Trabalho Prático 2
Relatório de Desenvolvimento
XML Workbench

Miguel Pereira Fernandes (44024)

5 de Junho de 2014

Conteúdo

Capítulo 1

Introdução

1.1 Enunciado

Neste projecto, pretende-se desenvolver uma plataforma para manipulação de documentos XML.
Esta plataforma terá dois níveis: num primeiro nível é preciso reconhecer um documento XML e construir uma sua representação em memória; num segundo nível pretende-se generalizar permitindo o carregamento de vários documentos para memória sobre os quais se poderão fazer várias operações: selecção de partes, geração de novos documentos a partir dos que estão carregados, estatísticas, ...
Podemos dividir este enunciado em 3 partes que se descrevem nas secções seguintes.

1.1.1 Reconhecedor de Documentos Estruturados

Como já foi referido, nesta fase os alunos deverão desenvolver um parser que valide um documento XML e crie em memória uma representação do mesmo.
A título apenas de exemplo apresenta-se uma possível gramática para um documento XML:

```
Documento --> ElemList '$'

ElemList --> ElemList Elem
           | Elem

Elem --> char
           | '&' id ';'
           | '<' id AttrList '>' ElemList '<' '/' id '>'
           | '<' id AttrList '/' '>'

AttrList --> Attr AttrList
           |&

Attr --> id '=' valor
```

No reconhecimento do documento, o parser desenvolvido deverá verificar os seguintes invariantes:

- todas as anotações correspondentes a elementos com conteúdo são abertas e fechadas correctamente (não há marcas cruzadas e nada fica por fechar ou nada é fechado sem ter sido aberto antes);
- o documento tem que obrigatoriamente começar com a abertura dum elemento (que irá englobar todo o documento).

1.1.2 Interpretador de Comandos

O parser desenvolvido no ponto anterior será uma peça de algo bem maior: o tal "XML Work- bench".
Pretende-se agora criar um ambiente de trabalho que aceite os seguintes comandos:

LOAD <path para o documento>id - Este comando irá usar o parser desenvolvido no ponto anterior para reconhecer e carregar um documento XML. No fim, deverá ainda criar uma entrada numa estrutura de dados interna em que o identificador id fica associado ao documento reconhecido;

LIST - Mostra no écran a lista de documentos carregados e respectivos ids;

SHOW id - Mostra no écran o documento associado ao identificador id em formato ESIS (ou noutro formato semelhante definido por si);

EXIT - Sai do programa;

HELP - Imprime no écran um texto parecido com esta lista de comandos.

Pode usar a imaginação à vontade para acrescentar comandos a esta lista.
Considere ainda a seguinte gramática proposta para este interpretador (pode alterá-la à vontade):

```
Interp --> ComList

ComList --> Comando
          | ComList Comando

Comando --> LOAD fich-id id
          | SHOW id
          | LIST
          | EXIT
          | HELP
```

1.1.3 Document Query Language

Neste ponto, todos grupos de trabalho deverão estar munidos dum interpretador de comandos que permite carregar documentos, visualizá-los, fornecendo assim um primeiro conjunto de facilidades básicas num sistema documental.
Nesta fase, vamos adicionar um novo comando à lista dos já existentes:

```
QLE: [selector de documentos] [query-exp]

[selector de documentos] --> * "todos os docs carregados"
                           | id "apenas o doc com ident=id"
                           | id1,id2,...,idn

[query-exp] --> "definida mais à frente"
```

contexto histórico das linguagens de query sobre ficheiros xml

A linguagem XSLT fornece um método bastante simples para descrever a classe de nodos que se quer seleccionar. É declarativa em lugar de procedimental. Apenas é preciso especificar o tipo dos nodos a procurar usando um tipo de padrões simples baseado na notação de directorias dum sistema de ficheiros (a sua estrutura é equivalente à de uma árvore documental). Por exemplo, livro/autor, significa: seleccionar todos os elementos do tipo autor contidos em elementos livro.
A XQL é uma extensão do XSLT. Adiciona operadores para a especificação de filtros, operações lógicas sobre conteúdo, indexação em conjuntos de elementos, e restrições sobre o conteúdo dos elementos. Basicamente, é uma notação para a especificação de operações de extracção de informação de documentos estruturados.

