



Universidade do Minho

## Projecto Java de Laboratório de Informática Relatório

Discentes:  
Axel Ferreira - a53064  
João Rua - a41841

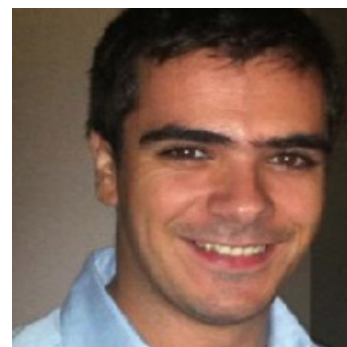
Docentes:  
F. Mário Martins  
João Miguel Fernandes  
João Luís Sobral

June 14, 2013

### Grupo



(a) nome : Axel Ferreira  
número : 53064  
mail : axelferreira@me.com



(b) nome : João Rua  
número : 41841  
mail : joaorua@gmail.com



## Contents

<b>1</b>	<b>Introdução</b>	<b>2</b>
1.1	Estrutura do Relatório . . . . .	3
<b>2</b>	<b>Tipos de Dados e Estruturas de Dados</b>	<b>3</b>
2.1	Tipos de Dados . . . . .	3
2.2	Macros . . . . .	3
2.3	Estruturas de Dados . . . . .	4
<b>3</b>	<b>Fase 1</b>	<b>5</b>
<b>4</b>	<b>Fase 2A</b>	<b>5</b>
<b>5</b>	<b>Fase 2B</b>	<b>5</b>
<b>6</b>	<b>Modularidade</b>	<b>6</b>
6.1	parser . . . . .	8
6.2	fread . . . . .	8
6.3	fwrite . . . . .	9
6.4	F . . . . .	9
6.5	main . . . . .	10
<b>7</b>	<b>Conclusão</b>	<b>10</b>
<b>8</b>	<b>Interface</b>	<b>10</b>

### Abstract

Foi desenvolvido um programa no âmbito da U.C. de Laboratório de Informática III , capaz de utilizar o conteúdo processado pelo anterior programa desenvolvido em C, também nesta UC, capaz de ler um ficheiro contendo um conjunto de autores e respetivos co-autores, e responder a alguns queries sobre estes dados. Estes dados de autorias, e co-autorias são retirados do website [DBLP](#).

## 1 Introdução

No âmbito da unidade curricular Laboratórios de Informática III foi proposta a realização de um projeto com 3 etapas. A primeira incidiu no parsing e filtragem de um conjunto de entradas de ficheiros de texto com uma sintaxe definida, em que cada linha contém uma entrada correspondente a uma publicação em revista ou conferência. Procurou-se, então, criar um programa que abra um ficheiro "lista.txt" que contém todos os ficheiros a serem processados, em cada linha deste ficheiro corresponde ao nome de um ficheiro a ser processado. Abra cada um desses ficheiros, lendo linha a linha cada entrada, processe essa entrada e consoante seja ou não válida, incremente os contadores correspondentes e finalmente imprima dois ficheiros em que o primeiro contém a lista de cada ficheiro processado seguido do número de rejeitados e o segundo os contadores do número de total de processados, rejeitados, artigos, conferencias e revistas. A segunda



etapa consiste na impressão de um ficheiro de estatísticas mais detalhado, que detalha o número de artigos existentes, por ano e por número de autores, o número de artigos existentes por número de autores, para cada intervalo de anos dado, o número de artigos existente, e para um dado ano, a percentagem de artigos para cada número de autores. A última fase incidiu no teste de performance e otimização da estrutura do programa e na implementação de uma estrutura de co-autores que permite responder a algumas queries sobre co-autoria.

### 1.1 Estrutura do Relatório

Este relatório inicia-se com uma capa, incluindo o título do projeto, a data, a identificação do autor e da equipa docente que acompanhou o projeto. Segue-se o Índice, o Abstract que resume o projeto, a Introdução ao mesmo (onde se explicita o objetivo a atingir) e a Estrutura do Relatório. O Desenvolvimento tem como secções os tipos e estruturas de dados, cada uma das tarefas Fase 1, Fase 2A e Fase 2B. Por fim apresentam-se a Conclusão e os anexos.

