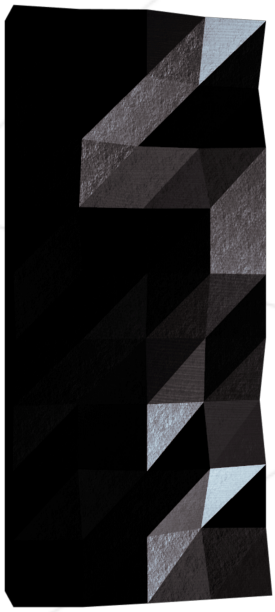

C++ - Módulo 02

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

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

Versão: 8.2

Conteúdo

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:Wikipédia).

O objetivo destes módulos é apresentar a você**Programação Orientada a Objetos**Este será o ponto de partida da sua jornada em C++. Muitas linguagens são recomendadas para aprender POO, mas decidimos escolher C++ por ser derivada da sua velha amiga C. Por ser 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 bastante diferente em muitos aspectos. Portanto, se você quer se tornar um desenvolvedor C++ proficiente, cabe a você ir além, seguindo os 42 princípios básicos!

Capítulo II

Regras gerais

Compilando

- Compile seu código comc++e as bandeiras -Parede -Wextra -Werror
- Seu código ainda deverá ser compilado se você adicionar o sinalizador -padrão=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 nas diretrizes.
- Escreva os nomes das classes em**UpperCamelCase**formato. Arquivos contendo código de classe sempre serão nomeados de acordo com o nome da classe. Por exemplo: NomeDaClasse.hpp/NomeDaClasse.h, NomeDaClasse.cpp,ouNomeDaClasse.tpp.Então, se você tiver um arquivo de cabeçalho contendo a definição de uma classe "BrickWall" que representa uma parede de tijolos, seu nome seráBrickWall.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 de que código que seus avaliadores não conseguem entender é código que eles não podem avaliar. Faça o possível para escrever um código limpo e legível.

Permitido/Proibido

Você não está mais programando em C. Hora de C++! Portanto:

- Você pode usar quase tudo da biblioteca padrão. Portanto, em vez de se ater ao que você já conhece, seria inteligente usar as versões em C++ das funções em C com as quais você está acostumado, sempre que possível.
- No entanto, você não pode usar nenhuma outra biblioteca externa. Isso significa C++11 (e formas derivadas) eImpulsionarBibliotecas são proibidas. As seguintes funções também são proibidas: *printf(), *alocação()elivre().Se você usá-los, sua nota será 0 e pronto.

- Observe que, a menos que explicitamente declarado de outra forma, ousando namespace <ns_name>e amigoPalavras-chave são proibidas. Caso contrário, sua nota será -42.
- **Você tem permissão para usar o STL somente nos Módulos 08 e 09.**Isso significa: não **Recipientes**(vetor/lista/mapa e assim por diante) e não**Algoritmos**(qualquer coisa que requeira incluir o <algoritmo>cabeçalho) até então. Caso contrário, sua nota será -42.

Alguns requisitos de design

- Vazamento de memória também ocorre em C++. Quando você aloca memória (usando onovo palavra-chave), você deve evitar**vazamentos de memória**.
- Do Módulo 02 ao Módulo 09, suas aulas devem ser projetadas no**Forma Canônica Ortodoxa, exceto quando explicitamente declarado 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. Portanto, eles devem incluir todas as dependências necessárias. No entanto, você deve evitar o problema de inclusão dupla adicionando**incluir guardas**. Caso contrário, sua nota será 0.

Leia-me

- Você pode adicionar alguns arquivos adicionais se precisar (por exemplo, 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! Sério, faça!
- Por Odin, por Thor! Use seu cérebro!!!

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

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

Você tem 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

Novas regras

A partir de agora, todas as suas aulas devem ser projetadas no**Forma Canônica Ortodoxa**, a menos que explicitamente indicado o contrário. Eles então implementarão as quatro funções-membro necessárias abaixo:

- Construtor padrão
- Construtor de cópias
- 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 origem (.cpp) contém a implementação.

Capítulo IV

Exercício 00: Minha Primeira Aula na Forma Canônica Ortodoxa

	Exercício: 00
Minha primeira aula sobre a forma canônica ortodoxa	
Diretório de entrega: *ex*00/	
Arquivos para entregar: Makefile, main.cpp, Fixo.{h, hpp}, Fixo.cpp	
Funções proibidas: Nenhum	

Você acha que entende de números inteiros e 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 que você usava em seu código era basicamente um número inteiro ou um número 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 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 ausentes nos 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, processamento de som ou programação científica, só para citar alguns.

Como C++ não possui números de ponto fixo, você irá adicioná-los.Este artigode Berkeley é um bom começo. Se você não tem ideia do que é a Universidade de Berkeley, leiaesta seçãoda sua página na Wikipédia.

