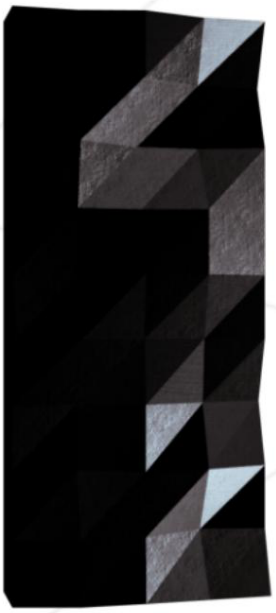

C++ - Módulo 01

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

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

Versão: 10

Conteúdo

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 com Classes" (fonte: Wikipedia).

O objetivo desses módulos é apresentar a Programação Orientada a Objetos.
Este será o ponto de partida de sua jornada em C++. Muitas linguagens são recomendadas para aprender OOP. Decidimos escolher C++ porque ele é derivado do 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. Então se você quer se tornar um desenvolvedor C++ proficiente, cabe a você ir além após o 42 Common Core!

Capítulo II

Regras gerais

Compilando

- Compile seu código com c++ e as tags -Wall -Wextra -Werror
- Seu código ainda deverá ser compilado se você adicionar a tag -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 exigido em As diretrizes.
- Escreva nomes de 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 finalizadas com uma nova linha caractere e exibido na saída padrão.
- Adeus Norminette! Nenhum estilo de codificação é imposto nos módulos C++. Você pode seguir o seu favorito. Mas tenha em mente 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. É hora de C++! Portanto:

- Você tem permissão para usar quase tudo da biblioteca padrão. Assim, em vez de se ater ao que você já sabe, seria inteligente usar o máximo possível as versões C++ das funções C com as quais você está acostumado.
- Entretanto, você não pode usar nenhuma outra biblioteca externa. Isso significa que C++ 11 (e formas derivadas) e bibliotecas 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, salvo indicação explícita em contrário, o namespace using <ns_name> e palavras-chave de amigos são proibidas. Caso contrário, sua nota será -42.
- É permitido utilizar o STL somente nos Módulos 08 e 09. Isso significa: nenhum contêiner (vetor/lista/mapa/e assim por diante) e nenhum algoritmo (qualquer coisa que exija a inclusão do cabeçalho <algorithm>) até então. Caso contrário, sua nota será -42.

Alguns requisitos de design

- 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 planejadas no estilo Ortodoxo Forma Canônica, exceto quando explicitamente indicado 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. Assim, eles devem incluir todas as dependências de que necessitam. No entanto, você deve evitar o problema da 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 tarefas não são verificadas por um programa, fique à vontade para fazê-lo, desde que entregue os arquivos obrigatórios.
- Às vezes, as orientações 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 consiga criar um script em seu editor de texto favorito.

Você tem uma certa liberdade para completar os exercícios.
Porém, siga as regras obrigatórias e não seja preguiçoso. Você poderia perca muitas informações úteis! Não hesite em ler sobre conceitos teóricos.

Capítulo III

Exercício 00: BraiiiiiiinnnzzzZ

	Exercício: 00
BraiiiiiiinnnzzzZ	
Diretório de entrega: ex00ÿ	
Arquivos para entregar: Makefile, main.cpp, Zombie.{h, hpp}, Zombie.cpp, newZombie.cpp, randomChump.cpp Funções proibidas: Nenhum	

Primeiro, implemente uma classe Zombie. Possui um nome de atributo privado de string.
Adicione uma função membro void anuncia(void); para a classe Zumbi. Zumbis anunciam-se da seguinte forma:

<nome>: BraiiiiiiinnnzzzZ...

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

Foo: BraiiiiiiinnnzzzZ...

Em seguida, implemente as duas funções a seguir:

- Zumbi* newZombie(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, nomeie-o e o zumbi se anuncia.

