

Usando vetores com o .NET Framework e C#

por Fabio Galuppo

Arrays, ou simplesmente vetores, são elementos dimensionais de um determinado tipo de dados, sendo eles tipo referência ou tipo valor, onde são agrupados num número fixo de elementos. Em C# os vetores são acessados através de um índice (valor numérico) cujo sempre inicia-se com zero (0).

Um vetor pode ser criado na forma unidimensional, multidimensional ou *jagged*, em C#. Um vetor é um tipo referência. Ele indica que a variável, a qual representa o vetor, aponta para os elementos em memória, bem como quando passada através de métodos, seus elementos podem ser alterados.

Os vetores de tamanho fixo são baseados na classe *System.Array* do .NET Framework.

Declarar, criar, armazenar e acessar um vetor

Na declaração de um *array* (vetor) selecionamos o tipo de dado seguido de colchetes ([]). Após, coloca-se a variável.

```
string[] Nomes; //array declarado, mas não criado ou inicializado
```

Antes do armazenamento dos valores, o operador *new* é obrigatório para a criação do número de elementos desejado. O mesmo aparece entre conchetes ([]) após o tipo determinado.

```
Nomes = new string[10]; //array criado com 10 elementos
```

Para armazenar um valor num array use o índice onde para o primeiro elemento do vetor é zero (0) e o último elemento é o tamanho total do vetor menos um (1). Depois é colocado o sinal de igual com o valor (constante ou uma variável) desejado a guardar naquela posição.

```
Nomes[0] = "Fabio Galuppo"; //armazenando uma string no primeiro elemento  
Nomes[1] = "Vanclei Matheus"; //armazenando uma string no segundo elemento  
Nomes[9] = "Wallace Santos"; //armazenando uma string no último elemento
```

Para acessar um valor do array basta usar o índice juntamente a variável que representa o vetor, como indicado anteriormente.

```
string nome = Nomes[0]; //Fabio Galuppo
```

Outro ponto interessante no ambiente do .NET Framework é as instâncias serem apenas criadas, ou seja, não existe um operador delete para eliminar a memória consumida. O trabalho é feito pelo coletor de lixo (*garbage collector*) o qual se responsabiliza pela liberação dos recursos gerenciados que foram consumido. Portanto, como os vetores possuem um tamanho fixo determinado na sua criação, eles não poderão ser redimensionados. Se um nova instrução *new* com um número maior de elementos for indicada num vetor já existente, isto não quer dizer que o vetor antigo será redimensionado, de fato ocorrerá que um novo vetor vai sobrescrever o antigo. A coleta de lixo libera o vetor antigo da memória desconsiderando seus valores no novo vetor.

```
Nomes = new string(10);  
  
Nomes[0] = "Fabio Galuppo"; //armazenando uma string no primeiro elemento  
Nomes[1] = "Vanclei Matheus"; //armazenando uma string no segundo elemento
```

Usando vetores com o .NET Framework e C#

por Fabio Galuppo

```
Nomes[9] = "Wallace Santos"; //armazenando uma string no último elemento  
Nomes = new string(20); //novo vetor, o antigo foi liberado e sobrescrito  
string nome = Nomes[0]; //Nome[0] é null
```

Se o vetor acessar um índice não existente, a exceção *IndexOutOfRangeException* será disparada.

```
string novonome = Nomes[21]; //Índice não existente  
(IndexOutOfRangeException)
```

Tipos de vetor

Existem três (3) tipos de vetores em C#. Eles são alinhados linearmente na memória, mas quanto as suas dimensões eles podem ser:

- Unidimensionais - uma (1) dimensão para acessar através de um índice;
- Multidimensionais - duas (2) ou mais dimensões para acessar através de uma combinação de índices. Eles são conhecidos de matrizes, principalmente por aparecerem na sua forma mais comum como um array de duas dimensões. São retangulares quanto suas dimensões;
- *Jagged* - array de arrays. São multidimensionais, mas não possuem a características retangulares de uma matriz, onde numa dimensão o número de elementos contidos poderão ser diferentes.