1.2 Objetivos e Organização

Este trabalho prático tem como principais objectivos:

- aumentar a experiência de uso do ambiente linux, da linguagem imperativa C (para codificação das estruturas de dados e respectivos algoritmos de manipulação), e de algumas ferramentas de apoio à programação;
- rever e aumentar a capacidade de escrever gramáticas independentes de contexto que satisfaçam a condição LR();
- desenvolver processadores de linguagens segundo o método da tradução dirigida pela sintaxe, suportado numa gramática tradutora;
- utilizar geradores de compiladores como o par lex/yacc

Para o efeito, esta folha contém vários enunciados, dos quais deverá resolver um.
O programa desenvolvido será apresentado a um dos membros da equipa docente, totalmente pronto e a funcionar (acompanhado do respectivo relatório de desenvolvimento) e será defendido por todos os elementos do grupo (3 alunos), em data a marcar.

O relatório a elaborar, deve ser claro e, além do respectivo enunciado, da descrição do problema, das decisões que lideraram o desenho da linguagem e a concepção da gramática, do esquema de tradução e respectivas acções semânticas (incluir as especificações lex e yacc), deverá conter exemplos de utilização (textos fontes diversos e respectivo resultado produzido). Como é de tradição, o relatório será escrito em LATEX.
O pacote de software desenvolvido (um ficheiro compactado, ".tgz", contendo os ficheiros ".l", ".y", algum ".c"ou ".h"que precise, os ficheiros de teste ".txt", o relato?rio ".tex"e a respectiva "makefile") deve ser entregue através do sistema de submissão de TPs.

Capítulo 2

Implementação

2.1 Decisões

Após análise do problema decidiu-se que este é composto por dois subproblemas:

- analizador léxico e sintático de ficheiros Xml
- analizador léxico e sintático de comandos inseridos pelo utilizador

Posto isto foram criados dois ficheiros ".y" e dois ficheiros ".l", um de cada tipo por subproblema em questão. Isto levantou duas questões que tiveram que ser resolvidas:

- conflitos de nomes de funções e variáveis - para ultrapassar este problema foram adicionados parâmetros à compilação do analizador léxico e do sintático para que o "yy" colocado por defeito nas funções e variáveis fosse substituído por "xml" no casa do analizador de ficheiros xml. Abaixo pode-se ver as duas instruções de compilação do analizador de ficheiros xml presentes na makefile:
  - yacc -d -b xml -p xml xml.y
  - flex -P xml xml.l
- invocação de um parser a partir do outro - ao invés de se definir uma função main no ficheiro "xml.y" correspondente ao analizador sintático de xml, adicionou-se a função "parseXmlFile" que recebe como parâmetro uma string referente ao path relativo ou absoluto do ficheiro a processar, abre esse mesmo ficheiro e o coloca no xmlin (yyin) do analizador de ficheiros xml. Essa função é depois invocada quando necessário pelo analizador sintático aquando da recepção de um comando LOAD. O resultado do parsing do ficheiro xml é depois colocado numa variável global definida no ficheiro "global.h" que é acessível a partir dos dois parsers e pode por isso ser lida pelo analizador de comnados.

Tal como descrito no enunciado, considerou-se que os ficheiros xml têm uma tag *root* que encapsula todas as outras. Considerou-se por isso que os ficheiros xml a processar não têm o típico header <xml.../>.
Foram ainda realizadas outras considerações relativamente a ficheiros xml que apesar de poderem não corresponder exatamente à realidade, são meras simplificações e poderão ser facilmente adaptáveis à realidade.

