Teil 7: Strukturen und Unionen

Gliederung

Strukturen

Typdefinitionen

Unionen

Strukturen

Strukturen

- Ursprung in Pascal als Datentyp **record, Verbunddatentyp**
- **Strukturtyp in C**
- kann Komponenten (Variablen) verschiedener Typen enthalten:

Personal-nummer	Nachname	Vorname	Straße	Haus-nummer	Postleit-zahl	Wohnort	Gehalt
`int`	`char[20]`	`char[20]`	`char[20]`	`int`	`int`	`char[20]`	`float`

- Anwendung: Gruppierung logisch zusammenhängender Daten
  z. B. Datum, Adresse, Personendaten, geometrische Objekte, …
- Behandlung beim Lesen und Schreiben immer als als Einheit

Strukturtyp-Definition

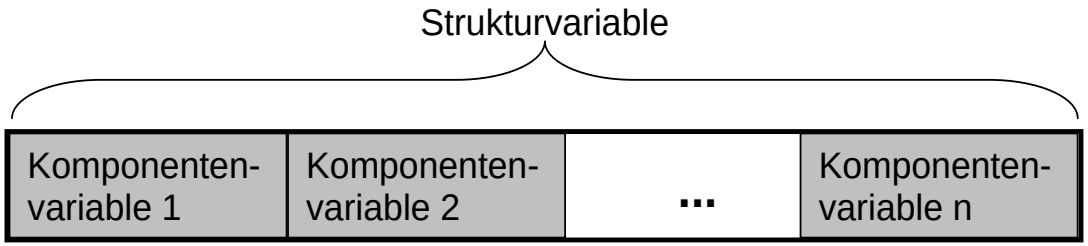
- frei definierter, zusammengesetzter Datentyp
- feste Anzahl von Komponenten

```
struct Name
{
    typ1 komponentenVariable_1;
    typ2 komponentenVariable_2;
    . . .
    typN komponentenVariable_n;
};
```

- der Typname ist `struct Name`

Strukturvariablen-Definition

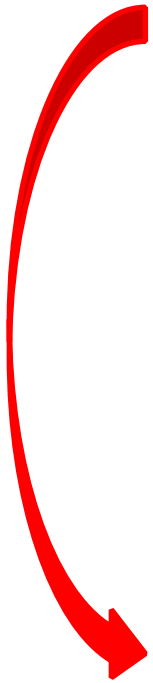
- Variable eines Strukturtyps = aus Komponentenvariablen zusammengesetzte Variable



- Erstellung einer Variablen des Typs **`struct`** `Name`

```
struct Name variable;
```

Beispiel-Definition



```
struct Adresse
{
    char  strasse[20];
    int   hausnummer;
    int   postleitzahl;
    char  stadt[20];
};

struct Student
{
    int             matrikelnummer;
    char            name[20];
    char            vorname[20];
    struct Adresse wohnort;
};

struct Student meyer, mueller;   // 2 Studenten
struct Student semester[50];     // Feld aus 50 Studenten
```

Übergabe an Funktionen

- *call-by-value* (wie bei elementaren Datentypen **`int`**, **`float`**, `etc.`)

Initialisierungslisten

```
struct Student student1 =
{
   66202,
   "Meyer",
   "Herbert",
   {
      "Schillerplatz",
      20,
      73730,
      "Esslingen"
   }
};
```

Zugriff auf Komponentenvariablen

```
struct Adresse
{
   char    strasse[20];
   int     hausnummer;
   int     postleitzahl;
   char    stadt[20];
};

struct Student
{
   int                 matrikelnummer;
   char                name[20];
   char                vorname[20];
   struct Adresse wohnort;
};

int main()
{
    struct Student studi;

   studi.matrikelnummer = 716347;
   strcpy(studi.name, "Meyer");
   studi.wohnort.postleitzahl = 73733;
   strcpy(studi.wohnort.stadt, "Esslingen");
   // ...
```

Zugriff auf Komponentenvariablen

```
    printf("%s %s\n", studi.vorname, studi.name);
    printf("%s %d\n", studi.wohnort.strasse,
                      studi.wohnort.hausnummer);
    printf("%d %s\n", studi.wohnort.postleitzahl,
                      studi.wohnort.stadt);
}
```

Kombination von Typ- und Variablendefinition

- ohne Strukturname/Etikett (**unüblich**)

```
struct
{
   float x;
   float y;
} punkt1, punkt2, punkt3;
```

- mit Strukturname/Etikett (**üblich**)

```
struct Koordinaten
{
   float x;
   float y;
} punkt1, punkt2, punkt3;

struct Koordinaten punkt4, punkt5, punkt6;
```

Struktur-Zuweisungen, Vergleich

- direkte Zuweisung möglich, alle Komponentenwerte der Variablen `punktA` werden den jeweiligen Komponenten der Variablen `punktB` zugewiesen

```
struct Koordinaten punktA = {1.5, 3.0}, punktB;
punktB = punktA;
```

- **ABER**: Prüfung auf Gleichheit `if (punktA == punktB)` ist **nicht** möglich!