Para declarar um array multidimensional, dentro dos colchetes ([]) separe por vírgulas (,) o número de dimensões. Na criação passe os valores para cada uma das dimensões.

```
int[,] MatrizBidimensional; //array multidimensional declarado para 2  
dimensões  
int[, ,] MatrizTridimensional; //array multidimensional declarado para 3  
dimensões
```

```
MatrizBidimensional = new int[3,3]; //criado com 9 elementos.  
MatrizBidimensional = new int[3,4,5]; //criado para 60 elementos.
```

```
MatrizBidimensional[0,0] = 50; //armazenando na matriz bidimensional  
MatrizBidimensional[2,1] = 100;  
MatrizBidimensional[2,2] = 200;
```

```
MatrizTridimensional[1,2,4] = 250; //armazenando na matriz tridimensional  
MatrizTridimensional[4,3,2] = 500;  
MatrizTridimensional[4,4,4] = 1000;
```

Para declarar um *array jagged* coloque o número de colchetes ([]) referente ao número de dimensões. Crie cada uma das dimensões independentemente. Passe os valores para cada uma delas.

```
int[][] JaggedBidimensional; //array jagged declarado para 2 dimensões  
int[][][] JaggedTridimensional; //array jagged declarado para 3 dimensões
```

Usando vetores com o .NET Framework e C#

por Fabio Galuppo

```
JaggedBidimensional = new int[2][]; //2 linhas na primeira dimensão
JaggedBidimensional[0] = new int[3]; //na primeira linha, 3 colunas
criadas
JaggedBidimensional[1] = new int[2]; //na segunda linha, 2 colunas
criadas

JaggedTridimensional = new int[1][][]; //3 linhas na primeira dimensão
JaggedTridimensional[0] = new int[2]; //primeira linha, 2 elementos
criadas
JaggedTridimensional[0][0] = new int[1]; //linha 1, coluna 1, 1 elemento
criado
JaggedTridimensional[0][1] = new int[2]; //linha 1, coluna 2, 2 elementos
criados
```

```
JaggedBidimensional[0][1] = 100;
JaggedBidimensional[0][2] = 400;

JaggedTridimensional[0][1][1] = 700;
JaggedTridimensional[0][0][0] = 900;
```

Os *arrays* retangulares e *jagged* podem ser misturados, por exemplo:

```
//jagged de 2 dimensões
double[,][,] Misturado = new double[2][,][,];
//linha 1 - criada com uma matriz 3x3
Misturado[0] = new double[3,3][,];
//linha 2 - criada com uma matriz 1x1
Misturado[1] = new double[1,1][,];
//linha 1, matriz(1,1) - criada com um vetor de 2 elementos
Misturado[0][0,0] = new double[2][,];
//linha 1, matriz(1,1), linha 2 - criada com uma matriz de 2x1x3
Misturado[0][0,0][1] = new double [2,1,3];
//e assim por diante...
```

Cada tipo de vetor possui uma representação lógica diferente na memória. Na figura 1, temos a representação dos vetores abaixo:

```
long[] ar1 = new long[5]; //vetor unidimensional
```

```
long[,] ar2 = new long[3,2]; //vetor multidimensional ou matriz
```

```
long[][] ar3 = new long[2][]; //vetor jagged
ar3[0] = new long[2];
ar3[1] = new long[4];
```

