Trabalho prático 2 – Filas de prioridade, baseadas em Heaps

1) Informação geral

O trabalho prático 2 consiste na implementação de uma pequena biblioteca de funções para manipulação de **heaps** em C.

Este trabalho deverá ser feito de forma autónoma por cada grupo na aula prática 8 e completado fora das aulas até à data limite estabelecida. A consulta de informação nas diversas fontes disponíveis é aceitável. No entanto, o código submetido deverá ser apenas da autoria dos elementos do grupo e quaisquer cópias detetadas serão devidamente penalizadas. A incapacidade de explicar o código submetido por parte de algum elemento do grupo implicará também numa penalização.

O prazo de submissão na página de Programação 2 do Moodle é 16 de Abril às 21:00.

2) Implementação do trabalho

O ficheiro *zip* PROG2_1718_T2 contém os ficheiros necessários para a realização deste trabalho, nomeadamente:

- `heap.h` inclui as declarações das funções a implementar - **não deve ser alterado**
- `heap.c` ficheiro onde deverão ser implementadas as funções da biblioteca
- `heap-teste.c` inclui os testes feitos à biblioteca - **não deve ser alterado**

A estrutura de dados `heap` é a base da biblioteca e tem a seguinte declaração:

```
typedef struct
{
    int tamanho;
    int capacidade;
    elemento **elementos;
} heap;
```

Nesta estrutura é guardado um apontador para o primeiro elemento da lista de elementos, assim como a capacidade e o tamanho da lista. Cada elemento desta lista é uma estrutura `elemento` que contém 2 campos: 1) inteiro com a prioridade, 2) *string* com o valor. A estrutura `elemento` é declarada da seguinte forma:

```
typedef struct
{
    int prioridade;
    char *valor;
} elemento;
```

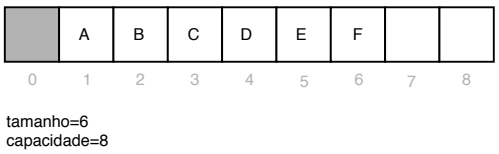
O trabalho consiste em implementar um conjunto de funções abaixo identificadas que permitem a realização de diversas operações sobre *heaps*. Estas funções estão declaradas no ficheiro `heap.h` e deverão ser implementadas no ficheiro `heap.c`.

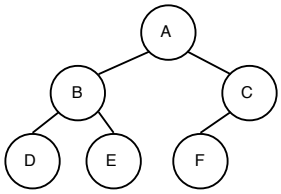
As funções a implementar e que estão associadas à estrutura de dados `heap` são:

1. **heap* heap_nova** (int tamanho_maximo);
   cria uma heap nova com o tamanho máximo indicado
2. **int heap_insere** (heap* h, const char* texto, int prioridade);
   adiciona um elemento à heap, com uma dada prioridade

3. **void heap_apaga** (heap* h);
   elimina uma heap, libertando toda a memória ocupada
4. **char* heap_remove** (heap* h);
   remove elemento da raíz da heap
5. **heap* heap_constroi** (elemento* v, int n_elementos);
   cria uma nova heap com os elementos do vetor v
6. **int heap_altera_prioridade** (heap* h, int indice_elemento, int nova_prioridade);
   altera a prioridade de um elemento da heap

A heap deve ser implementada com base num vetor (*array*) com a raiz guardada na posição 1. Na figura ao lado é mostrado um exemplo de como uma heap deve ser guardada. O tipo de heap a implementar é **min-heap**, ou seja, o elemento removido da heap deverá ser sempre o que tem menor valor de prioridade.





No ficheiro `heap.c` encontra-se já implementada a função `void mostraHeap(heap* h, int indice)` que imprime os elementos da *heap* e respetiva prioridade.

3) Descrição das funções a implementar

heap* heap_nova (int tamanho_maximo);

cria uma *heap* novo com o tamanho máximo inidicado

Parâmetros:

tamanho_maximo	tamanho máximo para a *heap*

Retorna:

Apontador para a *heap* criada ou NULL se ocorrer um erro.

Observações:

A *heap* é criada sem elementos.

int heap_insere (heap *h, const char* texto, int prioridade);

adiciona um novo elemento à *heap*

Parâmetros:

h	apontador para a *heap*
texto	string a guardar no elemento
prioridade	prioridade do elemento

Retorna:

1 se adicionou corretamente o novo elemento, 0 se a *heap* está cheia ou se não consegue criar o elemento.