- identificador de um elemento - considerou-se que o identificador de um elemento de um ficheiro xml segue a seguinte nomenclatura: uma primeira letra minúscula ou maiúscula ou um primeiro número, seguidos de 0 ou mais letras e números ou "-" e "_".
- identificador de um atributo - considerou-se que o identificador de um atributo de um ficheiro xml segue a seguinte nomenclatura: uma primeira letra minúscula ou maiúscula seguida de 0 ou mais letras e números ou "-" e "_".

2.2 Estruturas de dados e funções chave

Em seguida serão explicitados os extratos do código considerados mais relevantes e no cerne do funcionamento da aplicação. Para consultar ou realizar download do código fonte poderão consultar o seguinte repositório no github `https://github.com/migpfernandes/pl_tp2` .

2.2.1 Listas Genéricas

Uma vez que o parsing de ficheiros via Yacc está bastante inerente a uma estrutura de dados do tipo lista ligada, podendo esta ser de vários tipos, dependendo do que se está a processar optou-se por criar uma lista genérica assim como funções genéricas de manuseamento de listas. A estrutura que define um nó da lista é a seguinte:

```
typedef struct node_s {
  void *data;
  struct node_s *next;
} NODE;
```

As funções abaixo são as que habitualmente encontramos na interação com listas: inserção, remoção e pesquisa.

```
NODE *list_create(void *data);
NODE *list_insert_after(NODE *node, void *data);
NODE *list_insert_beginning(NODE *list, void *data);
NODE *list_insert_sorted(NODE *list, void *data,int(*func)(void*,void*))
    ;
int list_remove(NODE *list, NODE *node);
int list_foreach(NODE *node, int(*func)(void*));
NODE *list_find(NODE *node, int(*func)(void*,void*), void *data);
void list_destruct(NODE *node);
NODE* list_concat(NODE *list1,NODE* list2);
int list_length(NODE* list);
```

2.2.2 XmlData

A estrutura abaixo pretende criar uma abstração de um ficheiro Xml. Como se pode ver abaixo existem várias estruturas auxiliares como, por exemplo a estrutura sAttrList, que têm um mapeamento direto com as estruturas que se podem encontrar num ficheiro Xml, neste caso a estrutura sAttrList irá conter os atributos de um elemento. O ponto mais importante desta estrutura será talvez a "union" uNode, que contém os três tipos de nós que podem ser encontrados num ficheiro xml:

- Texto
- Elemento com conteúdo (elemento aberto e fechado por tags distintas)
- Elemento sem conteúdo (elemento fechado na própria declaração)

```
/* ---- Attributes ---- */

typedef struct sAttrList
  {
    char *name;
    char *value;
    struct sAttrList *next;
  } Attr, *AttrList;

AttrList consAttrList( char *n, char *v, AttrList l );
void showAttrList( AttrList l );
AttrList add2AttrList( AttrList l1, AttrList l2);

/* ---- Element Nodes ---- */
typedef union
  {
    TextNodePtr t;
    ElemNodePtr e;
    EmptyElemNodePtr ee;
  } uNode;

typedef struct sNode
  {
    int type;
    uNode val;
  } Node, *NodePtr;

NodePtr consNodefromText( TextNodePtr n );
NodePtr consNodefromElem( ElemNodePtr n );
NodePtr consNodefromEmptyElem( EmptyElemNodePtr n );
void    showNodeESIS( NodePtr node );
void    showNodeXML(NodePtr node);
NodePtr add2NodeList( NodePtr nl, NodePtr node );
NodePtr getNodeChild(NodePtr np);
NodePtr getNodeSibling(NodePtr nl);
int nodeTagNameIs(NodePtr nl,char* name);

struct sTextNode
  {
    char * contents;
    NodePtr sibling;
  };
TextNodePtr consTextNode( char *c, NodePtr s );

struct sElemNode
  {
```

```
    char *name;
    AttrList attrs;
    NodePtr sibling;
    NodePtr child;
  };
ElemNodePtr consElemNode( char * name, AttrList attrlist, NodePtr s,
    NodePtr c );
char* getAtributeValue(NodePtr node, char *key);
int containsAtribute(NodePtr node, char *key);

struct sEmptyElemNode
  {
    char *name;
    AttrList attrs;
    NodePtr sibling;
  };
EmptyElemNodePtr consEmptyElemNode( char * name, AttrList attrlist,
    NodePtr s);
```

