Processamento de Linguagens

BibTeXPro — Um processador de BibTeX

Relatório de Desenvolvimento

Bruno Arieira (A70565) | Jorge Cruz (A78895) | José Dias (A78494)

(7 de Maio de 2018)

Resumo

"BibTeX tem sido amplamente utilizado desde sua introdução por Oren Patashnik há 20 anos. Como o nome sugere, foi concebido para ser usado em combinação com o sistema de composição tipográfica LaTeX, mas tornou-se possível, por exemplo, incluir bibliografias do BibTeX, mesmo em documentos do Word, utilizando ferramentas de terceiros."

Este trabalho foi realizado no âmbito da unidade curricular Processamento de Linguagens, tem como principal objetivo a análise de documentos *BibText*, de acordo com o pedido pelo enunciado. Recorrendo ao gerador *FLEX*, demonstramos neste relatório o processo de desenvolvimento de filtros de texto.

Conteúdo

1 Introdução

Neste segundo trabalho prático ambicionamos aplicar os conceitos apreendidos nas aulas de Processamento de Linguagens, relativamente á ferramenta *FLEX*, com principal fundamento de aumentar a nossa experiência em ambiente Linux, a capacidade referente ao desenvolvimento de Expressões Regulares para retratar padrões de frases, como também aplicá-las de forma a torná-las úteis para análise de documentos.

Tal como no último trabalho prático, foram disponibilizados cinco problemas em concreto, que através de uma fórmula dada no enunciado (com a aplicação do número de aluno mais baixo do grupo de trabalho) foi-nos atribuído o primeiro problema, tal como referimos na capa deste relatório, *BibTeXPro — Um processador de BibTeX*. *BibTex* é uma ferramenta de formatação usada em documentos *Latex* que permite a separação da bibliografia de um dado documento com a apresentação do texto.

De acordo com o enunciado, necessitamos de analisar ficheiros **bib**, através da geração de programas FLex baseada em especificações com Expressões Regulares, cumprindo os requisitos solicitados, assim como outras funcionalidades implementadas por nós.

2 Análise do Texto Fonte

Este enunciado tem como objetivo a análise de ficheiros **bib**, que contém informação relativa a dados de bibliografias de documentos, fornecendo principais referências dos mesmos. Assim, foram fornecidos dois ficheiros **bib** diferentes. De acordo com este trabalho prático, este tipo de ficheiros podem conter diferentes bibliografias de vários documentos, que são caracterizadas pelas respetivas categorias (que fazem um total de 14), onde cada uma contém campos obrigatórios e opcionais. Estes campos, diferentes para cada categoria, podem ser:

- title
- journal
- year
- author/editor
- publisher
- chapter/pages
- booktitle
- school
- institution
- type
- address
- month
- note
- key
- series
- organization
- howpublished
- edition
- volume/number

Em continuação, apresentamos dois excertos, com categorias diferentes, do texto-fonte como exemplo de ilustração e para dar suporte à estratégia usada.

```
@InProceedings{CFPBH07d,
  author =    {Daniela da Cruz and Rúben
      and Maria João Varanda Pereira
      and Mário Béron and Pedro Rangel
               Henriques},
  title =     {Comparing Generators
      for Language-based Tools},
  booktitle = {CoRTA-07 - Compiler
      Related Techonlogies and
      Applications, Covilhã, Portugal},
  year =      {2007},
  month =     {July}
}
```

```
@Article{LARH09,
 author={Giovani Rubert Librelotto
        and Renato Preigschadt de Azevedo
        and José Carlos Ramalho
        and Pedro Rangel Henriques},
  title = {Topic Maps Constraint Languages:
        understanding and comparing},
  journal = {Int. Journal Reasoning-based
        Intelligent Systems},
  publisher = {Inderscience Entreprises, Ltd.},
  year = {2009},
  month = {Jun},
  volume = {1},
  number = {3/4},
  pages = {173-181},
  note = {ISSN: }
}
```

