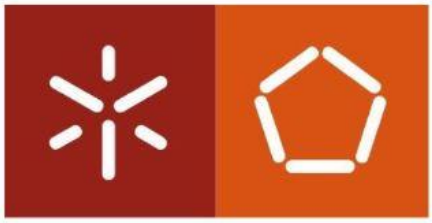


# UNIVERSIDADE DO MINHO

MIEI/ LEI+MEI

UC LI3- 2021/2022

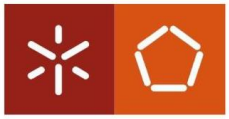
| Grupo 11




Luís Fernandes (A88539) David Duarte (A93253) João Castro (A97223)

# Relatório LI3 - 2021/22

## Guião 1. Grupo 11 - MIEI/LEI+MEI



Universidade do Minho

O guião 1 do projeto da UC Laboratórios de Informática III tinha como objetivos a consolidação de conhecimentos no que toca a operações de leitura e escrita sobre ficheiros, assim como a conversão da formatação de dados e gestões de memória.

O exercício 1 visava na realização de um programa que realizasse a leitura de 3 ficheiros em formato CSV e que validasse os campos que tivessem registos válidos, e consequentemente gerasse novos ficheiros com todos os campos corretos.

De modo a realizar esta primeira etapa definimos uma função 'ler_Ficheiro' que abria e lia cada um dos ficheiros através de um apontador e gerava os novos ficheiros corrigidos através do processo de encapsulamento. Em seguida, através da função pré-definida 'fgets' lemos cada uma das linhas dos nossos ficheiros e alocamos num array 'line', que seguidamente passará como argumento nas nossas funções 'build'.

Desta forma, enquanto que a leitura dos ficheiros não chegava ao seu fim, as funções build recebiam a linha e separavam as strings através da strsep que escrevia '\0' no final de cada uma das strings que estavam separadas pelo delimitador ";". Ainda na mesma função colocamos um campo de validação das componentes. Caso não fosse válida fazíamos free do apontador da estrutura de dados que estivesse em causa e retornava Null, e uma nova linha era lida. Caso não fosse, retornava o apontador da correspondente estrutura de dados.

De forma a validar o campo dos números não negativos, se o line não fosse um array vazio e se todos os seus elementos fossem dígitos(isDigit) então cabia à função atoi converter a string para o respetivo número inteiro e por conseguinte apenas nos restava validar se esse número era igual ou superior a 0. A verificação dos campos correspondentes às strings procedeu-se à averiguação se o array era vazio através da função strcmp, exeto o parâmetro descrição nos repos, pois este podia ser vazio, pois um repositório podia não ter descrição. Por outro lado, no que toca ao type enumerado, convertemos o array que armazenava esta componente para um tipo de dados "TIPO" e assim, em cada linha que recebíamos, verificávamos se o type era um Bot, Organization ou User, caso não fosse nenhum destes a componente seria inválida. A componente has_wiki convertemos o valor booleano (true/false) armazenado no char*valid para os valores 0 e 1 correspondentes.

Em relação à componente das listas, em primeiro lugar, alocamos memória para armazenar os números num array de inteiros(list). Em segundo, queríamos verificar se o array char não era vazio, assim como, se o primeiro elemento era um '[' e se a lista era não vazia. A não verificação de uma destes condições implicava retornar um array de inteiros null. Caso contrário, realizávamos o strsep do line de modo a retirar os parênteses retos e armazenávamos num array char auxiliar. De seguida, só nos restava percorrer esse array para saber se todos os elementos eram dígitos, se havia algum número negativo e se o contador que percorreu a lista tinha o mesmo valor que o valor inteiro de followers/following. Por fim, se a nossa lista fosse nula ou o tamanho não fosse superior a 0, então este componente estaria incorreta.

Quanto à resolução das datas, numa primeira fase, pretendíamos converter a string recebida para um tempo armazenando-o numa estrutura de dados tm dado pelo apontador time. De seguida

pretendíamos formatar o tempo e armazenámo-lo num novo array char new_time segundo a estrutura AAAA-MM-DD HH-MM-SS. Isto com o fim de comparar e verificar a construção da data.

Porventura, devido ao conflito que strdup de string.h fazia com o strdup do time.h, acabámos por construir um tipo de dados LTempo para as datas. Em pormenor, a primeira função irá testar a validação das datas segundo o processo referido anteriormente. Por outro lado, esta função averigua se o ano é ou não válido, de seguida, se consoante o mês dos anos limites o dia é valido(bissextitude do ano), e por fim, a validação dos dias consoante o mês. A parte respetiva ao tempo apenas verificámos se este se encontrava dentro dos limites habituais.

A seguir temos uma função da construção do tipo de dados LTempo na qual irá validar a data no array line e, deste modo, buil_time irá devolver o apontador do LTempo dessa data se estiver bem construído e Null caso contrário, sendo que nessa situação, este campo estará inválido.

No exercício 2 do guião 1 era pretendido que a partir dos ficheiros gerados do exercício 1, filtrássemos os dados incorretos e posteriormente, gerássemos novos ficheiros de saída.

Para a resolução deste exercício pensámos que seria mais conveniente a implementação de hash tables, uma vez que estas conseguem encontrar a key que procurámos em tempo constante O(1), independentemente do número de items que as nossas estruturas de dados users, commits ou repos pudessem vir a ter. Desta maneira, a nossa procura seria muito mais eficiente e levaria menos tempo.

De modo a executar esta tarefa, em primeiro lugar, construímos funções ler_files em que abríamos os ficheiros “ok” de saída do exercício 1 para leitura, e também abríamos os ficheiros “final” de saída do exercício 2 para escrita. Após a escrita do header, à medida que o nosso apontador lia o ficheiro, o nosso apontador LUser ia recebendo o user de cada linha e assim, através da função ‘g_hash_table_insert’ iamos criando uma hash table dos users com os vários valores de id. Desta modo, enquanto que o nosso ficheiro ‘commits_ok’ estava a ser lido, o commiter_id e o author_id de cada linha respetiva dos commits se encontrava estava a ser procurada na hash table dos users. Isto foi possível através da implementação da função pré-definida ‘GContainsKey’, e mais uma vez à técnica de encapsulamento que construímos em ordem para obter os commiter_id e author_id (get_Committer_ID, get_Author_ID). Por outro lado, a componente repo_id tem de estar presente na nossa hash table.

Assim, se uma linha do ficheiro commits estivesse bem construída, fazíamos o append(acrescento no final) de mais um elemento num GArray que armazenava os apontadores para estrutura de dados commits bem construídos (isto seria útil para mais tarde verificar a validação dos repos). No entanto, se a linha do commit estivesse errada fazíamos free do conteúdo, e essa mesma linha não seria escrita no ficheiro de saída.

No final, esta função ‘ler_file_commits_ok’ devolvia o GArray ‘commit_list’, como foi anteriormente referido, e verificava no ‘ler_file_repos_ok’ se cada uma das linhas do file possuía o owner_id correspondente na hash table users e se o mesmo componente estava ou não presente no GArray. Para isso, realizámos uma procura linear, em que verificávamos a cada commit do GArray se o seu item repo_id (por encapsulamento) era igual ao correspondente dessa linha. Assim, as linhas que não tivessem estes requisitos, seria feito o free do conteúdo delas. Caso contrário, era imprimida essa linha no ficheiro de saída.