

Processamento de Linguagens
(3º ano de Curso)
Trabalho Prático 2
Gedcom
Relatório de Desenvolvimento

Etienne Costa
(a76089)

Pedro Costa
(A85700)

Rui Azevedo
(a80789)

28 de Junho de 2020

Resumo

O presente relatório é referente ao segundo trabalho prático da cadeira **Processamento de Linguagens** e tem como objectivo demonstrar o processo de desenvolvimento de uma gramática tradutora. Essa gramática foi desenvolvida com o objectivo de converter um ficheiro escrito na Linguagem de Domínio Específico *Gedcom* para *HTML*.

Conteúdo

1	Introdução	3
2	Análise e Especificação	4
2.1	Descrição informal do problema	4
2.2	Especificação do Requisitos	4
2.3	Estruturas de dados	5
3	Codificação	6
3.1	Analisador léxico	6
3.2	Analisador sintáctico	8
4	Resultado final	11
5	Conclusão	13

Lista de Figuras

4.1	Página inicial	11
4.2	Página de uma família	12
4.3	Página de um indivíduo	12

Capítulo 1

Introdução

O objectivo do trabalho realizado era criar uma gramática tradutora capaz de traduzir um ficheiro na linguagem de domínio específico, *Gedcom* para *HTML* e, apresentar a informação de maneira a ser possível ser navegável.

A *Gedcom* é uma linguagem usada na área da Genealogia para transmissão e intercâmbio de informação nesta área. A gramática tradutora desenvolvida não abrange na sua totalidade o formato *Gedcom* devido à sua complexidade e dimensão e, por isso, o que foi feito foi reconhecer um sub-conjunto da linguagem.

A aplicação foi desenvolvida na linguagem de programação *C* e foram usadas as ferramentas *flex/yacc* em conjunto, onde o *flex* trata da análise léxica e o *yacc* da análise sintática. Para além disso, foram usadas também as bibliotecas da *glib* a fim de tirar partido de algumas estruturas de dados pertencentes à biblioteca.

Capítulo 2

Análise e Especificação

Neste capítulo será apresentada a descrição informal do problema a resolver bem como os requisitos impostos para a realização do mesmo. Por último, serão apresentadas as estruturas de dados usadas para armazenar a informação.

2.1 Descrição informal do problema

O problema imposto diz respeito ao desenvolvimento de um programa capaz de traduzir ficheiros escritos em *Gedcom 5.5* em *HTML*. A estrutura de um ficheiro *Gedcom*, de uma forma simplificada, é a seguinte:

$$\text{Gedcom_line} := \text{Level} + \text{Delim} + [\text{XRefID} + \text{Delim} +]\text{Tag} + [\text{Delim} + \text{LineValue} +] \text{Terminator}$$

O nível de uma linha representa uma relação hierárquica de informação. As linhas com números maiores estão relacionadas com as de número mais baixos. Esta estrutura hierárquica permite que uma linha tenha várias sub-linhas que, por sua vez pode ser as suas sub-linhas, formando assim um determinado contexto de informação pertence directamente à mesma coisa.

A referência cruzada (*XRefID*) é um valor opcional e o seu significado pode ser equiparado ao de uma chave primária num sistema de base de dados. Qualquer registo que pretenda referenciar outro, usará este valor.

A *tag* numa linha de um ficheiro *Gedcom* identifica o tipo de informação que essa determinada linha contém, da mesma maneira que o nome de um campo identifica o mesmo campo de um registo num sistema de base de dados.

Por fim, uma linha pode ter conteúdo associado à sua *tag*, dado pelo *LineValue*. A estrutura deste valor varia consoante a *tag*.

O resultado final da aplicação deverá ser um conjunto de páginas *HTML* com a informação sobre indivíduos e famílias, sendo a primeira página um índice para os mesmos. Deverá ser possível navegar sobre a informação lida de um ficheiro e a mesma deverá ser apresentada ao utilizador de uma forma clara e sucinta.

2.2 Especificação do Requisitos

Para o desenvolvimento da aplicação deverão ser usadas as ferramentas do ambiente *Linux*, *flex/yacc*. O *flex* é responsável pela análise léxica do ficheiro, gerando os *tokens* para a gramática. O *yacc* será responsável pela análise sintáctica do ficheiro e será através desta ferramenta que a gramática que reconhecerá o ficheiro de entrada vai ser definida. O programa será escrito na linguagem de programação *C*.

