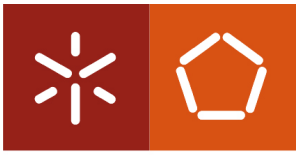


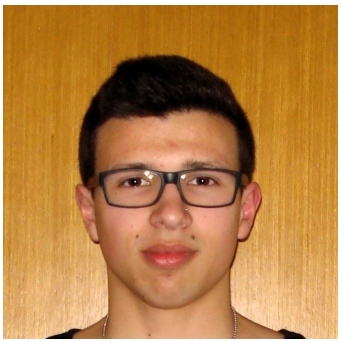
Universidade do Minho

LEI — Licenciatura de Engenharia Informática

Processamento de Linguagens

Trabalho pratico 1

Paulo Araujo - a58925, Orlando Costa - a67705, Rui Oliveira - a67661



Braga, 25 de março de 2015

Resumo

Este relatório descreve a resolução de um conjunto de exercícios propostos, que consistem no desenvolvimento de programas na linguagem C com o auxílio de geradores de filtros de texto, como o Flex. Para cada problema é realizada uma breve análise sobre o trabalho efetuado, as decisões que lideraram o seu desenvolvimento e as estruturas implementadas, assim como uma explicação do seu funcionamento.

Os problemas resolvidos consistem no desenvolvimento de filtros de texto que:

- processa um ficheiro XML com descrições de fotografias e gerar um álbum HTML.
- processa de um ficheiro XML anotado com tags Enamex e gera páginas HTML apresentando as "pessoas", "países", "cidades"e organizações nele identificadas.
- processa vários ficheiros de texto, compostos por letras de canções, e gera documentos em LATEX para cada uma delas.

Para cada problema é apresentado o código em linguagem C e as expressões regulares desenvolvidas, sendo estes suportados por exemplos e devidos resultados.

Conteúdo

Capítulo 1

Introdução

O presente projeto enquandra-se na unidade curricular de Processamento de Linguagnes do curso de Licenciatura em Engenharia Informática da Universidade do Minho. O projeto pretende aumentar as capacidades com as expressões regulares, desenvolvendo processadores de linguagens regulares utilizando o gerador de filtros de texto Flex. Para isso foram selecionados 3 exercícios dentro de um grupo de 8 exercícios, são eles: *2.1 Museu da Pessoa — tratamento de fotografias*, *2.2 Processamento de Entidades Nomeadas (Enamex)* e *2.5 Processamento de ficheiros com Canções*.

Capítulo 2

Museu da Pessoa — tratamento de fotografias

Neste problema é pretendido que, a partir de um ficheiro *XML*, seja criado um filtro de texto capaz de interpretar informação relativa às entrevistas feitas para construção o Museu da Pessoa. Com toda essa informação, deverá ser criado um álbum em *HTML* possuidor de um índice, ordenado alfabeticamente, contendo todos os nomes de pessoas presentes no álbum. Além disso, cada elemento do índice deverá estar referenciado para a página que contém todas as fotos da respetiva pessoa. Relativamente às fotos, essas deverão estar ordenadas cronologicamente e a descrição da foto deverá ser o seu título/cabeçalho.

2.1 Analise e Especificação

Após a análise do que é pedido no enunciado e de alguns dos *Datasets*, foi possível verificar qual a informação essencial a retirar do ficheiro *XML*. Além disso, como não é regra que as fotos e pessoas presentes no ficheiro *XML* estejam já na ordem pretendida, será necessário que toda a informação seja armazenada em estruturas de dados. O nome do ficheiro *HTML* resultante, poderá ser dado como argumento e caso não seja, por omissão será *AlbumGerado.html*. Além deste ficheiro, serão gerados tantos ficheiros *HTML* quantas as pessoas presentes no álbum. Esses ficheiros serão denominados numericamente (1.html,2.html,...) à medida que novos nomes são encontrados. Os ficheiros anteriormente referidos apenas serão gerados no final da leitura e filtragem de ficheiro *XML*.

2.2 Implementação

2.2.1 Estrutura de dados

De forma a dar seguimento ao que foi especificado na secção 2.1, foi necessária a criação de uma estrutura de dados que sirva de suporte aos dados recolhidos. Sendo assim, optamos por criar uma estrutura denominada `photo_node` que terá o formato de uma árvore binária e servirá para

armazenar toda a informação relativa a cada descrição de fotos encontrada no ficheiro *XML*. Nesta estrutura serão guardados o nome do ficheiro que contém a foto, a data e o local em que esta foi tirada, a sua descrição e o nome das pessoas nela presentes. A inserção de fotos nesta árvore será ordenada relativamente à sua data. Além desta estrutura, também criamos outra, denominada `person_node`, com o objetivo que esta guarde toda a informação acerca das pessoas presentes no álbum. Esta estrutura tem o formato de uma lista ligada e guardará informação como o nome da pessoa, assim como o nome do ficheiro da sua página *HTML* e terá ainda um apontador para uma estrutura `photo_node` em que será armazenada toda a informação relacionada com as suas fotos. Finalmente, criamos uma estrutura `Album` que apenas conterá um apontador para uma estrutura `person_node`, onde estará a informação relativa às pessoas nele presente, e um contador que servirá para contar o número de pessoas presentes no álbum. O contador servirá de auxílio na criação dás páginas *HTML* numeradas referidas na secção 2.1.

2.2.2 Filtro de Texto

Um dos objetivos deste trabalho prático é a utilização de geradores de filtro de texto, como o *Flex*. Sendo assim, foi criado um ficheiro Flex que permite encontrar determinados padrões de expressões regulares e executar uma determinada ação para cada uma delas.

No ficheiro referido, podemos encontrar algumas instruções em `linguagem C` como a inclusão de ficheiros de cabeçalhos (headers, .h) e a declaração de variáveis. De seguida, são definidas as expressões regulares e as respetivas ações que se pretendem realizar no caso da identificação positiva do referido padrão:

`\<foto[ \t]+[a-zA-Z]+=\".*\"` A partir da análise do ficheiro *XML* exemplificado no enunciado, foi possível verificar que a descrição de uma foto começa com o nome do ficheiro que a contem da seguinte forma: `<foto ficheiro="ficheiro.jpg">`. Sendo assim, quando este padrão é encontrado, é inicializado um `photo_node` com o nome encontrado entre as aspas. O nome é obtido retirando da expressão apenas o que se encontra à frente da primeira ocorrência de aspas e atrás da ocorrência seguinte.

`\<quando[ \t]+[a-zA-Z]+=\"[0-9]{4}(.|-)[0-9]{2}(.|-)[0-9]{2}` De seguida pretendemos encontrar a data em que a foto foi tirada. A partir do exemplo do enunciado, verificamos que a data é descrita da seguinte forma: `<quando data="1961-01-15"/>`. Sendo assim, pretendemos encontrar todas as expressões no formato referido, com a possibilidade que a data esteja separada por pontos em vez de traços. A obtenção da data faz-se retirando apenas o que se encontra à frente da primeira ocorrência de aspas.

`\<quem>[ \t\na-zA-ZÀ-û,\"0-9;:]+` Um padrão essencial a encontrar é o nome das pessoas presentes na foto. Esses nomes devem estar descritos da seguinte forma:
`<quem>Ana de Lourdes de Oliveira Chamine; Antonio Oliveira Machado</quem>`
Nesta fase, optamos por não separar ainda os nomes por tokens porque será mais útil fazê-lo apenas quando a descrição da foto estiver completa. Sendo assim, apenas retiramos tudo o que esteja entre `'>'` e `'<'` e adicionamos à estrutura que contem a descrição da foto.

