



**Universidade do Minho**

# Relatório - Laboratórios de Informática III

MIEI - 2º ANO - 2º SEMESTRE

UNIVERSIDADE DO MINHO

WIKIPEDIA

GRUPO 63

Ricardo Certo  
A75315

José Bastos  
A74696

Guilherme Guerreiro  
A73860

1 de Maio de 2017

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>2</b>
<b>2</b>	<b>Descrição dos Módulos</b>	<b>3</b>
2.1	Ficheiros . . . . .	3
2.2	Estruturas . . . . .	3
2.2.1	TAD_istruct . . . . .	3
2.2.2	Array . . . . .	6
2.2.3	Heap . . . . .	7
<b>3</b>	<b>Programa Principal</b>	<b>9</b>
<b>4</b>	<b>Makefile</b>	<b>10</b>
4.1	Estrutura . . . . .	10
4.2	Comandos . . . . .	10
4.2.1	make program ou make . . . . .	10
4.2.2	make clean . . . . .	10
4.2.3	make doc . . . . .	10
4.3	Dependências . . . . .	11
<b>5</b>	<b>Interface com o utilizador</b>	<b>12</b>
<b>6</b>	<b>Documentação</b>	<b>15</b>
<b>7</b>	<b>Conclusão</b>	<b>17</b>

# 1 Introdução

Este projeto foi nos solicitado pelos docentes da unidade curricular de Laboratórios de Informática III e tem como principal objetivo a realização de um programa que permita analisar os artigos presentes em backups da Wikipedia, realizados em diferentes meses, e extrair informação útil para esse período de tempo como, por exemplo, o número de revisões, o número de novos artigos, etc.

Outros objetivos estão também associados a este, como a consolidação de conhecimentos adquiridos em unidades curriculares anteriores, dentro das quais se incluem Programação Imperativa, Algoritmos e Complexidade e Arquitetura de Computadores.

O principal desafio do projecto seria a programação em larga escala, uma vez que pelo nosso programa passam milhares de dados, aumento assim a complexidade do trabalho. Para que a realização deste projeto fosse possível, foram-nos introduzidos novos princípios de programação, com especial relevo para a Modularidade e encapsulamento de dados.

## 2 Descrição dos Módulos

### 2.1 Ficheiros

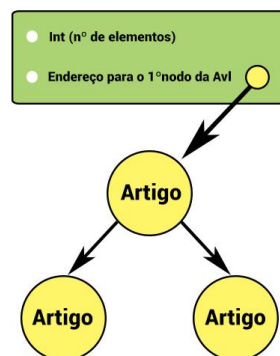
Os ficheiros que usamos para obter a informação e guardar o seu respetivo conteúdo foram três snapshots respetivos aos meses de Dezembro, Janeiro e Dezembro.

Os snapshots continham informação relativa aos artigos presentes nesses meses no Wipedia, desde criação de artigos até às revisões de artigos efetuadas por um utilizador numa data.

Esses snapshots estavam escritos em XML, o que nos obrigou a realizar um parser para podermos guardar a informação que criamos nas estruturas.

### 2.2 Estruturas

#### 2.2.1 TAD\_istruct



##### 2.2.1.1 Estrutura

```
struct avl {
    long idArticle;
    int height;
    void* info;
    struct avl *left, *right;
};

struct TCD_istruct {
    int total;
    Avl avl;
};
```

A estrutura TAD\_istruct é o módulo aonde vão ser armazenadas todas as informações retiradas dos snapshots que possam vir a ser úteis para responder às interrogações. Como se trata

de uma larga quantia de informação, necessitamos de os guardar numa estrutura adequada, estrutura essa que podemos verificar em cima.

A estrutura corresponde a uma Avl que vai conter toda a informação guardada e um inteiro para nos dizer o total de nodos dessa Avl. A Avl está ordenada em função do id do artigo e para além disso no nodo também contém o seu peso para garantir que ela está balanceada corretamente, tem também um apontador para uma estrutura que vai servir para guardar as várias informações sobre o artigo em questão e também tem dois apontadores um para a Avl da direita e outro para a Avl da esquerda.

#### 2.2.1.2 Typedefs

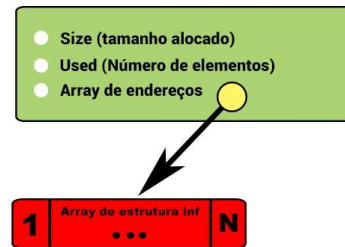
- `typedef void(*Func)(void*);`
- `typedef int BOOL;`
- `typedef char* STR;`
- `typedef struct myAvl *TAD_istruct;`
- `typedef struct avl *Avl;`

