CI3 Parte prática. Duração: 2h00m

Submeter a resolução num ficheiro zip (incluir os ficheiros *.cpp e *.h). Não necessita incluir o ficheiro Tests.cpp

Uma galeria de artes é especializada em pinturas e mantém uma coleção de obras de diferentes autores, produzidas ao longo de vários anos. A cada obra é atribuído um título, o nome do autor, um preço referente ao seu valor, e é também indicado o ano da sua produção. A classe ***Gallery***, que representa a galeria de arte, tem um vetor de apontadores para objetos da classe ***Paint***, que presenta a coleção de pinturas mantidas e geridas pela galeria. Considera-se a priori que cada pintura é única, e pode ser identificada pelo seu autor e pelo seu título, inequivocamente.

Para facilitar o acesso e consulta às obras disponíveis, a galeria implementa uma BST de objetos da classe ***PaintCatalogItem***, que fornece informação sobre pinturas da coleção no catálogo da galeria. A BST está ordenada por ordem decrescente de ano de produção da obra e, em caso de empate, por ordem alfabética de nome de autor. Em caso de empate destes dois critérios, a ordenação é efetuada por ordem alfabética de título da obra.

Para facilitar a contabilização das obras dos diferentes autores, a galeria implementa uma Tabela de Dispersão de objetos da classe ***AuthorRecord***, mantendo registo do número de obras disponíveis de cada autor, assim como a contabilização das transações de vendas, usando o nome do autor como chave do registo.

Com a finalidade de realizar exposições sempre muito atrativas, a galeria dá prioridade às suas obras mais relevantes, baseando-se no ano da sua produção e no seu valor de mercado. Esta informação é mantida numa Fila de Prioridade, que guarda objeitos da classe ***ExhibitionItem***, encontrando-se no topo a obra mais antiga e, em caso de empate, a de maior valor.

Paint, ***PaintCatalogItem***, ***AuthorRecord***, ***ExihibitionItem*** e ***Gallery*** estão parcialmente indicados abaixo.

```
class Paint {
    string author;
    string title;
    int year;
    double price;
public:
    //omitted...
};

class PaintCatalogItem {
    Paint* paintItem;
public:
    //omitted...
};

class AuthorRecord {
    string author;
    int availablePaints;
    double totalSells;
public:
    //omitted...
};

class ExhibitionItem {
    Paint* toShow;
public:
    //omitted...
};
```

```
class Gallery {
    vector<Paint*> collection;
    BST<PaintCatalogItem> catalog;
    HashTableAuthorRecord authorRecords;
    priority_queue<ExhibitionItem> paintsToShow;
public:
    Gallery(vector<Paint*> c);
    vector<Paint*> getCollection() const;
    void setCollection(vector<Paint*> c);
    priority_queue<ExhibitionItem> getPaintsToShow() const;

    void generateCatalog();
    BST<PaintCatalogItem> getCatalog() const;
    vector<Paint*> getPaintsBy(string a) const;
    vector<Paint*> getPaintsBetween(int y1, int y2) const;
    bool updateTitle(Paint* p, string tnew);

    int recordAvailablePainters();
    double totalSells() const;
    double sellPaint(string a, string t);

    void prepareExhibition();
    vector<Paint*> nBestExibition(int n, int maxPerYear);
    int itemExibitionOrder(string a, string t);
};
```

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.

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

```
vector<Paint*> Gallery::getPaintsBy(string a) const;
```

Esta função procura no catálogo da galeria (dado-membro *catalog*) todas as pinturas do autor **a**, independentemente do seu título e ano de produção, retornando-as num vetor. O vetor a retornar deve estar ordenado por ordem decrescente de ano de produção da obra e, em caso de empate, por ordem alfabética de nome de autor e, em caso de novo empate, por ordem alfabética de título da obra. Caso não se encontrem quaisquer obras do autor, o vetor retornado pela função estará vazio.