`\<onde>[ \t\n0-9a-zA-ZÀ-û.,;:\"]+<` Continuando com a análise do exemplo do enunciado, verificamos que uma foto pode ter descrito o local em que esta foi tirada:
`<onde>Casa Machado, Afurada, Vila Nova de Gaia</onde>`
Ora, quando este padrão é encontrado, só é necessário guardar o que está entre `'>'` e `'<'`, assim como no caso anterior.

`\<facto>[ \t\n0-9a-zA-ZÀ-û.,;:\"]+<` Finalmente, o último campo necessário para a descrição da foto é o facto que esta representa: `<facto> Os noivos cortam o bolo de casamento</facto>`. O que é essencial retirar neste caso é o mesmo que nos casos anteriores, ou seja, apenas o que se encontra entre '>' e '<'.

`\<\/foto>` Sempre que é encontrado o padrão que finaliza a descrição de uma foto, `</foto>`, é necessário adicionar o `photo_node` criado a todos os respetivos `person_node` das pessoas que se encontram na descrição da foto. É nesta fase que os nomes das pessoas presentes na foto é separado e identificado. Finalmente adiciona-se a descrição da foto a todas as pessoas nela presente.

`.|\n` Este padrão apenas é utilizado para indicar que sempre que é encontrado qualquer outro `byte` ou `\n`, deve ser ignorado.

2.2.3 Funcionamento

Antes do filtro de texto ser aplicado ao ficheiro *XML* é invocada a função `init` que inicializa uma variável `static Album`. Inicializada esta variável, é possível aplicar o filtro de texto. À medida que a informação do ficheiro *XML* é filtrada, são invocadas funções que tratam e guardam na estrutura a informação recolhida. Sempre que é encontrada uma expressão que defina o início de uma descrição de uma foto, a informação relativa ao nome da foto é recolhida e é invocada a função `initPhoto` que trata de alocar o espaço necessário para a informação que virá a ser recolhida posteriormente. Do mesmo modo, sempre que é encontrada uma expressão que identifique a data, a localização, a descrição ou as pessoas presentes na foto, são invocadas funções capazes de tratar e armazenar essa informação. As funções referidas são `setDate, setLoc, setFact e setWho`, respetivamente. Sempre que não seja possível identificar uma destas características anteriores, estas ficam com os seus valores por defeito, ou seja, caso uma foto não tenha a tag `<quem>`, o seu campo correspondente na estrutura ficará preenchido com `"Desconhecidos"`. Finalmente, quando é encontrada a expressão que define o final da descrição de uma foto, é invocada a função que trata de inserir a informação da nova foto na estrutura `Album`. Depois de tratada toda a informação, são invocadas funções auxiliares que criam e preenchem o(s) ficheiro(s) *HTML* necessários.

2.3 Testes realizados

<alguns exemplos>

Possíveis exemplos de aplicação deste filtro de texto:

- `cat\ <inputFile>\ |\ ./play\ <output1.html>\`
- `cat\ <inputFile>\ |\ ./play\`

Capítulo 3

Processamento de Entidades Nomeadas

Entidades nomeadas são "elementos atómicos em texto"pertencentes a categorias predefinidas tais como nomes de pessoas, organizações, localizações, quantidades, etc. Assim, Processamento de Entidades Nomeadas(PEN) é a tarefa de identificar estas entidades. Embora as categorias das entidades nomeadas serem predefinidas, existem várias opiniões sobre que categorias deve ser consideradas entidades nomeadas e quão abrangentes estas categorias devem ser. Por convenção, tags *"ENAMEX"* são utilizadas para nomes, tags *"NUMEX"* são utilizadas para entidades numéricas, e tags *"TIMEX"* são utilizadas para entidades temporais.

Neste exercício iremos apenas processar entidades com a tag *"ENAMEX"*, na forma:

- `<ENAMEX TYPE="PERSON">Francisco de Vilela Barbosa</ENAMEX>`
  (Pessoa)
- `<ENAMEX TYPE="LOCATION" SUBTYPE="COUNTRY">Portugal</ENAMEX>`
  (Localização, País)
- `<ENAMEX TYPE="LOCATION" SUBTYPE="CITY">Rio de Janeiro</ENAMEX>`
  (Localização, Cidade)
- `<ENAMEX TYPE="ORGANIZATION">Universidade do Minho</ENAMEX>`
  (Organização)

Como exercício extra iremos também abordar as entidades na forma:

- `<ENAMEX TYPE="LOCATION"> Santo Novo </ENAMEX>`
  (Localização não específica)

Todas as outras tags irão ser ignoradas.

O processamento de entidades nomeadas, apesar de ser aparentemente uma tarefa simples, enfrenta um dado numero de desafios. As entidades podem tornar-se difíceis de encontrar, e uma vez encontradas, difíceis de classificar. Localizações e nomes de pessoas podem ser as mesmas, e seguir estilos similares de formatação.

3.1 Analise e Especificação

Uma breve leitura do problema permite-nos entender algumas das funcionalidades necessárias, sendo estas definidas como:

- Necessidade de ordenação e não repetição na listagem de pessoas(alínea a):

  - A alínea A do problema requer a listagem de todas as pessoas identificadas, sem repetições. Este informação refere-se então às tags do tipo:
    `<ENAMEX TYPE="PERSON">...</ENAMEX>`.
    Por forma a armazenar e ordenar adequadamente toda a informação acerca das Pessoas, é necessária a utilização de uma estrutura capaz de suportar esta informação.

- Listar os países e cidades marcadas (alínea b):

  - Apesar de não estar especificado no enunciado, o grupo propôs uma implementação na qual seria possível associar cidades a certos países. Compreendemos que este tipo de implementação, para grande parte dos casos, não é viável e pode tornar a informação apresentada incoerente. No entanto, no intuito de aprender e aumentar o desafio proposto, decidimos que cidades mencionadas após países e antes de pontos finais pertenciam a esses países. Esta informação é também armazenada na estrutura implementada.

- Listar as organizações (alínea c):

  - Similarmente à alínea A, a alínea C requer a listagem de todas as organizações identificadas. Esta informação refere-se então às tags do tipo:
    `<ENAMEX TYPE="ORGANIZATION">...</ENAMEX>`
    E está implementada de forma similar à listagem de pessoas.

- Apresentar os resultados em formato HTML:

  - Por forma a visualizar facilmente os resultados do processamento do texto, estes são apresentados em formato HTML através da implementação de funções capazes de transformar a informação contida nas estruturas em documentos de texto com o formato requerido.

3.2 Implementação

3.2.1 Estrutura de dados

Com o intuito de cumprir todos os requisitos estruturais definidos anteriormente, foi desenvolvida uma estrutura de dados única capaz de armazenar todos os dados necessários. Assim, escolhemos implementar uma árvore binária de procura, na qual os nodos possuem a informação a guardar sobre a forma de array de caracteres. A escolha desta estrutura facilita a ordenação alfabética dos diversos nomes que possamos processar, e é de implementação relativamente simples. Creemos ser superior a outras estruturas tais como listas ligadas cuja implementação, apesar de mais

simples, torna-se mais complexa quando é necessária a ordenação dos seus elementos (O(N)) e a tabelas de hash cuja implementação é mais complexa, sendo que para elevadas quantidades de dados possuem ainda a necessidade de reHashing e garbage colection.

<-tree.png-> ???

Para casos em que apenas é necessário o armazenamento da entidade, sem qualquer tipo de associação, uma árvore com dois apontadores (esquerda e direita) e com a capacidade de armazenar a informação (array de caracteres) bastaria para abordar todos os casos. No entanto, de forma a armazenar a possível associação entre os países e as suas cidades, modificamos a árvore previamente referida, e adicionamos-lhe um apontador extra, que poderá ser visto como o apontador para a raiz de uma nova árvore, constituída por todas as cidades que pertencem a um dado país. Como curiosidade, adicionamos também um inteiro em cada nodo que servirá como contador para todas as ocorrências de um dado elemento. Todo o código referente a esta estrutura encontra-se implementado no ficheiros "tree.c"e "tree.h"e pode ser consultado em anexo.