2.1 Ações Semânticas

Após uma análise do ficheiro **bib** de exemplo presente na plataforma *elearning* foram descobertos alguns padrões no documento. Como já podemos verificar antes os elementos estão de forma geral definidos da seguinte forma:

```
@Categoria{Chave,
  author = {Autor1 and Autor2 and ...},
  title = {...},
  publisher = {...},
  year = {...},
  month = {...},
  etc.
}
```

Após nos depararmos com estes padrões a parte de análise e captura da informação relevante foi relativamente simples. O caratér "@"é apenas utilizado no início de uma nova referência e imediatamente após segue-se a respetiva categoria. Existe um caso especial na utilização do "@"para quando ocorre um *match* a `@string{.*\n` pois esses nao representam uma referência.

Dentro de cada referência surge o conteúdo encapsulado entre chavetas. Até á primeira virgula do conteúdo surge a chave da referência em questão e a partir deste ponto surgem os vários pontos relativos já referidos em cima entre os quais autores, títulos, anos de publicação, mês, etc.

Na construção deste projeto e utilizando o conhecimento já descrito, iniciamos vários campos:

- `CAT` :: Sempre que encontramos o caratér "`@`"iniciamos este campo respetivo à categoria, no qual guardamos todos os carateres que seja letras ou números.
- `KEY` :: Após captar a categoria da referência sabemos que se seguia a chave a seguir à primeira chaveta, como tal surge este campo, no qual guardamos todos os carateres que sejam letras ou números. Quando aparecer uma vírgula passa ao campo do conteúdo.
- `CONTENT` :: Posteriormente seguem-se todos os campos da referência, o seu conteúdo. Este campo permitiu separar o conteúdo em campos mais fáceis de trabalhar. Dentre os quais:

  `AUT` :: Corresponde ao campo de autores, percorrido caratér a caratér pois devido á separação de autores por uma palavra, neste caso "and"não era possível captar simplesmente todas as letras. Este campo é ativado quando, dentro do conteúdo, aparece "`author = {`"ou "`author = ¨`". Quando aparece a chaveta ou as aspas correspondentes ao fim, passa-se novamente para o campo do conteúdo.

  `TIT` :: Corresponde ao campo do título, ativado por "`title = {`"ou "`title = ¨`". Quando aparece a chaveta ou as aspas correspondentes ao fim, passa-se novamente para o campo do conteúdo.

2.2 Estrutura de Dados

A estrutura deste trabalho foi implementada recorrendo à utilização da biblioteca `glib`, que é bastante útil para armazenamento de dados.

Para guardar o nome de cada categoria/autor e as correspondentes informações, foram usadas árvores binárias balanceadas em que cada nodo possui uma chave e um valor. Todas as árvores foram inicializadas com uma função que permite ordenar e percorrer os nodos consoante a ordem alfabética das chaves. A árvore '`cat_tree`' corresponde às categorias, a árvore '`aut_tree`' corresponde aos autores, e a árvore '`list_aut_tree`' corresponde aos autores relacionados com o autor que poderá ser pesquisado.

```
GTree* cat_tree = g_tree_new((GCompareFunc)g_ascii_strcasecmp);
GTree* aut_tree = g_tree_new((GCompareFunc)g_ascii_strcasecmp);
GTree* list_aut_tree = g_tree_new((GCompareFunc)g_ascii_strcasecmp);
```

De seguida apresentamos a estrutura para os nodos da árvore onde guardamos as informações de uma categoria. O inteiro '`count`' corresponde ao número de vezes que a respetiva categoria apareceu, e o '`out` é um apontador

para o ficheiro HTML no qual vão ser escritos as informações de cada referência à categoria.

```
typedef struct Category {
        int count;
        FILE* out;
} * catInfo;
```

Por último, existe a estrutura para guardar as chaves das referências onde um dado autor aparece. As chaves serão armazenadas num 'GArray' e a sua contagem no inteiro 'count_keys'. Estas informações estarão presentes nos nodos dos respetivos autores.

```
typedef struct Author {
        GArray* list_keys;
        int count_keys;
} * autInfo;
```

Ao longo do programa são utilizados também 'arrays' que guardam informação temporária tal como, o nome da chave da referência atual.

