

## Operações em Ponto Flutuante no C#

Mário Leite

...

Quando são feitas operações em ponto flutuante (valores com casas decimais) é muito importante saber, a priori, qual é a precisão desejada e quais os valores extremos permitidos. Deste modo, o tipo de dado escolhido é fundamental para que os resultados saiam corretos e com a precisão esperada. Quando se trabalha com números e é necessário fazer cálculos que envolvam médias, ou qualquer tipo de operação em ponto flutuante, o mais indicado é considerar esses números e os resultados esperados nesses cálculos como valores *reais* (*float*, *double*, *decimal*); isto evita conversões desnecessárias e resultados inesperados; lembrando que a divisão entre dois inteiros sempre resulta em um resultado inteiro. Assim, a menos que se tenha certeza de que todos os cálculos serão inteiros, o mais seguro é declarar adequadamente as variáveis numéricas que entrarão nesses cálculos; mas o fator “precisão” também tem que ser observado. A **tabela 1** mostra os três tipos básicos de dados em ponto flutuante com as respectivas precisões, na linguagem C#.

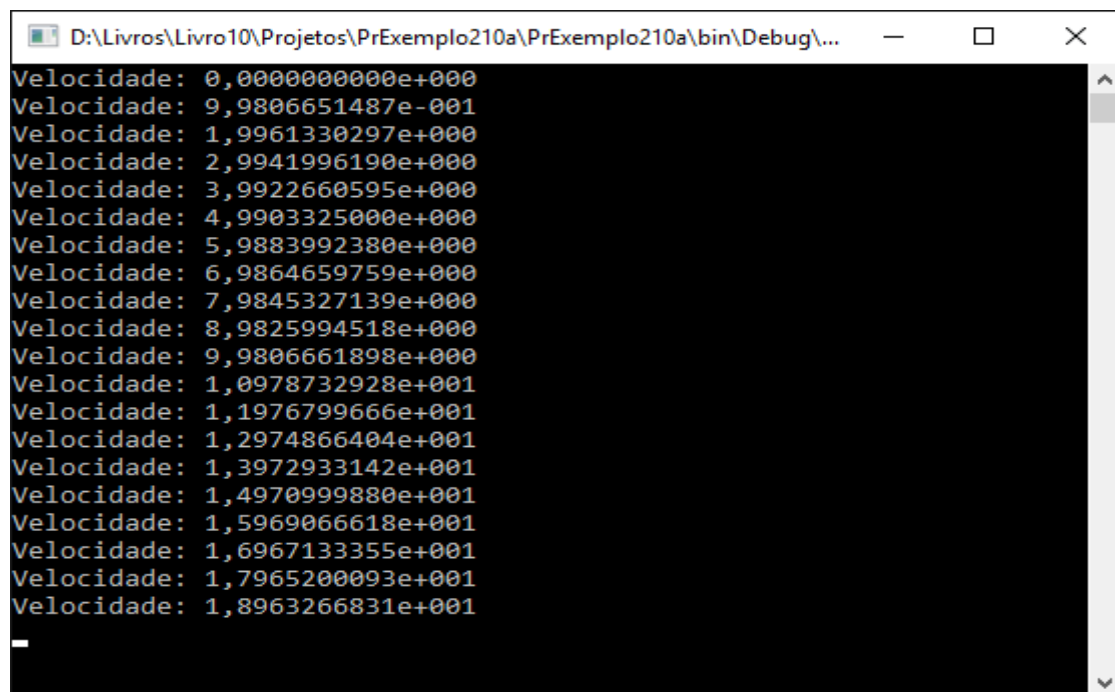
Tipo	Tamanho	Faixa	Precisão	Maior inteiro exato
<b>float</b>	4 <i>bytes</i>	$1.5 \times 10^{-45}$ a $3.4 \times 10^{38}$	7 a 8 dígitos	$2^{24}$
<b>double</b>	8 <i>bytes</i>	$5.0 \times 10^{-324}$ a $1.7 \times 10^{308}$	15 a 16 dígitos	$2^{53}$
<b>decimal</b>	16 <i>bytes</i>	$1.0 \times 10^{-28}$ a $7.9 \times 10^{28}$	28 a 29 dígitos	$2^{113}$

