

Processamento de Linguagens  
**Trabalho Prático 1**  
Relatório de Desenvolvimento

Higor Silva

João Monteiro

## **Resumo**

No âmbito da Unidade Curricular de Processamento de Linguagens, e com o intuito de aumentar a experiência de uso em ambiente Linux, e de algumas ferramentas de apoio à programação; aumentar a capacidade de escrever Expressões Regulares (ER) para descrição de padrões de frases; desenvolver, a partir de ERs, sistemática e automaticamente Processadores de Linguagens Regulares, que filtrem ou transformem textos e utilizar geradores de filtros de texto como o flex; para o efeito foram propostos vários enunciados dos quais poderíamos escolher pelo menos um. Foi então proposto a realização de uma lista de tarefas tais como, especificar os padrões de frases a encontrar no texto fonte, desenvolver um filtro de texto para fazer o reconhecimento dos padrões identificados e proceder à transformação pretendida, com recurso ao Flex, entre outras. Assim, este trabalho visa consolidar uma série de conceitos e conhecimentos adquiridos ao longo das aulas, sendo posto em prática tudo aquilo que nos foi lecionado.

**Área de Aplicação:** Processadores de Linguagens

**Palavras-Chave:** Expressões Regulares, filtros, flex, padrões, processadores de linguagens.

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>2</b>
<b>2</b>	<b>Análise e Especificação</b>	<b>3</b>
2.1	Descrição informal do problema (Enunciado)	3
2.2	Especificação dos requisitos	3
2.2.1	Dados	3
2.2.2	Pedidos	3
<b>3</b>	<b>Concepção/Desenho da solução</b>	<b>4</b>
3.0.1	TAG'S:	4
3.0.2	ESTRUTURAS E FUNÇÕES:	6
<b>4</b>	<b>Codificação e testes</b>	<b>7</b>
4.1	Alternativas, Decisões e Problemas de Implementação	7
4.2	Testes realizados e Resultados	7
<b>5</b>	<b>Conclusão</b>	<b>9</b>
<b>A</b>	<b>Código do Programa</b>	<b>10</b>

# Capítulo 1

## Introdução

Sendo que este projeto foi desenvolvido para a UC de Processamento de Linguagens, tem assim como principal objetivo uma melhor adaptação do grupo à escrita de Expressões Regulares e geradores de filtros. Para o seu desenvolvimento foi utilizado o *Flex*. O enunciado proposto pelo docente, continha diversos temas, dos quais o grupo teve de escolher pelo menos um. Optamos pelo tema "Template multi-file", pretendendo criar um programa "mkfromtemplate", capaz de aceitar um nome de projeto, e um ficheiro descrição de um template multi-file e que crie os ficheiros e pastas iniciais do projeto.

### Estrutura do relatório

O presente documento descreve o problema que nos foi posto em mãos nesta UC. No capítulo 1 é feita uma breve introdução ao problema. Nos capítulos seguintes vai-se aprofundando e especificando os detalhes.

No capítulo 2 é feita uma descrição informal do problema e uma especificação de requisitos. Aqui são expostos os dados que nos foram fornecidos no enunciado, tal como o que nos foi pedido para desenvolver.

No terceiro capítulo são descritas as estruturas de dados utilizadas e os algoritmos desenvolvidos.

No capítulo 4, são expostas as alternativas, decisões e problemas encontrados no desenvolvimento do projecto, bem como os resultados obtidos.

Por fim, na conclusão, é apresentada a análise crítica do projecto e o seu estado final.

## Capítulo 2

# Análise e Especificação

### 2.1 Descrição informal do problema (Enunciado)

Sabendo que em vários projectos de software, é habitual soluções envolvendo vários ficheiros e pastas, o que se pretende é criar um programa que seja capaz de aceitar um nome de projecto, e um ficheiro descrição de um template-multi-file, a partir do qual se crie os ficheiros e pastas iniciais do projecto. O template inclui:

- metadados (author, email) a substituir nos elementos seguintes
- tree (estrutura de directorias e ficheiros a criar)
- template de cada ficheiro

Como resultado da execução serão criados os ficheiros e directorias descritos em tree, com os conteúdos definidos nos templates de ficheiro, e as variáveis substituídas.

