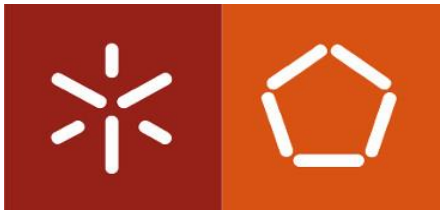
Relatório Guião 3

Grupo 37 | Laboratórios de Informática III

Hugo dos Santos Martins
a95125

João Bernardo Teixeira Escudeiro
a96075

José Pedro Batista Fonte
a91775



Departamento de Informática
Licenciatura em Engenharia Informática
2º ano | 1º Semestre
Laboratórios de Informática III

ÍNDICE

1. Introdução

O presente relatório visa descrever e explicar em detalhe o trabalho desenvolvido pelo grupo 37 no Guião 3 âmbito da cadeira de Laboratórios de Informática III, lecionada no curso de Licenciatura em Engenharia Informática na Universidade do Minho, no 1º semestre do ano letivo 2021/2022.

O guião 3 apresenta-se como a continuação do guião 1 e guião 2, já entregues pelo grupo. Os principais objetivos do guião 3 são: criar um menu interativo que torne a utilização do programa mais intuitiva e visualmente mais apelativa para um utilizador padrão, e também visar os testes de desempenho (tempo/qualidade) que o programa funciona.

O objetivo do guião 3 é, perante a ausência de um ficheiro de comandos passado como argumento, processar os ficheiros de dados padrão e apresentar ao utilizador um menu onde liste as diferentes funcionalidades que o programa disponibiliza e solicitar uma opção ao utilizador, permitindo assim uma fácil navegação pelos resultados, de uma forma interativa.

Resumidamente o guião 3 funciona do seguinte modo, após a escolha por parte do utilizador, o programa retorna, após recorrer às funções do guião 1 e 2, os resultados da opção escolhida num formato visualmente apelativo, facilitando a consulta e navegação pelos resultados obtidos.

2.Organização dos Ficheiros

Após analisar os principais objetivos desta U.C, o grupo verificou que as propriedades de modularidade e encapsulamento do código, já pedidas em guiões anteriores, não tinham sido postas em prática, como já foi explicado, pelo grupo, na apresentação do guião 2. Assim, foi decidido que seria uma das principais preocupações para este guião. Assim sendo, começamos por dividir o código em vários ficheiros consoante os argumentos e funcionalidades das funções, pondo em prática as técnicas mencionadas em cima. Na pasta ***src/*** encontram-se os ficheiros novos criados nesta etapa: ***commits.c, commits.h, data.c, data.h, generic.c, generic.h, g2.c, g3.c , g3.h , lang.c, lang.h, queries.c, queries.h, repos.c, repos.h, users.c, users.testes.c***.

- **g2.c** – contém o código principal constituído pela função main(), *readline*() e algumas funções auxiliares
- **g3.c** – contém o código auxiliar na realização do menu.
- **testes.c** – contém o código auxiliar à realização dos testes de desmpenho.
- **commits.c** – contém o código com todas as funções que utilizem COMMITS e COMMITS_AUX e que não gere dependências.
- **data.c** – contém o código com todas as funções que utilizem DATA e que não gere dependências.
- **generic.c** – contém o código com todas as funções que utilizem as diferentes *structs* e que tenham dependências entre si, assim como, todas as funções que utilizem AUX10 e IDS_ARR.
- ***lang*.c** – contém o código com todas as funções que utilizem LANG_ARR e que não gere dependências.
- ***queries*.c** – contém o código com todas as funções que utilizem STATS e que não gere dependências.
- ***repos*.c** – contém o código com todas as funções que utilizem REPOS e que não gere dependências.
- ***users*.c** – contém o código com todas as funções que utilizem USERS e que não gere dependências.

3.Explicação do Código

3.1 Estruturas de Dados

O grupo decidiu reutilizar as seguintes *structs* do guião 2 com o objetivo de responder a cada uma das *queries* da forma mais eficiente possível.