## 2 Tipos de Dados e Estruturas de Dados

### 2.1 Tipos de Dados

Foram utilizados os seguintes tipos de dados para transportar informação do fluxo do programa através de retornos de funções.

- struct sStats - é utilizada pelo parser para após validar uma linha devolver ao main o conteúdo útil dessa entrada.

```
typedef struct sStats
{   char * nomes;           // buffer com os autores
    int nAutores;           // número de autores
    int ano;                // Ano
} Stats;
```

- struct sList - Utilizado como nodo da lista na estrutura de dados.

```
struct sList
{   int nAut;               // número de Autores.
    int nArt;               // número de Artigos.
    struct sList * seg;     // Nodo seguinte
} List;
```

### 2.2 Macros

Estas macros foram utilizadas para facilitar a criação e manutenção do programa, bem como para facilitar a compreensão do mesmo. Desta forma, os seguintes tipos foram definidos.

- TRUE 1 - Representa o valor booleano verdadeiro e corresponde ao valor 1 utilizado pelo compilador.



- FALSE 0 - Análogo ao anterior para o valor falso (0).
- MAX\_XXXX - Valores para inicialização de valores como dimensão máxima ou dimensão inicial de arrays ou buffers.
- DEBUG\_MODE X - X toma um valor inteiro de forma a activar vários níveis do modo de debugging, que permite mostrar informação respeitante ao debugging ( como pr. ex. mostrar printf() ), em que cada nível diz respeito a debugging de partes diferentes de cada módulo do programa.
- MAX\_FILE X - X representa o número máximo de ficheiros com que o módulo fread pode trabalhar em simultâneo. Este valor é alterado na interface.

### 2.3 Estruturas de Dados

Estrutura.jpg

A estrutura de dados escolhida para armazenar os dados processados foi um Array de Listas Ligadas. Inicialmente este array tinha 100 posições possíveis para anos,



isto porque dado o contexto do problema seria suficiente(dado que a primeira publicação datava de 1936). Contudo foi redimensionado para 3000 posições, ocupando cerca de 24KB de memória, uma dimensão ainda assim razoável dado o contexto do problema, isto devido á existência, nos ficheiros de teste, de datas de publicações fictícias do ano 1300. O ano corresponde ao índice do array, que é calculado retirando o ano inicial ao ano em questão e somando 1. O índice 0 foi reservado para manter contadores totais para facilitar a pesquisa eficiente por número de artigos. Cada nodo do array tem um apontador para uma lista ligada. Estas listas ligadas têm 2 inteiros um que guarda o #Autores que publicaram, o outro o número de artigos. A razão pela qual esta estrutura foi escolhida deve-se á simplicidade de implementação bem como á vantagem em performance relativamente á utilização de apenas uma lista ligada. Sabendo que o acesso ao array é constante, e que os artigos se encontram relativamente bem distribuídos por ano. O acesso ás sub-listas ligadas de um determinado ano é constante. Sabendo também que a distribuição de artigos por número de autor se encontra entre as co-autorias de 1 a 20 elementos, conclui-se que esta lista nunca será muito grande, garantindo assim pouca penalização nos acessos á memória (relativamente a uma alternativa implementada apenas em listas ligadas).

### 3 Fase 1

A primeira fase incidiu na criação de um programa capaz de ler um ficheiro lista.txt, e para cada linha deste ficheiro, invocar um módulo separado "Parser" que abria e processava o ficheiro cujo nome correspondia á linha lida, armazenando os resultados estatísticos no próprio módulo. Finalmente, no modulo main, era invocada uma função da interface do módulo parser que imprimia os ficheiros D e E com o respetivo conteúdo.

### 4 Fase 2A

Nesta fase foi efetuado o refactoring e implementado o buffer dinâmico. Garantindo-se desta forma que o fluxo do programa é centrado no main e que o buffer tem sempre tamanho suficiente para ler uma linha.

