



# C++ - Módulo 05

## Repetição e Exceções

*Preâmbulo:*

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

*Versão: 11.0*

# Sumário

I	Introdução	2
II	Regras gerais	3
III	Instruções de IA	6
IV	Exercício 00: Mamãe, quando eu crescer, quero ser um burocrata!	9
V	Exercício 01: Em formação, vermes!	11
VI	Exercício 02: Não, você precisa do formulário 28B, não do 28C...	13
VII	Exercício 03: Pelo menos isso é melhor do que fazer café	15
VIII	Entrega e Avaliação por Pares	17

# 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 introduzi-lo à **Programação Orientada a Objetos**. Este será o ponto de partida da sua jornada em C++. Muitas linguagens são recomendadas para aprender OOP. Escolhemos C++ por ser derivado do seu velho amigo, C. Por ser uma linguagem complexa, e para manter as coisas simples, o seu código deverá cumprir o padrão C++98.

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

# Capítulo II

## Regras gerais

### Compilação

- Compile seu código com `c++` e as flags `-Wall -Wextra -Werror`
- Seu código ainda deve compilar se você adicionar a flag `-std=c++98`

### Formatação e convenções de nomenclatura

- Os diretórios dos exercícios serão nomeados desta forma: `ex00`, `ex01`, ... , `exn`
- Nomeie seus arquivos, classes, funções, funções membro e atributos conforme exigido nas diretrizes.
- Escreva os nomes das classes no formato **UpperCamelCase**. Arquivos contendo código de classe sempre serão nomeados de acordo com o nome da classe. Por exemplo:  
`NomeDaClasse.hpp`/`NomeDaClasse.h`, `NomeDaClasse.cpp`, ou `NomeDaClasse.tpp`. Então, se você tiver um arquivo de cabeçalho contendo a definição de uma classe "ParedeDeTijolo" representando uma parede de tijolos, seu nome será `ParedeDeTijolo.hpp`.
- A menos que especificado de outra forma, cada mensagem de saída deve terminar com um caractere de nova linha e ser exibida na saída padrão.
- *Adeus Norminette!* Nenhum estilo de codificação é imposto nos módulos C++. Você pode seguir o seu favorito. Mas lembre-se que o código que seus 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 programando em C. É hora de C++! Portanto:

- Você pode usar quase tudo da biblioteca padrão. Assim, em vez de se ater ao que você já conhece, seria inteligente usar as versões em C++ das funções C que você está acostumado o máximo possível.

- No entanto, você não pode usar nenhuma outra biblioteca externa. Isso significa que 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, as palavras-chave `using namespace <ns_name>` e `friend` são proibidas. Caso contrário, sua nota será -42.
- **Você só pode usar a STL nos Módulos 08 e 09.** Isso significa: nenhum **Contêiner** (`vector/list/map`, e assim por diante) e nenhum **Algoritmo** (qualquer coisa que exija a inclusão do cabeçalho `<algorithm>`) até lá. Caso contrário, sua nota será -42.

### Alguns requisitos de design

- Vazamento de memória ocorre em C++ também. Quando você aloca memória (usando a palavra-chave `new`), você deve evitar **vazamentos de memória**.
- Do Módulo 02 ao Módulo 09, suas classes devem ser projetadas na **Forma Canônica Ortodoxa, 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 de seus cabeçalhos independentemente de outros. Portanto, eles devem incluir todas as dependências de que precisam. No entanto, você deve evitar o problema de inclusão dupla adicionando **proteções 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 você 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 isso.
- Por Odin, por Thor! Use seu cérebro!!!



Em relação ao Makefile para projetos C++, as mesmas regras do C se aplicam (consulte o capítulo Norma sobre o Makefile).



Você terá que implementar muitas classes. Isso pode parecer tedioso, a menos que você seja capaz de criar scripts para 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 perderia muitas informações úteis! Não hesite em ler sobre conceitos teóricos.

# Capítulo III

## InSTRUÇÕES DE IA

### ● Contexto

Este projeto foi desenvolvido para ajudá-lo a descobrir os blocos de construção fundamentais do seu treinamento em TIC.

