Processamento de Linguagens (3º ano de Curso)

Trabalho Prático 2

Relatório de Desenvolvimento

Nome - João Amorim
(A7806)

13 de Março de 2021

Conteúdo

Capítulo 1

Introdução

O presente trabalho consiste no desenvolvimento de um conversor de GEDCOM 5.5 para XML/HTML, com o objetivo de se obterem várias páginas HTML. A página principal deverá ser constituída por um Índice, contendo os vários ID's de Pessoas Individuais ou Famílias, enquanto que as restantes páginas deverão ser constituídas pelas respetivas informações, sejam estas sobre uma Pessoa ou uma Família, em formato XML.

Para o desenvolvimento deste conversor, será necessária a criação de um analisador léxico, com recurso ao Flex, assim como um analisador sintático, fazendo uso do YACC.

Ao longo deste relatório será explicado, em detalhe, o caminho percorrido para a resolução deste enunciado assim como testes/exemplos e algumas conclusões.

Estrutura do Relatório

Este relatório encontrar-se-á dividido em 3 Capítulos. No primeiro será especificado o Analisador Léxico criado, apresentando todas as suas características.
No segundo capítulo será apresentado o Analisador Sintático, com toda a sua gramática e as acções semânticas mais simples.
Por fim, serão apresentados os passos para o desenvolvimento do Conversor de Gedcom, com todas as decisões tomadas. Ao longo do relatório será apresentado o código respetivo para que haja uma melhor percepção da matéria que está a ser explicada. No final, poderá na mesma ser encontrado em Apêndice todo o código presente neste trabalho.

Capítulo 2

Analisador Léxico

2.1 Expressões Regulares

2.1.1 pointer

O pointer diz respeito aos ID's que são passados, quer de uma Pessoa, quer de uma Família. Este pointer começará e acabará sempre com um '@', sendo o segundo caracter um dígito ou uma letra (maiúscula ou minúsculua). Todos os próximos caracteres poderão ser novamente dígitos ou letras, com a adição de símbolos como o '', o ' ' (espaço) e um conjunto de caracteres que mais à frente será demonstrado, denominado como otherchar.
A expressão regular criada e as suas instruções são :

```
[@][0-9a-zA-Z]([0-9a-zA-Z# ]|{otherchar})+[@]
{ yylval.s = strdup(yytext); return pointer; }
```

2.1.2 otherchar

Esta Expressão Regular está divida em duas partes. A primeira corresponde a uma abreviatura, uma vez que otherchar é constituída por vários intervalos de caracteres:

```
otherchar   [\x21-\x22\x24-\x2F\x3A-\x3F\x5B-\x5E\x60\x7B\-\x7E\x80-\xFE]
```

A segunda corresponde à expressão regular definida. Aquilo que era necessário era simplesmente definir que um otherchar seria o conjunto composto por um ou mais otherchar. A expressão regular criada e as suas instruções são :

```
{otherchar}+     { yylval.s = strdup(yytext); return otherchar; }
```

2.1.3 pal

A pal corresponde simplesmente a um ou mais caracteres do alfabeto, sejam estes minúsculos ou maiúsculos. Esta será posteriormente usada como possível elemento do Value que será demonstrado mais à frente. A expressão regular criada e as suas instruções são :

```
[a-zA-Z]+       { yylval.s = strdup(yytext); return pal; }
```

2.1.4 num

O num é facilmente descrito como um ou mais dígitos. Este será utilizado para caracterizar aquilo que será o Level e usado também como elemento para os Values. A expressão regular criada e as suas instruções são :

```
[0-9]+    { yylval.n = atoi(yytext); return num; }
```

2.1.5 TAGS TERMINATORS e Outros

Na análise léxica são ainda definidas regras para as Tags e Terminators que serão necessárias na gramática e que corresponderão a Palavras Reservadas e a Símbolos Terminais. Estas foram definidas no Analisador Léxico como:

```
\r              { return CARRIAGE; }
\n              { return NEWLINE; }

CHIL            { return CHIL;}
FAM             { return FAM;}
FAMC            { return FAMC;}
HUSB            { return HUSB;}
INDI            { return INDI;}
NAME            { return NAME;}
SEX             { return SEX;}
WIFE            { return WIFE;}
```

As outras duas expressões regulares existentes servem apenas para consumir todos os restantes espaços assim como qualquer outro caracter que não tenha sido definido, sem que seja executada qualquer acção. Estas correspondem a:

```
[ \t]           ;
.               ;
```

Capítulo 3

Compilador

3.1 Gramática

Nesta secção, será apresentada a gramática definida assim como explicadas todas as acções semânticas presentes e absolutamente necessárias para que o Conversor funcione corretamente.

3.1.1 Line

Será a partir deste Símbolo Não-Terminal Line que será estuturada toda a restante gramática. Num ficheiro de GEDCOM podem existir várias linhas de texto que terão que ser lidas e posteriormente analizadas/tratadas. Cada uma destas linhas terá 5 componentes fundamentais:

```
Level - Um ou mais digítos que estará presente no inicio de cada linha.

xRef - Corresponderá ao ID de uma pessoa ou de uma familia.
Poderá não estar presente na linha.

Tag - Conjunto pré-definido de palavras que indicará que tipo de linha estaremos a ler.

Value - Este Value poderá também ser um ID de uma pessoa ou de uma familia,
tendo também como função em certos casos atribuir um valor
à linha de acordo com a Tag fornecida

Terminator - Corresponde ao fim da linha. Será usado para saber que a linha acabou.
```

Uma vez que já sabemos todos os componentes de cada linha, é possível definir a frase para o Não Terminal Line:

```
Line   : Level xRef Tag Value Terminator
```

Sabendo a constiuíção de cada linha, é agora preciso definir a gramática para cada símbolo Não-Terminal presente na mesma. De notar que a análise semântica desta parte será explicada no final, uma vez que para a perceber, é necessário conhecer em primeiro lugar toda a restante gramática.

