

Benutzerdefinierte und zusammengesetzte Datentypen

Neben den einfachen Datentypen existiert die Möglichkeit, selbst Datentypen aus Grunddatentypen zusammenzusetzen.

Wir werden

- Aufzählungen,
- Arrays und
- Strukturen

kennen lernen.

Inhalt

1. Aufzählungstypen	2
2. Arrays	6
2.1. Sortieren in Feldern (erster Versuch)	10
2.2. Mehrdimensionale Felder	14
3. String	17
4. Struktur	22

1. Aufzählungstypen

Soll eine Variable eine **fest** vergebene **Menge** an **nicht-numerischen Werten** annehmen können, kann man **Aufzählungstypen** (auch Enumerationstypen genannt) verwenden.

Allgemeine Form (Deklaration Aufzählungstyp):

```
enum [typename] { enumeration } [list of variables];
```



optional

Beispiel:

```
enum Farbe {rot, gelb, gruen};  
enum Wochentag {Sonntag, Montag, Dienstag, Mittwoch,  
                Donnerstag, Freitag, Samstag};
```

Nach der Deklaration eines Aufzählungstypen können Variablen definiert werden, die zu diesem Typ gehören:

Beispiel:

```
Farbe eimer, auto;           // Definition
Wochentag heute = Dienstag; // Definition und Initialisierung
```

Typ

Name

Initialwert

Intern werden Aufzählungstypen mit **Integern** kodiert, in Aufschreibungsreihenfolge wird mit 0 begonnen und hoch gezählt.

Soll eine andere Kodierung verwendet werden, kann man die bei der Deklaration angeben.

Beispiel:

```
enum Farbe {rot=1, gelb=10, gruen=18};
```

In Ausdrücken werden `enum`-Typen in `int` umgewandelt, **nicht** umgekehrt.

Als Operation auf `enum`-Typen ist nur die Zuweisung erlaubt.

Beispiele:

```
enum Farbe {rot, gelb, gruen};
enum Wochentag {Sonntag, Montag, Dienstag, Mittwoch,
               Donnerstag, Freitag, Samstag};
enum Automarke {Audi, VW, Renault} meinAuto;
Wochentag heute;

heute = Montag;           // richtig
heute = 2;                // falsch, Datentyp nicht kompatibel
int i = Samstag;         // richtig, da Samstag in int gewandelt wird
i = heute + Montag;      // richtig
meinAuto = VW;           // richtig
meinAuto = VW + Audi;    // falsch, int-Ausdruck nicht in enum konvertierbar
```

Beispiel:

```
#include <iostream.h>
main()
{
    enum color {RED, GREEN, BLUE};
    color myColor = RED;

    if (myColor == RED) {
        cout << "hot" << endl;
    }
    if (myColor == BLUE) {
        cout << "cold" << endl;
    }
    if (myColor == GREEN) {
        cout << "Is not easy being" << endl;
    }
}
```

2. Arrays

Felder (engl. array) sind Zusammenfassungen von Elementen, die alle den gleichen Typ haben. Der Zugriff erfolgt indiziert über Integer. Arrays sind statische Objekte, d.h. die Anzahl der Elemente steht zur Compilezeit fest /später werden wir auch dynamische Objekte behandeln, z.B. Vektoren)

Allgemeine Form (Deklaration Array):

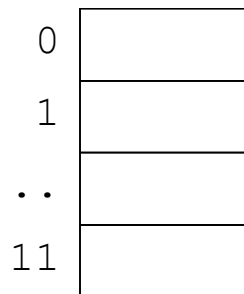
```
type name [number_elements];
```

Beispiel:

```
int tageProMonat[12];
```

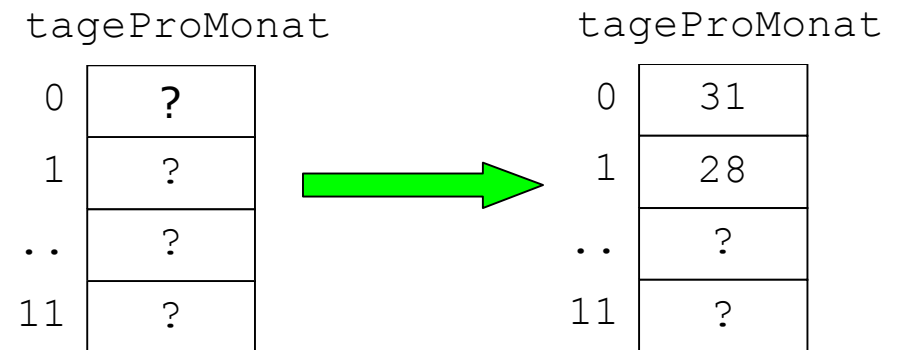
Dadurch wird ein Array mit 12 Integern angelegt.

