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

PCS 3111 - LABORATÓRIO DE PROGRAMAÇÃO ORIENTADA A OBJETOS PARA A ENGENHARIA ELÉTRICA

EXERCÍCIO PROGRAMA 1 – 2º SEMESTRE DE 2018

Resumo

Os EPs de PCS3111 têm como objetivo exercitar os conceitos de Orientação a Objetos aprendidos em aula ao implementar um software de gerenciamento de projetos de Engenharia. Esse software deve permitir o cadastro de atividades e pessoas em um projeto, permitindo calcular o custo e a duração estimada de cada atividade e do projeto como um todo.

1 Introdução

Deseja-se criar um software de gerenciamento de projetos *s*imples, no estilo do Trello (https://trello.com) – mas sem recursos gráficos. Para este primeiro EP o software deverá permitir a criação de atividades, as quais permitem calcular as estimativas de custo e de duração baseadas nas pessoas necessárias. As atividades são adicionadas a um projeto, o qual também calcula estimativas de custo e de duração. Por simplicidade, o número de atividades no projeto e de pessoas em uma atividade será limitado neste EP.

Este projeto será desenvolvido incrementalmente e em dupla nos dois Exercícios Programas de PCS 3111.

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.).

Sugere-se usar o EP1 para o estudo para a P1. Para isso veja a seção *Dicas* que explica como evitar os conceitos não ainda não aprendidos (conceitos da Aula 05).

2. Projeto

Deve-se implementar em C++ as classes **Atividade**, **Pessoa** e **Projeto**, 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 "Atividade.cpp" e "Atividade.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 outros `#define` para constantes além dos definidos neste documento. 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 Pessoa

Uma **Pessoa** representa um trabalhador que executa parte ou toda uma atividade dentro de um projeto. Cada **Pessoa** tem um nome, um valor que ela recebe por hora trabalhada (*valorPorHora*) e o número de horas que ela tem disponível por dia (*horasDiarias*).

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

```
Pessoa(string nome, double valorPorHora, int horasDiarias);
~Pessoa();

string getNome();
double getValorPorHora();
int getHorasDiarias();
double getCusto(int dias);

void imprimir();
```

Os métodos `getNome`, `getValorPorHora` e `getHorasDiarias` devem retornar, respectivamente, os valores do `nome`, `valorPorHora` e `horasDiarias` informados pelo construtor (conceito da Aula 05).

O método `getCusto` deve calcular o custo da **Pessoa** trabalhar um determinado número de dias (parâmetro *dias*). O custo deve ser calculado como:

$$custo = número\ de\ dias * número\ de\ horas\ disponíveis\ por\ dia * \ valor\ por\ hora\ trabalhada$$

Por exemplo, para uma **Pessoa** com nome Maria que recebe R$ 12 por hora e tem disponível 4 horas por dia, o custo dela trabalhar 2 dias será $2 * 4 * 12 = R\$\ 96$.

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

```
<nome> - R$<valor por hora> - <horas disponíveis por dia>h por dia
```

Por exemplo, a chamada do método `imprimir` para a **Pessoa** apresentada anteriormente seria:

```
Maria - R$12 - 4h por dia
```

2.2 Classe Atividade

Uma **Atividade** é um trabalho a ser executado por uma ou mais pessoas dentro de um projeto. Toda **Atividade** possui um nome e uma estimativa de horas necessárias para ser terminada. A seguir são apresentados os métodos **públicos** dessa classe e a constante a ser definida como um `#define`:

```
#define MAXIMO_RECURSOS 10

Atividade(string nome, int horasNecessarias);
~Atividade();

string getNome();
int getHorasNecessarias();

bool adicionar(Pessoa* recurso);
Pessoa** getPessoas();
int getQuantidadeDePessoas();

int getDuracao();

double getCusto();

void imprimir();
```

O construtor (conceito apresentado na Aula 05) deve receber o nome e a estimativa do número de horas necessárias para terminar a atividade (`horasNecessarias`). Essas informações devem ser retornadas pelos métodos `getNome` e `getHorasNecessarias`, respectivamente.

O método `adicionar` deve permitir adicionar uma **Pessoa** na **Atividade**, informando que ela trabalhará na **Atividade**. Deve ser possível adicionar até no máximo `MAXIMO_RECURSOS` **Pessoas** à uma **Atividade**. Caso não seja possível adicionar mais uma **Pessoa**, por existirem mais de `MAXIMO_RECURSOS` adicionadas, o método `adicionar` deve retornar `false`, ignorando a **Pessoa** passada. O método também deve retornar `false` caso aquele objeto **Pessoa** já tenha sido adicionado previamente – e nesse caso a **Pessoa** não deve ser adicionada novamente. Caso a **Pessoa** seja corretamente adicionada, o método deve retornar `true`.

