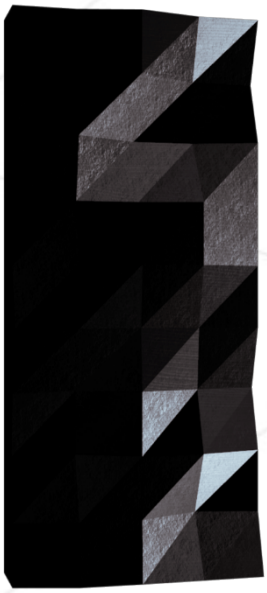

C++ - Módulo 03

Herança

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

Versão: 6

Conteúdo

Capítulo I

Introdução

C++ é uma linguagem de programação de uso geral criada por Bjarne Stroustrup como uma ex-tensão da linguagem de programação C, ou "C com Classes" (fonte: Wikipedia).

O objectivo destes módulos é apresentar-lhe a **Programação Orientada para Objectos**. Este será o ponto de partida da sua viagem em C++. Muitas linguagens são recomendadas para aprender OOP. Decidimos escolher o C++, uma vez que é derivado do seu velho amigo C. Como se trata de uma linguagem complexa, e para manter as coisas simples, o seu código obedecerá à norma C++98.

Sabemos que o C++ moderno é muito diferente em muitos aspectos. Por isso, se quiser tornar-se um programador C++ competente, cabe-lhe a si ir mais longe depois do 42 Common Core!

Capítulo II Regras gerais

Compilação

- Compile o seu código com c++ e os sinalizadores -Wall -Wextra -Werror
- O seu código deve continuar a compilar se adicionar o sinalizador -std=c++98

Convenções de formatação e nomeação

- As directorias de exercícios terão os seguintes nomes: ex00, ex01, ... , exn
- Dê aos seus ficheiros, classes, funções, funções membro e atributos os nomes exigidos nas directrizes.
- Escreva os nomes das classes no formato **UpperCamelCase**. Os ficheiros que contêm código de classe serão sempre nomeados de acordo com o nome da classe. Por exemplo: ClassName.hpp/ClassName.h, ClassName.cpp, ou ClassName.tpp. Então, se tiver um ficheiro de cabeçalho que contenha a definição de uma classe "BrickWall" que representa uma parede de tijolos, o seu nome será BrickWall.hpp.
- Salvo especificação em contrário, todas as mensagens de saída devem ser terminadas por um carácter de nova linha e apresentadas na saída padrão.
- *Adeus Norminette!* Não é imposto nenhum estilo de codificação nos módulos C++. Pode seguir o seu estilo preferido. Mas lembre-se de que um código que os seus colegas avaliadores não conseguem entender é um código que eles não podem classificar. Faça o seu melhor para escrever um código limpo e legível.

Permitido/Proibido

Já não está a codificar em C. Está na altura de usar C++! Portanto:

- É-lhe permitido utilizar quase tudo da biblioteca padrão. Assim, em vez de se cingir ao que já sabe, seria inteligente utilizar o mais possível as versões C++ das funções C a que está habituado.
- No entanto, não pode utilizar qualquer outra biblioteca externa. Isto significa que as bibliotecas C++11 (e formas derivadas) e Boost são proibidas. As seguintes funções também são proibidas: *printf(), *alloc() e free(). Se as utilizares, a tua nota será 0 e ponto final.

- Observe que, a menos que explicitamente declarado de outra forma, o namespace de uso <ns_name> e
  palavras-chave amigas são proibidas. Caso contrário, a sua nota será de -42.
- **Só é permitido usar a STL no Módulo 08.** Isso significa: nada de **Containers** (vector/lista/mapa/e assim por diante) e nada de **Algoritmos** (qualquer coisa que exija a inclusão do cabeçalho <algorithm>) até lá. Caso contrário, a tua nota será de -42.

Alguns requisitos de concepção

- A fuga de memória também ocorre em C++. Quando se aloca memória (usando o comando new
  ), é necessário evitar **fugas de memória**.
- Do módulo 02 ao módulo 08, as suas aulas devem ser concebidas segundo a **forma canónica ortodoxa, salvo indicação expressa em contrário**.
- Qualquer implementação de função colocada num ficheiro de cabeçalho (excepto para modelos de funções) significa 0 para o exercício.
- Deve ser possível utilizar cada um dos seus cabeçalhos independentemente dos outros. Assim, eles devem incluir todas as dependências que precisam. No entanto, tens de evitar o problema da dupla inclusão, adicionando **guardas de inclusão**. Caso contrário, a tua nota será 0.

Ler-me

- Pode acrescentar alguns ficheiros adicionais se for necessário (ou seja, para dividir o seu código). Como estes trabalhos não são verificados por um programa, pode fazê-lo à vontade, desde que entregue os ficheiros obrigatórios.
- Por vezes, as directrizes 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! A sério, faça-o.
- Por Odin, por Thor! Usem o vosso cérebro!!!

Terá de implementar muitas classes. Isto pode parecer entediante, a menos que seja capaz de programar o seu editor de texto favorito.

