



## 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*

# 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 conversando</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 com Classes" (fonte: [Wikipedia](#)).*

O objetivo desses módulos é apresentá-lo à **programação orientada a objetos**. Esse será o ponto de partida de sua jornada em C++. Muitas linguagens são recomendadas para aprender OOP. Decidimos escolher o C++, pois ele é derivado do seu velho amigo C. Como essa é uma linguagem complexa, e para manter as coisas simples, seu código obedecerá ao padrão C++98.

Sabemos que o C++ moderno é muito diferente em vários aspectos. Portanto, se você quiser se tornar um desenvolvedor C++ proficiente, cabe a você ir além do 42 Common Core!

# Capítulo II

## Regras gerais

### Compilação

- 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 de exercícios serão nomeados da seguinte forma: ex00, ex01, ... , exn
- Nomeie seus arquivos, classes, funções, funções de membro e atributos conforme exigido nas diretrizes.
- Escreva os nomes das classes no formato **UpperCamelCase**. Os arquivos que contêm código de classe sempre serão nomeados 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 que contenha 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 um caractere de nova linha e exibidas na saída padrão.
- *Adeus Norminette!* Nenhum estilo de codificação é imposto nos módulos C++. Você pode seguir seu estilo favorito. Mas lembre-se de que um código que seus colegas avaliadores não conseguem entender é um código que eles não podem avaliar. Faça o possível para escrever um código limpo e legível.

### Permitido/Proibido

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

- Você tem permissão para usar quase tudo da biblioteca padrão. Portanto, em vez de se ater ao que já sabe, seria inteligente usar o máximo possível as versões em C++ das funções em C com as quais 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 as usar, sua nota será 0 e pronto.



- Observe que, a menos que explicitamente declarado de outra forma, o namespace de uso `<ns_name>` e palavras-chave amigas são proibidas. Caso contrário, sua nota será -42.
- **Você tem permissão para usar o STL somente no Módulo 08.** Isso significa: nada de **contêineres** (vetor/lista/mapa/e assim por diante) e nada de **algoritmos** (qualquer coisa que exija a inclusão do cabeçalho `<algorithm>`) até lá. Caso contrário, sua nota será -42.

### Alguns requisitos de design

- O vazamento de memória também ocorre em C++. Quando você aloca memória (usando o comando `new`), você deve evitar **vazamentos de memória**.
- Do Módulo 02 ao Módulo 08, suas aulas devem ser elaboradas 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 dos seus cabeçalhos independentemente dos outros. Portanto, eles devem incluir todas as dependências de que precisam. No entanto, você deve evitar o problema da inclusão dupla adicionando **proteções de inclusão**. Caso contrário, sua nota será 0.

### Leia-me

- Você pode acrescentar alguns arquivos adicionais, se necessário (ou seja, para dividir seu código). Como essas tarefas não são verificadas por um programa, sinta-se à vontade para fazer isso, 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! De fato, faça isso.
- Por Odin, por Thor! Use seu cérebro!!!



Você terá de implementar muitas classes. Isso pode parecer entediante, a menos que você saiba programar seu editor de texto favorito.



Você tem certa liberdade para concluir os exercícios. Entretanto, siga as regras obrigatórias e não seja preguiçoso. Você perderia muitas informações úteis! Não hesite em ler sobre conceitos teóricos.



# Capítulo III

## Novas regras

De agora em diante, todas as suas classes devem ser projetadas na **Forma Canônica Ortodoxa**, a menos que explicitamente declarado de outra forma. Em seguida, elas implementarão as quatro funções de membro exigidas abaixo:


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

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



## Capítulo IV

# Exercício 00: Minha primeira aula na forma canônica ortodoxa

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

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

Leia este artigo de 3 páginas ([1](#), [2](#), [3](#)) para descobrir que você não o faz. Vá em frente, leia.

Até hoje, cada número que você usava em seu código era basicamente um número 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 os números inteiros e os números de ponto flutuante têm características opostas.

Mas hoje, as coisas vão mudar. Você descobrirá um novo e incrível tipo de número: **números de ponto fixo**! Sempre ausentes dos tipos escalares da maioria das linguagens, os números de ponto fixo oferecem um equilíbrio valioso entre desempenho, exatidão, intervalo e precisão. Isso explica por que os números de ponto fixo são particularmente aplicáveis à computação gráfica, ao processamento de som ou à programação científica, só para citar alguns exemplos.

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



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

- Membros particulares:
  - Um número **inteiro** para armazenar o valor do número de ponto fixo.
  - Um número **inteiro constante estático** para armazenar o número de bits fracionários. Seu valor será sempre o literal de número inteiro 8.
- Membros do público:
  - 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 destrutor.
  - Uma função membro `int getRawBits( void ) const;` que retorna o valor bruto do valor de ponto fixo.
  - Uma função membro `void setRawBits( int const raw );` que define o valor bruto do número de ponto fixo.