3.2.2 Filtro de Texto

De forma a processar as tags pertencentes a pessoas/organizações, foram criadas expressões regulares capazes de identificar essas tags. Assim, foram implementadas as seguintes expressões regulares para pessoas/organizações, assim como abreviaturas que facilitam a sua leitura e compreensão:

- pessoas: `{enamex}{person}{pal}{eclose}`
- organizações:`{enamex}{org}{pal}{eclose}`
- localizações gerais: `{enamex}{loc}{pal}{eclose}`
- cidades: `{enamex}{loc}{subcity}{pal}{eclose}`

sendo que as abreviaturas mencionadas correspondem a:

- Palavra
  `{pal} [a-zA-Z0-9Ç-ÑÀ-û ]+`
- Tag Enamex
  `{enamex} \<[ \t]*(?i:enamex)[ \t]+(?i:type)=`
- Tag de fecho
  `{eclose} \<[ \t]*\/(?i:enamex)[ \t]*\>`
- Elemento "Organization"
  `{org} \"[ \t]*(?i:organization)[ \t]*\"[ \t]*\>`
- Elemento "Person"
  `{person} \"[ \t]*(?i:person)[ \t]*\"[ \t]*\>`

De notar a capacidade das expressões regulares identificarem tags definidas tanto em letra maiuscula como minuscula, sendo também tolerantes à quantidade de espaços ou tabs presentes entre elementos destas. De forma a implementar a capacidade de associar cidades a um dado país, foram utilizados "operadores de contexto", de forma a que caso seja detectado uma tag correspondente a um país, o analisador entre no contexto não exclusivo (%s)country, e associa as seguintes tags correspondentes a cidades ao país em causa. Caso seja detetado um ponto final, o analisador lexico abandona esse contexto e continua a processar em contexto geral.

Nota: Esta foi uma funcionalidade assumida, pelo que poderá nem sempre ter sucesso e fazer as corretas associações.

Assim as expressões regulares correspondentes a localizações/países/cidades foram definidas na forma:

- paises:
  `{enamex}{loc}{subcountry}{pal}{eclose}`
- ponto final no contexto "country":
  `<country>`
- cidades no contexto "country":
  `<country>{enamex}{loc}{subcity}{pal}{eclose}`

sendo que as abreviaturas mencionadas correspondem a:

- Elemento "Country"
  `{subcountry}  (?i:subtype)=\"[ \t]*(?i:country)[ \t]*\"[ \t]*\>`
- Elemento "City"
  `{subcity} (?i:subtype)=\"[ \t]*(?i:city)[ \t]*\"[ \t]*\>`

Estas expressões devem encontrar-se no topo de todas as outras, devido à precedência que possuem sobre elas.

3.2.3 Funcionamento

No cabeçalho do ficheiro flex são declarados todos os apontadores referentes às estruturas onde irá ser armazenada a informação. Existe um apontador para cada tipo de estrutura, nomeadamente: pessoas, paises, cidades, organizações e outras localizações. São também declarados dois apontadores para arrays de caracteres que irão auxiliar o processamento das expressões capturadas. Após início do programa, é efetuada a chamada ao analisador léxico, responsável por capturar os dados referentes às expressões definidas. Cada vez que este efetua uma captura, estes dados são processados, sendo inseridos na estrutura correspondente, sendo que caso necessário lhes são retirados os espaços em branco que possam ter antes e depois da seu conteúdo, por forma a evitar inconsistência de dados. Após o término da leitura do input em questão, todos os dados presentes nas estruturas são escritos nos ficheiros HTML correspondentes, estando estes interligados através de hiperligações (tags <a href=/>). Cada estrutura (tipo de entidade) é escrita em ficheiro através da função treeToHTML (implementada no ficheiro tree.c), responsável por receber como parâmetros um apontador para estrutura e um identificador de ficheiro (previamente declarado), e transferir a informação para o ficheiro no formato adequado.

<- constituiçao do programa.png ->

3.3 Testes realizados

<alguns exemplos>

Capítulo 4

Processamento de ficheiros com Canções

Neste problema era pretendido que fosse criado um filtro de texto capaz de interpretar ficheiros com letras de músicas, e fosse gerado um ficheiro *latex* para cada música encontrada, contendo a informação processada. Ainda existe a particularidade de que cada ficheiro com músicas poder conter mais do que uma música, sendo que neste caso devem ser criados 2 ficheiros *latex*.

4.1 Analise e Especificação

Existem várias questões que são deixadas em aberto no enunciado que irão ser especificadas nesta secção. O programa lê do *stardard* input e os nomes dos ficheiros *latex* que irão ser gerados podem ser recebidos como argumento. Caso estes não sejam especificados, assumem um título numerado, começando em 0 até à n-ésima musica interpretada. Uma vez que não se sabe a ordem pela qual os cabeçalhos se apresentam nos ficheiros a serem interpretados, o mais seguro será guardar todas as musicas em memoria e apenas imprimir para o ficheiro *latex* após o fim de cada música. Após a análise dos *Datasets* verificou-se a existência de campos no cabeçalho que não são utilizados pelo programa, devendo estes ser ignorados. Durante a análise foi também verificada a existência de anotações em algumas músicas que serviriam para apresentar as pautas, pelo que optamos por ignorar estas marcas e tentar apenas imprimir a letra da música do ficheiro *latex*. Existe ainda outro cuidado na criação do ficheiro *latex* que é apresentação na música de caracteres especiais no *latex*.

4.2 Implementação

4.2.1 Estrutura de dados

De forma a complementar o enunciado na secção 4.1, foi criada uma estrutura principal denominada `Music`, onde se guarda a informação geral da música temporariamente até esta ser imprimida para um ficheiro.

Nesta estrutura ir-se-à guardar o titulo, o nome do autor e a letra da música, entre outros campos do cabeçalho que possam ser necessários.

A letra da musica é guardada numa lista ligada onde cada nodo é um linha da letra e é representada pela estrutura `MusicLine`. A estrutura pode ser encontrada em anexo 6.3.2.

4.2.2 Filtro de Texto

Para a filtragem do texto foram criadas várias expressões regulares, sendo que o ficheiro pode ser encontrado em anexo (6.3.1).

As primeiras expressões regulares, do tipo `^title:.+` servem para capturar os cabeçalhos que serão necessários. Existem também as expressões regulares do tipo: `from`, `author`, `lyrics`, `music` e `singer`, todas equivalentes. De forma a ignorar qualquer outro campo do cabeçalho que não tivesse sido previsto foi ainda criada a seguinte expressão regular: `^[a-zA-Z]+:.+`.

Quanto à deteção da letra da música existem duas expressões regulares: uma para capturar uma linha da letra, outra para capturar as linhas em branco entre os poemas, que são respetivamente: `[ ].*` e `$^\n`.

Tal como foi dito na análise (4.1), existem algumas anotações no meio da letra da música que são necessárias retirar. Para isso foram criadas as seguintes expressões regulares:

- `{abc}(.|\n)*{abcclose}` para retirar a pauta da musica.
- `[ ].*` que ignora as notas no meio dos poemas (pois estas tem um espaço no inicio).

Ainda assim estas duas expressões regulares não eram suficientes e na deteção de uma linha da letra, antes de guardar a linha, processa-se a linha com o auxílio de duas funções: `takeOffAnotations` e `takeOffUnderSccore`. A primeira retira anotações que estejam na mesma linha, e a segunda tira os caracteres '_' que estejam no meio da linha.

