

Processamento de linguagens (3º ano de MIEI)

Trabalho Prático 1

Relatório de Desenvolvimento

Célia Figueiredo
(a67637)

Diogo Tavares
(a61044)

Gil Gonçalves
(a67738)

5 de Abril de 2016

Resumo

Neste relatório será apresentado o desenvolvimento de um filtro de texto com a posterior aplicação sob um ficheiro BibTex, pretende-se que seja produzido um normalizador de ficheiros BibTex.

Implementou-se um filtro que permite fazer a contagem das categorias das referências bibliográficas, também foi implementado um filtro que permite a troca para chavetas do campo que está entre aspas. E ainda foi implementado um filtro que coloca os nomes dos autores escritos no formato "N. Apelido". Por fim, de modo a tornar a leitura mais fácil implementou-se um ferramenta de *pretty-printing*.

Será também mostrado um grafo que ilustra para um dado autor (escolhido pelo utilizador) todos os autores que publicam normalmente com o autor em causa. Utilizou-se a linguagem *Dot* do *GraphViz* esta, que gerou um ficheiro com um grafo de modo a que posteriormente fosse usada uma das ferramentas que processam *Dot* para desenhar o dito grafo de associações de autores.

Conteúdo

Capítulo 1

Introdução

Este trabalho envolverá o desenvolvimento de um normalizador de ficheiros *BibTex*, este é o tema do problema 2.2 do enunciado fornecido.

Enquadramento Utilização de expressões regulares e filtros de texto com o objetivo de produzir novos documentos a partir de padrões existentes no ficheiro de input.

Conteúdo do documento O presente documento contém a explicação do problema, assim como a apresentação das soluções produzidas.

Resultados Os resultados deste desafio serão as alíneas pedidas, sendo que serão apresentados em ficheiros *.html* e grafos.

Estrutura do Relatório

Este documento está dividido em seis partes. No capítulo 2 faz-se uma análise detalhada do problema proposto de modo a poder-se especificar as entradas, resultados e formas de transformação do ficheiro *.bib*.

No capítulo 3 serão descritas as estruturas de dados implementadas para a realização do problema descrito. No capítulo 4 serão mostradas as expressões regulares desenvolvidas para a implementação do caso de estudo, assim como os testes realizados e os resultados. No capítulo 5 termina-se o relatório com uma síntese do que foi dito, as conclusões e o trabalho futuro. No fim do documento estará incluído ficheiros anexos com o código desenvolvido.

Capítulo 2

Análise e Especificação

2.1 Descrição informal do problema

O problema escolhido consiste na análise da ferramenta de formatação de citações e referências bibliográficas em documentos \LaTeX .

O *BibTex* é uma ferramenta que foi criada em 1985 e utiliza um ficheiro (*.bib*) este que é uma base de dados que contém os dados bibliográficos (autor, título, ano de publicação, etc.) das fontes citadas no documento \LaTeX .

Deixámos um exemplo de um excerto de um ficheiro com a extensão *.bib*:

```
@inbook{Val90a,
author = "Jos\'e M. Valen\c{c}a",
title = "Processos, {0}bjectos e {C}omunica\c{c}\~ao
({0}p\c{c}\~ao I - {MCC})",
chapter = 2,
year = 1990,
month = Oct,
publisher = gdcc,
address = um,
annote = "programacao oobjectos, proc comunicantes, espec formal"
}
```

O ficheiro BibTex contém várias categorias de referência estas inicializadas sempre pelo caracter @, deixamos aqui alguns exemplos de entradas que podem ser encontradas no ficheiro BibTex:

```
@inbook
@misc
@incollection
@inproceedings
@techreport
@unpublished
```

2.2 Especificação do Requisitos

2.2.1 Dados

Os dados fornecidos são um ficheiro *.bib*, este que é um ficheiro com as características de um ficheiro *BibTex*.

Cada tipo de categoria tem os seus campos obrigatórios, neste acaso o objetivo de uma das tarefas será pesquisar através do campo **author** = o nome do autor e transformá-lo no formato "N. Apelido".

