Processamento de Linguagens (3º ano de Curso)

Trabalho Prático 3

Relatório de Desenvolvimento

José Pereira
(a82880@alunos.uminho.pt)

Ricardo Petronilho
(a81744@alunos.uminho.pt)

10 de Junho de 2019

Resumo

O projeto elaborado na Unidade Curricular de Processamento de Linguagens do Mestrado integrado em Engenharia Informática da Universidade do Minho, tem como principal objetivo o processamento de informação contida em ficheiros do tipo Biblio Thesaurus, utilizando para isso a ferramenta *flex + yacc*.

Conteúdo

1 Introdução **2**

2 Concepção/desenho da Resolução **3**

2.1 Gramática 3

2.2 Desenvolvimento do reconhecedor léxico - Flex 5

2.3 Desenvolvimento do reconhecedor léxico - Yacc 5

2.3.1 Implementação inicial 5

2.3.2 Guardar a informação em memória 6

2.3.3 Gerar uma página HTML 6

3 Utilização, Codificação e Testes **8**

4 Conclusão **9**

A Código do Programa Flex **10**

B Código do Programa Yacc **11**

Capítulo 1

Introdução

Na unidade curricular de Processamento de Linguagens, do Mestrado integrado em Engenharia Informática da Universidade do Minho, foi proposta a elaboração do exerício três do enunciado fornecido pela equipa docente.

O objetivo central é filtrar a informação mais importante de um ficheiro em formato *thesaurus ISO 2788 (T2788)*, utilizando as ferramentas flex + yacc.

Deste modo, a informação contida nos ficheiros referidos anteriormente, começa com metadados e um conjunto de conceitos, sendo que, cada um destes, contém ainda, um representante na linguagem base, traduções noutras línguas e ligações a outros conceitos.

Deste modo, os objetivos deste projeto são a especificação da gramática da linguagem de entrada, desenvolver um reconhecedor léxico e sintático para essa linguagem utilizando as ferramentas flex + yacc, já referidas anteriormente e por fim gerar uma página *HTML* com a informação recolhida, de forma organizada.

Capítulo 2

Concepção/desenho da Resolução

2.1 Gramática

Z = .F$
F = .L LB I listaC
L = .LANG ListaL
ListaL = .FLAG ListaL
ListaL = . —
LB = .BASELANG ListaLB
ListaLB = FLAG ListaLB
ListaLB = —
I = .INV FLAG FLAG
ListaC = C ListaC
ListaC = . —
C = .listaD
listaD = .D listaD
D = FLAG TXT
D = TXT

Tabela da gramática.

De seguida apresenta-se uma lengenda para facilitar a leitura da tabela da gramática:

- Z = estado inicial
- $ = fim do ficheiro
- F = ficheiro
- L = linguagens
- LB = linguagens base
- I = relações inversas
- listaC = lista de conceitos
- LANG = token que representa a expressão regular (%language) para capturar o metadado referente às linguagens

- ListaL = lista de linguagens
- FLAG = representa as palavras usadas para identificar, sendo estas totalmente escritas com letras maiúsculas, como por exemplo *BT, NT, PT, EN, ...*
- BASELANG = token que representa a expressão regular (%baselang) para capturar o metadado referente às linguagens de base
- listaLB = lista de linguagens de base
- INV = token que representa a expressão regular (%inv) para capturar o metadado referente às relações inversas
- C = conceito
- listaD = lista de dados
- D = dado, que será simplesmente texto ou texto seguido de flags
- TXT = expressão regular representativa de uma palavra, como por exemplo animal, cat, dog, ...
- — = representa o caso vazio

A gramática especificada acima, teve como base o exemplo fornecido pela equipa docente no enunciado.

Importante referir que inicialmente a gramática não era exatamente igual à atual, pois foi necessário mudar o estado *C* o qual representa o conceito, anteriormente era definido por *TXT listaD*, onde o TXT, tal como já foi dito representa palavras, pelo que, neste caso seria o termo da linguagem base (*baselang*) do conceito, daí pensar-se inicialmente que deveria estar no estado *C*.