### 2.2 Especificação dos requisitos

#### 2.2.1 Dados

Foi fornecido, no enunciado, pelo docente, um "template multi-file", de forma a que o grupo tivesse um noção do problema, texto esse que seria posteriormente processado.

#### 2.2.2 Pedidos

No contexto do enunciado escolhido, foi pedido ao grupo que especificasse os padrões de frases a encontrar no template, através de Expressões Regulares. Isto foi feito no ficheiro temp-multi.fl, que se encontra no Apêndice A, tendo-se especificado ER para identificar metadados (author, email), a estrutura das directorias a criar e o conteúdo de cada ficheiro. Foi pedido também que se identificasse as ações semânticas a realizar, o que também se encontra no ficheiro temp-multi.fl, juntamente com os filtros de texto desenvolvidos para reconhecer os padrões previamente identificados. As estruturas de dados utilizadas para armazenar a informação que vai sendo extraída do texto fonte, tasi como as *Tag's* desenvolvidas vão ser mencionadas na secção seguinte deste relatório.

## Capítulo 3

# Concepção/Desenho da solução

Neste capítulo, apresentaremos uma lista com as descrições das estruturas de dados, variáveis e funções, bem como as "tags" usadas na concepção da solução ao problema (implementação).

### 3.0.1 TAG'S:

- **%s META TREE FILEMODE**: Simplificamos a estrutura do template multi file em três seções principais, a de meta dados, onde se reconhece o tipo de seção e dados básicos como autor e e-mail. Como o processamento terá diferentes comportamentos para cada seção, então definimos três estados distintos para isso.
- **^ === ""\***: Essa ER é o identificador de meta dado de seção. Uma vez encontrada, ela muda estado para **<META>** e define que tipo de informação está procurado.
- **<META>[ \t]+**: No estado de meta dados, espaços não são necessários e por isso são consumidos sem nenhuma ação específica, apenas para manter somente o texto essencial dentro do token.
- **<META>e-?mail:** : essa ER identifica o meta dado do e-mail, ou seja, o próximo texto que será processado será o e-mail, então ele ativa uma flag que indica ao lex esse comportamento.
- **<META>author:** : assim como a ER anterior, essa identifica que o próximo token a ser processado será o nome do autor, ativando uma flag para que o lex identifique tal comportamento.
- **<META>[a-zA-Z@.\%]+[ ]\*[a-zA-z@.]\*** : essa ER lê o texto indicado pelas flags anteriores e o armazena no lugar devido. Notamos que ela está preparada para ler letras, os nomes reservados (os quais serão substituídos pelos meta dados autor, nome, email), espaço entre as palavras (no caso de nome composto do autor), pontos e caracteres necessários para e-mail.
- **<TREE>[ \t]\*** : No estado TREE também não nos interessa os espaços, então por isso temos essa ER para consumir esses caracteres. Esse comportamento é comum entre os estados META e TREE, no entanto não para o FILEMODE.
- **<TREE>^[ \t]\*** : Essa expressão regular reconhece os hífens no início dos tokens que constituem a árvore de arquivos e diretórios. Como esses hífens constituem a estrutura da árvore, eles são de suma importância para identificar dentro de qual diretório cada elemento está. Por isso, a quantidade de hífens lida é o nível onde a estrutura se encontra.

- **<TREE>**[a-zA-Z@.{%}] +/V : reconhece os nomes dos diretórios porque eles terminam com o caractere /. Depois disso, realizam o processamento criando cada diretório em seu lugar adequado.
- **<TREE>**[a-zA-Z@.{%}].+ : reconhece o nome dos arquivos a serem criados e os cria dentro do diretório especificado. Além de, adicioná-los em uma lista encadeada de arquivos, que posteriormente será utilizada para abrir esses arquivos para escrita no FILEMODE.
- **<FILEMODE>**"{%author%}": quando o estado estiver em FILEMODE, sempre que o lex reconhecer essa meta tag, ele imprimirá no arquivo determinado por yyout, não o conteúdo de yytext, mas o conteúdo da variável AUTHOR.
- **<FILEMODE>**"{%nome%}", **<FILEMODE>**"{%email%}": o mesmo vale para as outras meta tags, imprimindo o valor das suas respectivas variáveis.
- **<FILEMODE>**.\n : escreve no arquivo determinado por yyout todo e qualquer caracter.
- **.\n {**: reconhece todo e qualquer caractere não reconhecido pelas outras ERs e não faz nada com eles.

