



# C++ - Módulo 01

Alocação de memória, ponteiros para  
membros, referências, instrução switch

*Resumo:*

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

*Versão: 10*

# Conteúdo

<b>I</b>	<b>Introdução</b>	<b>2</b>
<b>II</b>	<b>Regras gerais</b>	<b>3</b>
<b>III</b>	<b>Exercício 00: Cérebrooo ...</b>	<b>5</b>
<b>IV</b>	<b>Exercício 01: Moar brainz!</b>	<b>6</b>
<b>V</b>	<b>Exercício 02: OLÁ, ESTE É O CÉREBRO</b>	<b>7</b>
<b>VI</b>	<b>Exercício 03: Violência desnecessária</b>	<b>8</b>
<b>VII</b>	<b>Exercício 04: Sed é para perdedores</b>	<b>10</b>
<b>VIII</b>	<b>Exercício 05: Harl 2.0</b>	<b>11</b>
<b>IX</b>	<b>Exercício 06: Filtro Harl</b>	<b>13</b>
<b>X</b>	<b>Submissão e avaliação por pares</b>	<b>14</b>

# 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: Cérebrooo ...

	Exercício: 00
BraiiiiiiiiinnzzzzZ	
Diretório de entrega: ex00/	
Arquivos para entregar: Makefile, main.cpp, Zombie.{h, hpp}, Zombie.cpp, newZombie.cpp, randomChump.cpp Funções proibidas: Nenhuma	

Primeiro, implemente uma classe **Zombie** . Ela tem um atributo privado string name.

Adicione uma função membro void announce( void ); à classe Zombie. Zumbis anunciam-se da seguinte forma:

<nome>: Cérebrooo ...

Não imprima os colchetes angulares (< e >). Para um zumbi chamado Foo, a mensagem seria:

Foo: Cérebrooo ...

Em seguida, implemente as duas funções a seguir:


- `Zumbi* novoZumbi( std::string nome );`  
Ele cria um zumbi, nomeia-o e retorna-o para que você possa usá-lo fora da função escopo.
- `void randomChump(std::string nome);`  
Ele cria um zumbi, dê um nome a ele e o zumbi se anuncia.

Agora, qual é o ponto real do exercício? Você tem que determinar em que caso é melhor alocar os zumbis na pilha ou heap.

Os zumbis devem ser destruídos quando você não precisar 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
Mais cérebros!	
Diretório de entrega: ex01/	
Arquivos para entregar: Makefile, main.cpp, Zombie.{h, hpp}, Zombie.cpp, zombieHorde.cpp	
Funções proibidas:	
Nenhuma	

Hora de criar uma **horda de zumbis**!

Implemente a seguinte função no arquivo apropriado:

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


Ela deve alocar N objetos Zumbis em uma única alocação. Então, ela tem que 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 `zombieHorde()` funcione conforme o esperado. Tente chamar `announce()` para cada um dos zumbis.

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

## Capítulo V

### Exercício 02: OLÁ, ESTE É O CÉREBRO

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

Escreva um programa que contenha:

- Uma variável de string inicializada como "OI, ESTE É O CÉREBRO".
- stringPTR: Um ponteiro para a string.
- stringREF: Uma referência à string.

Seu programa deve imprimir:

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

E então:


- 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: manipulação de endereços.



# Capítulo VI

## Exercício 03: Violência desnecessária

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

Implemente uma classe de arma que tenha:

- Um tipo de atributo privado, que é uma string.
- Uma função membro getType() que retorna uma referência const ao tipo.
- Uma função membro setType() que define o tipo usando o novo passado como parâmetro éter.

Agora, crie duas classes: **HumanA** e **HumanB**. Ambas têm uma Weapon e um nome. Elas também têm uma função membro attack() que exibe (claro, sem os colchetes angulares):

<nome> ataca com seu <tipo de arma>

HumanA e HumanB são quase iguais, exceto por esses dois pequenos detalhes:

- Enquanto HumanA pega a Arma em seu construtor, HumanB não.
- HumanoB pode **não ter sempre** uma arma, enquanto HumanoA **sempre** estará armado.

