



# C++ - Módulo 05

## Repetição e Exceções

*Resumo:*

*Este documento contém os exercícios do Módulo 05 dos módulos C++.*

*Versão: 10.1*

# Conteúdo

<b>I</b>	<b>Introdução</b>	<b>2</b>
<b>II</b>	<b>Regras gerais</b>	<b>3</b>
<b>III</b>	<b>Exercício 00: Mamãe, quando eu crescer quero ser burocrata! 5</b>	
<b>IV</b>	<b>Exercício 01: Formem-se, vermes!</b>	<b>7</b>
<b>V</b>	<b>Exercício 02: Não, você precisa do formulário 28B, não do 28C...</b>	<b>9</b>
<b>VI</b>	<b>Exercício 03: Pelo menos isso é melhor do que fazer café</b>	<b>11</b>
<b>VII</b>	<b>Submissão e avaliação por pares</b>	<b>13</b>

# Capítulo I

## Introdução

*C++ é uma linguagem de programação de uso geral criada por Bjarne Stroustrup como uma extensão da linguagem de programação C, ou "C com Classes" (fonte: [Wikipedia](#)).*

O objetivo desses módulos é apresentar **a Programação Orientada a Objetos**.

Este será o ponto de partida de sua jornada em C++. Muitas linguagens são recomendadas para aprender OOP. Decidimos escolher C++ porque ele é derivado do seu velho amigo C.

Por se tratar de uma linguagem complexa e para manter as coisas simples, seu código estará em conformidade com o padrão C++98.

Estamos cientes de que o C++ moderno é muito diferente em muitos aspectos. Portanto, se você deseja se tornar um desenvolvedor C++ proficiente, cabe a você ir além após o 42 Common Core!

## Capítulo II

### Regras gerais

#### Compilando

- Compile seu código com `c++` e os sinalizadores `-Wall -Wextra -Werror`
- Seu código ainda deverá ser compilado se você adicionar o sinalizador `-std=c++98`

#### Convenções de formatação e nomenclatura

- Os diretórios dos exercícios serão nomeados desta forma: `ex00`, `ex01`, ... , `ex`
- Nomeie seus arquivos, classes, funções, funções de membro e atributos conforme exigido em As diretrizes.
- Escreva nomes de classes no formato **UpperCamelCase** . Arquivos contendo código de classe serão sempre ser nomeado de acordo com o nome da classe. Por exemplo:  
`ClassName.hpp/ClassName.h`, `ClassName.cpp` ou `ClassName.hpp`. Então, se você tiver um arquivo de cabeçalho contendo a definição de uma classe "BrickWall" que representa uma parede de tijolos, seu nome será `BrickWall.hpp`.
- A menos que especificado de outra forma, todas as mensagens de saída devem ser finalizadas com uma nova linha caractere e exibido na saída padrão.
- *Adeus Norminette!* Nenhum estilo de codificação é imposto nos módulos C++. Você pode seguir o seu favorito. Mas tenha em mente que um código que seus pares avaliadores não conseguem entender é um código que eles não podem avaliar. Faça o seu melhor para escrever um código limpo e legível.

#### Permitido/Proibido

Você não está mais codificando em C. É hora de C++! Portanto:

- Você tem permissão para usar quase tudo da biblioteca padrão. Assim, em vez de se ater ao que você já sabe, seria inteligente usar o máximo possível as versões C++ das funções C com as quais você está acostumado.
- Entretanto, você não pode usar nenhuma outra biblioteca externa. Isso significa que C++ 11 (e formas derivadas) e bibliotecas Boost são proibidas. As seguintes funções também são proibidas: `*printf()`, `*alloc()` e `free()`. Se você usá-los, sua nota será 0 e pronto.

- Observe que, salvo indicação explícita em contrário, o namespace using <ns\_name> e palavras-chave de amigos são proibidas. Caso contrário, sua nota será -42.
- **É permitido utilizar o STL somente nos Módulos 08 e 09.** Isso significa: nenhum **contêiner** (vetor/lista/mapa/e assim por diante) e nenhum **algoritmo** (qualquer coisa que exija a inclusão do cabeçalho <algorithm>) até então. Caso contrário, sua nota será -42.

### Alguns requisitos de design

- O vazamento de memória também ocorre em C++. Quando você aloca memória (usando o novo palavra-chave), você deve evitar **vazamentos de memória**.
- Do Módulo 02 ao Módulo 09, suas aulas devem ser planejadas no **estilo Ortodoxo Forma Canônica, exceto quando explicitamente indicado de outra forma**.
- Qualquer implementação de função colocada em um arquivo de cabeçalho (exceto modelos de função) significa 0 para o exercício.
- Você deve ser capaz de usar cada um dos seus cabeçalhos independentemente dos outros. Assim, eles devem incluir todas as dependências de que necessitam. No entanto, você deve evitar o problema da inclusão dupla adicionando **guardas de inclusão**. Caso contrário, sua nota será 0.

