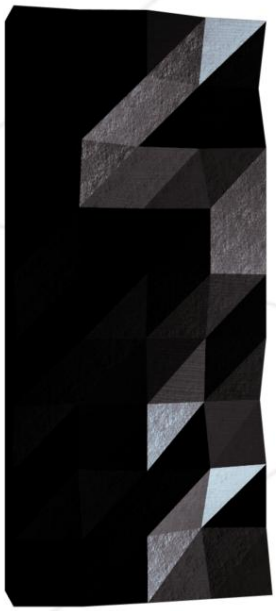
Machine Translated by Google



C++ - Módulo 08

Contêineres modelados, iteradores, algoritmos

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

Versão: 7

Machine Translated by Google

Conteúdo

Machine Translated by Google

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.

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

Machine Translated by Google

Capítulo II

Regras gerais

Compilando

- Compile seu código com c++ e os sinalizadores -Wall -Wextra -Werror
- Seu código ainda deverá ser compilado se você adicionar o sinalizador -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.

Machine Translated by Google

- 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 atribuições 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.

Machine Translated by Google

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.

Contudo, **utilizá-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 tanto quanto puder. Portanto, faça o possível para aplicá-los sempre que for apropriado.

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

Você pode definir seus modelos nos arquivos de cabeçalho normalmente. Ou, se desejar, 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.

Machine Translated by Google

Capítulo IV

Exercício 00: Fácil localização

	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 começar com o pé direito.

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

Assumindo que T é um contêiner **de inteiros,** esta função deve encontrar a primeira ocorrência do segundo parâmetro no primeiro parâmetro.

Se nenhuma ocorrência for encontrada, você poderá lançar uma exceção ou retornar um valor de erro da 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 esperado.

Você não precisa lidar com contêineres associativos.

Machine Translated by Google

Capítulo V

Exercício 01: Extensão

	Exercício: 01
Período	
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 int não assinada e será o único parâmetro passado ao construtor.

Esta classe terá uma função membro chamada addNumber() para adicionar um único número ao Span. Será utilizado para preenchê-lo. Qualquer tentativa de adicionar um novo elemento se já houver N elementos armazenados deverá gerar uma exceção.

A seguir, implemente duas funções-membro:shortSpan() e LongSpan()

Eles descobrirão respectivamente o intervalo mais curto ou o intervalo mais longo (ou distância, se preferir) entre todos os números armazenados e o retornarão. Se não houver nenhum número armazenado, 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.

Machine Translated by Google

Executando este código:

```
int principal()
{
    Span sp = Span(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 vários **iteradores.** Fazer milhares de chamadas para addNumber() é muito chato. Implemente uma função membro para adicionar vários números ao seu Span em uma chamada.

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

Machine Translated by Google

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 avançar para coisas mais sérias. Vamos desenvolver algo estranho.

O contêiner std::stack é muito bom. Infelizmente, é um dos únicos contêineres STL que NÃO é iterável. Isso é ruim.

Mas por que aceitaríamos isso? Especialmente se pudermos 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 .** Ele será **implementado em termos de** std::stack.
Ele oferecerá todas as funções de membros, além de um recurso adicional: **iteradores.**

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

Machine Translated by Google

Encontre um exemplo de teste abaixo.

```
int principal()
{
MutantStack<int>                pilha;

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();

++isto;
--isto;
enquanto (isso! = ite) {

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

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

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