ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO
Departamento de Engenharia de Computação e Sistemas Digitais

# PCS3111 - LABORATÓRIO DE PROGRAMAÇÃO ORIENTADA A OBJETOS PARA A ENGENHARIA ELÉTRICA

EXERCÍCIO PROGRAMA 1 – 2º SEMESTRE DE 2019

## Resumo

Os EPs de PCS3111 têm como objetivo exercitar os conceitos de Orientação a Objetos aprendidos em aula ao implementar um software para gerenciar competições esportivas. O software permite cadastrar diversas competições; cada competição envolve várias equipes e várias modalidades esportivas.

## 1 Introdução

Deseja-se criar um software para gerenciar competições esportivas (por exemplo, o InterUSP, o Engenharíadas e a Copa Zeus). Para este primeiro EP o software deverá permitir a criação de competições envolvendo diversas equipes e com a realização de diferentes modalidades esportivas (como, por exemplo, futebol, basquete e natação). A classificação da equipe em uma modalidade rende pontos que serão somados em uma tabela de pontos da competição. A equipe com maior número de pontos nessa tabela é a campeã da competição.

Este projeto será desenvolvido incrementalmente e em dupla nos dois Exercícios Programas de PCS3111.

A solução deve empregar adequadamente conceitos de Orientação a Objetos apresentados na disciplina: classe, objeto, atributo, método, encapsulamento, construtor e destrutor – o que representa o conteúdo até, *inclusive*, a Aula 5. A qualidade do código também será avaliada (nome de atributos/métodos, nome das classes, duplicação de código etc.).

## 2. Projeto

Deve-se implementar em C++ as classes **Competicao**, **Equipe**, **Modalidade** e **Tabela**, além de criar um `main` que permita o funcionamento do programa como desejado.

Cada uma das classes deve ter um arquivo de definição (".h") e um arquivo de implementação (".cpp"). Os arquivos devem ter exatamente o nome da classe. Por exemplo, deve-se ter os arquivos "Equipe.cpp" e "Equipe.h". Note que você deve criar os arquivos necessários. Não se esqueça de configurar o Code::Blocks para o uso do C++11 (veja a apresentação da Aula 03 para mais detalhes).

**Atenção:**

- O nome das classes e a assinatura dos métodos **devem seguir exatamente** o especificado neste documento. As classes não devem possuir outros membros (atributos ou métodos) **públicos** além dos especificados. Note que você poderá definir atributos e métodos privados, conforme necessário.
- Não é permitida a criação de outras classes além dessas.
- Não faça `#define` para constantes. Você pode (e deve) fazer `#define` para permitir a inclusão adequada de arquivos.

O não atendimento a esses pontos pode resultar erro de compilação na correção e, portanto, **nota `0` na correção automática**.

## 2.1 Classe Equipe

Uma **Equipe** representa uma equipe participante de uma competição. Cada **Equipe** tem um nome, como por exemplo, Poli, FEA e ESALQ.

A classe **Equipe** deve possuir os seguintes métodos **públicos**:

```
Equipe(string nome);
~Equipe();

string getNome();
void imprimir();
```

O método `getNome` deve retornar o nome da equipe informado pelo construtor[1].

O método `imprimir` deve jogar na saída padrão (`cout`) os dados da **Equipe** no seguinte formato (o texto entre "<" e ">" representa o valor):

```
Equipe <nome>
```

Por exemplo, a chamada do método `imprimir` para a **Equipe** com nome *Poli* seria:

```
Equipe Poli
```

## 2.2 Classe Modalidade

Uma **Modalidade** é uma modalidade esportiva realizada durante uma competição. Cada **Modalidade** possui um nome e diversas **Equipes** participantes. O resultado da **Modalidade** define a ordem de colocação obtida pelos participantes (por exemplo, 1º, 2º e 3º lugar). A seguir são apresentados os métodos **públicos** dessa classe:

[1] O método cujo nome é igual ao nome da classe é o *construtor*; o método cujo nome é o nome da classe precedido por um "~" é o *destrutor*. Esses conceitos serão apresentados na Aula 5.

```
Modalidade(string nome, Equipe** participantes, int quantidade);
~Modalidade();

string getNome();
Equipe** getEquipes();
int getQuantidadeDeEquipes();

void setResultado(Equipe** ordem);
int getPosicao (Equipe* participante);

void imprimir();
```

O construtor deve receber o nome, as **Equipes** participantes (`participantes`) e a quantidade de **Equipes** (`quantidade`). Essas informações devem ser retornadas pelos métodos `getNome`, `getEquipes` e `getQuantidadeDeEquipes`, respectivamente. Não destrua as **Equipes** no destrutor.

