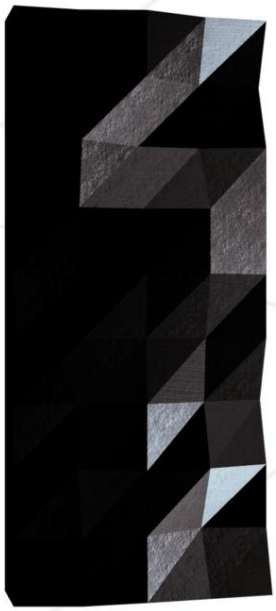

C++ - Módulo 02

Polimorfismo ad-hoc, sobrecarga de operador e forma canônica

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

Versão: 8

Índice

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).

C++ é uma linguagem de programação compilada que permite a programação sob vários paradigmas, incluindo programação processual, programação orientada a objetos e programação genérica. Seu bom desempenho e sua compatibilidade com C fazem dela uma das linguagens de programação mais utilizadas em aplicações onde o desempenho é crítico (fonte: Wikipedia).

Esses módulos têm como objetivo apresentá-lo à **Programação Orientada a Objetos.** Várias linguagens são recomendadas para aprender OOP. Por ser derivado do seu bom e velho amigo C, escolhemos a linguagem C++. Porém, por ser uma linguagem complexa e para não complicar sua tarefa, você cumprirá o padrão C++98.

Percebemos que o C++ moderno é diferente em muitos aspectos. Se você deseja melhorar seu domínio de C++, cabe a você buscar o núcleo comum de 42!

Capítulo II

Instruções gerais

Compilação

• Compile seu código com c++ e os sinalizadores -Wall -Wextra -Werror • Seu código deverá ser compilado se você adicionar o sinalizador -std=c++98

Convenções de formato e nomenclatura

• Os arquivos de exercícios terão os seguintes nomes: ex00, ex01, ..., ex

• Nomeie seus arquivos, classes, funções, funções-membro e atributos. homenagens conforme especificado nas instruções.

• Escreva os nomes das suas classes no **formato UpperCamelCase.** Os arquivos que contêm o código de uma classe terão o nome desta última. Por exemplo: ClassName.hpp/ClassName.h, ClassName.cpp ou ClassName.tpp.
Portanto, se um arquivo de cabeçalho contiver a definição de uma classe "BrickWall", seu nome será BrickWall.hpp.

• A menos que especificado de outra forma, todas as mensagens devem ser finalizadas com um retorno à linha e ser exibido na saída padrão.

• *Ciao Norminette!* Nenhum padrão é imposto durante os módulos C++. Você pode seguir o estilo de sua escolha. Mas tenha em mente que o código que seus colegas não conseguem entender é um código que seus colegas não conseguem avaliar. Portanto, faça o seu melhor para produzir um código limpo e legível.

O que é permitido e o que não é

A linguagem C acabou por enquanto. É hora de começar com C++! Portanto :

• Você pode usar quase toda a biblioteca padrão. Portanto, em vez de permanecer em terreno familiar, tente usar o máximo possível as versões C++ das funções C com as quais você está familiarizado.

• Entretanto, você não pode usar nenhuma outra biblioteca externa.
O que significa que C++ 11 (e derivados) e o conjunto Boost são proibidos. Além disso, certas funções permanecem proibidas. Usar as seguintes funções resultará

na nota 0: *printf(), *alloc() e free().

- Salvo indicação explícita em contrário, palavras-chave que usam namespace <ns_name> e amigo são proibidos. Seu uso resultará em uma pontuação de -42.
- **Você só tem direito ao STL nos Módulos 08 e 09.** Até então, é proibido o uso de **Containers** (vetor/lista/mapa/etc.) e Algoritmos **(qualquer** coisa que requeira incluir <algoritmo>). Caso contrário, você obterá uma pontuação de -42.

Algumas obrigações de design

- Vazamentos de memória também existem em C++. Ao alocar memória (usando a palavra-chave new), você **não deverá ter vazamentos de memória.**
- Do Módulo 02 ao Módulo 09, suas aulas devem estar em conformidade com a **forma canônica, conhecida como Coplian, a menos que seja explicitamente especificado o contrário.** • Uma função implementada em um arquivo de cabeçalho (exceto no caso de uma função de modelo) será equivalente a uma pontuação de 0. • Você deve ser capaz de usar seus arquivos de cabeçalho separadamente u É por isso que deverão incluir todas as dependências que serão necessárias para eles. No entanto, você deve evitar o problema da dupla inclusão protegendo-os com **protetores de inclusão.** Caso contrário, sua pontuação será 0.