3.1.2 Level

O Level está apenas definido como Não-Terminal apenas por uma questão de clareza na gramática. Poderíamos simplesmente ter substituído o Level por num. Ficamos então com:

```
Level : num                         {$$ = $1;}
```

A acção semântica, tal como as que vamos ver a seguir, servem apenas para atribuir o valor dos campos ao Não-Terminal. Neste caso, Level é do tipo Inteiro, logo estamos a atribuir o valor de num ao Level.

3.1.3 xRef

xRef corresponde a um Pointer. Este Pointer indica o ID, quer de uma Pessoa, quer de uma Familia e é caracterizado pela palavra variável pointer, que já foi definida como Expressão Regular no Analisador Léxico. Como já tinhamos visto, este pode não existir:

```
xRef : pointer              {$$ = strdup($1);}
     |                      {$$ = '\0';}
     ;
```

Como boa prática, e sendo xRef do tipo String, para o caso em que xRef é vazio, é atribuído a xRef o valor que indica o fim de uma String.

3.1.4 Tag

As Tags correspondem a um conjunto de Palavras Reservadas que indicam que tipo de linha está a ser lida. Apesar de exstirem dezenas de TAGS em GEDCOM, neste trabalho apenas foram definidas algumas, que permitissem obter as relações entre pai/mãe e filhos, assim como obter alguns detalhes sobre cada Pessoa/Familia. As TAGS definidas foram então:

```
Tag : FAM         {$$ = 1;}
    | INDI        {$$ = 2;}
    | FAMC        {$$ = 3;}
    | NAME        {$$ = 4;}
    | HUSB        {$$ = 5;}
    | WIFE        {$$ = 6;}
    | SEX         {$$ = 7;}
    | CHIL        {$$ = 8;}
    ;
```

```
FAM - Linha que indica o início de uma Família com um determinado ID.
INDI - Linha que indica o início de uma Pessoa com um determinado ID.
FAMC - Tag para Pessoa que indica o ID de uma Família no qual esta é filho.
NAME - Tag para Pessoa que indica o nome da Pessoa
```

```
HUSB - Tag para Família que indica o pai da Família
WIFE - Tag para Família que indica a mãe da Família
SEX - Tag para Pessoa que indica o seu sexo.
CHIL - Tag para Família que indica um filho nessa Família.
```

Mais uma vez, as acções semânticas correspondem a atribuições diretas.

3.1.5 Value

Por fim, e aquele que poderá ser visto como o mais complexo, temos o Não-Terminal Value. Este corresponde ao conjunto consituído por Pointers ou um conjunto de números, palavras ou 'otherchar' e poderá ainda não existir. No caso de ser um Pointer, tal como em xRef, este indica um ID. No outro caso, este será usado para caracterizar a Pessoa ou Família, dependendo obviamente da Tag que o precede. Para dar resposta a isto, a gramática criada ficou:

```
Value : pointer       {$$ = strdup($1);}
      | Line_Item     {asprintf(&$$, "%s", $1);}
      |               {$$ = '\0';}
      ;


Line_Item : Anychar               {asprintf(&$$, "%s", $1);}
          | Line_Item Anychar     {asprintf(&$$, "%s %s", $1, $2);}
          ;


Anychar : num            {asprintf(&$$, "%d", $1);}
        | pal            {$$ = strdup($1);}
        | otherchar      {$$ = strdup($1);}
        ;
```

Tal como vimos, Value poderá ser um pointer, vazio ou um Line Item, em que este último é nada mais do que a indicação de que poderemos ter um qualquer conjunto de Anychar (números + palavras + otherchar) em qualquer ordem. As acções semânticas vão atribuíndo os valores dos num, pal e otherchar até estes constituírem o Value final.

3.1.6 Terminator

O Terminator é bastante simples, indicando apenas o fim de uma linha. Este poderá ser aquilo que conhecemos como Carriage Return ou Newline:

```
Terminator : Carriage_Return
           | Newline
           | Carriage_Return Newline
```

```
          ;


Carriage_Return : CARRIAGE
                ;


Newline : NEWLINE
        ;
```

3.1.7 Axioma da Gramática e Recursividade

Para que a gramática ficasse completa eram ainda precisos 2 passos. O primeiro seria permitir que fossem lidas linhas sucessivamente, sem que fosse preciso voltar a executar o programa:

```
Linhas : Linhas Line
       | Line
```

O segundo, correspondente ao Axioma da noss Gramática, necessário para que possamos chamar a função que fará o tratamento de toda a informação em memória, fazendo então a criação dos ficheiros HTML, sendo este o objetivo final de todo o projeto:

```
Gedcom : Linhas
```

Capítulo 4

Criação do Conversor e Testes

Nesta secção será explicada a criação do Conversor, como que Estruturas de Dados foram utilizadas, técnicas utilizadas para escrever e criar ficheiros HTML utilizando os dados em Memória, descrição das análises semânticas mais complexas, entre outros.

