

Processamento de Linguagens (3º ano de MiEI)

Trabalho Prático 1

Relatório de Desenvolvimento de um Normalizador de ficheiros *BibTex*

Gustavo da Costa Gomes-Aluno
(72223)

José Carlos da Silva Brandão Gonçalves-Aluno
(71223)

Tiago João Lopes Carvalhais-Aluno
(70443)

5 de Abril de 2016

Resumo

Isto é um resumo do relatório da unidade curricular Processamento de Linguagens relativamente ao Trabalho Prático 1. Este visa a produção de um Normalizador de ficheiros *BibTex* permitindo a exploração da ferramenta *Flex* acompanhada de uma pequena demonstração de quão poderosa realmente é. Também é possível encontrar numa seção, no capítulo "Anexo", que possui a resolução de um segundo enunciado, que é a conversão de ficheiros *Zim-Wiki* para slides *Beamer*.

Conteúdo

1	Introdução	2
2	Análise e Especificação	3
2.1	Descrição informal do problema	3
2.2	Especificação do Requisitos	4
2.2.1	Dados	4
3	Concepção/desenho da Resolução	5
3.1	Estruturas de Dados	5
3.1.1	Alínea a	5
3.1.2	Alínea b	6
3.1.3	Alínea c	7
4	Codificação	8
4.1	Alternativas, Decisões e Problemas de Implementação	8
4.1.1	Alínea a	8
4.1.2	Alínea b	10
4.1.3	Alínea c	13
4.2	Testes realizados e Resultados	14
4.2.1	Alínea a	14
4.2.2	Alínea b	15
4.2.3	Alínea c	16
5	Conclusão	18
A	Código do Programa	19
A.1	Alínea a	20
A.2	Alínea b	25
A.3	Alínea c	36
A.4	Extra- Zim to Beamer	39

Capítulo 1

Introdução

Este trabalho envolveu o desenvolvimento de um Normalizador de ficheiros *BibTex*, um dos enunciados disponibilizados, no qual se procede á sua análise tendo em conta os seguintes pontos,

Enquadramento Utilização de Expressões Regulares e Filtros de Texto com o objetivo de produzir novos documentos a partir de padrões existentes num outro ficheiro.

Estrutura do documento Este documento possui um anexo com todo o código produzido em cada uma das alíneas, uma conclusão final que une o exercício e as respectivas soluções elaboradas e ainda capítulos elucidativos de cada tarefa a desempenhar em cada alínea.

Resultados Os resultados serão apreciados nos respectivos capítulos correspondentes a cada uma das alíneas desenvolvidas e cujo ficheiro(ou partes dele) estará(ão) no Anexo correspondente.

Conteúdo do documento Contém a explicação do problema em si, bem como a apresentação das soluções produzidas para colmatar essa situação, auxiliada com documentação e código produzido, presente nos Anexos.

Estrutura do Relatório

Este relatório possui quatro capítulos, uma conclusão, um capítulo extra dedicado a Anexos e a respectiva Bibliografia utilizada durante a realização deste projecto. Os capítulos são Análise e Especificação do problema, Especificação dos Requisitos, Arquitectura da solução para cada um dos sub-problemas indicados no enunciado global que envolverá a explicação das estruturas de dados utilizadas e por fim o capítulo designado Codificação, que incluirá alguns aspectos relevantes de todos os testes realizados para a verificação do correcto funcionamento.

Capítulo 2

Análise e Especificação

2.1 Descrição informal do problema

BibTex é uma ferramenta de formatação de citações bibliográficas em documentos *Latex*, que separa a bibliografia consultada do restante conteúdo. Um exemplo desse ficheiro, com a extensão .bib ilustra-se de seguida,

```
1 @InProceedings{CPBFH07e,
2   author = {Daniela da Cruz and Maria Joao Varanda Pereira
3             and Mario Beron and Ruben Fonseca and
4             Pedro Rangel Henriques},
5   title = {Comparing Generators for Language-based Tools},
6   booktitle = {Proceedings of the 1.st Conference on Compiler
7 }
8   year =
9   editor =
10  month =
11  Related Technologies and Applications , CoRTAâ07
12  — Universidade da Beira Interior , Portugal},
13 {2007},
14 {},
15 {Jul},
```

Este enunciado consiste em três alíneas, nas quais são requeridas diferentes tarefas a implementar. Na alínea a é pedido que se elabore um documento *HTML* que contenha a contagem de todas as diferentes categorias presentes no documento lpbib.txt, explicado num capítulo vindouro, *Código do Programa*.

Na alínea b é pedido que se desenvolva uma ferramenta de normalização que faça *pretty-printing*, fazendo a indentação correta em cada campo, escrevendo cada autor numa linha e que coloque no início da mesma os campos autor e título, e quando um campo estiver entre aspas modifique para chavetas e se escreva o nome dos autores com o seguinte formato, *N.Apelido*.

Por fim, na alínea c é pedido a construção de um grafo que para um determinado autor, á escolha do utilizador, mostre todos os autores que publicaram com esse autor. Para tal, recorrer-se-á á linguagem Dot do *Graph Viz2*, gerando um ficheiro com esse grafo de modo a que possa, posteriormente, desenhar o mesmo através de uma outra ferramenta que faça a leitura desses ficheiros.

2.2 Especificação do Requisitos

Neste trabalho, o objectivo é estimular a utilização de um ambiente *Linux*, da linguagem imperativa *C* e de outras ferramentas de apoio para a resolução de problemas de um modo diferente do habitual, que seria tentar resolver tudo apenas utilizando uma linguagem de programação e, para tal, visam o estudo e o desenvolvimento de Expressões Regulares, bem como, a sua manipulação por forma a atingir o resultado pretendido. Essas expressões são fundamentais para encontrar os padrões para os quais se irá tomar uma ação, que será a transformação do texto, filtrando ou removendo esses. Como auxiliar na realização de filtros de texto recorrer-se-á à utilização de um gerador designado, *Flex*.

Na realização deste problema é necessário concluir uma lista de tarefas que são, especificar os padrões de frases que se quer encontrar no texto fonte, através de Expressões Regulares, identificar as acções semânticas a realizar como reacção ao reconhecimento de cada um desses padrões, identificar as estruturas de dados globais que possa eventualmente precisar para armazenar temporariamente a informação que se vai extraindo do texto fonte ou que se vai construindo à medida que o processamento avança e por fim desenvolver um filtro de texto para fazer o reconhecimento dos padrões identificados e proceder à transformação pretendida, com recurso ao gerador *Flex*.

2.2.1 Dados

Os dados fornecidos são o ficheiro *lpbib.txt*, a ser abordado num capítulo posterior, a definição e utilização da ferramenta *Flex* e a definição de um ficheiro *BibTex*, isto é, as suas características. Para além destes, na alínea c) o ficheiro de *input* utilizado é referente ao resultado da alínea b) e é tem o nome de *graphTeste.txt*.

É também fornecido o nome de ferramentas de apoio à resolução do problema, sendo neste problema, a ferramenta *GraphViz2*, que permitirá colocar graficamente a informação dos grafos criados, sendo que as interações entre os autores se tornam mais perceptíveis.

Capítulo 3

Concepção/desenho da Resolução

3.1 Estruturas de Dados

3.1.1 Alínea a

Nesta alínea optou-se por utilizar uma *hashtable* como estrutura de dados auxiliar que vai guardando as categorias e o respectivo contador á medida que se vai encontrando um padrão no ficheiro lpbib.txt, ou seja, a acção ao padrão, que permite encontrar as categorias todas ao longo de todo o conteúdo.

Esta estrutura segue a lógica de *Open Addressing*, isto é, através de uma função de *hash* é encontrada a posição onde se irá inserir a categoria capturada pela expressão regular e no caso de essa estar já ocupada vai tentar inserir na posição seguinte, e se chegar á última reinicia, visto que é *circular*. Apenas se implementou funções essenciais, *inserir*, *remover*, *procurar* e *imprimir* o conteúdo desta estrutura e gerar o conteúdo do ficheiro *HTML* pedido. Para verificar se uma posição já possui ou não conteúdo basta verificar a *etiqueta* associada e designada por *state*, no Anexo encontra-se o conteúdo integral da implementação desta estrutura de dados.

3.1.2 Alínea b

Neste problema só é preciso guardar dados para ordenar os campos de forma a colocar o campo *author* em primeiro lugar, seguido do campo *title*. Uma estrutura de dados simples como um array de *strings* ou mais especificamente em contexto C um array de *char** seria o suficiente para ordenar os campos. No entanto, foi optado por utilizar duas *structs* que contenham toda a informação de um registo para resolver o problema. Esta opção, para além de permitir um maior controlo na formatação final do registo também simplifica o código em flex.

