CI2 Parte prática. Duração: 1h30m

Uma empresa realiza entrega de refeições ao domicílio, recorrendo a estafetas que vãos aos restaurantes levantar os pedidos e depois os entrega na morada do cliente. A empresa utiliza uma plataforma online para a gestão dos pedidos e das entregas.

Para facilitar a consulta dos restaurantes aderentes (classe ***Restaurant***), a empresa mantém um catálogo de anúncios dos restaurantes (classe ***RestaurantAd***) numa Árvore Binária de Pesquisa, BST. Os restaurantes são identificados pelo nome da cadeia a que pertencem, sendo distinguidos entre si pela sua morada. Os restaurantes anunciados no catálogo também incluem a popularidade do estabelecimento, pelo número de *likes*. A BST está organizada em ordem alfabética, pelo nome das cadeias de restaurante (membro-dado *name*, da classe ***Restaurant***), e depois pela ordem alfabética da morada do estabelecimento (membro-dado *address*, da classe ***Restaurant***).

Os estafetas que realizam as entregas (classe ***Courier***) estão guardados numa Tabela de Dispersão, e são identificados pelo seu e-mail, que é único. Os estafetas também mantêm um vetor com os pedidos que devem entregar (classe ***Order***). Cada pedido tem um apontador para o cliente (***Client***) que realizou o pedido, em cuja morada deve ser entregue, e um apontador para o Restaurante onde o pedido deve ser levantado, para além da sua descrição e do valor a pagar.

Os pedidos a serem processados pela empresa são organizados numa Fila de Prioridade. O critério de prioridade adotado pela empresa considera primeiro os pedidos realizado pelos clientes mais frequentes, com maior valor para o membro-dado *pastOrders*, da classe ***Client***. No caso de empate, são processados primeiros os pedidos com o maior valor a pagar, indicado pelo membro-dado *totalDue* da classe ***Order***.

As classes ***Restaurant***, ***RestaurantAd***, ***Client***, ***Courier***, ***Order*** e ***MealCourier*** estão parcialmente indicadas abaixo.

```
class Restaurant {
    string name;
    string address;
public:
    //omitted...
};

class RestaurantAd {
    Restaurant* restaurant;
    unsigned likes;
public:
    //omitted...
};

class Client {
    string name;
    string address;
    int nif;
    unsigned pastOrders;
public:
    //omitted...
};

class Courier {
    string email;
    vector<Order*> toDeliver;
public:
    //omitted...
};
```

```
class Order {
    Client* client;
    Restaurant* restaurant;
    string description;
    fload totalDue;
public:
    //omitted...
};


class MealCourier {
    vector<Restaurant*> restaurants;
    BST<RestaurantAd> catalog;
    HashTableCouriers couriers;
    priority_queue<Order> orders;
public:
    vector<Restaurant*> getChain(string name) const;
    int addLike(Restaurant* rest) const;

    float getTotalDueInOrders() const;
    void updateCourierEmail(string oldEmail, string newEmail);
    int fireCourier(string courierEmail);

    Order deliverNextOrder();
    Order withdrawOrder(Order o);

    //omitted...
};
```

Nota importante! A correta implementação das alíneas seguintes, referentes à utilização de Árvores Binárias de Pesquisa, Tabelas de Dispersão e Filas de Prioridade, pressupõe a implementação dos operadores adequados nas classes e estruturas apropriadas.

//BST

a) [3 valores] Implemente na classe ***MealCourier***, o membro-função

```
vector<Restaurant*> MealCourier::getChain(string name) const
```

Esta função retorna um vetor com todos os restaurantes de uma mesma cadeia, cujos *restaurantAd* estão disponíveis para consulta na BST *catalog*. Os restaurantes pertencentes a uma cadeia têm todos o mesmo nome (membro-dado *name* da classe ***Restaurant***). No caso de não haver restaurantes de tal cadeia, o método retorna um vetor vazio.

b) [3 valores] Implemente na classe ***MealCourier***, o membro-função

```
int MealCourier::addLike(Restaurant *rest)
```

Este método adiciona um "*like*" ao restaurante *rest* no catálogo *catalog*, e retorna o total de *likes* atualizados para aquele restaurante. Caso não haja anúncios do restaurante no catálogo, o método retorna 0 (zero).

//TABELA DE DISPERSÃO

c) [2.5 valores] Implemente na classe ***MealCourier***, o membro-função

```
float MealCourier::getTotalDueInOrders() const
```

Este método retorna a soma total dos preços devidos em todos os pedidos atribuídos a todos os estafetas na Tabela de Dispersão (membro-dado *couriers*). Caso a tabela de dispersão esteja vazia ou não haja pedidos atribuídos aos estafetas, o método deve retornar 0.0 (zero).

d) [2.5 valores] Implemente na classe ***MealCourier***, o membro-função

```
void MealCourier::updateCourierEmail(string oldEmail, string newEmail)
```

Este método atualiza a morada eletrónica do estafeta cujo e-mail é *oldEmail*, substituindo-o por *newEmail*. As restantes informações do estafeta devem permanecer inalteradas, nomeadamente o vetor de pedidos que lhe foram atribuídos para entrega. Caso o estafeta não exista na tabela de dispersão ou *newEmail* já esteja a ser utilizado por um estafeta existente, o método não faz nada.

e) [3 valores] Implemente na classe ***MealCourier***, o membro-função

```
int MealCourier::fireCourier(string courierEmail)
```

Ao despedir um estafeta cujo e-mail é *courierEmail*, este método o retira da Tabela de Dispersão e atribui todos os pedidos que antes lhe tinham sido destinados ao estafeta com menor número de pedidos por processar, retornando o número total de pedidos que ficam com o novo estafeta. Caso o estafeta a ser despedido não exista ou seja o único estafeta na Tabela de Dispersão, o método deve retornar 0 (zero). Se for o único estafeta, ao ser despedido a Tabela ficará vazia.

// FILA DE PRIORIDADE…

f) [3 valores] Implemente na classe ***MealCourier***, o membro-função

```
Order MealCourier::deliverNextOrder()
```

Este método retorna o próximo pedido de compra (*Order*) na fila de prioridade para processamento, e retira-o da fila. Se a fila estiver vazia, não havendo pedidos a espera de processamento, o método deve lançar a exceção ***NoOrderToProcess***. Nota: A exceção ***NoOrderToProcess*** já está implementada.

g) [3 valores] Implemente na classe ***MealCourier***, o membro-função

```
Order MealCourier::withdrawOrder(Order o)
```

Este método procura na fila de prioridade o pedido ***o*** e retira-o da fila. Se a fila estiver vazia, não havendo pedidos a espera de processamento, ou o pedido ***o*** não for encontrado na fila, o método deve lançar a exceção ***NoOrderToProcess***. Nota: A exceção ***NoOrderToProcess*** já está implementada.

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