2.3 Estruturas de dados

As primeiras estruturas que nos pareceram necessárias para tratar eficiente e eficazmente este conjunto de dados foi uma que guardasse informação sobre uma família e outra que guardasse informação sobre um indivíduo. Criamos então as seguintes estruturas:

```
typedef struct fam_info{
    char* idx;
    char* husb;
    char* wife;
    GPtrArray* children;
}*FamInfo;

typedef struct s_ind{
    char* idx;

    GHashTable* fields;
    GHashTable* display;

    char* last_updated_key;
}*S_Ind;
```

Uma família, tal como um indivíduo, guarda o seu identificador (*idx*). Além disso, no caso da família, interessa-nos apenas saber quem é o marido, a mulher e os filhos (que como podem ser vários, são guardados num array). No caso do indivíduo guardamos duas tabelas de hash. A primeira, chamada *fields* faz um mapeamento entre tags e o seu respetivo valor, por exemplo, "name" pode estar para "Joaquim". A tabela *display* faz o mapeamento entre tags e a forma como devem ser apresentadas nas páginas web. Por exemplo, podemos crer que a tag "name" apareça como "Nome da pessoa:". A variável *last_updated_key* diz-nos qual foi a última chave a ser inserida. É uma variável necessária para se lidar com as tags *cont* e *conc*. Assim, sempre que tivermos de lidar com uma dessas situações, sabemos qual foi a última tag atualizada e podemos simplesmente concatenar a nova informação lá.

Além destas duas estruturas, usamos duas tabelas de hash para fazer mapeamento entre id e família e id e indivíduo. Estas foram usadas numa fase de pós-processamento para adicionar informação assíncrona. Nunca podemos garantir que, por exemplo, quando recebemos um código de indivíduo como marido duma família, ele já existe enquanto indivíduo. Assim, fazemos uma travessia as tabelas no final para adicionar esse tipo de campos.

Capítulo 3

Codificação

Nesta secção serão apresentados os analisadores léxicos e sintáticos que permitem reconhecer um ficheiro de entrada e produzir um determinado conjunto de páginas *HTML* com a informação organizada para apresentação ao utilizador.

Antes disto, é necessário tecer algumas considerações que foram feitas a nível de implementação. Dado que a *Gedcom* é uma linguagem com um grau de complexidade e dimensão consideráveis, optou-se por reconhecer apenas um sub-conjunto da sua informação. Na secção que apresenta o analisador léxico pode-se observar quais as *tags* reconhecidas pela aplicação desenvolvida.

3.1 Analisador léxico

Como já foi referido anteriormente, o analisador léxico é responsável por gerar os *tokens* para o analisador sintático.

As expressões regulares definidas no *flex* para padronizar os *tokens* são referentes aos apontadores de referências cruzadas, aos níveis de uma linha, às *tags* normais e às *tags* de eventos.

Os padrões para reconhecer um nível e um apontador são bastante simples. Para isto, basta reconhecer, respectivamente, um conjunto de números no início da linha, que podem começar, ou não, por um conjunto de espaços ou *tabs*, e algo que na forma de *@anyChar@*. Já o comportamento das *tags* é ligeiramente diferente. As *tags* de eventos reconhecidas têm um comportamento bastante simples pois apenas têm que reconhecer a palavra correspondente à *tag*. As *tags* de eventos reconhecidas pelo analisador léxico são as seguintes:

- BIRT : evento de início de vida, nascimento.
- BURI : evento de disposição dos restos mortais de uma pessoa falecida.
- CHR : evento religioso do baptismo e/ou dar nome a uma criança.
- CHAN : evento que indica uma mudança, correção ou modificação.
- DEAT : evento de fim de uma vida mortal.
- MARR : evento de criação de uma união familiar, casamento.
- TRLR : fim de um ficheiro *gedcom*.

Outro conjunto de *tags* definidas, são as que recebem um *pointer* como valor. Neste caso, embora no analisador léxico não seja explícito que estas *tags* têm conteúdo, no analisador sintático isso é mais visível.

- CHIL : Filho biológico ou adotado.
- FAM : indica que o apontador que precede esta *tag* representa o identificador de uma determinada família.
- INDI : indica que o apontador que precede esta *tag* representa o identificador de um determinado indivíduo.
- FAMC : família onde um determinado indivíduo aparece como filho.
- FAMS : família onde um determinado indivíduo aparece como cônjuge.
- HUSB : indivíduo que tem o papel de homem/pai, no contexto da família.
- WIFE : indivíduo que tem o papel de mulher/mãe, no contexto da família.

