Prova com consulta (Moodle). Duração: 70min. Parte prática [15/20 valores]

No final deverá confirmar que submeteu corretamente no SIGEX o código fonte dos seus programas utilizando o nome indicado no enunciado.
Quaisquer cópias detetadas serão penalizadas com anulação da prova.

PARTE 1

Tendo por base a biblioteca de pilhas estudadas nas aulas práticas, e disponíveis nos ficheiros `pilha.h/.c`, implemente as funcionalidades pedidas nas duas alíneas seguintes no ficheiro **parte1.c** fornecido. Sempre que conveniente utilize as funções disponíveis.

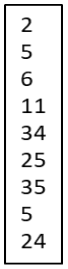
P1.1 [4 valores]

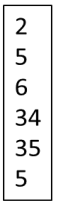
Pretende-se criar um programa para analisar os cromos de uma coleção. A lista de cromos está guardada num vetor de inteiros alocado dinamicamente. Os cromos são representados por números. Cada posição do vetor tem o número do cromo e estes não estão ordenados.

Imagine que tem uma coleção armazenada numa pilha e pretende retirar da pilha os cromos de um dado intervalo de números. Implemente a função

```
void retirar_cromos_intervalo(pilha *cromos, int inicio, int fim)
```

A função recebe a pilha `cromos`, e os valores **`inicio`** e **`fim`**, que representam o valores de início e de fim do intervalo de cromos a retirar (o valor **`inicio`** e o valor **`fim`** também se retiram). A função deve retornar a pilha sem os cromos a retirar e os cromos sobrantes devem estar na mesma ordem. Deve apenas recorrer às funções da pilha (não serão consideradas soluções em que os elementos da pilha são percorridos diretamente). **Sugestão**: utilize uma pilha auxiliar para manter a ordem.

Dado 

com intervalo entre 10 e 25 o resultado será 


```
Os cromos foram retirados corretamente (Certo)
Cromos: [2,5,6,34,35,5]
```

P1.2. [4 valores] Está a desenvolver um programa para gestão de correspondência numa estação de correios. Este programa utiliza as bibliotecas de vetores e listas ligadas que se encontram disponíveis no ficheiro `vetor.h/.c` e `lista.h/.c`. As bibliotecas foram adaptadas para guardar os dados necessários para este programa. Os dados constantes do ficheiro `correspondencia.txt` são carregados em três vetores: um contendo os nomes dos remetentes, outro contendo os nomes dos destinatários e um terceiro contendo os códigos postais do endereço de destino.

Implemente no ficheiro **parte1.c** fornecido a função `contar_correspondencia`:

```
lista* contar_correspondencia(vetor *vcp_dest, int *cpdistintos)
```

A função aceita como argumento um vetor com os códigos postais dos destinatários da correspondência depositada na estação (`vcp_dest`). A função efetua a contagem do número de cartas existentes para os diferentes destinos (código postal do destinatário) e coloca essa informação na lista ligada a devolver. A função deve ainda preencher o número de códigos postais distintos que foram encontrados, usando para o efeito o parâmetro `cpdistintos`. Em caso de erro ou argumentos inválidos, a função deve retornar NULL.

Depois de implementada a função**,** o programa deverá apresentar, não necessariamente pela mesma ordem:

```
Tamanho da lista OK: 11
Numero de codigos postais distintos retornado OK: 11

2440-000 BATALHA : 2 cartas
4411-801 ARCOZELO VNG : 7 cartas
2440-445 BATALHA : 2 cartas
2495-013 S. MAMEDE : 2 cartas
4150-740 PORTO : 7 cartas
4415-350 PEDROSO : 5 cartas
5400-582 CHAVES : 7 cartas
4615-019 FREIXO DE CIMA : 7 cartas
2125-998 MUGE : 7 cartas
7940-311 FARO DO ALENTEJO : 6 cartas
4935-204 VIANA DO CASTELO : 9 cartas
```

**** Submeta o ficheiro parte1.c no SIGEX ****

Prova com consulta (Moodle). Duração: 70min. Parte prática [15/20 valores]

PARTE 2

Responda às duas alíneas seguintes no ficheiro **parte2.c**. Utilize sempre que conveniente as bibliotecas de funções disponibilizadas para gestão e análise das estruturas de dados.

P2.1 [4 valores] Implemente a função `max_heap` que cria uma max-heap a partir de uma min-heap dada, isto é, quando removidos, os elementos da heap retornada devem vir pela ordem inversa dos elementos da heap dada. Após a execução da função, a heap dada deve preservar todo o seu conteúdo inicial.

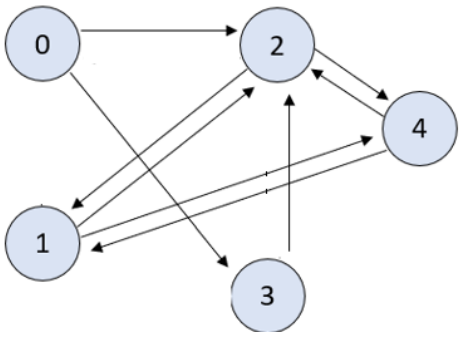
```
heap *max_heap(heap *h)
```

A função recebe o apontador para a heap dada e deverá retornar o apontador para a nova heap ou NULL em caso de erro. Utilize as funções disponíveis na biblioteca `heap.h/.c`.

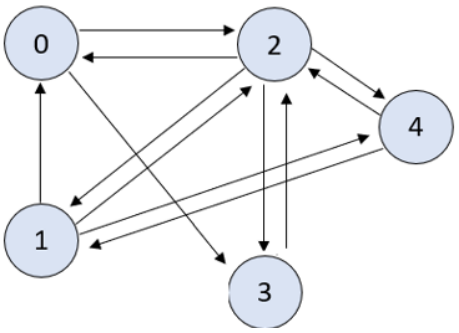
Depois de implementada a função, o programa deverá apresentar:

```
Sequência de elementos removidos da heap original (min-heap):
A B C D E F
Sequência de elementos removidos da heap retornada (max-heap):
F E D C B A
```

P2.2 [3 valores]



Grafo 1



Grafo 2

Implemente a função **subgrafo** que recebe dois **digrafos** e retorna: 1 se o grafo no primeiro argumento for subgrafo do segundo; 0 se o grafo no primeiro argumento **não** for subgrafo do segundo; e -1 em caso de erro. Um grafo **g1** é subgrafo de **g2 se e só se**: 1) todos os [identificadores de] vértices de **g1** existirem em **g2**; 2) todas as arestas de **g1** existirem em **g2**; 3) **g1** e **g2 não são** iguais. Todos os grafos envolvidos têm arestas sem pesos.

Considere a seguinte assinatura da função:

```
int subgrafo(grafo *g1, grafo *g2)
```

Para a chamada com g1 = **Grafo 1** e g2 = **Grafo 2** o programa deverá apresentar:

```
g1 E subgrafo de g2
```

Para a chamada com g1 = **Grafo 2** e g2 = **Grafo 1** o programa deverá apresentar:

```
g1 NAO E subgrafo de g2
```

Para a chamada com g1 = **Grafo 2** e g2 = **Grafo 2** o programa deverá apresentar:

```
g1 NAO E subgrafo de g2
```

***** Submeta o ficheiro parte2.c no SIGEX *****