# Lista de Exercícios 2

Universidade Federal de Santa Catarina
Campus de Joinville
Centro Tecnológico de Joinville
EMB5631 – Programação III
Prof. Valdir Pedrinho de Tomin Junior, Eng.

14 de agosto de 2018

## Exercícios do Livro

Resolver os seguintes problemas do capítulo 9 do livro (Deitel, H. M.; Deitel, P. J., **C++ Como Programar**, 5ª ed., Pearson, 2015.):

- 9.5
- 9.6
- 9.11 ao 9.14

**Dicas:**

1. Não é necessário resolver todos os exercícios, mas certifique-se de resolver um número suficiente de exercícios, de cada assunto, para consolidar seus conhecimentos;
2. Se for o caso, procure mais exercícios de determinado assunto; ou
3. Peça ajuda ao professor.

**Observação:**

As páginas do livro com os problemas propostos estão anexadas.

## Exercícios de revisão

**9.1** Preencha as lacunas em cada uma das seguintes sentenças:

a) Os membros de classe são acessados via operador _____ em conjunto com o nome de um objeto (ou referência a um objeto) da classe ou via operador _____ em conjunto com um ponteiro para um objeto da classe.

b) Os membros de classe especificados como _____ são acessíveis apenas às funções-membro da classe e `friend`s da classe.

c) Os membros de classe especificados como _____ são acessíveis em qualquer lugar que um objeto da classe esteja no escopo.

d) A _____ pode ser utilizada para atribuir um objeto de uma classe a outro objeto da mesma classe.

**9.2** Localize o(s) erro(s) em cada uma das seguintes seqüências e explique como corrigi-lo(s):

a) Suponha que o seguinte protótipo é declarado na classe `Time`:

```
void ~Time( int );
```

b) A seguinte definição é uma definição parcial da classe `Time`:

```
class Time
{
public:
// protótipos de função
private:
   int hour = 0;
   int minute = 0;
   int second = 0;
}; // fim da classe Time
```

c) Suponha que o seguinte protótipo é declarado na classe `Employee`:

```
int Employee( const char *, const char * );
```

## Respostas dos exercícios de revisão

**9.1** a) ponto (`.`), seta (`->`). b) `private`. c) `public`. d) atribuição-padrão de membro a membro (realizada pelo operador de atribuição).

**9.2** a) Erro: Os destrutores não têm permissão de retornar valores (nem mesmo de especificar um tipo de retorno) nem de aceitar argumentos.
Correção: Remova o tipo de retorno `void` e o parâmetro `int` da declaração.

b) Erro: Os membros não podem ser explicitamente inicializados na definição de classe.
Correção: Remova a inicialização explícita da definição de classe e inicialize os membros de dados em um construtor.

c) Erro: Os construtores não têm permissão de retornar valores.
Correção: Remova o tipo de retorno `int` da declaração.

## Exercícios

**9.3** Qual é o propósito do operador de resolução de escopo?

**9.4** (*Aprimorando a classe `Time`*) Forneça um construtor que seja capaz de utilizar a hora atual da função `time()` — declarada no cabeçalho `<ctime>` da C++ Standard Library — para inicializar um objeto da classe `Time`.

**9.5** (*Classe `Complex`*) Crie uma classe chamada `Complex` para realizar aritmética com números complexos. Escreva um programa para testar sua classe.

Os números complexos têm a forma

```
parteReal + parteImaginária * i
```

onde *i* é

$$\sqrt{-1}$$

Utilize as variáveis `double` para representar os dados `private` da classe. Forneça um construtor que permita que um objeto dessa classe seja inicializado quando ele for declarado. O construtor deve conter valores-padrão no caso de nenhum inicializador ser fornecido. Forneça as funções-membro `public` que realizam as tarefas a seguir:

a) Somar dois números `Complex`: as partes reais são somadas de um lado e as partes imaginárias são somadas de outro.

b) Subtrair dois números `Complex`: a parte real do operando direito é subtraída da parte real do operando esquerdo e a parte imaginária do operando direito é subtraída da parte imaginária do operando esquerdo.

c) Imprimir os números `Complex` na forma `(a, b)`, onde `a` é a parte real e `b` é a parte imaginária.

**9.6** (*Classe `Rational`*) Crie uma classe chamada `Rational` para realizar aritmética com frações. Escreva um programa para testar sua classe.

Utilize variáveis do tipo inteiro para representar os dados `private` da classe — o `numerador` e o `denominador`. Forneça um construtor que permita que um objeto dessa classe seja inicializado quando ele for declarado. O construtor deve conter valores-padrão no caso de nenhum inicializador ser fornecido e deve armazenar a fração na forma reduzida. Por exemplo, a fração

$$\frac{2}{4}$$

seria armazenada no objeto como 1 no `numerator` e 2 no `denominator`. Forneça funções-membro `public` que realizam cada uma das tarefas a seguir:

a) Somar dois números `Rational`. O resultado deve ser armazenado na forma reduzida.
b) Subtrair dois números `Rational`. O resultado deve ser armazenado na forma reduzida.
c) Multiplicar dois números `Rational`. O resultado deve ser armazenado na forma reduzida.
d) Dividir dois números `Rational`. O resultado deve ser armazenado na forma reduzida.
e) Imprimir os números `Rational` na forma `a/b`, onde `a` é o numerador e `b` é o denominador.
f) Imprimir os números `Rational` em formato de ponto flutuante.

**9.7** (*Aprimorando a classe `Time`*) Modifique a classe `Time` das figuras 9.8–9.9 para incluir uma função-membro `tick` que incrementa a hora armazenada em um objeto `Time` por um segundo. O objeto `Time` sempre deve permanecer em um estado consistente. Escreva um programa que testa a função-membro `tick` em um loop que imprime a hora no formato-padrão durante cada iteração do loop para ilustrar que a função-membro `tick` funciona corretamente. Certifique-se de testar os seguintes casos:

a) Incrementar para o próximo minuto.
b) Incrementar para a próxima hora.
c) Incrementar para o próximo dia (isto é, 11:59:59 PM para 12:00:00 AM).

