



C++ - Módulo 03

Herança

Resumo:

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

Versão: 6

Conteúdo

I	Introdução	2
II	Regras gerais	3
III	Exercício 00: Aaaae... ABRA!	5
IV	Exercício 01: Serena, meu amor!	7
V	Exercício 02: Trabalho repetitivo	8
VI	Exercício 03: Agora é estranho!	9

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 with Classes" (fonte: [Wikipedia](#)).

O objetivo desses módulos é apresentá-lo à **Programação Orientada a Objetos**. Este será o ponto de partida de sua jornada C++. Muitos idiomas são recomendados para aprender OOP. Decidimos escolher C++, pois é derivado de 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.

Sabemos 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 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 de exercícios serão nomeados desta forma: `ex00`, `ex01`, ... , `exn`
- Nomeie seus arquivos, classes, funções, funções de membro e atributos conforme exigido em As diretrizes.
- Escreva os nomes das 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 encerradas com uma nova linha caractere e exibido na saída padrão.
- *Adeus Norminette!* Nenhum estilo de codificação é aplicado nos módulos C++. Você pode seguir o seu favorito. Mas lembre-se de 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. Hora de C++! Portanto:

- Você tem permissão para usar quase tudo da biblioteca padrão. Portanto, em vez de se ater ao que você já sabe, seria inteligente usar o máximo possível as versões em C++ das funções C às 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-los, sua nota será 0 e pronto.

- Observe que, a menos que explicitamente declarado de outra forma, o namespace using <ns_name> e palavras-chave de amigos são proibidas. Caso contrário, sua nota será -42.
- **Você tem permissão para usar o STL apenas no Módulo 08 e 09.** Isso significa: sem **contêineres** (vetor/lista/mapa/e assim por diante) e sem **algoritmos** (qualquer coisa que requeira incluir o cabeçalho <algorithm>) até então. Caso contrário, sua nota será -42.

Alguns requisitos de projeto

- 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 elaboradas no **Ortodoxo 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 ser capaz de usar cada um de 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 (ou seja, 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! Realmente, faça isso.
- Por Odin, por Thor! Use seu cérebro!!!




Você terá que implementar muitas classes. Isso pode parecer tedioso, a menos que você seja capaz de criar o script de 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 poderia perder muita informação útil! Não hesite em ler sobre conceitos teóricos.

Capítulo III

Exercício 00: Aaaae... ABRA!

	Exercício: 00
Aaaae... ABRA!	
Diretório de entrega: ex00/	
Arquivos a serem entregues: Makefile, main.cpp, ClapTrap.{h, hpp}, ClapTrap.cpp	
Funções proibidas: Nenhuma	

Primeiro, você tem que implementar uma classe! Que original!

Ele será chamado de **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 público para que o ClapTrap pareça mais realista:

- void attack(const std::string& target);
- void takeDamage(quantia int não assinada);
- void beRepaired(quantia int não assinada);

Quando ClapTrap ataca, ele faz com que seu alvo perca <dano de ataque> pontos de vida. Quando o ClapTrap se repara, ele recupera <amount> pontos de vida. Atacar e reparar custa 1 ponto de energia cada. Claro, ClapTrap não pode fazer nada se não tiver pontos de vida ou pontos de energia sobrando.

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


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

Os construtores e destruidores também devem exibir uma mensagem, para que seus avaliadores 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 você nunca terá ClapTraps suficientes, agora criará um robô derivado. Ele será nomeado **ScavTrap** e herdará os construtores e destruidores de Clap Trap. No entanto, seus construtores, destruidor e attack() imprimirão mensagens diferentes. Afinal, ClapTraps tem consciência de sua individualidade.

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

ScavTrap usará os atributos de ClapTrap (atualize 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 de membro exibirá uma mensagem informando que o ScavTrap agora está no modo Gatekeeper.

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 a 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. É muito semelhante ao ScavTrap. No entanto, suas mensagens de construção e destruição devem ser diferentes. O encadeamento adequado de construção/destruição deve ser mostrado em seus testes. Quando um FragTrap é criado, o programa começa construindo um ClapTrap. A destruição está na ordem inversa.

Mesma coisa 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:


```
void highFivesGuys(void);
```

Esta função de membro exibe uma solicitação positiva de high fives na saída padrão.

Novamente, adicione mais testes ao seu programa.

Capítulo VI

Exercício 03: Agora é estranho!

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

Neste exercício, você criará um monstro: um ClapTrap que é meio FragTrap, meio ScavTrap. Ele será nomeado **DiamondTrap** e herdará tanto do FragTrap quanto do ScavTrap. Isso é tão arriscado!

A classe DiamondTrap terá um atributo privado de nome. Dê a este atributo exatamente o mesmo nome da variável (sem falar sobre o nome do robô aqui) que o da classe base ClapTrap.

Para ficar mais claro, aqui estão dois exemplos.

Se a variável do ClapTrap for name, dê o nome name ao do DiamondTrap.

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

Seus atributos e funções de membro serão escolhidos de qualquer 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)
- attack() (Scavtrap)

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

```
void whoAmI();
```

Esta função de membro exibirá seu nome e seu nome ClapTrap.

Claro, o subobjeto ClapTrap do DiamondTrap será criado uma vez, e apenas uma vez. Sim, há um truque.

Novamente, adicione mais testes ao seu programa.



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



Você pode passar neste módulo sem fazer o exercício 03.