5.5 Datenstrukturen II/Aufzählungen

5.5.1 Aufzählungs-Datentypen

In vielen Programmiersprachen gibt es so genannte *Aufzählungs-Datentypen*. Ein Aufzählungs-Datentyp dient dazu, dass man in eine Variable dieses Typs

a) nur bestimmte Werte schreiben darf

b) man auf jeden dieser Werte mit einem Namen zugreifen kann (oder muss)

Wenn man für ein Datum eine Variable `monat` braucht, wäre es gut, wenn man nur Werte zwischen 1 und 12 in `monat` schreiben kann. Alle anderen Werte sollten verboten werden. Und dann wäre es schön, wenn man statt `monat=1` auch `monat=JAN` schreiben kann.

5.5.2 Konstanten-Definition durch den Präprozessor

Die traditionelle Lösung für so etwas funktioniert mit dem C-Präprozessor (im folgenden auch CPP genannt).

In C werden symbolische Konstanten traditionell per define vereinbart:

```
1 #define PI 3.14
```

Der CPP ersetzt dann bei jedem Auftreten der symbolischen Konstanten `PI` dieses Wort durch die wörtliche (literale) Konstante `3.14`. Aus `omega=2*PI*f` wird `omega=2*3.14*f`. Und so kann man für die Monatsnamen 12 Konstanten definieren:

```
1 #define JAN 1
2 #define FEB 2
3 ...
4 #define DEZ 12
```

5.5.3 Konstanten-Definition in C

Konstanten per CPP zu vereinbaren, hat mehrere Nachteile:

a) Eine Konstante, die per CPP vereinbart wurde, ist für Testwerkzeuge nicht sichtbar. Testwerkzeuge arbeiten in der Regel auf derselben Ebene wie der Compiler, und der bekommt vom CPP nur `3.14` zu sehen, nicht `PI`.

b) Eine Konstante, die per CPP vereinbart wurde, unterliegt nicht den C-Regeln für Sichtbarkeit. Sie ist unterhalb ihrer Definition überall sichtbar. Hat man mehrere Konstanten desselben Namens in einem Programm, gibt es deshalb Konflikte.

Statt mit define definiert man Konstanten besser in C selbst. Man verwendet dazu das Schlüsselwort `const`:

```
1 const double PI=3.14;
```

5.5.4 Arten von Konstanten

Es gibt nun verschiedenen Arten von Konstanten:

a) Einzelne Konstanten, wie mathematische Konstanten und Naturkonstanten (s.o.)

b) Bitmusterkonstanten: 0x1=Bit 0, 0x2=Bit 1, 0x4=Bit 2, ...

c) Aufzählungskonstanten: 1=Januar, 2=Februar, ...

Einzelne Konstanten werden durch die const-Vereinbarung vollständig versorgt.

Für Bitmusterkonstanten nimmt man häufig den Präprozessor; es gibt aber in C auch eine Spezialform der Records, die Bitfelder (s.u.), die sich für Bitmuster ebenfalls eignen.

Und für Aufzählungskonstanten gibt es in C die Datenstruktur der *Aufzählung*.

5.5.5 Datenstruktur Aufzählung

Die Vereinbarung einer Aufzählung sieht in C ähnlich aus wie die eines Records:

```
1 enum monatstyp
2 {
3     JAN, FEB, MAE, APR, MAI, JUN, JUL, AUG, SEP, OKT, NOV, DEZ
4 };
```

`JAN` bis `DEZ` sind mögliche Werte, die der Datentyp `enum monatstyp` annehmen kann. Sie sind die *Aufzählungskonstanten* dieses Typs.

Nun kann man von dem Datentyp enum monatstyp Variablen erstellen:

```
1   enum monatstyp geburtsmonat, taufmonat;
```

Einer Variable dieses Datentyps kann nun jeden Wert aus der Liste JAN bis DEZ zuweisen:

```
1   geburtsmonat=APR;
2   taufemonat=MAI;
```

5.5.6 Aufzählung und int

Nun ist es in C (leider?) so, dass jeder Aufzählungsdatentyp vollständig kompatibel zum `int` ist: `JAN` ist eine symbolische Konstante für 0, `FEB` ist eine symbolische Konstante für 1 usw. Die Werte in der Liste sind aufsteigend belegt, beginnend mit 0.

```
1     geburtsmonat=APR;
2     printf("%i\n", geburtsmonat); // gibt: 3
```

Deshalb kann man mit Aufzählungsvariablen recht einfach rechnen und sie vergleichen:

```
1 ++geburtsmonat; // erlaubt
2 if(taufmonat>geburtsmonat) // erlaubt
3 {...}
```

Leider wird zur Laufzeit des Programms nicht kontrolliert, ob eine Aufzählungsvariable einen gültigen Wert (also einen Wert aus der Liste) zugewiesen bekommt:

```
1 geburtsmonat=13; // erlaubt
```

Der Programmierer muss also selbst darauf achten, dass nur mit gültigen Werten gearbeitet werden kann.

5.5.7 Definition einer Aufzählung mit Startwerten

Wenn nun in C schon die Aufzählung kompatibel zu int ist, dann möchte man häufig, dass die Aufzählungskonstanten den "richtigenSZahlenwerten zugeordnet sind. Das kann man bei der Typvereinbarung festlegen:

```
1 enum vorwahltyp { BIELEFELD=521, MINDEN=571, DETMOLD=5231 };
```

Werte ohne Zuordnung werden weiterhin aufsteigend numeriert:

```
enum monatstyp {JAN=1, FEB, MAE, APR, MAI, JUN,
                JUL, AUG, SEP, OKT, NOV, DEZ};
enum jahrtyp { DIESESJAHR=2022, NAECHSTESJAHR, UEBERNAECHSTESJAHR };
```

Dann hat `NAECHSTESJAHR` den Wert 2023 und `UEBERNAECHSTESJAHR` den Wert 2024.