O nome da categoria é seguida por uma chaveta, e o primeiro campo será o nome para a referência a ser introduzida, os campos de cada categoria são separados por vírgula e a seguir ao campo aparecerá o símbolo igual (=), deixámos um exemplo da sintaxe da categoria **@phdthesis** e os respetivos campos presentes:

```
@phdthesis{Mos75a,
author = "P. D. Mosses",
title = "Mathematical Semantics and Compiler Generation",
year = 1975,
school = "Oxford University",
annote = "compilacao incremental, atributos, ambientes prog"
}
```

É também fornecido o nome de ferramentas de apoio à resolução do problema, neste caso o *Graph Viz*, que permitirá colocar gráficamente a informação dos grafos criados, sendo que tornará as iterações entre os autores mais percetíveis.

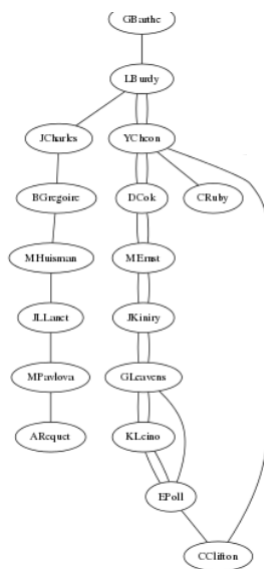


Figura 2.1: Exemplo de geração de um grafo com os nomes dos autores com recurso à ferramenta *Graph Viz*

Capítulo 3

Concepção/desenho da Resolução

3.1 Estruturas de Dados

3.1.1 Algoritmo alinea a)

Nesta alinea optou-se por utilizar listas ligadas porque vai guardando as categorias e o respetivo contador à medida que se encontra um padrão no ficheiro *.bib*.

Decidimos que as categorias que se diferenciavam apenas nas letras maiúsculas ou minúsculas pertenceriam à mesma categoria.

```
void* procura(char* c){
int flag = 1;

Est* aux;
aux = est;
while(aux!=NULL && flag && aux->cat != NULL){
if(!strcmp(minusculas(aux->cat), minusculas(c+1))){
aux->i++;
flag=0;
}
aux=aux->next;
}

if(flag){
Est* novo = (Est*) malloc(sizeof(Est));
novo->cat = strdup(c+1);
novo->i = 1;
novo->next = est;
est = novo;
}
}
```

3.1.2 Algoritmo alinea b1)

Utilizámos dois arrays, um que guarda a informação que é lida, e outro que guarda a informação tratada.

```
void trata()
```

3.1.3 Algoritmo alínea b2)

Utilizamos arrays strtok para apanhar o que está entre chavetas ou aspas strstr para cortar a string por autores

3.1.4 Algoritmo alínea c)

Nesta alínea utilizámos arrays

Capítulo 4

Codificação e Testes

4.1 Alternativas, Decisões e Problemas de Implementação

4.1.1 Expressões Regulares

Para garantir que uma secção do código do ficheiro input ficasse protegida de todas as acções accionadas por outras expressões regulares declaramos o estado %x.

Alinea a)

Nesta alinea é pedido que se faça a contagem das categorias existentes, e no final se produza um ficheiro em formato HTML como o nome e as respectivas contagens.

Para a realização deste desafio é necessário procurar a categoria inicializada pelo caractere @ até encontrar a chaveta de abertura ({}). A categoria encontrada será inserida na estrutura através da função *procura*, se a categoria já existir adiciona mais uma entrada ao contador, senão cria uma nova e insere-a na estrutura. Todas as categorias que obdecerem à expressão regular são capturadas com o *yytext*.

```
@[^{}]* {procura(yytext);}
```

O restante texto é ignorado com a seguinte expressão:

```
.\n {;}
```

Alínea b1)

Esta alínea está dividida em duas partes. A primeira pede que sempre que um campo está entre aspas estas sejam trocadas por chavetas e o nome dos autores deverá apresentar o formato "N. Apelido".