4.1 Estruturas

Sendo o objetivo deste Conversor, a criação de páginas HTML, que contenham informação sobre Pessoas ou Famílias, seria necessário, em primeiro lugar, fazer a leitura do ficheiro com linhas GEDCOM e ir guardando todos os dados obtidos em memória. Para isso foram criadas duas Estruturas:

```
typedef struct {
    char id[10];
    char name[40];
    char id_pai[10];
    char id_mae[10];
    char sex[2];
} T_PESSOA;

typedef struct {
    char id[10];
    char id_husb[10];
    char id_wife[10];
    int num_childs;
    char id_childs[15][100];
} T_FAMILIA;
```

A primeira struct diz respeito à informação de uma Pessoa. Aqui teremos variáveis como id (id da Pessoa), name (nome da Pessoa), id pai, (id da Mãe da Pessoa), sex (Sexo da Pessoa). Há que ter em atenção que o id pai terá duas funções. Inicialmente, assim que todas as linhas relativas a uma Pessoa são lidas, este campo terá em si guardado o id da Família a que pertence, vindo da TAG FAMC. Isto irá permitir procurar qual a Família em que esta Pessoa é filho. Depois de lido todo o

ficheiro, os dados serão cruzados e este id pai passará a possuír o então desejado id do Pai relativo à Pessoa em causa.
A segunda struct diz então respeito à informação relativa a uma Família. A variável id diz respeito ao id da Família, id husb ao id do pai dessa Família, id wife ao id da mãe dessa Família, num childs o número de filhos que essa Família possuí e id childs que contêm todos os id's dos mesmos filhos.

4.2 Acções Semânticas

Uma vez criadas as estruturas, é então necessário ler o ficheiro, usando os analisadores já explicados, e guardar os dados lidos dentro das mesmas structs. Para isso, de cada vez que temos uma nova linha é necessário verificar o que foi lido. Para isto temos a seguinte acção semântica para o Símbolo Não Terminal Line:

```
Line   : Level xRef Tag Value Terminator
{
if(pessoas == NULL && familias == NULL){
    pessoas = (T_PESSOA*) malloc(sizeof(T_PESSOA)*1000);
    familias = (T_FAMILIA*) malloc(sizeof(T_FAMILIA)*1000);
}
if($1==0){
  if(state=='p'){
     num_pessoas++;
  }else if (state=='f'){
     num_familias++;
  }
  state=' ';
  n_childs = 0;

  switch($3){
       case 1:
           state='f';
           if (num_familias==cap_familias){
               cap_familias+=1000;
               familias=(T_FAMILIA*)realloc(familias,cap_familias*sizeof(T_FAMILIA));
           }
           strncpy(familias[num_familias].id, $2 + 1, strlen($2)-2);
           familias[num_familias].id_husb[0]='\0';
           familias[num_familias].id_wife[0]='\0';
           break;

       case 2:
           state='p';
           if (num_pessoas==cap_pessoas){
               cap_pessoas+=1000;
```

```
                pessoas=(T_PESSOA*)realloc(pessoas,cap_pessoas*sizeof(T_PESSOA));
            }
            strncpy(pessoas[num_pessoas].id, $2 + 1, strlen($2)-2);
            pessoas[num_pessoas].name[0]='\0';
            pessoas[num_pessoas].id_pai[0]='\0';
            pessoas[num_pessoas].id_mae[0]='\0';
            break;
            }
            }else if (state=='p'){
                    switch($3){
                        case 3:
                            strncpy(pessoas[num_pessoas].id_pai, $4 + 1, strlen($4)-2);
                            break;
                        case 4:
                            strncpy(pessoas[num_pessoas].name, $4, strlen($4));
                            break;
                        case 7:
                            strncpy(pessoas[num_pessoas].sex, $4, strlen($4));
                            break;

                    }
            }else if (state=='f'){
                switch($3){
                    case 5:
                        strncpy(familias[num_familias].id_husb, $4 + 1, strlen($4)-2);
                        break;
                    case 6:
                        strncpy(familias[num_familias].id_wife, $4 + 1, strlen($4)-2);
                        break;
                    case 8:
                        strncpy(familias[num_familias].id_childs[n_childs],
                                $4 + 1, strlen($4)-2);
                        familias[num_familias].num_childs++;
                        n_childs++;
                           break;
                }

            }
}
       ;
```

Apesar de bastante grande, esta acção semântica, pode ser dividida inicialmente em 2 partes, em que cada uma destas partes fará praticamente o mesmo, escrever ou na estutura de dados de uma Pessoa ou de uma Família.

Na primeira divisão, é verificado se o Level corresponde ao valor 0. Se corresponder, é preciso perceber se a TAG que estamos a ler corresponde a FAM, onde estaremos a receber o id de uma Família, ou a INDI, onde recebemos o id de uma Pessoa. Em ambos os casos, iremos guardar o id recebido na sua respetiva estrutura, na posição da struct em que estamos, assim como incializados os restantes campos da mesma struct. Esta posição é dada pelo num pessoas ou num familias.
Assim que esta primeira linha é lida iremos também atualizar o Estado. Este Estado, correspondente aos caracteres 'f' e 'p', indica se as próximas linhas que serão lidas dirão respeito a Pessoas (para 'p') ou Familias (para 'f').

Como vimos, a primeira divisão corresponde a linhas começadas por 0. Ora, a segunda parte diz respeito a todas as outras linhas. ou seja, linhas que não tenham como Level o 0. Uma vez que após a leitura da primeira linha actualizamos o estado, sabemos se estamos neste momento a ler uma linha referente a uma Pessoa ou a uma Família. Com esta informação, apenas necessitamos de verificar a Tag da linha e guardar os dados nas Estruturas de acordo com a mesma.
Desta forma, da próxima vez que uma nova linha com Level 0 for lida, já teremos toda a informação relativa a uma Pessoa ou Família guardada, incrementando o respetivo número para que consigamos trabalhar devidamente com as estruturas e dando reset ao Estado novamente para ' ', pronto a ler uma nova Linha com Level 0.