tageProMonat



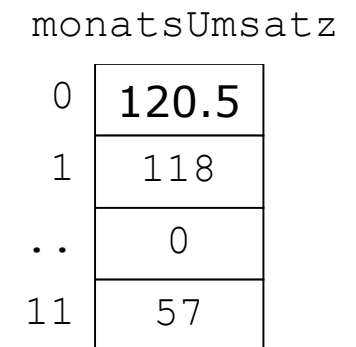
Der Zugriff auf ein Element erfolgt durch Name plus Index:

```
tageProMonat[0] = 31;  
tageProMonat[1] = tageProMonat[0] - 3;
```



Bei der Deklaration können auch Initialwerte mit angegeben werden:

```
const int anzahlMonate=12;  
double monatsUmsatz[anzahlMoate] =  
    {120.5, 118, 0, 0, 0, 0, 0, 0, 0, 0, 0, 57};
```



Beispiele:

```
int TageProMonat [] = {31,28,31,30,31,30,31,31,30,31,30,31};  
int AlleAufNull [Groesse] = {0}; // nicht initialisierte Elemente werden auf 0 gesetzt
```

Die Größe (in Bytes) eines Array kann durch `sizeof()` ermittelt werden.

Beispiel:

```
int TageProMonat [] = {31,28,31,30,31,30,31,31,30,31,30,31};  
cout << sizeof(TageProMonat) << endl;
```

48

Zuweisungen von Arrays als Ganzes sind **nicht** möglich.

Im Speicher des Rechners wird das o.a. Array wie folgt abgelegt:

abc	987
11	31
...	..
1	28
TageProMonat 0	31
xyz	123

Programmierbeispiel (BCD Darstellung von Integern: int->Ziffernfolge):

123 -> 1 2 3

```
$ cat bcd.cpp
#include <iostream.h>
main() {
    int i;
    int zahl, BCD[16]={0};
    i = 0;
    cout << "Integer: ";
    cin >> zahl;
    do {
        BCD[i] = zahl % 10;
        zahl = zahl / 10;
        i++;
    } while (zahl > 0);

    for (int j = i-1; j >=0 ; j--)
        cout << BCD[j] << " ";
    cout << endl;
}
$
```

```
$ bcd
Integer: 1234
1 2 3 4
$
```

Achtung:

Es findet keine Prüfung statt, ob stets ein erlaubter Bereich einer Array angesprochen wird!

```
#include <iostream>
using namespace std;
int a[4] = {10,11,12,13};
int y = 2;

int main() {
    cout << a[4] << endl;
}
```

```
$ t
2
$
```

Ein **Segmentierungsfehler** tritt auf, wenn man auf eine Speicherzelle zugreift, die in einem geschützten Segment (Textsegment) liegt.

```
#include <iostream>
using namespace std;
int a[4] = {10,11,12,13};
int main() {
    cout << a[-2000] << endl;
}
```