4.2.3 Funcionamento

De forma a perceber melhor o funcionamento do autómato esta secção irá detalhar a ponte entre o filtro de texto (4.2.2) e a estrutura de dados (4.2.1).

À medida que o autómato captura os campos do cabeçalho da música, guarda a informação com as funções de `append`, p.e. `appendAuthor`, `appendLyrics`, entre outras. Estas funções guardam os campos na variável `Music`.

As linhas da letra são guardadas através das funções `appendLine` e `appendWhiteLine`.

Quando é detetado o início de uma nova música, através de expressão regular, é executado `commitCheckNext()` que escreve a letra que está atualmente na variável `Music` para o ficheiro *latex*. Caso seja detetada a falta de algum item obrigatório então a escrita para o ficheiro é cancelada. De seguida a variável `Music` é reiniciada para a música seguinte com a função `Start()`.

Na escrita do ficheiro *latex* a letra é escrita entre as *tags* da *latex* de *Verbatim* para evitar erros no *latex* por falta de caracteres escape.

4.2.4 Testes realizados

Estão documentados neste secção 3 testes realizados ao autómato, utilizando como input os ficheiro que estão em anexo: 6.3.4, 6.3.4 e 6.3.4.

Teste nº 1

Após a utilização do autómato no ficheiro 6.3.4, este gerou o output (6.3.4). Este ficheiro não tem nenhuma situação excecional, é um caso normal.

Teste nº 2

Após a utilização do autómato no ficheiro 6.3.4, este gerou o output (6.3.4). Este ficheiro tem duas situações excecionais, o carácter '_' no meio de palavras e notas musicais no fim das frases. Podemos verificar no output que apenas tem a letra da musica.

Teste nº 3

Após a utilização do autómato no ficheiro 6.3.4, este gerou o output (6.3.4). Este ficheiro tem uma situação excecional, antes da letra da musica tem as tags `<abc>...</abc>` com anotações de notas musicas. Podemos verificar que no output já não está presente.

Um exemplo de uma possível utilização do autómato é:
`cat <inputFile> | ./play <output1.tex> <output2.tex> ...`

Capítulo 5

Conclusão

Terminado o desenvolvimento do trabalho, é importante referir que o mesmo nos permitiu aprofundar o conhecimento acerca do Gerador Léxico Flex assim como da análise léxica no geral, obrigando-nos também a utilizar ferramentas tais como HTML e Latex. Relativamente ao problema do "Museu da Pessoa", a dificuldade recaiu na definição da estrutura de suporte de dados, dado que devido à falta de claridade do enunciado foi necessário re-implementar a estrutura de forma a que esta admitisse a funcionalidade de ter um índice geral em HTML. No problema de "Processamento de Entidades Nomeadas"foi necessário chegar a um consenso acerca das tags que deveriam ser validadas e o relacionamento possível que estas teriam entre si. Após essa decisão, o desenvolvimento da estrutura que suporta esta informação tornou-se relativamente simples. O problema "Processamento de Ficheiros com Canções"foi resolvida através da implementação de uma estrutura capaz de evitar que a ordem dos dados no ficheiro não seja significativa (e então armazena em memória a informação). A presença de certos elementos em datasets mais diversos (tais como header's não esperados e anotações em músicas) deram origem a problemas, sendo que a solução consistiu em ignorar essa informação. Cada elemento do grupo realizou um exercício do enunciado proposto, apoiando-se mutuamente na existência de dificuldades. Apesar das dificuldades iniciais, encontramos-nos satisfeitos com o resultado final e estamos confiantes para o próximo trabalho.

Capítulo 6

Anexos

6.1 Museu da Pessoa — tratamento de fotografias

6.1.1 Filtro de Texto

```
%{
#include "album.h"
char* token;
photo_ptr picture;
%}

%%

\<foto[ \t]+[a-zA-Z]+=\".*\" {
yytext[strlen(yytext)-1]='\0';
picture=initPhoto(strchr(yytext,'"')+1);
}
\<quando[ \t]+[a-zA-Z]+=\"[0-9]{4}(.|-)[0-9]{2}(.|-)[0-9]{2} {
setDate(picture,strchr(yytext,'"')+1);
}
\<quem>[ \t\na-zA-ZÀ-û,\"0-9;:]+ {
token=strchr(yytext,'>')+1;
setWho(picture,token);
}
\<onde>[ \t\n0-9a-zA-ZÀ-û.,;:\"]+<{
yytext[strlen(yytext)-1]='\0';
setLoc(picture,strchr(yytext,'>')+1);
}
\<facto>[ \t\n0-9a-zA-ZÀ-û.,;:\"]+<{
yytext[strlen(yytext)-1]='\0';
setFact(picture,strchr(yytext,'>')+1);
}
\<\/foto>{
```

```
if(picture) {
token=strdup(picture->who);
token=strtok(token,";");
while (token){
insert_photo_person(token, picture);
token=strtok(NULL,";");
}
}
picture=NULL;
}
.|\n ;

%%
```

6.1.2 Estrutura de dados

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "album.h"

static Album album;

char* trimspace(char *str)
{
        char *end;

        while(isspace(*str)||(*str)=='\n') str++;

        if(*str == 0)
        return str;

        end = str + strlen(str) - 1;
        while(end > str && isspace(*end)) end--;

        *(end+1) = 0;
        return str;
}

char* subsPoints(char *str){
        int i;

        for(i=0; i<strlen(str);i++){
                if(str[i]=='.')
                        str[i]='-';
        }
        return str;
```

```
}

void insert_photo(photo_ptr p, photo_ptr *pics){
    if ((*pics) == NULL)
    {
        (*pics)=p;
        (*pics)->left=NULL;
                (*pics)->right=NULL;
        }
    else{
        int comp=strcmp(p->date, (*pics)->date);
        if(comp < 0)
        {
                insert_photo(p, &(*pics)->left);
        }
        else {
                insert_photo(p, &(*pics)->right);
        }
    }
}


void insert_photo_person(char *name, photo_ptr p){

    person_ptr aux, newPers, lastPers;
    int done=0, comp;

    name=trimspace(name);

    if (album.people == NULL)
    {
        album.people=malloc(sizeof(person_node));
        name=trimspace(name);
        album.people->name=strdup(name);
        album.people->href=malloc(16*sizeof(char));
        sprintf(album.people->href,"%d",album.count);
        album.people->href=strcat(album.people->href,".html");
        album.people->pics=p;
        album.count++;
        album.people->next=NULL;
        }
    else{
        for(aux=album.people;aux && (comp=strcmp(aux->name,name))<0;aux=aux->next){
                lastPers=aux;
        }

        if (comp==0){
                        insert_photo(p,&aux->pics);
                }
```

```
                else {
                        newPers = malloc(sizeof(person_node));
                        newPers->name=strdup(name);
                        newPers->href=malloc(16*sizeof(char));
                        sprintf(newPers->href,"%d",album.count++);
                        newPers->href=strcat(newPers->href,".html");
                        newPers->pics=p;
                        if(!aux) {
                                newPers->next=NULL;
                                if(lastPers) lastPers->next=newPers;
                        }
                        else {
                                newPers->next=aux;
                                if (lastPers) lastPers->next=newPers;
                                else album.people=newPers;
                        }
                }
        }
}

void setDate(photo_ptr p, char *photoD){
        photoD=trimspace(photoD);
        photoD=subsPoints(photoD);
        p->date=strdup(photoD);
}

void setLoc(photo_ptr p, char *photoL){
        photoL=trimspace(photoL);
        p->loc=strdup(photoL);
}

void setFact(photo_ptr p, char *photoF){
        photoF=trimspace(photoF);
        p->fact=strdup(photoF);
}

void setWho(photo_ptr p, char *photoW){
        photoW=trimspace(photoW);
        p->who=strdup(photoW);
}

photo_ptr initPhoto(char *photoN){
        photo_ptr pic = malloc(sizeof(photo_node));
        pic->file=strdup(photoN);
        pic->date=strdup("Desconhecida");
        pic->loc=strdup("Desconhecido");
        pic->fact=strdup("Desconhecida");;
        pic->who=strdup("Nao_identificados");
```