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

- Membros privados:
  - Um**inteiro**para armazenar o valor do número de ponto fixo.
  - UM**constante estática inteira**para armazenar o número de bits fracionários. Seu valor será sempre o literal inteiro8.
- Membros públicos:
  - Um construtor padrão que inicializa o valor do número de ponto fixo para0.
  - Um construtor de cópias.
  - Uma sobrecarga do operador de atribuição de cópia.
  - Um destruidor.
  - Uma função membroint getRawBits(void) const; que retorna o valor bruto do valor de ponto fixo.
  - Uma função membrovazio setRawBits(int const raw); que define o valor bruto do número de ponto fixo.

Executando este código:

```
# incluir <iostream>

int 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;

    retornar0;
}
```

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

função 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
Em direção a uma classe de números de ponto fixo mais útil	
Diretório de entrega:*ex*01/	
Arquivos para entregar:Makefile, main.cpp, Fixo.{h, hpp}, Fixo.cpp	
Funções permitidas:roundf (de <cmath>)	

O exercício anterior foi um bom começo, mas nossa classe é bastante inútil. Ela só pode representar o valor0,0.

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

- Um construtor que pega um**inteiro constante**como parâmetro.
  Ele o converte para o valor de ponto fixo correspondente. O valor dos bits fracionários deve ser inicializado como 8, como no exercício 00.
- Um construtor que pega um**número de ponto flutuante constante**como parâmetro. Ele o converte para o valor de ponto fixo correspondente. O valor dos bits fracionários deve ser inicializado como 8, como no exercício 00.
- Uma função membroflutuar para flutuar (void) const; que converte o valor de ponto fixo em um valor de ponto flutuante.
- Uma função membroint toInt(void) const; que converte o valor de ponto fixo em um valor inteiro.

E adicione a seguinte função ao**Fixo**arquivos de classe:

- 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:

```
# incluir <iostream>

int principal(vazio) {

    Fixo um;
    Fixoconstanteb(10); Fixo
    constantec(42,42f); Fixo
    constanted( b );

    a = Fixo(1234.4321f);

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

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


    retornar0;
}
```

Deve gerar algo semelhante a:

```
$> ./a.out
Construtor padrão chamado
Int construtor chamado
Construtor flutuante chamado Construtor
de cópia chamado Operador de atribuição
de cópia chamado Construtor flutuante
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 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 estamos falando

	Exercício 02
Agora estamos falando	
Diretório de entrega:*ex*02*/*	
Arquivos para entregar:Makefile, main.cpp, Fixo.{h, hpp}, Fixo.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 pelo menor representávelϵ, tal que 1 +ϵ >1.

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

- Uma função de membro estáticominque recebe duas referências a números de ponto fixo como parâmetros e retorna uma referência ao menor deles.
- Uma função de membro estáticominque leva duas referências a**constante**números de ponto fixo como parâmetros e retorna uma referência ao menor deles.
- Uma função de membro estáticomáx.que recebe duas referências a números de ponto fixo como parâmetros e retorna uma referência ao maior deles.
- Uma função de membro estáticomáx.que leva duas referências a**constante**números de ponto fixo como parâmetros e retorna uma referência ao maior deles.

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

```
# incluir <iostream>

int principal(vazio) {

    Fixo um;
    Fixoconstante 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;

    retornar0;
}
```

Deve gerar algo como (para maior legibilidade, as mensagens de construtor/destruidor foram removidas 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 trave

Capítulo VII

Exercício 03: BSP

	Exercício 03
BSP	
Diretório de entrega: *ex*03/	
Arquivos para entregar: Makefile, main.cpp, Fixo.{h, hpp}, Fixo.cpp, Ponto.{h, hpp}, Ponto.cpp, bsp.cpp	
Funções permitidas: roundf (de <cmath>)	

Agora que você tem um funcional **Fixo** classe, seria legal usá-lo.

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 completar o exercício 03.

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

- Membros privados:
  - Um atributo fixo constx.
  - Um atributo fixo conste.
  - Qualquer outra coisa útil.
- Membros públicos:
  - Um construtor padrão que inicializaxeepara0.
  - Um construtor que recebe dois números de ponto flutuante constantes como parâmetros. Ele inicializaxeecom esses parâmetros.
  - Um construtor de cópias.
  - Uma sobrecarga do operador de atribuição de cópia.
  - Um destruidor.
  - 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 amado triângulo.
- ponto: O ponto a ser verificado.
- Retorna: Verdadeiro se o ponto estiver dentro do triângulo. Falso caso contrário. Portanto, se o ponto for um vértice ou uma aresta, retornará Falso.

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

Capítulo VIII

Submissão e Avaliação por Pares

Entregue sua tarefa em seuGitrepositório como de costume. Somente o trabalho dentro do seu repositório será avaliado durante a defesa. Não hesite em verificar novamente os nomes das suas pastas e arquivos para garantir que estejam corretos.

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