Leia-me

- Se necessário, você pode renderizar arquivos adicionais (por exemplo, para separar seu código em mais arquivos). Como seu trabalho não será avaliado por um programa, faça o que achar melhor, desde que torne os arquivos obrigatórios.
- As instruções para um exercício podem parecer simples, mas os exemplos por vezes contêm instruções adicionais que não são explicitamente solicitadas.
- Leia cada módulo completamente antes de começar! Realmente. • Por Odin, por Thor! Use seu cérebro! ! !

Você terá que implementar um bom número de aulas, o que pode ser difícil... ou não!
Pode haver uma maneira de facilitar sua vida usando seu editor de texto favorito.

Você tem bastante liberdade para resolver os exercícios.
No entanto, siga as instruções e não se limite ao mínimo, pois você pode perder conceitos interessantes.
Sinta-se à vontade para ler alguma teoria.

Capítulo III

Nova instrução

A partir de agora, suas aulas deverão estar em conformidade com a **forma canônica do Coplian,** salvo indicação em contrário. Isso significa que eles devem incluir as seguintes quatro funções-membro:

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

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

Capítulo IV

Exercício 00: Meu primeiro canhão

	Exercício: 00
Meu primeiro canhão	
Pasta de renderização: ex00/	
Arquivos para renderizar: Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp Funções proibidas:	
Nenhuma	

Se você pensava que sabia números inteiros e flutuantes, este artigo de 3 páginas (1, 2, 3) irá mostrar-lhe que este provavelmente não é o caso. Vamos, leia.

Até hoje, todos os números usados em seu código eram inteiros, flutuantes ou possivelmente seus derivados (short, char, long, double, etc.).
Depois de ler o artigo acima, parece óbvio que as características dos números inteiros e dos números de ponto flutuante são opostas.

Mas hoje as coisas vão mudar. Você descobrirá um conceito novo e interessante: a **representação de números de ponto fixo !** Sempre ausentes dos tipos escalares da maioria das linguagens, os números de ponto fixo oferecem um equilíbrio interessante entre desempenho, correção, alcance e precisão. Isto explica por que estes números são amplamente utilizados em imagens digitais, som ou programação científica, para citar apenas três.

Como C++ não possui números de ponto fixo, você irá adicioná-los. Este item de Berkeley é um bom lugar para começar. Se você não sabe o que é UC Berkeley, leia esta parte de sua página da Wikipedia.

Crie uma classe na forma canônica para representar um número de ponto fixo.

- Membros privados: ÿ
  Um **número inteiro** para armazenar o valor do número de ponto fixo. ÿ Um
  número inteiro constante estático para armazenar o número de bits da peça
  fracionário, e cujo valor será sempre o número inteiro literal 8.

- Membros públicos:
  ÿ Um construtor padrão que inicializará o valor do número de ponto fixo
  para 0.

  ÿ Um construtor de cópia. ÿ Uma
  sobrecarga do operador de atribuição.
  ÿ Um destruidor.

  ÿ Uma função membro int getRawBits(void) const;
  que retorna o valor do número de ponto fixo sem convertê-lo.
  ÿ Uma função membro void setRawBits(int const raw);
  que inicializa o valor do número do ponto fixo com aquele passado como parâmetro.

Execute este código:

```
#incluir <iostream>

Inteiro principal( vazio ) {

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

    c = b;

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


    retornar 0;
}
```

Deve mostrar este resultado:

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

Construtor padrão chamado
Operador de atribuição de cópia chamado getRawBits
função membro chamada getRawBits função membro
chamada 0

função membro getRawBits chamada 0

função membro getRawBits chamada 0

Destruidor chamado
Destruidor chamado
Destruidor chamado
$>
```

Capítulo V

Exercício 01: Primeiros passos para uma aula útil

	Exercício: 01
Primeiros passos para uma aula útil	
Pasta de renderização: ex01/	
Arquivos para renderizar: Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp Funções	
permitidas: roundf (de <cmath>)	

O exercício anterior foi um bom ponto de partida, mas a nossa turma ainda não
não há muito interesse. Ele só pode representar o valor 0,0.

Adicione os seguintes construtores e funções-membro à sua classe em público:

- Um construtor que toma um **número inteiro constante** como parâmetro e o converte em um ponto fixo. O número de bits da parte fracionária é inicializado em 8 como no exercício 00.

- Um construtor que toma um **float constante** como parâmetro e o converte em um ponto fixo. O número de bits da parte fracionária é inicializado em 8 como no exercício 00.

- Uma função membro float toFloat(void) const;
  que converte o valor de ponto fixo em um número de ponto flutuante.
- Uma função membro int toInt(void) const;
  que converte o valor do ponto fixo em um número inteiro.

Adicione também a seguinte função aos seus arquivos de classe **fixa :**

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

Execute este código:

```
#incluir <iostream>

Inteiro principal( vazio ) {

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

    a = Fixo( 1234.4321f );

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

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

    retornar 0;
}
```

Deve mostrar este resultado:

```
$> ./a.fora
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
Destruidor chamado
a é 1234,43
b é 10
c é 42,4219
d é 10
a é 1234 como inteiro
b é 10 como inteiro
c é 42 como inteiro
d é 10 como inteiro
Destruidor chamado
Destruidor chamado
Destruidor chamado
Destruidor chamado
$>
```

Capítulo VI

Exercício 02: Agora podemos conversar

	Exercício: 02
Agora podemos conversar	
Pasta de renderização: ex02/	
Arquivos para renderizar: 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 e decremento (pré-incremento e pós-incremento, pré-decremento e pós-decremento) que diminuirão o valor do número de ponto fixo da unidade *ÿ* tal que 1 + *ÿ* > 1.

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

- Uma função membro estática que toma como parâmetros duas referências em números de ponto fixo e retorna o menor deles.
- Uma função membro estática min que toma como parâmetros duas referências a números **constantes** de ponto fixo e que retorna o menor deles.
- Uma função membro estática máxima tomando como parâmetros duas referências em números de ponto fixo e retorna o maior deles.
- Uma função membro estática max que toma como parâmetros duas referências a números **constantes** de ponto fixo e que retorna o maior deles.

Cabe a você testar todos os recursos do seu código. Mas execute este código:

```
#incluir <iostream>

Inteiro 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 << Fixo::max( a, b ) << std::endl;

    retornar 0;
}
```

Deverá exibir este resultado (para maior legibilidade, as mensagens do fabricante e do destrutivos foram removidos):

```
$> ./a.out 0

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

Se você dividir por 0, é aceitável que o programa trave.

Capítulo VII

Exercício 03: BSP

	Exercício: 03
BSP	
Pasta de renderização: ex03/	
Arquivos para renderizar: 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** funcionando , não seria bom usá-la?

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

BSP significa Particionamento de Espaço Binário. Não me agradeça. :)

Você pode concluir este módulo sem o exercício 03.

Primeiro, crie uma classe **Point** em forma canônica para representar um ponto 2D:

- Membros privados:
  - ÿ Um atributo Constante fixa x.
  - ÿ Um atributo y de constante fixa. ÿ E qualquer coisa que possa ser útil para você.
- Membros públicos:
  - ÿ Um construtor padrão que inicializa x e y como 0. ÿ Um construtor que usa dois valores flutuantes constantes como parâmetros e inicializa x e y com eles. ÿ Um construtor de cópia. ÿ Uma sobrecarga do operador de atribuição.
  - ÿ Um destruidor.
  - ÿ E qualquer coisa que possa ser útil para você.

Para finalizar, implemente a seguinte função no arquivo correspondente:

```
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 avaliado.
- Retorna: Verdadeiro se o ponto estiver dentro do triângulo. Caso contrário, falso. Isso significa que, se o ponto for um vértice ou colocado em uma aresta, a função retornará False.

Escreva e renderize seus próprios testes para demonstrar que sua aula funciona conforme solicitado.

Capítulo VIII

Submissão e avaliação por pares

Envie seu trabalho para o repositório Git normalmente. Somente o trabalho presente em seu depósito será avaliado na defesa. Verifique novamente os nomes de suas pastas e arquivos para que atendam às solicitações do sujeito.

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