Arrays, Listen & Schleifen

## Arrays

Wenn man in C# große Datenmengen speichern möchte, z.b. 500 int-Werte, dann wäre es ein viel zu großer Aufwand dafür 500 Variablen zu schreiben und mit diesen zu arbeiten. Stattdessen gibt es sogenannte Arrays. Arrays sind spezielle Variablen, die viele Variablen vom gleichen Datentyp zusammenfassen. Beispiel:

|  |  |
| --- | --- |
| Index | Wert |
| 0 | 0 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| ... | ... |
| 499 | 0 |

// 500 Int variablen einzeln

int x0 = 32;

int x1 = 24;

int x2 = 245;

int x3 = 23;

int x4 = 423;

int x5 = 34;

int x6 = 100;

//[...]

int x499 = 1000;

// 500 Int variablen in einem Array

int[] X\_Werte = new int[500];

Da in einem Array die einzelnen Variablen keinen Namen besitzen, greift man auf sie mit dem sogenannten Index zu:

// Normale Variable Setzen

x1 = 5;

// Array Element mit Index 1 setzen

X\_Werte[1] = 5;

In einem Array werden alle Elemente durchnummeriert, man startet immer bei der 0 und endet mit der **Anzahl der Elemente -1**, es ist ein häufiger Fehler, dass man auf eine Index zugreift, der außerhalb dieser Grenzen liegt, dies führt zu einer "**IndexOutOfRangeExecption**".

Oft will man eine bestimmte Operation mit allen Elementen eines Arrays durchführen. Dazu benutzt man **Schleifen!**

## Schleifen

Schleifen sind, ähnlich wie If-Blöcke, ein Element in C#, die eine bestimmte Bedingung haben. Sie sind dazu da, bestimmten Code zu wiederholen, bis bestimmte Bedingungen eintreffen.

Die einfachste Schleife ist die **while( )-schleife**. Eine while Schleife wiederholt ihren {}- Block **solange** die Bedingung in ihren ()-Klammern gilt:

int x = 0;

while (x < 100)

{

x += 1;

}

Diese Schleife wiederholt **solange** die "x += 1" Operation, bis X nichtmehr kleiner als 100 ist, also bis X == 100 ist. Nach dieser Schleife ist X = 100.

Schleifen sind besonders nützlich, wenn man mit Arrays arbeitet. Will man alle Elemente eines Int-Arrays auf einen Wert setzen, geht das so:

|  |  |
| --- | --- |
| Index | Wert |
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| ... | ... |
| 499 | 499 |

int[] X\_Werte = new int[500];

for (int i = 0; i < X\_Werte.Length; i += 1)

{

X\_Werte[i] = i;

}

Hier wird eine **for( )-Schleife** benutzt, sie ist speziell für den Gebrauch mit Arrays nützlich. Sie hat in ihren ( )-Klammern immer 3 Argumente mit ; getrennt. Zu erst wird eine **Iterationsvariable** und deren Startwert bestimmt, hier i. (i, j, n, m sind sehr oft dafür gebrauchte Namen).

Dann wird wie bei der While Schleife eine Bedingung festgelegt. Hier muss i immer kleiner sein als die Größe des X\_Werte Arrays, also immer kleiner als 500.

Das dritte ist ein Befehl der nach jedem durchlauf der Schleife ausgeführt wird. Hier wird also mit jedem Durchlauf i um 1 erhöht.

Zusammen mit der Bedingung heißt das, dass die Schleife genau 500 mal durchläuft, und i dabei am Anfang 0 ist und bis 499 hochzählt.

Weil das auch genau die Grenzen für den Array-Index sind, können wir i als Index angeben und somit auf alle 500 Elemente des Array zugreifen. Wir setzen sie auf i, also enthält das Array jetzt alle Zahlen von 0 bis 499.

|  |  |
| --- | --- |
| Index | Wert |
| 0 | 100 - 0\*2 = 100 |
| 1 | 100 - 1\*2 = 98 |
| 2 | 100 - 2\*2 = 96 |
| 3 | 100 - 3\*2 = 94 |
| ... | ... |
| 99 | 100 - 99\*2 = -98 |

Ein weiteres Beispiel:

int[] X\_Werte = new int[100];

for (int i = 0; i < X\_Werte.Length; i += 1)

{

X\_Werte[i] = 100 - i\*2;

}

## Listen

Listen sind spezielle Formen von Arrays. Wenn man ein Array erstellt, muss man die Größe dieses Arrays angeben. Man kann das Array danach nichtmehr vergrößern oder verkleiner. Listen funktionieren anders. Sie passen ihre Größe immer an die Anzahl der Elemente an. Listen kann man fast genauso benutzen wie Arrays, nur bei der Erstellung sieht es so aus:

// Int Array:

int[] X\_Werte = new int[500];

// Int Liste:

List<int> Y\_Werte = new List<int>();

Nach der Erstellung hat die Liste immer die Größe 0. Erst durch hinzufügen mit "Add()" steigt die Größe.

Wenn man nun ein Element der Liste hinzufügen möchte, schreibt man:

// Füge 5 hinzu, Größe steigt von 500 auf 501

Y\_Werte.Add(5);

Hierbei ist zu beachten, dass die Größe natürlich steigt! Der Index des neuen Elements ist dann entsprechend der Höchste, also 0.

Beim Entfernen gibt es nun 2 Möglichkeiten.

Einmal kann man ein Objekt mithilfe des Index entfernen, mit:

// Entferne Element an Stelle 0, Größe sinkt auf 0.

Y\_Werte.RemoveAt(0);

Hierbei sinkt die Größe dementsprechend, und **alle Elemente mit höherem Index sinken um 1.** Sie "fallen" praktisch nach wenn man Elemente unter ihnen entfernt.

Die Zweite Möglichkeit ist, ein Objekt direkt zu Entfernen, indem man es angibt.

// Entferne die erste 5, die in der Liste vorkommt!

Y\_Werte.Remove(5);

Wenn in diesem Fall die 5 mehrmals vorkommt, wird die 5 mit dem kleinsten Index entfernt. Auch hier fallen die höheren Elemente einen Index nach unten.

Wenn man jetzt mit For-Schleifen die Liste verändern, müssen wir aufpassen:

Einerseits gibt es kein Y\_Werte.length, sondern bei Listen heißt es Count.

Außerdem, da die Größe von der Liste sich verändern kann, ist es ganz wichtig in der For Schleife herunterzuzählen!

for (int i = 0; i < Y\_Werte.Count; i += 1)

{

Y\_Werte.RemoveAt(i);

}

Man würde erwarten, dass diese Schleife alle Elemente der Liste entfernt, aber sie entfernt nur die Hälfte, weil durch das "Nachrücken" die Indexe sich verändern.

for (int i = Y\_Werte.Count - 1; i >= 0; i -= 1)

{

Y\_Werte.RemoveAt(i);

}

Diese Schleife zählt runter und entfernt alle Elemente.