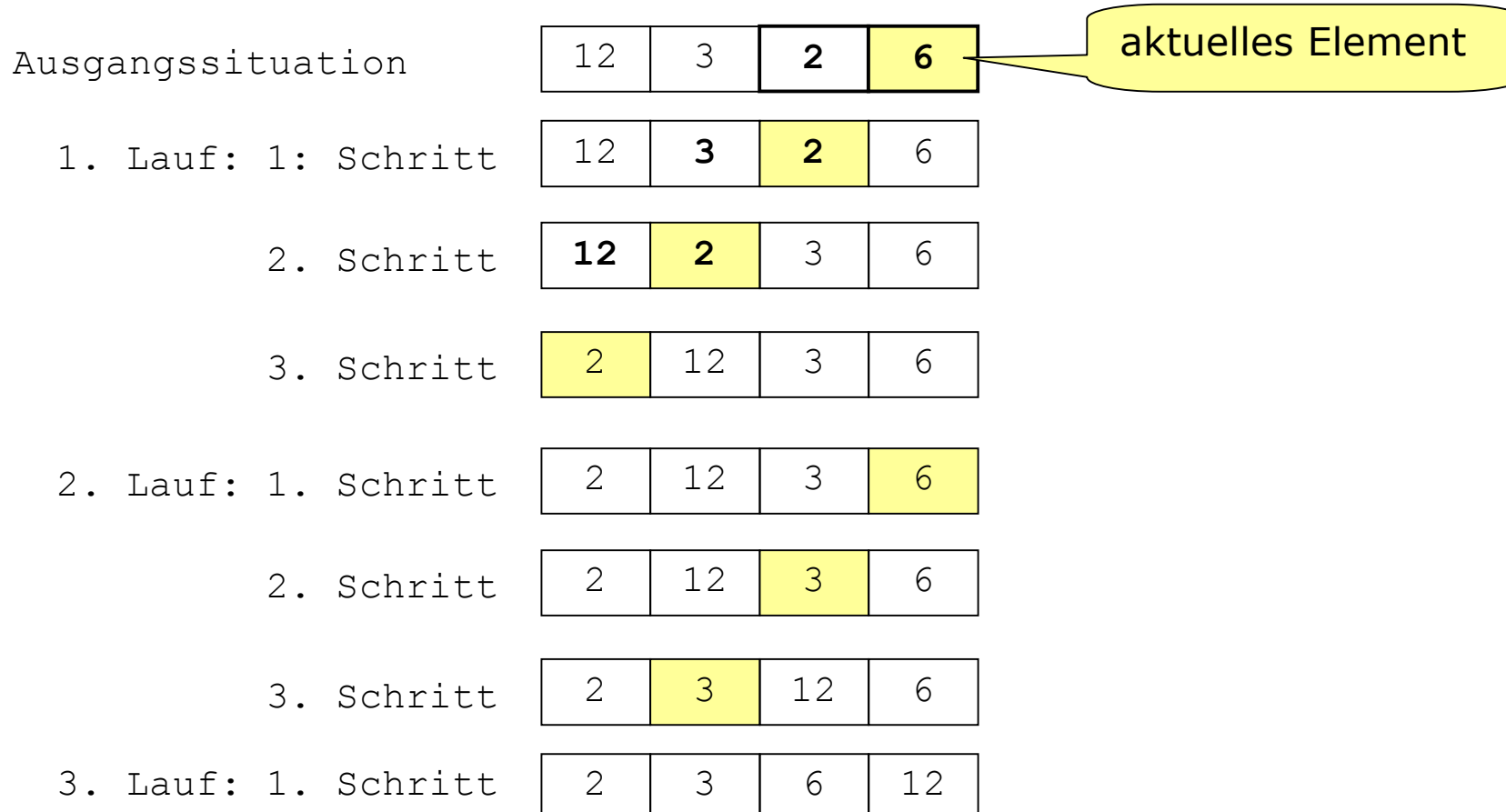
```
$ t1
Segmentation fault
$
```

2.1. Sortieren in Feldern (erster Versuch)

Bevor wir uns mit Sortieralgorithmen genauer beschäftigen, wird hier ein bekannter, aber verrufenener Algorithmus vorgestellt: **Bubble** Sort. Seine Popularität rührt von der Einfachheit – es ist aber einer der bzgl. Effizienz schlechtesten Verfahren.

Die Idee ist, wiederholt benachbarte Elemente zu vertauschen (falls „linker Nachbar“ > „aktuelles Element“.) bis alle Elemente sortiert sind

Beispiel (aufsteigend Sortieren):



Programm:

```
$ cat bubbleSort.cpp
#include <iostream.h>
main() {
    const int max=5;
    int zaehler=0, zahl,
    int daten[max]={0};

    do {
        cout << "Integer (0 Abbruch): ";
        cin >> zahl;
        daten[zaehler] = zahl;
        zaehler++;
    } while (zahl !=0 && zaehler<max);
}
```

```

// Sortieren
for (int lauf=1; lauf < zaehler; lauf++) // Läufe
    for (int element=zaehler-1; element>=lauf; element--) { // ein Lauf
        if (daten[element-1] > daten[element]) { // vertauschen
            int tmp = daten[element-1];
            daten[element-1] = daten[element];
            daten[element] = tmp;
        }
    }

for (int i= 0; i<zaehler ; i++) // Ausgeben
    cout << daten[i] << " ";
cout << endl;
}
$

```

Laufzeitanalyse (**Debuggen** Sie das Programm auf Ihrem Rechner):

Man erkennt an den zwei geschachtelten Schleifen, dass unabhängig davon, wie die Daten aussehen (also auch wenn sie bereits sortiert sind), immer die feste Anzahl von Durchläufen ausgeführt wird.

