Variablen und Konstanten von ANSI C und C# (Teil 2)

Datenstrukturen und Algorithmen entwerfen und anwenden

Inhaltsverzeichnis

1 Einleitung

Wie Sie bis jetzt bereits gelernt haben, stellen ANSI C und auch C# einige einfache (primitive) Datentypen zur Verfügung. So gibt es zum Beispiel für Ganzzahlen die Datentypen: `long`, `int`, `short`, `char` usw. sowie für Fliesskommazahlen die Datentypen: `double`, `float` usw. Es gibt aber auch Fälle, wo diese bestehenden Datentypen nicht mehr ausreichen und wir selber eigene Datentypen zu erstellen haben, um der vorgegebenen Problemstellung gerecht zu werden.

In Teil 2 wollen wir auf die Möglichkeiten eingehen, wie der Programmierer Arrays (Felder) und Listen erstellen kann. Diese kommen dann zum Einsatz, wenn sehr viele Variablen im gleichen Zusammenhang gebraucht werden.

2 Arrays (Felder)

In diesem Abschnitt wollen wir uns mit den Arrays auseinandersetzen. Bevor wir aber auf die Theorie der Array-Struktur eingehen, sollten wir uns überlegen, wo solche Arrays verwendet werden. Generell kann man sagen, dass diese Struktur überall dort zum Einsatz kommt, wo Datenwerte in einem engen Zusammenhang stehen. Als Beispiel können Sie sich eine Box mit 10 verschiedenen Unterteilungen vorstellen. In jeder Unterteilung befinden sich einzelne Aufkleber, welche die mathematischen Ziffern: 0...9 darstellen. So ist zum Beispiel im 1. Fach die Ziffer 0, im 2. Fach die Ziffer 1 ... und im 10. Fach die Ziffer 9 vorhanden. Um dies in der Programmiersprache ANSI C zu veranschaulichen, könnte man dies durch die folgende Struktur darstellen:

Beispiel:
```
int number[10];
```

Was dies bedeutet, und wie dieses neue Wissen angewendet wird, soll in diesem Dokument erklärt werden.

Merke: **!**	***Ein Array ist in der Programmiersprache C# ein Objekt, das als Instanzvariablen eine feste Anzahl von Elementen desselben Datentyps enthält. Aus diesem Grund muss ein Arrayobjekt auch mittels*** `new()` ***erstellt werden.***

2.1 Grundlagen zum Array

Definition:	***Ein Array ist eine Sammlung von Datenelementen, die alle den gleichen Datentyp aufweisen und den gleichen Namen tragen.***

Wie aus obiger Definition herauszulesen ist, handelt es sich beim Array um eine Gruppe von Speicherstellen, die den gleichen Namen tragen und sich voneinander nur durch einen Index unterscheiden. Dieser Index ist eine Zahl, welche in einer eckigen Klammer steht und auf den Variablennamen folgt.

Beispiel:
```
number[2];
```

2.2 Arrayarten

Nach einer ersten Einführung in die Grundlagen der Arrays wollen wir uns genauer mit diesem Strukturelement auseinandersetzen.

Um Arraystrukturen einteilen zu können, wird sehr oft der Aufbau eines Arrays genauer betrachtet.

Generell werden zwei verschiedene Arrayarten unterschieden. Diese heissen:

- Eindimensionale Arrays
- Mehrdimensionale Arrays

2.2.1 Eindimensionale Arrays

Bis jetzt haben wir nur das eindimensionale Array besprochen. Dieses Array kann man sich als lineare Anordnung der Array-Elemente vorstellen, wie das folgende Beispiel zeigt:

Index:	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
Wert:	*w1*	*w2*	*w3*	*w4*	*w5*	*w6*	*w7*	*w8*

Abbildung 1: Eindimensionales Array

2.2.1.1 Definition von eindimensionalen Arrays:

ANSI C:

```
Datentyp  arrayname[Elementanzahl];  // Arraybeschreibung
```

Codeumsetzung 1: Definition eines eindimensionalen Arrays in ANSI C

C#:

```
Datentyp[]  arrayname = new Datentyp[Elementanzahl];  // Arraybeschreibung
```

Codeumsetzung 2: Definition eines eindimensionalen Arrays in C#

Dieses Array besitzt insgesamt 8 Elemente mit jeweils dem gleichen Datentyp. Eine Definition eines solchen Arrays könnte z.B. folgendermassen aussehen:

ANSI C	C#
`int array1[8];`	`int[] array1 = new int[8];`

Codebeispiel 1: 1-dimensionale Array-Definitionen in ANSI C und C#

2.2.2 Mehrdimensionale Arrays

Bei dieser Array-Art werden die Elemente in mehreren Dimensionen angeordnet. So kann man sich zum Beispiel ein zweidimensionales Array als Fläche vorstellen, in dem die einzelnen Elemente linear in x- und y-Richtung verteilt sind.

y \ x	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
0	*w00*	*w01*	*w02*	*w03*	*w04*	*w05*	*w06*	*w07*
1	*w10*	*w11*	*w12*	*w13*	*w14*	*w15*	*w16*	*w17*
2	*w20*	*w21*	*w22*	*w23*	*w24*	*w25*	*w26*	*w27*
3	*w30*	*w31*	*w32*	*w33*	*w34*	*w35*	*w36*	*w37*

Abbildung 2: Zweidimensionales Array

2.2.2.1 Definition von mehrdimensionalen Arrays

ANSI C:

```
Datentyp  arrayname[EA1][EA2] … [EAn];  // Arraybeschreibung
```

Codeumsetzung 3: Definition eines mehrdimensionalen Arrays in ANSI C

C#:

```
Datentyp[, ,] arrayname = new Datentyp[EA1,EA2, …, EAn];  // Arraybeschreibung
```

Codeumsetzung 4: Definition eines mehrdimensionalen Arrays in C#

Hinweis: Die Bezeichnung "EA " steht für "Elementanzahl".

Dieses Array besitzt insgesamt 32 Elemente (4 * 8 Elemente) mit jeweils dem gleichen Datentyp. Eine Definition eines solchen Arrays könnte folgendermassen umgesetzt werden:

ANSI C	C#
`int array2[4][8];`	`int[,] array2 = new int[4,8];`

Codebeispiel 2: 2-dimensionale Array-Definitionen in ANSI C und C#

Möchte man hingegen ein dreidimensionales Array (Würfel) kreieren, so werden die einzelnen Elemente lineare in x-, y- und z-Richtung angeordnet. Eine Definition eines solchen Arrays könnte vom Aufbau her wie folgt aussehen:

ANSI C	C#
`int array3[4][8][3];`	`int[,,] array3 = new int[4,8,3];`

Codebeispiel 3: 3-dimensionale Array-Definitionen in ANSI C und C#

2.2.2.2 Hinweise zu mehrdimensionalen Arrays

- Theoretisch lässt sich ein Array mit beliebig vielen Dimensionen erzeugen. Meistens werden aber zwei- und dreidimensionale Arrays verwendet.
- Es gilt zu beachten, dass mehrdimensionale Arrays einen enormen Speicherbedarf nach sich ziehen. Siehe dazu Abschnitt: *"Array-Grösse"*.
- Neben dem enormen Speicherbedarf mehrdimensionaler Felder sorgt auch der relativ langsame Zugriff auf die einzelnen Elemente dafür, dass Felder mit mehr als zwei Dimensionen in der Praxis relativ selten vorkommen. Die Zugriffszeit ist eine Folge aufwendiger Index-Berechnungen. In Realität werden alle Elemente eines mehrdimensionalen Arrays nicht in Form einer Tabelle, sondern der Reihe nach im Speicher abgelegt. Soll ein einzelnes Element geschrieben oder gelesen werden, so muss dessen Adresse ausgehend von der Basisadresse des Arrays über die Multiplikation der Indexwerte mit der Datentypgrösse berechnet werden.