Para apanhar o conteúdo do campo **author** = , este que pode iniciar-se com chaveta de abertura ou aspas, sugerimos os seguintes filtros:

```
"author ="[ ]+"\" {BEGIN (AUT);}  
"author ="[ ]+"\" {BEGIN (AUT);}
```

Quando num autor se encontrar caracteres especiais no meio do nome tais como chavetas ou aspas, insere esses caracteres no array.

```
<AUT>\" {inser(yytext); }  
<AUT>\" {inser(yytext); }
```

Implementámos os filtros que nos permitem separar os nomes dos autores, estes que podem estar separados por vírgulas ou **and**.

```

<AUT>"\", "[ ]+ {trata();funcao();}
<AUT>[ ]+"and"[ ]+ {trata();funcao();}
<AUT>[ ]+"and"(\n) {trata();funcao();}

```

Estes filtros permitem-nos determinar quando um autor termina, tanto com aspas seguida de vírgula ou chaveta seguida de vírgula.

```

<AUT>"\"""\", " {trata();funcao();print();tmp=NULL;BEGIN INITIAL;}

```

```

<AUT>"\"\"\"\", " {trata();funcao();print();tmp=NULL;BEGIN INITIAL;}

```

Este filtro lê tudo o que está no à frente do campo autor:

```

<AUT>[ ' -}] {inser(yytext); }
\"[a-zA-Z] {printf("{%c", yytext[1]);}
[a-zA-Z]\" {printf("%c", yytext[0]);}

```

Alínea c)

Nesta alínea é pedido que seja construído um grafo que mostre para dado autor todos os autores que publicam com ele.

Para tal é necessário que seja feita a recolha de todos os nomes dos autores, depois para associá-los utiliza-se a expressão *and* que nos indica quais os autores que tem publicações conjuntas.

Este filtro tal como o anterior serve para apanhar o conteúdo do campo **author** = para os dois casos possíveis no ficheiro *.bf*.

```

"author ="[ ]*"\""" {BEGIN (AUT);}
"author ="[ ]*"\"{" {BEGIN (AUT);}

```

```

<AUT>"\"}" {;}

```

```

<AUT>"\", "[ ]+ {trata();}
<AUT>[ ]+"and"[ ]+ {trata();}
<AUT>[ ]+"and"(\n) {trata();}

```

```

<AUT>"\"""\", " {trata();print();tmp=NULL;BEGIN INITIAL;}
<AUT>"\"\"\"\", " {trata();print();tmp=NULL;BEGIN INITIAL;}
<AUT>[A-Za-z] {inser(yytext);}

```

4.1.2 Makefile

O principal objetivo da Makefile é facilitar a compilação e execução do programa. Para isso criamos o seguinte ficheiro:

4.2 Testes realizados e Resultados

Mostram-se a seguir alguns testes feitos e respectivos resultados obtidos:

4.2.1 Testes e Resultados alínea a)

Utilizámos este ficheiro de teste para testar as diferentes opções que apareciam no ficheiro

```
1 @book{
2 @book{
3 @cenas{
4 @book{
5 @coisas{
6 @Book{
7 @book{
8 @BOOK{
9     @BOOK{
10     @Book{
11     @author{
12
13 @AuThor{
14 @AUTHOR{
```

Mostrámos de seguida o ficheiro *.html* produzido:

```
proceeding -> 1
mastersthesis -> 2
proceedings -> 4
misc -> 61
manual -> 13
incollection -> 6
unpublished -> 15
inproceedings -> 209
article -> 142
phdthesis -> 21
book -> 47
inbook -> 3
techreport -> 140
string -> 31
```

4.2.2 Testes e Resultados alinea b)

4.2.3 Testes e Resultados alinea c)

Capítulo 5

Conclusão

Na alínea a pressupusemos que as categorias com o mesmo nome e que se diferenciavam apenas em maiúsculas ou minúsculas pertenciam à mesma categoria. Pois assim não estaríamos a repetir informação.