O método `setResultado` recebe a ordem de colocação das **Equipes** participantes e deve armazená-la. A **Equipe** na posição 0 do vetor é a campeã (1ª colocada); a **Equipe** na posição 1 do vetor é a 2ª colocada; e assim por diante. Considere que o vetor `ordem` passado como parâmetro possui a mesma quantidade de **Equipes** informada no construtor.

O método `getPosicao` deve retornar a colocação obtida pelo participante passado como parâmetro: 1 para o primeiro colocado, 2 para o segundo, 3 para o terceiro e assim por diante. Caso ainda não tenha sido definido o resultado da **Modalidade**, esse método deve retornar -1. O mesmo valor deve ser retornado caso a **Equipe** passada como parâmetro não seja um participante da competição.

O método `imprimir` deve jogar na saída padrão (`cout`) os dados da **Modalidade**. Caso ainda não tenha sido definido o resultado, o formato deve ser (os textos entre "<" e ">" representam valores):

```
Modalidade: <nome>
Participantes:
	<nome do participante 1>
	<nome do participante 2>
	...
	<nome do participante n>
```

Use um '\t' para indentar os nomes dos participantes. A ordem dos participantes deve ser a informada no construtor. Por exemplo, considere a modalidade "futebol", em que participam as equipes ESALQ, Poli e FEA. A chamada do método `imprimir` para essa **Modalidade** seria:

```
Modalidade: futebol
Participantes:
	ESALQ
	Poli
	FEA
```

Caso o resultado já tenha sido definido, o formato deve ser:

```
Modalidade: <nome>
Resultado:
	1o <nome do participante 1>
	2o <nome do participante 2>
	...
	no <nome do participante n>
```

Por exemplo, na modalidade "futebol", considerando que o resultado foi Poli em 1º, FEA em 2º e ESALQ em 3º, a saída seria:

```
Modalidade: futebol
Resultado:
	1o Poli
	2o FEA
	3o ESALQ
```

Use a letra "o" para o indicador do número ordinal (**não** use caracteres especiais).

## 2.3 Classe Tabela

Uma **Tabela** armazena a pontuação obtida pelas **Equipes** em uma **Competicao**. A seguir são apresentados os métodos **públicos** dessa classe:

```
Tabela(Equipe** participantes, int quantidade);
~Tabela();

void pontuar (Equipe* participante, int pontos);
int getPontos (Equipe* participante);
int getPosicao (Equipe* participante);

void imprimir();
```

O construtor da **Tabela** deve receber as **Equipes** participantes e a quantidade delas. Não destrua as **Equipes** no destrutor.

O método pontuar deve adicionar aos pontos atuais da **Equipe** informada como parâmetro (`participante`) o valor do parâmetro `pontos`. Inicialmente a pontuação das **Equipes** deve ser 0. Caso a **Equipe** informada não seja um participante, o método não deve fazer nada.

**Dica**

Sugere-se usar um vetor de inteiros para armazenar a pontuação[2]: o vetor teria como tamanho a quantidade de participantes e a posição da pontuação de uma **Equipe** ficaria na mesma posição que a **Equipe** está no vetor de participantes. Por exemplo, considere uma **Tabela** com três **Equipes** informadas no construtor na seguinte ordem: ESALQ, Poli e FEA. Inicialmente os valores serão [0, 0, 0]. Ao chamar o método pontuar com a **Equipe** Poli e 13 pontos, a pontuação ficará [0, 13, 0]. Se for chamado o método pontuar com a **Equipe** ESALQ e 8 pontos, a pontuação então ficará [8, 13, 0]. Caso seja chamado pontuar com a **Equipe** Poli e 10 pontos, a pontuação então será [8, 23, 0] (ou seja, será somado 10 à pontuação até então, de 13).

O método `getPontos` deve retornar a pontuação da **Equipe** passada como parâmetro. Caso a **Equipe** passada como parâmetro não seja um participante, o método deve retornar -1.

[2] A solução mais elegante seria o uso de um *dicionário*, que é implementado usando uma Tabela Hash (assunto visto em PCS 3110).