### Leia-me

- Você pode adicionar alguns arquivos adicionais se precisar (ou seja, para dividir seu código). Como essas atribuições não são verificadas por um programa, fique à vontade para fazê-lo, desde que entregue os arquivos obrigatórios.
- Às vezes, as orientações de um exercício parecem curtas, mas os exemplos podem mostrar requisitos que não estão explicitamente escritos nas instruções.
- Leia cada módulo completamente antes de começar! Realmente, faça isso.
- Por Odin, por Thor! Use seu cérebro!!!




Você terá que implementar muitas classes. Isso pode parecer tedioso, a menos que você consiga criar um script em seu editor de texto favorito.



Você tem uma certa liberdade para completar os exercícios. Porém, siga as regras obrigatórias e não seja preguiçoso. Você poderia perder muitas informações úteis! Não hesite em ler sobre conceitos teóricos.

## Capítulo III

### Exercício 00: Mamãe, quando eu crescer quero ser burocrata!

	Exercício: 00
Mamãe, quando eu crescer quero ser burocrata!	
Diretório de entrega: ex00/	
Arquivos para entrega: Makefile, main.cpp, Bureaucrat.{h, cpp}, Bureaucrat.cpp Funções proibidas: Nenhuma	



Observe que as classes de exceção não precisam ser elaboradas na Forma Canônica Ortodoxa. Mas todas as outras aulas precisam.

Vamos projetar um pesadelo artificial de escritórios, corredores, formulários e filas de espera. Parece divertido? Não? Muito ruim.

Primeiro, comecemos pela menor engrenagem desta vasta máquina burocrática: o **Burocrata**.

Um **burocrata** deve ter:

- Um nome constante.
- E uma nota que varia de **1** (nota mais alta possível) a **150** (nota mais baixa possível nota).

Qualquer tentativa de instanciar um Burocrata usando uma nota inválida deve gerar uma exceção: uma `Burocrata::GradeTooHighException` ou uma `Burocrata::GradeTooLowException`.

Você fornecerá getters para ambos os atributos: getName() e getGrade(). Implementar também duas funções de membros para aumentar ou diminuir o grau de burocrata. Se a nota estiver fora do intervalo, ambos lançarão as mesmas exceções que o construtor.



Lembrar. Como a nota 1 é a mais alta e 150 a mais baixa, incrementar a nota 3 deveria dar nota 2 ao burocrata.

As exceções lançadas devem ser capturáveis usando blocos try e catch:

```
tentar
{
    /* fazer algumas coisas com burocratas */
}
catch (std::exception & e) {

    /* trata exceção */
}
```


Você implementará uma sobrecarga do operador de inserção («) para imprimir algo como (sem os colchetes angulares):

<nome>, grau de burocrata <grau>.

Como sempre, faça alguns testes para comprovar que tudo funciona conforme o esperado.

## Capítulo IV

### Exercício 01: Preparem-se, vermes!

	Exercício: 01
Forme-se, vermes!	
Diretório de entrega: ex01/	
Arquivos para entrega: Arquivos do exercício anterior + Form.{h, cpp}, Form.cpp Funções proibidas: Nenhuma	

Agora que você tem burocratas, vamos dar-lhes algo para fazer. Que melhor atividade poderia haver do que preencher uma pilha de formulários?

Então, vamos fazer uma classe **Form**. Tem:

- Um nome constante.
- Um booleano que indica se está assinado (na construção, não está).
- Uma nota constante exigida para assiná-lo.
- E uma nota constante necessária para executá-lo.

Todos esses atributos são **privados** e não protegidos.

As notas do **Formulário** seguem as mesmas regras que se aplicam ao Burocrata. Por isso, as seguintes exceções serão lançadas se uma nota de formulário estiver fora dos limites: `Form::GradeTooHighException` e `Form::GradeTooLowException`.

Como antes, escreva getters para todos os atributos e uma sobrecarga do operador de inserção («) que imprime todas as informações do formulário.



Adicione também uma função membro `beSigned()` ao `Form` que usa um `Burocrata` como parâmetro. Altera o status do formulário para assinado se a nota do burocrata for alta o suficiente (maior ou igual à exigida). Lembre-se de que a nota 1 é superior à nota 2. Se a nota for muito baixa, lance um `Form::GradeTooLowException`.

Por último, adicione uma função membro `signForm()` ao `Burocrata`. Se o formulário foi assinado, ele imprimirá algo como:

<burocrata> assinado <formulário>


Caso contrário, imprimirá algo como:

<burocrata> não pôde assinar <formulário> porque <motivo>.

