



C++ - Módulo 01

Alocação de memória, indicações aos
membros, referências, declaração de
mudança

Resumo:

Este documento contém os exercícios do Módulo 01 dos módulos C++.

Versão: 9

Conteúdo

I	Introdução	2
II	Regras gerais	3
III	Exercício 00: Braiiiiiinnnnnnzzzz	5
IV	Exercício 01: Moar brainz!	6
V	Exercício 02: O QUE É ESTE CRÂNIO	7
VI	Exercício 03: Violência desnecessária	8
VII	Exercício 04: Sed é para os perdedores	10
VIII	Exercício 05: Harl 2.0	11
IX	Exercício 06: Filtro Harl	13

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-lhe a **Programação Orientada a Objetos**. Este será o ponto de partida de sua jornada C++. Muitas linguagens são recomendadas para aprender o OOP. Decidimos escolher C++, pois é derivado de seu velho amigo C. Como esta é uma linguagem complexa, e para manter as coisas simples, seu código estará de acordo com o padrão C++98.

Estamos conscientes de que o C++ moderno é muito diferente em muitos aspectos. Portanto, se você quer se tornar um desenvolvedor C++ proficiente, cabe a você ir mais além após o 42 Common Core!

Capítulo II Regras

gerais

Compilação

- Compile seu código com c++ e as bandeiras -Wall -Wextra -Werror
- Seu código ainda deve ser compilado se você adicionar a bandeira -std=c++98

Formatação e convenções de nomenclatura

- Os diretórios de exercícios serão nomeados desta forma: ex00, ex01, ... , exn
- Nomeie seus arquivos, classes, funções, funções dos membros e atributos, conforme exigido nas diretrizes.
- Escreva os nomes das classes no formato **UpperCamelCase**. Arquivos contendo código de classe serão sempre nomeados de acordo com o nome da classe. Por exemplo, os arquivos com o código de classe serão sempre nomeados de acordo com o nome da classe: ClassName.hpp/ClassName.h, ClassName.cpp, ou ClassName.tpp. Então, se você tiver um arquivo de cabeçalho contendo a definição de uma classe "BrickWall" representando uma parede de tijolo, seu nome será BrickWall.hpp.
- A menos que especificado de outra forma, todas as mensagens de saída devem ser terminadas por um novo caractere de linha e exibidas na saída padrão.
- *Adeus Norminette!* Nenhum estilo de codificação é aplicado nos módulos C++. Você pode seguir seu favorito. Mas tenha em mente que um código que seus colegas avaliadores não conseguem entender é um código que eles não conseguem avaliar. Faça seu melhor para escrever um código limpo e legível.

Permitido/Proibido

Você não está mais codificando em C. Tempo para C++! Portanto:

- Você tem permissão para usar quase tudo da biblioteca padrão. Assim, ao invés de se ater ao que você já sabe, seria inteligente usar o máximo possível as versões C++-ish das funções C às quais você está acostumado.
- Entretanto, não é possível utilizar 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.

- Note que, a menos que explicitamente declarado de outra forma, o uso do namespace <ns_name> e são proibidas as palavras-chave de amigos. Caso contrário, sua nota será -42.
- **Você está autorizado a usar o STL somente no Módulo 08.** Isso significa: nenhum **Container** (vetor/lista/mapa/e assim por diante) e nenhum **Algoritmo** (qualquer coisa que exija incluir o <algoritmo> cabeçalho) até lá. 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 08, suas aulas devem ser projetadas no **Formulário Canônico Ortodoxo, exceto quando explicitamente declarado em contrário**.
- Qualquer implementação de função colocada em um arquivo de cabeçalho (exceto para os 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. Entretanto, você deve evitar o problema da dupla inclusão, acrescentando **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 estas atribuições não são verificadas por um programa, sinta-se à vontade para fazê-lo desde que você entregue os arquivos obrigatórios.
- s vezes, as diretrizes de um exercício parecem curtas, mas os exemplos podem mostrar exigências que não estão explicitamente escritas nas instruções.
- Leia cada módulo completamente antes de começar! Realmente, faça-o.
- Por Odin, por Thor! Use seu cérebro!!!




Você terá que implementar muitas aulas. Isto pode parecer entediante, a menos que você seja capaz de escrever seu editor de texto favorito.



É-lhe dada uma certa liberdade para completar os exercícios. Entretanto, siga as regras obrigatórias e não seja preguiçoso. Você perderia muitas informações úteis! Não hesite em ler sobre conceitos teóricos.

Capítulo III

Exercício 00: BraiiiiiiinnnnnnzzzzZ

	Exercício : 00
BraiiiiiiinnnnnnzzzzZ	
Diretório de entrada : <i>ex00/</i>	
Arquivos para entregar : Makefile, main.cpp, Zombie.{h, hpp}, Zombie.cpp, newZombie.cpp, randomChump.cpp	
Funções proibidas : Nenhuma	

Primeiro, implemente uma classe **Zumbi**. Ela tem um nome de atributo privado de cadeia.

