Einführung in die Programmierung

Tutorium 3

Wintersemester 2025/2026

Arbeitsgruppe Systemsoftware
Angewandte Informatik 12

technische universität dortmund

Arbeitsgruppe Systemsoftware

```
div(a, b)                          q := 0, r := a
1  for (i := n − 1; i ≥ 0; --i) :
2     if (r ≥ (b << i)) :
3        r := r − (b << i)
4        q := q | (1 << i)
5  return q, r
```

Code

```
20 for (int i = 7; i >= 0; --i) {
21   if (result[REMAINDER] >= (b << i)) {
22     result[REMAINDER] += -(b << i);
23     result[QUOTIENT] |= (1 << i);
24 } }
```

'klassisches' Datum-Uhrzeit-Format vs. UNIX-Format

'klassisches' Datum-Uhrzeit-Format

- ⊕ Die Darstellung von Datum und Uhrzeit ist sehr einfach, wenn diese genau in dem Datum-Uhrzeit-Format dargestellt werden, indem sie auch dargestellt werden sollen
- ⊖ Das Zählen im 'klassischen' Datum-Uhrzeit-Format erfordert komplexe Zähllogik (Siehe DS1302*)

[*] Überlauf von Einer- und Zehner-Stelle der Jahreszahl: **Jahr-2000-Problem**!

UNIX-Format

Darstellung von Datum und Uhrzeit als Anzahl (Häufig 32⁑-Bit) der Sekunden seit dem 1.1.1970 ^{GMT} 00:00

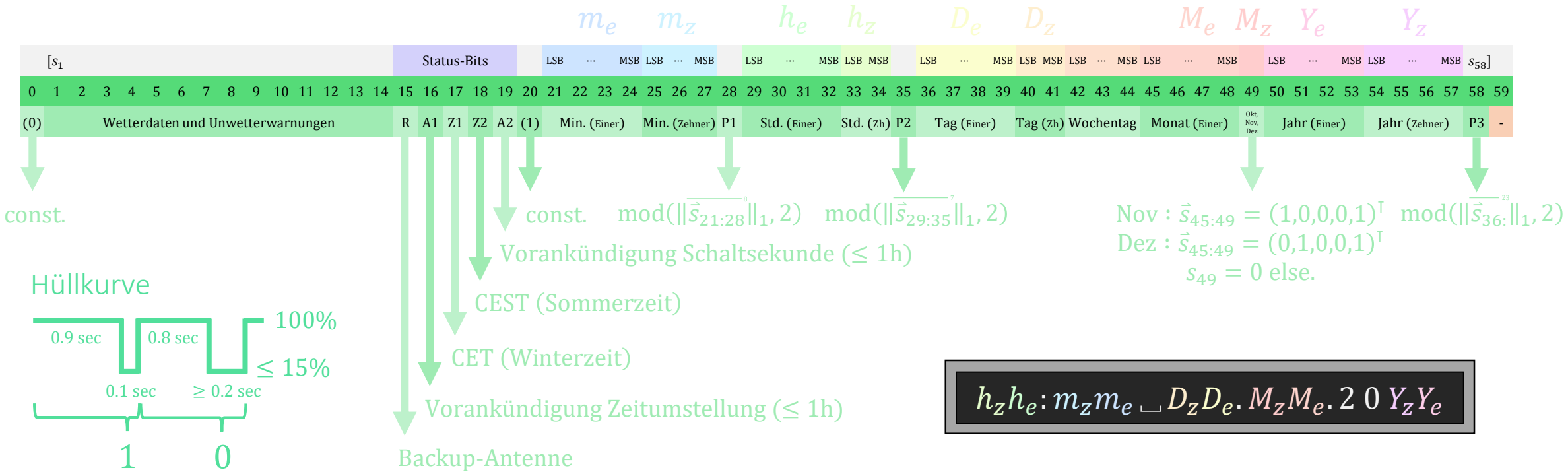
- ⊖ Umwandlung in das Datum-Uhrzeit-Format ist aufwendig (Berücksichtigung von Zeitzone, Schaltjahr, Zeitumstellung, …)
- ⊕ Das Zählen kann durch einen einfachen 32⁑-Bit Zähler implementiert werden

[⁑] Überlauf der UNIX-Zeit: **Jahr-2038-Problem**!

