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 2024

1 Introdução

Uma startup de ex-alunos de Engenharia Elétrica da Poli está desenvolvendo uma catraca eletrônica para ser usada em prédios de escritório. Além do controle de acesso, a catraca permite aos gestores cuidarem do controle de ponto dos funcionários, o que é importante do ponto de vista da legislação trabalhista.

Cada funcionário deve possuir um identificador único (o qual pode ser colocado em um cartão, por exemplo) que é lido pela catraca. Com essa informação, a catraca permite ou não o acesso. Uma vez que é comum que os funcionários tenham problemas na entrada e saída do prédio (por exemplo, esquecendo o cartão de identificação), o sistema também deve permitir o registro manual de entradas e saídas.

O software deve permitir o cadastro de novos usuários e também emitir um relatório mensal de acesso dos funcionários.

1.1 Objetivo

O objetivo deste projeto é fazer o software para o controle de catracas eletrônicas. 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, indentação etc.).

2 Projeto

Deve-se implementar em C++ as classes `Data`, `Registro`, `Usuario`, `GerenciadorDeUsuario` e `Catraca`, além de criar uma interface com o usuário que permita o funcionamento do programa como desejado.

Atenção:

1. 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, caso necessário**.
2. Não faça outros `#defines` de constantes além dos definidos neste documento. Você pode (e deve) somente fazer `#ifndef/#define` para permitir a inclusão adequada de arquivos.

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

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 "Data.cpp" e "Data.h". Note que você deve criar os arquivos necessários.

Todas as classes possuem construtores e destrutores (conceito da *Aula 5* – são os métodos com o mesmo nome da classe), mas é possível começar a implementar os métodos ainda sem esse conceito. Coloque a palavra *virtual* nos destrutores, como indicado. Não se preocupe com isso – será explicado o significado na Aula 7.

2.1 Classe Data

A `Data` representa um instante de tempo, com dia, mês, ano, hora, minuto e segundo. Embora a biblioteca padrão do C++11 possua classes para lidar com isso, elas usam conceitos avançados que só veremos mais adiante no curso. Para evitar confusões, criaremos uma classe e usaremos funções do C (disponíveis no C++).

Essa classe deve possuir apenas os seguintes **métodos públicos**:

```
Data(int hora, int minuto, int segundo, int dia, int mes, int ano);
virtual ~Data();

int getHora();
int getMinuto();
int getSegundo();
int getDia();
int getMes();
int getAno();

int diferenca(Data* d);
```

O construtor deve receber a hora (de 0 a 23), minuto (de 0 a 59), segundo (de 0 a 59), dia (de 1 a 31), mês (de 1 a 12) e ano (com 4 dígitos) da data. Note que não é feita uma verificação dos valores. Essas informações são retornadas pelos *getters*. Se você criar algum objeto, destrua-o no destrutor.

O método `diferenca` recebe uma outra `Data` e retorna a diferença em segundos entre a data atual e a data recebida. Não se preocupe com datas inválidas. Por exemplo, considere um objeto que guarda a data 10:00:00 de 21/10/2024. A chamada de `diferenca` nesse objeto passando a data 9:00:00 de 21/10/2024 deve retornar 3600. Caso a chamada de `diferenca` nesse mesmo objeto receba 10:00:00 de 22/10/2024 (mesmo horário no dia seguinte), o retorno deve ser -86400. Para implementar essa lógica, use as estruturas[1] `tm` e `time_t` definidas no include `ctime`. A estrutura `tm` representa uma data[2], possuindo os seguintes atributos públicos:

- tm_hour: a hora;
- tm_min: o minuto;
- tm_sec: o segundo;
- tm_isdst: se é horário de verão ou não (coloque sempre 0 para evitar problemas).
- tm_mday: o dia;
- tm_mon: o mês, começando por 0 (ou seja, janeiro é 0, fevereiro é 1 e dezembro é 11); e
- tm_year: o ano a partir de 1900 (ou seja, 2024 é representado por 2024 – 1900 = 124).