2.3 Generelle Hinweise für Arrays

- Arrays sind genau wie andere Variablen zuerst zu definieren. Eine Arraydefinition umfasst den Datentyp und die Grösse des Arrays (die Anzahl der Elemente im Array). Die Position der Array-Definition im Quellcode hat – wie bei normalen Variablen auch – Einfluss darauf, wie das Programm auf die Daten zugreifen kann.
- Der erste Index bei zweidimensionalen Arrays bestimmt nicht die Anzahl der Spalten, sondern der Zeilen.
- Die Array-Elemente sind immer fortlaufend nummeriert, wobei **das erste Element immer die Nummer 0 hat** und das letzte Element den Indexwert: *Elementanzahl - 1* besitzt. In anderen Sprachen, wie zum Beispiel BASIC, ist dem ersten Element im Array der Index 1 zugeordnet. C verwendet jedoch einen nullbasierten Index.
- Die Anzahl der Elemente (Elementanzahl, EA) in einem Array muss durch eine Integer-Konstante angegeben werden.
- Die Elemente eines Arrays belegen einen zusammenhängenden Speicherbereich, sie befinden sich der Reihe nach an aufeinanderfolgenden Adressen.
- Um eventuelle Änderungen der Arraygrösse (Elementanzahl) zu erleichtern, sollten bei der Definition von Arrays symbolische Konstanten eingesetzt werden. Eine solche lässt sich z.B. in den Programmiersprachen: ANSI C und C++ mit der Präprozessor-Anweisung: *#define* festlegen.

```
#define arrayMaxSize 10
int number[arrayMaxSize];
```

Codebeispiel 4: Angabe von Arraydimensionen mittels Konstanten in ANSI C

Achtung: Die #define-Direktive kann in der Programmiersprache: C# nicht zur Deklaration konstanter Werte verwendet werden. Definieren Sie Konstanten in C# stets als statische Member einer Klasse.

```
private static int arrayMaxSize = 10;
int[] array1 = new int[arrayMaxSize];
```

Codebeispiel 5: Angabe von Arraydimensionen mittels Konstanten in C#

2.4 Beispiele zur Definition von Arrays

2.4.1 Beispiele zur Definition von Arrays in ANSI C

```
// symbolic constants
#define yearsArrayMax  10               // number of years to be stored
#define monthsArrayMax 12               // number of months to be stored
#define daysArrayMax   31               // number of days to be stored

// array variable definitions
int   monthlyTurnover[monthsArrayMax];                      // turnover per month
float shareValue[monthsArrayMax][daysArrayMax];             // share value per day
int   turnover[yearsArrayMax][monthsArrayMax][daysArrayMax]; // turnover per day
```

Codebeispiel 6: Beispiel-Definitionen von Arrays in ANSI C

Merke: **!**	***Elemente in zweidimensionalen Arrays können in ANSI C nicht als*** `array[y,x]` ***definiert werden, sondern müssen als*** `array[y][x]` ***angegeben werden.***

2.4.2 Beispiele zur Definition von Arrays in C#

```
// symbolic constants
private static byte yearsArrayMax  = 10;  // number of years to be stored
private static byte monthsArrayMax = 12;  // number of months to be stored
private static byte daysArrayMax   = 31;  // number of days to be stored

// array variable definitions
int[]    monthlyTurnover = new int[yearsArrayMax];            // turnover per month
float[,] shareValue = new float[monthsArrayMax, daysArrayMax];// share value per day
                                                              // turnover per day
int[,,]  turnover = new int[yearsArrayMax, monthsArrayMax, daysArrayMax];
```

Codebeispiel 7: Beispiel-Definitionen von Arrays in C#

2.5 Zugriff auf Arrays

2.5.1 Lesender Zugriff auf ein eindimensionales Array-Element in ANSI C und C#

Der Zugriff auf ein Element eines Arrays funktioniert über deren Arraynamen plus dem in der Klammer stehenden Indexwert, welcher das einzelne Element indexiert.

```
variable =  arrayname[index];  // Variablenzuweisung
```

Codeumsetzung 5: Lesezugriff auf ein Arrayelement in einem eindimensionalen Array für ANSI C und C#

2.5.2 Lesender Zugriff auf ein mehrdimensionales Array-Element

ANSI C:

```
variable =  arrayname[index1]…[indexn];  // Variablenzuweisung
```

Codeumsetzung 6: Lesezugriff auf ein Arrayelement in einem mehrdimensionalen Array für ANSI C

C#:

```
variable = arrayname[index1,…, indexn];  // Variablenzuweisung
```

Codeumsetzung 7: Lesezugriff auf ein Arrayelement in einem mehrdimensionalen Array für C#

2.5.3 Schreibender Zugriff auf ein eindimensionales Array-Element in ANSI C und C#

```
arrayname[index] = Ausdruck;  // Zuweisung an ein Array-Element
```

Codeumsetzung 8: Schreibzugriff auf ein Arrayelement in einem eindimensionalen Array für ANSI C und C#

2.5.4 Schreibender Zugriff auf ein mehrdimensionales Array-Element

ANSI C:

```
arrayname[index1]…[indexn] = Ausdruck ;  // Zuweisung an ein Array-Element
```

Codeumsetzung 9: Schreibzugriff auf ein Arrayelement in einem mehrdimensionalen Array für ANSI C

C#:

```
arrayname[index1,…, indexn] = Ausdruck;  // Zuweisung an ein Array-Element
```

Codeumsetzung 10: Schreibzugriff auf ein Arrayelement in einem mehrdimensionalen Array für C#

2.5.5 Beispiele für den Zugriff auf Arrays

2.5.5.1 Beispiele für den Zugriff auf Arrays in ANSI C

```
// symbolic constants
#define measurementsArrayMax 10          // number of measurements to store
#define xArrayMax            100         // limit in x-direction
#define yArrayMax            1000        // limit in y-direction

// definition of arrays
float temperature[measurementsArrayMax]; // measured temperatures
int factor[xArrayMax][yArrayMax];        // conversion factors

…

// different access possibilities
temperature[0] = measuredValue;          // value assignment of array element [0]
factor[0][2] = inputValue;
temperature[1] = 12.5f;                  // value assignment with a constant
factor[2][6] = 12;
temperature[0] = temperature[1];         // copy of one array element
factor[0][0] = factor[99][999];
currentValue = temperature[0];           // read access of array element [0]
factorMax = factor[0][0];
outputValue = temperature[num - 1];      // read access of array element
factorMax = factor[xpos - 1][ypos - 1];
```