Devido a problemas que surgiram na implementação teve que ser alterado, estando atualmente a ser capturado no estado *D*, ou seja, nos dados do conceito.

O estado *I*, que representa as relações inversas é definido por *INV FLAG FLAG*, o que significa que apenas pode capturar duas relações. Como a equipa não tinha a certeza se era possível ocorrem mais do que duas relações inversas, fixou a captura a duas.

2.2 Desenvolvimento do reconhecedor léxico - Flex

A conceção da aplicação baseou-se na grámatica desenvolvida anteriormente, começando-se pelas expressões regulares, capturadas pela ferramenta *flex*.

Assim começou-se por identificar as mesmas, começando pelos metadados, dos quais :

- %language
- %baselang
- %inv

As expressões identificadas acima, estão associadas respetivamente a LANG, BASELANG e INV, tokens da aplicação *yacc*.

De seguida, tal como na gramática a *FLAG* corresponde às palavras identificadoras, como por exemplo: BT, NT, PT, EN, ..., ou seja, siglas, portanto a expressão regular utilizada foi [A-Z]+.

Do mesmo modo, *TXT*, identificado na gramática, representa palavras, como por exemplo: animal, cat, dog, ..., sendo a expressão regular utilizada a [A-Za-z1-9,_]+.

Importante referir uma das primeiras falhas do grupo, que não chegou a ficar inteiramente resolvida, devido à aproximação da entrega, pois tal como se pode verificar na expressão regular anterior, ela captura palavras, o que significa que não vai ser possível utilizar a aplicação desenvolvida pela equipa para ficheiros com campos como por exemplo *BT Life being*, apenas será capturado o *Life*.
Deste modo, a equipa encontrou uma forma de solucionar o problema, que é substituir os espaços pelo caracter *underscore* (_), sendo que, a equipa admite que não é a melhor solução. Do mesmo modo, segundo o exemplo fornecido pela equipa docente no enunciado, existe a possibilidade de se fazer *NT cat, dog*, onde se encontra o mesmo problema referido anteriormente, sendo que a solução neste caso foi adição da vírgula à expressão regular associada a *TXT* e remoção dos espaços, e desta forma, consegue-se capturar a frase.

Por fim, e não menos importante, também é necessário ignorar os comentários e outros caracteres, desta forma, adicionou-se a expressão regular #.* até ao fim da linha, sendo que os comentários são inicializados por #, não tendo qualquer processamento associado.

2.3 Desenvolvimento do reconhecedor léxico - Yacc

2.3.1 Implementação inicial

Analisando a gramática inicialmente definida, começou-se por identificar os tokens existentes, os quais redirecionados do programa *flex*.

De seguida, procedeu-se à criação e associação dos tipos das variáveis definidas no *union* das quais, *flags* e *texto*, sendo respetivamente FLAG e TXT, sendo que todos os outros estados estão associados à variável *texto*.

O passo seguinte, passou pela estruturação da gramática, onde para *debug*, se fez prints das capturas.

2.3.2 Guardar a informação em memória

De seguida e face aos objetivos a equipa tentou guardar a informação em memória, o que não conseguiu finalizar pois não se conseguiu descobrir o problema. No entanto, apesar de não concluido, a estrutura idealizada e implementada foi de uma árvore, onde não existe limite de *filhos*, para isso contou-se com o auxílio de uma biblioteca definida pelo alunos na Unidade Curricular de Laboratórios de Informática três, denominada por *ArrayList.c*. Ou seja, criou-se um *TAD_ARRAY_LIST* chamado de *conceitos* onde os seus elementos são do tipo definido de seguida:

```
typedef struct node{
    char* conceito; // nome do conceito
    char* traducao; // tradução do conceito
    char* scopeNote; // nota explicativa
    char* pai;
    TAD_ARRAY_LIST filhos; // filhos, relações inferiores
}*NODE;
```

Tal como se pode observar acima, um *node* corresponde à informação de um *conceito*, onde o campo *filhos* corresponde aos nós que tem relação superior com este conceito. Ou seja, que o campo *pai* dessa lista de conceitos apelidados de *filhos*, é igual ao campo *conceito* (termo da baselang) deste conceito, tendo-se assim uma estrutura organizada de forma hierárquica. Apesar de a equipa não ter conseguido finalizar este requisito, deixou-se o código até então desenvolvido comentado, para que no futuro se volte a tentar solucionar o problema.