### 3.0.2 ESTRUTURAS E FUNÇÕES:

- **char nameEspecial[9] = "{%name%}\0"**: variável auxiliar para comparar com o meta tag name.
- **char helper[2] = "/\0"**: variável auxiliar para adicionar o / no path do arquivo.
- **struct ftle{**
  - char \* name;**
  - char \* path;**
  - struct ftle \*next;};** estrutura para armazenar o caminho dos arquivos a serem acessados posteriormente.
- **struct ftle \*ftle list = NULL**: Head da lista ligada contendo os arquivos.
- **int LOOKUP = -1**: flag de controle que identifica qual tipo de meta dados está processando.
- **char OPTION[20]**: variável para armazenar a opção, ou o tipo de meta dados a ser processado.
- **char NAME[50]**: variável para armazenar o conteúdo da meta tag name.
- **char EMAIL[50]**: variável para armazenar o conteúdo da meta tag email.
- **char AUTHOR[50]**: variável para armazenar o conteúdo da meta tag author.
- **int opt = 0**: variável auxiliar que armazena o return da função validate.
- **int oldlvl = 0**: variável que identifica o nível do arquivo anterior, usada para auxiliar a posicionar os arquivos e diretórios corretamente.
- **int newlvl = 0**: variável que identifica o nível do arquivo atual, usada para auxiliar a posicionar os arquivos e diretórios corretamente.
- **char dirName[50]**: variável utilizada para armazenar o nome do diretório a ser criado ou manipulado.
- **char pathName[100]**: variável utilizada para armazenar o nome do caminho a ser armazenado.
- **char ftleName[50]**: variável utilizada para armazenar o nome do arquivo a ser armazenado, processado ou manipulado.
- **FILE \* aux**: variável auxiliar utilizada para abertura e criação dos arquivos, posteriormente atribuído ao yyout.
- **char auxftle[150]**: variável auxiliar utilizada para unir o pathName mais o fileName, para não ter perigo de exceder a memória alocada.
- **int validate(char\*)**: função que valida a meta tag do estado do arquivo.
- **int addFile(char\*,char\*)**: função que adiciona os dados do arquivo na estrutura de arquivos.
- **int ftndFile(char\*)**: função que encontra o arquivo na estrutura e copia o path para a variável pathName.
- **int getFileName(char\*)**: função que processa o nome do arquivo, usada principalmente para substituir as name tags, email tags, do nome do arquivo.
- **void printFileList()**: função teste para imprimir o nome dos arquivos dentro da estrutura.



## Capítulo 4

# Codificação e testes

### 4.1 Alternativas, Decisões e Problemas de Implementação

Alguns dos problemas encontrados na execução do trabalho foram: encontrar uma estrutura que armazenasse o caminho dos ficheiros, que respeitasse a estrutura das diretorias e como realizar tarefas tão distintas no mesmo interpretador léxico.

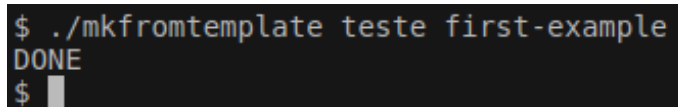
Para resolver o problema dos ficheiros e diretorias, a primeira estratégia foi construir uma lista encadeada que armazenasse a ordem de criação tanto das pastas, quanto dos arquivos. No entanto, utilizar uma estrutura em árvore parecia um tanto quanto desnecessário para isso. Por isso, criamos cada diretoria e cada ficheiro na ordem de execução e salvamos os ficheiros com seus respectivos caminhos para encontra-los quando necessário.