4.3 Conversor.c

Finalmente e não menos importante, temos o ficheiro onde será feita e conversão das linhas GEDCOM para XML/HTML. Aqui está presente uma função - criaHTML(). Esta função irá receber como parâmetros as Structs de Pessoa e Família, assim como o número de Pessoas e Famílias que foram lidas com sucesso, e vai, com os dados recebidos, criar os ficheiros HTML pretendidos.
De notar que a abordagem escolhida foi a de criar uma Mainpage, onde estarão os Índices de todas as Pessoas e Famílias (compostos pelos ID's das mesmas), em que cada Índice possuí uma Hiperligação para uma outra página, em que estará presente a informação da Pessoa ou Família, de acordo com aquilo em que se tiver escolhido. Esta informação será apresentada no formato XML, dentro de uma ¡textarea¿, tendo sido o único elemento que arranjei que mantinha a estrutura XML dentro de um ficheiro HTML.

De seguida é apresentada a função criaHTML():

```
void criaHTML(T_PESSOA * pessoas,int num_pessoas,
T_FAMILIA * familias, int num_familias){

    FILE * fp;

    fp = fopen("Mainpage.html","w");

    for(int p =0; p<num_pessoas;p++){
```

```
        // Encontrar a familia
        for(int f=0;f<num_familias;f++){
            if (strcmp(familias[f].id,pessoas[p].id_pai)==0){
                strcpy(pessoas[p].id_pai,familias[f].id_husb);
                strcpy(pessoas[p].id_mae,familias[f].id_wife);
                break;
            }
        }
    }

    // Construção da Mainpage.html

    fprintf(fp,"<html>\n<body>\n");
    fprintf(fp,"<div> Esta página tem como objetivo mostrar
            as Pessoas e Familias obtidas de um ficheiro no formato GEDCOM,
            apresentando para cada pessoa, as relações pai/mãe e filho obtidas
            da análise de cada Familia. Para mais informações sobre este
            formato de formato Genealógico, visite <a href=\"http://homepages.
            rootsweb.com/~pmcbride/gedcom/55gcch1.htm\">GEDCOM</a>");

    fprintf(fp,"<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n<genoa>\n");
    fprintf(fp,"<p><h2>Lista de Pessoas: </h2>");
    for(int p =0; p<num_pessoas;p++){
        fprintf(fp,"<p>Pessoa: <a href=\"Pessoa%s.html\">%s</a></p>",
                pessoas[p].id, pessoas[p].id);
    }
    fprintf(fp,"<p><h2>Lista de Familias: </h2>");
    for(int f =0; f<num_familias;f++){
        fprintf(fp,"<p>Familia: <a href=\"Familia%s.html\">%s</a></p>",
                familias[f].id, familias[f].id);
    }
    fprintf(fp,"</genoa>\n</body>\n</html>");


// Construção das páginas HTML referentes aos indices
// Páginas das Pessoas

    for(int p = 0; p<num_pessoas;p++){
        FILE * file;
        char buf[22];
        sprintf(buf, "Pessoa%s.html", pessoas[p].id);
        file = fopen(buf,"w");

        fprintf(file,"<html>\n<body>\n<textarea rows=\"50\" cols=\"150\"
                style=\"border:none;\">\n\n");
```

```
        fprintf(file,"<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n\n<genoa>\n");
        fprintf(file,"\t<pessoa>\n");
        fprintf(file,"\t\t<id>%s</id>\n", pessoas[p].id);
        fprintf(file,"\t\t<nome>%s</nome>\n", pessoas[p].name);
        if (strlen(pessoas[p].id_pai)==0)
            fprintf(file,"\t\t<pai/>\n");
        else
            fprintf(file,"\t\t<pai>%s</pai>\n", pessoas[p].id_pai);
        if (strlen(pessoas[p].id_mae)==0)
            fprintf(file,"\t\t<mae/>\n");
        else
            fprintf(file,"\t\t<mae>%s</mae>\n", pessoas[p].id_mae);

        if (strlen(pessoas[p].sex)==0)
            fprintf(file,"\t\t<sexo/>\n");
        else
            fprintf(file,"\t\t<sexo>%s</sexo>\n", pessoas[p].sex);
        fprintf(file,"\t</pessoa>\n</genoa>\n</textarea>\n</body>\n</html>\n");
    }

//Páginas das Familias

    for(int f = 0; f<num_familias;f++){
        FILE * file;
        char buf[22];
        sprintf(buf, "Familia%s.html", familias[f].id);
        file = fopen(buf,"w");

        fprintf(file,"<html>\n<body>\n<textarea rows=\"50\" cols=\"150\"
               style=\"border:none;\">\n\n");
        fprintf(file,"<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n\n<genoa>\n");
        fprintf(file,"\t<familia>\n");
        fprintf(file,"\t\t<id>%s</id>\n", familias[f].id);
        if (strlen(familias[f].id_husb)==0)
            fprintf(file,"\t\t<pai/>\n");
        else
            fprintf(file,"\t\t<pai>%s</pai>\n", familias[f].id_husb);
        if (strlen(familias[f].id_wife)==0)
           fprintf(file,"\t\t<mae/>\n");
        else
           fprintf(file,"\t\t<mae>%s</mae>\n", familias[f].id_wife);
        if(familias[f].num_childs==0){
           fprintf(file,"\t\t<filho/>\n");
        }else{
            for(int i = 0; i<familias[f].num_childs; i++){
```

```
            fprintf(file,"\t\t<filho>%s</filho>\n", familias[f].id_childs[i]);
            }
        }
        fprintf(file,"\t</familia>\n</genoa>\n</textarea>\n</body>\n</html>\n");
    }
}
```

Em primeiro lugar, é feita a procura da Família de cada Pessoa em memória para que as variáveis id pai e id mae possuam os valores corretos ou seja, os valores dos id husb e id wife presentes na respetiva Família.