### 5 Fase 2B

Para esta fase foi reservado o índice 0 do array de anos. Este índice contém um apontador para uma lista ligada de número de autores e número de artigos correspondentes, em tudo semelhante ás outras, com a particularidade desta lista conter o total de #artigos para cada #Autores sendo que a entrada com o #Autores 0 tem o #Artigos total de toda a estrutura. Foi também introduzido um primeiro nodo em cada lista com #Autores 0, de forma a garantir um contador do total de artigos para um determinado ano, com o intuito de facilitar e tornar mais eficiente o o cálculo das percentagens bem como o intervalo de anos por #Artigos para criação do Ficheiro G. O refactoring alterou o programa,



garantindo que o fluxo do mesmo passava pelo main. Como a partida parece evidente, devia ter havido uma perda na performance uma vez que todo o conteúdo é transportado entre módulos através do main, em vez de ser acedido diretamente. Contudo e após medições de tempos conclui-se que houve um ganho ligeiro na performance. Dado que a estrutura não cresce muito em tamanho à medida que o input aumenta uma vez que são apenas incrementadas as variáveis já contidas na estrutura, o tempo de procura ou impressão do ficheiro G na estrutura para valores sucessivamente maiores do input é constante.

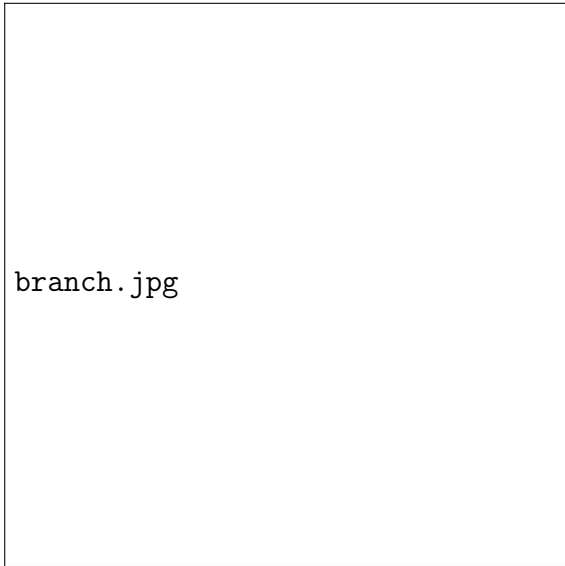
## 6 Modularidade

Após a primeira entrega, de forma a garantir a ocultação de dados e da implementação, a abstração de dados, o encapsulamento e alguma independência contextual, a arquitetura do programa sofreu um processo de refactoring tendo sido reformulados e criados novos módulos. Em cada módulo, os dados foram declarados como "static" tendo sido criados interfaces para mediar o manuseamento dos mesmos. Isto garante independência contextual, ocultação da implementação e dos dados. Estas interfaces foram posteriormente incluídas no programa main de forma a garantir que todo o fluxo do programa é controlado pelo módulo main. é possível ver a nova arquitetura dos módulos no diagrama abaixo.



Arquitetura.jpg

Toda a criação do programa, alteração e implementações experimentais foram feitas com recurso ao git com repositório no Github. O que permitiu a criação de branches para cada parte do projeto, bem como para experiências.



`branch.jpg`

Para além disto também foi com recurso ao Git que foi possível a compilação de uma versão pré-refactoring para comparação da performance.

### 6.1 parser

O módulo Parser é responsável pelo parsing e validação de cada linha que lhe é passada, devolvendo uma estrutura Stats contendo a informação relevante da linha (nomes de autores, número de autores e ano da publicação) caso seja válida.