Para realizar tarefas distintas no mesmo interpretador léxico, utilizamos a ferramenta de estados disponibilizadas pelo FLEX. O projeto possui três estados essenciais, que são: META, TREE e FILEMODE. Que alternando a execução de cada um, gerou o resultado esperado.

Uma decisão de projeto que tomamos foi de não processarmos letras especiais, acentos, números, hífens e underline tanto no texto, como no nome dos ficheiros e diretorias.

### 4.2 Testes realizados e Resultados

- Teste 1



```
$ ./mkfromtemplate teste first-example
DONE
$ █
```

Figura 4.1: Teste 1.a

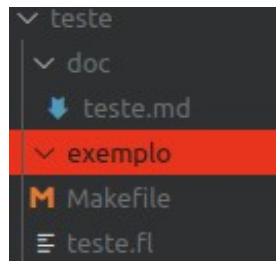


Figura 4.2: Teste 1.b

```
4  === tree
5  {%name%}/
6  - {%name%}.fl
7  - doc/
8  -- {%name%}.md
9  - exemplo/
10 - Makefile
```

Figura 4.3: Teste 1.c

```
# NAME
teste - o nosso fabuloso filtro ...FIXME
## Synopsis
teste file*
## Description
## See also
## Author
Comments and bug reports to J.Joao, jj@di.uminho.pt.
```

Figura 4.4: Teste 1.d

```
teste: teste.fl
flex teste.fl
cc -o teste lex.yy.c
install: teste
cp teste /usr/local/bin/
```

Figura 4.5: Teste 1.e

## Capítulo 5

# Conclusão

Em suma, pode-se afirmar que o projeto foi elaborado, a ver do grupo, com um bom aproveitamento, atingindo grande parte dos objetivos propostos.

Este trabalho permitiu com que o grupo se ambientasse melhor com o gerador léxico Flex, tal como com o desenvolvimento de Expressões Regulares. Podemos dizer que usando o Flex poupa-se muitas linhas de código se fosse em outra linguagem, caso o filtro tivesse sido construído integralmente em C, por exemplo. E é uma ferramenta simples, fácil de usar e cumpre a sua função.

Futuramente, uma das melhorias que se pode fazer é a implementação de uma interface para o utilizador.

## Apêndice A

# Código do Programa

```
%{
#include <string.h>
char nameEspecial[9] = "{ %name%}\0";
char helper[2] = "\0";
struct file{
    char * name;
    char * path;
    struct file *next;
};
struct file *file_list = NULL;
int LOOKUP = -1;
char OPTION[20];
char NAME[50];
char EMAIL[50];
char AUTHOR[50];
int opt = 0;
int oldlvl = 0;
int newlvl = 0;
char dirName[50];
char pathName[100];
char fileName[50];
FILE * aux;
char auxfile[150];
int validate(char*);
int addFile(char*,char*);
int findFile(char*);
int getFileName(char*);
void printFileList();
}%

%s META TREE FILEMODE

%%
^=== " " * {
```

```

                                if(LOOKUP == 3){
                                    fclose(aux);
                                }
                                LOOKUP = 0;
                                BEGIN META;}
<META>[ \t]+                    {}
<META>e-?mail:                  {LOOKUP = 1;}
<META>author:                   {LOOKUP = 2;}
<META>[a-zA-z@.{ %}]+[ ]*[a-zA-z@.]* {
                                if(LOOKUP == 0){
                                    sprintf(OPTION," %s",yytext);
                                    opt = validate(OPTION);
                                    switch(opt){
                                        case 1:
                                            LOOKUP = -1;
                                            break;
                                        case 2:
                                            LOOKUP = -1;
                                            BEGIN TREE;
                                            break;
                                        case 3:
                                            getFileName(OPTION);
                                            findFile(fileName);
                                            strcpy(auxfile,pathName);
                                            strcat(auxfile,helper);
                                            strcat(auxfile,fileName);

                                            aux = fopen(auxfile, "w");
                                            yyout = aux;
                                            LOOKUP = 3;
                                            BEGIN FILEMODE;
                                            break;
                                        default:
                                            printf("why are you here?\n");
                                            break;
                                    }
                                }else if(LOOKUP == 1){
                                    sprintf(EMAIL," %s",yytext);
                                    LOOKUP = -1;
                                }else if(LOOKUP == 2){
                                    sprintf(AUTHOR," %s",yytext);
                                    LOOKUP = -1;
                                    BEGIN INITIAL;
                                }else{
                                    }}
<TREE>[ \t]*                    {}
<TREE>[\\-]*                    {newlvl = yyleng;}
<TREE>[a-zA-z@.{ %}]+/V        { if(strcmp(nameEspecial,yytext)== 0){