Adicionar um anúncio de função de membro nulo (nulo); à classe

Zombie
Zombies

se anunciam como a seguir:

<nome>: BraiiiiiiinnnnnnzzzzZ...

Não imprima os suportes de ângulo (< e >). Para um zumbi chamado Foo, a mensagem seria:

Foo: BraiiiiiiinnnnnnzzzzZ...

Em seguida, implemente as duas funções a seguir:

- `Zombie* newZombie(std::string name);`
Ele cria um zumbi, nomeia-o e o devolve para que você possa utilizá-lo fora do escopo da função.
- `void randomChump(std::string name);`
Ele cria um zumbi, nomeia-o, e o zumbi se anuncia.


Agora, qual é o verdadeiro objetivo do exercício? É preciso determinar em que caso é melhor alocar os zumbis na pilha ou na pilha.

Os zumbis devem ser destruídos quando não se precisa mais deles. O destruidor deve

imprimir uma mensagem com o nome do zumbi para fins de depuração.

Capítulo IV

Exercício 01: Moar brainz!

	Exercício : 01
Moar brainz!	
Diretório de entrada : <i>ex01/</i>	
Arquivos para entregar : Makefile, main.cpp, Zombie.{h, hpp}, Zombie.cpp, ZombieHorde.cpp	
Funções proibidas : Nenhuma	

É hora de criar uma **horda de zumbis**!

Implementar a seguinte função no arquivo apropriado:

```
Zumbi*    zombieHorde( int N, std::string name );
```

Ele deve alocar N objetos Zumbi em uma única alocação. Em seguida, deve inicializar os zumbis, dando a cada um deles o nome passado como parâmetro. A função retorna um ponteiro para o primeiro zumbi.


Implemente seus próprios testes para garantir que sua função `zumbiHorde()` funcione como esperado.

Tente chamar o `anúncio()` para cada um dos zumbis.

Não se esqueça de apagar todos os zumbis e verificar se há **vazamentos de memória**.

Capítulo V

Exercício 02: O QUE É ESTE CRÉDITO

	Exercício : 02
OI ESTE É O CÉREBRO	
Diretório de entrada :	
ex02/	
Arquivos para entregar : Makefile, main.cpp	
Funções proibidas : Nenhuma	

Escreva um programa que contenha:

- Uma variável de corda inicializada para "HI THI THIS BRAIN".
- stringPTR: Um ponteiro para a corda.
- stringREF: Uma referência à corda.

Seu programa tem que ser impresso:

- O endereço de memória da variável string.
- O endereço de memória mantido por stringPTR.
- O endereço de memória mantido por


stringREF. E depois:

- O valor da variável string.
- O valor apontado por stringPTR.
- O valor apontado por stringREF.

Isso é tudo, sem truques. O objetivo deste exercício é desmistificar referências que podem parecer completamente novas. Embora existam algumas pequenas diferenças, esta é outra sintaxe para algo que você já faz: abordar a manipulação.

Capítulo VI

Exercício 03: Violência desnecessária

	Exercício : 03
Violência desnecessária	
Diretório de entrada : <i>ex03/</i>	
Arquivos para entregar : Makefile, main.cpp, Weapon.{h, hpp}, Weapon.cpp, HumanA.{h, hpp}, HumanA.cpp, HumanB.{h, hpp}, HumanB.cpp	
Funções proibidas : Nenhuma	

Implementar uma classe de armas que tenha:

- Um tipo de atributo privado, que é uma corda.
- Uma função de membro getType() que retorna uma referência constante ao tipo.
- Uma função de membro setType() que define o tipo usando a nova função passada como parâmetro.

Agora, crie duas classes: **HumanA** e **HumanB**. Ambas têm uma Arma e um nome. Eles também têm um ataque de função de membro() que exhibe (é claro, sem os ângulos entre parênteses):

<nome> ataques com seu <tipo de arma>

HumanA e HumanB são quase os mesmos, exceto por estes dois pequenos detalhes:

- Enquanto a HumanA pega a Arma em seu construtor, a HumanB não o faz.
- A HumanB pode **não** ter **sempre** uma arma, enquanto a HumanA estará **sempre** armada.

Se sua implementação estiver correta, a execução do código a seguir imprimirá um ataque com "clube de pico bruto", então um segundo ataque com "algum outro tipo de clube" para ambos os casos de teste:

```
int main()
{
    {
        Clube das armas = Weapon("clube de espiga bruta");

        HumanA bob("Bob", clube);
        bob.attack();
        clube.setType("algum outro tipo de clube");
        bob.attack();
    }
    {
        Clube das armas = Weapon("clube de espiga bruta");

        HumanB jim("Jim");
        jim.setWeapon(clube);
        jim.attack();
        clube.setType("algum outro tipo de clube");
        jim.attack();
    }

    retornar 0;
}
```


Não se esqueça de verificar **vazamentos de memória**.