Executando este código:

```
#include <iostream>

int main( void ) {

    Fixo a; Fixo
    b( a ); Fixo
    c;

    c = b;

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


}
```

O resultado deve ser algo semelhante a:

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

## Capítulo V

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

	Exercício 01
Em direção a uma classe de números de ponto fixo mais útil	
Diretório de entrada: <i>ex01/</i>	
Arquivos a serem entregues : <i>Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp</i>	
Funções permitidas : <i>roundf</i> (de <i>&lt;cmath&gt;</i> )	

O exercício anterior foi um bom começo, mas nossa classe é bastante inútil. Ela só pode representar o valor 0,0.

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

- Um construtor que recebe um **número inteiro constante** como parâmetro. Ele o converte para o valor de ponto fixo correspondente. O valor dos bits fracionários é inicializado em 8, como no exercício 00.
- Um construtor que recebe um **número de ponto flutuante constante** como parâmetro. Ele o converte para o valor de ponto fixo correspondente. O valor dos 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 membro `int toInt( void ) const;` que converte o valor de ponto fixo em um valor inteiro.

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

- Uma sobrecarga do operador `insertion (")` 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 main( void ) {

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

    a = Fixed( 1234.4321f );

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

    std::cout << "a is " << a.toInt() << " as integer" <<
    std::endl; std::cout << "b is " << b.toInt() << " as integer"
    << std::endl; std::cout << "c is " << c.toInt() << " como
    inteiro" << std::endl; std::cout << "d é " << d.toInt() << "
    como inteiro" << std::endl;
}
```


O resultado deve ser algo semelhante a:

```
$> ./a.out
Default constructor called
Int constructor called
Float constructor called
Copy constructor called
Operador de atribuição de cópia
chamado Float constructor chamado
Operador de atribuição de cópia
chamado Destructor chamado
a é 1234,43
b é 10
c é 42,4219
d é 10
a é 1234 como um número inteiro
b é 10 como um número inteiro
c é 42 como um número inteiro
d é 10 como
inteiro Destructor
called Destructor
called Destructor
called Destructor
called
```



## Capítulo VI

### Exercício 02: Agora estamos conversando

	Exercício 02
Agora estamos falando	
Diretório de entrada: <i>ex02/</i>	
Arquivos a serem entregues: <i>Makefile</i> , <i>main.cpp</i> , <i>Fixed.{h, hpp}</i> , <i>Fixed.cpp</i>	
Funções permitidas: <i>roundf</i> (de <i>&lt;cmath&gt;</i> )	

Adicione funções de membro público 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 de ponto fixo a partir do menor  $\epsilon$  representável, como  $1 + \epsilon > 1$ .

Adicione essas quatro funções-membro públicas sobrecarregadas à sua classe:

- Uma função de membro estático `min` que recebe como parâmetros duas referências em números de ponto fixo e retorna uma referência ao menor deles.
- Uma função de membro estático `min` que recebe como parâmetros duas referências à **constante** números de ponto fixo e retorna uma referência ao menor deles.
- Uma função de membro estático `max` que recebe como parâmetros duas referências em números de ponto fixo e retorna uma referência ao maior deles.
- Uma função de membro estático `max` que recebe como parâmetros duas referências à **constante** números de ponto fixo e retorna uma referência ao maior deles.





Cabe a você testar todos os recursos da sua classe. No entanto, ao executar o código abaixo:

```
#include <iostream>

int main( void ) {

    Fixo Fixo    a;
    const       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 facilitar a leitura, as mensagens do construtor/destrutor foram removidas no exemplo abaixo):

```
$> ./a.out
0
0.00390625
0.00390625
0.00390625
0.0078125
10.1016
10.1016
$>
```

# Capítulo VII

## Exercício 03:

### BSP

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

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 Binary space partitioning (particionamento de espaço binário). De nada. :)



Você pode ser aprovado neste módulo sem fazer o exercício 03.



Vamos começar criando a classe **Point** in Orthodox Canonical Form que representa um ponto 2D:

- Membros particulares:
  - Um atributo fixo const x.
  - Um atributo fixo const y.
  - Qualquer outra coisa útil.
- Membros do público:
  - Um construtor padrão que inicializa x e y como 0.
  - Um construtor que recebe 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 destrutor.
  - Qualquer outra coisa útil.

Para concluir, implemente a seguinte função no arquivo apropriado:

```
bool bsp( Point const a, Point const b, Point const c, Point const point);
```

- a, b, c: Os vértices de nosso amado triângulo.
- ponto: O ponto a ser verificado.
- Retorna: True se o ponto estiver dentro do triângulo. Caso contrário, falso. Portanto, se o ponto for um vértice ou uma aresta, ele retornará False.

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