b) [**2.5 valores**] Implemente na classe ***Gallery***, o membro-função

```
vector<Paint*> Gallery::getPaintsBetween(int y1, int y2) const;
```

Esta função procura no catálogo da galeria (dado-membro *catalog*) todas as pinturas que foram produzidas num determinado período de tempo, compreendido entre os anos ***y1*** e ***y2***, inclusivé, retornando-as num vetor. O vetor a retornar deve estar ordenado por ordem decrescente de ano de produção da obra e, em caso de empate, por ordem alfabética de nome de autor e, em caso de novo empate, por ordem alfabética de título da obra. Caso não se encontrem quaisquer obras no período indicado, o vetor retornado pela função estará vazio.

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

```
bool Gallery::updateTitle(Paint* p, string tnew);
```

Esta função permite a atualização do título de uma data pintura ***p***, consultada no catálogo da galeria (dado-membro *catalog*), a partir da atribuição de um novo título, ***tnew***, retornando ***true*** em caso de sucesso. No caso de a pintura não constar no catálogo da galeria, a função deverá retornar ***false***.

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

```
int Gallery::recordAvailablePainters();
```

Esta função cria os registos dos autores disponíveis na coleção da galeria (dado-membro *collection*), organizando-os numa tabela de dispersão (dado-membro *authorRecords*) e retornando o número de autores existentes na tabela de dispersão. A chave dos registos ***AuthorRercords*** são os nomes dos próprios autores; cada autor terá apenas um ***AuthorRecords*** associado a si, onde serão contabilizadas as suas obras disponíveis na galeria, e o total arrecado pelas vendas das suas obras.

e) [**2.5 valores**] Implemente na classe ***Gallery***, o membro-função

```
double Gallery::sellPaint(string a, string t);
```

Esta função realiza na galeria a venda da obra de título ***t***, do autor ***a***. O registo do autor é então atualizado na tabela de dispersão (dado-membro *authorRecords*), de forma a fazer-se refletir a transação. O número de pinturas do autor, ***availablePaints***, disponíveis na galeria é, portanto, decrementado enquanto o membro-dado ***totalSells*** é incrementado do valor recebido pela venda, equivalente ao preço da pintura. O vetor ***collection***, da galeria, também deve ser atualizado, deixando a obra vendida de fazer parte da coleção da galeria. A função retorna o valor da venda realizada.

f) [**2.5 valores**] Implemente na classe ***Gallery***, o membro-função

```
double Gallery::totalSells() const;
```

Este método retorna a totalidade das receitas arrecadadas como resultado das vendas das pinturas de todos os autores presentes tabela de dispersão (dado-membro *authorRecords*).

g) [**2.5 valores**] Implemente na classe ***Gallery***, o membro-função

```
int Gallery::itemExibitionOrder(string a, string t);
```

Este método procura, na fila de prioridade (dado-membro *paintsToShow*), a pintura do autor ***a***, com título ***t***, retornando a ordem em que a pintura se encontra na fila de prioridade. Considere que o elemento mais prioritário da fila tem ordem 1, o segundo ordem 2, e assim sucessivamente. Caso a pintura não esteja na fila de prioridade, o método retornará 0 (zero). A fila de prioridade *paintsToShow* não deverá ser alterada.

h) [**2.5 valores**] Implemente na classe ***Gallery***, o membro-função

```
vector<Paint*> Gallery::nBestExibition(int n, int maxPerYear);
```

Este método seleciona ***n*** pinturas da fila de prioridade (dado-membro *paintsToShow*) para uma exposição, retornando-as num vetor. Devem ser selecionadas as pinturas mais antigas e, para o mesmo ano, as mais caras. Só pode ser selecionado um número máximo de **maxPerYear** pinturas de um mesmo ano. As pinturas selecionadas são removidas da fila de prioridade *paintsToShow*.

Submeter a resolução da prova num ficheiro zip (inclui ficheiros *.cpp e *.h). Não necessita incluir o ficheiro Tests.cpp