O último conjunto de *tags* definidas são as que têm um valor associado. Para estas *tags* foram definidos um conjunto de estados para reconhecer esse mesmo valor. De seguida apresentam-se essas *tags* com o estado correspondente.

- ADDR → INCONTENT : representa o endereço de um local.
- CONC → INCONTENT : usado para concatenação de valores.
- CONT → INCONTENT : usado para concatenação de valores separados por um *carriage return*.
- DIV → INDIV : evento de dissolução de um casamento por meio de ação civil.
- DATE → INDATE : data de um evento.
- NAME → INNAME : conjunto de palavras usadas para reconhecer um indivíduo.
- GIVN → INNAME : nome usado para identificação oficial de um indivíduo.
- NICK → INNAME : apelido de um indivíduo.
- NOTE → INCONTENT : informação adicional sobre um determinado dado.
- PLAC → INCONTENT : identifica o sítio ou a localização de um evento.
- SEX → INSEX : identifica o sexo de um indivíduo.
- TITL → INCONTENT : uma descrição de uma escrita específica ou outra obra, como o título de um livro ou uma designação formal usada por um indivíduo em conexão com posições da realeza ou outro estatuto social.
- TIME → INTIME : hora de um determinado evento.
- NSFX → INCONTENT : texto que aparece numa linha de nome após ou atrás das partes GIVN e/ou NICK.

Os estados definidos têm como objectivo definir os seguintes padrões:

- INCONTENT : apanha qualquer caractere.
- INDIV : apanha o estado de um casal. O padrão definido apanha "yes"ou "y", que significa que um casal está divorciado, ou "no"ou "n", caso contrário. Ambos os casos são *case insensitive*.
- INDATE : reconhece uma data.
- INTIME : reconhece uma hora no formato: H:M:S.
- INNAME : reconhece um nome.
- INSEX : reconhece o sexo de um individuo que pode ser "male"ou "m"no caso do individuo do sexo masculino, ou "female"ou "f"no caso de um individuo do sexo feminino. Ambos casos são *case insensitive*.

Por fim, e uma vez que o *gedcom* aceita *tags* definidas pelo utilizador foi criada uma expressão que reconhece o seguinte padrão: *_alphadigit+* com o estado *INCONTENT* associado.

3.2 Analisador sintáctico

A gramática tradutora foi desenvolvida tendo como base a noção de registos. Começou-se por definir o que é uma *gedcom* com as seguintes produções:

$\langle Gedcom \rangle$	$:=$ Records zero Delim TRLR ;
$\langle Records \rangle$	$:=$ Records Record Record
$\langle Record \rangle$	$:=$ FamRecord IndiRecord

A produção inicial evidencia que uma *gedcom* é um conjunto de registos onde o último registo é a linha de terminação do ficheiro. As seguintes produções definem que um *record* pode ser relativo a uma família ou a um indivíduo. A produção *Records* permite que hajam vários registos tanto de família como de indivíduo.

Um registo de família tem a seguinte estrutura:

$\langle FamRecord \rangle$	$:=$ FamFstEntry FamFstEntry FamInEntries;
$\langle FamFstEntry \rangle$	$:=$ zero Delim Pointer FAM ;
$\langle FamInEntries \rangle$	$:=$ FamInEntry FamInEntries FamInEntry;

A estrutura de um registo de família é a apresentada acima. É composto, obrigatoriamente, pelo registo com o nível zero, de maneira a ser possível identificar cada família pelo seu identificador. Poderá existir também, ou não, mais registos relativos a uma determinada família. Esses registos dizem respeito a informação relativa a uma família e é composta por um sub-conjunto das *tags* definidas na análise léxica.

$\langle FamInEntry \rangle$	$:=$ Level Delim FamTagTypes ;
$\langle FamTagTypes \rangle$	$:=$ FamTags FamEvents EventDetail ;
$\langle FamTags \rangle$	$:=$ HUSB Delim Pointer WIFE Delim Pointer CHIL Delim Pointer DIV div ;
$\langle FamEvents \rangle$	$:=$ MARR ;

As produções acima dizem respeito a uma linha de um registo referente a uma família. Um registo tem que ter obrigatoriamente um nível e uma *tag*. As *tags* foram divididas em três categorias: as *tags* que necessitam de conteúdo, as *tags* que são eventos e as *tags* que caracterizam eventos.