O método `getPosicao` deve informar a ordem (posição) da **Equipe** na **Tabela** ao considerar o número de pontos. O método deve retornar 1 para a **Equipe** com o maior número de pontos, 2 para a **Equipe** com a segunda maior pontuação, 3 para a **Equipe** com a terceira maior pontuação e assim por diante. Caso diversas **Equipes** estejam com a mesma pontuação, deve-se considerar que as **Equipes** estão empatadas na mesma posição – que corresponde à menor posição. Por exemplo, considere as **Equipes** ESALQ, Poli e FEA. Caso a pontuação seja [8, 23, 0], o método `getPosicao` deve retornar 1 para a Poli, 2 para a ESALQ e 3 para a FEA. Caso a pontuação seja [8, 23, 23], o método `getPosicao` deve retornar 1 para a Poli, 1 para a FEA e 3 para a ESALQ. Com a pontuação [8, 23, 8], o método `getPosicao` deve retornar 1 para a Poli, 2 para a FEA e 2 para a ESALQ.

O método `imprimir` deve jogar na saída padrão (`cout`) os dados da **Tabela** no seguinte formato (os textos entre "<" e ">" representam valores):

```
<nome da Equipe 1> - <pontos da Equipe 1> pontos (<posição da Equipe 1>o)
<nome da Equipe 2> - <pontos da Equipe 2> pontos (<posição da Equipe 2>o)
...
<nome da Equipe n> - <pontos da Equipe n> pontos (<posição da Equipe n>o)
```

A ordem de apresentação das **Equipes** deve ser a ordem dos participantes no vetor passado no construtor (para simplificar). Por exemplo, para o vetor [ESALQ, Poli, FEA] e a pontuação [8, 23, 0], a impressão seria:

```
ESALQ - 8 pontos (2o)
Poli - 23 pontos (1o)
FEA - 0 pontos (3o)
```

Use "pontos" no plural mesmo se a Equipe possuir somente 1 ponto.

## 2.4 Classe Competicao

Uma **Competicao** é uma disputa realizada entre diversas **Equipes** em diferentes **Modalidades**. Cada **Competicao** possui um nome, diversas **Equipes** participantes e podem ser adicionadas diversas modalidades. A seguir são apresentados os métodos **públicos** dessa classe:

```
Competicao(string nome, Equipe** equipes, int quantidade, int maximoDeModalidades);
~Competicao();

string getNome();
Equipe** getEquipes();
int getQuantidadeDeEquipes();

bool adicionar(Modalidade* m);
Modalidade** getModalidades();
int getQuantidadeDeModalidades();

Tabela* getTabela();

void imprimir();
```

O construtor recebe o nome, o vetor de **Equipes**, a quantidade de **Equipes** e o número máximo de modalidades que podem ser adicionadas à **Competicao**. No destrutor você pode destruir todas as modalidades que foram adicionadas.

Os métodos `getNome`, `getEquipes` e `getQuantidadeDeEquipes` devem retornar o nome, o vetor de **Equipes** e a quantidade de **Equipes** conforme informados no construtor.

O método `adicionar` deve adicionar uma **Modalidade** à **Competicao**. Caso não seja possível adicionar, por já existirem o máximo de modalidades (informado no construtor) adicionados, esse método deve retornar *false*. Caso seja possível adicionar, o método deve retornar *true*. O vetor com as modalidades adicionadas deve ser retornado pelo método `getModalidades` e a quantidade de elementos desse vetor deve ser retornada pelo método `getQuantidadeDeModalidades`.

O método `getTabela` deve retornar um objeto **Tabela** com as pontuações atuais (no momento da chamada do método) das **Equipes**. Para isso deve-se acumular na **Tabela** a pontuação obtida por cada equipe em cada uma das **Modalidades** da **Competicao**. A pontuação por colocação é a apresentada abaixo:

1º lugar: 13 pontos
2º lugar: 10 pontos
3º lugar: 8 pontos
4º lugar: 7 pontos
5º lugar: 5 pontos
6º lugar: 4 pontos
7º lugar: 3 pontos
8º lugar: 2 pontos
9º lugar: 1 ponto
10º lugar em diante: 0 pontos

Não deve ser atribuído pontos para **Modalidades** sem resultado. **Equipes** que não participam da **Modalidade** também não devem ter pontuação atribuída. Para simplificar, crie uma nova **Tabela** a cada vez que o método `getTabela` for chamado. Portanto a **Tabela** deve representar os valores da pontuação no momento da chamada do método. Por exemplo, suponha uma competição com 3 **Equipes**: ESALQ, Poli e FEA. Essa competição possui 2 **Modalidades**: futebol e basquete. Suponha que a Modalidade basquete ainda não teve resultado e a ordem das **Equipes** no futebol foi [Poli, FEA, ESALQ]. Com isso, a tabela obtida pelo método `getTabela` deve possuir os valores ESALQ: 8, Poli: 13 e FEA: 10. Caso a modalidade basquete tenha como resultado [ESALQ, Poli] (ou seja, a FEA não participou), ao chamar novamente o método `getTabela` deve-se obter os valores ESALQ: 21, Poli: 23 e FEA: 10.

