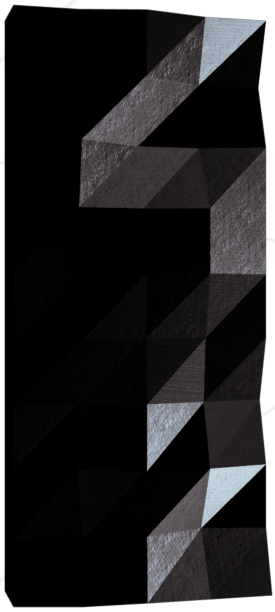

C++ - Módulo 01

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

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

Versão: 11.0

Sumário

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 você 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++ porque ela deriva 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.

Estamos cientes de que o C++ moderno é muito diferente em muitos aspectos. Então, se você quiser se tornar um desenvolvedor C++ proficiente, cabe a você ir mais longe após o 42 Common Core!

Capítulo II

Regras gerais

Compilação

- Compile seu código com `c++` e as flags `-Wall -Wextra -Werror`
- Seu código ainda deve compilar se você adicionar a flag `-std=c++98`

Formatação e convenções de nomenclatura

- Os diretórios dos exercícios serão nomeados desta forma: `ex00, ex01, ... , exn`
- Nomeie seus arquivos, classes, funções, funções membro e atributos conforme exigido nas 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:
  `NomeDaClasse.hpp/NomeDaClasse.h`, `NomeDaClasse.cpp`, ou `NomeDaClasse.tpp`.
  Então, se você tiver um arquivo de cabeçalho contendo a definição de uma classe "ParedeDeTijolo"representando uma parede de tijolos, seu nome será `ParedeDeTijolo.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 que o código que seus 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 programando em C. É hora de C++! Portanto:

- Você pode usar quase tudo da biblioteca padrão. Assim, em vez de se ater ao que você já conhece, seria inteligente usar as versões em C++ das funções C que você está acostumado o máximo possível.

- No entanto, você não pode usar nenhuma outra biblioteca externa. Isso significa que 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, as palavras-chave `using namespace <ns_name>` e `friend` são proibidas. Caso contrário, sua nota será -42.

- **Você só pode usar a STL nos Módulos 08 e 09.** Isso significa: nenhum **Contêiner** (vector/list/map, e assim por diante) e nenhum **Algoritmo** (qualquer coisa que exija a inclusão do cabeçalho `<algorithm>`) até lá. Caso contrário, sua nota será -42.

Alguns requisitos de design

- Vazamento de memória ocorre em C++ também. Quando você aloca memória (usando a palavra-chave `new`), você deve evitar **vazamentos de memória**.

- Do Módulo 02 ao Módulo 09, suas classes devem ser projetadas na **Forma Canônica Ortodoxa, 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 de seus cabeçalhos independentemente de outros. Portanto, eles devem incluir todas as dependências de que precisam. 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 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 você 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 (consulte o capítulo Norma sobre o Makefile).`

Você terá que implementar muitas classes. Isso pode parecer tedioso, a menos que você seja capaz de criar scripts para 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 perderia muitas informações úteis! Não hesite em ler sobre conceitos teóricos.

Capítulo III

Instruções de IA

Contexto

Este projeto foi desenvolvido para ajudá-lo a descobrir os blocos de construção fundamentais do seu treinamento em TIC.

Para ancorar adequadamente os conhecimentos e habilidades-chave, é essencial adotar uma abordagem criteriosa ao uso de ferramentas e suporte de IA.

A verdadeira aprendizagem fundamental exige um esforço intelectual genuíno — através de desafios, repetição e trocas de aprendizagem entre pares.

Para uma visão geral mais completa de nossa posição sobre a IA — como ferramenta de aprendizagem, como parte do currículo de TIC e como expectativa no mercado de trabalho — consulte as perguntas frequentes dedicadas na intranet.

Mensagem principal

- Construa bases sólidas sem atalhos.
- Desenvolva verdadeiramente habilidades técnicas e de poder.
- Experimente a verdadeira aprendizagem entre pares, comece a aprender como aprender e resolver novos problemas.
- A jornada de aprendizagem é mais importante que o resultado.
- Aprenda sobre os riscos associados à IA e desenvolva práticas eficazes de controle e contramedidas para evitar armadilhas comuns.

Regras para o aluno:

- Você deve aplicar o raciocínio às suas tarefas atribuídas, especialmente antes de recorrer à IA.
- Você não deve pedir respostas diretas à IA.
- Você deve aprender sobre a abordagem global da 42 em relação à IA.

Resultados da fase:

Nesta fase fundamental, você obterá os seguintes resultados:

- Obter bases sólidas em tecnologia e codificação.
- Saber por que e como a IA pode ser perigosa durante esta fase.