```

```

        strcpy(dirName,NAME);
    }else{
        strcpy(dirName,yytext);}
    if(newlvl == 0){ // root option
        mkdir(dirName, 0777);
        if(chdir(dirName) != 0){
            printf("didn't chage\n");
        }
        getcwd(pathName,100);
    }else{
        while(!(newlvl == oldlvl+1)){
            chdir("..");
            oldlvl--;
        }
        mkdir(dirName, 0777);
        if(chdir(dirName) != 0){
            printf("did'nt chage\n");
        }
        getcwd(pathName,100);
    }
    oldlvl = newlvl;
}
<TREE>[a-zA-Z@{ %}~.]+ {
    while(!(newlvl == oldlvl+1)){
        chdir("..");
        oldlvl--;
    }if(getFileName(yytext)){
        printf("ERROR, COULD'N MATCH THE FILENAME\n");
    }
    // add file to the structure and create the file.
    getcwd(pathName,100);
    if(!addFile(fileName,pathName)){
        printf("ERROR, FILE NOT ADDED TO THE STRUCTURE");
    }
    aux = fopen(fileName, "w");
    fclose(aux);
}
<FILEMODE>"{ %author %}" {fprintf(yyout," %s",AUTHOR);}
<FILEMODE>"{ %name %}" {fprintf(yyout," %s",NAME);}
<FILEMODE>"{ %email %}" {fprintf(yyout," %s",EMAIL);}
<FILEMODE>.\n {ECHO;}
.\n {}

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

```

```

int validate(char * option){

    if(!strcmp(option,"meta",strlen("meta"))){
        return 1; // META MODE
    }else if(!strcmp(option,"tree", strlen("tree"))){
        return 2; // TREE MODE
    }else {
        return 3; // FILE MODE
    }
}

int getFileName(char * text){

    if(strcmp(text,nameEspecial, 8)){ //don' t contain { %name%}
        strcpy(fileName,text);
        return 0;
    }else{ // files with { %name%}
        strcpy(fileName,NAME);
        strcat(fileName,text+8);
        return 0;
    }
    return 1;
}

int addFile(char *name,char*path){
    struct file *newfile;

    newfile = (struct file *) malloc(sizeof(struct file));
    newfile->next = file_list;
    newfile->name = (char *) malloc(strlen(name)+1);
    strcpy(newfile->name,name);
    newfile->path = (char *) malloc(strlen(path)+1);
    strcpy(newfile->path,path);
    file_list = newfile;

    return 1;
}

void printFileList(){
    struct file *fp = file_list;
    for(;fp;fp = fp->next){
        printf("f: %s\n", fp->name);
    }
}

int findFile(char * fileName){
    struct file *fp = file_list;

```

```

for(;fp;fp = fp->next){
    if(strcmp(fp->name, fileName)==0){
        strcpy(pathName,fp->path);
        return 0;
    }
}
return 1;
}

```

```

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

    int i = 0;
    FILE *file;

    if(argc < 3){
        printf("you should pass for the program a name and a template in this order\n");
    }
    else{
        sprintf(NAME," %s",argv[1]);
        file = fopen(argv[2], "r");
        if(!file){
            fprintf(stderr,"could not open  %s\n",argv[1]);
            exit(1);
        }
        yyin = file;
        yylex();
        printf("DONE\n");
    }

    return 0;
}

```