Por exemplo, para representar a data 10:00:00 21/10/2024, pode-se fazer:

```
tm* data1 = new tm;
data1->tm_hour = 10;
data1->tm_min = 0;
data1->tm_sec = 0;
data1->tm_isdst = 0;
data1->tm_mday = 21;
data1->tm_mon = 10 - 1;
data1->tm_year = 2024 - 1900;
```

A vantagem de usar essa estrutura é que existe uma forma simples de fazer diferença entre datas considerando o calendário gregoriano (note, por exemplo, que a diferença entre 23:59:59 de 31/10/2024 e 0:0:0 de 1/11/2024 deve ser -1). Para isso é primeiro necessário converter o `tm` para um `time_t`, que é um número que representa a data[3]. Essa conversão é feita com a função `mktime` (que está no cabeçalho `ctime`) que recebe um `tm` e retorna um `time_t`[4]. Por exemplo, para converter a `data1` mostrada anteriormente é só fazer:

```
time_t t1 = mktime(data1);
```

Com esse `time_t` é possível usar a função (também no cabeçalho `ctime`):

[1] Uma estrutura em C++ pode ser vista como uma classe em que, por padrão, todos os membros (atributos e métodos) são públicos.

[2] Mais detalhes em <https://cplusplus.com/reference/ctime/tm/>.

[3] Mais detalhes em <https://cplusplus.com/reference/ctime/time_t/>.

[4] Essa função também corrige a data no `tm`, alterando-a. Por exemplo, se você criar uma data 32 de janeiro, ao chamar `mktime` ele corrige para 1 de fevereiro. Mais detalhes em <https://cplusplus.com/reference/ctime/mktime/>.

`double difftime(time_t inicio, time_t fim)`

que calcula a diferença em segundos entre o `inicio` e o `fim`[5]. Note que ela retorna um `double`, mas no nosso caso usamos um inteiro (faça um cast). Por exemplo, considerando que em `t2` se tem o `time_t` representando uma outra data, pode-se fazer:

```
int diferenca = (int) difftime(t1, t2);
```

2.1.1 Classe Registro

O `Registro` é um evento de entrada ou saída, manual ou não, de um funcionário em uma catraca. Essa classe deve possuir apenas os seguintes **métodos públicos**:

```
Registro(Data* d, bool entrada, bool manual);
virtual ~Registro();

Data* getData();
bool isEntrada();
bool isManual();
```

O construtor deve receber uma `Data` em que o registro aconteceu, um booleano informado se é entrada (`true`) ou saída (`false`), e um booleano informando se o registro é manual (`true`) ou não (veja na classe `Usuario` uma explicação sobre registros manuais e feitos pela `Catraca`). No destrutor destrua a data.

Os métodos `getData`, `isEntrada` e `isManual` simplesmente retornam as respectivas informações passadas no construtor.

2.2 Classe Usuario

O `Usuario` representa alguém que terá acesso ao edifício pelo sistema e terá seu registro de horas armazenado. Essa classe deve possuir os seguintes métodos públicos:

```
Usuario(int id, string nome, int maximo);
virtual ~Usuario();

string getNome();
int getId();

bool entrar(Data *d);
bool sair(Data *d);

bool registrarEntradaManual(Data *d);
bool registrarSaidaManual(Data* d);

int getHorasTrabalhadas(int mes, int ano);
Registro** getRegistros();
int getQuantidade();
```

[5] Mais detalhes em <https://cplusplus.com/reference/ctime/difftime/>.

O construtor deve receber o identificador do usuário (usado pela `Catraca`), o nome e o número máximo de registros (crie um vetor de `Registro` com esse tamanho). O destrutor deve destruir os registros criados, assim como o vetor.

Os métodos `getNome` e `getId` devem retornar as informações passadas ao construtor.

O método `entrar` deve receber uma `Data` e verificar se o usuário pode entrar no edifício. Não é permitida a entrada caso o último registro dele foi uma entrada ou se a data informada for anterior à data do último registro. O método `sair` tem lógica similar: não deve permitir a saída caso o último registro foi uma saída ou se a data for anterior à data do último registro. Note que o primeiro registro pode ser de entrada ou de saída. Esses métodos também devem retornar `false` caso não haja espaço no vetor. Os registros criados por esse método não devem ser manuais, dado que eles serão feitos pela `Catraca`.

