3 Arrays, Strings

3.1 Arrays

In der Mathematik und Technik tauchen oft Schreibweisen auf wie

x_i .... i = 1, 2, 3,....n

$$\begin{bmatrix} G_{11} & G_{12} & \dots & G_{1n} \\ G_{21} & G_{22} & & G_{2n} \\ \dots & & & \\ G_{n1} & G_{n2} & & G_{nn} \end{bmatrix} * \begin{bmatrix} \varphi_1 \\ \varphi_2 \\ \\ \varphi_n \end{bmatrix} = \begin{bmatrix} I_1 \\ I_2 \\ \\ I_n \end{bmatrix}$$

$$R = R_1 + R_2 + \dots R_n = \sum_{i=1}^{n} R_i$$

Man spricht von indizierten Variablen, von Vektoren und Matrizen. Ein anderes Szenario ist die Verwendung von vielen Daten des gleichen Typs, Sie wollen z.B. mit einer Liste von Zahlen oder Namen in einem Programm arbeiten.

Nr	Zahl
1	23
2	56
3	78
9	78
10	12

Nr	Name
1	Max Fleissig
2	Felix Tschiggfrei
3	Judith Bader
n-1	Michael Franz
n	Elisabeth Broulikova

Dies ist in Programmen eine Standardsituation, denn Programme schreibt man primär für die Verarbeitung von großen Datenmengen. Eine Lösung mit dem bisherigen Ansatz

```
int z1, z2, z3, .... z10
string name1, name2, .....nameN
```

kann wohl nicht alles sein, was eine Programmiersprache bietet.

In praktisch allen Programmiersprachen gibt es das Konzept der Arrays (Datenfelder, indizierte Liste etc.). Es erlaubt die Vereinbarung von n Datenwerten unter einem Namen, wobei die einzelnen Elemente dann über den Namen des Arrays und einen Index angesprochen (ausgewählt) werden.

```
int [] numbers;
```

definiert zunächst nur den Namen `numbers` als Array. Die Erzeugung der Elemente erfolgt mit der Anweisung

```
numbers = new int[10];
```

Damit sind 10 int-Werte erzeugt, die im Prinzip noch nicht bestimmte Werte haben. (C# initialisiert sie allerdings mit 0). Man kann die Vereinbarung des Arrays und die Erzeugung auch zusammenfassen:

```
int [] numbers = new int[10];
```

In der Familie der C-Sprachen beginnt die Indizierung mit 0, d.h. die einzelnen Zahlen werden mit den Ausdrücken `numbers[0]` bis `numbers[9]` ausgewählt. Die Anzahl der Elemente im Array kann mit der Property `Length` ermittelt werden.

Typischer Programmcode um alle Elemente eines Arrays zu bearbeiten:

```
for (int i = 0; i < numbers.Length; i++)
{
  numbers[i] = i*i;
}
```

Die Tatsache, dass der Index programmierbar ist, also z.B. in einer Schleife beliebige Werte innerhalb der definierten Größe annehmen kann, ist der eigentliche große Vorteil des Konzepts.

Eine oft nützliche Möglichkeit ist folgende Variante der Vereinbarung eines Arrays:

```
string [] columnNames = {"Name", "Address", "City"};
int j;
for (j = 0; j < columnNames.Length; j++)
{
  Console.WriteLine(columnNames[j]);
}
columnNames[1] = "newName";
```

Eine bestimmte Anzahl von Werten wird dabei durch Initialisierung festgelegt. Damit ergibt sich auch die Länge automatisch.

Arrays können auch mehrdimensional sein. Das folgende Beispiel definiert eine Matrix mit 3 Zeilen und 4 Spalten, wie man sie z.B. in einem Gleichungssystem für 3 Unbekannte verwendet.

```
double [,] a = new double[3,4];
int i, j;
for (i = 0; i < 3; i++) // rows
{
  for (j = 0; j < 4; j++) // columns
    a[i,j] = (i+1)*10 + j + 1;
}

for (i = 0; i < 3; i++) // rows
{
  for (j = 0; j < 4; j++) // columns
    Console.Write("{0:f0}  ", a[i,j]);
    Console.WriteLine();
  }
}
```

Der Code erzeugt folgende Ausgabe:

```
11  12  13  14
21  22  23  24
31  32  33  34
```

Achtung: Bei der Arbeit mit Arrays muss man immer darauf achten, dass der Index im zulässigen Bereich liegt. Für ein Array mit der Größe (`Length`) n ist das der Bereich 0 bis n-1.

Der Nachteil des Arrays ist, dass es, wenn es einmal erzeugt ist, die Größe nicht bei Bedarf wieder ändern kann. Mit .NET 2.0 ist diese Einschränkung gefallen, es gibt `eine` Resize Methode.

Arrays sind (wie alles) in C# Objekte mit Eigenschaften und einigen interessanten Methoden für die wichtigsten Grundaufgaben mit Arrays (Suchen, Sortieren, Reihenfolge umdrehen etc). Die Methoden sind meist als statische Methode der Array-Klasse implementiert und das Array wird als Argument übergeben.

Beispiele:

`Array.Sort (Array a)`	Sortiert ein Array nach einem vordefinierten Sortierkriterium.
`Array.Reverse (Array a)`	Dreht die Reihenfolge der Elemente um.
`int Array.BinarySearch( Array a, object o);`	Sucht das Element o in einem sortierten Array.