2.3.3 Gerar uma página HTML

Tal como já foi referenciado, não se conseguiu concluir o requisito anterior, pelo que a equipa achava interessante ter utilizado a estrutura definida para esse mesmo requisito para a geração das páginas *HTML*, a qual iria facilitar o processo pois está organizada por níveis.

No entanto, a equipa após verificar que não conseguiria concluir o segundo requisito, de guardar a informação em memória, pensou numa segunda alternativa, gerar as páginas ao mesmo tempo que se faz a captura, tornando-se um processo fácil.

Deste modo, necessitou-se de guardar algumas variáveis, que por sorte já tinham sido guardadas na tentativa do segundo requisito, como o *conceito* (termo da baselang), que é muito importante, para se criar os ficheiros *.hmtl*, bem como as linguagens, linguagens base e relações inversas existentes . Assim, quando se captura uma palavra sozinha, estado *dado: TXT* sabe-se que se está perante o ínicio de um conceito, logo, cria-se o ficheiro *HTML*.

De seguida, quando se encontra um *dado: FLAG TXT*, vai se verificar o tipo de *FLAG*, existindo as seguintes possibilidades:

- uma tradução, como por exemplo *PT*, então escreve-se esse texto na página do conceito atual (ficheiro guardado como variável global).
- uma relação superior, como por exemplo *BT nomeConceito* e deste modo precisa-se de fazer um *link* na página *nomeConceito.html* do conceito superior para a página deste conceito atual.
- uma relação inferior, apenas se escreve esse texto na página deste conceito.

- uma scope note, onde se faz o mesmo que a anterior.

Capítulo 3

Utilização, Codificação e Testes

O grupo definiu uma *makefile*, que compila e corre o programa desenvolvido em flex+yacc. Também do mesmo modo, a *makefile* está preparada para instalar e desinstalar o programa.

Os testes realizados para verficação da resolução de cada problema foram por observações dos resultados no terminal, que ainda são possíveis verificar e também através de exemplos gerados pela equipa.

Capítulo 4

Conclusão

Em suma, os objetivos inicialmente propostos não foram totalmente cumpridos como foi referido ao longo do relatório, dos quais, capturar frases com palavras separadas por espaços e guardar a informação em memória, os restantes, como fazer a gramática, capturar a informação e gerar páginas *HTML* organizadas, foram conluídos.

A análise do ficheiro para encontrar os padrões necessários, foi importante para se conseguir capturar a informação necessária.

Em relação ao requisito de capturar frases, a solução, apesar de não ser a melhor, encontrada foi a substituição dos espaços por *underscore* e no caso de listagem de relações inferiores separadas por vírgula, remover os mesmos. Sendo que, obviamente a equipa sabe que não é a melhor solução, mas a mais funcional até ao momento, no entanto, pertende-se resolver este mesmo problema no futuro.

Importante referir também, que apesar de não se ter concluido o requisito dois, de guardar a informação em memória, se definiu a estrutura que a equipa achou mais adequada para guardar a mesma de forma hierárquica e organizada, e também se deixou comentado o código definido que apesar de não completamente funcional, se pertende melhorar em medida de trabalho futuro, para se conseguir atingir esse mesmo objetivo.

Por fim, apesar de alguns objetivos, não terem sido concluidos com sucesso, o grupo acha que os conseguidos, tem um bom resultado final.