A alínea c foi implementada pensando que o grafo seria criado com todos os autores. Porém após uma leitura mais atenta verificou-se que seria pedido que o utilizador escolhesse o autor e assim gerar o grafo com os autores que publicam diretamente com ele.

Apêndice A

Código do Programa

Lista-se a seguir o código que foi desenvolvido para a alínea a:

```
1  %{
2
3  typedef struct Est{
4      char* cat;
5      int i;
6      struct est *next;
7  }Est;
8
9  Est* est;
10
11 void* novo() {
12     est =(Est*) malloc(sizeof(Est));
13     est->next = NULL;
14     est->cat = NULL;
15     est->i = 0;
16 }
17
18 char* minusculas(char* d){
19     char* novo = strdup(d);
20     int j=0;
21     while(novo[j] != '\0'){
22         novo[j]= tolower(novo[j]);
23         j++;
24     }
25     return novo;
26 }
27
28 void* procura(char* c){
29     int flag = 1;
30
31     Est* aux;
32     aux = est;
33     while(aux!=NULL && flag && aux->cat != NULL){
34         if(!strcmp(minusculas(aux->cat), minusculas(c+1))){
35             aux->i++;
36             flag=0;
37         }
38         aux=aux->next;
39     }
40
41     if(flag){
```

```

42         Est* novo = (Est*) malloc(sizeof(Est));
43         novo->cat = strdup(c+1);
44         novo->i = 1;
45         novo->next = est;
46         est = novo;
47     }
48 }
49
50 %}
51
52 %%
53 @[^\{]* {procura(ytext);}
54 .|\n    {;}
55
56 %%
57
58 void print(){
59     FILE* fp;
60     fp = fopen("ficheiro.html", "w");
61     fprintf(fp, "<html>");
62     while(est->next!=NULL){
63         fprintf(fp, "%s -> %d<br>", est->cat, est->i);
64         est = est->next;
65     }
66     fprintf(fp, "<\html>");
67     fclose(fp);
68 }
69
70 int yywrap() {return 1;}
71 int main(){
72     est = novo();
73     yylex();
74     print();
75     free(est);
76     return 0;
77 }
78 }

```

Lista-se a seguir o código que foi desenvolvido para a alínea b:

```

1  %{
2  #include<string.h>
3  char  nomes[1000];
4  int  i=0;
5  char *tmp;
6
7
8  void inser(char *texto) {
9      nomes[i]=*texto;
10     i++;
11 }
12
13
14 void trata() {
15     nomes[i]='\0';
16
17     int variavel =1;
18
19     if(nomes[variavel]== ' ') {
20         char kill[1000]; int o=0;

```

```

21     while(nomes[variavel]== ' ') {variavel++;}
22     while(nomes[variavel]!='\0') {
23         kill[o]=nomes[variavel];
24         o++;
25         variabel++;
26     }     kill[o]='\0';
27     o=0;
28     while( kill[o]!='\0') {
29         nomes[o]=kill[o];
30         o++;
31     }
32
33     nomes[o]='\0';
34
35 }
36
37     int n= strlen(nomes);
38     int j=n-1;
39
40
41     while(nomes[j]== ' ') {j--;}
42
43     while(nomes[j]!=' ' && j!=0){
44         j--;
45     }
46     int k=1;
47     nomes[k]='.';
48     k++;
49
50     if(j!=0) {
51         while(nomes[j]!='\0') {
52
53             nomes[k]=nomes[j];
54             j++;
55             k++;
56         }
57
58         nomes[k]='\0';
59         i=0;
60
61     }
62     else {
63         nomes[k]='\0'; i=0;
64     }
65
66
67 }
68
69 void funcao() {
70
71     if (tmp) {
72
73         i=0;
74         strcat(tmp," and ");
75         int n=strlen(tmp);
76         int j=0;
77         tmp[n]=' ';
78         n++;
79

