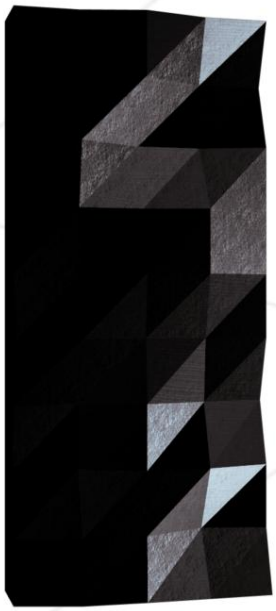

C++ - Módulo 08

Contêineres, iteradores e algoritmos modelados

Resumo:
Este documento contém os exercícios do Módulo 08 dos módulos C++.

Versão: 8

Conteúdo

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 C++. Muitas linguagens são recomendadas para aprender POO. Decidimos escolher C++, pois é derivado 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.

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 depois 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 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 nomes de classes no formato **UpperCamelCase .** Arquivos contendo código de classe serão sempre será 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" representando uma parede de tijolos, seu nome será BrickWall.hpp.
- A menos que especificado de outra forma, todas as mensagens de saída devem ser encerradas por uma nova linha caractere e exibido na saída padrão.
- *Adeus Norminette!* Nenhum estilo de codificação é imposto nos módulos C++. Você pode seguir seu favorito. Mas tenha em mente que um código que seus avaliadores pares não conseguem entender é um código que eles não conseguem classificar. 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. Assim, em vez de ficar preso ao que você já sabe, seria inteligente usar o máximo possível as versões C++-ish das funções C com as 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-las, sua nota será 0 e pronto.

- Observe que, a menos que explicitamente indicado de outra forma, o uso do namespace <ns_name> e palavras-chave de amigos são proibidas. Caso contrário, sua nota será -42.
- **Você tem permissão para usar o STL somente no Módulo 08 e 09.** Isso significa: nenhum **Container** (vetor/lista/mapa/e assim por diante) e nenhum **Algoritmo** (qualquer coisa que exija incluir o cabeçalho <algoritmo>) até então. Caso contrário, sua nota será -42.

Alguns requisitos de design

- Vazamento de memória ocorre em C++ também. 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 **Língua Ortodoxa Forma canônica, 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 dos 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 (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!!!

Você terá que implementar muitas classes. Isso pode parecer tedioso, a menos que você consiga criar um script no 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

Regras específicas do módulo

Você notará que, neste módulo, os exercícios podem ser resolvidos SEM os Containers padrão e SEM os Algoritmos padrão.

No entanto, **usá-los é precisamente o objetivo deste Módulo.** Você tem permissão para usar o STL. Sim, você pode usar os **Containers** (vetor/lista/mapa/e assim por diante) e os **Algoritmos** (definidos no cabeçalho <algoritmo>). Além disso, você deve usá-los o máximo que puder. Portanto, faça o seu melhor para aplicá-los onde for apropriado.

Você receberá uma nota muito ruim se não fizer isso, mesmo que seu código funcione como esperado. Por favor, não seja preguiçoso.

Você pode definir seus modelos nos arquivos de cabeçalho como de costume. Ou, se quiser, você pode escrever suas declarações de modelo nos arquivos de cabeçalho e escrever suas implementações em arquivos .tpp. Em qualquer caso, os arquivos de cabeçalho são obrigatórios enquanto os arquivos .tpp são opcionais.

Capítulo IV

Exercício 00: Fácil de encontrar

	Exercício: 00
Fácil de encontrar	
Diretório de entrega: ex00/	
Arquivos para entrega: Makefile, main.cpp, easyfind.{h, hpp} e arquivo opcional: easyfind.tpp Funções proibidas: Nenhuma	

Um primeiro exercício fácil é a maneira de começar com o pé direito.

Escreva um modelo de função easyfind que aceite um tipo T. Ele aceita dois parâmetros. O primeiro é do tipo T e o segundo é um inteiro.

Supondo que T seja um contêiner **de inteiros,** esta função precisa encontrar a primeira ocorrência do segundo parâmetro no primeiro parâmetro.

Se nenhuma ocorrência for encontrada, você pode lançar uma exceção ou retornar um valor de erro de sua escolha. Se precisar de inspiração, analise como os contêineres padrão se comportam.

Claro, implemente e entregue seus próprios testes para garantir que tudo funcione conforme o esperado.