#### 2.2.1.3 Funções

- **Avl initAvl();**  
Inicia a Avl, colocando-a a NULL.
- **TAD\_istruct initMyAvl();**  
Inicia a estrutura TAD\_istruct, alocando espaço para a mesma.
- **TAD\_istruct insertMyAvl(TAD\_istruct a,char\* line,void\* info,int aux);**  
Insere uma string e uma info numa TAD\_istruct.
- **int heightAvl(Avl a);**  
Retorna a altura de uma dada Avl.
- **Avl insert(Avl estrutura,STR line,void\* info);**  
Insere uma string e uma info num dado nodo de uma Avl.
- **void removeAvl(Avl estrutura,Func freeInfo);**  
Remove uma dada Avl, libertando o espaço ocupado pela mesma.
- **void removeMyAvl(TAD\_istruct estrutura,Func f);**  
Remove uma estrutura TAD\_istruct, libertando o espaço ocupado pela mesma.
- **void freeNodo(Avl a);**  
Remove um nodo de uma Avl, libertando o espaço ocupado por este.
- **BOOL existAvl(Avl estrutura, STR line);**  
Verifica se uma dada String existe numa dada Avl.
- **int totalElements(TAD\_istruct estrutura);**  
Retorna o numero de elementos de uma dada estrutura TAD\_istruct.
- **void\* findInfo (Avl a,STR line,int\* x);**  
Procura uma String que exista num dado nodo de uma Avl e retorna a info que lá se encontra retorna NULL caso nao exista.

- **Avl cloneAvl (Avl estrutura);**  
Faz um clone de uma dada Avl.
- **TAD\_istruct cloneMyAvl (TAD\_istruct estrutura);**  
Faz um clone de uma dada TAD\_istruct.
- **void removeFromTAD\_istruct (TAD\_istruct\* estrutura,int x);**  
Remove x nodos de uma dada TAD\_istruct.
- **int infoNULL(Avl a);**  
Conta em quantos nodos de uma Avl nao existe Info.
- **Avl getAvl(TAD\_istruct estrutura);**  
Retorna a Avl que se encontra numa estrutura TAD\_istruct.
- **Avl getAvlLeft(Avl a);**  
Retorna a Sub-arvore do lado esquerdo de uma dada Avl.
- **Avl getAvlRight(Avl a);**  
Retorna a Sub-arvore do lado direito de uma dada Avl.
- **char\* getAvlCode(Avl a);**  
Retorna a String presente no nodo de uma dada Avl.
- **void\* getInfo(Avl a);**  
Retorna a Info presente no nodo de uma dada Avl.
- **int getSize(TAD\_istruct a);**  
Retorna o tamanho de uma dada estrutura TAD\_istruct.
- **void setInfo(Avl a,void\* i);**  
Altera a Info presente no nodo de uma dada Avl.
- **void setAvl(TAD\_istruct a,Avl b);**  
Altera Avl presente na estrutura TAD\_istruct

### 2.2.2 Array



#### 2.2.2.1 Estrutura

```
struct inf {  
    long info;  
    long total;  
};
```

```
struct array {  
    int size;  
    int used;  
    struct inf *values;  
};
```

A estrutura Array foi criada para conseguirmos responder às interrogações pedidas inicialmente. O array é composto por um tamanho, por o número de posições ocupadas e por um apontador para uma outra estrutura chamada inf. A estrutura inf tem dois longs um para guardar informação relativa ao id de utilizador ou de artigo e outro para guardar um contador que vai ser útil para a realização de algumas queries.

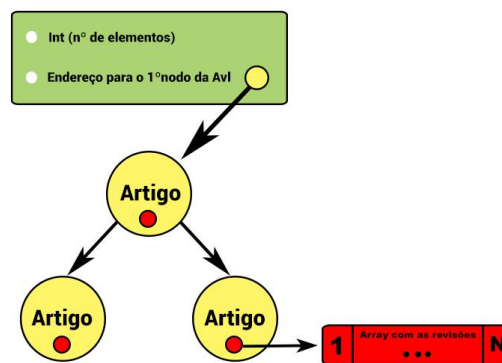
**2.2.2.2 Typedefs** A estrutura Array não possui nenhum typedef.

#### 2.2.2.3 Funções

- **Array initArray(int size);**  
Função responsável por inicializar um array.
- **int insertInf(Array a, long info, long total);**  
Função responsável por inserir informação num array.
- **int findIndex(long id, Array a);**  
Função de procura de um dado index.

- **int getArrayUsed(Array a);**  
Função que retorna o tamanho do array a ser usado.
- **long getInfTotal(Array a, int i);**  
Função que retorna a informação total.
- **long getInfInfo(Array a, int i);**  
Função que retorna a informação da estrutura apontada por uma dado posição do array.
- **void setInfTotal(Array a, int index, long i);**  
Função que altera a informação da estrutura apontada por uma dado posição do array.
- **static void swap(Array a, int x, int y);**  
Função responsável por fazer swap numa Array.
- **static void sortInf(Array a);**  
Função que aplica o algoritmo de ordenação Bubble-Up no Array.