3 Filtro de Texto - Gerador FLEX

Depois da análise e interpretação de ficheiros bib efetuada através da análise de padrões e estudo de como vamos tratar a informação e armazená-la podemos então prosseguir ao desenvolvimento de filtros de texto usando o gerador **FLEX**.

3.1 Contagem das Categorias e respetivas chaves, autores e títulos

Este exercício tem como objetivo encontrar as diversas categorias implícitas num ficheiro e gerar um documento `HTML` com os nomes das categorias e as respetivas contagens. É obtido também, para cada categoria, todas as chaves das referências e os respetivos autores e título.

A seguinte função será a responsável por adicionar as categorias e a respetiva contagem na árvore. Primeiramente, é necessário verificar se a categoria já existe na árvore através da função '`g_tree_lookup`'. Caso ainda não exista, é inserida na árvore, criando um novo ficheiro com o nome da categoria e, paralelamente, alocando a estrutura correspondente às informações das categorias, que armazenará o apontador para este ficheiro e a contagem inicializada com o valor '1'. Caso já esteja presente na árvore, basta incrementar o valor da contagem presente na estrutura do nodo devolvido.

```
void addCategory(char* category, int catleng, GTree *t) {

    gpointer* value = g_tree_lookup(t, category);

    if (!value) {

        catInfo category_info = malloc(sizeof(struct Category));
        char filename[100];

        sprintf(filename, "%s.html", g_ascii_strdown(category,-1));

        category_info->count = 1;
        category_info->out = fopen(filename, "w");
        out = category_info->out;
        openHTML(out);

        g_tree_insert(t, g_strdup(category), category_info);

    }
    else {

        ((catInfo)value)->count++;
        out = ((catInfo)value)->out;

    }
}
```

Este apontador de ficheiro guardado na árvore é importante pois, quando entramos numa nova referência, é necessário substituir o ficheiro de `output` para o qual será imprimido a chave e os respetivos nomes dos autores e título que irão aparecer de seguida. Sendo assim, existe um apontador de ficheiro global '`out`' que será substituído a cada nova referência pelo ficheiro presente no nodo da correspondente categoria.

Quando for atingido o fim do ficheiro `bib` de `input`, é imprimido num ficheiro `HTML` um índice com a contagem das categorias, sendo que, o nome da categoria é uma hiperligação para o correspondente ficheiro mencionado anteriormente. Para percorrer os nodos da árvore, recorreu-se à função '`g_tree_foreach`' que recebe como argumento uma função que será ativada para cada nodo, e que, neste caso, corresponde a imprimir o nome da categoria com hiperligação ao seu ficheiro e a sua contagem.

3.2 Índice de Autores

O seguinte exercício visava obter um índice de autores que mapeasse cada autor nos respetivos registos, de modo a que posteriormente uma ferramenta de procura do Linux pudesse fazer a pesquisa.

O processo para atingir este objetivo revelou-se semelhante ao anterior, porém agora pretendemos usar como chave dos nodos da árvore o nome dos autores e guardar para cada um, uma lista de chaves das referências onde estes são mencionados.

Deste modo, destaca-se a seguinte função que é chamada quando encontrado um autor e que adiciona a chave corrente ao nodo correspondente na árvore. Tal como anteriormente, caso o autor ainda não esteja presente na árvore, é necessário alocar a estrutura que irá guardar as chaves e a sua contagem. Caso o autor já esteja na árvore, adiciona-se a nova chave à estrutura existente no nodo e incrementa-se a contagem.

```
void addKeyToAuthor(gchar* author, gchar* key, GTree * t) {

        gchar * keyval = g_strdup(key);
        gpointer* value = g_tree_lookup(t, author);

        if (!value) {

                autInfo author_info = malloc(sizeof(struct Author));

                author_info->list_keys = g_array_new(FALSE, FALSE, sizeof(gchar*));
                author_info->count_keys = 1;

                g_array_append_val(author_info->list_keys, keyval);

                g_tree_insert(t, g_strdup(author), author_info);

        }
        else {

                ((autInfo)value)->count_keys++;
                g_array_append_val(((autInfo)value)->list_keys, keyval);

        }
}
```

