



# C++ - Módulo 05

## Repetição e Exceções

*Resumo:*

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

*Versão: 9.1*

# 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: Em forma, vermes!</b>	<b>7</b>
<b>V</b>	<b>Exercício 02: Não, precisa do formulário 28B, não do 28C...</b>	<b>9</b>
<b>VI</b>	<b>Exercício 03: Pelo menos isso supera o preparo do café</b>	<b>11</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 with Classes" (fonte: [Wikipedia](#)).*

O objetivo desses módulos é apresentá-lo à **Programação Orientada a Objetos**. Este será o ponto de partida de sua jornada C++. Muitos idiomas são recomendados para aprender OOP. Decidimos escolher C++, pois é derivado de 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.

Sabemos 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 do 42 Common Core!

## Capítulo II

### Regras gerais

#### Compilando

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

#### Convenções de formatação e nomenclatura

- Os diretórios de exercícios serão nomeados desta forma: `ex00`, `ex01`, ... , `exn`
- Nomeie seus arquivos, classes, funções, funções de membro e atributos conforme exigido em As diretrizes.
- Escreva os nomes das 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.tpp`. 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 encerradas com uma nova linha caractere e exibido na saída padrão.
- *Adeus Norminette!* Nenhum estilo de codificação é aplicado nos módulos C++. Você pode seguir o seu favorito. Mas lembre-se de 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. Hora de C++! Portanto:

- Você tem permissão para usar quase tudo da biblioteca padrão. Portanto, em vez de se ater ao que você já sabe, seria inteligente usar o máximo possível as versões em C++ das funções C às quais você está acostumado.
- No entanto, você não pode usar nenhuma outra biblioteca externa. Isso significa que as bibliotecas C++11 (e formas derivadas) e 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, a menos que explicitamente declarado de outra forma, o namespace using <ns\_name> e palavras-chave de amigos são proibidas. Caso contrário, sua nota será -42.
- **Você tem permissão para usar o STL apenas no Módulo 08 e 09.** Isso significa: sem **contêineres** (vetor/lista/mapa/e assim por diante) e sem **algoritmos** (qualquer coisa que requeira incluir o cabeçalho <algorithm>) até então. Caso contrário, sua nota será -42.

### Alguns requisitos de projeto

- 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 elaboradas no **Ortodoxo Forma Canônica, exceto quando explicitamente declarado de outra forma**.
- Qualquer implementação de função colocada em um arquivo de cabeçalho (exceto para modelos de função) significa 0 para o exercício.
- Você deve ser capaz de usar cada um de seus cabeçalhos independentemente dos outros. Assim, eles devem incluir todas as dependências de que precisam. No entanto, você deve evitar o problema de 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, sinta-se à vontade para fazê-lo, desde que entregue os arquivos obrigatórios.
- Às vezes, as diretrizes 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 seja capaz de criar o script de seu editor de texto favorito.



Você tem uma certa liberdade para completar os exercícios. No entanto, siga as regras obrigatórias e não seja preguiçoso. Você poderia perder muita informação útil! 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 a entregar: Makefile, main.cpp, Bureaucrat.{h, hpp}, Bureaucrat.cpp Funções proibidas: Nenhuma	



Observe que as classes de exceção não precisam ser projetadas na forma canônica ortodoxa. Mas todas as outras classes precisam.

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

Primeiro, comece pela menor engrenagem dessa vasta máquina burocrática: o **Burocrata**.

Um **burocrata** deve ter:

- Um nome constante.
- E uma nota que varia de **1** (maior nota possível) a **150** (menor nota 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()`. Implemente também duas funções de membro 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 o grau 1 é o mais alto e o 150 o mais baixo, incrementar um grau 3 deve dar um grau 2 ao burocrata.

As exceções lançadas devem ser capturadas usando blocos `try` e `catch`:

```
tente
{
    /* fazer algumas coisas com burocratas */
}
catch (padrão::exceção & 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 de costume, faça alguns testes para provar que tudo funciona conforme o esperado.

## Capítulo IV

### Exercício 01: Forme-se, vermes!

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

Agora que você tem burocratas, vamos dar a eles algo para fazer. Qual melhor atividade poderia haver do que preencher uma pilha de formulários?

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

- Um nome constante.
- Um booleano indicando se está assinado (na construção, não está).
- Uma nota constante exigida para assiná-lo.
- E um grau constante necessário para executá-lo.

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

As notas da **Forma** 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 de membro `beSigned()` ao `Form` que recebe um `Burocrata` como parâmetro. Muda o status do formulário para assinado se o grau do burocrata for alto o suficiente (maior ou igual ao exigido). Lembre-se, o grau 1 é superior ao grau 2. Se a nota for muito baixa, lance um `Form::GradeTooLowException`.

Por fim, adicione uma função de membro `signForm()` ao arquivo `Burocrata`. Se o formulário foi assinado, vai imprimir algo como:

<burocrata> assinou <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 28C...	
Diretório de entrega: ex02/	
Arquivos a serem entregues: Makefile, main.cpp, Bureaucrat.{h, hpp}, Bureaucrat.cpp + AForm.{h, hpp}, ShrubberyCreationForm.{h, hpp}, + RobotomiaRequestForm.{h, hpp}, PresidentialPardonForm.{h, hpp}	
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. Lembre-se de que os atributos do formulário precisam permanecer privados e que estão na classe base.

Adicione as seguintes classes concretas:

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

Todos eles levam apenas um parâmetro em seu construtor: o destino 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 o grau do burocrata que está tentando executar o formulário é alto o suficiente. Caso contrário, lance uma exceção apropriada.

Se você deseja verificar os requisitos em cada classe concreta ou na classe base (então chame outra função para executar o formulário) depende de você. No entanto, uma maneira é mais bonita que a outra.

Por fim, adicione a função de membro `executeForm(Form const & form)` ao `Bureaucrat`. Deve tentar executar o formulário. Se for bem-sucedido, imprima algo como:


```
<burocrata> executou <forma>
```

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 a entregar: Arquivos de exercícios anteriores + Intern.{h, hpp}, Intern.cpp Funções proibidas: Nenhuma	

Como o preenchimento de formulários é chato o suficiente, 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 grau, nem características únicas. A única coisa com que os burocratas se preocupam é que eles façam o seu trabalho.

No entanto, o estagiário tem uma capacidade importante: a função `makeForm()`. Leva 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 o passado como parâmetro) cujo destino será inicializado com o segundo parâmetro.

Ele imprimirá 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 em Piscine (piscina). Como de costume, você deve testar se tudo funciona conforme o esperado.

Por exemplo, o código abaixo cria um **RobotomiaRequestForm** direcionado para "Ben der":

```
{  
    Intern someRandomIntern;  
    Forma* rrf;  
  
    rrf = someRandomIntern.makeForm(" pedido de robotomia", "Bender");  
}
```