As produções para o reconhecimento de registos de indivíduos são praticamente iguais ao das famílias e são apresentadas de seguida.

$\langle IndiRecord \rangle$	$:=$ IndiFstEntry IndiFstEntry IndiInEntries;
$\langle FamFstEntry \rangle$	$:=$ zero Delim Pointer INDI ;
$\langle FamInEntries \rangle$	$:=$ IndiInEntry IndiInEntries IndiInEntry;
$\langle IndiInEntry \rangle$	$:=$ Level Delim IndiTagTypes ;
$\langle IndiTagTypes \rangle$	$:=$ IndiTags IndiEvents InditDetail ;

Como se pode observar, as produções são praticamente iguais, variando a *tag* correspondente ao nível zero e as *tags* referentes a um indivíduo.

Por fim, apresentam-se as produções que definem as *tags* de um indivíduo.

$\langle IndiTags \rangle$	$:=$ NAME name USRTAG content GIVN name NICK name SEX sex NOTE content CONT content CONC content TITL content FAMC Delim pointer FAMS Delim pointer ;
$\langle IndiEvents \rangle$	$:=$ BIRT BURI DEAT CHR ;
$\langle EventDetail \rangle$	$:=$ DATE date Time time Plac content ADDR content REFN Delim numb ;

Capítulo 4

Resultado final

O resultado final da aplicação, como já foi referido anteriormente, é uma página inicial com os índices das famílias e indivíduos e um conjunto de páginas para cada elemento.

A página inicial, para além do índice, contém também o nome do ficheiro fornecido pelo utilizador. As páginas da família e dos indivíduos, onde deveria aparecer um apontador para um indivíduo, aparece o nome do mesmo, contendo um *link* que redirecciona para a página desse mesmo indivíduo. Os apontadores das famílias redireccionam para as páginas da mesma família.

Index	
Loaded file - dataset/Royal92.ged.txt	
Families	Individuals
F1	I1
F2	I2
F3	I3
F4	I4
F5	I5
F6	I6
F7	I7
F8	I8
F9	I9
F10	I10
F11	I11
F12	I12
F13	I13

Figura 4.1: Página inicial

Family F1

Home

- Divorced: No
- Marriage
 - Date: 10 FEB 1840
 - Place: Chapel Royal, St. James Palace, England
- Husband: [Albert Augustus Charles](#)//
- Wife: [Victoria /Hanover/](#)
- Son: [Victoria Adelaide Mary](#)//
- Son: [Edward_VII /Wettin/](#)
- Son: [Alice Maud Mary](#)//
- Son: [Alfred Ernest Albert](#)//
- Son: [Helena Augusta Victoria](#)//
- Son: [Louise Caroline Alberta](#)//
- Son: [Arthur William Patrick](#)//
- Son: [Leopold George Duncan](#)//
- Son: [Beatrice Mary Victoria](#)//

Figura 4.2: Página de uma família

Individual I1

Home

- Name: [Victoria /Hanover/](#)
- Title: Queen of England
- Birth
 - Date: 24 MAY 1819
 - Place: Kensington, Palace, London, England
- Death
 - Date: 22 JAN 1901
 - Place: Osborne House, Isle of Wight, England
- Bury
 - Place: Royal Mausoleum, Frogmore, Berkshire, England
- Reference
- Family where appears as a husband/wife: [F1](#)
- Sex: Female
- Family where appears as a children: [F42](#)
- Father: [Edward Augustus /Hanover/](#)

Figura 4.3: Página de um indivíduo

Capítulo 5

Conclusão

Os objectivos impostos para a aplicação foram cumpridos na sua totalidade e, como se pode observar, o resultado final apresenta de forma clara a informação ao utilizador.

A maior dificuldade no trabalho foi definir a gramática para a *DSL Gedcom*. A solução desenvolvida para a gramática tornou-se mais simples quando se definiu que a gramática era um conjunto de registos de famílias e indivíduos. A análise léxica foi a parte mais simples do trabalho pois apenas foi necessário definir padrões, relativamente simples, para definir os *tokens* que seriam usados na análise sintáctica.

O resultado final, embora simples, apresenta de forma sucinta a informação e as *tags* definidas abrangem o sub-conjunto mais considerável de dados.