Uma vez efetuada esta procura, temos então todas as informações completas de todas as Pessoas e Famílias, apenas precisamos de criar as páginas HTML.
Esta criação é efetuada com o recurso ao fopen e ao fprintf. Com o primeiro iremos retornar um file pointer para o ficheiro onde iremos escrever, que será usado pelo fprintf, que será quem irá preencher o nosso ficheiro.

Para a Mainpage, é criada uma estrutura simples em HTML, com os Índices (ID's), com recurso às Hiperligações. No caso das restantes páginas, para cada Pessoa e cada Família, será criado um novo File Pointer, para um novo ficheiro, que terá igualmente uma estrutura em HTML, onde serão escritas as informações numa estrutura XML, ficando assim concluído o Conversor de GEDCOM.

4.4 Testes e Resultados Finais

4.4.1 Makefile

A Makefile usada, possui aquilo que foi fornecido nas aulas Práticas, com a adição do clean para fazer a remoção de todos os ficheiros HTML, uma vez que se existirem 20 Pessoas e 10 Famílias, irão ser criados 30 ficheiros.

```
gedcom.exe : y.tab.o lex.yy.o
           gcc -o gedcom.exe y.tab.o lex.yy.o -ll

y.tab. : y.tab.c
         gcc -c y.tab.c

lex.yy.o : lex.yy.c
           gcc -c lex.yy.c

y.tab.c y.tab.h : gedcom.y
                  yacc -d test.y

lex.yy.c : gedcom.l y.tab.h
           flex gedcom.l

clean:
      rm -f *.html
```

4.4.2 Ficheiro de Teste

Como ficheiro de teste, foi criado um ficheiro com o nome GEDCOM.txt, que contêm 15 Pessoas e 3 Famílias. Para a geração de todos os ficheiros HTML a partir do executável, é executado o seguinte comando:

```
$ cat GEDCOM.txt | ./gedcom.exe
```

Este enviará o output do cat do ficheiro GEDCOM.txt, para o Input do Executável, onde todas as linhas serão lidas e tratadas.

4.4.3 Resultados

Serão aqui apresentadas imagens da Mainpage, de uma Página gerada para uma Pessoa e uma para uma Família.

