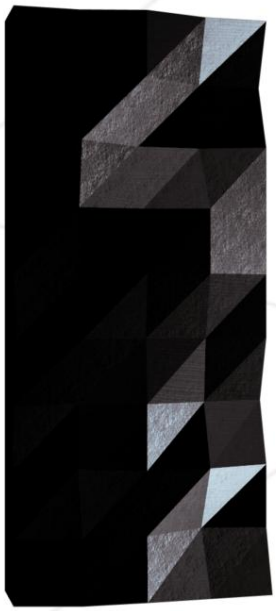

C++ - Módulo 02

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: 8

Conteúdo

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 apresentar **a Programação Orientada a Objetos.**
Este será o ponto de partida de sua jornada em C++. Muitas linguagens são recomendadas para aprender OOP. Decidimos escolher C++ porque ele 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.

Estamos cientes de 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 após o 42 Common Core!

Capítulo II

Regras gerais

Compilando

- 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 dos exercícios serão nomeados desta forma: ex00, ex01, ... , ex
- Nomeie seus arquivos, classes, funções, funções de membro e atributos conforme exigido em as diretrizes.
- Escreva nomes de 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 finalizadas com uma nova linha caractere e exibido na saída padrão.
- *Adeus Norminette!* Nenhum estilo de codificação é imposto nos módulos C++. Você pode seguir o seu favorito. Mas tenha em mente 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. É hora de C++! Portanto:

- Você tem permissão para usar quase tudo da biblioteca padrão. Assim, em vez de se ater ao que você já sabe, seria inteligente usar o máximo possível as versões C++ das funções C com as quais você está acostumado.
- Entretanto, você não pode usar nenhuma outra biblioteca externa. Isso significa que C++ 11 (e formas derivadas) e bibliotecas 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, salvo indicação explícita em contrário, o namespace using <ns_name> e palavras-chave de amigos são proibidas. Caso contrário, sua nota será -42.
- **É permitido utilizar o STL somente nos Módulos 08 e 09.** Isso significa: nenhum **contêiner** (vetor/lista/mapa/e assim por diante) e nenhum **algoritmo** (qualquer coisa que exija a inclusão do cabeçalho <algorithm>) até então. Caso contrário, sua nota será -42.

Alguns requisitos de design

- 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 planejadas no **estilo Ortodoxo Forma Canônica, exceto quando explicitamente indicado de outra forma.**
- Qualquer implementação de função colocada em um arquivo de cabeçalho (exceto 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 necessitam. No entanto, você deve evitar o problema da 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, fique à vontade para fazê-lo, desde que entregue os arquivos obrigatórios.
- Às vezes, as orientações 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 consiga criar um script em seu editor de texto favorito.

Você tem uma certa liberdade para completar os exercícios. Porém, siga as regras obrigatórias e não seja preguiçoso. Você faria perca muitas informações úteis! Não hesite em ler sobre conceitos teóricos.

Capítulo III

Novas regras

De agora em diante, todas as suas aulas deverão ser elaboradas na **Forma Canônica Ortodoxa,** salvo indicação explícita em contrário. Em seguida, eles implementarão as quatro funções-membro exigidas abaixo:

- Construtor padrão
- Copiar construtor
- Copiar operador de atribuição
- Destruidor

Divida o código da sua 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 para entrega: Makefile, main.cpp, Fixo.{h, hpp}, Fixo.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 número inteiro ou de ponto flutuante, ou qualquer uma de suas variantes (curto, char, longo, duplo e assim por diante). Depois de ler o artigo acima, é seguro assumir que números inteiros e 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, alcance 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, 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 Universidade de Berkeley, leia esta seção de sua página da Wikipedia.

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

• Membros privados:

ÿ Um **número inteiro** para armazenar o valor numérico de ponto fixo.

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

• Membros públicos:

ÿ Um construtor padrão que inicializa o valor numérico de ponto fixo como 0.

ÿ Um construtor de cópia.

ÿ Uma sobrecarga do operador de atribuição de cópia.

ÿ Um destruidor.

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

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

Executando este código:

```
#include <iostream>

interno principal( vazio ) {

    Corrigido um;
    Fixo b(uma);
    Fixo c;

    c=b;

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


    retornar 0;
}
```

Deve gerar 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 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 números de ponto fixo mais útil	
Diretório de entrega: ex01/	
Arquivos para entrega: 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. Ele só pode representar o valor 0,0.

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

- Um construtor que usa um **número inteiro constante** como parâmetro.
  Ele o converte no valor de ponto fixo correspondente. O valor dos bits fracionários é inicializado em 8 como no exercício 00.

- Um construtor que usa um **número constante de ponto flutuante** como parâmetro.
  Ele o converte no 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 da classe **Fixa :**

- 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>

interno principal( vazio ) {

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

    a = Fixo ( 1234.4321f );

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

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

    retornar 0;
}
```

Deve gerar algo semelhante a:

```
$> ./a.out
Construtor padrão chamado
Construtor int chamado
Construtor flutuante chamado
Construtor de cópia chamado
Operador de atribuição de cópia chamado
Construtor flutuante chamado
Operador de atribuição de cópia chamado
Destruidor 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
Destruidor chamado
Destruidor chamado
Destruidor chamado
Destruidor chamado
$>
```

Capítulo VI

Exercício 02: Agora estamos conversando

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

Adicione funções-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 a partir do menor $\ddot{y}$ representável , como $1 + \ddot{y} > 1$.

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

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

Cabe a você testar todos os recursos da sua classe. Porém, executando o código abaixo:

```
#include <iostream>

interno principal( vazio ) {

    Fixo
    Const fixa um; 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 << Fixo::max( a, b ) << std::endl;

    retornar 0;
}
```

Deve gerar algo como (para maior legibilidade, a mensagem construtor/destruidor sábios são removidos no exemplo abaixo):

```
$> ./a.out 0

0,00390625
0,00390625
0,00390625
0,0078125
10,1016
10.1016$>
```

Se 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 para entrega: 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 **fixa** 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 **Point** na Forma Canônica Ortodoxa que representa um ponto 2D:

- Membros privados:
  - ÿ Um atributo const fixo x.
  - ÿ Um atributo const fixo y.
  - ÿ Qualquer outra coisa útil.
- Membros públicos:
  - ÿ Um construtor padrão que inicializa x e y como 0.
  - ÿ Um construtor que toma como parâmetros dois números constantes de ponto flutuante. Ele inicializa x e y com esses parâmetros.
  - ÿ Um construtor de cópia.
  - ÿ Uma sobrecarga do operador de atribuição de cópia.
  - ÿ Um destruidor.
  - ÿ Qualquer outra coisa útil.

Para concluir, implemente a seguinte função no arquivo apropriado:

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

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

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

Capítulo VIII

Envio e avaliação por pares

Entregue sua tarefa em seu repositório Git normalmente. Somente o trabalho dentro do seu repositório será avaliado durante a defesa. Não hesite em verificar os nomes de suas pastas e arquivos para garantir que estão corretos.

?????????? XXXXXXXXXX = $3$$d6f957a965f8361750a3ba6c97554e9f