CI1 Parte prática. Duração: 1h30m

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`

- A resolução submetida será testada com um conjunto adicional de testes unitários, pelo que passar com sucesso os testes fornecidos não garante a cotação completa
- Se a resolução submetida passar apenas alguns dos testes, poderá obter cotação parcial na respetiva questão

O sistema de gestão de consultas de um hospital mantém um registo dos médicos disponíveis (vetor **`doctors`**), dos pacientes do hospital (lista **`patients`**) e das consultas efetuadas (*`consultations`*). O hospital mantém ainda um registo dos pacientes antigos (set **`oldPatients`**), ordenado por ordem crescente de ano da última consulta e, em caso de igualdade, por ordem crescente de id do paciente.

As consultas efetuadas (*`consultations`*) estão guardadas numa lista e organizadas por especialidade médica, sendo cada elemento da lista uma pilha contendo as consultas de uma mesma especialidade.

Um médico (objeto da classe **`Doctor`**), é caracterizado por um id (*`idDoctor`*), uma especialidade (*`speciality`*) e a fila de doentes que tem no momento para consulta (*`toAttend`*).

Um paciente (objeto da classe Patient) é caracterizado por um id (*`idPatient`*), o número de consultas que já realizou (*`numConsultations`*) e a identificação do ano da última consulta (*`lastConsultationYear`*).

Uma consulta (objeto da classe **`Consultation`**) é caracterizada por id do paciente (*`idPatient`*), especialidade médica da consulta (*`specialty`*) e uma descrição (*`description`*).

As classes **`Doctor`**, **`Patient`**, **`Consultation`** e **`Hospital`** estão parcialmente definidas a seguir:

```
class Doctor {
  int idDoctor;
  string specialty;
  queue<Patient> toAttend;
public:
   // …
};

class Patient {
  int idPatient;
  int numConsultations;
  int lastConsultationYear;
public:
  // …
};
```

```
class Consultation {
  int idPatient;
  string specialty;
  string description;
public:
  // …
};

class Hospital {
  vector<Doctor> doctors;
  list<Patient> patients;
  set<Patient> oldPatients;
  list<stack<Consultation> > consultations;
public:
  // …
};
```

Nota importante! A correta implementação das alíneas seguintes, referentes à utilização de Listas (*list*), Filas (*queue*), Pilhas (*stack*) e Árvores Binárias de Pesquisa (*set*), pressupõe a implementação dos operadores adequados nas classes e estruturas apropriadas.

a) **[2.5 valores]** Implemente na classe **`Hospital`** o membro-função:

```
void sortDoctors()
```

que ordena a lista de médicos (*`doctors`*), por ordem decrescente de ocupação (número de doentes a aguardar consulta: *`toAttend`*) e, em caso de igualdade, por ordem crescente de id (`id`).

b) **[3.0 valores]** Implemente na classe **`Hospital`** o membro-função:

```
float averageNPatients(string sp) const
```

que determina o número médio de pacientes atualmente à espera de consulta (fila *`toAttend`*) para os médicos da especialidade *`sp`*. Caso não existam médicos da especialidade `sp`, o valor retornado pela função será 0.0.

c) **[3.0 valores]** Implemente na classe **`Hospital`** o membro-função:

```
int removePatients(int minC)
```

que remove da lista *`patients`* todos os pacientes com número de consultas (*`numConsultations`*) inferior a *`minC`* e retorna o número de pacientes removidos. A lista *`patients`* deve manter a ordem relativa dos restantes pacientes.

d) **[3.0 valores]** Implemente na classe **`Doctor`** o membro-função:

```
void moveToFront(int idP)
```

que, por necessidade urgente de um paciente de código *`idP`*, coloca este paciente no início da fila *`toAttend`*. Os restantes pacientes presentes na fila mantêm a sua posição relativa. Se não existir nenhum paciente de código *`idP`* na fila *`toAttend`*, esta mantém-se inalterada.

e) **[3.0 valores]** Implemente na classe **`Hospital`** o membro-função:

```
vector<Patient> getOldPatients(int y) const
```

que procura no conjunto de pacientes antigos (*`oldPatients`*) aqueles que tiveram a última consulta antes do ano *y* (exclusivo), retornando-os num vetor ordenado por ordem crescente de ano da última

consulta (*lastConsultationYear*) e, em caso de igualdade, por ordem crescente de id (*idPatient*). Caso não exista nenhum paciente com última consulta antes do ano *y*, o vetor de retorno estará vazio.

nota: O conjunto de antigos pacientes do hospital (`set<Patient> oldPatients`) está ordenado por ordem crescente de ano da última consulta (*lastConsultationYear*) e, em caso de igualdade, por ordem crescente de id (*id*).

nota: implemente apropriadamente o overload dos operadores da classe **Patient**. Também é avaliada a correta construção/organização do set *oldPatients*.

f) **[3.0 valores]** Implemente, na classe **Hospital**, o membro-função:

```
void processConsultation(Consultation c)
```

que coloca na lista *consultations* a informação relativa à consulta `c`. Esta deve ser colocada na pilha correspondente à especialidade médica da consulta. Se não existe nenhuma pilha com informação relativa a consultas dessa especialidade, deve criar uma pilha nova e colocá-la no final da lista *consultations*.

g) **[2.5 valores]** Se um médico (*id1*) possui bastantes pacientes à espera (*toAttend*), o hospital contrata um novo médico para ajudar. Implemente, na classe **Hospital**, o membro-função:

```
bool addDoctor(int id2, string sp2, int id1)
```

que adiciona o médico de id *id2* e especialidade *sp2* ao vetor de médicos do hospital (*doctors*). Alguns dos pacientes do médico de id *id1* são transferidos para a fila (*toAttend*) do novo médico de id *id2*. Os pacientes a serem transferidos são aqueles que se encontram em posições pares na fila *toAttend* do médico de id *id1* (2.ª posição, 4.ª posição, …), devendo ser removidos desta fila.

Se não existe nenhum médico de id *id1* ou a especialidade do novo médico (*sp2*) não é a mesma do médico *id1*, o novo médico é adicionado ao vetor sem pacientes na sua fila e a função retorna *false*. Caso contrário, retorna *true*.

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`