Você não precisa manipular contêineres associativos.

Capítulo V

Exercício 01: Span

	Exercício: 01
Extensão	
Diretório de entrega: ex01/	
Arquivos para entrega: Makefile, main.cpp, Span.{h, hpp}, Span.cpp Funções proibidas:	
Nenhuma	

Desenvolva uma classe **Span** que possa armazenar no máximo N inteiros. N é uma variável unsigned int e será o único parâmetro passado ao construtor.

Esta classe terá uma função membro chamada addNumber() para adicionar um único número ao Span. Ela será usada para preenchê-lo. Qualquer tentativa de adicionar um novo elemento se já houver N elementos armazenados deve lançar uma exceção.

Em seguida, implemente duas funções de membro: shortestSpan() e longestSpan()

Eles descobrirão respectivamente o menor intervalo ou o maior intervalo (ou distância, se preferir) entre todos os números armazenados e retornarão. Se não houver números armazenados, ou apenas um, nenhum intervalo poderá ser encontrado. Portanto, lance uma exceção.

Claro, você escreverá seus próprios testes e eles serão muito mais completos do que os abaixo. Teste seu Span com pelo menos 10.000 números. Mais seria ainda melhor.

Executando este código:

```
int principal() {

    Período sp = Período(5);

    sp.addNumber(6);
    sp.addNumber(3);
    sp.addNumber(17);
    sp.addNumber(9);
    sp.addNumber(11);

    std::cout << sp.shortestSpan() << std::endl; std::cout << sp.longestSpan()
    << std::endl;

    retornar 0;
}
```

Deve produzir:

```
$> ./ex01 2

14
$>
```

Por último, mas não menos importante, seria maravilhoso preencher seu Span usando uma **variedade de iteradores.** Fazer milhares de chamadas para addNumber() é muito chato. Implemente uma função membro para adicionar muitos números ao seu Span em uma chamada.

Se você não tem ideia, estude os Containers. Algumas funções de membro usam uma série de iteradores para adicionar uma sequência de elementos para o contêiner.

Capítulo VI

Exercício 02: Abominação mutante

	Exercício: 02
Abominação mutante	
Diretório de entrega: ex02/	
Arquivos para entrega: Makefile, main.cpp, MutantStack.{h, hpp} e arquivo opcional: MutantStack.tpp Funções proibidas: Nenhuma	

Agora, é hora de passar para coisas mais sérias. Vamos desenvolver algo estranho.

O contêiner std::stack é muito bom. Infelizmente, é um dos únicos Con-tainers STL que NÃO é iterável. Que pena.

Mas por que aceitaríamos isso? Especialmente se podemos tomar a liberdade de massacrar o pilha original para criar recursos ausentes.

Para reparar essa injustiça, você precisa tornar o contêiner std::stack iterável.

Escreva uma classe **MutantStack .** Ela será **implementada em termos de** std::stack.
Ele oferecerá todas as suas funções de membro, além de um recurso adicional: **iteradores.**

É claro que você escreverá e entregará seus próprios testes para garantir que tudo funcione conforme o esperado.

Veja um exemplo de teste abaixo.

```
int principal()
{
Pilha Mutante<int>                pilha m;

mstack.push(5);
mstack.push(17);

std::cout << mstack.top() << std::endl;

mstack.pop();

std::cout << mstack.size() << std::endl;

mstack.push(3);
mstack.push(5);
mstack.push(737); //[...]

mstack.push(0);

MutantStack<int>::iterador it = mstack.begin();
MutantStack<int>::iterador ite = mstack.end();

++it; --
it;
enquanto (it != ite) {

        std::cout << *it << std::endl; ++it;

}
std::stack<int> s(mstack); retornar 0; }
```

Se você executá-lo uma primeira vez com seu MutantStack, e uma segunda vez substituindo o MutantStack por, por exemplo, um std::list, as duas saídas devem ser as mesmas. Claro, ao testar outro contêiner, atualize o código abaixo com as funções de membro correspondentes (push() pode se tornar push_back()).

Capítulo VII

Submissão e avaliação por pares

Entregue sua tarefa no 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 das suas pastas e arquivos para garantir que estejam corretos.

16D85ACC441674FBA2DF65195A38EE36793A89EB04594B15725A1128E7E97B0E7B47
111668BD6823E2F873124B7E59B5CE94B47AB764CF0AB316999C56E5989B4B4F00C
91B619C70263F