O método `getPessoas` deve retornar um vetor com todas as **Pessoas** adicionadas à atividade. O método `getQuantidadeDePessoas` deve retornar o número de **Pessoas** adicionadas, ou seja, a quantidade de **Pessoas** no vetor retornado em `getPessoas`.

A duração em dias estimada para a atividade deve ser retornada pelo método `getDuracao`. A duração deve ser calculada como:

$$duracao = \left\lceil \frac{horas\ necessárias\ para\ terminar\ a\ atividade}{\sum horas\ por\ dia\ de\ cada\ pessoa\ adicionada} \right\rceil$$

Use a função `ceil` para calcular o teto (arredondamento para cima). Para isso faça include da biblioteca `cmath` e faça "`using namespace std;`". Caso a **Atividade** não tenha **Pessoas** adicionadas, esse método deve retornar -1. Como exemplo de cálculo, considere que Maria e João foram adicionados à **Atividade** "Teste" e, portanto, trabalharão nela. Maria tem disponível 4 horas diárias; João tem disponível 5 horas. A atividade "Teste" necessita de 10h de trabalho. Portanto a duração será $\lceil 10 / (4 + 5) \rceil = \lceil 1{,}111 \rceil = 2$ dias.

O método `getCusto` deve retornar a estimativa de custo para realizar a **Atividade**. O custo deve ser calculado como:

$$custo = \sum duração * custo\ diário\ por\ pessoa$$

Caso a **Atividade** não tenha **Pessoas** adicionadas, esse método deve retornar -1. Use o método `getCusto(int dias)` em **Pessoa** para calcular o custo diário para cada **Pessoa**. Por exemplo, a **Atividade** "Teste" apresentada anteriormente terá duração de 2 dias. Maria e João trabalharão na atividade; Maria recebe R$ 12 por hora e João R$ 10, mas Maria tem disponível 4 horas por dia e João 5 horas. Com isso o custo será: $(2 * 4 * 12) + (2 * 5 * 10) = R\$196$.

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

```
<nome> - <duração> dias – R$<custo>
```

Por exemplo, a chamada do método `imprimir` para a **Atividade** "Teste" apresentada anteriormente seria:

```
Teste - 2 dias - R$196
```

2.3 Classe Projeto

Um **Projeto** é um "esforço temporário empreendido para criar um produto, serviço ou resultado exclusivo"[1]. Neste software, um **Projeto** é representado através de seu nome, as **Atividades** que devem ser executadas e os recursos necessários – no caso, as **Pessoas**. A seguir são apresentados os métodos **públicos** dessa classe e a constante a ser definida como um `#define`:

```
#define MAXIMO_ATIVIDADES 10

Projeto(string nome);
~Projeto();

string getNome();

bool adicionar(Atividade* a);
Atividade** getAtividades();
int getQuantidadeDeAtividades();

bool adicionarRecurso(Pessoa* p);
Pessoa** getPessoas();
int getQuantidadeDePessoas();

int getDuracao();
double getCusto();

void imprimir();
```

O nome do **Projeto** deve ser informado no construtor (conceito apresentado na Aula 05) e retornado pelo método `getNome`.

[1] PMI. *Um Guia do Conhecimento em Gerenciamento de Projetos (Guia PMBOK)*. PMI, 5ª edição, 2013, p.3.

O método `adicionar` deve adicionar as **Atividades** que devem ser executadas no **Projeto**. Deve ser possível adicionar até no máximo `MAXIMO_ATIVIDADES` **Atividades** à um **Projeto**. Caso não seja possível adicionar mais uma **Atividade**, por existirem mais de `MAXIMO_ATIVIDADES` adicionadas, o método `adicionar` deve retornar `false`, ignorando a **Atividade** passada. Caso a **Atividade** seja corretamente adicionada, o método deve retornar `true` (note que deve ser possível adicionar várias vezes a mesma **Atividade**). O método `getAtividades` deve retornar o vetor com as **Atividades** adicionadas e o método `getQuantidadeDeAtividades` deve retornar o número de **Atividades** adicionadas (a quantidade de **Atividades** no vetor).

Assim como se adiciona **Atividades**, deve ser possível adicionar **Pessoas** a um **Projeto**. Isso é feito pelo método `adicionarRecurso`. Deve ser possível adicionar até no máximo `MAXIMO_RECURSOS` **Pessoas** à um **Projeto**. O funcionamento desse método deve ser similar ao método na classe **Atividade**, ou seja, caso não seja possível adicionar mais uma **Pessoa**, por existirem mais de `MAXIMO_RECURSOS` adicionadas, o método `adicionarRecurso` deve retornar `false`, ignorando a **Pessoa** passada. O método também deve retornar `false` caso a **Pessoa** já tenha sido adicionada previamente – e nesse caso a **Pessoa** não deve ser adicionada novamente. Caso a **Pessoa** seja corretamente adicionada, o método deve retornar `true`. O método `getPessoas` deve retornar o vetor de **Pessoas** adicionadas e o método `getQuantidadeDePessoas` deve retornar a quantidade de **Pessoas** no vetor.