Comentários e exemplo:

- Sim, sabemos que a IA existe — e sim, ela pode resolver seus projetos. Mas você está aqui para aprender, não para provar que a IA aprendeu. Não perca seu tempo (nem o nosso) apenas para demonstrar que a IA pode resolver o problema dado.
- Aprender na 42 não é sobre saber a resposta — é sobre desenvolver a capacidade de encontrar uma. A IA lhe dá a resposta diretamente, mas isso o impede de construir seu próprio raciocínio. E o raciocínio leva tempo, esforço e envolve falhas. O caminho para o sucesso não deve ser fácil.
- Lembre-se de que durante os exames, a IA não estará disponível — sem internet, sem smartphones, etc. Você perceberá rapidamente se confiou demais na IA em seu processo de aprendizagem.
- A aprendizagem entre pares o expõe a diferentes ideias e abordagens, melhorando suas habilidades interpessoais e sua capacidade de pensar de forma divergente. Isso é muito mais valioso do que apenas conversar com um bot. Então não seja tímido — converse, faça perguntas e aprenda juntos!
- Sim, a IA fará parte do currículo — tanto como ferramenta de aprendizagem quanto como um tópico em si. Você até terá a chance de construir seu próprio software de IA. Para saber mais sobre nossa abordagem crescente, consulte a documentação disponível na intranet.

✓ Boa prática:

> Estou travado em um novo conceito. Pergunto a alguém próximo como ele abordou isso. Conversamos por 10 minutos — e de repente, clica. Entendi.

✗ Má prática:

> Uso secretamente a IA, copio algum código que parece certo. Durante a avaliação entre pares, não consigo explicar nada. Eu falho. Durante o exame — sem IA — estou travado novamente. Eu falho.

Capítulo IV

Exercício 00: BraiiiiiiinnnzzzZ

	Exercice : 00
BraiiiiiiinnnzzzZ	
Pasta de entrega : $ex00/$	
Arquivos para entregar : `Makefile, main.cpp, Zombie.{h, hpp}, Zombie.cpp, newZombie.cpp, randomChump.cpp`	
Funções não permitidas : `Nenhuma`	

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

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

```
<name>: BraiiiiiiinnnzzzZ...
```

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

```
Foo: BraiiiiiiinnnzzzZ...
```

Então, implemente as duas funções seguintes:

- `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 name );`
  Esta função cria um zumbi, dá um nome a ele e o faz se anunciar.

Agora, qual é o objetivo real do exercício? Você tem que determinar em qual caso é melhor alocar zumbis na stack ou no heap.

Zumbis devem ser destruídos quando você não precisar mais deles. O destrutor deve imprimir uma mensagem com o nome do zumbi para fins de depuração.

Capítulo V

Exercício 01: Moar brainz!

	Exercice : 01
Moar brainz!	
Pasta de entrega : $ex01/$	
Arquivos para entregar : `Makefile, main.cpp, Zombie.{h, hpp}, Zombie.cpp, zombieHorde.cpp`	
Funções não permitidas : `Nenhuma`	

Hora de criar uma **horda de Zumbis**!

Implemente a seguinte função no arquivo apropriado:

```
Zombie* zombieHorde( int N, std::string name );
```

Ela deve alocar `N` objetos Zombie em uma única alocação. Então, ela deve inicializar os zumbis, dando a cada um deles o nome passado como um parâmetro. A função retorna um ponteiro para o primeiro zumbi.

Implemente seus próprios testes para garantir que sua função `zombieHorde()` funcione como 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 VI

Exercício 02: HI THIS IS BRAIN

	Exercice : 02
HI THIS IS BRAIN	
Pasta de entrega : $ex02/$	
Arquivos para entregar : `Makefile, main.cpp`	
Funções não permitidas : `Nenhuma`	

Escreva um programa que contenha:

- Uma variável string inicializada com "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`.

É isso — sem truques. O objetivo deste exercício é desmistificar as referências, que podem parecer completamente novas. Embora existam algumas pequenas diferenças, esta é simplesmente outra sintaxe para algo que você já faz: manipulação de endereços.

Capítulo VII

Exercício 03: Unnecessary violence

	Exercice : 03
Violência desnecessária	
Pasta 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 não permitidas : `Nenhuma`	

Implemente uma classe **Weapon** que tenha:

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

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

```
<name> attacks with their <tipo de arma>
```

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

- Enquanto **HumanA** recebe a `Weapon` em seu construtor, **HumanB** não.
- **HumanB** pode **nem sempre** ter uma arma, enquanto **HumanA** sempre **sempre** estará armado.

