SAUER

DAS GROSSE

LOGO BUCH

ZU

CPC

UND

JOYCE

EIN DATA BECKER BUCH

ISBN 3-89011-157-2

Wichtiger Hinweis:
Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle Schaltungen, technischen Angaben und Programme in diesem Buch wurden von dem Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler ist der Autor jederzeit dankbar.

Vorwort

Wenn die Begrüßungszeile auf dem Monitor erscheint, kann das Programmieren mit LOGO sofort beginnen. Zunächst können Sie mit ein paar Vokabeln einfache Aktivitäten direkt veranlassen, dann wird damit ein neues LOGO-Wort definiert. Der Programmierer erweitert auf diese Weise nach und nach den Wortschatz von LOGO. Bald werden Sie feststellen, daß mit LOGO auch komplizierte, schwierige Aufgabenstellungen gemeistert werden können.

LOGO-Programme bestehen meist nur aus wenigen Zeilen. Umfangreichere Problemlösungen werden aus kleinen Einheiten, den Prozeduren, aufgebaut, die jeweils für eine ganz bestimmte Aufgabe formuliert werden. Die Struktur eines LOGO-Programms spiegelt deswegen unmittelbar den logischen Aufbau der Problemlösung wieder.

LOGO ist als Programmiersprache besonders durch die sogenannte Turtle-Grafik berühmt geworden, bei der grafische Effekte durch Befehle an einen Zeichenstift erzeugt werden. Mittlerweile wird diese Grafikmethode auch bei anderen Programmiersprachen benutzt. LOGO bietet aber sehr viel mehr als nur die Turtle-Grafik - diese Möglichkeiten auch dem Anfänger zugänglich zu machen, ist wesentliche Absicht dieses Buches.

Die Schneider-Rechner werden mit den beiden Programmiersprachen LOGO und BASIC geliefert. Der Schneider-Joyce ist als Textsystem darüber hinaus mit einem Textverarbeitungsprogramm ausgerüstet.

Die Käufer von Mikrocomputern werden in immer stärkerem Maße den Computer vorwiegend als Anwender von fertigen Programmsystemen und weniger als Programmierer nutzen. Weil es einfach ist, mit dem Programmieren in LOGO zu beginnen, ist es die ideale Sprache für den Anwender, der nur einen Einblick in das Programmieren bekommen will, im übrigen aber hauptsächlich Anwenderprogramme benutzen will. Die Möglich-

keiten von LOGO sind aber so vielfältig, daß er damit immer tiefer in das Problemlösen mit dem Computer eindringen kann.

Die Entwicklung von LOGO-Systemen für Mikrocomputer hat erst nach 1980 richtig begonnen. Es gibt noch keine verbindliche Sprachdefinition. Das kann man einerseits bedauern, während es andererseits auch eine rasche Weiterentwicklung möglich macht. Es gibt gegenwärtig im wesentlichen zwei LOGO-Familien auf Mikrocomputern: Das DR LOGO für die Schneider-Rechner stammt von der Softwarefirma Digital Research und unterscheidet sich von den Versionen deselben Herstellers für andere Rechner natürlich nur im Sprachumfang und einzelnen gerätespezifischen Elementen.

Diese LOGO-Versionen sind verwandt mit denen der Firma LCSI, die insbesondere auf den Apple-Rechnern und dem IBM-PC samt deren unzähligen kompatiblen Rechnern weite Verbreitung gefunden haben. Das vorliegende Buch kann in wesentlichen Teilen auch als Einführung in das LCSI-LOGO benutzt werden. Diese LOGO-Familie ist an typischen Fähigkeiten zu erkennen: den Eigenschaftslisten und der Möglichkeit von programmierten Fehlerausgängen. Stellvertretend für die drei seien die LOGO-Vokabeln pprop und catch genannt.

Die zweite LOGO-Familie wird meist als MIT-LOGO bezeichnet; sie ist in Deutschland durch das LOGO für den Commodore 64 und die von der Firma IWT für den APPLE II vertriebene Version verbreitet. Diese LOGO-Familie unterscheidet sich in den grundlegenden Kommandos nicht von der ersten. Der Sprachumfang ist deutlich geringer, die Version ist dafür sehr effizient implementiert. Für diese Familie eignet sich das vorliegende Buch weniger; es sei aber auf das im gleichen Verlag erschienene LOGO-Trainingsbuch des Autors für den C 64 verwiesen.

Das Buch ist zum schrittweisen Erlernen der Programmiersprache und ihren Anwendungsmöglichkeiten gedacht und in 19 Lektionen gegliedert. In den ersten zwölf Lektionen werden die wichtigsten Sprachelemente an einfachen Beispielen eingeführt, während in den letzten sieben Lektionen fortgeschrittene An-

wendungen an umfangreicheren Beispielen erläutert werden. In den Anfangslektionen werden keine Vorkenntnisse vorausgesetzt; Leser mit Vorwissen können diese Lektionen relativ schnell verlassen.

Wegen der einfach zu handhabenden Grafik bilden die grafischen Anwendungen einen Schwerpunkt bei der Einführung von LOGO. Dabei werden sowohl elementare Programme zur Turtle-Grafik als auch anspruchsvollere Beispiele wie solche zur dreidimensionalen Grafik betrachtet.

Auch die Anwendungen zur Listenverarbeitung erstrecken sich von einfachen Wortmanipulationen bis hin zu Anwendungen auf höhere Datenstrukturen. Schließlich werden die Möglichkeiten zur Dateiverwaltung in LOGO betrachtet. Im vorletzten Kapitel wird ein komplettes System von Programmen zur Dateiverwaltung entwickelt.

Um die Übersicht zu erleichtern, sind zu Beginn jeder Lektion die neu einzuführenden Sprachelemente zusammen mit den im Text behandelten Programmbeispielen aufgezählt. Auf ein LOGO-Lexikon mit der systematischen Beschreibung aller LOGO-Vokabeln wurde verzichtet, weil das Handbuch des Herstellers hierfür ausreicht. Am Ende der meisten Lektionen findet der Leser Übungsaufgaben.

Dem Verlagslektorat sei an dieser Stelle für die Zusammenarbeit bei der Fertigstellung des Buches gedankt.

Linden, Juni 1986
Gerhard Sauer

Inhaltsverzeichnis

Lektion 1: Start von DR LOGO

Mit LOGO können durch einfache Kommandos Berechnungen durchgeführt, Grafiken erstellt oder auch anspruchsvollere Aufgaben wie die Verwaltung von Dateien bewerkstelligt werden. Damit Ihr Computer wirklich einen Kreis auf dem Monitor zeichnet, wenn Sie den entsprechenden Befehl eingeben, muß dieser Befehl interpretiert, d.h. richtig erkannt und in unmittelbare Anweisungen an die eigentlichen Prozessoren umgesetzt werden.

Diese Aufgabe erledigt der LOGO-Interpreter; bei Ihrem Schneider-Computer heißt er DR LOGO und stammt von der Softwarefirma Digital Research. DR LOGO ist nun selbst wieder auf die Unterstützung eines Betriebssystems angewiesen, das beispielsweise dafür sorgt, daß der Rechner Ihre Eingaben auf der Tastatur verarbeitet oder die Disketten gelesen und beschrieben werden können. DR LOGO benutzt dabei das CP/M-Betriebssystem (Control-Program for Microprocessors) desselben Herstellers.

Bei den Schneider-CPC-Computern befinden Sie sich nach dem Einschalten des Rechners in einem eingebauten Betriebssystem, das mit der Programmiersprache BASIC unmittelbar verknüpft ist. Der Schneider Joyce ist dagegen nach dem Einschalten noch ohne jedes Betriebssystem.

Der erste Schritt zum Start von LOGO ist das Laden des CP/M-Betriebssystems. Bei den CPC-Rechnern haben Sie bei den älteren Typen 464 und 664 entweder nur die Version 2.2 oder beim 6128 auch die Version 3.0 (CP/M-Plus). Leider ist die unter CP/M 2.0 arbeitende LOGO-Version eine sehr abgemagerte Variante, wofür vermutlich der fehlende Speicherplatz verantwortlich ist. Hier wird in der Regel von der Version für CP/M-Plus ausgegangen; Abweichungen davon sollten Sie in Ihrem Benutzerhandbuch nachschlagen.

Schneider CPC

Zum Laden von CP/M wird die entsprechende Systemdiskette in das Lauferk eingelegt und mit dem Befehl

SHIFT+@ CPM (Abschluß mit der RETURN-Taste)

gestartet. Nach gleichzeitigem Drücken der Tasten SHIFT und @ erscheint ein senkrechter Strich als Zeichen auf dem Monitor, womit das Laufwerk angesprochen wird. Ist das System geladen, so erscheint als letztes auf dem Monitor das "Prompten" des Betriebssystems in der Form:

```
A>
```

Dabei bezeichnet der Buchstabe das Standardlaufwerk A. Das anschließende ">"-Zeichen stellt eine Aufforderung zur Eingabe von Befehlen dar.

Geben Sie nun ein

```
SUBMIT LOGO3
```

Damit wird ein Ladeprogramm mit der Bezeichnung LOGO3.SUB auf der eingelegten Diskette in Gang gesetzt. Dieses Vorprogramm sorgt auch dafür, daß anschließend die Tastatur richtig benutzt werden kann. Die Diskette darf bei der Verwendung des SUBMIT-Kommandos nicht schreibgeschützt sein, d.h. das Schreibschutzloch darf nicht offen sein, was man notfalls durch Überkleben erreichen kann.

Die Bezeichnung LOGO3 unterscheidet die Version von der unter CP/M 2.0 laufenden LOGO-Version. Da in der Regel nur die Version für CP/M 3.0 benutzt wird, ist es empfehlenswert, das Programm umzubenennen mit

```
RENAME LOGO.SUB=LOGO3.SUB
```

Dann lautet das Startkommando

```
SUBMIT LOGO
```

Das Laden von DR LOGO ist mit dem "Prompten" beendet; dabei erscheint am Anfang der Bildschirmzeile ein Fragezeichen als Aufforderung zur Eingabe und dahinter steht der Bildschirmcursor, der angibt, wo das als nächstes eingegebene Zeichen auf dem Bildschirm erscheint.

Schneider Joyce

Der Joyce benötigt unmittelbar nach dem Einschalten ein Betriebssystem auf Diskette im Standardlaufwerk. Legen Sie also einfach sofort nach dem Einschalten die CPM/Plus Diskette ein, dann wird das Betriebssystem automatisch eingeladen. Wenn Sie beispielsweise aus einer vorangegangenen Arbeit mit dem Textverarbeitungssystem heraus das CP/M-System starten wollen, können Sie den Neustart des Rechners mit der Tastenkombination

SHIFT+EXTRA+EXIT

auslösen.

Nach dem Einladen von CP/M sollte es nach der Beschreibung im Benutzerhandbuch wie beim CPC damit weitergehen, daß die Diskette mit dem LOGO-System eingelegt und der Befehl

```
SUBMIT LOGO
```

erteilt wird. Bei der dem Autor gelieferten Systemdiskette ergab sich sofort die Fehlermeldung

```
SUMBMIT?
```

Ein Blick in das Inhaltsverzeichnis der Diskette ergab, daß das CP/M-Kommando auf der Diskette fehlt. Wenn Sie diese Meldung nicht erhalten, dann haben Sie wahrscheinlich schon eine

neuere, vernünftig vorbereitete Diskette. Weitere Probleme ergeben sich mit den Zeichensätzen sowohl auf dem Bildschirm als auch auf dem angeschlossenen Drucker, weil das Joyce-System standardmäßig auf den deutschen Zeichensatz eingestellt ist, für LOGO aber der amerikanische Zeichensatz wegen der eckigen Klammern besser geeignet ist.

Da es ohnehin sehr ratsam ist, nur mit einer Systemkopie und nicht mit der Originaldiskette zu arbeiten, legen Sie sich am besten zunächst eine neue Diskette für DR LOGO an, die dann auch richtig präpariert wird. Eine neue Diskette muß als erstes formatiert werden. Dazu rufen Sie auf der CP/M-Systemdiskette das Programm disckit auf, indem Sie

```
DISCKIT
```

eingeben und anschließend den Anweisungen dieses Programms folgen.

Danach legen Sie die CP/M-Systemdiskette ein und rufen durch Eingabe von

```
PIP
```

das Kopierprogramm PIP auf. Als Aufforderung erscheint ein Sternchen am Zeilenanfang. Kopieren Sie nun nacheinander die folgenden Programme auf die neue Diskette. Dabei brauchen Sie das PIP-Programm nicht zu verlassen.

```
*b:logo.com=logo.com
*b:keys.drl=keys.drl
*b:submit.com=submit.com
*b:setkeys.com=setkeys.com
*b:language.com=language.com *b:ed.com=ed.com
*b:type.com=type.com
*b:erase.com=erase.com
*b:setlst.com=setlst.com
```

Die ersten beiden Teile (sogenannte Files) sind auf der LOGO-Diskette zu finden, die restlichen auf der CP/M-Systemdiskette. Bei Eingabe des Kopierbefehls muß die Quelldiskette eingelegt sein; anschließend fordert Sie das System zum Einlegen der neuen Diskette (für B) auf.

Es bietet es sich auch an, das CP/M-Betriebssystem mit auf die für LOGO vorbereitete Diskette zu kopieren (File J14GCPM3.EMS), weil dann für den Start von DR LOGO nur noch eine Diskette benötigt wird.

Zum Start muß ein Vorprogramm LOGO.SUB erzeugt werden, das die notwendigen Einstellungen von Tastatur, Bildschirm und Drucker vornimmt und anschließend das eigentliche LOGO-System einlädt. Ein solcher File kann mit dem Editor von CP/M geschrieben werden, dessen Beschreibung Sie im Handbuch nachlesen können, wobei auch eine Kurzbeschreibung mit Hilfe des HELP-Programms auf Seite 4 der Systemdisketten zu bekommen ist. Am einfachsten gehen Sie genau wie im folgenden beschrieben vor.

Geben Sie ein:

```
ed logo.sub
```

Die vorbereitete Diskette muß dabei eingelegt sein; als Antwort sollten Sie die Meldung NEW FILE und ein "Prompten" mit einem Sternchen vorfinden. Tippen Sie nun den Buchstaben i ein, wonach Sie auf dem Bildschirm

```
1:
```

sehen sollten. Jetzt werden folgende Zeilen nacheinander eingegeben, die jeweils mit der RETURN-Taste abgeschlossen werden:

```
1: language 0
2: setkeys keys.drl
3: setlst drucker.drl
4: logo.com
```

Dieser Vorgang wird mit der Tastenkombination ALT+Z beendet. Nach dem nun erscheinenden Sternchen wird der Buchstabe e und anschließend die RETURN-Taste gedrückt. Die Erzeugung des Files kann nun mit dem Kommando

```
type logo.sub
```

kontrolliert werden. Bei Fehlern wird der erzeugte File am besten mit

```
erase logo.sub
```

gelöscht und anschließend neu geschrieben.

In der zweiten Zeile wird eine Systemroutine setkeys aufgerufen, die zur richtigen Einstellung der Taststaur dient und den File keys.drl benutzt, den Sie von der Seite 4 der Systemdisketten schon kopiert haben. Weil die eckigen Klammern in LOGO wichtig sind, sollten diese auf dem Bildschirm auch als solche erscheinen, weswegen mit language 0 der amerikanische Zeichensatz eingestellt wird.

> Die eckigen Klanmern werden dann auf der Joyce-Tastatur mit den Ä- und Ü-Tasten erzeugt.

Um die eckigen Klammern auch auf dem Drucker richtig darzustellen, wird dieser mit der Routine setlst auf den amerikanischen Zeichensatz umgestellt. Dazu muß noch der File drucker.drl erzeugt werden, der vom Hersteller leider nicht mitgeliefert wird. Geben Sie deshalb wie oben ein

```
ed drucker.drl
```

und verfahren wie beim File LOGO.SUB bei der Eingabe. Es wird hier nur eine einzige Zeile benötigt:

```
1: ↑ 'ESC'R ↑ 'NUL'
```

Diese Zeile enthält eine sogenannte Escape-Sequenz zur Einstellung des Druckers.
Nun sollte Ihre Diskette zum Start von DR LOGO auf dem Joyce perfekt sein. Nach Eingabe von

```
submit logo
```

sollten die Zeilen des Files LOGO.SUB nacheinander auf dem Bildschirm erscheinen. Schließlich sollte die Einschaltmeldung von LOGO und zum Schluß ein Fragezeichen am Zeilenanfang erscheinen, das die Eingabebereitschaft signalisiert.

Lektion 2: Rechnen mit LOGO

Neue Sprachelemente:
*, /, +, -, Leerzeichen, make, Namen, Prozedurnamen, sin, cos, arctan, round, int, quotient, remainder, Editieren in LOGO

2.1 Erste Rechenaufgaben

Die Fähigkeit zum Rechnen hat den Computern ihren Namen gegeben; allerdings ist das Zahlenrechnen heute nur noch eine von vielen ihrer nützlichen Leistungen. Rechenaufgaben im beruflichen oder privaten Alltag werden kaum noch ohne Taschenrechner bearbeitet. Elektronische Taschenrechner sind dabei nichts anderes als winzige, nur für diesen speziellen Verwendungszweck gebaute Computer.

Mikrocomputer wie Ihr Schneider CPC oder ein anderer Personalcomputer sind natürlich nicht einfach komfortablere Taschenrechner. Ihr Vorteil besteht gerade in der breiten Palette ihrer Einsatzmöglichkeiten. Zunächst wollen wir aber untersuchen, was Ihr Mikro mit Hilfe von LOGO auf diesem Gebiet zustande bringt. Beim bloßen Rechenvorgang ist dies noch nachvollziehbar, und im Zweifelsfall kann man den Taschenrechner zur Kontrolle hinzuziehen.

Geben Sie also eine einfache Rechenaufgabe ein, z.B.:

```
3 * 5
```

Das Eingetippte erscheint dann als Protokoll auf dem Bildschirm. Damit ist die Aufgabe formuliert, die Bearbeitung durch den Computer kann erst beginnen, nachdem Sie ihn durch Drücken der RETURN-Taste dazu aufgefordert haben. Beim Taschenrechner muß man zu diesem Zweck die Taste "=" drücken.

LOGO antwortet in der folgenden Zeile mit:

```
15 (Ergebnis: 15)
```

Nun wird man die Aufgabe 3*5 schneller durch Kopfrechnen erledigen, was aber schon nicht mehr für eine Aufgabe folgenden Typs zutrifft:

```
723+3*(27+28*(411+198)-2*78)
```

Oder hätten Sie das Ergebnis 51492 auch so schnell im Kopf bekommen?

In einem Mathematikbuch würde die Aufgabe wahrscheinlich so stehen:

```
723+3(27+28(411+198)-2*78)
```

Es müssen aber alle Rechenoperationen angegeben werden. Wird nur das erste Multiplikationszeichen vor der runden Klammer vergessen, antwortet LOGO mit:

```
726

16923
```

Der Rechner hat zunächst die Addition 723+3 ausgeführt. Da kein Operationszeichen folgt, gilt die Rechenaufgabe als abgeschlossen, und das Ergebnis wird ausgegeben. Der Rest der Zeile ergibt wieder eine sinnvolle Rechenaufgabe, die separat gelöst wird und deren Resultat anschließend auf dem Bildschirm erscheint.

Wenn auch das zweite Multiplikationszeichen fehlt, erscheint die folgende Fehlermeldung:

```
I don't know what to do with (
```

Der Klammerausdruck ohne Operationszeichen kann nicht bearbeitet werden! Da der Zeilenrest durch die äußeren runden Klammern als eine zusammengehörige Aufgabe gekennzeichnet ist, kann der Anfang 27+28 nicht als sinnvolle Teilaufgabe betrachtet werden und es kommt zur Fehlermeldung.

Das Vergessen von Multiplikationszeichen ist nur einer der vielen Fehler beim Eintippen, die auch dem erfahrenen Computeranwender noch häufig unterlaufen. Wer sich beim Taschenrechner vertippt, muß fast immer mit der ganzen Aufgabe erneut beginnen, falls er den Fehler überhaupt bemerkt. Bei Ihrem Mikrocomputer können Sie die Aufgabenstellung korrigieren, solange Sie noch nicht die RETURN-Taste gedrückt haben.

Was zunächst auf dem Bildschirm erscheint, ist die *Eingabezeile*, d.h. die einzugebende Zeile mit der Aufgabenstellung. Diese Zeile steht Ihnen zunächst als eine Art Schmierzettel zur Verfügung, denn erst nach Drücken der RETURN-Taste macht LOGO ernst mit dem Inhalt dieser Zeile. Dann gibt es allerdings auch kein Pardon mehr, jeder Fehler wird bestraft!

Rechenaufgaben passen meist in eine Zeile. Das gilt sicher beim Schneider Joyce mit seinem besonders breiten Bildschirm, während man beim CPC gelegentlich auch über eine Bildschirmzeile hinauskommt, weil DR LOGO hier im 40-Zeichenmodus arbeitet. Wenn Sie mit der Eingabe am rechten Rand angelangt sind, wird automatisch in der nächsten Zeile weitergeschrieben. Die Überschreitung der Zeile wird rechts durch ein Ausrufezeichen sichtbar gemacht. Eine Eingabezeile darf sich demnach über mehrere Bildschirmzeilen erstrecken. Die maximale Länge können Sie durch Probieren selbst herausfinden. Der Rechner verweigert dann die Eingabe und gibt das mit einem Klingelton kund.

Das Ausrufezeichen wird auch im Text, insbesondere bei der Wiedergabe von Programmen verwendet, um anzuzeigen, daß die nächste Druckzeile noch zur selben logischen Zeile gehört. Solche Ausrufezeichen sind bei der Eingabe in den Rechner zu

übergehen. Beachten Sie, daß die Breite der Druckzeile nicht mit der der Bildschirmzeile übereinstimmt.

Wenn LOGO die Rechenaufgabe erledigt hat, ist diese im Rechner noch weiter verfügbar, da sie nämlich an einen Pufferspeicher übergeben wurde. Der Pufferinhalt kann für weitere Eingaben benutzt werden. Hierzu muß die Tastenkombination

CONTROL+Y

verwendet werden, wobei das Pluszeichen hier gleichzeitiges Drücken andeuten soll. Als Antwort erscheint der Inhalt der zuletzt eingegebenen Zeile erneut, und zwar in der nun gerade in Arbeit befindlichen Zeile. Es kann für eine einfache Wiederholung dieses Kommandos, aber auch in einer schon teilweise beschriebenen Zeile benutzt werden, um den Pufferinhalt anzuhängen. Die Zeile kann auch nach CONTROL+Y fortgesetzt werden; außerdem können Sie CONTROL+Y mehrfach wiederholen.

Beim Schneider Joyce kann für die gleiche Aufgabe auch die COPY-Taste verwendet werden - COPY ersetzt also CONTROL+Y. Die hier mit CONTROL bezeichnete Taste trägt beim Schneider Joyce die Aufschrift ALT.

Die Kombination CONTROL+Y löst eine von vielen sogenannten Editierfunktionen aus, die in LOGO verfügbar sind. Die CONTROL-Taste ist eine Umschalttaste wie die Shift-Tasten, die Sie vermutlich als Umschalttaste (Groß/Klein-Buchstaben) von der Schreibmaschine her kennen. Die Bedeutung der benutzten Taste wird hiermit verändert. Bei der Y-Taste z.B. so, daß auf dem Monitor kein Y ausgegeben wird. Wie alle Umschalttasten muß die CONTROL-Taste festgehalten werden, während die auslösende Y-Taste gedrückt wird.

2.2 Editieren der Eingabezeile

Zur Veränderung einer Eingabezeile stellt LOGO sogenannte Editierfunktionen bereit. Der Editor (so nennt man den dafür zuständigen Teil des Rechnerbetriebssystems) kann je nach LOGO-Version unterschiedlich viele Hilfsmittel zur Verfügung stellen. Die notwendigen Handgriffe sind auch bei gleichen Funktionen nicht immer dieselben. Hier muß man im Zweifelsfall im LOGO-Handbuch des Herstellers nachschauen - meist hilft auch unbefangenes Experimentieren. In diesem Buch wird der Editor des DR LOGO für die Schneidercomputer beschrieben; die Editierfunktionen sind hierbei im Handbuch nur lückenhaft beschrieben.

Cursorbewegungen

Der Cursor oder Zeiger gibt an, wo das nächste Zeichen auf dem Bildschirm erscheinen soll. Er kann mit den Cursorsteuertasten, die man an den Pfeilsymbolen erkennt, bewegt werden. Diese Tasten befinden sich im kleinen Tastenfeld neben der Buchstabentastatur.

Bei der Veränderung von Eingabezeilen bleiben Sie innerhalb einer (logischen) Zeile, weshalb hier hauptsächlich nur die Tasten für Cursorbewegungen nach rechts/links benötigt werden. Vertikale Bewegungen sind bei einer Eingabezeile nur dann möglich, wenn diese sich über mehrere Bildschirmzeilen erstrecken. Cursorbewegungen sind nur in dem bereits durch Zeichen gefüllten Bereich möglich.

LOGO stellt Ihnen nun noch zwei weitere Cursorbewegungen zur Verfügung:

- CONTROL+A: Cursor wird an den Anfang der Bildschirmzeile gesetzt, in der er sich gerade befindet

- CONTROL+E: Cursor wird entsprechend an das Ende der Bildschirmzeile gesetzt

Beachten Sie, daß sich die Cursorbewegungen auf den Bildschirmaufbau, nicht auf die logisch zusammengehörenden Zeilen beziehen.

Beim Joyce können Sie auch die Taste EZ/ZEILE zur Cursorbewegung an das Zeilenende bzw. den Zeilenanfang benutzen.

Einfügen von Zeichen

Um etwa bei der Zeichenfolge "3(" das fehlende Multiplikationszeichen einzufügen, setzt man den Cursor auf die runde Klammer und tippt das Multiplikationszeichen "*" ein. Die runde Klammer und alles, was danach noch folgt, wird dabei automatisch nach rechts verschoben. Die Eingabe von Zeichen über die Tastatur bewirkt also in LOGO ein Einfügen der neu eingegebenen Zeichen vor das unter dem Cursor liegende Zeichen. Bereits vorhandene Zeichen werden dabei nicht überschrieben.

Mit der Taste "TAB" (Tabulatortaste) können maximal vier Leerzeichen eingefügt werden, was bei Eingabezeilen allerdings kaum von Bedeutung ist. Der Tabulator ist fest in Abständen von je vier Zeichen (Spalten) eingestellt. Mit der Tab-Taste wird der Cursor auf die nächstfolgende Tabulatorposition gesetzt. Es werden dabei aber keine Zeichen übersprungen, sondern die erforderliche Zahl von Leerzeichen eingefügt und ein vorhandener Zeilenrest verschoben.

Löschen von Zeichen

Hier gibt es in LOGO mehrere Möglichkeiten:

- Taste "DEL" (im Hauptfeld oben ganz rechts)

  Hiermit wird das links vom Cursor stehende Zeichen gelöscht und gleichzeitig der mit dem Cursor beginnende Rest der Eingabezeile um eine Position nach links verschoben. Wiederholtes Drücken der Taste

"DEL" verschiebt also den mit der Cursorposition beginnenden Teil der Zeile als Ganzes schrittweise nach links.

- Taste "CLR" (links neben "DEL")
- CONTROL+D mit gleicher Funktion

Hiermit wird das unter dem Cursor stehende Zeichen gelöscht und der dann noch folgende Rest der Zeile um eine Position herangerückt. Die wiederholte Ausführung löscht einen mit der Cursorposition beginnenden Teil der Eingabezeile.

Wollen Sie einen Teil der Zeile löschen, müssen Sie bei der Verwendung von "DEL" den Cursor hinter das letzte Zeichen, bei der Benutzung von CLR dagegen auf das erste Zeichen des zu löschenden Teils stellen.

Beim Schneider Joyce trägt die CLR-Taste ebenfalls die Bezeichnung "DEL". Zur Unterscheidung ist bei der wie CONTROL+D funktionierenden Taste ein Pfeil nach rechts, bei der wie hier als "DEL"-Taste beschriebenen ein Pfeil nach links zusätzlich angebracht.

- CONTROL+K

Hiermit wird der Rest der Zeile, der mit der Cursorposition beginnt, auf einmal gelöscht. Durch CONTROL+A und CONTROL+K wird dann eine ganze Zeile gelöscht.

CONTROL+K wirkt im Gegensatz zu CONTROL+A und CONTROL+E auf die gesamte logische Zeile, was natürlich nicht ganz konsequent ist.

Korrektur von fehlerhaften Eingabezeilen

Wenn nach Eingabe einer Zeile eine Fehlermeldung erscheint, sollten Sie zunächst einmal die Fehlermeldung genau studieren. Anschließend bringen Sie mit der bereits besprochenen Tastenkombination CONTROL+Y (bzw. COPY beim Joyce) die fehlerhafte Zeile wieder auf den Bildschirm. Die zuvor betrachteten Editierfunktionen ermöglichen dann eine Fehlerkorrektur ohne komplette Neueingabe der betreffenden Zeile.

Rechenarten in LOGO

LOGO kennt natürlich die vier Grundrechenarten:

Rechenoperation	Zeichen	Beispiel
Addition	+	2.1 + 3.8
Subtraktion	-	7.2 - 5.5
Multiplikation	*	3.8 * 7
Division	/	10 / 3

Bruchaufgaben können bei Computern nicht mit Bruchstrichen geschrieben werden, weil Zahlen und Rechenzeichen alle in einer Zeile stehen müssen.

Es gilt die Regel: Punktrechnung (*, /) vor Strichrechnung (+, -). Abweichungen davon müssen durch runde Klammern gekennzeichnet werden. Mit Hilfe von Klammern können dann auch die Bruchaufgaben formuliert werden. Beispiel:

```
(3.14*5*5+12)/(1.44 - 0.7)
```

Dabei wurde jeweils Zähler und Nenner in Klammern eingeschlossen.

Infix- und Präfix-Notation

Bei der gewohnten Schreibweise

```
3*5, 4-8, ...
```

einer Grundrechenoperation steht das Operationszeichen zwischen den beiden Operanden, was man als Infixnotation bezeichnet. Ungewohnt ist Ihnen wahrscheinlich die sogenannte Präfixnotation

```
* 3 5 bzw. - 4 8
```

Dabei wird zuerst das Operationszeichen gesetzt, anschließend folgen die Operanden. DR LOGO läßt grundsätzlich beide Schreibweisen zu. Beachten Sie aber:

> In der Präfixnotation müssen Operationszeichen und Operanden jeweils durch Leerzeichen voneinander getrennt werden.

Wie Sie nach tieferem Eindringen in die Regeln von LOGO später noch feststellen werden, entspricht die Präfixnotation dem allgemein verwendeten Aufbau von LOGO-Anweisungen: Man gibt zuerst einmal an, was überhaupt gemacht werden soll. Anschließend folgen weitere Angaben, die für die betreffende Aufgabe benötigt werden. Die Infixnotation bei Rechenoperationen (auch bei Vergleichsoperationen) ist ein Zugeständnis an die vertraute Schreibweise der Algebra.

Die Präfixnotation hat den Vorteil, daß sie auch ohne Mißverständnis bei mehr als zwei Operanden verwendet werden kann. Probieren Sie:

```
(+ 1 2 3 4)
```

Ergebnis ist die Zahl 10=1+2+3+4! Die runden Klammern sind notwendig, damit alles zu einer einzigen Rechenaufgabe zusammengefaßt wird. Die runden Klammern werden in LOGO in dieser Weise immer verwendet, wenn die Anzahl der Ein-

gaben bei einer Operation unterschiedlich groß ist. In runde Klammern wird die gesamte Aufgabe gesetzt, wenn mehr bzw. weniger als die Standardanzahl von Operanden folgen.

Bei Operationen wie der Subtraktion und Division gibt es immer nur zwei Operanden.

2.3 Zahlen speichern

In allen Anwendungsbereichen gibt es Zahlenwerte, die häufig benötigt werden. Bei Rechnungen wird ständig der Mehrwertsteuersatz benötigt, Naturwissenschaftler brauchen die Werte von Naturkonstanten, Techniker genormte Richtwerte, Devisenhändler den Dollarkurs, usw. Wenn Zahlenwerte immer wieder in Rechenaufgaben auftauchen, will man sie nicht immer wieder neu eintippen. Auch bessere Taschenrechner bieten die Möglichkeit, solche Zahlenwerte zu speichern, damit sie nach Bedarf immer wieder abgerufen werden können.

In LOGO dient hierzu die Anweisung make:

```
make "MWStsatz 14/100
```

Der Sinn der Zeile ist klar: Es soll künftig der Mehrwertsteuersatz von 14% unter der Bezeichnung MWStsatz verfügbar sein. Anweisungen mit dieser Wirkung werden auch Wertzuweisungen genannt; die BASIC-Programmierer kennen dafür das Schlüsselwort LET.

Schauen wir uns zunächst die Syntax, d.h. den grammatikalischen Aufbau der Zeile an! Die Zeile beginnt mit dem Schlüsselwort make (auf Deutsch etwa SETZE), danach folgt die Bezeichnung "MWStsatz für den Steuersatz sowie der Zahlenwert, hier als Ergebnis einer Divisionsaufgabe.

Beachten Sie, daß diese drei Objekte durch Leerzeichen voneinander getrennt werden müssen! Der Verstoß gegen die Trennvorschrift wird nun nicht mehr geduldet. Probieren Sie

aus, was passiert, wenn eines der beiden Leerzeichen vergessen wird. Nach

```
make"MWStsatz 14/100
```

erscheint die Meldung: "I don't know how to make"MWStsatz". (Ich weiß nicht, wie das Kommando make"MWStsatz ausgeführt werden soll!) Aus der Meldung erkennt man, daß make und "MWStsatz ohne trennendes Leerzeichen als ein zusammengehörender Ausdruck angesehen wird.

Der Steuersatz kann natürlich auch direkt in der Form 0.14 eingesetzt werden:

```
make "MWStsatz 0.14
```

(Der Dezimalpunkt anstelle des Kommas ist Ihnen vermutlich vom Taschenrechner her geläufig.) Wird nun das zweite Leerzeichen weggelassen:

```
make "MWStsatz0.14
```

so reagiert LOGO mit der Fehlermeldung:

```
Not enough inputs to make
```

Zu Deutsch: Das Kommando make benötigt mehr Eingaben.

Die Zeichenfolgen "MWStsatz und 0.14 sind Eingaben für make. Die für ein Kommando erforderlichen Eingaben werden in LOGO grundsätzlich hinter dem Aufruf der angeforderten Aktivität in der jeweils vorgesehenen Anzahl und Reihenfolge aufgeführt und müssen durch Leerzeichen voneinander getrennt werden. Wird ein Leerzeichen vergessen, so stimmt die Anzahl nicht mehr. Daher erscheint die Fehlermeldung ... not enough inputs...

Die beiden Eingaben "MWStsatz und 0.14 haben eine unterschiedliche Form; zum Zahlenwert 0.14 gibt es nichts weiter zu sagen. Warum wird der Mehrwertsteuersatz mit "MWStsatz und nicht einfach mit MWStsatz bezeichnet?

Probieren Sie aus:

```
make MWStsatz 0.14
```

LOGO antwortet mit einer Fehlermeldung:

```
I don't know how to MWStsatz
```

Zu Deutsch: Ich weiß nicht, wie MWStsatz auszuführen ist. LOGO versucht offensichtlich, MWStsatz als Kommando zu deuten, kann es aber nicht ausführen. Das Anführungszeichen vor MWStsatz ist notwendig, um LOGO mitzuteilen, daß mit "MWStsatz kein Kommando gemeint ist.

make und "MWStsatz sind Namen für unterschiedliche Begriffe. make benennt eine Tätigkeit oder ein Kommado, "MWStsatz soll dagegen für einen Zahlenwert stehen. Alle Namen, die keine Kommandos benennen, müssen in LOGO zur Unterscheidung mit einem Anführungszeichen eingeleitet werden. Die Zahl 0.14 wird von LOGO ohne weiteren Zusatz richtig erkannt. Auch Zahlen sind für LOGO Wörter, aufgrund ihrer speziellen Bedeutung werden sie aber als besondere Wortart behandelt.

Groß-/Klein-Schreibung im LOGO

Beim Start von DR LOGO wird die Tastatur bei den Schneidercomputern auf den Normalbetrieb wie bei einer Schreibmaschine eingestellt, d.h. bei Buchstabentasten erscheinen zunächst Kleinbuchstaben, während zur Erzeugung von Großbuchstaben gleichzeitig die SHIFT-Taste gedrückt werden muß.

Die Verwendung der Groß- und Kleinschreibung wird bei den LOGO-Systemen auf verschiedenen Mikrocomputern unterschiedlich gehandhabt; hier muß man im Zweifelsfall erst einmal experimentieren.

Beim DR LOGO auf den Schneidercomputern ist Groß- und Kleinschreibung nebeneinander möglich, wobei das LOGO-System die unterschiedliche Schreibweise auch tatsächlich bemerkt. Mit den beiden unterschiedlichen Schreibweisen

```
make "MWStsatz 0.14 und make "MWSTSATZ 0.13
```

werden demnach zwei verschiedene Namen mit Werten versehen. Zwar gewinnt man damit mehr Möglichkeiten für die Wahl von Namen, andererseits stellt die gleichzeitige Verwendung auch eine Fehlerquelle dar, weil Sie dann neben dem Namen auch noch die genaue Schreibweise wissen und korrekt benutzen müssen.

Empfehlenswert ist eher, sich auf eine bestimmte Regel festzulegen. In diesem Buch werden meist die Namen mit Kleinbuchstaben geschrieben, die wie MWStsatz mit Werten verknüpft sind. Programmnamen werden dagegen zur Verdeutlichung in der Großschreibung benutzt. (Dabei soll es für diese Lektion allerdings bei MWStsatz bleiben.)

Beachten Sie aber:

> LOGO-eigene Vokabeln (z.B. make) müssen grundsätzlich mit Kleinbuchstaben geschrieben werden. "MAKE" führt demnach zu einer Fehlermeldung.

An wenigen Stellen, den sogenannten Systemeigenschaften, verwendet DR LOGO auch selbst Großbuchstaben, worauf dann jeweils noch genauer hingewiesen wird.

2.4 Mit gespeicherten Zahlen rechnen

Zurück zur Steuerberechnung. Wie muß die Eingabezeile zur Berechnung der Mehrwertsteuer bei einem Nettopreis von 50 DM lauten, wenn der Steuersatz jetzt von LOGO zur Verfügung gestellt werden kann?

```
5Ø * :MWStsatz
```

Welche Bedeutung hat der Doppelpunkt vor :MWStsatz, und warum steht nicht wie in der make-Zeile "MWStsatz? Daß "MWStsatz nicht richtig ist, zeigt sich in der dann folgenden Meldung:

```
* doesn't like MWStsatz as input
```

(Die Operation mit dem Zeichen * mag keine Eingabe der Form "MWStsatz.)

Damit sind Ihnen drei verschiedene Benennungsweisen begegnet: Ohne einleitendes Sonderzeichen, mit einleitendem Anführungszeichen und mit Doppelpunkt als Startsignal:

- Benennungen ohne Sonderzeichen bezeichnen in LOGO Kommandos (Tätigkeiten, auch Prozeduren genannt).

- Benennungen mit einleitendem Anführungszeichen bezeichnen Namen, denen bestimmte Eigenschaften zugeordnet worden sind (z.B. Zahlenwerte). In der Mathematik spricht man auch von einem Platzhalter. Anschaulicher kann man auch vom Namen einer Schublade sprechen.

- Benennungen, die mit einem Doppelpunkt beginnen, bezeichnen den Wert, für den der Platzhalter steht. Für :MWStsatz kann man im Klartext auch sagen *Wert von* MWStsatz.

Anders ausgedrückt:

- "MWStsatz gibt den Namen einer Schublade an,
- :MWStsatz bezeichnet dagegen den Inhalt der betreffenden Schublade.

Ihr Mikrocomputer hat natürlich für mehr als eine Schublade Platz. Wenn z.B. auch der Nettopreis mit

```
make "nettopreis 50
```

gespeichert wird, so werden Mehrwertsteuer und Bruttopreis in den beiden folgenden Zeilen berechnet:

```
:nettopreis * :MWStsatz
```

```
:nettopreis + :nettopreis * :MWStsatz
```

Wer mit anderen Programmiersprachen gearbeitet hat, muß sich an die Verwendung des Doppelpunktes für "Wert von" erst einmal gewöhnen. Die Meldung "I don't know to ... " wird am Anfang wahrscheinlich häufiger auftreten. Daß LOGO die klare Unterscheidung zwischen Namen und Inhalt einer Schublade benötigt, hängt damit zusammen, daß LOGO auch Fähigkeiten besitzt, die in den meisten Programmiersprachen nicht vorhanden sind.

So kann beispielsweise der Inhalt einer Schublade selbst wieder Name einer anderen Schublade sein; die Namen von Schubladen können auch manipuliert (zerlegt und zusammengesetzt) werden. Schließlich können den Namen auch noch andere Eigenschaften als Werte zugeordnet werden. Diese raffinierteren Fähigkeiten werden wir uns aber erst in einem zweiten Durchgang genauer anschauen.

2.5 Mathematische Funktionen

Einfache Taschenrechner besitzen meist eine Taste zur Berechnung der Quadratwurzel, besser ausgestattete zusätzliche Tasten für trigonometrische und andere mathematische Funktionen. Die DR LOGO-Version für die Schneidercomputer ist in dieser Hinsicht schwach ausgerüstet.

Funktion	Ergebnis	
round 3.2	3	Gerundeter Wert der Zahl 3.2
round 3.6	4	
int 4.3	4	Ganzzahliger Wert der Zahl 4.3
quotient 8 3	2	Ganzzahliger Teil des Bruchs 8/3
quotient 8 -3	-2	
remainder 11 4	3	Rest bei der Division 11 durch 4
remainder -10 3	-1	

Die Funktionen round, int, quotient, remainder haben gemeinsam, daß ihr Ergebnis eine ganze Zahl ist. round rundet einen Eingabewert auf eine ganze Zahl auf oder ab. Bei negativen Eingabewerten wird dabei allerdings der Betrag der Zahl gerundet. int liefert den ganzzahligen Teil einer Eingabe.

quotient und remainder liefern bei der Division ganzer Zahlen den ganzzahligen Teil des Quotienten und den Divisionsrest. Sind die Eingabewerte keine ganzen Zahlen, so werden sie zunächst wie mit int gerundet. DR LOGO verfährt übrigens immer so, daß dort, wo eine ganzzahlige Eingabe erwartet wird, im Zweifelsfall ein nichtganzzahliger Wert durch Abschneiden ganzzahlig gemacht wird.

Im Hinblick auf die Schreibweise von negativen Konstanten müssen Sie den Unterschied zwischen dem Minuszeichen als Operationszeichen für die Subtraktion und als Zeichen für das Vorzeichen beachten.

> DR LOGO interpretiert das Minuszeichen als Operationszeichen, wenn ein Leerzeichen folgt. Um die Bedeutung als Vorzeichen sicherzustellen, darf also kein Leerzeichen folgen.

Anmerkung: Nicht alle LOGO-Versionen arbeiten genauso!

Ein typischer Fehler tritt in diesem Zusammenhang beispielsweise in der folgenden Zeile auf:

```
quotient 8 - 3
```

Antwort: `Not enough inputs to quotient`

Das Minuszeichen wird als Operationszeichen interpretiert, d.h. es wird die Zahl 5=8-3 ausgerechnet und als Eingabe von quotient genommen. Weil quotient aber zwei Eingabewerte erwartet, kommt es zur Fehlermeldung. Die Meldung ist insofern typisch, als durch die Auswertung der Subtraktion die Zahl der Eingaben um eins veringert wird, was dann den zitierten Fehler verursacht. Die Fehlerursache ist dabei lediglich ein zuviel gesetztes Leerzeichen nach dem Minuszeichen.

Falls Ihnen später einmal wieder diese Fehlermeldung begegnet, sollten Sie nachschauen, ob Ihnen vielleicht ein so fehlinterpretiertes Minuszeichen den Streich gespielt hat.

Zunächst wurden die Funktionen benutzt, um das Ergebnis unmittelbar als Reaktion des Rechners auf den Bildschirm zu bringen. Die Resultate können aber natürlich auch aufgehoben werden.

```
make "zahl round 5.9
make "zahl round 5 / 3
make "zahl round :zahl
```

Anstelle einer Zahl kann auch eine Rechenaufgabe stehen, die selbst eine Zahl als Ergebnis erzeugt. In solchen Rechenaufgaben können auch Funktionen verwendet werden.

Sie können übrigens auch bei den Funktionen mit nur einem Eingabewert diesen in runde Klammern setzen und das trennende Leerzeichen dann weglassen:

```
round(5.9) round(:zahl/3)
```

Die runden Klammern sind in der Mathematik bei Funktionen üblich und werden auch in den meisten Programmiersprachen benutzt.

Als zweite Gruppe von Funktionen stehen noch die trigonometrischen Funktionen zur Verfügung.

Funktion	Ergebnis	
sin 30	0.5	Sinuswert für den Winkel 30 Grad
cos 30	0.866025388240814	Kosinuswert für den Winkel 30 Grad
arctan 0.5	26.565051177078	Arctan-Wert von 0.5 in Grad

Zur Bedeutung von sin und cos ist nichts weiter zu sagen. Von den Umkehrfunktionen ist die der Tangensfunktion vorhanden, aus der sich die Umkehrfunktionen der anderen trigonometrischen Funktionen berechnen lassen. Bei diesen Funktionen ist zu beachten, daß die Winkel im Gradmaß anzugeben sind, bzw. bei arctan im Gradmaß geliefert werden.

Die vielen Dezimalstellen bei den Ergebnissen sind eine Eigenart der Version auf den Schneiderrechnern. Dabei wird eine Genauigkeit vorgetäuscht, die überhaupt nicht vorhanden ist. Denn der Kosinus von 30 Grad sollte beispielsweise den Wert 0.866025403784439 haben, so daß sich nur die ersten sieben Stellen nach dem Dezimalpunkt als vertrauenswürdig erweisen.

Für Umrechnungen zwischen Winkel und Bogenmaß wird die Kreiszahl pi benötigt, die hier nicht eingebaut ist, wohl aber mit

```
make "pi 3.14159265
```

zur Verfügung gestellt werden kann. Die Umrechnung geschieht dann wie folgt:

```
make "grad :bogen * 180 / :pi
make "bogen :grad * :pi / 180
```

Übungen

1) Berechnen Sie den Ausdruck

$$\frac{5x^2-3x}{2x+1}$$

für verschiedene Werte von x.

2) 1. Zeile: `make "zahl pi make "zahlh 1`
2. Zeile: `make "zahlh :zahlh * :zahl :zahlh`
Was geschieht, wenn Sie die zweite Zeile (mit CONTROL+Y) erneut eingeben?

3) Mit (make "a 5 sin :a) kann durch Wiederholung und Ersetzung der Zahl 5 eine Wertetabelle für die Sinusfunktion erstellt werden. Stellen Sie in dieser Weise eine Tabelle für die Tangensfunktion (tan(x)=sin(x)/cos(x)) auf.

4) 1. Zeile: `make "w 1 make "r 2`
2. Zeile: `make "w (:w + :r / :w ) / 2 :w`
Untersuchen Sie, was bei fortgesetzter Wiederholung der zweiten Zeile geschieht.

Lektion 3: Würfeln und Wiederholen

Neue Sprachelemente:
random, rerandom, pr, type, copyon, copyoff, repeat, Listenbegriff, Unterbrechung mit CONTROL+G, ESC (STOP),CONTROL+Z, co

3.1 Zufallszahlen

Daß bei Computern auch der Zufall mit im Spiel sein kann, erscheint vielen am Anfang widersinnig. Überzeugen Sie sich, indem Sie folgende Zeile eingeben:

```
random 6
```

Mit der Wiederholungstaste (CONTROL+Y) kann diese Zeile unverändert wiederholt eingegeben werden. Trotz gleicher Anweisung gibt LOGO tatsächlich unterschiedliche Antworten. Gemeinsam ist den Resultaten:

- Das Ergebnis von random n ist stets eine ganze Zahl
- Der Wert liegt zwischen 0 und n-1.

Die Bezeichnung Zufallszahl gibt an, daß aus den möglichen Ergebnissen in gleicher Weise wechselnde Resultate erzeugt werden, wie dies bei einem echten Zufallsprozeß vom Typ des fairen Würfelns zu erwarten ist. Allerdings sollten beim Würfeln die Ergebnisse zwischen 1 und 6 liegen.

Das läßt sich leicht durch

```
1 + random 6
```

auch erreichen. Mit dieser Zeile haben wir also tatsächlich einen fairen Würfel simuliert.

Bei jedem Neustart von LOGO wird bei der Benutzung von random die gleiche Folge von Zufallszahlen erzeugt. Dieser Zustand kann auch mit Hilfe von

```
rerandom
```

wiederhergestellt werden. Wird also die Zeile zur Simulation eines Würfels mit rerandom eingeleitet

```
rerandom 1+random 6,
```

wird immer wieder dieselbe Augenzahl gewürfelt. Manche Rechner haben die Möglichkeit, gerade den umgekehrten Effekt zu erzielen, indem sie dafür sorgen, daß sich die Folgen von Zufallszahlen nicht wiederholen. Man kann dann den Beginn einer Folge von außen beeinflussen. In DR LOGO ist dieser Fall nicht vorgesehen. Man kann sich damit behelfen, am Anfang erst einmal eine variable Zahl von Würfelvorgängen vorzugeben, d.h. eine veränderbare Zahl von Aufrufen von random.

3.2 Ausgabe von Resultaten

Bisher hat LOGO alle Rechenaufgaben bearbeitet und das Ergebnis bekanntgegeben. Das ist zunächst einmal eine Gefälligkeit, denn Ergebnisse fallen eigentlich nur im Innern des Computers an. Der Bildschirm ist für den Computer ein Ausgabegerät genau wie ein Drucker oder ein externes Speichermedium wie das Floppylaufwerk. In den meisten Programmiersprachen (auch in manchen LOGO-Versionen) muß man deswegen die Ausgabe eines Ergebnisses immer durch eine eigene Anweisung bewerkstelligen. Bei einer Eingabezeile gibt DR LOGO die Rechenresultate automatisch aus, falls die Eingabezeile sich in sinnvoll interpretierbare Aufgaben zerlegen läßt.

Die Kommandos, die direkt in einer Eingabezeile gegeben werden, erfahren eine Sonderbehandlung durch DR LOGO. Hier versucht LOGO möglichst viele sinnvolle Ergebnisse zu produ-

zieren, auch wenn die sonst gültigen Sprachregeln nicht exakt eingehalten werden.

Für die Ausgabe von Ergebnissen dient im allgemeinen Fall die Anweisung pr (Abkürzung für Print). Hierzu einige Beispiele:

```
pr 3*5
(pr 3*5 7*9)
pr 1+random 6
```

pr bedeutet "Ausgabe von ... auf den Bildschirm". Die auszugebenden Daten oder Werte werden anschließend durch Leerzeichen voneinander getrennt. Standardmäßig wird ein Ergebnis bei einem Aufruf von pr erwartet. Falls wie beim zweiten Beispiel mehrere Ergebnisse ausgegeben werden sollen, müssen alle auszugebenden Werte zusammen mit dem Schlüsselwort pr in runde Klammern gesetzt werden. Die runden Klammern haben in LOGO die Aufgabe, mehrere Dinge als zusammengehörig zu erklären.

Druckerausgabe

Falls Sie einen Drucker an Ihren Computer angeschlossen haben, können Sie natürlich auch Ergebnisse ausdrucken lassen. In LOGO ist das normalerweise sehr einfach. Der Aufruf von

```
copyon
```

sorgt dafür, daß alle Ausgaben, die Sie im folgenden auf dem Bildschirm (genauer gesagt auf, dem sogenannten Textfenster) erscheinen lassen, gleichzeitig auch auf dem Drucker wiedergegeben werden. Während copyon den Drucker als zweites Ausgabemedium hinzuschaltet, kann die Druckerausgabe auch mit

```
copyoff
```

wieder abgeschaltet werden.

3.3 Wiederholung von LOGO-Anweisungen

Ob mit 1+random 6 wirklich die Ergebnisse eines guten Würfels simuliert werden können, läßt sich erst durch die Untersuchung vieler Würfe beurteilen, die der Computer macht. Die Wiederholung von Eingabezeilen durch die Tastenkombination Control+Y ist zwar in LOGO schon recht einfach, doch kann ein Computer die Wiederholung einer Tätigkeit mühelos selbständig und ohne manuellen Eingriff bewerkstelligen. Der Auftrag etwa zur zehnfachen Wiederholung wird LOGO durch repeat 10 erteilt. Beispiel:

```
repeat 10 [pr 1+ random 6]
```

repeat heißt *wiederhole*. Es benötigt zwei Angaben, nämlich zuerst wie oft etwas und zweitens was wiederholt werden soll.

Die Anzahl der Wiederholungen muß sinnvollerweise eine natürliche Zahl sein. Die Ganzzahligkeit wird von LOGO gegebenenfalls durch vorheriges Abschneiden hergestellt. LOGO läßt übrigens auch die Anzahl Null zu; worauf in der repeat-Zeile nichts geschieht. Die Anzahl der Wiederholungen muß nicht unbedingt eine Konstante sein, es kann an dieser Stelle auch der Wert eines Platzhalters stehen, z.B.

```
make "n 10
repeat :n [pr 1+random 6]
```

Die Konstante kann auch durch eine Rechenaufgabe ersetzt werden, falls die zu einem geeigneten Ergebnis führt. In LOGO kann man aber auch an einer solchen Stelle eine Tätigkeit in Gang setzen, wenn diese Tätigkeit eine brauchbares Ergebnis liefert, z.B. eine nichtnegative Zahl. Probieren Sie folgendes Beispiel mehrfach aus:

```
repeat random 10 [pr 1+random 6]
```

Hier wird durch random 10 die Anzahl der Wiederholungen selbst erst erwürfelt. Um die Zusammengehörigkeit von random und 10 optisch deutlich zu machen, können Sie beides auch in runde Klammern einschließen:

```
(random 10).
```

Was wiederholt werden soll, entspricht dem, was bisher in einer Eingabezeile allein stand. Wiederholt werden können Tätigkeiten und Abfolgen mehrerer Tätigkeiten. In unserem Beispiel ist die wiederholte Tätigkeit die Ausgabe - pr - des Ergebnisses der Rechenaufgabe 1+random 6. Beachten Sie, daß die Tätigkeit pr hier angegeben werden muß. Nur bei einfachen Eingabezeilen mit einem Ergebnis löst LOGO die Ausgabe selbst aus.

Die zu wiederholende Tätigkeit wird in eckige Klammern eingeschlossen. Die eckigen Klammern sind auch tatsächlich erforderlich und dienen nicht nur der optischen Zusammenfassung. Ohne die eckigen Klammern erhalten Sie die Fehlermeldung [pr 1 + random 6] didn't output to repeat. Warum gerade diese Fehlermeldung hier erscheint, soll erst einmal unbeantwortet bleiben.

Hier stoßen wir auf den zweiten, zentralen Begriff der Programmiersprache LOGO,den Begriff der *Liste*.

Bei der Wiederholung mit repeat kann man nicht nur eine, sondern auch eine ganze Reihe von Tätigkeiten wiederholen. Diese Tätigkeiten müssen in aufzählender Weise zusammengefaßt werden. Die Trennung erfolgt, wie inzwischen schon vertraut, durch Leerzeichen, die Zusammenfassung geschieht durch die eckigen Klammern. Später werden wir noch einige andere Möglichkeiten zur Listenbildung kennenlernen.

Als Beispiel für eine Wiederholung mehrerer Tätigkeiten soll eine Verbesserung des wiederholten Würfelns dienen.

Die Ausgabe der erwürfelten Zahlen geschah bisher jeweils auf einer neuen Zeile. Die Anweisung pr hat stets zur Folge, daß nach der Ausgabe automatisch ein Zeilenvorschub erzeugt wird.

Werden bei einer Ausgabe mit pr mehrere Ergebnisse ausgegeben, so erzeugt LOGO automatisch trennende Leerzeichen. Für viele Zwecke ist damit eine problemlos lesbare Ausgabe sichergestellt, ohne daß man über das Format der Ausgabe eigene Vorkehrungen treffen muß. Beide automatischen Formatierungsfunktionen, sowohl der automatische Zeilenvorschub als auch das trennende Leerzeichen, können unterbunden werden. Zu diesem Zweck muß pr, bzw. pr durch type ersetzt werden.

```
type 3*5 type 7*9
```

Antwort: 1563?

Die Antwort bedeutet natürlich nicht die Zahl 1563; die beiden Ergebnisse 15 und 63 werden vielmehr ohne Trennung aneinandergehängt. Werden beide Ergebnisse mit einem einzigen type-Kommando ausgegeben, so erfolgt eine Trennung durch Leerzeichen.

```
(type 3*5 7*9)
```

Antwort: 15 63

Wir verwenden type anstelle von pr zum wiederholten Würfeln:

```
repeat 10 [type 1+random 6]
```

Die zehn gewürfelten Zahlen werden damit zwar wie gewünscht in einer Zeile, aber ohne Trennzeichen ausgegeben. Wird durch die Anwendung von type auf die automatisch von LOGO bereitgestellte Formatierung verzichtet, so müssen Trennzeichen explizit vorgesehen werden. Als Trennzeichen kann beispielsweise ein Komma in Frage kommen, das nun aber ausdrücklich ausgegeben werden muß:

```
repeat 10 [type 1+random 6 type ", "]
```

Um ein Komma auszugeben, kann nicht einfach nur das Zeichen selbst niedergeschrieben werden, denn dann würde LOGO nach einer Prozedur unter diesem Namen suchen. Dem Komma muß deshalb ein Anführungszeichen vorangestellt werden.

Statt des Kommas können natürlich ebenso andere Sonderzeichen benutzt werden. Bei Trennung durch Leerzeichen sind besondere Vorkehrungen erforderlich. Das ist immer dann der Fall, wenn ein Zeichen, das in der Grammatik der Programmiersprache eine spezielle Bedeutung hat, wie ein gewöhnliches Zeichen behandelt werden soll. Die als Trennung wirkende Bedeutung des Leerzeichens wird abgeschaltet, wenn ein Backslashzeichen davorgesetzt wird:

```
repeat 10 [type 1+random 6 type "\ ]
```

Das Backslashsymbol (gespiegeltes Divisionszeichen) finden Sie beim CPC auf der Tastatur, es kann aber auch durch die Kombination CONTROL+Q erzeugt werden. Beim Joyce gibt es das Zeichen nicht auf der Tastatur, so daß Sie die Kombination ALT+Q benutzen müssen.

Das Backslashzeichen selbst wird nicht ausgegeben, denn es soll nur dafür sorgen, daß das nachfolgende Zeichen wirklich als solches Zeichen erkannt wird. Andere mögliche Bedeutungen in LOGO werden damit außer Kraft gesetzt. Ein Leerzeichen wirkt dann nicht mehr als Trennzeichen.

Statt zweimal die Anweisung type hintereinander zu wiederholen, können hier allerdings auch beide Ausgaben mit einer einzigen type-Anweisung bewerkstelligt werden, wenn type mit den entsprechenden Eingaben zusammen in runde Klammern eingeschlossen wird:

```
repeat 10 [(type 1+random 6 "\ )]
```

Hier wird dann tatsächlich noch ein weiteres Leerzeichen dazwischen gesetzt. Statt einer zehnfachen Wiederholung wollen wir uns einmal eine 10000-fache Wiederholung erlauben:

```
repeat 10000 [(type 1+random 6 "\ )]
```

Wer die Geduld dazu besitzt, kann am besten mit Hilfe eines Druckers untersuchen, ob LOGO mit Hilfe von random einen echten Würfel simulieren kann. Bei einem guten Würfel sollten alle Augenzahlen zwischen 1 und 6 mit gleicher Häufigkeit auftreten, wenn nur oft genug gewürfelt wird.

Nun wird es vielleicht doch zu langweilig, die vielen Würfe zu verfolgen. Es entsteht also manchmal das Bedürfnis, eine gerade ablaufende Aktion von LOGO zu unterbrechen.

Der unmittelbare Abbruch kann von außen durch Tastendruck verursacht werden:

- ESC (STOP beim Joyce) stoppt die laufende LOGO-Aktivität. LOGO antwortet mit der Meldung
- STOPPED!
- statt des ESC-Taste kann auch die Kombination CONTROL+G benutzt werden.

Mit diesem Eingriff wird die begonnene Aktivität, hier das 10000-fache Würfeln, vollständig abgebrochen.

Es gibt auch die Möglichkeit, falls das inhaltlich überhaupt einen Sinn ergibt, eine LOGO-Aktivität nur zeitweilig zu unterbrechen. Auch hierzu dient ein spezieller Tastendruck:

- CONTROL+Z unterbricht eine ablaufende Aktivität.
- Die Aktivität kann mit co (Abkürzung für Continue) fortgesetzt werden.

LOGO zeigt die Unterbrechung mit der Ausgabe von Pausing... ang. Später werden Sie bemerken, daß LOGO im allgemeinen bei Stopped und Pausing noch zusätzliche Angaben macht, an welcher Stelle der Abbruch bzw. die Unterbrechung erfolgte. Bei

direkter Ausführung einer Eingabezeile erscheinen deswegen vor dem Fragezeichen eckige Klammern.

In der durch die Unterbrechung eingetretene Pause können andere Aktivitäten stattfinden, ohne daß die unterbrochene Aktivität damit von ihrer Fortsetzung ausgeschlossen wird. Die gerade unterbrochene Anweisung kann sogar in dieser Pause verändert werden! Der Unterbrechungszustand wird erst durch co (Continue) wirklich aufgehoben.

Übungen

1) Es sollen zwei Würfel gleichzeitig geworfen werden. Als Ergebnis soll die Summe der erwürfelten Zahlen ausgegeben werden.

2) Es sollen Geburtsdaten aus unserem Jahrhundert mit Tag, Monat und Jahr zufällig verteilt ausgegeben werden.

Lektion 4: Programmieren

Neue Sprachelemente:
to, op

Programme:
KREISUMFANG, KREISFLÄCHE

4.1 Programmieren als Spracherweiterung

Mit LOGO arbeiten heißt, bestimmte Tätigkeiten des LOGO-Systems auszulösen. Nach dem Erscheinen der Begrüßungszeile steht ein Grundvokabular zur Verfügung, mit dem Sie Aufträge erteilen können. Für eine Programmiersprache ist dieses Vokabular schon sehr umfangreich, da LOGO von Hause aus schon viele Fähigkeiten mitbringt. Das faszinierende bei der Arbeit mit Computern ist nun aber gerade die Möglichkeit, aus bestimmten einfachen Grundtätigkeiten neue aufzubauen. Die Vielfalt denkbarer Tätigkeiten ist im Prinzip unbegrenzt.

Bevor eine neue Aktivität verwendet werden kann, muß LOGO lernen, was dabei geschehen soll. Programmieren bedeutet nichts anderes, als dem LOGO-System mitzuteilen, was es in Zukunft tun soll, wenn eine bisher unbekannte Tätigkeit aufgerufen wird.