O método **getDuracao** deve retornar a duração em dias do **Projeto**. Ela deve ser calculada como:

$$duração\ do\ projeto = \sum duração\ de\ cada\ atividade$$

Por exemplo, caso o **Projeto** "EP1" tenha duas **Atividades**, "Implementação" e "Teste", as quais duram, respectivamente, 3 dias e 2 dias, a duração do projeto deve ser $3 + 2 = 5\ dias$[2].

O método `getCusto` deve calcular o custo do **Projeto**. Ele deve ser calculado como:

$$custo\ do\ projeto = \sum custo\ de\ cada\ atividade$$

Por exemplo, no **Projeto** "EP1", as atividades "Implementação" e "Teste" custam, respectivamente, R$294 e R$196. Portanto o custo do **Projeto** será $294 + 196 = R\$490$.

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

```
<nome> - <duração> dias – R$<custo>
----
<impressão da atividade 1>
<impressão da atividade 2>
...
<impressão da atividade n>
```

Por exemplo, a chamada do método `imprimir` para o **Projeto** "EP1" apresentado anteriormente seria:

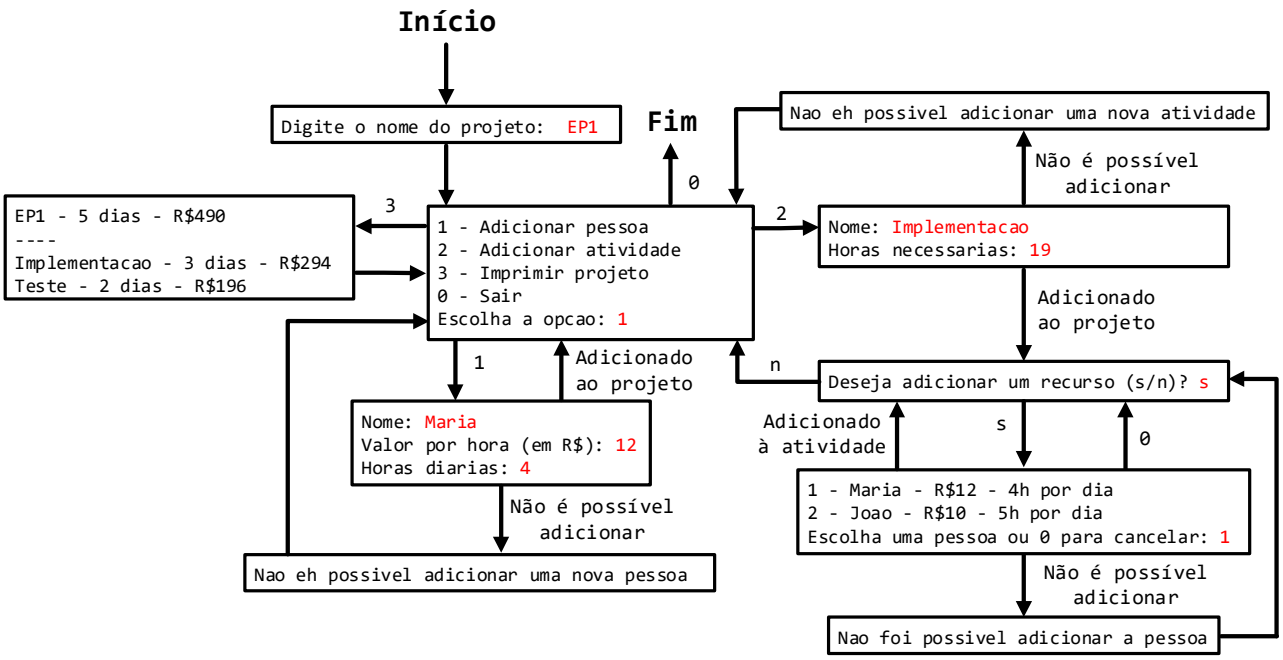
```
EP1 - 5 dias - R$490
----
Implementação - 3 dias - R$294
Teste - 2 dias - R$196
```

[2] Isso é só um exemplo: o EP1 demorará muito menos do que isso para ser implementado e testado!