Se sua implementação estiver correta, executar o código a seguir imprimirá um ataque com "clube bruto com espinhos" e depois um segundo ataque com "algum outro tipo de clube" para ambos os casos de teste:

```
int principal()
{
    {
        Porrete de arma = Arma(" porrete com pontas brutas");

        HumanA bob("Bob", clube);
        bob.attack();
        clube.setType("algum outro tipo de clube"); bob.attack(); }

        Porrete de arma = Arma(" porrete com pontas brutas");

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

    }

    retornar 0;
}
```


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



Em qual caso você acha que seria melhor usar um ponteiro para Weapon? E uma referência para Weapon? Por quê? Pense nisso antes de começar este exercício.

## Capítulo VII

### Exercício 04: Sed é para perdedores

	Exercício: 04
Sed é para perdedores	
Diretório de entrega: ex04/	
Arquivos para entrega: Makefile, main.cpp, *.cpp, *.{h, hpp}	
Funções proibidas: std::string::replace	

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


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

Usar funções de manipulação de arquivo C é proibido e será considerado trapaça. Todas as funções membro da classe std::string são permitidas, exceto replace. Use-as com sabedoria!

Claro, lidar com entradas e erros inesperados. Você tem que criar e entregar seu Faça seus próprios testes para garantir que seu programa funcione conforme o esperado.

# Capítulo VIII

## Exercício 05: Harl 2.0

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

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

- Nível **"DEBUG"** : As mensagens de depuração contêm informações contextuais. Elas são usadas principalmente para diagnóstico de problemas.  
Exemplo: *"Adoro ter bacon extra no meu hambúrguer 7XL-duplo-queijo-triplo-picles-especial-ketchup. Adoro mesmo!"*
- Nível **"INFO"** : Essas mensagens contêm informações extensas. Elas são úteis para rastreando a execução do programa em um ambiente de produção.  
Exemplo: *"Não acredito que adicionar bacon extra custa mais dinheiro. Você não colocou bacon suficiente no meu hambúrguer! Se tivesse colocado, eu não estaria pedindo mais!"*
- Nível **"AVISO"** : mensagens de aviso indicam um possível problema no sistema.  
No entanto, ele pode ser tratado ou ignorado.  
Exemplo: *"Acho que mereço um pouco mais de bacon de graça. Venho aqui há anos, enquanto você começou a trabalhar aqui no mês passado."*
- Nível **"ERROR"** : Essas mensagens indicam que ocorreu um erro irreversível.  
Geralmente, esse é um problema crítico que requer intervenção manual.  
Exemplo: *"Isso é inaceitável! Quero falar com o gerente agora."*

Você vai automatizar o Harl. Não será difícil, pois ele sempre diz a mesma coisa coisas. Você tem que criar uma classe **Harl** com as seguintes funções de membro privadas:

- void debug( void );
- informação vazia( void );
- aviso nulo( void );
- erro nulo( nulo );

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


```
vazio      reclamar(std::string nível);
```

O objetivo deste exercício é usar **ponteiros para funções membro**. Isso não é uma sugestão. Harl tem que reclamar sem usar uma floresta de if/else if/else. Ele não pensa duas vezes!

Crie e entregue testes para mostrar que Harl reclama muito. Você pode usar os exemplos dos comentários listados acima no assunto ou escolha usar seus próprios comentários.

# Capítulo IX

## Exercício 06: Filtro Harl

	Exercício: 06
Filtro Harl	
Diretório de entrega: ex06/	
Arquivos para entregar: Makefile, main.cpp, Harl.{h, hpp}, Harl.cpp 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 log que você deseja ouvir.

Crie um programa que tome como parâmetro um dos quatro níveis. Ele exibirá todas as mensagens deste nível e acima. Por exemplo:

```
$> ./harlFilter "AVISO"
[ AVISO ]
Acho que mereço um pouco mais de bacon de graça.
Eu venho aqui há anos, enquanto você começou a trabalhar aqui no mês passado.

[ ERRO ]
Isso é inaceitável. Quero falar com o gerente agora.

$> ./harlFilter "Não tenho certeza de quão cansado estou hoje..."
[Provavelmente reclamando de problemas insignificantes]
```

Embora existam várias maneiras de lidar com Harl, uma das mais eficazes é DESLIGAR Harl.

Dê o nome harlFilter ao seu executável.

Você deve usar, e talvez descobrir, a instrução switch neste exercício.



Você pode passar neste módulo sem fazer o exercício 06.

## Capítulo X

### 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\$4f1b9de5b5e60c03dcb4e8c7c7e4072c