

Universidade do Minho

LEI 2º Ano 2º Semestre

LI3 - Projeto em C

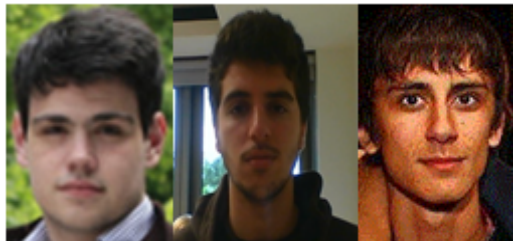
Gestauts

Uma aplicação de criação, gestão e consulta de um catálogo de publicações

Grupo 12

Tiago Cunha a67741, Daniel Caldas a67691, Xavier Rodrigues a67676

7 de Maio de 2014



Conteúdo

1	Introdução	3
2	Estruturas de Dados	4
2.1	Módulo Índice de Autores	4
2.1.1	A estrutura de dados do módulo	4
2.1.2	Variáveis privadas ao módulo	4
2.1.3	Desenho da estrutura de dados	5
2.1.4	Função de inicialização	5
2.1.5	Funções de inserção	5
2.1.6	Funções que libertam memória	6
2.1.7	Funções que providenciam encapsulamento do módulo . .	6
2.1.8	Outras funções	7
2.1.9	Função de fecho do módulo	7
2.2	Módulo Estatística	8
2.2.1	A estrutura de dados do módulo	8
2.2.2	Variáveis privadas ao módulo	8
2.2.3	Uma nota importante sobre a memória desperdiçada . . .	8
2.2.4	Desenho da estrutura de dados	9
2.2.5	Função de Inicialização	9
2.2.6	Funções de Inserção	9
2.2.7	Funções que libertam memória	10
2.2.8	Funções que providenciam o encapsulamento do módulo .	10
2.2.9	Função de fecho do módulo	10
2.3	Módulo Catálogo de Autores	12
2.3.1	A estrutura de dados do módulo...	12
2.3.2	Funções privadas ao módulo	13
2.3.3	Desenho da estrutura de dados	14
2.3.4	Funções de inserção	15
2.3.5	Outras funções	16
2.3.6	Funções que providenciam o encapsulamento do módulo .	16
2.3.7	Funções que libertam memória	18
2.3.8	Função de fecho do módulo	18
2.4	Ficheiro main.c e módulo leitura	19
2.4.1	Algumas funções auxiliares relevantes...	19
3	Interface do Utilizador e opções de navegação	20
4	Testes de Performance	22
4.1	Código utilizado nos testes	22
4.2	Resultados	22
5	Makefile	24
5.1	Opções de compilação	24
5.2	Grafo de dependências da makefile	25
6	Conclusão	26

1 Introdução

Foi proposto ao grupo de trabalho a elaboração de uma aplicação de gestão de um catálogo de autores em linguagem de programação C seguindo princípios de **modelaridade** e **encapsulamento** (propriedades naturais de linguagens OO como o JAVA). Para o desenvolvimento do projeto são fornecidos três ficheiros à volta dos quais se desenvolve todo o projeto, os quais foram minuciosamente analisados tendo a primeira análise (mais superficial) resultado nos seguintes pontos:

- O ficheiro contém uma forma de publicações fixa do estilo: “nome de autor, nome de autor . . . , ano da publicação”;
- Existem autores cujos nomes possuem caracteres especiais no início (e não só), existe pelo menos um autor cujo nome se inicia por letra minúscula (evahn), ora estes últimos dois casos serão tratados à parte dos restantes.
- Os ficheiros são de grande extensão pelo que devemos armazenar a informação recolhida do ficheiro em estruturas de dados que não só ocupem o menos memória possível mas que acima de tudo **organizem a informação** de modo a que esta possa ser posteriormente acedida mediante um *contexto* (query) e em *tempos aceitáveis*;

Relativamente à **arquitetura da aplicação**, não foi feita qualquer alteração à proposta pelos docentes, os módulos e suas funcionalidades são as previstas.

O **código** desenvolvido com fim a dar solução ao problema proposto segue algumas normas de sintaxe e estruturação pré-definidas pelo grupo uma delas é, o nome de cada função tem sempre um prefixo que nos indica qual o módulo sobre o qual opera a função, portanto todas as funções relacionadas com o módulo índice de autores têm o prefixo *IA* as do módulo estatística o prefixo *E* e as do módulo catálogo de autores o prefixo *CA*, mais tarde viemos a concluir que esta foi de facto uma boa medida.

Em cada módulo o código encontra-se seccionado, cada secção é composta por funções que tem um objetivo comum i.e operam com fins específicos, o que permitiu ao grupo de trabalho ao longo deste projeto fácil e rapidamente reimplementar soluções, consultar partes muito específicas do código e a manutenção do mesmo. Eis a estrutura externa generalizada, comum aos módulos Índice de Autores, Estatística e Catálogo de Autores:

- a) include dos respectivos ficheiros .h;
- b) Estrutura de dados, variáveis privadas ao módulo e assinaturas das funções privadas ao módulo;
- c) Função (ou funções) de inicialização do módulo;
- d) Funções de inserção na estrutura de dados do módulo;
- e) Funções standard de manipulação das árvores AVL para os módulos índice e catálogo de autores;
- f) Funções que libertam a memória alocada dinamicamente;
- g) Funções que permitem aceder a **cópias da informação** armazenada na estrutura de dados após leitura do ficheiro, **de modo a respeitar o encapsulamento dos dados**;
- h) Função de fecho que impossibilita inserções na estrutura de dados após término da leitura do ficheiro.

2 Estruturas de Dados

2.1 Módulo Índice de Autores

2.1.1 A estrutura de dados do módulo

No ficheiro índice-autores.h definimos o **tipo incompleto** (ou **tipo abstracto de dados**):

```
typedef struct publxAuthorTREE *publxAUTREE\_TAD;
```

No ficheiro índice-autores.c **concretiza-se o tipo incompleto de dados** com a estrutura de dados em si, i.e o **tipo completo de dados**:

```
typedef char TreeKey;
typedef enum balancefactor { LH , EH , RH } BalanceFactor;
typedef struct publxAuthorTREE{
    TreeKey *name; //o nome do autor
    BalanceFactor bf; //o fator de balanceamento de um nodo de uma árvore AVL
    struct publxAuthorTREE *left; //o apontador para o nodo à esquerda
    struct publxAuthorTREE *right; //o apontador para o nodo à direita
}publxAUTREE_TCD;
```

2.1.2 Variáveis privadas ao módulo

```
/*Modul private variables*/
static int countAUTHORS; /*número de autores diferentes encontrados*/
static char* smallest_name; /*nome mais pequeno*/
static unsigned int smallest_size; /*tamanho do nome mais pequeno*/
static char* biggest_name; /*nome maior*/
static unsigned int biggest_size; /*tamanho do nome maior*/
static int sum_names_length; /*somatório do comprimento dos nomes para cálculo da média*/
static int flagFULL; /*bandeira para fechar módulo no final da leitura*/

/*Array of pointers to AVL trees*/
static publxAUTREE_TCD** pointersToAVLtrees; /*array de apontadores para árvores*/
static int* numberAuthorsEachLetter; /*array que contém total de autores (nodos) de cada árvore*/
```

