



# C++ - Módulo 01

Alocação de memória, ponteiros para membros,  
referências e instruções switch

*Resumo:*

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

*Versão: 10.1*

# 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>6</b>
<b>IV</b>	<b>Exercício 01: Mais cérebros!</b>	<b>7</b>
<b>V</b>	<b>Exercício 02: OLÁ, ESTE É O CÉREBRO</b>	<b>8</b>
<b>VI</b>	<b>Exercício 03: Violência desnecessária</b>	<b>9</b>
<b>VII</b>	<b>Exercício 04: Sed é para perdedores</b>	<b>11</b>
<b>VIII</b>	<b>Exercício 05: Harl 2.0</b>	<b>12</b>
<b>IX</b>	<b>Exercício 06: Filtro Harl</b>	<b>14</b>
<b>X</b>	<b>Submissão e avaliação por pares</b>	<b>15</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 em C++. Muitas linguagens são recomendadas para aprender POO. Escolhemos C++ por ser derivada da sua velha amiga, 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. Portanto, se você quer se tornar um desenvolvedor C++ proficiente, cabe a você ir além, seguindo os 42 princípios básicos!

## 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, ... , exn
- Nomeie seus arquivos, classes, funções, funções de membro e atributos conforme necessário em as diretrizes.
- Escreva os nomes das classes no formato **UpperCamelCase** . Arquivos contendo código de classe sempre serão nomeados 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 da classe "BrickWall", que representa uma parede de tijolos, seu nome será BrickWall.hpp.
- A menos que especificado de outra forma, cada mensagem de saída deve terminar com um caractere de nova linha e ser exibida na saída padrão.
- *Adeus, Norminette!* Nenhum estilo de codificação é imposto nos módulos C++. Você pode seguir o seu favorito. Mas lembre-se de que código que seus avaliadores não conseguem entender é código que eles não podem avaliar. Faça o possível para escrever um código limpo e legível.

#### Permitido/Proibido

Você não está mais programando em C. Hora de C++! Portanto:

- Você pode usar quase tudo da biblioteca padrão. Portanto, em vez de se ater ao que você já conhece, seria inteligente usar as versões em C++ das funções C com as quais você está acostumado, sempre que possível.
- No entanto, você não pode usar nenhuma outra biblioteca externa. Isso significa que as bibliotecas C++11 (e 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 o contrário, o uso do namespace <ns\_name> e Palavras-chave de amigos são proibidas. Caso contrário, sua nota será -42.
- **Você só pode usar o STL nos Módulos 08 e 09.** Isso significa que não há **Contêineres** (vetor/lista/mapa e assim por diante) nem **Algoritmos** (qualquer coisa que exija a inclusão do cabeçalho <algoritmo>) até lá. Caso contrário, sua nota será -42.

### Alguns requisitos de design

- 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 na Igreja **Ortodoxa 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 conseguir usar cada um dos seus cabeçalhos independentemente dos outros. Portanto, eles devem incluir todas as dependências necessárias. No entanto, você deve evitar o problema de inclusão dupla adicionando **proteções de inclusão**. Caso contrário, sua nota será 0.

### Leia-me

- Você pode adicionar alguns arquivos adicionais, se necessário (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 isso.
- Por Odin, por Thor! Use seu cérebro!!!



Em relação ao Makefile para projetos C++, as mesmas regras do C se aplicam (veja o capítulo Norm sobre o Makefile).




Você terá que implementar muitas classes. Isso pode parecer tedioso, a menos que você saiba programar 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  
perca muitas informações úteis! Não hesite em ler sobre  
conceitos teóricos.

# Capítulo III

## Exercício 00: Cérebrooo ...

	Exercício: 00
Cérebrooo ...	
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 **Zumbi** . Ela tem um atributo string privado chamado name.

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

<nome>: Cérebroooooooooo...

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

Foo: Cérebrooo ...

Em seguida, implemente as duas funções a seguir:


- `Zombie* newZombie( std::string name );` Esta função cria um zumbi, dá um nome a ele e o retorna para que você possa usá-lo fora do escopo da função.
- `void randomChump(std::string nome);`  
Esta função cria um zumbi, dá um nome a ele e faz com que ele se anuncie.

Agora, qual é o objetivo real do exercício? Você precisa determinar em qual caso é melhor alocar zumbis na pilha ou no 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: Mais cérebros!

	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. Em seguida, deve inicializar os zumbis, atribuindo 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 usar `delete` para desalocar 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 de 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.

É isso — sem truques. O objetivo deste exercício é desmistificar referências, que podem parecer completamente novas. Embora existam algumas pequenas diferenças, esta é apenas 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 **Weapon** que tenha:

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

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

<nome> ataca com seu <tipo de arma>

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

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

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

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

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

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

        HumanB jim("Jim");
        jim.setWeapon(clube);
        jim.attack();
        clube.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 "Arm"? E uma referência a "Arm"? 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 deve abrir o arquivo <nome do arquivo> e copiar seu conteúdo para um novo arquivo <filename>.replace, substituindo cada ocorrência de s1 por s2.

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

Claro, lide com entradas e erros inesperados. Você deve criar e entregar seu testes próprios 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 entrega: Makefile, main.cpp, Harl.{h, hpp}, Harl.cpp Funções proibidas:	
Nenhuma	

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

- Nível **"DEBUG"** : As mensagens de depuração contêm informações contextuais. São usadas principalmente para diagnóstico de problemas.  
Exemplo: *"Adoro ter bacon extra no meu hambúrguer 7XL com queijo duplo, pickles triplos, ketchup especial. Adoro mesmo!"*
- Nível **"INFO"** : Essas mensagens contêm informações abrangentes. São úteis para rastreamento a execução do programa em um ambiente de produção.  
Exemplo: *"Não acredito que adicionar mais bacon custe mais. 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 precisa criar uma classe **Harl** com as seguintes funções-membro privadas:

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

**Harl** também tem uma função membro pública que chama as quatro funções 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 precisa 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 entrega: Makefile, main.cpp, Harl.{h, hpp}, Harl.cpp Funções proibidas:	
Nenhuma	

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

Crie um programa que receba como parâmetro um dos quatro níveis. Ele exibirá todas as mensagens deste nível em diante. 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. Somente o trabalho dentro do seu repositório será avaliado durante a defesa. Não hesite em verificar os nomes das suas pastas e arquivos para garantir que estejam corretos.



??????????? XXXXXXXXXXXX = \$3\$4f1b9de5b5e60c03dcb4e8c7c7e4072c