE="COUNTRY"> Países Baixos Espanhóis </ENAMEX> e em
 <ENAMEX TYPE="LOCATION"> Rousillon </ENAMEX>. Como meio de fixar ainda mais a paz e as fronteiras
 dos dois reinos, uma união entre as duas famílias reais, de <ENAMEX TYPE="COUNTRY"> Espanha
</ENAMEX> e de <ENAMEX TYPE="COUNTRY"> França </ENAMEX>, foi proposta. Tal proposta estava
seduzindo imensamente <ENAMEX TYPE="PERSON"> Ana de Áustria </ENAMEX>, a mãe de
<ENAMEX TYPE="PERSON"> Luís XIV </ENAMEX>, que sempre desejou ver seu filho casado com uma
parente sua da Casa de Habsburgo.

Entretanto, a hesitação espanhola conduziu a um esquema em que o <ENAMEX TYPE="PERSON"> Cardeal
Jules Mazarin </ENAMEX>, primeiro-ministro da  <ENAMEX TYPE="COUNTRY"> França </ENAMEX>, fingia
procurar uma união para o rei com <ENAMEX TYPE="PERSON"> Catarina de Bragança </ENAMEX>. Quando
<ENAMEX TYPE="PERSON"> Filipe IV de Espanha </ENAMEX> ouviu sobre a reunião em <ENAMEX TYPE="CITY">
Lyon </ENAMEX> entre as casas de <ENAMEX TYPE="COUNTRY"> França </ENAMEX> e de
<ENAMEX TYPE="COUNTRY"> Portugal </ENAMEX>, enviou uma mensagem especial para a corte francesa,
a fim de abrir as negociações de paz e de um casamento real. Então <ENAMEX TYPE="PERSON"> Luís
XIV </ENAMEX> aceita de boa vontade casar-se com a infanta <ENAMEX TYPE="PERSON"> Maria Teresa
de Áustria </ENAMEX>, filha de <ENAMEX TYPE="PERSON"> Filipe IV </ENAMEX>, rei da Espanha, e
<ENAMEX TYPE="PERSON"> Isabel da França </ENAMEX>, sua tia, irmã de seu pai. O casamento teve
lugar em <TIMEX TYPE="DATE"> 9 de Junho de 1660 </TIMEX> em <ENAMEX TYPE="CITY"> Saint-Jean-de-Luz
</ENAMEX>. Para prevenir uma união das duas coroas, os diplomatas espanhóis incluíram uma
cláusula na qual <ENAMEX TYPE="PERSON"> Maria Teresa </ENAMEX> e seus descendentes seriam
desprovidos de qualquer direito ao trono espanhol. Contudo, pela habilidade de
<ENAMEX TYPE="NAME"> Jules Mazarin </ENAMEX>, a cláusula só seria válida mediante pagamento
de um grande dote. A <ENAMEX TYPE="COUNTRY"> Espanha </ENAMEX> estava empobrecida após
décadas de guerra e foi incapaz de pagar um dote de tais proporções, e
a <ENAMEX TYPE="COUNTRY"> França </ENAMEX> nunca recebeu a quantia acordada de 500.000 Escudos.

</document>
```