

Vorlesung 12

- Struct (Strukturen)
 - Arrays von Strukturen
 - Verschachtelte Strukturen
 - Strukturen und Funktionen
- Zeit im Computer
 - Datum

Aufbau einer Struct

GRUNDLEGENDE TYPEN

```
int  
char  
float
```

Durch das Symbol
[]



ARRAY

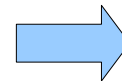
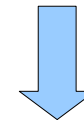
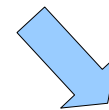
```
char [30]  
char [20]
```

Durch das Symbol *



ZEIGER

```
char *
```



STRUCT

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

StructBeispielFreund.cpp

Deklaration und Initialisierung einer Struktur

Etikett der Struktur (keine Variable!)

```
struct mensch {  
    char name[30];  
    char vorname[20];  
    unsigned short alter;  
};
```

i) Definition der Struktur

Variable des Strukturtypen

```
struct mensch student1 = { "Kohl", "Christian", 19 };
```

```
struct mensch student2 = { "Schmidt", "Anne", 20 };
```

**ii) Deklaration und
Initialisierung der
Strukturvariablen**

Komplexere Strukturen:

Arrays von Strukturen.

Aufgabe:

Schreiben Sie ein Programm zur Verwaltung einer
Telefonnummerliste:

Komplexere Strukturen:

Arrays von Strukturen.

Lösungsweg:

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 mehreren 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];
    int telefonnummer;
};
```

Komplexere Strukturen:

Arrays von Strukturen.

Lösungsweg:

```
struct eintrag
{
    char vname[15];
    char nname[16];
    int telefonnummer;
};
```

2-Nachdem die Struktur definiert ist, kann man den Arrays wie folgt vereinbaren.

Diese Anweisung deklariert einen Array namens *liste* mit 1000 Elementen. Jedes Element ist ein Struktur von Datentyp Eintrag.

```
eintrag liste[1000];
```

Arrays_von_struct.cpp

Komplexere Strukturen:

Strukturen, die Strukturen enthalten

Wie bereits erwähnt, kann eine Struktur jeden beliebigen C++-Datentypen enthalten.

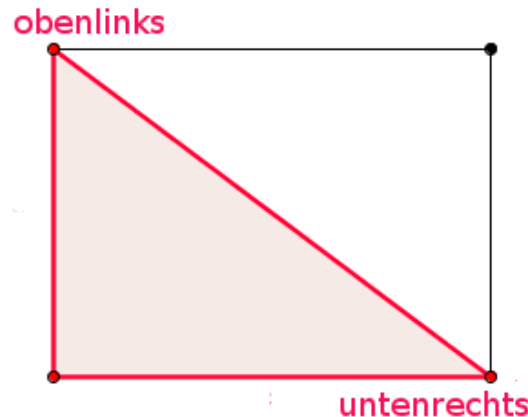
So kann eine Struktur unter anderem auch andere Struktur aufnehmen.

Komplexere Strukturen:

Strukturen, die Strukturen enthalten

Programm:

Wir möchten ein Programm programmieren, wo Rechtecke als *struct* gespeichert werden und deren Fläche berechnet wird.



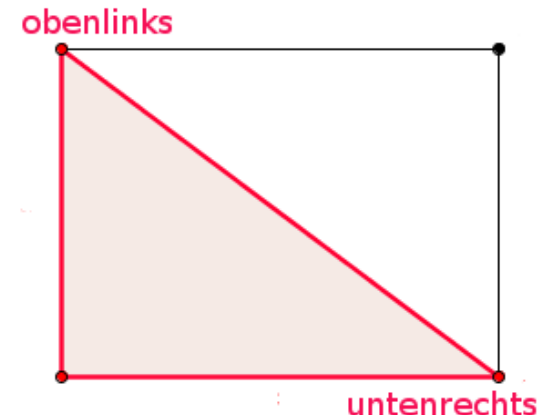
Komplexere Strukturen:

Strukturen, die Strukturen enthalten

Voraussetzung:

Ein Rechteck kann bloß durch 2 Punkte auf der selben Diagonal bestimmt werden.

Bei uns sind die ausgewählten Punkte *obenlinks* und *untenrechts*.



Anmerkung für die Programmierung: Man muss daher im Programm darauf aufpassen, dass die y-Koordinate der unteren rechten Ecke größer ist als die y-Koordinate der oberen linken Ecke und dass die x-Koordinate der unteren rechten Ecke größer ist als die x-Koordinate der unteren linken Ecke und dass alle Koordinaten positiv sind.

Komplexere Strukturen:

Strukturen, die Strukturen enthalten

Lösungsweg:

1-Struktur für einen Punkt definieren:

2-Struktur für das Rechteck (anhand der Struktur Punkt) definieren:

3-Struktur nutzen um

- Eingabe
- Berechnungen
- und Ausgabe

auszuführen.

Komplexere Strukturen:

Strukturen, die Strukturen enthalten: **Verschachtelte Strukturen**

Lösungsweg:

1-Struktur für einen Punkt definieren

2-Struktur für das Rechteck (anhand der Struktur Punkt) definieren:

```
struct punkt
{
    float x;
    float y;
};
```

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

Komplexere Strukturen:

Verschachtelte Strukturen

Lösungsweg:

```
struct punkt
{
    float x;
    float y;
};
```

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

- Die Struktur **punkt** hat zwei Elemente (x und y) von Datentyp *float*.
- Die Struktur **rechteck** hat zwei Elemente (*obenlinks* und *untenlinks*) von Datentyp **punkt**

Komplexere Strukturen:

Verschachtelte Strukturen

Weitere Komponenten in rechteck

Lösungsweg:

```
struct punkt
{
    float x;
    float y;
};
```

```
struct rechteck
{
    punkt obenlinks;
    punkt untenrechts;
    float flaeche;
};
```

- In kann der Struktur **rechteck** weitere benötigte Komponenten hinzufügen (z.B. **flaeche**)

Komplexere Strukturen:

Verschachtelte Strukturen

Lösungsweg:

```
rechteck figur1;
```

Durch diese Anweisung wird eine Variable von Datentyp *rechteck* vereinbart.

Die Variable heißt *figur1*.

Nun kann man Eingabe, Berechnungen und Ausgabe in einem Programm bequem und veranschaulichend.

struct_in_struct.cpp

Zeit in C/C++

- Die Bibliothek ist
 - `ctime` (C++ Version) *include*<ctime>
 - oder
 - `time.h` (C Version) *include*<time.h>
- Es gibt 2 Datentypen um sich mit der Uhrzeit zu beschäftigen
 - `time_t`
 - einfacher Typ (ganzzahliger Datentyp)
 - Zeitabstand zum
1. Januar 1970, 00:00:00 GMT (Greenwich Mean Time) in Sekunden
 - `tm`
 - struct
 - Information über Jahr, Monatstag, Wochentag...

Zeit C/C++: Zeit ermitteln

- Die Ermittlung der Zeit läuft ausschließlich über die Funktion *time*.

- Deklaration `time_t time(time_t *t);`

- Zuerst ist eine Variable *time_t* notwendig, dann wird die Funktion aufgerufen mit der Adresse der Variable. (Adressenoperator: &)

```
time_t Zeit;  
time (&Zeit);  
cout << "Sekunden Seit 01.01.1970 ist" << Zeit;
```

Zeit C/C++: time_t Zeit formatieren

- Die einfachste Variante ist über die Funktion *ctime*
 - Die Deklaration der Funktion ist:
`char *ctime(const time_t *time);`
 - Also wir brauchen den Wert in einer Variable `time_t`, der formatiert wird in einer Zeichenkette.

```
time ( &Zeit );
```

```
cout << "Datum " << ctime ( &Zeit ) << Zeit;
```

datum.cpp

Zeit C/C++: struct tm

- Die struct tm ist benutzerfreundlicher als der Datentyp time_t

Struktur mit folgendem Aufbau:

```
struct tm {  
    int tm_sec;    /* Sekunden */  
    int tm_min;    /* Minuten */  
    int tm_hour;   /* Stunde (0 bis 23) */  
    int tm_mday;   /* Tag im Monat (1 bis 31) */  
    int tm_mon;    /* Monat (0 bis 11) */  
    int tm_year;   /* Jahr (Kalenderjahr minus 1900) */  
    int tm_wday;   /* Wochentag (0 bis 6, Sonntag = 0) */  
    int tm_yday;   /* Tag im Jahr (0 bis 365) */  
    int tm_isdst;  /* Ungleich null bei US-Sommerzeitkonvertierung */  
};
```

Zeit C/C++: struct tm

- Die *struct tm* kann mit der Funktion *localtime* aus einem Wert *time_t* belegt werden.

Durch die besondere Datentypen muss die Funktion *localtime* mit Verweisoperator *** aufgerufen.

```
tm strukturTM;
```

```
time(&t);
```

```
strukturTM = *localtime(&t);
```

Zeit C/C++: tm Zeit Komponenten

- Wenn ich mit localtime eine struct tm belegt habe, kann ich aus dem structur die Komponenten erfahren (Wochentag, Monat, Jahrestag, Stunden, Minuten....);
 - Also wir übergeben die Adresse der struct tm und bekommen wir die formatierte Zeichenkette zurück.

```
tm strukturTM;
```

```
time(&t);
```

```
strukturTM = *localtime(&t);
```

```
cout << "Wochentag " << strukturTM.tm_wday;
```

struct_tm.cpp

Zeit C/C++: time_t Zeit formatieren

- Probleme bei time_t

- time_t ist als long implementiert

Der hat ein maximaler Wert 2147483647 (Der größte Wert für long: MAX_LONG)

- Was für ein Datum ist das?

Früher als 2050 !

Funktionen mit Struct

- Eine Funktion mit einem struct Datentyp unterscheidet sich nicht von einer üblichen Funktion.

Ich muss den struct-Datentyp für den Parameter deklarieren und die entsprechende Struktur-Variable in die Funktion übergeben.