```
        pic->left=NULL;
        pic->right=NULL;
        return pic;
}

void init(){
        album.people=NULL;
        album.count=1;
}

void appendPhoto(photo_ptr p, FILE* file){

        if(p!=NULL) {

                appendPhoto(p->left, file);
                fprintf(file, "  <div class=\"photo\">\n</br>\n");
                fprintf(file, "    <div class=\"fact\"><h3>Descricao:</h3> %s</div>\n
                fprintf(file, "    <div class=\"img\" align=\"center\">\n");
                fprintf(file, "      <img src=\"%s\" alt=\"%s\" width=800 height=600>
                fprintf(file, "    <div class=\"data\"><b>Data:</b> %s</div>\n",p->da
                fprintf(file, "    <div class=\"local\"><b>Local:</b> %s</div>\n",p->
                fprintf(file, "    <div class=\"who\"><b>Quem:</b> %s</div>\n",p->who
                fprintf(file, "  </div>\n");
                appendPhoto(p->right, file);

        }
}

void createPage(char* filename, char* name, photo_ptr p){
        FILE* file;
        photo_ptr pic_ptr;

        file=fopen(filename,"w+");

        fprintf(file, "<!DOCTYPE html>\n<html>\n");
        fprintf(file, "<head>\n  <meta charset=\"UTF-8\">\n</head>\n<body>\n");

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

        fprintf(file, "<div class=\"title\" align=\"center\">\n");
        fprintf(file, "  <h1>Album de fotografias</h1>\n");
        fprintf(file, "  <h2>%s</h2>\n</div>\n</br>\n</br>\n", name);

        fprintf(file, "<div class=\"gallery\">\n");
        appendPhoto(p, file);
        fprintf(file, "</div>\n</body>\n</html>");

        fclose(file);
}
```

```
void createAlbum(char* filename) {
        FILE* file;
        person_ptr aux = album.people;

        file=fopen(filename,"w+");

        fprintf(file, "<!DOCTYPE␣html>\n<html>\n");
        fprintf(file, "<head>\n␣␣<meta␣charset=\"UTF-8\">\n</head>\n<body>\n");

        fprintf(file, "<div␣class=\"title\"␣align=\"center\">\n");
        fprintf(file, "␣␣<h1>Album␣de␣fotografias</h1>\n</div>\n");

        fprintf(file, "<div␣class=\"index\">\n");
        fprintf(file, "␣␣<div␣class=\"indextitle\"␣␣align=\"center\">\n");
        fprintf(file, "␣␣␣␣<h2>Indice␣de␣Pessoas</h2>\n␣␣</div>\n");
        fprintf(file, "␣␣<div␣class=\"indexitems\">\n␣␣␣␣<ul>\n");

        while(aux){
        fprintf(file, "␣␣␣␣␣␣<li><a␣href=\"%s\">%s</a></li>\n", aux->href,aux->name);
        createPage(aux->href,aux->name, aux->pics);
        aux=aux->next;
    }

        fprintf(file, "␣␣␣␣</ul>\n␣␣</div>\n</div>\n");
        fprintf(file, "</body>\n</html>");

        fclose(file);
}


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

        init();
        yylex();


        if(argc==2){
                createAlbum(argv[1]);
        }
        else{
                createAlbum("AlbumGerado.html");
        }
        return 0;
}
```

6.1.3 Cabeçalho ficheiro C

```
#ifndef __album_h__
```

```
#define __album_h__

typedef struct sPhoto *photo_ptr;

typedef struct sPhoto {
        char* file;
        char* date;
        char* loc;
        char* fact;
        char* who;
        photo_ptr left;
        photo_ptr right;
} photo_node;

typedef struct person *person_ptr;

typedef struct person
{
    char* name;
    char* href;
    photo_ptr pics;
    person_ptr next;
} person_node;

typedef struct sAlbum {
        person_ptr people;
        int count;
} Album;


void setDate(photo_ptr p, char *photoD);
void setLoc(photo_ptr p, char *photoL);
void setFact(photo_ptr p, char *photoF);
void setWho(photo_ptr p, char *photoW);
photo_ptr initPhoto(char *photoN);
void insert_photo_person(char *name, photo_ptr p);

#endif
```

6.2 Processamento de Entidades Nomeadas (Enamex)

6.2.1 Filtro de Texto

```
%s country

%{
    #include "tree.h"
```

```
    char *aux;
    char* aux_subcity;
    int aux_len;
    tree_ptr persons=NULL;
    tree_ptr countrys = NULL;
    tree_ptr citys = NULL;
    tree_ptr organizations = NULL;
    tree_ptr otherlocations = NULL;
    char* trimspace(char *str);
%}

pal         [a-zA-Z0-9Ç-ÑÀ-û ]+
enamex      \<[ \t]*(?i:enamex)[ \t]+(?i:type)=
eclose      \<[ \t]*\/(?i:enamex)[ \t]*\>
org         \"[ \t]*(?i:organization)[ \t]*\"[ \t]*\>
person      \"[ \t]*(?i:person)[ \t]*\"[ \t]*\>
loc         \"[ \t]*(?i:location)[ \t]*\"[ \t]*(\>)?
subcountry  (?i:subtype)=\"[ \t]*(?i:country)[ \t]*\"[ \t]*\>
subcity     (?i:subtype)=\"[ \t]*(?i:city)[ \t]*\"[ \t]*\>

%%

    BEGIN 0;

{enamex}{loc}{subcountry}{pal}{eclose}    {
        aux=strdup(strchr(yytext,'>')+1);
        *strchr(aux,'<')='\0';
        insert(trimspace(aux), &countrys);
        BEGIN country;
    }

<country>\.                       BEGIN 0;

<country>{enamex}{loc}{subcity}{pal}{eclose}   {
        tree_ptr aux_tree = search_tree(aux, countrys);
        aux_subcity=strdup(strchr(yytext,'>')+1);
        *strchr(aux_subcity,'<')='\0';
        insert_subtree(trimspace(aux_subcity), &countrys);
        insert(trimspace(aux_subcity), &citys);
        BEGIN 0;
    }
{enamex}{person}{pal}{eclose}   {
        aux=strdup(strchr(yytext,'>')+1);
        *strchr(aux,'<')='\0';
        insert(trimspace(aux), &persons);
    }
{enamex}{org}{pal}{eclose}  {
        aux=strdup(strchr(yytext,'>')+1);
        *strchr(aux,'<')='\0';
```

```
        insert(trimspace(aux), &organizations);
    }
{enamex}{loc}{subcity}{pal}{eclose}   {
       aux=strdup(strchr(yytext,'>')+1);
       *strchr(aux,'<')='\0';
       insert(trimspace(aux), &citys);
   }
{enamex}{loc}{pal}{eclose}  {
        aux=strdup(strchr(yytext,'>')+1);
        *strchr(aux,'<')='\0';
        insert(trimspace(aux), &otherlocations);
    }

<*>.|\n                     ;


%%

/*Função que retira os espaços brancos dos nomes*/
char* trimspace(char *str)
{
    char *end;

    while(isspace(*str)) str++;

    if(*str == 0)
        return str;

    end = str + strlen(str) - 1;
    while(end > str && isspace(*end)) end--;

    *(end+1) = 0;
    return str;
}

void printHeader(FILE *f){

    fprintf(f, "<DOCTYPE !html>\n<html>\n\t<head> <meta charset=\"UTF-8\">\n Grupo 7</head>\n");
    fprintf(f, "<link type=\"text/css\" rel=\"stylesheet\" href=\"stylesheet.css\"/>");
    fprintf(f, "\t<title>Processamento de Linguagens</title><body>");

