CI3 Parte prática. Duração: 2h00m

Um grupo de amigos, leitores inveterados, muito habituados a trocarem entre si livros e outras obras literárias, decidiram instituir um clube de leitura, representado pela classe **ReadingClub**.

Os leitores, representados pela classe **User**, são identificados pelo seu nome (*name*) e um contacto de e-mail (*eMail*). Um leitor pode estar a ler uma obra literária, *titleReading*, representada pelo par ordenado (título, autor), que são *strings*. Guarda um registo das obras que já leu, no vetor *titlesRead*, de pares (título, autor). A classe **UserRecord** encapsula um apontador para um objeto da classe **User**, a fim de facilitar a gestão dos registos dos leitores do clube.

A classe **Book** representa uma obra literária, como um livro, e tem os membros-dado *title*, *author*, e *year*, representando título, autor (ou autores) e ano da edição, respetivamente. Um livro poderá ter várias edições, identificadas por anos distintos; é possível também que diferentes livros tenham o mesmo título, diferindo entretanto pelos seu autores, o que caracteriza obras distintas. O membro dado *reader*, que é um apontador para um objeto da classe **User**, guarda a referência para o leitor que possa estar a ler o livro num dado momento, tornando-o indisponível para ser emprestado a outro leitor.

Um objeto da classe **BookCatalogItem** é um item de catálogo para um livro de título (*title*) e autores (*author*) conhecidos, mantendo um registo de todas os exemplares daquele livro no clube (*items*).

A classe **ReadingClub** mantém referência para todas as unidades de livros existentes no clube, emprestados ou disponíveis, no vetor *books*.

Para facilitar a consulta dos livros do clube, objetos da classe **BookCatalogItem** são mantidos numa Árvore Binária de Pesquisa (BST) (*catalogItems*), enquanto a gestão dos leitores é feita a partir de uma Tabela de Dispersão (*userRecords*), de objetos da classe **UserRecord**. O clube também classifica os seus leitores, a partir de uma Fila de Prioridade (*readerCandidates*) e oferece, de tempos em tempos, um prémio ao seu melhor leitor.

As classes **User**, **UserRecord**, **Book**, **BookCatalogItem** e **ReadingClub**, estão parcialmente definidas a seguir.

```
class User {
  string name;
  string eMail;
  pair<string, string> titleReading;
  vector<pair<string, string> > titlesRead;
public:
  friend class UserRecord;
  User(string name, string eMail);
  string getName() const;
  string getEMail() const;
  pair<string, string> getReading() const;
  vector<pair<string,string>> getReadings() const;
  int numReadings() const;
  bool operator<(const User& u1) const;
};

class UserRecord {
  User* userPtr;
public:
  UserRecord(User* user);
  string getName() const;
  void setEMail(string eMail);
  string getEMail() const;
};

class Book {
  const string title;
  const string author;
  const unsigned year;
  User* reader;
public:
  Book(string t, string a, unsigned y);
  string getTitle() const;
  string getAuthor() const;
  unsigned getYear() const;
  void setReader(User* reader);
  User* getReader() const;
};
```

```
class BookCatalogItem {
  string title;
  string author;
  vector<Book*> items;
public:
  BookCatalogItem(string t, string a, unsigned y);
  string getTitle() const;
  string getAuthor() const;
  vector<Book*> getItems() const;
  void addItems(Book* book);
  bool operator<(const BookCatalogItem &bci1) const;
  bool operator==(const BookCatalogItem &bci1) const;
};

class ReadingClub {
  vector<Book*> books;
  BST<BookCatalogItem> catalogItems;
  HashTabUserRecord userRecords;
  priority_queue<User> readerCandidates;
public:
  ReadingClub(vector<Book*> books);
  void addBook(Book* book);
  void addBooks(vector<Book*> books);
  vector<Book*> getBooks() const;
  Book* boorrowBook(string t, string a, User* u);

  // Part BST
  void generateCatalog();
  vector<Book*> getAvailableItems(Book* book) const;
  bool borrowBookFromCatalog(Book* b, User* u);
  // Part Hash Table
  void addUserRecord(User* user);
  void changeUserEMail(User* user, string newEMail);
  // Part Piority Queue
  void addBestReaderCandidates(const vector<User>& us,
int min);
  int awardReaderChampion(User& champion) const;
};
```

