Signatur

Wir betrachten nun das Thema Prozedurschichtung. Solange man nur Programme schreibt, deren Größe eine DIN A4 Seite nicht übersteigt, gibt es keinen Grund, sich über eine besondere Strukturierung von Programmen Gedanken zu machen. Dies wird aber selbstverständlich dringend notwendig, wenn Software entwickelt werden muss, deren Größe Tausende von DIN A4 Seiten ausmacht. Dann muss man gegeneinander abgrenzbare Programmteile haben, die von unterschiedlichen Entwicklern unabhängig voneinander gestaltet werden können. Die Grundidee, die hinter einer Prozedurschichtung steht, ist die Vorstellung des Verhältnisses zwischen einem Auftraggeber und einem Auftragnehmer. Der Auftraggeber ist für die Erledigung einer umfassenden Aufgabe zuständig, aber er lässt Teilaufgaben von anderen ausführen. Hierzu bedarf es einer klaren Schnittstelle zwischen dem Auftraggeber und dem Auftragnehmer. Im Falle der Prozedurschichtung nennt man diese Schnittstelle Signatur. Die Signatur umfasst 3 Bestandteile:

(1) die Identifikation des Auftragnehmers in Form des Prozedurnamens,
(2) die Liste der Informationen, die vom Auftraggeber an den Auftragnehmer gegeben werden müssen, damit die Aufgabe erledigt werden kann
(die sog. formalen Argumente),
(3) der Typ der Information, die der Auftragnehmer als Ergebnis seiner Arbeit zurückliefern soll.

Als Beispiel eines Auftragnehmers betrachten wir den Wurzelzieher, der aus einem Radikanden r die k-te Wurzel ziehen kann. Seine Signatur soll wie folgt ausssehen:

```
REAL: root( INTEGER: grad; REAL: radikand )
```

Als Ergebnis liefert der Wurzelberechner eine `REAL`-Zahl zurück. Als Argument braucht er die beiden Zahlen grad und radikand.

Eine Signatur ist zu betrachten wie die Definition einer Steckdose, die den Nutzern bekannt sein muss, damit sie ihre Stecker so gestalten können, dass sie in die Steckdose passen.

Kommunikation zwischen rufender und gerufener Prozedur

Im betrachteten Beispiel könnte das Programm eines Nutzers von `root` den folgenden Abschnitt enthalten:

```
x:= 12.5 * 38.4;
y:= root(4, x);
```

Die Speicherzellen x und y gehören dem Auftraggeber, der dem Auftragnehmer keinen Zugang zu diesen Zellen gewähren will. Wir können uns vorstellen, x und y seien die Namen von Kochtöpfen, und der Koch will niemanden in seine Küche lassen. Da zwischen ihm und dem Auftragnehmer nur Information und keine Materie fließen soll, kann die gesamte Kommunikation akustisch durch die geschlossene Küchentüre hindurch erfolgen:

Auftraggeber: Hallo `root`, berechne mir das Ergebnis für den Fall, dass grad den Wert 4 hat und der radikand den Wert ... (Hier teilt der Auftraggeber den Inhalt des Topfes x mit.)
Auftragnehmer: Hallo Auftraggeber, das Ergebnis ist ...

Gesprochene Information ist dem Wesen nach flüchtig, d.h. die Information ist verloren, wenn sie der Hörer nicht speichert. Der Auftragnehmer speichert die gehörten Argumentwerte in seinen Speicherzellen grad und radikand, und der Auftraggeber speichert das gehörte Ergebnis in seiner Speicherzelle y.

Diese Art der Kommunikation zwischen dem Auftraggeber und dem Auftragnehmer, die wir uns als ausschließlich akustische Kommunikation vorstellen können, wird „Call by Value“ genannt. Daneben gibt es aber auch noch den Fall, dass der Auftraggeber den Auftragnehmer in seine Küche hereinkommen lässt und ihm Zugang zu „seinen Kochtöpfen“ gewährt. Zur Veranschaulichung denke man an Aufträge, welche die Anwesenheit des Auftragnehmers in der Küche unbedingt erfordern – beispielsweise „Rühren Sie so lange in der Pfanne, bis die Zwiebeln gelb und glasig geworden sind.“