### 2.2.3 Heap



#### 2.2.3.1 Estrutura

```
struct elem{
    long idCont;
    char* nameCont;
    int totalWords;
    int nbytes;
    char* timestamp;
    long revisionId;
    char* title;
};
```

```
struct heap{
    int size;
```



```

    int used;
    long total;
    struct elem *values;
};

```

A estrutura Heap é passada na Avl e foi criada com o intuito de guardar toda a informação necessária das revisões para responder às queries pretendidas. A Heap é constituída por um tamanho, que corresponde ao tamanho alocado para ela, o número de posições ocupadas e um array de apontadores para a estrutura elem chamado values. Finalmente, a estrutura elem é onde guardamos toda a informação de cada revisão: o id do Contribuidor, o nome do Contribuidor, o número total de palavras da revisão, o número de bytes da revisão, o timestamp, o id da revisão e o título do artigo.

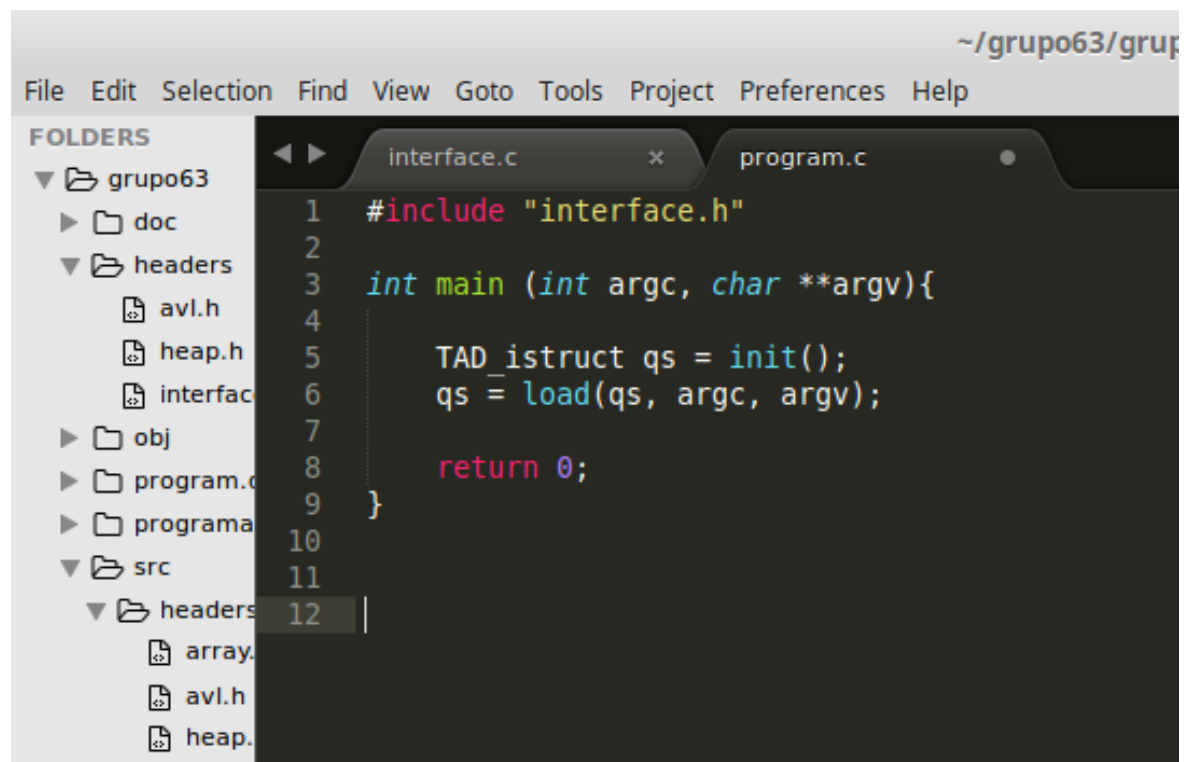
### 2.2.3.2 Typedefs

- typedef struct article \*Article;

### 2.2.3.3 Funções

- **Heap initHeap (int size);**  
Inicia a estrutura Heap com um dado tamanho, alocando espaço para a mesma.
- **int insertHeap (Heap h, double x,int y,char\* ct);**  
Insere numa Heap um determinado valor e uma string.
- **int getHeapUsed(Heap h);**  
Retorna o numero de elem a serem usados pela Heap
- **double getIdCont(Heap h, int i);**  
Retorna o idCont do elem na posição i do Heap
- **char\* getNameCont(Heap h, int i);**  
Retorna o nameCont do elem na posição i do Heap
- **int getTotalWords(Heap h, int i);**  
Retorna o totalWords do elem na posição i do Heap
- **char\* getTimestamp(Heap h, int i);**  
Retorna o timestamp do elem na posição i do Heap
- **double getRevisionId(Heap h, int i);**  
Retorna o revisionId do elem na posição i do Heap
- **double getArticleId(Heap h, int i);**  
Retorna o articleId do elem na posição i do Heap