Codebeispiel 8: Zugriff auf Arrays in ANSI C

2.5.5.2 Beispiele für den Zugriff auf Arrays in C#

```
// symbolic constants
private static byte measurementsArrayMax = 10;   // number of measurements to store
private static byte xArrayMax            = 100;  // limit in x-direction
private static int  yArrayMax            = 1000; // limit in y-direction

// definition of arrays
float[] temperature = new float[measurementsArrayMax]; // measured temperatures
int[,] factor = new int[xArrayMax, yArrayMax];         // conversion factors

…
// different access possibilities
temperature[0] = measuredValue;           // value assignment of array element [0]
factor[0, 2] = inputValue;
temperature[1] = 12.5f;                   // value assignment with a constant
factor[2, 6] = 12;
temperature[0] = temperature[1];          // copy of one array element
factor[0, 0] = factor[99, 999];
currentValue = temperature[0];            // read access of array element [0]
factorMax = factor[0, 0];
outputValue = temperature[num - 1];       // read access of array element
factorMax = factor[xpos - 1, ypos - 1];
```

Codebeispiel 9: Zugriff auf Arrays in C#

2.5.6 Hinweise zum Zugriff auf Arrays

- Um auf den 1. Eintrag eines Arrays zuzugreifen, muss der Indexwert *0* gewählt werden; um den letzten Eintrag anzusprechen, muss der Indexwert: *Elementanzahl -1* verwendet werden.
- Ein Array-Element lässt sich in einem Programm überall dort verwenden, wo man auch eine normale Variable des gleichen Typs einsetzen kann.
- Beim Zugriff auf die einzelnen Elemente eines Arrays können Sie für den Index beliebige Integer-Ausdrücke verwenden, um den Indexwert zu berechnen.
- Der Versatz zwischen der Position eines Elements und dessen Index-Wert ist häufige Quelle tückischer Programmierfehler (Adressüberschreitungen). **ANSI C und C# überprüfen während der Kompilierung nicht, ob über das Ende eines Feldes hinaus geschrieben wird! Der Fehler wird erst während der Laufzeit des Programms zum Vorschein kommen.**
- Beachten Sie, dass es sich bei den eckigen Klammern um Operatoren handelt. Ihre Priorität liegt über jener von arithmetischen Operatoren. Der Ausdruck innerhalb der eckigen Klammern wird daher völlig ausgewertet, bevor ein Feldelement für irgendwelche Operationen herangezogen wird. Eine Ausnahme bildet jedoch der Inkrement- bzw. Dekrementoperator im Postfixmodus.
- Um möglichst effizient auf alle Elemente eines Arrays zuzugreifen, empfiehlt sich der Einsatz einer `for`-Schleife. *(Siehe dazu das Dokument: „Kontrollstrukturen von ANSI C und C# (Teil 1)“)*

2.6 Definition und Initialisierung von Arrays

Genau gleich wie Variablen können auch Arrays **bei der Definition mit Werten gefüllt werden (Initialisierung)**. Gleich bei der Definition kann eine Liste mit Anfangswerten zugewiesen werden. Die einzelnen Werte müssen rechts vom Zuweisungsoperator in geschweiften Klammern stehen und durch Kommas voneinander getrennt sein.

2.6.1 Initialisierung eines eindimensionalen Arrays in ANSI C und C#

```
Datentyp arrayname[Elementanzahl] = {Wert0,  … Wertn};  // A-Initialisierung
```

Codeumsetzung 11: Initialisierung eines eindimensionalen Arrays in ANSI C und C#

2.6.2 Initialisierung eines zweidimensionalen Arrays

ANSI C:

```
Datentyp arrayname [EAy][EAx] =  {{Wert00, … Wert0x},   // A-Initialisierung
                                  {Wert10, … Wert1x},   // x = EAx - 1
                                  {Werty0, … Wertyx}};  // y = EAy - 1
```

Codeumsetzung 12: Initialisierung eines zweidimensionalen Arrays in ANSI C

C#:

```
Datentyp[,] arrayname Datentyp[EAy,EAx]{{Wert00, … Wert0x},   // A-Init.
                                        {Wert10, … Wert1x},   // x = EAx - 1
                                        {Werty0, … Wertyx}};  // y = EAy - 1
```

Codeumsetzung 13: Initialisierung eines zweidimensionalen Arrays in C#

Hinweis: Die Bezeichnungen "EAy" und "EAx" stehen für "Elementanzahl in y-Richtung" und "Elementanzahl in x-Richtung".

2.6.3 Beispiele zur Definition und gleichzeitiger Initialisierung von Arrays

2.6.3.1 Beispiele für ANSI C

```
// definition of arrays with initialisation
float conversionFactors [] = {1.1f, 0.25f, 2.3f}; // array has 3 elements
const float currency[3] = {1.4, 1.5, 1.6}; // conversion factors of currency
int value1[3]   = {10, 100, 1000};         // explicit definition of array size
int value2[]    = {10, 100, 1000};         // compiler defines array size
int list1[10]   = {9, 6, 3};               // partly initialisation of array elements
int list2[10]   = {0};                     // all array elements initialized with 0
int list3[2][3] = {{ 1, 2, 3}, {4, 5, 6}}; // initializing a 2-dimensional array
int list4[2][3] = {1, 2, 3, 4, 5, 6};      // initializing with same effect
```

Codebeispiel 10: Definition mit gleichzeitiger Initialisierung eines Arrays in ANSI C

2.6.3.2 Beispiel für C#

```
// definition of arrays with initialisation
float[] conversionFactors = new float[] {1.1f, 0.25f, 2.3f}; // array has 3 elements
int[] value1 = new int[3] {10, 100, 1000};       // explicit definition of array size
int[] value2 = new int[]  {10, 100, 1000};       // compiler defines array size
int[,] list3 = new int[2, 3] {{1, 2, 3}, {4, 5, 6}}; // initializing a 2-dim. array
int[,] list5 = new int[,] {{1, 2, 3}, {4, 5, 6}};    // initializing with same effect
```

Codebeispiel 11: Definition mit gleichzeitiger Initialisierung eines Arrays in C#

2.6.4 Hinweise zur Initialisierung von Arrays

- Wird ein Array in ANSI C nicht explizit initialisiert, so sind seine Elemente beim Start des Programms unbestimmt. Anders sieht diese Situation bei C# aus. Hier werden alle Einträge des Arrays auf den Standardinitialisierungswert gesetzt.
- Wird in ANSI C die Arraygrösse fest definiert und nicht alle Werte für ein Array angegeben, so werden diese vom Index: `0` her aufgefüllt. Die verbleibenden Elemente werden mit dem Wert: `0` initialisiert. Diese Eigenart bei der Initialisierung von Arrays kann dazu ausgenützt werden, ohne grossen Aufwand alle Elemente eines Feldes mit dem Wert: `0` zu initialisieren. *(Siehe obige Beispiele mit den Arraynamen:* `list1`*,* `list2`*)*
- Es ist in C# nicht möglich ein Array mit dem Wert: `0` bei der Definition auf einmal zu initialisieren. Es muss aber gesagt werden, dass ohne Wertangabe das Array sowieso mit dem Standardwert belegt wird. Hat ein Array z.B. den Datentyp: `int` so erhält jedes Feld bei der Definition des Arrays den Standardinitialisierungswert: `0`.
- Eine unvollständige Initialisierungswerteangabe welche bei ANSI C möglich ist, kann in C# nicht verwendet werden (*Siehe obige Beispiel mit dem Arraynamen:* `list1`)!
- Initialisieren Sie, wenn immer möglich, Arrays gleich bei ihrer Definition. Bei einer späteren Initialisierung benötigen Sie dazu immer zusätzlichen Code, der im Normalfall in Form einer Schleifenkonstruktion aufgebaut ist.
- Wird eine komplette Liste der Werte zu einem Array angegeben, so kann auf die Grössenangabe (Elementanzahl) des Arrays bei der Definition verzichtet werden. *(Siehe obiges Beispiel mit dem Arraynamen:* `conversionFactors`*)*
- Die Initialisierung eines zweidimensionalen Arrays erfolgt in der Weise, dass zuerst alle Werte einer Zeile angegeben werden die den gleichen 1. Indexwert besitzen. Sobald alle Werte angegeben wurden (Anzahl Elementwerte pro Zeile = 2. Indexwert), werden alle Werte angegeben, die den gleichen nächsten 1. Indexwert besitzen. Diese Angaben gehen in gleicher Art weiter, bis im Normalfall alle Elemente des Arrays initialisiert wurden. *(Siehe obige Beispiele mit den Arraynamen:* `list3`*,* `list4`*)*
- Aus obigen Beispielen erkennen Sie, dass ein zweidimensionales Array in ANSI C auf zwei Arten initialisiert werden kann. In der ersten anzustrebenden Möglichkeit werden kleine Gruppen gebildet, welche die einzelnen Initialisierungsteile darstellen. In der zweiten Möglichkeit werden alle Werte der Reihe nach aufgeschrieben. Von dieser zweiten Möglichkeit rate ich Ihnen jedoch ab, da eine Zuordnung zu den Arrayfeldern (Dimensionen) viel schwieriger wird. *(Siehe obige Beispiele mit den Arraynamen:* `list3`*,* `list4`*)*