Programm-Beispiel:

```
class Program
{
    static void Main(string[] args)
    {
       int n = 10;
       int[] numbers = new int[n];
       Random random = new Random();
       for (int i = 0; i < n; i++) {
          numbers[i] = random.Next(101);
          Console.WriteLine("{0,3:d}: {1,3:d}", i, numbers[i]);
       }
       Console.WriteLine("\nSort the array, Press <Enter> to start!");
       Console.ReadLine();
       Array.Sort(numbers);
       for (int i = 0; i < n; i++) {
          Console.WriteLine("{0,3:d}: {1,3:d}", i, numbers[i]);
       }
       Console.Write("\nFind an element: Enter a number: ");
       string input = Console.ReadLine();
       int numberToFind = int.Parse(input);
       int index = Array.BinarySearch(numbers, numberToFind);
       if (index >= 0)
          Console.WriteLine("{0,3:d} found at position {1,3:d}",
             numberToFind, index);
       else
          Console.WriteLine("{0,3:d} not found!", numberToFind);

       Console.WriteLine("\nPress <Enter> to end program!");
       Console.ReadLine();
    }
}
```

3.2 Strings

Eine Besonderheit sind in C# die Zeichenketten (*strings*). Strings sind Objekte, der Typ `string` ist also ein Referenztyp, andererseits unterstützt C# den Typ so gut, dass man Strings fast wie einen elementaren Datentyp verwenden kann. Strings sind keine char-Arrays (wie in C++ oder in C), erlauben aber über einen Indexer den Zugriff auf die einzelnen Unicode-Zeichen in der Form `s[index]`. String Objekte sind unveränderbar. Methoden, die einen String verändern liefern als Resultat eine modifizierte Kopie des originalen Strings. Für das Entfernen nicht mehr benötigter Strings sorgt der Garbage-Collector. Das hat Auswirkungen auf die Performance. Für aufwendige String Operationen ist die Verwendung des StringBuilder die bessere Lösung.

3.2.1 Strings erzeugen

Eine gebräuchliche Art ist

string s1 = "This is a string literal";

Man nennt eine fixen Zeichenkette (Zeichenkettenkonstante) ein *string literal*. Innerhalb der Apostrophe kann man Escape-Zeichen wie `\n \t` verwenden, sie sind auch wichtig, um das Backslash-Zeichen selber `\\` und den Apostroph `\"` als Zeichen einzubauen.

Eine Variante ist ein "*verbatim string literal*", (verbatim = wortwörtlich) das mit einem Klammeraffen beginnt:

```
string filePath =
  @"\\M:\AInf\C#Projects\BankAccounts\bin\Debug\BankAccounts.exe
```

In der normalen Schreibweise muss man dafür schreiben:

```
string filePath =
  "\\\\M:\\AInf\\C#Projects\\BankAccounts\\bin\\Debug\\BankAccounts.exe
```

Zeichen, die man auf der Tastatur nicht findet, kann man mittels des Unicodes der Zeichen in einer Zeichenkette definieren:

```
string mikroOhm = "\u03bc\u03a9";
```

ist zum Beispiel der String "μΩ".

Zur Erzeugung von Strings kann man aber auch einen der verschiedenen Konstruktoren verwenden.

```
string line = new string('-', 20);
```

erzeugt einen String mit 20 Minus-Zeichen.

```
string nullString;
```

Nach einer Vereinbarung ohne Initailisierung oder Erzeugung ist die Variable eine Null-Referenz, also der Wert von `nullString` ist `null`.

Will man einen zunächst leeren String erzeugen, so verwendet man

```
string emptyString = string.Empty;
```

das ist besser als

```
string emptyString = "";
```

3.2.2 Operatoren

Für Strings sind die Operatoren =, +, += , == und != definiert (überladen).

```
string s1 = "ABCD";
string s2 = "1234";
string s3;
s3 = s1;              // Copy s1 to s3 using the overlodad operator =
s3 = string.Copy(s1); // Using the string.Copy method
s3 = s1 + s2;         // Concatenation
s3 += "****";         // Append "****"
if (s1 == s2) ...
```

3.2.3 Eigenschaften und Methoden

Es gibt jede Menge äußerst nützliche statische Methoden in der String-Klasse:

```
Copy
Format
Compare
EndsWith
Insert
Remove
StartsWith
Substring
Trim
TrimEnd
TrimStart
Format
...
```

Eine Beschreibung der Methoden und Eigenschaften findet man in der Online Hilfe. Oft genügt schon der vom Editor angezeigte Hilfetext, um die benötigte Methode auszuwählen und richtig zu verwenden.

3.2.4 String-Arrays

Mit String-Arrays kann man genau so umgehen wie mit Arrays, die einfache Datentypen enthalten. Einige String- und auch andere Framework-Methoden liefern als Resultat ein String-Array.

Beispiel: Zerlegung einer Eingabezeile mit bestimmten Trennzeichen in die einzelnen Bestandteile (Wörter).

```
string line = "  12, 34, 56, 16; 76,3  ";
char [] seperators = new char [] {',', ';'};
string[] tokens;
line = line.Trim();
Debug.WriteLine("After Trim: " + line);      // "12, 34, 56, 16; 76,3"
line = line.Replace(" ", string.Empty);
Debug.WriteLine("After Replace: " + line);   // // "12,34,56,16;76,3"

tokens = line.Split(seperators);
for (int i = 0; i < tokens.Length; i++)
  Debug.WriteLine(tokens[i]);
```

Die Ausgabe der "tokens" ergibt:

```
12
34
56
16
76
3
```

Will man die einzelnen Teile jetzt noch in Zahlen umwandeln, so schaut das (ohne Absicherung der Parse-Methode) so aus:

```
int[] numbers = new int[tokens.Length];
for (i = 0; i < tokens.Length; i++)
{
  numbers[i] = int.Parse(tokens[i]);
}
Array.Sort(numbers);
for (i = 0; i < tokens.Length; i++)
{
  Debug.WriteLine(numbers[i].ToString());
}
```