Observações:

A inserção de um elemento deve manter a consistência da *heap*. Exemplo de erro é o apontador para a *heap* ser NULL.

void heap_apaga (heap* h);

elimina uma *heap*, libertando toda a memória ocupada

Parâmetros:

h	apontador para a *heap* a apagar

Observações:

Não esquecer de libertar toda a memória alocada para evitar *memory leaks*.

char* heap_remove (heap *h);

remove o elemento da raiz da *heap*

Parâmetros:

h	apontador para a *heap*

Retorna:

A *string* do elemento de menor prioridade, NULL se a *heap* estiver vazia ou em caso de erro.

Observações:

Para manter a consistência da *heap*, a remoção da raiz implica a reordenação dos restantes elementos de acordo com a sua prioridade.

heap* heap_constroi (elemento* v, int n_elementos);

cria uma nova heap com os elementos do vetor v

Parâmetros:

v	vetor de elementos
n_elementos	número de elementos do vetor v

Retorna:

Apontador para a *heap* criada ou NULL se ocorrer um erro.

int heap_altera_prioridade (heap *h, int indice_elemento, int nova_prioridade);

altera a pripridade de um elemento da *heap*

Parâmetros:

h	apontador para a *heap*
indice_elemento	identificador do elemento
nova_prioridade	nova prioridade atribuída ao elemento da *heap*

Retorna:

1 se a prioridade for alterada com sucesso, -1 em caso de erro.

4) Teste da biblioteca de funções

É fornecido um ficheiro `heap-teste.c` que permite realizar um conjunto de testes à biblioteca desenvolvida. Note que os testes não são exaustivos, não verificando, por exemplo, *memory leaks* e por isso os testes devem ser considerados apenas como um indicador de uma aparente correta implementação das funcionalidades esperadas.

Inicialmente o programa `heap-teste` quando executado apresentará o seguinte resultado:

```
** Testes com filmes
ERRO: Erro ao criar nova heap.
** Mais testes com filmes
ERRO: Erro ao criar segunda heap.
** Teste com livros
ERRO: Erro ao construir heap.
```

Note que é fortemente aconselhável que as funções `heap_nova` e `heap_insere` sejam implementadas antes de todas as outras. É também aconselhável que as funções sejam implementadas pela ordem indicada.

Depois de todas as funções corretamente implementadas o resultado do programa apresentará o seguinte resultado:

```
** Testes com filmes
Numero de elementos: 7
Primeiro valor retirado da heap: Logan (OK)
Segundo valor retirado da heap: Guardians of the Galaxy Vol. 2 (OK)
Numero de elementos: 5 (OK)
OK: Heap vazia.
** Mais testes com filmes
Nova heap tem 5 elementos.
Numero de elementos: 5 (capacidade: 5)
** Teste com livros
Tamanho de heap=8 (OK) e capacidade=8 (OK).
Valor na raiz do heap OK.
Alterar prioridade de "Milk and Honey" para 9.
OK: Elemento alterado esta' na posicao certa.
Alterar prioridade de "The Handmaid's Tale" para 1.
OK: Elemento alterado esta' na posicao certa.
```

5) Ferramenta de desenvolvimento

A utilização de um IDE ou do Visual Studio Code é aconselhável no desenvolvimento deste trabalho uma vez que permite fazer depuração de uma forma mais eficaz. Poderá encontrar informações sobre a utilização do Visual Studio Code num breve tutorial disponibilizado no Moodle.

6) Avaliação

A classificação do trabalho é dada pela avaliação feita à implementação submetida pelos estudantes, mas também pelo desempenho dos estudantes na aula dedicada a este trabalho. A classificação final do trabalho (T2) é dada por:

T2 = 0.7 Implementação + 0.1 Memória + 0.2 Desempenho

A classificação da implementação é essencialmente determinada por testes automáticos adicionais. No caso da implementação submetida não compilar, esta componente será de 0%.

A gestão de memória também será avaliada, sendo considerados 3 patamares: 100% nenhum *memory leak*, 50% alguns *memory leaks* mas pouco significativos, 0% muitos *memory leaks.*

O desempenho será avaliado durante a aula e está dependente da entrega do formulário "Preparação do trabalho" que se encontra disponível no Moodle. A classificação de desempenho poderá ser diferente para cada elemento do grupo.

7) Submissão da resolução

A submissão é apenas possível através do Moodle e até à data indicada no início do documento. Deverá ser submetido um ficheiro *zip* contendo:

- o ficheiro **heap.c** com as funções implementadas
- um ficheiro **autores.txt** indicando o nome e número dos elementos do grupo

Nota importante: apenas as submissões com o seguinte nome serão aceites: `T2_G<numero_do_grupo>.zip`. Por exemplo, `T2_G999.zip`