5.8 Datenstrukturen II/Datenstrukturen dokumentieren

5.8.1 Auf der Suche nach Diagramm für Daten

Mit den Datenstrukturen Zeiger, Array, Record und Union kann man beliebig umfangreiche Datentypen und Variablen erzeugen. Aber wie stellt man sie so dar, dass auch Nicht-C-Programmierer sie verstehen können?

Zur Erinnerung: Mit dem Struktogramm (=Nassi-Shneiderman-Diagramm, DIN 66261), dem Flussdiagramm (=Nassi-Shneiderman-Diagramm, DIN 66261) und dem Jackson-Diagramm können wir beliebige Programmstrukturen mehr oder weniger gut darstellen. Aber was ist mit Daten?

Für die Darstellung von Daten kommt man mit dem Struktogramm nicht so recht weiter. Zwar gibt es mittlerweile sogenannte *Erweiterte Struktogramme*, in denen auch die Vereinbarung von Variablen dargestellt wird. Qber die Struktur der Variablen wird dadurch nicht deutlich.

Bei Flussdiagramm gibt es zum Programm-Fluss-Diagramm (bekannt als Programm-Ablaufplan PAP) noch das Daten-Fluss-Diagramm. Aber dort kann man nur ablesen, von woher nach wohin die Daten fließen. Man kann aber nicht zeigen, wie die Daten aussehen.

Bleibt noch das Jackson-Diagramm: Auch hier haben wir bisher nur Programmabläufe dargestellt. Aber Jackson hat einen Zusammenhang festgestellt: Wie man die Daten anlegt, so verarbeitet man sie meistens auch. Das heißt: Wenn ich in einem Record drei Komponenten (nennen wir sie x, y und z) habe, dann verarbeite ich sie in der Regel nacheinander: Erst x, dann y, dann z.

```
1  struct beispieltyp{
2      int x;
3      double y;
4      char z;
5  };
6  struct beispieltyp beispiel={3,4.5,'x'};
7  void verarbeite(struct beispieltyp bsp)
8  {
9      printf("x=%i\n", bsp.x);
10     printf("y=%lf\n", bsp.y);
11     printf("z='%c'\n", bsp.z);
12 }
```

Das Record enthält drei Komponenten; die Funktion enthält eine Sequenz (=Folge) von drei Anweisungen. Merke: Zur Datenstruktur Record gehört meisten die Programmstruktur Sequenz.

Genauso ist es mit der Datenstruktur Array:

```
1  double beispiel[8]={1.,2.,3.,4.,5.,6.,7.,8.};
2  void verarbeite(double bsp[], int anzahl)
3  {
4      for(int lauf=0; lauf<anzahl; ++lauf)
5      {
6          printf("bsp[%i]=%lf\n", lauf, bsp[lauf]);
7      }
8  }
```

Das Array enthält Elemente vom Typ double; die Funktion enthält eine Zählschleife, die jedes Element nacheinander verarbeitet. Merke: Zur Datenstruktur Array gehört meistens die Programmstruktur Schleife.

Mit der Datenstruktur Union ist es etwas schwierig, weil man einen Diskriminator (=Unterscheider) zur Auswahl einer der Alternativen braucht. Den muss man der Funktion mitgeben (wie oben beim Array die Anzahl):

```
1  union beispieltyp{
```

```
2     int x;
3     double y;
4  };
5  union beispieltyp beispiel={3}; // nullte Alternative: 3
6  int diskriminator=0; // waehle nullte Alternative
7  void uverarbeite(union beispieltyp bsp, int diskri)
8  {
9     if(diskri==0)
10       printf("x=%i\n", bsp.x);
11    else
12       printf("y=%lf\n", bsp.y);
13 }
```

Die Union enthält zwei Alternativen; die Funktion eine Verzweigung, die entweder die eine oder die andere Alternative verarbeitet. Merke: Zur Datenstruktur Union gehört oft die Programmstruktur Verzweigung (bei zwei Alternativen) oder Mehrfachauswahl (bei mehr als zwei).

Wozu hilft einem das bei der Suche nach einem Diagramm? Jackson nutzt diesen Zusammenhang zwischen Daten- und Programmstrukturen aus: Er verwendet dasselbe Diagramm mit denselben Symbolen für Daten *und* für Programme.