  Bei C# gibt es nur die erste Möglichkeit welche sowieso auch ich ANSI C eingesetzt werden sollte. (Siehe obige Beispiel mit dem Arraynamen: `list3`*)*
- Wird bei der Initialisierung eines Arrays die Arraygrösse nicht angegeben, so berechnet der Compiler selbständig die benötigte Grösse. Um dies zu tun, betrachtet er die Anzahl und die Struktur der Initialisierungswerte. *(Siehe obige Beispiele mit den Arraynamen:* `list2`*,* `list5`*)*
- In der praktischen Softwareentwicklung kommt es häufig vor, dass man in einem Array Werte hinterlegt, die konstant bleiben und niemals verändert werden sollen. Um solche Werte, die als Initialisierungsliste angegeben werden, vor einem versehentlichen Überschreiben zu schützen, muss das Schlüsselwort: *const in ANSI C* verwendet werden. *(Siehe obiges Beispiel mit den Arraynamen:* `currency`*)*

  Achtung: Bei C# gibt es diese Möglichkeit nicht!

2.7 Array-Grösse

Die maximale Grösse eines Arrays ist nur durch die Grösse des zur Verfügung stehenden Speichers beschränkt. Als Programmierer brauchen Sie sich mit den internen Abläufen nicht zu befassen; das Betriebssystem kümmert sich um alles, ohne dass Sie eingreifen müssen.

Die Anzahl der Bytes, die ein Array im Speicher belegt, ergibt sich aus der Anzahl der für das Array deklarierten Elemente und der Grösse der Elemente selbst. Die Elementgrösse hängt wiederum vom Datentyp des Arrays und von der Darstellung dieses Datentyps auf dem konkreten Computer ab.

Um den Speicherbedarf eines Arrays zu ermitteln, multipliziert man die Anzahl der Elemente im Array mit der Elementgrösse.

Merke: **!**	***Wenn ein Array definiert wird, reserviert der Compiler einen Speicherblock, der das gesamte Array aufnimmt.*** ***Die einzelnen Array-Elemente sind nacheinander im Speicher abgelegt.***

2.7.1 Berechnungsbeispiel zur Bestimmung der Speichergrösse eines Arrays

In diesem Beispiel sollen Temperaturwerte, welche in einem zeitlichen Abstand von 10 Minuten gemessen wurden, in einem Array dargestellt werden. Das Array wird in ANSI C wie folgt aufgebaut:

```
double temperature[12][31][24][6]; // temperature values per 10 minutes in a year
```

Rechnet man den benötigten Speicherplatz für dieses Array aus, so bekommt man den folgenden Wert:

Array-Speichergrösse = 12 * 31 * 24 * 6 * 8 = 428‘544 Bytes!

2.7.2 Bestimmung der Speichergrösse eines Arrays mittels Programmbefehl

- Mit dem sizeof()-Operator von ANSI C lässt sich die Anzahl der Bytes ermitteln, die von einem Array belegt werden.

- Bei C# kann dieser Befehl nicht mehr in diesem Zusammenhang verwendet werden. Um trotzdem den Speicherbedarf eines Arrays in C# bestimmen zu können, muss zuerst mittels dem Befehl: `array.Length` die Anzahl der Felder bestimmt und dann mit der Anzahl Bytes des Datentyps multipliziert werden.

2.8 Zeichenketten (Strings)

2.8.1 Generelles zu Zeichenketten

2.8.1.1 Zeichenketten in ANSI C

Die Programmiersprache ANSI C kennt keinen Datentyp, der speziell für Zeichenketten geschaffen wurde. Bei ihr ist eine Zeichenkette nichts anderes als ein Array von `char`-Werten.

Allen Zeichenketten gemeinsam ist der Aufbau: Nachdem jedes einzelne Zeichen eines Textes nacheinander im eindimensionalen Array abgespeichert wurde, wird dieser Text mit einem binären `0` abgeschlossen. Um dieses Zeichen selber im Array abzuspeichern, wird die Schreibweise '`\0`' benutzt. Man beachte bitte, dass die Hochkommas ebenfalls eingegeben werden müssen, um dieses Zeichen korrekt einzutragen. Man spricht in diesem Zusammenhang von einem sogenannten „nullterminierten String“.

2.8.1.2 Zeichenketten in C#

Für Zeichenketten in C# existiert ein eigener Datentyp mit der Bezeichnung: `string`. Tatsächlich ist es so, dass der Datentyp: `string` ein Synonym für die Klasse: `String` ist. Objekte der Klasse: String, also Zeichenketten, verfügen somit über Eigenschaften und Methoden, ähnlich wie Sie dies bereits bei den Array sehen konnten.

2.8.2 Kopieren von Strings

2.8.2.1 String-Kopierung in ANSI C

Soll ein String kopiert werden, so geht dies am einfachsten über die Bibliotheksfunktion: `strcpy()` aus der Bibliothek: `string` mit der Prototypendatei*:* `string.h`.

```
strcpy(zeichenkette1, zeichenkette2);  // Zeichenkette1 = Zeichenkette2
```

Codeumsetzung 14: Kopieren eines Strings in ANSI C

```
#include <string.h>         // string library

char text[10];              // array definition for string

strcpy(text, "Hello");      // copying "Hello" in array: text
printf("%s", text);
```

Codebeispiel 12: Kopieren von Strings in ANSI C

Hinweis:

In ANSI C wird für den Kopiervorgang eines Strings die Funktion: `strcpy()` *eingesetzt. Da wir aber in unserem Unterricht zur Entwicklung unserer Programme Visual Studio verwenden und dort als Programmiersprache C++ einsetzen, muss als Funktion:* `strcpy_s()` *eingesetzt werden.*

Die Kopie des Textes: "`Hello`" wird im Array: `text` wie folgt abgelegt:

0	1	2	3	4	5	6	7	8	9	← Index des Arrays
H	e	l	l	o	\0					← Inhalt der einzelnen Felder