Usando vetores com o .NET Framework e C#

por Fabio Galuppo

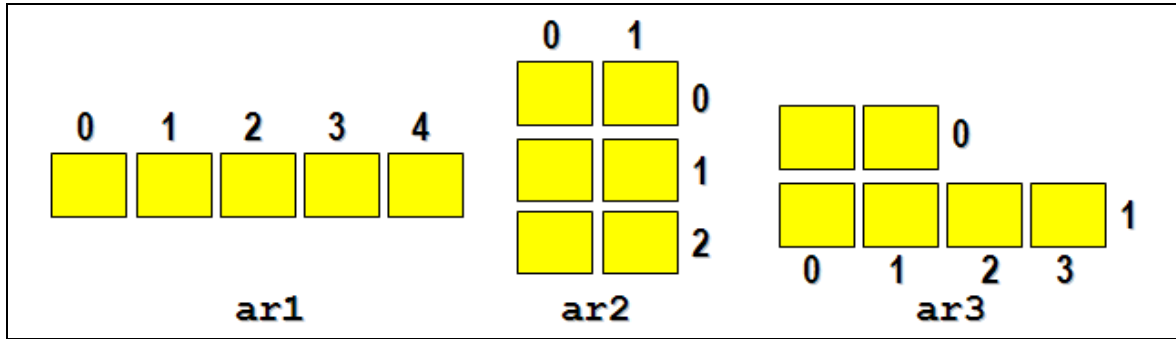


Figura 1: Representação dos vetores unidimensional, multidimensional e *jagged*

Inicializando os vetores

Os vetores podem ser inicializados durante a declaração. Os valores de inicialização são colocados entre chaves. É normal e permitido neste caso a omissão do tamanho do *array* e do operador *new*.

```
//Array unidimensional
double[] ar1 = new double[4]{1.1, 2.2, 3.3, 4.4};
double ar1 = new double[]{1.1, 2.2, 3.3, 4.4};
double ar1 = {1.1, 2.2, 3.3, 4.4};
```

```
//Array multidimensional
int[,] ar2 = new int[2,3]{ {1,2,3}, {4,5,6} };
int[,] ar2 = new int[,]{ {1,2,3}, {4,5,6} };
int[,] ar2 = { {1,2,3}, {4,5,6} };
```

```
//Array Jagged
byte[][] ar3 = new byte[2][]{new byte[2]{0xE,0xF}, new
byte[4]{0xA,0xB,0xC,0xD}};
byte[][] ar3 = new byte[][]{new byte[]{0xE,0xF}, new
byte[]{0xA,0xB,0xC,0xD}};
byte[][] ar3 = {new byte[2]{0xE,0xF}, new byte[4]{0xA,0xB,0xC,0xD}};
```

```
//Array Jagged
byte[][] ar3 = new byte[2][];
ar3[0] = new byte[2]{0xE, 0xF};
ar3[1] = new byte[4]{0xA, 0xB, 0xC, 0xD};
```

Percorrendo os vetores

Os vetores podem ser percorridos de diversas formas. Por exemplo, usando comandos de iteração como *for* e *while*. Eles são acessados por seus índices. Há duas possibilidades de obter-se o tamanho do *array*. A primeira é guardá-lo numa variável durante sua criação. Já a outra forma, mais eficiente, é utilizar a propriedade *Length* ou o método *GetLength* para obter o total de elementos dele.

Usando vetores com o .NET Framework e C#

por Fabio Galuppo

```
//csc Vetores.cs

using System;

public class Unidimensional
{
    string[] s = {"um", "dois", "três", "quatro"};

    public void Processar()
    {
        Console.WriteLine("\nUnidimensional = ");
        for(int idx = 0, l = s.Length; idx < l; ++idx)
            Console.WriteLine("{0} ", s[idx]);
    }
}

public class Multidimensional
{
    string[,] s = { {"um", "dois"}, {"três", "quatro"} };

    public void Processar()
    {
        Console.WriteLine("\nMultidimensional = ");
        for(int idx = 0, l = s.GetLength(0); idx < l; ++idx)
            for(int idx2 = 0, l2 = s.GetLength(1); idx2 < l2; ++idx2)
                Console.WriteLine("{0} ", s[idx, idx2]);
    }
}

public class Jagged
{
    string[][] s = { new string[]{"um"}, new string[]{"dois", "três", "quatro"} };

    public void Processar()
    {
        Console.WriteLine("\nJagged = ");
        for(int idx = 0, l = s.GetLength(0); idx < l; ++idx)
            for(int idx2 = 0, l2 = s[idx].GetLength(0); idx2 < l2; ++idx2)
                Console.WriteLine("{0} ", s[idx][idx2]);
    }
}

public class Vetores
{
    public static void Main()
    {
        Unidimensional uni = new Unidimensional();
        Multidimensional multi = new Multidimensional();
        Jagged jag = new Jagged();

        uni.Processar();
        multi.Processar();
        jag.Processar();
    }
}
```

Nos casos acima a propriedade *Length* aplicada ao vetor unidimensional retorna a quantidade de elementos dele. O método *GetLength(0)* aplicado ao vetor multidimensional retorna a quantidade de elementos da primeira dimensão, enquanto o método *GetLength(1)* retorna a quantidade de

Usando vetores com o .NET Framework e C#

por Fabio Galuppo

elementos da segunda dimensão. O método *GetLength(0)* aplicado ao vetor jagged retorna ao número de *arrays* dentro do *array*. As outras quantidades devem ser acessadas através de *GetLength(0)* da dimensão desejada. Por exemplo, *s[0].GetLength(0)* retorna o número de elementos contido na primeira dimensão do *array*.

