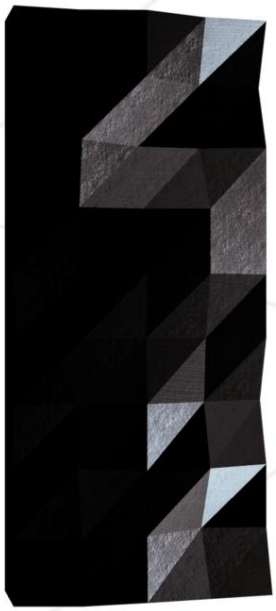

C++ - Módulo 03

Herança

Resumo:
Este documento contém os exercícios do Módulo 03 dos módulos C++.

Versão: 7.1

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 em C++. Muitas linguagens são recomendadas para aprender POO, mas decidimos escolher C++ por ser derivada da sua velha amiga 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. Portanto, se você quer se tornar um desenvolvedor C++ proficiente, cabe a você ir além, além dos 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 os nomes das classes no formato **UpperCamelCase .** Arquivos contendo código de classe sempre serão nomeados 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 da classe "BrickWall", que representa uma parede de tijolos, seu nome será BrickWall.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 C com as quais você está acostumado, sempre que possível.
- No entanto, você não pode usar nenhuma outra biblioteca externa. Isso significa que as bibliotecas C++11 (e 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 o contrário, o uso do namespace <ns_name> e Palavras-chave de amigos são proibidas. Caso contrário, sua nota será -42.
- **Você só pode usar o STL nos Módulos 08 e 09.** Isso significa que não há **Contêineres** (vetor/lista/mapa e assim por diante) nem **Algoritmos** (qualquer coisa que exija a inclusão do cabeçalho <algoritmo>) até lá. Caso contrário, sua nota será -42.

Alguns requisitos de design

- 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 elaboradas na Igreja **Ortodoxa Forma canônica, exceto quando explicitamente declarado 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 conseguir 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 **proteções de inclusão.** Caso contrário, sua nota será 0.

Leia-me

- Você pode adicionar alguns arquivos adicionais, se necessário (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 isso.
- 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 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

Exercício 00: Aaaaand... ABERTO!

	Exercício: 00
Eeeee... ABERTO!	
Diretório de entrega: ex00/	
Arquivos para entrega: Makefile, main.cpp, ClapTrap.{h, hpp}, ClapTrap.cpp Funções proibidas: Nenhuma	

Primeiro, você precisa implementar uma classe! Que original!

Ele será chamado **ClapTrap** e terá os seguintes atributos privados inicializados com os valores especificados entre colchetes:

- Nome, que é passado como parâmetro ao construtor
- Pontos de vida (10), representando a saúde do ClapTrap
- Pontos de energia (10)
- Dano de ataque (0)

Adicione as seguintes funções de membro públicas para que o ClapTrap se comporte de forma mais realista:

- ataque vazio(const std::string& alvo);
- void takeDamage(unsigned int quantidade);
- void beReparado(unsigned int quantidade);

Quando ClapTrap ataca, ele faz com que seu alvo perca <dano de ataque> pontos de vida.
Quando ClapTrap se repara, ele recupera <quantidade> pontos de vida. Atacar e reparar cada um custa 1 ponto de energia. É claro que ClapTrap não pode fazer nada se não tiver pontos de vida ou pontos de energia restantes. No entanto, como estes exercícios servem como uma introdução, as instâncias de ClapTrap não devem interagir diretamente umas com as outras, e os parâmetros não farão referência a outra instância de ClapTrap.

Em todas essas funções membro, você precisa imprimir uma mensagem para descrever o que acontece. Por exemplo, a função attack() pode exibir algo como (claro, sem os colchetes angulares):

ClapTrap <nome> ataca <alvo>, causando <dano> pontos de dano!

Os construtores e destruidores também devem exibir uma mensagem, para que seus avaliadores pares podem ver facilmente que foram chamados.

Implemente e entregue seus próprios testes para garantir que seu código funcione conforme o esperado.

Capítulo IV

Exercício 01: Serena, meu amor!