Die äußere Schleife wird $(n-1)$ mal ausgeführt, die innere Schleife wird $n/2$ mal ausgeführt, also werden stets $\frac{1}{2} * (n^2 - n)$ Vergleiche durchgeführt! Man sagt deshalb, **Bubble Sort** hat **quadratische Laufzeit** (wegen n^2 in der o.a. Formel). Dies ist ein **schlechter** Wert! Wir werden noch Verfahren mit Laufzeit $n * \log(n)$ kennen lernen.

2.2. Mehrdimensionale Felder

Bisher waren die Arrays eindimensional; in C++ sind auch mehrdimensionale Arrays möglich.

```
const int Quartale = 4; // Zeilen
const int Regionen = 3; // Spalten
int Umsatz [Quartale][Regionen] = {
    {00, 01, 02},
    {10, 11, 12},
    {20, 21, 22},
    {30, 31, 32}
};

cout << Umsatz[3][1];
```

31

Im Speicher wird das o.a. Array wie folgt abgelegt:

	xvz	987	
	2	32	
	3 1	31	
	0	30	
	2	22	
	2 1	21	
	0	20	
	2	12	
Quartale	1 1	11	
	0	10	
	2	02	Regionen
	0 1	01	
Umsatz	0	00	
	abc	123	

Beispiel (Ausgabe einer Matrix):

```
$ cat matrixAus.cpp
#include <iostream.h>
main() {
    const int m=2, n=3;
    int A[m][n] =
        {
            {1, 0, 2},
            {2, 1, 1}
        };

    for (int i = 0; i < m ; i++) {
        for (int j = 0; j < n ; j++)
            cout << A[i][j] << " ";
        cout << endl;
    }
}
$
```

```
$ matrixAus.exe
1 0 2
2 1 1
$
```

Hörsaalübung:

Sei A eine (m,n) Matrix und B eine (n,r) Matrix. Die Matrizenmultiplikation $C=A*B$ ist wie folgt definiert:

$$c_{i,j} = \sum_{l=1}^n a_{i,l} * b_{l,j} \quad \text{für } (i=1,2, \dots, m; j=1,2, \dots, r)$$

Beispiel: $(2,3)$ Matrix * $(3,2)$ Matrix

$$\begin{pmatrix} 1 & 0 & 2 \\ 2 & 1 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 2 \\ 0 & 2 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 3 & 2 \\ 3 & 6 \end{pmatrix}$$

[-> Programm](#)

3. String (Zeichenkette)

Wenn man als Komponententyp Arrays `char` nimmt, hat man ein Feld von einzelnen Zeichen, eine Zeichenkette. Die Konvention ist, dass die Zeichenkette mit dem speziellen Zeichen `\` abgeschlossen wird.

```
char zeile[80]
```

Dies ist in C eine Möglichkeit, mit Strings umzugehen.

Wir werden das später noch verdeutlichen.

Tafelübung

- „Länge eines String“,
- „Enthält String spezielles Zeichen“
- „Aneinanderhängen von Strings“
- „Reversieren eines Strings“

In C++ existiert eine elegantere Möglichkeit, die Standardklasse `string`.

Ein String wird wie folgt definiert:

```
string beispiel1 = "Hallo";  
string beispiel2("Welt");
```

Es existieren viele Funktionen, z.B.:

- Verkettung +
- Vergleiche ==, !=, <, <=, >, >=

Das Hallo-Welt-Programm kann damit auch so formuliert werden:

```
$ cat string_HalloWelt.cpp
#include <iostream.h>
#include <string>
main() {
    string teil1 = "Hallo";
    string teil2("Welt");
    string zusammen;

    zusammen = teil1 + ", " + teil2;

    cout << zusammen << endl;
}
$
```

Damit wird String-Bibliothek bekannt gemacht.

```
$ string_HalloWelt.exe
Hallo, Welt
$ edit
string_HalloWelt.cpp
$
```

Strings sind hier dynamische Objekte, d.h. der erforderliche Speicher wird automatisch allokiert. Es kommt nicht zu einer Speicherverletzung, wenn der String zu groß wird (wie bei `char b[80]`).

Strings werden in den Beispielen immer wieder verwendet, um weitere Sprachkonstrukte zu erklären.