As duas *structs* estão definidas no ficheiro *prettybibtex.h*. A struct *registo* contém toda a informação de um registo. O campo *categoria* é usado para guardar o nome da categoria, da mesma maneira o campo *chave* guarda a chave. Apesar da informação dos autores ser considerada mais um campo de uma entrada *BibTeX* (não confundir com campos da struct registo) foi guardada num array de *strings* separando-se dos restantes campos. Esta distinção tem duas vantagens significativas, em primeiro lugar a divisão do conteúdo do campo *author* num array de *strings* permite guardar o nome formatado de cada autor numa string diferente, isto facilita bastante a tarefa de imprimir um autor em cada linha. A segunda vantagem é que facilita a função de ordenação dos campos, assim só é preciso colocar o campo *title* em primeiro lugar do array de *Campo*. Esta componente é um array de um apontador para a struct *campo* que contém um par de *strings* com o nome do campo a guardar e o seu respectivo conteúdo.

```
typedef struct registo{
    char* categoria;
    char* chave;
    char** autores;
    int iautores;
    Campo* campos;
    int r;
}*Registo;

typedef struct campo{
    char* nome;
    char* valor;
}*Campo;
```

Esta estruturada de dados é reutilizada a cada leitura de uma entrada *BibTeX* de modo a o programa não utilizar mais memória do que efetivamente necessita. É de notar que se a memória da estrutura não fosse devidamente libertada e/ou reutilizada, o programa para ficheiros *BibTeX* de elevada dimensão ocuparia uma elevada quantidade de memória possivelmente capaz de ultrapassar a memória da heap atribuída pelo *Sistema Operativo* e terminar interruptamente.

3.1.3 Alínea c

Nesta alínea, optou-se pela utilização de duas listas ligadas, uma para guardar os autores que publicaram com o autor fornecido e outra para guardar os autores de uma publicação para, posteriormente, verificar se o autor principal lá estava contido e, em caso afirmativo, guardar-se na primeira estrutura os outros autores, sendo que, caso um autor já exista na estrutura principal, então incrementava um contador que indica quantas vezes esse autor fez publicações em conjunto com o principal.

O grupo optou por escolher esta estrutura de dados, uma vez que é bastante simples de implementar, tendo como uma das desvantagens a pouca eficiência na procura e inserção de informação. Contudo, como trabalhamos com pouca quantidade de dados (em termos computacionais), chegámos à conclusão que as desvantagens da lista ligada não se iriam fazer sentir na execução do programa.

Capítulo 4

Codificação

4.1 Alternativas, Decisões e Problemas de Implementação

4.1.1 Alínea a

Um dos problemas de implementação passou por conseguir contabilizar as categorias de forma independente, isto é, após uma análise do documento fonte `lpbib.txt` verificou-se a existência de categorias que contem exatamente os mesmos caracteres mas escritos de diferentes formas. Um exemplo disso é a categoria *inproceedings* que pode também se encontrar como *InProceedings* e como *INPROCEEDINGS*. Então o grande desafio foi separar esta categoria em três, porque apesar de terem os mesmos caracteres, elas representam a mesma categoria mas de forma independente visto que na contagem destas se pretende ter um resultado que mostre o que realmente está no ficheiro fonte.

Na fase de implementação surgiram pequenos problemas relacionados com a expressão regular que foi definida por forma a filtrar apenas a categoria. Visto que essa estava delimitada por dois caracteres, o '@' e o '. Por vezes *yytext* não continha o conteúdo correto, o que revelava que a expressão regular ainda não estava a funcionar corretamente.

Após esse problemas estarem resolvidos conseguiu-se produzir o ficheiro *HTML* sem nenhuma dificuldade, apenas se efetuou a impressão do conteúdo da estrutura de dados que foi armazenando as categorias e atualizando os seus contadores com a indentação e os *headers* que permitem visualizar o ficheiro obtido num *browser*. Esse ficheiro *HTML* produzido chama-se `indexA.html` e pode ser visto no Anexo A.1, bem como o filtro de texto produzido, recorrendo á ferramenta *Flex*, neste documento.

Para terminar falta proceder á análise das expressões regulares utilizadas e a respectiva acção a tomar quando estas forem encontradas.

- i. `@[a?zA?Z]+\{`
- ii. `.\|\\n`

A expressão regular ii. é para filtrar todo o texto, mas é absorvente e por isso é preciso muito cuidado com a sua utilização e ,associado a esta, a acção de fazer *print* que é a acção por defeito, quando se fornece

`}`

A expressão regular i. é a expressão que foi desenvolvida para a resolução do problema pedido, contagem das categorias, que exige inicialmente a captação das categorias e apenas das categorias presentes no ficheiro `lpbib.txt`, que contem muita mais informação. O objectivo desta é encontrar todos os padrões que estejam contidos entre os caracteres '@' e '.', visto que essa é a definição de categoria num ficheiro *BibTex*. E como as categorias apenas podem conter letras temos de restringir os padrões encontrados entre esses dois caracteres ao facto de que só podem ter letras, quer minúsculas quer maiúsculas, daí `'[a?zA?Z]+'`. O símbolo '+' refere-se á possibilidade de encontrar uma

ou mais ocorrências, isto é, entre esses dois caracteres encontra-se apenas uma ou mais letras. Visto que não existe nenhuma restrição quanto á forma da palavra categoria, isto é, por exemplo a exigência de começar por letra maiúscula e seguida apenas de letras minúsculas não é necessário efetuar mais nenhuma restrição ao padrão que permitirá filtrar as categorias.

Associado a esta, última expressão regular, está a acção de copiar o conteúdo de *yytext+1*, que apenas considera o texto após o '@' até *yytext-2*, onde está o '' para um *char** local que será então inserido na estrutura de dados através da instrução *insertTable(ht, str, (int ?) count)* ; sendo que *count* é um contador inicial que apenas serve para iniciar o contador na estrutura de dados. Esta instrução está codificada por forma a verificar logo se a categoria já existe ou não, e caso exista apenas incrementa a ocorrência dessa categoria e ignora o parâmetro *count* recebido. E caso não exista procede então ao início do contador com valor de *count* recebido e insere na estrutura de dados.

Por fim é necessário criar o ficheiro *HTML* pretendido com o conteúdo da estrutura de dados que foi sendo atualizada até se chegar ao fim do ficheiro *lpbib.txt* e para tal na função *main* do ficheiro *tp1A.l* recorreu-se á chamada da função *printHashTable (ht)* ; que foi codificada no ficheiro *hashtable.c* por forma a criar o ficheiro *HTML* com a formatação necessária, produzindo, desse modo, o resultado final desta alínea.

4.1.2 Alínea b

A codificação deste programa acarretou alguns desafios. Inicialmente, o grande desafio foi filtrar os campos delimitados por chavetas e aspas, no entanto os dois problemas foram resolvidos de uma maneira simples e elegante. Outro grande desafio foi a leitura e formatação do conteúdo do campo *author*, este provou-se ser um problema complexo e a sua solução não tão elegante e simples como as anteriores. Isto apenas porque o conteúdo do campo pode aparecer em 2 formatos distintos mais um conjunto de casos particulares.

Para a resolução do problema foram criadas 6 variáveis de condição cujo propósito de cada uma será explicado ao longo da secção.

O estado *INITIAL* tem como função identificar as entradas de *BibTeX* com registos de interesse, estes são os que contêm referências bibliográficas. O *BibTeX* para além das entradas de interesse contem 3 tipos de entradas *@string*, *@preamble* e *@comment*. O ficheiro de dados deste trabalho não contem as duas últimas entradas e, portanto, a gramática não precisou de regras para lidar com as mesmas. No entanto, para a gramática funcionar com as mesmas, basta adicionar duas regras que identifiquem as entradas e as imprimem. Essas regras e respetivas ações seriam quase idênticas à primeira regra para identificar a entrada *@string*. Por exemplo, para apanhar a entrada *@preamble* bastaria adicionar a regra `@[Pp][Rr][Ee][Aa][Mm][Bb][Ll][Ee]{` com a ação *ECHO*. Ignorando as entradas não relevantes para o objetivo do problema, temos todas as categorias restantes iniciadas também por '@' seguidas pelo nome da categoria contendo uma sequência de uma ou mais letras maiúsculas ou minúsculas `[A-Za-z]+` finalizadas com o símbolo `''`. Dada a ocorrência desse padrão é inicializada a estrutura *Registo* que conterá a informação da entrada, e é já adicionado à estrutura o nome da categoria. Após lida a categoria, vem a leitura da chave no estado *CHAVE*.

```
<INITIAL>@[Ss][Tt][Rr][Ii][Nn][Gg]\{ { ECHO;}
<INITIAL>@[A-Za-z]+\{ {
registo = initRegisto();
yytext[yytext - 1] = '\0'; yytext++;
registo->categoria = strdup(yytext);
BEGIN CHAVE;}
<INITIAL>.\|n { ECHO;}
```

A chave está obrigatoriamente separada do próximo campo através do símbolo `''` e possivelmente contem espaços no início e fim. A regra `.—`limpa esses espaços e a regra `[A-Za-z0-9:]+` filtra o id da chave. Sabe-se que a chave pode conter letras, números e o símbolo `':'`, no entanto, se essa informação fosse desconhecida ou a chave abrangesse um conjunto maior de símbolos, a expressão regular poderia ser substituída por algo mais genérico, como por exemplo:

```
[^ ]+/,
```

Esta expressão apanharia todos os símbolos excepto o espaço até apanhar o símbolo delimitador `''`, não consumindo o mesmo, uma vez que não faz parte da chave. Gravada a chave na estrutura *registo* é inicializado o estado *CAMPOS*.

```
<CHAVE>[A-Za-z0-9:]+ {registo->chave = strdup(yytext);BEGIN CAMPOS;}
<CHAVE>.\|n {}
```

