Datenstrukturen (Wertebereiche für strukturierte Werte)

Bei der Behandlung der Sprache LISP haben wir Binärbäume als Argumente und Ergebnisse von Funktionen kennengelernt. Man kann in LISP solche Bäume als Werte betrachten, die man benannten Variablen zuordnen kann. So kann man nicht nur elementare Werte zuordnen, beispielsweise

```
(SETQ  x  5)
```

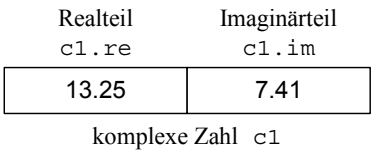
sondern man kann einer Variablen auch einen strukturierten Wert zuordnen:

```
(SETQ  x  '(2  3  (4  5) )  )
```

In LISP gibt es nicht die Möglichkeit, die Nutzung einer Variablen derart einzuschränken, dass man diese Variable nur für eine bestimmte Art von Werten zulässt, beispielsweise nur für `INTEGER`-Zahlen oder für Binärbäume mit 3 Knoten. Im Gegensatz hierzu ist C eine getypte Sprache. Dort gibt es die sogenannten Vereinbarungen, in denen festgelegt wird, dass einer bestimmten benannten Variable nur Exemplare aus einem ganz bestimmten Wertebereich zugeordnet werden dürfen.

Bisher haben wir nur Wertebereiche für „atomare Werte“ kennengelernt: `INTEGER, REAL oder FLOAT, CHAR, BOOLEAN.` Mit der Kennzeichnung *atomar* weisen wir auf den Sachverhalt hin, dass ein solcher Wert als informationelles Individuum zu betrachten ist, bei dem es sinnlos wäre, nach einer inneren Struktur zu fragen.

Aus atomaren Werten kann man aber strukturierte Werte zusammenbauen. So kann man beispielsweise aus zwei `REAL`-Zahlen eine komplexe Zahl zusammenbauen:



komplexe Zahl `c1`

Um diesen strukturierten Typ zu definieren, muss man in C schreiben

```
typedef   struct { float re; float im; }  complex;
```

Man kann sich solch eine zusammengebaute Information als Paket vorstellen, in welchem mehrere Teile enthalten sind. Manchmal will man ein solches Paket als Ganzes ansprechen, nämlich immer dann, wenn man es transportieren und speichern will. Man stelle sich vor, man will ein Paket, welches auf dem Tisch liegt, in den Schrank stellen. Dann interessiert es nicht, wie viele unterschiedliche Teile innerhalb des Pakets unterscheidbar sind. Wenn man dagegen in den Bereich der Verarbeitung kommt, muss man auf die einzelnen Teile innerhalb des Pakets zugreifen können, denn nur diese sind Gegenstand der expliziten Verarbeitung. Man denke an ein Paket, worin die Zutaten zum Backen eines Kuchens enthalten sind. Solange man dieses Paket nur verschickt und speichert, interessieren die Zutaten nicht. Wenn man dagegen den Kuchen backen will, muss man das Paket aufmachen und die einzelnen Bestandteile getrennt benutzen.

Mit der Vereinbarung `complex: c1, c2;`

wird Speicherplatz für zwei Pakete reserviert. Diese Vereinbarung hat den gleichen strukturellen Aufbau wie die Vereinbarung `int j1, j2;`
mit der zwei Speicherplätze für Integerzahlen reserviert werden. Mit den Bezeichnungen

`c1.re` `c1.im` `c2.re` `c2.im`

kann man die einzelnen Bestandteile innerhalb der Pakete identifizieren.

Im Falle der Sprache LISP würden wir solche Pakete als Listen gestalten. Ein Paket vom Typ `complex` wäre also eine Liste mit 2 Positionen. Da man in LISP den einzelnen Positionen in der Liste keine Namen geben kann, muss man die Identifikation der einzelnen Listenpositionen durch Verwendung der Funktionen CAR und CDR realisieren. Wir nehmen an, dass wir den Realteil in die 1. Position und den Imaginärteil in die 2. Position einer Liste gepackt haben, die wir als Wert der Variablen c1 zugewiesen haben. Dann wird durch den Ausdruck `(CAR c1)` der Realteil im Paket c1 umschrieben, den man in C mit `c1.re` identifizieren konnte. Entsprechend wird durch `(CADR c1)` der Imaginärteil identifiziert, den man in C mit `c1.im` identifizieren kann. Man erkennt hier, dass die Schreibweise in C für das Verständnis günstiger ist.

****

Bei den strukturierten Werten bzw. Wertebereichen muss man zwischen geschlossenen und offenen Strukturen unterscheiden. Eine Struktur liegt immer vor, wenn die jeweiligen Werte aus Komponenten zusammengesetzt sind und man den gesamten Wert nicht atomar verstehen kann. Bei einer geschlossenen Struktur ist die Anzahl der Komponenten festgelegt, wogegen die Anzahl der Komponenten kein Merkmal einer offenen Struktur sein kann. So ist beispielsweise die Struktur `complex` eine geschlossene Struktur, und der Wertebereich „Vektor“ ohne Angabe der Komponentenanzahl ist eine offene Struktur.

Aus der Mathematik kennt man die Strukturen Vektoren und Matrizen. Sowohl die Vektoren als auch die Matrizen werden programmiersprachlich als `ARRAY` bezeichnet. Unter einem `ARRAY` versteht man eine ein- oder mehrdimensionale Matrix von Elementen gleichen Typs. Die Elemente müssen nicht Zahlen sein. Es können auch andere informationelle Elemente sein. Die Elemente eines `ARRAY`s dürfen sogar selbst wieder aus einem strukturierten Wertebereich stammen, wobei lediglich die Bedingung gilt, dass alle Elemente eines `ARRAY`s aus dem gleichen Wertebereich stammen.