In LOGO werden die vom Benutzer erfundenen Tätigkeiten genauso behandelt wie die im Grundvokabular vorgegebenen Fähigkeiten.

> Programmieren in LOGO heißt: Den Wortschatz von LOGO erweitern.

Wäre der Speicher Ihres Rechners unbegrenzt, könnten Sie im Prinzip bei jeder LOGO-Sitzung neue LOGO-Tätigkeiten erklären und den erweiterten Wortschatz am Ende der Sitzung auf Diskette abspeichern. Vor dem nächsten Arbeiten mit LOGO könnten die in allen vorangegangenen Sitzungen entstandenen

LOGO-Erweiterungen eingeladen werden. Ihr LOGO-System würde also ständig wachsen und der Umfang des Wortschatzes sich immer mehr erweitern.

Sinnvoll arbeiten kann man nur mit einem begrenzten System. Das ist auch ausreichend, denn für eine bestimmte Anwendung braucht man schließlich nur eine darauf bezogene Auswahl von Tätigkeiten.

4.2 Der Umgang mit Prozeduren

Die Tätigkeiten in LOGO heißen auch Prozeduren (Procedures). Alle Prozeduren, sowohl die im Grundwortschatz von LOGO stets vorhandenen als auch die durch Programmieren hinzugekommenen, werden vom LOGO-System in gleicher Weise behandelt. Wie Operationen mit Hilfe von Prozeduren richtig in Gang gesetzt werden, läßt sich deshalb am einfachsten an den Ihnen bereits bekannten LOGO-Vokabeln wie pr, make, repeat, random, usw. studieren.

Zuerst wird die Tätigkeit benannt, die ausgeführt werden soll. Der Aufruf einer Prozedur beginnt also grundsätzlich mit dem zugehörigen Prozedurnamen, der ohne einleitendes Sonderzeichen geschrieben wird. Manche Tätigkeiten benötigen keine weiteren Angaben.

Beispiele hierfür, die bereits betrachtet wurden, sind etwa:

```
copyon, co
```

In den meisten Fällen benötigen Prozeduren eine oder mehrere Eingaben. Hierfür seien einige Beispiele genannt, deren Bedeutung Sie schon kennen:

```
pr "Hallo
make "n 2
repeat :n [pr 1+random 6]
```

Im ersten Beispiel wird zum Ausdrucken mit pr natürlich die auszugebenden Daten als Eingabe für die mit pr bezeichnete Tätigkeit benötigt. Hier soll z.B. das Wort Hallo gedruckt werden. Weil Hallo ein Wort und nicht Prozedurname ist, müssen zur Unterscheidung die Anführungszeichen am Anfang gesetzt werden. An der Stelle von "Hallo kann aber auch eine Zahl, wie bei

```
pr 3.14
```

oder der Name einer Tätigkeit, welche die Ausgabe als Ergebnis liefert, stehen, z.B.

```
pr random 6.
```

Weil random selbst wieder eine Angabe über den Bereich benötigt, aus dem die Zufallszahlen herausgegriffen werden sollen, ersetzt bei diesem Beispiel der vollständige Aufruf random 6 das Wort "Hallo beim ersten Beispiel. Die Zahl 6 ist hier eine Eingabe von random. Um die Zugehörigkeit einer Eingabe festzustellen, muß man nach links den letzten unmittelbar vorausgegangenen Prozedurnamen suchen.

Damit eine eindeutige Zuordnung der Eingaben bei der Ausführung der Tätigkeit sichergestellt ist, muß die Reihenfolge der Eingaben genau beachtet werden.
Auch die Anzahl der Eingaben ist normalerweise festgelegt und muß dementsprechend eingehalten werden. Ausnahmen bilden einige im Grundwortschatz von LOGO vorhandenen Vokabeln wie pr oder type.

pr erfordert normalerweise eine Eingabe, es sind aber auch mehrere zulässig, wie bei

```
(pr "pi= 3.14159)
```

Bei Abweichung von der Standardanzahl müssen dann runde Klammern um den gesamten Prozeduraufruf gesetzt werden.

Als Eingabe kommt aber auch der Inhalt einer Schublade in Frage, d.h. der Wert des durch ein Wort bezeichneten Platzhalters:

```
pr :mwstsatz
```

LOGO sieht nach, ob der Name mwstsatz bekannt ist. Falls das Programm diesen Namen nicht findet, antwortet LOGO mit

```
mwstsatz has no value
```
(mwstsatz hat keinen Wert).

Wurde der Name mwstsatz dagegen zuvor erklärt, so gibt LOGO den gültigen Wert auf dem Bildschirm aus.

```
make "mwstsatz .14
pr :mwstsatz
```

Beachten Sie den Unterschied zu

```
pr "mwstsatz
```

In diesem Fall wird das Wort mwstsatz als solches auf dem Bildschirm ausgegeben!

Betrachten wir nun als zweites Beispiel den Gebrauch von make genauer. make ist eine Prozedur mit zwei Eingaben: die erste ist ein Wort, mit dem ein Name benannt wird, die zweite der Inhalt der Schublade gleichen Namens. Die Bedeutung beider Eingaben ist eindeutig durch die Reihenfolge festgelegt.

Probieren Sie z.B. folgendes aus:

```
make "Vorname "Hans
pr :Vorname
```

LOGO gibt dann wie beabsichtigt das Wort HANS aus. Wenn Sie bei make beide Eingaben vertauscht haben, so wird genau umgekehrt das Wort VORNAME unter dem Namen HANS gespeichert.

Ein wichtiges Kennzeichen bei Eingaben läßt sich an einem weiteren, schon bekannten Beispiel demonstrieren. In der vorangegangenen Lektion wurde das LOGO-Wort repeat zur Wiederholung von Eingabezeilen vorgestellt. repeat ist eine Prozedur mit zwei Eingaben:

```
repeat :n [pr random 6]
```

Die zweite Eingabe ist jetzt eine *Liste*, was an den eckigen Klammern deutlich wird. An Stelle dieser Liste könnte auch der Inhalt einer Schublade treten, falls der Inhalt wirklich vom Typ einer Liste ist.

Das vorangegangene Beispiel könnte auch ersetzt werden durch:

```
make "aktion [pr random 6]
repeat :n :aktion
```

Dieses Beispiel ist übrigens schon eine bemerkenswerte Eigenschaft von LOGO! Beim Aufruf von repeat wird die zu wiederholende Befehlsfolge nicht nochmal hingeschrieben. LOGO holt sich die Anweisungen als Wert eines Namens, also aus der entsprechenden Schublade, deren Inhalt selbst manipuliert werden kann. In den meisten Programmiersprachen ist diese Operation überhaupt nicht möglich.

Bei repeat unterscheiden sich die beiden Eingaben hinsichtlich des Typs. Die erste Eingabe muß eine Zahl oder ein Prozeduraufruf mit einer Zahl als Ergebnis sein, während die zweite Eingabe eine Liste oder etwas, was zu einer Liste führt, sein soll. Wird statt einer Zahl eine Liste oder umgekehrt eingegeben, z.B.

```
repeat [10] [pr random 6]
```

dann antwortet LOGO mit einer Fehlermeldung:

```
repeat doesn't like [10] as input
```

4.3 Prozeduren mit Ergebnis

Mit make wird eine Tätigkeit ausgelöst, deren Wirkung man zunächst nicht bemerkt. make benötigt zwei Eingaben, es werden aber keine Ergebnisse in der Form von Ausgaben zurückgegeben.

Ganz anders bei der Eingabe von

```
random 6
```

bei der LOGO beispielsweise mit

```
1
```

antwortet; random erzeugt also ein Ergebnis. Das ist ja der Sinn der Sache, dennn schließlich soll der Rechner nicht nur in seinem Inneren still vor sich hin würfeln, sondern die gewürfelte Zahl auch als Ergebnis präsentieren. Meist möchte man damit noch weitere Tätigkeiten durchführen wie z.B. in der Zeile

```
pr 1+random 6.
```

Hier soll zum Ergebnis noch 1 addiert und die Summe ausgedruckt werden. Umgekehrt wird in der Zeile

```
pr make "n 2
```

kein Ergebnis durch make erzeugt, und die Ausgabeprozedur pr findet nichts zur Ausgabe vor. LOGO antwortet deswegen mit der Fehlermeldung:

```
[make "n 2] didn't output to pr
(make hat kein Ergebnis erzeugt)
```

Typische Beispiele für Tätigkeiten mit Ausgabe sind auch die mathematischen Funktionen, z.B.

```
sin 30
```

Es gibt also auch Tätigkeiten, die einen bestimmten Wert als Ergebnis ausgeben, der dann weiter verarbeitet werden kann.

Beachten Sie dabei:

- Eine Prozedur kann mehrere Eingaben haben.
- Eine Prozedur hat dagegen keinen oder genau einen Ausgabewert!

Prozeduren, die genau ein Ergebnis ausgeben, werden oft Funktionen genannt. Insofern ist LOGO eine Programmiersprache, die auf die Verwendung von Funktionen ausgerichtet ist. Die Tatsache, daß als Ergebnis nur eine und nicht mehrere Ausgaben möglich sind, ist aber nur auf den ersten Blick eine Einschränkung. Mehrere Ausgaben können nämlich zu einer einzigen Liste zusammengefaßt werden, so daß dann am Ende daraus wieder nur ein Ausgabewert gebildet wird. Darauf wird im Detail noch bei der Listenverarbeitung eingegangen werden.

4.4 Schreiben einfacher Programme

Nachdem ausführlich der Umgang mit Prozeduren an Hand einiger im Grundwortschatz des LOGO-Systems vorhandener Vokabeln dargestellt wurde, kann nun endlich mit ersten Spracherweiterungen begonnen werden.

Bisher wurden stets Tätigkeiten des LOGO-Systems ausgelöst, und zwar solche, die schon definiert sind. Beim Einführen neuer Vokaabeln muß LOGO diese neu erlernen. Dieser Vorgang ist grundsätzlich ein anderer Zustand des Systems, verglichen mit dem Auslösen bereits eingebauter Aktivitäten. LOGO muß deshalb dazu in einen speziellen Lernmodus versetzt werden. Wenn Sie schon einmal mit programmierbaren Taschenrechnern gearbeitet haben, ist Ihnen diese Umschaltung in einen solchen speziellen Modus vertraut.

Diese Tätigkeit nennt man bei Computern *Editieren* von Programmen. Das Lernen oder Editieren kann sich sowohl auf

etwas vollständig Neues beziehen, als auch bedeuten, daß dabei etwas bereits Gelerntes verändert, insbesondere korrigiert werden soll. Bei DR LOGO besteht die Möglichkeit, eine neue Prozedur im Textfenster zeilenweise einzugeben. Diese Variante soll hier zunächst besprochen werden. Im allgemeinen, insbesondere zur Korrektur bzw. Veränderung bereits bestehender Prozeduren geschieht dies in einem eigenen, dem sogenannten Edit-Fenster. Darauf wird in der folgenden Lektion eingegangen.

Beispiel 1: LOGO soll das Würfeln erlernen. Zu diesem Zweck kennen Sie schon die Eingabezeile

```
1+random 6
```

Hier muß die Anweisung zum Würfeln aus vorhandenen Vokabeln zusammengesetzt werden. Einfacher wäre es, wenn LOGO direkt auf das Kommando WUERFELN eine entsprechende Antwort geben könnte. Um LOGO die Tätigkeit WUERFELN beizubringen, geben Sie ein:

```
to WUERFELN
```

Im Englischen heißt laufen 'to walk'; der Beginn mit 'to' leitet sich vom englischen Infinitiv ab. Frei Übersetzt signalisiert die eingegebene Zeile etwa 'jetzt soll das neue Verb würfeln erklärt werden'.

Manchen wird hier vielleicht der sprachliche Mischmasch aus der englischen 'Muttersprache' von LOGO und deutschen Wörtern stören. Tatsächlich gibt es auch deutschsprachige Versionen von LOGO. Nun ist dieses Durcheinander von Vokabeln aus deutschen und englischen Wörtern bei Programmiersprachen Gewohnheit geworden, wenn auch ein feinfühliges Sprachempfinden dabei Qualen erleiden mag.

Die Schlüsselwörter sind heutzutage bei Programmiersprachen durchgängig der englischen Sprache entlehnt, genauso wie beispielsweise im internationalen Flugverkehr ein 'gebrochenes Englisch' zur Verständigung dient. Wollte man für alle natürlichen Sprachen eine der jeweiligen Muttersprache angepaßte

Version entwickeln, würde dieses Vorhaben mehr Probleme schaffen als Nutzen stiften.

Bei LOGO ist der Wunsch nach einer deutschsprachigen Version allerdings größer als bei anderen Programmiersprachen, weil LOGO auch als eine für Kinder leicht zu erlernende Sprache angesehen wird. Andererseits sind es doch nur relativ wenige Vokabeln der englischen Sprache, deren Bedeutung man sich durch praktischen Umgang am besten einprägt. Die wirklichen Schwierigkeiten beim Programmieren treten sicher nicht beim Verständnis einfacher englischer Vokabeln auf!

Doch zurück zum Erlernen des Tätigkeitswortes WUERFELN!

Nach Übergabe der Zeile to WUERFELN durch Drücken der Return-Taste zeigt Ihr Rechner einen neuen Zustand durch ein anderes "Prompten", d.h. ein anderes Zeichen zur Verdeutlichung der Eingabebereitschaft, an. Anstelle des Fragezeichens erscheint ein >-Zeichen!

Geben Sie nun ein:

```
op 1+random 6
```

op (Abkürzung von Output) steht für Ausgabe. Op zeigt an, daß die Tätigkeit 'WUERFELN' ein Ergebnis hat, genauso wie etwa random, cos, usw. op veranlaßt aber auch gleichzeitig die Beendigung der Tätigkeit WUERFELN. Es hat deswegen keinen Sinn, nach der Zeile mit op noch weitere Zeilen schreiben zu wollen.

Damit ist 'WUERFELN' vollständig erklärt. Es muß nur noch das Ende markiert werden. Geben Sie dazu ein:

```
end
```

DR LOGO merkt nun, daß der Lernvorgang, nämlich das Erlernen der Tätigkeit WUERFELN, beendet ist. Das >-Zeichen am Anfang der Zeile verschwindet. Es erschent nunmehr eine Erfolgsmeldung:

```
WUERFELN defined
```
(Die Tätigkeit WUERFELN ist definiert.)

Damit ist das Wort WUERFEL, d.h. die Prozedur mit dem Namen WUERFELN verfügbar.

Beachten Sie aber bitte, daß das LOGO-System beim Editieren nicht prüft, ob die Erklärung der Tätigkeit WUERFELN grammatikalisch korrekt und inhaltlich sinnvoll ist. Eventuell vorhandene Fehler können somit erst bei der Ausführung auftauchen.

Prüfen Sie nun nach, ob mit WUERFELN auch wirklich gewürfelt werden kann, indem Sie einfach diesen Term eingeben. Tatsächlich hat die Eingabe von WUERFELN die gleiche Wirkung wie die Eingabezeile 1+random 6!

Weitere Beispiele für einfache Programme

Kreisberechnungen

Wie beim Programmieren des Wortes WUERFELN lassen sich einfache Programme für die Berechnung von Kreisumfang und Kreisfläche als neue LOGO-Vokabeln KREISUMFANG, KREISFLAECHE einführen.

Nach der Eingabe von 'to KREISUMFANG' ist dann die Formel für den Kreisumfang als Ausgabe der Prozedur KREISUMFANG anzugeben:

```
to KREISUMFANG
op 2* :pi*:r
```

Entsprechend für die Kreisfläche:

```
to KREISFLAECHE
op :pi*:r*:r
```

Beide Formeln machen davon Gebrauch, daß die Kreiszahl pi von DR LOGO unter dem Namen pi bereits zur Verfügung gestellt wird.

```
make "pi 3.14159265
```

Falls keine Tippfehler aufgetreten sind, sollten die beiden neuen LOGO-Vokabeln KREISUMFANG und KREISFLAECHE die gleiche Wirkung wie die entsprechenden Eingabezeilen haben.

Vielleicht antwortet LOGO aber mit der Fehlermeldung:

```
I don't know how to r in KREISUMFANG: op 2 * :pi * r
```

Es ist keine Tätigkeit unter dem Namen r bekannt; LOGO weiß also nicht, wie "r" ausgeführt werden soll. Der Prozedurname r wird in der Prozedur KREISUMFANG verwendet und zwar in der Zeile op 2 * :pi * r. Der aufgetretene Fehler ist leicht zu erkennen: Aufgrund eines Tippfehlers fehlt der Doppelpunkt vor r, der LOGO mitteilt, daß der Wert von r, und nicht eine Prozedur r gemeint ist.

Wie Sie sehen, nimmt DR LOGO ausführlich Stellung. In der Meldung erscheint, wo welches Problem aufgetreten ist. Auch die Zeile, in der der Fehler erkannt wird, ist zu sehen. In dieser Situation kennt das System die Stelle, an der die Schwierigkeit entdeckt worden ist. Das können Sie nutzen, indem Sie

```
ed
```

eingeben. Als Reaktion darauf geht das LOGO-System in einen speziellen Zustand über. Sie sehen anschließend den Text des Programms KREISUMFANG vor sich, wobei der urprüngliche

Inhalt des Bildschirms verschwunden ist und am unteren Bildschirmrand eine Fußzeile mit der Bezeichnung

```
Edit
```

entstanden ist. In diesem Fall steht der Cursor auf dem als problematisch erkannten Buchstaben r in der zweiten Zeile.

Was jetzt zu sehen ist, gibt die Stelle an, bei der das LOGO-System wegen des aufgetretenen Fehlers die Weiterarbeit verweigert hat. Tatsächlich kann es vorkommen, daß der eigentliche Fehler schon etwas früher passiert ist, wenn nämlich ein Programm selbst von einem anderen aufgerufen worden ist und dabei etwas nicht in Ordnung war.

Wie die Korrektur durchzuführen und das verbesserte Programm dem LOGO-System wieder zu übergeben ist, wird in der nächsten Lektion behandelt.

Lektion 5: Programme Editieren

Neue Sprachelemente:
ed, po, edall, poall, pops, pons, pots, er, ern, erall, Editor-Kommandos, Globale und lokale Namen

5.1 Korrektur von Programmen

Programme können im Prinzip korrigiert werden, indem man sie nochmals ohne Fehler eintippt. Das ist nicht gerade bequem und birgt vor allem die Gefahr neuer Fehler in sich. DR LOGO bietet Ihnen daher einen komfortablen Editor. Darunter versteht man den Teil des LOGO-Systems, der zum Gestalten der Progammtexte vorgesehen ist.

Auch wenn Sie neue Prozeduren schreiben, empfiehlt es sich, den Text der Prozedur von vorne herein mit diesem Editor zu schreiben. Nur wenn Sie eine fertige Prozedur abtippen, von der Sie annehmen können, daß keine Korrekturen notwendig sind, ist es zweckmäßig, direkt den Programmmtext im Textfenster einzugeben.

Der Editor ist ein eigenes Textverarbeitungsprogramm, allerdings nicht so komfortabel wie ein für diesen Zweck gedachtes kommerzielles Textsystem. Programmtexte können damit verändert und ergänzt werden. Sie können einzelne Prozeduren oder auch alle im Arbeitsspeicher befindlichen Programmtexte gleichzeitig bearbeiten. Sogar die Werte, die Sie bestimmten Namen zugewiesen haben, können damit beeinflußt werden. In der Regel sollten Sie aber immer nur eine einzige Prozedur editieren.

In den Editiermodus kann man - wie bereits beschrieben - gelangen, indem man bei einer Fehlermeldung ed eingibt. Die Bearbeitung einer Prozedur wird mit dem Kommando ed eingeleitet:

```
ed "KREISUMFANG
```

Es soll die Prozedur mit dem Namen KREISUMFANG bearbeitet werden. Beachten Sie, daß die Anführungszeichen vor Kreisumfang notwendig sind. Das entspricht der bei make angewandten Regel. Wenn Sie dagegen einen Prozedurtext mit to KREISUMFANG im Textfenster eingeben, wird der Prozedurname ohne Anführungszeichen geschrieben.

Nach ed wird der gesamte Bildschirm zum Editieren bereitgestellt; es wird dann im Edit-Fenster gearbeitet. Das zuvor sichtbare Textfenster wird dadurch gelöscht, was übrigens auch für das Grafikfenster gilt.

Wenn die angesprochene Prozedur bereits existiert, erscheint der Programmtext. Wird die Tätigkeit zum ersten Mal angesprochen, erscheint dagegen nur

```
to KREISUMFANG
end
```

Hinter ed ist nur der Prozedurname, eingeleitet durch Anführungszeichen, zu setzen. LOGO fügt dann von sich aus ein end hinzu. Ihre Aufgabe ist es also, den Programmtext zwischen Prozedurnamen und dem Ende der Prozedur einzugeben.

Sie können bei einem Editiervorgang mehrere Prozeduren hintereinander schreiben. Jede muß mit der Zeile end abgeschlossen werden. DR LOGO gibt Ihnen ein end vor, die übrigen müssen Sie selbst schreiben.

Wie wird der Editiermodus wieder verlassen?

Hierzu wird die Kombination CONTROL+C benutzt.

CONTROL+C beendet das Editieren.

Beim Schneider CPC kann auch stattdessen die Taste

COPY

benutzt werden. (Beim Joyce gibt es auch eine Taste mit dieser Aufschrift, die dort aber die Funktion von CONTROL+Y übernimmt.)

LOGO reagiert mit dem Kommentar:

```
KREISUMFANG defined
```
(KREISUMFANG nun definiert.)

Falls Sie mit dem Editiervorgang völlig unzufrieden sind, können Sie die ganze Sache auch mit ESC (STOP) abbrechen. Damit wird die gesamte Editierung annulliert.

Cursorbewegungen

Beim Editieren von Programmzeilen sind Bewegungen des Cursors in alle vier Richtungen möglich, also auch nach oben und unten. Zusätzlich kann der Cursor mit CONTROL+A an den Anfang der Zeile, mit CONTROL+E an das Ende der Zeile bewegt werden.

Wird eine Programmzeile durch den rechten Bildschirmrand unterbrochen, so markiert ein Ausrufungszeichen am Ende der Bildschirmzeile die Fortsetzung in der nächsten.

Eine Programmzeile wird durch Drücken der RETURN-Taste beendet. Die in der Bildschirmzeile dann noch folgenden Leerzeichen gehören nicht zum Programmtext. LOGO-Programme können mit Hilfe von Zeilen optisch gegliedert werden. Diese Gliederung ist im Hinblick auf die grammatikalischen Regeln von LOGO meistens unwichtig, macht den Programmtext aber besser lesbar. Auf alle Fälle sollten Sie lange Programmzeilen vermeiden, wenn sie nicht notwendig sind.

Wenn eine Bildschirmzeile durch RETURN abgeschlossen wird, ist das Ende durch die aktuelle Cursorposition gegeben. Folgt dann auf dem Bildschirm noch weiterer Programmtext hinter dem Cursor, wird der Rest automatisch in die nächste Zeile geschoben, was dann auch auf dem Bildschirm deutlich wird. Auf diese Weise kann auch eine längere Programmzeile in mehrere Zeilen zerlegt werden.

Der LOGO-Editor erlaubt darüber hinaus Cursorbewegungen über größere Entfernungen hinweg.

- CONTROL+R setzt den Cursor an den Anfang des Programmtextes, der sich gerade in der Bearbeitung befindet.
- CONTROL+X setzt den Cursor umgekehrt ans Ende des bearbeiteten Textes.
- CONTROL+L sorgt dafür, daß die gerade angesprochene Textzeile in die Mitte des Bildschirms kommt. Damit kann die Textumgebung einer Zeile besser sichtbar gemacht werden.
- Mit CONTROL+V wird im Text um eine Seite vorwärts geblättert,
- mit CONTROL+U um eine Seite zurückgeblättert.

Löschen

Zum Löschen gibt es die Tasten DEL und CLR wie bei den einfachen Eingabezeilen. Mit CONTROL+K wird der Rest der Zeile gelöscht und zwar beginnend mit der Cursorposition.

Die Taste DEL löscht das vor dem Cursor stehende Zeichen. Wenn der Cursor am Anfang einer Zeile steht, so wird das intern vorhandene Trennsymbol zur Markierung eines Zeilenendes gelöscht. Damit wird die Programmzeile, an deren Beginn der Cursor steht, an die vorangegangene Zeile angehängt. Auf diese Art wird die mit der Taste RETURN erreichte Trennung zweier Programmzeilen gerade wieder aufgehoben. Auf diese Weise können auch leere Zeilen entfernt werden.

Einfügen von Zeichen und Zeilen

Beim Editieren von Programmen können ganze Zeilen eingefügt werden. Hierzu gibt man ein:

CONTROL+O (Eröffne neue Zeile)

Der auf die Cursorposition folgende Programmtext wird nach hinten verschoben, so daß Platz für den Beginn einer neuen Zeile entsteht.

Das Einfügen von Zeichen geschieht wie bei den Eingabezeilen. Die eingetippten Zeichen werden an die Cursorposition geschrieben, der vorhandene Programmtext wird zuvor jeweils um eine Stelle nach hinten geschoben und dabei nicht überschrieben. Mit der schon bei den Eingabezeilen besprochenen Tastenkombination CONTROL+Y kann der gerade im Eingabepuffer befindliche Textteil eingefügt werden, insbesondere auch der versehentlich gelöschte Zeilenrest.

Beim Programm KREISUMFANG aus der vorangegangenen Lektion fehlte z.B. einmal der Doppelpunkt vor dem Namen r (Radius). Zur Korrektur wird der Cursor auf das r gestellt und der Doppelpunkt eingegeben. Mit CONTROL+C (bzw. COPY) wird das korrigierte Programm in den LOGO-Wortschatz übernommen, wobei der alte Programmtext automatisch überschrieben wird.

5.2 Programme mit Eingaben

In Lektion 2 wurde als Beispiel die Berechnung der Mehrwertsteuer betrachtet, wobei der Computer die Rechnung sofort nach der Eingabe der Zeile ausführte. Lösen wir diese Aufgabe nun mit einem Programm:

```
to MEHRWERTSTEUER
op :mwstsatz * :preis
end
to BRUTTOPREIS
op ( 1 + :mwstsatz ) * :preis
end
```

Beide Prozeduren benötigen den Wert des Steuersatzes, der also mit einer Zeile der Form

```
make "mwstsatz 0.14
```

unter diesem Namen abgelegt sein muß. Es ist sicher recht sinnvoll, für alles folgende den Wert von mwstsatz einmal dem LOGO-System mitzuteilen, um nicht jedesmal die Ziffernfolge selbst eingeben zu müssen.

Der für den Steuersatz stehende Name mwstsatz sollte allen Tätigkeiten, die innerhalb dieser LOGO-Sitzung stattfinden, zugänglich sein. Solche Namen werden als globale Namen bezeichnet. Global heißt dabei, daß sie nach einmaliger Festlegung Teil des LOGO-Systems sind und darauf bei allen LOGO-Aktivitäten zurückgegriffen werden kann.

Daß es sich bei dem betrachteten Beispiel um den Wert des gerade geltenden Mehrwertsteuersatzes handelt, ist hierfür unerheblich. Ein Physiker würde vielleicht gern wichtige Naturkonstanten unter sinnvoll gewählten Namen verfügbar halten. Für einen Arzt ist dagegen vielleicht wichtig, welchen Gebührensatz er seinen Klienten abverlangt.

Damit die Prozeduren für die Steuerberechnung einen Wert ergeben können, muß natürlich auch der des Preises bekannt sein. In der bisher vorliegenden Form sollte also vor dem Aufruf von MEHRWERTSTEUER, bzw. BRUTTOPREIS der Preis

```
make "preis 156.30
```

definiert werden.

Wie der Mehrwertsteuersatz wird der Preis damit zu einem globalen Namen. Alle folgenden LOGO-Aktivitäten können unter diesem Namen auf die Zahl 156.30 solange zugreifen, bis durch einen erneuten Aufruf von make mit dem Namen preis dessen Wert möglicherweise verändert wird.

Diese Behandlung des speziellen Wertes 156.30 erscheint unnötig und schwerfällig. Die Zahl 156.30 hat keine allgemeingültige Bedeutung, die zu einem festen Platz für alle folgenden LOGO-Tätigkeiten berechtigt. Schwerfällig ist das Verfahren deshalb, weil bei jedem neuen Preis eine ganze Zeile der Form

```
make "preis 520.10
```

eingegeben werden muß.

Zweckmäßiger wäre hier ein Methode wie bei einer mathematischen Funktion, etwa

```
cos 30
```

Dabei muß nur der Eingabewert 30 verändert werden, um einen neuen Wert für die Kosinusfunktion zu bekommen. Die spezielle Zahl 30 hat auch nur im Augenblick des Funktionsaufrufes eine besondere Bedeutung. Die Programme zur Steuerberechnung wären deshalb leichter handzuhaben, wenn der aktuelle Preis als Eingabe der Prozeduren MEHRWERTSTEUER, bzw. BRUTTOPREIS ähnlich wie bei der eingebauten Funktion cos behandelt werden könnte.

Um die beiden Programme zu verändern, wählen Sie den LOGO-Editor mit

```
edall
```
(Editieren des Arbeitsspeichers)

an. Damit werden alle im Arbeitsspeicher vorhandenen Tätigkeiten und auch die Namen für die Editierung bereitgestellt. So finden sich beispielsweise Zeilen der folgenden Form vor:

```
make "mwstsatz 0.14
```

Man kann auf diese Weise auch den Zahlenwert von MWSTSATZ oder anderer globaler Namen eingeben oder verändern. Die Benutzung des Editors ist für diesen Zweck praktisch, wenn die "make-Zeile" relativ lang ist und nur wenig daran geändert werden soll. Man kann den Namen allerdings auf diese Weise nicht löschen.

Ändern Sie nun die Titelzeilen der beiden kleinen Prozeduren um in

```
to MEHRWERTSTEUER :preis
to BRUTTOPREIS :preis
```

Die Editierung wird mit CONTROL+C (Joyce) oder COPY (CPC) abgeschlossen. Danach können MEHRWERTSTEUER und BRUTTOPREIS wie mathematische Funktionen benutzt werden.

```
MEHRWERTSTEUER 100   Antwort: 14
BRUTTOPREIS 100      Antwort: 114
```

LOGO hat jetzt zwei neue mathematische Funktionen erlernt.

5.3 Globale und lokale Namen

Die speziellen Zahlenwerte 156.30, 100 ... haben nur dort Bedeutung, wo sie im Prozeduraufruf eingegeben werden. Im Programmtext werden sie unter dem Namen preis verwendet. Welchen Wert hat jetzt preis beispielsweise nach den drei oben angegebenen Zeilen?

Überprüfen Sie den Wert mit:

```
pr :preis
```

Die Antwort von LOGO lautet aber:

```
preis has no value (Preis hat keinen Wert)
```

Für den Namen preis gibt es keinen Wert, obwohl er doch in den beiden Prozeduren MEHRWERTSTEUER und BRUTTOPREIS vorkommt. Da beide Prozeduren richtige Werte liefern, muß der Wert von preis innerhalb der Tätigkeiten auch tatsächlich vorhanden sein. Der Name preis ist also offensichtlich innerhalb der Prozeduren, die preis als Eingabewert vorsehen, bekannt - außerhalb aber nicht. Solche Namen heißen lokale Namen.

Möglicherweise haben Sie die Fehlermeldung nicht bekommen und LOGO hat stattdessen einen Wert ausgegeben. Das kommt dann daher, daß Sie in derselben Sitzung schon einmal eine Zeile make "preis ... eingegeben haben. Überzeugen Sie sich davon, indem Sie eingeben:

```
edall
```

Wenn Sie tatsächlich bei pr :preis einen Wert und keine Fehlermeldung erhalten haben, sollte nach edall eine Zeile der Form make "preis ... im Edit-Fenster erscheinen, wobei diese Zeilen hinter den Programmtexten stehen. Stattdessen kann man die gesuchte Zeile auch gezielt mit ed "preis herausgreifen. Die Zeile make "preis ... verschwindet aus dem Arbeitsspeicher, nachdem mit

```
ern "preis (steht für erase name "preis)
```

der globale Name preis gelöscht worden ist. Sie können sich dann davon überzeugen, daß die Zeile nach dem Aufruf von BRUTTOPREIS nicht wieder auftaucht.

Global-lokal zeigt an, wann das LOGO-System die betreffenden Namen verarbeiten kann:

- GLOBALE NAMEN sind für alle LOGO-Aktivitäten verfügbar.
- LOKALE NAMEN haben nur innerhalb einer Prozedur Geltung.

Mit "alle LOGO-Aktivitäten" ist natürlich alles gemeint, was in einer LOGO-Sitzung in Gang gesetzt wird, nachdem der globale Name erklärt wurde. Man kann globale Namen auch wieder löschen, gelöschte Namen sind für LOGO dann nicht mehr vorhanden.

Die Gültigkeit lokaler Namen innerhalb einer einzigen Prozedur scheint ihre Anwendung sehr einzuschränken. Das ist aber durchaus nicht der Fall, weil Prozeduren selbst wieder andere aufrufen können. LOGO kennt einen lokalen Namen solange, bis das Ende der Prozedur, in der der lokale Name eingeführt worden ist, erreicht wird.

Die Bedeutung der Unterscheidung in globale und lokale Namen wird erst später bei komplizierteren Programmbeispielen deutlich. Zunächst ist festzuhalten:

Eingaben von Prozeduren sind lokale Namen.

5.4 Ausgabe des erweiterten Wortschatzes

Den Grundwortschatz von LOGO und seine Bedeutung müssen Sie im Zweifelsfall in einer Kurzfassung der LOGO-Wörter (siehe Anhang) nachschlagen. Wenn sich der Wortschatz durch Programmieren, bzw. durch die Einführung globaler Namen erweitert hat, muß man immer wieder den erreichten Stand der Erweiterung einsehen können.

Diesem Ziel dient die LOGO-Vokabel

po (Abkürzung für Printout)

po erfordert als Eingabe den Prozedurnamen oder auch eine aufzählende Liste von Prozedurnamen. Der Programmtext der angegebenen Prozedur wird dann im Textfenster ausgegeben. Wie beim Aufruf des Editos gibt es hierzu noch Varianten:

poall (für Printout All)

erklärt sich selbst: LOGO schreibt alle vorhandenen Prozeduren und globalen Namen auf den Bildschirm. Dabei achtet das LOGO-System auf die Größe des Bildschirms und unterbricht die Ausgabe, wenn eine Seite geschrieben ist, so daß der ausgegebene Text nicht am oberen Bildschirmrand wieder entschwindet, bevor sie ihn lesen konnten. Nach Drücken einer Taste wird dann die jeweils nächste Seite vollgeschrieben.

Wenn Sie einen Drucker angeschlossen haben, können Sie mit

```
copyon poall
```

einen Ausdruck des gesamten Arbeitsspeichers auf Papier bringen. Wenn man die Ausgabe nur auf dem Bildschirm verfolgt, ist es meist zweckmäßig, die Ausgabe mit po von vorne herein zu begrenzen.

Interessiert man sich nur für Prozeduren oder nur für Namen, so teilt man dies mit durch:

pops (für Printout Procedures)
pons (für Printout Names)

Beim Ausgeben der Namen werden nicht nur diese selbst, sondern auch die zugehörigen Werte auf dem Bildschirm sichtbar, z.B.

```
mwstsatz is 0.14
```

Die Einzelausgabe von Prozeduren ist ebenfalls möglich:

```
po "MEHRWERTSTEUER

po [MEHRWERTSTEUER BRUTTOPREIS]
```

Manchmal will man nur wissen, welche neuen LOGO-Vokabeln, d.h. welche Prozedurnamen erklärt wurden. Die Prozedurnamen werden als Titles bezeichnet:

```
pots (für Printout Titles)
```

damit erhält man sehr leicht eine Übersicht über die selbstdefinierten LOGO-Aktivitäten.

5.5 Löschen von Programmen und Namen

Gelegentlich will man Erweiterungen des Wortschatzes auch wieder aus dem System entfernen. Manchmal ist es sinnvoller, ein neues Programm zu schreiben, anstatt ein vorhandenes zu korrigieren. Irgendwann stößt jeder Programmierer einmal an die Speichergrenzen und muß dann versuchen, durch Beseitigung nicht mehr benötigter Objekte Platz zu schaffen. Zum Löschen dient die LOGO-Vokabel

```
er
```

er benötigt eine Eingabe, z.B.

```
er "MEHRWERTSTEUER
er "BRUTTOPREIS
```

Bereits erwähnt wurde die Möglichkeit, einzelne Namen löschen zu können. Hierfür gibt es das LOGO-Wort ern:

```
ern "mwstsatz
```

Man kann auch den Inhalt des gesamten Arbeitsspeichers auf einnmal löschen. (Größte Vorsicht!)

```
erall (für Erase All)
```

Damit werden sowohl alle Namen wie auch alle Prozeduren entfernt.

Leerzeichen als Trennzeichen

Vielleicht ist Ihnen aufgefallen, daß bei der Formulierung von Programmen Leerzeichen mit unterschiedlicher Häufigkeit benutzt wurden.

```
op (1+:mwstsatz)*:preis
op ( 1 + :mwstsatz ) * :preis
```

Dies sind zwei Zeilen, die zum gleichen Ergebnis führen. In der zweiten Zeile wird alles, was nicht zu einem Objekt gehört, durch Leerzeichen voneinander getrennt. Das entspricht der allgemein für LOGO geltenden Regel:

> Das Leerzeichen dient in LOGO als Trennzeichen. Solange Zeichen in einer Zeichenkette nicht durch Leerzeichen voneinander getrennt werden, gehören sie zusammen und stellen für LOGO ein einziges Objekt dar.

Daß Zeichenfolgen zu getrennten Begriffen gehören, ist bei Sonderzeichen aus der Bedeutung zu erkennen. Bei 1+:mwstsatz sind die Zahl 1 und der Wert von mwstsatz als Operanden für die Addition erkennbar. LOGO geht in solchen Fällen nachsichtig mit dem Anwender um, d.h. die Trennung wird auch ohne Leerzeichen richtig erkannt. Sie werden aber feststellen, daß beim Editieren vom Editor selbstständig Leerzeichen in Ihren Programmtext eingefügt werden, wenn das System das für notwendig hält. Sollen Sonderzeichen, dazu gehören neben den Operationszeichen auch Klammern, Anführungszeichen und das Semikolon, ihre besondere Funktion nicht ausüben, muß das Backslashzeichen vorangestellt werden.

Ratsam ist es aber dennoch, eher ein Leerzeichen zuviel als zuwenig zu schreiben. Ein wichtiger Sonderfall ist dagegen das Minuszeichen.

```
MEHRWERTSTEUER -100
MEHRWERTSTEUER - 100
```

In der ersten Zeile wird ein negativer Wert (-100) als Preis verwendet, ob das sinnvoll sein kann sei dahingestellt, und in der zweiten wird das Minuszeichen dagegen als Operationszeichen gedeutet, weil es von der Ziffernfolge 100 durch ein Leerzeichen getrennt ist. Das führt hier zu einer Fehlermeldung.

Lektion 6: Einführung in die Grafikprogrammierung

Neue Sprachelemente:
fs, ts, ss , setsplit, fd, bk, rt, lt, home, window, fence, wrap, cs, clean, pu, pd, ht, st, watch, nowatch, trace, notrace

Programme:
QUADRAT, STERN, VERSCHIEBE

6.1 Das Grafikfenster

Das Monitorbild Ihres Computers entsteht aus einem Raster vieler einzelner Bildpunkte. Damit können auch Grafiken dargestellt werden, wie das auch in der üblichen Drucktechnik geschieht, wenn dort auch das Raster sehr viel enger ist.

Bisher wurde der Bildschirm im wesentlichen für die Darstellung von Rechenaufgaben benutzt, deren Ergebnissen sowie zum Schreiben von Programmen. Die für den jeweiligen Zweck genutzten Bildschirmbereiche werden Fenster genannt: das Textfenster, das Edit-Fenster und das Grafikfenster. Unter Fenstern versteht man abgegrenzte Bereiche des Bildschirms, in denen sich jeweils bestimmte Aktivitäten abspielen. Im LOGO-System auf den Schneiderrechnern können sich solche Fenster nicht überlappen, man kann also den Bildschirm lediglich aufteilen.

Das Edit-Fenster wurde in der letzten Lektion betrachtet; es umfaßt bis auf eine Fußzeile im wesentlichen den gesamten Bildschirm. Für Grafiken kann entweder ebenfalls der gesamte Bildschirm verwendet werden (Fullscreen) oder dieser wird in einen oberen Teil für die Grafik und einen unteren für die Textausgabe geteilt (Splitscreen).

Wie der Bildschirm genutzt werden soll, wird durch die folgenden Kommandos festgelegt:

- fs (für Fullscreen: Gesamter Bildschirm als Grafikfenster)
- ts (für Textscreen: Nur Textfenster)
- ss (für Splitscreen: Geteilter Bildschirm)

Für den Grafikbetrieb wird der geteilte Bildschirm benutzt, wenn Sie nicht ausdrücklich fs verlangen. Das ist auch vernünftig, weil man dann im unteren Teil des Monitorbildes die eigene Tastatureingabe verfolgen kann.

Die Größe des Textfenster im Splitscreen-Modus kann mit der Anweisung setsplit eingestellt werden.

```
setsplit 10
```

Damit werden zehn Textzeilen im unteren Bereich reserviert, wobei das Grafikfenster natürlich entsprechend eingeschränkt wird. Meist ist es sinnvoll, einfach mit der vom System vorgegebenen Größe zu arbeiten.

Das Kommanndo fs wirkt sich erst aus, wenn anschließend das Grafikfenster mit cs oder clean gelöscht wird.

```
fs cs oder fs clean
```

Die LOGO-Vokabel cs (für Clearscreen) bringt das Grafikfenster in die Ausgangsposition. Sie sollten deshalb den Grafikbetrieb grundsätzlich mit

```
cs
```

beginnen. Ohne cs zu Beginn kommt man zwar auch in die Ausgangsstellung, wegen eines kleinen Systemfehlers bleibt dann aber anschließend eine störende "Pfeilspitze" in der Bildschirmmitte stehen. Beachten Sie also:

Die Öffnung des Grafikfensters sollte in der Regel mit cs beginnen, um den beschriebenen Systemfehler zu korrigieren. Das ist beispielsweise auch nach einem Editiervorgang notwendig.

Das Programmieren einer Computergrafik bedeutet, einem imaginären Zeichenstift auf dem Grafikfenster Befehle zu erteilen. Da Sie diesen Zeichenstift nicht unmittelbar in der Hand haben, müssen Sie ihn durch Kommandos steuern. Zur Orientierung auf der Zeichenfläche gibt es zwei Methoden:

- die "körperzentrierte" Sprache der sogenannten Turtle-Grafik,
- die Orientierung im Koordinatensystem.

Die Turtle-Grafik (Turtle = Schildkröte) ist typisch für LOGO, weil sie ursprünglich hierfür entwickelt wurde. Heute sehen auch andere Programmiersprachenumgebungen diese Grafikmethode vor, denn der Vorzug der Turtle-Grafik hat sich herumgesprochen.

In gewissem Umfang besitzen die meisten Programmiersprachen für grafikfähige Mikrocomputer Anweisungen zum Zeichnen im Koordinatensystem. In LOGO ist nicht nur die typische Turtle-Grafik vorhanden, auch für die koordinatenorientierte Grafikprogrammierung hält DR LOGO einige Hilfsmittel bereit. Zunächst wird hier auf die Turtle-Grafik ausführlich eingegangen, die Kommandos, die Koordinaten benutzen, werden etwas später betrachtet.

6.2 Bewegungen der Schildkröte

Welche Bedeutung hat die Turtle?

In der Mitte des Grafikfensters erscheint in der Ausgangsstellung nach dem Start des Grafikbtriebs mit cs ein Pfeilsymbol, die Turtle, offensichtlich eine abstrakte Form einer Schildkröte.

Dieser kleine Pfeil ist das Symbol für einen Zeichenstift. Denn dort, wo die Turtle sitzt, kann unmittelbar gezeichnet werden. Der Zeichenstift wird ganz bewußt nicht durch einen Punkt oder einen kleinen Kreis dargestellt, weil die Spitze der Turtle angibt, in welche Richtung sich der Zeichenstift bewegt.

Die Kennzeichnung von Position (Ort) und Richtung des Zeichenstifts macht gerade das besondere der Turtle-Grafik aus. Der Schildkröte wird nicht ein Auftrag der Art 'gehe von da nach dort' erteilt.

Der Grundbefehl für die Turtle lautet vielmehr:

> Gehe eine bestimmte Strecke in einer bestimmten Richtung!

Die gerade gültige Richtung wird durch die Spitze der Turtle gekennzeichnet. Probieren Sie jetzt:

```
fd 60
```

Die Schildkröte hat sich über eine gewisse Strecke nach oben bewegt. Gleichzeitig hat sie dabei eine Spur in der Gestalt einer Geraden auf dem Bildschirm hinterlassen. Zeichnen mit der Turtle-Grafik heißt:

> Die Turtle so über den Grafikschirm zu bewegen, daß die von ihr hinterlassene Spur die gewünschte Zeichnung ergibt.

Es liegt auf der Hand, daß es möglich sein muß, die Richtung der Turtle zu verändern. Geben Sie ein:

```
rt 90
```

Die Schildkröte hat sich tatsächlich um 90 Grad nach rechts bewegt. Der Richtungssinn 'rechts' bezieht sich dabei auf die Turtle selbst. Bei einem Fahrzeug würde rt den Übergang zu einer Rechtskurve anzeigen.

Eigentlich würden fd und rt schon für alle Richtungen genügen, denn eine Rechtsdrehung um 270 Grad hat die gleiche Wirkung wie eine Linksdrehung um 90 Grad. LOGO kennt aber auch die Bewegung nach rückwärts und die Drehung nach links

```
bk 50
lt 90
```

Jetzt wird es Zeit, daß Sie diese Kommandos benutzen, um die Turtle über den Grafikschirm zu hetzen! Zögern Sie auch nicht, die Schildkröte einmal über die Grenzen des Bildschirms hinauszutreiben! Was beobachten Sie?

Wenn die Schildkröte über den oberen Bildschirmrand hinaus gerät, so ist das Pfeilsymbol auch nicht mehr sichtbar. Die Turtle kann aber wieder mit

home (ab nach Hause)

in den Ausgangszustand gebracht werden.

Das beschriebene Verhalten beim Versuch, die Turtle über die Grenzen des Grafikfensters hinauszutreiben ist nach dem Start von LOGO zu beobachten. DR LOGO kennt hierfür drei Betriebsarten:

- window (Fenster)
- fence (Einzäunung)
- wrap (Wickel-Zustand)

Der Window-Zustand bedeutet, daß die Schildkröte unbeschadet den Bereich des Grafikfenster verlassen kann. Sichtbar ist natürlich nur das, was sich innerhalb des Grafikfensters abspielt.

Im Fence-Zustand trifft die Schildkröte dagegen am Rand des Grafikfensters auf einen Zaun, der sie am Verlassen hindert. Es öffnet sich dann das Fehlerfenster mit der Meldung:

```
Turtle out of bounds
```

Beachten Sie, daß die Schildkröte in diesem Zustand dabei in ihrer Ausgangsposition verharrt. Ein Kommando wird also im Fence-Zustand nur dann befolgt, wenn die Turtle nach der Ausführung im Grafikfenster bleibt.

Bei welchen Eingaben der Rand des Grafikfensters erreicht wird, hängt neben der eingestellten Größe des Textfensters auch noch davon ab, ob Sie mit dem Schneider CPC oder dem Joyce arbeiten. In der Standardeinstellung werden die Ränder von der home-Position aus erreicht mit:

- CPC Senkrecht: 192 nach oben, 193 (112) nach unten
  Waagerecht: 320 nach links bzw. rechts
- Joyce: Senkrecht: 264 nach oben, 266(95) nach unten
  Waagerecht: 360 nach links bzw. rechts.

Beim Wrap-Modus (Wickelzustand) hängen oberer und unterer Rand des Fensters zusammen, ebenso rechter und linker Rand. Wenn die Turtle dann am oberen Rand aus dem Grafikfenster hinaus gedrängt wird, kommt sie am unteren Rand wieder ins Bild. Dementsprechend wandert sie z.B. über den linken Rand hinaus und taucht von rechts wieder im Bild auf.

Probieren Sie aus (die Eingaben müssen eventuell an die Größe des Grafikfenstes angepaßt werden):

```
home window fd 250
home fence fd 250
home wrap fd 250
home wrap rt 90 fd 400
```

Die Aussagen über die Ränder beziehen sich allerdigs nur auf das Grafikfenster, nicht auf den Bildschirm.

Beim Wickelmodus hängen oberer und unterer, rechter und linker Rand des Grafikschirms zusammen. Die Schildkröte benimmt sich so, als würde sie auf der Oberfläche eines Fahrradschlauches kriechen. Die Turtle-Bewegung wickelt sich gewissermaßen um den Grafikschirm herum, man spricht deshalb vom Wrap-(Wickel-)Zustand.

Das Wickeln gestattet lustige grafische Effekte, wogegen der window-Zustand mehr dem entspricht, was man von einer Grafikfläche erwartet.

6.3 Einfache Grafikprogramme

Mit einer Vorwärtsbewegung und anschließender Drehung um 90 Grad

```
fd 80 rt 90
```

entsteht bei vierfacher Wiederholung ein Quadrat. Die Eingabezeile kann mit Hilfe von CONTROL+Y wiederholt werden. LOGO führt diese Wiederholung mit Hilfe von repeat automatisch durch, was die Eingabezeile

```
repeat 4 [fd 80 rt 90]
```

zum Zeichnen eines Quadrates der Länge mit 80 Einheiten liefert.

Während man beim Rechnen auch ohne Programmieren einen Computer noch mit Gewinn einsetzen kann, wird man es beim Zeichnen sehr schnell als viel zu mühsam empfinden, die Kommandos für die Schildkröte immer wieder eintippen zu müssen. Für die Grafik wird der Computer erst richtig interessant, wenn aus den zwei Grundbewegungen der Turtle - der geradlinigen Verschiebung und der Drehung - komplexe Bewegungen aufgebaut werden.

Als erste Spracherweiterung soll LOGO die Tätigkeit 'Zeichne ein Quadrat mit gegebener Länge' lernen. Dazu könnte das Wort 'Quadratzeichnen' erklärt werden; Weil das ein sehr langes Wort ist, soll stattdessen stehen:

```
to QUADRAT
```

Hier muß man halt zwischen einer Vergewaltigung der deutschen Sprache und dem Risiko vieler Tippfehler bei langen Wörtern abwägen. Sie können auch direkt den LOGO-Editor aufrufen mit

```
ed "QUADRAT
```

Damit öffent sich das Editierfenster mit der Titelzeile der Prozedur QUADRAT.

Die neue LOGO-Tätigkeit soll die bereits benutzte Eingabezeile ausführen, das gesamte Programm erscheint dann im Editiermodus so:

```
to QUADRAT


>repeat 4 [fd 100 rt 90]
```

Das Programm wird mit der Zeile

```
>end
```

abgeschlossen, wenn Sie mit to QUADRAT begonnen haben. Wenn man sofort den Editor zum Schreiben des kleinen Programms benutzt hat, wird die Prozedur QUADRAT mit CONTROL+C (bzw. COPY beim CPC) in den LOGO-Wortschatz übernommen.

Der Vorgang wird natürlich schnell langweilig, schließlich entsteht ja immer das gleiche Quadrat. Es liegt nahe, eine variable Seitenlänge vorzusehen. Dazu muß die Prozedur QUADRAT die Seitenlänge als Eingabe vorsehen.

```
to QUADRAT :seite
repeat 4 [fd :seite rt 90]
```

Machen Sie sich bei dieser Gelegenheit noch einmal die Bedeutung des einleitenden Doppelpunktes vor dem Namen seite klar: Die Schreibweise :seite bedeutet 'Wert der Seite'.

Wenn das Zeichnen von Quadraten verschiedener Seitenlänge, etwa mit

```
QUADRAT 10 QUADRAT 20 QUADRAT 30
```

monoton wird, versuchen Sie einmal, das Würfeln mit in die Grafik einzubeziehen. Mittlerweile ist der Grafikschirm schon ziemlich vollgemalt, und es entsteht das Bedürfnis, den Ausgangszustand im Grafikfenster wieder herzustellen. Neben dem dazu geigneten Kommando cs, das bereits zum Eintritt in den Grafikbetrieb verwendet wurde, gibt es auch die Möglichkeit, auf dem Grafikschirm nur die Turtle-Spuren selbst wegzuwischen.

```
clean
```

Damit wird der Inhalt des Grafikschirms ohne weitere Nebenwirkungen gelöscht, während cs einen vorgegebenen Anfangszustand der Grafik einstellt, was auch die Wirkung von home beinhaltet.

Wie gefällt Ihnen die folgende Grafik?

```
repeat 100 [QUADRAT random 150]
```

Diese kurze LOGO-Zeile erzeugt die räumlich wirkende Grafik von Abb. 6.1, nämlich ein einfaches Kippbild, das je nach Sichtweise des Betrachters aus dem Bildschirm nach vorne heraus oder nach hinten in den Monitor hinein zu ragen scheint.

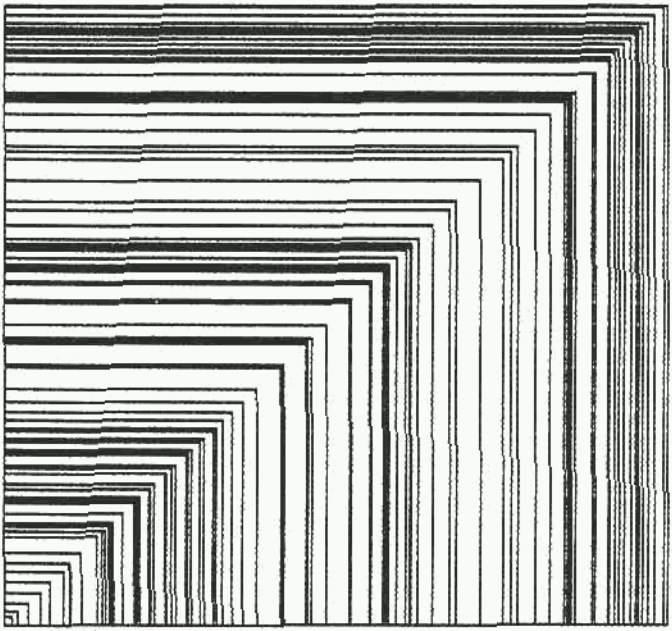


Abb. 6.1: Kippbild durch Wiederholung von QUADRAT

Vergegenwärtigen Sie sich hier noch einmal den Aufruf der Tätigkeit QUADRAT in dieser Zeile. Die Eingabe der Seitenlänge geschieht nicht als Zahl, sondern durch eine andere LOGO-Tätigkeit random. Die Zahl 150 ist Eingabe der davor stehenden Prozedur, also random. random 150 liefert als Ergebnis eine einzige Zahl, die dann von QUADRAT als Seitenlänge verwandt wird.

6.4 Drehen von Figuren

Nach der Ausführung der Prozedur QUADRAT hat die Turtle wieder ihre Ausgangsstellung in der Mitte des Bildschirms erreicht. Eine Wiederholung von QUADRAT mit gleicher Eingabe erzeugt natürlich die gleiche Figur. Wenn also überhaupt etwas geschieht, ist dies nur an der Bewegung der Turtle zu erkennen. Wird die Schildkröte aber vorher noch bewegt, so ändert sich beim Aufruf von QUADRAT zwar nicht die Figur, wohl aber ihre Position auf dem Bildschirm:

```
QUADRAT 60 rt 45 QUADRAT 60
```

Das zweite Quadrat ist um 45 Grad gegenüber dem ersten gedreht. Die Figuren sind aber deckungsgleich (kongruent).

So einfache Figuren wie Quadrate werden erst durch vielfache Wiederholung interessant; probieren Sie daher einmal die folgende Zeile aus:

```
cs repeat 36 [QUADRAT 60 rt 10]
```

Diese einfache Art, ganze Figuren drehen und verschieben zu können, macht einen wesentlichen Reiz der Turtle-Grafik aus. Wenn das mit der Zeile

```
repeat 36 [QUADRAT 60 rt 10]
```

entstandene Muster gefällt, dann könnten Sie es auch als Spracherweiterung für die gegenwärtige LOGO-Sitzung aufnehmen.

```
to STERN :seite
repeat 36 [QUADRAT :seite rt 10]
end
```

Mit einer Zeile der Art

```
cs STERN 40 STERN 60 STERN 90
```

können die ineinandergeschachtelten Muster schon recht beeindruckende Bilder erzeugen.

Bei der Prozedur STERN ist die Seitenlänge variabel, der Drehwinkel aber noch fest mit 10 Grad gewählt. Bei 36 Drehungen ergibt sich so eine volle Umdrehung. Mit dem erklärten Wort STERN kann man mehr anfangen, wenn die Zahl der Drehungen veränderbar ist.

Um die Prozedur STERN abzuändern, muß sie editiert werden. Mit

```
ed "STERN
```

wird das Edit-Fenster geöffnet, und der Text von STERN erscheint. Beachten Sie, daß der Prozedurname STERN mit Anführungszeichen eingeleitet werden muß. Ohne Anführungszeichen erscheint die Fehlermeldung:

```
Not enough inputs to STERN
```

DR LOGO wendet die Regeln hier streng an:

- STERN ohne Anführungszeichen löst in jedem Fall die Ausführung der Tätigkeit STERN aus. Das scheitert dann an der fehlenden Eingabe für die Seitenlänge.

- Mit Anführungszeichen wird der LOGO-Vokabel ed der Name der Tätigkeit STERN als Eingabe mitgeteilt. Wenn die Prozedur STERN existiert, gibt ed den Text im Edit-Fenster aus, andernfalls werden dort nur die Titel- und Schlußzeile erzeugt.

```
to STERN :seite :n
repeat :n [QUADRAT :seite rt 360/:n]
end
```

Die Übergabe des veränderten Textes von STERN an das LOGO-System geschieht wie bereits gewohnt mit CONTROL+C oder mit COPY beim CPC.

Der Drehwinkel 360/:n sorgt dafür, daß insgesamt eine volle Umdrehung zustande kommt. Probieren Sie einmal

```
STERN 80 2 STERN 40 4 STERN 20 8.
```

Oder lassen Sie sich überraschen mit

```
STERN random 100 random 72.
```

Bei diesem Aufruf werden beide Eingaben von STERN durch Zufallszahlen bestimmt.

Für den LOGO-Ungeübten ist die Schreibweise 360/:n etwas seltsam. Die Division wird nicht durch den Doppelpunkt, sondern mit dem schrägen Bruchstrich angezeigt. Der Doppelpunkt gehört zum Namen n.

6.5 Verschieben von Figuren

Wird nach der Ausführung von QUADRAT die Turtle nicht um einen bestimmten Winkel gedreht, sondern um eine bestimmte Strecke vorwärts oder rückwärts bewegt, so erscheint das folgende Quadrat jeweils verschoben.

```
repeat 5 [QUADRAT 20 fd 20]
```

Hier werden fünf Quadrate aneinandergehängt. Um bei den folgenden Beispielen den Bildschirm besser ausnutzen zu können, wird die Schildkröte erst einmal nach unten bewegt, denn dann steht oben mehr Platz zur Verfügung.

```
cs bk 50 repeat 5 [QUADRAT 20 fd 30]
```

Die Quadrate hängen jetzt nicht mehr aneinander. Es hat sich aber etwas Unbeabsichtiges bemerkbar gemacht: Die verschobenen Quadrate sind durch eine gemeinsame Senkrechte miteinander verbunden. Bei der Verschiebung der Schildkröte mit fd 30 hinterläßt diese eben auch eine Spur. Das sollte man natürlich auch verhindern können, sonst würden alle Figuren auf dem Bildschirm zusammenhängen.

Eine stubenreine Schildkröte erhält man durch

pu (Abkürzung für Penup)

pu heißt: Hebe den Zeichenstift an. Nach diesem Befehl kann der Zeichenstift wie vorher bewegt werden, zur Fortsetzung der Zeichnung läßt er sich wieder absenken mit

pd (Abkürzung für Pendown)

Um eine Verschiebung zu bewerkstelligen, muß vor der Verschiebung der Zeichenstift angehoben, danach wieder abgesenkt werden. Ebenso können natürlich die Bewegungen des Zeichenstifts mit dem Zeichnen der Figur verbunden werden.

Es sind aber nicht nur Verschiebungen nach oben, sondern in beliebige Richtungen möglich. Dazu kann ein eigenes LOGO-Wort nützlich sein.

```
to VERSCHIEBE :winkel :strecke
ht pu rt :winkel
fd :strecke
lt :winkel pd st
end
```

Die Richtung wird dabei durch den Winkel zur Senkrechten nach oben gekennzeichnet. Auf der Landkarte würde z.B. Osten dem Winkel +90 Grad, Westen -90 Grad, Südwesten -135 Grad usw. entsprechen. In der Prozedur VERSCHIEBE muß die Rechtsdrehung nach der Vorwärtsbewegung wieder durch eine entsprechende Linksdrehung rückgängig gemacht werden, da sonst neben der beabsichtigten Verschiebung auch eine zusätzliche Drehung der Figur entstünde.

Versuchen Sie z.B. folgende Zeile:

```
cs repeat 5 [QUADRAT 20 VERSCHIEBE -45 35]
```

Sicher haben Sie im Programm VERSCHIEBE die beiden nicht erklärten Vokabeln ht und st entdeckt. Deren Auswirkung war bei der Zeichnung der fünf Quadrate allerdings kaum zu bemerken.

ht (Verstecke die Schildkröte - Hide Turtle)
st (Zeige die Schildkröte - Show Turtle)

Damit kann die Turtle unsichtbar, bzw. sichtbar gemacht werden. Das Verstecken der Schildkröte ist während des eigentlichen Zeichenvorgangs meist nicht ratsam, da die Turtle-Bewegungen schließlich Veränderungen auf dem Grafikschirm deutlich machen. Wenn die Schildkröte bei der dann erzeugten Grafik störend wirkt, kann sie am Ende immer noch mit ht versteckt werden.

Bei längeren Zeichenprogrammen führt eine versteckte Turtle zur Beschleunigung des Zeichenvorgangs. Es ist leicht zu verstehen, daß eine Darstellung der bewegten Schildkröte den Computer auch Zeit kostet.

6.6 Prozeduren rufen Prozeduren

Bei den betrachteten, einfachen Grafikprogrammen stand das Zeichnen auf dem Bildschirm im Vordergrund. Die dabei entstandenen Prozeduren sind teilweise komplizierter als die einfachen Programme zur Mehrwertsteuerberechnung. Um die Programmstruktur zu verdeutlichen, soll die letzte Version des Programms STERN nochmals unter die Lupe genommen werden. Benötigt wird dabei auch die Prozedur QUADRAT.

```
to STERN :seite :n
repeat :n [QUADRAT :seite rt 360/:n]
end
to QUADRAT :seite
repeat 4 [fd :seite rt 90]
end
```

In der Prozedur STERN wird das selbstdefinierte LOGO-Wort QUADRAT verwendet, während im Programm QUADRAT im Gegensatz dazu nur Tätigkeiten aus dem Grundwortschatz von LOGO benutzt werden.

