A pizzeria do Sr. Mário, especificada pela classe **Pizzeria**, vende pizzas para entrega ao domicílio. A pizzeria guarda registo dos seus clientes, que já tenham comprado uma pizza ao Sr. Mário, e um vetor de menus disponíveis.

A informação sobre um menu está descrita na classe **Menu**, e inclui o número de identificação, *id*, o nome do menu, *name*, o número de vezes que os clientes declararam gostar desse menu, guardado no membro-dado *likes*, e os ingredientes com que a pizza do menu é confecionada, guardados no vetor *ingredients*; os ingredientes são identificados pelos seus respetivos nomes. Os clientes são objetos da classe **Customer** e são caracterizados pelo número de contribuinte, *nif*, e pelo nome, *name*.

Independentemente do seu ID, ou nome, um menu é identificado pelos seus ingredientes. Assim, dois menus distintos não terão os mesmos ingredientes. O cliente, portanto, realiza a seu pedido identificando o menu pelos seus ingredientes.

As classes **Menu**, **Customer** e **Pizzeria**, estão parcialmente definidas a seguir.

NÃO PODE acrescentar membros-dado nas classes **Customer** e **Pizzeria.**

```
class Menu {
   int id;
   string name;
   int likes;
   vector<string> ingredients;
public:
   //...
   Menu(string n1, vector<string> is1);
   void setIngredients(vector<string> is1);
   //...
};
```

```
class Customer {
   int nif;
   string name;
public:
   //...
};
```

```
class Pizzeria {
   vector<Menu> menus;
   vector<Customer*> customers;
public:
   //...
   vector<Menu> optionsWithinIngredientLimits(int i1, int
i2) const;
   vector<Menu> popularityRank() const;
   Customer* chefCustomer();
   Menu& removeIngredient(vector<string> is1, string i1);
   //...
  };
```

a) [**2.5 valores**] Implemente na classe **Menu** o construtor abaixo, que atribui a cada objecto criado um novo ID, sequencial e incremental; o primeiro menu a ser criado terá ID igual a 1, o segundo terá ID igual a 2, e assim sucessivamente.

```
Menu(string n1, vector<string> is1)
```

O construtor recebe como argumentos o nome que designa o menu, *n1*, e um vetor, *is1*, contendo o nome dos ingredientes que confecionam a pizza. O membro-dado *likes* deverá ser inicializado a 0.

b) [**2.5 valores**] Implemente a função template abaixo:

```
bool isSet(const vector<Comparable>& v1)
```

que recebe um vetor de elementos e verifica se todos são únicos. No caso de não haver elementos repetidos, a função retornará *true*; no caso de haver mais do que uma ocorrência de um mesmo elemento, a função retornará *false*.

c) [**3 valores**] Implemente na classe **Menu** o membro-função abaixo:

```
void setIngredients(const vector<string> &is1)
```

que atribui ao menu os ingredientes do argumento *is1*. No caso de haver ingredientes repetidos, o membro-função deverá lançar a exceção ExceptionIngredientsRepeated (definida na classe Menu). Sugestão: utilize o template implementado na alínea b).

d) [**3 valores**] Implemente na classe **Pizzeria** o membro-função abaixo::

```
vector<Menu> optionsWithinIngredientLimits(int i1, int i2) const
```

A função recebe dois inteiros, *i1* e *i2*, que indicam o número mínimo e o número máximo de ingredientes pretendidos para as opções de menu desejadas, respectivamente. A função retorna um vetor com todos os menus confecionados com um numero de ingredientes no intervalo *i1* e *i2*, inclusive. No caso de i1<1 ou i2<1 ou i1>i2, deverá lançar a exceção ExceptionInvalidIngredientLimits (definida na classe Pizzeria).

e) [**3 valores**] Implemente na classe **Pizzeria** o membro-função abaixo:

```
vector<Menu> popularityRank() const
```

Que retorna um vetor de menus, ordenado do menu mais popular para o menos popular. A popularidade de um menu é avaliada pelo número de *likes* que tem; quanto mais *likes*, mais popular será o menu. No caso de empate, será o menu com menos ingredientes. No caso de haver múltiplos menus nesta condição, deverão ser ordenados alfabeticamente. Sugestão: utilize a função `insertionSort`, fornecida.

f) [**3 valores**] Implemente na classe **Pizzeria** o membro-função abaixo:

```
Customer* chefCustomer()
```

Que retorna o apontador para o cliente mais criativo. A criatividade de um cliente é avaliada em função dos menus cujo nome é o nome do cliente e do número de *likes* desses menus (ao realizar um pedido, se o menu não existe, este é criado com o nome do cliente que o pediu). O cliente mais criativo será aquele com maior rácio *likes* por total de menus com o nome do cliente.

g) [**3 valores**]] Implemente na classe **Pizzeria** o membro-função abaixo:

```
Menu& removeIngredient(vector<string> is1, string i1)
```

O método procura entre os menus oferecidos pela Pizzeria aquele cujos ingredientes são identificados no vetor *is1*, passado como argumento, e apaga do menu o ingrediente pretendido, identificado pela string *i1*, passada como argumento. Caso não exista um menu com ingredientes is1, o método lança a exceção ExceptionInexistentMenu (definida na classe Pizzeria). No caso do menu ser localizado mas não conter o ingrediente *i1*, o método lança a exceção `ExceptionIngredientNotPresent` (definida na classe Menu); esta classe exceção deve incluir o membro-função `getIngredient()` que retorna o nome do ingrediente não existente. Quando o método é executado com sucesso, apagando do menu o ingrediente *s1*, retorna a referência para o menu atualizado com a nova lista de ingredientes.