



## C++ - Módulo 02

Polimorfismo ad-hoc, sobrecarga de operador  
e forma de classe canônica ortodoxa

*Resumo:*

*Este documento contém os exercícios do Módulo 02 dos módulos C++.*

*Versão: 7.1*

# Conteúdo

<b>I</b>	<b>Introdução</b>	<b>2</b>
<b>II</b>	<b>Regras gerais</b>	<b>3</b>
<b>III</b>	<b>Novas regras</b>	<b>5</b>
<b>IV</b>	<b>Exercício 00: Minha Primeira Aula na Forma Canônica Ortodoxa</b>	<b>6</b>
<b>V</b>	<b>Exercício 01: Rumo a uma classe de números de ponto fixo mais útil</b>	<b>8</b>
<b>VI</b>	<b>Exercício 02: Agora estamos falando</b>	<b>10</b>
<b>VII</b>	<b>Exercício 03: BSP</b>	<b>12</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 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.

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.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 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

### Novas regras


De agora em diante, todas as suas aulas devem ser projetadas na **Forma Canônica Ortodoxa**, a menos que seja explicitamente declarado o contrário. Em seguida, eles implementarão as quatro funções de membro necessárias abaixo:

- Construtor padrão
- Construtor de cópia
- Operador de atribuição de cópia
- Destruidor

Divida o código da classe em dois arquivos. O arquivo de cabeçalho (.hpp/.h) contém a classe definição, enquanto o arquivo de origem (.cpp) contém a implementação.

## Capítulo IV

### Exercício 00: Minha Primeira Aula em Forma Canônica Ortodoxa

	Exercício: 00
Minha primeira aula na forma canônica ortodoxa	
Diretório de entrega: ex00/	
Arquivos a serem entregues: Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp	
Funções proibidas: Nenhuma	

Você acha que conhece inteiros e números de ponto flutuante. Que fofo.

Por favor, leia este artigo de 3 páginas ([1](#), [2](#), [3](#)) para descobrir que não. Vá em frente, leia.

Até hoje, cada número usado em seu código era basicamente um inteiro ou um número de ponto flutuante, ou qualquer uma de suas variantes (short, char, long, double e assim por diante). Depois de ler o artigo acima, é seguro presumir que inteiros e números de ponto flutuante têm características opostas.

Mas hoje as coisas vão mudar. Você vai descobrir um novo e incrível tipo de número: **números de ponto fixo!** Sempre ausente dos tipos escalares da maioria dos idiomas, os números de ponto fixo oferecem um equilíbrio valioso entre desempenho, precisão, alcance e precisão. Isso explica por que os números de ponto fixo são particularmente aplicáveis à computação gráfica, processamento de som ou programação científica, apenas para citar alguns.

Como C++ não possui números de ponto fixo, você irá adicioná-los. [Este artigo](#) de Berkeley é um bom começo. Se você não tem ideia do que é a Berkeley University, leia [esta seção](#) de sua página da Wikipédia.

Crie uma classe na Forma Canônica Ortodoxa que represente um número de ponto fixo:

- Membros privados:

- Um **número inteiro** para armazenar o valor do número de ponto fixo.

- Um **inteiro constante estático** para armazenar o número de bits fracionários. Seu valor sempre será o literal inteiro 8.

- Membros públicos:

- Um construtor padrão que inicializa o valor do número de ponto fixo como 0.

- Um construtor de cópia.

- Uma sobrecarga do operador de atribuição de cópia.

- Um destruidor.

- Uma função de membro `int getRawBits( void ) const`; que retorna o valor bruto do valor de ponto fixo.

- Uma função de membro `void setRawBits( int const raw );` que define o valor bruto do número de ponto fixo.

Executando este código:

```
#include <iostream>

int principal( vazio ){

    Fixo um;
    b(a) fixo;
    C fixo;

    c = b;

    std::cout << a.getRawBits() << std::endl; std::cout <<
    b.getRawBits() << std::endl; std::cout << c.getRawBits() <<
    std::endl;

    retornar 0;
}
```

Deve produzir algo semelhante a:


```
$> ./a.out
Construtor padrão chamado
Construtor de cópia chamado
Operador de atribuição de cópia chamado // <-- Esta linha pode estar faltando dependendo da sua implementação função de membro
getRawBits chamada
Construtor padrão chamado
Operador de atribuição de cópia chamado
função de membro getRawBits chamada função
de membro getRawBits chamada 0 função de
membro getRawBits chamada 0
função de membro getRawBits chamada 0

Destruidor chamado
Destruidor chamado
Destruidor chamado
$>
```



## Capítulo V

### Exercício 01: Rumo a uma classe de números de ponto fixo mais útil

	Exercício 01
Rumo a uma classe de número de ponto fixo mais útil	
Diretório de entrega: ex01/	
Arquivos a serem entregues: Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp	
Funções permitidas: roundf (de <cmath>)	

O exercício anterior foi um bom começo, mas nossa aula é bastante inútil. Só pode representar o valor 0,0.

Adicione os seguintes construtores públicos e funções de membro públicas à sua classe:

- Um construtor que usa um **número inteiro constante** como parâmetro.  
Ele o converte no valor de ponto fixo correspondente. O valor de bits fracionários é inicializado em 8 como no exercício 00.
- Um construtor que usa um **número de ponto flutuante constante** como parâmetro.  
Ele o converte no valor de ponto fixo correspondente. O valor de bits fracionários é inicializado em 8 como no exercício 00.
- Uma função membro float toFloat( void ) const;  
que converte o valor de ponto fixo em um valor de ponto flutuante.
- Uma função de membro int toInt( void ) const;  
que converte o valor de ponto fixo em um valor inteiro.