O estado *campos* tem uma função apenas que é identificar se o registo acabou ou se há mais campos. O registo acaba se aparecer depois do último campo o símbolo `''`, o último campo pode ser finalizado com uma `''` ou não. Entre a `''` opcional e a `''` de finalização pode aparecer os símbolos `''` e `''`. No final da entrada, é invocada a função `'printRegisto'` para imprimir a *struct registo* com a informação da entrada na sua totalidade, a função também liberta a memória alocada. Caso a chaveta de finalização não apareça significa que há mais campos e dá-se a leitura do nome do próximo campo no estado *NOME CAMPO*.

```
<CAMPOS>(",")\n \t]*\{ { printRegisto(registo); BEGIN INITIAL;}
<CAMPOS>\, \n \t]* {BEGIN NOME CAMPO;}
```

O nome do campo pode conter letras e o símbolo '-' e é obtido pela ER `[a-zA-Z]+`. O separador entre o nome do campo e o seu conteúdo é o símbolo '=', quando este é filtrado dá-se início à leitura do conteúdo do campo no estado `VALORCAMPO`. O campo *author* é tratado de maneira diferente, uma vez que um dos objetivos do trabalho é formatar nomes. Este estado identifica o campo *author* e dá início ao estado `AUT` que irá processar o conteúdo do campo de forma diferente dos restantes. Para além disso, o estado aproveita para facilitar o trabalho do estado `AUT`, cobrindo o caso do conteúdo do campo *author* estar vazio e o caso do campo conter apenas um único nome.

```
<NOMECAMPO>[ ]*=[ ]* { BEGIN VALORCAMPO; }
<NOMECAMPO>[a-zA-Z-]+
```

```
<NOMECAMPO>[ ]*[Aa][Uu][Tt][Hh][Oo][Rr][ ]*=[ ]*["{}][ ]*["}] // Caso vazio
<NOMECAMPO>[ ]*[Aa][Uu][Tt][Hh][Oo][Rr][ ]*=[ ]*["{} //Dá início a AUT
<NOMECAMPO>[ ]*[Aa][Uu][Tt][Hh][Oo][Rr][ ]*=[ ]*["{}[ ]*[^ ]+[ ]*["}]/, //Único nome
```

O conteúdo do campo está sempre delimitado por aspas ou chavetas, com a exceção no caso do conteúdo ser um número ou uma única palavra. Esse caso é tratado pela expressão `[a-z][A-Z][0-9]+`. Se o campo estiver delimitado por aspas o conteúdo do campo pode conter todos os símbolos possíveis. O delimitador "" pode aparecer também no próprio campo, no entanto este tem de ser precedido pelo símbolo \; aparecendo sempre \". Devido à elevada quantidade de símbolos possíveis o problema só pode ser resolvido através da negação do conjunto de símbolos não permitidos. Portanto, o conteúdo do campo é tudo excepto o delimitador "" com a exceção do aparecimento do símbolo \" que resulta na expressão regular

```
\"([^\"](\"\\\"\"?))*\"
```

Quando o campo está delimitado por chavetas a mesma técnica não pode ser utilizada uma vez que o delimitador final \" pode aparecer livremente no conteúdo. A única restrição é que sempre que é aberta uma chaveta tem de aparecer outra a fechar. O estado identifica o delimitador \" e dá início à variável de condição `VALORCAMPOT2`. O problema é resolvido nesse estado através de um contador de chavetas abertas. Esse estado finaliza quando encontrar o símbolo \" e o número de chavetas abertas ser 1, o que significa que o símbolo \" está a fechar a chaveta inicial do campo. Finalizada a leitura do conteúdo do campo é invocado o estado `CAMPOS` para verificar se há mais campos no registo ou se o registo acabou.

```
<VALORCAMPO>[a-zA-Z0-9]+
<VALORCAMPO>\"([^\"](\"\\\"\"?))*\"
<VALORCAMPO>{
```

Este estado tem como função filtrar os nomes individuais dos autores para formatar os nomes no formato \"N. Apelido\". Para entender a solução é necessário conhecer as formatações dos nomes dos autores. A formatação 1 consiste nos nomes completos ou primeiro e ultimo nome de cada autor separados por \"and\", por exemplo \"Nome Nome Apelido and Nome Apelido\". Na formatação 2 os nomes dos autores também são separados por \"and\", no entanto o apelido do autor vem sempre em primeiro lugar, separado com uma \",\" do primeiro e restantes nomes, por exemplo \"Apelido, Nome Nome\". O estado identifica o final do campo através do padrão \",', ou ','. Uma vez que o símbolo \" não é um bom delimitador, visto que os nomes podem conter chavetas o que obrigaria à contagem de chavetas abertas como feito nos outros campos. Todavia tal método aumentaria significativamente a complexidade da solução. A solução do problema encontra-se explicada detalhadamente de seguida, cada regra contém uma pequena explicação e a sua ação é explicada.

```
// Regra identifica o final do nome completo de um autor
<AUT>[ ]+[Aa]nd[ ]+ {
    guarda o nome e o apelido do autor
    reinicializa as variáveis de controlo
temApelido = 0; nomeCompleto = 0; primeiroNomeParsed = 0;
}
// Identifica o final do campo
```

```

<AUT>["}], [ ]*\n {
    guarda o nome e o apelido do último autor e salta para o estado CAMPOS
}
// Apanha o último nome do campo e não deixa consumir o padrão de saída do estado
<AUT>([^\n]("\\\\"?))+/, " {
    É lido o último nome ou apelido do registo e é invocada a função formatAuthor que lida com o nome conforme
    formatAuthor(); // explicado no final
}
// Identifica a formatação 2 através do padrão "Apelido,"
<AUT>([^\n]("\\\\"?))+/, {
    identifica o formato 2
    guarda o apelido
}
// Apanha os restantes nomes
<AUT>([^\n]("\\\\"?))+ {
    É lido um nome ou apelido e é invocada a função formatAuthor que lida com o nome conforme as variáveis de c
    formatAuthor();
}
<AUT>.\n {
    limpa espaços, e vírgulas no caso da formatação 2 */
}

formatAuthor(){
    Dado um nome e o estado das variáveis de controlo, comporta-se de 3 maneiras diferentes.
    Na situação do formato2 e o nome do autor não estar completo(ter 2 nomes), identifica o nome como primeiro
    Na situação formato1 e primeiro nome não lido, guarda o nome como primeiro nome e ativa flag 'primeiroNomeF
    Na situação formato1 e primeiro nome lido, guarda o nome como apelido. É de notar que guarda o nome sempre
}

```

4.1.3 Alínea c

Inicialmente, o problema incidiu no pouco conhecimento do grupo acerca da linguagem *Dot*, pelo que foi necessário efetuar um estudo aprofundado sobre a linguagem em questão. Posteriormente, houve uma grande dificuldade em conseguir captar informação do ficheiro *lpbib.txt*, uma vez que alguns caracteres desapareciam e, obviamente, ao fazer depois a comparação com o autor principal, este nunca surgia e, como tal, o grafo resultante seria sempre incorreto. Mas, como o trabalho de filtrar os autores e mudar os seus nomes para o formato "N. Apelido", já havia sido feito, então o grupo optou por receber como input o resultado da alínea b), que foi guardado no ficheiro *graphTeste.txt*. Desta forma, como os autores estão todos no mesmo formato, sendo que anteriormente uns estavam no formato "Nome Apelido" e outros em "Apelido, Nome", foi mais fácil agora proceder à recolha dos mesmos, aproveitando o trabalho efetuado anteriormente. No Anexo encontra-se um exemplo de um ficheiro *.dot* e o grafo resultante do mesmo. Convém referir que, para proceder à execução desta alínea, deve-se exutar o comando *make ARGS="N. Autor"*. Desta forma o programa irá executar uma *Makefile*, que vai executar o programa com o argumento dado.

Iremos proceder, agora, a uma breve análise às expressões regulares utilizadas nesta alínea e respetivas ações.

```
i. author[ ]*=[ ]*{" {linha = NULL; BEGIN AUTOR;}
ii. <AUTOR>[\t" "\n]+ {}
iii. <AUTOR>[^\\t\\n]*\\/\\}, " {char *authorName = strdup(yytext); linha = insert(linha, authorName);}
iv. <AUTOR>"\\},\\n" {autores = getAutores(autores, linha, author); BEGIN INITIAL;}
v. <AUTOR>[^\\t\\n]+ {char *authorName = strdup(yytext); linha = insert(linha, authorName);}
vi. .|\\n {}
vii. <*><<EOF>>{printAutores(); free(linha); free(autores); return 0;}
```

A primeira expressão corresponde à situação em que se encontra o campo "author" de um registo. O programa inicializa a estrutura de dados referente aos autores encontrados num registo e entra no estado "AUTOR". A segunda expressão ocorre já dentro do estado "AUTOR" e serve apenas para limpar caracteres que estejam a mais, ou seja, espaços, TAB's ou mudanças de linha. A terceira expressão apanha qualquer carater, exceto mudanças de linha e TAB's, eliminando as aspas, até encontrar uma chaveta para fechar o campo, apanhando assim o nome do autor (dado por *yytext*) e, de seguida efetua a inserção do mesmo na estrutura *linha*. A quarta expressão regular ocorre logo após a anterior e efetua a chamada da função *getAutores*, que verifica se um autor existe na *linha* e, em caso afirmativo, insere os restantes autores na estrutura principal (de nome *autores*) e, de seguida, abandona o estado "AUTOR". A quinta expressão efetua o mesmo trabalho da terceira, mas serve para autores que não sejam os últimos da lista, tendo as mesmas ações. A sexta expressão serve para, quando no estado inicial, ignorar qualquer carater ou mudança de linha. Convém notar que, como esta expressão regular vem depois das anteriores, não serão ignorados os caracteres importantes, nomeadamente, os presentes no campo "author". Por fim, a última expressão ocorre aquando do final do ficheiro, sendo que aí serão imprimidos os autores no ficheiro *graph.dot* e efetuadas as devidas limpezas na memória, devido às estruturas criadas.