3 Interface com o usuário

Coloque o main em um arquivo em separado, chamado `main.cpp`. Ela é apresentada esquematicamente no diagrama abaixo. Cada retângulo representa uma "tela", ou seja, o conjunto de informações apresentadas e solicitadas. As setas representam as transições de uma tela para outra – os textos na seta representam o valor que deve ser digitado para ir para a tela destino ou a condição necessária (quando não há um texto é porque a transição acontece incondicionalmente). Em **vermelho** são apresentados exemplos de dados inseridos pelo usuário.

> **Atenção:** A interface com o usuário deve seguir exatamente a ordem definida (e exemplificada). Se a ordem não for seguida, haverá desconto de nota.



Alguns detalhes:

- Use o método `getPessoas` do **Projeto** para listar as **Pessoas** disponíveis no **Projeto**. Apresente sempre todas as **Pessoas** adicionadas ao **Projeto** para adicionar uma **Pessoa** à uma **Atividade**, mesmo que já adicionadas à **Atividade**.
- No diagrama não são apresentados casos de erro de digitação (por exemplo, uma opção inválida no menu inicial). Não é necessário fazer tratamento disso. Assuma que o usuário *sempre* digitará um valor válido.
- Assuma que todos os nomes (e strings em geral) não contém espaços. Isso facilita o processamento da string.

4 Entrega

O projeto deverá ser entregue até dia **03/10** em um Judge específico, 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 3 problemas (Parte 1, Parte 2 e Parte 3). 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**: não são executados testes – o Judge apenas tenta chamar todos os métodos definidos neste documento para todas as classes.

5 Dicas

- Separe o `main` em várias funções para organizar o código.
- Implemente a solução aos poucos – não deixe para implementar tudo no final.
- É muito trabalhoso testar o programa ao executar o `main` *com menus*, já que é necessário informar vários dados para inicializar o **Projeto**. Para testar, crie um `main` mais simples, que cria os objetos do jeito que você quer testar. Só não se esqueça de entregar o `main` correto!
- 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.

5.1 Usando o EP1 para estudar para a P1

O EP1 cobre conceitos da Aula 05, os quais não serão abordados na P1. Para usar o EP1 para estudar para a P1, *ignore* os métodos que tem o mesmo nome da classe (são os construtores e destrutores), ou seja:

- `Atividade(string nome, int horasNecessarias);`
- `~Atividade();`

- `Pessoa(string nome, double valorPorHora, int horasDiarias);`
- `~Pessoa();`
- `Projeto(string nome);`
- `~Projeto();`

Comente esses métodos no ".h" e não os implemente no ".cpp". Além disso, para usar as classes, crie os seguintes métodos *setters*:

- Classe `Atividade`
  - `void setNome(string nome);`
  - `void setHorasNecessarias(int horas);`
- Classe `Pessoa`
  - `void setNome(string nome);`
  - `void setValorPorHora(double valor);`
  - `void setHorasDiarias(int horas);`
- Classe `Projeto`
  - `void setNome(string nome);`

Após assistir a Aula 05, descomente no ".h" os métodos com o mesmo nome da classe (construtores e destrutores) e os implemente. Além disso, remova (do ".h" e do ".cpp") os métodos *setters* adicionados.

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 – classe Pessoa

- Construtor e destrutor
- getNome getValorPorHora e getHorasDiarias
- getCusto várias chamadas
- getCusto valor com decimal várias chamadas
- imprimir

Parte 2 – classe Atividade

- Construtor e destrutor
- getNome e getHorasNecessarias
- getQuantidadeDePessoas para 0 pessoas
- adicionar e getPessoas para 1 pessoa
- adicionar e getPessoas para 3 pessoas
- adicionar até MAXIMO_RECURSOS
- adicionar repetidos
- getDuracao sem pessoas
- getDuracao com 1 pessoa
- getDuracao com várias pessoas sem arredondamento
- getDuracao com várias pessoas com arredondamento
- getCusto sem pessoas
- getCusto com 1 pessoa
- getCusto com várias pessoas
- getCusto com várias pessoas e decimal
- imprimir

Parte 3 – classe Projeto

- Construtor e destrutor
- getNome
- getQuantidadeDeAtividades para 0 atividades
- adicionar e getAtividades para 1 atividade
- adicionar e getAtividades para 3 atividades
- adicionar até MAXIMO_ATIVIDADES
- adicionar repetidos
- getQuantidadeDePessoas para 0 pessoas
- adicionarRecurso e getPessoas para 1 pessoa
- adicionarRecurso e getPessoas para 3 pessoas
- adicionarRecurso até MAXIMO_RECURSOS
- adicionarRecurso repetidos
- getDuracao sem atividades
- getDuracao com 1 atividade
- getDuracao com várias atividades
- getCusto sem atividades
- getCusto com 1 atividade e decimal
- getCusto com várias atividades
- imprimir