O método `imprimir` deve jogar na saída padrão (`cout`) os dados da **Competicao** no seguinte formato (os textos entre "<" e ">" representam valores):

```
<Nome da competição>
<Chamada do método imprimir da Tabela>
```

Por exemplo, para a competição InterUSP, com uma tabela com ESALQ: 21, Poli: 23 e FEA: 10, a impressão deve ser:

```
InterUSP
ESALQ - 21 pontos (2o)
Poli - 23 pontos (1o)
FEA - 10 pontos (3o)
```

## 3 Interface com o usuário

Coloque o main em um arquivo em separado, chamado `main.cpp`. O comportamento do main é apresentado esquematicamente no quadro abaixo. Entre "<" e "> são apresentadas meta-informações: em **vermelho** as entradas do usuário; em **verde** os valores que dependem da quantidade de equipes (n) ou da quantidade de modalidades (m); as reticências (...) representam informações que devem ser repetidas dependendo da quantidade de equipes ou de modalidades. Ao final deve ser chamado o método `imprimir` do objeto competição.

```
Informe o nome da competicao: <nome da equipe>

Informe a quantidade de equipes: <quantidade de equipes (n)>
Informe o nome da equipe 1: <nome da equipe 1>
Informe o nome da equipe 2: <nome da equipe 2>
...
Informe o nome da equipe <n>: <nome da equipe n>

Informe a quantidade de modalidades: <quantidade de modalidades (m)>
Informe o nome da modalidade 1: <nome da modalidade 1>
Informe a equipe 1a colocada: <número de 1 a quantidade de equipes>
Informe a equipe 2a colocada: <número de 1 a quantidade de equipes>
...
Informe a equipe <N>a colocada: <número de 1 a quantidade de equipes>

Informe o nome da modalidade 2: <nome da modalidade 1>
Informe a equipe 1a colocada: <número de 1 a quantidade de equipes>
Informe a equipe 2a colocada: <número de 1 a quantidade de equipes>
...
Informe a equipe <N>a colocada: <número de 1 a quantidade de equipes>

...

Informe o nome da modalidade <M>: <nome da modalidade 1>
Informe a equipe 1a colocada: <número de 1 a quantidade de equipes>
Informe a equipe 2a colocada: <número de 1 a quantidade de equipes>
...
Informe a equipe <N>a colocada: <número de 1 a quantidade de equipes>

<Chamada do método imprimir do objeto competição>
```

**Atenção:** a saída do método `imprimir` deve seguir exatamente o especificado no enunciado.

Não se preocupe com erros de digitação: considere que o usuário sempre digita informações corretas. Por exemplo, não verifique se o usuário digitou um número de **Equipe** válida ou se informou a mesma **Equipe** repetidamente ao informar a colocação em uma **Modalidade**. Também considere que todas as **Equipes** participam de todas as **Modalidades**. Por simplicidade, considere que os nomes da **Competição**, das **Equipes** e das **Modalidades** **não possuem espaço**.

Um exemplo de saída é apresentado a seguir; em **vermelho** são apresentados exemplos de entradas.

```
Informe o nome da competicao: InterUSP

Informe a quantidade de equipes: 3
Informe o nome da equipe 1: Poli
Informe o nome da equipe 2: FEA
Informe o nome da equipe 3: ESALQ

Informe a quantidade de modalidades: 4
Informe o nome da modalidade 1: Futebol
Informe a equipe 1a colocada: 1
Informe a equipe 2a colocada: 2
Informe a equipe 3a colocada: 3

Informe o nome da modalidade 2: Natacao
Informe a equipe 1a colocada: 3
Informe a equipe 2a colocada: 1
Informe a equipe 3a colocada: 2

Informe o nome da modalidade 3: Volei
Informe a equipe 1a colocada: 1
Informe a equipe 2a colocada: 3
Informe a equipe 3a colocada: 2

Informe o nome da modalidade 4: Atletismo
Informe a equipe 1a colocada: 2
Informe a equipe 2a colocada: 3
Informe a equipe 3a colocada: 1

InterUSP
Poli – 44 pontos (1o)
FEA – 39 pontos (3o)
ESALQ – 41 pontos (2o)
```

# 4 Entrega

O projeto deverá ser entregue até dia **04/10** em um Judge diferente, disponível em <http://judge.pcs.usp.br/pcs3111/ep/> (nos próximos dias vocês receberão um login e uma senha). **As duplas devem ser formadas por alunos da mesma turma e elas devem ser informadas no e-Disciplinas até a data de entrega do EP**. Caso não seja informada a dupla, será considerado que o aluno está fazendo o EP sozinho.