Após a criação do ficheiro *graph.dot*, será executado um comando que irá criar um ficheiro de imagem com o grafo resultante. Convém referir, mais uma vez, que todos os comandos importantes estão contidos numa *Makefile*.

4.2 Testes realizados e Resultados

Mostram-se a seguir alguns testes feitos (valores introduzidos) e os respectivos resultados obtidos: lpbib.txt nas diferentes alíneas deste problema.

4.2.1 Alínea a

Para este problema fizeram-se testes sucessivos até encontrar a expressão regular que permitisse armazenar apenas a categoria na estrutura de dados evitando desse modo que a manipulação de *strings* fosse feita através de funções definidas na linguagem *C*, tal como, *strtok* ou outra semelhante, uma vez que o objectivo é a utilização de expressões regulares que façam todo o trabalho de manipulação de texto.

Os testes realizados passaram todos por correr a seguinte *makefile*

```
1 all: 1 2
2
3 1:
4         flex tp1.l
5
6 2:
7         gcc lex.yy.c hashtable.c -ll -o tp1
```

E de seguida efetuar

`./tp1 < lpbib.txt`

E verificando se o resultado obtido era realmente o resultado pretendido. Antes de se chegar ao resultado correcto teve-se de corrigir a situação de contagem de categorias com os mesmos caracteres mas que representavam categorias distintas porque o que acontecia era, para cada uma dessas categorias o contador estava a incluir as outras, mas só mostrava as ocorrências de uma dessas categorias. Por exemplo, para a categoria *INPROCEEDINGS* verifica-se que no ficheiro fonte lpbib.txt apenas ocorre cinco vezes, mas o que acontecia era que se se imprimisse o conteúdo da estrutura de dados ela apresentava apenas essas cinco ocorrências mas o valor do contador não correspondia á contagem real porque estava a incluir os contadores das categorias *InProceedings* e *inproceedings*.

Como resultado final obteve-se o ficheiro *HTML* com o formato pretendido, que é categoria x e o valor do contador para essa categoria x. Pode ser verificada abrindo o ficheiro lpbib.txt e procurando uma categoria qualquer e verificar que o número de ocorrências coincidem e para visualizar o aspecto da solução apenas é necessário abrir o indexA.html com um *browser* qualquer.

4.2.2 Alínea b

Para testar o programa, para além de sucessivas análises gerais do output a processar o ficheiro *lpbib.txt*, foi criado um ficheiro de dimensões mais reduzidas denominado *bugs.txt* com entradas *BibTeX* com particularidades diferentes umas das outras. Esse ficheiro contém as várias possibilidades de formatação no campo dos autores, nomes de autores com aspas e chavetas, campos com ”, último campo terminado com ou sem vírgula e mais. Foi também criada uma gramática extra que filtrou com sucesso os nomes de todos os autores do ficheiro *lpbib.txt* já formatado pelo programa desenvolvido.

4.2.3 Alínea c

Para executar os vários testes à aplicação, foi corrida a seguinte *Makefile*:

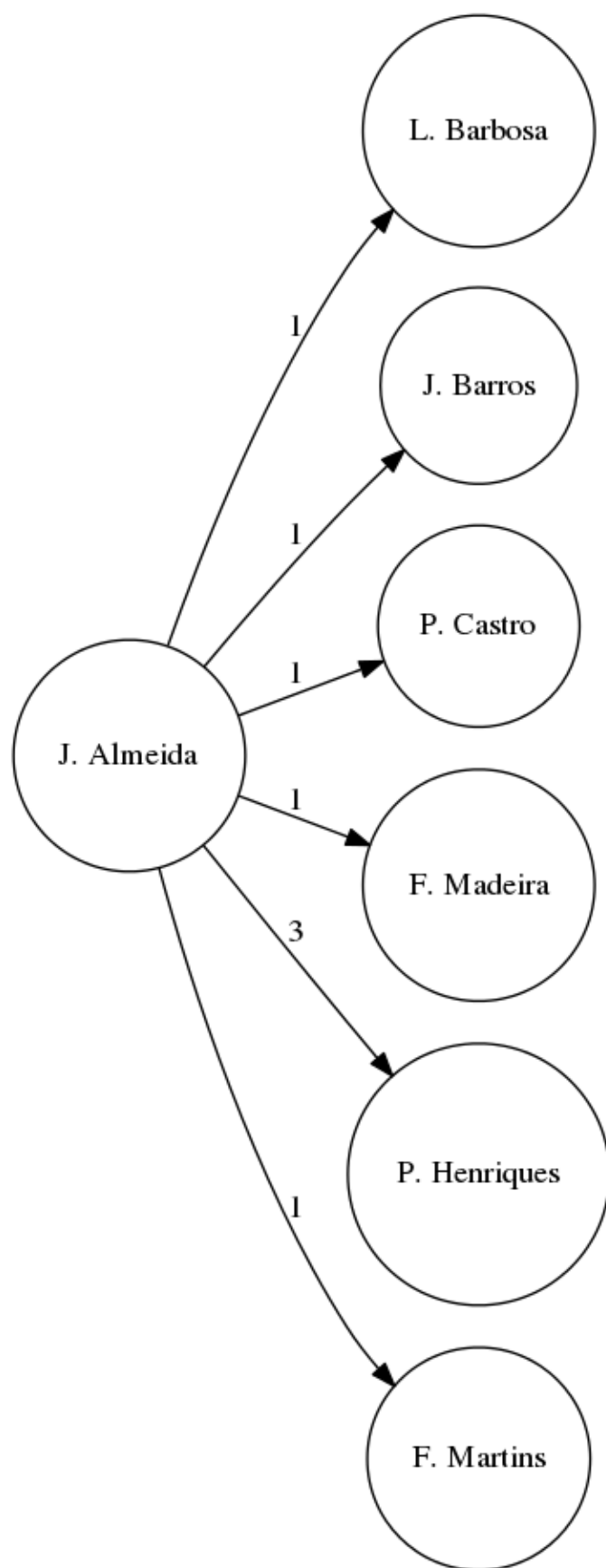
```
1 all: flexTree compileTree execTree createGraph
2
3 flexTree:
4     flex tp1C.fl
5
6 compileTree:
7     gcc lex.yy.c lista.c -ll -o tp1C
8
9 execTree:
10    ./tp1C < graphTeste.txt
```

E, de seguida, correr o seguinte comando:

```
make ARGS="J. Almeida"
```

De seguida, seguem abaixo o ficheiro *graph.dot* e o grafo respetivo:

```
1 digraph finite_state_machine {
2     rankdir=LR;
3     size="20,20"
4     node [shape = circle]; "J. Almeida" node [shape = circle]; "L. Barbosa" node [shape = circle]; "J. Barros" node [shape = circle]; "P. Castro" node [shape = circle]; "F. Madeira"
5     node [shape = circle]; "P. Henriques" node [shape = circle]; "F. Martins"
6     "J. Almeida" "J. Almeida" -> "L. Barbosa" [label = "1"]; "J. Almeida" -> "J. Barros" [label = "1"]; "J. Almeida" -> "P. Castro" [label = "1"]; "J. Almeida" -> "F. Madeira" [label = "1"]; "J. Almeida" -> "P. Henriques" [label = "3"]; "J. Almeida" -> "F. Martins" [label = "1"];
7 }
```



Capítulo 5

Conclusão

Como pudemos verificar ao longo da realização deste trabalho prático, o *Flex* é uma ferramenta bastante útil para efetuar um *parsing* a um ficheiro que contenha informação que possa ser útil em alguma situação.

Em relação ao trabalho efetuado, o grupo demonstra-se satisfeito com o resultado final do mesmo, apesar das dificuldades sentidas em alguns momentos. Conseguimos, numa primeira fase, proceder à contagem de todas as categorias existentes no ficheiro *BibTex* e, de seguida, efetuar a normalização dos campos e desenvolver uma ferramenta de *pretty-printing*, que embelezou o ficheiro inicial, sendo que o resultado desta tarefa revelou-se extremamente importante para a realização da última fase, que consistiu, na criação de um grafo de todos os autores que efetuaram publicações com um autor passado como argumento.

Para além destas tarefas, ainda optamos pela realização de mais uma tarefa (explicada em anexo), que consistiu na criação de *slides* em *Beamer*, utilizando para tal a ferramenta *Zim-wiki*. Mais uma vez e, à semelhança da tarefa anterior, os resultados gerados foram bastante satisfatórios, o que nos deixa bastante satisfeitos.

Futuramente, pretendemos aperfeiçoar os nossos conhecimento acerca desta e de outras ferramentas de *parsing* de documentos, a fim de estarmos habilitados a construir programas ainda mais sofisticados e de melhor qualidade.