Abbildung 3: Inhalt der Arrayfelder für Text: „Hallo“

2.8.2.2 String-Kopierung in C#

Beim Kopieren verhält sich ein Objekt der Klasse: `String` wie eine einfache Variable und nicht wie ein Objekt. Wenn eine Zeichenkette einer anderen Zeichenkette zugewiesen wird, so sind diese beiden Zeichenketten voneinander unabhängig. Eine Veränderung des Originals hat keine Veränderung der Kopie zur Folge!

```
zeichenkette1 = zeichenkette2;  // Zeichenkette1 = Zeichenkette2
```

Codeumsetzung 15: Kopieren eines Strings in C#

```
string text1 = "Hello friends";
string text2;

text2 = text1;
                                              // text1 is copied in text2
Console.WriteLine("text 1: " + text1);        // output: "Hello friends"
Console.WriteLine("text 2: " + text2);        // output: "Hello friends"

text1 = "other text";                         // text1 has a new content

Console.WriteLine("text 1: " + text1);        // output: "other text"
Console.WriteLine("text 2: " + text2);        // output: "Hello friends"
```

Codebeispiel 13: Kopieren von Strings in C#

2.8.3 Verknüpfen von Strings

2.8.3.1 String-Verknüpfung in ANSI C

Soll ein String mit einem zweiten String verknüpft (zusammengeführt) werden, so verwendet man am einfachsten die Bibliotheksfunktion: `strcat()` aus der Bibliothek: `string` mit der Prototypendatei*:* `string.h`.

```
strcat(zeichenkette1, zeichenkette2);  // Zeichenkette1 += Zeichenkette2
```

Codeumsetzung 16: Zusammenfügen von Strings in ANSI C

```
#include <string.h>         // string library

char text[10];              // array definition for string

strcpy(text, "Hello");
strcat(text, " Joe");       // text = "Hello" + " Joe" = "Hello Joe"
printf("%s", text);
```

Codebeispiel 14: Verknüpfung von Strings in ANSI C

Hinweis:

In ANSI C wird für das Verknüpfen von Strings die Funktion: `strcat()` *eingesetzt. Da wir aber in unserem Unterricht zur Entwicklung unserer Programme Visual Studio verwenden und dort als Programmiersprache C++ einsetzen, muss als Funktion:* `strcat_s()` *eingesetzt werden.*

Die Verknüpfung der beiden Texte: "`Hello`" und " `Joe`" führt zu folgendem Ergebnis:

0	1	2	3	4	5	6	7	8	9	← Index des Arrays
H	e	l	l	o		J	o	e	\0	← Inhalt der einzelnen Felder

Abbildung 4: Inhalt der Arrayfelder für Text: „Hallo“ + „Joe“

2.8.3.2 String-Verknüpfung in C#

Um dies zu tun, verwendet man einfach den '+' – Operator.

```
zeichenkette1 + zeichenkette2;  // Zeichenkette1 + Zeichenkette2
```

Codeumsetzung 17: Zusammenfügen von Strings in C#

```
string text1 = "Hello";
string text2 = " Joe";
string text3;

text3 = text1 + text2;                      // text = "Hello" + " Joe"

Console.WriteLine("text 1: " + text1);      // output: "Hello"
Console.WriteLine("text 2: " + text2);      // output: " Joe"
Console.WriteLine("text 3: " + text3);      // output: "Hello Joe"
```

Codebeispiel 15: Verknüpfung von Strings in C#

2.8.4 Vergleich von Strings

2.8.4.1 String-Vergleich in ANSI C

Soll ein String mit einem zweiten String verglichen werden, so verwendet man am einfachsten die Bibliotheksfunktion: `strcmp()` aus der Bibliothek: `string` mit der Prototypendatei: `string.h`.

```
strcmp(zeichenkette1, zeichenkette2);  // Vergleich Zeichenketten
```

Codeumsetzung 18: Vergleichen von Strings in ANSI C

Diese Funktion vergleicht zwei Zeichenketten (Zeichenkette1, Zeichenkette2) miteinander und gibt das Ergebnis des Vergleiches als Rückgabewert zurück. Der Rückgabewert wird wie folgt berechnet:

- Zeichenkette1 **kleiner** als Zeichenkette2 → Rückgabewert **<0**
- Zeichenkette1 **gleich** der Zeichenkette2 → Rückgabewert **0**
- Zeichenkette1 **grösser** als Zeichenkette2 → Rückgabewert **>0**

```
#include <string.h>                 // string library

char text1[] = "Hello";             // array definition and initialising for strings
char text2[] = "Hello Joe";

if (strcmp(text1, text2) == 0) {    // comparing string of equal text
  …
}
```

Codebeispiel 16: Vergleich von Strings in ANSI C

Im obigen Beispiel wird der `if`-Anweisungsblock nicht ausgeführt, da die beiden Texte nicht identisch (gleich) sind.

2.8.4.2 String-Vergleich in C#

Um dies zu tun, verwendet man einfach den '==' – Operator.

```
zeichenkette1 == zeichenkette2;  // Zeichenkette1 = Zeichenkette2?
```

Codeumsetzung 19: Vergleichen von Strings in C#

```
string text1 = "Hello";
string text2 = " friends";
string text3 = "Hello";

if (text1 == text2)                          // comparing text1 and text2
{                                            // output: "different text"
  Console.WriteLine("identical text ");
}
else
{
  Console.WriteLine("different text ");
}

if (text1 == text3)                           // comparing text1 and text3
{                                             // output: "identical text"
  Console.WriteLine("identical text");
}
else
{
  Console.WriteLine("different text ");
}
```

Codebeispiel 17: Vergleich von Strings in C#

2.8.5 Bestimmung der Stringlänge

2.8.5.1 Stringlängenabfrage in ANSI C

Soll die Länge eines Strings bestimmt werden, so verwendet man am einfachsten die Bibliotheksfunktion: `strlen()` aus der Bibliothek: `string` mit der Prototypendatei*:* `string.h`.

```
strlen(zeichenkette1);  // Länge der zeichenkette1
```

Codeumsetzung 20: Bestimmung der Stringlänge in ANSI C

```
#include <string.h>                          // string-library

char text[20] = "Hello friends";             // define string variable: text

printf("text:        %s\n", text);           // output: text
printf("String length: %d", strlen(text));   // calculate string length
```

Codebeispiel 18: Bestimmung der Stringlänge in ANSI C

2.8.5.2 Stringlängenabfrage in C#

In C# kann das Attribut: Length abgefragt werden.

```
zeichenkette1.Length;  // Laenge der Zeichenkette 1
```

Codeumsetzung 21: Bestimmung der Stringlänge in C#

```
string text1 = "Hello class";

Console.WriteLine("text:       " + text1);          // output: "text:  Hello class"
Console.WriteLine("text length: " + text1.Length);  // output: "text length: 11"
```

Codebeispiel 19: Bestimmung der Stringlänge in C#

3 ArrayList-Klasse in C#

Eine Arrayliste vergrössert sich beim Hinzufügen von Elementen bzw. verkleinert sich beim Löschen von Elementen. Die Elemente einer ArrayList können wir wie bei einem Array mit einem Index ansprechen. Wenn wir in die Elemente einer ArrayList wieder eine ArrayList hinzufügen, erzeugen wir eine mehrdimensionale Struktur.

Merke: ***!***	***Eine Arrayliste (ArrayList) ist die Kombination eines Arrays mit einer dynamischen Liste.*** ***In Listen lassen sich am Kopf, am Ende und in der Mitte Elemente einfügen bzw. löschen.***

Intern ist die Arrayliste eine doppelt verkettete Liste, bei der wir einerseits nach vorn und hinten und andererseits über den Index navigieren können.