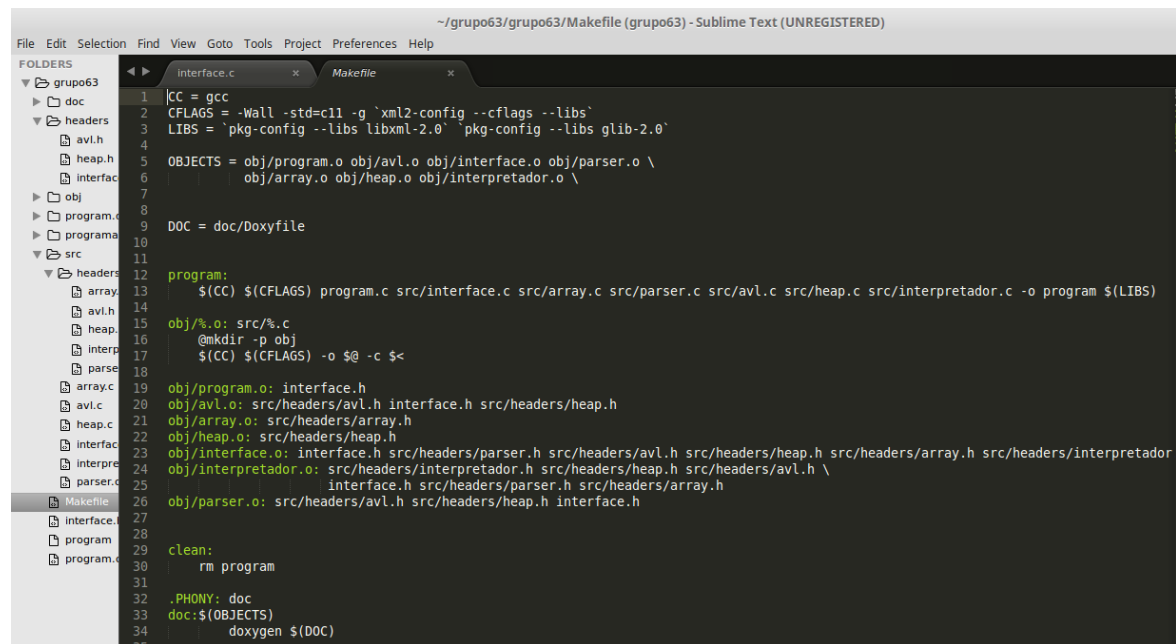
### 3 Programa Principal



O nosso programa principal começa por inicializar todas as estruturas necessárias para armazenar a informação contida nos ficheiros XML que o programa se prepara para ler.

Depois de os dados estarem guardados devidamente na estrutura, vamos implementar dez queries que queremos que o nosso programa responda da maneira mais eficaz possível. Este programa dá include de outro chamado `interface.h`, que é onde estão contidos todos os protótipos das queries a implementar pois a sua implementação vai ser feita no `interface.c`.

## 4 Makefile



```
1 CC = gcc
2 CFLAGS = -Wall -std=c11 -g `xml2-config --cflags --libs`
3 LIBS = `pkg-config --libs libxml-2.0` `pkg-config --libs glib-2.0`
4
5 OBJECTS = obj/program.o obj/avl.o obj/interface.o obj/parser.o \
6           obj/array.o obj/heap.o obj/interpretador.o \
7
8
9 DOC = doc/Doxyfile
10
11
12 program:
13     $(CC) $(CFLAGS) program.c src/interface.c src/array.c src/parser.c src/avl.c src/heap.c src/interpretador.c -o program $(LIBS)
14
15 obj/%.o: src/%.c
16     @mkdir -p obj
17     $(CC) $(CFLAGS) -o $@ -c $<
18
19 obj/program.o: interface.h
20 obj/avl.o: src/headers/avl.h interface.h src/headers/heap.h
21 obj/array.o: src/headers/array.h
22 obj/heap.o: src/headers/heap.h
23 obj/interface.o: interface.h src/headers/parser.h src/headers/avl.h src/headers/heap.h src/headers/array.h src/headers/interpretador.h
24 obj/interpretador.o: src/headers/interpretador.h src/headers/heap.h src/headers/avl.h \
25                     interface.h src/headers/parser.h src/headers/array.h
26 obj/parser.o: src/headers/avl.h src/headers/heap.h interface.h
27
28
29 clean:
30     rm program
31
32 .PHONY: doc
33 doc:$(OBJECTS)
34     doxygen $(DOC)
```

### 4.1 Estrutura

A Makefile permite-nos compilar todo o nosso programa para que seja possível executá-lo.

Como podemos verificar pela figura, estamos a guardar na variável CC o compilador que estamos a usar que é o gcc, seguida da variável CFLAGS onde adicionamos todas as flags utilizadas por opção ao compilar. Seguido destas temos LIBS onde referimos todas as bibliotecas externas que estamos a utilizar. De seguida temos OBJECTS onde guardamos todos os ficheiros objetos que vamos utilizar, e por fim em DOC o Doxyfile necessário para gerar a documentação.

