

Strukturen

- Komplexe Strukturen.
- Strukturen und Strukturen.
- Strukturen und Array.
- Strukturen und Zeiger.
- Vordefinierte Strukturen.

Aufbau einer Struct

GRUNDLEGENDE TYPEN

```
int  
char  
float
```

Durch das Symbol
[]



ARRAY

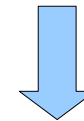
```
char [30]  
char [20]
```

Durch das Symbol *



ZEIGER

```
int *
```



STRUCT

```
char nname [30];  
char vname [20];  
float groesse;  
int alter;  
int *d1;
```

structGross.cpp

Struktur vereinbaren

in C++ (☺ bequemer als in C!)	in C und C++
<pre>struct STank { char Datum[11]; int Fahrstrecke; float Fuellmenge, Durchschnittsverbrauch; };</pre>	<pre>typedef struct { char Datum[11]; int Fahrstrecke; float Fuellmenge, Durchschnittsverbrauch; } STank;</pre>

```
typedef unsigned int WORD;
typedef char* PCHAR;
```

```
// WORD steht nun für unsigned int
// PCHAR steht nun für char*
```

Initialisieren von Strukturvariablen

Anfangswerte

BEISPIELE:

```
// Anfangswerte für einfache Variable:
```

```
int Baudrate = 9600;
```

```
unsigned char Byte = 0;
```

```
// Anfangswerte für Felder:
```

```
int Lottotip[6] = {6,21,24,30,40,44};
```

```
double E12[12] = {1.0,1.2,1.5,1.8,2,2,2.7,3.3,3.9,4.7,5.6,6.8,8.2};
```

```
char Sprache[10] = "C++";
```

Initialisieren von Strukturvariablen

Anfangswerte

```
STank Tank = {"30.09.2002", 645, 38.16};
```

Datum , Fahrstrecke , Füllmenge.

Initialisieren von Strukturvariablen

```
STank Tank = {,,645,38.16};
```

```
// oder:
```

```
STank Tank = {"30.09.2002",,38.16};
```

} wäre nicht erlaubt!

Initialisieren von Strukturvariablen

Wir möchten ein Feld mit drei Elementen vom Typ `STank` definieren und in allen drei Elementen lediglich die Komponente `Fahrstrecke` mit dem Wert `200` (z.B. die Länge einer Teststrecke) initialisieren. So geht's:

```
STank Tank[3] = { { "",200},           // Die Datum-Komponente wurde jeweils
                  { "",200},           // mit einem Leerstring initialisiert;
                  { "",200}           // die Komponenten Fuellmenge und Durch-
                };                     // schnittsverbrauch werden jeweils
                                     // automatisch mit 0 vorbelegt ( s. o. )!
```

Komplexere Strukturen:

Arrays in Struktur

Die Struktur kann auch
Arrays beinhalten

```
struct Person
{
    char name[31];
    char vname[21];
    char gebdatum[9];
    char geschlecht;
    int MatN;
}student;
```


Strukturen und Array

```
1 #include <iostream>
2 using namespace std;
3 struct STank
4 {
5     char Datum[11];
6     int Fahrstrecke;
7     float Fuellmenge, Durchschnittsverbrauch;
8 };
9
10
```

Zugriff auf die Komponenten eines Struktur-Objekts

```
1 #include <iostream>
2 using namespace std;
3 struct STank
4 {
5     char Datum[11];
6     int Fahrstrecke;
7     float Fuellmenge, Durchschnittsverbrauch;
8 };
9
10
11 int main ( void )
12 {
13     STank Tank, aTank[10], *pTank;
14
15
16 }
17
18
```

Vereinbarung von Strukturen Variablen

```
1 #include <iostream>
2 using namespace std;
3 struct STank
4 {
5     char Datum[11];
6     int Fahrstrecke;
7     float Fuellmenge, Durchschnittsverbrauch;
8 };
9
10
11 int main ( void )
12 {
13     STank Tank, aTank[10], *pTank;
14
15
16 }
17
18 ..
```

Der Reihe nach werden hier die einfache **STank**-Variable Tank, das Feld aTank mit 10 Elementen vom Typ **STank** und der **STank**-Zeiger pTank definiert.