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) [**3 valores**] Implemente na classe ***ReadingClub*** o membro-função

```
void ReadingClub::generateCatalog()
```

que adiciona ao catálogo *catalogItems* novos itens de catálogo para os livros do vector *books*. Os itens do catálogo (objetos da classe *BookCatalogItem*) estão organizados na BST alfabeticamente, pelo título do livro e pelo autor do livro. Poderá haver livros com mesmo título, mas de autores diferentes – isto implicará itens distintos no catálogo. Cada item do catálogo agrupa no vector *items*, da classe *BookCatalogItems*, todos os livros de mesmo título e autor, independentemente do seu ano de edição.

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

```
vector<Book*> ReadingClub::getAvailableItems(Book* book) const
```

que retorna um vector com apontadores para todos os livros de mesmo título e mesmo autor do argumento *book*, que estejam disponíveis. Um livro da lista *items* de um *BookCatalogItem* está disponível quando não tem associado qualquer leitor ao seu membro-dado *reader*.

c) [**3 valores**] Os utilizadores do clube, ao requisitarem um livro, passam a estar associados como leitores de um exemplar em particular, que não necessariamente tenha o mesmo ano de edição do livro que pretendem requisitar. Implemente na classe ***ReadingClub*** o membro-função

```
bool ReadingClub::borrowBookFromCatalog(Book* book, User* reader)
```

que procura através do catálogo *catalogItems* se haverá algum exemplar disponível de mesmo título e mesmo autor do livro *book*, passado como argumento. Se houver um exemplar disponível (o primeiro encontrado), o utilizador *reader* fica associado àquele exemplar em particular, que passará a estar indisponível. Se houver um exemplar disponível e a operação for bem-sucedida, a função retorna ***true***; caso contrário, a função retornará ***false***. Use o membro-função ***addReading***, da classe **User**, para estabelecer a associação do livro ao leitor.

d) [**3 valores**] A fim de gerir melhor os leitores do clube, pretende-se guardar os seus registos (objetos da classe ***UserRecord***) numa Tabela de Dispersão (membro-dado *userRecords*), em que os utilizadores são reconhecidos pelos seus e-mails, que são únicos para cada utilizador (diferentes utilizadores poderão ter o mesmo nome, mas terão e-mails distintos). Implemente na classe ***ReadingClub*** o membro-função

```
void ReadingClub::addUserRecord(User* user)
```

que insere na Tabela de Dispersão *userRecords* um novo registo para o leitor *user* passado como argumento.

e) [**2,5 valores**] Com o objetivo de manter os registos dos leitores sempre atualizados, implemente na classe ***ReadingClub*** o membro-função

```
void ReadingClub::changeUserEMail(User* user, string newEMail)
```

que permite ao leitor *user* atualizar o seu endereço eletrónico para *newEMail*.

f) [**2,5 valores**] De tempos em tempos, o clube oferece um prémio ao seu melhor leitor, ou seja, àquele que tem acumulado até à data o maior número de leituras, incluindo a sua leitura atual, no caso de estar a ler algum livro de momento. Para ordenar os candidatos, o clube utiliza uma *heap* the máximos (membro-dado *readerCandidates*). Implemente na classe ***ReadingClub*** o membro-função

```
void ReadingClub::addBestReaderCandidates(const vector<User>& candidates, int min)
```

que insere na *heap readerCandidates* apenas os candidatos aptos a concorrer ao prémio. Um candidato está apto a concorrer ao prémio quando o seu número de leituras for igual ou superior a um valor mínimo estipulado, identificado pelo argumento *min*.

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

```
int ReadingClub::awardReaderChampion(User& champion) const
```

que identifica o campeão entre os leitores concorrentes, se houver um. Só poderá haver um único campeão, ou seja, o leitor no topo da *heap* deverá ter o maior número absoluto de leituras, não podendo o segundo (ou demais) ter número de leituras iguais a ele. Se houver um campeão absoluto, a função passa este leitor para o argumento *champion*, passado por referência, e retorna o número de candidatos efectivos que concorreram ao prémio. Caso contrário, o argumento *champion* não é modificado, e a função retorna 0 (zero).