Figura 1: Variáveis privadas ao módulo Índice de Autores

2.1.3 Desenho da estrutura de dados

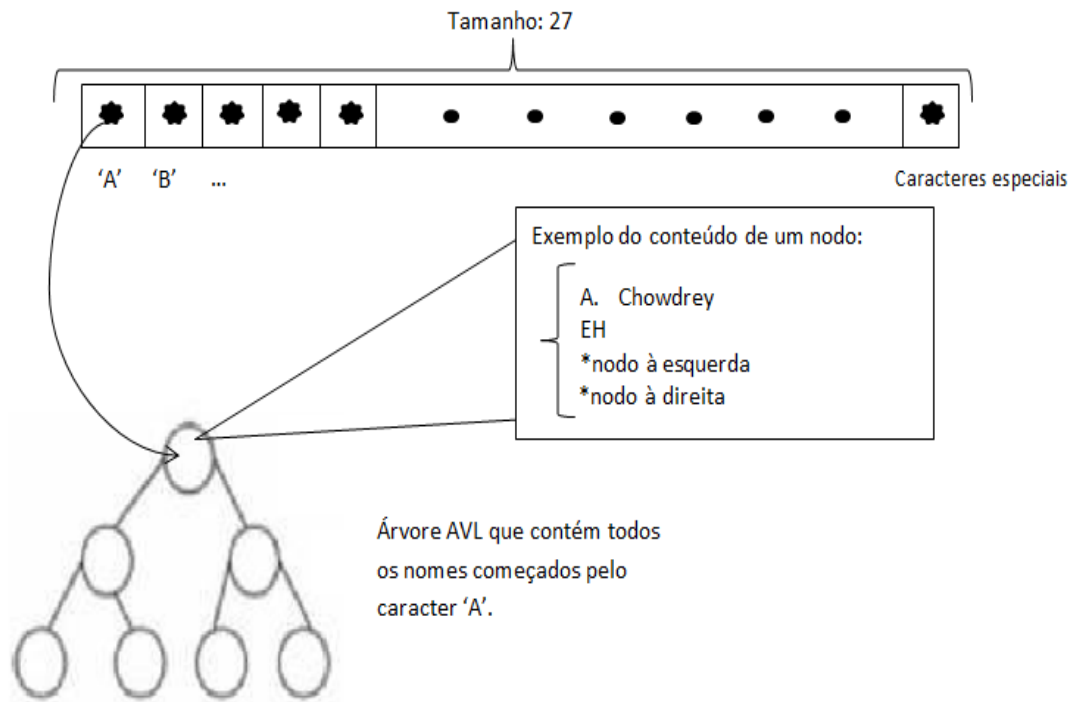


Figura 2: Desenho da estrutura de dados do módulo. O array *pointersToAVLtrees* do módulo índice autores como podemos ver pela figura é um array de 27 apontadores para diferentes árvores AVL, cada apontador representa por assim dizer uma letra, note-se que o apontador **aponta sempre para a raiz da árvore** pelo que a quando inserções de nomes na árvore o mesmo apontador tem de ser actualizado.

2.1.4 Função de inicialização

void IA_Init() - inicializa as variáveis privadas ao módulo;

2.1.5 Funções de inserção

*void IA_Store_Authors (char *author_names[], int number_of_authors)*
- controla ciclo de chamadas de inserção nas árvores.

static publxAUTREE_TCD IA_Insert_AuthorsTree(publxAUTREE_TCD* authorsTREE, char * name , int *cresceu, int avlTreeIndex)* - insere o nome passado como argumento na devida árvore caso ainda não exista.

2.1.6 Funções que libertam memória

void IA_ReleaseAuthorsTree() - função que liberta memória ocupada pelas AVL.

static publxAUTREE_TCD* IA_DeallocateTREE (publxAUTREE_TCD* authorsTREE) - liberta memória de cada nodo de uma árvore através de uma travessia.

2.1.7 Funções que providenciam encapsulamento do módulo

int IA_getCountAUTHORS() - retorna o número autores diferentes encontrados no ficheiro;

char* IA_getSmallest() - retorna um apontador para uma string que é uma cópia do nome mais pequeno encontrado no ficheiro.

char* IA_getBiggest() - retorna um apontador de uma string que é uma cópia do nome maior encontrado no ficheiro.

unsigned int IA_getSmallestSize() - retorna o tamanho do nome mais pequeno.

unsigned int IA_getBiggestSize() - retorna o tamanho do nome maior.

int IA_getNamesMediumLength() - devolve o comprimento médio de todos os nomes diferentes encontrados.

int IA_getTotalLetter(char c) - retorna o total de nomes começados pela letra passada como parâmetro.

char** IA_getAuthorsListNames(char c) - retorna um apontador para um array que contém a cópia dos nomes da árvore correspondentes à letra passada como parâmetro.

static char** IA_CopyTree(char** names, publxAUTREE_TCD* a) - retorna o apontador para o array de nomes atualizado. Função que na realidade faz a travessia da árvore copiando nome a nome de cada nodo da árvore para o array de nomes passando recursivamente o apontador para esse array como parâmetro.

2.1.8 Outras funções

static void IA_BiggerSmallerNames (char * name) - efetua cálculos intermédios tais como: cálculo do maior e mais pequeno nome e média do comprimento dos nomes.

static int IA_checkIndex (char c) - retorna o índice da árvore onde o nome começado pela letra passada como parâmetro deve ser inserido. (*uma espécie de HashCode()*).

2.1.9 Função de fecho do módulo

void IA_CloseInsertions() - fecha as inserções ao módulo através de uma flag privada ao mesmo, reforçando a robustez do módulo e o encapsulamento.

2.2 Módulo Estatística

2.2.1 A estrutura de dados do módulo

No ficheiro estatistica.h definimos o **tipo incompleto**:

```
typedef struct publxEAR_TCD *publxEAR_TAD
```

No ficheiro estatistica.c está definido o **tipo concreto de dados**:

```
typedef struct publicxYEAR{  
    int year; //o ano  
    int *counter_authors;  
    int max_number_authors; //publicação com mais autores nesse ano  
    int counter_total; //total de publicações nesse ano  
}publxEAR_TCD;
```

counter_authors – Array que contém para um dado ano as quantidades de artigos publicados com 1, 2, 3 ... max_number_authors autores. Através desta implementação obtivemos tempo constante (após encontrado o ano) para perguntas do tipo: "No ano X quantos artigos foram publicados com Y autores? A **resposta** é automática apenas temos de consultar a posição **Y-1** do array **counter_authors**, claro que se Y é maior max_number_authors a resposta é 0 autores. Esta implementação tem apenas a desvantagem de deixar por vezes células do array a 0 desperdiçando memória como veremos mais à frente.