Implemente e entregue alguns testes para garantir que tudo funcione conforme o esperado.

## Capítulo V

### Exercício 02: Não, você precisa do formulário 28B, não do 28C...

	Exercício: 02
Não, você precisa do formulário 28B, não do 28C...	
Diretório de entrega: ex02/	
Arquivos para entrega: Makefile, main.cpp, Bureaucrat.{h, cpp}, Bureaucrat.cpp + AForm.{h, cpp}, ShrubberyCreationForm.{h, cpp}, + RobotomyRequestForm.{h, cpp}, PresidentialPardonForm.{h, cpp}	
Funções proibidas: Nenhuma	

Como agora você tem formulários básicos, é hora de criar mais alguns que realmente façam alguma coisa.

Em todos os casos, a classe base Form deve ser uma classe abstrata e, portanto, deve ser renomeada como AForm. Tenha em mente que os atributos do formulário precisam permanecer privados e que estão na classe base.

Adicione as seguintes classes concretas:

- **ShrubberyCreationForm:** notas exigidas: sinal 145, exec 137  
Crie um arquivo <target>\_shrubbery no diretório de trabalho e grave árvores ASCII dentro dele.
- **RobotomyRequestForm:** Notas exigidas: sinal 72, exec 45 Faz alguns ruídos de perfuração. Em seguida, informa que <target> foi robotizado com sucesso em 50% das vezes. Caso contrário, informa que a robotia falhou.
- **PresidentialPardonForm:** Notas exigidas: sinal 25, exec 5  
Informa que <target> foi perdoado por Zaphod Beeblebrox.

Todos eles levam apenas um parâmetro em seu construtor: o alvo do formulário. Para por exemplo, “casa” se quiser plantar arbustos em casa.

Agora, adicione a função membro `execute(Bureaucrat const & executor) const` ao formulário base e implemente uma função para executar a ação do formulário das classes concretas. Você deve verificar se o formulário está assinado e se a nota do burocrata que tenta executá-lo é alta o suficiente. Caso contrário, lance uma exceção apropriada.

Se você deseja verificar os requisitos em cada classe concreta ou na classe base (depois chamar outra função para executar o formulário), depende de você. Porém, uma forma é mais bonita que a outra.

Por último, adicione a função membro `executeForm(AForm const & form)` ao `Bureau-`car. Ele deve tentar executar o formulário. Se der certo, imprima algo como:


```
<burocrata> executado <formulário>
```

Caso contrário, imprima uma mensagem de erro explícita.

Implemente e entregue alguns testes para garantir que tudo funcione conforme o esperado.

## Capítulo VI

### Exercício 03: Pelo menos isso é melhor do que fazer café

	Exercício: 03
Pelo menos isso é melhor do que fazer café	
Diretório de entrega: ex03/	
Arquivos para entregar: Arquivos de exercícios anteriores + Intern.{h, hpp}, Intern.cpp Funções proibidas: Nenhuma	

Como preencher formulários já é bastante chato, seria cruel pedir aos nossos burocratas que fizessem isso o dia todo. Felizmente, existem estagiários. Neste exercício, você deve implementar a classe **Intern**. O estagiário não tem nome, nem nota, nem características únicas. A única coisa com que os burocratas se preocupam é que façam o seu trabalho.

Porém, o estagiário tem uma capacidade importante: a função `makeForm()`. São necessárias duas cordas. O primeiro é o nome de um formulário e o segundo é o destino do formulário. Ele retorna um ponteiro para um **objeto Form** (cujo nome é aquele passado como parâmetro) cujo alvo será inicializado para o segundo parâmetro.

Irá imprimir algo como:

Estagiário cria <formulário>

Se o nome do formulário passado como parâmetro não existir, imprima uma mensagem de erro explícita.

Você deve evitar soluções ilegíveis e feias, como usar uma floresta `if/elseif/else`. Esse tipo de coisa não será aceito durante o processo de avaliação. Você não está mais na Piscine (piscina). Como sempre, você deve testar se tudo funciona conforme o esperado.

Por exemplo, o código abaixo cria um **RobotomyRequestForm** direcionado a “Ben-der”:

```
{
    Estagiário someRandomIntern;
    Formulário* rrf;

    rrf = someRandomIntern.makeForm(" solicitação de robotomia", "Bender");
}
```

## Capítulo VII

### Envio e avaliação por pares

Entregue sua tarefa em seu repositório Git normalmente. Somente o trabalho dentro do seu repositório será avaliado durante a defesa. Não hesite em verificar os nomes de suas pastas e arquivos para garantir que estão corretos.



16D85ACC441674FBA2DF65190663F9373230CEAB1E4A0818611C0E39F5B26E4D774F1  
74620A16827E1B16612137E59ECD492E468A92DCB17BF16988114B98587594D12810  
E67D173222A