E adicione a seguinte função aos arquivos da classe **Fixed** :

- Uma sobrecarga do operador de inserção («) que insere uma representação de ponto flutuante do número de ponto fixo no objeto de fluxo de saída passado como parâmetro.

Executando este código:

```
#include <iostream>

int principal( void ){

    Fixo      a;
    Const fixo b( 10 ); const
    fixo c( 42.42f ); Const fixo d( b );

    a = Fixo( 1234.4321f );

    std::cout << "a é std::cout" << a << std::endl; << b
    << "b é" << std::endl; << c <<
    std::cout << "c é std::cout" << std::endl; << d << std::endl;
    << "d é"

    std::cout << "a é" << a.toInt() << << " como inteiro" << std::endl; como
    std::cout << "b é std::cout" << b.toInt() << << c.toInt() << " inteiro" << std::endl; como inteiro"
    << "c é std::cout << "d é" << << d.toInt() << << std::endl; como inteiro" <<
    std::endl;


    retornar 0;
}
```

Deve produzir algo semelhante a:

```
$> ./a.out
Construtor padrão chamado
Construtor int chamado
Construtor float chamado
Construtor de cópia chamado
Operador de atribuição de cópia chamado
Construtor float chamado
Operador de atribuição de cópia chamado
Destruído chamado
a é 1234,43
b é 10
c é 42,4219
d é 10
a é 1234 como número inteiro
b é 10 como número inteiro
c é 42 como número inteiro
d é 10 como número inteiro
Destruído chamado
Destruído chamado
Destruído chamado
Destruído chamado
$>
```

# Capítulo VI

## Exercício 02: Agora estamos falando

	Exercício 02
Agora estamos a falar	
Diretório de entrega: ex02/	
Arquivos a serem entregues: Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp Funções permitidas: roundf (de <cmath>)	

Adicione funções de membro públicas à sua classe para sobrecarregar os seguintes operadores:

- Os 6 operadores de comparação: >, <, >=, <=, == e !=.
- Os 4 operadores aritméticos: +, -, \* e /.
- Os 4 operadores de incremento/decremento (pré-incremento e pós-incremento, pré-decremento e pós-decremento), que aumentarão ou diminuirão o valor do ponto fixo do menor  $\tilde{y}$  representável, como  $1 + \tilde{y} > 1$ .

Adicione estas quatro funções de membro sobrecarregadas públicas à sua classe:

- Uma função de membro estático min que toma como parâmetros duas referências em números de ponto fixo e retorna uma referência ao menor.
- Uma função de membro estático min que toma como parâmetros duas referências a **constante** números de ponto fixo e retorna uma referência ao menor.
- Uma função de membro estático max que usa como parâmetros duas referências em números de ponto fixo e retorna uma referência ao maior.
- Uma função de membro estática máxima que toma como parâmetros duas referências a **constante** números de ponto fixo e retorna uma referência ao maior.

Cabe a você testar todos os recursos de sua classe. No entanto, executando o código abaixo:

```
#include <iostream>

int principal( vazio ){

    Fixo          a;
    const fixo     b( Fixo( 5.05f ) * Fixo( 2 ) );

    std::cout << a << std::endl; std::cout <<
    ++a << std::endl; std::cout << a <<
    std::endl; std::cout << a++ << std::endl;
    std::cout << a << std::endl;

    std::cout << b << std::endl;

    std::cout << Fixed::max( a, b ) << std::endl;

    retornar 0;
}
```

Deve produzir algo como (para maior legibilidade, o construtor/destruidor mes  
sábios são removidos no exemplo abaixo):


```
$> ./a.out 0
0,00390625
0,00390625
0,00390625
0,0078125
10,1016
10.1016
$>
```



Se alguma vez você fizer uma divisão por 0, é aceitável que o programa  
falhas

# Capítulo VII

## Exercício 03: BSP

	Exercício 03
BSP	
Diretório de entrega: ex03/	
Arquivos a serem entregues: Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp, Point.{h, hpp}, Point.cpp, bsp.cpp Funções permitidas: roundf ( de <cmath>)	

Agora que você tem uma classe **Fixed** funcional , seria bom usá-la.

Implemente uma função que indique se um ponto está dentro de um triângulo ou não.  
Muito útil, não é?



BSP significa particionamento de espaço binário. De nada. :)



Você pode passar neste módulo sem fazer o exercício 03.

Vamos começar criando a classe **Ponto** na Forma Canônica Ortodoxa que representa um ponto 2D:

- **Membros privados:**

- Um atributo const fixo x.
- Um atributo const fixo y.
- Qualquer outra coisa útil.

- **Membros públicos:**

- Um construtor padrão que inicializa x e y como 0.
- Um construtor que usa como parâmetros dois números de ponto flutuante constantes.  
Ele inicializa x e y com esses parâmetros.
- Um construtor de cópia.
- Uma sobrecarga do operador de atribuição de cópia.
- Um destruidor.
- Qualquer outra coisa útil.

Para finalizar, implemente a seguinte função no arquivo apropriado:

```
bool bsp( Ponto constante a, Ponto constante b, Ponto constante c, Ponto constante ponto);
```

- a, b, c: Os vértices do nosso amado triângulo.
- ponto: O ponto a ser verificado.
- Retorna: Verdadeiro se o ponto estiver dentro do triângulo. Falso caso contrário.  
Assim, se o ponto for um vértice ou aresta, retornará False.

Implemente e entregue seus próprios testes para garantir que sua classe se comporte conforme o esperado.