Para ancorar adequadamente os conhecimentos e habilidades-chave, é essencial adotar uma abordagem criteriosa ao uso de ferramentas e suporte de IA.

A verdadeira aprendizagem fundamental exige um esforço intelectual genuíno — através de desafios, repetição e trocas de aprendizagem entre pares.

Para uma visão geral mais completa de nossa posição sobre a IA — como ferramenta de aprendizagem, como parte do currículo de TIC e como expectativa no mercado de trabalho — consulte as perguntas frequentes dedicadas na intranet.

### ● Mensagem principal

- 👉 Construa bases sólidas sem atalhos.
- 👉 Desenvolva verdadeiramente habilidades técnicas e de poder.
- 👉 Experimente a verdadeira aprendizagem entre pares, comece a aprender como aprender e resolver novos problemas.
- 👉 A jornada de aprendizagem é mais importante que o resultado.
- 👉 Aprenda sobre os riscos associados à IA e desenvolva práticas eficazes de controle e contramedidas para evitar armadilhas comuns.

## ● Regras para o aluno:

- Você deve aplicar o raciocínio às suas tarefas atribuídas, especialmente antes de recorrer à IA.
- Você não deve pedir respostas diretas à IA.
- Você deve aprender sobre a abordagem global da 42 em relação à IA.

## ● Resultados da fase:

Nesta fase fundamental, você obterá os seguintes resultados:

- Obter bases sólidas em tecnologia e codificação.
- Saber por que e como a IA pode ser perigosa durante esta fase.

## ● Comentários e exemplo:

- Sim, sabemos que a IA existe — e sim, ela pode resolver seus projetos. Mas você está aqui para aprender, não para provar que a IA aprendeu. Não perca seu tempo (nem o nosso) apenas para demonstrar que a IA pode resolver o problema dado.
- Aprender na 42 não é sobre saber a resposta — é sobre desenvolver a capacidade de encontrar uma. A IA lhe dá a resposta diretamente, mas isso o impede de construir seu próprio raciocínio. E o raciocínio leva tempo, esforço e envolve falhas. O caminho para o sucesso não deve ser fácil.
- Lembre-se de que durante os exames, a IA não estará disponível — sem internet, sem smartphones, etc. Você perceberá rapidamente se confiou demais na IA em seu processo de aprendizagem.
- A aprendizagem entre pares o expõe a diferentes ideias e abordagens, melhorando suas habilidades interpessoais e sua capacidade de pensar de forma divergente. Isso é muito mais valioso do que apenas conversar com um bot. Então não seja tímido — converse, faça perguntas e aprenda juntos!
- Sim, a IA fará parte do currículo — tanto como ferramenta de aprendizagem quanto como um tópico em si. Você até terá a chance de construir seu próprio software de IA. Para saber mais sobre nossa abordagem crescente, consulte a documentação disponível na intranet.

**✓ Boa prática:**

Estou travado em um novo conceito. Pergunto a alguém próximo como ele abordou isso. Conversamos por 10 minutos — e de repente, clica. Entendi.

**✗ Má prática:**

Uso secretamente a IA, copio algum código que parece certo. Durante a avaliação entre pares, não consigo explicar nada. Eu falho. Durante o exame — sem IA — estou travado novamente. Eu falho.

## Capítulo IV

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

	Exercice : 00
	Mamãe, quando eu crescer, quero ser um burocrata!
	Pasta de entrega : <i>ex00/</i>
	Arquivos para entregar : Makefile, main.cpp, Bureaucrat.{h, hpp}, Bureaucrat.cpp
	Funções não permitidas : Nenhuma



Observe que as classes de exceção não precisam ser projetadas no Formato Canônico Ortodoxo. No entanto, todas as outras classes devem segui-lo.

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

Primeiro, comece com a menor engrenagem nesta vasta máquina burocrática: o **Bureaucrat**.

Um **Bureaucrat** deve ter:

- Uma constante name.
- Uma classificação que varia de **1** (classificação mais alta possível) a **150** (classificação mais baixa possível).