Wenn beim Programmieren neue LOGO-Vokabeln entstanden sind, werden sie in gleicher Weise wie die Grundwörter ('LOGO Primitive') behandelt. Nach Aufruf der Tätigkeit QUADRAT wird von dieser Prozedur die LOGO-Tätigkeit fd in Gang gesetzt. Wichtig ist dabei vor allem, daß die Eingabe für die Prozedur fd aus der des rufenden Programms QUADRAT stammt.

Wird das Programm STERN mit

```
STERN 50 4
```

aufgerufen, wird die Zahl 50 zunächst einmal unter dem Namen seite an die Prozedur STERN übergeben. Innerhalb von STERN wird der Name seite zuerst an die Prozedur QUADRAT, im Programm QUADRAT dann wiederum an die LOGO-Grundtätigkeit fd weitergereicht. Erst an dieser Stelle wird die Zahl 50 in einer Aktion auf dem Bildschirm umgesetzt.

- Prozeduren können andere Prozeduren aufrufen.
- Die Eingabewerte können mit ihrem Namen an aufgerufene Prozeduren weitergereicht werden.

Die Möglichkeit, Eingabewerte von Prozeduren an andere aufgerufene Prozeduren weiterreichen zu können, fördert einen modularen, baukastenähnlichen Aufbau von Programmen. Für die richtige Anwendung einer Prozedur muß man nur wissen, welche Eingabewerte in welcher Reihenfolge benötigt werden. Wie diese Eingabewerte im Programm genannt werden, und was damit im einzelnen geschieht, ist unerheblich. Sofern das Programm auch Ergebnisse hat, die weiter verarbeitet werden sollen, gilt entsprechendes auch für die Ausgaben.

Wenn die Tätigkeit STERN erst einmal programmiert und erfolgreich getestet wurde, kann sie in komplexere Grafikprogramme eingebaut werden, ohne daß man sich um das Programm STERN selbst noch kümmern muß. Diese Art der Übergabe von Eingabewerten funktioniert allerdings nur deswegen problemlos, weil die Eingaben von Prozeduren lokale Namen sind. In der folgenden Version wird das noch deutlicher:

```
to STERN :laenge :n
repeat :n [QUADRAT :laenge rt 360/:n]
end
to QUADRAT :seite
repeat 4 [fd :seite rt 90]
end
```

Innerhalb der Prozedur STERN hat die Eingabe den Wert laenge, in der Prozedur QUADRAT, der Stelle der eigentlichen Auswertung, heißt sie dagegen seite. Man kann das folgendermaßen beschreiben:

Benötigt wird beim Bearbeiten von fd in der Prozedur QUADRAT ein Eingabewert; DR LOGO findet dort die Aufforderung, beim 'Wert von seite' nachzuschauen. seite ist aber Eingabe von QUADRAT, also muß LOGO nachsehen, was innerhalb der rufenden Prozedur STERN als Eingabewert für QUADRAT übergeben wurde. Die Eingabe heißt dort aber nun 'Wert von laenge; laenge ist wiederum Eingabewert von STERN, also muß LOGO den tatsächlichen Eingabewert der Prozedur STERN ermitteln.

Die Tätigkeit fd erhält so ihren Eingabewert erst um mehrere Ecken!

Bei der hier dargestellten Übergabe von Eingaben an eine Prozedur wird jeweils der Wert, nicht der zugehörige Name weitergegeben. Man spricht deswegen von einem Wertaufruf (englisch: call by value). Wenn Sie Programmiersprachen wie PASCAL, ALGOL oder PLI kennen, so werden Sie vielleicht nach dem sogenannten Referenzaufruf (englisch: call by name)

fragen. Keine Sorge, LOGO kennt auch dieses; es wird uns im Zusammenhang mit der LOGO-Vokabel thing noch beschäftigen.

6.7 Watch und Trace - Die Kontrolleure von DR LOGO

Das Verschachteln von Prozeduraufrufen kann zu mächtigen Tätigkeiten führen, wobei man allerdings wachsam sein muß, um bei wachsender Komplexität nicht den Überblick zu verlieren. DR LOGO stellt Ihnen nun zwei Hilfsmittel zur Verfügung, um den Ablauf eines Programms Schritt für Schritt verfolgen zu können. Geben Sie zu diesem Zweck ein:

```
trace
```
(etwa: Protokolliere den Ablauf von Prozeduren)

Dieser Betriebszustand bleibt solange erhalten, bis Sie wieder eingeben:

```
notrace
```
(Protokollieren benden)

Für eine Fehlersuche kann dann die Kombination mit der Ausgabe auf dem Drucker, soweit vorhanden, besonders nützlich sein, also die Kombination

```
copyon trace.
```

Machen Sie nun einen Versuch mit

```
STERN 50 2.
```

Es erscheint eine Folge von Meldungen.

```
[1] Evaluating STERN
[1] n is 2
[1] laenge is 50
[2] Evaluating QUADRAT
[2] seite is 50
[2] Evaluating QUADRAT
[2] seite is 50
```

Im Trace-Modus wird die Ausführung von Prozeduren protokolliert. Evaluating STERN bedeutet, daß jetzt mit der Prozedur STERN begonnen wurde. Die Eingabewerte werden anschließend mit Namen und dem übergebenen Wert offengelegt.

Beobachten Sie die Wirkung von trace bei der Prozedur

```
to MEHRWERTSTEUER :preis
make "mwstsatz .14
op :mwstsatz* :preis
end
```

Hier wird einem Namen mit Hilfe von make mit ein Wert zugeordnet, was zur Protokollzeile

```
[1] making "mwstsatz Ø.14
```

führt. MEHRWERTSTEUER hat auch eine Ausgabe, die im Trace-Modus mit der Bemerkung

```
[1] MEHRWERTSTEUER returns ...
```

wiedergegeben wird.

Welche Bedeutung haben die am Anfang der Protokollzeilen in eckigen Klammern ausgegebenen Zahlen?

Im Beispiel STERN wird beim Aufruf STERN 50 2 die Prozedur STERN direkt in Gang gesetzt, QUADRAT dagegen erst von STERN gerufen. QUADRAT wird also auf einer anderen Ebene benutzt als STERN, was durch in eckige Klammern stehende Ziffern verdeutlicht wird.

Bei stärkerer Verschachtelung von Prozeduraufrufen wird das dann noch deutlicher. Beginnt die Protokollzeile mit [3], so wird die angezeigte Prozedur von einer anderen Tätigkeit gerufen, die selbst wiederum von einer dritten in Gang gesetzt wurde usw.

Der Trace-Zustand ist nicht nur zur Fehlersuche nützlich, er hilft auch, einen möglicherweise komplizierteren, verschachtelten Ablauf von LOGO-Programmen verstehen zu lernen. Weil LOGO eine mächtige Programmiersprache ist, kann die innere Logik eines LOGO-Programms auch durchaus komplizierter werden als das bei primitiven Programmiersprachen überhaupt möglich ist.

Neben dem Trace-Modus zur Protokollierueng des Programmablaufs gibt es noch einen weitere Möglichkeit zur Kontrolle, den Watch-Modus.

Probieren Sie

```
watch
STERN 50 2

[1] In STERN, repeat :n [QUADRAT :laenge rt 360 / :n]
```

Nach Drücken der RETURN-Taste geht es jeweils weiter.

```
[2] In QUADRAT, repeat 4 [fd :seite rt 90]
[2] In QUADRAT, repeat 4 [fd :seite rt 90]
```

Auch hier wird ein Protokoll angefertigt! Die Ziffern in eckigen Klammern am Anfang der Protokollzeile haben dieselbe Bedeutung wie bei trace. Allerdings wird die im Trace-Zustand wiedergegebene Abfolge von Prozeduraufrufen und die eventuell erfolgenden Änderungen von Werten nicht gezeigt. Protokolliert wird vielmehr die Folge der ausgeführten Programmzeilen. Die von LOGO gerade durchgeführte Prozedur sowie die Verschachtelungstiefe werden am Anfang der Protokollzeile sichtbar gemacht.

Der Watch-Zustand wird mit:

```
nowatch
```

abgeschaltet. Die ausführlichste Dokumentation bekommen Sie natürlich, wenn Sie sowohl die Watch- als auch die Trace-Funktion benutzen.

Wenn Sie watch und den Aufruf STERN 50 2 in einer Eingabezeile aufgerufen haben, ergab sich eine Fehlermeldung, da man nämlich watch auch mit Eingaben versehen kann.

```
watch "STERN
```

Hiermit wird eine Protokollierung der Kommandoabarbeitung auf die Prozedur STERN eingeschränkt. Man kann auch mehrere Prozeduren gleichzeitig zur Überwachung mit watch anmelden, muß daraus aber erst eine Liste bilden, was noch später erklärt wird.

Die Einschränkung des Watch-Zustands auf einige Prozeduren ist für die Fehlersuche sehr nützlich, weil die dann ausgegebene Information auf die Umgebung eines aufgetretenen Fehlers beschränkt werden kann.

Übungen

1) Die Turtle soll zufällig verteilt über den Bildschirm jagen. Richtung und Schrittweite sollen mit Hilfe der Funktion random erzeugt werden. Eingabewert für das Programm soll die Zahl der Wiederholungen sein.

2) Schreiben Sie Prozeduren, die Rechtecke und Dreiecke variabler Größe zeichnen.

3) Der Bildschirm soll mit einem Muster aus gleichgroßen, lückenlos aneinander hängenden Quadraten bedeckt werden.

4) Formulieren Sie eine LOGO-Tätigkeit HAUS, die ein einfach gebautes Haus zeichnet.

5) Erzeugen Sie durch Verschiebung eines Hauses mehrere Häuser.

6) Zaubern Sie viele Sterne zufällig verteilt auf den Bildschirm.

Lektion 7: Grafikprogramme mit Wiederholung

Neue Sprachelemente:
local, if, stop, Rekursion, Vergleiche

Programme:
VIELECK, Programme für Kreise, Kreisbogen, ROSETTE, POLYGON, STICKEN (Chaotische Kurven), ECKEN, NECKEN, DOPPELECKEN

7.1 Vom Quadrat zum Kreis

Nach Ausführung der Prozedur QUADRAT

```
to QUADRAT :seite
repeat 4 [fd :seite rt 90]
end
```

steht die Turtle wieder in ihrer Ausgangsposition. Das ist zwar Absicht des Programms, aber an sich ist es ja nicht selbstverständlich, daß die vierfache Wiederholung einer Zeichenanweisung zu einer geschlossenen Figur führt. Wie kann man das dem Programm ansehen?

Einmal liegt das an den Drehbewegungen. Eine viermalige Drehung um 90 Grad bedeutet eine Drehung der Schildkröte um 360 Grad, also eine volle Umdrehung. Die Turtle hat deswegen am Ende die gleiche vertikale Ausrichtung wie zu Beginn. Es könnte aber immer noch passieren, daß sich keine geschlossene Figur ergibt, da die Turtle am Ende einen anderen Punkt als den Ausgangspunkt erreicht haben könnte.

Sie muß sich also um die gleiche Strecke nach unten bewegen, wie sie nach oben geschoben worden ist; gleiches gilt für die Bewegungen nach rechts bzw. links. Alle Strecken, die die Schildkröte zurücklegt, haben die gleiche Länge, so daß

zusammen mit dem gesamten Drehwinkel von 360 Grad ein Quadrat entsteht.

Eine volle Umdrehung der Turtle läßt sich natürlich auch mit einer anderen Zahl von Drehungen erreichen. Als Verallgemeinerung von QUADRAT versuchen wir deshalb ein Programm VIELECK:

```
to VIELECK :seite :n
repeat :n [fd :seite rt 360/:n]
end
```

Spielen Sie selbst mit diesem neuen LOGO-Wort Vieleck!

```
VIELECK 60 3
VIELECK 60 5
VIELECK 60 6
...
```

Es ergeben sich tatsächlich regelmäßige Dreiecke, Vierecke, Fünfecke, Sechsecke usw.!

An dieser Stelle ließen sich interessante Überlegungen zu Vielecken anschließen. Betrachten wir hier nur noch einmal das regelmäßige, d.h. gleichseitige Dreieck.

```
VIELECK 80 3
```

In der Prozedur VIELECK erfolgt eine Drehung um den Winkel 360/3=120 Grad. In der Dreiecksgeometrie bezeichnet man den Drehwinkel der Turtle als den Außenwinkel. Der Innenwinkel ergibt sich aus dem Außenwinkel jeweils durch 180 Grad minus Innenwinkel, also 60 Grad. Daraus folgt die bekannte Aussage, daß beim Dreieck die Summe der Innenwinkel 180 Grad beträgt.

Um zu sehen, was die Prozedur VIELECK bei wachsender Zahl der Ecken produziert, kann folgendes kleine Programm dienen:

```
to VIELE.VIELECKE :seite :nmax
cs pu lt 90 fd 100 rt 90 pd
local "k make "k 2
repeat :nmax -1 [VIELECK :seite :k make "k :k+1]
```

Probieren Sie z.B. aus

```
VIELE.VIELECKE 18 40
```

Es sollte jetzt auf dem Bildschirm ein kegelartiges Bild entstehen. In diesem Programm sind einige bisher noch nicht verwendete Sprachelemente enthalten.

Die Prozedur hat den Namen VIELE.VIELECKE bekommen. Die Länge des Namens ist zwar beim Aufruf etwas unhandlich, lange Namen haben aber den Vorteil, daß sie sich meist selbst erklären. 'Viele Vielecke' sind zwei Worte, die eigentlich auseinander geschrieben werden sollten. Das Leerzeichen ist aber in LOGO grundsätzlich ein Trennzeichen. In der Zeile

```
VIELE VIELECKE :seite :nmax
```

ist der Prozedurname VIELE. Dagegen steht VIELECKE für eine Eingabe.

Der Punkt bei VIELE.VIELECKE dient als optisches Trennsymbol, für LOGO entsteht aber ein einziges Wort. Die Verwendung des Punktes als nichttrennendes Trennsymbol hat sich nicht nur in LOGO eingebürgert. Mit der gleichen Funktion wird häufig auch das Unterstreichungszeichen benutzt, das Sie beim CPC über der Null auf dee obersten, beim Joyce über dem Gedankenstrich in der untersten Reeihe der Tastatur finden.

Die zweite Zeile von VIELE.VIELECKE bewirkt die Löschung des Grafikschirms, wobei der Zeichenstift nach links verschoben wird, damit genügend Platz für die Vielecke vorhanden ist. Die

dritte Zeile enthält ein neues LOGO-Wort, local, auf das im nächsten Abschnitt noch eigens eingegangen wird.

In der vierten Zeile wird das Zeichnen von Vielecken mit repeat (nmax-1)mal wiederholt. nmax stellt die maximale Eckenzahl dar, mit der VIELECK aufgerufen werden soll. Da mit einem Zweieck begonnen wid, muß die Wiederholung nur einmal weniger ausgeführt werden.

Damit die Eckenzahl beim nächsten Aufruf von VIELECK größer wird, erhöht sich der Wert der aktuellen Zahl der Ecken k mit make "k :k+1 um eins.

Somit werden nacheinander Vielecke mit 2, 3, 4, ... bis nmax Ecken gezeichnet. Die Seiten haben immer die gleiche Länge. Der Umfang der Vielecke wird dann mit wachsender Eckenzahl immer größer.

Natürlich ist Ihnen schon längst aufgefallen, daß die begrenzte Auflösung des Grafikschirms dafür sorgt, daß manche Strecken gar nicht vollständig gerade sind, das Lineal des Computers scheint manchmal Bruchstellen aufzuweisen. Bereits beim 20-Eck ist es kaum noch möglich, die zwanzig Ecken richtig zu identifizieren. Wenn das Computerlineal Bruchstellen hat, so tritt entsprechendes natürlich auch beim Zirkel des Computers auf. Wegen der begrenzten Auflösung des Grafikschirms sind regelmäßige Vielecke mit großer Eckenzahl nicht mehr von Kreisen zu unterscheiden: Kreise werden praktisch durch regelmäßige Vielecke erzeugt!

7.2 Strukturiertes Programmieren mit lokalen Namen

Untersuchen wir nun die Bedeutung der dritten Zeile im Programm VIELE.VIELECKE:

```
local "k make "k 2
```

Die Bedeutung von make "k 2 ist klar. Der Name k wird mit einem Anfangswert 2 versehen. Die Prozedur kann auch ohne local "k geschrieben werden und erzeugt dann auch dieselbe Zeichnung. Die Wirkung von local "K können Sie feststellen, wenn Sie pons nach einem Aufruf von VIELE.VIELECKE eingeben. Sie werden vergeblich nach dem Namen k suchen. Wird local "k dagegen weggelassen, so finden Sie bei pons den Namen k mit seinem gültigen Wert.

> local macht aus dem nachfolgenden Namen einen lokalen Namen.

Der Name K ist nur innerhalb der Prozedur bekannt, in der er mit local erklärt wurde. Hat LOGO diese Prozedur abgeschlossen, so ist der Name nicht mehr verfügbar und wird genauso behandelt wie eine Prozedureingabe. Der Unterschied besteht nur darin, daß diese beim Aufruf einen Wert bekommt, der mit local eingeführte Name dagegen erst durch make mit einem Wert versehen wird.

Welchen Sinn hat nun eigentlich die Einführung lokaler Namen mit local?

Im Beispiel der Prozedur VIELE.VIELECKE ist es relativ unerheblich, ob k als lokaler oder globaler Name behandelt wird. Die Verwendung lokaler Namen ist dennoch überall dort empfehlenswert, wo globale Namen nicht notwendig sind. Bei lokalen Namen ist ihre Geltung von vornherein auf den Bereich eingeschränkt, wo die Namen benötigt werden. Es kann deswegen keinen Konflikt und keine unbeabsichtigte Verwechslung mit Namen außerhalb dieses Bereichs geben. Der Name k kann außerhalb der Prozedur VIELE.VIELECKE benutzt werden, die

Verwendung des gleichen Namens für einen lokalen Namen stört überhaupt nicht.

Verwendet man dagegen globale Namen, so muß sorgfältig darauf geachtet werden, daß keine Konflikte in der Bedeutung von Namen auftreten.

> Die Verwendung lokaler Namen fördert das modulare, strukturierte Programmieren.

Für die Benutzung einer fertigen Prozedur müssen dann nur die Eingaben und gegebenenfalls die Ausgabe bekannt sein.

Globale Namen, die nicht mehr benötigt werden, stellen auch eine Art Programmiermüll dar, der u. a. Platz im System beansprucht. Bei den bisher betrachteten kleinen Programmbeispielen gibt es keine Platzprobleme im Rechner. Die Verwendung lokaler Namen ist aber auch in dieser Hinsicht sinnvoll.

7.3 Die Schildkröte auf krummen Wegen

Kreise

Wird ein Kreis mit Hilfe von Vielecken näherungsweise dargestellt, sind die Seitenlänge des Vielecks und die Seiten- bzw. Eckenzahl N vorgegeben. Das N-fache der Seitenlänge ist der Umfang des Vielecks. Ein Kreis wird aber üblicherweise nicht durch den Umfang, sondern durch den Kreisradius festgelegt. Um also einen Kreis mit vorgegebenem Radius als Vieleck zu erzeugen, muß deswegen die zugehörige Seitenlänge für das als Näherung benutzte Vieleck aus dem Radius berechnet werden. Weil der Kreisumfang aus dem Radius r durch 2*:pi*r resultiert, ergibt sich die benötigte Seitenlänge nach

SEITENLÄNGE = 2*:pi*r/n

Damit kann ein Programm für Kreise formuliert werden, wobei die Kreiszahl pi vor dem Aufruf bereitgestellt werden muß:

```
to KREIS.1 :r :n
VIELECK 2*:pi*:r/:n :n
end
to VIELECK :seite :n
repeat :n [fd :seite rt 360/:n]
end
```

KREIS.1 soll die erste Version eines Kreisprogramms andeuten. Da die Zahl der Ecken als Eingabewert variabel ist, kann man mit dem Programm noch experimentieren:

```
make "pi 3.141593
KREIS.1 70 360
KREIS.1 70 72
KREIS.1 70 36
```

Mit 360 Ecken läßt sich die Schildkröte natürlich viel Zeit, eigentlich reichen 36 Ecken schon aus, um einen hinreichend runden Eindruck der Kreislinie zu erzeugen. Ein zweiter Vorschlag benutzt 36 Ecken und unterscheidet sich noch zusätzlich dadurch vom ersten, daß die Position des Zeichenstifts beim Aufruf als Kreismittelpunkt genommen wird.

```
to KREIS.2 :r
pu fd :r rt 95 pd
VIELECK :pi*:r/18 36
pu lt 95 bk :r pd
end
```

In der zweiten Zeile wird die Turtle zunächst vom Mittelpunkt aus nach oben auf einen Punkt der Kreislinie befördert und dann in Richtung der Kreislinie um 95 Grad gedreht. Im Vergleich zur ersten Version wird hier eine zusätzliche Anfangsdrehung um 5 Grad ausgeführt. (Der Zusatzwinkel ist gerade die Hälfte des dann bei VIELECK verwendeten Winkels von 10 Grad.) Ohne diese Anfangsdrehung würde der Kreis etwas nach links verschoben.

Sie können sich durch Ausprobieren davon überzeugen, daß ohne diesen Zusatzwinkel die Turtle am Ende nicht wie beabsichtigt im Mittelpunkt des Kreises stünde. Die Vorwärtsbewegung vom Ausgangspunkt zur Kreislinie muß auf einer Symmetrieachse des Vielecks erfolgen. Weil im Programm VIELECK mit einer Vorwärtsbewegung begonnen wird, ist die Zusatzdrehung um 5 Grad notwendig, und zum Schluß wird mit lt 95 der Gesamtdrehwinkel 360 erreicht.

Da die Prozedur VIELECK nur eine einzige repeat-Zeile enthält, kann man diese auch in die KREISPROZEDUR aufnehmen; die folgende Version KREIS.3 ist dann in sich abgeschlossen.

```
to KREIS.3 :r
pu fd :r rt 95 pd
local "seite make "seite :pi*:r/18
repeat 36 [fd :seite rt 10]
pu lt 95 bk :r pd
end
```

Der Name seite wurde hier eingeführt, damit nicht bei jedem Schritt die Zahl pi*R/18 ausgerechnet werden muß.

Kreisbogen

Für Zeichenprogramme sind Kreisbogen, also Teile der Kreislinie, eigentlich noch interessanter als ganze Kreise, weil sie als Standard für gekrümmte Kurventeile verwendet werden können. Aus einem Kreisprogramm kann natürlich relativ leicht ein Programm für Kreisbogen gemacht werden. Damit können dann Rechts- oder Linkskurven gezeichnet werden.

Die aktuelle Position der Turtle gibt den Anfang der Kurve vor, weswegen man wirklich zwei verschiedene Tätigkeiten für Rechts- bzw. Linkskurven braucht. Hier soll nur die Rechtskurve beschrieben werden, da Sie die Übersetzung in Links-

kurven mühelos durch Übertragung aller Rechts- in Linksdrehungen erzeugen können.

Wird zur Beschleunigung wie beim Kreisprogramm bei jedem Schritt ein größerer Drehwinkel von 10 oder etwa auch 5 Grad genommen, so ergibt sich ein kleines Problem, weil der für den Kreisbogen gewünschte Gesamtdrehwinkel nicht immer ein ganzzahliges Vielfaches des Teilwinkels ist. Bei einem Gesamtwinkel von beispielsweise 86 Grad kann ein Teilbogen mit 80 Grad wie im Kreisprogramm gezeichnet werden.

Für die restlichen 6 Grad muß ein eigener Programmteil vorgesehen werden. Die Winkelaufteilung in 86=80+6 Grad geschieht mit den Ganzzahlfunktionen quotient und remainder.

```
(quotient :winkel 10)
(remainder :winkel 10)
```

Die erste Zeile liefert die Anzahl der 10-Grad-Schritte, die zweite Zeile den Restwinkel. Die runden Klammern sind hier nicht notwendig, weil sie die drei Objekte nur optisch zusammenfassen. Beachten Sie, daß sich jeweils eine einzige Zahl daraus ergibt. Alles, was in runde Klammern eingeschlossen ist, steht an Stelle einer Zahl, hier zum Beispiel der Anzahl von Wiederholungen.

Um die Krümmung der Kurve festzulegen, können entweder der Radius des zugehörigen Kreises oder die Seitenlänge des zugehörigen Vielecks verwendet werden. In der folgenden Version wird direkt die Seitenlänge benutzt.

```
to RECHTSKURVE :seite :winkel
rt 5
repeat (quotient :winkel 10) [fd :seite rt 10]
RESTRKURVE :seite/10 (remainder :winkel 10)
end
to RESTRKURVE :seite :winkel
fd :seite*:winkel
rt :winkel-5
end
```

Wie oben beim Programm KREIS.3 kann auf die zweite Prozedur verzichtet werden, wenn die restliche Rechtskurve in der Prozedur RECHTSKURVE erzeugt wird. Der Aufruf einer eigenen Prozedur RESTRKURVE ist die elegantere Lösung, weil die Übergabe von remainder :winkel 10 als Eingabe einer Prozedur dafür sorgt, daß diese Funktion nur einmal ausgewertet werden muß. Beim Einbau der Prozedur VIELECK im Beispiel KREIS.3 wurde stattdessen ein eigener lokaler Name eingeführt, dem mit make ein Wert zugewiesen wurde. Solche Lösungen sind aber weniger elegant und nicht typisch für LOGO-Programme.

Mit RECHTSKURVE können auch Kreise gezeichnet werden, wobei der Winkel dann eben 360 Grad betragen muß.

```
RECHTSKURVE :pi*70/18 360
```

Diese Zeile erzeugt einen Kreis mit einem Radius von 70 Einheiten.

Figuren mit Kreisbogen

Mit Hilfe von Rechts- und Linkskurven können viele hübsche Figuren erzeugt werden. Eine Anregung dazu:

```
to AUSSCHNITT :groesse :winkel
repeat 2 [RECHTSKURVE :groesse :winkel rt 180 - :winkel]
end

to ROSETTE :groesse :winkel :n
repeat :n [AUSSCHNITT :groesse :winkel rt 360/:n]
KREIS.3 :groesse ht
end
```

Wenn Sie beispielsweise versuchen

```
fs cs ht ROSETTE 15 80 9
```

oder

```
fs cs ht ROSETTE 17 80 9 ROSETTE 25 40 18
```

erhalten Sie folgendes Bild:

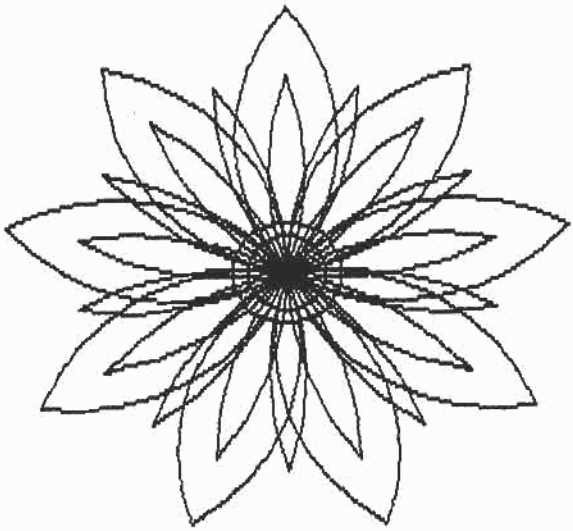


Abb. 7.1: Rosetten

Die Kombination mit Linkskurven gibt natürlich noch lustigere Möglichkeiten für Grafikspielereien!

7.4 Wiederholung durch Rekursion

Die bisher betrachteten Grafikprogramme hatten alle ein ganz einfaches Schema: ein Grundschritt, bestehend aus einer Vorwärtsbewegung und anschließender Rechtsdrehung; danach erfolgt eine Wiederholung des Grundschritts.

```
fd :seite rt :winkel;Grundschritt
```

Hinter dem Strichpunkt ist hier nicht etwa ein neues LOGO-Wort zu entdecken, mit Grundschritt soll vielmehr der Inhalt der Zeile erläutert werden. Eine Kommentierung des Programmtextes ist bei ganz einfachen, kurzen Programmen meist nicht nötig, wenn man für Namen im Programm möglichst sprechende, d.h. selbsterklärende, Namen verwendet.

Es kann aber leicht passieren, daß die Bedeutung eines Programms erst durch langes Nachdenken bzw. Probieren erkennbar wird. Hier helfen Kommentare im Programmtext, die gegenüber anderen Formen der Erläuterung den Vorteil haben, beim Arbeiten mit einem Programm stets präsent zu sein.

Der Beginn eines Kommentars wird durch das Semikolon und das Ende durch das Zeilenende festgelegt. Das Semikolon hat in der Syntax von LOGO genau diese Aufgabe. Alles, was in einer Programmzeile hinter dem Strichpunkt vorkommt, wird vom LOGO-System als Kommentar überlesen und das Programm wird mit der nächsten Programmzeile fortgeführt. Versehentlich in den Programmtext hineingekommene Strichpunkte führen also zu fehlerhafter Ausführung.

Die Wiederholung des Grundschritts geschah bisher mit repeat

```
repeat 1000 [fd :seite rt :winkel]
```

Stattdessen kann die Wiederholung auch in folgender Weise geschehen:

```
to POLYGON :seite :winkel
fd :seite rt :winkel;Grundschritt
POLYGON :seite :winkel;Rekursionsaufruf
end
```

In der vierten Zeile wird innerhalb der Tätigkeit POLYGON eben diese Prozedur selbst aufgerufen, ein solcher Ablauf wird Rekursion genannt.

Programme können also nicht nur andere Programme aufrufen, sie dürfen auch sich selbst rufen! Das hört sich zunächst etwas nach Münchhausen an, der sich am eigenen Schopf aus dem Sumpf zieht.

Probieren Sie zunächst einmal aus, ob denn das Programm Polygon überhaupt funtkioniert:

```
POLYGON 70 60
```

Es entsteht ein regelmäßiges Sechseck. Die Schildkröte scheint aber nun von Sinnen zu sein, denn sie läuft ständig auf dem Sechseck herum und findet offenbar kein Ende. Immerhin, die gewünschte Wiederholung des Grundschritts wird durch den Selbstaufruf erreicht.

Das Programm kann zwischendurch mit CONTROL+Z angehalten werden. In der Pause kann in den Ablauf, sogar in den Programmtext selbst eingegriffen werden. Mit co wird der Ablauf fortgesetzt.

Der Anblick der um das Sechseck hetzenden Turtle ist bald langweilig, das Programm sollte dann endgültig mit ESC (STOP beim Joyce) abgebrochen werden.

Bei diesem einfachen Beispiel ist die Wirkung des rekursiven Aufrufs offensichtlich: POLYGON veranlaßt das LOGO-System, die Prozedur neu zu starten, ein Prozeß, der so kein Ende

findet. Eine sinnvolle Anwendung dieser einfachen Art der Rekursion ist eigentlich nur bei Wiederholungen gegeben, deren Anzahl offengehalten wird und wobei der Abbruch bewußt durch einen Eingriff von außen erzielt werden soll.

7.5 Rekursion mit veränderten Eingaben

Ganz anders sieht das aus, wenn eine Prozedur sich selbst aufruft, dabei aber die Eingaben für die Prozedur geändert werden.

```
to POLYGON.2 :seite :winkel
fd :seite rt :winkel;Grundschritt
POLYGON.2 :seite+2 :winkel;Rekursionsaufruf
end
```

Die einzige Änderung besteht darin, daß beim Selbstaufruf die erste Eingabe nicht mehr :seite sondern :seite+2 heißt.

Am Ende müssen Sie

```
ed "POLYGON
```

eingeben. Im Edit-Fenster erscheint der Text der bereits definierten Prozedur POLYGON. Hängen Sie nun an Polygon (erste und vorletzte Zeile) ".2" an und schreiben hinter seite noch "+1". Nach Abschluß mit COPY beim CPC bzw. ALT+C beim Joyce wird Polygon.2 definiert, wobei die alte Prozedur POLYGON unverändert erhalten bleibt.

Experimentieren Sie ruhig erst einmal mit dem Programm, bevor Sie die Struktur des Programms genauer unter die Lupe nehmen, beispielsweise mit

```
cs POLYGON.2 1 90
cs POLYGON.2 1 60
cs POLYGON.2 1 45
```

Die entstehenden Figuren wachsen rasch über den Bildschirmrand hinaus, was übrigens im Wickelmodus zu ganz reizvollen Effekten führen kann. Statt den Ablauf manuell durch ESC (STOP) abzubrechen, kann man hier auch einen Abbruch mit fence erreichen, sobald der Zeichensstift über den Rand hinauskommandiert wird.

```
cs fence POLYGON.2 1 12Ø
```

Mit CONTROL+Y kann diese Eingabezeile immer wieder erzeugt werden, wobei nur noch der Winkel verändert werden muß. Wenn zu viele Linien übereinanderlaufen, kann man die vierte Zeile so abändern, daß die Vergrößerung der Seite schneller erfolgt, etwa :seite+2.

Versuchen Sie es jetzt einmal mit Winkeln, die nicht Teiler von 360 oder Vielfachen davon sind, z.B.

```
cs fence POLYGON.2 1 49
```

Raten Sie erst einmal, was wohl bei folgendem Aufruf entsteht

```
cs fence POLYGON.2 1 91.
```

Leichter ist jetzt vielleicht schon, die Gestalt zu erraten, die bei einem Winkel von 89 Grad entsteht!

Verblüffend ist bei solchen Programmen, welche unterschiedlichen Figuren allein durch Änderung des Winkels entstehen können. Solche Grafiken können auch ohne Rekursion erzeugt werden, die hier allerdings besonders elegante Möglichkeiten liefert.

Nun zurück zum rekursiven Aufruf selbst. Um genau zu verstehen, wie ein rekursives Programm ausgeführt wird, kann mit großem Nutzen die Option trace eingesetzt werden. Schalten Sie also vor dem Aufruf mit trace den Protokollmodus ein, und falls ein Drucker angeschlossen ist, schalten Sie auch diesen mit copyon hinzu.

```
cs trace fence POLYGON.2 20 90
```

Auf dem Bildschirm finden Sie dann das Protokoll der Prozeduraufrufe:

```
[1] Evaluating POLYGON.2
[1] winkel is 90
[1] seite is 20
[2] Evaluating POLYGON.2
[2] winkel is 90
[2] seite is 22
[3] Evaluating POLYGON.2
[3] Evaluating POLYGON.2
[3] winkel is 90
[3] seite is 24
...
```

Bei jedem Prozeduraufruf erscheint im Trace-Modus zunächst eine Zeile, die die Ausführung bestimmter Prozeduren anzeigt. Anschließend werden die aktuellen Werte der Eingaben ausgegeben. Am Protokoll erkennt man, daß die Rekursion hier genauso arbeitet, als wenn dieselbe Prozedur dauernd mit neuen, veränderten Eingabewerten aufgerufen würde.

Es ergibt sich daraus, daß diese einfache Art der Rekursion auch durch eine andere Programmkonstruktion ersetzt werden kann.

```
to POLYGON.3 :seite :winkel :n;ohne Rekursion
epeat :n [fd :seite rt :winkel make "seite :seite+1]
end
```

Die Anzahl der Wiederholungen müssen bei repeat angegeben werden, man könnte hier praktisch denselben Endloseffekt mit einem sehr hohen Wert für n erreichen:

```
POLYGON.3 10 90 10000
```

Wiederholungen von Tätigkeiten ohne Rekursion nennt man auch die Iteration von Tätigkeiten.

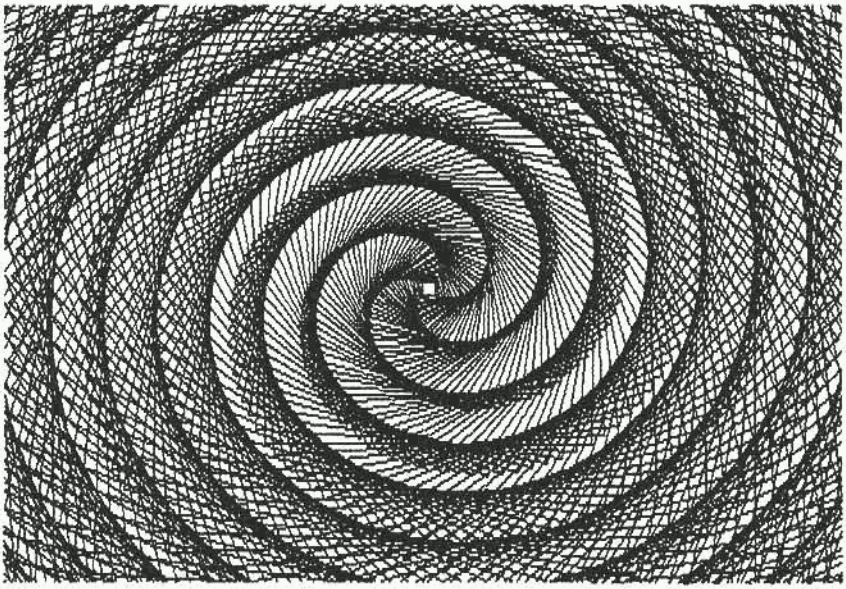


Abb. 7.2: Ergebnis von POLYGON.3 mit einem Winkel von 89 Grad

Die Wiederholung mit Rekursion ist deswegen eine elegante, wenn auch nicht unbedingt erforderliche Lösung, weil die Erhöhung der Schrittweite seite hier über den Prozeduraufruf erfolgt. Typisch für iterative Programmierung desselben Vorgangs ist die Verwendung von make in POLYGON.3.

Die bisher besprochenen Rekursionen sind von sehr einfacher Art: Eine Prozedur ruft sich am Ende selbst wieder auf. Dieser Rekursionstyp ist leicht zu durchschauen, und Sie sollten auch nicht zögern, die Rekursion dafür zu nutzen. Programmieren mit Rekursion stellt ein leistungsfähiges Instrument dar, vorläufig soll aber eine erste Einführung in die einfache Form genügen.

Leider wird die Rekursion in der LOGO-Version auf den Schneidercomputern nicht sehr effizient abgewickelt - andere Versionen auf vergleichbaren Mikrocomputern sind hier deutlich überlegen!

7.6 Stickmuster: Chaotische Kurven

Was kann bei einem so einfachen Programm wie dem folgenden schon entstehen?

```
to STICKEN :groesse :n
fd :groesse (rt :n)
STICKEN :groesse :n +1
end
```

Preisfrage: Was passiert beim Aufruf STICKEN 10 0? Lassen Sie sich überraschen!

Der Zeichenvorgang ist einfach: Es erfolgt jeweils eine Vorwärtsbewegung mit fester Länge und anschließend eine Rechtsdrehung. Dabei wächst der Drehwinkel bei jedem Schritt an. Es entsteht eine Folge von Drehwinkeln, wobei die Folge hier durch :n erzeugt wird; es handelt sich einfach um die Folge der natürrlichen Zahlen. Stattdessen kann man aber irgendwelche andere Zahlenfolgen verwenden, was zu sehr unterschiedlichem Bildern führen kann.

Einfache Beispiele für andere Zahlenfolgen sind:

a) :n * 10	b) :n * 10 +1
c) :n * 10 +3	d) :n * 10 +4
e) :n * 7	f) :n * :pi

Die Beispiele a) bis f) sind so zu verstehen, daß in der zweiten Zeile von STICKEN der jeweils angegebene Ausdruck den Aufruf von :n hinter rt ersetzen soll. Beim Experimentieren werden Sie feststellen, daß kleine Änderungen erhebliche Auswirkungen auf die entstehenden Stickmuster haben können.

Noch weniger sind die Kurven voraussagbar, wenn etwa kompliziertere Formeln in der Prozedur A benutzt werden:

g) :n * :n

Dem Experimentieren sind hier keine Grenzen gesetzt. Die Größe der entstehenden Muster kann nach Belieben mit der ersten Eingabe von STICKEN gesteuert werden. Zusätzlich kann das Wickeln noch zu besonderen Effekten führen.

Wenn Ihnen das Programm zu langsam arbeitet oder Sie die Fehlermeldung I'm out of space bekommen, können Sie auch mit der iterativen Formulierung arbeiten.

```
to STICKE :groesse :anzahl
local "n make "n 1
repeat :anzahl [fd :groesse rt :n * :pi make "n :n+1]
end
```

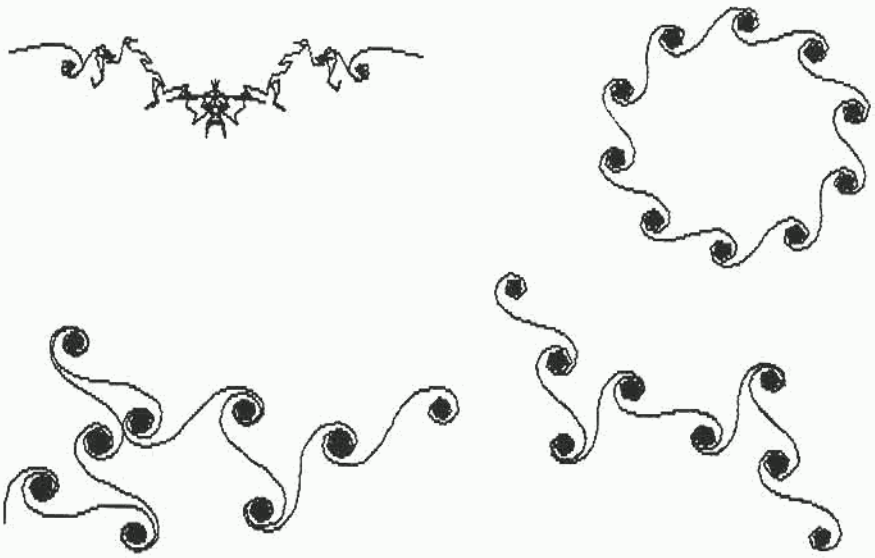


Abb. 7.3: Figuren, die durch STICKEN erzeugt werden

Ein Hinweis für den mathematisch interessierten Leser: Wenn die Zeichenebene als Ebene der komplexen Zahlen gedeutet wird, dann kann die Wirkung von STICKEN als Aufsummierung komplexer Zahlen gleichen Betrages interpretiert werden. Die Prozedur A stellt die Phase der komplexen Zahl dar. Diese Deutung der Zeichenebene legt es auch nahe, die Turtle-Grafik für eine grafische Methode zur sogenannten Fourier-Transformation zu nutzen.

7.7 Abbruch mit Bedingungen

Bei Zeichenprogrammen kann der manuelle Abbruch durch Tastendruck zweckmäßig sein, wenn man beispielsweise vorher keine klare Vorstellung von dem erzeugten Bild hat. Bei der Wiederholung mit repeat steht die Anzahl derselben schon vorher fest. Im allgemeinen Fall soll die Ausführung eines Programms dann abgebrochen werden, wenn eine bestimmte Bedingung erfüllt ist. Der einfachste Fall ist eine Abzählung der Wiederholungen analog zur Wirkung von repeat:

```
to POLYGON.2 :seite :winkel
if :seite > 100 [stop]
fd :seite rt :winkel;Grundschritt
POLYGON.2 :seite+1 :winkel;Rekursionsaufruf
end
```

Neu ist die zweite Zeile, sie hat die Form

```
if ... [...]
```
Wenn ... dann

Nach if steht ein Vergleich, im obigen Beispiel

```
:seite > 100.
```

Dieser Vergleich kann mit der Antwort wahr oder falsch enden, was man im Direktbetrieb an einer Eingabezeile beispielhaft demonstrieren kann.

```
1 > 100
```
Antwort: FALSE (Ergebnis: Falsch)

```
99 = 99
```
Antwort: TRUE (Ergebnis: Wahr)

```
4 < 10
```
Antwort: TRUE (Ergebnis: Wahr)

Ist das Ergebnis des Vergleichs nach if falsch, dann wird der Rest der Programmzeile bis zum nächsten RETURN ignoriert. Ist die Antwort dagegen wahr, wird der Rest der Programmzeile normal abgearbeitet, also auch der Inhalt der eckigen Klammern.

Im Programmbeispiel folgt das LOGO-Wort

```
stop
```

in eckigen Klammern, das sich eigentlich ja von selbst erklärt, denn hiermit wird die Ausführung der Prozedur beendet. Wurde die Tätigkeit selbst von einer anderen Prozedur aufgerufen, so wird darin weitergefahren. Eine ähnliche Wirkung hat das LOGO-Wort op, das Sie schon kennen. op beendet die Prozedur aber im Gegensatz zu stop mit der Ausgabe eines Ergebniswertes. Ob die Tätigkeit mit stop oder op beendet wird, hängt von der Aufgabe der betreffenden Prozedur ab. Sie wird natürlich auch beendet, wenn keine weitere LOGO-Anweisung mehr folgt, z.B. bei den einfachen Grafikprogrammen.

Kombinierte Vielecke

Schon die Wiederholung von Vorwärtsbewegung und anschließender Drehung kann zu sehr unterschiedlichen Figuren führen.

```
to ECKEN :seite :winkel
ECKE :seite :winkel :winkel
end

to ECKE :seite :winkel :gesamt
fd :seite rt :winkel
if (remainder :gesamt 360 ) = 0 [stop]
ECKE :seite :winkel :gesamt+:winkel
end

to NECKEN :seite :winkel
NECKE :seite :winkel :winkel
end

to NECKE :seite :winkel :gesamt
fd :seite rt :winkel fd :seite rt 2*:winkel
if (remainder :gesamt 360) = 0 [stop]
NECKE :seite :winkel :gesamt+:winkel
end
```

Die eigentlichen Zeichenprogramme sind ECKE und NECKE. Dagegen wurden ECKEN und NECKEN nur zum bequemeren Aufrufen zusätzlich formuliert. ECKE und NECKE funktionieren rekursiv, die Rekursion wird deswegen abgebrochen, weil sich stets geschlossene Figuren ergeben.

Probieren Sie ruhig einmal die verschiedensten Winkel aus. Eine Veränderung von seite hat nur Einfluß auf die Größe, nicht aber die Gestalt. Man muß aber immer erst herausfinden, welche Seitenlängen bei gegebenem Winkel sinnvoll sind.

Die Veränderung von seite bzw. winkel führt zu spiraligen Figuren. Die Eingabegröße d gibt den jeweiligen Zuwachs an:

```
to SPIRALE :seite :winkel :d
fd :seite rt :winkel
SPIRALE :seite+:d :winkel :d
end

to NETZ :seite :winkel :d
fd :seite rt :winkel
NETZ :seite :winkel+:d :d
end
```

Kompliziertere Figuren bekommt man, wenn verschiedene Vielecke miteinander kombiniert werden. Hier soll eine iterative Formulierung gewählt werden.

```
to DOPPELECKEN :s1 :w1 :s2 :w2 :n
local "w make "w :w2
repeat :n [rt :w1 fd :s1 rt :w fd :s2 lt :w make "w :w+:w2]
end
```

s1, w1 und s2, (w2+w1) sind Seiten und Winkel zu zwei verschiedenen Vielecken. Hier werden die Grundschritte zu beiden Vielecken abwechselnd hintereinander ausgeführt, wobei der zweite Winkel durch die beiden Drehungen um w1 und w2 zustandekommt.

Probieren Sie einmal die Kombinationen von Eingabewerten:

(50 90 50 230 36), (50 90 50 210 12),
(10 10 10 (-25) 72), (15 19 15 (-39) 360)

Es sollten sich dann die Grafiken von Abb. 7.4 ergeben.

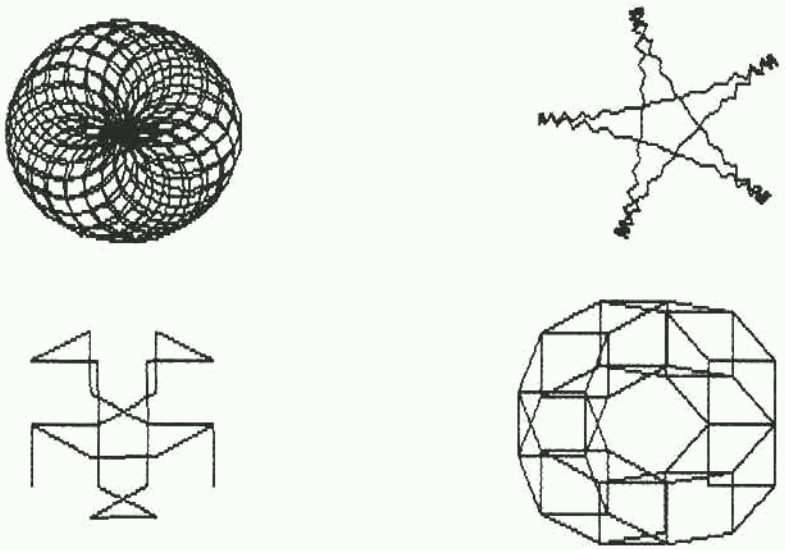


Abb. 7.4: Grafiken mit DOPPELECKEN

Übungen

1) Es sollen konzentrische Kreise mit wachsenden Radien gezeichnet werden.

2) Schreiben Sie eine Prozedur SONNE, die eine Sonne mit vielen Sonnenstrahlen auf dem Bildschirm eerzeugt.

3) Das Programm MONDPHASEN soll Mondsicheln in den verschiedenen Mondphasen vom Vollmond bis zum Neumond zeichnen.

4) Definieren Sie die Prozedur LINKSKURVE analog zum Programm RECHTSKURVE.

Lektion 8: Die Bedienung des LOGO-Systems

Neue Sprachelemente:
save, load, dir, erasefile, changef, bye, savepic, loadpic, erasepic, dirpic, ct, clean, cs, recycle, nodes

8.1 Der Umgang mit Disketten

Die anfangs betrachteten Programmbeispiele waren so kurz, daß die Eingabe über Tastatur wenig Mühe macht. Nach einiger Zeit werden Sie aber sicher das Ergebnis Ihrer Programmieranstrengungen für spätere Verwendung aufheben wollen. Es kann auch häufig vorkommen, daß eine LOGO-Sitzung unterbrochen werden muß, bevor Sie das gewünschte Ziel erreicht haben. Auch dann sollte der beim Abbruch vorhandene Zustand für die nächste Sitzung verfügbar gemacht werden.

Abspeichern

Der Grundwortschatz von DR LOGO ist auf einer eigenen Systemdiskette abgespeichert, die am besten nur dafür verwendet wird. Das aktuelle LOGO-System enthält darüber hinaus die durch Programmierung entstandenen Spracherweiterungen, also die Prozeduren, die Sie entweder selbst geschrieben oder von einer Diskette eingelesen haben.

Zum LOGO-System gehören auch die globalen Namen mit ihrem zugehörigen Inhalt.

Zur Sicherung des nicht auf der System-Diskette vorhandenen Teils des LOGO-Systems wird die LOGO-Prozedur save benutzt:

```
save "kreise
```

Damit wird alles, was sowohl an Spracherweiterungen wie Namen zum gegenwärtigen Zeitpunkt vorhanden ist, auf der im Laufwerk befindlichen Diskette abgespeichert. Der Name "kreise würde kenntlich machen, daß die zum Zeichnen von Kreisen in

der vorangegangenen Lektion betrachteten Prozeduren abgespeichert werden sollen.

Mit save kann bei DR LOGO ein vorhandener File auf der Diskette nicht überschrieben werden; bei übereinstimmenden Filenamen wird eine Fehlermeldung ausgegeben. Sie können aber das Programm SPEICHERN aus Lektion 12 benutzen. Bei diesem Programm wird der vorhandene File umbenannt und zur Reserve aufgehoben; der aktuelle Inhalt des Arbeitsspeichers wird wie verlangt abgespeichert.

Wenn Sie nur einen Teil der im Laufe der LOGO-Sitzung entstandenen Prozeduren oder Namen speichern wollen, können Sie die nicht gewünschten Teile löschen. In Lektion 15 finden Sie unter der Bezeichnung BEHALTEN auch ein nützliches Hilfsprogramm, mit dessen Hilfe alle Programme außer den als Eingabe festgelegten gelöscht werden, was dann von Vorteil ist, wenn sich schon viele, nicht mehr benötigte Prozeduren angesammelt haben.

Am besten sehen Sie sich vor dem Arbeiten mit save erst einmal an, was überhaupt vorhanden ist:

```
poall
```

Übersichtlicher wird die Ausgabe, wenn Sie sich nur die Namen und Prozedurnamen auflisten lassen mit

```
pots pons
```

Das Löschen geschieht mit der LOGO-Vokabel er (Abkürzung von Erase).

```
er "KREIS.3
```

Damit wird die Prozedur KREIS.3 gelöscht, mit der Zeile

```
er [KREIS.2 KREIS.3 ROSETTE]
```

werden die aufgezählten Prozeduren gelöscht.

Durch

```
ern "mwstsatz
```

wird der globale Name mwstsatz aus dem System entfernt. Mehrere Namen werden gelöscht, indem diese mit eckigen Klammern zusammen gefaßt werden.

Man kann auch alle Prozeduren bzw. alle globalen Namen mit Hilfe von

```
er glist ".DEF bzw. ern glist ".APV
```

auf einmal löschen. Dabei wird die LOGO-Funktion glist benutzt, die im Zusammenhang mit den sogenannten Eigenschaftslisten in der folgenden Lektion erst ausfühlicher betrachtet werden soll.

Beachten Sie, daß in LOGO alle zum jeweiligen Zeitpunkt vorhandenen Inhalte des Systems durch save abgespeichert werden können. Die Unterbrechung einer LOGO-Sitzung bereitet deshalb keine Probleme. Zur Fortsetzung muß dann der zuletzt mit save abgespeicherte Teil eingelesen werden. Danach ist die Ausgangssituation wiederhergestellt.

Mit save "kreise wird ein sogenannter LOGO-File auf der Diskette erzeugt. Wenn Sie einen langen Filenamen verwenden, werden nur die ersten acht Zeichen berücksichtigt. Das System hängt dabei automatisch die Verlängerung ".LOG" an den Filenamen an, worum Sie sich innerhalb von LOGO aber gar nicht kümmern müssen. Erst auf der Ebene des CP/M-Betriebssystems wird diese Kennzeichnung sichtbar. Der File ist für das Betriebssystem ein Textfile, dessen Inhalt Sie sich übrigens auch

ohne DR LOGO direkt auf dem Bildschirm ansehen oder über den Drucker ausgeben können (z.B. CP/M-Kommando type).

Das Verzeichnis der vorhandenen LOGO-Files

Mit dem Kommando

```
dir
```

werden die Namen der auf der Diskette vorhandenen LOGO-Files zu einer Liste zusammengefaßt, was an den eckigen Klammern erkenntlich ist. Das Inhaltsverzeichnis kann auch mit einem Namen verbunden und damit aufgehoben werden.

```
make "inhalt dir
```

Einlesen von Diskette

Ein LOGO-File kann mit der Prozedur load eingelesen werden:

```
load "kreise
```

LOGO reagiert auf den Ladevorgang wie bei der Übergabe von Programmtexten nach dem Editieren. Die einzelnen Prozeduren werden als nunmehr definiert gemeldet. Es können mehrere LOGO-Files hintereinander eingelesen werden, ebenso wie man mit dem Editor in mehreren Etappen Programmtexte erstellen und dem LOGO-System einverleiben kann.

Beim Einlesen können keine Jokerzeichen, die in den Schneiderhandbüchern auch Universalzeichen heißen, verwendet werden, d.h. man muß den Filenamen richtig und vollständig angeben.

Löschen und Ändern von Eintragungen auf der Diskette

Der LOGO-File kreise kann mit

```
erasefile "kreise
```

gelöscht werden. Beachten Sie aber, daß LOGO bei erasefile keine Bestätigung verlangt, mit diesem Befehl sollten Sie demnach vorsichtig umgehen.

Bevor man einen File endgültig mit erasefile löscht, kann man ihn vorsichtshalber unter anderem Namen aufheben.

```
changef "kreisalt "kreise
```

Damit sollte der alte File den Namen kreisalt erhalten. In den dem Autor voliegeden LOGO-Versionen funktioniert dieser Befehl bedauerlicherweise nicht. LOGO reagiert mit der Antwort

```
File kreis not found
```

Es ist zu hoffen, daß dieser Systemfehler vom Hersteller beseitigt wird.

Weitere Diskettenoperationen

Vor dem ersten Abspeichern müssen Disketten formatiert werden, und zwar außerhalb von LOGO auf der Ebene des CP/M-Betriebssystems. Auf dieser befinden Sie sich vor dem Laden des LOGO-Systems. In das Betriebssystem kann man auch durch die Beendigung der LOGO-Sitzung mit

```
bye
```

hineinkommen. Beachten Sie aber, daß der Hersteller hier unverständlicherweise keine Sicherheitsabfrage eingebaut hat, so daß es keine Rettung für eventuell verlorengegangene Programme oder Daten gibt, wenn der Befehl eingegeben wurde.

Die Disketten werden mit dem Dienstprogramm diskkit bzw. diskkit3 formatiert, wobei Sie zum Absspeichern von LOGO-Files eine Datendiskette (keine Systemdiskette) benötigen.

Mit demselben Dientsprogramm können Disketten auch kopiert werden. Die Anfertigung von Sicherheitskopien in regelmäßigen Zeitabständen ist empfehlenswert.

Der Hersteller hat das Diskettenformat geändert; erfahrungsgemäß wirkt sich das bei LOGO-Disketten "aufwärtskomatibel" aus, d.h. Disketten im alten Format können von den neueren Systemen, z.B. beim Joyce, zum Abspeichern wie zum Lesen von LOGO-Files benutzt werden. Disketten im neuen Format werden dagegen von den älteren Systemen nicht verarbeitet.

8.2 Laden, Speichern, Löschen und Drucken von Grafiken

Auch der Inhalt des Grafikfensters kann unmittelbar auf Diskette gespeichert und von dort auch wieder eingelesen werden, wofür die beiden LOGO-Kommandos savepic und loadpic vorgesehen sind.

```
savepic "muster cs
loadpic "muster
```

Beim Laden eines Bildes geht LOGO standardmäßig in den splitscreen-Modus, d.h. der Bildschirm wird in ein Grafik- und ein Textfenster aufgeteilt. Wenn Sie ein Bild einladen, das den ganzen Bildschirm beansprucht, sollten Sie diesen Zustand vor dem Laden herbeiführen.

```
fs cs ht loadpic "muster
```

In einer Bildschirmgrafik mit großer Auflösung ist viel Information enthalten. Die zugehörigen Files beanspruchen viel Platz, und beim Laden wie beim Speichern muß man einige Sekunden warten.

Eine abgespeicherte Grafik kann mit dem Kommando

```
erasepic "muster
```

wieder von der Diskette gelöscht werden.

Das Inhaltsverzeichnis der auf der Diskette gespeicherten Bilder bekommt man mit dem Kommando dirpic.

```
dirpic
```

Zum Ausdrucken von Grafiken haben die neueren Mikrocomputer eine sogenannte Hardcopyfunktion eingebaut, mit der eine direkte Kopie des Bildschirms auf dem Drucker erzeugt wird. Beim Schneider-Joyce wird diese Funktion mit der Tastenkombination

EXTRA+PTR

ausgelöst. So wurden beispielsweise die Grafiken für das vorliegende Buch zu Papier gebracht.

Beim CPC ist die Hardcopyfunktion bedauerlicherweise gegenwärtig noch nicht vorhanden. Hier ist man zunächst auf die Hilfe der Fotografie angewiesen, wozu es einer gewissen Übung bedarf, die Belichtung richtig zu wählen. Ein Bildschirmfoto kann auf der anderen Seite natürlich auch die Farben wiedergeben, wenn Sie einen Farbmonitor besitzen.

Als Alternative bietet sich auch die Ausgabe des auf Diskette abgespeicherten Bildfiles mit einem geeigneten Programm an.

8.3 Löschen

Bildschirm Löschen

Wenn Sie das Dialogfenster mit Eingaben und Systemantworten beschreiben, so sind Sie bald am unteren Rand angekommen. Wenn Sie weitere Zeilen benötigen, wird durch das sogenannte Scrollen der gesamte Bildschirminhalt automatisch nach oben geschoben, um am unteren Rand Platz zu schaffen.

Da es als störend empfunden wird, wenn der Bildschirm vollgeschrieben ist, kann der Textschirm gelöscht werden mit:

```
ct (Abkürzung für Cleartext)
```

Das Grafikfenster wird gelöscht mit

```
clean
```

clean hat sonst keine Folgen. Soll gleichzeitig die Turtle in die Aussgangsposition (einschließlich der Ausrichtung nach "Norden") gebracht werden, empfiehlt sich stattdessen

```
cs (Abkürzung für Clearscreen)
```

Programme und Namen Löschen

Der Speicherplatz Ihres Mikrocomputers ist begrenzt. Wegen seiner hohen Fähigkeite braucht das LOGO-System selbst eine Menge an Speicherplatz. Andererseits kann man leistungsfähige Programme in LOGO sehr kurz formulieren. Trotzdem kann es passieren, daß der Speicherplatz knapp wird.

Wenn Prozeduren editiert und neue Versionen unter anderem Namen in das LOGO-System aufgenommen werden, können sich auf Dauer eine ganze Reihe von Programmen anhäufen, die gar nicht mehr benötigt werden. Wenn der Platz knapp wird, sollte man erst einmal mit pots (Abkürzung für Printout Titles) eine

Übersicht über die vorhandenen Prozeduren erstellen, um überflüssige Prozedurnamen herauszufinden.

Wie Prozeduren und Namen mit Hilfe der LOGO-Vokabeln

```
er und ern
```

gelöscht werden können, wurde bereits am Anfang der Lektion erläutert. Eingaben sind jeweils entweder ein Prozedur- bzw. Variablenname, der dann mit Anführungszeichen eingeleitet werden muß, oder eine in eckigen Klammern eingeschlossene Liste solcher Namen.

```
er "kreis , er [kreis steuer], er glist ".DEF
ern "mwsteuersatz , ern [pi a zahl], er glist ".APV
```

Die Verwendung der Funktion glist wurde ebenfalls schon zu Anfang erläutert.

Müllbeseitigung

Das LOGO-System erzeugt Müll, also intern belegter Speicherplatz, der nicht mehr benötigt wird. Es ist für das System nicht ökonomisch, bei jedem Schritt zu überprüfen, welcher Speicherplatz möglicherweise freigegeben werden kann. Man kann aber durch das Kommando

```
recycle
```

eine "Müllbeseitigung" veranlassen. Bevor man daran geht, etwas mit er bzw. ern zu löschen, sollte erst versucht werden, über eine Müllsammlung Platz zu schaffen. Die vom System gelegentlich selbst durchgeführte Müllsammlung kann störende Pausen hervorrufen. Mit recycle kann der Benutzer dagegen den Zeitpunkt der Pause selbst steuern.

Freier Platz

Sie können sich jederzeit über die Entwicklung des für LOGO zur Verfügung stehenden Platzes informieren lassen durch

```
nodes
```

nodes steht für Knoten. Ein Knoten ist so etwas wie die Grundeinheit in Datenstrukturen, hier in dem von LOGO zu verwaltenden Programmen und Namen. Mit dieser Angabe können Sie im Grunde erst durch häufige Beobachtung etwas anfangen. Wird eine Zahl über 2000 ausgegeben, ist noch viel Platz frei. Bei 100 freien Knoten ist der Platz natürlich schon knapper geworden. Um sich über den tatsächlichen Speicherbedarf zu informieren, sollten Sie zuvor recycle befehlen.

