Eigene Datentypen

>> Das C-Schlüsselwort `typedef` ermöglicht es, Kurzformen oder alternative Namen für Datentypen zu erstellen.

Dadurch können wir es vermeiden, lange Typen wie `unsigned long int` jedes Mal komplett ausschreiben zu müssen.

Außerdem kann man komplexere Datentypen erstellen.

Die Grundidee ist, einen Typ zunächst zu definieren und ihm dann einen Alias zu geben.

Syntax

```
typedef <old name> <new name>;
```

Syntax

```
typedef <old name> <new name>;
```

```
typedef unsigned char byte;
```

```
byte buffer[1024];
```

Syntax

```
typedef <old name> <new name>;
```

```
typedef char* string;
```

```
string name;
```

Strukturen

```
struct car
{
    int year;
    char model[10];
    char plate[7];
    int odometer;
    double engine_size;
};
typedef struct car car_t;
```

Strukturen

```c
struct car
{
    int year;
    char model[10];
    char plate[7];
    int odometer;
    double engine_size;
};
typedef struct car car_t;
```

```c
// Variable-Deklaration
struct car mycar;       // ohne typedef
car_t mycar;            // mit typedef

// Zugriff auf Felder
mycar.year = 2011;
strcpy(mycar.plate, "CS50");
mycar.odometer = 50505;
```

Strukturen: Kurzschreibweise

```
typedef struct car
{
    int year;
    char model[10];
    char plate[7];
    int odometer;
    double engine_size;
}
car_t;
```

Vorteil von typedef

```c
// Ohne typedef - umständlich:
struct car
{
    int year;
    char model[10];
    char plate[7];
    int odometer;
    double engine_size;
};
struct car mycar; // struct muss immer angegeben werden

// Mit typedef - elegant:
typedef struct car car_t;
car_t mycar; // kürzere Schreibweise
```

EXTRAS IN 3 MINUTEN

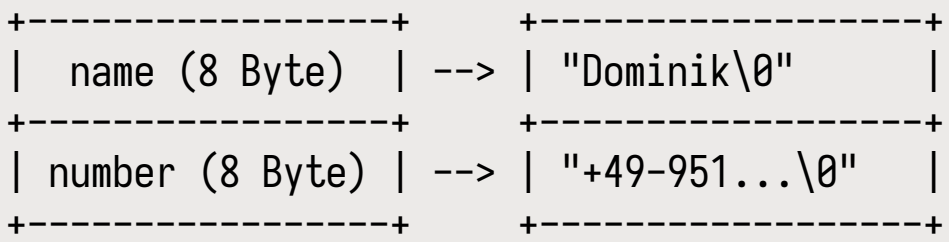
FRAGEN – ANTWORTEN – RÄTSEL
UND KURZE ZUSAMMENFASSUNG

Unterschiede bei Strings

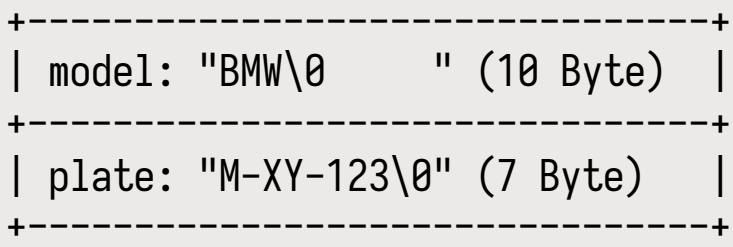
```
// Variante 1: Struktur mit strings (char*)
typedef struct
{
    string name;    // entspricht char*
    string number;  // entspricht char*
}
person;

// Variante 2: Struktur mit char arrays
typedef struct
{
    char model[10];  // fest eingebautes Array
    char plate[7];   // fest eingebautes Array
}
car;
```

Unterschiede bei Strings

Person-Struktur:



Car-Struktur:



```c
// Mit char*:
typedef struct
{
    string name;
    string number;
} person;
```

```c
// Mit Arrays
typedef struct
{
    char model[10];
    char plate[7];
} car;
```

```c
person p;
p.name = "Kurzer Name";      // Funktioniert!
p.name = "Ein sehr sehr sehr langer Name";  // Funktioniert auch!
```

```c
car c;
strcpy(c.model, "BMW");      // Funktioniert!
strcpy(c.model, "Mercedes-Benz");  // Buffer Overflow!
```

typedef ermöglicht das Erstellen von Aliasnamen für Datentypen

Strings (char*) erlauben flexible Längenverwaltung

Besonders nützlich bei der Arbeit mit Strukturen (structs)

Character-Arrays haben feste, vorhersagbare Größe

Strukturnamen nach struct sind optional

Die Wahl zwischen string und char[] hängt vom Anwendungsfall ab