Qualquer tentativa de instanciar um **Bureaucrat** com uma classificação inválida deve lançar uma exceção:

seja **Bureaucrat::GradeTooHighException** ou **Bureaucrat::GradeTooLowException**.

Você deverá fornecer getters para ambos os atributos: `getName()` e `getGrade()`. Você também deve implementar duas funções membro para incrementar ou decrementar a classificação do burocrata. Se a classificação sair do intervalo, ambas as funções devem lançar as mesmas exceções que o construtor.



Lembre-se, já que a classificação 1 é a mais alta e 150 a mais baixa, incrementar uma classificação 3 deve resultar em uma classificação 2 para o burocrata.

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

```
try
{
    /* faça algumas coisas com burocratas */
}
catch (std::exception & e)
{
    /* lide com a exceção */
}
```

Você deve implementar uma sobrecarga do operador de inserção («) para imprimir a saída no seguinte formato (sem os colchetes angulares):

<name>, burocrata com classificação <grade>.

Como de costume, envie alguns testes para provar que tudo funciona como esperado.

# Capítulo V

## Exercício 01: Em formação, vermes!

	Exercice : 01
	Em formação, vermes!
Pasta de entrega :	<i>ex01/</i>
Arquivos para entregar :	Arquivos do exercício anterior + Form.{h, hpp}, Form.cpp
Funções não permitidas :	Nenhuma

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

Vamos criar uma classe **Form**. Ela tem:

- Uma constant name.
- Um booleano indicando se está assinado (na construção, não está).
- Uma classificação constante necessária para assiná-lo.
- Uma classificação constante necessária para executá-lo.

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

As classificações do **Form** seguem as mesmas regras que as do **Bureaucrat**. Assim, as seguintes exceções serão lançadas se a classificação de um formulário estiver fora dos limites:

`Form::GradeTooHighException` e `Form::GradeTooLowException`.

Como antes, escreva getters para todos os atributos e sobrecarregue o operador de inserção («) para imprimir todas as informações do formulário.

Além disso, adicione uma função membro `beSigned()` ao `Form` que receba um `Bureaucrat` como parâmetro. Ele muda o status do formulário para assinado se a classificação do burocrata for alta o suficiente (maior ou igual à exigida). Lembre-se, a classificação 1 é maior que a classificação 2.

Se a classificação for muito baixa, lance uma `Form::GradeTooLowException`.

Em seguida, modifique a função membro `signForm()` na classe `Bureaucrat`. Esta função deve chamar `Form::beSigned()` para tentar assinar o formulário. Se o formulário for assinado com sucesso, ele imprimirá algo como:

```
<bureaucrat> assinou <form>
```

Caso contrário, ele imprimirá algo como:

```
<bureaucrat> não conseguiu assinar <form> porque <reason>.
```

Implemente e envie alguns testes para garantir que tudo funcione como esperado.

# Capítulo VI

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

	Exercice : 02
	Não, você precisa do formulário 28B, não do 28C...
Pasta de entrega :	<i>ex02/</i>
Arquivos para entregar :	Makefile, main.cpp, Bureaucrat.{h, hpp},cpp], + AForm.{h, hpp},cpp], ShrubberyCreationForm.{h, hpp},cpp], + RobotomyRequestForm.{h, hpp},cpp], PresidentialPardonForm.{h, hpp},cpp]
Funções não permitidas :	Nenhuma

Agora que você tem formulários básicos, é hora de criar mais alguns que realmente façam algo.

Em todos os casos, a classe base Form deve ser uma classe abstrata e, portanto, deve ser renomeada para AForm. Lembre-se de que os atributos do formulário precisam permanecer privados e que eles pertencem à classe base.

Adicione as seguintes classes concretas:

- **ShrubberyCreationForm:** Classificações necessárias: assinar 145, executar 137  
Cria um arquivo `<target>_shrubbery` no diretório de trabalho e escreve árvores ASCII dentro dele.
- **RobotomyRequestForm:** Classificações necessárias: assinar 72, executar 45  
Faz alguns ruídos de perfuração e, em seguida, informa que `<target>` foi robotomizado com sucesso 50robotomia falhou.
- **PresidentialPardonForm:** Classificações necessárias: assinar 25, executar 5  
Informa que `<target>` foi perdoado por Zaphod Beeblebrox.