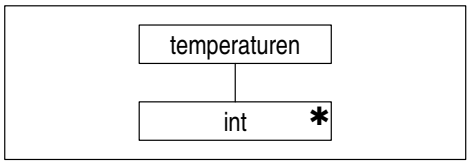
5.8.2 Grundelemente des Jackson-Diagramms

5.8.2.1 Das Array im Jackson-Diagramm

Der erste Grundbaustein ist das Array:

```
1     int temperaturen[10]; /* Array */
```

Das wird im Jackson-Diagramm so dargestellt:



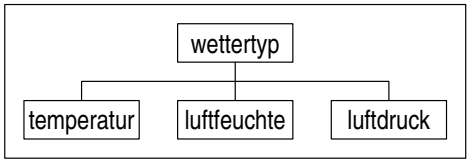
Der Stern ist hier ein Symbol für Wiederholung.

5.8.2.2 Der Record im Jackson-Diagramm

Der zweite Grundbaustein ist der Record:

```
1     struct wettertyp
2     {
3        int temperatur;
4        double luftfeuchte;
5        unsigned int luftdruck;
6     };
```

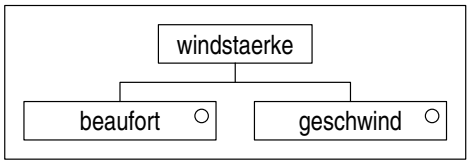
Er wird im Jackson-Diagramm so dargestellt:



5.8.2.3 Die Union im Jackson-Diagramm Der dritte und letzte Grundbaustein ist die Union:

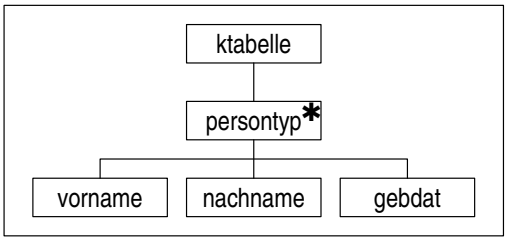
```
1  union windstaerke
2  {
3     int beaufort;      // windstaerke 9=Sturm usw.
4     double geschwind; // geschwindigkeit in km/h
5  };
```

Sie wird im Jackson-Diagramm so dargestellt:



5.8.3 Konstruktion

Die genannten Grundbausteine kann man beliebig zu ganzen Datenhierarchien zusammenfügen. Man erhält dann zu jedem Datentyp bzw. zu jeder Variablen ein entsprechendes Baumdiagramm. Hier ist das Baumdiagramm zur Variablen `ktabelle` aus dem weiter oben erstellten Programm:



Hier nochmal im Vergleich die Datenstruktur in C-Schreibweise:

```
1  struct persontyp
2  {
3     char nachname[ANZAHL+1];
4     char vorname[ANZAHL+1];
5     long gebdat;
6  };
7  struct persontyp persondb[MAXANZ];
```

5.8.4 Jenseits von Daten und Funktionen

Bei der modularen Programmierung werden Daten und die jeweils dazugehörigen Funktionen in einem Modul zusammengefasst. So ein Modul stellt man oft in Form eines Topfes dar, der die betreffenden Daten und Funktionen enthält. Und dieser Topf (Modul) tauscht noch Daten mit anderen Modulen aus. Struktogramm und Jackson-Diagramm helfen an der Stelle nicht weiter, wenn es um diese Kommunikation (=Daten- und Programmfluss) zwischen den Modulen geht.

Ähnlich ist es bei der objektorientierten Programmierung. Hier kann es mehrere Module derselben Art geben (die Module heißen dann Objekte und die Art nennt man Klasse). Man kann mit den bisherigen Diagrammen noch den Innenaufbau eines Objekts beschreiben, aber die Kommunikation (=Daten- und Programmfluss) zwischen Objekten derselben oder verschiedener Klassen ist nicht anschaulich darstellbar.

Hier helfen die verschiedenen Diagramme der Sprache UML (=unified modelling language) weiter, die für fast jede Gelegenheit eine eigene Möglichkeit bieten.