```

```

80 for (j=0;nomes[j]!='\0';j++) {
81
82     tmp[n]=nomes[j];
83     n++;
84
85 }
86 tmp[n]='\0';
87
88 }
89
90 else {
91     i=0;
92     int j=0;
93     tmp=(char *) malloc(1000);
94
95     for (j=0;nomes[j]!='\0';j++) {
96         tmp[j]=nomes[j];
97
98     }
99     tmp[j]='\0';
100
101 }
102
103 }
104 }
105
106
107 void print() {
108
109     char h[1000]="author = {";
110
111     strcat(h,tmp);
112     int n=strlen(h);
113     h[n]='}';
114     n++;
115     h[n++]=',';
116
117     h[n]='\0';
118     printf("%s",h);
119
120 }
121
122
123
124
125
126
127 %}
128
129
130 %x AUT
131
132 %%
133
134
135 "author ="[ ]*"\" {BEGIN (AUT);}
136 "author ="[ ]*"\" {BEGIN (AUT);}
137
138

```



```

139 <AUT>"\" {inser(yytext); }
140 <AUT>"\" {inser(yytext); }
141 <AUT>"\",[ ]+ {trata();funcao();}
142 <AUT>[ ]+"and"[ ]+ {trata();funcao();}
143 <AUT>[ ]+"and"(\n) {trata();funcao();}
144 <AUT>"\"\"\", {trata();funcao();print();tmp=NULL;BEGIN INITIAL;}
145 <AUT>"\"\"\", {trata();funcao();print();tmp=NULL;BEGIN INITIAL;}
146 <AUT>[ '-}] {inser(yytext); }
147
148 \"[a-zA-Z] {printf("{%c", yytext[1]);}
149 [a-zA-Z]\ " {printf("%c", yytext[0]);}
150
151
152 %%
153
154 int yywrap() {return 1;}
155 int main(){
156     yylex();
157
158     free(tmp);
159     return 0;
160 }

```

Lista-se a seguir o código que foi desenvolvido para a alínea c:

```

1 %{
2 #include<string.h>
3 char nomes[1000];
4 int i=0;
5 char *tmp;
6 FILE *fp;
7
8
9 void inser(char *texto) {
10     nomes[i]=*texto;
11     i++;
12 }
13
14 void trata() {
15     nomes[i]='\0';
16     if(tmp) {
17         i=0;
18
19         int n=strlen(tmp);
20         tmp[n++]='-';
21         tmp[n++]='-';
22         tmp[n]='\0';
23         strcat(tmp,nomes);
24     }
25
26     else {
27         tmp=(char *) malloc(1000);
28         strcat(tmp,nomes);
29         i=0;
30
31     }
32
33 }
34
35 void print() {

```

```

36  i=0;
37      fprintf(fp,"%s",tmp);
38      fprintf(fp,"\n");
39
40  }
41
42
43
44  void abrir(){
45
46      fp = fopen("ficheiro.dot", "w");
47      fprintf(fp, "graph {");
48  }
49
50  void fechar() {
51
52
53      fprintf(fp, "}");
54      fclose(fp);
55
56  }
57
58
59
60
61
62  %}
63
64  %x AUT
65
66
67
68  %%
69
70  "author ="[ ]*"\" {BEGIN (AUT);}
71  "author ="[ ]*"\" {BEGIN (AUT);}
72  <AUT>"\}"      {;}
73  <AUT>"\,"[ ]+   {trata();}
74  <AUT>[ ]+"and"[ ]+ {trata();}
75  <AUT>[ ]+"and"(\n) {trata();}
76  <AUT>"\"""\""," {trata(); print();tmp=NULL;BEGIN INITIAL;}
77  <AUT>"\}"\""," {trata(); print();tmp=NULL;BEGIN INITIAL;}
78  <AUT>[A-Za-z]    {inser(ytext);}
79
80
81
82  %%
83
84
85  int yywrap() {return 1;}
86  int main(){
87      abrir();
88      yylex();
89      fechar();
90      free(tmp);
91      return 0;
92  }

```

Bibliografia