8.4 Selbstladende Programmfiles

Anwenderprogramme bestehen häufig aus mehreren Prorammteilen. Es ist deshalb sinnvoll, den Ladevorgang eines ganzen zusammengehörenden Programmpakets automatisch in Gang setzen zu lassen. Bei DR LOGO wird nach dem Laden des LOGO-Systems selbst ein LOGO-File mit der Bezeichnung STARTUP gesucht und automatisch geladen. Wenn Sie also erreichen wollen, daß nach dem Start des LOGO-Systems automatisch eine Reihe von Programmen ohne weiteres Zutun geladen wird, dann können Sie diese Programme (oder auch den Inhalt von Namen) unter der speziellen Bezeichnung STARTUP abspeichern. Wenn sich beim Start des LOGO-Systems von der Language-Disk aus dort ein LOGO-File mit der Bezeichnung STARTUP.LOGO befindet, wird er ohne zusätzliche Aufforderung geladen.

Diese Option ist nur sinnvoll für gebrauchsfertige Programme. Man kann damit häufig benötigte Spracherweiterungen zu DR LOGO mit dem LOGO-Grundwortschatz zusammen auf eine Diskette schreiben. Es ist dann nicht mehr notwendig, beim Laden und Starten des Systems die Hilfsprogramme herauszusuchen und hinzuzuladen.

Bei anderen LOGO-Versionen gibt es unter der Bezeichung STARTUP zusätzlich bzw. alternativ die Möglichkeit, ein Programm nach dem Laden automatisch in Gang zu setzen. Bei DR LOGO muß eine Startprozedur explizit aufgerufen werden. Sie können sich natürlich angewöhnen, bei größeren Programmpaketen immer eine Prozedur unter einem festen Namen (z.B. STARTUP) vorzusehen, mit dem dann das Programm gestartet wird.

8.5 Arbeiten mit zwei Laufwerken

Das Laufwerk, das Sie beim Laden von DR LOGO benutzt haben, wird als Standardlaufwerk (Default disk) bei den Kommandos load, save, erasefile benutzt. Wenn Sie ein zweites Laufwerk ansteuern wollen, müssen Sie es ausdrücklich benennen.

```
dir "b:
```

Alle LOGO-Files der Diskette in Laufwerk B (2. Floppy) werden ausgegeben.

```
save "b:Kreise
```

Damit wird der Arbeitsspeicher unter dem Filenamen Kreise auf dem Laufwerk B abgespeichert.

```
load "b:Steuer
```

Der LOGO-File Steuer wird vom Laufwerk B geladen.

Es gibt auch die Möglichkeit, mit Hilfe von

```
setd "b:
```

sämtliche Diskettenoperationen auf das genannte Laufwerk zu steuern. Man erspart sich dann, die Laufwerksbezeichnung jeweils vor die Filenamen setzen zu müssen.

Beim Schneider Joyce gibt es auch eine sogenannte RAM-Disk unter der Bezeichnung m. Dabei handelt es sich eigentlich um einen Speiecherbereich im Rechner, auf den man in gleicher Weise wie auf ein Diskettenlaufwerk zugreifen kann. Es wird also ein Laufwerk simuliert, wobei der Zugriff schneller als bei einem echten Laufwerk erfolgt. Weil LOGO-Programme wenig Platz beanspruchen, kommt dieser Vorteil allerdings bei LOGO kaum zum Tragen.

Lektion 9: Wörter und Listen

Neue Sprachelemente:
item, count, piece, se, show, first, bf, last, bl, emptyp, fput, lput, memberp, where, numberp, wordp, listp, equalp, not, shuffle

Programme:
ZERLEGE, PLAPPERN, KEIL, SPIEGEL, SPIEGEL.SATZ, DADA, STO, RT.TAUSCH, LISTEN.TAUSCH, LOTTO, ZIEHUNG, KUGELN, ERSETZE

9.1 Zeichenketten als Wörter

In einer Sprache werden Sätze aus Wörtern gebildet, Wörter selbst sind wieder aus Buchstaben aufgebaut. Für weitere sinnvolle Verknüpfungen reichen die Buchstaben des Alphabets nicht aus, deshalb verwenden wir Sonderzeichen wie „.;" usw. zur Unterstützung. Auf der Tastatur Ihres Rechners finden Sie ebenfalls neben den Buchstaben eine größere Zahl von Sonderzeichen, wobei als zulässige Zeichen natürlich auch die zehn Ziffern zählen. Für den Computer haben einige Zeichen zwar spezielle Aufgaben, eigentlich werden alle zunächst einmal gleich behandelt.

Für LOGO bedeutet irgendeine Folge von Zeichen, Buchstaben, Ziffern und Sonderzeichen ein Wort.

Beispiele:

```
"Sonne
"Computer
"3.14159
"<<.>>
```

Nur die ersten beiden sind im normalen Sprachgebrauch ordentliche Wörter. In Computersprachen steht zunächst nicht die inhaltliche Bedeutung, sondern die Übereinstimmung mit den Regeln der Sprache im Vordergrund.

Alle Wörter beginnen mit einem einleitenden Anführungszeichen, dessen Bedeutung schon ganz zu Anfang im Zusammenhang mit Namen untersucht wurde. Das Anführungszeichen gehört nicht zum Wort, selbst, wenn es ganz am Anfang steht. Es signalisiert aber, daß die folgenden Zeichen ein Wort bilden. Was von LOGO als Wort angenommen wird, läßt sich durch pr leicht überprüfen:

```
pr "Sonne
pr ""Hurra"
pr "Zwei Woerter
```

Im ersten Beispiel sollte Sonne, beim zweiten "Hurra" auf dem Bildschirm erscheinen. Beim dritten gibt es eine Fehlermeldung:

```
I don't know how to Woerter
```

Zwei wird als Wort erkannt, Woerter wird dagegen als Prozedurname gedeutet und die Fehlermeldung zeigt an, daß eine solche Tätigkeit nicht bekannt ist.

> Der Beginn eines Wortes wird durch ein Anführungszeichen angezeigt, das nicht zum Wort selbst gehört. Alle Zeichen bis zum nächsten Trennzeichen bilden das Wort.

Als Trennzeichen dient in LOGO das Leerzeichen. Am Ende einer Zeile erübrigt sich ein Trennzeichen.

```
(pr "Zwei "Woerter)
```

Damit wird richtig Zwei Woerter ausgedruckt. Die runden Klammern sind notwendig, weil pr standardmäßig nur eine Eingabe erwartet. Die schließende runde Klammer ist ebenfalls ein Trennzeichen und gehört nicht zum Wort. DR LOGO führt die

Zeile auch richtig aus, wenn Sie die zweite Klammer weglassen. Am Ende einer Zeile werden Klammern automatisch geschlossen, was übrigens auch für eckige Klammern gilt.

Das Leerzeichen ist wie in der natürlichen Sprache auch zunächst nicht als Zeichen innerhalb von Wörtern möglich. Es ist allerdings auch möglich, die trennende Wirkung des Leerzeichens abzuschalten. Dazu muß vor das Leerzeichen ein Backslashsymbol, also ein gespiegeltes Divisionszeichen gesetzt werden:

```
pr "Zwei\ Woerter
```

Dieses Zeichen finden Sie beim CPC auf der Tastatur; es kann aber auch durch die Tastenkombination CONTROL + Q erzeugt werden, die auch beim Joyce in der Form Alt+Q zu benutzen ist.

9.2 Sätze als Listen

Aus Wörtern kann ein vollständiger Satz gebildet werden, wobei die Wörter üblicherweise durch Leerzeichen voneinander getrennt werden.

Eine durch Leerzeichen getrennte Folge von Wörtern ist für LOGO ein einfaches Beispiel einer Liste. Listen werden in eckige Klammern eingeschlossen, die selbst nicht zur Liste gehören, sondern nur Anfang und Ende der Liste angeben.

```
[Heute ist Sonntag]
[10 25 27 31 32 46]
```

Die erste Liste besteht aus drei, die zweite aus sechs Wörtern. Eine Liste kann als ganzes ausgedruckt werden:

```
pr [Heute ist Sonntag]
```

Die Antwort von LOGO ist wie gewünscht:

```
Heute ist Sonntag
```

Die eckigen Klammern werden also nicht mit ausgegeben. Die gleiche Ausgabe würde auch bei

```
(pr "Heute "ist "Sonntag )
```

erfolgen. Die Zeile mit der Liste ist offensichtlich die bequemere Variante.

Man kann der Ausgabe nicht ansehen, ob mehrere Wörter nacheinander oder eine aus mehreren Wörtern gebildete Liste gedruckt wird. Wollen Sie auf dem Bildschirm erkennen, ob eine Liste ausgegeben wurde, dann haben Sie die Möglichkeit, statt pr die LOGO-Prozedur show zu benutzen.

```
show [Heute ist Sonntag]
```

Dabei werden die eckigen Klammern auch zur Kennzeichnung der Liste ausgedruckt.

Listen werden natürlich nicht nur ausgedruckt; es gibt vielmehr eine ganze Reihe von LOGO-Vokabeln um Listen zu handhaben. So können sie beispielsweise einem Namen zugeordnet werden. Dabei ist ein Name selbst ein Wort.

```
make "Liste [Heute ist Sonntag]
```

Danach ist die Liste [Heute ist Sonntag] unter dem Namen Liste verfügbar.

```
pr :Liste
```

Beachten Sie den Doppelpunkt vor Liste, der anzeigt, daß der Inhalt der mit dem Namen Liste versehenen Schublade gemeint ist, also: Drucke den Wert von Liste!

Genau so können auch Wörter selbst mit Namen verbunden werden

```
make "Wort "Donaudampfschiffahrtskapitaensmuetze
```

Damit wird eine Abkürzung für das lange Wort festgesetzt.

Zum erstenmal sind Listen im Zusammenhang mit der Wiederholung durch repeat vorgekommen:

```
repeat 10 [pr 1+random 6]
```

Eine Liste kann also auch Prozedurnamen enthalten. Die angegebene repeat-Liste kann mit einem Namen versehen werden und dann unter diesem benutzt werden

```
make "REP.LISTE [pr 1+random 6]
repeat 10 :REP.LISTE
```

9.3 Sätze aus Sätzen: Hierarchische Listen

Aus Wörtern können Sätze gebildet werden. Viele Sätze können wiederum ein ganzes Kapitel ausmachen, mehrere Kapitel ein Buch, viele Bücher eine Bibliothek.

Auf diese Weise entsteht eine Hierarchie, die auf der Grundeinheit des Wortes aufbaut. Aus einer Bibliothek kann dann umgekehrt etwa das 500. Buch gesucht werden, in diesem das 7. Kapitel, dort wieder der dritte Satz, im Satz selbst das zweite Wort. Hierbei kann auf jede Einheit (Buch, Kapitel, Satz, Wort) unabhängig als Ganzes zugegriffen werden.

Was man in dieser Weise aus Wörtern aufbauen kann, nennt man eine Liste. Die bisher betrachteten Sätze in der Liste bei der Textausgabe mit pr oder der repeat-Liste stellen die nächst höhere Ebene nach den Wörtern selbst dar. Die Möglichkeit, eine Hierarchie von Ebenen aufbauen zu können, erhebt die Listen zu einem vielfältigen Instrument.

Kompliziertere Listenstrukturen werden erst später betrachtet. Zunächst sollen die wichtigsten Operationen mit Listen und Wörtern an Hand einfacher Beispiele erläutert werden.

9.4 Zerlegen von Listen und Wörtern

Listen und Wörter können im Programm nach Wunsch manipuliert werden. Notwendig sind dazu Grundoperationen zum Zerlegen und Zusammenbauen, die sowohl auf Listen wie auch auf Wörter angewandt werden können.

Wörter und Listen können einmal in ihre einzelnen Bestandteile zerlegt werden, so wie ein Satz in die einzelnen Wörter und ein Wort in einzelne Buchstaben zerlegt werden kann. Um ein bestimmtes Element herauszugreifen, dient die Funktion item:

```
make "Satz [Heute scheint die Sonne]
item 2 :Satz
```

Antwort: scheint

Die Operation item liefert auch ein bestimmtes Zeichen in einem Wort.

```
make "Wort "Donaudampfschiffahrtskapitaensmuetze
item 25 :Wort
```

Antwort: i

Es kann passieren, daß ein gewünschtes Element gar nicht existiert, weil die Liste zu klein, bzw. das Wort zu kurz ist.

```
item 5 :Satz
```

Es erscheint die Fehlermeldung:

```
Too few items in [Heute scheint die Sonne]
```
(Es gibt zu wenig Elemente in der Liste ...)

Die Anzahl der vorhandenen Elemente kann durch Abzählen mit der LOGO-Funktion count bestimmt werden.

```
count :Satz
```
Antwort: 4

```
count :Wort
```
Antwort: 36

Vorwärts und rückwärts Zerlegen

Damit können wir schon eine kleine Prozedur schreiben, die eine Liste oder ein Wort in alle vorhandenen Elemente zerlegt und diese untereinander ausdruckt.

```
to ZERLEGE :objekt :n
if :n > count :objekt [stop]
(pr :n item :n :objekt)
ZERLEGE :objekt :n+1
end
```

Die Prozedur ZERLEGE ruft sich selbst rekursiv auf, um das jeweils nächste Element zu bestimmen. In der zweiten Zeile wird der Abbruch ausgelöst, wenn die Anzahl der Elemente durch die Zählvariable n schon überschritten ist.

```
ZERLEGE :Satz 1
```

LOGO antwortet mit:

```
1 Heute
2 scheint
3 die
4 Sonne
```

In der dritten Zeile von ZERLEGE wird die Zählvariable n zum besseren Lesen mit ausgegeben. Beim langen Wort Donau... soll der Aufruf erst mit dem 31. Element beginnen, um Platz zu sparen.

```
ZERLEGE :Wort 31
```

Antwort:

```
31 m
32 u
33 e
34 t
35 z
36 e
```

Das folgende Programm schreibt eine Liste, bzw. ein Wort mit umgekehrter Reihenfolge der Elemente auf den Bildschirm.

```
TO RUECKWAERTS :objekt :n
if :n = 0 [stop]
type item :n :objekt
RUECKWAERTS :objekt :n-1
end
```

Bei der Anwendung auf Wörter mit

```
RUECKWAERTS :Wort count :Wort
```

entsteht das gespiegelte Wort auf dem Schirm. Die Ausgabe erfolgt mit type, so daß die einzelnen Elemente unmittelbar hintereinander geschrieben werden.

Beachten Sie auch die Eingabe. Das Objekt wird von hinten her zerlegt, beginnend mit dem Element, dessen Nummer der zweite Eingabewert von RUECKWAERTS angibt. Deswegen wird als zweiter Eingabewert hier beim Aufruf das Ergebnis der Funktion count übergeben.

Soll eine Liste, z.B. ein Satz rückwärts geschrieben werden, empfiehlt es sich, zwischen die Elemente Leerzeichen auszudrucken. Dazu kann die dritte Zeile von RUECKWAERTS wie folgt abgeändert werden:

```
(type item :n :objekt "\ )
```

Nach dem Aufruf RUECKWAERTS :Satz count :Satz sollte dann erscheinen

```
Sonne die scheint Heute
```

Teilstücke von Wörtern und Listen

Mit item wird genau ein Element aus einem Wort, bzw. einer Liste herausgeholt. Häufig will man aber ein ganzes Teilstück eines Wortes, bzw. einer Liste erhalten. LOGO stellt für diesen Zweck eine eigene Vokabel zur Vefügung:

```
piece 6 16 :Wort
```
Antwort: `Dampfschiff`

Mit piece wird ein Teilstück aus der angegebenen Liste oder dem angegebenen Wort ausgegeben, wobei die ersten beiden Eingaben von piece Anfang und Ende des Teilstücks angeben.

Auch Zahlen sind für LOGO Wörter, probieren Sie doch mal:

```
make "pi 3.1415926535898 piece 3 7 :pi
```
Antwort: `14159`

Hier wird die dritte bis siebte "Stelle" der Zahl pi ausgegeben. Der Vergleich mit dem Resultat von pi zeigt, daß der Dezimalpunkt dabei mitgezählt wird. Es wird nämlich das dritte bis siebte Zeichen im Wort 3.141592 genommen.

Versuchen Sie es nun mit einer Liste als Eingabe:

```
piece 2 3 :Satz
```
Antwort: [scheint die]

Die äußeren eckigen Klammern zeigen an, daß das Ergebnis eine Liste ist. Bei einer hierarchischen Liste auf der nächsten Stufe, d.h. einem aus Sätzen gebildeten Abschnitt

```
make "abschnitt!
[[Heute ist Sonntag] [Das Wetter ist scheusslich]
[Ich habe keine Lust] [Wir warten auf Godot]]

piece 2 3 :abschnitt
```

erzeugt piece als Ergebnis einen Abschnitt aus dem zweiten bis dritten Satz:

```
[[Das Wetter ist scheusslich] [Ich habe keine Lust]]
```

Beachten Sie den Unterschied zwischen dem Ergebnis von item und piece:

- Bei piece ist das Ergebnis vom gleichen Typ wie die dritte Eingabe; ein Teilstück eines Wortes ergibt wieder ein Wort, das einer Liste eine Liste.
- Das Ergebnis von item ist dagegen ein Teilelement, denn man gelangt dabei um eine Stufe tiefer in der Hierarchie. Bei einem Wort ergibt sich demnach ein Zeichen, bei einem Satz ein Wort, bei einem Abschnitt ein Satz.

9.5 Zusammensetzen von Wörtern und Listen

Sätze

Mit der Prozedur RUECKWAERTS wird ein Satz von hinten her zerlegt und daraus ein neuer Satz auf dem Bildschirm aufgebaut. Hiermit kann man aber nichts weiter anfangen, da es nur zeitweilig als Druckbild existiert.

Um einen neuen Satz, also eine Liste aus Wörtern, aufzubauen, wird das LOGO-Wort

se (Abkürzung von Sentence)

verwendet.

```
se "Guten "Tag
```

Damit wird eine Liste aus den zwei Wörtern erzeugt, hier der Satz Guten Tag. LOGO antwortet deswegen mit

```
[Guten Tag]
```

Die eckigen Klammern weisen dabei das Ergebnis als Liste aus.

se (se) erwartet normalerweise zwei Eingaben. Soll ein Satz aus mehr als zwei Wörtern gebildet werden, muß der Prozedurname zusammen mit den Eingaben in runde Klammern gesetzt werden.

```
(se "Heute "scheint "die "Sonne)
Antwort: [Heute scheint die Sonne]
```

Persönliche Anrede

```
to HERR :name
op se [Sehr geehrter Herr] :name
end
to FRAU :name
op se [Sehr geehrte Frau] :Name
end
```

Aufruf:

```
pr HERR "Logomeister
```

Antwort:

```
Sehr geehrter Herr Logomeister
```

Zur Verdeutlichung wiederholen wir den Aufruf mit show anstelle von pr

```
show FRAU "Schmidt
```

Antwort:

```
[Sehr geehrte Frau Schmidt]
```

show gibt auch die äußeren eckigen Klammern mit aus, das Ergebnis von FRAU wie auch von HERR ist also eine Liste. Es handelt sich aber um eine spezielle Listenform, nämlich einen Satz, wie aus der Übersetzung von se zu erwarten ist. Die erste Eingabe ist selbst ein Satz, die zweite ein Wort.

Geben Sie nun ein:

```
show HERR [Logomeister]
```

Auch hier wird ein Satz ausgegeben, obwohl die zweite Eingabe kein Wort, sondern eine Liste ist.

Wenn eine Eingabe zu se kein Wort, sondern eine Liste ist, dann entfernt LOGO sozusagen die äußeren eckigen Klammern, aber auch nur diese.

Wörter

Wörter können mit der LOGO-Prozedur word ähnlich wie Sätze mit se gebildet werden:

```
show word "Halle "luja
```

LOGO antwortet mit Halleluja. Daß es sich dabei tatsächlich um ein Wort und nicht um eine Liste handelt, wird daran sichtbar, daß auch show keine eckigen Klammern ausgibt. Weil dabei ein einziges Wort entstanden ist, gibt es nur ein Element und dementsprechend kein trennendes Leerzeichen.

Wie bei se müssen runde Klammern gesetzt werden, wenn mehr als zwei Eingaben zu einem Wort zusammengesetzt werden sollen.

```
(word "Donau "dampf "schiffahrt)
```

Zusammen mit der Zerlegung von Wörtern erlaubt das Zusammensetzen mit word umfangreiche Wortmanipulationen.

Sprachliche Anwendungen bieten sich an:

Ich gehe, du gehst, er geht, wir gehen, ihr geht, sie gehen.
Alles entsteht aus der Grundform 'gehen'. Natürlich erschweren die Unregelmäßigkeiten in den natürlichen Sprachen die Programmierung beliebiger Verben.

```
to MEHRZAHL :einzahl
op word :einzahl "n
end
```

Aufruf:

```
pr MEHRZAHL "Strasse
```

Antwort:

```
Strassen
```

Ganz so einfach ist die Mehrzahlbildung allerdings doch nicht, denn das Programm MEHRZAHL liefert nur für eine bestimmte Klasse von Substantiven die richtige Form.

Weil auch die Zahlen für LOGO Wörter darstellen, kann man mit word auch Zahlen aufbauen.

```
(word 3 ". 14159)
```

Antwort: 3.14159

Die Manipulation von Wörtern und Zahlen bietet in LOGO mehr Möglichkeiten, als man zunächst vermuten könnte. Die LOGO-Vokabeln selbst, sowohl die aus dem Grundwortschatz als auch die durch Prozeduren zusätzlich definierten, sind Wörter. Der Programmtext der Prozeduren ist eine Liste. Damit ergeben sich weitreichende Möglichkeiten der Programmanipulation durch Programme. Darauf wird in Lektion 17 noch eingegangen.

Durch die Kombination von word bzw. se mit piece können mehrere Teile von Wörtern und Listen zusammengestzt werden:

```
word piece 1 5 :wort piece 31 36 :wort
```

Antwort: `Donaumuetze`

oder:

```
(se piece 1 2 :satz [den ganzen Tag] piece 3 4 :satz)
```

Antwort: `[Heute scheint den ganzen Tag die Sonne]`

9.6 Zufallssätze

Mit Zufallszahlen können viele einfache Programme lustige Ergebnisse erzeugen. Ein Programm, das deutsche Sätze mit zufälligem Inhalt bei richtigem grammatikalischen Aufbau erzeugt, wird kompliziert, wenn Sie eine große Vielfalt möglicher Satzkonstruktionen zulassen.

Beschränken wir uns auf den simplen Aufbau von Sätzen der folgenden Art:

Anton geht weinend nach Hause

Schwierig wird es schon, wenn das Subjekt in der Mehrzahl vorkommt, oder andere Zeiten als die Gegenwart verwendet werden usw.

```
to PLAPPERN
pr (se SUBJEKT PRAEDIKAT ADVERB ORTS.ADVERB)
end
```

Das ist ein Rahmenprogramm, das die Satzstruktur festlegt: Subjekt, Prädikat, adverbiale Bestimmung der Art und Weise, adverbiale Bestimmung des Ortes.

Der Satz wird durch die Ausgaben der Prozeduren SUBJEKT, PRAEDIKAT, ADVERB, ORTS.ADVERB erzeugt.

```
to SUBJEKT
op "Anton
end
to PRAEDIKAT
op "geht
end
to ADVERB
op "weinend
end
```

```
to ORTS.ADVERB
op [nach Hause]
end
```

Das sind natürlich keine aufregenden Prozeduren, da sie nur die Aufgabe haben, ein Wort als Ergebnis auszugeben.

Auch solch einfache Prozeduren können sinnvoll sein. Damit kann die Zuweisung an globale Namen mit make vermieden werden. Bei Prozeduren wird auch nur der Prozedurname selbst hingeschrieben, bei einem Namen SUBJEKT, der etwa mit

```
make "Subjekt "Anton
```

mit dem Wort Anton verknüpft wird, müßte dagegen der Wert von Subjekt, also :Subjekt stehen.

Die Ausgabe von ORTS.ADVERB ist kein Wort, sondern eine Liste in der einfachen Form eines Satzes. Beim Aufruf von PLAPPERN streicht se die äußeren eckigen Klammern. Genauso könnte etwa das Subjekt aus mehreren Wörtern zusammengesetzt werden:

```
to SUBJEKT
op [Der arme Anton]
end
```

Jetzt sollte nach Eingabe von PLAPPERN auf dem Bildschirm erscheinen:

```
Der arme Anton geht weinend nach Hause
```

Als Alternative kann man aus mehreren Wörtern ein einziges bilden, wenn die Leerzeichen mit Hilfe von vorangesetzten Doppelkreuzsymbolen als normale Zeichen statt Trennzeichen einbezogen werden

```
"Der\ arme\ Anton\ statt [Der arme Anton]
```

Bis jetzt wird noch nicht geplappert, sondern nur ein einziger Satz erzeugt. Die Prozeduren für die Satzteile sollen einen Vorrat möglicher Antworten besitzen, aus dem dann eine zufällig herausgegriffen wird.

```
to SUBJEKT
op item 1+random 10 [Hans [Der arme Anton] [Die Bundes
regierung] [Ein Elefant] [Das Auto] Er Sie Es
[Der Dackel] Helmut]
end
```

Plappern sollte natürlich viele Sätze produzieren:

```
to PLAPPERN
pr (se SUBJEKT PRAEDIKAT ADVERB ORTS.ADVERB)
PLAPPERN
end
```

Jetzt könnte etwa auf dem Bildschirm erscheinen:

```
Die Bundesregierung geht weinend nach Hause
Ein Elefant geht weinend nach Hause
Sie geht weinend nach Hause
```

Genauso wie für das Subjekt müssen die Prozeduren für die anderen Satzteile mit einer Auswahl von Antworten versehen werden, damit wirklich Abwechslung in die Geschichte kommt.

```
to PRAEDIKAT
op item 1+random 8 [geht schreit pinkelt erklaert
schlaeft lacht rollt sitzt]
end
```

Die Vervollständigung der Programme sei nun Ihrer Phantasie überlassen.

9.7 Aufteilen von Listen und Wörtern

Mit item können einzelne Elemente unter Angabe der betreffenden Nummer aus Listen oder Wörtern herausgetrennt werden, während sich mit der LOGO-Vokabel piece die Möglichkeit bietet, ganze Teilstücke aus Wörtern und Listen herauszuschneiden.

Eine weitere Art der Zerlegung ist das Abtrennen eines Elementes am Anfang oder Ende einer Liste oder eines Wortes.

Zum Ausprobieren definieren wir uns noch eine Probeliste als Prozedur:

```
to PROBE
op [Logo ist eine tolle Sprache]
end
```

Geben Sie jetzt ein:

```
pr first "Auto
pr first Probe
pr last "Fahrrad
pr last PROBE

pr bf "Auto
pr bf Probe
pr bl "Fahrrad
pr bl PROBE
```

Antworten:

```
A
Logo
d
Sprache
uto
ist eine tolle Sprache
Fahrra
Logo ist eine tolle
```

Die Wirkung der eingeführten LOGO-Vokabeln läßt sich hieraus ablesen:

- first gibt das erste Element einer Liste bzw. das erste Zeichen eines Wortes aus,
- last entsprechend das letzte Element.
- bf (Abkürzung von Butfirst) liefert alles außer dem ersten Element der Liste, bzw. das restliche Wort bis auf das erste Zeichen,
- bl (Abgkürzung von Butlast) entsprechend alles außer dem letzten Element.
- Die LOGO-Vokabeln first, last, bf, bl haben eine Eingabe, die entweder eine Liste oder ein Wort sein muß.

Auch diese Kommandos sind umfangreicher, als es auf den ersten Blick erscheinen mag. Wenn Sie z.B. ausprobieren:

```
pr last bl PROBE
pr first last bl PROBE
pr bl bl bl PROBE
```

Die Wirkung dieser Aufteilung entfaltet sich besonders durch die Verkettung von Aufteilungen. Wenn von einer Liste, schrittweise vorne und hinten Teile abgetrennt werden, können beliebige Kombinationen zusammengestellt werden.

Die Verkettung bedeutet beim ersten Beispiel:

> Von der durch die Prozedur PROBE gebildeten Liste wird mit bl zunächst das letzte Element abgetrennt, und es verbleibt [Logo ist eine tolle]. last greift dann das letzte Element vom Rest heraus und gibt damit das zweitletzte Wort tolle aus.

Die Verkettung der Operationen kann insbesondere durch rekursiven Aufruf von Prozeduren zum Tragen kommen.

```
to KEIL :objekt
pr :objekt
KEIL bf :objekt
end
```

Aufruf mit KEIL "Melibokus liefert:

```
Melibokus
elibokus
libokus
ibokus
bokus
okus
kus
us
s
```

Und die Fehlermeldung

```
bf doesn't like an empty word as input in:
KEIL bf :objekt
```

Die Fehlermeldung bedeutet, daß bf (bf) auf ein nicht mehr zerlegbares Objekt gestoßen ist.

Was wird als Ergebnis von

```
bf "$
```

ausgegeben? Wenn das erste Element abgetrennt wird, bleibt nichts mehr übrig. LOGO akzeptiert dieses Kommando dennoch - die Antwort ist eine leere Zeile, also ein Ergebnis ohne Inhalt!

Die Anzahl der Elemente des Resultats ist auch tatsächlich Null, wie man durch

```
count bf "S
```
mit der Antwort: ø

feststellen kann.

Das entspricht der Zahl 0=1-1 oder noch besser dem Begriff der leeren Menge. Es gibt also tatsächlich leere Wörter und leere Listen.

```
to LEER.WORT
op "
end
to LEER.LISTE
op []
end
```

LEER.WORT erzeugt das gleiche wie bf "S. Das leere Wort hat genausowenig einen Inhalt wie die leere Liste, sie unterscheidet sich aber durch ihren Typ.

Um die Fehlermeldung in der Prozedur KEIL zu vermeiden, muß ein programmierter Abbruch der Rekursion vorgesehen sein. Dazu kann eine if-Zeile

```
to KEIL :objekt
if :objekt = " [stop]
pr :objekt
KEIL bf :objekt
end
```

eingebaut werden.

Diese Abfrage kann allerdings nur dann benutzt werden, wenn objekt ein Wort ist, weil eine leere Liste etwas anderes als ein leeres Wort ist.

```
KEIL PROBE
```

liefert auch die Fehlermeldung:

```
bf doesn't like [] as input in ...
```

Die Prozedur wurde nicht durch stop beendet, obwohl der Inhalt von objekt leer war. Soll die Prozedur auf Listen angewandt werden, so muß die if-Abfrage lauten:

```
if :objekt = [] [stop]
```

Es gibt aber eine Lösung, die in beiden Fällen funktioniert:

```
to KEIL :objekt
if emptyp :objekt [stop] pr :objekt
KEIL bf :objekt
end
```

Hier tritt zum erstenmal ein LOGO-Wort auf, das mit einem Fragezeichen endet. Diese Vokabeln sind Prüfwörter, mit deren Hilfe eine Bedingung überprüft werden kann.

emptyp :objekt stellt die Frage: Ist der Inhalt von objekt leer?

Die Antwort kann nur *wahr* oder *falsch* lauten.

```
emptyp "Melibokus
```

Antwort: FALSE (Ergebnis: Falsch)

Genauso bei emptyp PROBE.

Bei folgenden Eingaben:

```
emptyp "
emptyp []
```

TRUE (Ergebnis: Wahr)

Die mit dem Prüfwort emptyp formulierte Bedingung ermöglicht den Abbruch der Prozedur KEIL unabhängig davon, ob die Eingabe eine Liste oder ein Wort ist.

```
KEIL PROBE
```

Antwort:

```
Logo ist eine tolle Sprache
ist eine tolle Sprache
eine tolle Sprache
tolle Sprache
Sprache
```

Beim Ausprobieren der LOGO-Vokabeln first, last, bf, bl ist eine kleine Feinheit verloren gegangen, weil die Ausgabe mit pr automatisch die äußeren eckigen Klammern wegstreicht.

```
show first PROBE
show bf PROBE
```

Antwort:

```
Logo
[ist eine tolle Sprache]
```

Bei einem Satz, wie er hier von PROBE erzeugt wird, macht first aus der Liste ein Wort als Ergebnis, das Resultat von bf ist dagegen wieder eine Liste, was an den eckigen Klammern deutlich wird. In der Hierarchie Wort, Satz, Kapitel, Buch, Bibiliothek ... führen first und last auf die nächst niedrigere Ebene zurück, während bf, bl auf derselben Ebene bleiben.

```
first first [Ein Satz]
bf bf [Ein Satz]
```

Im ersten Beispiel ergibt sich der Buchstabe E als Ergebnis, im zweiten wird eine leere Liste [] erzeugt.

Wörter spiegeln

Um ein Wort auf dem Bildschirm gespiegelt auszugeben, reicht es aus, die Buchstaben von hinten her auszudrucken. Die vorderen Buchstaben müssen nach hinten bzw. die hinteren nach vorn befördert werden, damit das gespiegelte Wort verfügbar ist.

```
to ANS.ENDE :wort
op word (bf :wort) (first :wort)
end
```

Aufruf: `pr ANS.ENDE "hallo`

Antwort: `alloh`

Das Wort wäre durch diese Operation gespiegelt, wenn vorher der mit bf erzeugte Rest schon gespiegelt worden wäre. Hier kann nun die Rekursion zu einer eleganten Lösung verhelfen.

Bevor der Rest des Wortes mit dem nach hinten gebrachten Buchstaben wieder verbunden wird, muß der Prozeß selbst auf den Rest angewandt werden.

Es muß deswegen ersetzt werden

`bf :wort` durch `ANS.ENDE bf :wort`

Die durch den Selbstaufruf ausgelöste Wiederholung muß gesteuert abgebrochen werden. Ohne eine Zeile für den Abbruch wird eine Fehlermeldung auftreten, wenn das nach und nach zerlegte Wort keine Zerlegung mehr zuläßt, d.h. wenn ein leeres Wort entstanden ist.

```
to SPIEGEL :wort
if emptyp :wort [op :wort]
op word (SPIEGEL bf :wort) (first :wort)
end
```

Aufruf: `SPIEGEL "hallo`

Antwort: `ollah`

Der Transport von Buchstaben beginnt nun erst, wenn der letzte Buchstabe des eingegebenen Wortes, hier das O, bei der Zerlegung erreicht ist.

Sätze spiegeln

Um Sätze nach dem gleichen Prinzip zu spiegeln, muß im Programm für das Spiegeln von Wörtern nur word durch se ersetzt werden.

```
to SPIEGEL.SATZ :satz
if emptyp :satz [op :satz]
op se (SPIEGEL.SATZ bf :satz) (first :satz)
end
```

Aufruf: `pr SPIEGEL.SATZ [wie geht es]`
Antwort: `es geht wie`

Aufruf: `pr SPIEGEL.SATZ PROBE`
Antwort: `Sprache tolle eine ist Logo`

Zur Verschönerung wurde auch wort in satz umgeändert, was aber für das Programm keine Auswirkung hat.

9.8 Listen Verlängern

Zu first und last gibt es entsprechende Umkehrungen fput und lput (fput, lput stehen für Firstput bzw. Lastput).

Beispiele:

```
show fput "Guten [Tag]
show fput "Wie [spaet ist es ?]

show lput "Zuhause [es ist schoen]
show lput "wie? [Spaet ist es]
```

Antworten:

```
[Guten Tag]
[Wie spaet ist es ?]
[Es ist schoen Zuhause]
[Spaet ist es wie?]
```

Die Ausgabe mit show zeigt, daß mit fput und mit lput grundsätzlich Listen erzeugt werden. Insofern sind beide keine vollständigen Umkehrungen zu first und last, die ja auch auf Wörter angewandt werden können.

fput und lput haben genau zwei Eingaben. Die zweite muß eine Liste sein, die erste kann eine Liste oder ein Wort sein. Die Aufgabe besteht also darin, am Anfang oder am Ende einer bestehenden Liste ein Wort oder eine ganze Liste anzuhängen.

In manchen Fällen kann die gleiche Wirkung mit anderen LOGO-Vokabeln erzeugt werden.

```
show se "Guten "Tag
show se "Guten [Tag]
show se [Guten] [Tag]
```

Antworten jeweils: `[Guten Tag]`

Das Ergebnis ist jedesmal dasselbe wie bei fput "Guten [Tag]. Das zweite Beispiel stimmt mit der Anwendung von fput völlig überein. Die Eingabe des ersten Beispiels wäre für fput unzulässig, weil hier die zweite Eingabe eine Liste sein muß.

```
show se [Guten] [Tag]
show fput [Guten] [Tag]
```

Antworten: `[Guten Tag]`, bzw. `[[Guten] Tag]`

Hier wird ein weiterer Unterschied zu se sichtbar. Bei fput werden nicht wie bei se die äußeren eckigen Klammern der ersten Eingabe weggestrichen. Davor wird jetzt kein Wort, sondern eine Liste mit einem Element gesetzt.

lput arbeitet genauso wie fput, nur wird die erste Eingabe an das Ende der Liste, die als zweiter Eingabewert erwartet wird, gehängt.

Anwendungen von fput und lput

fput ist als Umkehrung von first gedacht.

```
pr fput (first PROBE) (bf PROBE)
```

Antwort: `Logo ist eine tolle Sprache`

Mit first und bf wird die von Probe erzeugte Liste in das erste Element, hier also das Wort LOGO, und den Rest zerlegt. Mit fput wird beides wieder zusammengesetzt, wobei sich die Ausgangsliste von PROBE ergibt.

Interessant wird es erst, wenn die Operationen, die auf den Anfang einer Liste wirken, mit denen kombiniert werden, die am Listenende aktiv werden. Also fput mit last, lput mit first und Kombinationen aus allen.

Wichtiger noch ist die Kombination von first/fput und last/lput, wenn vor dem Zusammensetzen mit fput oder lput einer der abgetrennten Teile verändert wird.

Elemente abändern

```
to DADA :liste
op fput (word first :liste "dada ) (bf :liste)
end

pr DADA [Wie geht es]
```

Antwort: `Wiedada geht es`

Da die runden Klammern nur zur optischen Zusammenfassung gesetzt sind, können sie entfallen. DADA hängt an das erste Wort des Satzes die Zeichenkette dada an.

Wird stattdessen das Wort selbst wieder angehängt, so entsteht ein Stottereffekt.

```
to STOTT :liste
op fput (word first :liste first :liste) bf :liste
end

pr STOTT [wie geht es]
```

Antwort: `wiewie geht es`

Um echt zu stottern, sollten alle Worte verdoppelt werden. Wenn Sie den Übergang von ANS.ENDE zur Prozedur SPIEGEL betrachten, dann können Sie erkennen, wie STOTT erweitert werden muß.

Der mit bf :liste erzeugte Rest der Liste muß selbst dem Stotterprozeß unterworfen werden; im Klartext muß also ersetzt werden

```
bf :liste durch STOTT bf :liste
```

Damit ist das Stottern für alle Wörter in Gang gesetzt. Jetzt muß aber noch ein Abbruch der Wiederholung vorgesehen werden, der genauso wie bei SPIEGEL aussehen kann.

```
to STOTTERN :liste
if emptyp :liste [op []]
op fput (word first :liste first :liste) !
(STOTTERN bf :liste)
end

pr STOTTERN [wie geht es]
```

Antwort: `wiewie gehtgeht eses`

9.9 Feststellen, worum es sich handelt: Prüfwörter

emptyp ist ein LOGO-Prüfwort, emptyp :a hat das Ergebnis TRUE oder FALSE, genauso wie eine Vergleichsoperation der Art :a > :b.

Die Prüfwörter enden im LOGO-Grundwortschatz mit dem Buchstaben p, wobei die Umkehrung allerdings nicht gilt. Beim Aufruf geschieht mit der Eingabe selbst nichts. Solche LOGO-Vokabeln werden benutzt, um Sachverhalte erst einmal auf ihre Eigenschaften hin zu überprüfen und Entscheidungen über die weitere Verwendung treffen zu können. emptyp läßt sich sehr häufig für den programmierten Abbruch von Wiederholungen beim rekursiven Aufruf einsetzen, was bei den letzten Beispiele der Fall war.

Andere Prüfwörter dienen dazu, den Typ eines Objekts festzustellen:

- numberp :a stellt fest, ob der Wert von A eine Zahl ist.
- wordp :a prüft, ob die Eingabe ein Wort ist.
- listp :a untersucht, ob es sich beim Wert von A um eine Liste handelt.

Nach den Regeln von LOGO gilt also eine Zahl auch als Wort.

```
numberp 5
wordp 5
listp 5
```

Antworten: TRUE, TRUE, FALSE.

Das Prüfwort listp sucht quasi nach den äußeren eckigen Klammern, wobei die Anzahl der Elemente keine Rolle spielt. Ein einziges Wort, das in eckige Klammern gesetzt wird, gilt nicht als solches, sondern als Liste.

```
listp [eine Liste]
listp [Wort]
listp "Wort
listp []
```

Antworten: TRUE, TRUE, FALSE, TRUE.

Ein sehr nützliches Prüfwort ist

```
memberp
```

Mit memberp kann festgestellt werden, ob ein Wort oder eine Liste Teil einer vorgegebenen Liste sind, oder ob ein Zeichen in einem Wort enthalten ist.

```
memberp "Schall [Worte sind Schall und Rauch]
memberp "schlecht [Heute ist das Wetter schoen]
memberp "a "abc
memberp [ist] [dies [ist] eine geschachtelte Liste]
memberp ". "5.16
```

Beim zweiten Beispiel ergibt sich FALSE, sonst überall TRUE.

memberp prüft, ob die erste Eingabe in der zweiten als Teil enthalten ist.

In der Hierarchie Zeichen, Wort, Kapitel, Buch ... muß die erste Eingabe auf einer niedrigeren Ebene erfolgen als der zweite Eingabewert.

DR LOGO ist nun bereit, Ihnen sogar zu verraten, an welcher Stelle das gesuchte Zeichen, bzw. das gesuchte Listenelement gefunden wurde. Dazu muß nur die Frage Wo? (where?) gestellt werden.

```
memberp "Schall [Worte sind Schall und Rauch] where
```

Nach der positiven Antwort TRUE wird die Position 3 ausgegeben. Bei einzelnen Zeichen funktioniert where und memberp genauso:

```
(pr memberp "b "abc where)
```

Antwort: TRUE 2

Ist das gesuchte Zeichen oder Listenelement nicht vorhanden, gibt where Null als Position aus:

```
(pr memberp "d "abc where)
```

Antwort: FALSE ∅

Mit piece können Teile aus Listen und Zeichenketten herausgeschnitten werden, wobei aber die Positionen von Anfang und Ende bekannt sein müssen. Mit where können dann Teillisten, bzw. Teilketten über das erste und letzte Element bestimmt werden.

```
to ANFANG :objekt :ziel
if memberp :objekt :ziel [op piece 1 where :ziel] !
[op :ziel]
end
```

Beispiele:

```
ANFANG "Schall [Worte sind Schall und Rauch]
Antwort: [Worte sind Schall]
ANFANG "x "Wort Antwort: Wort
```

Die Prozedur Anfang gibt den Anfang einer Liste bzw. eines Wortes bis zum Listenelement bzw. dem Zeichen OBJEKT aus. Wird das Objekt nicht gefunden, ist die ganze Liste bzw. Zeichenkette das Ergebnis. Soll in diesem Fall eine leere Liste oder ein leeres Wort zurückgegeben werden muß zusätzlich geprüft werden, ob es sich bei ziel um ein Wort oder eine Liste handelt, damit das Ergebnis vom richtigen Typ ist.

```
to ANFANG2 :objekt :ziel
if memberp :objekt :ziel [op piece 1 where :ziel]
if wordp [op "] [op []] end
```

Beispiele:

```
ANFANG2 "Fassbinder [Worte sind Schall und Rauch]
```
Antwort: []

```
ANFANG2 "x "Wort
```
Antwort:

Sicher haben Sie bemerkt, daß die LOGO-Vokabel if in beiden Fällen für zwei Alternativen benutzt wurden, während in Lektion 7 nur eine einzige auszuführende Liste zum Tragen kam. Die Bedeutung liegt auf der Hand:

- Bei if folgt nach dem zu überprüfenden logischen Ausdruck zunächst die Liste von LOGO-Anweisungen, die beim Resultat TRUE ausgeführt werden soll.
- Wird eine zweite Liste von Anweisungen angefügt, so kommt diese beim Ergebnis FALSE zum Zug. Wenn diese leer ist, kann sie weggelassen werden.

Lottoziehung: Ein weiteres Beispiel für memberp

Ergebnisse für das Zahlenlotto (6 aus 49) wurden schon im ersten Kapitel über Rechnen mit LOGO vorgestellt. Dabei konnte es vorkommen, daß bereits gewürfelte Zahlen sich wiederholen, bevor der Lottotip komplett war. Um Wiederholungen zu vermeiden, müssen auf jeden Fall die Ergebnisse des Würfelns zum Vergleich aufgehoben werden. Mit memberp läßt sich nun leicht erfragen, ob eine Wiederholung aufgetreten ist, was aber weitere Zufallszahlen notwendig macht.

```
to LOTTO :tip
local "zahl
if count :tip = 7 [op :tip]
make "zahl (1+random 49)
if memberp :zahl :tip [op LOTTO :tip]
op LOTTO lput :zahl :tip
end
```

Die Prozedur LOTTO ruft sich an zwei verschiedenen Stellen selbst auf, und besitzt, worauf Sie achten sollten, einen Ausgabewert. Der Selbstaufruf muß deshalb mit op LOTTO erfolgen.

Wenn das Prüfwort memberp zur Antwort Ja führt, d.h. die gezogene Zahl im Ziehungsgerät nicht mehr zur Verfügung steht, dann wird per Selbstaufruf nur eine Wiederholung des Ziehungsvorgangs ausgelöst. Falls die Zahl noch nicht gezogen wurde, wird die Liste tip übergeben, die bereits gezogene Gewinnzahlen mit LPUT ergänzt. Die ermittelte Zahl muß mit Hilfe von make notiert werden, weil bei der Funktion random eine Wiederholung zu einem anderen Resultat führen würde.

Etwas komfortabler wird der Aufruf, wenn ein Rahmenprogramm benutzt wird.

```
to ZIEHUNG
make "tip LOTTO []
(pr [Ergebnis:] bl :tip [-Zusatzzahl:] last :tip)
end
```

Beispiel:

```
ZIEHUNG
```

Ergebnis: 2 27 44 13 49 46 -Zusatzzahl: 6

Nach einer Lottoziehung werden die Lottozahlen in sortierter Reihenfolge ausgegeben. Da DR LOGO auf den Schneidercomputern über keinen eigenen Sortierbefehl verfügt, kann das Sortieren hier am einfachsten dadurch geschehen, daß alle Kugeln daraufhin durchmustert werden, ob sie gezogen worden sind oder nicht. Die richtige Reihenfolge entsteht dann beim Durchmustern.

```
to SORT :i :l :e;Sortieren von Zahlen aus dem Bereich 1-i
repeat :i [if memberp :i :l [make "e se :i :e] make "i :i-1] op :e
end
to ZIEHUNG
make "tip LOTTO []
(pr [Ergebnis: ] SORT 49 bl :tip [] [- Zusatzzahl:] last :tip)
end
```

9.10 Vergleichsoperationen

Um die Gleichheit zweier Objekte festzustellen, kann das Prüfwort equalp benutzt werden.

```
make "liste [Worte sind Schall und Rauch]
make "satz [Heute ist Mittwoch]
make "kopie :liste

equalp :liste :satz        Antwort: FALSE
equalp :liste :kopie       Antwort: TRUE
equalp 10 20               Antwort: FALSE
equalp "wort "wort         Antwort: TRUE
```

equalp kann auf Liste, Wörter und Zahlen angewandt werden. Das Prüfwort equalp entspricht der sonst verwendeten Syntax

für Prüfwörter, kann aber auch einfach durch ein Gleichheitszeichen ersetzt werden:

```
= :liste :satz          Antwort: FALSE
= "wort "wort           Antwort: TRUE
```

Das Gleichheitszeichen hat auch den Vorteil, daß es neben der gerade benutzten Präfixnotation auch in der Infixnotation benutzt werden darf:

```
:liste = :satz          Antwort: FALSE
"wort = :satz"          Antwort: FALSE
```

Ungleichheit kann durch Verneinung der Gleichheit überprüft werden. Dafür gibt es wie in fast allen Programmiersprachen die Vokabel not:

```
not equalp :liste :satz   Antwort: TRUE
not = "wort "wort         Antwort: FALSE
```

Beachten Sie, daß die in anderen LOGO-Versionen bzw. anderen Programmiersprachen gegebene Möglichkeit, die Ungleichheit mit der Kombination <> oder >< zu prüfen, hier nicht erlaubt ist.
Die Vergleichsoperationen "größer als" und "kleiner als" lassen sich sowohl auf Zahlen wie auch auf Wörter anwenden.

```
> "Logo "Basic          Antwort: TRUE
"Helmut < "Johannes     Antwort: TRUE
10 > 20                 Antwort: FALSE
< 1E10 1E15             Antwort: TRUE
```

Der Größenvergleich ist bei Wörtern im Sinne der lexikalischen Ordnung zu verstehen. Ein Wort ist dann größer als ein zweites, wenn es in einem Lexikon nach dem Alphabet weiter hinten aufgeführt würde.

9.11 Listen durcheinanderschütteln

Wie man den Zufall mit Hilfe von random ins Spiel bringt, ist Ihnen nun schon bekannt. Sie können aber auch ganze Wörter bzw. ganze Listen auf einmal in zufälliger Weise durcheinanderbringen. Dazu ist die LOGO-Vokabel shuffle vorgsehen.

```
shuffle :liste
```

Probieren Sie mit Control+Y aus, was bei wiederholter Eingabe der Zeile geschieht! Es werden jeweils willkürliche Anordnungen der Wörter ausgegeben. shuffle kann nur auf Listen angewandt werden, diese dürfen aber auch verschachtelt sein.

```
make "Schachtel !
[dies [ist eine [sehr]] verschachtelte [Liste]]
shuffle :Schacht l
```

Antwort: `[verschachtelte [ist eine [sehr]] dies [Liste]]`

Mit shuffle kann auch eine Lottoziehung dargestellt werden. Dabei werden ja in ähnlicher Weise 49 Kugeln zufällig durcheinander g bracht und davon sieben ausgewählt.

```
to KUGELN :n :trommel
if :n=0 [op :trommel]
op KUGELN :n-1 fput :n :trommel
end
```

Aufruf: `KUGELN 49 []`

Das Ergebnis von Kugeln ist eine Liste der Zahlen 1 bis 49. Beim Aufruf wird mit der leeren Liste begonnen, in der dritten Zeile wird mit fput eine Zahl von links her angefügt. Die Wiederholung geschieht rekursiv, wobei die hinzugefügte Zahl N um eins verringert wird, bis sie beim Wert Null zum Abbruch der Rekursion führt. Wenn sie durch

```
make "trommel KUGELN 49 []
```

eine Trommel mit Kugeln bereit stellen, kann der Ziehungsprozeß an Hand von

```
piece 1 7 shuffle :trommel
```

dargestellt werden.

9.12 Eigenschaftslisten

Ein Objekt ist festgelegt durch seinen Namen und seine Eigenschaften. Bei Eigenschaften müssen wir zwischen Begriff, z.B. die Farbe, und zugehörigem Wert, z.B. rot, unterscheiden. Eine Eigenschaft besteht demnach aus einem Paar, der eigentlichen Eigenschaft und deren Wert. Die Eigenschaftsbezeichnung ist ein Wort, der zugehörige Wert kann ein Wort oder auch eine Liste sein.

Eigenschaftslisten sind Listen, die aus solchen Paaren gebildet werden.

Um die Eigenschaften zu erfahren, die sich hinter einem Namen verbergen, dient die LOGO-Vokabel plist.

```
make "Zahl 10
plist "Zahl          Antwort: [.APV 10]
```

Das erste Element eines Paares ist die Bezeichnung der Eigenschaft, hier .APV (für associated property value). Die mit einem Punkt beginnenden Eigenschaften sind LOGO-Systemeigenschaften, von denen Sie sechs in Ihrem mitgelieferten Handbuch aufgelistet finden. Beachten Sie, daß diese Eigenschaften mit Großbuchstaben geschrieben werden müssen.

Die Systemeigenschaft .APV speziell bedeutet den Wert einer globalen Variablen.

Sicher sind noch Prozeduren im Arbeitsspeicher verfügbar. Fragen Sie mit plist nach den Eigenschaften von Prozeduren.

```
plist "Prozedurname
```

Sie werden dann die Eigenschaft .DEF vorfinden, deren zugehöriger Wert ist der Programmtext in der Form einer Liste. Daneben finden sich möglicherweise noch Eigenschaften, die die Formatierung des Programmtextes sowie Kommentare kennzeichnen, worauf hier nicht eingegangen werden soll.

Probieren Sie auch aus:

```
plist "fd          Antwort: [.PRM 7020]
```

Die Eigenschaft .PRM zeigt die Zugehörigkeit zum Grundwortschatz von DR LOGO an. Mit dem Zahlenwert (hier die Antwort beim Joyce) kann man nichts anfangen, er ist intern für den Interpreter von Bedeutung.

Den Wert einer Eigenschaft erfährt man mit Hilfe von gprop.

```
gprop "Zahl ".APV              Antwort: 10
gprop "Prozedurname ".DEF      Antwort: Programmtext
gprop "Zahl "Farbe             Antwort: []
```

Wenn die gefragte Eigenschaft nicht vorhanden ist, so wird die leere Liste zurückgegeben.

Einem Objekt wird mit Hilfe der LOGO-Vokabel pprop eine Eigenschaft zugewiesen.

```
pprop "Auto "Farbe "rot
pprop "Auto "Preis "20000DM
pprop "Auto "Marke "Aucedes
plist "Auto

Antwort: [Marke Aucedes Preis 20000DM Farbe rot]

gprop "Auto "Farbe  Antwort: rot
```

Mit der LOGO-Vokabel remprop können Eigenschaften wieder entfernt werden. Wird eine Eigenschaft entfernt, ändert sich nicht ihr Wert, vielmehr erlischt die ganze Eigenschaft.

```
remprop "Auto "Preis
```

Damit hat im genannten Beispiel das Auto keinen Preis mehr. Eine Veränderung des Wertes geschieht mit pprop. Man kann auch umgekehrt mit Hilfe des Kommandos glist feststellen, welche Namen eine bestimmte Eigenschaft besitzen.

```
pprop "Himmel "Farbe "blau
glist "Farbe Antwort: [Auto Himmel]
```

Schließlich gibt es noch die LOGO-Vokabel pps, durch die alle Namen mit den zugehörigen Eigenschaften ohne Berücksichtigung der Systemeigenschaften ausgegeben werden.

```
pps                    Antwort: Himmels's Farbe is blau
                       Autos's Farbe is rot ...
```

9.13 Ersetzen mit Eigenschaftslisten

Stellen Sie sich vor, es sollen Inserate für Gebrauchtwagen gedruckt werden. Der Anzeigentext wird eine Standardform besitzen, sodaß immer nur spezielle Eigenschaften eines Autos wie Marke, Farbe, Alter und Preis eingesetzt werden müssen.

```
make "Inserat [[Günstig zu verkaufen:] [MARKE TYP Baujahr
JAHR] [Farbe: FARBE] [Preis: PREIS DM] [Auto Metzger
Rüsselstadt]]
```

Bei der Eingabe dieser langen Zeile für den Wert von Inserat kann natürlich leicht ein Tippfehler auftreten. Wird der Fehler sofort bemerkt, kann die Eingabezeile mit CONTROL+Y, bzw. beim Joyce auch mit der COPY-Taste, wieder erzeugt und anschließend korrigiert werden. Ist das nicht mehr möglich, so

kann die Variable Inserat auch mit dem Editor bearbeitet werden. Nach Eingabe von

```
ed "Inserat
```

erscheint die Zeile im Editorbetrieb und kann wie ein Programmtext bearbeitet werden, wobei der Editor dann auch mit CONTROL+C verlassen werden muß.

```
pprop "Auto "MARKE "Aucedes
pprop "Auto "TYP "100GL
pprop "Auto "FARBE "rot
pprop "Auto "PREIS 9000
pprop "Auto "JAHR 1983
```

Damit sind die benötigten Eigenschaften eines Fahrzeugs beschrieben, die nun anstelle der Platzhalterwörter in das Inserat eingesetzt werden müssen. Das Programm ERSETZE erledigt das für einen einzigen Satz, wenn die benötigten Eigenschaften bereits in der Variablen eliste zur Verfügung stehen.

```
make "eliste plist "Auto

to ERSETZE :satz :neu
if emptyp :satz [op :neu]
local "wort make "wort first :satz
if memberp :wort :eliste [op ERSETZE bf :satz se :neu item
1+where :eliste] [op ERSETZE bf :satz se :neu :wort]
end
```

Weil ERSETZE zunächst nur einzelne Sätze verarbeitet, müssen zum Testen Sätze mit item aus dem Inserattext herausgenommen werden.

```
ERSETZE item 2 :Inserat []
Antwort: [Aucedes 100GL Baujahr 1983]

ERSETZE item 1 :Inserat []
Antwort: [Günstig zu verkaufen:]
```

Die Prozedur ERSETZE hat zwei Eingaben: den zu bearbeitenden Satz und den daraus entstehenden neuen Satz, der beim Aufruf zunächst als leere Liste vorgesehen wird. Mit dem Prüfwort memberp wird festgestellt, ob das gerade untersuchte Wort in der Eigenschaftsliste von Auto enthalten ist. Wenn dies der Fall ist, dann findet sich der Wert der Eigenschaft als nächstes Element in eliste, wobei die Position mit Hilfe von where ermittelt werden kann.

Der Satz wird nun durch Selbstaufruf Wort für Wort analysiert; dabei wird beim rekursiven Aufruf jeweils die erste Eingabe um ein Wort verkürzt, an den neu entstehenden Satz wird mit se jeweils ein Wort angehängt. Weil ERSETZE ein Ergebnis zurückgibt, muß auch der Selbstaufruf mit op eingeleitet werden.

```
to EABSATZ :absatz :objekt
(local "eliste "i)
make "eliste plist :objekt make "1
repeat count :absatz [pr ERSETZE item :i :absatz [] make "i
:i+1]
end
```

Aufruf: `EABSATZ :Inserat "Auto`

In EABSATZ wird ein ganzer Absatz, also eine Liste von Sätzen, mit ERSETZE bearbeitet. Die Wiederholung geschieht hier mit repeat, wobei eine Zählvariable i benutzt wird, um jeweils den i.ten Satz abzutrennen. Die Wiederholung könnte genausogut rekursiv formuliert werden.

Natürlich ist es hier noch sehr umständlich, die Eigenschaften des Fahrzeugs nacheinander mit pprop eingeben zu müssen. Dafür kann das folgende Programm benutzt werden, das schon die LOGO-Vokabel rq für eine Eingabe mit der Tastatur benutzt, die im folgenden Kapitel ausführlicher erläutert wird.

```
to DAUTO
type "Marke: pprop "Auto "MARKE rq
type "Typ: pprop "Auto "TYP rq
type "Baujahr: pprop "Auto "JAHR rq
type "Farbe: pprop "Auto "FARBE rq
type "Preis: pprop "Auto "PREIS rq
end
```

Nach Eingabe von DAUTO werden die benötigten Eigenschaften auf dem Bildschirm angefordert, wobei die Antwort jeweils mit Return abgeschlossen wird. Die Eigenschaften dürfen übrigens auch Leerzeichen enthalten, als Farbe beispielsweise "blau metallic".

Übungen

1) Vervollständigen Sie die Programme zur Erzeugung von Zufallssätzen. Schreiben Sie Prozeduren ADVERB und ORTS.ADVERB, die auch für diese Satzteile zufällig variierende Ausgaben haben.

2) Schreiben Sie eine Prozedur SPITZE, die bei jedem Schritt sowohl von vorn als auch von hinten einen Buchstaben beim eingegebenen Wort wegnimmt.

3) Eine Prozedur soll feststellen, ob die Eingabe ein Palindrom ist, d.h. ein Wort, das vorwärts wie rückwärts gelesen den gleichen Sinn ergibt (z.B. OTTO). Als Ausgabe soll entweder das Wort "TRUE oder das Wort "FALSE zurückgegeben werden.

4) Formulieren Sie eine LOGO-Tätigkeit, die die Buchstaben des eingegebenen Wortes willkürlich durcheinander bringt.

5) Wir schreiben Dezimalbrüche gewöhnlich als Kommazahlen. Das Programm soll die Kommazahl in die angelsächsische Schreibweise mit dem Dezimalpunkt überführen. (Mit dem Komma muß die Zahl als Eingabewert als Wort mit einleitenden Anführungszeichen geschrieben

werden, z.B. "1,57; im Vorgriff auf die nächste Lektion können Sie als Eingabe auch ausprobieren: first rq!)

6)
```
to WANDLER :n :b;wandelt n vom Dezimalsystem ins System
mit der Basis B um
if :n = 0 [op "]
op word (WANDLER quotient :n :b :b) (remainder :n :b)
end
```

Untersuchen Sie das Programm! Wenn die Basis >10 ist, verwendet man für die folgenden Ziffern Buchstaben (10=a, 11=b usw.). Verallgemeinern Sie WANDLER für solche Fälle, speziell für die Basis 16.

7) Vielleicht sind Sie ein abergläubischer Lottospieler? Verändern Sie die Prozeduren zur Lottoziehung so, daß 13 nicht als Gewinnzahl auftreten kann.

8) shuffle kann nicht auf Wörter und damit auch nicht auf Zahlen angewandt werden. Um ein Wort bzw. eine Zahl mit Hilfe von shuffle zu schütteln, muß das Wort erst in Zeichen zerlegt und zu einer Liste zusammengesetzt werden. Schreiben Sie jeweils eine Prozedur, die ein Wort in eine aus Zeichen bestehende Liste überführt und umgekehrt eine solche Liste wieder zu einem Wort zusammensetzt.

Lektion 10: Eingabe und Ausgabe

Neue Sprachelemente:
rq, rl, rc, keyp, ascii, char, go, label, wait

Programme:
BEGRUESSUNG, KREIS, REAKTION

10.1 Eingabelisten

Wenn Sie Ihren Computer eingeschaltet haben, verbraucht er Strom. Zwar nicht soviel wie ein Heizgerät, Sie können aber mit Recht erwarten, daß dabei auch etwas geschieht. Wenn das LOGO-System nicht gerade damit beschäftigt ist, Ihre Aufträge auszuführen, wartet es auf eine Eingabe, die normalerweise über die Rechnertastatur erfolgt. Dazu wird ein Fragezeichen auf dem Bildschirm erzeugt und der Cursor ist zu sehen. Für den Rechner ist das ebenso eine Tätigkeit wie die Erledigung von Rechenaufgaben.