Struct stats

Para as *queries* estatísticas, o grupo decidiu criar uma estrutura que contemplasse todos os campos que as estas necessitavam para a sua execução. Estes campos são contabilizados à medida que o *parsing* dos dados é efetuado. Todos os campos são do tipo *int*.

```
struct stats
{
    int user;
    int bot;
    int organization;
    int repos ;
    int colaboradores ;
    int bots_colab;
    int utilizadores;
    int commits;
};
```

***Figura 1**-Struct stats*

Struct data

Esta *struct* serve de auxílio à realização das *queries* que tratam datas. Possui os campos ano, mês e dia, que são todos do tipo inteiro.

```
struct data
{
    int ano;
    int mes;
    int dia;

};
```

***Figura 2**- Struct data*

Struct aux10

Esta *struct* serve de auxílio à realização das *queries* que tratam datas. Possui os campos ano, mês e dia, que são todos do tipo inteiro.

```
struct aux10
{
    KEY author_id;
    int size;

};
```

***Figura 3**- Struct aux10*

Struct lang_arr

Esta *struct* que guarda a *language*, o *number*, que é o número de ocorrências da linguagem, a *pos* é a posição no array, o *size* é o tamanho do array ocupado.

```
struct lang_arr
{
    char* language;
    int number;
    int pos;
    int size;

};
```

***Figura 4** - Struct lang_arr*

Struct ids_arr

Esta *struct* guarda o id de um utilizador, o *number* que é o número de commits daquele ID, a *pos* é a posição no array, o *size* é o tamanho do array ocupado.

```
struct ids_arr
{
    KEY id;
    int number;
    int pos;
    int size;

};
```

***Figura 5**- Struct ids_arr*

KEY: Todas as principais estruturas possuem uma *key* que diz respeito ao id do respetivo parâmetro (ou id do user ou do repositório).

Struct users

Para os users, separamos os seguintes campos, que nos são uteis à realização de cada uma das *queries*. A *Key* da tabela de *hash* que é uma string contempla o id de cada user. Os campos: login, *type*, *following_list*, *repos* são tratados como uma string. Os outros parâmetros *followers*, *following*, são inteiros enquanto campo *state* é um *char* e indica se a posição na tabela de *hash* está ou não ocupada.

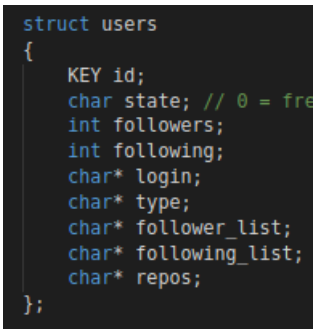


Figura 6 *- Struct users*

Struct repos

Para os repos, separou-se os seguintes campos, que são uteis à realização de cada uma das *queries*. Assim sendo, também temos a KEY que é o *id* de repositório, um *state*, que indica se a posição está ou não ocupada. O *owner_id* do repositório, a linguagem do mesmo que é uma string, um DATA *updated_at* que indica a data em que o repositório foi atualizado, e a *description* que também é uma string.

Figura 7 *- Struct repos*

Struct commits

Para os commits, separou-se os seguintes campos, que são úteis à realização de cada uma das *queries*. Assim sendo, também temos a KEY que é o *id* de repositório, um *state*, que indica se a posição está ou não ocupada. Os três inteiros (*size*, *p*, *bot*), indicam, respetivamente, o tamanho alocado do array de *strings* (commits), tamanho utilizado desse array e o campo *bot* que nos diz se o repositório tem ou não *bots* como colaboradores.

Figura 8 *- Struct commits*

Struct commits_aux

Esta *struct* apenas serve no auxílio da query2, para verificar quantos colaboradores possuímos

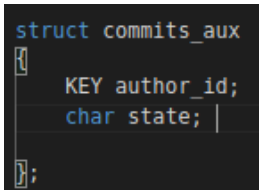


Figura 9 *- Struct commits_aux*

3.2. Encapsulamento e Modularidade :

Como um dos maiores objetivos da UC de LI III, o encapsulamento dos dados e a modularidade foram objetos de trabalho pelo grupo. Como consequência do grupo não ter utilizado estes princípios na realização do Guião Anterior, uma das tarefas que ocupou maior parte do tempo foi a reformulação do código de forma a obter o código final, de acordo com estes princípios. Assim, e de forma a por em prática a modularidade, separou-se o código em diferentes módulos (*.h e *.c) cada um com um nome sugestivo ao seu conteúdo.