Em que caso você acha que seria melhor usar um ponteiro para Arma? E uma referência a Arma? Por quê? Pense sobre isso antes de iniciar este exercício.

Capítulo VII

Exercício 04: Sed é para os perdedores

	Exercício : 04
	Sed é para os perdedores
	Diretório de entrada : <i>ex04/</i>
	Arquivos para entregar : Makefile, main.cpp, *.cpp, *.{h, hpp}
	Funções proibidas : <code>std::string::replace</code>

Crie um programa que leve três parâmetros na seguinte ordem: um nome de arquivo e duas cordas, *s1* e *s2*.

Ele abrirá o arquivo *<filename>* e copiará seu conteúdo para um novo arquivo *<nome do arquivo>.substituir*, substituindo cada ocorrência de *s1* por *s2*.


O uso de funções de manipulação de arquivos C é proibido e será considerado trapaça. Todas as funções de membro da classe `std::string` são permitidas, exceto a substituição. Use-as sabiamente!

É claro, lidar com entradas e erros inesperados. Você tem que criar e entregar seus próprios testes para garantir que seu programa funcione como esperado.

Capítulo VIII

Exercício 05: Harl

2.0

	Exercício : 05 Harl 2.0
Diretório de entrada : <i>ex05/</i>	
Arquivos para entregar : <i>Makefile, main.cpp, Harl.{h, hpp}, Harl.cpp</i>	
Funções proibidas : <i>Nenhuma</i>	

Você conhece Harl? Todos nós conhecemos, não é mesmo? Caso você não conheça, veja abaixo o tipo de comentário que Harl faz. Eles são classificados por níveis:

- Nível **"DEBUG"**: As mensagens de debug contêm informações contextuais. Elas são usadas principalmente para diagnóstico de problemas.
Exemplo: *"Eu adoro ter bacon extra para o meu 7XL-duplo queijo-triplo picles-especial-hambúrguer de ketchup. Eu realmente faço!"*
- Nível **"INFO"**: Estas mensagens contêm amplas informações. Elas são úteis para rastrear a execução do programa em um ambiente de produção.
Exemplo: *"Não posso acreditar que acrescentar bacon extra custa mais dinheiro. Você não colocou bacon suficiente no meu hambúrguer! Se o fizesse, eu não estaria pedindo mais!"*
- Nível **"ADVERTÊNCIA"**: As mensagens de advertência indicam um problema potencial no sistema. Entretanto, ele pode ser tratado ou ignorado.
Exemplo: *"Acho que mereço ter um pouco mais de bacon de graça. Eu tenho vindo por anos, enquanto você começou a trabalhar aqui desde o mês passado".*
- Nível **"ERROR"**: Estas mensagens indicam a ocorrência de um erro irreversível. Esta é geralmente uma questão crítica que requer intervenção manual.
Exemplo: *"Isto é inaceitável! Quero falar agora com o gerente".*

Você vai automatizar o Harl. Não vai ser difícil, pois sempre diz as mesmas coisas. Você tem que criar uma classe **Harl** com as seguintes funções de membro particular:

- `void debug(void);`
- `informação vazia(vazio);`
- `aviso de nulidade (void);`
- `erro nulo(vazio);`

Harl também tem uma função de membro público que chama as quatro funções de membro acima, dependendo do nível passado como parâmetro:


`nulo complaint(std::string level);`

O objetivo deste exercício é utilizar **indicadores para as funções dos membros**. Isto não é uma sugestão. Harl tem que reclamar sem usar uma floresta de `if/else if/else`. Não pensa duas vezes!

Criar e entregar testes para mostrar que Harl reclama muito. Você pode usar os comentários de exemplo.

Capítulo IX

Exercício 06: Filtro Harl

	Exercício : 06
	Filtro Harl
	Diretório de entrada : <i>ex06/</i>
	Arquivos para entregar : <i>Makefile, main.cpp, Harl.{h, hpp}, Harl.cpp</i>
	Funções proibidas : Nenhuma

Às vezes você não quer prestar atenção a tudo o que Harl diz. Implemente um sistema para filtrar o que Harl diz, dependendo dos níveis de registro que você quer ouvir.

Criar um programa que tome como parâmetro um dos quatro níveis. Ele exibirá todas as mensagens a partir deste nível e acima. Por exemplo, um programa que tenha como parâmetro um dos quatro níveis:

```
$> ./harlFiltro
"ADVERTÊNCIA" [
ADVERTÊNCIA ]
Acho que mereço ter um pouco mais de bacon de graça.
Eu venho há anos, enquanto você começou a trabalhar aqui desde o mês passado.

A
Isto é inaceitável, quero falar agora com o gerente.

$> ./harlFilter "Não tenho certeza do cansaço que estou"
```

Embora existam várias maneiras de lidar com Harl, uma das mais efetivas é a de SWITCH.

Dê o nome de `harlFilter` ao seu executável.

Você deve usar, e talvez descobrir, a declaração de troca neste exercício.



Você pode passar este módulo sem fazer o exercício 06.