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

Os zumbis devem ser destruídos quando você não precisar mais deles. O destruidor deve imprima uma mensagem com o nome do zumbi para fins de depuração.

Capítulo IV

Exercício 01: Moar brainz!

	Exercício: 01
Moar brainz!	
Diretório de entrega: ex01ÿ	
Arquivos para entregar: Makefile, main.cpp, Zombie.{h, hpp}, Zombie.cpp, zombieHorde.cpp	
Funções proibidas:	
Nenhum	

É hora de criar uma horda de zumbis!

Implemente a seguinte função no arquivo apropriado:

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

Deve alocar N objetos Zombie em uma única alocação. Em seguida, deve-se 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 anúncio() para cada um dos zumbis.

Não se esqueça de deletar 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: ex02ÿ	
Arquivos para entregar: Makefile, main.cpp Funções	
proibidas: Nenhum	

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 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: ex03ÿ	
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 para type.
- Uma função 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 membro attack() que exibe (é claro, sem os colchetes angulares):

<nome> ataca com seu <tipo de arma>

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

- Enquanto HumanA leva a Arma em seu construtor, HumanB não.
- HumanB nem sempre terá uma arma, enquanto HumanA estará sempre armado.

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

```
int principal()
{
    {
        Clube de armas = Arma(" clube com pontas brutas");

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

    }
    {
        Clube de armas = Arma(" clube 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.

Nesse caso você acha que seria melhor usar um ponteiro para Arma? E uma referência à 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: ex04ÿ	
Arquivos para entregar: 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 strings, s1 e s2.

Ele abrirá o arquivo <nome do arquivo> e copiará seu conteúdo para um novo arquivo <nome do arquivo>.replace, substituindo todas as ocorrências de s1 por s2.

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

Claro, lide com entradas e erros inesperados. Você tem que criar e entregar 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: ex05ÿ	
Arquivos para entregar: Makefile, main.cpp, Harl.{h, hpp}, Harl.cpp Funções proibidas:	
Nenhum	

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

- Nível "DEBUG": As mensagens de depuração contêm informações contextuais. Eles são usados principalmente para diagnóstico de problemas.
  Exemplo: "Adoro comer bacon extra no meu hambúrguer 7XL com queijo duplo, triplo picles e ketchup especial. Adoro mesmo!"

- Nível "INFO": Estas mensagens contêm informações extensas. Eles são úteis para rastrear a execução do programa em um ambiente de produção.
  Exemplo: "Não acredito que adicionar bacon extra custe mais dinheiro. Você não colocou bacon suficiente no meu hambúrguer! Se colocasse, eu não estaria pedindo mais!"

- Nível "WARNING": Mensagens de aviso indicam um possível problema no sistema.
  No entanto, ele pode ser tratado 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 "ERROR": Estas mensagens indicam que ocorreu um erro irrecuperável.
  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 a mesma coisa coisas. Você deve criar uma classe Harl com as seguintes funções de membro privado:

- depuração nula(void);
- informação nula(void);
- aviso de void(void);
- erro de void(void);

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. Isto 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 dos comentários listados acima no assunto ou opte por usar seus próprios comentários.

Capítulo IX

Exercício 06: Filtro Harl

	Exercício: 06
Filtro Harl	
Diretório de entrega: ex06ÿ	
Arquivos para entregar: Makefile, main.cpp, Harl.{h, hpp}, Harl.cpp Funções proibidas:	
Nenhum	

Às vezes você não quer prestar atenção em tudo que Harl diz. Implementar um sistema para filtrar o que Harl diz dependendo dos níveis de registro 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.
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 quão cansado estou hoje..."
 [Provavelmente reclamando de problemas insignificantes]
```

Embora existam diversas 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

Envio e avaliação por pares

Entregue sua tarefa em seu repositório Git normalmente. Somente o trabalho dentro do seu repositório será avaliado durante a defesa. Não hesite em verificar os nomes de suas pastas e arquivos para garantir que estejam corretos.

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