> **Atenção:** não copie código de um outro grupo. Qualquer tipo de cópia será considerada plágio e os grupos envolvidos terão **nota 0 no EP**. Portanto, **não envie** o seu código para um colega de outro grupo!

Entregue todos os arquivos, inclusive o `main` (que deve **obrigatoriamente** ficar em um arquivo "main.cpp"), em um arquivo comprimido no formato ZIP (outros formatos, como RAR e 7Z, *podem* não ser reconhecidos e acarretar **nota 0**). Os códigos fonte não devem ser colocados em pastas.

> **Atenção:** faça a submissão do mesmo arquivo nos 2 problemas (Parte 1 e Parte 2). Isso é necessário por uma limitação do Judge. Caso isso não seja feito, parte do seu EP não será corrigido – impactando a nota.

Siga a convenção de nomes para os arquivos ".h" e ".cpp". O não atendimento disso pode levar a erros de compilação (e, consequentemente, **nota zero**).

Ao submeter os arquivos no Judge será feita **apenas** uma verificação básica de modo a evitar erros de digitação no nome das classes e dos métodos públicos. **Você poderá submeter quantas vezes você desejar, sem desconto de nota**. Mas note que a nota dada **não é a nota final**: nesse momento não são executados testes – o Judge apenas tenta chamar todos os métodos definidos neste documento para todas as classes.

# 5 Dicas

- Implemente a solução aos poucos – não deixe para implementar tudo no final.
- Para testar o programa faça o `main` chamar uma função de teste que cria objetos com valores interessantes para testar, sem pedir entrada para o usuário. Não se esqueça de remover a função de teste ao entregar a versão final do EP.
- Caso o método `setResultado` em **Modalidade** apenas armazene a *referência ao vetor* (o que é o mais simples), cuidado com o reuso no `main` do mesmo vetor para informar resultados em diferentes modalidades. Note que nesse caso todas as **Modalidades** apontariam para o mesmo vetor de resultados. Para evitar isso, crie um vetor para o resultado de cada modalidade.
- Submeta no Judge o código com antecedência para descobrir problemas na sua implementação. É normal acontecerem *RuntimeErrors* e outros tipos de erros.
- Use o "Fórum de dúvidas do EP" para esclarecer dúvidas no enunciado ou problemas de submissão no Judge.
- **Evite submeter nos últimos minutos do prazo de entrega.** É normal o Judge ficar sobrecarregado com várias submissões e demorar para compilar.

# 6 Testes do Judge

Ao submeter o Judge só testará se as classes **possuem** todos os métodos especificados. Ele **não** testará se os métodos são corretos. **Você poderá submeter quantas vezes você desejar, sem desconto de nota**. Após o fim do prazo, os seguintes testes serão executados:

Parte 1 – classes Equipe, Modalidade e Tabela

- Equipe: Construtor, destrutor e getNome
- Equipe: imprimir
- Modalidade: construtor, destrutor, getNome, getEquipe e getQuantidade
- Modalidade: getPosicao sem resultado
- Modalidade: getPosicao com chamada de setResultado e 2 equipes
- Modalidade: getPosicao não participante
- Modalidade: getPosicao com chamada de setResultado e 4 equipes
- Modalidade: getResultado com chamada de setResultado 2 vezes
- Modalidade: imprimir sem resultado
- Tabela: Construtor e destrutor
- Tabela: getPontos sem pontuar
- Tabela: getPontos não participante
- Tabela: getPontos com pontuar sem acumular
- Tabela: getPontos com pontuar acumulando

- Tabela: getPosicao sem pontos
- Tabela: getPosicao com pontos
- Tabela: getPosicao com empate no início
- Tabela: getPosicao com empate no meio
- Tabela: getPosicao com empate no fim
- Tabela: imprimir

Parte 2 – classe Competicao

- Competicao: construtor, destrutor, getNome, getEquipe e getQuantidade
- Competicao: adicionar uma modalidade
- Competicao: adicionar várias modalidades até limite
- Competicao: adicionar modalidades além limite
- Competicao: getTabela com modalidades sem resultado
- Competicao: getTabela com algumas modalidades sem resultado
- Competicao: getTabela com todas modalidades com resultado