In diesen Zustand können Sie den Rechner aber auch innerhalb eines Programms mit Hilfe der LOGO-Vokabeln rl und rq versetzen. Betrachten wir zur Demonstration ein einfaches Programm:

```
to BEGRUESSUNG
pr [Wie heissen Sie?]
make "Name rl
(pr [Habe ich richtig verstanden, Sie heissen]!
:Name "?)
if rq = "nein [BEGRUESSUNG] [(pr [O.K. Guten Tag,]!
:Name)]
end
```

Nach dem Aufruf von BEGRUESSUNG wird die Textzeile

```
Wie heissen Sie?
```

ausgegeben, in der folgenden Bildschirmzeile erscheint dann der Cursor. Das Fragezeichen, das vor den Cursor gesetzt wird, wenn LOGO auf neue Aufträge wartet, das sogenannte 'Prompten', erscheint nicht. Das System wartet jetzt auf eine Eingabe, befindet sich dabei aber in der Prozedur BEGRUESSUNG. Davon können Sie sich überzeugen, indem Sie die Ausführung mit STOP(ESC) unterbrechen. LOGO antwortet mit:

```
Stopped! in BEGRUESSUNG: ...
```

Nach Eingabe einer Folge von Zeichen wird der Dialog fortgesetzt.

> rl versetzt das LOGO-System in einen Wartezustand. Erst nach dem Drücken der Returntaste wird im Programm fortgefahren.

Auch im Direktbetrieb läßt sich rl verwenden mit einer Antwort vom Typ

```
[...]
```

Die eckigen Klammern zeigen an, daß rl als Ergebnis eine Liste ausgibt. Das läßt sich auch leicht mit dem Prüfwort listp bestätigen:

```
listp rl
```

Antwort nach beliebiger Eingabe: TRUE

Das Ergebnis von rl kann auch mit LOGO-Vokabeln bearbeitet werden, die auf Listen wirken, z.B.

```
count rl
first bf rl
```

Damit ist das weite Feld der Programme mit Eingabedialog eröffnet.

In der vorletzten Zeile von BEGRUESSUNG taucht die LOGO-Vokabel rq auf. Sie unterscheidet sich von rl nur insofern, als ein Wort und keine Liste erwartet wird.

```
wordp rq
```

Nach beliebiger Eingabe folgt die Antwort TRUE, und zwar auch dann, wenn Sie mehrere Wörter durch Leerzeichen getrennt eingeben oder einen Satz mit mehreren Wörtern in eckige Klammern einschließen. Das sonst als Trennzeichen interpretierte Leerzeichen, wie auch die eckigen Klammern, wird bei rq als Zeichen innerhalb eines Wortes gedeutet. Davon können Sie sich durch Abzählen mit COUNT überzeugen.

> rq gibt die eingetippten Zeichen als Wort zurück. Mit dem Cursor zeigt der Rechner die Eingabebereitschaft an. Die Eingabe gilt erst mit Betätigung der Returntaste als abgeschlossen.

10.2 Sprünge nach vorne und hinten

Im Programm BEGRUESSUNG ist bereits eine gewisse Kontrolle der Tastatureingabe vorgesehen. Der Benutzer muß seine Eingabe, hier den Namen, eigens bestätigen. Falls keine Bestätigung, sondern die Antwort Nein erfolgt, wird die Prozedur BEGRUESSUNG rekursiv aufgerufen. Ein vorausschauender Programmierer stellt sich auf mögliche Eingabefehler ein, indem im Programm selbst eine durch rl oder rq erzeugte Eingabe vor jeder weiteren Verarbeitung zunächst einmal auf ihre Brauchbarkeit untersucht wird.

Als Beispiel soll ein Programm zur Kreisberechnung betrachtet werden.

```
to KREIS
pr [KREISBERECHNUNG]
pr [Wie gross soll der Radius sein?]
make "r rq
(pr [Umfang =] 6.283185*:r)
(pr [Flaeche=] 3.14159*:r*:r)
end
```

Die Eingabe des Radius wird in der vierten Zeile durch rq veranlaßt. Nun kann es ja beim Eintippen passieren, daß versehentlich statt einer Zahl etwas anderes eingegeben wird. Bei dem anschließenden Versuch, das Rechenbeispiel auszuführen, wird demnach eine Fehlermeldung erfolgen. Es ist also sinnvollerweise zu überprüfen, ob wirklich ein Zahlenwert vorliegt. Das entsprechende Prüfwort lautet numberp.

Falls keine Zahl eingegeben wurde, soll nicht mit der Kreisberechnung fortgefahren, sondern eine neue Eingabe verlangt werden. Die zugehörige Bedingung würde also eigentlich lauten 'Falls keine Zahl, dann ...'. Die Negation eines Prüfwortes läßt sich erreichen mit dem LOGO-Wort not:

```
if not numberp [...]
```

Was soll anstelle der Pünktchen treten? Das Programm soll wieder zurückkehren zur Zeile

```
pr [Wie gross soll der Radius sein?]
```

Bisher kennen Sie nur die Möglichkeiten, andere Tätigkeiten aufzurufen, eine Prozedur rekursiv aufzurufen oder eine Prozedur zu benden. Es gibt aber auch die Möglichkeit, innerhalb einer Prozedur zu Programmzeilen zurückzukehren bzw. an später folgende Zeilen zu gelangen und dabei bestimmte Programmzeilen zu überspringen. Solche Sprünge innerhalb einer Prozedur werden durch go veranlaßt.

```
to KREIS
pr [KREISBERECHNUNG]
label "eingabe pr [Wie gross soll der Radius sein ?]
make "r rq
if not numberp :r [go "eingabe]
(pr [Umfang =] 6.283185*:r)
(pr [Flaeche=] 3.14159*:r*:r)
end
```

Das Sprungziel muß markiert werden, damit ein Sprung überhaupt ausgeführt werden kann. Am Beginn der Zielprogrammzeile wird dazu das LOGO-Wort label und anschließend ein markierendes Wort angegeben. Diese Markierung gewinnt erst bei der Bearbeitung von go Bedeutung.

Sprunganweisungen gibt es in fast allen Programmiersprachen, sie sind auf der Ebene der sogenannten Maschinensprache sogar sehr häufig. Für LOGO ist die Verwendung von go nicht gerade typisch, weil ja in LOGO das viel mächtigere Instrument des Prozeduraufrufs einschließlich des rekursiven Aufrufs zur Verfügung steht. Sprünge können in LOGO aber nicht beliebig, sondern nur innerhalb einer Prozedur selbst erfolgen. Weil LOGO-Prozeduren typischerweise nur relativ wenige Zeilen enthalten, kommen auch nur Sprünge über wenige Zeilen hinweg in Frage.

Die Programmstruktur wird deswegen auch mit der go-Anweisung kaum in ihrer Überschaubarkeit beinträchtigt. Ganz unsinnig wäre es aber, längere Prozeduren mit einer ganzen Reihe von Sprungzielen und go-Anweisungen zu formulieren. Eine konsequent problemorientierte, mächtige Sprache wie LOGO ist nicht dafür konzipiert, primitive, maschinennahe Programmstrukturen zu realisieren.

Reaktionstest

Es soll ein Programm entwickelt werden, daß die Reaktionszeit testet. Ein Testkandidat soll aufgefordert werden, eine bestimmte Taste zu drücken, wenn das entsprechende Zeichen auf dem Dialogfenster erscheint. Mit einem solchen Programm kann man auch die Tastatur kennenlernen.

Zunächst schreiben wir kleine Hilfsprozeduren für die Überschrift sowie eine weitere zum zeitweiligen Anhalten.

```
to UEBERSCHRIFT
ct pr "
pr "REAKTIONSTEST pr "
pr [Druecken Sie so schnell wie moeglich] pr "
pr [die Taste, die gleich auf dem Bildschirm erscheint!]
pr "
end

to HALTEN
repeat 150 []
end
```

Mit HALTEN wird eine programmgesteuerte längere Pause veranlaßt, die ohne äußere Eingriffe beendet wird.

Die Anzeige des einzugebenden Zeichens soll natürlich in einem nicht genau berechenbaren Zeitintervall erfolgen. Hierzu können Zufallszahlen dienen, mit deren Hilfe eine Warteschleife so oft wie per Zufall bestimmt durchlaufen wird.

```
repeat random 300 []
```

DR LOGO auf den Schneider-CPC-Computern, also nicht auf dem Joyce, besitzt sogar eine eigene Vokabel für derartige Warteschleifen.

```
wait 6000
```

Damit wird der Programmablauf ca. 100 Sekunden lang angehalten. Die repeat-Zeile in HALTEN läßt sich mit Hilfe von wait ersetzen durch

```
wait random 500
```

Dabei kann der Eingabewert für random natürlich auch anders gewählt werden.

Um feststellen zu können, ob eine Taste gedrückt wurde, ist in LOGO das Prüfwort

```
keyp
```

vorgesehen. Nach einem Tastendruck liefert keyp den Wert TRUE, sonst FALSE.

Die Kontrolle der Tastendrücke soll natürlich die Reaktionsgeschwindigkeit bei einem Reaktionstest messen. Dies geschieht in der Prozedur TASTE

```
to TASTE :t
local "I make "I 0
label "Anfang
if keyp [make "ta rc] [make "I :I+1 go "Anfang]
if ascii :ta = :t [op :I] [op -1]
end

to WERTUNG :I
if :I < 0 [op [Ein Fehler!]]
if :I < 6 [op [Hervorragend!]]
if :I < 12 [op [Sehr Gut!]]
if :I > 18 [op [Langsam!]] [op [Gut!]]
end
```

In TASTE wird eine Wiederholung der Tastaturabfrage mit GO zur Sprungmarke ANFANG befohlen. Die Anzahl der Wiederholungen wird in der lokalen Variablen I gezählt und als Ergebnis zurückgegeben. Das Ergebnis von Taste ist ein Maß für die

Geschwindigkeit, mit dessen Wert dann in der Prozedur WERTUNG ein Bewertung vorgenommen werden soll.

Vor der näheren Erläuterung von TASTE soll erst noch das Hauptprogramm REAKTION vorgestellt werden.

```
to REAKTION
UEBERSCHRIFT
repeat random 300 []
if keyp [(pr "Disqualifiziert rc]
[wurde bereits gedrueckt!]) HALTEN REAKTION]
make "t 97+random 26 pr char :t
pr " (type [Ihre Reaktion war] WERTUNG TASTE :t)
pr " HALTEN REAKTION
end
```

In der vierten Zeile von REAKTION wird überprüft, ob vorzeitig eine Taste gedrückt wurde, obwohl noch keine Aufforderung ergangen ist, was wiederum zur Disqualifizierung Anlaß gibt. Mit der LOGO-Vokabel rc wird das entsprechende Zeichen der vorzeitig gedrückten Taste gelesen und als Ergebnis ausgegeben. Eine positive Antwort des Prüfwortes keyp bedeutet nicht, daß gerade eine Taste gedrückt ist, sondern weist vielmehr auf eine von LOGO bisher noch nicht verarbeitete Tastatureingabe hin. Erst das Einlesen mit rc löscht diesen Hinweis, weshalb auch bei vorzeitigem Tastendruck eingelesen werden muß. HALTEN verursacht eine künstliche Pause, mit dem Selbstaufruf REAKTION wird alles neu begonnen.

Der einzutippende Buchstabe wird im Hauptprogramm REAKTION in der fünften Zeile ausgewählt

```
make "t 97+random 26
```

Unter dem Namen t wird so eine ganze Zahl zwischen 97 und 122 erwürfelt, der Wert von t wird dabei an die Prozedur TASTE übergeben. Der Zusammenhang zwischen diesem Zahlenwert und einem Buchstaben wird hergestellt mit:

```
pr char :t
```

Hier kommt mit char eine neue LOGO-Funktion ins Spiel, die am besten gleich im Zusammenhang mit der in der letzten Zeile von TASTE vorkommenden Funktion

```
ascii rc
```

betrachtet wird.

- char weist einer Zahl ein einzelnes Zeichen zu.
- ascii ordnet umgekehrt einem einzelnen Zeichen eine Zahl zu.

Beispiele:

```
char 97          Antwort: a
char 122         Antwort: z
char 65          Antwort: A
char 90          Antwort: Z

ascii "B         Antwort: 66
ascii "q         Antwort: 113

ascii char 97    Antwort: 97
char ascii "z    Antwort: z
```

Die Zeichen sind für LOGO Wörter. Mit der Zuordnung von Zahlen werden Zeichen codiert; hierbei beziehen sich char und ascii auf den Standardcode ASCII (American Standard Code for Information Interchange). Die letzten zwei Aufrufe demonstrieren, daß sich char und ascii gegenseitig aufheben.

Beachten Sie, daß Groß- und Kleinschreibung beim ASCII-Code zu unterschiedlichen Werten führen. Im Fall des betrachteten Programms REAKTION wird man am einfachsten nur Kleinbuchstaben verwenden. Ein Problem stellt in diesem Zusammenhang immer der deutsche Zeichensatz dar, der zunächst im ASCII-Code natürlich nicht berücksichtigt ist. Die CPC-Computer haben standardmäßig eine ASCII-Tastatur und arbeiten ohne deutsche Umlaute usw. Der Schneider-Joyce hat

zwar eine deutsche Tastatur, für DR LOGO wird aber besser der amerikanische Zeichensatz eingestellt.

Neben den Buchstabentasten können natürlich auch die meisten anderen Tasten für den Reaktionstest verwendet werden. Den zugehörigen ASCII-Code wird am einfachsten mit der Zeile

```
ascii rc
```

bestimmt.

Die Wirkung von TASTE läßt sich leicht an Beispielen erproben:

```
TASTE 97
```

Das Programm Taste wartet auf einen Tastendruck, ohne daß dabei ein Cursor auf dem Bildschirm erscheint. Das Warten ist in der ersten Zeile von TASTE programmiert. Wenn das Ergebnis von keyp FALSE ist, aber keine Taste gedrückt ist, dann wird der Wert der Zählgröße I um eins erhöht. Tippen Sie nun die zum ascii-Code 97 gehörende Buchstabentaste a, so erscheint z.B.

```
Antwort: 85
```

Die Zahl gibt an, wie häufig die erste Zeile von TASTE durchlaufen wurde. Ist nämlich beim Prüfwort keyp in der ersten Zeile von TASTE tatsächlich eine Taste gedrückt, dann wird mit rc das gedrückte Zeichen festgestellt und mit dem Namen TA verknüpft. Die Wiederholungsschleife der dritten Zeile wird abgebrochen, und in der vierten Zeile wird nun geprüft, ob der ASCII-Code der gedrückten Taste mit dem in der zweiten Eingabevariablen T übergebenen Wert übereinstimmt. Bei Übereinstimmung, hier also beim Drücken der richtigen Taste a, wird als Ergebnis die Anzahl I der Wiederholungen ausgegeben, was ein Maß für die Wartezeit darstellt. Wird dagegen keine Übereinstimmung festgestellt, so erscheint eine Fehlermeldung. Falls Sie also statt der richtigen Taste a irgendeine andere Taste tippen, etwa b, reagiert TASTE mit

```
b ist ein Fehler!
```

Antwort: -1

Zur Unterscheidung gibt TASTE jetzt den Wert (-1) als Ergebnis zurück.

Bildschirm-Layout

Ausgaben auf dem Bildschirm erscheinen dort, wo sich der Cursor gerade befindet, was wir bisher weitgehend dem LOGO-System überlassen haben. In der Regel wird der Cursor dabei an den Anfang der nächsten freien Bildschirmzeile gesetzt.

In der Prozedur UEBERSCHRIFT wurde mit ct in den Bildschirmaufbau eingegriffen. Der Textschirm wird gelöscht und der Cursor in die obere linke Ecke gesetzt. Der Cursor kann mit Hilfe des LOGO-Wortes setcursor beliebig auf dem Bildschirm positioniert werden.

```
setcursor [20 12]
```

Damit wird der Cursor in die Spalte 20 in der Zeile Nr. 12 gesetzt. Wenn Sie diese Zeile im Direktbetrieb eingegeben haben, dann erscheint in der Bildschirmmitte das Fragezeichen und daneben der blinkende Cursor. Umgekehrt kann man mit cursor die geltende Lage des Cursors erfragen

```
cursor
```

Ergebnis: [0 13]

Das Ergebnis von cursor ist eine Liste aus zwei Zahlen. Die erste bedeutet wie bei setcursor die Spalten-, die zweite die Zeilennummer der beim Aufruf gültigen Cursorposition. Die gültige Position ist diejenige, bei der die nächste Ausgabe erfolgt bzw. erfolgen würde.

Beachten Sie, daß das Ergebnis eine Liste ist. Weil LOGO immer nur einen Ausgabewert für Prozeduren zuläßt, so wie eine mathematische Funktion einen Zahlenwert liefert, müssen die beiden Angaben für die Lage des Cursors in einer Liste zu einem einzigen Objekt zusammengefaßt werden. Um beispielsweise die Zeilennummer allein zu bekommen, muß die Ausgabeliste zerlegt werden

```
last cursor
```

Für die Spaltennummer wird entsprechend first verwendet. In dieser Weise muß auch bei selbstprogrammierten Prozeduren verfahren werden, wenn mehrere Resultate zurückgegeben werden sollen.

- setcursor benötigt eine Liste aus zwei Werten als Eingabe. Der erste gibt die Spaltennummer (0-39), der zweite die Zeilennummer (0-24) an.
- cursor verlangt keine Eingabe. Als Ergebnis wird die gültige Position des Cursors in der Form einer zweielementigen Liste zurückgegeben.

Die Verwaltung von Angaben zu Personen ist eine wichtige, in mancher Hinsicht auch problematische Anwendung von Computern. Es soll hier nur eine kleine Teilaufgabe bearbeitet werden. Das Programm soll zur Eingabe von Name, Vorname und Adresse einer Person dienen. Ein solches Programm soll anwenderfreundlich sein, d.h. der Benutzer soll nur die unbedingt notwendigen Tastendrücke erledigen. Er soll dabei vom Programm her mit Kommentaren und einer Steuerung des Cursors unterstützt werden.

Komfortable Programme nehmen auch eine Prüfung der Eingaben vor, weisen unzulässige Eingaben zurück, erlauben Korrekturen usw. Es sollen hier nur Anregungen für die eigene Programmierarbeit gegeben werden.

Einfache Eingabemaske

Die Bedienerführung geschieht bei Programmen dieser Art durch sogenannte Bildschirmmasken. Dabei wird ein Teil des Bildschirms vom Steuerprogramm her beschrieben. Nur in bestimmten Bereichen sind Lücken vorgesehen, die dann bei der Eingabe gefüllt werden. Der Cursor muß dabei so gesteuert werden, daß nur die vorgesehenen Lücken vom Anwender beschrieben werden können - in der Regel in einer vorher festgelegten Reihenfolge. Hier sollen Name, Vorname, Straße, Hausnummer, Postleitzahl, und Ort aufgenommen werden.

Die einfachste Art einer solchen Eingabemaske läßt in einer Bildschirmzeile genau eine Eingabe zu. Wenn die vom Benutzer eingegebenen Zeichen eine Zeile abschließen, kann der Anwender nichts löschen, solange er in der Zeile bleibt.

```
to MASKE1
ct
setcursor [0 1] pr [Name:..............]
setcursor [0 3] pr [Vorname:..............]
setcursor [0 5] pr [Strasse:..............]
setcursor [0 7] pr [Nr.:....]
setcursor [0 9] pr [PLZ:....]
setcursor [0 11] pr [Ort:..............]
end

to EINGABE1
MASKE1
setcursor [5 1] make "Name rq
setcursor [8 3] make "Vorname rq
setcursor [8 5] make "Strasse rq
setcursor [4 7] make "Nr rq
setcursor [4 9] make "PLZ rq
setcursor [4 11] make "Ort rq
op (se :Name :Vorname :Strasse :Nr :PLZ :Ort)
end
```

Es wird keine Überprüfung der Eingaben vorgenommen. Man könnte in diesem Zusammenhang daran denken, bei der Eingabe von Hausnummer und Postleitzahl mit Hilfe des Prüfwortes numberp festzustellen, ob tatsächlich eine Zahl eingegeben worden ist. Wenn nicht, dann kann die entsprechende Zeile wiederholt werden. Bei den anderen Angaben könnte umgekehrt dafür gesorgt werden, daß Zahlen zurückgewiesen werden.

Nach dem Aufruf von

```
EINGABE1
```

erscheint die programmierte Maske auf dem Bildschirm, der Cursor steht dabei an der für den Namen vorgesehenen Stelle. Wird die Eingabe des Namens durch die Returntaste abgeschlossen, springt der Cursor automatisch an die für den Vornamen vorgesehene Stelle usw. Als Antwort erscheint schließlich beispielsweise

Ergebnis: `[Wurst Hans Haudegenweg 52 4000 Düsseldorf]`

Um eine ganze Adressenliste aufzunehmen, muß die Eingabe wiederholt werden

```
to ADRESSEN :liste
local "Person
label "ein make "Person EINGABE1
if first :Person = "* [op :liste]
make "liste lput :Person :liste
go "ein
end
```

Die Wiederholung ist hier zur Abwechslung einmal nicht rekursiv, d.h. nicht durch Selbstaufruf programmiert worden. Die Eingabe von einzelnen Adressen, die in der Prozedur EINGABE1 erfolgt, wird solange wiederholt, bis als Name ein Sternchen (Multiplikationszeichen) erscheint. Der Abbruch geschieht also dadurch, daß beim Namen ein *-Zeichen eingetippt wird. Bei den restlichen Angaben kann einfach mit der Returntaste weitergeschaltet werden.

Weil die Adressenliste als Eingabe erscheint, kann mit der Prozedur ADRESSEN eine bereits vorhandene Liste verlängert werden. Beim ersten Aufruf gibt man eine leere Liste ein.

Erster Aufruf: `make "Verein ADRESSEN []`

Danach folgt etwa die Eingabe der Adressen für mehrere Personen.

Weitere Aufrufe: `make "Verein ADRESSEN :Verein`

Eingabemasken mit rc

Die Eingabe mit rq ist einfach zu handhaben, weil bei der Eingabe die Zeile wie eine normale LOGO-Eingabezeile behandelt wird. Vor dem Übergeben der Eingabezeile mit der Returntaste kann mit den Cursorsteuertasten nach rechts und links gegangen werden. Daneben gibt es die Editiermöglichkeiten mit der CONTROL-Taste. Auf der anderen Seite kann man die Länge der Eingabezeile nicht weiter einschränken, eine Eingabemaske kann dadurch vom Anwender zerstört werden. Man kann rq also nicht ohne weiteres verwenden, wenn bei einer Eingabemaske mehrere Eingaben innerhalb einer Zeile erfolgen sollen.

Als Alternative bietet sich die zeichenweise Eingabe mit rc an, die sich leichter kontrollieren läßt. Bei dieser Eingabemethode müssen dann Cursorsteuerungen selbst programmiert werden. Bei rc wird auch das eingegebene Zeichen nicht automatisch auf dem Bildschirm ausgegeben. Bei einer Eingabemaske müssen dann die eingegebenen Zeichen mit type eigens in die vorgesehenen Lücken gebracht werden. Als Grundlage für diesen Typ der Eingabe kann die folgende Prozedur EIN dienen.

```
to EIN :S :Z :M :W
setcursor se :S :Z type char 32
setcursor se :S :Z
(local "c "ac)
label "taste make "c rc make "ac ascii :c
if :ac = 248 [op EIN :S-1 :Z :M bl :W]
if :ac = 243 [op :W]
if and :ac > 31 :ac < 123]
[make "W word :W :c type :c] [go "taste]
if :S = :M [op :W]
op EIN :S+1 :Z :M :W
end
```

Diese Prozedur hat die Aufgabe, beginnend mit der Cursorposition, die durch die Eingabewerte S und Z (Spalte und Zeile) gegeben ist, ein Wort als Eingabe zu bestimmen, das sich bis maximal zur Spaltenposition M (3. Eingabewert) erstreckt.

In der ersten Zeile von EIN wird mit type char 32 die aktuelle Cursorposition zunächst gelöscht. Als Korrekturmöglichkeit ist nur die DEL-Taste vorgesehen, die zum ASCII-Code 248 gehört. Die Returntaste hat den Code 243 und bewirkt die Beendigung des jeweiligen Eingabefeldes.

In der achten Zeile wird überprüft, ob eine Taste gedrückt wurde, die im ASCII-Code zwischen 32 (Leertaste) und 122 (z) liegt. Die Annahme anderer Tasten wird verweigert. Dieser Test verwendet die LOGO-Vokabel

```
and
```

die in der vorliegenden LOGO-Version nur in der Präfix-Notation benutzt werden darf, d.h. das and muß vor den zu verknüpfenden logischen Ausdrücken stehen.

Wird das eingegebene Zeichen akzeptiert, dann wird es an das entstehende Wort W angehängt und gleichzeitig auf dem Bildschirm ausgegeben. Ist die maximale Spaltenposition M erreicht, dann erfolgt ebenfalls ein Abbruch der Eingabe. Damit können vorher beschriebene Teile der Eingabemaske vor dem Über-

schreiben durch eine Benutzereingabe geschützt werden. Die Prozedur EIN ruft sich schließlich selbst auf, wobei die Spaltenposition um eins erhöht wird.

```
EIN Ø 1Ø 5 "
```

Der Cursor erscheint nun am Anfang der Zeile 10. Bei einer Eingabe werden die Zeichen wie gewohnt auf den Bildschirm geschrieben, der Cursor wandert dabei jeweils um eine Position nach rechts. Wird die Spalte 5 erreicht, dann werden weiter eingegebene Tasten zunächst nicht angenommen, es erscheint vielmehr zuerst.

Ergebnis: ...

Die Prozedur EIN wurde also abgebrochen. Damit kann die vorher verwendete Prozedur EINGABE1 umgeschrieben werden, so daß die Eingabemaske mehrere Daten pro Zeile möglich macht.

```
to EINGABE
MASKE
make "Name EIN 5 1 19 "
make "Vorname EIN 28 1 39 "
make "Strasse EIN 8 3 19 "
make "Nr EIN 23 3 27 "
make "PLZ EIN 4 5 8 "
make "Ort EIN 24 5 39 "
op (se :Name :Vorname :Strasse :Nr :PLZ :Ort)
end
to MASKE
ct
setcursor [Ø 1] pr [Name:.............]
setcursor [2Ø 1] pr [Vorname:.........]
setcursor [Ø 3] pr [Strasse:.........]
setcursor [2Ø 3] pr [Nr.....]
setcursor [Ø 5] pr [PLZ:....]
setcursor [2Ø 5] pr [Ort:.............]
end
```

Natürlich lassen sich hier noch mancherlei Verbesserungen hinsichtlich der Prüfung der Eingabewerte anbringen. Die Beispiele lassen aber erkennen, wie man in LOgo professionelle Eingabemasken programmieren kann. In Lektion 14 wird das Thema noch einmal aufgegriffen. Dort wird ein Programm behandelt, das es dem Benutzer erlaubt, die Maske direkt auf den Bildschirm zu schreiben. Er muß das Programm MASKE nicht selbst schreiben, das erledigt dort das Programm für ihn. In LOGO kann man nämlich mit Programmen andere Programme erzeugen.

Übungen

1) In den ersten Lektionen mußten Zahlen, Wörter, Listen direkt beim Aufruf von Prozeduren in der Eingabezeile geschrieben werden. Jetzt können die Programmbeispiele mit Hilfe von rl, rq und rc mit aktuellen Eingabewerten versorgt werden. Greifen Sie auf alte Beispiele zurück und benutzen die Programme so, daß Sie die Werte im Programmablauf eingeben.

2) Es sollen Dezimalzahlen eingegeben und anschließend mit zwei Nachkommastellen ausgegeben werden. Vor die Zahl sollen soviele Leerzeichen gesetzt werden, daß das entstehende Zahlwort immer acht Zeichen (einschließlich der Leerzeichen) enthält.

3) Formulieren Sie eine Prozedur, die mit Hilfe von Tastendrücken die Schildkröte über den Bildschirm kommandiert (etwa Taste V für vorwärts, Taste R für Rechtsdrehen, usw.).

4) Erzeugen Sie aus den Buchstaben des Alphabets durch zufällige Auswahl von Buchstaben Zufallsworte. Benutzen Sie zur Auswahl die Vokabeln ascii und char.

5) Die Prozedur soll jedem Buchstaben den jeweils übernächsten im Alphabet zuordnen. Am Ende des Alphabets soll zum Anfang zurückgekehrt werden, d.h. Y wird A zugeordnet usw.

6) Entwickeln Sie (vgl. Aufg. 5) ein Programm für einen Geheimcode, bei dem jeder Buchstabe durch einen anderen in unregelmäßiger Weise ersetzt wird. Für die Zuordnung können Zufallszahlen nutzbar gemacht werden, wenn Sie mit Hilfe von rerandom dafür sorgen, daß reproduzierbare Folgen durchlaufen werden.

7) Es soll eine Tabelle eingegeben werden, die in einer Spalte genau zehn Werte enthält. Die Spalten sollen einen festen Abstand voneinander haben. Schreiben Sie ein Programm, das eingegebene Zahlen in dieser Weise tabelliert auf dem Bildschirm ausgibt.

Lektion 11: Grafik mit Koordinaten

Neue Sprachelemente:
setpos, setx, sety, seth, towards, sf, setbg, pe, px, setpc, tf, fs, ss, setsplit, setscrunch

Programme:
TURTLE.JAGD, SONNENSYSTEM, BOX, ELLIPSE, POLYGON, KREISZAHL, FEUERWERK, SWS, ZUCKEN, SINUSKURVE, PLOTTER (Funktionszeichner mit Hilfsprogrammen)

11.1 Zeichnen im Koordinatensystem

Die Grundbefehle der Turtle-Grafik bewirken Bewegungen nach vorwärts bzw. rückwärts und Drehungen nach rechts bzw. links. Die Zeichnungen entstehen als auf dem Grafikschirm aufgezeichnete Spuren der Schildkröte. Diese Grundkommandos sind auf den jeweiligen Ort und die jeweilige Richtung der Turtle bezogen, sozusagen als 'egozentrische' Anweisungen.

Grundlage der meisten Grafiksysteme sind dagegen Anweisungen, die den Zeichenstift an eine bestimmte Stelle auf dem Bildschirm oder Papier bringen. Um einen Punkt festzulegen, braucht man ein Koordinatensystem.

Auf dem Globus wird die geographische Lage beispielsweise durch Längen- und Breitengrad eines Ortes angegeben. Beim ebenen Bildschirm verwendet man die kürzesten Abstände des Punktes von zwei Koordinatenachsen, nämlich einer waagerechten und einer senkrechten Achse, die sich im sogenannten Koordinatenursprung schneiden.

Welche der beiden Methoden am besten geeignet ist, hängt von der jeweiligen Aufgabenstellung ab. Wenn Sie die Schildkröte an einen bestimmten Platz im Grafikfenster bringen wollen, müssen Sie mit den bisher betrachteten Grafikbefehlen (fd, bk, rt, lt)

möglicherweise länger herumprobieren, bis dies wirklich exakt gelingt.

LOGO kennt auch die Beschreibung mit Koordinaten.

Ein bestimmter Punkt, nämlich die Bildschirmmitte, kann bereits unmittelbar angesteuert werden durch

home (Turtle zur Mitte des Grafikfensters).

Der Mittelpunkt ist auch gleichzeitig der Koordinatenursprung für LOGO. Das Kommando home kann also auch so beschrieben werden: Setze den Zeichenstift an den Koordinatenursprung, also an den Punkt mit der x-Koordinate (waagerecht) 0 und der y-Koordinate (senkrecht) 0.

Tatsächlich wird bei home auch noch zusätzlich die Turtle senkrecht nach oben ausgerichtet, vorläufig soll aber die Richtung der Schildkröte außer Betracht bleiben.

Um den Zeichenstift auf einen bestimmten Punkt zu setzen dient das LOGO-Wort setpos. Dieses erfordert als Eingabe die Liste der beiden Koordinatenwerte, zuerst die x- dann die y-Koordinate.

```
setpos [Ø Ø]
setpos [5Ø 5Ø]
setpos [-1ØØ 78]
setpos [-5Ø -1ØØ]
```

Wahrscheinlich sind jetzt Strecken auf dem Bildschirm zu sehen, die die angegebenen Punkte miteinander verbinden. Die Grafikbefehle sind in LOGO grundsätzlich als Bewegungen des Zeichenstifts anzusehen. Ob dann eine Linie gezeichnet wird oder nicht hängt davon ab, ob der Zeichenstift abgesenkt oder angehoben war. Wird erst pu vorausgeschickt, dann wird die Schildkröte nur an die angegebenen Punkte gesetzt.

Die Koordinaten müssen als Liste übergeben werden. Handelt es sich, wie bei den vorangegangenen Beispielen um Konstanten, so wird die Liste einfach durch die eckigen Klammern erzeugt. In allen anderen Fällen muß eine Liste mit se gebildet werden:

```
make "x 50 make "y -60 setpos se :x :y
```

Statt se können auch andere Vokabeln zum Aufbau von Listen benutzt werden.

Welche Werte dürfen die Koordinaten annehmen?

Die Antwort hängt davon ab, wie groß Sie das Grafikfenster gewählt haben. Die Bildschirmgröße des Schneider CPC unterscheidet sich auch wesentlich von der des Schneider Joyce.

Standardgrößen

Mit der Standardgröße ist die Größe des Grafikfensters gemeint, über die sie verfügen können, wenn Sie entweder mit dem normalgeteilten Grafikschirm oder mit dem gesamten verfügbaren Bildschirm arbeiten. Dazu gibt es die Kommandos

ss (Splitscreen) und fs (Fullscreen)

In den Splitscreen-Zustand kommt man auch durch ein beliebiges Grafik-Kommando.

CPC: x-Koordinate -320 - 320,
y-Koordinate -192(-112) - 191

Joyce: x-Koordinate -360 - 360,
y-Koordinate -265(-94) - 263

Dabei beziehen sich die unteren Grenzen in Klammern auf die Standardeinstellung des geteilten Bildschirms.

Mit dem LOGO-Kommando

```
setsplit 5
```

kann die Größe des Textfensters beim geteilten Bildschirm festgelegt werden. Wie dann die untere Grenze für die y-Koordinate aussieht, können Sie jeweils selbst ausprobieren, indem Sie mit fence eine Überschreitung des Grafikfensters verbieten und dann die Turtle soweit nach unten schieben, bis die Fehlermeldung "Turtle out of bounds" erscheint.

Achsenparallele Bewegungen

Bei setpos müssen beide Koordinaten angegeben werden. Es gibt aber auch eine vereinfachte Form, bei der nur eine der beiden Koordinaten des Punktes nötig ist. Die zweite Koordinate wird dann beim gegenwärtigen Wert festgehalten.

setx 100: Setzt die x-Koordinate auf den Wert 100, die y-Koordinate bleibt erhalten.

sety 50: Setzt entsprechend die y-Koordinate auf den neuen Wert 50, ohne die x-Koordinate zu verändern.

Hier ist die Eingabe keine Liste sondern im Sinne der LOGO-Syntax ein Wort. Ist die Eingabe keine Konstante, dann kann auch ein entsprechender Ausdruck stehen, der einen Zahlenwert liefert:

```
setx 3*:x oder sety random 100
```

Bei abgesenktem Zeichenstift (pd) werden mit setx, sety jeweils Strecken gezeichnet, die parallel zu den beiden Koordinatenachsen verlaufen. Damit kann man etwa die Bildschirmachsen selbst zeichnen:

```
to ACHSENKREUZ
pu setpos [-320 0] pd setx 320;x-Achse
pu setpos [0 -112] pd sety 191;y-Achse
home
end
```

Mit Hilfe der Zufallszahlen kann mit setx und sety auch eine wilde Schildkrötenjagd veranstaltet werden:

```
to TURTLE.JAGD
setx (random 320) - 160
sety (random 260) - 129
TURTLE.JAGD
end
```

Die Kommandos setpos, setx, sety wirken sich zunnächst nur auf die Position der Turtle aus; Zeichnungen entstehen dabei mittelbar bei Bewegungen der Schildkröte mit gesenktem Stift. Es gibt in DR LOGO daneben auch eine Vokabel, um unabhängig von der Turtle einen Punkt an eine ausgewählte Stelle im Koordinatensystem zu setzen:

```
dot [100 120]
```

Die Eingabe ist für dot wie bei setpos eine Liste aus den beiden Koordinatenwerten, die bei konstanten Werten mit eckigen Klammern, bei Namen mit Hilfe von se gebildet wird. Dieser Befehl entspricht dem Grundbefehl in den üblichen Grafiksystemen. Anwendungen zu dot werden erst in Lektion 14 betrachtet.

11.2 Turtleposition im Koordinatensystem

Die Schildkröte kann mit setpos, setx, sety an bestimmte Punkte im Koordinatensystem kommandiert werden. Ebenso will man auch umgekehrt den Ort der Turtle im Koordinatensystem feststellen. Mit der Funktion tf (Abkürzung für Turtlefacts) wird alle Information über den Zustand der Schildkröte offengelegt.

```
cs setpos [100 120] show tf
```

Antwort: `[00 120 0 PD 1 TRUE]`

Als Ergebnis von tf wird eine Liste von sechs Elementen ausgegeben. Die ersten beiden Elemente geben die Position der Turtle im Koordinatensystem wieder. Wir können auch eine eigene Vokabel POS definieren, die nur die Koordinatenwerte ausgibt.

```
to POS
op piece 1 2 tf
end
```

Das letzte Element der von tf erzeugten Liste gibt an, ob die Turtle sichtbar ist, während das vierte den aktuellen Zustand des Zeichenstifts angibt (PD, PU, PE oder PX).

```
ht pu tf
```

Antwort: `[100 120 0 PU 1 FALSE]`

Das vorletzte Element zeigt die Farbe des Zeichenstifts an, hier beispielsweise den Farbcode 1 und die Bedeutung des dritten Wertes beschreibt die Richtung der Turtle, worauf etwas später eingegangen wird. Zunächst soll noch eine Anwendung der Koordinatenbestimmung betrachtet werden.

Bestimmung der Kreiszahl

Die Kreiszahl PI ist das Verhältnis von Kreisumfang zu Kreisdurchmesser. Dieses Verhältnis kann man näherungsweise berechnen, wenn ein Kreis durch ein Vieleck angenähert wird. Die mathematische Berechnung des Umfangs bei einem regelmäßigen Vieleck ist kompliziert. Mit LOGO läßt sich das experimentell sehr einfach machen: Der Umfang eines Vielecks ergibt sich einfach durch die Summe der Seitenlängen, den Durchmesser können wir nun mit tf messen.

```
to KREISZAHL
setx -140
repeat 180 [fd 2 rt 1]
op 360*2/((first tf)+140)
end
```

Tatsächlich wird nur ein Halbkreis gezeichnet, damit der Durchmesser als Änderung der Turtleposition gegenüber dem Anfangspunkt gemessen werden kann. Die Abweichung vom exakten Wert der Kreiszahl PI ist nur 1/50 Promille!

11.3 Die Richtung der Turtle

Die Schildkröte oder Turtle ist ein gerichteter Zeichenstift, dessen Richtung auf dem Bildschirm durch die Spitze des Dreiecks angezeigt wird. Bei der Ausführung der Kommandos

```
cs und home
```

wird die Turtle in der Bildschirmmitte plaziert und senkrecht nach oben ausgerichtet. Diese Richtung ist die Nullstellung des Zeichenstifts. Absolute Angaben über die Richtung der Schildkröte beziehen sich auf die Ausrichtung nach oben.

Die Drehung der Turtle geschieht meist auf die aktuelle Stellung bezogen.

```
rt 10 bzw. lt 10
```

drehen die Schildkröte jeweils um zehn Grad. Drehwinkel können auch negativ sein; eine Rechtsdrehung um (-10) Grad entspricht einer Linksdrehung um 10 Grad. Die Winkel dürfen auch über 360 Grad liegen.

Es gibt daneben auch die Möglichkeit, die Richtung absolut, d.h. bezogen auf die Ausgangsstellung anzugeben, so wie man eine Himmelsrichtung angeben kann. Hierfür dient die LOGO-Vokabel seth.

seth richtet die Turtle unabhängig von ihrer gegenwärtigen Ausrichtung nach dem angegebenen Winkel aus.

```
seth 90
```
(Abkürzung von Set Heading)

richtet die Schildkröte waagerecht nach rechts orientiert aus, im Vergleich zu einer Landkarte also nach Osten. Die Eingabe für seth ist der Winkel im Uhrzeigersinn, bezogen auf die Nullstellung.

Die aktuelle Ausrichtung der Turtle erfährt man umgekehrt als drittes Element bei der Ausgabe von tf:

```
cs seth 37 pr item 3 tf
```

Antwort: 37

Probieren Sie etwa

```
to FEUERWERK
seth random 360
fd random 130
home
FEUERWERK
end
```

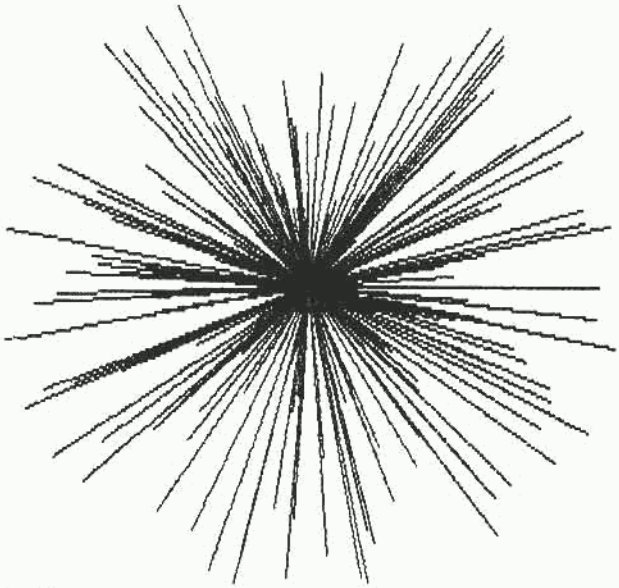


Abb. 11.1: Feuerwerk

Die Schildkröte kann auch auf einen bestimmten Punkt zielen und die zugehörige Richtung angeben.

towards gibt als Wert die "Himmelsrichtung" an, in der ein angegebener Punkt von der gegenwärtigen Turtle-Position aus liegt. towards benötigt als Eingabewerte die X- und die Y-Koordinate des angepeilten Punktes, die wie bei SETPOS zu einer Liste zusammengefaßt werden.

```
home towards [50 50]
```

Antwort: 45

Mit der Kombination

```
seth towards [50 70]
```

kann die Schildkröte auf den Punkt mit den Koordinaten (50;70) ausgerichtet werden.

Beispiel: Dreieckskonstruktion

Als Beispiel für die LOGO-Vokabeln zur absoluten Orientierung kann die Prozedur SWS dienen.

```
to SWS :c :alpha :b;Dreieck Seite Winkel Seite
(local "bk "gamma)
make "bk tf;Position von Punkt B
seth 90 bk :c lt :alpha fd :b
make "gamma 180 + (item 3 tf) - towards :bk
(pr [Winkel Alpha:] :alpha [Beta: ] 180 - :alpha - :gamma
[Gamma: ] :gamma)
setpos :bk
end
```

Die Prozedur zeichnet ein Dreieck, von dem zwei Seiten und der eingeschlossene Winkel vorgegeben sind. In der fünften Zeile wird die Richtung des Ausgangspunktes und daraus wiederum der Winkel Gamma, anschließend auch noch Beta, bestimmt.

Unter dem Namen bk wird das Ergebnis von tf aufgehoben, wenn sich die Turtle beim Dreieckspunkt B befindet. Mit towards :bk wird anschließend auf den Punkt B gezielt, wobei eigentlich für towards nur eine Liste aus den beiden Koordinatenwerten erwartet wird. Durch Experimentieren stellt sich heraus, daß DR LOGO keinen Anstoß daran nimmt, daß die Liste noch weitere Elemente enthält. Auch bei setpos :bk wird das gewünschte mit der sechselementigen Liste erreicht.

11.4 Figuren im Koordinatennetz

DR LOGO auf den Schneidercomputern besitzt im Grundwortschatz keine eigenen Vokabeln, um Figuren wie Rechtecke, Kreise, Ellipsen usw. zu zeichnen. Untersuchen Sie die beiden folgenden Beispiele.

```
to BOX :l
pu ht setpos :l
pd seth 90
repeat 2 [fd item 3 :l lt 90 fd last :l lt 90]
end

to POLY :ko
pu setpos :ko pd
POL bf bf :ko
end

to POL :ko
if emptyp :ko [stop]
setpos :ko
POL bf bf :ko
end
```

Mit BOX wird ein Rechteck gezeichnet, dessen linke untere Ecke durch die ersten beiden Elemente, und dessen Länge und Breite jeweils durch das dritte bzw. vierte Element der Eingabeliste gegeben sind. Probieren Sie aus:

```
fs cs repeat 70 [BOX (se rx ry random 120 random 120)]
```

Zur zufälligen Auswahl von Punkten auf dem Grafikschirm werden hier zwei Hilfsprozeduren rx, ry benutzt.

```
to RX
op (random 500) - 300
end
to RY
op (random 300) - 200
end
```

Damit wurde die Grafik in Abb. 11.2 gezeichnet.

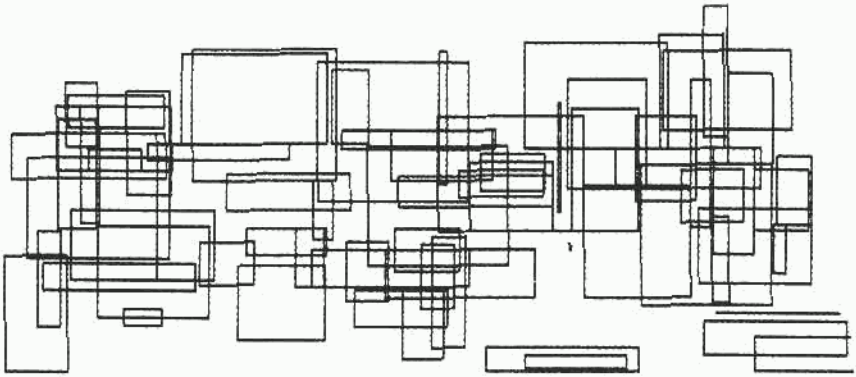


Abb. 11.2: Zufallsrechtecke

Um aus den Angaben für ein Rechteck die für BOX benötigte Liste aufzubauen, wird se verwendet; eckige Klammern können nur bei Konstanten genommen werden. Die runden Klammern sind notwendig, weil mehr als zwei Eingaben für se folgen.

Mit der Prozedur POLY wird ein Polygonzug gezeichnet, wobei die Koordinaten der Eckpunkte durch die Eingabeliste festgelegt werden. Das einfachste Polygon ist eine Strecke, ein Dreieck ein weiteres Beispiel:

```
cs
POLY [-100 0 120 150] POLY [-100 -50 90 -30 0 180 -100 -50]
```

Um mit POLY etwas spielen zu können, soll zuerst eine Prozedur definiert werden, die die Koordinaten eines Polygonzugs zufällig erzeugt.

```
to POLYGON :anzahl
local "l make "l []
repeat :anzahl [make "l (se :l rx ry)]
op :l
end
```

Dabei werden die Hilfsfunktionen rx und ry verwendet, die jeweils an de gewünschten Bereich angepaßt werden können. Probieren Sie nun aus:

```
fs cs ht POLY POLYGON 50
```

11.5 Bildschirmzustände

Mit der LOGO-Funktion sf (Abkürzung für Screenfacts) wird die Information über den Zustand des Bildschirms ausgegeben.

```
sf
```

Antwort: `[0 SS 5 WINDOW 1]` bzw. `[0 SS 10 WINDOW 0.46875]`

Die erste Antwort ist die Standardantwort beim Schneider CPC, die zweite die beim Joyce.

Die Bedeutung der Elemente ist:

- Hintergrundfarbe des Grafikfensters
- Fensterzustand (TS SS oder FS)
- Fensteraufteilung (Anzahl der Textzeilen bei SS)
- Begrenzung des Grafikfensters (FENCE, WINDOW oder WRAP)
- Achsenverhältnis

Die einzelnen Werte können mit den folgenden Kommandos beeinflußt werden.

- setbg 1 legt den Farbcode für den Hintergrund des Grafikfensters fest.
- ss, fs, ts stellen den Fensterzustand ein.
- setsplit 7 definieren die Größe des Textfensters.
- window, fence, wrap legen die Reaktion bei der Überchreitung der Granzen des Grafikschirms fest.
- setscrunch 0.7 ändert das Achsenverhälnis auf 0.7 ab.

(Farbänderungen werden erst nach Löschen mit cs bzw. clean, ct wirksam.)

Hier soll speziell das Achsenverhältnis interessieren. Die Abstände der einzelnen Bildpunkte eines Monitors sind in waagerechter und senkrechter Richtung unterschiedlich, wobei die tatsächlichen Größenverhältnisse auch vom jeweiligen Monitor abhängen können. In LOGO wird dafür gesorgt, daß bei den Zeichenbefehlen darauf keine Rücksicht genommen werden muß. Die unterschiedlichen Voreinstellungen beim CPC und dem Joyce stellen die unterschiedlichen Monitore bzw. deren Betriebsweisen in Rechnung.

Mit setscrunch kann das Achsenverhältnis gezielt verändert werden. Dann zeichnet beispielsweise die Prozedur QUADRAT in Wirklichkeit ein Paralellogramm, also ein gestrecktes bzw. gestauchtes Quadrat. Beachten Sie, daß sich ein verändertes Achsenverhältnis auch auf die Lage eines Punktes auswirkt, wenn er beispielsweise mit setposs angesprochen wird. Damit ändert sich nicht nur die Gestalt einer Figur, sondern auch ihre Lage auf dem Grafikfenster.

Mit Zufallszahlen kann man nun allein mit der Prozedur QUADRAT spielerisch allerhand erzeugen. So hat die folgende Zeile die Grafik in Abb. 11.3 erstellt.

```
fs cs repeat 50 [pu setpos se (random 600) - 300 !
(random 200) - 100 pd seth random 360 !
setscrunch 0.1 + (random 20)/10 quadrat 10]
```

Dabei hat die Eingabe von QUADRAT noch einen festen Wert

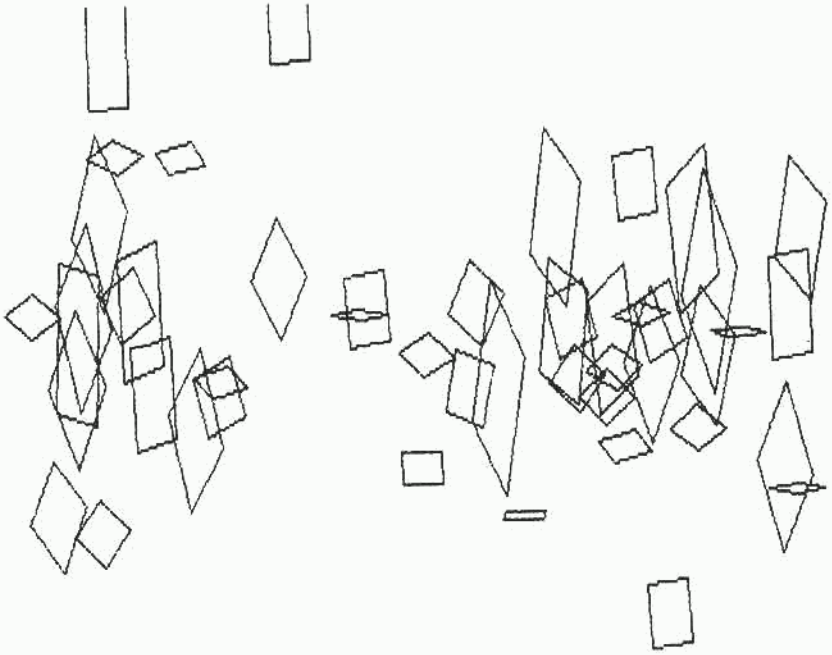


Abb. 11.3: Spielereien mit QUADRAT und setscrunch

Bei krummlinigen Figuren ist die Nutzung von setscrunch noch interesanter. So werden durch Streckung bzw. Stauchung aus Kreisen Ellipsen, die ohne die Anwendung von setscrunch nur mit komplizierteren Umrechnungen gezeichnet werden könnten.

```
to KREIS :r
pu fd :r rt 95 pd
local "s make "s 0.1745*:r
repeat 36 [fd :s rt 10]
pu lt 95 bk :r
end
```

```
to ELLIPSE :a :b
setscrunch :b/:a;beim Joyce mit 0.46875 multiplizieren
KREIS :a
end
```

Die beiden Halbachsen der Ellipse sind die Eingabe von ELLIPSE, wobei sich a auf die waagerechte und b auf die senkrechte Achse bezieht. Es sind allerdings nicht beliebige Werte für das Achsenverhältnis möglich, weil DR LOGO nur Achsenverhältnisse bis 1:10 bzw. 10:1 zuläßt. (Das wirkt sich dann beim Joyce wegen des zusätzlichen Fektors 0.46875 etwas anders aus).

Bei der Wiederkehr des berühmten Halleyschen Kometen 1985/86 soll uns seine Bahn als Beispiel dienen; dabei sollen die Winkel zwischen den Bahnebenen unbeachtet bleiben. In der folgenden Prozedur sind darüber hinaus alle Planeten des Sonnensystems vorgesehen.

```
to SONNENSYSTEM :r
local "k
ELLIPSE 0.387 * :r 0.379 * :r;Merkur
ELLIPSE 0.723 * :r 0.723 * :r;Venus
ELLIPSE :r :r;Erde
ELLIPSE 1.524 * :r 1.517 * :r;Mars
ELLIPSE 5.203 * :r 5.197 * :r;Jupiter
ELLIPSE 9.539 * :r 9.524 * :r;Saturn
ELLIPSE 19.18 * :r 19.16 * :r;Uranus
ELLIPSE 30.06 * :r 30.06 * :r;Neptun
make "k tf seth 0 fd 9.9 * :r
ELLIPSE 39.75 * :r 38.5 * :r;Pluto
setpos :k seth 90 fd 17.75 * :r seth 0
ELLIPSE 18.34 * :r 4.58 * :r;Halley-Komet
setpos :k
end
```

Der Eingabewert von SONNENSYSTEM ist der Radius der Erdbahn. Je nach Eingabe werden alle eingegebenen Bahnen oder nur ein Teil des Planetensystems sichtbar. Hier muß man erst einmal etwas mit der Eingabegröße spielen.

Damit der Halley-Komet ganz ins Blickfeld kommt, ist es zweckmäßig, auch den Mittelpunk zu verschieben, das Ergebnis ist in Abb. 11.4 zu sehen:

```
fs cs ht pu setx -140 SONNENSYSTEM 12
```

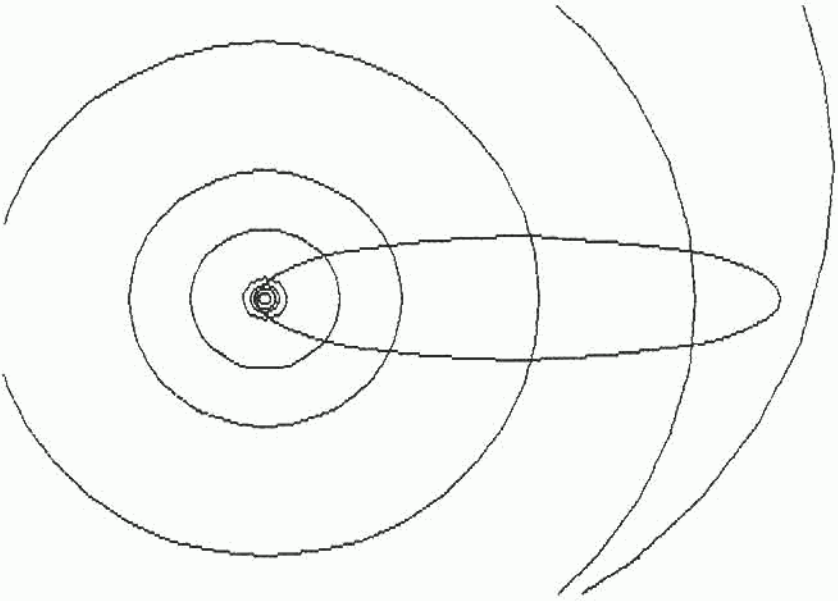


Abb. 11.4: Die Bahn des Halleykometen

11.6 Funktionen darstellen

Der Zeichenbefehl setpos ist für die grafische Darstellung von Funktionswerten geeignet. Um die Wellenlinie der Sinusfunktion zu zeichnen, kann das folgende Programm

```
to SINUSKURVE :x
if :x > 319 [stop]
setpos se :x 100 * sin :x * 2
pd
SINUSKURVE :x+6
end
```

beispielsweise mit

```
cs pu SINUSKURVE -319
```

aufgerufen werden. Die Wiederholung wird hier wieder durch Rekursion erreicht. Beim Rekursionsaufruf wird die Bildschirmkoordinate um 6 erhöht. Diese Genauigkeit reicht für viele Zwecke aus, bei geringerer Schrittweite dauert das Zeichnen entsprechend länger.

Da die Turtle beim Zeichnen von Funktionsgraphen keine wesentliche Rolle spielt, kann man sie auch verstecken. Beim Aufruf von SINUSKURVE wird der Zeichenstift mit pu angehoben, und erst einmal an den Kurvenanfang gebracht, ohne daß eine Verbindung zur aktuellen Turtle-Position sichtbar wird.

Bei der grafischen Darstellung von Funktionen müssen einige Probleme gelöst werden, damit ein Zeichenprogramm nicht nur bei einer speziellen Funktion ein sinnvolles Bild erzeugt.

Zunächst einmal muß eine flexible Umrechnung zwischen den absoluten Einheiten auf dem Bildschirm und den für die Berechnung der Funktionswerte benötigten Einheiten vorgesehen werden. Im Programm SINUSKURVE bedeutet x die absolute Bildschirmkoordinate, die Umrechnung erfolgt dann beim Funktionsaufruf. Der berechnete Funktionswert muß

anschließend mit einem geeigneten Skalenfaktor auf sinnvolle Bildschirmwerte umgerechnet werden.

Gelegentlich wird auch die Lage des Koordinatenursprungs in der Bildschirmmitte zur optimalen Nutzung des Monitors nicht zweckmäßig sein. Dann müssen die Koordinatenwerte umgerechnet werden, um eine passende Verschiebung zu erreichen.

Für das Verständnis eines Funktionsgraphen ist ein Achsenkreuz wünschenswert, wobei geeignete Markierungen die Einheiten deutlich machen sollen.

Ein Funktionszeichner mit Komfort

Verglichen mit den bisher recht kleinen Beispielen, stellt das folgende, relativ komfortable Zeichenprogramm ein größeres Programmierprojekt dar.

```
to PLOTTER :xØ :yØ :ex :ey :schritt;Funktionsplotter
fs window
local "orand make "orand 190;Oberer Rand
local "urand make "urand -190;Unterer Rand
local "lrand make "lrand -310;Linker Rand
local "rrand make "rrand 310;Rechter Rand
KREUZ;Achsenkreuz zeichnen
MARKIEREN :ex :ey;Skalen markieren
pu setpos se :lrand :yØ + FUNKTION (:lrand - :xØ)/:ex pd
KURVE :lrand - :xØ
end
```

Dies ist das Rahmenprogramm, das die einzelnen Bausteine in Gang setzt. Zu Anfang werden die Begrenzungen des Grafikfensters mit den Größen orand, urand, lrand und rrand festgelegt (beim Joyce sind die Werte geeignet abzuändern).

```
to KREUZ;Achsenkreuz mit Rahmen
ht cs
pu setpos se :lrand :orand
pd setx :rrand sety :urand setx :lrand sety :orand
pu setpos se :xØ :orand pd sety :urand pu
setpos se :lrand :yØ pd setx :rrand
end
```

KREUZ zeichnet ein Achsenkreuz. Zusätzlich wird ein Rahmen um den Bildschirm gezeichnet.

```
to MARKIEREN :ex :ey
seth Ø XSKALA :xØ - :ex * quotient :lrand - :xØ -:ex
seth 9Ø YSKALA :yØ - :ey * quotient :urand - :yØ -:ey
end
```

Beachten Sie jeweils die Schreibweise bei den Minuszeichen!

Das Programm MARKIEREN setzt kleine Striche an die Achsen in einem den Einheiten ex und ey entsprechenden Abstand. Diese Markierungen werden auch am Rahmen eingezeichnet, was eine bessere Orientierung ermöglicht.
Bei den Prozeduren KREUZ und MARKIEREN werden keine Eingaben gemacht. Die benötigten Größen sind für diese beiden Vokabeln globale Namen, deren Wert aus dem rufenden Programm PLOTTER entnommen werden, wo sie wiederum lokale Namen sind. Die Hilfprogramme zu Zeichenprogramm können in dieser Formulierung nur innerhalb von PLOTTER aufgerufen werden.

MARKIEREN verwendet die Hilfstätigkeiten XSKALA und YSKALA, um die Skalenmarkierungen zu setzen.

```
to XSKALA :x;Markierung in x-Richtung
if :x > :rrand [stop]
pu setpos se :x :urand pd fd 8
pu sety :yØ - 5 pd fd 1Ø
pu sety :orand - 8 pd fd 8
XSKALA :x + :ex
end
```

```
to YSKALA :y;Markierung in y-Richtung
if :y > :orand [stop]
pu setpos se :lrand :y pd fd 10
pu setx :x0 - 5 pd fd 10
pu setx :rrand - 8 pd fd 8
YSKALA :y + :ey
end
```

Der Funktionsgraph wird in der Prozedur KURVE erzeugt.

```
to KURVE :x
if :x > :rrand [stop]
setpos se :x0 + :x :y0 + :ey * FUNKTION :x/:ex
KURVE :x+:schritt
end
```

Der Funktionswert wird zunächst mit Hilfe der Prozedur FUNKTION berechnet. Zum Zeichnen werden die x- und y-Werte dann in die richtigen Bildschirmkoordinaten umgerechnet. Es wird nicht geprüft, ob die Kurve den oberen bzw. unteren Rand überschreitet. Mit der Option window ist das auch nicht notwendig, wenn orand und urand gerade am tatsächlichen Rand des Grafikfensters liegen.

Schließlich fehlt noch die Prozedur FUNKTION, die die Berechnung der darzustellenden Funktion übernimmt. Das Zeichenprogramm wäre noch flexibler, wenn der Name der Funktion nicht festgelegt wäre, sondern als Eingabewert an das Zeichenprogramm übergeben würde. Das ist in LOGO auch möglich, wird aber erst später behandelt.

```
to FUNKTION :x
op :x * :x
end
```

Hier wird eine Parabel entstehen.