2.2.2 Variáveis privadas ao módulo

```
/*Modul private variables*/  
static publxEAR_TCD *datesStore; /*array de estruturas do módulo*/  
static int countYEAR; /*número de anos diferentes em que foram publicados artigos*/  
static int min_year; /*menor ano encontrado*/  
static int max_year; /*maior ano encontrado*/  
static int flagFULL; /*bandeira para fechar módulo no final da leitura*/
```

Figura 3: Variáveis privadas ao módulo Estatística

2.2.3 Uma nota importante sobre a memória desperdiçada

A função que fecha o acesso a tentativas de inserções na estrutura de dados após término da leitura do ficheiro, também faz o **acerto das dimensões dos arrays counter_authors** com o fim de **reduzir a memória desperdiçada**, que se conclui após efectuados os cálculos ser de 352 bytes para publicx.txt e 356 bytes para os restantes ficheiros. **Por ser tão pouca a memória desperdiçada e tendo também em conta a capacidade de qualquer máquina nos dias de hoje o grupo optou por esta solução pela sua simples implementação e facilidade de consulta.**

2.2.4 Desenho da estrutura de dados

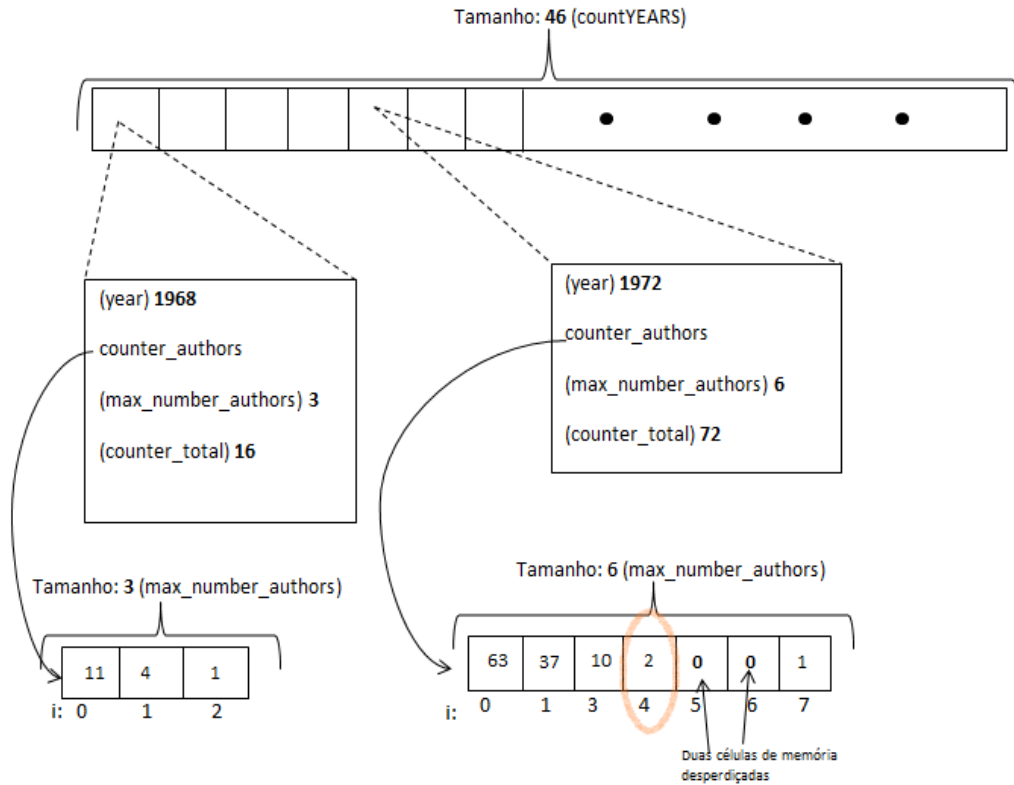


Figura 4: Desenho da estrutura de dados do módulo (array de estruturas encontra-se ordenado por ordem crescente dos anos). *Quantas publicações em 1972 com 5 autores?* $5-1=4$, na posição 4 do array co_authors da estrutura do ano 1972 temos **2 publicações**.

2.2.5 Função de Inicialização

void E_Init() - inicializa as variáveis privadas ao módulo.

2.2.6 Funções de Inserção

void E_Estatistica (int year_of_publication, int number_of_authors) - armazena na estrutura de dados o ano da publicação e inicializa os campos desse elemento da estrutura de dados caso o ano ainda não exista, caso contrário atualiza os devidos campos da estrutura de dados correspondentes ao ano passado como parâmetro.

static int E_Insertion_Sort (int year_of_publication) - percorre o array de estruturas a partir do fim fazendo shifts à direita enquanto o ano que

queremos inserir for menor que o da estrutura que estamos a visitar, no final devolve o índice onde deve ser inserido o novo elemento.

static void E_Copy_Structure (int a, int b) - copia a estrutura de dados da posição a para a posição b.

static void E_Copy_Array (int a, int b) - auxiliar da função *E_Copy_Structure*, apenas faz a cópia do array *counter_authors* (campo dos elementos do array de estruturas).

2.2.7 Funções que libertam memória

void E_Release_Mem () - liberta a memória da estrutura de dados e respectivos campos alocados dinamicamente.

2.2.8 Funções que providenciam o encapsulamento do módulo

int** E_getAuthorsPublicationsTable() - retorna uma matriz com todos os arrays (cópias) *counter_authors* de todas as estruturas (útil para gerar ficheiro com ext. .csv).

int E_getMaxYear () - retorna o máximo ano em que foi publicado um artigo.

int E_getMinYear() - retorna o mínimo ano em que foi publicado um artigo.

int E_getCountYEAR() - retorna o número diferente de anos em que foram publicados artigos (tamanho do array de estruturas de dados).

int* E_getYearsOfPublications() - retorna um apontador para um array que recolhe da estrutura de dados todos os anos diferentes em que foram publicados artigos (por ordem crescente do ano).

int* E_getPublicationsPerYear() - retorna um apontador para um array que recolhe da estrutura de dados a quantidade de publicações ano a ano.

void E_getLongestPublication(int * year, int * number_of_authors) - atualiza os inteiros cujo endereço foi passado como parâmetro colocando nesses inteiros informação relativa à publicação mais longa do ficheiro, i.e o ano dessa publicação e o número de autores.

int E_getPublicationsInYear(int year_of_publication) - retorna o número de publicações desse ano.

2.2.9 Função de fecho do módulo

void E_CloseInsertions() - Fecha as inserções ao módulo através de uma flag privada ao mesmo, reforçando a robustez do código e o encapsulamento. Faz

também operações adicionais de ajustar o tamanho dos arrays `conter_authors` dos diferentes elementos do array de estruturas de modo a reduzir o máximo possível o desperdício de memória.

