Aula prática 1

Os exercícios 1 e 3 têm como objetivo aplicar conhecimentos de alocação dinâmica de memória. O exercício 2 tem como objetivo aplicar conhecimentos de representação de dados e de ficheiros binários.

Para cada exercício, consulte a respetiva pasta incluída em **P01.zip**, disponível no Moodle.

0 – Este exercício pode ser validado no CodeRunner do Moodle.

0.1 – Implemente um programa que (a) pede ao utilizador o número de elementos do vetor, (b) cria o vetor de forma dinâmica (dica: `calloc`), (c) pede ao utilizador números inteiros e preenche o vetor com os mesmos, e (d) no final, chama a função `imprime_vetor`, já implementada.

0.2 – Implemente uma função

```
int vetor_tem_impares(int *vetor, int n)
```

que retorna 0 caso o vetor não tenha números ímpares ou 1 caso o vetor tenha números ímpares.

1 – Os ficheiros `vetor.c/h` contêm uma biblioteca implementada em C para a manipulação de vetores de *strings*. Esta biblioteca é composta por diversas funções que podem ser usadas, por exemplo, para criar um novo vetor, apagar um vetor já existente, adicionar elementos, eliminar elementos, ou ordenar os elementos.

1.1 – Estude a implementação da biblioteca fornecida.

1.2 – Crie um pequeno programa de teste da biblioteca que deverá realizar as seguintes operações:

1. Criar um novo vetor vazio.
2. Solicitar ao utilizador 5 *strings* para inserir no vetor.
3. Imprimir o conteúdo do vetor.
4. Solicitar ao utilizador uma *string*; se essa *string* existir no vetor, apagar a *string*.
5. Imprimir o conteúdo do vetor.
6. Ordenar o vetor.
7. Imprimir o conteúdo do vetor.

1.3 – Adicione as seguintes funções à biblioteca:

```
vetor* vetor_concatena(vetor *vec1, vetor *vec2)
```

cria um novo vetor que resulta da concatenação de dois outros vetores

```
int vetor_inverte(vetor *vec)
```

inverte os elementos do vetor

```
vetor* vetor_baralha(vetor *vec)
```

cria um novo vetor com os mesmos elementos do vetor `vec`, mas guardados em posições aleatórias

1.4 – Altere o programa de teste implementado na alínea 1.2, de forma a verificar também estas funções.

2 – Os ficheiros áudio MP3 incorporam, para além da informação áudio, informação associada normalmente designada de metadata; esta informação pode incluir, entre outros, dados sobre o nome do artista, da faixa e do álbum, comentários, etc. Os dois formatos de metadata em MP3 são o ID3v1 e ID3v2.

Escreva um programa que leia a metadata de ficheiros MP3 no formato ID3v1. Quando essa informação está presente, os campos de informação encontram-se no fim do ficheiro. São utilizados no total 128 bytes, sendo inicializados por 3 bytes contendo o valor "TAG". A tabela em baixo resume os campos utilizados pelo formato ID3v1.

Campo	Comprimento (bytes)	Descrição
Cabeçalho	3	Valor de verificação "TAG"
Título	30	Título da música representado com carateres ASCII
Artista	30	Nome do artista representado com carateres ASCII
Álbum	30	Nome do álbum representado com carateres ASCII
Ano	4	Ano representado por 4 dígitos (carateres ASCII e não um inteiro)
Comentário	28 ou 30	Comentário associado à música
Byte-zero	1	Se o número da faixa estiver incluído então deverá ser 0. Nesse caso apenas são usados 28 bytes no comentário
Número	1	Número da faixa no álbum representado por um inteiro
Género	1	Género da música representado por um inteiro e definido de acordo com a tabela disponibilizada em: https://en.wikipedia.org/wiki/List_of_ID3v1_Genres (considere apenas os géneros 0 a 79)

Utilize o ficheiro *musica.mp3* para testar o programa que desenvolveu.

Exemplo usando o ficheiro *musica.mp3*

```
Titulo - Particule
Artista - Silence
Album - L'autre endroit
Ano - 2006
Número - 7
Género - Instrumental
```

3 – Pretende-se preencher um vetor de 3 *strings* lidas do teclado, cada uma com um máximo de 80 caracteres. Para não desperdiçar espaço em memória, cada *string* deve ser guardada no vetor, alocando dinamicamente apenas o número de caracteres necessários.

Estude a implementação proposta para este exercício (ficheiro **`ex_3.c`**), compile e corra o programa. Poderá utilizar a ferramenta `valgrind` para detetar erros de memória e "memory leaks", executando-a no terminal da seguinte forma:

```
valgrind --tool=memcheck --leak-check=yes <nome_do_programa>
```

Identifique os erros e corrija o código, de forma a que não seja indicado qualquer aviso pelo `valgrind`, como apresentado no exemplo em baixo.

Exemplo (sem erros e fugas de memória)

```
==24252== Memcheck, a memory error detector
==24252== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==24252== Using Valgrind-3.10.0.SVN and LibVEX; rerun with -h for copyright
info
==24252== Command: ./a.out
==24252==
[1] (vazio)
[2] (vazio)
[3] (vazio)
Posicao para nova string (1 a 3): 1
Nova String: Frank Lloyd Wright
[1] Frank Lloyd Wright
[2] (vazio)
[3] (vazio)
Posicao para nova string (1 a 3): 2
Nova String: Oscar Niemeyer
[1] Frank Lloyd Wright
[2] Oscar Niemeyer
[3] (vazio)
Posicao para nova string (1 a 3): 3
Nova String: Alvar Aalto
[1] Frank Lloyd Wright
[2] Oscar Niemeyer
[3] Alvar Aalto
Posicao para nova string (1 a 3): 0
==24252==
==24252== HEAP SUMMARY:
==24252==     in use at exit: 0 bytes in 0 blocks
==24252==   total heap usage: 4 allocs, 4 frees, 70 bytes allocated
==24252==
==24252== All heap blocks were freed -- no leaks are possible
==24252==
==24252== For counts of detected and suppressed errors, rerun with: -v
==24252== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```