Os métodos `registrarEntradaManual` e `registrarSaidaManual` tem o mesmo comportamento de `entrar` e `sair`, com a diferença que eles registram entradas e saídas manuais (ou seja, que serão registradas manualmente pela segurança do prédio). Esses métodos devem seguir as mesmas regras explicadas anteriormente: não permitir a entrada caso o último registro foi uma entrada (se `registrarEntradaManual`) ou saída (se `registrarSaidaManual`), se a data for anterior à data do último registro, ou não houver espaço.

O método `getRegistros` deve retornar o vetor de registros desse Usuario, enquanto que o método `getQuantidade` deve retornar a quantidade de registros adicionadas a esse vetor.

O método `getHorasTrabalhadas` deve fazer o cálculo das horas trabalhadas pelo usuário naquele mês e ano. Ou seja, ele deve somar a diferença entre as saídas e entradas naquele mês. O valor deve ser truncado, ou seja, caso a soma dê 10,5, o método deve retornar 10. Ou seja, primeiro some as diferenças e depois faça o cálculo necessário para que o resultado seja em horas. O filtro considera apenas a data de entrada, ou seja, se a chamada do método foi com 5 e 2024 (05/2024) e a entrada foi no mês 05/2024 e a saída foi no mês 06/2024, essas horas devem ser computadas; porém se a entrada foi no mês 04/2024 e a saída foi em 05/2024, essas horas não devem ser consideradas para a chamada do método com 5 e 2024. Além disso, se a entrada no mês solicitado for o último registro do usuário, ela não deve ser considerada no cálculo de horas trabalhadas (já que não há ainda uma saída registrada). Por exemplo, considere um usuário com os seguintes registros:

Entrada	Saída
09:04:30 de 30/09/2024	15:00:20 de 30/09/2024
23:00:00 de 30/09/2024	08:06:00 de 01/10/2024
09:05:30 de 02/10/2024	12:08:00 de 02/10/2024
13:10:00 de 02/10/2024	18:10:30 de 02/10/2024
09:10:30 de 03/10/2024	

A chamada `getHorasTrabalhadas(9, 2024)` deve retornar 15, enquanto que `getHorasTrabalhadas(10, 2024)` deve retornar 8.

2.3 Classe GerenciadorDeUsuario

A classe `GerenciadorDeUsuario` gerencia os usuários conhecidos pelas catracas. Essa classe deve possuir os seguintes métodos públicos:

```
GerenciadorDeUsuario(int maximo);
virtual ~GerenciadorDeUsuario();

bool adicionar(Usuario* u);
Usuario* getUsuario(int id);
Usuario** getUsuarios();
int getQuantidade();
```

O construtor recebe a quantidade máxima de usuários que podem ser armazenadas por esse gerenciador. O destrutor deve destruir os usuários e o vetor criado para armazená-los.

O método `adicionar` deve adicionar um `Usuario` ao gerenciador. Caso não haja espaço disponível ou o usuário já tenha sido adicionado (considere que usuários são iguais caso tenham ids iguais), o método deve retornar `false` e não adicionar o objeto. Caso contrário, ele deve retornar `true` e adicionar o objeto.

O método getUsuario deve retornar o objeto Usuario com o id informado que foi adicionado ao gerenciador. Caso não tenha sido adicionado um Usuario com esse id, o método deve retornar nullptr.

O método `getUsuarios` deve retornar o vetor de `Usuario` desse gerenciador, enquanto que o método `getQuantidade` deve retornar a quantidade de usuários adicionadas ao gerenciador.