}

void createIndex(FILE *f){
    printHeader(f);
    fprintf(f, "<h1>Processamento de Linguagens</h1></a>\n");
    fprintf(f, "<h2>Processamento de Entidades Nomeadas</h2></a>\n");
    fprintf(f, "<a href=\"persons.html\"><p class=\"ind\">Persons</p></a>\n");
    fprintf(f, "<a href=\"countrys.html\"><p class=\"ind\">Countrys</p></a>\n");
```

```
    fprintf(f, "<a href=\"organizations.html\"><p class=\"ind\">Organizations</p></a>\n");
    fprintf(f, "<a href=\"citys.html\"><p class=\"ind\">City's</p></a>\n");
    fprintf(f, "<a href=\"otherlocations.html\"><p class=\"ind\">Other Locations</p></a>\n");
    fprintf(f, "\t</body>\n</html>\n");
}

int yywrap()
{ return(1); }

int main()
{
    yylex();

    FILE *i, *p, *c, *o, *ac, *ol;

    i = fopen("web/index.html", "w");
    createIndex(i);
    fclose(i);

    p = fopen("web/persons.html", "w");
    printHeader(p);
    fprintf(p, "<h1>Persons</h1>");
    treeToHtml(persons,p);
    fprintf(p, "<a href=\"index.html\"><p>Voltar</p></a>\n\t</body>\n</html>\n");
    fclose(p);

    c = fopen("web/countrys.html", "w");
    printHeader(c);
    fprintf(c, "<h1>Countrys</h1>");
    treeToHtml(countrys,c);
    fprintf(c, "<a href=\"index.html\"><p>Voltar</p></a>\n\t</body>\n</html>\n");
    fclose(c);

    o = fopen("web/organizations.html", "w");
    printHeader(o);
    fprintf(o, "<h1>Organizations</h1>");
    treeToHtml(organizations,o);
    fprintf(o, "<a href=\"index.html\"><p>Voltar</p></a>\n\t</body>\n</html>\n");
    fclose(o);

    ac = fopen("web/citys.html", "w");
    printHeader(ac);
    fprintf(ac, "<h1>City's</h1>");
    treeToHtml(citys,ac);
    fprintf(ac, "<a href=\"index.html\"><p>Voltar</p></a>\n\t</body>\n</html>\n");
    fclose(ac);

    ol = fopen("web/otherlocations.html", "w");
    printHeader(ol);
```

```
    fprintf(ol, "<h1>Other Locations</h1>");
    treeToHtml(otherlocations,ol);
    fprintf(ol, "<a href=\"index.html\"><p>Voltar</p></a>\n\t</body>\n</html>\n");
    fclose(ol);

    return 0;
}
```

6.2.2 Estrutura de dados

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

#include "tree.h"




typedef struct tree
{
    char desig[MAX_SIZE];
    int num;
    tree_ptr left, right, subtree;
} Tree;


void insert(char *novo_desig, tree_ptr *p)
{
    if ((*p) == NULL)
    {
        (*p)=(tree_ptr)malloc(sizeof(Tree));
        strcpy((*p)->desig,novo_desig);
        (*p)->num=1;
        (*p)->left=NULL;
        (*p)->right=NULL;
        (*p)->subtree=NULL;
    }
    else{
        int comp=strcmp(novo_desig,(*p)->desig);
        if(comp < 0)
        {
            insert(novo_desig, &(*p)->left);
        }
        else if (comp > 0)
            {
                insert(novo_desig, &(*p)->right);
            }
```

```
            else
                {
                (*p)->num++;
                }
        }
}

void insert_subtree(char *novo_desig, tree_ptr *p){
    if ((*p) != NULL)
    {
        insert(novo_desig, &(*p)->subtree);
    }
}


void subtree_to_html(tree_ptr sub_arv, FILE *f){
    if(sub_arv != NULL){
        subtree_to_html(sub_arv->left, f);
        fprintf(f, "<ul>");
        fprintf(f, "\t<li><p class=\"node\">%s</p></li>\n",sub_arv->desig);
        fprintf(f, "</ul>");
        subtree_to_html(sub_arv->right, f);
    }
}

void treeToHtml(tree_ptr btree, FILE *f){
    if (btree != NULL)
    {
        treeToHtml(btree->left, f);
        fprintf(f, "<p class=\"root\">%s</p>\n",btree->desig);

        if(btree->subtree != NULL)
            subtree_to_html(btree->subtree, f);

        treeToHtml(btree->right, f);
    }
}

void imprime_inorder_tree(tree_ptr p)
{
    if (p!=NULL)
    {
        imprime_inorder_tree(p->left);
        printf("%s -> %d\n",p->desig, p->num);
        if(p->subtree != NULL) imprime_inorder_tree(p->subtree);
        imprime_inorder_tree(p->right);
    }
}
```

```
tree_ptr copyTree (tree_ptr p)
{
    tree_ptr aux=NULL;
    if (p!=NULL) {
        insert(p->desig,&aux);
        aux->left=copyTree(p->left);
        aux->right=copyTree(p->right);
    }
    return aux;
}


void imprime_mais_freq(tree_ptr p, int n)
{
    if (p!=NULL)
    {
        imprime_mais_freq(p->left,n);
        if ((p->num)>n) printf("%s\n",p->desig);
        imprime_mais_freq(p->right,n);
    }
}


int total_Pubs(tree_ptr p)
{
    if (!p) return 0;
    else return (p->num)+total_Pubs(p->left)+total_Pubs(p->right);
}


int total_elems(tree_ptr p) {
    if (!p) return 0;
    else return 1 + total_elems(p->left)+total_elems(p->right);
}


tree_ptr search_tree(char *desig, tree_ptr p)
{
    if (!p) return NULL;
    else {
        int comp=strcmp(p->desig,desig);
        if (comp==0) return (p);
        else {
            if (comp>0)  return search_tree(desig, p->left);
            else return search_tree(desig, p->right);
        }
    }
}
```

```
void imprime_pos_order_by_N(tree_ptr p, int *N) {
    if (p){
        imprime_pos_order_by_N(p->right,N);
        if (*N){
            printf("Elemento: %s\n", p->desig);
            printf("Nmr: %d\n\n", p->num);
            (*N)--;
        }
        imprime_pos_order_by_N(p->left,N);
    }
}


void makeempty(tree_ptr p)
{
    if (p != NULL)
    {
        makeempty(p->left);
        makeempty(p->right);
        free(p);
    }
}
```

6.2.3 Cabeçalho ficheiro C

```
#ifndef ___TREE_H___
#define ___TREE_H___

#define MAX_SIZE 200

typedef struct tree *tree_ptr;

/*Insere um novo elemento na BS-tree*/
void insert(char *novo_desig, tree_ptr *p);

/*Imprime todos os nodos(elementos) da BS-tree por ordem alfabetica*/
void imprime_inorder_tree(tree_ptr p);

/*Imprime os elementos que aparecem no minimo n vezes*/
void imprime_mais_freq(tree_ptr p, int n);

/*Devolve numero de ocorrencias de todos os elementos da BS-tree*/
int total_Pubs(tree_ptr p);

/*Devolve numero de nodos(elementos) da BS-tree*/
int total_elems(tree_ptr p);
```

```
/*Copia uma BS-tree*/
tree_ptr copyTree (tree_ptr p);

/*Procura um elemento numa BS-tree e devolve um apontador ele caso se encontrar.*/
tree_ptr search_tree(char *desig, tree_ptr p);

/*Imprime por ordem decrescente de numero de ocorrencias.*/
void imprime_pos_order_by_N(tree_ptr p, int *N);

/*Liberta memoria de BS-tree*/
void makeempty(tree_ptr p);

/*Insere elementos em subarvore*/
void insert_subtree(char *novo_desig, tree_ptr *p);

/*Imprime para ficheiro uma dada subarvore*/
void treeToHtml(tree_ptr btree, FILE *f);

void subtree_to_html(tree_ptr sub_arv, FILE *f);

#endif
```