Weil das Zeichenprogramm flexibel ist, können Sie damit spielerisch umgehen. Um eine wirklich gute Darstellung zu erreichen, müssen Sie die günstigste Wahl der Einheiten wie auch der Lage

des Koordinatenursprungs durch mehrmaliges Ausprobieren herausfinden. Bei der Parabel sollte man die waagerechte x-Achse mehr an den unteren Bildschirmrand verlegen. Die Einheiten sollten so gewählt sein, daß gerade interessierende Eigenschaften gut dargestellt werden.

```
cs PLOTTER Ø -19Ø 7Ø 2Ø 3
```

Bei der Sinusfunktion, die in LOGO ja im Gradmaß berechnet wird, ist die Einheit nicht zur Markierung geeignet, da ein sinnvoller Abstand auf dem Bildschirm wenigstens einige Punkte betragen muß. Hier kann eine geeignete Skalierung im Funktionsprogramm Abhilfe schaffen. Hier soll das Beispiel noch etwas interessanter gemacht werden, indem der Sinuswert anschließend durch x dividiert wird.

```
to FUNKTION :x;Besselfunktion jØ
op (sin :x*9Ø)/:x
end

cs PLOTTER Ø Ø 4Ø 1ØØ 3
```

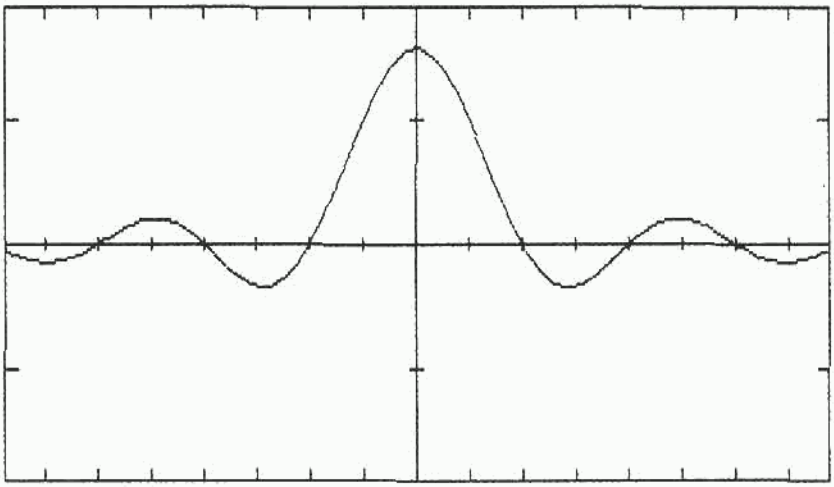


Abb. 11.5: Graph der Funktion sin(x)/x

Mit dieser Änderung erhält die auf den Achsen markierte Einheit die Bedeutung 90 Grad. Ähnlich kann man bei anderen Beispielen vorgehen, wenn ein großer Bereich auf der x-Achse dargestellt werden soll.

Was sollte nun der Anwender dieses Programms zumindest verstanden haben?

Er muß wissen, wie die Tätigkeit PLOTTER in Gang gesetzt wird, d.h. er muß verständlicherweise den Prozedurnamen sowie die Zahl und Bedeutung der Eingabewerte kennen. In der vorliegenden Version muß er darüber hinaus noch wissen, daß die darzustellende Funktion unter dem Prozedurnamen FUNKTION bereitgestellt wird.

Der Anwender muß das Zeichenprogramm selbst nicht verstehen, kann es aber wie eine "Utility", d.h. eine komfortable Spracherweiterung behandeln.

Übungen

1) Verallgemeinern Sie das Programmpaket zur Darstellung von Funktionen so, daß mehrere Funktionsgraphen in einer Zeichnung gleichzeitig dargestellt werden können.

2) Zeichnen Sie den Graphen der Funktion sin(1/x). Umgehen Sie dabei die Stelle Null entweder durch eine geeignete Abfrage im Programm FUNKTION oder durch eine geignete Schrittweite.

Lektion 12: Prozeduren

Neue Sprachelemente:
throw, catch, error, TOPLEVEL

Programme:
XQUADRAT, HORNER, SQRT, TEILER, POCALL
(Programmanalyse mit Hilfsprogrammen), SPEICHERN

12.1 LOGO-Tätigkeiten als Prozeduren

Programmieren als Erfinden neuer LOGO-Vokabeln haben Sie nun schon mehrfach an Hand einiger Beispielen kennengelernt. Die neuen LOGO-Wörter benennen Tätigkeiten, die wiederum Prozeduren genannt werden. In dieser Lektion soll das Prozedurkonzept noch einmal systematisch untersucht werden.

Mit einer Prozedur wird eine Logo-Aktivität als Einheit verfügbar gemacht. Die Tätigkeit kann aus einer einzelnen oder auch aus mehreren Tätigkeiten bestehen. Im einfachsten Fall kann eine Prozedur nur eine einzige Aktivität des LOGO-Systems beinhalten, was dann nur hinsichtlich der Bezeichnung interessant ist. So können wir etwa die Grundvokabeln von LOGO durch deutsche Wörter ersetzen:

```
to VORWAERTS :a
fd :a
end
```

Danach kann das LOGO-Wort fd grundsätzlich durch VORWAERTS ersetzt werden. Genauso können alle anderen LOGO-Vokabeln ins Deutsche übertragen werden.

Dieses einfache Beispiel mit nur einer einzigen LOGO-Aktivität macht natürlich nicht den Sinn des Prozedurkonzepts aus. Prozeduren werden vielmehr mit dem Ziel formuliert, mehrere Operationen unter einem Namen zusammenzufassen, um damit aus einfachen Tätigkeiten mächtige zu machen. Als einfache,

aber doch schon nützliche Tätigkeit kann etwa VIELECK genannt werden.

```
to VIELECK :seite :n
repeat :n [fd :seite rt 360/:n]
end
```

Die zu einer Prozedur zusammengefaßten Tätigkeiten sollten eine überschaubare Einheit bilden. Es ist unsinnig, LOGO-Aktivitäten zur Erklärung einer einzigen Prozedur über mehrere Bildschirmseiten hinweg aneinanderzureihen. Bei den natürlichen Sprachen erwartet man ja auch nicht, daß einzelne Wörter zur Erläuterung ihrer Bedeutung seitenlange Erklärungen im Lexikon benötigen.

In sich abgeschlossene Teilprogramme werden in anderen Programmiersprachen auch Unterprogramme genannt. Insofern können die Prozeduren in LOGO auch als Unterprogramme aufgefaßt werden. Kennzeichnend für LOGO ist der gesamte Aufbau des Programmierens mit selbstdefinierten Tätigkeiten. Während Unterprogramme in anderen höheren Programmiersprachen ein Programmierwerkzeug von vielen darstellt, sind die Prozeduren in LOGO das zentrale Instrument überhaupt. Ein Hauptprogramm gibt es in LOGO im eigentlichen Sinn nicht. Bei komplizierteren Aufgaben gibt es höchstens eine Rahmenprozedur, die dann die Abfolge von Teilaktivitäten unter einer Bezeichnung zusammenfaßt.

Dieser Sprachaufbau macht es erforderlich, eine Problemstellung in Teilprobleme zu zerlegen, die sich als wohldefinierte, unabhängige Problemstellungen selbst bearbeiten lassen. Diese Methode der Problemanalyse und deren Umsetzung in Computerprogramme nennt man strukturiertes Programmieren. In LOGO kann im Grunde genommen nur strukturiert programmiert werden.

12.2 Prozedurtypen

Aufgabe einer Prozedur vom Typ VIELECK ist die Erledigung einer bestimmten Aufgabe, hier dem Zeichnen eines Vielecks, wobei das Ergebnis auf dem Grafikschirm sichtbar wird. Es kann aber nicht von anderen Prozeduren weiter verarbeitet werden und ist somit eine Prozedur ohne Ausgaben.

Wenn ein Ergebnis zur weiteren Bearbeitung zur Verfügung gestellt werden soll, muß das erzeugte Resultat beim Verlassen der Prozedur dem LOGO-System übergeben werden. Hierbei handelt es sich dann um eine Prozedur mit Ausgabe. Prozeduren können auch für beide Aufgabenstellungen gleichzeitig eingesetzt werden, die Bestimmung des auszugebenden Ergebnisses kann aber auch nur einen Teil der Tätigkeit darstellen. Das ist eine Frage der inhaltlichen Bedeutung. Für die LOGO-Syntax kommt es zunächst nur darauf an, ob eine Ausgabe erfolgt oder nicht, weil der Umgang mit Prozeduren mit bzw. ohne Ausgabe sich voneinander unterscheidet.

Im Grundwortschatz von LOGO gibt es Wörter, die mit einem P enden. Meistens handelt es sich dabei um Prüfwörter wie memberp, wordp, LISTP, emptyp , numberp, deren Resultat FALSE oder TRUE (falsch oder wahr) ist.

```
wordp [LOGO]          Antwort: FALSE
wordp "LOGO           Antwort: TRUE
numberp "LOGO         Antwort: FALSE
numberp 3.14          Antwort: TRUE
```

Diese LOGO-Vokabeln haben ein Ergebnis, sind also um Prozeduren mit Ausgabe. Das Ergebnis ist aber ganz speziell, da nur die beiden Wahrheitswerte TRUE/FALSE vorkommen. Diese Wörter werden in erster Linie im Zusammenhang mit Bedingungen verwendet, d.h. in Zeilen, die mit if beginnen.

```
if numberp :a [pr [a ist eine Zahl !]]
```

Solche Prüfwörter können auch durch Prozeduren erklärt werden. Als Beispiel soll TEILERP dienen, die überprüft, ob eine natürliche Zahl Teiler einer zweiten ist.

```
to TEILERP :zahl :teiler
op remainder :zahl :teiler = 0
end
```

Diese Prozedur gibt als Ergebnis entweder TRUE oder FALSE aus, je nachdem, ob bei der Division zahl/teiler ein Rest entsteht oder nicht.

Solche selbstdefinierten Prüfwörter werden wie die im LOGO-Grundwortschatz vorhandenen Prüfwörter benutzt. Prozeduren mit dieser Aufgabe spielen eine spezielle Rolle und stellen somit einen dritten Prozedurtyp dar. Wir können demnach drei Typen unterscheiden:

- Prozeduren ohne Ausgabe
- Prozeduren mit Ausgabe
- Prüfwörter, d.h. Prozeduren mit der Ausgabe TRUE bzw. FALSE.

Prozeduren mit Ausgaben

In LOGO kann eine Prozedur nur ein Ergebnis zurückgeben, sie wird dann Funktion genannt. Im Hinblick auf das Rechnen stellt LOGO im Grundwortschatz nur einige mathematische Funktionen zur Verfügung. Reichen diese nicht aus, so können nach Bedarf eigene Funktionen programmiert werden. Auf manchen Taschenrechnern finden Sie etwa eine Funktionstaste zum Quadrieren einer Zahl. Dafür läßt sich schnell eine LOGO-Funktion schreiben.

```
to XQUADRAT :x
op :x*:x
end
```

Ein komplizierteres Beispiel ist die folgende Funktionsprozedur.

```
to HORNER :x :a; Horner-Schema
if emptyp :a [op 0]
op (first :a ) + :x * HORNER :x bf :a
end
```

Aufruf: `HORNER 2.4 [2 3 0 5]`
Antwort: 78.31999999999

Beachten Sie, daß die runden Klammern um (first :a) notwendig sind, damit zuerst first, die Addition danach ausgeführt wird. Die Rechenoperationen haben also gegenüber first - und anderen auf Wörter und Listen wirkende LOGO-Vokabeln - eine höhere Priorität, wenn nicht geklammert wird.
Die Prozedur HORNER berechnet den Wert eines Polynoms nach dem sogenannten Hornerschema. Die Koeffizienten werden unter dem Namen A als Liste übergeben. Bei dem gezeigten Aufruf von HORNER wird folgendes Polynom berechnet:

$$2 + 3*x + 0*x^2 + 5*x^3$$

Für die näherungsweise Berechnung mathematischer Funktionen sind Polynome besonders wichtig.

Weil der Hersteller aus unerfindlichen Gründen die Berechnung der Quadratwurzel nicht im Grundworttschatz berücksichtigt hat, soll hier auch noch eine geeignete Prozedur angeführt werden.

```
to SQRT :x;Quadratwurzel nach Heron/Newton
if :x < 0 [pr [negatives Argument] op :x] [op MITTEL (1 + :x)/2]
end
to MITTEL :w
local "v make "v (:w + :x /:w)/2
if :w - :v < 1.e-03 [op :v] [op MITTEL :v]
end
```

Prozeduren mit mehreren Ergebnissen

Wenn eine Prozedur mehrere Resultate hat, müssen diese in Form einer Liste zusammengefaßt werden, da es der Syntax entsprechend nur ein Objekt als Resultat geben kann. Weil in LOGO im Grunde alles auf der Ebene von Listen verarbeitet wird, können mehrere, von der inhaltlichen Bedeutung aber ganz unterschiedliche Ergebnisse immer als Liste zusammengefaßt werden. Die Beschränkung auf einen einzigen Ausgabewert ist deswegen keine Einschränkung der Programmiermöglichkeiten.

Als kleines Beispiel soll eine Prozedur betrachtet werden, die die Teiler einer natürlichen Zahl bestimmt.

```
to TEILER1 :zahl
(local "tl "t)
make "tl [1] make "t 1
repeat (:zahl - 2) [make "t :t+1 !
if remainder :zahl :t =0 [make "tl lput :t :tl]]
op lput :zahl :tl
end
```

Aufruf: `TEILER1 30`
Antwort: `[1 2 3 5 6 10 15 30]`

Das Programm untersucht die Teilereigenschaft aller natürlichen Zahlen bis zur vorgegebenen Zahl. Schneller arbeitet folgendes Programm:

```
to TEILER :zahl
op TEIL1 :zahl int SQRT :zahl [] []
end
to TEIL1 :zahl :t :tl :tr
if :t=1 [op (se 1 :tl :tr :zahl)]
if remainder :zahl :t = 0 !
[op TEIL1 :zahl (:t-1) (se :t :tl) (se :tr :zahl/:t)]
op TEIL1 :zahl (:t-1) :tl :tr
end
```

Hier sind nur die runden Klammern in der zweiten Zeile von TEIL1 notwendig, die übrigen sind nur zur optischen Zusammenfassung gesetzt. Diese Version macht sich den Umstand zunutze, daß immer ein kleinerer und ein größerer Teiler gleichzeitig gefunden werden. Sie haben sicher bemerkt, daß TEIL1 rekursiv geschrieben ist, TEILER1 dagegen ohne Rekursion arbeitet.

12.3 Das Zusammenwirken von Prozeduren

Eine LOGO-Tätigkeit kann selbst andere LOGO-Tätigkeiten in Gang setzen. Dies geschieht schon beim einfachsten Programm, da zumindest LOGO-Wörter aus dem Grundwortschatz benutzt werden müssen, um überhaupt etwas zu bewirken. Weil die selbstdefinierten LOGO-Wörter, also die Prozeduren, genau wie die des Grundwortschatzes behandelt werden, wird das Prinzip, nach dem Prozeduren zusammenwirken, schon bei den einfachsten Programmen deutlich.

Wird eine Prozedur formuliert, die das LOGO-Wort random benutzt, muß bekannt sein:

- welche Eingaben benötigt die Prozedur random
- welches Resultat liefert random.

Wie random selbst arbeitet, ist für die Verwendung in anderen Programmen unerheblich. Die Verbindungsglieder beim Zusammenwirken von Prozeduren sind demnach Eingaben und Ausgabe, sofern überhaupt eine solche erfolgt. Bei selbstdefinierten Prozeduren gibt es über die globalen Namen (globalen Variablen) möglicherweise noch eine weitere Verbindung der Prozeduren untereinander.

Prozeduren ohne Ausgabe werden genauso benutzt wie LOGO-Vokabeln vom Typ fd oder make. Solche Tätigkeiten haben selbstverständlich ein Resultat, z.B. bestimmte grafische Ereignisse, wirken sich aber nicht auf andere Prozeduren aus. Tätigkeiten, die eine Zahl als Ergebnis liefern, können überall eine Zahl selbst ersetzen. Wo etwa die Zahl 3 verwendet werden

kann, könnte ebenso SQRT 3 wie HORNER 3 [2 5 7 9] stehen. Wenn Prozeduren Eingaben erfordern, dann kann diese auch durch den Aufruf einer LOGO-Tätigkeit ersetzt werden, die selbst ein entsprechendes Resultat liefert.

Dagegen können Elemente von Listen nicht direkt durch Prozeduraufrufe ersetzt werden. Wird etwa beim Aufruf von HORNER eine Koeffizientenliste [2 5] durch [random 10 SQRT 7] ersetzt, besteht die Eingabeliste aus den vier Wörtern random, 10, SQRT und 7, was nicht beabsichtigt ist. Die Eingabeliste muß vielmehr mit se erzeugt werden:

```
HORNER 2 (se random 10 SQRT 7)
```

Beachtet werden muß beim Zusammenwirken von Prozeduren natürlich die Anzahl und der Typ der benötigten Eingaben bzw. der erfolgenden Ausgabe.

Lokale und globale Größen

Bei LOGO-Grundvokabeln kennt man, sofern überhaupt vorhanden, nur die Anforderungen an die Eingabewerte und die Art des Ergebnisses. Was bei der Ausführung der Grundtätigkeit von LOGO intern geschieht, ist für den Anwender unwichtig, er erhält darüber auch keinen Aufschluß. Alle Größen, mit Ausnahme der LOGO "System Variables", die bei der Ausführung der benutzten Prozedur möglicherweise vorkommen, sind nur von lokaler Bedeutung, d.h. nur im Moment der Ausführung intern bekannt.

Am besten ist es, dieses Prinzip auch auf selbstprogrammierte LOGO-Wörter anzuwenden. Dann sind für das Zusammenwirken von Prozeduren nur die Werte für Eingaben bzw. Ausgabe bedeutsam. Alle Größen, die weder Eingabe- noch Ausgabewerte sind, sollten als lokale Größen der jeweiligen Prozedur definiert werden. Hierzu dient die Anweisung local.

Die Eingabewerte von Prozeduren sind Verbindunsglieder zwischen denselben. Die Verbindung besteht nur im Augenblick des Aufrufs einer Tätigkeit. Während des Ablaufs der Prozedur sind Eingabewerte wieder lokal. Beim Aufruf einer Tätigkeit wird sozusagen ein Paket übergeben, dessen weiteres Schicksal nur noch innerhalb der Prozedur bekannt ist, die es erhalten hat. Man kann also innerhalb einer Prozedur Eingabewerte ohne äußere Auswirkungen verändern.

Namen, die weder Eingabewerte sind noch mit Hilfe von local zu lokalen Namen erklärt werden, sind allen Prozeduren als globale Namen bekannt. Welche Namen dieser Art und mit welchem Inhalt sie vorhanden sind, kann man durch PONS erfahren. Es kann von der inhaltlichen Bedeutung her sinnvoll sein, bestimmte Größen mit globalen Namen zu benennen, wenn diese wie z.B. die Kreiszahl PI allgemein für alle möglichen Prozeduren verfügbar sein sollen.

Man kann sich durch Benutzung globaler Namen ersparen, die benötigten Werte als Eingabewerte von Prozeduren zu übergeben. Damit keine Konflikte auftreten, muß man bei jeder zukünftigen Verwendung von LOGO-Wörtern die bereits benutzten globalen Namen im Gedächtnis behalten.

Auch bei der Ausgabe können globale Namen verwandt werden. Anstelle von

```
op 1 + random 6
```

kann auch die Zeile

```
make "A 1 + random 6
```

benutzt werden, wenn A nicht als lokaler Namen erklärt wurde. Statt als Resultat der betreffenden Prozedur kann das Ergebnis als Wert der Größe A anschließend verwendet werden.

Der Einsatz von globalen Namen kann, insbesondere bei mehrfach ineinandergeschachtelten Prozeduren, zu kürzeren Programmen führen. Hier muß man die Vorteile der modularen

Programmiertechnik gegen mögliche Bequemlichkeiten durch knappere Programmtexte gegeneinander abwägen.

12.4 Untersuchung des Programmaufbaus

Programmanalyse mit POCALL

In LOGO arbeiten kleine Programmbausteine zusammen und erfüllen damit kompliziertere Aufgaben. Prozeduren können andere Prozeduren in Gang setzen, die selbst wiederum Prozeduren aufrufen usw. Prozeduren können sich auch selbst aufrufen, sei es unmittelbar oder über mehrere Zwischenstufen. Dieses Zusammenwirken von Prozeduren stellt eigentlich die logische Struktur eines größeren, in LOGO geschriebenen Programms dar.

Ein LOGO-Programm setzt mit LOGO-Vokabeln Tätigkeiten in Gang, wobei es im Prinzip nicht darauf ankommt, ob es sich um selbstdefinierte Vokabeln oder um solche aus dem Grundwortschatz handelt. Für den Umgang mit selbstdefinierten Prozeduren ist neben der inhaltlichen Wirkung zunächst nur die Anzahl und die Bedeutung der Eingaben wichtig. Man muß nicht unbedingt wissen, welche Prozeduren als Folge des Aufrufs indirekt in Aktion gesetzt werden.

Im Verlauf einer Programmentwicklung kann es für einen Programmierer dennoch nützlich sein, den logischen Ablauf auch über alle Ebenen hinweg im Detail zu verfolgen. Manche umfangreicheren LOGO-Versionen stellen für eine solche Analyse des Programmablaufs eigene Vokabeln bereits im Grundwortschatz zur Verfügung. Hier soll in diesem Abschnitt eine kleines Programmpaket vorgestellt werden, das Sie dann nach Bedarf einladen und anwenden können.

Tatsächlich ist es möglich, LOGO-Programme mit Hilfe von LOGO-Programmen zu analysieren. Dazu werden auch Sprachelemente benutzt, die erst in späteren Kapiteln genauer besprochen werden. An solchen Stellen sollten Sie den Standpunkt

eines Anwenders einnehmen, für den es erst einmal auf die Funktionstüchtigkeit des Programms ankommt.

Die Programme haben die Aufgabe, die LOGO-Vokabel

```
POCALL
```

zu realisieren. (Die Bezeichnung orientiert sich übrigens an einer entsprechenden Vokabel in der DR LOGO-Version für den ATARI ST.) Die Vokabel POCALL läßt sich auch auf sich selbst anwenden, sehen wir uns deshalb einfach das Ergebnis von

```
POCALL "POCALL
```

an.

```
Ø : POCALL 1 : ...$ANALYSE
2 : ......$LISTE
3 : .........$LSATZ
2 : ......$ANALYSE*
```

Damit hat die Prozedur POCALL offengelegt, welche Prozeduren bei ihrem Aufruf benutzt werden. Innerhalb von POCALL selbst, also auf der Ebene 1, wird die Prozedur $ANALYSE aufgerufen. Danach wird eine Aufruf von $LISTE auf der Ebene 2 protokolliert, d.h. $LISTE wird von $ANALYSE in Gang gesetzt. $LISTE ruft dann selbst wiederum das Programm $LSATZ, der Aufruf erfolgt deswegen auf Ebene 3. Anschließend wird wieder ein Aufruf auf Ebene 2 festgestellt, d.h. dieser erfolgt von der Prozedur $ANALYSE aus. Es wird dabei das Programm $ANALYSE selbst aufgerufen, es handelt sich also um einen rekursiven Aufruf.

Bei einem Selbstaufruf oder auch einem wechselseitigen rekursiven Aufruf von Prozeduren hat es keinen Sinn, die Analyse der gegenseitigen Aufrufe weiter zu verfolgen, weil das zu einer endlosen Wiederholung führen und ohnehin keine neue Information liefern würde. Der rekursive Aufruf wird stattdessen durch ein angehängtes Sternchen kenntlich gemacht.

An den angehängten *-Zeichen kann man somit leicht herausfinden, wo in einem komplexeren Programm Rekursionen benutzt werden.

Das Programmpaket zur Strukturanalyse

```
to POCALL :p
;ern glist ".APV (local "$funk "$li)
make "$funk glist ".DEF
make "$li []
$ANALYSE :p ∅
end

to $ANALYSE :$ :n
(type :n ": ) repeat :n [type "...] type :$
if memberp :$ :$li [pr "* stop] [pr "]
if not namep :$ [$LISTE :$]
(local "ta "l)
make "ta thing :$
make "l 1 make "$li lput :$ :$li
if emptyp :ta [make "$li bl :$li stop]
repeat count :ta [$ANALYSE item :l :ta :n+1 make "l :l+1]
make "$li bl :$li
end
```

Sicher ist Ihnen aufgefallen, daß im betrachteten Programmpaket viele Namen mit dem $-Zeichen beginnen. Damit sollen möglichst zufällige Übereinstimmungen mit Namen bei den zu analysierenden Programmen vermieden werden.

Zu Beginn der Prozedur POCALL werden die Namen $funk und $li für Listen vereinbart. Mit der Zeile - beachten Sie die Großschreibung von DEF -

```
make "$funk glist ".DEF
```

werden unter dem Namen $funk alle zum Zeitpunkt des Aufrufs definierten Prozedurnamen zusammengefaßt. Die Liste $funk stellt das Reservoir aller Prozeduren dar, die überhaupt in Frage kommen.

In der Liste $li, die in POCALL zunächst nur als leere Liste angelegt wird, werden die im Lauf der Analyse nacheinander als aufgerufene Prozeduren erkannten Namen aufgehoben. An Hand dieser Liste kann dann ein rekursiver Aufruf festgestellt werden.

Das Programm $ANALYSE führt die eigentliche Untersuchung der als Wert :p eingegebenen Prozedur aus, wobei die zweite Eingabe die Ebene des Prozeduraufrufs angibt. Immer dann, wenn $ANALYSE aufgerufen wird, ist ein Programmaufruf identifiziert worden, der dann sogleich zu Protokoll gegeben wird. Falls der betreffende Prozedurname bereits in der Liste $li enthalten ist, liegt ein rekursiver Aufruf vor, dessen Auftreten mit der Ausgabe des *-Zeichens und der Beendigung der weiteren Analyse beantwortet wird.

In der vierten Zeile wird mit der Abfrage

```
namep :$
```

überprüft, ob der eingegebene Prozedurname als Name mit einem Wert verbunden ist. Die Logo-Vokabel namep antwortet mit TRUE, wenn das eingegebene Wort eine bereits definierte Variable bezeichnet. Geben Sie etwa ein

```
pr :unbekannt,
```

so wird LOGO mit der Fehlermeldung "unbekannt has no value" reagieren. Daß diese Meldung erscheint, läßt sich an der Antwort FALSE bei der Anwendung des Prüfworts namep schon erkennen

```
namep "unbekannt      Antwort: FALSE
```

Wozu dient nun die Abfrage von :$ mit namep? Wenn der Aufruf einer Prozedur erkannt worden ist, muß die betreffende Prozedur selbst untersucht werden. Dieser Aufgabe dient die Prozedur $LISTE, denn $LISTE erzeugt zu einer gegebenen Prozedur eine Liste, welche selbst wiederum die Namen der dort aufgerufenen Prozeduren enthält. Die Liste der vorkommenden

Prozedurnamen wird dabei einem Namen als Wert zugewiesen, der mit dem Prozedurnamen übereinstimmt. Probieren Sie aus:

```
$LISTE "$ANALYSE pr :$ANALYSE
```

Antwort: `[$LISTE $ANALYSE]`

Beachten Sie, daß der Name $ANALYSE jetzt sowohl Prozedurname als auch Name einer Variablen ist, die gerade die in der Prozedur gleichen Namens gerufenen Programmnamen beinhaltet. Die gleichzeitige Verwendung eines Namens sowohl für eine Prozedur als auch für eine Variable ist in LOGO demnach zulässig. Um unbeabsichtige Verwechslungen zu vermeiden, sollte man von dieser Möglichkeit aber mit Vorsicht Gebrauch machen. Die Übereinstimmung hat auch zur Folge, daß Sie nun beim Aufruf des Editors hinter dem Programmtext die Zeile

```
make "$ANALYSE [$LISTE $ANALYSE]
```

finden.

Mit der Abfrage namep :$ wird demnach festgestellt, ob bereits bekannt ist, welche Prozeduren im angesprochenen Programm gerufen werden. Wenn das nicht der Fall ist, d.h. die zur Debatte stehende Prozedur im Verlauf der Analyse noch nicht vorgekommen ist, dann wird durch den Aufruf von $LISTE eben diese Liste erzeugt.

Diese Liste wird nun in der sechsten Zeile von $ANALYSE dem Namen ta zwischendurch zugewiesen, weil die Liste mehrfach benötigt wird. Dazu wird die LOGO-Vokabel

```
thing
```

benutzt. Würde die Zeile nämlich stattdessen

```
make "ta :$
```

lauten, so würde unter ta der Name der Liste, hier etwa $ANALYSE, nicht aber die gewünschte Liste selbst bereit

stehen. Man könnte sich vorstellen, daß zu diesem Zweck ein zweiter Doppelpunkt angebracht wäre, was aber gegen die LOGO-Syntax verstoßen würde.

```
thing :$ bedeutet also: Wert von :$
```

In der siebten Zeile wird schließlich der gerade untersuchte Programmname mit lput in die Liste der bisher gerufenen Prozeduren eingereiht.

Nun können Programme nicht über beliebig viele Ebenen hinweg andere Programme aufrufen. Ob eine Prozedur selbst keine anderen Prozeduren mehr aufruft, ist daran zu erkennen, daß $LISTE eine leere Liste erzeugt. In diesem Fall wird in der achten Zeile der rekursive Aufruf von $ANALYSE beendet. Ruft eine Prozedur dagegen andere auf, so muß für jeden Prozeduraufruf das Programm $ANALYSE in der neunten Zeile rekursiv aufgerufen werden.

Ist diese Analyse abgeschlossen, so wird in der letzten Zeile die betrachtete Prozedur aus der Buchhaltungsliste $li wieder gestrichen.

```
to $LISTE :$t
(local "p "l)
make "p $LSATZ text :$t
make "l 1 make "$t []
repeat count :p [make "a item :l :p if memberp :a :$funk !
[make :$t se thing :$t :a] make "l :l+1]
end

to $LSATZ :$
local "l$ make "l$ []
label "ANF if emptyp :$ [op :l$]
if listp first :$ [make "$ se first :$ bf :$ go "ANF]
make "l$ se :l$ first :$
make "$ bf :$ go "ANF
end
```

Die Aufgabe der Prozedur $LISTE, die $LSATZ als Hilfprogramm verwendet, ist schon besprochen worden.

In der dritten Zeile wird zunächst der gesamte Text der Prozedur mit

```
text :$t
```

als Liste erzeugt und diese Liste wird dem Hilfsprogramm $LSATZ übergeben. Die LOGO-Vokabel text wird später noch ausführlicher behandelt. Damit wird der Programmtext unter dem Namen einer Variablen verfügbar und kann damit selbst durch Programme verarbeitet werden. Aus dem Beispielaufruf POCALL "POCALL wird ersichtlich, daß Prozeduren auch den eigenen Programmtext bearbeiten können.

Probieren Sie einmal aus

```
text "POCALL
```

Der Programmtext wird als Liste zurückgegeben, deren Elemente jeweils eine Programmzeile in der Form einer Liste darstellen. Das erste Element hat die Namen von Eingabeparametern zum Inhalt. Weil bei LOGO-Vokabeln wie repeat und if auch wieder Listen vorkommen, können auch einzelne Programmzeilen selbst wieder mehrfach verschachtelte Listen sein.

Für die Suche nach Prozedurnamen im Programmtext ist diese Form nicht geeignet, weil die LOGO-Vokabel memberp erst mit dem gewünschten Effekt auf Sätze angewandt werden kann. Deshalb wird mit der Prozedur $LSATZ der Programmtext zunächst einmal in die Form eines Satzes, d.h. einer Liste von Wörtern gebracht. Die Prozedur beseitigt Verschachtelungen bei Listen und kann für diesen Zweck auch auf andere Listen angewandt werden.

```
$LSATZ [Dies [war [eine [verschachtelte]] Liste]]
```

Ergebnis: `[Dies war eine verschachtelte Liste]`

Die Beseitigung der Verschachtelung bei Listen geschieht in $LSATZ mit den beiden LOGO-Vokabeln

```
listp und se
```

Dabei stellt das Prüfwort listp fest, ob die Eingabe eine Liste ist. Die Liste wird mit Hilfe von first und bf zerlegt und anschließend mit se wieder zusammengesetzt, wobei jeweils äußere eckige Klammern beseitigt werden. Wenn Sie diesen Prozeß im Detail verfolgen wollen, sollten Sie vor dem Aufruf der Prozedur einmal den Ablaufverfolger mit trace einschalten.

Abschließende Bemerkungen

Beim Beispiel der Selbstanalyse POCALL "POCALL werden die im Programm $ANALYSE gerufenen Prozeduren in der globalen Variablen $ANALYSE abgelegt. Wird POCALL mehrfach benutzt, wird diese Liste nicht neu erzeugt, wenn sie bereits vorhanden ist. Bei nachfolgenden Programmänderungen kann es im Prinzip vorkommen, daß Prozeduraufrufe verschwinden oder hinzukommen, was dann aber bei weiteren Aufrufen von POCALL nicht bemerkt wird.

Wenn ein solcher Fall auftritt, löscht man am besten vor dem Aufruf von POCALL alle globalen Namen mit der Zeile

```
ern glist ".APV
```

Diese Zeile ist oben als Kommentarzeile am Anfang der Prozedur POCALL selbst zu finden. Weil dabei möglicherweise auch andere Namen unbeabsichtigt gelöscht werden, wurde diese Zeile durch das Semikolon vorsichtshalber außer Kraft gesetzt.

Zum Schluß sei noch verraten, daß man mit später noch zu diskutierenden Methoden während des Programmablaufs Programmnamen aufbauen, ja sogar Prozeduren neu definieren darf. Bei solch raffinierten Anwendungen ist es überhaupt nicht möglich, anhand des Programmtextes alle tatsächlich vorkommenden Programmaufrufe vorher festzustellen.

12.5 Sprünge über mehrere Ebenen

Im Regelfall werden Prozeduren mit stop oder op bzw. dem Erreichen des Prozedurendes verlassen. Das LOGO-System kehrt damit zum rufenden Programm zurück. Gelegentlich, beispielsweise bei speziellen Eingaben, will man ohne weitere Aktionen zum Direktbetrieb zurückkehren. Dafür kann throw "TOPLEVEL benutzt werden. Man erspart sich so in allen rufenden Prozeduren Maßnahmen für einen Sonderfall vorzusehen.

Bei der Rückkehr zum Direktbetrieb muß das LOGO-System selbst die notwendigen Schritte zum Abschluß der durchlaufenen Prozeduren übernehmen. Es sollte deshalb auch möglich sein, nicht vollständig zum Direktbetrieb, sondern nur um einige Ebenen zurückzugehen.

Ein Sprung innerhalb einer Prozedur kann mit go ausgeführt werden. Der Rücksprung in eine rufende Prozedur über mehrere Ebenen des Prozeduraufrufs hinweg wird durch throw bewerkstelligt. Wie go braucht auch throw ein Sprungziel. Ohne Vereinbarung kennt LOGO das Sprungziel TOPLEVEL. Um selbst Sprungziele für throw zu definieren, wird die LOGO-Vokabel catch eingesetzt, die für throw das Gegenstück ist wie LABEL für go.

```
to DEMO
catch "ziel [EBENE2]
pr [Sprungziel in Demo ist erreicht]
end

to EBENE2
pr [EBENE2 ist erreicht]
EBENE3 pr [in EBENE2, zurueck aus EBENE3]
end
```

```
to EBENE3
pr [Jetzt auf EBENE3]
pr [Soll ich zu Demo zurueck?]
if lc first rq ="j [throw "ziel]
pr [Soll ich sofort abbrechen?]
if lc first rq ="j [throw "TOPLEVEL]
pr [Ich mache weiter!]
end
```

Mit diesem Beispiel sollen throw und catch nur demonstriert werden. Wenn Sie die Frage in EBENE3 mit ja beantworten, erscheint als nächstes die Meldung "Sprungziel erreicht". Bei negativer Antwort wird gefragt, ob aus EBENE3 der direkte Abbruch erfolgen soll. Falls Ihre Antwort nicht ja ist, gibt EBENE2 die Rückkehr aus EBENE3 bekannt. Auch in diesem Fall wird nach Beendigung von EBENE2 das Sprungziel erreicht, denn dieses ist die nächste Anweisung nach dem Aufruf von EBENE2.

Das Sprungziel TOPLEVEL (Großschreibung beachten!) ist intern im LOGO-System vorhanden. LOGO kehrt danach in den Direktbetrieb zurück, ohne in die rufenden Prozeduren zurückzukehren, wie das bei stop der Fall ist. Das throw "TOPLEVEL wirkt sich so ähnlich wie ein Abbruch mit der Stoptaste aus, ohne daß dabei allerdings eine spezielle Meldung auf dem Bildschirm erscheint.

catch hat zwei Eingaben. Die erste ist ein Wort als Bezeichnung des Sprungziels, während die zweite eine Liste mit ausführbaren Kommandos darstellt.

Beachten Sie: Der Sprung zu dem in der catch-Anweisung genannten Ziel ist nur von Prozeduren aus möglich, die von den in der Eingabeliste aufgeführten Prozeduren gerufen wurden! Das Ziel muß mit anderen Worten in einer Prozedur gefunden werden, die bei der Rückkehr zum Direktbetrieb durchlaufen wird. Bezogen auf diesen Pfad ist der Name des Sprungziels lokal, d.h. außerhalb nicht bekannt. Er kann deshalb in verschiedenen Pfaden ohne Konflikt mehrfach benutzt werden.

12.6 Programmierte Fehlerbehandlung

Bei einem Fehler erscheint im Normalfall eine Fehlermeldung auf dem Textschirm, danach kehrt LOGO in den Direktbetrieb zurück. Sie können dann durch Eingabe von

```
ed
```

sofort in den Editor gelangen, meist befindet sich der Cursor dann an der Fehlerquelle.

Diese Reaktion auf Fehler ist in der Praxis sehr bequem. LOGO ist eine typische Sprache für einen Interpreter. Tritt ein Fehler auf, kann das fehlerhafte Programm korrigiert und mit der Ausführung neu begonnen werden. Auch falsche Werte, die einen Fehler hervorrufen, können vor einem weiteren Versuch geändert werden. Globale Namen können sogar mit dem Editor bearbeitet werden, was bei komplizierteren Daten sehr viel einfacher als eine Neubesetzung mit make ist.

Etwas anders sieht das für den Anwender eines LOGO-Programms aus, der LOGO selbst nicht beherrscht. Gute Anwenderprogramme sollten natürlich selbst keine Fehler enthalten, sie müssen aber mit falschen Eingaben des Anwenders fertig werden. Dafür können die vorhandenen Möglichkeiten genutzt werden, um einen Fehler abzufangen.

Wenn ein Fehler auftritt, versucht das LOGO-System von sich aus die Anweisung throw "error, also einen Rücksprung zu einem catch "error durchzuführen. Wenn dieses Sprungziel tatsächlich im Programm vorgesehen ist, wird der Sprung auch ausgeführt und es gibt keine Fehlermeldung Dabei muß sich catch "error in einer Prozedur befinden, die bei der Rückkehr zum Direktbetrieb durchlaufen wird, genauso wie bei der throw-Anweisung sonst. catch "error darf aber in verschiedenen Pfaden gleichzeitig vorgesehen werden.

Bei den meisten sinnvollen Anwendungen ist die Fehlerquelle bereits beim Programmieren klar. Man kann aber auch mit Hilfe der Funktion error die Meldung über den zuletzt aufgetretenen Fehler als Liste bekommen. Probieren Sie einfach aus:

```
show error
```

Die nach dem catch "error folgende Zeile kann auch durch reguläre Beendigung der in der Anweisungsliste aufgeführten Kommandos erreicht werden. Welcher Fall vorliegt, kann bei Bedarf durch eine Abfrage festgestellt werden.

```
if emptyp error [...]
```

Dabei ergibt sich ohne Fehler der Wahrheitswert TRUE.

Die Prozedur SPEICHERN stellt eine einfache Anwendung dar. Beim Abspeichern auf Diskette mit Hilfe von save können vorhandene Files nicht überschrieben werden. Wenn der Filename schon vorhanden ist, gibt es eine entsprechende Fehlermeldung, die sich nun mit catch "error abfangen läßt.

```
to SPEICHERN :file
catch "error [save :file]
if emptyp error [stop]
pr [File] :file [existiert bereits!])
pr [Alten File löschen? j/n]
if lc rc ="j [erasefile :file save :file] !
[pr [Ohne Speichern abgebrochen!]]
end
```

Wenn der Filename bereits existiert, gibt es bei einem Versuch mit save normalerweise eine Fehlermeldung, auf die man dann natürlich im Direktbetrieb reagieren kann. Mit der Prozedur SPEICHERN werden programmgesteuerte Reaktionen angeboten. Auf Wunsch wird der alte File gelöscht und der Inhalt des Arbeitsspeichers wird nun abgespeichert.

Häufig wird bei gutem Programmsystemen so vorgegangen, daß der bereits vorhandene File unter einem anderen Namen aufgehoben wird, damit bei einem Versehen der Inhalt noch verfügbar ist. Das ist im Prinzip mit DR LOGO auch möglich, denn im Handbuch finden Sie die LOGO-Vokabel

```
changef
```

zum Umbenennen von Files. Leider funktioniert der Befehl nicht in den gegenwärtig vorliegenden LOGO-Systemen auf den Schneidercomputern: es erscheint stets die Fehlermeldung File ... not found.

Übungen

1) Benutzen Sie POCALL für Programme aus vorangegangenen Lektionen.

2) Formulieren Sie ein Prüfwort VOKALP, das TRUE bzw. FALSE als Ergebnis ausgibt, je nachdem, ob das eingegebene Zeichen eine Vokal ist oder nicht.

3) Die Prozedur soll aus dem eingegebenen Wort die Vokale heraussuchen und diese als Liste ausgeben.

4) Eingabe soll eine Liste von Zahlen sein. Es soll die Summe dieser Zahlen berechnet und als Ergebnis zurückgegeben werden.

Lektion 13: Rekursion

Neue Sprachelemente:
Rekursion mit Selbstaufruf am Ende, Ablauf der Rekursion mit Selbstaufruf in der Prozedurmitte

Programme:
ANORDNUNGEN (Fakultät), TIPS (Binomialkoeffizient), GGT, KGV (Euklid-Algorithmus), TETRA, BAUM (grafische Rekursion), TURM (Turm von Hanoi mit Hilfsprozeduren), STRATEGIE (iterative Version der Türme von Hanoi)

13.1 Rekursiver Aufruf am Prozedurende

Programme mit Rekursion sind für LOGO typisch, weshalb rekursive Programme auch schon früh eingeführt wurden. In dieser Lektion soll die Rekursion systematisch betrachtet werden.

Im einfachsten Fall dient der Selbstaufruf nur zu einer Wiederholung ohne jede Veränderung. Ein grafischer Grundschritt wie das Zeichnen eines Winkels kann damit beliebig of wiederholt werden. Vergleichen Sie nun zwei Programme, die das mit gleichem Ergebnis ausführen.

```
to WINKEL1
fd 60 rt 100
WINKEL1
end

to WINKEL2
label "anfang fd 60 rt 100 go "anfang
end
```

Die Wiederholung des Grundschritts aus einer Vorwärtsbewegung und einer Rechtsdrehung wird in WINKEL1 durch Selbstaufruf, in WINKEL2 durch die Sprunganweisung GO

erreicht. WINKEL1 ist eine rekursive, WINKEL2 eine iterative Formulierung für dieselbe Aufgabenstellung.

Beim Selbstaufruf können Eingabegrößen verändert werden. Die Änderung von Größen bei Wiederholungen läßt sich mittels Rekursion besonders einfach formulieren:

```
to WINKEL3 :winkel :laenge
fd :laenge rt :winkel
WINKEL3 :winkel :laenge+1
end

to WINKEL4 :winkel :laenge
label "anfang fd :laenge rt :winkel
make "laenge :laenge + 1
go "anfang
end
```

Das erste Programm ist wieder eine rekursive, das zweite eine iterative Lösung für die gleiche Aufgabe. Der unterschiedliche Ablauf kann mit dem Trace- oder auch mit dem Watch-Modus von LOGO sichtbar gemacht werden. Bei der iterativen Formulierung WINKEL4 wird im Trace-Betrieb nur die Veränderung von laenge durch make protokolliert, bei der rekursiven Variante WINKEL3 der fortlaufende Selbstaufruf mit den aktuellen Eingabewerten.

Was bei einer Rekursion geschehen soll, ist einfach zu ersehen: am Ende von WINKEL3 soll erneut der Schenkel eines Winkels gezeichnet werden, dessen Länge um eins größer sein soll. Bei der iterativen Wiederholung in WINKEL4 muß man sich erst klar machen, wie die Wiederholung abläuft, bevor man deren inhaltliche Aufgabe richtig versteht. Das bereitet bei diesem einfachen Beispiel zwar kaum Schwierigkeiten, bei komplizierteren Aufgabenstellungen kann es aber sehr schnell schwierig werden, die inhaltliche Bedeutung von Wiederholungen zu erfassen.

Bei der Veränderung mit make muß man sich darüber klar werden, daß hier der Inhalt einer Schublade (laenge) herausgenommen, ein Ergebnis damit berechnet und dieses wieder in derselben Schublade abgelegt wird.

Bei der Rekursion kann es andererseits schwierig werden, wenn man den internen Ablauf verstehen will. Was geschieht im Rechner, wenn ein Programm sich selbst aufruft? Beißt sich die Katze dabei nicht selbst in den Schwanz? Solche Fragen stellen sich dem LOGO-Anwender, der längere Praxis mit Programmiersprachen ohne Rekursion hat. LOGO ist so konzipiert, daß der Programmierer sich im Grunde genommen nicht um interne Vorgänge des Rechners kümmern muß. Im LOGO-Programm wird formuliert, was inhaltlich geschehen soll, den Rest übernimmt das LOGO-System. Für die Formulierung der inhaltlichen Ziele sind rekursive Lösungen oftmals direkter an der Problemstellung orientiert, während iterative meist besser auf die interne Bearbeitung im Rechner zugeschnitten und deshalb schneller in der Ausführung sind.

Die einfache Rekursion, bei der der rekursive Aufruf die letzte Anweisung der Prozedur darstellt und diese keine Eingaben besitzt, erfolgt in LOGO aber mit relativ großer Geschwindigkeit und sie kann auch beliebig oft durchlaufen werden. Im Watch- oder Trace-Modus zeigt DR LOGO dabei immer das niedrigste Niveau [1] an.

Hat die Prozedur Eingaben, und zwar unabhängig davon, ob diese bei der Rekursion verändert werden oder nicht, so bereitet die Rekursion schon mehr Mühe. Im Watch- bzw. Trace-Modus wird nun nach dem rekursiven Aufruf jeweils ein höheres Niveau angezeigt ([2], [3], ...). Bei einem Vergleich wird deutlich, daß die Rekursion in der vorliegenden Version nicht so effizient abgewickelt wird, wie das bei anderen LOGO-Versionen auf vergleichbaren Rehnern der Fall ist.

13.2 Rekursive Aufrufe vor dem Prozedurende

Bei der Rekursion, die nur zur Wiederholung dient, erfolgt der Selbstaufruf als letzte Tätigkeit innerhalb der Prozedur. Folgen darauf noch weitere Programmzeilen, so ist der Ablauf des Programms bei der Rekursion schwieriger zu durchschauen. Wie kann eine Tätigkeit sich selbst erneut in Gang setzen, wenn sie noch nicht abgeschlossen worden ist?

Betrachten Sie ein einfaches Beispiel, bei dem nur Wiederholungen abgezählt werden:

```
to ZAEHLEN1 :zahl
if :zahl > 5 [stop]
pr :zahl
ZAEHLEN1 :zahl + 1
end

to ZAEHLEN2 :zahl
if :zahl > 5 [stop]
ZAEHLEN2 :zahl + 1
pr :zahl
end
```

In beiden Programmen soll bis 5 gezählt werden. Damit die Wiederholung abbricht, ist eine Abbruchbedingung mit Hilfe von if ... [stop] in beiden Beispielen vorgesehen. Bei ZAEHLEN1 ist der Selbstaufruf wieder die letzte Anweisung der Prozedur, bei ZAEHLEN2 folgt noch die Druckanweisung.

Nach dem Aufruf von

```
ZAEHLEN1 1
```

wird der Anfangswert hochgezählt und als Ergebnis wird 1,2,3,4,5 ausgegeben. Ohne die Programmzeile mit if würden jetzt bis zum manuellen Abbruch von außen die natürlichen Zahlen der Reihe nach ausgedruckt werden. Nach dem Aufruf von

```
ZAEHLEN2 1
```

erscheint dagegen 5,4,3,2,1 auf dem Bildschirm. Die Ursache für die umgekehrte Reihenfolge muß an der veränderten Stellung der Ausgabeanweisung pr :zahl liegen.

Was geschieht, wenn die Abbruchbedingung mit if aus der Prozedur ZAEHLEN2 herausgenommen wird? Am einfachsten kann man eine Zeile zeitweilig aus der Programmausführung herausnehmen, indem man sie zum Kommentar macht. Setzen Sie also an den Anfang der Zeile ein Semikolon. Nun sieht man beim Aufruf von ZAEHLEN2 zunächst einmal nichts; bald meldet LOGO einen Abbruch mit

```
Out of LOGO stack space (LOGO-Kellerspeicher erschöpft)
```

Der Rekursionsaufruf läßt sich also nicht beliebig oft wiederholen; das LOGO-System benötigt hier offensichtlich Speicherplatz, den sogenannten Stack oder Kellerspeicher, da es nur einen gewissen Bereich vorgesehen hat.

Auch bei Zaehlen1 wird der Kellerspeicher benötigt und die Wiederholung kann nicht beliebig oft erfolgen. In diesem Fall ist das aber eine unnötige Schwäche der LOGO-Version für die Schneidercomputer, denn beim Rekursionsaufruf am Prozedurende muß eigentlich nichts für später aufgehoben werden.

Der unterschiedlichen Ablauf von ZAEHLEN2 und ZAEHLEN1 kann wieder mit dem Ablaufverfolger, am kürzesten nur mit watch, untersucht werden.

```
watch
ZAEHLEN1 3
[1] In ZAEHLEN1, if :zahl > 5 [stop]
[1] in ZAEHLEN1, pr :zahl - Antwort: 3
[1] In ZAEHLEN1, ZAEHLEN1 :zahl+1
[2] In ZAEHLEN1, if :zahl > 5 [stop]
[2] in ZAEHLEN1, pr :zahl - Antwort: 4
[2] In ZAEHLEN1, ZAEHLEN1 :zahl+1
[3] In ZAEHLEN1, if :zahl > 5 [stop]
[3] in ZAEHLEN1, pr :zahl - Antwort: 5
[3] In ZAEHLEN1, ZAEHLEN1 :zahl+1
[4] In ZAEHLEN1, if :zahl > 5 [stop]

ZAEHLEN2 3
[1] In ZAEHLEN2, if :zahl > 5 [stop]
[1] In ZAEHLEN2, ZAEHLEN2 :zahl+1
[2] In ZAEHLEN2, if :zahl > 5 [stop]
[2] In ZAEHLEN2, ZAEHLEN2 :zahl+1
[3] In ZAEHLEN2, if :zahl > 5 [stop]
[3] In ZAEHLEN2, ZAEHLEN2 :zahl+1
[4] In ZAEHLEN2, if :zahl > 5 [stop]
[3] in ZAEHLEN2, pr :zahl - Antwort: 5
[2] in ZAEHLEN2, pr :zahl - Antwort: 4
[1] in ZAEHLEN2, pr :zahl - Antwort: 3
```

In beiden Versionen wird zunächst schrittweise die nächste Ebene im Prozeduraufruf erreicht. In ZAEHLEN1 wird vor dem Selbstaufruf die Druckzeile ausgeführt. Dementsprechend erscheinen die ausgegebenen Zahlen in aufsteigender Reihenfolge. Nachdem die Abbruchbedingung erfüllt ist, wird die Ausführung von ZAEHLEN1 in einem Schritt beendet.

Zum Abarbeiten von ZAEHLEN2 wird bei der Rekursion die Ausführung der gerade gültigen Ebene unterbrochen und zur nächsten Ebene des Prozeduraufrufs übergegangen. Nachdem die Abbruchbedingung erreicht ist, findet nun ein schrittweises Zurückgehen in die Ausgangsebene von ZAEHLEN2 statt.

Die Druckzeile wird dann dreimal hintereinander ausgeführt und protokolliert, zuerst die in der Ebene vor dem Erreichen der Abbruchbedingung.

Wie an der zu Anfang der Protokollzeile in eckige Klammern eingeschlossenen Kennziffer zu erkennen ist, wird diese bei ZAEHLEN2 bei jedem Aufruf erhöht und anschließend beim Zurückgehen auf die Ausgangsebene reduziert. Bei ZAEHLEN1 geschieht das nicht.

Um den Wert von zahl an dieser Stelle ausgeben zu können, muß der auf der jeweiligen Ebene gültige Wert verfügbar sein. Das kann nur geschehen, wenn LOGO bei der Unterbrechung der Prozedur ZAEHLEN2 durch den Selbstaufruf den gültigen Wert für die spätere Verwendung aufgehoben hat. Im betrachteten einfachen Beispiel handelt es sich nur um einen Namen, im allgemeinen müssen alle lokalen Namen bei der Unterbrechung der Prozedurausführung in einem dafür vorgesehenen 'Keller' (Stack) eingelagert werden. Ist der für diese Zwecke vorgesehene Speicherplatz erschöpft, kann die Rekursion nicht mehr weiter durchgeführt werden, das LOGO-System reagiert mit der Meldung Out of LOGO stack space!

Die umgekehrte Reihenfolge, nämlich die Ausgabe der Zahlen in ZAEHLEN1 und ZAEHLEN2, hat ihre Ursache darin, daß in ZAEHLEN1 die Druckzeile beim Vorwärtsschreiten in der Rekursion ausgeführt wird, in ZAEHLEN2 dagegen die Zahlen beim Zurückgehen auf die vorangegangene Ebene des Prozeduraufrufs gedruckt werden.

13.3 Rekursive Funktionen

Anordnung verschiedener Objekte

Wieviele Möglichkeiten gibt es, elf verschiedene Bücher auf einem Regal anzuordnen? Die Antwort lautet: 39916800. Lehrreicher ist die folgende Anwort: Es gibt elfmal mehr Möglichkeiten elf Bücher anzuordnen als bei zehn Büchern. Bei jeder möglichen Anordnung von zehn Büchern kann das elfte Buch

vor das 1., 2., ..., 10. und hinter das 10. gesetzt werden, jede Anordnung von zehn Büchern gibt elf Möglichkeiten für das letzte Buch.

Diese Antwort liefert zwar noch keine Zahl, aber Einsicht in die allgemeine Problemlösung für eine beliebige Zahl von Objekten. Sie führt unmittelbar zu einer rekursiven Lösung:

> Um die Zahl der möglichen Anordnungen von n Objekten zu bestimmen, muß die Zahl der Anordnungen für (n-1) Objekte berechnet werden.

```
to ANORDNUNGEN :n
op :n * ANORDNUNGEN :n-1
end
```

Bei der gegebenen Problemstellung soll ein Wert als Ergebnis berechnet werden, weshalb die entsprechende Prozedur durch op einen Wert ausgeben muß. Der Prozeduraufruf muß dann in einer Programmzeile dort stehen, wo ein Wort stehen kann, etwa anstelle einer Zahl oder für den Wert eines Namens. Auch die rekursiven Prozeduren mit Ausgabewert müssen so benutzt werden. Die Prozedur ANORDNUNGEN wird in der Zeile rekursiv aufgerufen, in der die Ausgabe mit op erfolgt, der Selbstaufruf gehört also zur Eingabe von op.

Die Prozedur Anordnungen ist noch nicht komplett, denn die Rekursion findet noch kein Ende. Tatsächlich antwortet LOGO bei dieser Version rasch mit der Fehlermeldung Out of LOGO stack space! Obwohl der Selbstaufruf am Ende steht, handelt es sich nicht um die einfache Form der Rekursion, die beliebige Wiederholungen zuläßt. Der Grund ist leicht einzusehen: Das Ergebnis, hier die Zahl der Anordnungen, muß für die Multiplikation aufgehoben werden, weil diese ja erst ausgeführt werden kann, wenn die Rekursion ein Ende gefunden hat.

Für die rekursive Berechnung eines Wertes ist eine Abbruchbedingung erforderlich. Weil ein Ergebnis benötigt wird, muß beim Abschluß der Rekursion wirklich ein Ausgabewert vorhanden sein. Eine rekursive Berechnung kann nur funktionieren,

wenn ein spezieller Wert bekannt ist, mit dem die Rekursion beendet werden kann. Im betrachteten Fall kann sie bei Erreichen eines einzigen Objekts abgebrochen werden, dann gibt es natürlich auch nur eine Möglichkeit zur Anordnung.

```
to ANORDNUNGEN :n
if :n=1 [op 1]
op :n * ANORDNUNGEN :n-1
end
```

Die hier erklärte Funktion Anordnungen wird in der Mathematik mit Fakultät bezeichnet.

Wieviele Möglichkeiten gibt es beim Lotto?

Beim Zahlenlotto 6 aus 49 können 13983816 verschiedene Tips abgegeben werden. Diese Zahl läßt sich rekursiv mit folgender Überlegung bestimmen:

Zunächst werden nur fünf Zahlen ausgewählt, dann stehen für die sechste Zahl noch 44=49-5 Möglichkeiten offen. Danach sollte es (49-5)mal mehr Möglichkeiten für die Auswahl von sechs Zahlen geben als für die Auswahl von fünf Zahlen. Tatsächlich gibt es aber weniger Möglichkeiten, weil ein bestimmter Sechsertip aus mehreren Fünfertips hervorgehen kann. Der TIP (1,5,13,24,35,46) kann aus allen Fünfertips erzeugt werden, die durch Streichung einer der sechs Zahlen entstehen. Es gibt also genau sechs derartiger Fünfertips. Daraus folgt:

Es gibt (49-5)/6mal mehr Sechsertips als Fünfertips, (49-4)/5mal mehr Fünfertips als Vierertips usw. Das läßt sich in eine geeignete LOGO-Vokabel fassen.

```
to TIPS :n
if :n=1 [op 49];49 Mögl. für einen Einertip
OP (50 * :n)/:n * TIPS :n - 1
end
```

Probieren Sie:

```
TIPS 6
```
Antwort: 13983816

Die Prozedur läßt sich leicht für ein beliebiges Lotto n aus m verallgemeinern:

```
to TIPS2 :n :m
if :n=1 [op :m];M Mögl. für einen Einertip
OP (:m+1-:n)/:n * TIPS :n-1 :m
end
```

13.4 Weitere Beispiele zur Rekursion

Polynomberechnung

Eine rekursiv definierte Funktion ist zum Beispiel die in Lektion 10 betrachtete Berechnung von Werten eines Polynoms nach dem Hornerschema.

```
to HORNER :x :a
if emptyp :a [op 0]
op (first :a) + :x * HORNER :x bf :a
end
```

Die Rekursion bezieht sich hier auf die Liste der Koeffizienten A. Das Ergebnis wird aus dem Wert des Polynoms berechnet, das durch Weglassen des ersten Koeffizienten entsteht.

GGT und KGV: Euklidscher Algorithmus

GGT: Größter gemeinsamer Teiler
KGV: Kleinstes gemeinsames Vielfaches

Vielleicht rufen die Bezeichnungen bei Ihnen unangenehme Erinnerungen an die Bruchrechnung hervor. Mit LOGO wird das ganz einfach! Das nach Euklid benannte Verfahren gestattet die Bestimmung des größten gemeinsamen Teilers zweier Zahlen auf

elegante Weise. Man nimmt die größere der beiden Zahlen und bestimmt den Divisionsrest beim Teilen durch die kleinere:

$$54=12*4 + 6$$

Der Divisionsrest und die kleinere Zahl haben den gleichen GGT wie die Ausgangwerte, das Paar (12,6) enthält aber kleinere Zahlen als (54,12). Das Verfahren endet, wenn kein Rest mehr übrig bleibt. Dieser Algorithmus läßt sich rekursiv sehr einfach formulieren.

```
to GGT :a :b
if :b = 0 [op :a]
op GGT :b remainder :a :b
end
```

Die Prozedur funktioniert auch dann, wenn die erste Zahl kleiner als die zweite ist. Dann werden beide beim ersten Schritt gerade miteinander vertauscht. Das kleinste gemeinsame Vielfache ergibt sich dann durch:

```
to KGV :a :b
op :a * :b / GGT :a :b
end
```

Grafische Rekursionen: Bäume

```
to DREIECK :seite
rt 150 fd :seite * 1.732
rt 150 fd :seite
rt 180
end
```

Probieren Sie aus:

```
fd 80 DREIECK 80
```

Es entsteht ein gleichschenkliges Dreieck. Wiederholen Sie nun diese Zeile noch zweimal! Es entstehen zwei weitere deckungsgleiche Dreiecke, die zusammen den Eindruck eines Tetraeders ergeben.

Die Prozedur DREIECK zeichnet nur zwei Dreiecksseiten. Dahinter steckt natürlich eine spezielle Absicht. Nachdem die erste Dreiecksseite gezeichnet ist, kann eine Unterbrechung stattfinden und an dem gerade erreichten Punkt eine andere Figur dargestellt werden. Hier soll das Verfahren zur Projektion eines Tetraeders rekursiv benutzt werden.

```
to TETRA :seite :e
if :e = 0 [stop]
repeat 3 [fd :seite TETRA :seite/1.9 :e-1 DREIECK :seite]
end
```

Die Größe e gibt die Ebene der Verschachtelung an. Probieren Sie:

```
cs ht bk 30 TETRA 90 4
```

Ein einfaches Programm, das die Grafik in Abb. 13.1 erzeugt!

