



# 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: 9.2*

# 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: Brailiiiiinnzzz</b>	<b>5</b>
<b>IV</b>	<b>Exercício 01: Moar brainz!</b>	<b>6</b>
<b>V</b>	<b>Exercício 02: OI ISSO É 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 de 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 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: BraiiiiiiinnzzzzZ

	Exercício: 00
BraiiiiiiinnzzzzZ	
Diretório de entrega: ex00/	
Arquivos a serem entregues: Makefile, main.cpp, Zombie.{h, hpp}, Zombie.cpp, newZombie.cpp, randomChump.cpp	
Funções proibidas: Nenhuma	

Primeiro, implemente uma classe **Zombie**. Ele tem um nome de atributo privado de cadeia de caracteres.

Adicione uma função de membro void anunciar( void ); para a classe Zumbi. zumbis se anunciam da seguinte forma:

<nome>: BraiiiiiiinnzzzzZ...

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

Foo: BraiiiiiiinnzzzzZ...

Em seguida, implemente as duas funções a seguir:


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

Agora, qual é o objetivo 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
Moar brainz!	
Diretório de entrega: ex01/	
Arquivos a serem entregues: 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);
```


Ele deve alocar N objetos Zombie em uma única alocação. Em seguida, ele 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 zombieHorde() funcione conforme o esperado. Tente chamar anuncie() para cada um dos zumbis.

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

## Capítulo V

### Exercício 02: OI ISSO É CÉREBRO

	Exercício: 02
OI ISSO É CÉREBRO	
Diretório de entrega: ex02/	
Arquivos a serem entregues: Makefile, main.cpp	
Funções proibidas: Nenhuma	

Escreva um programa que contenha:

- Uma variável de string inicializada como "HI THIS IS BRAIN".
- stringPTR: Um ponteiro para a string.
- stringREF: Uma referência à string.

Seu programa deve imprimir:

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

E então:


- O valor da variável de 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 endereço.



# Capítulo VI

## Exercício 03: Violência desnecessária

	Exercício: 03
violência desnecessária	
Diretório de entrega: ex03/	
Arquivos a 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 de membro getType() que retorna uma referência const ao tipo.
- Uma função de membro setType() que define o tipo usando o novo passado como parâmetro éter.

Agora, crie duas classes: **HumanA** e **HumanB**. Ambos têm uma arma e um nome. Eles também têm uma função de membro attack() que exibe (é claro, sem os colchetes):

<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.
- HumanB pode **nem sempre** ter uma arma, enquanto HumanA **sempre** estará armado.

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

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

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

        Porrete de arma = Arma(" Porrete bruto com pontas");

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

    }

    retorna 0;
}
```


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



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

## Capítulo VII

### Exercício 04: Sed é para perdedores

	Exercício: 04
Sed é para perdedores	
Diretório de entrega: ex04/	
Arquivos a serem entregues: 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 <filename> e copiará seu conteúdo em um novo arquivo <filename>.replace, substituindo cada ocorrência de s1 por s2.

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

Claro, lide com entradas e erros inesperados. Você tem que criar e entregar o seu 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 a serem entregues: Makefile, main.cpp, Harl.{h, hpp}, Harl.cpp	
Funções proibidas: Nenhuma	

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

- Nível **"DEBUG"** : As mensagens de depuração contêm informações contextuais. Eles são usados principalmente para diagnóstico de problemas.  
Exemplo: *"Adoro ter bacon extra para meu hambúrguer de ketchup especial 7XL-queijo duplo-picles-triplo. Eu realmente amo!"*
- Nível **"INFO"** : Estas mensagens contêm informações extensas. Eles 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 colocasse, eu não estaria pedindo mais!"*
- Nível **"AVISO"** : As mensagens de aviso indicam um possível problema no sistema.  
No entanto, pode ser manipulado ou ignorado.  
Exemplo: *"Acho que mereço um pouco de bacon extra de graça. Venho há anos, enquanto você começou a trabalhar aqui desde o mês passado."*
- Nível **"ERRO"** : Estas 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 Harl. Não será difícil, pois sempre diz o mesmo coisas. Você precisa criar uma classe **Harl** com as seguintes funções de membro privado:

- void debug( void );
- void info( void );
- aviso nulo (vazio);
- erro void( void );

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


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

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

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

# Capítulo IX

## Exercício 06: Filtro de Harl

	Exercício: 06
filtro Harl	
Diretório de entrega: ex06/	
Arquivos a serem entregues: Makefile, main.cpp, Harl.{h, hpp}, Harl.cpp	
Funções proibidas: Nenhuma	

Às vezes você não quer prestar atenção em tudo 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 de bacon extra de graça.
Eu venho aqui há anos, enquanto você começou a trabalhar aqui desde o mês passado.

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

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

Embora existam várias maneiras de lidar com Harl, uma das mais eficazes é desligá-lo.

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 atribuição em 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 de suas pastas e arquivos para garantir que estejam corretos.