### 6.2 fread

O módulo fread (file read) é responsável pela abertura de ficheiros para leitura. Está preparado para trabalhar com um número de ficheiros máximo a definir na interface. Este módulo está preparado para trabalhar com múltiplos ficheiros simultaneamente devido a duas razões. A primeira é para permitir que este módulo possa trabalhar com o ficheiro lista.txt e com um ficheiro de revista ou conferência em simultâneo. A segunda razão deve-se a inicialmente ter sido pensada a possibilidade de otimizar o programa através da criação de várias threads de forma a que cada ficheiro de revista ou conferência pudesse ser tratado num processo separado, implementando assim paralelismo. Contudo no decorrer do projeto, e após medições de tempos de execução tanto no teste 5 (stress) como posteriormente no TESTE1\_FASE2, foi concluído que esta otimização era desnecessária tendo em conta o tempo que a implementação gastaria, bem como o pouco benefício (em tempo de execução) que traria. Esta conclusão é suportada pelo facto dos testes utilizados terem dimensões razoavelmente grandes e o tempo de execução nestes testes anda na ordem das décimas de segundo tal como nos testes mais simples. Inicialmente foi implementado um buffer de 1KB, tendo sido alterado posteriormente para um buffer dinâmico em que o tamanho inicial deste buffer é ligeiramente superior à dimensão da maioria do tamanho das entradas.





grafo.jpg

### 6.3 fwrite

Este módulo é semelhante ao anterior, difere apenas no propósito de escrever ficheiros.

### 6.4 F

O módulo F consiste na implementação da fase 2A, possui uma interface que é utilizada pelo módulo main de forma a inserir os dados relevantes que o main recebe através do módulo parser. O módulo F é responsável pela criação e gestão da estrutura de dados que permite posteriormente imprimir o ficheiro G.



## 6.5 main

Este módulo controla o fluxo do programa, fazendo a ponte entre os módulos anteriores. Nomeadamente faz um pedido de linha do ficheiro lista.txt ao modulo fread, para cada linha devolvida faz um novo pedido a esse módulo com o nome de cada ficheiro, sendo devolvida cada linha (entrada) deste último ficheiro.

Posteriormente o main passa a linha obtida ao parser, de onde é devolvido um resultado que pode ser NULL sendo incrementado o contador de artigos rejeitados (static no main) ou uma estrutura com informação que é passada ao módulo F para inserção. Finalmente, quando todos os ficheiros contidos no ficheiro "lista.txt" forem processados, o módulo main dá ordem de impressão dos ficheiros D.txt, E.txt e G.csv aos respetivos módulos.

## 7 Conclusão

A principal dificuldade que foi ultrapassada, nomeadamente na 2ª Fase, foi a implementação das estruturas de dados uma vez que a função de inserção não funcionava corretamente tendo sido re implementada numa versão recursiva correta. A dificuldade não ultrapassada devido, ao tempo despendido no processo de refactoring, resume-se á falta de tempo para implementação da estrutura de co-autoria.

## 8 Interface

- Parser

```
/**
 * Parses a Line
 * @param buffer the buffer containing the line to be parsed
 * @param ty+he type of the line conference or journal)
 * @return Struct with all counters.
 */
Stats parseLine(char * buffer, char t);
```

- fread

```
/**
 * Initializes the control for the files
 */
void init_file_control();

/**
 * Opens a file.
 * @return file index in control, or -1 if failed.
 */
int openFile(char * file_name, char * mode);
```



```
/**
 * Reads a line from a file growing the buffer as needed.
 * @param a reference to the buffer reference.
 * @param a reference for the size of the buffer.
 * @param file index in control.
 * @return 0 if success -1 otherwise.
 */
```

```
int dynamic_read_line(char** buf, int* size, int file);
```

```
/**
 * Closes a file.
 * @param index of the file to be closed.
 */
```

```
void closeFile(int index);
```

- fwrite

```
/**
 * This function outputs the number of rejected entries in each file to the "E
 * @param int counter number of invalid entries in the file.
 * @param char * fileN the name of the courent file.
 */
```

```
void imprimeE(int bool, int counter, char * file, char * path);
```

```
/**
 * This function creates the file "D.txt" and it's content. If the file already
 * @param nRej the number of rejected entries
 * @param nJou the number of Journals
 * @param nCou the number of Counferences
 * @param path the path of the file to be written to.
 */
```

```
void imprimeD(int nRej, int nJou, int nCou, char * path);
```

- F

```
/**
 * Adds data to the structure.
 * @return returns a pointer to a structure with the linked list
 */
```

```
struct sList * addList();
```

```
/**
 * This function creates the file "G.csv" and it's content. If the file already
 * @param bool if it's first time opens in "w" mode else in "a".
 */
```

```
int imprimeG();
```