**Tabela 1 - Os três tipos de dados em ponto flutuante em C#**

Qualquer um dos três tipos de dados reais apresentados na **tabela 1** pode ser empregado em cálculos que envolvam valores com casas decimais; entretanto, o tipo *decimal* é o mais indicado (talvez até obrigatório) em operações financeiras devido à sua alta precisão quando comparado com os outros dois tipos flutuantes. Por exemplo, sabemos da Física Clássica que um corpo abandonado de uma certa distância do solo aumenta, aproximadamente, a sua velocidade de queda em **9.80665 m/s** a cada segundo; assim, podemos criar uma aplicação simples para mostrar a evolução da sua velocidade instantânea em queda livre a cada intervalo de **0.1** segundo.

O programa “**PrExercício210a**” abaixo é uma solução para o exemplo citado. A **figura 1** mostra a saída do programa quando usamos variável do tipo *float* para o tempo, e a **figura 2** quando são usadas variáveis do tipo *decimal*

```
namespace PrExercício21a
{
    internal class Program
    {
        public const double G = 9.980665; //declara a constante da gravidade
        static void Main(string[] args)
        {
            double V = 0.0;
            for (float t=0.1F; t <=2.0F; t += 0.1F)
            {
                V = (double) 9.980665*t; //expressão da velocidade em queda livre
                Console.WriteLine("Velocidade: {0}", V.ToString("e10"));
            }
            Console.ReadKey();
        }
    }
}
```



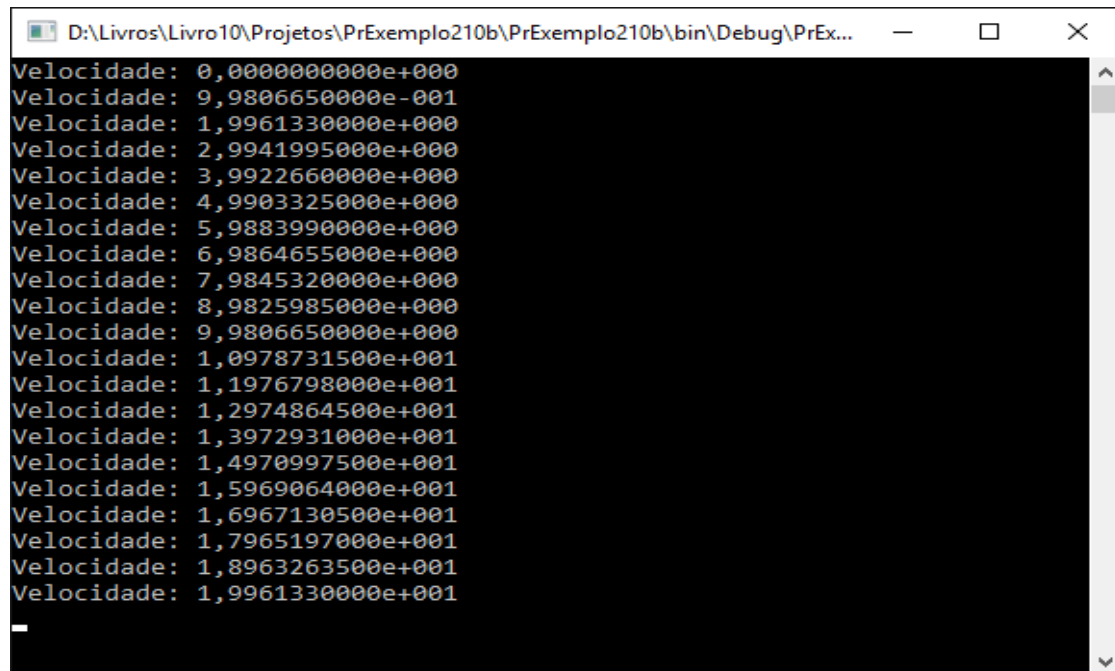
```
D:\Livros\Livro10\Projetos\PrExemplo210a\PrExemplo210a\bin\Debug\...
Velocidade: 0,000000000e+000
Velocidade: 9,9806651487e-001
Velocidade: 1,9961330297e+000
Velocidade: 2,9941996190e+000
Velocidade: 3,9922660595e+000
Velocidade: 4,9903325000e+000
Velocidade: 5,9883992380e+000
Velocidade: 6,9864659759e+000
Velocidade: 7,9845327139e+000
Velocidade: 8,9825994518e+000
Velocidade: 9,9806661898e+000
Velocidade: 1,0978732928e+001
Velocidade: 1,1976799666e+001
Velocidade: 1,2974866404e+001
Velocidade: 1,3972933142e+001
Velocidade: 1,4970999880e+001
Velocidade: 1,5969066618e+001
Velocidade: 1,6967133355e+001
Velocidade: 1,7965200093e+001
Velocidade: 1,8963266831e+001
```