2.3 Módulo Catálogo de Autores

2.3.1 A estrutura de dados do módulo...

Tipo incompleto definido no ficheiro catálogo_autores.h:

```
typedef struct publicxCAUTHORS* publxCAUTHORS_TAD;
```

Concretização do tipo incompleto com a estrutura de dados primária (**tipo concreto de dados**) do módulo no ficheiro .c:

```
typedef struct publicxCAUTHORS{
    char *name; //o nome do autor
    struct StoreCountCoAutRelations *co_authors;
    int number_of_co_authors; //número de co-autores diferentes do autor
    int maximus; //número de publicações do autor com o co-autor com quem mais publicou
    struct yearsNPUBLICATIONS *years_npublications;
    int nyears; //número de anos diferentes em que publicou
    int total_publications; //total de publicações do autor
    BalanceFactor bf; //bf, left e right igual à estrutura índice de autores
    struct publicxCAUTHORS *left;
    struct publicxCAUTHORS *right;
}publxCAUTHORS_TCD;
```

*struct StoreCountCoAutRelations *co_authors* – um array de estruturas que armazena por ordem decrescente do número de coautorias todos os co-autores do autor da estrutura de dados primária. A estrutura secundária de dados em questão tem a seguinte forma:

```
typedef struct StoreCountCoAutRelations{
    publxCA coaut; //apontador para co-autor (um nodo de uma AVL)
    int counter; //nº de co-autorias
}publxCOAUTR;
```

*struct yearsNPUBLICATIONS *years_npublications* – array de estruturas que guarda o número de publicações do autor associado a um dado ano. A forma da estrutura de dados é a seguinte:

```
typedef struct yearsNPUBLICATIONS{
    int year; //ano das publicações
    int counter; //nº de publicações do autor nesse ano
}yearNPUBLX;
```

2.3.2 Funções privadas ao módulo

```
/*Modul private variables*/  
static int countYEAR; /*número de anos diferentes*/  
static publxCa *pointersToCoAuthors; /*guarda apontadores para autores "recém-inseridos"*/  
static int pointersINDEX; /*nº de autores efetivamente inseridos*/  
static int flagFULL; /*flag para fecho do módulo*/  
  
/*Pointer to AVL tree of each letters same algorithm in modul indice_autores*/  
static publxCa* authors_catalog; /*array de AVL's BASE DO GRAFO*/  
static publxCYEARS* years_catalog; /*array de estruturas (optimização)*/
```

Figura 5: Variáveis privadas ao módulo

2.3.3 Desenho da estrutura de dados

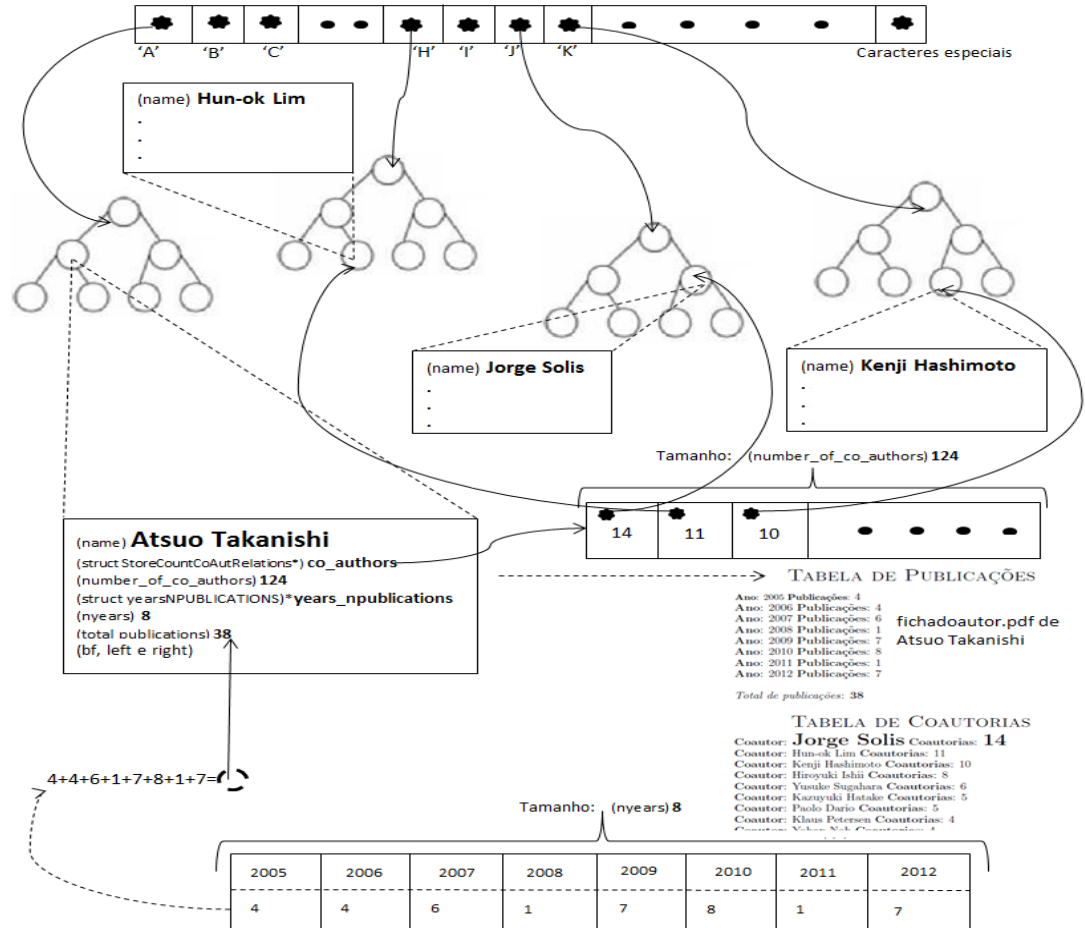


Figura 6: Os dados usados no exemplo são retirados da aplicação com o fim de ilustrar um cenário próximo do real. É de salientar que os rectângulos dos co-autores são estruturas de dados primárias tais como as do autor *Atsuo Takanishi*. A miniatura na figura é um excerto do ficheiro (*fichadoautor.pdf*) que é gerado automaticamente pela querie 101.

2.3.4 Funções de inserção

void CA_Store_Authors (char *author_names[], int number_of_authors, int year_of_publication) - Primeiro a função faz inserções de nomes numa árvore AVL à semelhança do módulo índice de autores criando assim a "base" do grafo em que consiste este módulo. De seguida a função itera sobre o array de nomes e um array de apontadores para nodos da AVL relacionando os autores (coautorias) ao mesmo tempo que cria uma estrutura dados um pouco diferente que para cada ano armazena por ordem decrescente do número de publicações a lista de autores que publicaram nesse ano.

static void CA_InsertPublicationAuthor (publxCa this, int year_of_publication)

- Esta função ordena à medida que constroí um array de publicações para um autor "this" em que ou se atualiza o número de publicações do autor "this" no ano year_of_publication ou se aloca espaço para um novo ano em que esse mesmo autor tenha publicado.

static void CA_InsertCoAuthor (publxCa this, publxCa co_author) -

Associa o coautor ao autor "this" incrementando o contador de publicações em conjunto no array de coautorias de this ou caso estes ainda não tenham sido relacionados aloca memória para tal efeito.

static void CA_InsertAuthorInYearsCatalog (publxCa author, int year_of_publication)

- Função responsável por criar um "catálogo de anos" i.e numa estrutura de dados própria armazena associado a um ano a lista de todos os autores que publicaram nesse ano por ordem decrescente do número de artigos publicados.