Es muss sich also um Aufträge handeln, bei denen der Auftragnehmer den Zustand in bestimmten Speichern, die dem Auftraggeber gehören, gezielt verändern soll. Als Beispiel aus der Programmierung betrachten wir den Auftrag, n Personennamen, die in beliebiger Reihenfolge als Liste in einem Speicher des Auftraggebers stehen, alphabetisch zu ordnen und die neue Liste an die gleiche Stelle zu schreiben, wo vorher die alte Liste stand. Da genügt es nicht, dass der Auftraggeber dem Auftragnehmer die alte Liste durch die geschlossene Türe hindurch vorliest und der Auftragnehmer ihm anschließend die sortierte Liste als Ergebnis ebenfalls akustisch mitteilt.

Für die rein akustische Kopplung, also für den Fall des Call by Value würde die Signatur lauten:

```
LISTE: sortieren(LISTE: argumentliste)
```

Und beim Auftraggeber könnte die Erteilung des Auftrags im Programm die folgende Form haben:

```
meineliste:= sortieren(meineliste);
```

Wenn dagegen der Auftragnehmer Zugang zum Listenspeicher des Auftraggebers haben soll, muss die Signatur lauten:

```
VOID sortieren(LISTE: *argumentlistenpointer)
```

`argumentlistenpointer` ist eine Information, die dem Auftragnehmer Zugang zum Speicher der Argumentliste gibt. Durch den vorangestellten Stern wird in der Programmiersprache C ausgedrückt, dass man nicht den Pointer selbst, sondern die Zelle bzw. deren Inhalt meint, auf die der Pointer zeigt. Durch den Ausdruck

```
LISTE: *argumentlistenpointer
```

wird also ausgesagt, dass die Zelle, auf die der Pointer zeigt, eine Zelle sein soll, die nur mit einem Wert belegt werden darf, der aus dem Wertebereich `LISTE` stammt.

Beim Auftraggeber könnte nun die Erteilung eines Auftrags die folgende Form haben:

```
sortieren(&meineliste);
```

Durch das vorangestellte Zeichen `&` wird in der Programmiersprache C ausgedrückt, dass nicht der Inhalt der Zelle `meineliste` gemeint ist, sondern ein Pointer, der auf die Zelle zeigt.

Diese Art der Kommunikation, bei der ein Auftraggeber dem Auftragnehmer einen Pointer übergibt, der dem Auftragnehmer den Zugang zu Speicherzellen gewährt, die dem Auftraggeber gehören, wird „Call by Reference“ genannt.

Die Variablen, die in der Klammer einer Signatur stehen, werden formale Variable genannt. Diese Bezeichnung weist darauf hin, dass der Name einer solchen Variablen so lange keinen konkreten Speicherplatz identifiziert, wie die Prozedur noch nicht aufgerufen wurde, d.h. solange noch kein Auftrag erteilt wurde.

Mit jedem Aufruf der Prozedur wird zu jeder formalen Variablen ein konkreter Speicherplatz festgelegt. Wir werden später sehen, dass im Falle der Rekursion einer formalen Variablen mehrere Speicherplätze zugeordnet werden können.

Wenn eine Prozedur mehrere formale Variable hat, ist es möglich, dass bei einem Prozeduraufruf einige dieser Variablen mit Call by Value und andere mit Call by Reference angesprochen werden.

Schichtungsplan

Die Auftraggeber- Auftragnehmerbeziehung kann man grafisch sehr schön in einem sogenannten Schichtungsplan darstellen. Im Zusammenhang mit der 4. Übung wurde der folgende Schichtungsplan vorgegeben:

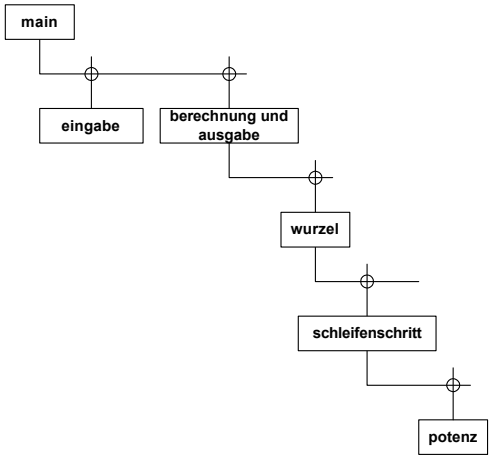

Bild 36 Beispiel eines Schichtungsplans

Hinter dieser grafischen Darstellung stehen die folgenden Überlegungen:

Eine Schichtung ist eine spezielle zweistellige Relation. Ganz allgemein ist eine Relation eine Teilmenge eines karthesischen Produkts. Als Beispiel betrachten wir die zweistellige Relation

$$\text{Wohnrelation} \subseteq \text{Personenmenge} \times \text{Ortsmenge}$$

Eine solche Relation kann, falls die Mengen endlich sind, als Matrix veranschaulicht werden, wie dies in Bild 37 gezeigt ist.

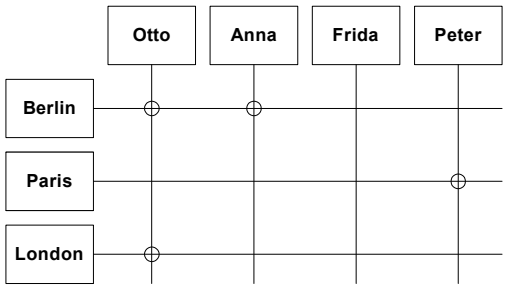

Bild 37 Matrixdarstellung einer zweistelligen Relation

Eine Schichtung ist eine quadratische Relation, denn es wird nur eine einzige Menge, die Menge der Prozeduren, betrachtet, wobei aber jedes Mengenelement in zwei Rollen vorkommen kann, in der Rolle des Rufenden und in der Rolle des Gerufenen. Die Schichtungsstruktur, die im Bild 36 bereits in der typischen Schichtungsform gezeigt ist, kann selbstverständlich auch als allgemeine quadratische Relation dargestellt werden. Sie sieht dann so aus, wie es in Bild 38 gezeigt ist.

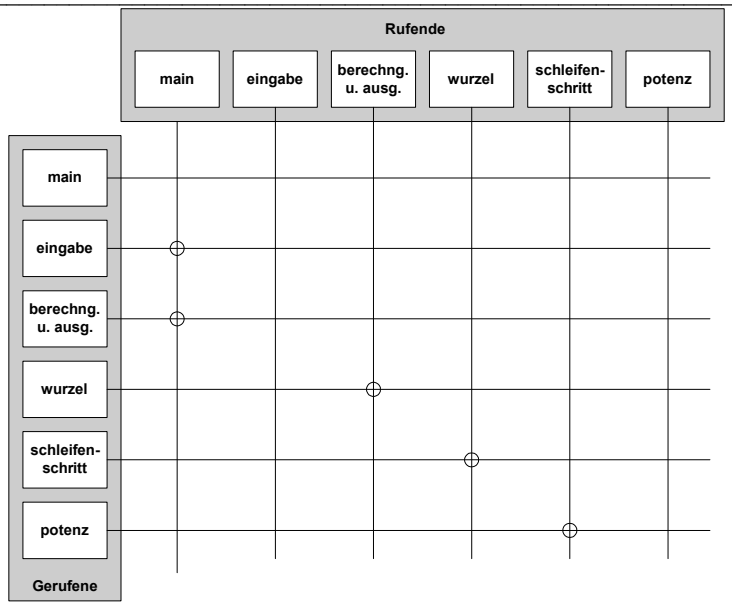

Bild 38 Quadratische Relationsmatrix zur Schichtung aus Bild 36

Während man in Bild 36 unmittelbar sieht, wer Auftraggeber von wem ist – denn die Auftraggeber stehen oben, die Auftragnehmer unten, sieht man dies in Bild 38 nicht. Dennoch ist Bild 38 eine formal äquivalente Darstellung zu Bild 36. In Bild 36 kommen 5 Relationskringel vor, und diese findet man in Bild 38 wieder.