6.3 Processamento de ficheiros com Canções

6.3.1 Filtro de Texto

```
%{
#include "musica.h"

char* takeOffAnotations(char* c){
    int i=0;
    while(c[i] != '\0' && !(c[i] == ' ' && c[i + 1] == ' ')) {
        i++;
    }
    c[i] = '\0';
    return c;
}

char* takeOffUnderSccore(char* c){
    int i=0, w=0;
    while(c[i] != '\0') {
        //ignore if '_'
        if(c[i] != '_') {
            c[w] = c[i];
            w++;
        }
        i++;
    }
```

```
    c[w] = '\0';
    return c;
}

%}


abc      \<[ ]*(?i:abc)[ ]*\>
abcclose      \<\/[ ]*(?i:abc)[ ]*\>

%%
^title:.+       appendTitle(yytext + 6);
^from:.+        appendFrom(yytext + 5);
^author:.+      appendAuthor(yytext + 7);
^lyrics:.+      appendLyrics(yytext + 7);
^music:.+       appendMusic(yytext + 6);
^singer:.+      appendSinger(yytext + 7);

^[a-zA-Z]+:.+   ;// Ignore other headers:

{abc}(.|\n)*{abcclose}  ;


##-+    {commitCheckNext(); start();}// Music End

^\n     appendWhiteLine(); // white music line.
[ ].*   ; //ignore lines that start with space (they are music anotations);
.*      {appendLine(takeOffAnotations(takeOffUnderSccore(yytext)));
        //poem line, taking of underscore and music annotations}
.|\n    ;
%%
```

6.3.2 Estrutura de dados

```
#include <stdio.h>
#include <unistd.h>
#include "musica.h"
#include <stdlib.h>
#include <string.h>

// ##############################################################################
//                              private header area
// ##############################################################################

extern int yylex();

typedef struct sMusicLine {
    char* line;
```

```
    struct sMusicLine* next;
} MusicLine;

typedef struct sMusic {
    char* _Title;
    char* _From;
    char* _Author;
    char* _Lyrics;
    char* _Music;
    char* _Singer;

    MusicLine* poem;
    MusicLine* poemEnd;
    int error;
} Music;

void writeLatex();

// ##############################################################################
//                          Implementation Area
// ##############################################################################
static Music music;
static char** argv;
static int argc;
static int argIndex;


#define append(name) \
    void append##name(char* t) { \
        if (music._##name != NULL) { \
            music.error = 1; \
        } else { \
            music._##name = strdup(t); \
        } \
    } \


append(Title)
append(From)
append(Author)
append(Lyrics)
append(Music)
append(Singer)

void appendLine(char* line) {
    if (music.poemEnd == NULL){
        music.poem = (MusicLine*) malloc(sizeof(MusicLine));
        music.poemEnd = music.poem;
    } else {
```

```
        music.poemEnd->next = (MusicLine*) malloc(sizeof(MusicLine));
        music.poemEnd = music.poemEnd->next;
    }
    music.poemEnd->next = NULL;
    music.poemEnd->line = strdup(line);
}

void appendWhiteLine() {
    // don't append white line in the init of the file

    if (music.poemEnd != NULL){
        music.poemEnd->next = (MusicLine*) malloc(sizeof(MusicLine));
        music.poemEnd = music.poemEnd->next;
        music.poemEnd->next = NULL;
        music.poemEnd->line = strdup("");
    }
}

void start() {
    music._Title = NULL;
    music._From = NULL;
    music._Author = NULL;
    music._Lyrics = NULL;
    music._Music = NULL;
    music._Singer = NULL;

    music.poem = NULL;
    music.poemEnd = NULL;
    music.error = 0;
}

void writeLatex(Music m, FILE* f) {
    fprintf(f, "\\title{%s}\n",m._Title);
    if (m._Author != NULL) {
        fprintf(f, "\\author{%s}\n",m._Author);
    } else {
        fprintf(f, "\\author{%s,%s}\n",m._Lyrics,m._Music);
    }

    fprintf(f, "\\documentclass[12pt]{article}\n");
    fprintf(f, "\\begin{document}\n");
    fprintf(f, "\\maketitle\n");
    fprintf(f, "\\section*{Letra}\n");
    fprintf(f, "\\begin{center}\n");
    MusicLine* it;
    fprintf(f, "\\begin{verbatim}\n");
    for (it = m.poem; it ; it = it->next){
        // insert line
        //if (it->line[0] != '\0'){
```

```
            fprintf(f, "%s\n",it->line);
            //} else {
                // insert white space

                //fprintf(f, "\\vspace{5mm}\n");

            //}
        }
        fprintf(f, "\\end{verbatim}\n");
        if (m._Singer){
            fprintf(f, "\\vspace{5mm}\n");
            fprintf(f, "\\hfill %s\n",m._Singer);
        }
        fprintf(f, "\\end{center}\n");
        fprintf(f,"\\end{document}\n");
    }

    int commitCheckNext() {
        if (music.error != 0) {
            // need something here?
        } else {
            FILE* f;
            if (argIndex < argc) {
                f= fopen(argv[argIndex],"w");
                writeLatex(music,f);
                fflush(f);
                fclose(f);
    //              execl("/usr/bin/pdflatex","pdflatex",argv[argIndex], NULL);
            } else {
                char buffer[20];
                sprintf(buffer,"%d.tex",argIndex);
                f = fopen(buffer,"w");
                writeLatex(music,f);
                fflush(f);
                fclose(f);
    //              execl("/usr/bin/pdflatex","pdflatex",buffer, NULL);
            }
            argIndex++;
        }
        return music.error;
    }

    int main(int _argc, char* _argv[]){
        argc = _argc - 1 ; // first argument is the program name.
        argv = (char**) malloc(sizeof(char) * _argc );

        for(_argc--;_argc>0;_argc--){
            argv[_argc - 1] = strdup(_argv[_argc]);
        }
```

```
    start();
    yylex();
    commitCheckNext();
}

int yywrap() {

    return -1;
}
```

6.3.3 Cabeçalho ficheiro C

```
#ifndef __musica_h__
#define __musica_h__


void appendTitle(char* title);
void appendFrom(char* from);
void appendAuthor(char* author);
void appendLine(char* line);
void appendWhiteLine();
void start();
int commitCheckNext();


#endif
```

6.3.4 Testes

Input teste 1

```
title: Amêndoa Amarga
lyrics: José Carlos Ary dos Santos
music: Alain Oulman
singer: Amália Rodrigues

Port ti falo
e ninguém pensa
mas eu digo
minha amêndoa, meu amigo
meu irmão
meu tropel de ternura
minha casa
meu jardim de carência
minha asa.

Por ti vivo
e ninguém pensa
mas eu sigo
```

```
um caminho de silvas
e de nardos
uma intensa ternura
que persigo
rodeada de cardos
por tantos lados.

Por ti morro
e ninguém sabe
mas eu espero
o teu corpo que sabe
a madrugada
o teu corpo que sabe
a desespero

ó minha amarga amêndoa
desejada.

ó minha amarga amêndoa
desejada.
```

