Aula prática 2

Os exercícios 1 e 3 têm como objetivo aplicar conhecimentos de alocação dinâmica de memória apresentados na segunda aula teórica de Programação 2. Adicionalmente, nos exercícios 2 e 4 será feita uma aplicação prática destes conceitos, utilizando uma biblioteca de manipulação de vetores de *strings*.

1 – Escreva um programa que leia, para um vetor V, um número N (escolhido pelo utilizador) de elementos inteiros. Utilizando funções, construa e imprima dois novos vetores baseados no vetor V: um com os seus números pares e outro com os seus números ímpares (ver exemplo). Todos os vetores deverão ser criados dinamicamente.

Exemplo

```
Número de elementos? 5
Introduza o 1º elemento: 1
Introduza o 2º elemento: 0
Introduza o 3º elemento: 3
Introduza o 4º elemento: 8
Introduza o 5º elemento: 10

Vetor original: [ 1 0 3 8 10 ]
Números pares : [ 0 8 10 ]
Números ímpares: [ 1 3 ]
```

2 – O ficheiro vetor.zip contém 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, ordenar os elementos, etc.

2.1 – Estude a implementação da biblioteca fornecida.

2.2 – 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.

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

Estude a implementação proposta para este exercício (ficheiro ex3.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)
```

4 – Considere novamente a biblioteca de vetores do exercício 2.

4.1 – 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

4.2 – Altere o programa de teste implementado no exercício 2, de forma a verificar também estas funções.