Para percorrer um vetor o comando *foreach* pode ser utilizado, principalmente porque um *array* possui o método *GetEnumerator* que retorna a interface *IEnumerator*. Usando o comando *foreach* para iterar um vetor o resultado é disposição sequencial dele, ou seja, o resultado final é a disposição dos elementos na memória. Os *jagged arrays* não conseguem ser acessado somente pelo comando *foreach* devido suas dimensões variáveis.

```
//csc Vetores.cs

using System;

public class Unidimensional
{
    string[] s = {"um", "dois", "três", "quatro"};

    public void Processar()
    {
        Console.WriteLine("\nUnidimensional = ");
        foreach(string e in s)
            Console.WriteLine("{0} ", e);
    }
}

public class Multidimensional
{
    string[,] s = { {"um", "dois"}, {"três", "quatro"} };

    public void Processar()
    {
        Console.WriteLine("\nMultidimensional = ");
        foreach(string e in s)
            Console.WriteLine("{0} ", e);
    }
}

public class Jagged
{
    string[][] s = { new string[]{"um"}, new string[]{"dois", "três", "quatro"} };

    public void Processar()
    {
        Console.WriteLine("\nJagged = ");
        for(int idx = 0, l = s.GetLength(0); idx < l; ++idx)
            foreach(string e in s[idx])
                Console.WriteLine("{0} ", e);
    }
}

public class Vetores
{
    public static void Main()
    {
        Unidimensional uni = new Unidimensional();
        Multidimensional multi = new Multidimensional();
        Jagged jag = new Jagged();
    }
}
```

Usando vetores com o .NET Framework e C#

por Fabio Galuppo

```
        uni.Processar();  
        multi.Processar();  
        jag.Processar();  
    }  
}
```

A classe *System.Array*

Os vetores são baseados na classe *System.Array* do .NET Framework. É possível usar as propriedades e os membros da classe para executar operações diversas. Os principais membros estão relacionados abaixo:

Length retorna a quantidade total de elementos em todas as dimensões.

```
int tamanho = elems.Length; //6
```

Rank retorna o *rank* ou o número de dimensões do *array*.

```
int dimensoes = elems.Rank; //1
```

BinarySearch faz uma busca usando o algoritmo de binary search apenas numa dimensão.

```
Array.BinarySearch(elems, 'F'); //2
```

GetLength retorna a quantidade de elementos de uma dimensão especificada.

```
int dim1 = elems.GetLength(0); //6
```

GetLowerBound retorna o limite inferior do vetor. Em C# é sempre zero (0).

```
int inferior = elems.GetLowerBound(0); //0
```

GetUpperBound retorna o limite superior do vetor.

```
int superior = elems.GetUpperBound(0); //5
```

GetValue obtém o valor de um elemento determinado do vetor.

```
char ch = (char)elems.GetValue(1); //A
```

SetValue define um valor para um elemento determinado do vetor.

```
elems.SetValue('C', 5);
```

IndexOf retorna o índice da primeira ocorrência de um valor numa dimensão do vetor.

```
Array.IndexOf(elems, 'A'); //0
```

LastIndexOf retorna o índice da última ocorrência de um valor numa dimensão do vetor.

Usando vetores com o .NET Framework e C#

por Fabio Galuppo

```
Array.LastIndexOf(elems, 'A'); //5
```

Copy e *CopyTo* efetuam cópias de um *array* para outro.

```
elems.CopyTo(elems2, 0);
```

```
Array.Copy(elems, elems3, elems.Length);
```

Sort organiza um vetor na ordem crescente.

```
Array.Sort(elems2); //A B C D E F
```

Reverse inverte a ordem de um vetor.

```
Array.Reverse(elems3); //C D E F A B
```

Clear limpa os elementos do *array* com zero (0), falso (*false*) ou nulo (*null*), dependendo do tipo de dados. Os parâmetros especificados são o vetor, o índice inicial e a quantidade de elementos a serem limpados.

```
Array.Clear(elems2, 0, elems2.Length); //Todos elementos
```

```
Array.Clear(elems2, 1, 2); //2 elementos a partir da 2ª posição
```

O exemplo a seguir usa todos os métodos discutidos anteriormente:

```
//csc Vetores.cs

using System;

public class Vetores
{
    public static void Main()
    {
        //Criação
        char[] elems = new char[6];

        //Atribuição
        elems[1] = 'A';           //A
        elems[0] = (char)0x42;    //B
        elems[5] = "C# is COOL"[0]; //C
        elems[4] = 'D';           //D
        elems[3] = (char)69;       //E
        elems[2] = "EFI"[1];       //F

        //Percorrer
        foreach(char ch in elems) Console.WriteLine(ch);

        //Usando os membros do System.Array
        Console.WriteLine("Tamanho do array = {0}", elems.Length);

        Console.WriteLine("Número de dimensões = {0}", elems.Rank);
    }
}
```