Apêndice A

Código do Programa Flex

Lista-se a seguir o CÓDIGO feito em flex:

```
%{
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

%}

%option noyywrap

%%

#.*\n  {}

\%language\  {return LANG;}
\%baselang\  {return BASELANG;}
\%inv\  {return INV;}


[A-Z]+ {yylval.flag = strdup(yytext); return FLAG;}

[A-Za-z\,1-9\_\,]+ {yylval.texto = strdup(yytext);return TXT;}

[\ \t\n] {}

.  {}

%%
```

Apêndice B

Código do Programa Yacc

Lista-se a seguir o CÓDIGO feito em yacc, pelo que a equipa recomenda consultar o ficheiro original, pois o mesmo é de grande dimensões para ser apresentado no relatório:

```
%{
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "ArrayList.h"
int yylex();
void yyerror(char *s);

typedef struct node{
char* conceito; // nome do conceito
char* traducao; // tradução do conceito
char* scopeNote; // nota explicativa
char* pai;
TAD_ARRAY_LIST filhos; // filhos, relações inferiores
}*NODE;

char* tmpTraducoes[10];
int indexTraducoes = 0;
char* tmpPai;
char* tmpFilho;

TAD_ARRAY_LIST conceitos;

char* conceito;
char* traducao;
char* scopeNote;
char* pai;
char* filho;

FILE* file;
char* ficheiro_atual;
```

```
void print(TAD_ARRAY_LIST lista){
if(getArraySize(lista) != 0){
for(int i = 0; i < getArraySize(lista); i++){

NODE node = getElem(lista, i);
if(node != NULL){
printf("conceito:%s\n", node->conceito);
printf("tradução:%s\n", node->traducao);

if(node->filhos != NULL){
printf("------------------ filhos ------------------\n");
print(node->filhos);
printf("--------------------------------------------\n");
}
}
}
}
}

// procura pelo pai e adiciona ao filho do mesmo
int adicionarNode(TAD_ARRAY_LIST lista, NODE node){

if(lista == NULL || node == NULL){
perror("ERRO: a lista está a NULL!");
exit(-1);
}

for(int i = 0; i < getArraySize(lista) ; i++){

NODE nodeTmp = (NODE) getElem(lista, i);

if(nodeTmp == NULL){
perror("ERRO: nodeTmp está a NULL!");
exit(-1);
}

if(strcmp(nodeTmp->conceito, node->pai) == 0){
addElem(nodeTmp->filhos, node);
return 1;
}
else{
adicionarNode(nodeTmp->filhos, node);
}
}

return 0;
}
```

```
void beginHTML(FILE* file, char* title){
        fprintf(file, "<html>\n\t<head>\n\t\t<meta charset='UTF-8'/>\n\t</head>\n<body>\n<h1>%s</
    }

void endHTML(FILE* file){
     fprintf(file, "</ul></body></html>");
}
%}

%token LANG BASELANG INV TXT FLAG

%union{
char* texto;
char* flag;
}

%type<flag> FLAG
%type<texto> TXT linguagens listaLinguagens linguagensBase listaLinguagensBase inversas conceitos

%%

ficheiro:  linguagens linguagensBase inversas conceitos     {
printf("pai=%s | filho=%s | traducoes[0] = %s | traducoes[1] = %s\n", tmpPai, tmpFilho, tmpTraduc
printf("linguagens:%s\nlinguagensBase:%s\ninversas:%s\nconceitos:\n%s\n", $1, $2, $3, $4);
}
;

linguagens:  LANG listaLinguagens   {asprintf(&$$, "%s", $2);}
    ;

listaLinguagens: FLAG listaLinguagens                                    {
tmpTraducoes[indexTraducoes] = strdup($1); // guardar as traduções existentes
indexTraducoes++;
asprintf(&$$, "%s %s", $1, $2);
}
| {$$="";}
;


linguagensBase:  BASELANG listaLinguagensBase   {asprintf(&$$, "%s", $2);}
    ;

listaLinguagensBase: FLAG listaLinguagensBase                            {asprintf(&$$, "%s %s",
