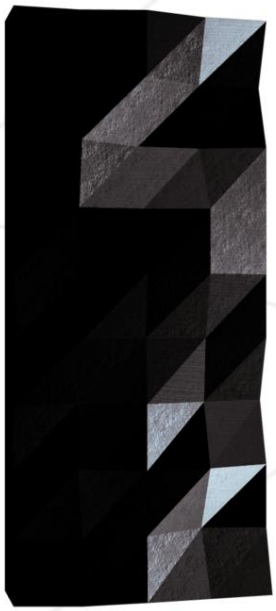

C++ - Módulo 06

Conversões C++

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

Versão: 6.2

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: Wikipedia).

O objetivo destes módulos é apresentar a **Programação Orientada a Objetos.**
Este será o ponto de partida da sua jornada C++. Muitas linguagens são recomendadas para aprender POO. Decidimos escolher C++, pois é derivado do seu velho amigo C.
Como esta é 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 muito diferente em muitos aspectos. Então, se você quer se tornar um desenvolvedor C++ proficiente, cabe a você ir além depois do 42 Common Core!

Capítulo II

Regras gerais

Compilando

- Compile seu código com c++ e os sinalizadores -Wall -Wextra -Werror
- Seu código ainda deve compilar se você adicionar o sinalizador -std=c++98

Convenções de formatação e nomenclatura

- Os diretórios dos exercícios serão nomeados desta forma: ex00, ex01, ... , exn
- Nomeie seus arquivos, classes, funções, funções de membro e atributos conforme necessário em as diretrizes.
- Escreva nomes de classes no formato **UpperCamelCase .** Arquivos contendo código de classe serão sempre será 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" representando uma parede de tijolos, seu nome será BrickWall.hpp.
- A menos que especificado de outra forma, todas as mensagens de saída devem ser encerradas por uma nova linha caractere e exibido na saída padrão.
- *Adeus Norminette!* Nenhum estilo de codificação é imposto nos módulos C++. Você pode seguir seu favorito. Mas tenha em mente que um código que seus avaliadores pares não conseguem entender é um código que eles não conseguem classificar. Faça o seu melhor para escrever um código limpo e legível.

Permitido/Proibido

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

- Você tem permissão para usar quase tudo da biblioteca padrão. Assim, em vez de ficar preso ao que você já sabe, seria inteligente usar o máximo possível as versões C++-ish das funções C com as quais você 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 usá-las, sua nota será 0 e pronto.

- Observe que, a menos que explicitamente indicado de outra forma, o uso do namespace <ns_name> e palavras-chave de amigos são proibidas. Caso contrário, sua nota será -42.
- **Você tem permissão para usar o STL somente no Módulo 08 e 09.** Isso significa: nenhum **Container** (vetor/lista/mapa/e assim por diante) e nenhum **Algoritmo** (qualquer coisa que exija incluir o cabeçalho <algoritmo>) até então. Caso contrário, sua nota será -42.

Alguns requisitos de design

- Vazamento de memória ocorre em C++ também. 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 elaboradas na **Língua Ortodoxa Forma canônica, 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. Assim, eles devem incluir todas as dependências de que precisam. No entanto, você deve evitar o problema de inclusão dupla adicionando **guardas de inclusão.** 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!!!

Você terá que implementar muitas classes. Isso pode parecer tedioso, a menos que você consiga criar um script no seu editor de texto favorito.

Você tem uma certa liberdade para completar os exercícios. No entanto, siga as regras obrigatórias e não seja preguiçoso. Você iria 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 usando um tipo específico de conversão. 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 para um int, um float ou um double. Isso ajudará, mas não fará todo o trabalho.	

Escreva uma classe ScalarConverter que conterá apenas um método estático "convert" que receberá como parâmetro uma representação de string de um literal C++ em sua forma mais comum e emitirá seu valor na seguinte série de tipos escalares:

- carvão
- inteiro
- flutuar
- dobro

Como esta classe não precisa armazenar nada, ela não deve ser instanciável por Usuários.

Com exceção dos parâmetros char, somente a notação decimal será usada.

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

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

Exemplos de literais float: 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 precisa lidar com esses pseudo literais (sabe, por diversão): -inf, +inf e nan.

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

Você tem que primeiro detectar o tipo do literal passado como parâmetro, convertê-lo de string para seu tipo real, então convertê-lo **explicitamente** para os três outros tipos de dados. Por fim, 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 que você precise 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
./converter 42.0f
caracteres: '*'
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	

Implementar uma classe Serializer, que não será inicializável pelo usuário de forma alguma, com os seguintes métodos estáticos:

uintptr_t serializar(Dados* ptr);
Ele pega um ponteiro e o converte para o tipo inteiro sem sinal uintptr_t.

Dados* desserializar(uintptr_t raw);
Ele pega um parâmetro inteiro sem sinal 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 (ou seja, com membros de dados).

Use serialize() no endereço do objeto Data e passe seu valor de retorno para deserialize(). Então, garanta que o valor de retorno de deserialize() seja comparado igual ao ponteiro original.

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

Capítulo VI

Exercício 02: Identificar o tipo real

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

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

Essas quatro classes não precisam ser projetadas no Ortodoxo Forma canônica.

Implemente as seguintes funções:

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

void identify(Base* p); Ele imprime o tipo real do objeto apontado por p: "A", "B" ou "C".

vazio identificar(Base& p); Ela imprime o tipo real do objeto apontado por p: "A", "B" ou "C". Usar um ponteiro dentro desta função é proibido.

É proibido incluir o cabeçalho typeinfo.

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

Capítulo VII

Submissão e avaliação por pares

Entregue sua tarefa no seu repositório Git como de costume. Apenas 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.

16D85ACC441674FBA2DF65190663E136253996A5020347143B460E2CF3A3784D794B
104265933C3BE5B62C4E062601EC8DD1F82FEB73CB17AC57D49054A7C29B5A5C1D8
2027A997A3E24E387