### 4.2 Comandos

#### 4.2.1 make program ou make

Comando default da Makefile para compilar o nosso programa. Este comando gera o executável e se o executável já existir remove-o e volta a criar um.

#### 4.2.2 make clean

Comando responsável por remover o executável criado no ato de compilação.

#### 4.2.3 make doc

Comando usado para gerar a documentação feita no código do programa.

## 4.3 Dependências

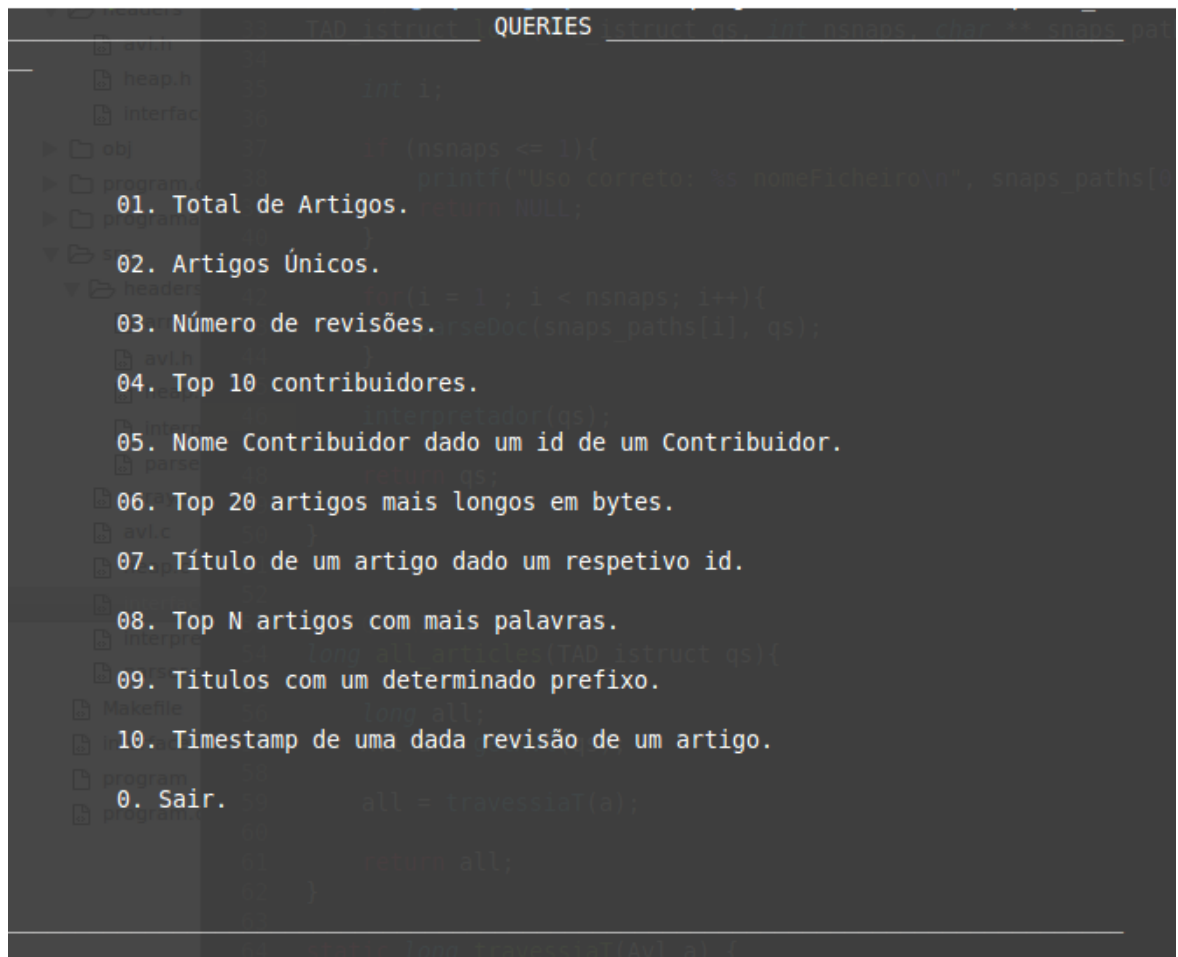
```
obj/program.o: interface.h
obj/avl.o: src/headers/avl.h interface.h src/headers/heap.h
obj/array.o: src/headers/array.h
obj/heap.o: src/headers/heap.h
obj/interface.o: interface.h src/headers/parser.h src/headers/avl.h src/headers/heap.h src/headers/array.h src/headers/interpretador
obj/interpretador.o: src/headers/interpretador.h src/headers/heap.h src/headers/avl.h \
    interface.h src/headers/parser.h src/headers/array.h
obj/parser.o: src/headers/avl.h src/headers/heap.h interface.h
```

Na figura acima podemos verificar a dependência que cada ficheiro objeto têm com os outros ficheiros.