Wir definieren zwei Speicherobjekte vom Typ `STank` :

```
STank Tank, *pTank;    // Tank ist eine Variable vom Typ STank
                        // pTank ist ein Zeiger auf den Typ STank und
pTank = new STank;      // bekommt hier die Adresse eines dynamischen
                        // ( namenlosen ) STank-Objekts zugewiesen!
```

Zugriff auf die Komponenten eines Struktur-Objekts

Auf die Komponenten eines Struktur-Objekts greift man zu mit dem	
Punktoperator .	wenn das Objekt eine Struktur- Variable (wie z.B. Tank) ist.
Pfeiloperator ->	wenn man "nur" die Adresse des Struktur-Objekts, d.h. einen Zeiger auf eine Struktur hat. Solche Strukturzeiger kommen vor allem vor als Funktionsparameter und als Zeiger auf dynamische Objekte, die mit <code>new</code> oder <code>malloc</code> erzeugt wurden!

```
#include <iostream>
using namespace std;
struct STank
{
    char Datum[11];
    int Fahrstrecke;
    float Fuellmenge, Durchschnittsverbrauch;
};

int main ( void )
{
    STank Tank, aTank[10], *pTank;
    int i;
    cout<<"Bitte das Tank-Datum für die Variable Tank eingeben: ";
    gets (Tank.Datum);
    cout<<"Bitte die Fahrstrecke für die Variable Tank eingeben: ";
    cin>>Tank.Fahrstrecke;
    cout<<"Bitte die Fuellmenge für das dynamische Objekt eingeben: ";
    cin>> pTank->Fuellmenge;
    Tank.Durchschnittsverbrauch = 100*Tank.Fuellmenge/Tank.Fahrstrecke;
    cout<<"Durchschnittsverbrauch ( Variable Tank ): "
    <<Tank.Durchschnittsverbrauch<<" L/100 km\n";
    strcpy (pTank->Datum, Tank.Datum);
    pTank->Fahrstrecke = Tank.Fahrstrecke;
    *pTank = Tank;
```

Komplexere Strukturen:

Strukturen, die Strukturen enthalten

Wie bereits erwähnt, kann eine Struktur jeden beliebigen C++-Datentypen enthalten. So kann eine Struktur unter anderem auch andere Strukturen aufnehmen.

Komplexere Strukturen:

Strukturen, die Strukturen enthalten

Lösungsweg:

1-Strukturen von x und y definieren:

2-Strukturen von Rechteck definieren:

```
struct koord
{
    int x;
    int y;
};
```

```
struct rechteck
{
    koord obenlinks;
    koord untenrechts;
};
```

struck_in_struck.cpp

Komplexere Strukturen:

Arrays von Strukturen.

Aufgabe:

Schreiben Sie ein Programm zur Verwaltung einer Telefonnummerliste:

Man sollte dabei eine Struktur definieren, die für jeden Eintrag in der Telefonliste den Namen der zugehörigen Person und die Telefonnummer enthält.

Komplexere Strukturen:

Arrays von Strukturen.

Lösungsweg:

1-Struktur definieren.

2-Eine Telefonliste besteht aus vielen Einträgen.Deshalb wäre eine einzige Instanz der Struktur nicht besonders nützlich

Komplexere Strukturen:

Arrays von Strukturen.

Lösungsweg:

1-Struktur definieren.

```
struct eintrag
{
    char vname[15];
    char nname[16];
    char telefon[12];
};
```

Komplexere Strukturen:

Arrays von Strukturen.

Lösungsweg:

2-Nachdem die Struktur definiert ist.

```
struct eintrag
{
    char vname[15];
    char nname[16];
    char telefon[12]
};
```

Kann man das Arrays wie folgt vereinbaren.

Diese Anweisung deklariert ein array namens liste mit 1000 Elementen. jedes Element ist ein Struktur von Typ eintrag und wird durch einen Index identifiziert. Jede der Strukturvariablen besteht aus drei Elementen von Typ char

eintrag liste[1000];

Arrays_von_struct.cpp

Was ist neu bei struct

- Die geschweiften Klammern bei der Deklaration
- Der Punktoperator oder pfeiloperator bei Zugriff auf Komponente

Ansonsten sind übliche Variablen.

