



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

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

Este será o ponto de partida da sua jornada C++. Muitas linguagens são recomendadas para aprender POO. Decidimos escolher C++, pois é derivado do seu velho amigo C.

Como esta é 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. Então, se você quer se tornar um desenvolvedor C++ proficiente, cabe a você ir além depois 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 dos exercícios serão nomeados desta forma: `ex00`, `ex01`, ... , `exn`
- Nomeie seus arquivos, classes, funções, funções de membro e atributos conforme necessário em as diretrizes.
- Escreva nomes de classes no formato **UpperCamelCase** . Arquivos contendo código de classe serão sempre será 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" representando uma parede de tijolos, seu nome será `BrickWall.hpp`.
- A menos que especificado de outra forma, todas as mensagens de saída devem ser encerradas por uma nova linha caractere e exibido na saída padrão.
- *Adeus Norminette!* Nenhum estilo de codificação é imposto nos módulos C++. Você pode seguir seu favorito. Mas tenha em mente que um código que seus avaliadores pares não conseguem entender é um código que eles não conseguem classificar. 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. Assim, em vez de ficar preso ao que você já sabe, seria inteligente usar o máximo possível as versões C++-ish das funções C com as 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-las, sua nota será 0 e pronto.

- Observe que, a menos que explicitamente indicado de outra forma, o uso do namespace `<ns_name>` e palavras-chave de amigos são proibidas. Caso contrário, sua nota será -42.
- **Você tem permissão para usar o STL somente no Módulo 08 e 09.** Isso significa: nenhum **Container** (vetor/ lista/mapa/e assim por diante) e nenhum **Algoritmo** (qualquer coisa que exija incluir o cabeçalho `<algoritmo>`) até então. Caso contrário, sua nota será -42.

### Alguns requisitos de design

- Vazamento de memória ocorre em C++ também. 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 na **Língua Ortodoxa Forma canônica, exceto quando explicitamente indicado 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 dos 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 (por exemplo, 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! Sério, faça.
- 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 no 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 iria 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 classes de exceção não precisam ser projetadas em Orthodox Canonical Form. Mas todas as outras classes precisam.

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

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 lançar uma exceção: uma `Bureaucrat::GradeTooHighException` ou uma `Bureaucrat::GradeTooLowException`.

Você fornecerá getters para ambos os atributos: `getName()` e `getGrade()`. Implemente também duas funções de membro para incrementar ou decrementar a nota do burocrata. Se a nota estiver fora do intervalo, ambas lançarão as mesmas exceções que o construtor.



Lembre-se. Como o grau 1 é o mais alto e 150 o mais baixo, incrementar um grau 3 deve dar um grau 2 ao burocrata.

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

```
try {
    /* fazer algumas coisas com burocratas */
}
catch (std::exception & e) {
    /* manipular 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: Formem-se, vermes!

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

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

Então, vamos fazer uma classe **Form**. Ela tem:

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

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

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

Da mesma forma que 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 recebe um `Burocrata` como parâmetro. Ela muda o status do formulário para assinado se a nota do burocrata for alta o suficiente (maior ou igual à necessária). Lembre-se, a nota 1 é maior que a nota 2. Se a nota for muito baixa, lance uma `Form::GradeTooLowException`.

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

`<burocrata> assinou <formulário>`


Caso contrário, ele imprimirá algo como:

`<burocrata> não conseguiu assinar <formulário> por <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 entregar: Makefile, main.cpp, Bureaucrat.{h, hpp}, Bureaucrat.cpp + AForm. {h, hpp}, ShrubberyCreationForm.{h, hpp}, + RobotomyRequestForm.{h, hpp}, PresidentialPardonForm.{h, hpp}	
Funções proibidas: Nenhuma	

Já que 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 para AForm. Tenha em mente que os atributos do formulário precisam permanecer privados e que eles estão na classe base.

Adicione as seguintes classes concretas:

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

Todos eles levam apenas um parâmetro em seu construtor: o alvo do formulário. Para por exemplo, "casa" se você 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 tem que verificar se o formulário está assinado e se o grau do burocrata que tenta executar o formulário é alto o suficiente. Caso contrário, lance uma exceção apropriada.

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

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


`<burocrata> executou <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, cpp}, Intern.cpp Funções proibidas: Nenhuma	

Como preencher formulários é irritante o suficiente, seria cruel pedir aos nossos burocratas para fazer isso o dia todo. Felizmente, os estagiários existem. Neste exercício, você tem que implementar a classe **Intern**. O estagiário não tem nome, nem nota, nem características únicas. A única coisa com que os burocratas se importam é que eles façam seu trabalho.

No entanto, o estagiário tem uma capacidade importante: a função `makeForm()`. Ela recebe duas strings. A primeira é o nome de um formulário e a segunda é o alvo do formulário. Ela retorna um ponteiro para um **objeto Form** (cujo nome é o passado como parâmetro) cujo alvo será inicializado para 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 (pool). Como de costume, você tem que testar se tudo funciona como esperado.

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

```
{  
    Estagiário algumEstagiárioAleatório;  
    Formulário* rrf;  
  
    rrf = someRandomIntern.makeForm(" solicitação de robotomia", "Bender");  
}
```

## Capítulo VII

### Submissão e avaliação por pares

Entregue sua tarefa no seu repositório Git como de costume. Apenas o trabalho dentro do seu repositório será avaliado durante a defesa. Não hesite em verificar novamente os nomes das suas pastas e arquivos para garantir que estejam corretos.



16D85ACC441674FBA2DF65190663F9373230CEAB1E4A0818611C0E39F5B26E4D774F1  
74620A16827E1B16612137E59ECD492E468A92DCB17BF16988114B98587594D12810  
E67D173222A