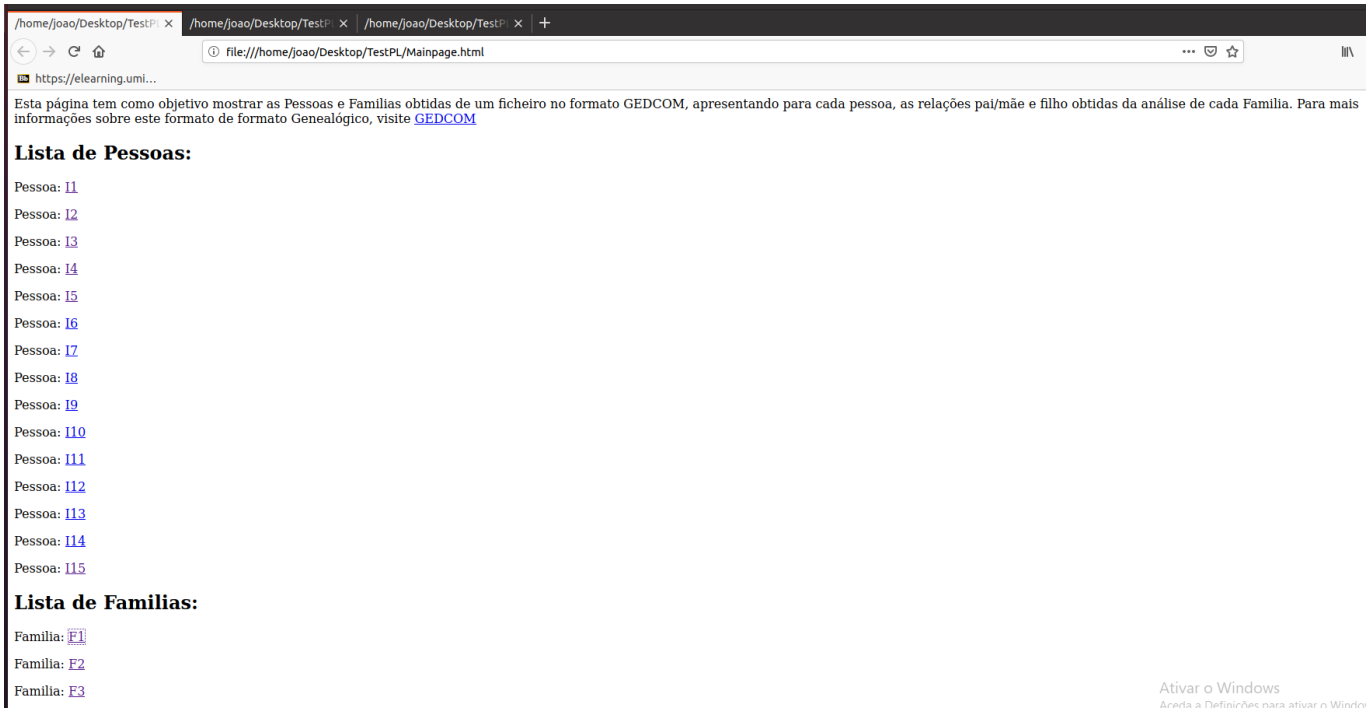

Figura 4.1: Mainpage.html

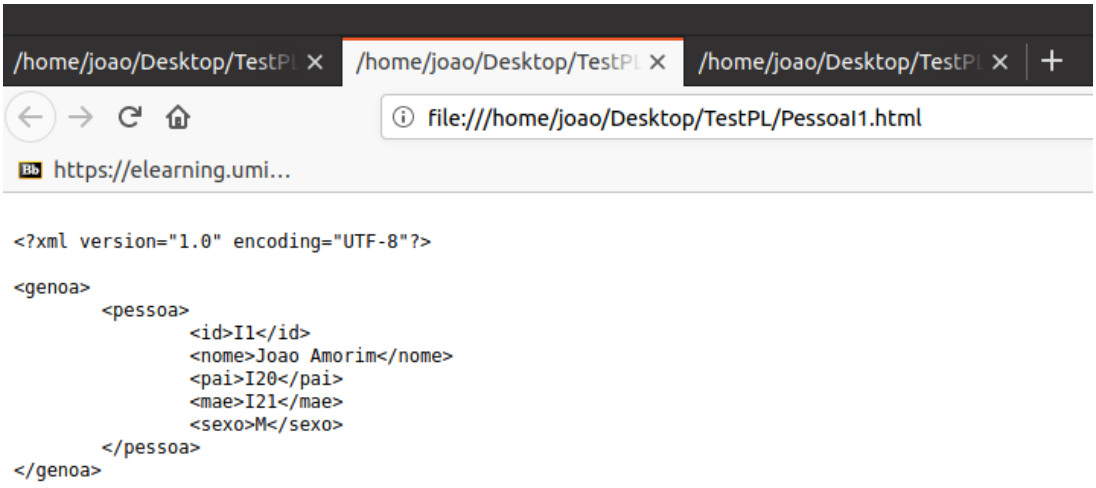

Figura 4.2: PessoaI1.html

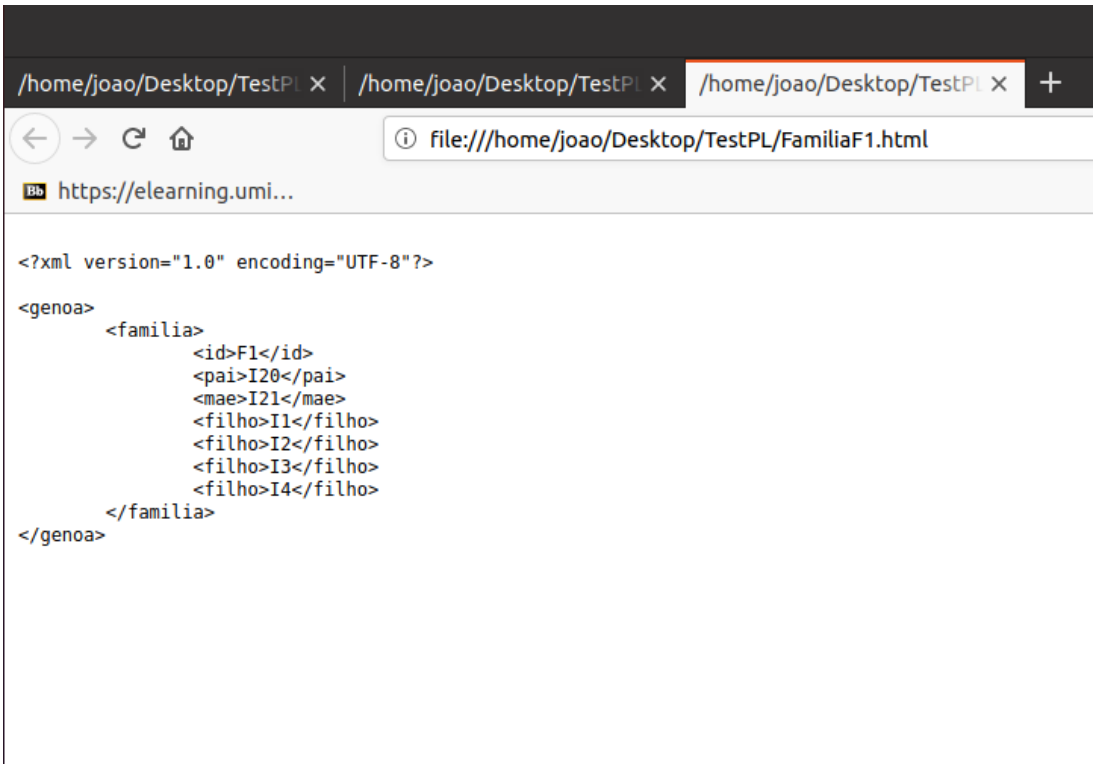

Figura 4.3: FamiliaF1.html

Capítulo 5

Conclusão

Com a conclusão deste projecto, foi-me possível aprofundar os conhecimentos sobre FLEX e YACC assim como relembrar alguns conceitos sobre a linguagem de programação C. Penso que no final, o reusltado obtido é bastante positivo, especialmente tendo em conta que realizei o trabalho todo sozinho, tendo particular dificuldade em ir às aulas devido ao trabalho. Na minha opinião, a maior dificuldade foi conseguir perceber exatamente aquilo que estava a ser pedido, quer em termos da gramática, uma vez que esta pode realmente ser muito extensa, quer em termos do resultado final que deveria ser apresentado em XML/HTML.
Como desenvolvimento futuro, sem qualquer dúvida que o tratamento de mais TAG's tornaria este conversor de GEDCOM para XML/HTML muito mais avançado, permitindo ler uma variedade de linhas muito mais alargada. Outra opção seria ainda estilizar o XML/HTML enviado para os ficheiros, uma vez que as páginas geradas são bastante simples.