Eine ArrayList ist nicht wie ein Array auf einen bestimmten Datentyp festgelegt. Dies bedeutet:

Merke: ***!***	***Eine ArrayList kann zur gleichen Zeit unterschiedliche Datentypen beinhalten.***

Damit Sie die ArrayList verwenden können, müssen Sie mit Hilfe von `using` den Namensraum: `System.Collections` einbinden.

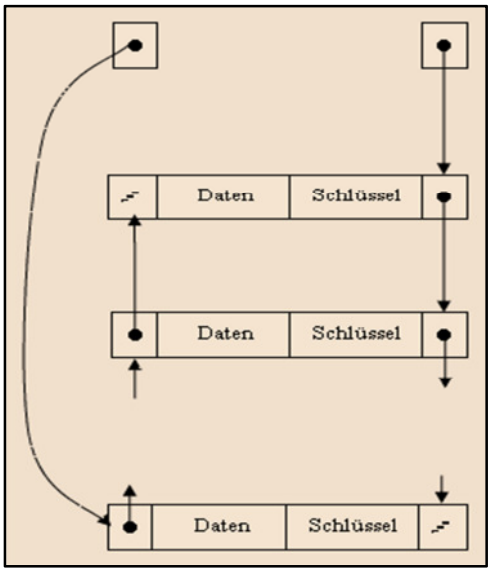

Abbildung 5: Doppelt verkettete Liste

Eine doppelt verkettete Liste unterscheidet sich von einer einfach verkettete Liste durch einen zweiten Zeiger, der auf das Vorgängerelement zeigt. Hierdurch ist es möglich, sowohl vorwärts wie auch rückwärts zu navigieren.

Bei dieser Gelegenheit ist es sinnvoll, neben der Erde einen zweiten Zeiger auf das Ende der Liste einzuführen, um so schnell dorthin zu gelangen.

3.1 Definition einer ArrayList in C#

```
ArrayList arrayListName = new ArrayList();  // Erzeugung einer ArrayListe
```

Codeumsetzung 22: Definition einer ArrayList in C#

```
using System.Collections;

ArrayList aList1 = new ArrayList();                 // create ArrayList: alist1
```

Codebeispiel 20: Definition einer ArrayList in C#

3.2 Verwendung einer ArrayList in C#

3.2.1 Bestimmung der Speichergrösse (Kapazität) einer ArrayList

```
arrayListName.Capacity;  // Angabe zur Speichergroesse der ArrayList
```

Codeumsetzung 23: Bestimmung der Speichergrösse einer ArrayList in C#

Die Anfangskapazität (Anzahl Elemente) einer ArrayList ist immer 0. Aus diesem Grund müssen wir die Anzahl der Elemente selbst vorgeben. Dies geschieht mit der Methode: `Add()`. Jedes Mal wenn wir diesen Befehl verwenden, wird der ArrayList ein weiteres Element hinzugefügt.

```
using System.Collections;

ArrayList aList1 = new ArrayList();                    // create ArrayList: aList1
Console.WriteLine("memory space: " + aList1.Capacity); // output: "memory space: 0"
```

Codebeispiel 21: Bestimmung der Speichergrösse einer ArrayList in C#

3.2.2 Hinzufügen eines Elementes am Ende einer ArrayList

Fügt am Ende der Liste ein Element hinzu.

```
arrayListName.Add(object);  // Objekt einer ArrayListe anfuegen
```

Codeumsetzung 24: Hinzufügen von Elementen am Ende einer ArrayList in C#

Beim Hinzufügen müssen wir uns aber merken, dass sich die ArrayList nach folgendem Muster in der Grösse (Kapazität) verändert:

4, 8, 16, 32, 64, 128 …

Wir erkennen somit, dass sich die Grösse beim Hinzufügen des 5. Elementes auf 8 verändert und dann bis zum 8 Element gleich bleibt. Fügen wir dann das 9. Element ein, so vergrössert sich die Grösse des ArraysList auf 16 und bleibt dann bis zum 16. Element gleich. Diese Reihe kann somit immer in dieser Art und Weise fortgesetzt werden.

```
using System.Collections;

ArrayList aList1 = new ArrayList();                        // create ArrayList: aList1

for (int count = 0; count < 9; count++)                    // output: content: aList1
{
  aList1.Add(count);                                       // add element
  Console.Write("element: " + count + "  -> ");            // output: element-no.
  Console.Write("number of elements: " + aList1.Count);    // outp: number of elements
  Console.WriteLine("memory space: " + aList1.Capacity);   // output: used memory spa.
}
```

Codebeispiel 22: Verwendete Speichergrösse einer ArrayList in C#

Programmausgabe:

```
element: 0  -> number of elements: 1  -> memory space: 4
element: 1  -> number of elements: 2  -> memory space: 4
element: 2  -> number of elements: 3  -> memory space: 4
element: 3  -> number of elements: 4  -> memory space: 4
element: 4  -> number of elements: 5  -> memory space: 8
element: 5  -> number of elements: 6  -> memory space: 8
element: 6  -> number of elements: 7  -> memory space: 8
element: 7  -> number of elements: 8  -> memory space: 8
element: 8  -> number of elements: 9  -> memory space: 16
```

Mit der Methode: Add() können verschiedene Objekte (Datentypen) innerhalb einer ArrayList gespeichert werden wie folgendes Beispiel zeigt:

```
using System.Collections;

ArrayList aList1 = new ArrayList();         // create ArrayList: aList1

aList1.Add(100);                            // add integer value
aList1.Add(12.3456);                        // add double value
aList1.Add("hello friend");                 // add string

for (int count = 0; count < 3; count++)     // output: content: aList1
  Console.WriteLine("content of ArrayList on position: " + count +
                    ": " + aList1[count]);
}
```

Codebeispiel 23: Elementen mit unterschiedlichen Datentypen in eine ArrayList von C# hinzufügen

Programmausgabe:

```
content of ArrayList on position: 0: 100
content of ArrayList on position: 1: 12.3456
content of ArrayList on position: 2: hello friend
```

3.2.3 Hinzufügen eines Elementes an einer beliebigen Stelle im ArrayList

Fügt am angegebenen Index ein Element in die ArrayList ein.

```
arrayListName.Insert(index, object);  // Objekt einer ArrayListe hinzufuegen
```

Codeumsetzung 25: Hinzufügen von Elementen an beliebiger Stelle in eine ArrayList für C#

```
using System.Collections;

ArrayList aList1 = new ArrayList();         // create ArayList: aList1

aList1.Add(100);                            // add integer value
aList1.Add(12.3456);                        // add double value
aList1.Add("hello friend");                 // add string
aList1.Insert(1, "new value");              // add new value at position 1

for (int count = 0; count < 4; count++)     // output: content: aList1
{
  Console.WriteLine("content of ArrayList on position: " + count +
                    ": " + aList1[count]);
}
```

Codebeispiel 24: Hinzufügen eines Elementes in eine ArrayList an einer bestimmter Position in C#

Programmausgabe:

```
content of ArrayList on position: 0: 100
content of ArrayList on position: 1: new value
content of ArrayList on position: 2: 12.3456
content of ArrayList on position: 3: hello friend
```

3.2.4 Bestimmung der Elementanzahl in einer ArrayList

```
arrayListName.Count;  // Angabe zur Anzahl der Elemente in der ArrayList
```

