

## Eine Rekursionsgeschichte

Jede Funktion kann man sprachlich beschreiben. Dies zu versuchen, kann jeweils sehr hilfreich sein, da man dabei oft erkennt, ob man wirklich verstanden hat, was man programmieren will – und wenn man das nicht genau weiß, kann man natürlich auch kein Programm dafür entwickeln.

### Beispiel

Unser Beispiel, an dem wir die Ausformulierung der sprachlichen Beschreibung einsetzen wollen, ist das Ersetzen eines bestimmten Elementes in einer Liste durch ein vorgegebenes anderes.

### Die Aufgabe ist allgemein zu formulieren

Wenn diese Aufgabe nicht zu verallgemeinern wäre, es also wirklich nur darum ginge, z.B. in der Anschrift den einen Nachnamen "Müller" durch den Namen "Meier" zu ersetzen, weil die Person geheiratet hat, dann würden wir dazu sicherlich kein Programm schreiben, sondern das "per Hand" machen. Wir gehen daher davon aus, dass wir dies verallgemeinern wollen, also zwar ein bei der Anwendung fest vorgegebenes, aber sonst beliebiges Element durch ein eben solches ausgetauscht werden soll und auch die Liste muss während der Bearbeitung zwar bekannt sein, bei unserer Beschreibung des Vorgehens bei der Problemlösung aber beliebig.

### Notwendige Parameter

Diese Überlegungen zeigen, dass wir beim Aufruf der Funktion, der wir einen passenden Namen geben wie "ersetze" mindestens folgende Parameter übergeben müssen:

- das zu ersetzende Element,
- das Element, dass dafür eingesetzt werden soll und
- die Liste, in der es ersetzt werden soll.

### Ungenaue Formulierung

Bei der Bearbeitung von Programmen taucht immer wieder das Problem auf, dass die Aufgaben nicht eindeutig beschrieben sind. Und die erste Frage, die hier auftaucht, ist die Frage, was eigentlich gemacht werden soll, wenn das Element mehrfach auftaucht. Ebenso ist zu klären, was passieren soll, wenn das Element nicht in der Liste enthalten war. Außerdem ist nicht angegeben, ob das Element ggf. auch in Teillisten auftauchen könnte – so etwas nennt man eine tiefe Liste – und ob es auch dort ersetzt werden soll oder ob es sich um eine flache Liste handelt, bei der sich diese Frage nicht stellt.

### Festlegung

Wir legen uns fest: Alle auftretenden Fälle des Elementes in einer flachen sollen ersetzt werden und wenn es nicht enthalten war, wird die Liste nicht verändert und auch keine Meldung ausgegeben.

### Was kann passieren?

Bei der Bearbeitung von Listen greifen wir grundsätzlich am Kopf zu. Deshalb müsste die angegebene Frage eigentlich lauten:

"Was kann uns am Kopf der Liste passieren?"

- Die erste Möglichkeit wird oft vergessen:  
Die Liste hat gar keinen Kopf [mehr].  
Wenn die Liste leer ist, führt der Zugriff auf ihren Kopf zu einem Fehler. Wir brauchen dann aber auch nichts mehr zu tun, in einer leeren Liste ist nichts zu ersetzen.  
Die Liste wird dann leer – wie sie ist – zurückgegeben.
- Das zu ersetzende Element steht am Kopf.  
Dann ersetzen wir dies Element, indem wir das neue an seiner Stelle am Kopf der Liste einfügen.  
Was machen wir mit dem Rest der Liste? Die Antwort haben wir oben festgelegt: Auch im restlichen Teil der Liste soll das Element durch das neue ersetzt werden. Wer macht das? Ganz einfach, das macht ja gerade unsere Funktion, die wir oben mit `ersetze` bezeichnet haben!  
Also müssen wir den Satz von etwas weiter oben leicht modifizieren:  
Dann ersetzen wir dies Element, indem wir das neue an seiner Stelle am Kopf der mit `ersetze` bearbeiteten Restliste einfügen.
- Das zu ersetzende Element steht *nicht* am Kopf.  
Anstatt das neue Element am Kopf der Liste einzufügen, fügen wir das bisherige Kopfelement – und nun haben wir zugehört – am Kopf der mit `ersetze` bearbeiteten Restliste ein.

Das Schreiben der Funktion sollte so gelingen.

### **Probleme beim Nachdenken über die Lösung**

Probleme beim Nachdenken über die Lösung zeigen sich immer dann, wenn man es nicht schafft zu akzeptieren, dass man bei funktionaler Programmierung nicht die Abarbeitung des Programms beschreibt, also nicht die einzelnen Schritte. Typische Formulierungen dieser Art sind: *"Erst einmal muss das Programm ... und dann muss es ... und am Schluss hat es ..."* Dies ist eine Beschreibung eines Ablaufs.

### **Kein Ablauf sondern Auswertung**

Bei der funktionalen Programmierung beschreibt man alle verschiedenen Möglichkeiten, auf die man bei einem Schritt innerhalb des Ablaufes stoßen kann. Es sind nämlich Auswertungsschritte und jeder Auswertungsschritt muss eindeutig mit allen denkbaren Alternativen beschrieben sein.

Die Rekursion finden wir dann in dem Selbstaufzuruf der Funktion, in diesem Fall in der Angabe, dass die "mit `ersetze` bearbeiteten Restliste" verwendet wird. Wenn wir bei der funktionalen Analyse sonst alles richtig gemacht haben, "weiß" die Funktion, wie das geht und unser Scheme sorgt dafür, dass sie das richtig macht, wenn wir ihr die richtigen Parameter übergeben.