char** CA_getMaxCoAuthories (char * name , int * number_of_publications)

- Retorna um apontador para uma lista de nomes (cópias) (a lista na maioria das situações contém só um nome) dos autores com quem o autor passado como parâmetro mais publicou.

char** CA_getNAuthorsInYear (int year_of_publication, int * N, int * values) - Retorna um array de cópias de nomes que correspondem aos N autores que mais publicaram no ano passado como parâmetro as respetivas quantidades de publicações ficaram guardadas no array values.

void CA_Init() - Quando chamada externamente inicializa as variáveis privadas ao módulo

static int CA_InsertionSort (publxCa this, int year_of_publication) - o índice onde deve ser inserido o novo ano em que o autor publicou, a função garante que o array fica sempre ordenado por ordem crescente do ano.

static int CA_Already_Exists_Year (publxC A this, int year_of_publication)- caso o autor ainda não tenha publicado nesse ano, ou o índice correspondente a esse ano no array de publicações do autor (years_npublications) para que na função chamadora se incremente o respetivo contador.

static void CA_OrderCoAuthorsByPublications (publxC A this, int index)- Ordena o array recentemente modificado de modo a que this tenha sempre um array de coautores ordenados por ordem decrescente do número de coautorias, i.e o autor com quem this mais publicou fica sempre à cabeça do array co_authors.

static void CA_SortAutsByPublications (publxC OAUTR * auts, int index)- Função que mantém ordem no array de estruturas de dados.

static int CA_InsertionYear_Sort (int yea_of_publication)-retorna o índice onde deve ser colocado o novo ano de modo a garantirmos ordem crescente pelo ano de publicação

static void CA_Copy_Structure (int a, int b)-Copia a estrutura de dados da posição a para a posição b.

static void CA_Copy_ArrayOfPointerToAuthors (int a, int b)- Apenas faz o swap de apontadores dos arrays de autoresj-çcontador.

2.3.5 Outras funções

static int CA_checkIndex (char c)- tem a mesma finalidade que a implementada no índice de autores.

2.3.6 Funções que providenciam o encapsulamento do módulo

int CA_getNumberOfPublicAuthor (char * author, int year_of_publication)- o número de publicações do autor passado como parâmetro no dado ano.

char** CA_getListCoAuthorsGivenName (char * name)- um apontador para um array de cópias dos nomes de todos os coautores do autor passado como parâmetro.

int CA_getNumberOfCoAuthorsGivenName(char * name)- o número de diferentes coautores associados ao nome do autor passado como para parâmetro.

int CA_getNumberOfSoloAuthors()- o numero de autores que publicaram a solo.

int CA_getNumberDiferentYearsOfPublications(char * name)- devolve o número de anos diferentes em que o autor passado como parâmetro publicou artigos.

static publCA CA_SearchSoloAuthors(publCA aux)-um apontador para um nodo da AVL (será NULL no final da pesquisa).

int* CA_getYearsArrayPublications(int * years, char * name)-um apontador de um array que contém os diferentes anos em que o autor passado como parâmetro publicou artigos

int* CA_getPublicationsPerYear(int * publications, char * name)-um novo apontador para um array que contém as quantidades de publicações do autor passado como parâmetro posteriormente a serem associadas a um ano.

char** CA_getMaxCoAuthories (char * name , int * number_of_publications)-um apontador para uma lista de nomes (cópias) (a lista na maioria das situações contém só um nome) dos autores com quem o autor passado como parâmetro mais publicou.

char** CA_AuthorsYearsInterval(int minYEAR, int maxYEAR, int * size)- um apontador para uma lista de cópias de nomes dos autores que publicaram em cada ano do intervalo fechado passado como parâmetro, adicionalmente guarda no inteiro size o tamanho dessa lista de autores.

static char** CA_searchInTree(publCA author, int minYEAR, int maxYEAR, int number_of_years, char* authors_names[], int * size)-um novo apontador para o array de nomes que é atualizado a cada chamada da função

char** CA_getNAuthorsInYear(int year_of_publication, int * N, int * values)-um array de cópias de nomes que correspondem aos N autores que mais publicaram no ano passado como parâmetro as respectivas quantidades de publicações ficaram guardadas no array values.

float CA_getPercentilsInYear(char * name, int year_of_publication)-um float que representa a percentagem de publicações do autor passado como parâmetro relativamente ao total de publicações do ano (também passado como parâmetro).

char* CA_getMaximumPublications(int * n)-o nome do autor com maior

total de publicações no ficheiro.

static publxC A CA_searchMaxInTree(publxC A author, int * max)- Função que procura em todas as árvores o máximo valor para total de publicações de um autor.

int* CA_getPublicationsInYear(int index)-um apontador para um array que contém os valores de publicações nesse ano a serem associados a um array de nomes de autores.

int CA_getNumberOfAuthorsInYear(int index)-o número diferente de autores que publicaram nesse ano.

char** CA_getNamesList(int index)-um array de nomes (cópias) complementa à função em cima CA_getPublicationsInYear.

int* CA_getCoAutsPublications(char * name)- um apontador para um array que contém as vezes que o autor passado como parâmetro publicou associado a cada um dos seus coautores.

2.3.7 Funções que libertam memória

void CA_ReleaseAuthorsCatalog()-Liberta memória para a estrutura de dados que armazena as árvores do catálogo de autores.

static publxC A CA_Deallocate(publxC A tree)-Liberta memória nodo a nodo da árvore AVL libertando também as respetivas variáveis de cada nodo.

void CA_ReleaseYears()- Liberta memória para a estrutura de dados que armazena o "catálogo de anos".

2.3.8 Função de fecho do módulo

void CA_CloseInsertions()-Fecha as inserções ao módulo através de uma flag privada ao mesmo, reforçando a robustez do código e o encapsulamento.

2.4 Ficheiro main.c e módulo leitura

- a) include dos respectivos ficheiros .h;
- b) Definição de macros e assinaturas das funções auxiliares ao módulo;
- c) Função **main**:
 - i) Imprime painel de entrada e lê opção escolhida pelo utilizador;
 - ii) Inicializar os módulos;
 - iii) Ciclo de leitura e tratamento da informação nos diversos módulos;
 - iv) Fecho dos módulos;
 - v) Menu Gestauts onde o utilizador interaje com o programa até que decida sair da aplicação;
 - vi) Quando o autor sai da aplicação liberta-se a memória utilizada pelas estruturas de dados dos diversos módulos;
- d) Funções de controlo input/output, são um canal de comunicação intermédio entre input do utilizador e funções auxiliares que fazem o trabalho que é solicitado pelo utilizador em função de cada querie, aqui também fazemos o tratamento de erros i.e validação do input do utilizador em função do contexto da querie.
- e) Funções auxiliares que utilizam os mecanismos por nós desenvolvidos para irem buscar **cópias** da informação às estruturas dos módulos e devolve ao utilizador tal informação com um certo formato.

2.4.1 Algumas funções auxiliares relevantes...

char* trim (char * token) - retorna apontador para nova string após remoção de espaços em branco no início e no fim do token ("aparar" a string).

void printMenu () - Imprime no ecrã o menu principal da aplicação;

void IA_Print_NamesStartedByLetter(char c) - Imprime páginas e escreve num ficheiro a listagem dos nomes começados pela letra passada como parâmetro.