2.2.3 FileInfo

Esta estrutura pretende representar a informação referente a um ficheiro Xml. Quando um ficheiro é carregado, é criada uma instância desta estrutura e colocada numa lista para posterior manipulação através de outros comandos. Basicamente o ficheiro tem três caraterísticas:

- ID
- Nome
- Conteúdo

que são as retratadas pela estrutura.

```
typedef struct sFileInfo{
    char *Id;
    char *filename;
    NodePtr ficheiroXml;
} *FileInfo, FileInfoNode;

FileInfo createFileInfo(char *Id,char *filename,NodePtr ficheiroXml);
NODE* addFile(NODE* list,FileInfo ficheiro);
void showFile(NODE* list,char* Id);
void listFiles(NODE* list);
void destructList(NODE* list);
FileInfo findFile(NODE* list,char *Id);
```

As funções de manipulação desta estrutura estão fortemente ligadas aos comando disponíveis na aplicação: LIST (listFiles), SHOW (showFile), LOAD (addFile), EXIT (destructList).

2.2.4 XmlPath

Esta estrutura pretende criar uma representação abstracta de um comando Xql. A estrutura é composta pelos seguintes campos:

name - Identificador do elemento ou do atributo

filters - Lista de filtros aplicados sobre o nó do XQL em questão. Cada elemento da lista é uma lista de elementos do tipo XmlPath

isAtrib - Identifica se o elemento em questão se refere a um atributo

slashPrefixNo - Número de barras (‘) que precedem o elemento do XQL

isDirectChild - Identifica se o elemento deve ser aplicado a partir do nó atual (corresponde ao ponto no XQL)

Esta estrutura será utilizada conjuntamente com a xmldata em funções que extraem a informação do ficheiro Xml.

```
typedef struct sXmlPath{
    char* name;
    NODE* filters;
    int isAtrib;
    int slashPrefixNo;
    int isDirectChild;
} *XmlPath,XmlPathNode;

XmlPath createXmlNode(char *name, NODE* Filters,int isAtrib,int
    slashprefixno);
NODE* addXmlPathNode(NODE* list, XmlPath node);
void setDirectChild(NODE* list,int directChild);
void printXpathExpression(NODE *list);

NODE* filterSelectedNodes(NodePtr xmlfile,NODE* xmlpath);
void printFilteredNodesForFile(NODE* files, NODE* fileList,NODE* xmlpath
    );
```

Todas estas funções são vitais ao funcionamento do comando QLE, contudo existem três funções aqui que são de particular interesse:

filterSelectedNodes - executa a filtragem do comando QLE e constroí uma lista de nós para posterior apresentação ao utilizador.

printXpathExpression - reconstroí o comando XQL a partir da estrutura em memória.

printFilteredNodesForFile - recebe uma lista de Id's de ficheiros, uma lista dos ficheiros em memória e uma estrutura XmlPath com a XQL que se pretende aplicar aos vários ficheiros e imprime no écran o resultado do processamento.