De igual modo, e com o objetivo de ver concluído o princípio do encapsulamento, o grupo decidiu criar as estruturas no respetivo ficheiro (*.h) e dizer o seu conteúdo apenas no ficheiro (*.c) correspondente, para que o conteúdo de cada uma das estruturas apenas seja acedido naquele ficheiro, e consequentemente ficar com a estrutura "encapsulada".

Um outro ponto a ter em conta é o acesso a dados da estrutura que apenas é feito no ficheiro em que esta se encontra declarada, e caso se pretenda obter um dado da estrutura num outro ficheiro para (por exemplo) a realização das *queries*, então nas respetivas funções, são retornadas cópias para que os dados da estrutura nunca sejam diretamente acedidos pelo utilizador num ficheiro que não seja aquele que a estrutura está declarada.

3.3 Funções Principais

Função main ()

A função main() é responsável por abrir os ficheiros presentes na pasta entrada/ e criar as Hash Tables dos dados dos users.csv, repos.csv e commits.csv seguindo os procedimentos indicados em cima. De seguida, invoca a função *readfile*() que vai ler as linhas do commands.txt e vai direcionar para a função das *queries* em questão.

Função readfile ()

Para o leitor de linha o grupo usou a função *readfile*(), que recebe todos as *structs* inicializadas na main, bem como o *filepath* para o ficheiro de entrada. Assim e mediante o ficheiro de comandos, esta função separa cada uma das linhas mediante os "*tokens*" da instrução e direciona para cada uma das *queries* com os respetivos parâmetros que são necessários para a sua execução.

Função menu ()

A função menu, baseia-se na implementação de um menu interativo recorrendo à biblioteca <NCurses> em que, mediante inputs, se realiza a *query* correspondente, bem como apresenta, de uma forma apelativa os resultados da mesma. Assim o utilizador consegue de uma forma iterativa, realizar a *query* que pretenda, podendo navegar pelos resultados da mesma. Esta função recorre a funções auxiliares que mediante o input, realizam a *query* correspondente, para que os resultados apresentados sejam corretos.

3.4 Gestão de Dados

Em relação à gestão de dados, o grupo optou por não incorporar a utilização de ficheiros adicionais para gestão de memória, pois os tempos das *queries* são aceitáveis, quando temos em conta os novos ficheiros, e são abaixo do tempo útil (5 segundos).

3.5 Otimizações

Em relação ao guião 2, o grupo fez algumas alterações , principalmente nas *queries* 5 e 6 , pois a maneira abordada no anterior guião ,não era , de todo a mais eficiente, tanto que a média calculada no guião anterior , e para os ficheiros anteriores ,era cerca de 17 segundos , o que é , de todo impensável nesta altura do projeto.

Assim, o grupo optou por utilizar tabelas de *hash* também nestas *queries* com o objetivo de diminuir o tempo de acesso ao array, o que diminuiu significativamente.

3 Testes de desempenho

Em relação às *queries*, estas foram testadas com os ficheiros (previamente limpos) do guião 3. Quanto à *query*9, face a erros que apareceram, não conseguimos testar com os ficheiros do guião 3, sendo que o fizemos com ficheiros do guião anterior. Assim esta *query* terá uma média de tempo ligeiramente inferior, comparativamente ao tempo que poderia demorar com os novos ficheiros.