## 5 Interface com o utilizador

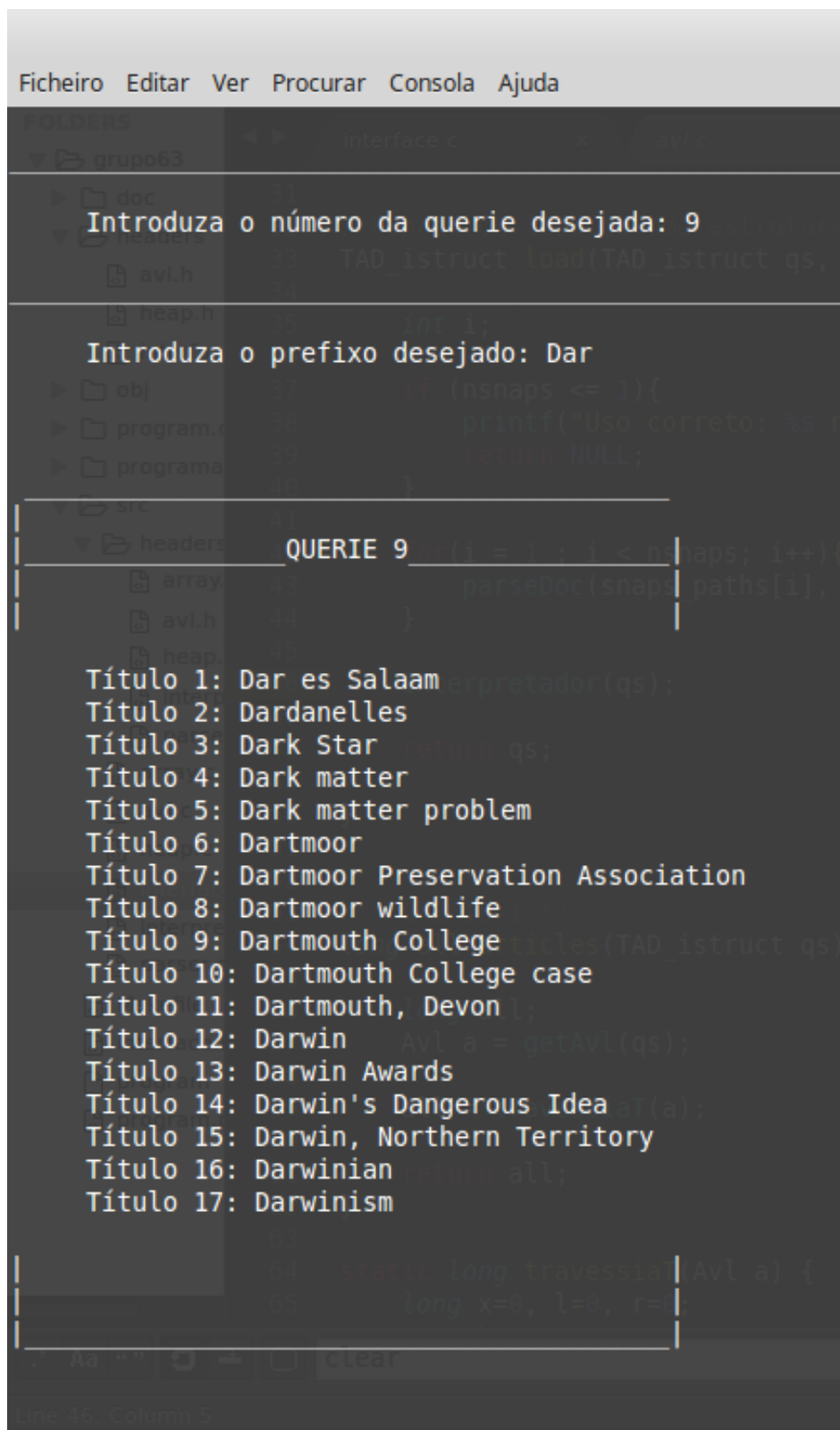
Quando um dado utilizador executa o nosso programa, a primeira coisa que lhe é apresentada é um Menu, onde estão presentes todas as funcionalidades do nosso programa.

Quando o Menu é apresentado o init e o load da estrutura já foram previamente executados, ou seja, a estrutura já foi inicializada e os dados já se encontram carregados na respetiva estrutura.