	Exercício: 01
Serena, meu amor!	
Diretório de entrega: ex01/	
Arquivos a entregar: Arquivos do exercício anterior + ScavTrap.{h, hpp}, ScavTrap.cpp Funções proibidas: Nenhuma	

Como nunca é possível ter ClapTraps suficientes, agora você criará um robô derivado.
Ele se chamará **ScavTrap** e herdará os construtores e o destrutor de Clap-Trap. No entanto, seus construtores, destrutor e attack() imprimirão mensagens diferentes.
Afinal, os ClapTraps têm consciência de sua individualidade.

Observe que o encadeamento adequado de construção/destruição deve ser mostrado em seus testes. Quando uma ScavTrap é criada, o programa começa construindo uma ClapTrap. A destruição ocorre na ordem inversa. Por quê?

O ScavTrap usará os atributos do ClapTrap (atualize o ClapTrap de acordo) e deve inicializá-los para:

- Nome, que é passado como parâmetro ao construtor
- Pontos de vida (100), representando a saúde do ClapTrap
- Pontos de energia (50)
- Dano de ataque (20)

ScavTrap também terá sua própria habilidade especial:

```
void guardGate();
```

Esta função de membro exibirá uma mensagem indicando que o ScavTrap agora está no modo Gate Keeper.

Não se esqueça de adicionar mais testes ao seu programa.

Capítulo V

Exercício 02: Trabalho repetitivo

	Exercício: 02
Trabalho repetitivo	
Diretório de entrega: ex02/	
Arquivos para entrega: Arquivos de exercícios anteriores + FragTrap.{h, hpp}, FragTrap.cpp Funções proibidas: Nenhuma	

Fazer ClapTraps provavelmente está começando a te dar nos nervos.

Agora, implemente uma classe **FragTrap** que herda de ClapTrap. Ela é muito semelhante a ScavTrap. No entanto, suas mensagens de construção e destruição devem ser diferentes. O encadeamento adequado de construção/destruição deve ser demonstrado em seus testes. Quando uma FragTrap é criada, o programa começa construindo uma ClapTrap. A destruição ocorre na ordem inversa.

O mesmo vale para os atributos, mas com valores diferentes desta vez:

• Nome, que é passado como parâmetro ao construtor

• Pontos de vida (100), representando a saúde do ClapTrap

• Pontos de energia (100)

• Dano de ataque (30)

FragTrap também tem uma habilidade especial:

vazio highFivesGuys(vazio);

Esta função membro exibe uma solicitação de cumprimento positivo na saída padrão.

Novamente, adicione mais testes ao seu programa.

Capítulo VI

Exercício 03: Agora ficou estranho!

	Exercício: 03
Agora ficou estranho!	
Diretório de entrega: ex03/	
Arquivos para entregar: Arquivos de exercícios anteriores + DiamondTrap.{h, hpp}, DiamondTrap.cpp	
Funções proibidas:	
Nenhuma	

Neste exercício, você criará um monstro: um ClapTrap que é metade FragTrap, metade Scav-Trap. Ele se chamará **DiamondTrap** e herdará de FragTrap e ScavTrap. Isso é muito arriscado!

A classe DiamondTrap terá um atributo privado chamado name. Este atributo deve ter exatamente o mesmo nome de variável da classe base ClapTrap (sem fazer referência ao nome do robô).

Para ser mais claro, aqui estão dois
exemplos: se a variável do ClapTrap for nome, dê à variável do DiamondTrap o nome nome.
Se a variável do ClapTrap for _name, atribua à variável do DiamondTrap o nome _name.

Seus atributos e funções de membro serão herdados de suas classes pai:

- Nome, que é passado como parâmetro para um construtor
- ClapTrap::name (parâmetro do construtor + sufixo "_clap_name")
- Pontos de vida (FragTrap)
- Pontos de energia (ScavTrap)
- Dano de ataque (FragTrap)
- ataque() (ScavTrap)

Além das funções especiais de ambas as classes pai, DiamondTrap terá sua própria habilidade especial:

vazio quemSouEu();

Esta função membro exibirá seu nome e seu nome ClapTrap.

É claro que a instância ClapTrap de DiamondTrap será criada uma vez, e somente uma vez. Sim, há um truque.

Novamente, adicione mais testes ao seu programa.

Você conhece os sinalizadores do compilador -Wshadow e -Wno-shadow?

Você pode passar neste módulo sem completar o exercício 03.

Capítulo VII

Submissão e Avaliação por Pares

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

?????????? XXXXXXXXXX = $3$$cf36316f07f871b4f14926007c37d388