2.3 Parser de XML

2.3.1 FLEX

O analizador léxico para ficheiros XML é bastante simples uma vez que se colocou a “responsabilidade” de validar a estrutura dos ficheiros no analizador sintático. A maior dificuldade no processamento do ficheiro XML foi a identificação dos locais onde a remoção dos espaços e quebras de linha deve ser realizada, tudo o resto é bastante simples. Basicamente existem três contextos:

intag, atr e endtag que representam respectivamente o processamento dentro de uma tag de abertura de elemento ou de um elemento único, o processamento dos atributos e o procesamento da tag de fecho de elemento. As transições entre os vários contextos são realizadas através dos símbolos <e >essencialemente.

```
%{
#include "xmldata.h"
#include "xml.tab.h"
#include <string.h>
%}

%x intag atr endtag

%%

\<\/     {BEGIN endtag; return ENDTAGB;}
\<       {BEGIN intag; return '<';}
[^\<]*     {xmllval.str = strdup(xmltext); return texto;}
[ \n\t]    ;

<intag>[a-zA-Z0-9][\_a-zA-Z0-9]*
                        {
                                xmllval.str = strdup(xmltext);
                                BEGIN atr;
                                return id;
                        }

<endtag>[a-zA-Z0-9][\_a-zA-Z0-9]*
                        { xmllval.str = strdup(xmltext); return id; }

<endtag>\>[ \n\t\r]*            { BEGIN 0; return '>'; }

<atr>[a-zA-Z][a-zA-Z0-9\_\-]*
                        { xmllval.str = strdup(xmltext); return id;}

<atr>\=                     return '=';
<atr>\"[^\"]*\"         { xmllval.str = strdup(xmltext); return valor; }
<atr>\>[ \n\t\r]*       { BEGIN 0; return '>';}
<atr>\/\>[ \n\t\r]*     { BEGIN 0; return SINGLETAGE;}

%%
int xmlwrap(){
//  return 1;
}
```

2.3.2 YACC

```
%{
#include <stdio.h>
#include <stdlib.h>
#include "xmldata.h"
#include "global.h"

extern int xmllineno;
extern FILE *xmlin;
%}

%token ENDTAGB SINGLETAGE texto id valor
%start Documento

%union {
  char *str;
  AttrList alist;
  NodePtr node;
```

```
}


%type <str> ENDTAGB SINGLETAGE texto id valor
%type <alist> AttrList Attr
%type <node> Documento NodeList Node Tag

%%
Documento : Tag {$$ = $1; xmlFile = $$;};

Tag     : '<' id AttrList '>' NodeList ENDTAGB id '>' {
                    if (strcmp($2,$7)!=0) {
                     xmlerror("O ficheiro xml não tem uma estrutura
                          válida.");
                     YYABORT;
                    } else
                     $$ =consNodefromElem(consElemNode($2,$3,NULL,$5));
                    };

AttrList  : AttrList  Attr {$$ = add2AttrList($1,$2);}
      | { $$ = NULL;}
      ;

Attr    : id '=' valor {$$ = consAttrList($1,$3,NULL);};

NodeList  : NodeList Node { $$ = add2NodeList($1,$2); }
      | Node { $$ = $1; }
      ;


Node    : texto   { $$ = consNodefromText(consTextNode($1,NULL)); /*
    contents, sibling */ }
      | Tag  { $$ = $1;}
      | '<' id AttrList SINGLETAGE { consNodefromEmptyElem(
          consEmptyElemNode($2,$3,NULL)); }
      ;

%%
int xmlerror(char *s){
  fprintf(stderr,"Documento com sintaxe inválida.\nErro: %s\nLinha: %d\n
      ",s,xmllineno);
}

//int main(){
//  xmlparse();
//}

void parseXmlFile(char *path){
  FILE *fp=fopen(path,"r");
  if(!fp)
  {
      printf("O ficheiro não foi encontrado!\n");
  } else {
    xmlin=fp;
    xmlparse();

    fclose(fp);
  }
}
```