**9.8** (*Aprimorando a classe `Date`*) Modifique a classe `Date` das figuras 9.17–9.18 para realizar a verificação de erros nos valores inicializadores para membros de dados `month`, `day` e `year`. Além disso, forneça uma função-membro `nextDay` para incrementar o dia por um. O objeto `Date` sempre deve permanecer em um estado consistente. Escreva um programa que testa a função `nextDay` em um loop que imprime a data durante cada iteração para ilustrar que `nextDay` funciona corretamente. Certifique-se de testar os seguintes casos:

a) Incrementar para o próximo mês.
b) Incrementar para o próximo ano.

**9.9** (*Combinando a classe `Time` e a classe `Date`*) Combine a classe `Time` modificada do Exercício 9.7 e a classe `Date` modificada do Exercício 9.8 em uma classe chamada `DateAndTime`. (No Capítulo 12, discutiremos a herança, que permitirá realizar essa tarefa rapidamente sem modificar as definições de classe existentes.) Modifique a função `tick` para chamar a função `nextDay` se a hora for incrementada para o dia seguinte. Modifique as funções `printStandard` e `printUniversal` para gerar saída da data e da hora. Escreva um programa para testar a nova classe `DateAndTime`. Especificamente, teste incrementar a hora para o dia seguinte.

**9.10** (*Retornando indicadores de erros das funções set da classe `Time`*) Modifique as funções *set* na classe `Time` das figuras 9.8–9.9 para retornar valores de erros apropriados se uma tentativa de *configurar* um membro de dados de um objeto da classe `Time` para um valor inválido for feita. Escreva um programa que testa sua nova versão da classe `Time`. Exiba mensagens de erro quando funções *set* retornarem valores de erros.

**9.11** (*Classe `Rectangle`*) Crie uma classe `Rectangle` com atributos `length` e `width`, cada um dos quais assume o padrão de 1. Forneça funções-membro que calculam os atributos `perimeter` e `area` do retângulo. Além disso, forneça as funções *set* e *get* para os atributos `length` e `width`. As funções *set* devem verificar se `length` e `width` são números de ponto flutuante maiores que 0,0 e menores que 20,0.

**9.12** (*Classe `Rectangle` aprimorada*) Crie uma classe `Rectangle` mais sofisticada do que aquela que você criou no Exercício 9.11. Essa classe armazena somente as coordenadas cartesianas dos quatro vértices do retângulo. O construtor chama uma função *set* que aceita quatro conjuntos de coordenadas e verifica se cada um deles está no primeiro quadrante sem coordenadas *x* ou *y* individualmente maiores que 20,0. A função *set* também verifica se as coordenadas fornecidas especificam de fato um retângulo. Forneça funções-membro que calculam `length`, `width`, `perimeter` e `area`. O comprimento é o maior das duas dimensões. Inclua uma função predicado `square` que determina se o retângulo é um quadrado.

**9.13** (*Aprimorando a classe `Rectangle`*) Modifique a classe `Rectangle` do Exercício 9.12 para incluir uma função `draw` que exibe o retângulo dentro de uma caixa de 25 por 25 para incluir a parte do primeiro quadrante em que o retângulo reside. Inclua uma função `setFillCharacter` para especificar o caractere a partir do qual o corpo do retângulo será desenhado. Inclua uma função `setPerimeterCharacter` para especificar o caractere que será utilizado para desenhar a borda do retângulo. Se você se sentir ambicioso, talvez queira incluir funções para dimensionar o tamanho do retângulo, rotacioná-lo e movê-lo dentro da parte designada do primeiro quadrante.

**9.14** (*Classe `HugeInteger`*) Crie uma classe `HugeInteger` que utiliza um array de 40 elementos de dígitos para armazenar inteiros com até 40 dígitos cada. Forneça as funções-membro `input`, `output`, `add` e `substract`. Para comparar objetos `HugeInteger`, forneça funções `isEqualTo`, `isNotEqualTo`, `isGreaterThan`, `isLessThan`, `isGreaterThanOrEqualTo` e `isLessThanOrEqualTo` — cada uma

dessas é uma função 'predicado' que simplesmente retorna `true` se o relacionamento se mantém entre os dois `HugeIntegers`, e retorna `false` se o relacionamento não se mantém. Além disso, forneça uma função predicado `isZero`. Se estiver motivado, forneça as funções-membro `multiply`, `divide` e `modulus`.

**9.15** (*Classe `TicTacToe`*) Crie uma classe `TicTacToe` que permitirá escrever um programa completo para jogar o jogo-da-velha (Tic-Tac-Toe). A classe contém como dados `private` um array bidimensional 3 por 3 de inteiros. O construtor deve inicializar a grade vazia com todos como zero. Permita dois jogadores humanos. Para onde quer que o primeiro jogador se mova, coloque um 1 no quadrado especificado. Coloque 2 para onde quer que o segundo jogador se mova. Todo movimento deve ocorrer em um quadrado vazio. Depois de cada movimento, determine se houve uma derrota ou um empate. Se você se sentir motivado, modifique seu programa de modo que o computador faça o movimento para um dos jogadores. Além disso, permita que o jogador especifique se quer ser o primeiro ou o segundo. Se você se sentir excepcionalmente motivado, desenvolva um programa que jogue um jogo-da-velha tridimensional em uma grade 4 por 4 por 4. [*Atenção:* Esse é um projeto extremamente desafiador que pode exigir muitas semanas de esforço!]