Man kann auch einer Aufzählungskonstanten einen negativen Wert geben:

```
enum fehlertyp { ALLGEMEIN=-1, SPEZIAL=-5, EXTRA=-42 };
```

Ürigens ist auch erlaubt, dass mehrere Aufzählungskonstanten denselben Wert haben:

```
enum autotyp { OPEL=1, VW=37, VOLKSWAGEN=37, FORD=48 };
```

5.5.8 Definition einer Aufzählung mit Begrenzungswerten

Bei einer Aufzählungsvariable muss man als Programmierer selbst dafür sorgen, dass die Variable keinen falschen Wert bekommt. Bei einer zusammenhängenden Liste von Werten (wie `MONTAG=1` bis `SONNTAG=7`) ist das recht einfach: Vor Beginn und nach dem Ende der Liste wird je eine Konstante definiert, die ungültig ist und einen passenden Namen bekommt:

```
enum wochentagtyp{ WT_MIN=0, MO, DI, MI, DO, FR, SA, SO, WT_MAX };
enum wochentyptyp wt;
wt=99;
if(wt<=WT_MIN || wt>=WT_MAX)
    printf("Fehler");
```

Diese Überprüfung funktioniert auch dann zuverlässig, wenn man die Typvereinbarung um weitere Aufzählungskonstanten erweitert (was in diesem Beispiel natürlich nicht zu erwarten ist).

5.5.9 Anwendungsbeispiel Mehrfachauswahl

Weil Aufzählungsdatentypen zu int kompatibel sind, lassen sie sich ohne Probleme in einer Mehrfachauswahl benutzen:

```
enum wochentagtyp{ WT_MIN=0, MO, DI, MI, DO, FR, SA, SO, WT_MAX };
enum wochentagtyp wt=WT_MIN;
do{
    printf("Wochentag eingeben (1=Montag bis 7=Sonntag):");
    scanf("%d", &wt);
}while(wt<=WT_MIN || wt>=WT_MAX);
switch(wt)
{
    case MO: printf("Montag\n");       break;
    case DI: printf("Dienstag\n");     break;
    case MI: printf("Mittwoch\n");     break;
    case DO: printf("Donnerstag\n");   break;
    case FR: printf("Freitag\n");      break;
    case SA: printf("Samstag\n");      break;
    case SO: printf("Sonntag\n");      break;
    default: printf("Fehler!\n");
}
```

5.5.10 Anwendungsbeispiel Index-Array

Aufzählungsdatentypen eignen sich auch als Index eines Arrays. Hier ein Beispiel:

```
enum wochentagtyp{ WT_MIN=0, MO, DI, MI, DO, FR, SA, SO, WT_MAX };
enum wochentagtyp wt=DI;
char wochentagsnamen[9][21]={"Fehler!","Montag","Dienstag","Mittwoch",
                             "Donnerstag","Freitag","Samstag","Sonntag"};
printf("%s\n", wochentagsnamen[2]);   // literale Konstante im Index
printf("%s\n", wochentagsnamen[DI]);  // Aufzaehlungskonstante im Index
printf("%s\n", wochentagsnamen[wt]);  // Aufzaehlungsvariable im Index
```

In den neueren C-Versionen ist es erlaubt, ein Array so zu initialisieren, dass man den Array-Index angibt:

```
char wochentagsnamen[9][21]={[0]="Fehler!",[1]="Montag",[2]="Dienstag",
        [3]="Mittwoch"}; // usw.
```

Hier bietet es sich an, dass man den Array-Index als Aufzählungskonstante angibt:

```
char wochentagsnamen[9][21]={[WT_MIN]="Fehler!", [MO]="Montag",
                             [DI]="Dienstag",[MI]="Mittwoch"}; // usw.
```

Das obige Beispiel sieht dann so aus:

```
    enum wochentagtyp{ WT_MIN=0, MO, DI, MI, DO, FR, SA, SO, WT_MAX };
    enum wochentagtyp wt=WT_MIN;
    char wochentagsnamen[9][21]={"Fehler!","Montag","Dienstag",
        "Mittwoch","Donnerstag","Freitag","Samstag","Sonntag"};
    do{
        printf("Wochentag eingeben (1=Montag bis 7=Sonntag):");
        scanf("%d", &wt);
    }while(wt<=WT_MIN || wt>=WT_MAX);

    printf("%s\n", wochentagsnamen[wt]);
```

Man kann auch die Eingabe von Aufzählungsvariablen über die Liste vornehmen:

```
    enum wochentagtyp{ WT_MIN=0, MO, DI, MI, DO, FR, SA, SO, WT_MAX };
    enum wochentagtyp wt=WT_MIN;
    char wochentagsnamen[9][21]={"Fehler!","Montag","Dienstag",
        "Mittwoch","Donnerstag","Freitag","Samstag","Sonntag"};
    char wtagname[21]="";
    printf("Wochentag eingeben:");
    scanf("%s", wtagname);
    while(getchar()!='\n'){}
    for(wt=WT_MIN+1; wt<WT_MAX; ++wt)
    {
        if(strcmp(wtagname, wochentagsname[wt])==0)
            break;
    }
    if(wt==WT_MAX)
        printf("Fehler!");
    else
        printf("%s\n", wochentagsnamen[wt]);
```