



C++ - Módulo 03

Herança

Resumo:

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

Versão: 7

Conteúdo

I	Introdução	2
II	Regras gerais	3
III	Exercício 00: Eeeee... ABERTO!	5
IV	Exercício 01: Serena, meu amor!	7
V	Exercício 02: Trabalho repetitivo	8
VI	Exercício 03: Agora ficou estranho!	9
VII	Submissão e avaliação por pares	11

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`, ... , `ex`
- 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.hpp`. 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 perder muitas informações úteis! Não hesite em ler sobre conceitos teóricos.

Capítulo III

Exercício 00: Eeeee... ABERTO!

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

Primeiro, você tem que 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 para um construtor
- Pontos de vida (10), representam a saúde do ClapTrap
- Pontos de energia (10)
- Dano de ataque (0)

Adicione as seguintes funções de membro públicas para que o ClapTrap pareça mais realista:

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

Quando ClapTrap ataca, ele faz com que seu alvo perca <dano de ataque> pontos de vida. Quando ClapTrap se repara, ele recebe <quantidade> pontos de vida de volta. Atacar e reparar custam 1 ponto de energia cada. Claro, ClapTrap não pode fazer nada se não tiver pontos de vida ou pontos de energia restantes.

Em todas essas funções membro, você tem que 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 de 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 para 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 será chamado de **ScavTrap** e herdará os construtores e destrutor de ClapTrap. No entanto, seus construtores, destrutor e attack() imprimirão mensagens diferentes. Afinal, os ClapTraps estão cientes de sua individualidade.

Observe que o encadeamento adequado de construção/destruição deve ser demonstrado em seus testes. Quando um ScavTrap é criado, o programa começa construindo um ClapTrap. A destruição é em ordem reversa. Por quê?

O **ScavTrap** usará os atributos do ClapTrap (atualizará o ClapTrap em consequência) e deve inicializá-los para:

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

O ScavTrap também terá sua própria capacidade especial:


```
void guardGate();
```

Esta função membro exibirá uma mensagem informando 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 entregar: 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 similar a ScavTrap. Entretanto, suas mensagens de construção e destruição devem ser diferentes. O encadeamento de construção/destruição apropriado deve ser mostrado em seus testes. Quando um FragTrap é criado, o programa começa construindo um ClapTrap. A destruição é na ordem inversa.

As mesmas coisas para os atributos, mas com valores diferentes desta vez:

- Nome, que é passado como parâmetro para um construtor
- Pontos de vida (100), representam a saúde do ClapTrap
- Pontos de energia (100)
- Dano de ataque (30)

O FragTrap também tem uma capacidade 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 ScavTrap. Ele será chamado de **DiamondTrap** e herdará tanto do FragTrap quanto do ScavTrap. Isso é muito arriscado!

A classe DiamondTrap terá um atributo privado name. Dê a esse atributo exatamente o mesmo nome de variável (não estou falando do nome do robô aqui) que o da classe base ClapTrap.

Para ser mais claro, aqui estão dois exemplos.

Se a variável do ClapTrap for name, atribua o nome name à variável do DiamondTrap.

Se a variável do ClapTrap for _name, dê o nome _name à variável do DiamondTrap.

Seus atributos e funções de membro serão escolhidos de uma 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-mãe, o DiamondTrap terá sua própria capacidade especial:

```
vazio quemSouEu();
```

Esta função membro exibirá seu nome e seu nome ClapTrap.

Claro, o subobjeto ClapTrap do DiamondTrap será criado 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 fazer o exercício 03.

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.



??????????? XXXXXXXXXXXX = \$3\$cf36316f07f871b4f14926007c37d388