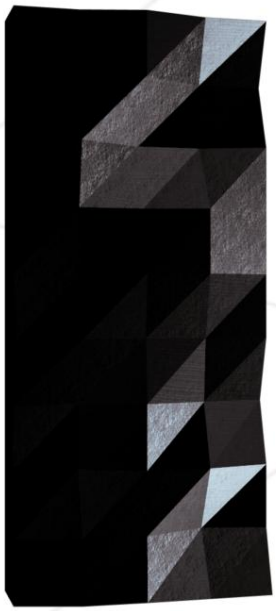

# C++ - Módulo 03

## Herança

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

*Versão: 7*

# Conteúdo

# 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 com Classes" (fonte: Wikipedia).*

O objetivo desses módulos é apresentar **a Programação Orientada a Objetos.**
Este será o ponto de partida de sua jornada em C++. Muitas linguagens são recomendadas para aprender OOP. Decidimos escolher C++ porque ele é derivado do 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.

Estamos cientes de 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 após o 42 Common Core!

# Capítulo II

# Regras gerais

### Compilando

- Compile seu código com c++ e os sinalizadores -Wall -Wextra -Werror
- Seu código ainda deverá ser compilado 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 exigido em as diretrizes.
- Escreva nomes de 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 finalizadas com uma nova linha caractere e exibido na saída padrão.
- *Adeus Norminette!* Nenhum estilo de codificação é imposto nos módulos C++. Você pode seguir o seu favorito. Mas tenha em mente 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. É hora de C++! Portanto:

- Você tem permissão para usar quase tudo da biblioteca padrão. Assim, em vez de se ater ao que você já sabe, seria inteligente usar o máximo possível as versões C++ das funções C com as quais você está acostumado.
- Entretanto, você não pode usar nenhuma outra biblioteca externa. Isso significa que C++ 11 (e formas derivadas) e bibliotecas 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, salvo indicação explícita em contrário, o namespace using <ns_name> e palavras-chave de amigos são proibidas. Caso contrário, sua nota será -42.
- **É permitido utilizar o STL somente nos Módulos 08 e 09.** Isso significa: nenhum **contêiner** (vetor/lista/mapa/e assim por diante) e nenhum **algoritmo** (qualquer coisa que exija a inclusão do cabeçalho <algorithm>) até então. Caso contrário, sua nota será -42.

**Alguns requisitos de design**

- 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 planejadas no **estilo Ortodoxo Forma Canônica, exceto quando explicitamente indicado de outra forma.**
- Qualquer implementação de função colocada em um arquivo de cabeçalho (exceto 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 necessitam. No entanto, você deve evitar o problema da 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, fique à vontade para fazê-lo, desde que entregue os arquivos obrigatórios.
- Às vezes, as orientações 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 consiga criar um script em seu editor de texto favorito.

Você tem uma certa liberdade para completar os exercícios. Porém, siga as regras obrigatórias e não seja preguiçoso. Você faria perca muitas informações úteis! Não hesite em ler sobre conceitos teóricos.

# Capítulo III

# Exercício 00: Aaaaae... ABRA!

|  | Exercício: 00 |
|---|---|
| Eaaaa... ABERTO! | |
| Diretório de entrega: ex00/ | |
| Arquivos para entrega: 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 público para que o ClapTrap pareça mais realista:

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

Quando ClapTrack 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 custam 1 ponto de energia cada. Claro, ClapTrap não pode fazer nada se não tiver mais pontos de vida ou energia.

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

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

Os construtores e destruidores também devem exibir uma mensagem, para que seus avaliadores pode facilmente ver que eles 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 entrega: Arquivos do exercício anterior + ScavTrap.{h, hpp}, ScavTrap.cpp Funções proibidas: Nenhuma | |
| | |

Como você nunca terá ClapTraps suficientes, agora você criará um robô derivado.
Ele se chamará **ScavTrap** e herdará os construtores e destruidores do Clap-Trap. No entanto, seus construtores, destruidor 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 um ScavTrap é criado, o programa começa construindo um ClapTrap. A destruição ocorre na ordem inversa. Por que?

**ScavTrap** usará os atributos do ClapTrap (atualizará o ClapTrap em conseqüência) 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 capacidade especial própria:

```
void guardGate();
```

Esta função 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 | |
| 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 irritar.

Agora, implemente uma classe **FragTrap** que herda de ClapTrap. É muito semelhante ao ScavTrap. Contudo, as 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 ocorre na ordem inversa.

A mesma coisa para os atributos, mas desta vez com valores diferentes:

• 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 membro exibe uma solicitação de cumprimentos positiva na saída padrão.

Novamente, adicione mais testes ao seu programa.

# Capítulo VI

# Exercício 03: Agora está estranho!

|  | Exercício: 03 |
|---|---|
| Agora é 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 se chamará **DiamondTrap** e herdará do FragTrap E do ScavTrap. Isso é tão arriscado!

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

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

Seus atributos e funções-membro serão escolhidos em 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 acerto (FragTrap)
- Pontos de energia (ScavTrap)
- Dano de ataque (FragTrap)
- ataque() (Scavtrap)

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

void quemAmI();

Esta função membro exibirá seu nome e seu nome ClapTrap.

Obviamente, 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 do compilador -Wshadow e -Wno-shadow?

Você pode passar neste módulo sem fazer o exercício 03.

# Capítulo VII

# Envio e avaliação por pares

Entregue sua tarefa em seu repositório Git normalmente. Somente o trabalho dentro do seu repositório será avaliado durante a defesa. Não hesite em verificar os nomes de suas pastas e arquivos para garantir que estão corretos.

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