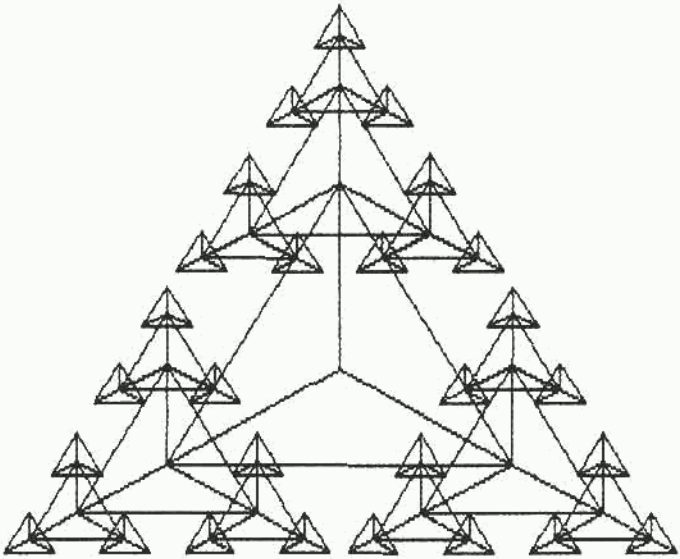


Abb. 13.1: Der Tetraederbaum vierter Stufe

Mit POCALL kann die einfache Struktur deutlich gemacht werden.

```
pocall "TETRA
Ø : TETRA
1 : ... TETRA*
1 : ... DREIECK
```

Vielleicht sehen Sie sich noch ein weiteres Beispiel für einen rekursiv erzeugten Baum an, wobei das Programm hier nicht mehr näher erläutert werden soll.

```
to BAUM :laenge :winkel :n
if :n = Ø [stop]
fd :laenge
lt :winkel BAUM :laenge :winkel :n-1
rt 2*:winkel BAUM :laenge * .9 :winkel :n-1
lt :winkel bk :laenge
end
```

Aufruf: `cs pu setpos [30 -170] pd BAUM 30 17 9`

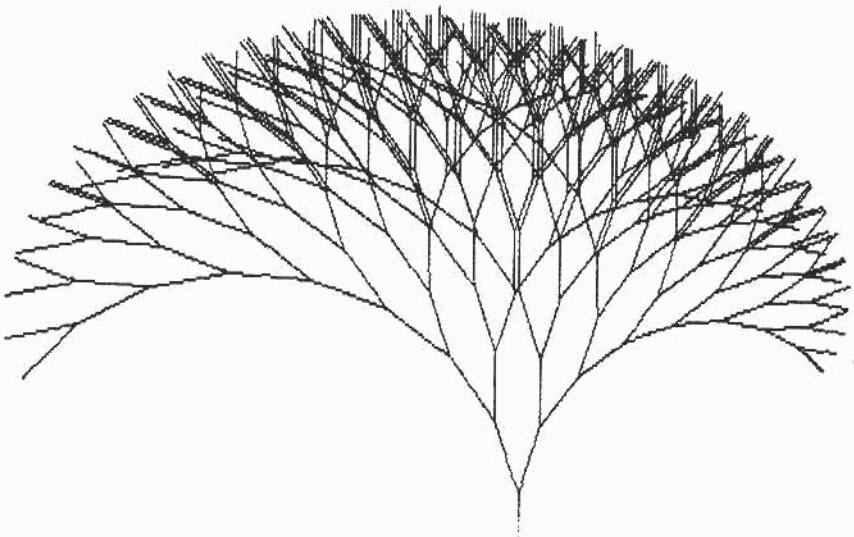


Abb. 13.2: Ein rekursiv erzeugter Binärbaum

13.5 Die Türme von Hanoi

Das von dem Mathematiker Lucas erfundene mathematische Spiel "Türme von Hanoi" ist ein klassisches Beispiel für Rekursion. Eine bestimmte Anzahl von Scheiben mit unterschiedlichem Durchmesser ist auf einem Stapel der Größe nach aufgetürmt, die kleineren zuoberst. Daneben gibt es noch zwei weitere Plätze, wo die Scheiben aufgetürmt werden können. Ziel ist es, den Turm vom Ausgangsplatz auf einen der beiden anderen umzusetzen. Dabei dürfen Scheiben immer nur auf einen leeren Platz oder auf Scheiben mit größerem Durchmesser gelegt werden.

Den Computer kann man dabei als Ersatz für reales Spielmaterial benutzen und die jeweilige Situation der drei Plätze auf dem Bildschirm anzeigen. Mit den unten aufgeführten Hilfsprozeduren kann man ein solches Spielprogramm formulieren.

Hier soll es um die Programmierung der optimalen Spielstrategie gehen.

Bei nur zwei Scheiben ist es leicht, die benötigten Züge anzugeben, um von Platz 1 nach Platz 2 umzuschichten:

```
1 nach 3 / 1 nach 2/ 3 nach 2
```

Bei drei Scheiben geht man so vor, daß zunächst der Teil des Turms ohne die unterste Scheibe nach Platz 3 umgeschichtet wird. Das Problem, einen Turm mit zwei Scheiben umzustapeln wurde aber schon gelöst. Anschließend wird die unterste Scheibe auf den gewünschten Platz 2 gelegt. Dann muß der Stapel mit zwei Scheiben von Platz 3 auf Platz 2 umgeschichtet werden, was wieder zum bereits gelösten Problem mit zwei Scheiben führt.

```
1 nach 2/1 nach 3/ 2 nach 3
1 nach 2
3 nach 1/ 3 nach 2/ 1 nach 2
```

Bei n Scheiben wird der obere Teil aus (N-1) Scheiben auf den dritten Platz umgestapelt, der nicht der Zielplatz ist, dann die unterste Scheibe umgelegt, zuletzt der Turm mit (N-1) Scheiben vom Rangierplatz auf den Zielplatz versetzt. Damit bietet sich eine rekursive Lösung an, die jeweils die Tätigkeit für einen Turm mit einer um eins verminderten Zahl von Scheiben benutzt. Die Rekursion ist zu Ende, wenn nur noch eine Scheibe umzustapeln ist.

Die Grundtätigkeit ist das Umlegen einer Scheibe von einem zu einem anderen Platz. Ohne Grafikausgabe kann man die entsprechende Anweisung ausgeben, und das Spiel mit echten Scheiben nach den Befehlen des LOGO-Programms ausführen.

Die Prozeduren zu grafischen Darstellung werden etwas später angegeben.

```
to UMLEGEN :von :nach
(pr [Lege um von Platz] :von [ nach Platz ] :nach)
end
```

Die eigentliche Lösung ist in der Prozedur VERSETZE programmiert.

```
to VERSETZE :n :von :nach
if :n=1 [UMLEGEN :von :nach stop]
local "rangierpl
make "rangierpl word "| 6 - (last :von) - (last :nach)
VERSETZE (:n-1) :von :rangierpl
UMLEGEN :von :nach
VERSETZE (:n-1) :rangierpl :nach
end
```

Die Türme werden nicht einfach durch die Nummer sondern mit 11, 12, 13 bezeichnet. Dieses Verfahren erweist sich für die Verwaltung der Scheiben als zweckmäßig. Hier ist eine programmierte Verwaltung noch nicht notwendig, aber die Programme können bei der nachfolgenden Erweiterung mit grafischer Darstellung weiter benutzt werden. Mit last wird aus dem Namen die Nummer gewonnnen, während umgekehrt mit word aus der Nummer wieder der richtige Name erzeugt wird.

Die Nummer des Rangierplatzes wird in der dritten Programmzeile mit Hilfe der dort angegebenen Formel bestimmt. Man kann sich leicht überzeugen, daß damit tatsächlich der freie Platz - weder VOR noch NACH - bestimmt werden kann. Die übrigen Zeilen geben das oben beschriebene Verfahren wieder. Es fehlt noch ein Rahmenprogramm.

```
to TURM :anzahl
type [Von Platz 1 nach Platz?]
VERSETZE :anzahl "|1 word "| rq
```

Die Untersuchung mit POCALL liefert:

```
POCALL "TURM
Ø : TURM
1 : ... VERSETZE
2 : ..... UMLEGEN
2 : ..... VERSETZE*
2 : ..... UMLEGEN
2 : ..... VERSETZE*
```

Grafikprozeduren für die Türme von Hanoi

Die folgenden Prozeduren können benutzt werden, um die optimale Lösung des Spiels auf dem Bildschirm grafisch darzustellen. Die Grafikprogramme sind komplizierter als die Programmierung des Spiels selbst.

Mit Hilfe der Prozeduren LEGE, AUF, AB kann man ein ausgefülltes Rechteck sozusagen als Scheibe an eine bestimmte Stelle zeichnen, bzw. diese Scheibe auf einen Stapel legen oder davon wegnehmen.

```
to LEGE :n :g :p
if :n < 1 [stop]
pu setpos se(200 * last :p) - 400 - :g/2 22*:n - 100
SCHEIBE :g end

to SCHEIBE :gr
type char 7;Ton
px seth 90
repeat 2 [fd :gr lt 90 fd 16 lt 90]
end

to AUF :p :g
make :p se thing :p :g
pd
LEGE count thing :p :g :p
end
```

```
to AB :p
(local "l "g)
make "l thing :p make "g last :l
pe
LEGE count :l :g :p
make :p bl :l
op :g
end
```

In LEGE bedeuten n die Lage der Scheibe auf einem Stapel, g die Scheibengröße und p die Platznummer. Damit wird eine Scheibe an eine ganz bestimmte Position gesetzt, die in der dritten Zeile ausgerechnet wird.

Die Scheibe selbst wird mit Hilfe der Prozedur SCHEIBE gezeichnet. Dabei wird zunächst das ASCII-Zeichen 7 (Bell) ausgegeben, was ein akustisches Signal erzeugt. Der Zeichenstift wird mit

```
px
```

in einen Zustand versetzt, bei dem die Turtle die Farbe beim Zeichnen invertiert, so daß ein Rechteck gezeichnet wird, wenn der Grafikschirm dort gelöscht ist. Bei nochmaligem Aufruf wird das Rechteck dann wieder gelöscht. Probieren Sie aus:

```
SCHEIBE 100 repeat 200 [] SCHEIBE 100
```

AUF dient dazu, auf den Stapel mit der Numer p eine weitere Scheibe der Größe g aufzulegen, während mit AB die oberste Scheibe eines Stapels gelöscht werden soll; beide benutzen natürlich dazu die Tätigkeit LEGE. Darüber hinaus muß eine Buchhaltung über den Zustand der drei Plätze stattfinden, denn für die grafische Darstellung der Türme muß man wissen, wieviele und welche Scheiben jeweils auf einem Platz liegen. Das geschieht mit Hilfe der Listen l1, l2 und l3.

In LOGO kann man Namen im Programm selbst zusammenbasteln, und zwar bei l1, l2, l3 aus dem Buchstaben l und den Ziffern 1, 2, 3. Wenn die Stapelnummer unter dem Namen p

verfügbar ist, wird der zugehörige Name mit word l :p erzeugt. Will man nun den Wert dieses Namens bekommen, so müßte nach der üblichen LOGO-Syntax ein Doppelpunkt davor gesetzt werden. Das reicht aber nicht aus, weil der Wert dann der eben gebildete Name ist. Das liegt daran, daß der Name nicht direkt gegeben ist, sondern selbst als Wert von etwas zustandekommt. Der Wert, hier der Wert der Liste l1 (12,13), wird durch die LOGO-Vokabel thing bestimmt.

Beim Auflegen einer Scheibe muß die entsprechende Liste um die Größe der hinzugekommenen Scheibe ergänzt und beim Herunternehmen aus der Liste die entsprechende Angabe gestrichen werden. Sie wird als Ergebnis zurückgegeben, weil die Scheibe auf einen anderen Stapel gelegt werden muß.

Das Umlegen wird grafisch nun durch AB und AUF bewerkstelligt:

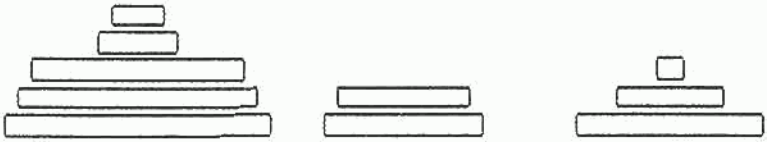
```
to UMLEGEN :von :nach
AUF :nach AB :von
end
```

Das Ergebnis von AB dient als Eingabewert für AUF. Zur Abrundung fehlen nur noch die Prozeduren TURM und BAUE, in denen die Ausgangssituation des Spiels herbeigeführt und grafisch dargestellt wird.

```
to TURM :anzahl
ht cs
make "l1 [] make "l2 [] make "l3 []
BAUE :anzahl 20 "l1
type [Von Platz 1 nach Platz ?]
VERSETZE :anzahl "l1 word l rq
end
```

```
to BAUE :n :g :p
if :n =0 [stop]
lege :n :g :p
make :p se :g thing :p
BAUE :n - 1 :g + 20 :p
end
```

Probieren Sie dies zunächst einmal mit kleinen Türmen und beobachten Sie dabei das Anwachsen der benötigten Zeit, wenn die Scheibenzahl erhöht wird.



von platz 1 nach platz :2

Abb. 13.3: Eine Situation mitten im Spiel

Das gesamte Paket der Prozeduren für die Türme von Hanoi kann mit Hilfe von POCALL (aus der vorangeganenen Lektion) untersucht werden. Die Analyse liefert folgendes Ergebnis:

```
0 : TURM
1 : ...BAUE
2 : ......LEGE
3 : .........SCHEIBE
2 : ......BAUE*
1 : ...VERSETZE
2 : ......UMLEGEN
3 : .........AUF
4 : ............LEGE
5 : ..............SCHEIBE
3 : .........AB
4 : ............LEGE
5 : ..............SCHEIBE
2 : ......VERSETZE*
2 : ......UMLEGEN
3 : .........AUF
4 : ............LEGE
5 : ..............SCHEIBE
3 : .........AB
4 : ............LEGE
5 : ..............SCHEIBE
2 : ......VERSETZE*
```

Iterative gegen rekursive Problemlösungen

Abgesehen von der Rekursion als Wiederholungsmethode bereitet das Verständnis rekursiver Problemlösungen manchem Anwender Schwierigkeiten. Die Rekursion wird deswegen hier ausführlich erläutert. Sie sollten sich gelegentlich auch genügend Zeit zum Verständis der Abwicklung dieser Proramme nehmen.

Wenn das inhaltliche Problem komplizierter ist, lassen sich öfter rekursive Lösungsvorschläge viel leichter als iterative finden. Man kann auch so vorgehen, daß zunächst eine rekursive Lösung gesucht wird, um überhaupt eine in der Hand zu haben. Später kann man dann versuchen, iterative Methoden herauszufinden, da diese in der Regel schneller arbeiten.

Als Beispiel sollen noch einmal die Türme von Hanoi herhalten.

Zunächst dürfte es Ihnen kaum gelingen, direkt eine iterative Lösungsmethode anzugeben denn Sie müssen dafür einen eigenen Algorithmus entwickeln. Beobachten Sie nun, was bei der rekursiven Methode passiert. Sie werden bald feststellen, daß bei jedem zweiten Schritt gerade die kleinste von allen Scheiben bewegt wird. Anschließend haben Sie für den nächsten Zug keine Wahl mehr, wenn Sie die Wiederholung von Zügen vermeiden. An den Platz der kleinsten Scheibe kann nichts gelegt werden. Bei den verbleibenden zwei Turmpositionen gibt es dann nur eine Möglichkeit, weil eine von den beiden zuoberst liegenden Scheiben die kleinere ist.

Jetzt müssen Sie nur noch die Bewegung der kleinsten Scheibe beobachten! Es stellt sich heraus, daß diese bei gegebener Anfangsbedingung und Zielposition immer in die gleiche Richtung marschiert. Dabei muß man sich die drei Türme auf einem Kreis angeordnet denken. Ob die kleinste Scheibe gegen oder mit dem Uhrzeigersinn bewegt wird, hängt von der Scheibenzahl und der vorgegebenen Ziellage ab. Das Verfahren muß demnach wie folgt ablaufen:

- Bewegung der kleinsten Scheibe um eine Einheit, z.B. nach rechts.
- Herausfinden, welche der beiden zuoberst liegenden Scheiben bewegt werden darf.
- Prüfen auf das Ende des Verfahrens.

Wenn man die wesentlichen Punkte des Verfahrens einmal gesehen hat, erscheint die Lösung ganz einfach. Nur muß man darauf natürlich erst einmal kommen.

```
to STRATEGIE :t1 :t2 :t3
;Iterative Loesung fuer die Tuerme von Hanoi
label "anfang
make "t3 :t1 make "t1 :t2
UMLEGEN :t3 :t2; Bewegung der kleinsten Scheibe
make "t2 NACHFOLGER :t1
if emptyp thing :t3 [UMLEGEN :t2 :t3 go "anfang]
if emptyp thing :t2 [UMLEGEN :t3 :t2 go "anfang]
if BREITE :t3 > BREITE :t2 [UMLEGEN :t2 :t3 go "anfang]
UMLEGEN :t3 :t2 go "anfang
end

to NACHFOLGER :l
op item last :l [l2 l3 l1]
end

to BREITE :l
op last thing :l
end
```

Die Prozedur STRATEGIE ersetzt bei der iterativen Lösung die Prozedur VERSETZE, so daß die letzte Zeile von Turm dann lauten muß:

```
STRATEGIE "l1 "l2 "l3
```

STRATEGIE benutzt die beiden Hilfsprogramme NACHFOLGER und BREITE. NACHFOLGER gibt den Namen des nachfolgenden Turms in zyklischer Reihenfolge aus. Bei l1 wird l2, bei l2 wird l3 und bei l3 wird l1 ausgegeben. BREITE ergibt die Breite der auf dem betreffenden Stapel zuoberst liegenden Scheibe.

Zunächst wird die kleinste Scheibe bewegt, und das Ziel wird durch die Prozedur NACHFOLGER bestimmt. Beim darauffolgenden Zug werden die beiden Türme betrachtet, auf denen nicht die kleinste Scheibe liegt. Ist einer von beiden leer, so steht der benötigte Zug fest. Im anderen Fall muß geprüft werden, auf welchem Turm die größere Scheibe zuoberst liegt, was mit Hilfe von BREITE geschieht.

Die angebenene iterative Lösung ist noch nicht so allgemein wie die rekursive, weil man dabei nicht vorgeben kann, ob der Turm auf den zweiten oder dritten Platz umgeschichtet wird, das hängt nämlich von der Scheibenzahl ab. Wollen Sie die Zielposition vorher festlegen, muß das Programm noch verallgemeinert werden, indem vorher bestimmt wird, mit welchem Drehsinn die kleinste Scheibe bewegt wird. Diese Verallgemeinerung sei Ihnen zur Übung aufgegeben.

Es gibt auch noch einen weiteren Schönheitsfehler: Nach Beendigung der Umschichtung wird mit einer Fehlermeldung abgebrochen, weil bei einer Wiederholung mit der Sprunganweisung GO noch kein Abbruch vorgesehen ist. Am einfachsten ist eine kleine Änderung in der Prozedur AB

```
to AB :p
local l make l thing :p
if emptyp :l [throw "TOPLEVEL]
...
```

Throw "TOPLEVEL bewirkt eine Beendigung des Programms wie stop, wobei aber nicht ins rufende Programm zurückgesprungen wird. Der gesamte Programmablauf wird abgebrochen und das System ist anschließend wieder eingabebereit.

Übungen

1) Stellen Sie fest, wie häufig bei 2, 3, 4,... Scheiben für einen Turm von Hanoi vom Programm Scheiben umgelegt werden!

2) Ein Anfangskapital K soll mit einem bestimmten Prozentsatz p verzinst werden. Schreiben Sie eine rekursiv arbeitende Prozedur, die das verzinste Kapital nach n Jahren (Zinsperioden) als Ergebnis ausgibt.

3) Ein Sparer zahlt jährlich eine bestimmte Summe R auf ein Konto, das mit einem Zinssatz p verzinst wird. Formulieren Sie mit Hilfe der Rekursion eine Programm, das den Kontostand nach n Jahren ausgibt.

4) Verallgemeinern Sie die iterative Lösung für die Türme von Hanoi so, daß der Zielplatz vorgegeben werden kann. Zusätzlich zur Hilfsprozedur NACHFOLGER muß eine analoge Prozedur VORGAENGER formuliert werden.

Lektion 14: Fortgeschrittene Grafikprogrammierung

Neue Sprachelemente:
dot, dotc, fill

Programme:
STAMPWORD, DREIECK (Fall SSS), WELLE, C, LDRACHE, RDRACHE, HILBERT, SIERPINSKI, KIPPBILD, RINGE, SICHEL, FLUCHT, WUERFEL, PUNKT, VEKTOR, TETRAEDER

14.1 Beschriftung von Grafiken

Wenn auf dem Textschirm ein Zeichen ausgegeben werden soll, dann muß auf ein Raster von 8*8 Punkten ein fertiges Bild zur Darstellung des Zeichens gebracht werden. Die Bilder werden aus dem gerade aktuellen Zeichensatz entnommen, der eine Auswahl von 224 Darstellungen zur Verfügung stellt. Den Zeichensatz können Sie sich mit der Zeile

```
make "i 32 repeat 224 [type char :i make "i :i+1]
```

ansehen.

Im Gegensatz dazu wird bei Zeichenvorgängen auf einzelne Punkte des Grafikschirms zugegriffen. Wenn die angebotene Auswahl von Zeichen auch vielleicht schon reichlich erscheint, so sind 224 mögliche Bilder fast nichts gegen die tatsächlich möglichen Varianten auf einem 8*8 Punktraster (deren Anzahl benötigt selbst bei einfarbiger Darstellung 20 Dezimalstellen!).

Zwischen der Erzeugung von Grafiken und der Ausgabe von Textzeichen bestehen deshalb wesentliche Unterschiede. Manche Mikrocomputer verwalten dementsprechend Text- und Grafikschirm intern als völlig getrennte Bereiche. Bei den Schneider-Computern sind beide Schirme Teile des gesamten Bildschirms,

sofern überhaupt Text und Grafik simultan benötigt werden. Man spricht auch von der Aufteilung in ein Text- und ein Grafikfenster.

Die Beschriftung von Grafiken wird von DR LOGO leider nicht durch eigene Kommandos unterstützt, sodaß man teilweise zu Tricks greifen muß.

Text auf dem Grafikschirm beim Schneider Joyce

Beim Joyce ist die Textausgabe auf dem Grafikschirm besonders einfach, weil es ohne weiteres möglich ist, den Textcursor in den Grafikschirm hineinzusteuern. Wenn Sie einen Grafikbefehl wie etwa home erteilen, finden Sie sich standardmäßig anschließend in einem Modus mit geteiltem Bildschirm. Die Postion des Textcursors erfahren Sie mit

```
cursor
```

Ergebnis etwa: `[0 22]`

Das Ergebnis gibt den Anfang der 22ten Zeile an; das ist die Zeile, in der auch die Antwort ausgegeben wurde, also die zweite Zeile des Textfensters. Man kann aber auch den Textcursor in das Grafikfenster hinein dirigieren.

```
setcursor [20 10] type [Hier ist der Textcursor]
```

Damit ist schon klar, wie beim Joyce Grafiken beschriftet werden können:

Der Textcursor wird an die gewünschte Stelle des Grafikfensters dirigiert. Die Ausgabe erfolgt dann wie auf dem Textschirm.

Kehrt LOGO dann in den Direktbetrieb zurück, erscheint der Cursor und das Fragezeichen (Prompten) in der auf die zuletzt erreichte Position des Textcursors folgende Zeile, häufig also mitten im Grafikfenster. Um den Cursor ins Textfenster zurück-

zubringen kann man am einfachsten mit ct das Textfenster löschen.

```
to TBEISPIEL
cs setcursor [20 10] pr [Testbeipiel]
repeat 100 [fd random 80 rt random 360]
ct;Textcursor zurück
end
```

Wenn das Löschen unerwünscht ist, kann man zu Beginn den aktuellen Stand des Textursor aufheben und am Schluß wieder zurücksetzen:

```
make "textcursor cursor ... setcursor :textcursor
```

Auf den Ursprung des Koordinatensystems, also die durch home erreichte Position, zielt man beim Joyce etwa mit der Textcursorposition [45 15]. In vertikaler Richtung müssen acht, in vertikaler Richtung 16 Schritte für ein Zeichen veranschlagt werden.

Beschriften von Grafik beim Schneider-CPC

Beim CPC darf der Textcursor nicht in das Grafikfenster hineingesteuert werden, wobei ein solcher Versuch zu unerwünschten Effekten im Textfenster führt.

Der CPC kenn aber einen Grafikcursor, der in LOGO natürlich durch die Position der Turtle bestimmt wird und dementsprechend auch mit Kommandos an die Schildkröte verändert werden kann. An dieser Position kann ein Zeichen ausgegeben werden.

```
to STAMPWORD :wort :m if emptyp :wort [stop]
type (word char 23 :m char 5 first :wort)
STAMPWORD bf :wort :m
end
```

Die Ausgabe an der Position des Grafikcursors wird durch das in der dritten Zeile von STAMPWORD vorangesetzte Steuerzeichen char 5 erreicht. Mit der Prozedur STAMPWORD kann dann ein ganzes Wort ausgegeben werden. Tatsächlich werden bei unmittelbar aufeinanderfolgenden Ausgaben von Zeichen diese hintereinandergesetzt, ohne daß die Turtle verschoben werden muß.

Vor char 5 in der dritten Zeile wird noch das Steuerzeichen 23 sowie der eigegebene Wert von m in die Ausgabe eingefügt. Dabei steht m für den Zeichenmodus des CPC, wobei Null den Normalwert darstellt.

```
cs STAMPWORD "Hallo Ø
```

Die Eingabe des Zeichenmodus ist insofern erforderlich, als das LOGO-System von sich aus den Zeichenmodus 1 wählt, was sich in einer Invertierung bzw. bei unterschiedlichen Zeichenfarben in einem Farbwechsel bemerkbar macht.

```
cs STAMPWORD "Hallo Ø fd Ø STAMPWORD "Hallo 1
```

Mit dieser Zeile wird ersichtlich, daß mit fd 0 der Grafikcursor wieder an die durch die Position der Turtle bestimmte Stellung gebracht wird. Bei Bewegungen der Turtle muß für STAMPWORD die Turtle sichtbar sein, damit die Veränderung des Grafikcursors richtig ausgwertet wird.

Die nachfolgende Ausgabe von Hallo bewirkt nun bei gleicher Zeichenfarbe eine Löschung. Bei der Überlagerung unterschiedlicher Zeichenfolgen wird die Schrift auf diese Weise meist unlesbarer werden.

Für den LOGO-Programmierer ist diese Verwendung gerätespezifischer Steuercodes im Grunde genommen suspekt. Hier hätte der Hersteller besser eine eigene LOGO-Vokabel vorsehen sollen. Im Handbuch ist die Beschrifung von Grafiken überhaupt nicht erwähnt.

14.2 Einzelpunkte

Eine Zeichung entsteht bei der Turtle typischerweise durch die Spur, die die Schildkröte bei ihren Bewegungen auf dem Bildschirm hinterläßt.

Die betrachtete LOGO-Version bietet aber auch Kommandos zur Erzeugung einzelner Punkte auf dem Bildschirm. Dazu dienen die LOGO-Vokabeln dot und dotc.

```
dot [200 100]
```

Damit wird ein Punkt an der durch die Eingabe festgelegten Position in der aktuellen Farbe des Zeichenstifts gesetzt. Probieren Sie aus:

```
repeat 1000 [dot se -320+random 640 -120+random 320]
```

Mit der Zeile

```
dot tf
```

wird ein Punkt an der aktuellen Turtleposition gesetzt. Stattdessen kann auch das Kommando

```
fd 0
```

benutzt werden, was für diesen Zweck natürlich einfacher ist. Der Befehl dot kann im Prinzip vollständig durch

```
setpos [200 100] fd 0
```

ersetzt werden.

Die LOGO-Vokabel dotc liefert die Farbnummer des eingegebenen Punktes als Resultat.

```
setpc 1 dot [100 100] dotc [100 100]
```

Ergebnis: 1

Der Hintergrund wird als Voreinstellung mit der Farbe 0 versehen, wenn Sie ihn nicht mit Hilfe setbg verändern. Damit kann dotc zum Test einzelner Punkte eingesetzt werden.

Beim Joyce gibt dotc unmittelbar an, ob ein Punkt auf dem Grafikschirm gesetzt ist oder nicht, weil hier der Hintergrund immer den Farbcode 0 und gesetzte Punkte die Farbnummer 1 haben. Beim CPC kann mit Hilfe von sf die Hintergrundfarbe festgestellt werden, wenn diese nicht von vorne herein bekannt ist.

```
if dotc [100 100] = 1 [(pr "Punkt [100 100] [ist besetzt)]
if not dotc [100 100] = first sf [ ... ]
```

Mit dotc kann man auch feststellen, ob der angesprochene Punkt auf dem Bildschirm liegt, weil dotc den Wert -1 als Ergebnis zurückgibt, wenn die Koordinaten aus dem Grafikschirm hinausweisen.

Schnittpunkte: Dreieckskonstruktionen

Mit dem Einsatz von dotc als Testbefehl für Einzelpunkte wird es auch möglich, festzustellen, ob der Zeichenstift bereits vorhandene Linien trifft. Man kann also damit Schnittpunkte von Geraden unnd Kurven grafisch bestimmen. In der Mathematik gibt es hierfür viele Anwendungsmöglichkeiten.

Als Beispiel soll hier die Konstruktion eines Dreiecks aus den drei Seitenlängen betrachtet werden. Diese Aufgabe wird auf dem Papier so gelöst, daß eine Seite gezeichnet und anschließend ein Schnittpunkt zweier Kreise gesucht wird. Mit der LOGO-Funktion dotc soll die Konstruktion auf dem Bildschirm vorgenommen werden.

```
to DREIECK :a :b :c
ht (local "ak "bk "ck)
make "ak se -:c/2 -80;Punkt A
make "bk se :c/2 -80;Punkt B
pu setpos :bk pd setpos :ak;Seite c
halbkreis :b;Kreis um A mit Radius b
pu setpos :bk
make "ck SCHNITTKREIS :a;Kreis um B mit Radius a
if emptyp :ck [pr [keine Lösung] stop]
pd setpos :bk setpos :ak setpos :ck
(pr "Gamma: (towards :ak) - (towards :bk))
setpos :ak (pr "Alpha: (towards :bk) - (towards :ck))
setpos :bk (pr "Beta: (towards :ck) - (towards :ak))
end

to SCHNITTKREIS :r
(local "u "w)
pu seth -90 fd :r rt 90
make "u 1.5708*:r make "w 180/:u rt :w/2
repeat 4 [fd 2 rt :w]
repeat :u [fd 2 rt :w if or (dotc tf >0) (dotc se first
tf 1+item 2 tf>0) [type char 7 op tf] [op []]
end

to HALBKREIS :r
pu seth 90 fd :r lt 95 pd
local "s make "s .17453 * :r
repeat 18 [fd :s lt 10]
pu lt 85 fd :r
end
```

Mit der Hilfsprozedur HALBKREIS wird ein Halbkreis um den Punkt A gezeichnet. Anschließend wird die Turtle auf einem zweiten Halbkreis um den Punkt B mit Hilfe vom SCHNITTKREIS geführt. Der Zeichenstift bleibt dabei abgehoben, weil der Testbefehl dotc ja nur die Punkte des zuerst gezeichneten Kreises feststellen darf.

Während beim ersten Halbkreis tatsächlich nur die Hälte eines 36-Ecks gezeichnet wird, muß die Turtle beim zweiten Mal ganz langsam vorangeschickt werden, damit der Schnittpunkt auch tatsächlich erreicht wird. Hier wurde die Schrittweite 2 gewählt (fd 2). Anschließend wird dotc mit den gerade erreichten Koordinatenwerten, die mit tf festgestellt werden, und auf den Punkt mit einer um eine größeren y-Koordinate zur Abfrage auf den Schnittpunkt benutzt. (Beim Joyce erhöht man besser um zwei.)

Die Prozedur Schnittkreis gibt im Erfolgsfall unter Begleitung durch einen Piepton die Koordinaten des Schnittpunktes, bei Mißerfolg dagegen die leere Liste als Ergebnis zurück.

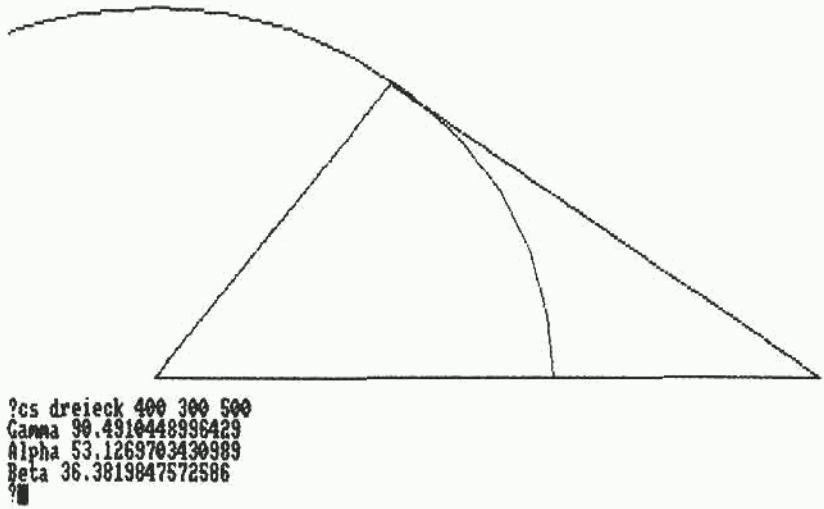


Abb. 14.1: Dreieckskonstruktion

Die Lösung der Konstruktionsaufgabe wird zum Schluß auch zur Bestimmung der Winkel im Dreieck benutzt. Dazu wird von einem Eckpunkt jeweils mit Hilfe der LOGO-Vokabel towards

auf die beiden anderen Ecken gezielt und daraus der gesuchte Winkel bestimmt.

14.3 Dichteverteilungen

Interessante Grafiken können mit Strecken, Polygonzügen sowie mit ausgefüllten, gefärbten oder schraffierten Flächenstücken erzeugt werden. Aber auch die Verteilung einzelner Punkte kann für grafische Zwecke genutzt werden; diese werden dann meist mit Hilfe von Zufallszahlen unregelmäßig gestreut, womit man bei Flächen raffinierte Effekte hervorrufen kann, z.B. den räumlichen Eindruck unter verschiedenartigen Beleuchtungen.

Hier soll nur an Hand einfacher Beispiele das Prinzip erläutert werden. Eine gleichmäßige Zufallsverteilung von Punkten ist natürlich langweilig und es wird erst dann interessant, wenn in bestimmten Bereichen wenig, in anderen viele Punkte gesetzt werden.

Das Grafikfenster ist ein zweidimensionales Gebilde. Um für unterschiedliche Häufigkeiten von Punkten zu sorgen, muß zu jedem möglichen Bildpunkt eine Zahl gehören, deren Wert darüber entscheidet, ob ein Punkt gesetzt wird. Um eine zufällige Streuung von Punkten zu erreichen, wird die jedem Punkt zugeordnete Zahl mit einer erwürfelten Zufallszahl verglichen. Wird nun ein Punkt gerade dann gesetzt, wenn die Zufallszahl kleiner ist als der jeweils zugeordnete Wert, ergibt sich in Bereichen großer zugeordneter Zahlenwerte eine große Wahrscheinlichkeit für einzelne Punkte.

Derartige Verteilungen von einzelnen Punkten werden demnach durch die Vorgabe einer Funktion festgelegt, die jedem möglichen Bildpunkt auf dem Grafikschirm einen Zahlenwert zuordnet. Weil die mit random erzeugten Zufallszahlen nicht negativ sind, müssen auch die Funktionswerte sinnvollerweise positiv sein. Der größtmögliche Wert sollte mit demjenigen übereinstimmen, der beim Aufruf von random als Eingabe benutzt wird, was sich stets durch geeignete Faktoren erreichen

läßt. Die möglichen Abstufungen des erzeugten Bildes werden durch den Bereich bestimmt, in dem die Zufallszahlen streuen.

Kreiswellen

Wenn Sie auf die ruhige Oberfläche eines Teiches einen Stein werfen, so breiten sich kreisförmig Wellen aus. Bei einer Momentaufnahme gibt es an bestimmten Orten Wellenberge, an anderen Wellentäler. Eine solche Momentaufnahme kann nun durch eine Verteilung von Punkten dargestellt werden.

Die Funktion, die zu jedem möglichen Bildpunkt einen geeigneten Zahlenwert liefert, muß bei festem Abstand vom Erregerzentrum gleiche Werte liefern. Um den wellenartigen Charakter darzustellen, bieten sich die Funktionen SIN bzw. COS an.

```
to WELLE :y
if :y < 0 [stop]
make "x 0
repeat 150 [make "x :x+1 if 10*(1+cos F :x :y) > random
22 [dot se :x :y dot se -:x :y dot -:x -:y dot :x :-y]]
WELLE :y-2
end
```

Aufruf z.B. mit: `fs cs ht WELLE 151`

Das oben beschriebene Verfahren findet in der Liste der REPEAT-Zeile von WELLE statt. Diese Zeile bearbeitet die Punkte auf einer horizontalen Geraden im Grafikfenster. Zur Darstellung einer Wellenform wird der Kosinus des Abstands vom Koordinatenursprung genommen. Damit ein nichtnegativer Funktionswert entsteht, wird zunächst die Zahl 1 addiert, so daß die Funktionswerte zwischen Null und zwei erreicht werden. Durch Multiplikation mit 10 werden die möglichen Werte dann auf den Bereich bis 20 gestreckt. Anschließend erfolgt der Vergleich mit der gewürfelten Zufallszahl in einem entsprechenden Bereich, mit dessen Größe man noch etwas spielen kann.

Zur Beschleunigung wird hier die Symmetrie der Verteilung ausgenutzt. Tatsächlich wird nämlich die Berechnung nur in einem Quadranten des Grafikschirms durchgeführt, während die anderen drei Quadranten durch Spiegelungen erzeugt werden.

Das Programm ist einfach und erzeugt mit der Zufallsstreuung eine grafische Wirkung. Weil aber nach und nach alle möglichen Punkte durchgemustert werden, ist der Rechenaufwand erheblich. Derartige Anwendungen sollten Sie in Gang setzen, wenn Sie den Computer einige Stunden sich selbst überlassen können. Man kann durchaus auch den Abstand der einzelnen Punkte größer wählen und damit die Anzahl verringern, wegen der begrenzten Bildschirmauflösung macht sich das zunächst kaum bemerkbar. Das gilt insbesondere für den Joyce hinsichtlich der vertikalen Auflösung.

Wenn die Grafik fertig ist, können Sie sie nach Gefallen abspeichern.

```
savepic "KWELLE
```

Die Verteilungsfunktion ist hier in der Prozedur F versteckt, die für andere Verteilungen dementsprechend geändert werden muß. Das Ergebnis von F soll im betrachteten Beispiel ein Zehnfaches vom Abstand des Punktes mit den Koordinaten x und y vom Koordinatenursprung sein. Multiplikation mit dem Faktor 10 gewährleistet einen sinnvollen Bildausschnitt. Diesen Faktor können Sie nach Belieben variieren.

Der Abstand eines Punktes vom Nullpunkt wird als Quadratwurzel aus der Summe der Koordinatenquadrate berechnet. Nun hat der Hersteller aber dummerweise bei der vorliegenden LOGO-Version keine Funktion für die Quadratwurzel bereitgestellt. Dieser Mangel läßt sich natürlich durch eine selbstdefinierte Funktion beheben, was aber hier wegen der großen Zahl von Punkten zu gewaltigem Zeitbedarf führt.

Der gesuchte Abstand kann auch mit Hilfe von trigonometrischen Funktionen berechnet werden, die im Grundwortschatz von DR LOGO vorhanden sind.

```
to F :x :y
if :x=0 [op :y]
if :y=0 [op :x]
op :x/sin arctan :x/:y
end
```

Für Punkte auf den Koordinatenachsen versagt die Berechnung mit Hilfe der arctan-Funktion, weswegen die Abfragen in F erfolgen müssen. Weil diese Abfragen allerdings auch wieder die Rechnung verlangsamen, kann es hier sinnvoll sein, die Punkte auf den Koodinatenachsen im rufenden Programm Welle in getrennten Zeilen zu bearbeiten. Die Funktion F wird dann dort nicht benötigt, sodaß die Abfragen entfallen können.

Für die in Abb. 14.2 zu sehende Grafik wurde allerdings eine schnellere Näherungslösung gewählt.

```
to F :x :y
make "s :x + :y make "p :x * :y / :s
op 10 * (:s - :p - :p * :p / (:s + :s)
end
```

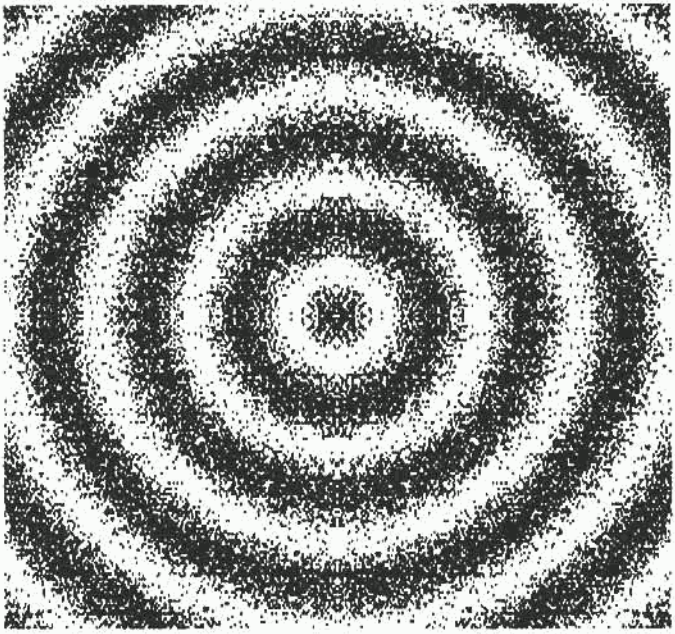


Abb. 14.2: Kreiswellen

Andere Verteilungen

Durch Abänderung der Prozedur F können andere Verteilungen erzeugt werden, wobei es keine Grenzen zum Experimentieren gibt. Es kann dabei leicht passieren, daß ein Bild ohne klar erkennbare Struktur entsteht. Dann muß der Funktionswert in der Prozedur F mit einem geeigneten Faktor multipliziert werden.

Als grafisches Hilfmittel kann auch eine Streckung in vertikaler Richtung eingesetzt werden. Dafür stellt LOGO das Kommando setscrunch zur Verfügung:

```
setscrunch .2
sf
```

Antwort z.B.: `[Ø Ø WINDOW Ø.2]`

Damit kann das Achsenverhältnis variiert werden. Die Standardeinstellung hat beim CPC den Wert eins, beim Joyce 0.46875. Diese stellen sicher, daß etwa eine Bewegung mit fd 100 auf dem Bildschirm in waagerechter wie in senkrechter Richtung auch tatsächlich gleichlange Strecken erzeugen.

Das Kommando sf (Abkürzung von Screenfacts) gibt als letzten Wert den aktuellen Wert des Faktors für das Achsenverhältnis aus.

Versuchen Sie einmal folgendes Beispiel, das relativ wenig Rechenzeit benötigt.

```
to F :x :y
op Ø.Ø5 * :x * :y
end
```

Anmerkung: In solchen Bereichen, wo der von F erzeugte Funktionswert übereinstimmt, werden auch gleiche Punktdichten erzeugt. Deswegen entstehen Verteilungen, die sich an den Kurven orientieren, die zur Gleichung F=konstant gehören. Das sind in diesem Fall Hyperbeln mit den Koordinatenachsen als Asymptoten.

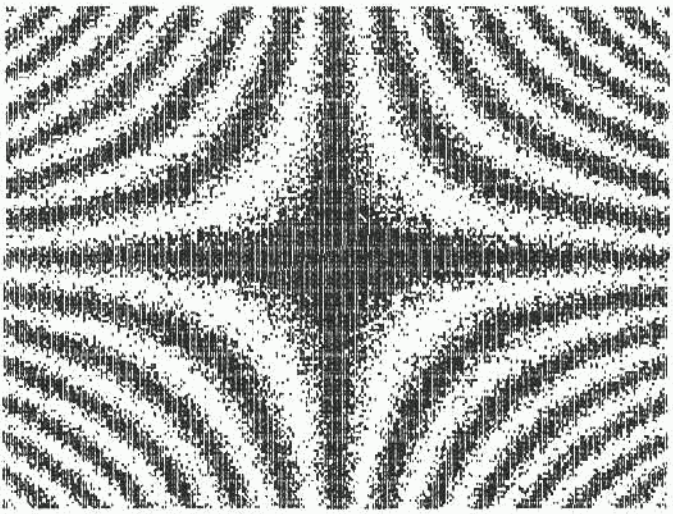


Abb. 14.3: Hyperbelwellen

Die Grafik in Abb. 14.4 wird schließlich mit folgender Funktion F erzeugt.

```
to F :x :y
op (2*towards [0 0]) + 700 * log :x*:x + :y*:y
end

to LOG :x
make "b (word ". :x)
make "t (:b - 1) / (:b + 1)
op :t + 0.8182 * count :x
end

to WELLE :y
if :y < 0 [stop]
make "x -300
repeat 300 [make "x :x+1 if 10*(1+cos F :x :y) > random 22 !
[dot se :x :y dot se -:x -:y]]
WELLE :y-2
end
```

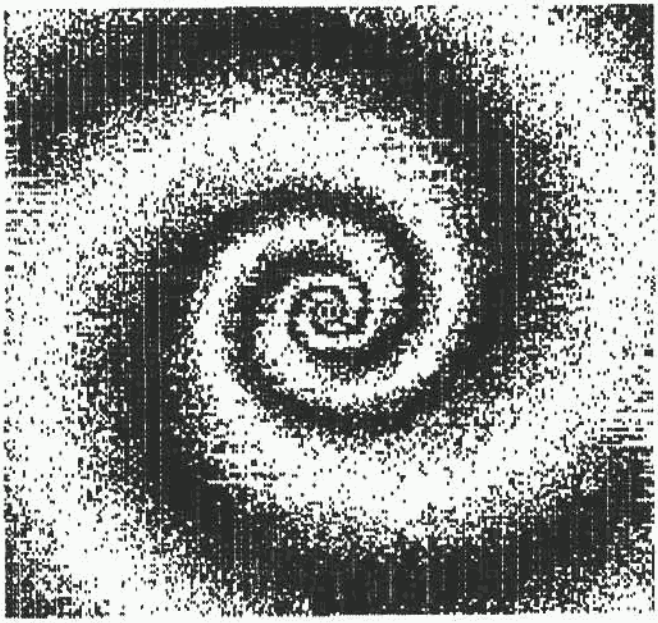


Abb. 14.4: Verteilung mit spiralförmiger Struktur

Das Hauptprogramm WELLE wird hier noch einmal wiedergegeben, weil die erzeugte Figur eine geringere Symmetrie besitzt und nur noch eine Spiegelung am Nullpunkt stattfindet.

Die Grundstruktur des Spiralnebels ist eine logarithmische Spirale. Weil DR LOGO bei den Schneidercomputern auch keine Logarithmusfunktion eingebaut hat, wurde als Ersatz die Prozedur LOG vorgesehen. Diese gibt eine sehr grobe Näherung für einen Logaritmus mit einer Basis, die etwa bei 18 liegt. Für die Erzeugung der Punkteverteilung ist sie brauchbar, als Ersatz zur Berechnung des Logarithmus bei mathematischen Aufgaben ist sie aber keineswegs geeignet.

Das logarithmische Verhalten der Funtkion F wird übrigens im wesentlichen durch Abzählen der vorhandenen Dezimalstellen mit count erreicht (die Eingabe ist hier eine natürliche Zahl).

Außer purer Freude an der Grafik, wie sie in den Abbildungen 14.2 bis 14.4 zu sehen ist, finden solche Verteilungen auch Anwendungen in Bereichen der Naturwissenschaft, in denen Zufallsverteilungen eine Rolle spielen.

Ein bekanntes Beispiel ist die Antreffwahrscheinlichkeit von Elektronen im Inneren eines Atoms bzw. Moleküls. Die Naturwissenschaft behauptet, daß die Elektronen - das sind die elektrisch geladenen Teilchen in der Atomhülle - keinen festen Ort besitzen, sondern mit gewissen Wahrscheinlichkeiten an bestimmten Stellen anzutreffen sind. Die Quantenphysik bzw. die Quantenchemie beschreiben dieses Verteilungen durch entsprechende Funktionen. Dies ist ein Bereich, wo Sie die beschriebenen Techniken zur Veranschaulichung einsetzen können.

14.4 Rekursive Ästhetik

Vielleicht haben auch Sie sich als Kind an der Schönheit von Mustern in einem Kaleidoskop erfreut. Der ganze Reiz entsteht ja nur aus der mehrfachen Spiegelung eines aus Glasstücken zufällig entstandenen Grundmusters. Der rekursive Gebrauch eines einfachen grafischen Grundschritts kann in ähnlicher Weise zu reizvollen Mustern führen.

Der Tetraederbaum aus Lektion 13 ist ein solches Beispiel, hier sollen vier weitere Beispiele zur Anregung dienen. Zunächst zwei Beispiele, bei denen die Grundtätigkeit nur aus dem Zeichnen einer kleinen Strecke besteht. Es ist überraschenschend, was daraus durch mehrfachen Selbstaufruf und Drehungen um rechte Winkel entstehen kann, ist überraschend. Die beiden folgenden Beispiele sind dem Buch "Turtle Geometry" von H. Abelson und A. di Sessa entnommen.

```
to C :laenge :niveau
if :niveau =0 [fd :laenge stop]
C :laenge :niveau - 1 rt 90
C :laenge :niveau - 1 lt 90
end
```

```
to LDRACHE :laenge :niveau
if :niveau=0 [fd :laenge stop]
LDRACHE :laenge :niveau - 1 lt 90
RDRACHE :laenge :niveau - 1 end
to RDRACHE :laenge :niveau
if :niveau=0 [fd :laenge stop]
LDRACHE :laenge :niveau - 1 rt 90
RDRACHE :laenge :niveau - 1
end
```

Probieren Sie C mit wachsender Verschachtelungstiefe aus:

```
fs cs C 5 1 cs C 5 2 ...
```

Die entstehenden Muster könnten als Vorlage für Broschen dienen, deren typische Gestalt ab Niveau 4 deutlich wird. Sie werden feststellen, daß sich die Achse der Figur bei wachsender Niveauzahl jeweils um 45 Grad dreht. Vor dem Aufruf von C kann dann der Zeichenstift noch auf einen geeigneten Platz gesetzt und mit einer vorangesetzten Drehung in die günstigste Richtung gebracht werden.

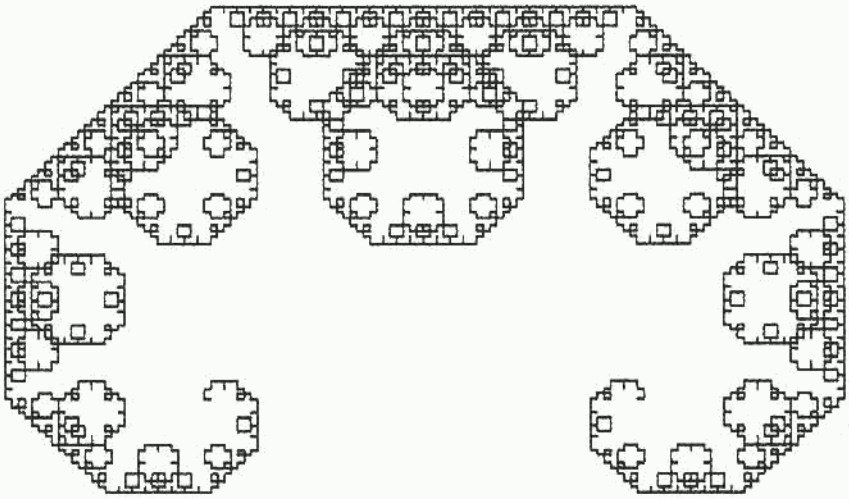


Abb. 14.5: Das von C auf der 12. Stufe erzeugte Muster

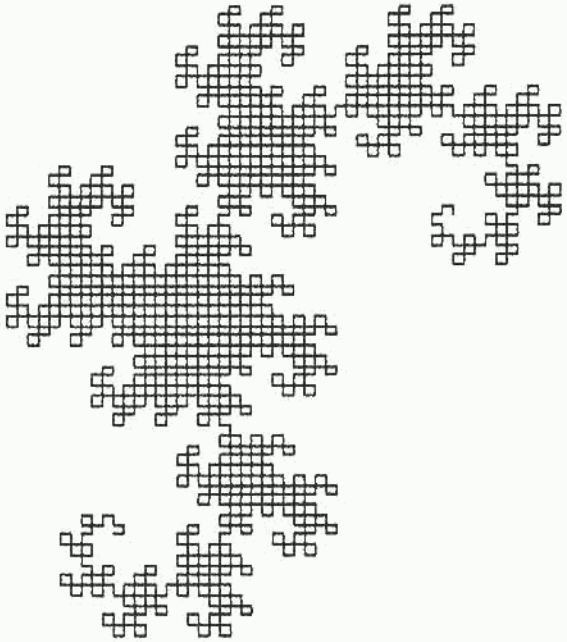


Abb. 14.6: Der Drachen 11. Ordnung

Was beim zweiten Beispiel mit LDRACHE und RDRACHE entsteht, wird ab Niveau 7 deutlich. Probieren Sie aber auch:

```
fs cs pu setpos [80 50] pd ht LDRACHE 5 11
```

Bei diesen Programmen sollte man sinnvollerweise die Schildkröte mit ht verstecken, weil das wackelnde Dreieck sonst eher störend wirkt.

Hilbert- und Sierpinski-Kurven

Die nächsten zwei Beispiele sind Kurven, mit denen Flächen in regelmäßiger Weise ausgemalt werden können. Das erste hat der deutsche Mathematiker David Hilbert, das zweite der Pole Wraclaw Sierpinski erfunden.

```
to HILBERT :laenge :niveau :drehsinn
if :niveau =0 [stop]
if :drehsinn [lt 90] [rt 90]
HILBERT :laenge :niveau - 1 not :drehsinn
fd :laenge
if :drehsinn [rt 90] [lt 90]
HILBERT :laenge :niveau - 1 :drehsinn
fd :laenge
HILBERT :laenge :niveau - 1 :drehsinn
if :drehsinn [rt 90] [lt 90]
fd :laenge
HILBERT :laenge :niveau - 1 not :drehsinn
if :drehsinn [lt 90] [rt 90]
end
```

Die Hilbertkurven sind nicht geschlossene Kurven. Ihr Grundmuster hat die Gestalt eines U, die Sie mit

```
HILBERT 50 1 "TRUE
```

erzeugen. Die dritte Eingabe gibt einen Drehsinn vor, bei TRUE ist das U nach rechts, bei FALSE nach links offen. Beim nächsten Niveau werden insgesamt vier solche Figuren mit beiden Orientierungen erzeugt und durch einen Vorwärtsschritt aneinandergefügt.

Die Verwendung der Größe Drehsinn hat den Vorteil, daß für beide möglichen Orientierungen dasselbe Programm benutzt werden kann, und nicht wie beim vorangegangenen Beispiel zwei Prozeduren LDRACHE und RDRACHE benötigt werden. Drehsinn steht für einen Wahrheitswert, nämlich TRUE oder FALSE für dessen Umkehrung die LOGO-Vokabel not sorgt.

```
not "TRUE
```

Antwort: FALSE

Besonders reizvoll ist es, die Hilbertkurven mit wachsender Verschachtelungstiefe in einer Grafik zusammenzustellen. Wenn die Schrittlänge jeweils halbiert und der Anfangspunkt geeignet verschoben wird, windet sich die Hilbertkurve der nächst höheren Ordnung um die des vorangegangenen Niveaus. Dazu dienen die Rahmenprogramme HILBERTK und HILBERTPLOT.

```
to HILBERTPLOT; Verschachtelte Hilbertkurven
fs cs HILBERTK 75 -75 1 150 6
end

to HILBERTK :x0 :y0 :i :h :n
if :n=:i [stop]
pu setpos se :x0 :Y0 pd
HILBERT :h :i "TRUE
HILBERTK :x0+:h/4 :y0-:h/4 :i+1 :h/2 :n
end
```

HILBERTPLOT löscht den Grafikschirm und setzt die richtige Anfangsbedingung so, daß alle folgenden Kurven bis zur fünften Ordnung gezeichnet werden.

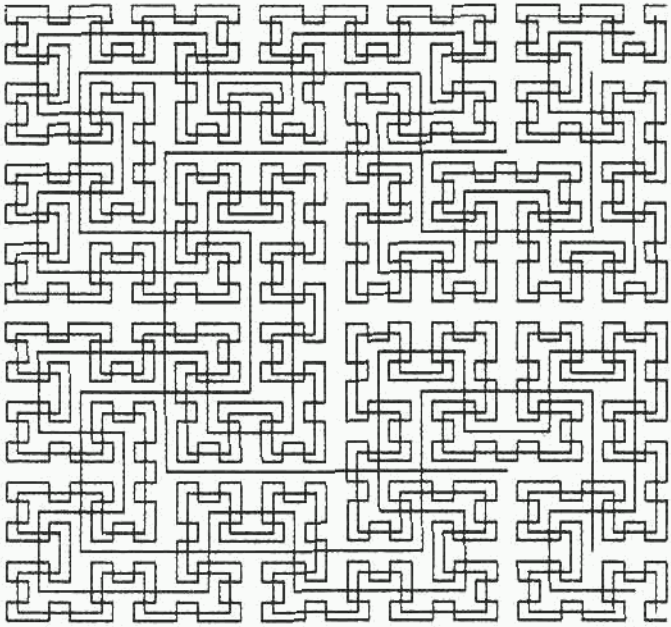


Abb. 14.7: Überlagerte HILBERT-Kurven

Mit den Sierpinski-Kurven kann man Flächen noch schöner ausmalen, der rekursive Aufbau ist aber etwas schwieriger zu verstehen. Zunächst die grundlegende Prozedur zur Erzeugung einer Sierpinski-Kurve:

```
to A :i;Sierpinski-Teilstück
if :i=0 [stop]
A :i-1 rt 135 fd :l lt 45
A :i-1 fd :l2 lt 180
A :i-1 rt 135 fd :l lt 45
A :i-1
end
```

Sehen Sie zunächst einmal über den am Anfang einer jeden Zeile erfolgenden Selbstaufruf hinweg. Die Prozedur A beinhaltet dann drei Vorwärtsschritte mit den beiden Längen l und l2. Dazwischen erfolgen jeweils Drehungen um 45 Grad, wobei Sie bedenken müssen, daß 180 minus 135 auch 45 ergibt! Probieren Sie aus:

```
make "l 50 make "l2 :l*1.4142 A 1
```

Die beiden verwendeten Längen verhalten sich zueinander wie die Seite eines Quadrats zur Diagonalen. Beobachten Sie nun, was durch den Selbstaufruf jeweils zu Beginn der vier Zeilen bewirkt wird.

```
fs cs A 2 cs A 3 ...
```

Die eigentliche Sierpinski-Kurve besteht aus vier solchen durch die Prozedur A erzeugten Stücken, die zueinander im rechten Winkel stehen. Durch einen Vorwärtsschritt mit der jeweils absoluten Richtung 45, 135, -135, -45 Grad werden diese vier Stücke miteinander zur geschlossenen Sierpinski-Kurve verbunden.

```
repeat 4 [a :i rt 135 fd :l lt 45]
```

Dabei bedeutet i das Rekursionsniveau. Daß die Drehungen sowohl bei A als auch in dieser repeat-Zeile gerade richtig gewählt sind, stellt man am besten durch Ausprobieren fest.

Auch bei diesen Kurven ist es reizvoll, mehrere Ordnungen zu überlagern. Das wird von den beiden Rahmenprogrammen SIERPIN und SIERPINSKI geleistet.

```
to SIERPIN :l :l2 :x0 :y0 :i
if :i=:n [stop]
pu setpos se :x0 :y0 pd
repeat 4 [A :i rt 135 fd :l lt 45]
SIERPIN :l/2 :l2/2 :x0-:l2/2 :y0+:l2/4 :i+1
end
```

```
to SIERPINSKI :GROESSE :N
fs cs ht
SIERPIN :GROESSE :GROESSE*1.4142 -:GROESSE 2.1*:GROESSE 1
end
```

Mit SIERPINSKI werden der Anfangspunkt und die beiden Längen für das erste Niveau an SIERPIN übergeben; in dieser Prozedur werden die Kurve auf dem Niveau I in der repeat-Zeile erzeugt und anschließend die nächste Ebene mit geeignet verschobenem Anfangspunkt und halbierten Längen in Gang gesetzt. Sie können es aufrufen mit:

```
SIERPINSKI 60 5
```

Auch hier sollten sich die Kurven höherer Ordnung um die der vorangegangenen Ordnung herumwinden.

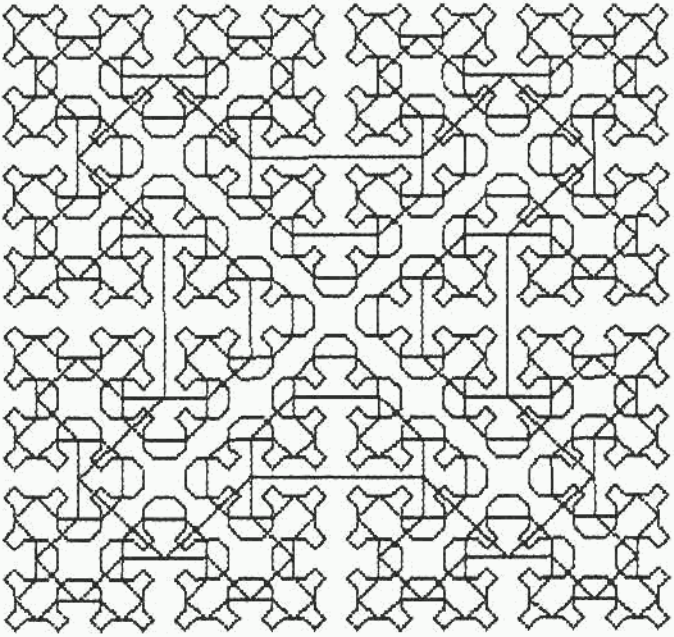


Abb. 14.8: Überlagerung von SIERPINSKI-Kurven

14.5 Grafik mit Flächen

Während in Abschnitt 14.2 Grafiken mit Einzelpunkten auf dem Bildschirm ausfgebaut wurden, waren Streckenzüge in regelmäßiger Anordnung die grafischen Hilfsmittel im vorangegangenen Abschnitt. Nun sollen Flächenelemente als Instrumente eingesetzt werden. Flächelemente werden zunächst einmal durch geschlossene Linien wie Vielecke und Kreise dargestellt.

Mit Flächen können in der Ebene Muster aufgebaut werden, denken Sie etwa an die Struktur von Bienenwaben. Flächen sind aber auch wesentlich bei der Darstellung von räumlichen Objekten in einer Zeichnung. Neben der Darstellung des Flächenrandes ist auch eine Gestaltung des Inneren der Fläche von grafischer Wirkung.

Um das Flächeninnere hervorzuheben, kann die Fläche schraffiert, mit regelmäßigen oder zufälligen Punktverteilungen versehen oder auch als Ganze in verschiedenen Farbtönen dargestellt werden. DR LOGO auf dem Schneider CPC bietet keine eigenen Kommandos zu Gestaltung von Flächen. Etwas besser dran sind die Besitzer eines Joyce, weil in dessen LOGO-Version ein Befehl zum Ausfüllen von Flächen vorgesehen ist.

Ausfüllen von Flächen beim Schneider Joyce (nicht CPC)

Flächen können mit dem Kommando fill ausgemalt werden. Probieren Sie etwa:

```
to sechseck :l
repeat 6 [fd :l rt 60]
end

sechseck 40 pu rt 60 fd 40 pd fill
```

Das mit sechseck erzeugte Sechseck wird vollständig ausgefüllt, wobei beim Joyce keine Farbvariationen möglich sind. An der aufrufenden Zeile erkennen Sie die Voraussetzungen:

- Die Turtle muß sich innerhalb eines geschlosenen Linienzuges befinden, wobei sie keine Verbindung zum Rand haben darf; sie muß sozusagen frei in der Fläche stehen.
- Der Status des Zeichenstifts muß pd (oder pe bzw. px) sein.

