

**Submeter** apenas a resolução da prova, num ficheiro zip (inclui ficheiros \*.cpp e \*.h). O ficheiro zip não deve conter pastas. Não necessita incluir o ficheiro tests.cpp

- A resolução submetida será testada com um conjunto adicional de testes unitários, pelo que passar com sucesso os testes fornecidos não garante a cotação completa
- Se a resolução submetida passar apenas alguns dos testes, poderá obter cotação parcial na respetiva questão

A **UverEats** é uma empresa de entrega de refeições ao domicílio, com base em plataformas tecnológicas. O sistema utilizado pela UverEats mantém um registo dos condutores disponíveis para a realização das entregas (set **drivers**) e dos restaurantes com os quais a empresa mantém contrato de fornecimento de refeições *take-away* (lista **restaurants**). A empresa também mantém um registo dos clientes que recorrem aos seus serviços de entregas (vector **clients**).

Cada restaurante (objeto da classe **Restaurant**), da rede de restaurantes da empresa, é caracterizado por um nome (*name*) e uma avaliação (*rating*). Os pedidos de refeição que o restaurante recebe (objetos da classe **Order**) têm associado um número de controlo (*orderNumber*), o nome do cliente (*client*) e o nome do restaurante (*restaurant*). Todos os pedidos recebidos são mantidos na fila **orders**, do respetivo restaurante.

Um cliente (objeto da classe **Client**) é identificado pelo seu nome (*name*) e pela sua antiguidade no sistema, definida pela propriedade *seniority*. Um motorista (objeto da classe **Driver**) é identificado pelo seu ID (*driverId*) e pelo seu nome (*driverName*). Cada motorista mantém os pedidos que devem entregar organizados em um pilha (*toDeliver*).

As classes **UverEats**, **Restaurant**, **Client**, **Order** e **Driver** estão parcialmente definidas a seguir.

<pre>class UverEats {     set &lt;Driver&gt; drivers;     list &lt;Restaurant&gt; restaurants;     vector&lt;Client&gt; clients; public:     //TODO:     void sortClients();     float averageRestaurantRating(int numMinOrders) const;     vector&lt;Driver&gt; getDriversWithOrders(int n) const;     vector&lt;string&gt; checkCommonClients(unsigned d1, unsigned d2);     void createRestaurantBranch(string restMain, string restBranch); };  class Restaurant {     string name;     float rating;     queue&lt;Order*&gt; orders; public:     //TODO:     void removeOrder(string client, string orderNumber ); };</pre>	<pre>class Client {     string name;     int seniority; public:     //TODO:     bool operator&lt;(const Client &amp;c) const; };  class Order {     string orderNumber;     string client;     string restaurant; };  class Driver {     unsigned driverId;     string driverName;     stack&lt;Order&gt; toDeliver; public:     //TODO:     vector&lt;string&gt; checkCommonClients(const Driver &amp;d2);     bool operator&lt;(const Driver &amp;d1) const; };</pre>
--	---

**Nota importante!** A correta implementação das alíneas seguintes, referentes à utilização de Listas (list), Filas (queue), Pilhas (stack) e Árvores Binárias de Pesquisa (set), pressupõe a implementação dos operadores adequados nas classes e estruturas apropriadas.

- a) **[2.5 valores]** Implemente, na classe **UverEats**, o membro-função:

```
void sortClients()
```

que ordena o vetor de clientes (*clients*) por valor decrescente de antiguidade (*seniority*) e, em caso de igualdade, alfabeticamente pelo nome do cliente.

- b) **[3.0 valores]** A empresa deseja monitorizar a qualidade média dos restaurantes mais solicitados pelos seus clientes. Implemente, na classe **UverEats**, o membro-função:

```
float averageRestaurantRating(int numMinOrders) const
```

que determina a média das avaliações (membro-dado *rating*) dos restaurantes com número de pedidos superior a um dado valor (argumento *numMinOrders*). Caso não haja restaurantes com número de pedidos superior a *numMinOrders*, o valor retornado pela função será 0.0.

- c) **[3.0 valores]** Implemente, na classe **Restaurant**, o membro-função:

```
void removeOrder(string client, string orderNumber)
```

que permite a um cliente (*client*) cancelar um pedido (*orderNumber*), removendo-o da fila *orders*.

- d) **[3.0 valores]** Com o objetivo de manter um serviço de qualidade, a empresa **UverEats** pretende manter contratos com restaurantes bem avaliados apenas. Implemente, na classe **UverEats**, o membro-função:

```
void removeRestaurants(float minRating)
```

que remove da lista de restaurantes (*restaurants*) da rede da empresa todos aqueles cuja avaliação (*rating*) é inferior a uma avaliação mínima admissível (argumento *minRating*). A lista *restaurants* deve manter a ordem relativa dos restantes restaurantes.

- e) **[3.0 valores]** Um condutor desconfia que as suas encomendas possuem clientes em comum com as de outro colega, e quer saber quais são esses clientes. Implemente, na classe **Driver**, o membro-função:

```
vector<string> checkCommonClients(const Driver &d2)
```

que verifica se entre a sua pilha de pedidos e a do condutor *d2* há pedidos de clientes comuns. Se houver, os nomes desses clientes devem ser retornados num vetor, senão, o vetor a ser retornado estará vazio. A ordem dos elementos no vetor de retorno é indiferente.

- f) **[3.0 valores]** Implemente, na classe **UverEats**, o membro-função:

```
vector<Driver> getDriversWithOrders(int numOrders) const
```

que procura no conjunto de condutores (membro-dado **drivers**) todos os condutores que tenham menos de um determinado número de pedidos (argumento *numOrders*), retornando-os num vetor ordenado por ordem decrescente de número de pedidos e, em caso de igualdade, por ordem crescente dos seus ids. Caso não se encontrem quaisquer condutores com menos de *numOrders*, o vetor retornado pela função estará vazio.

O conjunto de condutores da empresa (`set<Driver> drivers`) está ordenado por ordem decrescente de número de pedidos (pedidos nas pilhas *orders* respetivas) e, em caso de igualdade, por ordem crescente dos seus id (*idDriver*).

**Importante:** implemente apropriadamente o overload dos operadores da classe **Driver**. Também é avaliada a correta construção/organização do `set drivers`.

- g) **[2.5 valores]** Um dos restaurantes da empresa **UverEats** está com muitos pedidos (*orders*) e precisa de ajuda, criando para isso um restaurante filial. Implemente, na classe **UverEats**, o membro-função:

```
void createRestaurantBranch(string restMain, string restBranch)
```

que cria um novo restaurante chamado *restBranch*, na lista de restaurantes da empresa, e coloca na sua fila alguns dos pedidos do restaurante sede, chamado *restMain*. Os pedidos a serem transferidos para o novo restaurante são aqueles que se encontram em posições pares na fila do restaurante sede (2.ª posição, 4.ª posição, ...), removendo-os da sua fila. O novo restaurante deve ser adicionado no final da lista **restaurants**.

Submeter apenas a resolução da prova, num ficheiro zip (inclui ficheiros \*.cpp e \*.h). O ficheiro zip não deve conter pastas. Não necessita incluir o ficheiro tests.cpp