Das folgende Beispiel fasst einige Operationen auf Strings zusammen.

```
$ cat perationenAufStrings.cpp
```

```
#include <iostream.h>
```

```
#include <string>
```

```
main() {
```

```
    string einString= "hallo";
```

```
    // Stringausdruck ausgeben
```

```
    cout << einString + "\n";
```

```
    // String zeichenweise ausgeben
```

```
    for (long i = 0; i < einString.size(); i++)
```

```
        cout << einString[i] << " ";
```

```
    cout << endl;
```

```
    // String zeichenweise ausgeben
```

```
    for (long i = 0; i < einString.length(); i++)
```

```
        cout << einString[i] << " ";
```

```
    cout << endl;
```

```
    // Zeichen eines String ohne Indexprüfung ausgeben
```

```
    cout << einString[100] << endl;
```

```
$ operationenAufStrings
```

```
hallo
```

```
h a l l o
```

```
h a l l o
```

```
// String zeichenweise mit Indexprüfung ausgeben
for (long i = 0; i < einString.length(); i++)
    cout << einString.at(i) << " ";
cout << endl;
```

h a l l o

```
// Zeichen eines String mit Indexprüfung ausgeben
cout << einString[100] << endl;
```

```
// Kopie eines String erzeugen
string einStringKopie(einString);
cout << einStringKopie << endl;
```

hallo

```
// Kopie durch Zuweisung
string neuerString("neu!");
einStringKopie = neuerString;
cout << einStringKopie << endl;
```

neu!

```
// Zuweisung Zeichenkettenkonstante
einStringKopie = "Zeichenkette";
cout << einStringKopie << endl;
```

Zeichenkette

```
// String verketteten
```

```
einStringKopie += " mit Blanks";  
cout << einStringKopie << endl;
```

```
// Strings vergleichen
```

```
if (einString > einStringKopie)  
    cout << "ENDE Beispiel!" << endl;
```

```
}  
$
```

Zeichenkette mit Blanks

ENDE Beispiel!

Hörsaalübung

Formulieren Sie die Programme, die wir mit C-Strings erarbeitet haben nun mit C++-Strings.

- Länge eines String“,
- „Enthält String spezielles Zeichen“
- „Aneinanderhängen von Strings“
- „Reversieren eines Strings“

4. Struktur

Ein Array war eine Zusammenfassung von Elementen des gleichen Typs. Will man eine Komposition von Elementen auch unterschiedlichen Typs, dann sind Strukturen die Lösung.

Hier soll nur auf Grundlegende Eigenschaften von Strukturen eingegangen werden, das sie nochmals genauer im Zusammenhang mit Objekten behandelt werden.

Allgemeine Form (Strukturdeklaration):

```
struct [Typname] { Components } [Variable_list];
```

Beispiel:

```
enum Wochentag {Sonntag, Montag, Dienstag, Mittwoch,  
               Donnerstag, Freitag, Samstag};  
  
struct Datum {  
    int Tag;  
    int Monat;  
    int Jahr;  
    Wochentag TagBezeichnung;  
};
```

Soll eine Variable dieses Typs definiert werden, so erfolgt dies durch:

```
Datum heute;
```

Auf eine Komponenten der Variablen kann durch den Selektionsoperator „.“ zugegriffen werden.

```
heute.Monat = 7;  
heute.TagBezeichnung = Montag;
```

Beispiel (Bruchrechnung):

```

$ cat bruch.cpp
#include <iostream.h>
main() {
    struct bruch {
        int zaehler;
        int nenner;
    };

    bruch a, b, mul, sum;

    a.zaehler = 2;
    a.nenner = 3;
    b.zaehler = 2;
    b.nenner = 3;

    sum.zaehler = a.zaehler * b.nenner + a.nenner * b.zaehler;
    sum.nenner = a.nenner * b.nenner;

    mul.zaehler = a.zaehler * b.zaehler;
    mul.nenner = a.nenner * b.nenner;

    cout << "a+b = " << sum.zaehler << "/" << sum.nenner << endl;
    cout << "a*b = " << mul.zaehler << "/" << mul.nenner << endl;
}
$

```

// Deklaration

// Definition von Variablen

```

$ bruch
a+b = 12/9
a*b = 4/9
$

```

Tafelübung:

1. Deklarieren Sie eine Struktur, die eine Adresse aufnehmen kann.

Hörsaalübung:

Ein Messwert besteht aus den Komponenten Wert und laufende Nummer.

Schreiben Sie ein Programm, das Messwerte einliest und die Nummer des größten Messwertes ausgibt.