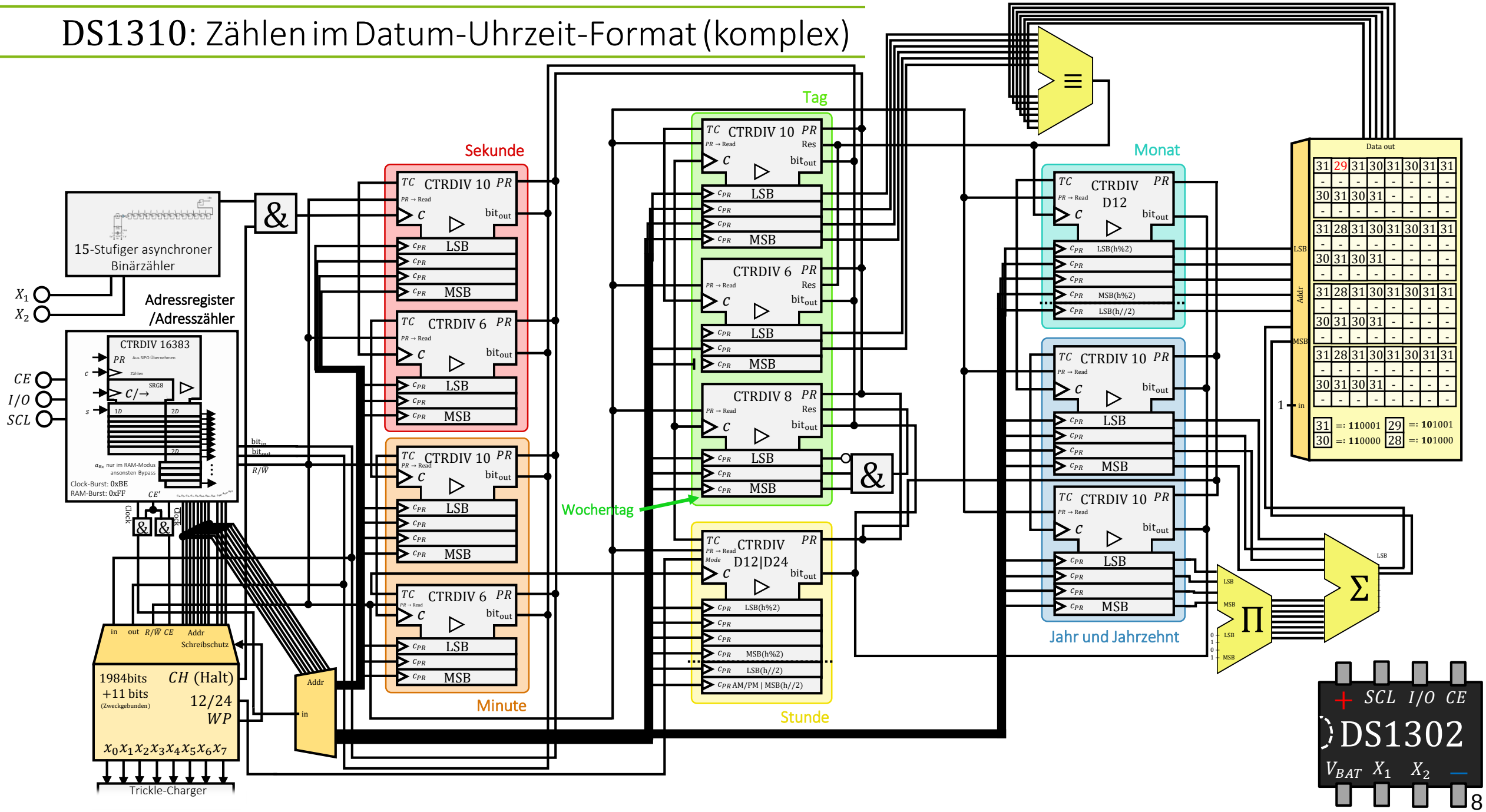
- **Funkuhren** zeichnen sich dadurch aus, dass sie sich von Zeit zu Zeit mit einer zentralen **Atom-Uhr** synchronisieren
- Die Uhrzeit einer solchen sehr genauen Uhr wird dann in serieller Form als AM-Radiowelle übermittelt
- Auf dem europäischen Kontinent kann z.B. das Signal **DCF77** (**77.5 kHz**) empfangen werden, welches neben der Uhrzeit auch noch weitere Informationen (Datum, Unwetterwarnungen, …) enthält

Foto: S. Terfloth

Aufbau des DCF77-Signals



DS1310: Zählen im Datum-Uhrzeit-Format (komplex)