Output teste 1

```
\title{ Amêndoa Amarga}
\author{ José Carlos Ary dos Santos, Alain Oulman }
\documentclass[12pt]{article}
\begin{document}
\maketitle
\section*{Letra}
\begin{center}
\begin{verbatim}
Port ti falo
e ninguém pensa
mas eu digo
minha amêndoa, meu amigo
meu irmão
meu tropel de ternura
minha casa
meu jardim de carência
minha asa.

Por ti vivo
e ninguém pensa
mas eu sigo
um caminho de silvas
e de nardos
uma intensa ternura
que persigo
```

```
rodeada de cardos
por tantos lados.

Por ti morro
e ninguém sabe
mas eu espero
o teu corpo que sabe
a madrugada
o teu corpo que sabe
a desespero

ó minha amarga amêndoa
desejada.

ó minha amarga amêndoa
desejada.
\end{verbatim}
\vspace{5mm}
\hfill  Amália Rodrigues
\end{center}
\end{document}
```

Input teste 2

```
title: Amêndoa Amarga
lyrics: José Carlos Ary dos Santos
music: Alain Oulman
singer: Amália Rodrigues

Port ti falo
e ninguém pensa
mas eu digo
minha amêndoa, meu amigo
meu irmão
meu tropel de ternura
minha casa
meu jardim de carência
minha asa.

Por ti vivo
e ninguém pensa
mas eu sigo
um caminho de silvas
e de nardos
uma intensa ternura
que persigo
rodeada de cardos
por tantos lados.
```

```
Por ti morro
e ninguém sabe
mas eu espero
o teu corpo que sabe
a madrugada
o teu corpo que sabe
a desespero

ó minha amarga amêndoa
desejada.

ó minha amarga amêndoa
desejada.
```

Output teste 2

```
\title{ * Tejo que levas as águas}
\author{ Manuel da Fonseca, Adriano Correia de Oliveira}
\documentclass[12pt]{article}
\begin{document}
\maketitle
\section*{Letra}
\begin{center}
\begin{verbatim}
Tejo que levas as águas
correndo de par em par
lava a cidade de mágoas
leva as mágoas para o mar

Lava-a de crimes espantos
de roubos, fomes, terrores,
lava a cidade de quantos
do ódio fingem amores

Leva nas águas as grades
de aço e silêncio forjadas
deixa soltar-se a verdade
das bocas amordaçadas

Lava bancos e empresas
dos comedores de dinheiro
que dos salários de tristeza
arrecadam lucro inteiro

Lava palácios vivendas
casebres bairros da lata
leva negócios e rendas
```

```
que a uns farta e a outros mata

Tejo que levas as águas
correndo de par em par
lava a cidade de mágoas
leva as mágoas para o mar

Lava avenidas de vícios
vielas de amores venais
lava albergues e hospícios
cadeias e hospitais

Afoga empenhos favores
vãs glórias, ocas palmas
leva o poder dos senhores
que compram corpos e almas

Leva nas águas as grades
...

Das camas de amor comprado
desata abraços de lodo
rostos corpos destroçados
lava-os com sal e iodo

Tejo que levas nas águas
...
\end{verbatim}
\vspace{5mm}
\hfill  Adriano Correia de Oliveira
\end{center}
\end{document}
```

Input teste 3

```
title: = Raúl tinha um Ioio
singer: Bando dos Gambozinos
lyrics: Manuel António Pina
music: Suzana Ralha
in: "o beco dos gambozinos"
from: jj

<abc>
X: 1
M: 2/4
K: C
Q: 1/4=60
L: 1/8
```

```
dc Ad | dc Ad | dc Ad | GG AA |
w:Ra-ul ti-nhaum i-oi-o que io-io-ia-va to-do o di-a
z/2 D/2E/2F/2 G>G | AG F2 |1 z/2 D/2E/2F/2 G>G | AG F2 :|2 z F/2A/2 GG | FE DD |]
w:quan-doo Ra-úl fa-zia ó-ó o i-o-io a-dor-me-cia
</abc>

Raúl tinha um ioio
que ioioiava todo o dia
quando o Raúl fazia ó-ó
o ioio adormecia

E quando o Raúl chorava
porque o ó-ó não vinha
o ioio embalava
para baixo e para cima

Raúl dormia e sonhava
e quando sonhava sorria
porque o io-io ioioiava
nos sonhos que Raúl via
```

Output teste 3

```
\title{ = Raúl tinha um Ioio}
\author{ Manuel António Pina, Suzana Ralha}
\documentclass[12pt]{article}
\begin{document}
\maketitle
\section*{Letra}
\begin{center}
\begin{verbatim}
Raúl tinha um ioio
que ioioiava todo o dia
quando o Raúl fazia ó-ó
o ioio adormecia

E quando o Raúl chorava
porque o ó-ó não vinha
o ioio embalava
para baixo e para cima

Raúl dormia e sonhava
e quando sonhava sorria
porque o io-io ioioiava
nos sonhos que Raúl via
\end{verbatim}
\vspace{5mm}
\hfill  Bando dos Gambozinos
```

```
\end{center}
\end{document}
```

Amndoa Amarga

Jos Carlos Ary dos Santos, Alain Oulman

April 1, 2015

Letra

```
Port ti falo
e ningum pensa
mas eu digo
minha amndoa, meu amigo
meu irmo
meu tropel de ternura
minha casa
meu jardim de carncia
minha asa.

Por ti vivo
e ningum pensa
mas eu sigo
um caminho de silvas
e de nardos
uma intensa ternura
que persigo
rodeada de cardos
por tantos lados.

Por ti morro
e ningum sabe
mas eu espero
o teu corpo que sabe
a madrugada
```


Figura 6.1: PDF gerado por o ficheiro latex (teste 1). Pagina 1 de 2

Por ti morro
e ningum sabe
mas eu espero
o teu corpo que sabe
a madrugada

o teu corpo que sabe
a desespero

minha amarga amndoa
desejada.

minha amarga amndoa
desejada.

Amlia Rodrigues

Figura 6.2: PDF gerado por o ficheiro latex (teste 1). Pagina 2 de 2

* Tejo que levas as guas

Manuel da Fonseca, Adriano Correia de Oliveira

April 1, 2015

Letra

Tejo que levas as guas
correndo de par em par
lava a cidade de mgoas
leva as mgoas para o mar

Lava-a de crimes espantos
de roubos, fomes, terrores,
lava a cidade de quantos
do dio fingem amores

Leva nas guas as grades
de ao e silncio forjadas
deixa soltar-se a verdade
das bocas amordaadas

Lava bancos e empresas
dos comedores de dinheiro
que dos salrios de tristeza
arrecadam lucro inteiro

Lava palcios vivendas
casebres bairros da lata
leva negcios e rendas
que a uns farta e a outros mata

Figura 6.3: PDF gerado por o ficheiro latex (teste 2). Pagina 1 de 2

Tejo que levas as guas
correndo de par em par
lava a cidade de mgoas
leva as mgoas para o mar

Lava avenidas de vcios
vielas de amores venais
lava albergues e hospcios
cadeias e hospitais

Afoga empenhos favores
vs glrias, ocas palmas
leva o poder dos senhores
que compram corpos e almas

Leva nas guas as grades
...

Das camas de amor comprado
desata abraos de lodo
rostos corpos destroados
lava-os com sal e iodo

Tejo que levas nas guas
...

Adriano Correia de Oliveira

Figura 6.4: PDF gerado por o ficheiro latex (teste 2). Pagina 2 de 2

= Ral tinha um Ioio

Manuel Antnio Pina, Suzana Ralha

April 1, 2015

Letra

```
Ral tinha um ioio
que ioioiava todo o dia
quando o Ral fazia -
o ioio adormecia

E quando o Ral chorava
porque o - no vinha
o ioio embalava
para baixo e para cima

Ral dormia e sonhava
e quando sonhava sorria
porque o io-io ioioiava
nos sonhos que Ral via
```

Bando dos Gambozinos

Figura 6.5: PDF gerado por o ficheiro latex (teste 3)