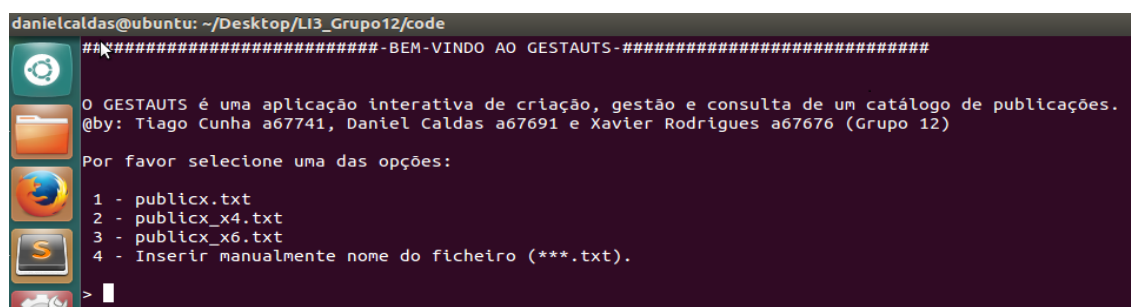
void E_Print_Publications_Per_Years(int number_of_publications) - Imprime no ecrã uma tabela de publicações descriminando a quantidade por ano.

void CA_TableOfPublicationsAuthor (char * name) - Escreve num ficheiro e imprime no terminal a tabela de publicações do autor cujo nome é passado como parâmetro.

void CA_NAuthorsGivenYear(int year_of_publication, int N) - Gera a lista dos N (passado como parâmetro) autores que mais publicaram no ano passado como parâmetro.

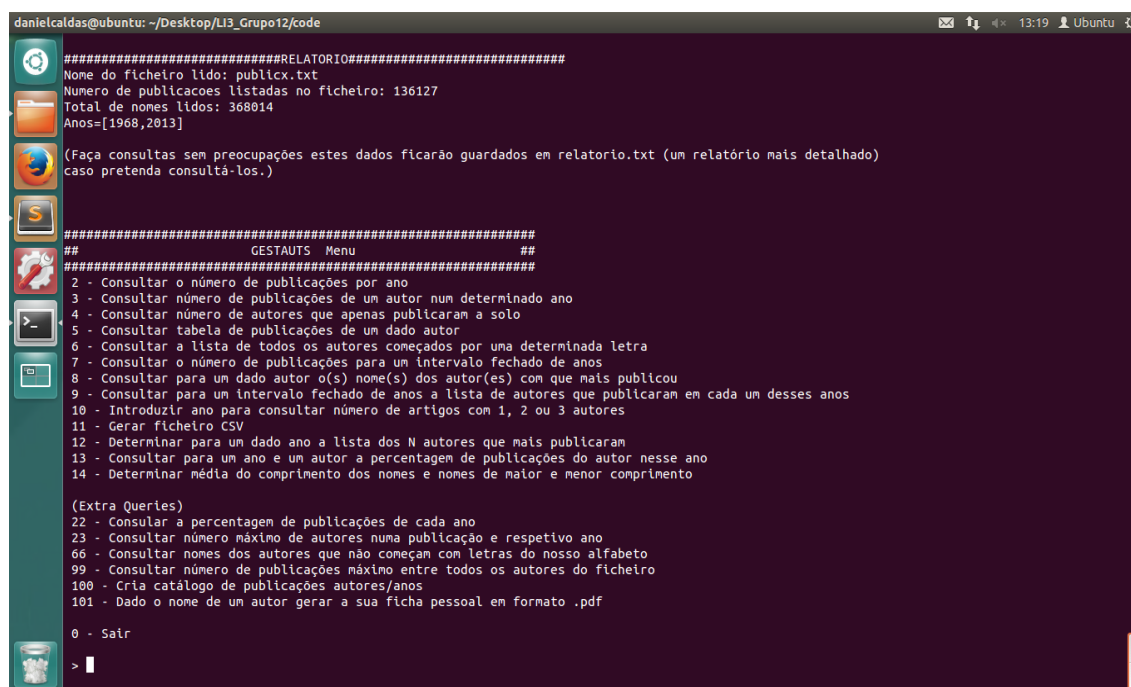
3 Interface do Utilizador e opções de navegação

Podemos observar na figura 7 o painel de entrada da aplicação e na figura 8 o menu principal onde o utilizador escolhe qual a operação que pretende efectuar baseando-se no índice numérico das queries.



```
danielcaldas@ubuntu: ~/Desktop/LI3_Grupo12/code
#####-BEM-VINDO AO GESTAUTS-#####
O GESTAUTS é uma aplicação interativa de criação, gestão e consulta de um catálogo de publicações.
@by: Tiago Cunha a67741, Daniel Caldas a67691 e Xavier Rodrigues a67676 (Grupo 12)
Por favor selecione uma das opções:
1 - publicx.txt
2 - publicx_x4.txt
3 - publicx_x6.txt
4 - Inserir manualmente nome do ficheiro (***.txt).
> 
```