Apêndice A

Código do Programa

Lista-se a seguir um excerto do ficheiro *BibTex* que foi utilizado para demonstrar o funcionamento, correto, do código desenvolvido para a resolução do problema. Ficheiro esse que foi disponibilizado em <http://di.uminho.pt/~prh/lp.bib>

```
1 @string{ eth = "Institut fur Informatik , ETH Zurich" }
2
3 @techreport{BW83a,
4   author = "Manfred Broy and Martin Wirsing",
5   title = "Generalized Heterogeneous Algebras and Partial Interpretations",
6   year = 1983,
7   month = Feb,
8   institution = "Institut fur Informatik , TUM",
9   note = "(draft version)",
10  annote = "espec algebrica"
11 }
12
13 @inbook{Val90a ,
14   author = "Jos\'e M. Valen\c{c}a",
15   title = "Processos , {O}bjectos e {C}omunica\c{c}\~ao
16           ({O}p\c{c}\~ao I - {MCC})",
17   chapter = 2,
18   year = 1990,
19   month = Oct,
20   publisher = gdcc ,
21   address = um,
22   annote = "programacao oobjectos , proc comunicantes , espec formal"
23 }
```

A.1 Alínea a

O código do programa desenvolvido em *Flex*, tal como está no ficheiro fonte *tp1A.l* encontra-se de seguida.

```
1 %{
2     #include <stdlib.h>
3     #include <stdio.h>
4     #include <string.h>
5     #include "hashtable.h"
6     HashTable ht;
7     int count=1;
8     int position=0;
9 %}
10
11
12
13 %%
14
15 @[a-zA-Z]+\{
16         {
17             char* str = (char *) malloc(sizeof(char)*1000);
18             strcpy(str,yytext+1);
19             str[yytext-2] ='\0';
20
21             insertTable(
22                 ht, str,
23                 (int *)
24                 count);
25
26         }
27
28 .|\n
29         {}
30
31
32
33
34
35
36 %%
37 int main(){
38     initializeTable(ht);
39     int s;
40     s=yylex();
41     while(s){printf("%d",s);s=yylex();}
42     printHashTable(ht);
43     printf("\n");
44     return 0;
45 }
```

Apesar de existirem bibliotecas disponíveis com estruturas de dados já implementadas, tomou-se a liberdade de reutilizar uma biblioteca já produzida numa unidade curricular anterior em vez de utilizar *hsearch.h* disponível em *glib2.h*.

O código da estrutura de dados utilizada na resolução da alínea *a* deste problema foi desenvolvido na linguagem *C* e apresenta-se de seguida o código na íntegra, que pode ser encontrado no ficheiro fonte *hashtable.h*. Lembra-se que as operações de inserir, remover e procura nesta estrutura de dados encontra-se em *hashtable.c*.

Ficheiro *.h*,

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define HASHSIZE 31
6 #define KEYTYPE_SIZE 30
7 #define EMPTY 0
8 #define DELETED -1
9 #define FULL 1
10
11 //Open Addressing – Linear Probing (insertTable) and Quadratic Probing (insertTable2)
12 //After hash_function calculated the position if it is not available it try insert in the
   next position with
13 //label "empty" or "deleted". "deleted" is required when retrieving data only stops when
   finds "empty"
14
15 typedef char Keytype[KEYTYPE_SIZE];
16 typedef void *Info;
17 typedef struct entry{
18     int state;
19     Keytype key;
20     Info info;
21 }Entry;
22
23 typedef Entry* HashTable[HASHSIZE];
24
25 int hash_function(Keytype key);
26 void initializeTable(HashTable ht);
27 void clearTable(HashTable ht);
28 void insertTable(HashTable ht, Keytype k, Info i);
29 int retrieveTable(HashTable ht, Keytype k);
30 void deleteTable(HashTable ht, Keytype k);
31 void printHashTable(HashTable ht);
```

```

1 #include "hashtable.h"
2
3
4 int hash_function(Keytype key){
5     int i=0,sum=0;
6     for (;i<KEYTYPE_SIZE-1;i++){ //ends with '\0' so we must subtract one iteration
7         sum+=key[i];
8     }
9     return (sum%HASHSIZE);
10 }
11
12 void initializeTable(HashTable ht){
13     int i=0;
14     for (;i<HASHSIZE;i++){
15         Entry* new = (Entry*) malloc(sizeof(struct entry));
16         new->state=EMPTY;
17         strncpy(new->key," Empty",KEYTYPE_SIZE);
18         new->info=NULL;
19         ht[i]=new;
20     }
21 }
22
23 void clearTable(HashTable ht){
24     int i=0;
25     for (;i<HASHSIZE;i++) free(ht[i]);
26 }
27
28 void insertTable(HashTable ht, Keytype k, Info i){
29     int position=hash_function(k);
30     int found=retrieveTable(ht,k);
31     if(found!=0){ // k já existe
32         ht[found]->info=ht[found]->info + 1;
33     }
34     else{ //k não existe
35         if(ht[position]->state!=FULL){
36             strncpy(ht[position]->key,k,KEYTYPE_SIZE);
37             ht[position]->info=i;
38             ht[position]->state=FULL;
39         }
40         else{
41             //try to insert in the next and so on — Linear Probing
42             while(ht[position]->state==FULL) position=(position+1)%HASHSIZE;
43             //it founded an EMPTY or DELETED
44             strncpy(ht[position]->key,k,KEYTYPE_SIZE);
45             ht[position]->info=i;
46             ht[position]->state=FULL;
47         }
48     }
49 }
50
51 //retrieving with linear probing from the initial position
52 int retrieveTable(HashTable ht,Keytype k){
53     int position=hash_function(k);

```



```

54  Entry* aux;
55  int res=0;
56  for (; ht[position]->state!=EMPTY && position<HASHSIZE; position=(position+1)%HASHSIZE){
57      if (strncmp(ht[position]->key,k,KEYTYPE.SIZE)==0){
58          aux=ht[position]; res=position;
59      }
60  }
61  return res;
62 }
63
64 void deleteTable(HashTable ht, Keytype k){
65     int position=hash_function(k), i=0;
66     for (; ht[position]->state!=EMPTY && i<HASHSIZE; i++){
67         if (strncmp(ht[position]->key,k,KEYTYPE.SIZE)==0){
68             ht[position]->state=DELETED;
69             strncpy(ht[position]->key," Deleted",KEYTYPE.SIZE);
70             ht[position]->info=NULL;
71         }
72     }
73 }
74
75 void printHashTable(HashTable ht){
76     int i=0;
77     Entry* aux;
78     FILE * fp = fopen("index.html","wr");
79     fprintf(fp,"<HTML>\n\t<BODY>\n");
80     fprintf(fp,"<h1> BibTex File </h1>\n");
81     for (; i<HASHSIZE; i++){
82         aux=ht[i];
83         if (aux->state!=EMPTY && aux->state!=DELETED){
84             fprintf(fp," \t\t<li>Categoria: %s, Contagem: %d\n",aux->key,(int) aux->info);
85         }
86     }
87     fprintf(fp," \t<BODY/>\n<HTML/>");
88 }

```

O ficheiro com a resolução da alínea *a* é apresentado em baixo, e o seu conteúdo está no ficheiro *indexA.html*.

```
1 <HTML>
2     <BODY>
3         <li>Categoria: MISC, Contagem: 1
4         <li>Categoria: TechReport, Contagem: 2
5         <li>Categoria: mastersthesis, Contagem: 2
6         <li>Categoria: techreport, Contagem: 138
7         <li>Categoria: proceeding, Contagem: 1
8         <li>Categoria: ARTICLE, Contagem: 1
9         <li>Categoria: MISC, Contagem: 1
10        <li>Categoria: unpublished, Contagem: 15
11        <li>Categoria: INPROCEEDINGS, Contagem: 5
12        <li>Categoria: phdthesis, Contagem: 21
13        <li>Categoria: string, Contagem: 31
14        <li>Categoria: incollection, Contagem: 6
15        <li>Categoria: manual, Contagem: 13
16        <li>Categoria: BOOK, Contagem: 2
17        <li>Categoria: InProceedings, Contagem: 20
18        <li>Categoria: inbook, Contagem: 3
19        <li>Categoria: inproceedings, Contagem: 184
20        <li>Categoria: book, Contagem: 44
21        <li>Categoria: misc, Contagem: 39
22        <li>Categoria: proceedings, Contagem: 4
23        <li>Categoria: article, Contagem: 131
24        <li>Categoria: Article, Contagem: 10
25        <li>Categoria: Misc, Contagem: 20
26        <li>Categoria: Book, Contagem: 1
27    <BODY/>
28 <HTML/>
```

A.2 Alínea b

Ficheiro *prettybibtex.l*:

```
1  %{
2  #include <stdlib.h>
3  #include <stdio.h>
4  #include <ctype.h>
5  #include "prettybibtex.h"
6
7  /*
8      Idente cada Campo,
9      Um autor por linha,
10     Colocar no inicio os campos "author title"
11  */
12
13  Registo registo;
14
15  // Variaveis para campos de token chaveta
16  int it2 = 0;
17  int chavetaAberta = 0;
18
19  char apelido[20];
20  char primeiroNome[3]; // SEMpre {[A-Z],., }
21  int nomeCompleto = 0;
22  int temApelido = 0;
23  int formato2 = 0; // Formato dos nomes Apelido, nomes and ...
24  int primeiroNomeParsed = 0; // Usado para o formato com AND
25
26  void formatAuthor(){
27  if(formato2 && !nomeCompleto){
28  // parsing nome formato 2
29  nomeCompleto = 1; // Apelido vem primeiro
30      primeiroNome[0] = yytext[0];
31      primeiroNome[1] = '.';
32      primeiroNome[2] = ' ';
33  }else if(!formato2){
34      if(!primeiroNomeParsed){
35  // parsing primeiro nome formato
36      primeiroNome[0] = yytext[0];
37      primeiroNome[1] = '.';
38      primeiroNome[2] = ' ';
39      primeiroNomeParsed = 1;
40      temApelido = 0;
41  }else{
42      /* coloca todos os nomes da pessoa com exceção do primeiro no apelido
43      como o apelido @ o ultimo nome neste formato o ultimo nome vai
44      sobrepor os outros */
45  // Parsing possivel apelido
46      temApelido = 1;
47      strcpy(apelido, yytext); apelido[yyleng] = '\0';
48  // if(apelido[yyleng-1] == '}') apelido[yyleng-1] = '\0';
49  }
50  }
51  }
52
53  %}
```