Dass eine Teilmenge eines karthesischen Produkts Relation genannt wird, ist in der folgenden Anschauung begründet: Das karthesische Produkt entspricht der Menge aller Einsetzungskombinationen zu einer gegebenen Aussageform. Eine Aussageform hat die Struktur einer Aussage, welche die Existenz bestimmter Beziehungen zwischen Individuen bestimmter Art ausdrückt. Diese Individuen sind aber in der Aussageform nur als Variable und nicht als konkrete Fälle bezeichnet. Nur diejenigen Elemente aus dem karthesischen Produkt, deren Einsetzung in die Aussageform daraus eine wahre Aussage machen, werden in die Relation übernommen. Die Aussageformen für die beiden in den Bildern 37 und 38 dargestellten Relationen lauten:

Person *p* hat eine Wohnung in der Stadt *s*.

In der Prozedur *p1* steht ein Aufruf der Prozedur *p2*.

Schichtungssemantik

Ich habe gesagt, dass durch die Schichtung eine Auftraggeber/ Auftragnehmerbeziehung erfasst wird. Nun will ich Ihnen am Beispiel der Schichtung aus Bild 36 bzw. 39 zeigen, dass es zwei völlig unterschiedliche Gründe für eine Prozedurschichtung gibt. Wir vergleichen hierzu die beiden Prozeduren `potenz` und `schleifenschritt`.

Die Prozedur `potenz` musste nur geschrieben werden, weil im Unterschied zu den arithmetischen Operatoren + und – der Potenzoperator nicht bereits Teil der Sprache C ist. In Bild 39 liegt die Potenzprozedur zwischen zwei waagerechten Linien. Oberhalb einer solchen waagerechten Linie steht jeweils die Funktionalität einer Programmiersprache zur Benutzung zur Verfügung. Die unterste Linie entspricht somit der Bereitstellung der Funktionalität der Sprache C. Die darüber liegende Linie entspricht der um die Potenzfunktion erweiterten Funktionalität der Sprache C. Man kann eine solche Funktionalität vollständig erklären und verstehen, ohne die Prozeduren, die oberhalb der jeweiligen Linie liegen, zu betrachten.

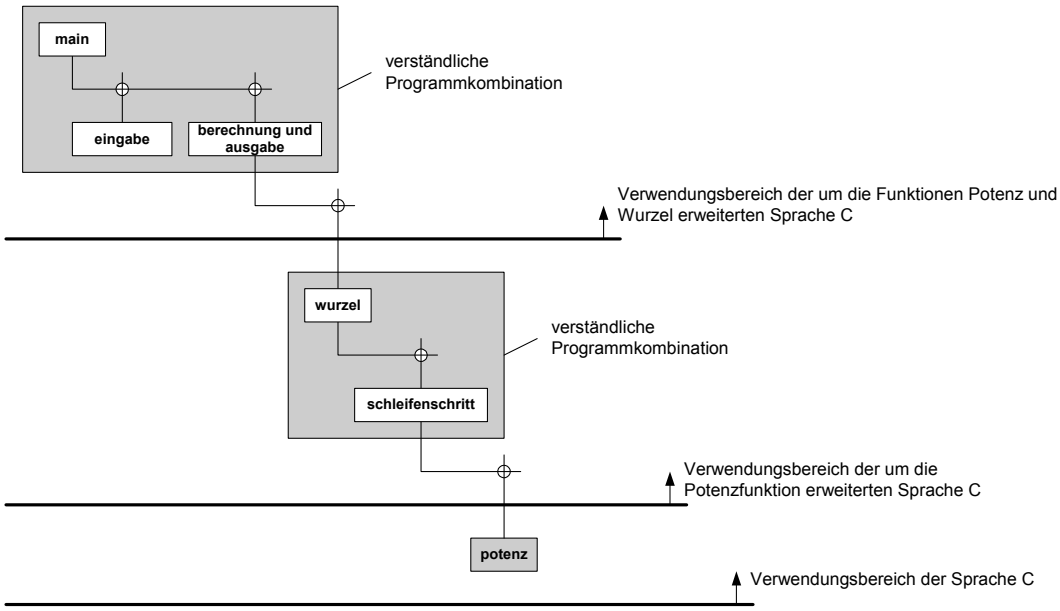

Bild 39 Beispiel zur Veranschaulichung der zwei unterschiedlichen Schichtungsgründe