Por fim, quando atingido o fim do ficheiro de `input`, é imprimido para um ficheiro `HTML` as informações recolhidas. Ou seja, percorrendo a árvore, para cada nodo imprime-se o nome do autor e as chaves que se encontram na sua estrutura. Sempre que quisermos obter um registo de um certo autor, basta procurar a chave no ficheiro bib fornecido.

3.3 Grafo de Autores

O último exercício tinha como objetivo construir um grafo que mostrasse, para um dado autor (definido à partida) todos os autores que publicam normalmente com o autor em causa. Recorrendo à linguagem `Dot` do `GraphViz`, queria-se gerar um ficheiro com esse grafo de modo a que se pudesse, posteriormente, usar uma das ferramentas que processam `Dot` para desenhar o dito grafo de associações de autores.

Após considerar o problema, chegou-se à conclusão que seria necessário um `array` que guardasse a lista dos nomes dos autores do registo corrente. Chegando ao fim do campo dos autores, teria-se de avaliar esta lista, procurando o nome do autor definido no início. É neste momento que entra a seguinte função.

```
void searchAuthor(GArray* list_aut, int size, GTree* t) {

        int found = 1, pos;

        for (pos = 0; pos < size && found != 0; pos++)
                found = g_ascii_strcasecmp(author_search, g_array_index(list_aut, gchar*, pos));

        pos--;

        if (found == 0)
        {
                for (int i = 0; i < size; i++)
                        if (i != pos) {
                                gchar * aut = g_array_index(list_aut, gchar*, i);
                                gpointer* value = g_tree_lookup(t, aut);

                                if (!value)
                                        g_tree_insert(t, aut, (gpointer*)1);
                                else
                                        g_tree_replace(t, aut, (int)value + 1);
                        }
        }
        else
                for (int i = 0; i < size; i++)
                        g_free( g_array_index(list_aut, gchar*, i) );

        g_array_free(list_aut, FALSE);
}
```

Esta função, chamada quando é atingido o fim do campo dos autores, recebe a lista com o nome dos autores e procura o autor que é pedido. Caso encontre o autor, guarda numa árvore o nome dos outros autores e o número de vezes em que já partilharam registos. Assim, para esta árvore, cada nodo tem como chave o nome dos autores, e como valor a contagem.

Novamente, no fim do ficheiro de `input`, percorre-se a árvore, sendo que para cada nodo, imprime-se, em linguagem `Dot` para um ficheiro, o nome do autor que foi procurado, uma seta '`->`' a representar a associação, o nome do autor no nodo, e a contagem dentro de uma `label`. Deste modo, usando o comando do `GraphViz` no ficheiro imprimido, obtemos um grafo com nodos contendo o nome dos autores, e ligações entre eles contendo como atributo a contagem.

4 Conclusão

Tal como já foi referenciado, o principal objetivo deste trabalho prático foi aplicar os conhecimentos adquiridos nas aulas práticas ao longo do semestre em relação á utilização de filtros de texto em **FLEX**. Com isto foi nos atribuído, conforme o mais baixo número de aluno, o primeiro enunciado.

Na realização deste trabalho prático, as dificuldades prenderem-se em identificar os autores para cada categoria, pois o que separava os diferentes autores era a palavra "and"(constituida por caratéres), sendo que complicava o processo de seleção dos respetivos autores. No entanto, conseguimos resolver este problema percorrendo caratér a caratér o campo AUT, considerando que o que separava os autores era a palavra "and".

Outra das dificuldades que sentimos com este o desenvolvimento deste trabalho foi o uso da biblioteca **GLIB**, que com o auxílio do estudo e interpretação da sua documentação (que se encontra explícita nas referências), conseguimos ultrapassar.