2.4 Classe Catraca

A `Catraca` representa um equipamento que permite a entrada e saída de usuários conhecidos. Essa classe deve possuir os seguintes métodos públicos:

```
Catraca(GerenciadorDeUsuario* g);
virtual ~Catraca();

bool entrar(int id, Data* d);
bool sair(int id, Data* d);
```

O construtor deve receber um `GerenciadorDeUsuario`, o qual será consultado para encontrar o `Usuario` com o id informado na tentativa de entrada ou saída. Note que um sistema pode ter várias catracas e elas podem compartilhar o gerenciador. Por isso, o destrutor não deve destruir o `GerenciadorDeUsuario`.

O método `entrar` deve procurar no `GerenciadorDeUsuario` o `Usuario` com o id informado e, para o `Usuario` encontrado, chamar o método `entrar` (representando uma entrada automática) e retornar o valor que foi retornado por esse método. Caso o usuário não seja

encontrado, o método deve retornar `false`. O método sair tem comportamento similar – a única diferença é que ele chama o método `sair` do `Usuario` ao invés do `entrar`.

3 Main e menu.cpp

Coloque a `main` em um arquivo separado, chamado `main.cpp`. Nele você deverá simplesmente chamar uma função `menu`, a qual ficará no arquivo `menu.cpp`. Não faça include de menu no arquivo com o main (*jamais faça include de arquivos .cpp*). Portanto, o `main.cpp` deve ser.

```
void menu();

int main() {
    menu();
    return 0;
}
```

O menu deve criar um programa que se utiliza as classes especificadas para permitir a entrada e saída de pessoas usando a catraca, registrar a entrada e a saída manual (chamando os métodos de `Usuario` de registro manual), cadastrar usuários e gerar um relatório de horas trabalhadas pelos usuários em um determinado mês.

No menu você deve criar um `GerenciadorDeUsuario` e dois objetos `Catraca` (catraca 0 e 1), os quais usam a mesmo `GerenciadorDeUsuario`. Considere o máximo de 10 usuários e 10 registros.

3.1 Interface

A seguir é apresentado um diagrama que contém as telas da interface em console que deve ser criada. Cada retângulo representa uma "tela", ou seja, um conjunto de texto impresso com possivelmente entrada de dados. 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.

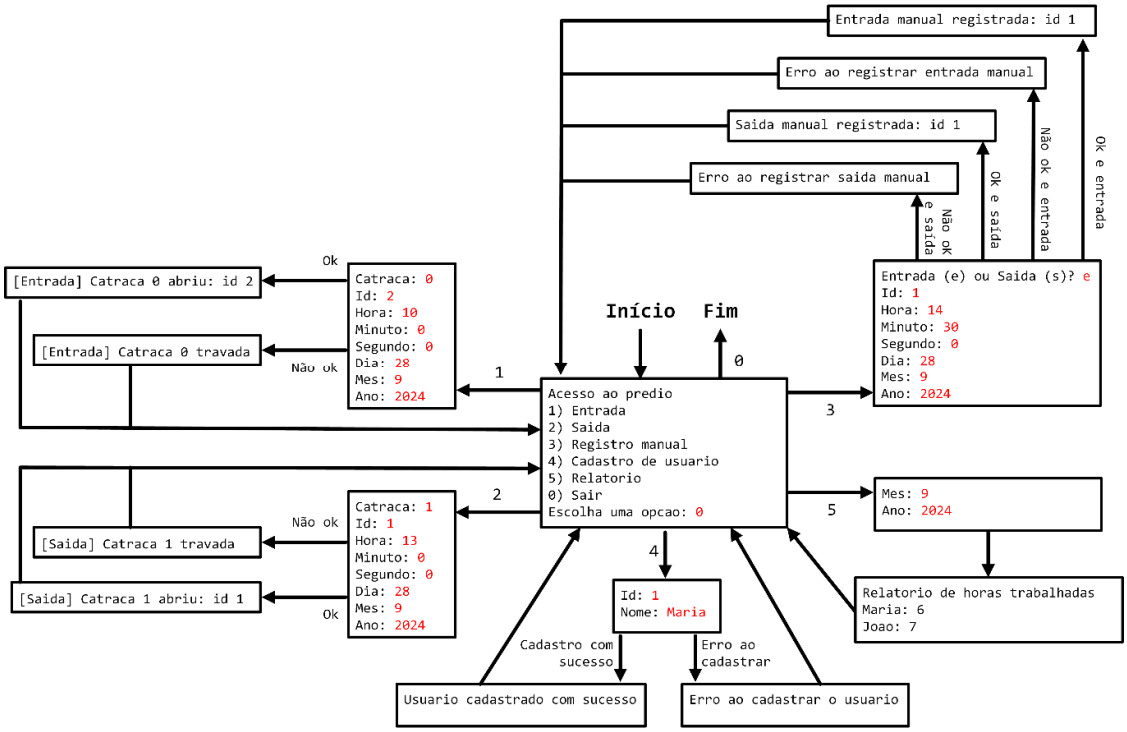
Observação: pode-se considerar que o usuário somente digitará entradas com escolhas possíveis. Ou seja, não é preciso tratar entradas incorretas.