Apêndice A

Código do Programa

Lista-se de seguida todo o código dos 3 diferentes ficheiros, gedcom.y, gedcom.l e Conversor.c
gedcom.y

```
%{
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Conversor.c"

extern int yylex();
int yyerror();
int count = 1;

T_PESSOA * pessoas = NULL;
int num_pessoas=0;
int cap_pessoas=1000;
T_FAMILIA * familias = NULL;
int num_familias=0;
int cap_familias=1000;
char state = ' ';
int n_childs = 0;
%}

%union{ char * s; int n; }

%token CHIL FAM FAMC HUSB INDI NAME SEX WIFE
%token CARRIAGE NEWLINE
%token <s> pal otherchar pointer
%token <n> num
%type <s> xRef Value Line_Item Anychar
%type <n> Level Tag
```

```
%%

Gedcom : Linhas           { // Ultima pessoa/familia
                            if (state==’p’){
                                num_pessoas++;
                            }else if (state==’f’){
                                num_familias++;
                            }
                            criaHTML(pessoas,num_pessoas,familias,num_familias);
                            }


Linhas : Linhas Line
       | Line


Line   : Level xRef Tag Value Terminator
 {
 if(pessoas == NULL && familias == NULL){
     pessoas = (T_PESSOA*) malloc(sizeof(T_PESSOA)*1000);
     familias = (T_FAMILIA*) malloc(sizeof(T_FAMILIA)*1000);
 }
 if($1==0){
   if(state==’p’){
      num_pessoas++;
   }else if (state==’f’){
      num_familias++;
   }
   state=’ ’;
   n_childs = 0;

   switch($3){
        case 1:
            state=’f’;
            if (num_familias==cap_familias){
                cap_familias+=1000;
                familias=(T_FAMILIA*)realloc(familias,cap_familias*sizeof(T_FAMILIA));
            }
            strncpy(familias[num_familias].id, $2 + 1, strlen($2)-2);
            familias[num_familias].id_husb[0]=’\0’;
            familias[num_familias].id_wife[0]=’\0’;
            break;

        case 2:
```

```
            state='p';
            if (num_pessoas==cap_pessoas){
                cap_pessoas+=1000;
                pessoas=(T_PESSOA*)realloc(pessoas,cap_pessoas*sizeof(T_PESSOA));
            }
            strncpy(pessoas[num_pessoas].id, $2 + 1, strlen($2)-2);
            pessoas[num_pessoas].name[0]='\0';
            pessoas[num_pessoas].id_pai[0]='\0';
            pessoas[num_pessoas].id_mae[0]='\0';
            break;
            }
            }else if (state=='p'){
                    switch($3){
                        case 3:
                            strncpy(pessoas[num_pessoas].id_pai, $4 + 1, strlen($4)-2);
                            break;
                        case 4:
                            strncpy(pessoas[num_pessoas].name, $4, strlen($4));
                            break;
                        case 7:
                            strncpy(pessoas[num_pessoas].sex, $4, strlen($4));
                            break;

                    }
            }else if (state=='f'){
                switch($3){
                    case 5:
                        strncpy(familias[num_familias].id_husb, $4 + 1, strlen($4)-2);
                        break;
                    case 6:
                        strncpy(familias[num_familias].id_wife, $4 + 1, strlen($4)-2);
                        break;
                    case 8:
                        strncpy(familias[num_familias].id_childs[n_childs],
                                $4 + 1, strlen($4)-2);
                        familias[num_familias].num_childs++;
                        n_childs++;
                           break;
                }

            }
}
        ;
Level : num                     {$$ = $1;}
      ;
```

```
xRef : pointer            {$$ = strdup($1);}
     |                    {$$ = '\0';}
     ;


Tag : FAM         {$$ = 1;}
    | INDI        {$$ = 2;}
    | FAMC        {$$ = 3;}
    | NAME        {$$ = 4;}
    | HUSB        {$$ = 5;}
    | WIFE        {$$ = 6;}
    | SEX         {$$ = 7;}
    | CHIL        {$$ = 8;}
    ;


Value : pointer       {$$ = strdup($1);}
      | Line_Item     {asprintf(&$$, "%s", $1);}
      |               {$$ = '\0';}
      ;


Line_Item : Anychar               {asprintf(&$$, "%s", $1);}
          | Line_Item Anychar     {asprintf(&$$, "%s %s", $1, $2);}
          ;


Anychar : num            {asprintf(&$$, "%d", $1);}
        | pal            {$$ = strdup($1);}
        | otherchar      {$$ = strdup($1);}
        ;


Terminator : Carriage_Return
           | Newline
           | Carriage_Return Newline
           ;


Carriage_Return : CARRIAGE
                ;
```

```
Newline : NEWLINE
        ;


%%

int yyerror(char *s)
{
  fprintf(stderr, "ERRO: %s \n", s);
}

int main(){

    yyparse();

    return(0);
}
```

gedcom.l

```
    %{

#include <stdlib.h>
#include "y.tab.h"




%}


        /* Abreviaturas de ER */

otherchar   [\x21-\x22\x24-\x2F\x3A-\x3F\x5B-\x5E\x60\x7B\-\x7E\x80-\xFE]

%%

[ \t]           ;

\r              { return CARRIAGE; }
\n              { return NEWLINE; }

CHIL            { return CHIL;}
FAM             { return FAM;}
```

```
FAMC            { return FAMC;}
HUSB { return HUSB;}
INDI            { return INDI;}
NAME            { return NAME;}
SEX             { return SEX;}
WIFE            { return WIFE;}


[@][0-9a-zA-Z]([0-9a-zA-Z# ]|{otherchar})+[@]
{ yylval.s = strdup(yytext); return pointer; }

{otherchar}+     { yylval.s = strdup(yytext); return otherchar; }
[a-zA-Z]+       { yylval.s = strdup(yytext); return pal; }
[0-9]+         { yylval.n = atoi(yytext); return num; }


.               ;

%%
```

