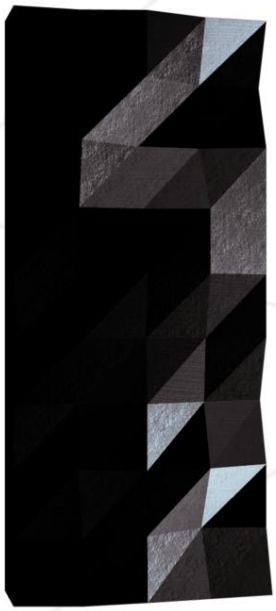

C++ - Módulo 06

Conversões C++

Resumo:
Este documento contém os exercícios do Módulo 06 dos módulos C++.

Versão: 5.1

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 poderia perca muitas informações úteis! Não hesite em ler sobre conceitos teóricos.

Capítulo III

Regra adicional

A regra a seguir se aplica a todo o módulo e não é opcional.

Para cada exercício, a conversão de tipo deve ser resolvida utilizando um tipo específico de casting. Sua escolha será verificada durante a defesa.

Capítulo IV

Exercício 00: Conversão de escalar tipos

	Exercício 00
Conversão de tipos escalares	
Diretório de entrega: ex00/	
Arquivos para entrega: Makefile, *.cpp, *.{h, hpp}	
Funções permitidas: qualquer função para converter de uma string em um int, float ou double. Isso ajudará, mas não fará todo o trabalho.	

Escreva uma classe estática ScalarConverter que conterá um método "convert" que toma como parâmetro uma representação em string de um literal C++ em sua forma mais comum. Este literal deve pertencer a um dos seguintes tipos escalares:

- Caracteres
- interno
- flutuar
- dobro

Exceto para parâmetros char, apenas a notação decimal será usada.

Exemplos de literais char: 'c', 'a', ...
Para simplificar as coisas, observe que caracteres não exibíveis não devem ser usados como entradas. Se uma conversão para char não puder ser exibida, imprime uma mensagem informativa.

Exemplos de literais int: 0, -42, 42...

Exemplos de literais flutuantes: 0.0f, -4.2f, 4.2f...
Você também tem que lidar com esses pseudo literais (você sabe, para a ciência): -inff, +inff e nanf.

Exemplos de literais duplos: 0,0, -4,2, 4,2...
Você também tem que lidar com esses pseudo literais (você sabe, por diversão): -inf, +inf e nan.

Escreva um programa para testar se sua classe funciona conforme o esperado.

Você deve primeiro detectar o tipo do literal passado como parâmetro, convertê-lo de string para seu tipo real e, em seguida, convertê-lo **explicitamente** para os outros três tipos de dados. Por último, exiba os resultados conforme mostrado abaixo.

Se uma conversão não fizer sentido ou estourar, exiba uma mensagem para informar ao usuário que a conversão de tipo é impossível. Inclua qualquer cabeçalho necessário para lidar com limites numéricos e valores especiais.

```
./convert 0
char: Não exibível int: 0 float:
0.0f
double: 0.0

./convert nan
char: impossível int:
impossível float: nanf

duplo: nan
./convert 42.0f
caractere:
'*' int: 42
flutuante: 42,0f
duplo: 42,0
```

Capítulo V

Exercício 01: Serialização

	Exercício: 01
Serialização	
Diretório de entrega: ex01/	
Arquivos para entrega: Makefile, *.cpp, *.{h, hpp}	
Funções proibidas: Nenhuma	

Implemente uma classe estática Serializer com os seguintes métodos:

uintptr_t serialize(Dados* ptr);
Ele pega um ponteiro e o converte para o tipo inteiro não assinado uintptr_t.

Dados* desserialize(uintptr_t raw);
Ele pega um parâmetro inteiro não assinado e o converte em um ponteiro para Dados.

Escreva um programa para testar se sua classe funciona conforme o esperado.

Você deve criar uma estrutura de dados não vazia (significa que possui membros de dados).

Use serialize() no endereço do objeto Data e passe seu valor de retorno para deserialize(). Em seguida, certifique-se de que o valor de retorno de deserialize() seja igual ao ponteiro original.

Não esqueça de entregar os arquivos da sua estrutura de dados.

Capítulo VI

Exercício 02: Identifique o tipo real

	Exercício: 02
Identifique o tipo real	
Diretório de entrega: ex02/	
Arquivos para entrega: Makefile, *.cpp, *.{h, hpp}	
Funções proibidas: std::typeinfo	

Implemente uma classe **base** que possua apenas um destruidor virtual público. Crie três vazios classes A, **B** e C, que herdam publicamente de Base.

Estas quatro classes não precisam ser elaboradas na tradição ortodoxa.
Forma canônica.

Implemente as seguintes funções:

Base * gerar(void); Ele instancia
A, B ou C aleatoriamente e retorna a instância como um ponteiro Base. Sinta-se à vontade para usar o que quiser para a implementação de escolha aleatória.

identificação nula(Base* p);
Imprime o tipo real do objeto apontado por p: "A", "B" ou "C".

identificação nula (Base & p);
Imprime o tipo real do objeto apontado por p: "A", "B" ou "C". É proibido usar um ponteiro dentro desta função.

Incluir o cabeçalho typeinfo é proibido.

Escreva um programa para testar se tudo funciona conforme o esperado.