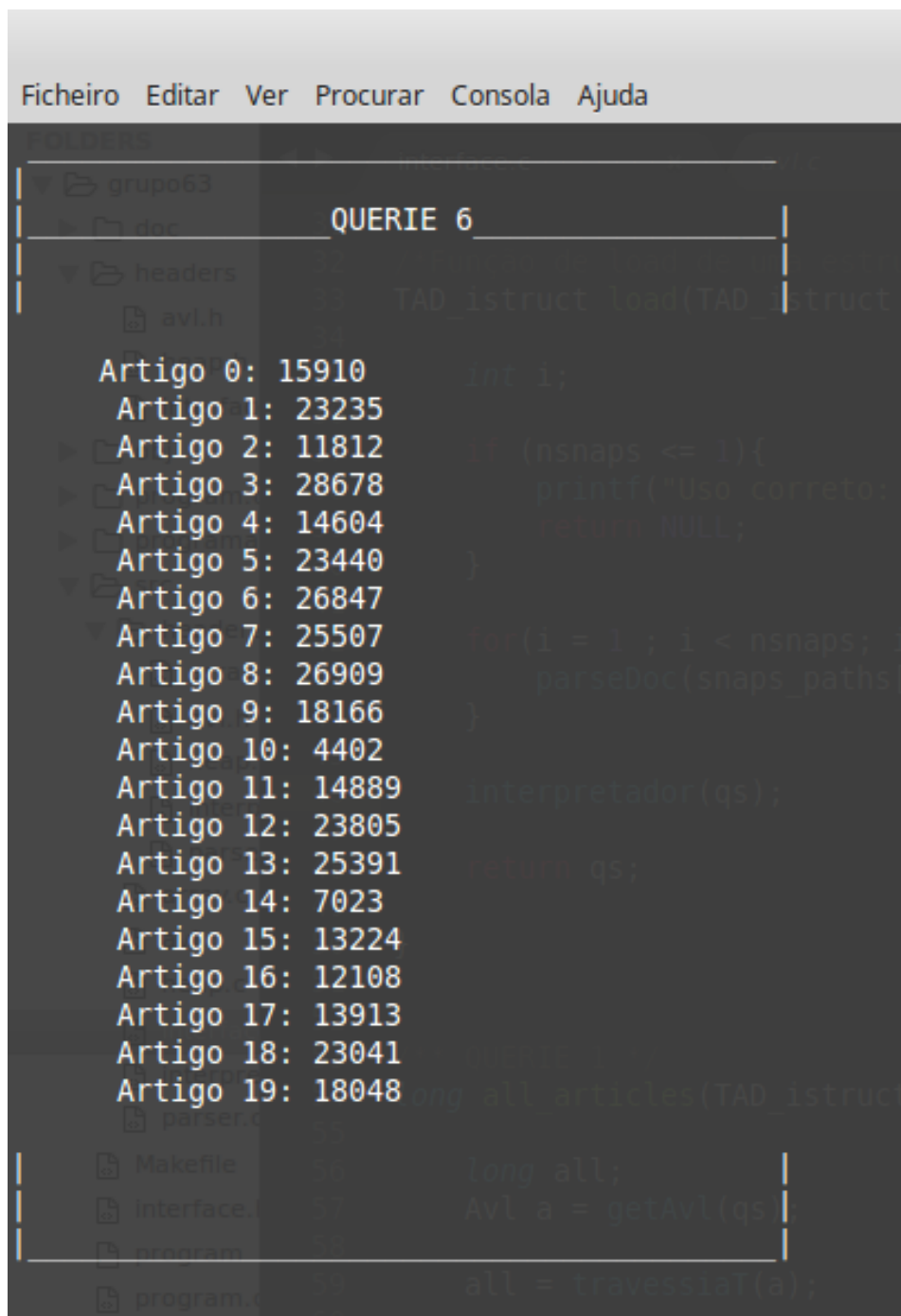


```
01. Total de Artigos.
02. Artigos Únicos.
03. Número de revisões.
04. Top 10 contribuidores.
05. Nome Contribuidor dado um id de um Contribuidor.
06. Top 20 artigos mais longos em bytes.
07. Título de um artigo dado um respetivo id.
08. Top N artigos com mais palavras.
09. Titulos com um determinado prefixo.
10. Timestamp de uma dada revisão de um artigo.
0. Sair.
```

A nossa interface recebe grande parte das vezes números como comandos, é exemplo disso o comando 0 que funciona sempre como um comando para sair da interface. Existem no entanto excepções a esta regra que é o caso de algumas queries presentes na interface, quando pedem um prefixo, ou um id de um artigo, por exemplo.



Tendo em conta que os resultados por vezes são apresenatdos na forma de conjuntos numéricos ou de strings decidimos apresentar estes na forma de uma lista.



Toda esta interface para o utilizador, apesar de simples, foi feita por nós de maneira a que o utilizador pudesse tirar o maior proveito da mesma, com comandos simples e fáceis de memorizar. Foi também preocupação nossa fazer com que a interface fosse fácil para o utilizador se acostumar, com pouco tempo de utilização.

# 6 Documentação

LI3\_grupo63

Main PageFiles

Search

File List

Here is a list of all files with brief descriptions:

src

headers

array.h

avl.h

heap.h

interpretador.h

parser.h

Protótipos das funções que trabalham com Arrays

Protótipos das funções que trabalham com AVL's

Protótipos das funções que trabalham com Heaps

Protótipos das funções relacionadas com o interpretador

Protótipos das funções responsáveis por tratar dos ficheiros

Generated on Sun Apr 30 2017 17:26:30 for LI3\_grupo63 by Doxygen 1.8.13

src	
headers	
array.h	Protótipos das funções que trabalham com Arrays
avl.h	Protótipos das funções que trabalham com AVL's
heap.h	Protótipos das funções que trabalham com Heaps
interpretador.h	Protótipos das funções relacionadas com o interpretador
parser.h	Protótipos das funções responsáveis por tratar dos ficheiros

array.h File Reference

Protótipos das funções que trabalham com Arrays. More...