É-te dada uma certa liberdade para realizares os exercícios. No entanto, segue as regras obrigatórias e não sejas preguiçoso. Perderia muitas informações úteis! Não hesites em ler sobre conceitos teóricos.

Capítulo III

Exercício 00: Aaaaand... ABRIR!

	Exercício : 00
Aaaaand... ABERTO!	
Directório de entrada : *ex00/*	
Ficheiros a entregar : `Makefile, main.cpp, ClapTrap.{h, hpp}, ClapTrap.cpp`	
Funções proibidas : `Nenhuma`	

Primeiro, é preciso implementar uma classe! Que original!

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

- 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(unsigned int amount);
- void beRepaired(unsigned int amount);

Quando ClapTrack ataca, faz com que seu alvo perca <dano de ataque> pontos de vida. Quando a ClapTrap se repara, recupera <quantidade> de pontos de vida. Atacar e reparar custam 1 ponto de energia cada. É claro que a ClapTrap não pode fazer nada se não tiver pontos de vida ou de energia.

Em todas estas funções membro, é necessário imprimir uma mensagem para descrever o que acontece. Por exemplo, a função attack() pode apresentar algo do género (claro, sem os parênteses angulares):

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

Os construtores e o destruidor devem também apresentar uma mensagem, para que os avaliadores possam ver facilmente que foram chamados.

Implementar e apresentar os seus próprios testes para garantir que o seu código funciona como esperado.

Capítulo IV

Exercício 01: Serena, meu amor!

	Exercício : 01
Serena, meu amor!	
Directório de entrada : *ex01/*	
Ficheiros a entregar : `Ficheiros do exercício anterior + ScavTrap.{h, hpp}, ScavTrap.cpp`	
Funções proibidas : `Nenhuma`	

Como nunca há ClapTraps suficientes, vais agora criar um robô derivado. Ele vai chamar-se **ScavTrap** e vai herdar os construtores e o destruidor do Clap-Trap. No entanto, os seus construtores, destruidor e ataque() vão imprimir mensagens diferentes. Afinal de contas, as ClapTraps têm consciência da sua individualidade.

Note-se que o encadeamento correcto de construção/destruição deve ser mostrado nos testes. Quando uma ScavTrap é criada, o programa começa por construir uma ClapTrap. A destruição é feita na ordem inversa. Porquê?

ScavTrap utilizará os atributos de ClapTrap (actualizará 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)

A ScavTrap também terá a sua própria capacidade especial:

```
void guardGate();
```

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

Não se esqueça de adicionar mais testes ao seu programa.

Capítulo V

Exercício 02: Trabalho repetitivo

	Exercício : 02
Trabalho repetitivo	
Directório de entrada : *ex02/*	
Ficheiros a entregar : `Ficheiros de exercícios anteriores + FragTrap.{h, hpp}, FragTrap.cpp`	
Funções proibidas : `Nenhuma`	

Fazer ClapTraps está provavelmente a começar a irritar-te.

Agora, implemente uma classe **FragTrap** que herda de ClapTrap. É muito semelhante a ScavTrap. No entanto, as suas mensagens de construção e destruição devem ser diferentes. O encadeamento correcto de construção/destruição deve ser mostrado nos seus testes. Quando uma FragTrap é criada, o programa começa por construir uma ClapTrap. A destruição faz-se pela ordem inversa.

A 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)

A FragTrap também tem uma capacidade especial:

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

Esta função membro apresenta um pedido positivo de high fives na saída padrão.

Mais uma vez, adicione mais testes ao seu programa.

Capítulo VI

Exercício 03: Agora é que é estranho!

	Exercício : 03
Agora é que é estranho!	
Directório de entrada : *ex03/*	
Ficheiros a entregar : `Ficheiros de exercícios anteriores + DiamondTrap.{h, hpp}, DiamondTrap.cpp`	
Funções proibidas : `Nenhuma`	

Neste exercício, vais criar um monstro: um ClapTrap que é metade FragTrap, metade ScavTrap. Ele será chamado de **DiamondTrap**, e herdará tanto do FragTrap quanto do ScavTrap. Isto é tão arriscado!

A classe DiamondTrap terá um atributo privado name. Atribui a este atributo exactamente o mesmo nome de variável (não estamos a falar do nome do robô) que o da classe base ClapTrap.

Para ser mais claro, eis dois exemplos.
Se a variável da ClapTrap for name, atribuir o nome name à variável da DiamondTrap.
Se a variável da ClapTrap for _name, atribuir o nome _name à variável da DiamondTrap.

Os seus atributos e funções de membro serão seleccionados de qualquer uma das suas classes-mãe:

- 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)

Para além das funções especiais de ambas as classes-mãe, a DiamondTrap terá a sua própria capacidade especial:

void whoAmI();

Esta função de membro apresentará o seu nome e o seu nome ClapTrap.

É claro que o subobjecto ClapTrap do DiamondTrap será criado uma vez, e apenas uma vez. Sim, há um truque.

Mais uma vez, adicione mais testes ao seu programa.

Conhece os sinalizadores de compilação -Wshadow e -Wno-shadow?

Pode passar este módulo sem efectuar o exercício 03.