- Quanto aos testes, os parâmetros que foram utilizados foram:

Nas primeiras quatro, não foi necessário qualquer parâmetro. Os parâmetros utilizados para as restantes foram:

Q5 : Número : "100"; Data_1 ”2010-01-01";Data_2 : “2015-01-01";

Q6 : Número: "5"; Linguagem: "Python";

Q7 : Data_1 "2014-04-25";

Q8 : Número : "100"; Data_1 ”2010-01-01";Data_2 : “2015-01-01";

Q9 : Número "4";

Q10 : Número :"50";

O grupo correu as *queries* várias vezes fazendo comparações entre os ficheiros de saída, e para os mesmos parâmetros estas produzem os mesmos outputs.

Para os testes de desempenho foi utilizado o computador portátil Acer Predator Triton 500, com as seguintes especificações:

- CPU: Intel i7 9750H @ 2.60 GHz/core (6 cores/12 Threads)
- GPU: NVIDIA GeForce RTX 2060

…

	Teste1	Teste2	Teste3	Teste4	Teste5	Teste6	Teste7	Teste8	Teste9	Teste10	Média
Query1	0.000091	0.000121	0.000100	0.000105	0.000123	0.000106	0.000115	0.000141	0.000153	0.000104	0.000114
Query2	0.000081	0.000117	0.000156	0.000188	0.000166	0.000119	0.000129	0.000174	0.000105	0.000180	0.000143
Query3	0.000108	0.000145	0.000125	0.000106	0.000105	0.000114	0.000111	0.000137	0.000129	0.000119	0.000118
Query4	0.000105	0.000154	0.000101	0.000112	0.000122	0.000105	0.000086	0.000149	0.000122	0.000154	0.000121
Query5	1.777084	1.744599	1.812798	1.749208	1.744063	1.745243	1.760413	1.810010	1.739778	1.757347	1.761093
Query6	0.278678	0.279869	0.277669	0.283022	0.279016	0.279731	0.284141	0.285112	0.278533	0.284782	0.280971
Query7	0.388574	0.399131	0.386153	0.390746	0.381232	0.383873	0.386374	0.385056	0.391003	0.400954	0.389091
Query8	1.417967	1.406349	1.386178	1.395531	1.357367	1.404943	1.413982	1.391091	1.386885	1.390001	1.393268
Query9	1.170537	1.204553	1.241858	1.184529	1.173245	1.174456	1.257434	1.202093	1.210686	1.257531	1.206106
Query10	2.676631	2.708849	2.689340	2.664744	2.729413	2.746261	2.720796	2.688815	2.685101	2.785695	2.705650

Tabela1.Os resultados apresentados na tabela ,são fruto da execução das *queries* com os parâmetros acima mencionados, descartando o maior e menor valor;

Foram realizados 10 testes para cada *query* para garantir uma maior fiabilidade dos resultados

5. Conclusão

No geral o grupo autoavalia-se positivamente visto que demos por concluído este guião. Encapsular e modularizar o código foi uma das tarefas mais difíceis dada a inexistência de conhecimento prévio, no entanto o objetivo foi alcançado. O método usado para contornar as dependências geradas por algumas funções após este procedimento, foi criar um ficheiro genérico. Assim todas as funções que tivessem argumentos de estruturas guardadas noutros ficheiros, foram feitas neste novo *generic.c*, assim como as poucas funções que usam IDS.ARR e AUX10.

O grupo também teve dificuldade em utilizar a biblioteca <NCurses> para a construção do menu, dada a inexistência de conhecimento e experiência nessa área.

Assim, e olhando para o trabalho desenvolvido durante o semestre, o grupo considera que apesar de ter desenvolvido todas as *queries* no guião 2 de uma forma

correta, poderia ter obtido uma melhor classificação à UC, se tivesse posto em prática os princípios de modularidade e encapsulamento no guião anterior.

Em geral o grupo acha que a tarefa 3 foi concluída com sucesso. Um aspeto passível de ser melhorado é a *query9* que no guião anterior funcionava na perfeição, com ficheiros ligeiramente menores, mas neste guião deixou de funcionar, possivelmente por erros de tamanho que o grupo não conseguiu encontrar.

Em relação aos resultados produzidos por cada *query* , o grupo acredita que os tempos de execução são bastante aceitáveis ,pelo que ,no geral o programa se encontra bastante otimizado .