Die Datum/Zeit–Struktur **SYSTEMTIME**

Der Struktur–Datentyp **SYSTEMTIME** ist in der Headerdatei `winbase.h` so definiert:

```
typedef struct {  
    WORD wYear;  
    WORD wMonth;  
    WORD wDayOfWeek;  
    WORD wDay;  
    WORD wHour;  
    WORD wMinute;  
    WORD wSecond;  
    WORD wMilliseconds;  
} SYSTEMTIME;
```

Das ist eine ebenso nützliche wie einfache Struktur; viel einfacher als unsere **STank**–Struktur; denn alle acht Komponenten sind vom selben Typ **WORD**, also **unsigned short int** (16 Bit !).(In `windef.h` finden wir: `typedef unsigned short WORD; !`)

Die Datum/Zeit–Struktur **SYSTEMTIME**

- eine Variable vom Typ **SYSTEMTIME** definieren und
- eine Funktion finden, die die aktuellen Datum– und Zeitwerte vom System holt und in unserer **SYSTEMTIME**–Variablen speichert!

Die Datum/Zeit–Struktur **SYSTEMTIME**

Auf der Hilfeseite zu **SYSTEMTIME** finden wir einen Verweis auf die Funktion `GetSystemTime` und von dort einen weiteren Verweis auf die noch besser geeignete Funktion `GetLocalTime`. Wir sehen dort, dass diese Funktion in `winbase.h` so deklariert ist:

```
void GetLocalTime ( LPSYSTEMTIME lpSystemTime );
```


Die Datum/Zeit–Struktur **SYSTEMTIME**

Der Parametername "spricht" zu uns (Ungarische Namenskonvention !): Das Präfix lp verrät uns, dass der Parameter ein Zeiger ist. Aber auf welchen Typ? Und wo ist überhaupt der Operator *, den wir zur Definition eines Zeigers nehmen müssen? Vor dem Parameter steht der Typbezeichner **LPSYSTEMTIME**. In der Windows-Headerdatei winbase.h finden wir des Rätsels Lösung: **LPSYSTEMTIME** steht für **SYSTEMTIME***, also für: Zeiger auf **SYSTEMTIME**! (Daran müssen Sie sich im Windows–API gewöhnen: Sehr oft steckt der Zeigerdefinitions-* für uns nicht mehr sichtbar in einem mit typedef definierten Typbezeichner. Am Präfix **LP** oder **P** können wir aber trotzdem erkennen, dass es sich um einen Zeigertyp handelt!)

Die Datum/Zeit–Struktur **SYSTEMTIME**

Warum hat `GetLocalTime` einen Zeiger–Parameter vom Typ Zeiger auf **SYSTEMTIME** ?

`GetLocalTime` sagt uns: "Legt im aufrufenden Programmteil eine Variable vom Typ **SYSTEMTIME** an; denn ich möchte in diese Variable das aktuelle Systemdatum und die Systemzeit schreiben! Damit ich das kann, müsst ihr mir sagen, wo ich Eure Ergebnisvariable finde. Ihr müsst also die Adresse dieser Variablen meinem Parameter `lpSystemTime` übergeben!"

Und damit dieser Parameter die Adresse einer **SYSTEMTIME**–Variablen übernehmen kann, muss er eben ein Zeiger auf den Strukturtyp **SYSTEMTIME** sein!

-
- 1 In der Funktion, die `GetLocalTime` aufrufen möchte (hier: `main`), definieren wir eine Ergebnisvariable vom Typ `SYSTEMTIME`. (Um hier Platz zu sparen, mit dem wenig sprechenden Namen `T`). Sie enthält in diesem Augenblick noch einen undefinierten Inhalt!
 - 2 Wir rufen die Funktion `GetLocalTime` auf und übergeben ihr die Adresse unserer Variablen `T`, damit sie diese Variable finden und mit den aktuellen Systemdatum/Systemzeit–Werten füllen kann!
 - 3 `GetLocalTime` hat die acht Komponenten unserer `SYSTEMTIME`–Variablen `T` gefüllt und wir können ihre Inhalte ausgeben.