Codeumsetzung 26: Bestimmung der Anzahl Elemente einer ArrayList in C#

```
using System.Collections;

ArrayList aList1 = new ArrayList();                  // create ArrayList

for (int count = 0; count < 9; count++)
{
  aList1.Add(count);                                 // add element
}
Console.Write("number of elements: " + aList1.Count);// output: "no. of elements: 9"
```

Codebeispiel 25: Bestimmung der Anzahl der Elemente innerhalb einer ArrayList in C#

3.2.5 Löschen einer ArrayList

Entfernt alle Elemente aus der ArrayList.

```
arrayListName.Clear();  // Loescht alle Elemente einer ArryList
```

Codeumsetzung 27: Löschen einer ArrayList in C#

```
using System.Collections;

ArrayList aList1 = new ArrayList();                        // create ArrayList

for (int count = 0; count < 9; count++)
{
  aList1.Add(count);                                       // add element
}
Console.WriteLine("number of elements: " + aList1.Count); // outp: "no. elements: 9"

aList1.Clear();                                            // delete all elements

Console.WriteLine("number of elements: " + aList1.Count); // outp: "no. elements: 0"
```

Codebeispiel 26: Komplettes Löschen einer ArrayList in C#

3.2.6 Löschen eines bestimmten Elementes in einer ArrayList

Löscht das Element mit dem angegebenen Wert.

```
arrayListName.Remove(object);  // Loescht ein spezifisches  Element
```

Codeumsetzung 28: Löschen eines bestimmten Elementes in einer ArrayList für C#

```
using System.Collections;

ArrayList aList1 = new ArrayList();                  // create ArrayList

aList1.Add(100);                                     // add integer value
aList1.Add(12.3456);                                 // add double value
aList1.Add("hello friend");                          // add string

for (int count = 0; count < aList1.Count; count++)  // output: content: aList1
{
  Console.WriteLine("content of ArrayList on position: " + count +
                    ": " + aList1[count]);
}

Console.WriteLine("\nsingle element: \"12.3456\" delete");

aList1.Remove(12.3456);                              // delete element: "12.3456"

for (int count = 0; count < aList1.Count; count++)  // output: content: aList1
{
  Console.WriteLine("content of ArrayList on position:: " + count +
                    ": " + aList1[count]);
}
```

Codebeispiel 27: Löschen eines bestimmten Elementes in C#

Programmausgabe:

```
content of ArrayList on position: 0: 100
content of ArrayList on position: 1: 12.3456
content of ArrayList on position: 2: hello friend
single element: "12.3456" delete
content of ArrayList on position: 0: 100
content of ArrayList on position: 2: hello friend
```

3.2.7 Löschen eines Elementes an einer bestimmten Position in einer ArrayList

Löscht ein Element an der angegebenen Position.

```
arrayListName.RemoveAt(index);  // Loescht das Element mit diesem Index
```

Codeumsetzung 29: Löschen eines Elementes an einer bestimmten Position in einer ArrayList für C#

```
using System.Collections;

ArrayList aList1 = new ArrayList();        // create ArrayList

aList1.Add(100);                                     // add integer value
aList1.Add(12.3456);                                 // add double value
aList1.Add("hello friend");                          // add string

for (int count = 0; count < aList1.Count; count++)  // output: content: aList1
{
  Console.WriteLine("content of ArrayList on position: " + count +
                    ": " + aList1[count]);
}

Console.WriteLine("\nsingle element: \"12.3456\" delete");

aList1.RemoveAt(1);                                 // delete element: "12.3456"

for (int count = 0; count < aList1.Count; count++)  // output: content: aList1
```

```
{
  Console.WriteLine("content of ArrayList on position: " + count +
                    ": " + aList1[count]);
}
```

Codebeispiel 28: Löschen eines Elementes an einer bestimmten Position in einer ArrayList bei C#

Programmausgabe:

```
content of ArrayList on position: 0: 100
content of ArrayList on position: 1: 12.3456
content of ArrayList on position: 2: hello friend

single element: "12.3456" delete
content of ArrayList on position: 0: 100
content of ArrayList on position: 2: hello friend
```

3.2.8 Durchsuchen einer ArrayList nach einem bestimmten Objekt

Bestimmt, ob sich ein bestimmtes Objekt innerhalb einer ArrayList befindet.

```
arrayListName.Contains(object);  // Durchsucht ArrayList nach Objekt
```

Codeumsetzung 30: Durchsuchen einer ArrayList nach einem bestimmten Objekt in C#

```
using System.Collections;

ArrayList aList1 = new ArrayList();         // create ArrayList

aList1.Add(100);                            // add integer value
aList1.Add(12.3456);                        // add double value
aList1.Add("hello friend");                 // add string

if (aList1.Contains("hello friend") == true)
{
  Console.WriteLine("found object");        // output: "found object"
}
```

Codebeispiel 29: Durchsuchen einer ArrayList nach einem bestimmten Objekt in C#

3.2.9 Sortieren einer ArrayList

Sortiert die Elemente einer ArrayList nach alphabetischer Reihenfolge

```
arrayListName.Sort();  // Sortiert die ArrayList
```

Codeumsetzung 31: Sortieren einer ArrayList in C#

```
using System.Collections;

ArrayList aList1 = new ArrayList();                 // create ArrayList

aList1.Add("BMW");                                  // add car make
aList1.Add("Ford");
aList1.Add("Audi");
aList1.Add("Mercedes");
aList1.Add("VW");
aList1.Add("Porsche");

for (int count = 0; count < aList1.Count; count++) // output: car make unsorted
{
  Console.Write(aList1[count] + ", ");
```

```
}

aList1.Sort();                                          // sort car make

Console.WriteLine();

for (int count = 0; count < aList1.Count; count++) // output: car make sorted
{
  Console.Write(aList1[count] + ", ");
}
```

Codebeispiel 30: Sortieren einer ArrayList in C#

Programmausgabe:

```
BMW, Ford, Audi, Mercedes, VW, Porsche,
Audi, BMW, Ford, Mercedes, Porsche, VW,
```

3.2.10 Reihenfolge der Elemente einer ArrayList umkehren

Kehrt die Reihenfolge der vorhandenen Elemente um.

```
arrayListName.Reverse();  // Reihenfolge umkehren
```

Codeumsetzung 32: Ändern der Reihenfolge einer ArrayList in C#

```
using System.Collections;

ArrayList aList1 = new ArrayList();                 // create ArrayList

aList1.Add("Audi");                                 // add car make
aList1.Add("BMW");
aList1.Add("Ford");
aList1.Add("Mercedes");
aList1.Add("Porsche");
aList1.Add("VW");

for (int count = 0; count < aList1.Count; count++) // output: car make orig. order
{
  Console.Write(aList1[count] + ", ");
}

aList1.Reverse();                                   // reverse order

Console.WriteLine();

for (int count = 0; count < aList1.Count; count++) // output: car make reverse order
{
  Console.Write(aList1[count] + ", ");
}
```

Codebeispiel 31: Umkehren der Reihenfolge einer ArrayList in C#

Programmausgabe:

```
Audi, BMW, Ford, Mercedes, Porsche, VW,
VW, Porsche, Mercedes, Ford, BMW, Audi,
```

3.2.11 Weiter Methoden und Eigenschaften einer ArrayList

In diesem Abschnitt wurden nur einige wichtige Eigenschaften und Methoden einer ArrayList aufgeführt. Um die weiteren Möglichkeiten kennenzulernen, sei auf die MSDN-Library verwiesen:

https://msdn.microsoft.com/de-de/library/system.collections.arraylist(v=vs.110).aspx

4 List<T>-Klasse in C#

Bei der List<T>-Klasse handelt es sich um eine Liste von Elementen, welche alle den gleichen Datentyp aufweisen müssen. Dies stellt auch den Hauptunterschied zur ArrayList-Klasse dar, welche innerhalb der Liste unterschiedliche Datentypen zulässt.