Conversor.c

```
    typedef struct {
    char id[10];
    char name[40];
    char id_pai[10];
    char id_mae[10];
    char sex[2];
} T_PESSOA;

typedef struct {
    char id[10];
    char id_husb[10];
    char id_wife[10];
    int num_childs;
    char id_childs[15][100];
} T_FAMILIA;


void criaHTML(T_PESSOA * pessoas,int num_pessoas,
T_FAMILIA * familias, int num_familias){

    FILE * fp;
```

```
    fp = fopen("Mainpage.html","w");

    for(int p =0; p<num_pessoas;p++){
        // Encontrar a familia
        for(int f=0;f<num_familias;f++){
            if (strcmp(familias[f].id,pessoas[p].id_pai)==0){
                strcpy(pessoas[p].id_pai,familias[f].id_husb);
                strcpy(pessoas[p].id_mae,familias[f].id_wife);
                break;
            }
        }
    }

    // Construção da Mainpage.html

    fprintf(fp,"<html>\n<body>\n");
    fprintf(fp,"<div> Esta página tem como objetivo mostrar
            as Pessoas e Familias obtidas de um ficheiro no formato GEDCOM,
            apresentando para cada pessoa, as relações pai/mãe e filho obtidas
            da análise de cada Familia. Para mais informações sobre este
            formato de formato Genealógico, visite <a href=\"http://homepages.
            rootsweb.com/~pmcbride/gedcom/55gcch1.htm\">GEDCOM</a>");

    fprintf(fp,"<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n<genoa>\n");
    fprintf(fp,"<p><h2>Lista de Pessoas: </h2>");
    for(int p =0; p<num_pessoas;p++){
        fprintf(fp,"<p>Pessoa: <a href=\"Pessoa%s.html\">%s</a></p>",
                pessoas[p].id, pessoas[p].id);
    }
    fprintf(fp,"<p><h2>Lista de Familias: </h2>");
    for(int f =0; f<num_familias;f++){
        fprintf(fp,"<p>Familia: <a href=\"Familia%s.html\">%s</a></p>",
                familias[f].id, familias[f].id);
    }
    fprintf(fp,"</genoa>\n</body>\n</html>");


// Construção das páginas HTML referentes aos indices
// Páginas das Pessoas

    for(int p = 0; p<num_pessoas;p++){
        FILE * file;
        char buf[22];
        sprintf(buf, "Pessoa%s.html", pessoas[p].id);
        file = fopen(buf,"w");
```

```
        fprintf(file,"<html>\n<body>\n<textarea rows=\"50\" cols=\"150\"
                style=\"border:none;\">\n\n");
        fprintf(file,"<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n\n<genoa>\n");
        fprintf(file,"\t<pessoa>\n");
        fprintf(file,"\t\t<id>%s</id>\n", pessoas[p].id);
        fprintf(file,"\t\t<nome>%s</nome>\n", pessoas[p].name);
        if (strlen(pessoas[p].id_pai)==0)
            fprintf(file,"\t\t<pai/>\n");
        else
            fprintf(file,"\t\t<pai>%s</pai>\n", pessoas[p].id_pai);
        if (strlen(pessoas[p].id_mae)==0)
            fprintf(file,"\t\t<mae/>\n");
        else
            fprintf(file,"\t\t<mae>%s</mae>\n", pessoas[p].id_mae);

        if (strlen(pessoas[p].sex)==0)
            fprintf(file,"\t\t<sexo/>\n");
        else
            fprintf(file,"\t\t<sexo>%s</sexo>\n", pessoas[p].sex);
        fprintf(file,"\t</pessoa>\n</genoa>\n</textarea>\n</body>\n</html>\n");
    }

//Páginas das Familias

    for(int f = 0; f<num_familias;f++){
        FILE * file;
        char buf[22];
        sprintf(buf, "Familia%s.html", familias[f].id);
        file = fopen(buf,"w");

        fprintf(file,"<html>\n<body>\n<textarea rows=\"50\" cols=\"150\"
               style=\"border:none;\">\n\n");
        fprintf(file,"<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n\n<genoa>\n");
        fprintf(file,"\t<familia>\n");
        fprintf(file,"\t\t<id>%s</id>\n", familias[f].id);
        if (strlen(familias[f].id_husb)==0)
            fprintf(file,"\t\t<pai/>\n");
        else
            fprintf(file,"\t\t<pai>%s</pai>\n", familias[f].id_husb);
        if (strlen(familias[f].id_wife)==0)
           fprintf(file,"\t\t<mae/>\n");
        else
           fprintf(file,"\t\t<mae>%s</mae>\n", familias[f].id_wife);
        if(familias[f].num_childs==0){
```

```
            fprintf(file,"\t\t<filho/>\n");
        }else{
            for(int i = 0; i<familias[f].num_childs; i++){
            fprintf(file,"\t\t<filho>%s</filho>\n", familias[f].id_childs[i]);
            }
        }
        fprintf(file,"\t</familia>\n</genoa>\n</textarea>\n</body>\n</html>\n");
    }
}
```