```

55 %cx AUT CHAVE CAMPOS NOMECAMPO VALORCAMPO VALORCAMPOT2
56
57 %%
58 <INITIAL>@[Ss][Tt][Rr][Ii][Nn][Gg]\{ { ECHO;}
59 <INITIAL>@[A-Za-z]+\{ {
60 registo = initRegisto();
61 yytext[yytext - 1] = '\0'; yytext+=1;
62 registo->categoria = strdup(yytext);
63 BEGIN CHAVE;}
64 <INITIAL>.\n { ECHO;}
65 <*><EOF> { return 0;}
66 <CHAVE>[A-Za-z0-9:]+ {registo->chave = strdup(yytext);BEGIN CAMPOS;}
67 <CHAVE>.\n {
68 <CAMPOS>(",")[\n\t]*\} { printRegisto(registo); BEGIN INITIAL;}
69 <CAMPOS>\,[\n\t]* {BEGIN NOMECAMPO;}
70 <NOMECAMPO>[ ]*[Aa][Uu][Tt][Hh][Oo][Rr][ ]*=[ ]*["{}][ ]*["}] { /* Lista de Autores vazia */
71 registo->autores = NULL;
72 BEGIN CAMPOS;}
73 <NOMECAMPO>[ ]*[Aa][Uu][Tt][Hh][Oo][Rr][ ]*=[ ]*["{}][ ]*["}] {
74 registo->iautores=0;
75 temApelido = 0; nomeCompleto = 0; primeiroNomeParsed = 0;formato2=0;
76 BEGIN AUT;
77 }
78 <NOMECAMPO>[ ]*[Aa][Uu][Tt][Hh][Oo][Rr][ ]*=[ ]*["{}][ ]*["}] {
79 int iAspas = first(yytext, '"');
80 int iPar = first(yytext, '{'); int iToken;
81 if( iAspas < iPar) iToken = iAspas; else iToken = iPar;
82 yytext[yytext-1] = '\0';
83 char* singleName = yytext + iToken + 1;
84 registo->autores[registo->iautores++] = strdup(singleName);
85 BEGIN CAMPOS;
86 }
87 <NOMECAMPO>[ ]*=[ ]* { BEGIN VALORCAMPO;}
88 <NOMECAMPO>[a-zA-Z-]+ {registo->campos[registo->r]->nome = strdup(yytext);}
89 <NOMECAMPO>[\n\t]* {
90 <AUT>[ ]+[Aa]nd[ ]+ {
91 char* authorName = malloc(sizeof(char) * 30);
92 strcpy(authorName, primeiroNome);
93 if(temApelido) strcpy(authorName + 3, apelido);
94 registo->autores[registo->iautores++] = authorName;
95 // reinicializar as variaveis de controlo
96 temApelido = 0; nomeCompleto = 0; primeiroNomeParsed = 0;
97 }
98 <AUT>["}], [ ]*\n {
99 // Paragem de author
100 char* authorName = malloc(sizeof(char) * 30);
101 strcpy(authorName, primeiroNome);
102 if(temApelido) strcpy(authorName + 3, apelido);
103 registo->autores[registo->iautores++] = authorName;
104 BEGIN NOMECAMPO;
105 }
106 <AUT>([^\n]("\\""\n"?))+/"}, {
107     formatAuthor();
108 }
109 <AUT>([^\n]("\\""\n"?))+/, { /*identifica o formato Apelido, Nomes and .. */ formato2 = 1;
110 // Parsing apelido formato 2
111 strcpy(apelido, yytext);
112 nomeCompleto = 0;
113 temApelido = 1;

```

```

114 }
115 <AUT>([\^, "\n]("\\\\""?)) + {
116     formatAuthor();
117 }
118 <AUT>.\n {
119     /* Em principio so limpa espaÃ§os e virgulas no caso da formataÃ§ao 2 */ }
120 <VALORCAMPO>[a-zA-Z0-9]+ { registro->campos[registro->r++]->valor= strdup(yytext); BEGIN
    CAMPOS;}
121 <VALORCAMPO>\"([\^]"\\\\""?))*\" { yytext[yyleng -1] = '}''; yytext[0] = '{';
122 registro->campos[registro->r++]->valor = strdup(yytext);
123 BEGIN CAMPOS;}
124 <VALORCAMPO>\{ {
125     chavetaAberta = 1;
126     registro->campos[registro->r]->valor = malloc( CAMP_BUFFER_SIZE * sizeof(char));
127     it2 = 0;
128     registro->campos[registro->r]->valor[it2++] = '{';
129     BEGIN VALORCAMPOT2; }
130 <VALORCAMPOT2>\{ {
131     chavetaAberta++;
132     registro->campos[registro->r]->valor[it2++] = '{';
133 }
134 <VALORCAMPOT2>\} {
135     chavetaAberta--;
136     if(chavetaAberta == 0){
137         registro->campos[registro->r]->valor[it2++] = '}'';
138         registro->campos[registro->r++]->valor[it2++] = '\\0';
139         BEGIN CAMPOS;
140     } else {
141         registro->campos[registro->r]->valor[it2++] = '}'';
142     }
143 }
144 <VALORCAMPOT2>.\n {
145     registro->campos[registro->r]->valor[it2++] = yytext[0];
146 }
147 %%
148
149 //int yywrap(){return 0;}
150
151 int main(){
152     int s;
153     while(s = yylex()){printf("%d",s);}
154     return 0;
155 }

```

```

1
2 #include <stdlib.h>
3 #include <stdio.h>
4 #include <ctype.h>
5 #include "prettybibtex.h"
6
7 Registo initRegisto(){
8     Registo reg = malloc( sizeof(struct registo));
9     reg->r = 0; reg->iautores = 0;
10    reg->autores = malloc( 30 * sizeof(char*));
11    int k;
12    int n = MAXNUMBERCAMPS;
13
14    reg->campos = malloc( n * (sizeof(Campo)));
15    for(k=0; k < n ; k++){
16        reg->campos[k] = malloc(sizeof(struct campo));
17    }
18    return reg;
19
20 void orderCampos(Registo reg){
21     int j,k, t = -1; /* index a author t title */
22     for(j = 0; j < reg->r ; j++){
23         char* valor = (reg->campos[j]->nome);
24         for(k = 0; valor[k] != '\0' ; k++ )
25             valor[k] = tolower(valor[k]);
26
27         if( strcmp(valor,"title") == 0 ) {t = j; break;}
28     }
29     if( t > 0){
30         Campo sub = reg->campos[0];
31         reg->campos[0] = reg->campos[t];
32         reg->campos[t] = sub;
33     }
34 }
35
36 void printRegisto(Registo reg){
37     int k;
38     orderCampos(reg);
39     printf("@s{s,\n",reg->categoria,reg->chave);
40     if(reg->autores != NULL && reg->iautores > 0)
41         if(reg->iautores == 1)
42             printf("    author = {s},\n",reg->autores[0]);
43         else{
44             printf("    author = {s\n",reg->autores[0]);
45             for(k= 1; k < reg->iautores - 1; k++)
46                 printf("\t    s\n",reg->autores[k]);
47             printf("\t    s},\n",reg->autores[k]);
48         }
49
50     for(k = 0; k < (reg->r - 1) ; k++)
51         printf("    s = s,\n",reg->campos[k]->nome,reg->campos[k]->valor);
52
53     printf("    s = s\n",reg->campos[reg->r - 1]->nome,reg->campos[reg->r-1]->valor);
54     printf("}\n");
55     free(reg); reg = NULL;
56 }

```

```
57
58 int first(char* s,char c ){
59     int t;
60     for(t = 0; s[t]!='\0'; t++)
61         if( s[t]== c) break;
62     return t;
63 }
```

```
1 #ifndef PRETTYBIBTEX_H
2
3 #define PRETTYBIBTEX_H
4
5 #define CAMP_BUFFER_SIZE 1000
6 #define MAX_NUMBER_CAMPS 50
7
8 typedef struct campo{
9     char* nome;
10    char* valor;
11 }*Campo;
12
13 typedef struct registo{
14     char* categoria;
15     char* chave;
16     char** autores;
17     int iautores;
18     Campo* campos;
19     int r;
20 }*Registo;
21
22 Registo initRegisto();
23
24 void orderCampos(Registo reg);
25
26 void printRegisto(Registo reg);
27
28 int first(char* s, char c);
29
30 #endif
```

A.3 Alínea c

Ficheiro *tp1C.fl*:

```
1  %{
2
3  #include <stdio.h>
4  #include <string.h>
5  #include <errno.h>
6  #include <stdlib.h>
7  #include <string.h>
8  #include <ctype.h>
9  #include "lista.h"
10
11 char *author;
12 GRAPH autores;
13 GRAPH linha;
14
15 void printLinha(GRAPH linha);
16 void printAutores();
17
18  %}
19
20 %x AUTOR
21
22 %%
23
24                                     autores = insert(NULL, author);
25 author[ ]*\=[ ]*[{ " ]           {linha = NULL; BEGIN AUTOR;}
26 <AUTOR>[\t" "\n]+               {}
27 <AUTOR>[^\\t\\n]*/"\}, "         {char *authorName = strdup(yytext); linha = insert(linha, authorName
   );}
28 <AUTOR>"\},\n"                   {autores = getAutores(autores, linha, author); BEGIN INITIAL;}
29 <AUTOR>[^\\t\\n]+               {char *authorName = strdup(yytext); linha = insert(linha, authorName
   );}
30 .|\n                             {}
31 <*><<EOF>>                       {printAutores(); free(linha); free(autores); return 0;}
32
33 %%
34
35 void printAutores() {
36     FILE *fp = fopen("graph.dot", "w");
37     GRAPH aux = autores;
38     fprintf(fp, "digraph finite_state_machine {\n");
39     fprintf(fp, "    rankdir=LR;\n");
40     fprintf(fp, "    size=\n20,20\n");
41     while(aux != NULL) {
42         fprintf(fp, "node [shape = circle];");
43         fprintf(fp, " \n\"%s\" ", aux->author);
44         aux = aux->next;
45     }
46     aux = autores;
47     while(aux != NULL) {
48         if(strncmp(author, aux->author, strlen(author)) == 0) fprintf(fp, "\n\"%s
   \", author);
49         else fprintf(fp, "\"%s\" -> \"%s\" [label = \"%d\"];", author, aux->author,
   aux->num);
50         aux = aux->next;
```

```

51     }
52     fprintf(fp, "\n");
53     fclose(fp);
54 }
55
56 int main(int argc, char *argv[]) {
57     if(argc == 3) {
58         int s;
59         char *firstName = argv[1];
60         char *lastName = argv[2];
61         char *result = malloc(strlen(firstName) + strlen(lastName) + 2);
62         strcpy(result, firstName);
63         strcat(result, " ");
64         strcat(result, lastName);
65         author = strdup(result);
66         s=yylex();
67         while(s) {
68             printf("%d",s);
69             s=yylex();
70         }
71         free(result);
72     }
73     return 0;
74 }

```

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <errno.h>
4 #include <stdlib.h>
5 #include <string.h>
6 #include <ctype.h>
7 #include "lista.h"
8
9 GRAPH insert(GRAPH g, char *author) {
10     if(g == NULL) {
11         g = (GRAPH) malloc(sizeof(struct grafo_autores));
12         g->author = strdup(author);
13         g->num = 1;
14         g->next = NULL;
15     }
16     else if(strncmp(author, g->author, strlen(author)) == 0) g->num++;
17     else g->next = insert(g->next, author);
18     return g;
19 }
20
21 GRAPH getAutores(GRAPH autores, GRAPH linha, char *author) {
22     GRAPH aux = linha;
23     while(aux != NULL) {
24         if(strncmp(author, aux->author, strlen(author)) == 0) {
25             aux = linha;
26             while(aux != NULL) {
27                 if(strncmp(author, aux->author, strlen(author)) != 0)
28                     autores = insert(autores, aux->author);
29                 aux = aux->next;
30             }
31             break;
32         }
33         else aux = aux->next;
34     }
35     return autores;
36 }
```

Ficheiro *lista.h*:

```
1 #ifndef LISTA_H
2
3 #define LISTA_H
4
5 typedef struct grafo_autores {
6     char *author;
7     int num;
8     struct grafo_autores *next;
9 } GA;
10
11 typedef struct grafo_autores *GRAPH;
12
13 GRAPH insert(GRAPH g, char *author);
14 GRAPH getAutores(GRAPH autores, GRAPH linha, char *author);
15
16 #endif
```

A.4 Extra- Zim to Beamer

Decidiu-se também resolver o último enunciado que fiz respeito á conversão de ficheiros *Zim-wiki* para slides *Beamer*, que é uma variação de *LaTeX*, cujos slides são criados consoante o tipo de *Header* lido (apenas os tipos 3,4,5 permitem criar novo slide).

Ficheiro *Zim-Wiki* escrito em .txt,

```
1 ===== Home =====
2 Created Tuesday 22 March 2016!
3 This is a test!
4
5
6 ===== Header 2 =====
7 Should be a new slide!!
8 Next items should produce a List in Bold with three different bullets!
9
10 * item 1
11 * item 2
12 * item 3
13     1. item 3
14         a. item 3a
15
16
17 ===== Header 3 =====
18 Should be a new slide!!
19 Now it should be appearing //check boxes in italic//
20 ===== head 4 =====
21 Should not be a new slide underlined!!
22
23 [ ] empty
24 [*] checked wright
25     [*] sub checked wright
26     [ ] sub empty
27 [x] checked wrong
28
29
30 ===== Header 3 =====
31 Should be a new slide!!
32
33 ===== head 5 =====
34 Should not be a new slide!!
35
36 Now an example of ~~strike through~~
37
38 To finish this presentation an image must be shown!
39 An import of an image!!!
40
41 {{./atom.jpg}}
42
43
44 ===== End =====
45 Now an example of verbatim working ''int main() {return(0);} in verbatim ''!!!
```

Ficheiro *Ficheiro.l* com as expressões regulares que permitem resolver o problema,

```

1  %{
2      #include <stdio.h>
3      int firstSlide=0;
4      FILE *fp;
5  %}
6  %%
7  ^[=]{6}[ ]{1}[a-zA-Z0-9 ]+[ ]{1}[=]{6}          {
8
9
10
11
12
13
14
15
16
17
18
19
20 ^[=]{5}[ ]{1}[a-zA-Z0-9 ]+[ ]{1}[=]{5}          {
21
22

```

```

if (firstSlide==0){
fp=fopen("zim2beamer.tex","w
");
fprintf(fp,"%cdocumentclass{
beamer}\n %chypersetup{
pdfstartview={Fit}}\n %
cusetheme{Madrid}\n
",92,92,92);
fprintf(fp,"%cusecolortheme{
beaver}\n %cusepackage{
pifont}\n %cusepackage{
cancel}\n %cbegin{
document}\n
",92,92,92,92);
fprintf(fp,"%c title [Teste
Beamer]{The Real Deal}\n
%c subtitle {Example of
Beamer}\n",92,92);
fprintf(fp,"%c author [Ze
Carlos]\n",92);
fprintf(fp,"%c F.~ Author %
cinst{1} }\n %cinstitute
[University of Minho]{\n
%cinst{1}\n",92,92,92);
fprintf(fp,"Department of
Computer Science%c%c\n
",92,92);
fprintf(fp,"University of
Minho\n}\n %cdate [PT
2016]{Try number 3,
2016}\n %csubject{
Computer Science}\n
",92,92);
}
yytext[yytext-7]='\0';
fprintf(fp," \n%cbegin{
frame}\n\t%cframetitle{%
s}\n",92,92,yytext+7);
firstSlide++;
}

if (firstSlide==0){
fprintf(fp,"%cdocumentclass{beamer}\n
%c hypersetup{pdfstartview={Fit
}}\n %cusetheme{Madrid}\n
",92,92,92);

```

```

23 fprintf(fp,"%cusecolortheme{beaver}\n
   %cusepackage{pifont}\n %
   cusepackage{cancel}\n %cbegin{
24 document}\n",92,92,92,92);
   fprintf(fp,"%ctitle[Teste Beamer]{
   The Real Deal}\n %csubtitle{
25 Example of Beamer}\n",92,92);
   fprintf(fp,"%cauthor[Ze Carlos]\n
26 ",92);
   fprintf(fp,"%{F.~ Author %cinst{1} }\n
   %cinstitute[University of Minho
   ]{\n %cinst{1}\n",92,92,92);
27 fprintf(fp,"Department of Computer
   Science%c%c\n",92,92);
28 fprintf(fp,"University of Minho\n\n
   %cdate[PT 2016]{Try number 3,
   2016}\n %csubject{Computer
   Science}\n",92,92);
29 }
30 yytext[yytext-6]='\0';fprintf(fp,"\n
   %cbegin{frame}\n\t%cframetitle{%
   s}\n",92,92,yytext+6);firstSlide
31 ++;

32 ^[=]{4}[ ]{1}[a-zA-Z0-9 ]+[ ]{1}[=]{4} {
33
34 if (firstSlide==0){
   fprintf(fp,"%cdocumentclass{beamer}\n %
   chypersetup{pdfstartview={Fit}}\n %
   cusetheme{Madrid}\n",92,92,92);
35   fprintf(fp,"%cusecolortheme{beaver}\n
   %cusepackage{pifont}\n %
   cusepackage{cancel}\n %cbegin{
   document}\n",92,92,92,92);
36   fprintf(fp,"%ctitle[Teste Beamer]{
   The Real Deal}\n %csubtitle{
   Example of Beamer}\n",92,92);
37   fprintf(fp,"%cauthor[Ze Carlos]\n
   ",92);
38   fprintf(fp,"%{F.~ Author %cinst{1} }\n
   %cinstitute[University of Minho
   ]{\n %cinst{1}\n",92,92,92);
39   fprintf(fp,"Department of Computer
   Science%c%c\n",92,92);
40   fprintf(fp,"University of Minho\n\n
   %cdate[PT 2016]{Try number 3,
   2016}\n %csubject{Computer
   Science}\n",92,92);
41 }
42 yytext[yytext-5]='\0';fprintf(fp,"\n
   %cbegin{frame}\n\t%cframetitle{%
   s}\n",92,92,yytext+5);firstSlide
43 ++;

44 \n\n\n { fprintf(fp,"%cend{frame}\n",92);}
45 ^[=]{3}[ ]{1}[a-zA-Z0-9 ]+[ ]{1}[=]{3} {yytext[yytext-4]='\0';fprintf(fp,"\n%s\n",
   yytext+4);}