4.1 Definition einer Liste in C#

```
List<Datentyp> listName = new List<Datentyp>();  // Erzeugung eine Liste
```

Codeumsetzung 33: Definition einer Liste in C#

```
using System.Collections;

List <string> list1 = new List<string>();                  // create List
```

Codebeispiel 32: Listendefinition in C#

4.2 Verwendung einer List in C#

Die Verwendung einer Liste in C# ist nahezu identisch zur Verwendung einer ArrayList. Aus diesem Grund sei hier direkt auf den folgenden MSDN-Link verwiesen:

https://msdn.microsoft.com/de-de/library/6sh2ey19(v=vs.110).aspx

4.2.1 Verwendungsbeispiel einer Liste in C#

```
using System.Collections;

List<string> list1 = new List<string>();           // create List

list1.Add("Audi");                                  // add  car make
list1.Add("BMW");
list1.Add("Ford");
list1.Add("Mercedes");
list1.Add("Porsche");
list1.Add("VW");

for (int count = 0; count < list1.Count; count++)  // output: car make orig. order
{
  Console.Write(list1[count] + ", ");
}

list1.Reverse();                                   // reverse order

Console.WriteLine();

for (int count = 0; count < list1.Count; count++)  // output: car make reverse order
{
  Console.Write(list1[count] + ", ");
}
```

Codebeispiel 33: Verwendungsbeispiel einer Liste in C#

Programmausgabe:

```
Audi, BMW, Ford, Mercedes, Porsche, VW,
VW, Porsche, Mercedes, Ford, BMW, Audi,
```

5 ObservableCollection<T>-Klasse in C#

Bei der ObservableCollection<T>-Klasse handelt es sich um eine dynamische Datenauflistung von Elementen, welche alle den gleichen Datentyp aufweisen müssen. Sie ähnelt somit sehr der Klasse: List<T>. Der Unterschied besteht aber darin, dass diese Klasse Benachrichtigungen bereitstellt, wenn Elemente hinzugefügt, entfernt oder die gesamte Liste aktualisiert wird. Dies stellt auch den Hauptunterschied zur ArrayList- oder List<T>-Klasse dar, welche keine solchen Benachrichtigungen zur Verfügung stellen.

Die ObservableCollection werden wir meistens im Zusammenhang mit Datenbindung (Binding) einsetzen. Auf diese Weise können wir z.B. Benutzerschnittstellen entwerfen, welche komplett objektorientiert umgesetzt werden und somit dem Standard entsprechen, den wir anstreben wollen.

(Siehe Dokument: „Binding in C#“)

5.1 Definition einer ObservableCollection in C#

```
ObservableCollection<Datentyp> collectionName =
  new ObservableCollection<Datentyp>();            // Erzeugung eine dyn. Liste
```

Codeumsetzung 34: Definition einer ObservableCollection in C#

```
using System.Collections.ObjectModel;

// create dynamic list
ObservableCollection<Point> collectionName = new ObservableCollection<Point>();
```

Codebeispiel 34: Dynamische Listendefinition mit einer ObservableCollection<T>-Klasse in C#

5.2 Verwendung einer ObservableCollection in C#

Die Verwendung einer ObservableCollection<T>-Klasse in C# ist nahezu identisch zur Verwendung einer List<T>-Klasse. Sie besitzt mehr oder weniger die gleichen Eigenschaften und Methoden dieser Klasse. Aus diesem Grund sei hier direkt auf den folgenden MSDN-Link verwiesen:

https://msdn.microsoft.com/de-de/library/ms668604(v=vs.110).aspx

Die Verwendung selbst werden wir im Dokument: *„Binding in C#“ anschauen.*

5.2.1 Verwendungsbeispiel einer ObservableCollection in C#

```
ObservableCollection<string> dynList1 = new ObservableCollection<string>();

// create dynamic list

dynList1.Add("Audi");                                  // add car make
dynList1.Add("BMW");
dynList1.Add("Ford");
dynList1.Add("Mercedes");
dynList1.Add("Porsche");
dynList1.Add("VW");

for (int count = 0; count < dynList1.Count; count++)  // output: car make
{
  Console.Write(dynList1[count] + ", ");
}
```

```
dynList1.Remove("Mercedes");                          // delete "Mercedes"

Console.WriteLine();

for (int count = 0; count < dynList1.Count; count++)  // output: car make
{
  Console.Write(dynList1[count] + ", ");
}
```

Codebeispiel 35: Verwendungsbeispiel einer ObservableCollection in C#

Programmausgabe:

```
Audi, BMW, Ford, Mercedes, Porsche, VW,
VW, Porsche, Ford, BMW, Audi,
```

Abbildungen:

Codeumsetzungen:

Codebeispiele:

Historie

Vers.	Bemerkungen	Verantwortl.	Datum
0.1	- erstes Zusammenführen der Informationen	W. Odermatt	18.07.2005
1.0	- letzte Layout Anpassungen gemacht	W. Odermatt	13.09.2005
1.1	- Korrekturen durchgeführt	W. Odermatt	12.09.2006
1.2	- Korrekturen durchgeführt	W. Odermatt	22.11.2010
1.3	- Korrekturen durchgeführt	W. Odermatt	29.04.2013
2.0	- Anpassungen an neuen Modulinhalt: M403 umgesetzt	W. Odermatt	13.06.2014
3.0	- Anpassungen, Neuaufteilung und Erweiterungen an neue Programmiersprache C#	W. Odermatt	14.12.2015
3.1	- Anpassungen an neuen Modulinhalt: M411 umgesetzt	W. Odermatt	16.02.2016
3.2	- letzte Layout Anpassungen gemacht	W. Odermatt	22.02.2016

Referenzunterlagen

LfNr.	Titel / Autor / File / Verlag / ISBN	Dokument-Typ	Ausgabe
1	„Programmiersprache C“, Implementierung C, Grundkurs / H.U. Steck	Theorieunterlagen	2003
2	„Programmieren in C“ / W. Sommergut / Beck EDV-Berater im dtv / 3-423-50158-8	Fachbuch	1997
3	„C Kompakt Referenz“ / H. Herold / Addison Wesley / 3-8273-1984-6	Fachbuch	2002
4	„C in 21 Tagen“ / P. Aitken, B. L. Jones / Markt + Technik / 3-8272-5727-1	Fachbuch	2000
5	„C Programmieren von Anfang an“ / H. Erlenkötter / rororo / 3-499-60074-9	Fachbuch	2001
6	„Programmieren in C“ / B.W. Kernighan, D.M. Ritchie / Hanser / 3-446-25497-3	Fachbuch	1990
7	„Einstieg in Visual C# 2013“ / Thomas Theis / Galileo Computing / 978-3-8362-2814-5	Fachbuch	2014
8	“Visual C# 2012” / Andreas Kühnel / Rheinwerk Computing / 978-3-8362-1997-6	Fachbuch	2013
9	“Objektorientiertes Programmieren in Visual C#” / Peter Loos / Microsoft Press / 3-86645-406-6	Fachbuch	2006