Figura 1 - Saída da aplicação do programa “PrExercício210a”: com e em tipo *float*

Agora, observe na **figura 2** que de  $t=0.0$  até  $t=2.0$  deveria ser mostradas **21** iterações: uma a cada intervalo de tempo; entretanto, foram mostradas apenas **20**. Este erro no número de iterações foi devido à falha de precisão ao utilizar valores *float* (para a variável  $t$ ) no *loop* que calcula as velocidades instantâneas. Agora, observe quando utilizamos valores no tipo *decimal* na versão “PrExercício210b” do programa.

```
namespace PrExercício210b
{
    internal class Program
    {
        public const decimal G=9.980665m; //constante da gravidade
        static void Main(string[] args)
        {
            decimal V = 0.0m;
            for (decimal t=0.1m; t <=2.0m; t += 0.1m)
            {
                V = (decimal) G*t; //expressão da velocidade em queda livre
                Console.WriteLine("Velocidade: {0}", V.ToString("e10"));
            }
            Console.ReadKey();
        }
    }
}
```

A **figura 2** mostra o resultado correto: **21** iterações. Isto foi possível devido ao fato de termos utilizado valores numéricos do tipo *decimal* (para **t**) nas operações e observando, também, a compatibilidade na utilização desses valores.



```
D:\Livros\Livro10\Projetos\PrExemplo210b\PrExemplo210b\bin\Debug\PrEx...
Velocidade: 0,0000000000e+000
Velocidade: 9,9806650000e-001
Velocidade: 1,9961330000e+000
Velocidade: 2,9941995000e+000
Velocidade: 3,9922660000e+000
Velocidade: 4,9903325000e+000
Velocidade: 5,9883990000e+000
Velocidade: 6,9864650000e+000
Velocidade: 7,9845320000e+000
Velocidade: 8,9825985000e+000
Velocidade: 9,9806650000e+000
Velocidade: 1,0978731500e+001
Velocidade: 1,1976798000e+001
Velocidade: 1,2974864500e+001
Velocidade: 1,3972931000e+001
Velocidade: 1,4970997500e+001
Velocidade: 1,5969064000e+001
Velocidade: 1,6967130500e+001
Velocidade: 1,7965197000e+001
Velocidade: 1,8963263500e+001
Velocidade: 1,9961330000e+001
```

**Figura 2.** - Saída da aplicação do programa “PrExercício210b”: com **t** em tipo *decimal*

Por outro lado, é importante notar que se a velocidade **V** fosse declarada como *double* na expressão **V = (decimal)G\*t** seria gerado um erro de incompatibilidade; o sistema responderia com a seguinte mensagem de erro:

*Não é possível converter implicitamente “decimal” em “double”*

E se a conversão (decimal) fosse retirada e ficasse apenas **V = g\*t**, mesmo assim outro erro seria gerado devido ao operador **\***, e a mensagem de erro seria a seguinte:

*O operador “\*” não pode ser aplicado a operadores do tipo “double” e “decimal”*

Então, devemos ter bastante cuidado ao trabalhar com tipos numéricos em expressões que podem resultar em valores com casas decimais. Em transações financeiras erros deste tipo podem ser fatais, já que nesses casos estaria sendo considerada a possibilidade de um prejuízo real se transformar num falso lucro (ou vice-versa). Portanto, todo cuidado é pouco quando estivermos trabalhando com valores em ponto flutuante: EM QUALQUER LINGUAGEM!

---

**Nota:** Postagem baseada no livro: “*Linguagem C#: Com acessos a Bancos de Dados*” publicado pelo autor na “Amazon” e no “Clube de Autores”.

<https://www.amazon.com.br/Linguagem-Com-Acesso-Bancos-Dados-ebook/dp/B0BRBKNV4N>

---