#include <string.h>  
#include <stdlib.h>  
Include dependency graph for array.h:

Go to the source code of this file.

Typedefs

typedef struct array \* Array  
Declaração do tipo Array, um tipo abstrato. More...

Functions

Array

initArray (int size)  
Inicia a estrutura Array com um dado tamanho, alocando espaço para a mesma. More...

int insertAt (Array a, long info, long total)  
Insere num Array a informação pretendida. More...

int getArrayUsed (Array a)  
Retorna o tamanho do Array utilizado até ao momento. More...

long getTotal (Array a, int i)  
Retorna o total da inf. More...

long getInfInfo (Array a, int i)  
Retorna a info da inf. More...

int findIndex (long id, Array a)  
Obtém a localização da inf pretendida. More...

parser.h File Reference

Protótipos das funções responsáveis por tratar dos ficheiros. More...

#include <stdio.h>  
#include <string.h>  
#include <time.h>  
#include <stdlib.h>  
#include <libxml/xmlmemory.h>  
#include <libxml/parser.h>  
#include "../interface.h"  
Include dependency graph for parser.h:

Go to the source code of this file.

Functions

int \* count (xmlChar \*key, int \*contagem)  
Responsável por contar as palavras e bytes do texto. More...

void parseToStruct (char \*title, long idCont, char \*nameCont, int totalWords, char \*timestamp, long revisionId, long id, int nBytes)  
Responsável por guardar o parsing na estrutura. More...

void parsePage (int number, xmlDocPtr doc, xmlNodePtr cur)  
Responsável por fazer o parsing das páginas do documento. More...

TAD\_istruct parseDoc (char \*docname, TAD\_istruct parser)  
Responsável por fazer o parsing do documento. More...



▼ src	
▼ headers	
array.h	Protótipos das funções que trabalham com Arrays
avl.h	Protótipos das funções que trabalham com AVL's
heap.h	Protótipos das funções que trabalham com Heaps
interpretador.h	Protótipos das funções relacionadas com o interpretador
parser.h	Protótipos das funções responsáveis por tratar dos ficheiros

array.h File Reference	Typedefs   Functions
Protótipos das funções que trabalham com Arrays. More... <pre>#include &lt;string.h&gt; #include &lt;stdlib.h&gt;</pre> Include dependency graph for array.h:  Go to the source code of this file.	
<b>Typedefs</b> typedef struct array * Array Declaração do tipo Array, um tipo abstrato. More...	
<b>Functions</b> Array InitArray (int size) Inicia a estrutura Array com um dado tamanho, alocando espaço para a mesma. More... int InsertInt (Array a, long info, long total) Insere num Array a informação pretendida. More... int getArrayUsed (Array a) Retorna o tamanho do Array utilizado até ao momento. More... long getTotalTotal (Array a, int i) Retorna o total da int. More... long getTotalInfo (Array a, int i) Retorna a info da int. More... int findIndexes (long id, Array a) Retorna a localização dos id encontrados. More...	

As figuras apresentadas acima remetem-nos para documentação .erada pelo código do nosso programa.

A documentação desempenha um papel importante nas questões de segurança pois sem a devida documentação, bug's e pontos vulneráveis no programa demoram a ser encontrados e corrigidos.

## 7 Conclusão

Este foi o projeto que até ao momento gerou um maior número de dificuldades a todos elementos do grupo. Ao longo da realização do mesmo fomos confrontados com situações novas que puxaram pelo nosso espírito de autonomia e também desenvolvemos uma maior capacidade avaliativa de quando tomamos uma decisão passando pelos prós e pelos contras da mesma.

Todas as estruturas que usamos na realização deste projecto, desde as AVLs até às Heaps, requeriam um conhecimento prévio de conceitos abordados na unidade curricular de Algoritmos e Complexidade.

Tivemos necessidade de aprender como funciona a biblioteca do XML de modo a conseguirmos efetuar o parse dos snapshots e guardar as informações úteis nas estruturas referidas acima.

O facto de haver necessidade de utilizar tipos opacos foi também uma das nossas grandes dificuldades, algo novo para nós e que nos obrigou a estruturar o trabalho de maneira completamente distinta do que estávamos habituados até então, como por exemplo a clonagem de uma string/estrutura antes de a retornar, caso contrário o utilizador poderia ter acesso á mesma através do apontador.

Todo este processo de realização do projeto serviu para o grupo consolidar matéria lecionada em outras unidades curriculares e ganhar uma maior experiência na realização de trabalhos em equipa.