Code

```
23 sec = t % 60; t = t / 60;
24 min = t % 60; t = t / 60;
25 hour = t % 24; t = t / 24;
26 weekday = t % 7;
27
28 while (true){
29   temp = is_leap(year) ? 366 : 365;
30   if (year) {t -= temp; year++; } else break;
31 }
32
33 if (is_leap(year)) { mdays[1] = 29; }
34
35 month = 0;
36 while (t >= mdays[month]){ t -= mdays[month]; month++; }
37
38 day = (int)(t+1)
39
40 printf("%02d.%02d.%d; %s; %02d:%02d:%02d\n\r", ...);
```

Code

```
24 switch (month){
25   case 1:
26   case 3:
27   case 5:
28   case 7:
29   case 8:
30   case 10:
⋮    case 12:last_day = 31; break;
34   case 2:{ /* Check for leap year */ }; break;
⋮    case 4:
44   case 6:
45   case 9:
⋮    case 11: last_day = 30; break;
49   default:
50     day = 99
51     month = 99
52     year = 9999
53 }
```

Code

```
35 if (((year % 100 == 0) && (year % 400 != 0))
36   || (year % 4 != 0)) {
37     last_day = 28
38 } else {
39     last_day = 29
40 }
```

… oder besser: Vorgegebene Funktion `is_leap` nutzen:

```
34 case 2: last_day = is_leap(year) ? 29 : 28; break;
```

Code

```
55 if ((day < 1) || (day > last_day) || (year < 1600) || (year > 2600)){
56   day = 99;
57   month = 99;
58   year = 9999;
59 } else {
60   if (day == last_day) {
61     day = 1;
62     month++;
63   } else { day++; }
64   if (month == 13) {
65     month = 1;
66     year ++;
67   }
68 }
```

Der **Enumerator** mit dem Schlüsselwort **enum** bietet eine Alternative zu Platzhalter-Makros, die mit Fortlaufenden natürlichen Zahlen belegt sind.

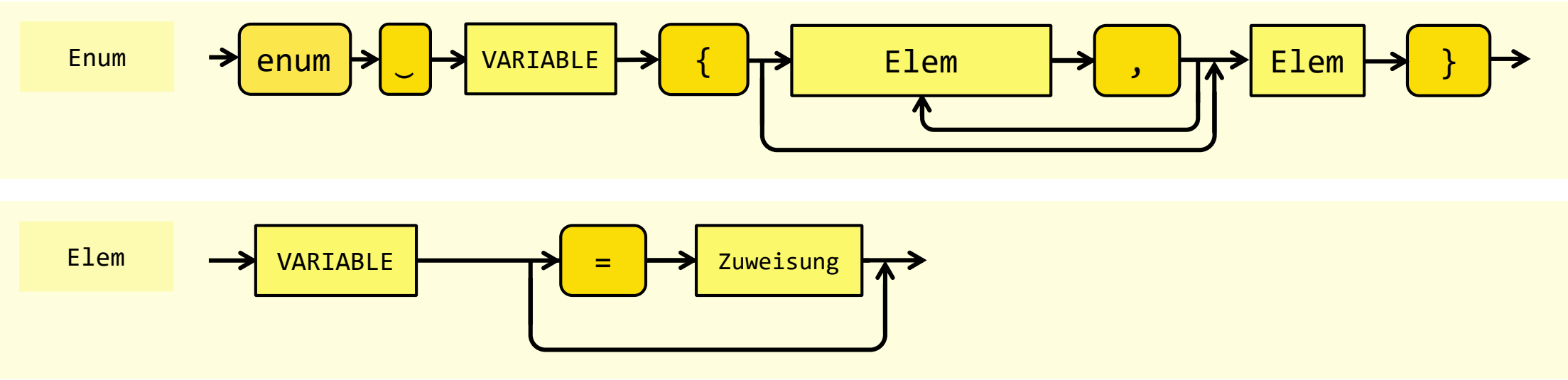
```
#define JAN  1
#define FEB  2
#define MAR  3
#define APR  4
#define MAI  5
        ⋮
#define DEZ  12
```

```
enum Monate {
NUL, JAN, FEB,
MAR, APR, MAI,
JUN, JUL, AUG,
SEP, OKT, NOV,
DEC }
```

Platzhaltermenge

```
enum Monate {
JAN = 1,
FEB, MAR, APR,
MAI, JUN, JUL,
AUG, SEP, OKT,
NOV, DEC }
```

```
enum Monat m = NOV; // = 11
```


Enum
enum
␣
VARIABLE
{
Elem
,
Elem
}
Elem
VARIABLE
=
Zuweisung

Noch Fragen?