



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

EU	Introdução	2
II	Regras gerais	3
III	Exercício 00: Cérebrooo ...	6
4	Exercício 01: Mais cérebros!	7
V	Exercício 02: OLÁ, ESTE É O CÉREBRO	8
VI	Exercício 03: Violência desnecessária	9
VII	Exercício 04: Sed é para perdedores	11
VIII	Exercício 05: Harl 2.0	12
IX	Exercício 06: Filtro Harl	14
X	Submissão e avaliação por pares	15

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: [Wikipédia](#)).

O objetivo destes módulos é apresentar a você **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. Por ser 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 `g++` e as bandeiras `-Wall -Wextra -Werror`
- Seu código ainda deverá ser compilado 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 nas diretrizes.
- Escreva os nomes das classes em **UpperCamelCase** formato. Arquivos contendo código de classe sempre serão nomeados de acordo com o nome da classe. Por exemplo:
`NomeDaClasse.hpp`/`NomeDaClasse.h`, `NomeDaClasse.cpp`, ou `NomeDaClasse.hpp`. 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, 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 em C com as quais você está acostumado, sempre que possível.
- No entanto, você não pode usar nenhuma outra biblioteca externa. Isso significa C++11 (e formas derivadas) e **Impulsionar Bibliotecas** são proibidas. As seguintes funções também são proibidas: `*printf()`, `*alocação()` e `free()`. Se você usá-los, sua nota será 0 e pronto.

- Observe que, a menos que explicitamente declarado de outra forma, usando namespace <ns_name> e amigo Palavras-chave são proibidas. Caso contrário, sua nota será -42.
- **Você tem permissão para usar o STL somente nos Módulos 08 e 09.** Isso significa: não **Recipientes** (vetor/lista/mapa e assim por diante) e não **Algoritmos** (qualquer coisa que requeira incluir o <algoritmo> cabeçalho) até então. 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 projetadas no **Forma Canônica Ortodoxa, exceto quando explicitamente declarado 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. Portanto, eles devem incluir todas as dependências necessárias. No entanto, você deve evitar o problema de inclusão dupla adicionando **incluir guardas**. 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!!!



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 certa liberdade para completar os exercícios. No entanto, siga as regras obrigatórias e não seja preguiçoso. Você perderia 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, Zumbi.{h, hpp}, Zumbi.cpp, newZombie.cpp, randomChump.cpp	
Funções proibidas:Nenhum	

Primeiro, implemente um **Zumbi** classe. Possui um atributo de string privado **nome**.

Adicionar uma função de membro **vazio anunciar(vazio);** para a classe **Zumbi**. se **Zumbis** anunciam 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:


- **Zumbi* novoZumbi(std::string nome);**
Esta função cria um zumbi, nomeia-o 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.

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, Zumbi.{h, hpp}, Zumbi.cpp, zombieHorde.cpp	
Funções proibidas:Nenhum	

Hora de criar um **horda de zumbis**!

Implemente a seguinte função no arquivo apropriado:

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


Deve alocar **N** objetos zumbis em uma única alocação. Em seguida, ele 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 seu **Horda Zumbi()** a função funciona conforme o esperado. Tente chamar **anunciar()** para cada um dos zumbis.

Não se esqueça de usar **excluir** para desalocar todos os zumbis e verificar **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 entregar:Makefile, main.cpp	
Funções proibidas:Nenhum	

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 porstringPTR.
- O endereço de memória mantido porstringREF.


E então:

- O valor da variável string.
- O valor apontado porstringPTR.
- O valor apontado porstringREF.

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, Arma.{h, hpp}, Arma.cpp, HumanA.{h, hpp}, HumanA.cpp, HumanB.{h, hpp}, HumanB.cpp	
Funções proibidas: Nenhum	

Implementar um **Arma** classe que tem:

- Um atributo privado `tipo`, que é uma string.
- `getTipo()` função membro que retorna uma referência constante para `tipo`.
- `setTipo()` função membro que define `tipo` usando o novo valor passado como parâmetro.

Agora, crie duas classes: **Humana** e **HumanoB**. Ambos têm um `Arma` e um `nome`. Eles também têm uma função de membro `ataque()` 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** `arma` o `Arma` em seu construtor, **HumanoB** não.
- **HumanoB** poderia **nem sempre** ter uma arma, enquanto **Humana** **sempre** está 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()
{
    {
        Arma   clube = Arma("clube bruto com cravos");

        HumanoA bob("Prumo", clube);
        bob.ataque();
        clube.setType("algum outro tipo de clube");
        bob.ataque();
    }
    {
        Clube de armas = Arma("clube bruto com cravos");

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

return 0;
```


Não se esqueça de verificar **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 entregar: Makefile, main.cpp, *.cpp, *.{h, hpp}	
Funções proibidas: std::string::substituir	

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


Deve abrir o arquivo `<nome do arquivo>` e copie seu conteúdo para um novo arquivo `<nome do arquivo>.replace`, substituindo cada ocorrência de `s1` com `s2`.

O uso de funções de manipulação de arquivos C é proibido e será considerado trapaça. Todas as funções-membro da classe sequência padrão são permitidos, exceto `substituir`. Use-os com sabedoria!

Claro, lide com entradas e erros inesperados. Você deve criar e entregar 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:Nenhum

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:

- **"DEPURAR"** nível: As mensagens de depuração contêm informações contextuais. São usadas principalmente para diagnóstico de problemas.
Exemplo: *"Adoro bacon extra no meu hambúrguer 7XL com queijo duplo, pickles triplos e ketchup especial. Adoro mesmo!"*
- **"INFORMAÇÕES"** nível: Essas mensagens contêm informações abrangentes. Elas são úteis para rastrear a execução de programas em um ambiente de produção.
Exemplo: *"Não acredito que adicionar bacon extra custa mais caro. Você não colocou bacon suficiente no meu hambúrguer! Se tivesse colocado, eu não estaria pedindo mais!"*
- **"AVISO"** nível: Mensagens de aviso indicam um possível problema no sistema. No entanto, ele pode ser resolvido 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."*
- **"ERRO"** nível: Essas mensagens indicam que ocorreu um erro irreversível. Geralmente, trata-se de 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, já que ele sempre diz as mesmas coisas. Você precisa criar um **Harl** classe com as seguintes funções de membro privadas:

- void debug(void);
- informação vazia(void);
- aviso nulo(nulo);
- 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 de 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 de comentários listados acima no assunto ou optar por 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:Nenhum

Às vezes, você não quer prestar atenção a tudo o que o Harl diz. Implemente um sistema para filtrar o que o 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 `Filtro harl` para 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 em seu Git repositório como de costume. Somente 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.



???????????? XXXXXXXXXX = \$3\$4f1b9de5b5e60c03dcb4e8c7c7e4072c