2.4 Parser de Comandos

2.4.1 FLEX

```
%{
```

```
#include "list.h"
#include "xmlpath.h"
#include "y.tab.h"
%}

%x cmdload cmdshow cmdunknown cmdqle queryexp

%%

(E|e)(X|x)(I|i)(T|t)     return EXIT;
(H|h)(E|e)(L|l)(P|p)     return HELP;
(L|l)(I|i)(S|s)(T|t)     return LIST;
(S|s)(H|h)(O|o)(W|w)     { BEGIN cmdshow; return SHOW; }
(L|l)(O|o)(A|a)(D|d)     { BEGIN cmdload; return LOAD; }
(Q|q)(L|l)(E|e)(\:\ )      {BEGIN cmdqle; return QLE; }

[a-zA-Z]+          {
                  BEGIN cmdunknown;
                  yylval.str = strdup(yytext);
                  printf("O comando '%s' não foi reconhecido.\nTente
                      escrever 'HELP' para mais informações.\n",yytext);
                  return UNKNOWN;
                }
.                 ;

<cmdload>[a-zA-Z0-9]+        {yylval.str = strdup(yytext); BEGIN 0;
    return id; }
<cmdload>[a-zA-Z0-9\-\_/\.][a-zA-Z0-9\-\_/ \.]+\.xml  {yylval.str =
    strdup(yytext); return fichId; }
<cmdload>\"[^\"]+\"          {yylval.str = strdup(yytext+1); yylval.str[
    strlen(yylval.str)-1] = '\0'; return fichId; }

<cmdshow>[a-zA-Z0-9]+      {yylval.str = strdup(yytext); BEGIN 0; return
    id;}

<cmdqle>,             {return ','; }
<cmdqle>\*              {return '*'; BEGIN queryexp; }
<cmdqle>[a-zA-Z0-9]+       {yylval.str = strdup(yytext); return id; }
<cmdqle>[ \t]           {BEGIN queryexp; }
<cmdqle>.             ;

<queryexp>\/\/       return DOUBLESLASH;
<queryexp>\[     return '[';
<queryexp>\]     return ']';
<queryexp>\/     return SLASH;
<queryexp>\*     return '*';
<queryexp>\.     return PERIOD;

<queryexp>[a-zA-Z0-9][\_a-zA-Z0-9]*                { yylval.str = strdup(
    yytext); return tagname; }
<queryexp>\@[a-zA-Z][a-zA-Z0-9\-\_]*       { yylval.str = strdup(yytext
    +1); return atribname; }

<queryexp>\n                          { BEGIN 0; return END; }
<queryexp>[ \t]             ;

<cmdunknown>.           ;
<cmdunknown>\n            { BEGIN 0; }

<*>[ \n\t]            ;

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

2.4.2 YACC

```
%{
#include <stdio.h>
#include "FileInfo.h"
#include "global.h"
#include "list.h"
#include "xmlpath.h"

#define RESET   "\033[0m"
#define BOLDRED     "\033[1m\033[31m"      /* Bold Red */

NODE* list=NULL;
extern int parseXmlFile(char* path);

NodePtr xmlFile=NULL;

void showHelpMessage();
void showPrompt();
int printStringList(void *s);
%}

%token LOAD SHOW LIST EXIT HELP QLE UNKNOWN END SLASH DOUBLESLASH ERROR
    PERIOD
%token fichId id tagname atribname
%start Interp

%union{
  char *str;
  NODE* no;
  XmlPath pathxml;
  int num;
}

%type<str> fichId id tagname LOAD SHOW LIST EXIT HELP QLE UNKNOWN
    Comando ComList Interp atribname END
%type<no>  Idlist DocSelector QueryExp QueryExp2 TagList FilterList
    Filter
%type<num> Context Context2
%type<pathxml> Tag

%%
Interp     : ComList;

ComList    : Comando { showPrompt(); }
        | ComList Comando { showPrompt();}
      ;

Comando    : LOAD fichId id {xmlFile = NULL; parseXmlFile($2);
                 if (xmlFile) {
                   FileInfo info = createFileInfo($3,$2,xmlFile);
                   list = addFile(list, info);
                   printf("Ficheiro adicionado com sucesso!\n");
                 }
                }
           | SHOW id { showFile(list,$2); }
           | LIST    { listFiles(list); }
           | EXIT    { printf("Programa terminado!\n"); YYACCEPT; }
           | HELP    { showHelpMessage(); }
           | QLE DocSelector QueryExp END { printf("DOCS:\n");
               list_foreach($2,printStringList); printf("TAGS:\n");
               printXpathExpression($3);
                     printFilteredNodesForFile(list,$2,$3); }
      | UNKNOWN
      ;