Se sua implementação estiver correta, a execução do seguinte código imprimirá um ataque com "crude spiked club"seguido por um segundo ataque com "some other type of club"para ambos os casos de teste:

```
int main()
{
  {
    Weapon  club = Weapon("crude spiked club");

    HumanA bob("Bob", club);
    bob.attack();
    club.setType("some other type of club");
    bob.attack();
  }
  {
    Weapon  club = Weapon("crude spiked club");

    HumanB jim("Jim");
    jim.setWeapon(club);
    jim.attack();
    club.setType("some other type of club");
    jim.attack();
  }

  return 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 sobre isso antes de começar este exercício.

Capítulo VIII

Exercício 04: Sed is for losers

	Exercice : 04
Sed é para perdedores	
Pasta de entrega : $ex04/$	
Arquivos para entregar : `Makefile, main.cpp, *.cpp, *.{h, hpp}`	
Funções não permitidas : `std::string::replace`	

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

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

Usar funções de manipulação de arquivos 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 seus próprios testes para garantir que seu programa funcione como esperado.

Capítulo IX

Exercício 05: Harl 2.0

	Exercice : 05
Harl 2.0	
Pasta de entrega : $ex05/$	
Arquivos para entregar : `Makefile, main.cpp, Harl.{h, hpp}, Harl.cpp`	
Funções não permitidas : `Nenhuma`	

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

- Nível "**DEBUG**": Mensagens de depuração contêm informações contextuais. Elas são usadas principalmente para diagnóstico de problemas.
  Exemplo: *"Eu amo ter bacon extra para o meu hambúrguer 7XL-duplo-queijo-triplo-picles-ketchup-especial. Eu realmente amo!"*

- Nível "**INFO**": Estas mensagens contêm informações extensas. Elas são úteis para rastrear a execução do programa em um ambiente de produção.
  Exemplo: *"Eu não acredito que adicionar bacon extra custa mais dinheiro. Vocês não colocaram bacon suficiente no meu hambúrguer! Se vocês tivessem colocado, eu não estaria pedindo por mais!"*

- Nível "**WARNING**": Mensagens de aviso indicam um problema potencial no sistema. No entanto, ele pode ser tratado ou ignorado.
  Exemplo: *"Eu acho que mereço ter bacon extra de graça. Eu venho aqui há anos, enquanto você começou a trabalhar aqui apenas no mês passado."*

- Nível "**ERROR**": Estas mensagens indicam que ocorreu um erro irrecuperável. Este é geralmente um problema crítico que requer intervenção manual.
  Exemplo: *"Isto é inaceitável! Eu quero falar com o gerente agora."*

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

- `void debug( void );`
- `void info( void );`
- `void warning( void );`
- `void error( void );`

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

```
void    complain( std::string level );
```

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. 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 escolher usar comentários de sua autoria.

Capítulo X

Exercício 06: Harl filter

	Exercice : 06
Harl filter	
Pasta de entrega : $ex06/$	
Arquivos para entregar : `Makefile, main.cpp, Harl.{h, hpp}, Harl.cpp`	
Funções não permitidas : `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 quer ouvir.

Crie um programa que receba como parâmetro um dos quatro níveis. Ele irá exibir todas as mensagens a partir deste nível e acima. Por exemplo:

```
$> ./harlFilter "WARNING"
[ WARNING ]
Eu acho que mereço ter bacon extra de graça.
Eu venho aqui há anos, enquanto você começou a trabalhar aqui apenas no mês passado.

[ ERROR ]
Isto é inaceitável! Eu quero falar com o gerente agora.

$> ./harlFilter "Eu não tenho certeza do quão cansado eu estou hoje..."
[ Provavelmente reclamando sobre 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 declaração switch neste exercício.

`Você pode passar este módulo sem fazer o exercício 06.`

Capítulo XI

Entrega e avaliação por pares

Entregue sua tarefa 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 eles estejam corretos.

Durante a avaliação, uma breve **modificação do projeto** pode ser ocasionalmente solicitada. Isso pode envolver uma pequena mudança de comportamento, algumas linhas de código para escrever ou reescrever, ou um recurso fácil de adicionar.

Embora esta etapa possa **não ser aplicável a todos os projetos**, você deve estar preparado para ela se for mencionada nas diretrizes de avaliação.

Esta etapa visa verificar sua compreensão real de uma parte específica do projeto. A modificação pode ser realizada em qualquer ambiente de desenvolvimento que você escolher (por exemplo, sua configuração usual), e deve ser viável em poucos minutos — a menos que um prazo específico seja definido como parte da avaliação.
Você pode, por exemplo, ser solicitado a fazer uma pequena atualização em uma função ou script, modificar uma exibição ou ajustar uma estrutura de dados para armazenar novas informações, etc.

Os detalhes (escopo, alvo, etc.) serão especificados nas **diretrizes de avaliação** e podem variar de uma avaliação para outra para o mesmo projeto.

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