Em suma, conseguimos realizar todas as funcionalidades dos requisitos impostos no enunciado com sucesso, que nos deixou com um conhecimento mais aprofundado relativamente á construção de filtros de texto e expressões regulares usando **FLEX**.

5 Ficheiros

```
%option noyywrap
%x STR KEY CAT CONTENT AUT TIT

%{

#include <unistd.h>
#include <stdio.h>
#include <glib.h>
#include <glib/gprintf.h>

FILE* out;
gchar* author_search = NULL;

typedef struct Category {
       int count;
       FILE* out;
} * catInfo;

typedef struct Author {
       GArray* list_keys;
       int count_keys;
} * autInfo;

void openHTML(FILE* file) {
       fprintf(file, "<!DOCTYPE html>\n<html>\n");
       fprintf(file, "<head>\n<meta charset=\"utf-8\">\n</head>\n");
       fprintf(file, "<body>\n");
}

void openGraph(FILE* file) {
       fprintf(file, "digraph Author {\n");
       fprintf(file, " ranksep = 4.0\n\n");
}

void closeGraph(FILE* file) {
       fprintf(file, "}\n");
}

void closeHTML(FILE* file) {
       fprintf(file, "</body>\n</html>\n");
}

gboolean printGraph(gpointer author, gpointer count, gpointer data) {

       g_fprintf(out, " \"%s\" -> \"%s\" [ label = \"%d\" , arrowhead = none ]\n",
           author_search, author, (int)count);
       g_free(author);
       return FALSE;
}

gboolean printCategories(gpointer category, gpointer category_info, gpointer data) {

       g_fprintf(out, "<a href=\"%s.html\">%s</a> = %d\n<br>\n",
              g_ascii_strdown(category,-1),
              g_ascii_strdown(category,-1),
              (((catInfo)category_info)->count));
```

```
	return FALSE;
}

gboolean printAuthors(gpointer author, gpointer author_info, gpointer data) {

	GArray* list = ((autInfo)author_info)->list_keys;
	int count = ((autInfo)author_info)->count_keys;

	g_fprintf(out, "Autor: %s\n<br>\n", author);

	g_fprintf(out, "<ul>\n");
	for (int i = 0; i < count; i++)
	{
		g_fprintf(out, "<li>%s</li>\n", g_array_index(list, gchar*, i));
		g_free( g_array_index(list, gchar*, i) );
	}
	g_fprintf(out, "</ul>\n");

	g_array_free(list, FALSE);
	free((autInfo)author_info);

	return FALSE;
}

gboolean closeFiles(gpointer category, gpointer category_info, gpointer data) {

	FILE* file = ((catInfo)category_info)->out;
	closeHTML(file);
	fclose(file);
	free(category_info);
	return FALSE;
}

void addCategory(char* category, int catleng, GTree *t) {

	gpointer* value = g_tree_lookup(t, category);

	if (!value) {

		catInfo category_info = malloc(sizeof(struct Category));
		char filename[100];

		sprintf(filename, "%s.html", g_ascii_strdown(category,-1));

		category_info->count = 1;
		category_info->out = fopen(filename, "w");

		out = category_info->out;
		openHTML(out);

		g_tree_insert(t, g_strdup(category), category_info);
	}
	else {
		((catInfo)value)->count++;
		out = ((catInfo)value)->out;
	}
}
```

```
void addKeyToAuthor(gchar* author, gchar* key, GTree * t) {

        gchar * keyval = g_strdup(key);
        gpointer* value = g_tree_lookup(t, author);

        if (!value) {

                autInfo author_info = malloc(sizeof(struct Author));

                author_info->list_keys = g_array_new(FALSE, FALSE, sizeof(gchar*));
                author_info->count_keys = 1;

                g_array_append_val(author_info->list_keys, keyval);

                g_tree_insert(t, g_strdup(author), author_info);
        }
        else {
                ((autInfo)value)->count_keys++;
                g_array_append_val(((autInfo)value)->list_keys, keyval);
        }
}

void searchAuthor(GArray* list_aut, int size, GTree* t) {

        int found = 1, pos;

        for (pos = 0; pos < size && found != 0; pos++)
                found = g_ascii_strcasecmp(author_search, g_array_index(list_aut, gchar*, pos
                    ));

        pos--;

        if (found == 0)
        {
                for (int i = 0; i < size; i++)
                        if (i != pos) {
                                gchar * aut = g_array_index(list_aut, gchar*, i);
                                gpointer* value = g_tree_lookup(t, aut);

                                if (!value)
                                        g_tree_insert(t, aut, (gpointer*)1);
                                else
                                        g_tree_replace(t, aut, (int)value + 1);
                        }
        }
        else
                for (int i = 0; i < size; i++)