> **Atenção:** A interface com o usuário deve seguir exatamente o especificado (incluindo ":" e espaços entre ")" e ":" e o restante do texto - os demais espaços não serão verificados). Se ela não for seguida, haverá desconto de nota. Não adicione outros textos além dos apresentados no diagrama e especificados.

As telas funcionam da seguinte forma:

- Opção 1: deve-se chamar o método entrar na `Catraca` informada com o id e `Data` informados.

- Opção 2: deve-se chamar o método sair na `Catraca` informada com o id e `Data` informados.
- Opção 3: deve-se fazer um registro manual (de entrada ou de saída, dependendo do que foi informado) no `Usuario` com id e data informados.
- Opção 4: deve-se cadastrar um novo `Usuario` com os dados informados, o qual deve ser adicionado ao `GerenciadorDeUsuario`.
- Opção 5: deve-se gerar um relatório de horas trabalhadas para **todos** os `Usuarios`, apresentando o nome e as horas trabalhadas (método `getHorasTrabalhadas`).



3.2 Sugestões de implementação

Crie funções auxiliares para reaproveitar comportamento das telas – há muita semelhança entre algumas delas (por exemplo, a obtenção de uma data é sempre igual).

Uma outra dica é usar um `do-while` para o menu principal (evite recursão) e funções para cada item do menu principal.