Ganz anders liegt der Fall bei der Prozedur `schleifenschritt`. Es wäre sinnlos, unmittelbar oberhalb der Prozedur `schleifenschritt` eine waagerechte Linie einzutragen, welche die Verfügbarkeit der Funktionalität einer bestimmten Sprache symbolisieren könnte. Denn die Funktionalität, die durch die Prozedur `schleifen-schritt` bereit gestellt wird, kann nur erklärt werden, indem man gleichzeitig die Prozedur `wurzel` betrachtet. Deshalb sind die beiden Prozeduren `wurzel` und `schleifenschritt` durch ein Rechteck zu einer höheren Einheit zusammengefasst. Entsprechendes gilt für die Prozeduren `main`, `eingabe` und `berechnung und ausgabe`. Denn die beiden Prozeduren `eingabe` und `berechnung und ausgabe` stellen keine universell verwendbare Funktionalität bereit. Beim Ablauf der Eingabeprozedur werden interaktiv vom Benutzer der Wurzelgrad k und der Radikand r erfragt. Diese Funktionalität kann man für nichts anderes benutzen als für die Berechnung der Wurzel.

Bei der Frage nach der Universalität einer bereitgestellten Funktionalität darf man nicht nur die Prozedurbezeichnung betrachten, sondern man muss die durch die konkrete Prozedur bereitgestellte Funktionalität ansehen. So gibt es zwar in sehr vielen Programmen einen Bedarf an einer Prozedur `eingabe`, aber für die unterschiedlichen Aufgaben werden diese jeweils mit `eingabe` bezeichneten Prozeduren eine völlig unterschiedliche Funktionalität bereitstellen müssen. Deshalb kann man oberhalb der Prozedur `eingabe` keine waagerechte Linie

einziehen, welche den Schichtungsplan in zwei Abschnitte unterteilt, die unabhängig von einander verstanden werden können.

Man kann sich die hier betrachteten Verhältnisse an Hand einer Analogie aus dem Bereich des Automobilbaus veranschaulichen. Ein Auto besteht aus sehr vielen unterschiedlichen Komponenten, die man in zwei Klassen einteilen kann. In die eine Klasse fallen diejenigen Komponenten, für die es keine universelle Verwendbarkeit gibt, die also nur für einen ganz bestimmten Zweck im Auto konstruiert wurden. Man denke hier beispielsweise an ein ganz bestimmtes Gussteil, welches für die Verbindung der Benzinleitung mit dem Vergaser gebraucht wird. Wenn man dieses Gussteil isoliert betrachtet, wird man normalerweise gar nicht erkennen können, für was dieses Gussteil gebraucht werden könnte. Völlig anders liegt der Fall beim Betrachten einer Gewindeschraube, die zur Verbindung zweier Komponenten im Auto benutzt wird. Eine solche Schraube kann unabhängig von ihrer Benutzung verstanden werden und kann auch für viele andere Aufgaben verwendet werden.

Es erscheint mir wichtig, an dieser Stelle darauf hinzuweisen, dass die Frage nach der Grenzziehung im Schichtungsplan keine formal entscheidbare Frage ist. Sie sollten also nicht danach fragen, an Hand welcher formalen Kriterien man erkennen könnte, oberhalb welcher Prozeduren man einen Sprachtrennungsschnitt im Schichtungsplan einführen darf. Denn programmiersprachlich gibt es keinerlei Hinweise für eine solche Entscheidung.

Während im Schichtungsplan die benutzten Prozeduren immer unten liegen und die benutzenden Prozeduren oben, muss dies im Programmtext normalerweise genau umgekehrt sein. Es muss zuerst die Definition einer benutzbaren Prozedur im Programmtext vorkommen, bevor eine solche Prozedur in einem anderen Programmstück aufgerufen werden kann.

Ganz allgemein gilt, dass Bezeichner, die der Programmierer einführt, von der Maschine nicht verstanden werden können, wenn sie nicht zuvor vom Programmierer programmiersprachlich erklärt wurden. Eine Prozedurdefinition stellt eine solche Erklärung bezüglich des Prozedurnamens dar. Eine Typdefinition entspricht der Erklärung für den Typbezeichner.

Es gibt Programmiersprachen, die es erlauben, Prozeduren, die nur in Verbindung mit einer anderen Prozedur verstanden werden können, innerhalb des Prozedurtextes der verwendenden Prozedur zu definieren.