Usando vetores com o .NET Framework e C#

por Fabio Galuppo

```
Console.WriteLine("Índice do 'F' = {0}", Array.BinarySearch(elems, 'F'));

int dim1 = elems.GetLength(0);
Console.WriteLine("Número de elementos na 1ª dimensão = {0}", dim1);

Console.WriteLine("Lower Bound = {0}", elems.GetLowerBound(0));

Console.WriteLine("Upper Bound = {0}", elems.GetUpperBound(0));

Console.WriteLine("Valor na posição 2 = {0}", elems.GetValue(1));

elems[5] = 'A';

//Primeira ocorrência
Console.WriteLine("Índice de 'A' = {0}", Array.IndexOf(elems, 'A'));

//Última ocorrência
Console.WriteLine("Índice de 'A' = {0}", Array.LastIndexOf(elems, 'A'));

elems.SetValue('C', 5);

char[] elems2 = new char[6];
elems.CopyTo(elems2, 0);

char[] elems3 = new char[6];
Array.Copy(elems, elems3, elems.Length);

Array.Sort(elems2);
Console.WriteLine("Ordem Crescente");
foreach(char ch in elems2) Console.Write(ch);
Console.WriteLine("\nOrdem Decrescente");
for(int a = elems2.Length - 1; a > -1; --a) Console.Write(elems2[a]);

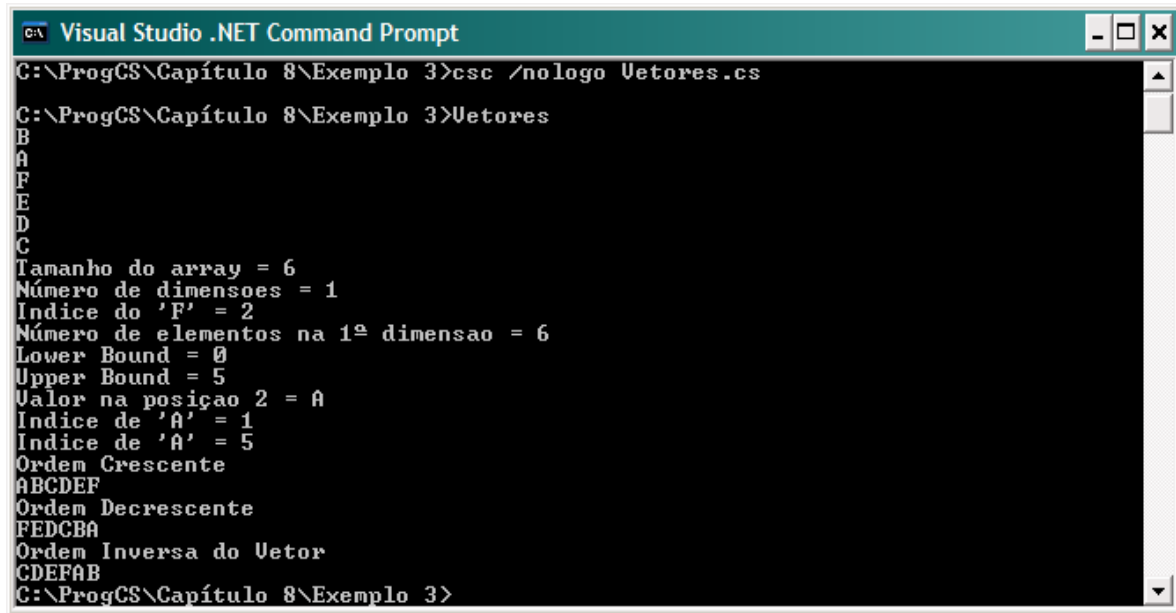
Console.WriteLine("\nOrdem Inversa do Vetor");
Array.Reverse(elems3);
foreach(char ch in elems3) Console.Write(ch);

Array.Clear(elems2, 0, elems2.Length);
Array.Clear(elems3, 0, elems3.Length);
}
}
```

Para compilar o exemplo acima no *Command Prompt* digite *csc /nologo Vetores.cs*. Execute o programa digitando *Vetores*. A figura 2 mostra a compilação e execução da aplicação em C#.