                        g_free( g_array_index(list_aut, gchar*, i) );

        g_array_free(list_aut, FALSE);
}

%}

space  [\ \t]+
alpha  [a-zA-Z]
author  (?i:author){space}?\={space}?(\"|\{)
```

```
title   [^a-zA-Z](?i:title){space}?\={space}?(\"|\{)
and     [^a-zA-Z0-9]and[^a-zA-Z0-9]

%%

        GTree* cat_tree = g_tree_new((GCompareFunc)g_ascii_strcasecmp);
        GTree* aut_tree = g_tree_new((GCompareFunc)g_ascii_strcasecmp);
        GTree* list_aut_tree = g_tree_new((GCompareFunc)g_ascii_strcasecmp);
        int delim = 0, pos = 0, size_list_aut;
        gchar* curr_author = malloc(sizeof(gchar) * 100);
        gchar* curr_title = malloc(sizeof(gchar) * 200);
        gchar* curr_key = malloc(sizeof(gchar) * 50);
        GArray* list_aut;

\@string\{.*\n              { }

"@"                         BEGIN CAT;

<STR>{
        "}"                 BEGIN 0;
        .|\n                { }
}


<CAT>{
        [a-zA-Z0-9]+        { addCategory(yytext, yyleng, cat_tree); }
        "{"                 BEGIN KEY;
        .|\n                { }
}

<KEY>{
        [^\,]+              { fprintf(out, "Chave: %s\n<br>\n", yytext);
                              g_stpcpy(curr_key, yytext);
                            }
        ","                 { delim = 0; BEGIN CONTENT; }
}

<CONTENT>{
        {author}            { fprintf(out, "Autores:\n<ul>\n");
                              list_aut = g_array_new(FALSE, FALSE, sizeof(gchar*));
                              size_list_aut = 0;
                              BEGIN AUT;
                            }
        {title}             BEGIN TIT;
        \{                  delim++;
        \}                  { if (delim > 0)
                                    delim--;
                              else {
                                    fprintf(out, "<hr>\n");
                                    BEGIN 0;
                              }
                            }
        .|\n                { }
}

<AUT>{
        {and}               { curr_author[pos] = '\0';
                              curr_author = g_strstrip(curr_author);
```

```
                        addKeyToAuthor(curr_author, curr_key, aut_tree);

                        if (author_search != NULL) {
                              gchar* author = g_strdup(curr_author);
                              g_array_append_val(list_aut, author);
                              size_list_aut++;
                        }

                        g_fprintf(out, "<li>%s</li>\n", curr_author);
                        pos = 0;
                      }
\n|\t                 { }
\\\"                  { curr_author[pos++] = yytext[0];
                        curr_author[pos++] = yytext[1];
                      }
"{"                   { delim++;
                        curr_author[pos++] = yytext[0];
                      }

"}"                   { if (delim > 0) {
                              delim--;
                              curr_author[pos++] = yytext[0];
                        }
                        else {
                              curr_author[pos] = '\0';
                              curr_author = g_strstrip(curr_author);

                              addKeyToAuthor(curr_author, curr_key, aut_tree);

                              if (author_search != NULL) {
                                    gchar* author = g_strdup(curr_author);
                                    g_array_append_val(list_aut, author);
                                    size_list_aut++;

                                    searchAuthor(list_aut, size_list_aut,
                                        list_aut_tree);
                              }

                              g_fprintf(out, "<li>%s</li>\n</ul>\n", curr_author);
                              pos = 0;
                              BEGIN CONTENT;
                        }
                      }
\"                    { if (delim > 0)
                              curr_author[pos++] = yytext[0];
                        else {
                              curr_author[pos] = '\0';
                              curr_author = g_strstrip(curr_author);

                              addKeyToAuthor(curr_author, curr_key, aut_tree);

                              if (author_search != NULL) {
                                    gchar* author = g_strdup(curr_author);
                                    g_array_append_val(list_aut, author);