| {$$="";}
;

inversas:  INV FLAG FLAG
```

```
{ // assumimos que apenas existem duas possíveis relaçõs
tmpFilho = strdup($2);
tmpPai = strdup($3);
asprintf(&$$, "%s %s", $2, $3);
}
    ;

conceitos: conceito conceitos {asprintf(&$$, "%s %s", $1, $2);}
|  {$$="";}
;

conceito: dados   {
asprintf(&$$, "%s", $1);
}

dados: dado dados {asprintf(&$$, "%s\n%s", $1, $2);}
| {
// este código foi a tentativa para o requisito 2 de guardar a info em memória
/*NODE tmpNode = (NODE) malloc(sizeof(struct node));
tmpNode->filhos = ARRAY_LIST(10);

int flag = 0;

if(conceito != NULL){
flag++;
tmpNode->conceito = strdup(conceito);
conceito = NULL;
}
if(traducao != NULL){

tmpNode->traducao = strdup(traducao);
traducao = NULL;
}
if(scopeNote != NULL){

tmpNode->scopeNote = strdup(scopeNote);
scopeNote = NULL;
}
if(pai != NULL){

tmpNode->pai = strdup(pai);
pai = NULL;
}

adicionarNode(conceitos, tmpNode);
*/
$$="";
}
```

```
;

dado:  FLAG TXT {
if(strcmp(strdup($1), tmpTraducoes[0]) == 0 || strcmp(strdup($1), tmpTraducoes[1]) == 0){
traducao = strdup($2);
asprintf(&$$, "TRADUÇÃO=[%s %s]", $1, $2);
fprintf(file, "TRADUÇÃO: %s<br>", traducao);
}else if(strcmp(strdup($1), tmpPai) == 0){
asprintf(&$$, "PAI=[%s %s]", $1, $2);
pai = strdup($2);

// HTML -------------------------------------
fprintf(file, "RELAÇÃO INVERSA SUPERIOR: %s<br>", pai);
char* ref1 = malloc(((strlen($2)+6+8)*sizeof(char)));
char* ref2 = malloc((strlen($2)+6*sizeof(char)));
                                                sprintf(ref1,"paginas/%s.html",$2);
                                                sprintf(ref2,"%s.html",conceito);
FILE* tmpFile = fopen(ref1, "a");
fprintf(tmpFile, "<a href=\"%s\">%s</a><br>", ref2, conceito);
// HTML -------------------------------------

}else if(strcmp(strdup($1), tmpFilho) == 0){
asprintf(&$$, "FILHO=[%s %s]", $1, $2);
filho = strdup($2);
fprintf(file, "RELAÇÃO INVERSA INFERIOR: %s<br>", filho);
}else if(strcmp(strdup($1), "SN") == 0){
scopeNote = strdup($2);
asprintf(&$$, "StickNote=[%s %s]", $1, $2);
fprintf(file, "NOTA EXPLICATIVA: %s<br>", scopeNote);
}
}
| TXT                                             {
// HTML -------------------------------------
char* ref = malloc(((strlen($1)+6+8)*sizeof(char)));
                                                sprintf(ref,"paginas/%s.html",$1);
file = fopen(ref, "w");
beginHTML(file, strdup($1));
// HTML -------------------------------------

conceito = strdup($1);
asprintf(&$$, "|%s|", $1);
}
;

%%

#include "lex.yy.c"
```

```
void yyerror(char *s){
fprintf(stderr,"Erro:%sLine:%d\n",s, yylineno);
}

int main(){

system("rm -f -r paginas/");
    system("mkdir paginas/");
    system("rm -r *.html");

file = fopen("index.html", "w");

beginHTML(file, "Thesaurus");

fprintf(file, "<a href=\"paginas/Life.html\">Life</a>");

// este código foi a tentativa para o requisito 2 de guardar a info em memória
/*
conceitos = ARRAY_LIST(10);

NODE novoNo = (NODE)malloc(sizeof(struct node));
novoNo->conceito = "Life";
novoNo->traducao = "life";
novoNo->filhos = ARRAY_LIST(10);
*/

file = fopen("paginas/Life.html", "w");

beginHTML(file, "Life");

//addElem(conceitos, novoNo);
yyparse();
}
```