Usando vetores com o .NET Framework e C#

por Fabio Galuppo



```
C:\> Visual Studio .NET Command Prompt
C:\ProgCS\Capítulo 8\Exemplo 3>csc /nologo Vetores.cs
C:\ProgCS\Capítulo 8\Exemplo 3>Vetores
B
A
F
E
D
C
Tamanho do array = 6
Número de dimensões = 1
Índice do 'F' = 2
Número de elementos na 1ª dimensão = 6
Lower Bound = 0
Upper Bound = 5
Valor na posição 2 = A
Índice de 'A' = 1
Índice de 'A' = 5
Ordem Crescente
ABCDEF
Ordem Decrescente
FEDCBA
Ordem Inversa do Vetor
CDEFAB
C:\ProgCS\Capítulo 8\Exemplo 3>
```

Figura 2: Compilação e Execução do exemplo Vetores

Cálculos com Matrizes

O exemplo abaixo mostra um programa de cálculos com matrizes bidimensionais de inteiros. Ele cria uma classe *Matriz* onde possui os métodos *Somar*, *Subtrair* e *Imprimir* para efetuar operações de soma com matrizes, subtração com matrizes e exibir a matriz:

```
//csc /nologo Matrizes.cs

using System;

public class Matriz
{
    private int[,] matriz;
    private int linhas, colunas;

    public Matriz(int linhas, int colunas)
    {
        this.linhas = linhas;
        this.colunas = colunas;
        matriz = new int[linhas, colunas];
    }

    public Matriz Somar(Matriz m)
    {
        Matriz res = new Matriz(this.linhas, this.colunas);

        for(int a = 0, l = linhas; a < l; ++a)
            for(int b = 0, l2 = colunas; b < l2; ++b)
                res.SetValue(a + 1, b + 1, matriz[a, b] + m.GetValue(a + 1, b + 1));

        return res;
    }

    public Matriz Subtrair(Matriz m)
```

Usando vetores com o .NET Framework e C#

por Fabio Galuppo

```
{
    Matriz res = new Matriz(this.linhas, this.colunas);

    for(int a = 0; a < linhas; ++a)
        for(int b = 0; b < colunas; ++b)
            res.SetValue(a + 1, b + 1, matriz[a, b] - m.GetValue(a + 1, b + 1));

    return res;
}

public void SetValue(int linha, int coluna, int valor)
{
    matriz[linha - 1, coluna - 1] = valor;
}

public int GetValue(int linha, int coluna)
{
    return matriz[linha - 1, coluna - 1];
}

public void Imprimir()
{
    for(int a = 0; a < linhas; ++a)
    {
        Console.WriteLine("");
        for(int b = 0; b < colunas; ++b)
            Console.Write("{0} ", matriz[a, b]);
    }
}

public static void Main()
{
    Console.Write("\nM1");

    Matriz m1 = new Matriz(3, 3);
    m1.SetValue(1, 1, 50); m1.SetValue(1, 2, 100); m1.SetValue(1, 3, 1);
    m1.SetValue(2, 1, 33); m1.SetValue(2, 2, 300); m1.SetValue(2, 3, 2);
    m1.SetValue(3, 1, 22); m1.SetValue(3, 2, 700); m1.SetValue(3, 3, 5);

    m1.Imprimir();

    Console.Write("\n\nM2");

    Matriz m2 = new Matriz(3, 3);
    m2.SetValue(1, 1, 8); m2.SetValue(1, 2, 200); m2.SetValue(1, 3, 15);
    m2.SetValue(2, 1, 9); m2.SetValue(2, 2, 700); m2.SetValue(2, 3, 1);
    m2.SetValue(3, 1, 3); m2.SetValue(3, 2, 100); m2.SetValue(3, 3, 0);

    m2.Imprimir();

    Console.Write("\n\nSomar = M1 + M2");

    Matriz resultado = m1.Somar(m2);

    resultado.Imprimir();

    Console.Write("\n\nSubtrair = M1 - M2");

    resultado = m1.Subtrair(m2);

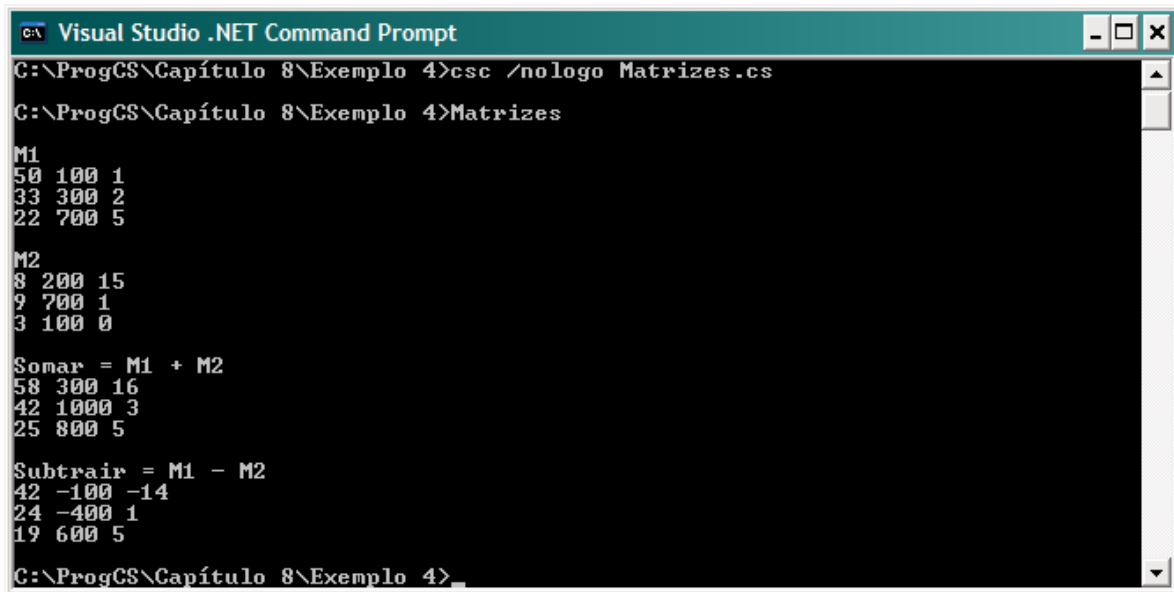
    resultado.Imprimir();
}
```