- Bestimmung der Größe in Bytes einer Strukturvariablen:

```
sizeof(struct Koordinaten)
```

Selektion über Zeiger

- **Punktoperator .** und **Pfeiloperator ->**

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

struct Koordinaten punktA = {1.5, 3.0};
struct Koordinaten *ppunkt;
ppunkt = &punktA;
```

- Wie greife ich auf Komponenten von `punktA` über den Zeiger `ppunkt` zu?
- Komponenten-Selektion über Zeiger auf 2 Arten möglich:

```
(*ppunkt).x          bzw.    ppunkt->x

(*ppunkt).x = 3;             ppunkt->x = 3;
```

Zeiger und geschachtelte Strukturen

- Strukturvariablen als Komponenten

```
struct Vektor
{
   struct Koordinaten *anfang;
   struct Koordinaten *ende;
};
```

- Zeiger vom Typ `Vektor`

```
struct Vektor vec, *pvec;
vec.anfang = &punkt2;
vec.ende = &punkt3;
pvec = &vec;
...
pvec->anfang->x
```

bzw.

Strings in Strukturen

- String als Komponentenvariable

```
struct Name
{
   char nachname[20];
   char *vorname;
};

struct Name person1 = {"Maier", "Herbert"};
```

- `char nachname[20];` struktur**intern**, String gehört zur Strukturvariablen
- `char *vorname;` struktur**extern**, (in diesem Falle konstanter String)

Typdefinition mit typedef

Was ist typedef?

- Vergabe eines neuen Namens (**Alias**) *neuername* für einen
  - schon bekannten oder
  - soeben definierten Datentyp

```
typedef bekanntertyp neuername;
```

- Beispiele

```
typedef unsigned long int UINT;
typedef float REAL;

UINT a;
REAL b;
```

Anwendung

- spart Schreibarbeit
- Portierbarkeit von Programmen mit maschinenabhängigen Datentypen (maschinenabhängige Datentypen treten NUR in der **typedef** Zeile auf)
- Beispiel: Windows-API

  https://msdn.microsoft.com/en-us/library/windows/desktop/aa383751%28v=vs.85%29.aspx

Anwendung

64-bit data models

Data model	short (integer)	int	long (integer)	long long	pointers, size_t	Sample operating systems
LLP64, IL32P64	16	32	32	64	64	Microsoft Windows (x86-64 and IA-64) using Visual C++; and MinGW
LP64, I32LP64	16	32	64	64	64	Most Unix and Unix-like systems, e.g., Solaris, Linux, BSD, macOS. Windows when using Cygwin; z/OS
ILP64	16	64	64	64	64	HAL Computer Systems port of Solaris to the SPARC64
SILP64	64	64	64	64	64	*Classic* UNICOS[41] (versus UNICOS/mp, etc.)

- Beispiel: einheitlicher 64-Bit long Integer `LINT`

```
typedef long long int LINT;    // Anpassung auf LLP64-System

typedef long int LINT;         // Anpassung auf LP64-System
```

typedef und Strukturen

```
struct Datum
{
  short int tag;
  char monat[10];
  short int jahr;
};

struct Datum heute;
```

```
typedef struct
 {
   short int tag;
   char monat[10];
   short int jahr;
 } DATUM;

DATUM heute;
```

Empfohlene Variante

Beispiel: Mitarbeiter-Datenbank

```
typedef struct
{
    char vorname[80];
    char nachname[80];
    int alter;
    double gehalt;
} MITARBEITER;

int main()
{
    MITARBEITER mitarbeiter[100];     // legt 100 mitarbeiter an

    // Ersten Mitarbeiter eintragen
    strcpy(mitarbeiter[0].vorname, "Hans");
    strcpy(mitarbeiter[0].nachname, "Schmidt");
    mitarbeiter[0].alter = 23;
    mitarbeiter[0].gehalt = 5302.32;

    // Zweiten Mitarbeiter eintragen
    strcpy(mitarbeiter[1].vorname, "Dieter");
    strcpy(mitarbeiter[1].nachname, "Mueller");
    mitarbeiter[1].alter = 19;
    mitarbeiter[1].gehalt = 6230.43;
    // usw.
```

Unionen

Union-Definition

- Benutzung wie Struktur
- ABER: alle Komponenten (**Alternativen**) beginnen bei **der selben Adresse**
- speichert jeweils **nur eine einzige** Komponente (aus Reihe von Alternativen)
- Programmierer ggf. muss verfolgen, welcher Typ momentan in der Union gespeichert ist

```
union Variant
{
   int      intAlternative;
   float    floatAlternative;
};
```

Zugriff auf Alternativen

```
union Variant
{
   int      intAlternative;
   float    floatAlternative;
};

int x;
union Variant vario;
```

- Alternative schreiben / auslesen

```
vario.floatAlternative = 123.0;
x = vario.intAlternative;
```

- Zeiger auf Union

```
union Variant *pVario = &vario;

x = pVario->intAlternative;
```

Beispiel: Prozessorregister

```
struct WORDREGS
{
    unsigned int ax;
    unsigned int bx;
    unsigned int cx;
    unsigned int dx;
};

struct BYTEREGS
{
    unsigned char al,ah;
    unsigned char bl,bh;
    unsigned char cl,ch;
    unsigned char dl,dh;
};

union REGS
{
    struct WORDREGS w;
    struct BYTEREGS b;
};

union REGS inregs;

inregs.w.ax = 0x0815;  /* AX Register auf Hex 0815 setzen */
inregs.b.ch = 0x1A;    /* CH Register auf Hex 1A setzen */
```