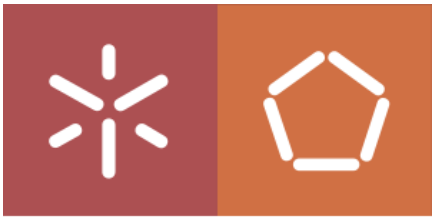

Universidade do Minho
Escola de Engenharia

Universidade do Minho

Laboratórios de Informática III

Licenciatura em Engenharia Informática

Guião 1

Relatório

João Ferreira (A89487)

Marisa Soares (A92926)

Luís Araújo (A96351)

10 de novembro de 2021

Exercício 1

Primeiramente, começamos por escrever a função **parseUsers()** utilizando algumas funções que nos foram sugeridas pelo professor na 1ª aula pratica, nomeadamente a **fopen()**, **fprintf()** e **fclose()**.

O propósito desta função é:

1. Abrir o ficheiro ".csv" e ler cada linha, com o auxílio do **strsep()**.
2. Fazer o parsing de cada linha do ficheiro, ou seja, verificar se elas são válidas. Para este caso criámos uma outra função – **linhaValida()**
3. Escrever linha a linha no novo ficheiro, *".-ok.csv"*, apenas se estas forem válidas.
4. Fechar ambos os ficheiros.

Devemos acrescentar que esta função será implementada da mesma forma para os restantes 2 ficheiros, os "commits.csv" e os "repos.csv", que se chamam respetivamente **parseCommits()** e **parseRepos()**. A única diferença nestas funções é no ficheiro que lê e onde o ficheiro escreve.

Para verificar se as linhas são validas (passo 2), usamos a função **linhaValida()**. Através do uso do **strsep()**, criamos um ciclo para incrementar uma variável sempre que o **strsep()** é chamado, de modo a que verifiquemos cada coluna do ficheiro .csv separadamente (por exemplo: quando a variável *i* é igual a 0, estamos perante o primeiro campo). Sendo assim podemos facilmente verificar se cada campo é válido com a utilização das funções do ficheiro "parsing.c" e um *switch-case*.

É de notar as seguintes funções:

I. Usamos a função **isInt(***)* quando queremos verificar se a coluna selecionada é um número inteiro não negativo;

II. A **verificarData()** verifica se o formato da data está no pretendido e se a data e hora são válidas;

III. A **validaLista()** averigua se o campo que estamos a verificar é uma lista de números inteiros não negativos;

IV. Em relação ao ficheiro "users.csv" era necessário que o número de followers/following fosse igual ao número de elementos das respetivas listas, sendo assim usamos a função **getFollow()** em conjunto com a **validaLista()**, ou seja, o campo do *followers/following* só será válido se o número de elementos da lista for igual ao *int* devolvido pela **getFollow().**

Usamos assim estas funções para verificar os respetivos campos das respetivas linhas de cada ficheiro.

Tivemos de escrever 3 funções **linhaValida()** diferentes, uma vez que os ficheiros cumprem requisitos diferentes para diferentes colunas.

Numa nota final queremos acrescentar que nos campos que deve ser válida uma string, optamos por não utilizar nenhuma função, nem utilizar nenhum *case* dentro dos *switch*, uma vez que uma string é sempre válida.

Desta forma acabamos por fazer o exercício 1, cumprindo todas as etapas.

Exercício 2

Para a resolução do exercício 2, compreendemos que seria necessário o armazenamento de *Users* e de *Repos*. Para esse efeito utilizamos *Hash Tables*, da libraria '*glib.h'*.

Criamos assim as *struct user* e *struct repos*, contidas nos ficheiros *user.h* e *repos.h*, respetivamente. E posteriormente as listas de Users e Repos, nos ficheiros *listaUsers.h* e *listaRepos.h*.

```
typedef struct user{
    int id;
    char* username;
} *USER;
```

```
typedef struct repos{
    int id;
    int owner_id;
} *REPOS;
```

```
typedef struct lUsers{
    GHashTable *user;
} *LISTAUSERS;
```

```
typedef struct lRepos{
    GHashTable *repos;
}*LISTAREPOS;
```

Depois, nos respetivos ficheiros de parse de cada ficheiro, lemos o ficheiro "-ok.csv" anteriormente criado com as linhas corretas e válidas. No caso do "users-ok.csv" apenas adicionamos os utilizadores à lista dos users se não existirem, ou seja, no ficheiro "users-final.csv" não existem utilizadores duplicados.

Utilizamos as funções **adicionarUsers()** para ler o ficheiro e adicionar as linhas ao ficheiro final e **adicionarU()** para criar um User adicioná-lo à lista (caso não exista nesta) – a key utilizada nesta *Hash Table* é o *id* do utilizador (campo 1). Para isso usamos um ciclo for e a **strsep()** novamente para armazenar os primeiros dois campos em variáveis, para que possamos verificar se o *id* já existe na lista de utilizadores.

Seguidamente, verificamos os repositórios, ou seja lemos o ficheiro "repos-ok.csv", e só adicionamos à lista de repositórios e a linha ao ficheiro "repos-final" os repositórios cujos id ainda não existam na lista de repositórios e *owner_id* existam na lista de utilizadores – a key utilizada nesta *Hash Table* será o *id* do repositório.

A função utilizada neste caso não difere muito da referida acima para os users: usamos o *id* do repositório e o *owner_id* para averiguar a condição acima.

Por fim, para os commits não utilizamos nenhuma lista, uma vez que não é necessário armazenar nada, apenas gerar o ficheiro *commits-final* cujos *committer_id e author_id* existam na lista de utilizadores e *repos_id* exista na lista de repositórios. Para isso utilizamos a função **adicionarC()**, e passamos pelos primeiros 3 campos, que guardamos em variáveis para que possamos confirmar as condições mencionadas anteriomente.

A nossa maior dificuldade na concretização do Guião 1 passou pela utilização da libraria do "glib.h", nomeadamente as *Hash Tables*, uma vez que ainda não estávamos familiarizados com esta estrutura de dados.

Outras dificuldades foram a implementação da *Makefile* e do uso da função **strsep()**.