```

```

46  ^[=]{2}[ ]{1}[a-zA-Z0-9 ]+[ ]{1}[=]{2}          {yytext[yylen-3]='\0';fprintf(fp,"\\n%s\\n",
      yytext+3);}
47  \\{\\.\\*\\}
      {yytext[yylen-2]='\0';fprintf(fp,"\\n%cbegin{
      figure}[!ht]\\n\\t %ccentering\\n\\t %cincludegraphics[width=0.25 %ctextwidth]{%s} \\n\\t %
      ccaption{Atom}\\n %cend{figure}\\n",92,92,92,92,yytext+4,92,92);}
48  \\*+\\*
      {yytext[yylen-2]='\0';fprintf(
      fp,"%ctextbf{%s}\\n",92,yytext+2);}
49  \\-+\\-
      {yytext[yylen-2]='\0';fprintf(
      fp,"\\n%cunderline{%s}\\n",92,yytext+2);}
50  [/]{2}[a-zA-z0-9 ]+[/]{2}
      {yytext[yylen-2]='\0';fprintf(fp,"\\n%ctextit{%s}\\n",92,yytext+2);}
51  [~]{2}[a-zA-z0-9 ]+[~]{2}
      {yytext[yylen-2]='\0';fprintf(fp,"\\n%ccancel{%s}\\n",92,yytext+2);}
52  [']{2}.+[']{2}
      {yytext[yylen-2]='\0';fprintf(fp,"\\n%ccend{frame}\\n%
      cbegin{frame}[fragile]\\n%cbegin{verbatim}\\n%s\\n%ccend{verbatim}\\n%ccend{frame}\\n
      ",92,92,92,yytext+2,92,92);}
53  [0-9]\\. [A-Za-z0-9]+
      {fprintf(fp,"\\n%cbegin{itemize}\\n\\n%cbegin{itemize}\\n%ccitem %s\\n%ccend{
      itemize}\\n%ccend{itemize}\\n",92,92,92,yytext,92,92);}
54  [a-z]\\. [ ]{1}[A-Za-z0-9]+
      {fprintf(fp,"\\n%cbegin{itemize}\\n\\n%cbegin{itemize}\\n%ccitem %s\\n%ccend{itemize}\\n
      %ccend{itemize}\\n",92,92,92,yytext,92,92);}
55  ^[A-Z][A-Za-z0-9 \\!]+
      {fprintf(fp,"%s\\n",yytext);}
56  \\[[ ]\\}\\*
      {fprintf(fp,"\\n%cding{112} %s %cpar\\n",92,yytext+4,92);}
57  .\\[[ ]\\}\\*
      {fprintf(fp,"%chspace{5ex}%cding{112} %s %cpar\\n",92,92,
      yytext+4,92);}
58  .\\[\\*\\]\\*
      {fprintf(fp,"%chspace{5ex}%cverb|%ccheckmark%s %cpar\\n
      ",92,92,92,yytext+4,92);}
59  \\[\\*\\]\\*
      {fprintf(fp,"%cverb|%ccheckmark %s %cpar\\n",92,92,yytext
      +4,92);}
60  \\[[X|x]\\}\\*
      {fprintf(fp,"%cverb|%cxmark X %s %cpar\\n",92,92,yytext+4,92);}
61  \\n\\*[ ]{1}[a-zA-Z0-9 ]+
      {fprintf(fp,"\\n%cbegin{itemize}\\n%ccitem %s\\n%ccend{itemize}\\n",92,92,yytext+3,92)
      ;}
62  .|\\n
      {}
63  %%
64  int main(){
65      int s;
66      s=yylex();
67      while(s){fprintf(fp,"%d",s);s=yylex();}
68      fprintf(fp,"%ccend{document}\\n",92);
69      fclose(fp);
70      return 0;
71  }

```

Ficheiro *Ficheiro.tex* com os slides criados a partir de *zim2beamer.txt*, que representa o conteúdo de um ficheiro *Zim-Wiki*.

Para criar os respetivos slides apenas bastará compilar o ficheiro *.tex* para se obter o ficheiro PDF.

```
1 \documentclass{beamer}
2 \hypersetup{pdfstartview={Fit}}
3 \usetheme{Madrid}
4 \usecolortheme{beaver}
5 \usepackage{pifont}
6 \usepackage{cancel}
7 \begin{document}
8 \title[Teste Beamer]{The Real Deal}
9 \subtitle{Example of Beamer}
10 \author[Ze Carlos]
11 {F.~Author \inst{1} }
12 \institute[University of Minho]{
13 \inst{1}
14 Department of Computer Science\\
15 University of Minho
16 }
17 \date[PT 2016]{Try number 3, 2016}
18 \subject{Computer Science}
19
20 \begin{frame}
21     \frametitle{Home}
22     Created Tuesday 22 March 2016!
23     This is a test!
24 \end{frame}
25
26 \begin{frame}
27     \frametitle{Header 2}
28     Should be a new slide!!
29     Next items should produce a
30     \textbf{List in Bold}
31
32 \begin{itemize}
33 \item item 1
34 \end{itemize}
35
36 \begin{itemize}
37 \item item 2
38 \end{itemize}
39
40 \begin{itemize}
41 \item item 3
42 \end{itemize}
43
44 \begin{itemize}
45
46 \begin{itemize}
47 \item 1. item 3
48 \end{itemize}
49 \end{itemize}
50
51 \begin{itemize}
```

```

53 \begin{itemize}
54 \item a. item
55 \end{itemize}
56 \end{itemize}
57 \end{frame}
58
59 \begin{frame}
60     \frametitle{Header 3}
61     Should be a new slide!!
62     Now it should be appearing
63
64     \textit{check boxes in italic}
65
66     head 4
67     Should not be a new
68
69     \underline{slide underlined}
70
71     \ding{112} empty \par
72     \verb|\checkmark checked wright \par
73     \hspace{5ex}\verb|\checkmark sub checked wright \par
74     \hspace{5ex}\ding{112} sub empty \par
75     \verb|\xmark X checked wrong \par
76 \end{frame}
77
78 \begin{frame}
79     \frametitle{Header 3}
80     Should be a new slide!!
81
82     head 5
83     Should not be a new slide!!
84     Now an example of
85
86     \cancel{strike through}
87     To finish this presentation an image must be shown!
88     An import of an image!!!
89
90     \begin{figure} [!ht]
91         \centering
92         \includegraphics [width=0.25 \textwidth]{atom.jpg}
93         \caption{Atom}
94     \end{figure}
95 \end{frame}
96
97 \begin{frame}
98     \frametitle{End}
99     Now an example of verbatim working
100
101 \end{frame}
102 \begin{frame}[fragile]
103 \begin{verbatim}
104 int main() {return(0);} in verbatim
105 \end{verbatim}
106 \end{frame}
107 \end{document}

```

Alguns *prints* do ficheiro PDF final, visto que não fazia sentido incluir um PDF dentro de outro. Pretende-se apenas mostrar três slides que contém a maior parte das características de uma apresentação.

Header 2

Should be a new slide!! Next items should produce a **List in Bold**

- item 1
- item 2
- item 3
 - 1. item 3
 - a. item

Navigation icons

Ze Carlos (University of Minho)

Teste Beamer

PT 2016 2 / 6

Header 3

Should be a new slide!!

head 5 Should not be a new slide!! Now an example of ~~strike through~~

To finish this presentation an image must be shown! An import of an image!!!



Figure: Atom

Navigation icons

Ze Carlos (University of Minho)

Teste Beamer

PT 2016 4 / 6

Header 3

Should be a new slide!! Now it should be appearing

check boxes in italic

head 4 Should not be a new

slide underlined

☐ empty

✓checked wright

 ✓sub checked wright

☐ sub empty

X checked wrong



Durante a elaboração deste enunciado os testes realizados passaram pela geração do seus ficheiros PDF e ir comparando com o ficheiro .txt que possui o conteúdo de um ficheiro *Zim-Wiki* e observando se os slides estavam de acordo com a sua especificação.

O ficheiro .l contém todo o conjunto de expressões regulares que permitem capturar os padrões que dizem respeito às regras de escrita de um ficheiro *zim* e as ações que toma, escrever o respectivo conteúdo no ficheiro *LaTeX Beamer*.

Os problemas que ocorreram relacionam-se com o o facto de por vezes não estar alinhado mas nada de especial. O maior problema foi apenas o facto de ao fazer "fprintf" para o ficheiro .tex não colocava 'é como tal recorreu-se á utilização do seu valor *ASCII*, que é o número 92 e verificava-se quando surge o símbolo percentagem seguido de 'c'.

Bibliografia

- [1] *V. Aho, Alfred , S. Lam, Monica , Sethi, Ravi and D. Ullman, Jeffrey* Second Edition, *Compilers Principles Techniques and Tools*.
- [2] ShareLatex examples and tutorials, <https://www.sharelatex.com>