DocSelector : '*' { $$ = NULL; }
      | Idlist { $$ = $1; }
      ;
```

```
Idlist     : Idlist ',' id { $$ = list_insert_beginning($$,$3); }
      | id { $$ = list_insert_beginning(NULL,$1); }
      ;

QueryExp   : PERIOD Context2 QueryExp2  { setDirectChild($3,1); ((XmlPath
    ) $3->data)->slashPrefixNo = $2; $$ = $3; }
           | QueryExp2 { $$ = $1; }
           ;

QueryExp2 : TagList Context2 atribname  { XmlPath atribNode =
    createXmlNode($3,NULL,1,$2);
                                          $$ = addXmlPathNode($1,
                                              atribNode); }
           | Context atribname        { XmlPath atribNode =
              createXmlNode($2,NULL,1,$1);
                                          $$ = addXmlPathNode(NULL,
                                              atribNode); }
           | TagList              { $$ = $1; }
           ;

Context    : Context2     { $$ = $1; }
           |          { $$ = 0; }
           ;

Context2   : SLASH      { $$ = 1; }
           | DOUBLESLASH    { $$ = 2; }
           ;

TagList    : TagList Context2 Tag  { $3->slashPrefixNo = $2; $1 =
    addXmlPathNode($1,$3); $$ = $1;  }
           | Context Tag      { $2->slashPrefixNo = $1; $$ =
              addXmlPathNode(NULL,$2); }
           ;

Tag      : tagname FilterList  { $$ = createXmlNode($1, $2,0,0); }
           | '*' FilterList       { $$ = createXmlNode("*", $2,0,0); }
           ;

FilterList  : FilterList Filter {
                                    NODE* atribs = $1;
                                    if (!atribs) atribs = list_create($2
                                        );
                                    else atribs = list_insert_after(
                                        atribs,$2);
                                    $$ = atribs;
                                }
           |           { $$ = NULL; }
           ;

Filter     : '[' QueryExp ']' {$$ = $2; }

%%
void showPrompt(){
  printf("\n>");
}

int printStringList(void* s){
  if(s) printf("Item: %s\n",(char*) s);
  return 0;
}

int yyerror(char *s){
   fprintf(stderr,"%s",s);
}

int main(){
  showAppLogo();
  yyparse();
```

```
}
```

Capítulo 3

Casos de estudo

3.1 Ficheiro XML a processar (simple.xml)

```
<breakfast_menu>
  <food>
    <name>Belgian Waffles</name>
    <price>$5.95</price>
    <description>Two of our famous Belgian Waffles with plenty of real
        maple syrup</description>
    <calories>650</calories>
  </food>
  <food>
    <name>Strawberry Belgian Waffles</name>
    <price>$7.95</price>
    <description>Light Belgian waffles covered with strawberries and
        whipped cream</description>
    <calories>900</calories>
  </food>
  <food>
    <name>Berry-Berry Belgian Waffles</name>
    <price>$8.95</price>
    <description>Light Belgian waffles covered with an assortment of
        fresh berries and whipped cream</description>
    <calories>900</calories>
  </food>
  <food>
    <name>French Toast</name>
    <price>$4.50</price>
    <description>Thick slices made from our homemade sourdough bread</
        description>
    <calories>600</calories>
  </food>
  <food>
    <name>Homestyle Breakfast</name>
    <price>$6.95</price>
    <description>Two eggs, bacon or sausage, toast, and our ever-popular
         hash browns</description>
    <calories>950</calories>
  </food>
</breakfast_menu>
```

3.2 Processamento do ficheiro

Para executar o programa executa-se a seguinte instrução na bash:

```
./interp
```

Em seguinda, tendo em vista o carregamento do ficheiro de exemplo na base de dados, executam-se as seguintes instruções:

```
$ ./interp
.-..-..-..-..-.      .-.    .-.            .-.    .-.
     .-.
: `' :: `' :: :      : :.-.: :            : :.-.: :                        :
     :
`    ' : .. :: :      : :: :: : .--. .--. : `'.': `-.  .--. ,-.,-. .--. :
    `-.
.'   `.: :; :: :__    : `' `' ;' .; :: ..': . `.' .; :' '_.': ,. :'  ..':
    .. :
:_;:_;:_;:_;:___.'    `.,`.,' `.__.':_;  :_;:_;`.__.'`.__.':_;:_;`.__.':_
    ;:_;
```

```
>list
Não existem ficheiros para listar.

>load xml_examples/simple.xml aa
Ficheiro adicionado com sucesso!

>list
Ficheiros:
ID     NAME
------------------------------
aa     xml_examples/simple.xml
```

3.3 Resultado do processamento

Para finalizar a conversão do ficheiro XML para ESIS executam-se os seguintes passos:

```
>show aa
Ficheiro ID:"aa" NAME:"xml_examples/simple.xml"
(breakfast_menu
(food
(name
-Belgian Waffles
)name
(price
-$5.95
)price
(description
-Two of our famous Belgian Waffles with plenty of real maple syrup
)description
(calories
-650
)calories
)food
(food
(name
-Strawberry Belgian Waffles
)name
(price
-$7.95
)price
(description
```

```
-Light Belgian waffles covered with strawberries and whipped cream
)description
(calories
-900
)calories
)food
(food
(name
-Berry-Berry Belgian Waffles
)name
(price
-$8.95
)price
(description
-Light Belgian waffles covered with an assortment of fresh berries and
    whipped cream
)description
(calories
-900
)calories
)food
(food
(name
-French Toast
)name
(price
-$4.50
)price
(description
-Thick slices made from our homemade sourdough bread
)description
(calories
-600
)calories
)food
(food
(name
-Homestyle Breakfast
)name
(price
-$6.95
)price
(description
-Two eggs, bacon or sausage, toast, and our ever-popular hash browns
)description
(calories
-950
)calories
)food
)breakfast_menu

>exit
Programa terminado!
```

Capítulo 4

Conclusões

Após a realização deste trabalho foi possível comprovar as inúmeras funcionalidades que um parser desenvolvido através do YACC conjuntamente com o FLEX tem. Sem "grande" esforço, produziu-se um parser capaz de pegar em qualquer ficheiros Xml e colocá-lo numa estrutura em memória, sujeitando o ficheiro XML a uma validação estrutural. A partir deste ponto as possibilidades são inúmeras. Neste trabalho converteu-se o ficheiro Xml num ficheiro ESIS, mas poderia ter-se facilmente convertido para Json ou HTML, ou até para scripts de geração de base de dados com os dados do ficheiro XML.
Por outro lado, a realização deste trabalho deu também para verificar que utilizando estas ferramentas de geração de aplicações, se consegue construir aplicações com alguma complexidade com relativa facilidade.

Capítulo 5

Trabalho Futuro

Devido à complexidade deste projeto, houve algumas tarefas que ficaram por executar:

- a aplicação de XQL não foi completamente finalizada, apesar de já conseguir realizar o parse de expressões complexas, dos objectivos propostos apenas não consegue processar operadores, não os traduz depois na filtragem dos documentos. Neste momento, a aplicação é apenas capaz de processar expressões simples do tipo "//teste" ou "teste/teste2".
- Tolerânica a erros. No processador de comandos, existem situações em que uma sintaxe inválida provoca o fim abrupto da aplicação. Isto não deveria acontecer, deveria ser devolvida uma mensagem informativa e a aplicação deveria voltar ao estado antes da execução desse comando.
- Seria interessante permitir voltar a guardar os ficheiros carregados em memória ou os resultados das seleções no filesystem.