Todos eles recebem apenas um parâmetro em seu construtor: o alvo do formulário. 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 nas classes concretas. Você deve verificar se o formulário está assinado e se a classificação do burocrata que tenta executar o formulário é alta o suficiente. Caso contrário, lance uma exceção apropriada.

Se você verifica os requisitos em cada classe concreta ou na classe base (e então chama outra função para executar o formulário) depende de você. No entanto, uma maneira é mais elegante do que a outra.

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

```
<bureaucrat> executou <form>
```

Caso contrário, imprima uma mensagem de erro explícita.

Implemente e envie alguns testes para garantir que tudo funcione como esperado.

# Capítulo VII

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

	Exercice : 03
	Pelo menos isso é melhor do que fazer café
	Pasta de entrega : <i>ex03/</i>
	Arquivos para entregar : Arquivos dos exercícios anteriores + Intern.{h, hpp}, Intern.cpp
	Funções não permitidas : Nenhuma

Como preencher formulários o dia todo seria muito cruel para nossos burocratas, os estagiários existem para assumir essa tarefa tediosa. Neste exercício, você deve implementar a classe **Intern**. O estagiário não tem nome, classificação e nem características únicas. A única coisa com que os burocratas se preocupam é que eles façam seu trabalho.

No entanto, o estagiário tem uma habilidade fundamental: a função `makeForm()`. Esta função recebe duas strings como parâmetros: a primeira representa o nome de um formulário, e a segunda representa o alvo do formulário. Ele retorna um ponteiro para um objeto **AForm** (correspondente ao nome do formulário passado como parâmetro), com seu alvo inicializado para o segundo parâmetro.

Ele deve imprimir algo como:

```
Intern cria <form>
```

Se o nome do formulário fornecido não existir, imprima uma mensagem de erro explícita.

Você deve evitar soluções ilegíveis e confusas, como usar uma estrutura excessiva if/else/if/else. Este tipo de abordagem não será aceito durante o processo de avaliação. Você não está mais na Piscina. Como de costume, você deve testar tudo para garantir que funcione como esperado.

Por exemplo, o código a seguir cria um **RobotomyRequestForm** direcionado para "Bender":

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

# Capítulo VIII

## Entrega e Avaliação por Pares

Envie sua tarefa para o seu repositório `Git` como de costume. Apenas o trabalho dentro do seu repositório será avaliado durante a defesa. Certifique-se de verificar novamente os nomes de suas pastas e arquivos para garantir que estejam corretos.

Durante a avaliação, uma breve **modificação do projeto** pode ser ocasionalmente solicitada. Isso pode envolver uma pequena mudança de comportamento, algumas linhas de código para escrever ou reescrever, ou um recurso fácil de adicionar.

Embora esta etapa possa **não ser aplicável a todos os projetos**, você deve estar preparado para ela se for mencionada nas diretrizes de avaliação.

Esta etapa visa verificar sua compreensão real de uma parte específica do projeto. A modificação pode ser realizada em qualquer ambiente de desenvolvimento que você escolher (por exemplo, sua configuração usual), e deve ser viável em poucos minutos — a menos que um prazo específico seja definido como parte da avaliação.

Você pode, por exemplo, ser solicitado a fazer uma pequena atualização em uma função ou script, modificar uma exibição ou ajustar uma estrutura de dados para armazenar novas informações, etc.

Os detalhes (escopo, alvo, etc.) serão especificados nas **diretrizes de avaliação** e podem variar de uma avaliação para outra para o mesmo projeto.



16D85ACC441674FBA2DF65190663F9373230CEAB1E4A0818611C0E39F5B26E4D774F1  
74620A16827E1B16612137E59ECD492E468A92DCB17BF16988114B98587594D12810  
E67D173222A