3.3 Exemplo

Segue um exemplo de funcionamento do programa com a saída esperada e ressaltando em **vermelho** os dados digitados pelo usuário.

```
Acesso ao predio
1) Entrada
2) Saida
3) Registro manual
4) Cadastro de usuario
```

```
5) Relatorio
0) Sair
Escolha uma opcao: 4

Id: 1
Nome: Maria
Usuario cadastrado com sucesso

Acesso ao predio
1) Entrada
2) Saida
3) Registro manual
4) Cadastro de usuario
5) Relatorio
0) Sair
Escolha uma opcao: 4

Id: 2
Nome: Joao
Usuario cadastrado com sucesso

Acesso ao predio
1) Entrada
2) Saida
3) Registro manual
4) Cadastro de usuario
5) Relatorio
0) Sair
Escolha uma opcao: 1

Catraca: 0
Id: 2
Hora: 10
Minuto: 0
Segundo: 0
Dia: 28
Mes: 9
Ano: 2024
[Entrada] Catraca 0 abriu: id 2

Acesso ao predio
1) Entrada
2) Saida
3) Registro manual
4) Cadastro de usuario
5) Relatorio
0) Sair
Escolha uma opcao: 1

Catraca: 0
Id: 1
Hora: 10
Minuto: 10
Segundo: 0
```

```
Dia: 28
Mes: 9
Ano: 2024
[Entrada] Catraca 0 abriu: id 1

Acesso ao predio
1) Entrada
2) Saida
3) Registro manual
4) Cadastro de usuario
5) Relatorio
0) Sair
Escolha uma opcao: 2

Catraca: 1
Id: 1
Hora: 13
Minuto: 0
Segundo: 0
Dia: 28
Mes: 9
Ano: 2024
[Saida] Catraca 1 abriu: id 1

Acesso ao predio
1) Entrada
2) Saida
3) Registro manual
4) Cadastro de usuario
5) Relatorio
0) Sair
Escolha uma opcao: 2

Catraca: 1
Id: 2
Hora: 13
Minuto: 5
Segundo: 0
Dia: 28
Mes: 9
Ano: 2024
[Saida] Catraca 1 abriu: id 2

Acesso ao predio
1) Entrada
2) Saida
3) Registro manual
4) Cadastro de usuario
5) Relatorio
0) Sair
Escolha uma opcao: 1

Catraca: 0
Id: 2
```

```
Hora: 14
Minuto: 0
Segundo: 0
Dia: 28
Mes: 9
Ano: 2024
[Entrada] Catraca 0 abriu: id 2

Acesso ao predio
1) Entrada
2) Saida
3) Registro manual
4) Cadastro de usuario
5) Relatorio
0) Sair
Escolha uma opcao: 2

Catraca: 0
Id: 1
Hora: 18
Minuto: 10
Segundo: 0
Dia: 28
Mes: 9
Ano: 2024
[Saida] Catraca 0 travada

Acesso ao predio
1) Entrada
2) Saida
3) Registro manual
4) Cadastro de usuario
5) Relatorio
0) Sair
Escolha uma opcao: 3

Entrada (e) ou Saida (s)? e
Id: 1
Hora: 14
Minuto: 30
Segundo: 0
Dia: 28
Mes: 9
Ano: 2024
Entrada manual registrada: id 1

Acesso ao predio
1) Entrada
2) Saida
3) Registro manual
4) Cadastro de usuario
5) Relatorio
0) Sair
Escolha uma opcao: 2
```

```
Catraca: 0
Id: 1
Hora: 18
Minuto: 15
Segundo: 0
Dia: 28
Mes: 9
Ano: 2024
[Saida] Catraca 0 abriu: id 1

Acesso ao predio
1) Entrada
2) Saida
3) Registro manual
4) Cadastro de usuario
5) Relatorio
0) Sair
Escolha uma opcao: 2

Catraca: 0
Id: 2
Hora: 18
Minuto: 18
Segundo: 0
Dia: 28
Mes: 9
Ano: 2024
[Saida] Catraca 0 abriu: id 2

Acesso ao predio
1) Entrada
2) Saida
3) Registro manual
4) Cadastro de usuario
5) Relatorio
0) Sair
Escolha uma opcao: 5

Mes: 9
Ano: 2024

Relatorio de horas trabalhadas
Maria: 6
Joao: 7

Acesso ao predio
1) Entrada
2) Saida
3) Registro manual
4) Cadastro de usuario
5) Relatorio
0) Sair
Escolha uma opcao: 0
```

4 Entrega

O projeto deverá ser entregue até o dia **21/10** em um Judge específico, disponível em <https://laboo.pcs.usp.br/ep/> (nos próximos dias vocês receberão um login e uma senha).

As duplas podem ser formadas por alunos de qualquer turma e elas devem ser informadas no e-Disciplinas até o dia 04/10. Caso não seja informada a dupla, será considerado que o aluno está fazendo o EP sozinho. **Note que no EP2 deve-se manter a mesma dupla do EP1 (será apenas possível *desfazer* a dupla, mas não formar uma nova).**

> **Atenção:** não copie código de um outro grupo. Qualquer tipo de cópia será considerado plágio e **todos** os alunos dos 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` e o `menu` (que devem **obrigatoriamente** ficar nos arquivos "main.cpp" e "menu.cpp", respectivamente), em um arquivo comprimido no formato ZIP (outros formatos, como RAR e 7Z, *podem* não ser reconhecidos e acarretar **nota 0**). O nome do arquivo não pode conter espaço, ".", acentos ou ter mais de 11 caracteres. Os códigos fonte não devem ser colocados em pastas. A submissão pode ser feita por qualquer um dos membros da dupla – recomenda-se que os dois submetam.