Usando vetores com o .NET Framework e C#

por Fabio Galuppo

```
        Console.WriteLine("");  
    }  
}
```

Para compilar o exemplo acima no *Command Prompt* digite *csc /nologo Matrizes.cs*. Execute o programa digitando *Matrizes*. A figura 3 mostra a compilação e execução da aplicação em C#.



```
C:\> Visual Studio .NET Command Prompt  
C:\ProgCS\Capítulo 8\Exemplo 4>csc /nologo Matrizes.cs  
C:\ProgCS\Capítulo 8\Exemplo 4>Matrizes  
M1  
50 100 1  
33 300 2  
22 700 5  
  
M2  
8 200 15  
9 700 1  
3 100 0  
  
Somar = M1 + M2  
58 300 16  
42 1000 3  
25 800 5  
  
Subtrair = M1 - M2  
42 -100 -14  
24 -400 1  
19 600 5  
C:\ProgCS\Capítulo 8\Exemplo 4>
```

Figura 3: Compilação e Execução do exemplo Matrizes

Arrays dinâmicos

Os *arrays* dinâmicos em C# permite que sejam alteradas as dimensões de um vetor com operações como *Insert* e *Remove*. Ele é utilizado através de uma instância da classe *ArrayList* que se encontra no *namespace System.Collections* do *.NET Framework*.

```
//csc VetorDin.cs  
  
using System;  
using System.Collections;  
  
public class VetorDin  
{  
    public static void Main()  
    {  
        ArrayList arr = new ArrayList();  
        arr.Add("C#");  
        arr.Add("C++");  
        arr.Add("Perl");  
        arr.Add("Visual Basic");  
        int a = arr.Count; // 4  
        arr.RemoveAt(0); //Remove da 1ª posição  
        arr.Remove("Perl");  
        a = arr.Count; // 2  
    }  
}
```

Usando vetores com o .NET Framework e C#

por Fabio Galuppo

Um dos construtores da classe *ArrayList* possui um inteiro para indicar a capacidade (*capacity*) inicial do vetor que será alocada. Procure sempre passar este valor para o construtor, mesmo que seja um valor aproximado, para mais ou para menos onde a alocação interna para o vetor passa a ser mais eficaz.

```
ArrayList arr = new ArrayList(10); //Já reserva 10 elementos antes de ser usado
```