In der Vereinbarung eines `ARRAY`s müssen drei unterschiedliche Dinge angegeben werden: Der Name der Variablen bzw. des Behälters, denen Werte vom `ARRAY`-Typ zugeordnet werden; die Dimension des `ARRAY`s und die Anzahl der `ARRAY`-Positionen in jeder Dimension; der Typ der einzelnen Elemente im `ARRAY`.

Beispiel: `Koeffizientenmatrix ARRAY [ 1:5, 1:10 ] OF REAL`

In diesem Beispiel handelt es sich um eine zweidimensionale Matrix mit 5 Zeilen und 10 Spalten, deren 50 Elemente alle vom Typ `REAL` sind. Jedes dieser 50 Elemente kann eindeu-

tig über die Angabe des Paares [Zeilennummer, Spaltennummer] identifiziert werden, beispielsweise `Koeffizientenmatrix[2, 7]`.

Nun betrachten wir offene Strukturen für informationelle Werte. Diese Strukturen stellt man sich als eine Menge von Zellen vor, in denen elementare Werte stehen, und Linien, die jeweils zwei solcher Zellen strukturell verbinden. Das typische Beispiel für eine solche Struktur ist ein Binärbaum. Bei der Sprache LISP sind alle Werte Binärbäume und deshalb muss man im Programm nicht explizit darauf hinweisen, dass die eingeführten benannten Speicherzellen Behälter für Binärbäume sein sollen. Sie können ja in der Sprache LISP gar nichts anderes sein. Anders dagegen liegt der Fall in Programmiersprachen, wo man für jede Speicherzelle angeben muss, welche Art von Werten sie aufnehmen können soll. Und da stehen wir nun vor dem Problem, in einer Sprache, die den Begriff des Binärbaumes nicht schon auf Grund der Sprachkonstruktion kennt, den Begriff formulieren zu müssen. Die Sprache kennt typischerweise die elementaren Wertebereiche `INTEGER, REAL, BOOLEAN` und `CHARACTER`. Damit man in einer solchen Sprache auch strukturierte Wertebereiche beschreiben kann, muss die Sprache noch einen bestimmten weiteren elementaren Wertebereich per Konstruktion kennen. Dies ist der Wertebereich der sogenannten Zeiger oder `Pointer`. Es handelt sich um interne Ortsidentifikatoren. Wir haben bereits die Möglichkeit kennen gelernt, Orte durch Namen zu bezeichnen. Sonst hätten wir ja gar keine Zuweisungsanweisungen formulieren können, in denen verlangt wird, dass ein bestimmter Wert an einen bestimmten Ort gespeichert wird.

Die grundlegende Idee bei der Einführung strukturierter Wertebereiche besteht darin, dass man gar keinen Ort einführt, an dem der strukturierte Wert liegt, sondern dass man den strukturierten Wert verteilt speichert. Verteilte Speicherung bedeutet, dass man jeder Komponente des strukturierten Wertes ihren eigenen Ort zuordnet und dass man bei den Komponenten auch noch die Information hinterlegt, wie sie mit anderen Komponenten verbunden sein müssen, damit sich insgesamt der strukturierte Wert ergibt. In Bild 40 ist veranschaulicht, wie man sich die verteilte Speicherung eines Binärbaums vorstellen muss, bei dem nur die Blätter beschriftet sind.

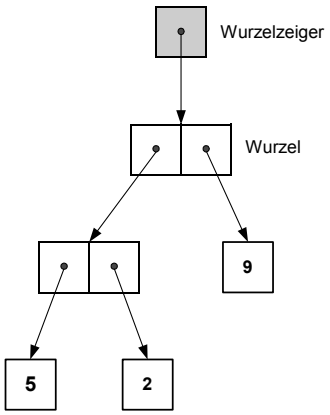


Bild 40 Speicherung eines Binärbaums als sog. Pointer-Struktur

Im Falle dieses Beispiels gibt es 6 unterschiedliche Orte, an denen Informationen über den Baum abgelegt sind. Da der Binärbaum erst als Ergebnis einer Programmausführung entstanden sein kann, können die Orte für die Baumknoten nicht vom Programmierer benannt worden sein. Denn es hätte ja auch sein können, dass als Ergebnis kein Baum mit 5 Knoten, son-

dern einer mit 25 Knoten entstanden wäre. Der einzige Ort, der vom Programmierer benannt werden konnte, ist der Ort, in dem der Wurzelzeiger steht, denn diesen Ort benötigt man unabhängig von der tatsächlichen Gestalt des zu speichernden Binärbaums.

Wenn man offen strukturierte Werte im Speicher unterbringen muss, benötigt man eine Speicherverwaltung. Bei dieser Speicherverwaltung kann Speicherplatz angefordert werden und es kann Speicherplatz zurückgegeben werden. Bei der Anforderung von Speicherplatz muss die Speicherverwaltung einen Ortsidentifikator zurückliefern, damit der nun reservierte Speicherplatz benutzt werden kann. Umgekehrt muss der Speicherverwaltung ein Ortsidentifikator mitgeteilt werden, wenn sie eine Speicherreservierung aufheben soll. Die Ortsidentifikatoren, die als Eingabe oder Ausgabe der Speicherverwaltung vorkommen, sind außer für die interne Identifikation von Speicherorten für nichts zu benutzen. Der Programmierer weiß nichts über die Form, in der solche Ortsidentifikatoren maschinenintern gespeichert werden. Während die Namen, die der Programmierer selbst als Ortsidentifikatoren einführt, selbstverständlich peripher eingegeben und ausgegeben werden können, tauchen die internen Ortsidentifikatoren nicht in der Peripherie auf.

****