Parte prática. Duração: 2h00m

Uma empresa de gestão de condomínios pretende implementar um sistema para armazenamento e manipulação da informação relativa às urbanizações onde efetua a gestão de condomínios. Uma urbanização (classe **Urbanizacao**) possui múltiplos imóveis. Um imóvel pode ser de dois tipos: apartamento ou vivenda.

Um **Imovel** possui como membros-dado um identificador, a área habitacional e o nome do seu proprietário. Um **Apartamento** possui ainda como membro-dado o andar em que se encontra. Uma **Vivenda** possui também como membros-dado a área exterior e informação sobre existência de piscina ou não.

O valor da mensalidade do condomínio de um imóvel é *mensImovel=50+0.2*área habitacional*. O valor da mensalidade de um apartamento é *mensImovel+1*andar*. O valor da mensalidade de uma vivenda, é *mensImovel+0.1*areaExterior+10*temPiscina*.

As classes **Urbanizacao**, **Imovel**, **Apartamento, Vivenda** e **ECondominio** estão parcialmente definidas a seguir.

NÃO PODE acrescentar membros-dado nas classes **Imovel**, **Apartamento** e **Vivenda.**

```
class Imovel {
   int areaHabitacao;
   string proprietario;
   int ID;
public:
   Imovel(int areaH, string prop, int id=0);
   // …
};

class Apartamento: public Imovel {
   int andar;
public:
   Apartamento(int areaH, int anda, string prop, int id=0);
   // …
};

class Vivenda: public Imovel {
   int areaExterior;
   bool piscina;
public:
   Vivenda(int areaH, int areaExt, bool pisc, string prop, int id=0);
   // …
};
```

```
class Urbanizacao {
   string nome;
   int ID;
   vector<Imovel *> imoveis;
public:
   Urbanizacao(string nm, int id);
   void adicionaImovel(Imovel *im1);
   // …
};
```

```
class ECondominio {
   vector<Urbanizacao> urbanizacoes;
public:
   void adicionaUrbanizacao(Urbanizacao
                                    urb1);
   // …
};
```

Parte prática. Duração: 2h00m

a) [**2.5 valores**] Calcule o valor da mensalidade a pagar por um condómino de um **Apartamento** ou **Vivenda**. Implemente nas classes **Apartamento** e **Vivenda** o membro-função:

float mensalidade() const

Esta função retorna o valor da mensalidade de condomínio a pagar.

b) [**3 valores**] Um proprietário pretende efectuar o pagamento da mensalidade de todos os seus imóveis. Implemente na classe **ECondominio** o membro-função:

float mensalidadeDe(string nomeProp) const

Esta função calcula a soma das mensalidades a pagar pelo proprietário de nome *nomeProp*. Nota: uma mesma pessoa pode ser proprietária de um ou mais imóveis existentes em uma ou várias urbanizações.

c) [**3 valores**] Um proprietário incompatibilizou-se com a empresa que gere o condomínio. Este convenceu todos os outros condóminos das urbanizações onde possui algum imóvel a rescindir o contrato com a empresa. Implemente na classe **ECondominio** o membro-função:

vector<Urbanizacao> removeUrbanizacaoDe(string nomeProp)

Esta função retira da empresa todas as urbanizações onde o proprietário de nome *nomeProp* possui algum imóvel. Retorna um vetor com as urbanizações eliminadas da empresa.

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

*vector<Imovel *> areaSuperiorA(int area1) const*

Esta função retorna um vetor com os imóveis existentes na urbanização de área total superior a *area1*. A área total de um imóvel é a soma da área de habitação com a área exterior.

e) [**3 valores**] Implemente na classe **Urbanizacao** o operador >. Uma urbanização é maior que outra se tiver menor razão nº_imóveis/nº_proprietarios (considerar apenas os proprietários não repetidos).

f) [**2.5 valores**] Implemente na classe **ECondominio** o operador de função que aceita um argumento do tipo *string* (representando o nome de uma urbanização) e retorna o vetor de imóveis (*vector<Imovel *>*) dessa urbanização. Deve retornar o vetor vazio, caso não exista nenhuma urbanização com o nome especificado.

g) [**3 valores**] Todas as urbanizações são agora identificadas por um ID diferente e sequencial (inteiro). A primeira urbanização possui ID igual a 1.

i. [**2 valores**] Implemente um novo construtor: *Urbanizacao(string nm)*

e o membro-função estático que reinicia o contador ID: *resetID()*

ii. [**1 valor**] Os imóveis dentro de cada urbanização também são identificados por IDs diferentes e sequenciais (com início em 1 em cada urbanização). Efetue as alterações que considerar necessárias.