                                    size_list_aut++;
```

```
                                        searchAuthor(list_aut, size_list_aut,
                                            list_aut_tree);
                                    }

                                    g_fprintf(out, "<li>%s</li>\n</ul>\n", curr_author);
                                    pos = 0;
                                    BEGIN CONTENT;
                                  }
                                }
        .                       curr_author[pos++] = yytext[0];
}

<TIT>{
        \n|\t                   { }
        \\\"                    { curr_title[pos++] = yytext[0];
                                  curr_title[pos++] = yytext[1];
                                }
        "{"                     { delim++;
                                  curr_title[pos++] = yytext[0];
                                }
        "}"                     { if (delim > 0) {
                                        delim--;
                                        curr_title[pos++] = yytext[0];
                                  }
                                  else {
                                        curr_title[pos] = '\0';
                                        curr_title = g_strstrip(curr_title);
                                        g_fprintf(out, "Título: %s\n<br>\n", curr_title);
                                        pos = 0;
                                        BEGIN CONTENT;
                                  }
                                }
        \"                      { if (delim > 0)
                                        curr_title[pos++] = yytext[0];
                                  else {
                                        curr_title[pos] = '\0';
                                        curr_title = g_strstrip(curr_title);
                                        g_fprintf(out, "Título: %s\n<br>\n", curr_title);
                                        pos = 0;
                                        BEGIN CONTENT;
                                  }
                                }
        .                       curr_title[pos++] = yytext[0];

}

<<EOF>> {                       out = fopen("index.html", "w");
                                openHTML(out);
                                g_tree_foreach(cat_tree, (GTraverseFunc)printCategories, NULL);
                                closeHTML(out);
                                g_tree_foreach(cat_tree, (GTraverseFunc)closeFiles, NULL);
                                fclose(out);

                                out = fopen("authorindex.html", "w");
                                openHTML(out);
                                g_tree_foreach(aut_tree, (GTraverseFunc)printAuthors, NULL);
                                closeHTML(out);
                                fclose(out);
```

```
                                if (author_search != NULL) {
                                        out = fopen("graph.dot", "w");
                                        openGraph(out);
                                        g_tree_foreach(list_aut_tree, (GTraverseFunc)printGraph,
                                             NULL);
                                        closeGraph(out);
                                }

                                g_free(curr_key);
                                g_free(curr_title);
                                g_free(curr_author);

                                g_tree_destroy(list_aut_tree);
                                g_tree_destroy(cat_tree);
                                g_tree_destroy(aut_tree);

                                fclose(out);
                                yyterminate();
}

.|\n                            { }

%%

int main(int argc, char** argv){

        ++argv, --argc; /* skip over program name */
        if ( argc > 0 ) {
                yyin = fopen( argv[0], "r" );
                if (argc > 1) {
                        author_search = malloc( sizeof(gchar) );
                        g_stpcpy( author_search, argv[1] );
                }
        }
        else
                yyin = stdin;


        yylex();
}
```

6 Referências

- www.bibtex.org
- https://developer.gnome.org/glib/2.56/
- https://www.graphviz.org/
- https://www.ibm.com/developerworks/linux/tutorials/l-glib/index.html