Figura 7: Painel de entrada do Gestauts



```
danielcaldas@ubuntu: ~/Desktop/LI3_Grupo12/code
#####RELATORIO#####
Nome do ficheiro lido: publicx.txt
Numero de publicacoes listadas no ficheiro: 136127
Total de nomes lidos: 368014
Anos=[1968,2013]
(Faça consultas sem preocupações estes dados ficarão guardados em relatorio.txt (um relatório mais detalhado) caso pretenda consultá-los.)
#####
##          GESTAUTS  Menu          ##
#####
2 - Consultar o número de publicações por ano
3 - Consultar número de publicações de um autor num determinado ano
4 - Consultar número de autores que apenas publicaram a solo
5 - Consultar tabela de publicações de um dado autor
6 - Consultar a lista de todos os autores começados por uma determinada letra
7 - Consultar o número de publicações para um intervalo fechado de anos
8 - Consultar para um dado autor o(s) nome(s) dos autor(es) com que mais publicou
9 - Consultar para um intervalo fechado de anos a lista de autores que publicaram em cada um desses anos
10 - Introduzir ano para consultar número de artigos com 1, 2 ou 3 autores
11 - Gerar ficheiro CSV
12 - Determinar para um dado ano a lista dos N autores que mais publicaram
13 - Consultar para um ano e um autor a percentagem de publicações do autor nesse ano
14 - Determinar média do comprimento dos nomes e nomes de maior e menor comprimento

(Extra Queries)
22 - Consultar a percentagem de publicações de cada ano
23 - Consultar número máximo de autores numa publicação e respetivo ano
66 - Consultar nomes dos autores que não começam com letras do nosso alfabeto
99 - Consultar número de publicações máximo entre todos os autores do ficheiro
100 - Cria catálogo de publicações autores/anos
101 - Dado o nome de um autor gerar a sua ficha pessoal em formato .pdf

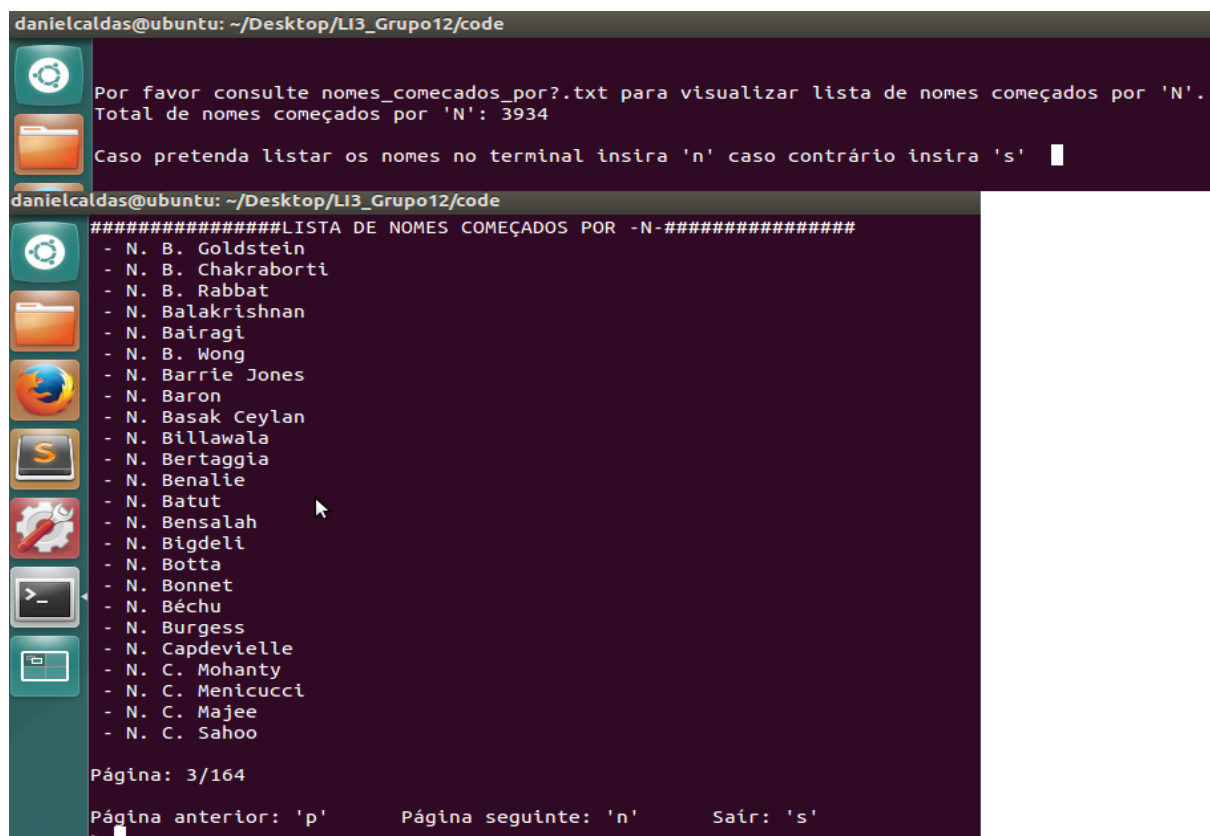
0 - Sair
> 
```

Figura 8: Menu principal

Para as queries em que é necessário listar maiores conteúdos foi pensada uma solução de apresentação de resultados diferente das restantes queries, que **permite ao utilizador navegar para trás, para a frente ou parar a consulta a qualquer instante** (queries 6, 9 e 12).

A função base para este tipo de navegação tem como nome ***IA_PrintPagesInTerminal*** e localiza-se na secção do código que diz respeito ao módulo índice de autores no ficheiro main.c. A função recebe sempre arrays com os conteúdos a listar, recebe um número total de páginas necessário para a listagem de tal conteúdo e é usado uma variável índice que incrementa ou decrementa consoante o utilizador queira navegar nos dados para a frente ou para trás respectivamente.

Nas queries atrás mencionadas *é sempre guardada em ficheiro a informação listada* pelo que oferecemos **uma alternativa de consulta ao utilizador** bem como a possibilidade de este ficar com uma cópia dos dados que visualizou no terminal. É o utilizador quem decide sempre se quer listar tais conteúdos ou se pretende continuar a utilizar a API sabendo que ficou com uma cópia do conteúdo ficheiro.



```
danielcaldas@ubuntu: ~/Desktop/LI3_Grupo12/code
Por favor consulte nomes_comecados_por?.txt para visualizar lista de nomes começados por 'N'.
Total de nomes começados por 'N': 3934
Caso pretenda listar os nomes no terminal insira 'n' caso contrário insira 's'

danielcaldas@ubuntu: ~/Desktop/LI3_Grupo12/code
#####LISTA DE NOMES COMEÇADOS POR -N-#####
- N. B. Goldstein
- N. B. Chakraborti
- N. B. Rabbat
- N. Balakrishnan
- N. Bairagi
- N. B. Wong
- N. Barrie Jones
- N. Baron
- N. Basak Ceylan
- N. Billawala
- N. Bertaggia
- N. Benalie
- N. Batut
- N. Bensalah
- N. Bigdeli
- N. Botta
- N. Bonnet
- N. Béchu
- N. Burgess
- N. Capdevielle
- N. C. Mohanty
- N. C. Menicucci
- N. C. Majee
- N. C. Sahoo

Página: 3/164
Página anterior: 'p'      Página seguinte: 'n'      Sair: 's'
>
```

Figura 9: Neste exemplo da *querie 12* podemos observar todas as ferramentas e alternativas de consulta que fornecemos ao utilizador

4 Testes de Performance

4.1 Código utilizado nos testes

```
#include <sys/time.h>
#include <time.h>
.
.
.

struct timeval inicio;
struct timeval fim;

gettimeofday(&inicio,NULL);
.
.
//Chamada da função
.
.
gettimeofday(&fim,NULL);

printf("inicio: %ld microseconds\n", (long int)inicio.tv_usec);
printf("fim: %ld microseconds\n\n", (long int)fim.tv_usec);

long int tempo_gasto = (long int)fim.tv_usec - (long int)inicio.tv_usec;
printf ("elapsed CPU time:      %ld microseconds\n", tempo_gasto);
```

4.2 Resultados

	Carregar ficheiros em memória	Querie 8.	Querie 9.	Querie 12.
Ficheiros de teste	Tempos			
<i>publicx.txt</i>	3,78 s	3194 microseg	13857 microseg	4207 microseg
<i>publicx_x4.txt</i>	9,73 s	3699 microseg	14548 microseg	4563 microseg
<i>publicx_x6.txt</i>	15,56 s	4175 microseg	15401 microseg	4595 microseg

Figura 10: Tabela de tempos de execução

Algumas informações relevantes sobre a máquina que correu os testes ...