> **Atenção:** faça a submissão do mesmo arquivo nos 4 problemas (Parte 1, Parte 2, Parte 3 e Parte 4). 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 buscando evitar erros de compilação devido à erros de digitação do nome das classes e dos métodos públicos. **Note que a nota dada não é a nota final**: neste momento não são executados testes – o Judge apenas tenta chamar todos os métodos definidos neste documento para todas as classes. Por exemplo, essa verificação é a seguinte para a classe `Registro`:

```
Registro* r = new Registro(new Data(1, 1, 1, 1, 1, 2024), true, true);
Data* d = r->getData();
bool b = r->isEntrada();
b = r->isManual();
delete r;
```

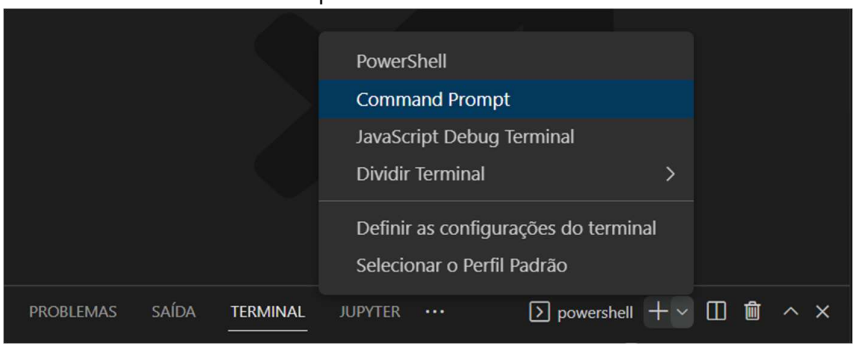
Você pode submeter quantas vezes quiser, sem desconto na nota.

5 Dicas

- Caso o programa esteja travando, execute o programa no modo de depuração. O depurador informará o erro que aconteceu – além de ser possível depurar para descobrir onde o erro aconteceu!
- Faça `#include` apenas das classes que são usadas naquele arquivo. Por exemplo, se o arquivo .h não usa a classe `X`, mas o .cpp usa essa classe, faça o `include` da classe `X` *apenas* no .cpp. Incluir classes desnecessariamente pode gerar erros de compilação estranhos (por causa de referências circulares).
    - Inclua todas as dependências necessárias. Não dependa de `#includes` feitos por outros arquivos incluídos.
- É muito trabalhoso testar o programa ao executar o `main` *com menus*, já que é necessário informar vários dados para inicializar os registradores e a memória de dados. 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.
    - Uma opção é usar o recurso de paste no powershell (o terminal padrão do VSCode). É só clicar com o botão direito no terminal do VSCode.
    - Uma outra opção para testar é usar o comando:

      ```
      ep < entrada.txt > saida.txt
      ```

      Esse comando executa o programa `ep` usando como entrada do teclado o texto no arquivo `entrada.txt` e coloca em `saída.txt` os textos impressos pelo programa (sem os valores digitados). No caso do Windows, para rodar esse comando você precisa de um prompt de comando (por uma limitação do *PowerShell*). Para fazer isso, clique na seta para baixo do lado do + no terminal e escolha Command Prompt.



- Implemente a solução aos poucos – não deixe para implementar tudo no final.
- Submeta no Judge o código com antecedência para descobrir problemas na sua implementação. É normal acontecerem *RuntimeErrors* e outros tipos de erros no Judge que não aparecem ao executar o programa no Windows. Veja a mensagem de erro do Judge para descobrir em qual classe acontece o problema. Caso você queira testar o projeto em um compilador similar ao do Judge, use o site https://github.com/features/codespaces.
    - Em geral *RuntimeErrors* acontecem porque você não inicializou um atributo que é usado. Por exemplo, caso você não crie um vetor ou não inicialize o atributo quantidade, para controlar o tamanho do vetor, ocorrerá um *RuntimeError*.

Atenção: jamais deixe o código fonte do seu EP público na Internet - algum aluno pode usá-lo e isso será identificado como plágio. Se você usar o GitHub, deixe o repositório privado (adicionando o outro membro da dupla como colaborador).

- Use o Fórum de dúvidas do EP no e-Disciplinas 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.**