Máquina virtual Ubuntu 12.04 LTS Intel Core i7 3630QM 2.4GHz

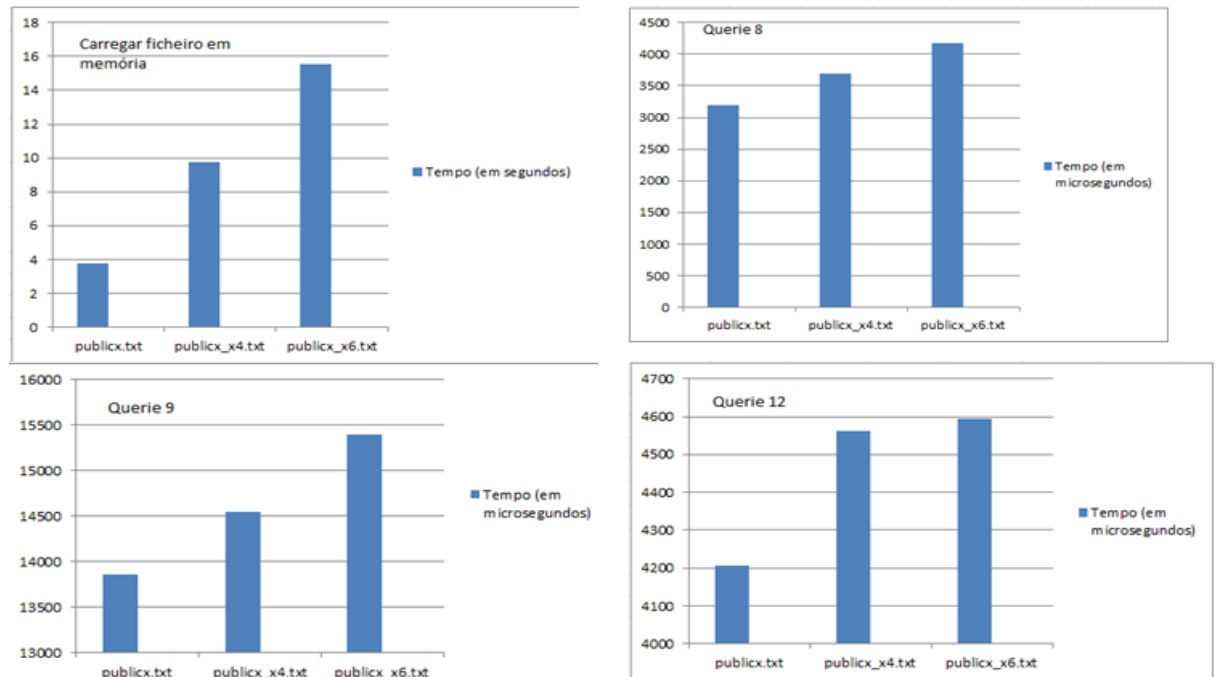


Figura 11: Gráficos sobre os resultados obtidos. Note-se que no gráfico da querie 12 o desfazimento dos tempos têm a escala de intervalos mais pequena. Para os três ficheiros os inputs para os testes foram: **Querie8** - Donald F. Towsley; **Querie9**: [2000,2010] ; **Querie12**: ano - 2012 N - 500;

```

Por favor consulte NAutores.txt para visualizar a lista de 500 autores que mais artigos publicaram em 2012.
inicio: 206020 microseconds
fim: 210227 microseconds
elapsed CPU time: 4207 microseconds

```

Figura 12: Exemplo de medição de tempo da querie 12.

5 Makefile

5.1 Opções de compilação

`-ansi -Wall -Wextra -O3 -Wno-unused-result -g`

Notas sobre as opções de compilação:

-ansi : diz ao compilador para implementar a opção ANSI.

-Wall : diz ao compilador para implementar todas as opções de aviso (warning);

-Wextra : ativa alguns avisos extra.

-O3 : o nível de optimização do código.

-Wno-unused-result : porque versões do gcc como a 4.6.3 dão warnings quando se ignoram valores de retorno da função `scanf` ou até mesmo com a função `fgets` ou com chamadas ao sistema como `system("clear")`, que foi uma ferramenta usada no sentido de formatar o output da informação no terminal.

-g : Muito útil para fazer debugging em qualquer API principalmente quando estas são de grandes dimensões. Esta opção faz com que o gdb diga ao utilizador qual a linha exacta do código num ficheiro `.c` em que ocorreu o erro.

5.2 Grafo de dependências da makefile

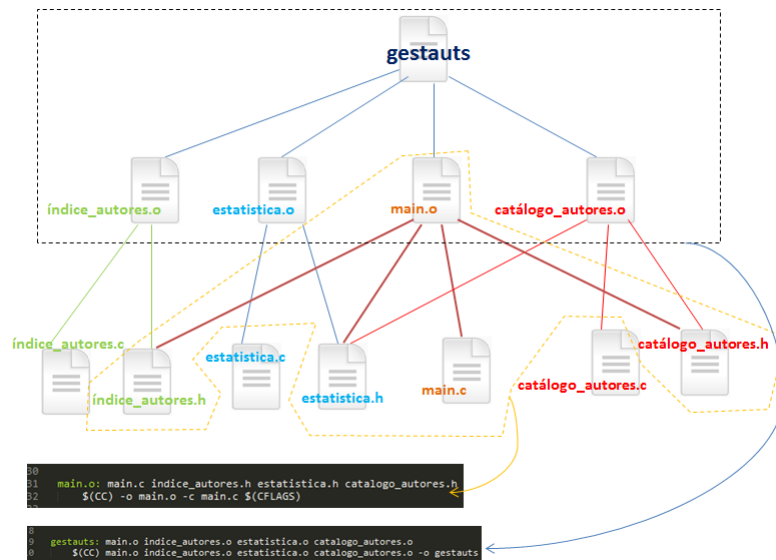


Figura 13: Podemos observar representações esquemáticas das linhas de código da makefile no grafo.

6 Conclusão

O grupo conseguiu desenvolver uma solução aceitável para o problema proposto, construindo uma aplicação extremamente interactiva e que respeita os princípios de modularidade e encapsulamento postos como desafio inicial. Não sendo a linguagem de programação C possuidora natural de tais propriedades de modularidade e encapsulamento estas tiveram forçosamente de ser implementadas através de uma codificação mais trabalhosa.

O grupo de trabalho ficou satisfeito com os resultados finais obtidos, pois as escolhas feitas levaram a tempos de execução satisfatórios a nível de manuseamento da aplicação por outro lado temos também a percepção de que o tempo que os ficheiros demoram a ser carregados pelo programa para a memória não é tão satisfatório mas este foi um custo necessário para que a aplicação tivesse tempos de resposta rápidos durante a consulta dos resultados (e esse é o verdadeiro objectivo do problema).

O Gestauts foi sobretudo um passo importante na absorção de conceitos para a **boa prática da Engenharia de Software**, conceitos como **encapsulamento** e **modularidade**, tendo o grupo concluído que usando uma linguagem para objectos como o JAVA estas propriedades são “grátis” pelo que não temos de nos preocupar com os detalhes de implementação de tão baixo nível como implementar árvores AVL ou tabelas de hash, temos sim de nos preocupar em **saber quando, como** e o **porquê** de essa estrutura de dados pela qual optamos ser uma solução mais eficiente para o problema proposto.