Man kann den Vorgang am Bildschirm verfolgen: Es werden übereinander Punkte gesetzt, bis bereits gesetzte Punkte erreicht werden. Wird fill benutzt, wenn keine geschlossene Berandung existiert, dann wird das Ausfüllen immer weiter ausgedehnt - möglicherweise über den ganzen Bildschirm hinweg. Beachten Sie, daß vor Beendigung von fill nicht unterbrochen werden kann, wewegen fill mit gewisser Vorsicht benutzt werden muß.

Als einfaches Muster können Sie etwa versuchen:

```
to secken :l
sechseck :l rt 60 pu fd :l pd fill fd :l
sechseck :l rt 60 pu fd 2*:l pd
end
```

Aufruf: `fs cs repeat 3 [secken 30]`

Schraffieren von Flächen

Mit Schraffuren können Flächenelemente in vielfältiger Weise gestaltet werden. Ohne Unterstützung durch eigene Sprachelemente, die relativ schnell in der Ausführung sein können, ist das Schraffieren bei beliebigen Berandungen sehr aufwendig, weil ja die dazu verwendeten Strecken auf das Innere beschränkt werden müssen.

Ein bekanntes Kippbild soll als Beispiel dienen.

```
to PARA :w :n :s
repeat :n [fd :l lt :w fd :s rt :w bk :l lt :w fd :s rt :w]
end

to PAR :w :n :s
fd :l lt :w fd :ns rt :w bk :l lt :w bk :ns rt :w
PARA :w :n :s
end
```

Das Programm PARA erzeugt eine Schraffur mit der äußeren Gestalt eines Parallelogramms. Probieren Sie etwa aus:

```
make "l 60 PARA 60 12 2 PARA 120 4 12
```

Beim ersten Fall ist die Schraffur so eng, daß der Eindruck einer ausgefüllten Fläche mit zusätzlicher Struktur entsteht, während beim zweiten der Weg des Zeichenstifts deutlich zu erkennen ist. Die Orientierung der begrenzenden Paralellogramme zeigt die Bedeutung des Winkels als erster Eingabe von PARA an. Die Prozedur PAR zeichnet nun zuerst noch zusätzlich eine Begrenzung ein.

```
to FELD :y
if :y < -180 [stop]
pu setx 320 sety :y seth 0 pd
repeat 8 [PAR 120 2 12 PARA 60 12 2]
pu setx 320 sety :y - :ns - :l seth 0 pd
repeat 8 [PARA 60 12 2 PAR 120 2 12]
FELD :y - :ns - :l - :l
end

to KIPPBILD
(local "l "ns)
make "l 40 make "ns 48
fs cs ht
FELD 150
end
```

Die Grafik wird in FELD erzeugt. In den beiden repeat-Zeilen wird jeweils eine Reihe von aneinanderhängenden, verschieden orientierten und unterschiedlich schraffierten Parallelogrammen gezeichnet. Die beiden Reihen werden dann so gegeneinander versetzt, daß sich das Muster in Abb. 14.9 ergibt.

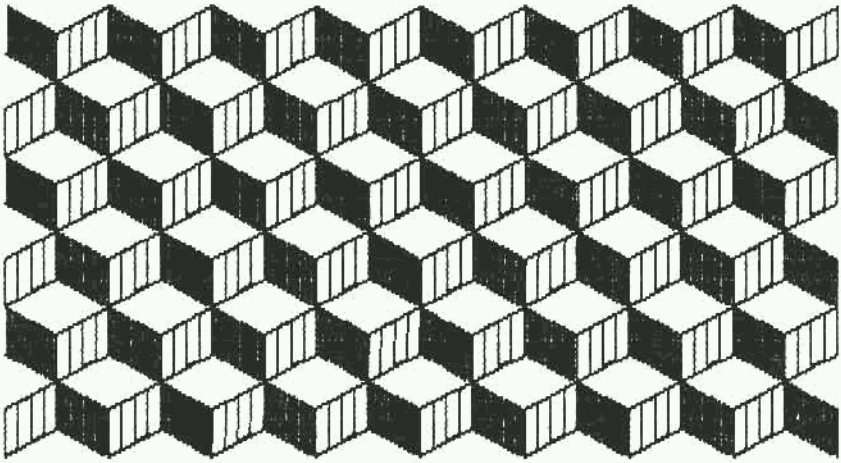


Abb. 14.9: Kippbild aus schraffierten Paralellogrammen

Solche Bilder können in doppelter Weise räumlich gesehen werden, deswegen die Bezeichnung Kippbild. Das Umschalten zwischen den beiden Interpretationen geschieht natürlich im Gehirn des Betrachters; manchmal gibt es auch Schwierigkeiten, von der einmal gewonnenen Perspektive abzulassen. In der durch KIPPBILD, das offensichtlich nur noch ein Rahmenprogramm ist, erzeugten Grafik finden Sie dunkle Flächenstücke, die Sie einmal bei der räumlichen Deutung von oben, zum zweiten auch von unten betrachten.

Wenn Sie sich vorzustellen versuchen, daß die intensiv schraffierten Flächenstücke von rechts beleuchtet werden, dann erblicken Sie die dunklen von unten. Wenn diese dagegen von links beleuchtet werden, schauen Sie von oben auf die dunklen Flächen.

Weil man beim Betrachten von Grafiken oft nur Ausschnitte auf einmal wirklich räumlich interpretiert, kann das räumliche Sehen auch leicht in die Irre geführt werden. Vielleicht haben Sie schon einmal Grafiken des holländischen Malers Cornelius Escher gesehen, die daraus ihren besonderen Reiz beziehen. Eine einfache Grafik dieser Art sei noch angefügt.

Es sollen 18 Kreisscheiben mit Radius R übereinandergelegt werden. Die Mittelpunkte liegen an den Ecken eines 18-Ecks mit der Seitenlänge L. Werden Kreise räumlich übereinandergelegt, so bleibt vom untersten Kreis nur noch ein sichelförmiger Ausschnitt sichtbar.

```
to SICHEL :l
rt 110
repeat 4 [rt 10 bk :l]
repeat 24 [fd :l lt 10]
lt 110
repeat 8 [lt 10 bk :l]
repeat 24 [fd :l rt 10]
end
```

Aufruf: `SICHEL 10`

Das Grundprogramm SICHEL bildet mit Hilfe von zwei Kreisausschnitten eine Sichel. Die Kreisausschnitte haben jeweils einen Öffnungswinkel von 240 Grad, sind aber gedreht. Wird SICHEL neunfach wiederholt, so entstehen die 18 Sicheln in Abb. 14.10.

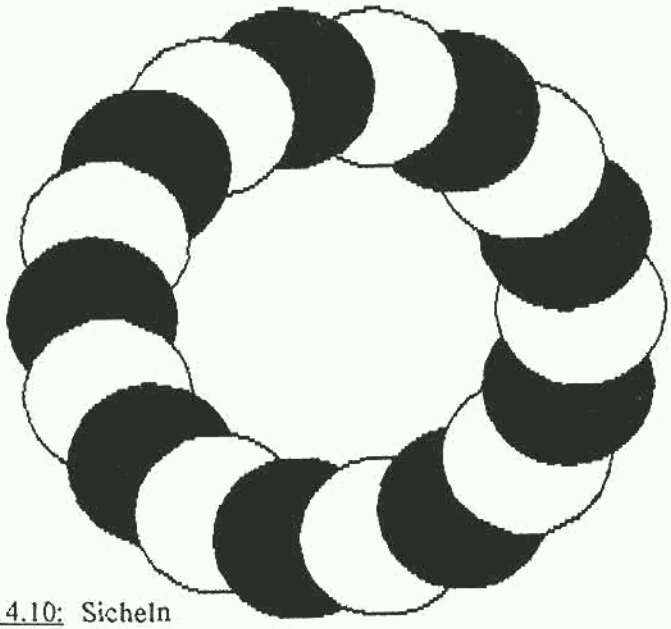


Abb. 14.10: Sicheln

In Abb. 14.10 ist jede zweite Sichel vollständig ausgefüllt, dieses Bild ist auf dem Schneider-Joyce gezeichnet worden. Das zugehörige Program lautet:

```
to RINGE :l
make "p tf
make "r 5.7296 * :l make "r2 2.8648 * :r
pu fd :r2 + 0.866 * :r pd seth 65
repeat 24 [fd :l rt 10 ]
repeat 9 [SICHEL :l]
pu setpos :p seth 0 fd :r2 seth 100;fuer Joyce
repeat 9 [pd fill pu fd :r rt 20 fd :r rt 20];fuer Joyce
end
```

Zum Ausfüllen der Fläche wird die Turtle nach dem Zeichnen der Sicheln noch einmal auf einem 18-Eck entlang geschickt, dessen Eckpunkte innerhalb der Sichelflächen liegt. Aufrufen können Sie das Programm beispielsweise mit der Zeile:

```
cs pu ht setpos [-100 100] pd RINGE 5
```

Beim CPC sind die beiden Zeilen mit dem Hinweis auf den Joyce im Kommentar wegzulassen, wobei die dann natürlich auch nicht ausgefüllt werden. Vielleicht probieren Sie aber die folgende Version aus, bei der jede zweite Sichel schraffiert wird. Das Programm wird dadurch wesentlich umständlicher, obwohl das Schraffieren sicher nicht perfekt ist.

```
to SICHEL :l
rt 110 repeat 4 [rt 10 bk :l] repeat 8 [fd :l lt 10]
bk :r3 fd :r3 repeat 2 [fd :l lt 10]
repeat 40 [fd :r bk :r rt 5] lt 200;schraffieren
fd :l lt 10 fd :r2 bk :r2
repeat 13 [fd :l lt 10]
lt 110
repeat 8 [lt 10 bk :l] repeat 24 [fd :l rt 10]
end

to RINGE :l
make "r 5.7296 * :l make "r2 4.5 * :l make "r3 3.5 * :l
rt 5 repeat 24 [fd :l rt 10]
repeat 9 [SICHEL :l]
end
```

Die Schraffierbewegungen sind an den Kombinationen der Form fd :r bk :r zu erkennen.

14.6 Zeichnen in drei Dimensionen

Dreidimensionale Grafik mit dem Computer ist ein sehr interessantes, aber auch schwieriges Thema, zu dem gewisse mathematische Grundlagen benötigt werden, auf die hier aber nicht eingegangen werden kann. Es soll deshalb nur ein kurzer Ausflug in die Darstellung der dritten Dimension unternommen werden. Anstelle mathematischer Berechnungen wollen wir die Möglichkeiten der Turtle-Grafik nutzen.

Fluchtpunkte

Eine mögliche Darstellung der dritten Dimension in der Zeichenebene ist die Verwendung eines Fluchtpunktes. Dabei scheinen sich alle Geraden parallel zur dritten räumlichen Achse im Fluchtpunkt zu schneiden.

Die räumliche Darstellung ist dann schon bei der einfachen Gestalt eines Würfels nicht leicht zu konstruieren, weil Strecken, die im Bild etwas weiter hinten liegen, kürzer werden, die Verkürzung aber von der jeweiligen Lage zum Fluchtpunkt abhängt.

Grundlage ist das Programm FLUCHT.

```
to FLUCHT :seite :richtung :zielx :ziely :fluchtx :fluchty
seth towards se :fluchtx :fluchty
fd :seite seth :richtung
label "anfang
fd 1 if parallelp :zielx :ziely :fluchtx :fluchty 2 [
[op piece 1 2 tf] [go "anfang]
end

to PARALLELP :x1 :y1 :x2 :y2 :tol
make "re remainder (towards se :x1 :y1) - (towards se :x2 :y2) 180
if :re < 0 [op -:re > -:tol] [op :re < :tol]
end
```

FLUCHT zeichnet von der Position der Schildkröte aus zunächst eine Strecke der Länge seite in Richtung des Fluchtpunktes. Danach schlägt die Schildkröte die als Eingabewert angegebene Richtung ein. Nach jedem kleinen Vorwärtsschritt wird sowohl der Fluchtpunktes als auch der durch zielx, ziely angegebene Punkt auf dem Bildschirm anvisiert. Wenn beide Punkte in zueinander parallelen Richtungen gepeilt werden, wird FLUCHT beendet. Die Koordinaten des dabei zuletzt erreichten Punktes werden als Ergebnis zurückgegeben. Der ganze Vorgang ist relativ langsam, erfordert aber keinerlei Mathematik.

Das Prüfwort PARALLELP untersucht, ob die Schildkröte die beiden Punkte (x1,y1) und (x2,y2) in gleicher bzw. entgegengesetzter Richtung sieht. Wegen der begrenzten Auflösung kann die Übereinstimmung der Winkel nur innerhalb einer Toleranzgrenze geprüft werden. Je enger die angepeilten Punkte der Schildkröte benachbart sind, desto ungenauer wird die Winkelbestimmung.

```
to WUERFEL :seite :fluchtx :fluchty
ht
home repeat 4 [fd :seite lt 90]
home fd :seite
make "u FLUCHT :seite/2 180 0 0 :fluchtx :fluchty
home fd :seite
make "o FLUCHT :seite/2 270 -:seite :seite :fluchtx :fluchty
setpos se -:seite :seite
end
```

Das Programm ist für einen Fluchtpunkt gedacht, der auf der rechten Bildschirmseite liegt. Der Aufruf kann beispielsweise erfolgen durch:

```
cs WUERFEL 100 200 200
```

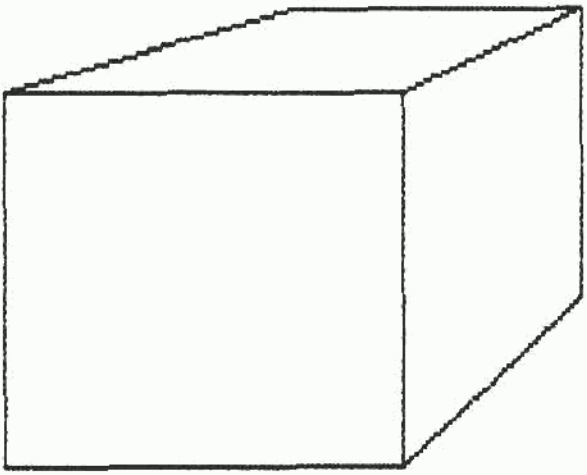


Abb. 14.11: Würfel mit Fluchtpunktperspektive

Vektoren und Punkte im dreidimensionalen Koordinatensystem

Es ist nicht einfach, sich die Lage von Punkten, Strecken und Körpern im Raum vorzustellen. Eine Darstellung in der Ebene mit Hilfe einer Parallelprojektion ist zwar leicht zu konstruieren, aber auch zeitraubend. Hier kann der Computer helfen, wobei im folgenden nur ein Beispiel für eine Perspektive dargestellt werden soll.

```
to ACHSEN :n :einheit
seth 0 ht pd
repeat 2 [ursprung achse :n :einheit rt 90]
seth 60 ursprung achse :n 0.7*:einheit
end
```

```
to ACHSE :n :l
fd :n * :l
rt 30 repeat 3 [rt 120 fd 10] lt 30;Pfeilspitze
URSPRUNG
repeat :n-1 [pu fd :l pd rt 90 fd 3 bk 6 fd 3 lt 90]
end

to URSPRUNG
pu setpos [-50 -50] pd
end
```

Die Prozedur ACHSEN zeichnet drei Koordinatenachsen, wobei die zweite scheinbar nach hinten in den Bildschirm hineinläuft. Die in diese Richtung weisenden Strecken werden mit einem Richtungswinkel 60 Grad parallel zueinander dargestellt, die Streckenlängen werden einheitlich mit dem Faktor 0.7 multipliziert. Winkel und Faktor können natürlich verändert werden, wenn der räumliche Eindruck unbefriedigend erscheint. Die Verkürzung in der dritten Raumrichtung ist notwendig, weil diese Richtung sonst überdimensional erscheinen würde.

Auf den Achsen werden Einheiten markiert, wobei der Eingabewert n in ACHSEN die gewünschte Länge der Achse in der Einheit angibt. Der Koordinatenursprung wird durch die Prozedur URSPRUNG festgelegt, wodurch er leicht zu verändern ist.

Um einen durch drei Koordinaten gegeben Punkt auf dem Bildschirm darzustellen, muß die Schildkröte vom Ursprung ausgehen und Vorwärtsbewegungen parallel zu den Koordinatenachsen machen. Die Länge der Bewegung ist bei der ersten und dritten Achse unmittelbar durch die Koordinate gegeben, bei der zweiten muß allerdings wieder der zusätzliche Faktor 0.7 berücksichtigt werden. Die Einheit (in LOGO-Grafikschritten) wird in den Prozeduren als globale Größe aufgefaßt und muß also zu Beginn mit make "einheit ... festgelegt werden.

Diese Methode mit Hilfe der TURTLE-Grafik errechnet nicht die Koordinatenwerte in der Zeichenebene, sondern ermittelt sie vielmehr durch zeichnerische Konstruktion.

```
to PUNKT :punkt
URSPRUNG
op VEKTOR :punkt
end

to VEKTOR :punkt
st pu
seth 90 fd :einheit * item 1 :punkt
seth 60 fd .7 * :einheit * item 2 :punkt
seth 0 fd :einheit * item 3 :punkt
ht pd
fd 2 bk 4 fd 2 rt 90 fd 2 bk 4 fd 2;Kreuzchen
op piece 1 2 tf end

to VEKTOR.RICHTUNG :vektor
URPSRUNG
op towards PUNKT :vektor
end

to STRECKE :anfang :ende
(local "ak "ek)
make "ak PUNKT :anfang
make "ek PUNKT :ende
pd setpos :ak
end
```

Ein Punkt bzw. ein Vektor wird im dreidimensionalen Raum durch drei Zahlen festgelegt. Diese Angaben werden bei den Prozeduren zu einer Liste zusammengefaßt. PUNKT und VEKTOR geben die Bildschirmkoordinaten von Punkten in der Form eines Satzes aus. Sie können für weitere Zeichnungen benutzt werden.

PUNKT ruft VEKTOR auf; der Unterschied besteht nur darin, daß die Schildkröte zuerst zum Ursprung bewegt wird, während bei VEKTOR von der aktuellen Position aus begonnen wird. Die Koordinaten eines Punktes beziehen sich immer auf den Ursprung, während ein Vektor an beliebige Punkte im Raum angeheftet werden kann.

Die Prozedur STRECKE bestimmt die Lage zweier im Raum vorgegebener Punkte auf dem Bildschirm und verbindet sie dann durch eine Strecke. Hiermit können auch Flächen und Körper im Raum aufgebaut werden.

Die Prozedur VEKTOR.RICHTUNG gibt als Ergebnis die Richtung auf dem Bildschirm aus - immer bezogen auf die Nullstellung der Schildkröte senkrecht nach oben - so daß z.B. Geraden mit vorgegebener Richtung gezeichnet werden können.

Wenn Punkte bzw. Vektoren auf dem Bildschirm bestimmt werden, sind die Bewegungen der Turtle sichtbar und der Zeichenstift oben. Der Punkt wird durch ein Kreuzchen markiert, die Turtle wird anschließend versteckt, damit die genaue Lage des Punktes sichtbar ist.

Beispiele:

```
make "einheit 30 STRECKE [2 2 1] [-2 2 -2]
```

Es wird eine Strecke gezeichnet.

```
PUNKT [1 -1 2] VEKTOR [0 1 1]
```

Es werden zwei Punkte dargestellt. Der zweite Punkt ergibt sich durch Vektoraddition. Damit kann die Addition von Vektoren im dreidimensionalen Raum betrachtet werden. Der zweite Punkt, der als Endpunkt der Vektoraddition zustande kommt, muß nach den Regeln der Vektorrechnung auch in einem Stück erzeugt werden mit

```
PUNKT [1 0 3]
```

Als Beispiel für die Darstellung eines Körpers kann folgende Prozedur dienen:

```
to TETRAEDER :g
(local "a "b "c "d)
make "a (se Ø Ø 1.5 * :g)
make "b (se :g :g Ø )
make "c (se -:g Ø Ø )
make "d (se Ø.5 * :g -:g Ø )
setpc 2 DS :a :c :b Ø.Ø5
setpc 3 DS :b :d :c Ø.Ø5
setpc 1 DS :a :c :d Ø.Ø2
end

to DS :a :b :c :df
pu setpos :a pd setpos :b setpos :c setpos :a
end
```

Für die Darstellung der Grenzflächen wird hier in TETRAEDER mit dem Kommando setpc von der Farbgebung Gebrauch gemacht, was natürlich auch nur bei einem CPC mit Farbmonitor richtig zur Geltung kommt.

In der Prozedur DS wird jeweils eine der begrenzenden Dreiecksseiten gezeichnet. Mit der Formulierung einer eigenen Tätigkeit für diese Aufgabe wird auch das Ziel verfolgt, die Darstellung der Oberflächen anschließend noch verbessern zu können. Die letzte Eingabe hat zunächst keine Bedeutung.

Schraffieren von Dreiecken

Die Dreieckseiten sollen schraffiert werden. Weil diese Aufgabe auch unabhängig von der Darstellung eines Tetraeders von Interesse ist, soll die Problemlösung hier getrennt präsentiert werden.

```
to SCHRAFF :f :ax :ay :bx :by :cx :cy
if :f > 0.99 [stop]
pu setpos se :f * :ax + :cx :f * :ay + :cy
pd setpos se :f * :bx + :cx :f * :by + :cy
schraff :f + :df :ax :ay :bx :by :cx :cy
end

to DS :a :b :c :df
pu setpos :a pd setpos :b setpos :c setpos :a
SCHRAFF :df (first :a) - (first :c) (last :a) - (last :c) !
(first :b) - (first :c) (last :b) - (last :c) first :c !
last :c
end
```

Eine Fläche wird durch eine Schar paralleler Strecken schraffiert. Bei Dreiecken erzeugt man am einfachsten Parallelen zu einer der drei Seiten. Die Anfangs- und Endpunkte werden in der Prozedur SCHRAFF bestimmt und die Strecken dann mit setpos gezeichnet.

Die Bestimmung der Parallelen ist einfacher, als man vielleicht zunächst erwarten würde. In der Ausdrucksweise der Mathematik gehen sie durch zentrische Streckungen bezogen auf den gegenüberliegenden Eckpunkt aus der Dreiecksseite hervor.

Probieren Sie das Verfahren einmal unabhängig von der Darstellung des Tetraeders an einem einzigen Dreieck aus:

```
DS [0 0] [100 100] [-100 100] 0.1
```

Die letzte Eingabe bestimmt die Anzahl und den Abstand der schraffierenden Parallelen, was man am besten durch Experimentieren studiert.

Beim Schneider-Joyce kann für eine Dreiecksfläche auch der Befehl fill zum Ausfüllen benutzt werden. Die drei letzten Zeilen von Tetraeder könnten dann etwa ersetzt werden durch:

```
DS :a :c :d 1
pu setpos :c seth 90 fd 10 pd fill pu
DS :a :c :b 0.1
DS :b :d :c 0.1
```

Damit wurde die Darstellung in Abb. 14.11 gewonnen.

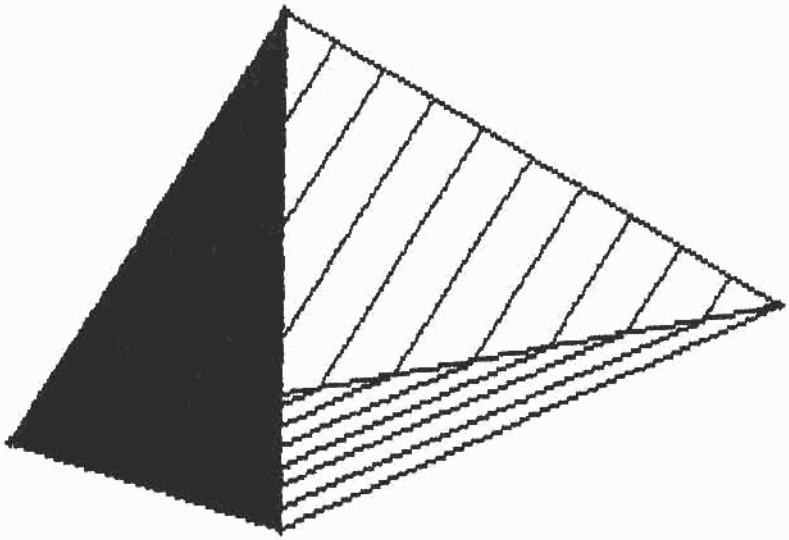


Abb. 14.12: Darstellung eines Tetraeders

Lektion 15: Vertieftes Arbeiten mit Listen

Neue Sprachelemente:
or, and

Programme:
REPLACE, BEHALTE, EINFUEGEN, ENTFERNE, ABSATZTAUSCH, BEGRUESSUNG (mit Hilfsprogrammen; Dialogprogramm mit Schablonen für Redewendungen)

Zeichenfolgen ohne trennende Leerzeichen heißen in LOGO Wörter, unabhängig von ihrer inhaltlichen Bedeutung. Aus Wörtern können Sätze gebildet werden, indem sie durch Leerzeichen getrennt und als Satz durch eckige Klammern zusammengefaßt werden. Ein Satz wird dann Liste genannt. Listen können nicht nur Wörter als Elemente haben, sondern die Elemente können auch selbst wieder Listen sein.

15.1 Ersetzen von Listenelementen

Zur Manipulation von Sätzen sind Operationen notwendig, um einzelne Wörter, also Elemente einer Liste, durch andere ersetzen zu können. Hierzu müssen die einzelnen Elemente herausgelöst, überprüft, gegebenenfalls durch andere Wörter ersetzt und anschließend wieder eingefügt werden. Löschen von Elementen und Einfügen vor oder nach bestimmten Elementen sind Manipulationen, die nach einem ähnlichen Schema ablaufen.

Herausgelöst werden Listenelemente mit der LOGO-Prozedur first oder entsprechend von hinten her mit last. Wieder eingefügt werden kann ein Wort dann mit fput:

```
fput (OPERATION first :LISTE) bf :LISTE
```

oder auch mit se

```
se (OPERATION first :LISTE) bf :LISTE
```

Je nach Wirkung der Prozedur OPERATION wird mit dieser Zeile das erste Element manipuliert. Soll etwa das Wort durch Operation gespiegelt werden, könnte OPERATION durch die Prozedur SPIEGEL ersetzt werden, die in Lektion 9 entwickelt wurde.

```
to SPIEGEL :WORT
if emptyp :WORT [op :WORT]
op word SPIEGEL bf :WORT first :WORT
end
```

Beispiel:

```
make "SATZ [MANIPULATION VON SAETZEN]
se SPIEGEL first :SATZ bf :SATZ
```

Antwort: `[NOITALUPINAM VON SAETZEN]`

Um die Manipulation an jedem Wort des Satzes vorzunehmen, muß die Veränderung des ersten Wortes auf den Satzrest selbst wieder angewandt werden. Der Satzrest ist um ein Wort kürzer geworden, damit kann eine rekursive Ersetzung formuliert werden.

```
to ERSATZ :SATZ
if emptyp :SATZ [op :SATZ]
op se TAUSCH first :SATZ ERSATZ bf :SATZ
end

to TAUSCH :WORT
op SPIEGEL :WORT
end
```

Aufruf: `ERSATZ [Heute scheint die Sonne]`
Antwort: `[etueH tniehcs eid ennoS]`

Aufruf: `ERSATZ ERSATZ [Heute scheint die Sonne]`
Antwort: `[Heute scheint die Sonne]`

Die Prozedur TAUSCH hätte in ERSATZ natürlich gleich SPIEGEL heißen können. Das in der Prozedur ERSATZ verwendete Schema läßt sich für alle Manipulationen an Sätzen übertragen, deswegen wurde diese Prozedur allgemeiner gehalten.

Die Manipulation der Liste geschieht in ERSATZ in einem typisch rekursiven Verfahren: Es wird immer das erste Element einer Liste entfernt und mit der restlichen Liste weiter verfahren. Erst wenn das letzte Listenelement erreicht ist, wird die modifizierte Liste nach und nach mit se wieder zusammengesetzt. Typisch für solche Vorgehensweisen sind die LOGO-Vokabeln first, bf, last, bl, fput, lput.

Das beschriebene Verfahren sollte bei allen LOGO-Versionen funktionieren. DR LOGO stellt aber als mächtige LOGO-Implementation noch weitere Hilfsmittel zur Verfügung, die einen nichtrekursiven Zugriff auf Listenelemente gestatten.

- item :N :SATZ liefert das N-te Listenelement.
- piece :I :N :SATZ liefert die Teilliste vom I.ten bis zum N.ten Element.

Ersetzen und Einfügen

Man kann zwar mit item direkt auf eine Listelement zugreifen, es gibt aber keine LOGO-Vokabel, mit dem umgekehrt ein Listenelement ersetzt bzw. verändert werden kann. Deswegen formulieren wir eine eigene Vokabel REPLACE für diesen Zweck, auf die dann später auch zurückgegriffen wird.

```
to REPLACE :i :L :W
if :i > 1 [make "W lput :W piece 1 :i-1 :L]
if :i < count :L [op se :W piece :i+1 count :L :L] [op :W]
end
```

```
to EINFUEGEN :i :L :W
if :i > 1 [make "W lput :W piece 1 :i-1 :L]
if :i > count :L [op :W] [op se :W piece :i count :L :L]
end
```

Mit REPLACE wird in der Liste L das i-te Element durch den Wert von W ersetzt. Im Geegensatz zu dem oben beim Programm ERSATZ angewandten rekursiven Zerlegen und Zusammenbauen, wird hier die vor bzw. hinter dem ausgewählten Element stehenden Teile der Liste mit Hilfe von piece abgetrennt. Anschließend wird die Liste zusammen mit der Eingabe W mit se wieder zusammengesetzt.

Die Prozedur EINFUEGEN unterscheidet sich nur geringfügig in der dritten Zeile von REPLACE, wobei das mit i ausgewählte Listenelement mit in den hinteren Teil der Liste hineingenommen wird. Mit EINFUEGEN wird demnach die dritte Eingabe an die i.te Stelle der Liste L eingefügt.

In beiden Prozeduren kann die Liste L sowohl ein einfacher Satz als auch eine zusammengesetzte Liste sein.

```
REPLACE 2 [a x c d e] "b Ergebnis [a b c d e]
REPLACE 3 [[Wasser Kraft] [Stein Kohle] [Kern Energie]]
[Solar Energie]
```

Ergebnis: `[[Wasser Kraft] [Stein Kohle] [Solar Energie]]`

```
EINFUEGEN 2 [Johann von Goethe] "Wolfgang
```

Ergebnis: `[Johann Wolfgang von Goethe]`

Gelegentlich will man auch ein Listenelement entfernen, was natürlich analog wie das Ersetzen gemacht werden kann.

```
to ENTFERNE :i :L
if :i =1 [op bf :L]
if :i = count :L [op bl :L] [op se piece 1 :i-1 :L piece
:i+1 count :L :L]
end

ENTFERNE 3 [Rot Grün Blau]
```

Ergebnis: `[Rot Grün]`

Löschen eines Teils der vorhandenen Prozeduren

Es kommt häufiger vor, daß sich im Arbeitsspeicher viele nicht mehr benötigte Prozeduren angesammelt haben. Es kann dann recht mühsam sein, diese nach und nach zu löschen, wobei es auch leicht passieren kann, daß Prozeduren vesehentlich gelöscht werden. Die Prozedur ENTFERNE kann an dieser Stelle eine nützliche Anwendung finden.

```
to BEHALTE :l
(pr [Sollen alle Prozeduren ausser] :l [geloescht werden? j/n]
if lc first rq = "n [stop]
er REST se :l [ENTFERNE BEHALTE REST] glist ".DEF
end

to REST :l :p
if emptyp :l [op :p]
if memberp first :l :p [op REST bf :l ENTFERNE where :p]
[op REST bf :l :p]
end
```

Nach einer Sicherheitsabfrage werden alle Prozeduren ausser den in der Eingabeliste aufgezählten und den hier benötigten Prozeduren REST, ENTFERNE und BEHALTE gelöscht. Wollen Sie auch diese löschen, dann muß die vierte Zeile in BEHALTE lauten

```
er REST :l glist ".DEF
```

Mit der Anweisung glist ".DEF (beachten Sie die Großschreibung von DEF) wird eine Liste aller momentan vorhandenen, selbstdefinierten Prozeduren erzeugt.

Ersetzen eines Suchwortes durch ein anderes Wort

Jeder kennt heute die per Computer als Massendrucksache erzeugten Briefe, die den Anschein einer persönlichen Anrede erwecken. An den Textstellen für den Namen des Adressaten kann im Ausgangstext ein Platzhalterwort geschrieben werden. Das Programm muß dann dieses spezielle Wort suchen und durch den jeweiligen Namen ersetzen. Zunächst soll nur ein einziger Satz ohne Satzzeichen betrachtet werden.

Es soll ein bestimmtes Wort in einem Satz durch ein anderes ersetzt werden. Unter der Verwendung von REPLACE ergibt sich die folgende Lösung.

```
to AUSTAUSCH :satz :alt :neu
if memberp :alt :satz [
[op AUSTAUSCH REPLACE where :satz :neu :alt :neu] [op :satz]
end
```

Diese Formulierung nutzt die Fähigkeit des Prüfwortes memberp aus. DR LOGO muß ja dann auf die Suche gehen, ob das unter dem Namen alt eingegebene Wort in der Liste satz vorkommt. Die Antwort ist bei memberp natürlich TRUE oder FALSE; man kann nun aber mit der LOGO-Vokabel

```
where
```

auch erfahren, an welcher Stelle das Wort gefunden wurde. Diese Angabe wird dann beim Aufruf von REPLACE zum Ersetzen durch das Wort mit dem Namen neu benötigt. Weil where nur das jeweils erste Auftreten meldet, muß die Abfrage solange wiederholt werden, bis das Suchwort nicht mehr vorhanden ist.

Häufig hat das Suchwort nur die Funktion eines Platzhalters, der für ein Verb, einen Namen usw. steht. Man kann dann einfach nur ein Sonderzeichen als Platzhalter benutzen, beispielsweise &. *Bei Sonderzeichen, die eine spezielle Bedeutung in LOGO* haben, beispielweise *, + usw., muß ein Backslash vorangestellt werden, was auch durch CONTROL+Q (beim Joyce durch ALT+Q oder f3-Taste) erzeugt werden kann.

```
AUSTAUSCH [Hans Verb Erika] "Verb "liebt Antwort: [Hans liebt Erika]
AUSTAUSCH [Wir & gern] "& "singen
```

Antwort: `[Wir singen gern]`

```
AUSTAUSCH [& , Sie haben gewonnen!] "& "Herr\ Zwerghuber
AUSTAUSCH [Paul & Monika] "& "ist\ ganz\ verrueckt\ nach
```

Mit REPLACE kann ein Listenelement nur durch genau ein neues Element ersetzt werden. Wenn ein Wort durch mehrere Wörter ersetzt werden soll, kann man aus den Wörtern eine Liste bilden und dann das alte Wort durch die neue Liste ersetzen. Das Ergebnis ist dann aber kein Satz, sondern eine verschachtelte Liste, was an den eckigen Klammern sichtbar wird.

Bei den letzten zwei Beispielen wird aus mehreren Wörtern ein einziges Wort im Sinne der LOGO-Syntax dadurch gebildet, daß vor das Leerzeichen ein Backslashzeichen gesetzt wird. Man könnte das Programm AUSTAUSCH aber mit Hilfe der Prozedur EINFUEGEN geeignet verallgemeinern.

Sollen mehrere Wörter im Satz ersetzt werden, so muß die Schleife in AUSTAUSCH für jedes Wort durchlaufen werden. Bei der folgenden Erweiterung des Programms werden unter den Namen ALT bzw. NEU jeweils selbst Sätze erwartet, die dann mit first und bf zerlegt werden.

```
to ATAUSCH :satz :alt :neu
if emptyp :alt [op :sat]]
op ATAUSCH AUSTAUSCH :satz first :alt first :neu bf :alt bf :neu
end
```

Beispiel:

```
ATAUSCH [& scheint $] [& $] [Heute die\ Sonne]
```

Antwort: `[Heute scheint die Sonne]`

15.2 Ersetzen in Texten

Ein umfangreicherer Text sollte in Kapitel, Abschnitte und Sätze gegliedert werden. Für die Syntax von LOGO muß die Gliederung durch eckige Klammern erfolgen, da die Satzzeichen zum Inhalt des Textes gehören. Um in einem Absatz ein Wort zu ersetzen, geht man den Text Satz für Satz durch und tauscht in jedem Satz.

```
to ABSATZTAUSCH :liste :alt :neu
if emptyp :liste [op []]
op fput ATAUSCH first :liste :alt :neu ABSATZTAUSCH bf
:liste :alt :neu
end

to TEXT1
op [[& ist eine vielseitig verwendbare !
Programmiersprache.]
[Schon Kinder koennen & lernen und ihre eigenen !
Programme in & schreiben.]
[Aber auch fuer komplizierte Problemstellungen eignet !
sich &, weil in & Prozeduren, Rekursionen und !
Listenverarbeitung moeglich sind.]
[Programme koennen in & durch Programme untersucht und !
veraendert werden.]
[Fuer Fragestellungen der kuenstlichen Intelligenz !
bietet & damit gute Voraussetzungen]]
end
```

Hier wurde ein Probetext als Prozedur TEXT1 programmiert, wobei der gesamte Text als eine Zeile eingegeben werden muß, was bei dieser Länge nur mit dem LOGO EDITOR, also nach der Eingabe von ed "TEXT1 geht. Probieren Sie nun

```
ABSATZTAUSCH TEXT1 [& &,] [LOGO LOGO,]
```

Vielleicht sind Sie nicht einverstanden und plädieren für

```
ABSATZTAUSCH TEXT1 [& &,] [PASCAL PASCAL,]
```

15.3 Suchen im Kontext: Intelligente Dialogprogramme

Viele Anwenderprogramme, etwa Computerspiele oder kommerzielle Programme, führen zu Beginn einen Dialog mit dem Benutzer. Dabei werden auch Angaben wie Name und Datum verlangt, die dann in späteren Phasen vom Programm benutzt werden. Der Benutzer hat kaum Freiheit bei der Eingabe und er wird in der Regel durch genaue Anweisungen oder Bildschirmmasken geführt; bei falscher Form der Eingabe muß diese wiederholt werden, und sogar fatale Fehler können auftreten.

Ganz anders verlaufen Dialoge zwischen Menschen ab. Die angesprochene Person kann auf eine Frage mit sehr unterschiedlich formulierten Antworten reagieren und die Verständigung klappt trotzdem. Wir sind eben auf ein breites Spektrum sprachlicher Ausdrucksformen eingestellt und haben gelernt, die Bedeutung von Wörtern und Begriffen aus ihrem Kontext heraus zu beurteilen. Im Zweifelsfall wird nachgefragt, wobei die Nachfrage auch möglichst mehr als nur eine Wiederholung der Ausgangsfrage sein sollte.

Es ist sicher wünschenswert, die Dialoge in Computerprogrammen diesem Vorbild stärker anzugleichen. Eine kleine Vorübung hierzu soll ein Programm sein, das bestimmte Wörter aus einem Text herausfinden kann, die in vorgegebenen Kontexten vorkommen dürfen.

Am Anfang eines Dialogs soll die Frage nach dem Namen stehen:

Wie heisst du?

Ein primitives Programm würde als Antwort nur den Namen selbst, also genau ein Wort akzeptieren. Ein intelligentes Programm sollte dagegen auch Sätze der folgenden Art richtig verarbeiten können:

Ich heisse Linda.
Mein Name ist Anton.
Hans ist mein Name.
Ich denke, ich heisse Karl.
Ja, Jens ist mein Name.

Die Antworten sind zwar verschieden, die Namen kommen aber jeweils in typischen Wortumgebungen vor. Es soll eine Prozedur entworfen werden, die eine derartige Wortumgebung in einem eingegebenen Satz feststellt. Das als Name zu interpretierende Wort soll dann herausgefunden und für die weitere Verwendung bereitgestellt werden.

Die möglichen Wortumgebungen dienen als Schablone, mit der der eingegebene Satz untersucht wird. Nur Teile eines Satzes müssen auf die Schablone passen, während die für den Zusammenhang unwichtigen Satzteile durch ein Jokerzeichen repräsentiert werden sollen, etwa durch ein $-Zeichen. Das als Namen zu deutende Wort wird durch ein weiteres Jokerzeichen, etwa das &-Zeichen, dargestellt. Gültige Schablonen wären danach:

Ich heisse & , Ich bin $ &

Antworten der Form "Ja, ich heisse Helmut, Ich denke, ich heisse Alice, wie heisst du, Ich bin die Lisa!" würden zu diesen Schablonen passen. Das gesuchte Programm muß die Schablone über den eingegebenen Satz hinwegziehen. Die verbindlichen Worte müssen übereinstimmen, während bei Jokerzeichen jedes Wort akzeptiert wird. Falls alle Vergleiche die Übereinstimmung

mit der Schablone bestätigen, wird das an der dem Jokerzeichen & entsrechenden Stelle zu findende Wort als Name akzeptiert.

Natürlich kann ein solches Programm auch zu unsinnigen Ergebnissen führen, da die Namen nur auf Grund ihrer Stellung im Satz identifiziert werden. Hier kann man über eine Plausibilitätsprüfung des Ergebnisses nachdenken, mit der vorhersehbare Irrtümer ausgeschlossen werden können.

Beim Vergleich mit der Schablone kann man sich zunächst auf das erste verbindliche Wort der Schablone konzentrieren. Der Satz wird mit MEMBERP auf dieses Wort hin untersucht; wenn es tatsächlich vorkommt, so wird mit der LOGO-Vokabel WHERE die Stellung im Satz ermittelt.

```
to SUCHE :satz :schab
if or first :schab = "$ first :schab = "& [op SUCHE bf :satz bf :schab]
if memberp first :schab :satz [op where] [op 0]
end
```

Die zweite Zeile dient in SUCHE zum Überlesen von Jokerzeichen. Wenn das Ergebnis nicht Null ist, also das erste verbindliche Wort gefunden wurde, soll der Rest verglichen werden. Dazu dient die Prozedur VERGLEICHEP.

```
to VERGLEICHEP :satz :schab
if emptyp :schab [op "TRUE]
if emptyp :satz [op "FALSE]
local "wort make "wort first :schab
if (or :WORT = first :satz :wort = "$ :wort = "&)
[op VERGLEICHEP bf :satz bf :schab] [op "FALSE]
end
```

VERGLEICHEP ist ein Prüfwort und liefert also TRUE oder FALSE als Resultat. Falsch ergibt sich, wenn der Satz vor der Schablone endet und das jeweils erste Wort der Schablone weder ein Jokerzeichen ist noch mit dem ersten Wort des Satzes übereinstimmt. Wahr kommt heraus, wenn die Schablone ohne eine der Bedingungen für Falsch durchlaufen wurde.

In der drittletzten Zeile werden drei Bedingungen mit or logisch miteinander verknüpft.

SUCHE und VERGLEICHEP werden im Programm FINDEN aufgerufen. Die Eingabe ist für FINDEN eine Liste von Schablonen, die rekursiv durchlaufen werden.

```
to FINDEN :schablone
if emptyp :schablone [op "]
(local "schab "st "s)
make "schab first :schablone
make "st SUCHE :satz :schab
if (or not memberp "& :schab :st=0 :st=:sl)!
[op FINDEN bf :schablone]
make "S where
if VERGLEICHEP piece :st+1 :sl :satz bf :schab !
[op item :st+:s - 1 :satz] [op FINDEN bf :schablone]
end
```

In den Größen st bzw. s wird festgehalten, wo das erste verbindliche Wort im Satz erkannt wurde bzw. wo das Jokerzeichen & für den gesuchten Namen in der Schablone steht.

Beachten Sie, daß der Selbstaufruf mit op eingeleitet werden muß, weil FINDEN eine Prozedur mit Ergebnis ist. Wird ein Wort als Name identifiziert, so ist dieses das Resultat, andernfalls wird das leere Wort zurückgegeben.

Das Prüfwort VERGLEICHEP wird in der siebten Zeile aufgerufen.

Ein Sonderfall muß noch bei der Untersuchung des Satzes berücksichtigt werden. Die Antwort auf die Frage "Wie heißt du?" kann nur aus dem Namen selbst bestehen. Deswegen wird FINDEN in einem kleinen Rahmenprogramm FINDE aufrufen. Um einen kompletten Dialog führen zu können, wird dann noch ein Steuerprogramm BEGRUESSUNG formuliert.

```
to BEGRUESSUNG
label "anf pr [Wie heisst du?]
make "name FINDE ENTSETZEN rl []
if emtyp :name !
[pr [Verzeihung. Ich habe den Namen nicht verstanden.]!
go "anf]
pr (se [Nett mit dir zu sprechen] :name "! )
end

to FINDE :satz
local "sl make "sl count :satz
if :sl=1 [op first :satz] [op FINDEN SCHABLONEN]
end
```

In Rahmenprogramm BEGRUESSUNG wird die Antwort auf die Frage "Wie heisst du?", die mit rl erfragt wird, zunächst einmal der Prozedur ENTSETZEN überantwortet. Erstes Ziel von ENTSETZEN ist die Entfernung von Satzzeichen aus der Antwort. Satzzeichen hängen im allgemeinen ohne Zwischenraum an Wörtern und werden deshalb von LOGO als Bestandteile von Wörtern angesehen. Dann werden einmal Namen möglicherweise mit einem angehängten Satzzeichen identifiziert, zum anderen macht es für den Vergleich mit der Schablone einen Unterschied, ob beispielsweise "mein Name" mitten im Satz, vor einem Nebensatz oder am Satzende steht.

```
to ENTSETZEN :l :a
if emptyp :l [op :a]
local "w make "w lc first :l if ascii last :w < 65 make "w bl :w
op ENTSETZEN bf :l se :a :w
end
```

Daneben gibt es in DR LOGO für die Schneider-Computer Groß/Kleinschreibung und beides wird auch unterschiedlich behandelt. Deswegen werden die Wörter der gegebenen Antwort in der Prozedur ENTSETZEN grundsätzlich in die Kleinschreibung übersetzt. Dann muß bei den Schablonen nicht darauf Rücksicht genommen werden, daß Wörter mit gleicher Bedeutung je nach Stellung im Antwortsatz unterschiedlich geschrie-

ben werden. Dafür muß dann am Ende der gefundene Name wieder in die Großschreibung übertragen werden.

Am Ende von FINDE wird das Programm FINDEN mit SCHABLONEN als Eingabe aufgerufen. Ohne einleitenden Doppelpunkt muß es sich um einen Prozedurnamen handeln. Es wird also noch eine Prozedur benötigt, die eine Liste von Schablonen ausgibt. Wenn der Vorrat an Schablonen erweitert werden soll, muß nur diese Prozedur verändert werden.

```
to SCHABLONEN
op [[& ist $ name] [ich heisse &] [& heisse ich]!
[mein name ist &]]
end
```

Das Programm kann jetzt immerhin auf so unterschiedliche Antworten wie die folgenden richtig reagieren.

Hans
Wie bitte, Werner ist der Name
Anton ist mein Name, bitte sehr
Monika heisse ich schon immer
Mein Gott, Walter heisse ich natuerlich
Hallo, Gantenbein ist mein Name
Jo mai, mein Name ist Xaver, wos host du gdocht?
Guten Tag, ich heisse Helmut

Die Namen Hans, Werner, Monika ... und Helmut werden als solche erkannt.

Das vorgestellte Programm ist noch lange nicht perfekt! Werden beispielsweise zwei Worte für den Namen, etwa Vor- und Nachname, eingegeben, wird nur das erste Wort als Namen identifiziert. Es ist auch noch nicht möglich, überlappende Schablonen zu verwenden, also etwa Ich bin & und Ich bin der &. Wem diese Fassung nicht gefällt, der soll das Dialogprogramm selbst mit mehr Intelligenz ausstatten.

LOGO und My Fair Lady

Ein berühmtes Dialogprogramm ist das von J. Weizenbaum entwickelte Programm Eliza, so benannt nach der Hauptfigur in Shaws Komödie Pygmalion, der Vorlage für My Fair Lady. Dieses Programm soll in der Lage gewesen sein, einen scheinbar vernünftigen und verständigen Dialog zwischen Arzt und Patient - von der Arztposition her - zu führen.

Programme dieser Art nutzen die Tatsache aus, daß in bestimmten Zusammenhängen charakteristische Floskeln und Redeweisen verwendet werden. Das Bemühen von Therapeuten zielt auch zunächst darauf ab, den Gesprächspartner zu Äußerungen zu veranlassen, selbst aber unverbindlich zu bleiben. Die Reaktionen des Arztes lassen in der Anfangsphase kaum erkennen, ob er die Bedeutung der Patientenäußerungen überhaupt erkannt hat.

Es ist deshalb auch nicht sehr schwer, einfache Programme für den Beginn eines solchen Dialogs zu schreiben. Die Reaktionen des Programms auf die Antworten des Patienten können am Anfang in standardisierten Floskeln wie "Erzählen Sie weiter", "Wie lange geht das schon so?", "Haben Sie noch andere Beschwerden?" oder auch in abgewandelten Wiederholungen der Patientenäußerungen bestehen. Damit die Abfolge der Antworten variiert, kann im Programm mit Hilfe von Zufallszahlen aus einem Repertoire von Sätzen bzw. Satzteilen ausgewählt werden. Schwieriger wird es schon, eingegebene Äußerungen grammatisch richtig zu wiederholen, weil zumindest die Person im Satz und damit auch die Form der Verben richtig abgeändert werden müssen.

Im Programm BEGRUESSUNG wurde schon mehr an Intelligenz des Programms gewonnen, weil dort die Bedeutung von Wörtern im Kontext der Wortumgebung herausgefunden wird. Für ein intelligentes Programm zum Dialog mit Patienten könnten Sie sich überlegen, welche typischen Wortumgebungen bei Patientenäußerungen vorkommen könnten. Wenn daraus Schablonen entwickelt werden, mit denen man die Bedeutung von Wörtern

in solchen Antworten möglicherweise herausfinden kann, dann sind auch intelligentere Reaktionen des Programms möglich.

Eine gezielte Frage, z.B. "Wie heißt du?", schränkt die darauf von einem ernsthaften Dialogpartner zu erwartenden Antworten schon weit ein. Man kann sich deshalb auch vorstellen, daß ein die Position des Arztes simulierendes Programm die Antworten des Gesprächspartners stark vorprägt.

Seine Gesundheit einem solchen Programm anzuvertrauen, wird man zu Recht als töricht bezeichnen. Eine nützlichere Anwendung derartiger Techniken kann man bei Programmen zu Lehr- bzw. Schulungszwecken sehen. Je flexibler, abwechslungsreicher und intelligenter Lernprogramme auf Reaktionen des Benutzers eingehen, um so größer kann der Lernerfolg sein.

Wenn hier von intelligenten Programmen gesprochen wird, so ist das nur im Vergleich mit "dummen Programmen" gerechtfertigt. Von Intelligenz im Sinn menschlicher Intelligenz ist man hier noch weit entfernt. Über sogenannte künstliche Intelligenz von Computerprogrammen wird zur Zeit allerdings intensiv geforscht.

Ein wesentlicher Teil von Intelligenz ist das Wissen über viele Fakten und deren Zusammenhänge. Vor allem der effiziente Zugriff auf vorhandenes Wissen stellt ein enormes Problem dar. In der nächsten Lektion soll ein ganz bescheidener Blick auf dieses Thema geworfen werden. Fortschritte gibt es im Bereich der künstlichen Intelligenz bisher nur auf relativ eng begrenzten Feldern. Die für Arbeiten auf diesem Gebiet besonders geeignete Programmiersprache heißt LISP. LOGO ist ein Abkömmling von LISP.

Übungen

1) Name, Wohnort, Geschlecht, Alter, Beruf, Einkommen bilden einen Satz. In einer Liste sind diese Datensätze für eine Personengruppe enthalten. Formulieren Sie ein Programm, mit dem Personen nach mehreren wählbaren Merkmalen herausgesucht werden können (z.B. alle weiblichen Personen zwischen 20 und 50 Jahren).

2) Eine eingegebene Dezimalzahl soll im Klartext als Zahlwort ausgegeben werden. Lösen Sie zuerst das Problem für zwei-, dann für dreistellige Zahlen.

3) Erweitern Sie das Dialogprogramm durch eine zweite Frage der Art: Wie geht es Ihnen? Nach der Antwort des Benutzers auf beide Fragen, nämlich nach Namen und Wohlbefinden, soll das Programm eine Reaktion ausgeben, die beide Antworten verwendet.

Lektion 16: Datenstrukturen in LOGO

Neue Sprachelemente:
thing, list, listp, throw, TOPLEVEL

Programme:
SIEB (Primzahlsieb), TABELLE, TABAUS, TABEIN, TABDRUCK (Tabellenfunktionen), EINGABE (Tabellenprogramm), TIERERATEN (Aufbau und Erweiterung von Baumstrukturen), QUICKSORT (Sortierprogramm)

Zahlen, Wörter und Zeichen sind Daten. Im einfachsten Fall gehört zu jedem einzelnen Datum ein eigener Name. Wird aus mehreren Wörtern ein Satz gebildet, wobei die Wörter in LOGO genauso gut die Bedeutung von Zahlen haben können, gehört der Name des Satzes zu einer Datenmenge. Diese muß eine Struktur aufweisen, da die Daten im Computer so gespeichert werden müssen, daß gezielt darauf zugegriffen werden kann.

Ein aus Wörtern aufgebauter Satz ist ein einfaches Beispiel einer Liste. Diese bilden eine sehr komplexe Grundlage für Datenstrukturen in LOGO, weswegen im Grundwortschatz keine anderen Datenstrukturen vorgesehen sind. Listen sind ein sehr vielseitiges Instrument, man kann damit komplexe Datenstrukturen aufbauen.

Während alle LOGO-Versionen mit Listen arbeiten, gibt es in DR LOGO und einer Reihe weiterer Implementationen mit den sogenannten Eigenschaftslisten noch eine besondere Spielart von Listen, die sich auch für Datenstrukturen eignet.

Bei den Wörtern eines Satzes ist die Wortlänge sehr unterschiedlich, so daß man die interne Darstellung im Computer nicht von vorneherein festlegen kann. Im Gegensatz dazu kann man beim Arbeiten mit Zahlen die Länge und damit die interne Darstellung von Anfang an fest vereinbaren. Daten, die im Computer alle einen einheitlich großen Platz beanspruchen und deren Anzahl schon vorher bekannt ist, lassen sich einfacher und

schneller verwalten als Listen. Bei Aufgabenstellungen, bei denen intensiv mit vielen Zahlen gerechnet werden muß, sind LOGO-Programme relativ langsam und werden dazu kaum eingesetzt.

16.1 Einfache Listen: Primzahlsieb

Die einfachste Form der Liste ist ein aus einzelnen Wörtern bestehende Satz, also eine Aneinanderreihung von Elementen, die durch Leerzeichen getrennt werden.

Als weiteres Beispiel hierfür soll ein Programm zur Berechnung von Primzahlen betrachtet werden. Hierzu werden aus der Menge der natürlichen Zahlen bis zu einer bestimmten Größe nach und nach die teilbaren Zahlen herausgestrichen, man spricht von einem Primzahlsieb.

In der klassischen Version von Eratosthenes werden jeweils die Vielfachen einer gefundenen Primzahl gestrichen; mit 2 beginnend sind das also alle geraden Zahlen, anschließend alle Vielfachen von 3, 5 usw. Sucht man die Primzahlen bis 100, dann ist man schon nach dem Streichen der Vielfachen von 7 fertig. Eine Zahl, die nicht Primzahl ist, muß ja mindestens zwei Primfaktoren haben, wobei der kleinere dann unterhalb von 10 liegen muß. Im allgemeinen Fall braucht deswegen das Streichen nur bis zur Quadratwurzel der größten Zahl durchgeführt werden.

Wir legen uns nun eine Liste der gestrichenen Zahlen an. Um festzustellen, ob eine natürliche Zahl Primzahl ist, muß dann nur in der Liste der gestrichenen Zahlen nachgesehen werden. Außer der Zahl 2 sind alle Primzahlen ungerade, weshalb die geraden Zahlen gleich weggelassen werden können.

```
to SIEB :N;Primzahlen bis N
(type 2 ",)
make "PRIM []; Liste der gestrichenen Zahlen
make "I 1
make "W SQRT :N
repeat (:N/2-1) [make "I :I+2 !
if not memberp :I :PRIM [(type :I ",)
if :I < :W [STREICHE :I :I+:I]]
end

to STREICHE :J :Z
if :J > :N [stop]
make "PRIM se :PRIM :J
STREICHE :J+Z :Z
end
```

(Die Funktion SQRT zur Berechnung der Quadratwurzel wird weiter unten betrachtet.)

In der Prozedur STREICHE werden die ungeraden Vielfachen von J gestrichen, also der Liste PRIM als Wort hinzugefügt. Die Vielfachen werden durch Addition beim rekursiven Aufruf erzeugt. In dieser Version werden Zahlen mehrfach gestrichen, 15 beispielsweise als Vielfaches von 3 und von 5. Beim Aufruf mit SIEB 500 werden beispielsweise 240 Eintragungen erzeugt, obwohl es nur 163 unterschiedliche gibt. Durch eine Abfrage kann die mehrfache Aufnahme in die Liste unterbunden werden, die dritte Zeile von STREICHE müßte dazu lauten

```
if not memberp :J :PRIM [make "PRIM se :PRIM :J]
```

Damit werden die Liste kürzer und die Ausführungszeit wegen der zusätzlichen Abfrage mit memberp geringfügig länger. Man kann übrigens auch das ganze Verfahren noch verbessern, worauf aber nicht eingegangen werden soll, weil das Programm dann erheblich komplizierter würde.

Die Primzahlbestimmung geschieht im Programm SIEB in der repeat-Zeile unter Benutzung einer iterative Lösung. Damit nur die ungeraden Zahlen untersucht werden müssen, wird die

Größe I jeweils um zwei erhöht. Ist die Zahl bereits gestrichen, liefert das Prüfwort memberp die Antwort ja (TRUE), durch not wird daraus FALSE, und der Rest der repeat-Zeile wird nicht ausgeführt. Ist die Zahl nicht gestrichen, liefert not memberp den Wert TRUE. In diesem Fall wurde eine Primzahl gefunden, die dann ausgedruckt wird. Danach wird das Streichen der Vielfachen mit der Prozedur STREICHE in Gang gesetzt. Die zweite Eingabe von STREICHE gibt die Schrittweite beim Streichen an; es wird das doppelte der Primzahl deswegen eingegeben, damit die geraden Zahlen übergangen werden.

Mit STREICHE werden die Zahlen eigentlich nicht gestrichen, sondern notiert. Man kann natürlich auch umgekehrt vorgehen und am Anfang für alle in Frage kommenden Zahlen, z.B. die ungeraden, eine Liste anlegen, und dann in dem Sinn streichen, daß das Wegfallen geeignet notiert wird. Am Anfang könnte für alle ungeraden Zahlen ein bestimmtes Zeichen in der Liste vorgesehen werden, das dann beim Streichen umgewandelt wird. Eine bereits gestrichene wird dann nicht mit memberp sondern am jeweils zugeordneten Zeichen überprüft. Eine solche Version sei Ihnen als Übungsaufgabe überlassen.

Leider ist das DR LOGO auf den Schneider-Computern im Hinblick auf mathematische Funktionen recht dürftig ausgestattet worden, denn sogar die auf allen Taschenrechnern vorhandene Berechnung der Quadratwurzeln ist nicht eingebaut.

Quadratwurzeln werden mit dem klassischen Iterationsverfahren nach Heron-Newton berechnet.

```
to SQRT :x;Berechnung der Quadratwurzel
op MITTEL (1+:x)/2
end

to MITTEL :w
make "v (:w+:x/:w) / 2
if :w-:v < 1 [op :v] [op MITTEL :v]
end
```

Das Programm wurde schon in Lektion 12 betrachtet. Weil es im Zusammenhang mit dem Primzahlsieb nur auf die Stellen vor dem Komma ankommt, wird die Rekursion bereits abgebrochen, wenn aufeinanderfolgende Näherungen sich weniger als eins voneinander unterscheiden.

16.2 Tabellen: Logoplan

Statt sich wie beim vorangegangenen Beispiel Vielfache von Primzahlen auf einer Liste zu notieren, könnte man z.B. einen Kassenzettel erstellen. Mit den aufgelisteten Preisen können dann beispielsweise die Gesamtsumme und Zwischensummen bestimmt werden.

Interessanter wird es, wenn mehrere Kassenzettel nebeneinander gelegt werden, etwa die Liste der Ausgaben für einen Haushalt monatlich. Damit werden Querverbindungen möglich, die für eine rationale Planung von Ausgaben sehr nützlich sein können.

Die Ausgaben für jeden Monat müssen hierzu jedesmal in einer festgelegten Reihenfolge in die Listen eingetragen werden. Bei einer fehlenden Ausgabe für einen bestimmten Posten muß dann auch ein entsprechender Vermerk, in der Regel die Zahl 0, eingetragen werden. Zusammen ergibt sich somit eine zweidimensionale Tabelle, bei der die Zeilen für bestimmte Posten stehen, z.B. Lebensmittel oder Autokosten, während die Spalten der Tabelle dann zu je einem Monat gehören.

In der ersten Spalte einer solchen Tabelle wird man einen erläuternden Begriff für die jeweiligen Zeilen im Klartext setzen, damit die Bedeutung sichtbar bleibt. In der ersten Zeile werden dann ebenfalls Überschriften eingetragen. Eine weitere Spalte bietet sich für die Jahressumme aller Ausgaben eines Postens, entsprechend als letzte Reihe die monatliche Summe aller Ausgaben an. Auch an Spalten bzw. Zeilen für prozentuale Angaben könnte man denken.

Der Computer hat in diesem Zusammenhang einmal eine Verwaltungsaufgabe. Die Tabellen können gedruckt, verändert, abgespeichert und damit aufgehoben werden. Daneben kann der Computer zu Berechnungen, etwa zur Addition von Monats- bzw. Jahressummen, eingesetzt werden. Bei jeder Änderung eines Tabellenwertes können damit problemlos alle berechneten Werte auf den aktuellen Stand gebracht werden.

Programme für solche Zwecke werden Tabellenkalkulationsprogramme genannt. Bekannte Beispiele sind die professionellen Programmsysteme Visicalc und Multiplan. Diese Programme leisten noch wesentlich mehr, als hier vorgesehen werden soll.

Wie kann die Tabelle in LOGO dargestellt werden?

Die Liste der monatlichen Ausgaben ist ein Satz, hat also die einfache Form einer Liste. Die Tabelle ist auch eine Liste, deren Elemente aber nicht Wörter, sondern Sätze sind; sie ist also eine aus Listen gebildete Liste. Es kann bei Tabellen auch vorkommen, daß ein einzelnes Tabellenelement schon mehrere Wörter beinhaltet. Hier soll zunächst aber nur der Fall betrachtet werden, daß eine Eintragung durch ein Wort dargestellt werden kann, was ja auch für Zahlen zutrifft.

Tabellenfunktionen

Für das Arbeiten mit Tabellen werden Prozeduren benötigt, die häufig auftretende Operationen ausführen.

Eine Tabelle muß als spezielle Liste angelegt werden und jede Spalte gleichviele Zeilen bekommen. Alle Punkte der Tabelle müssen vorhanden sein, damit richtig auf die Werte zugegriffen werden kann. Wo kein Eintrag vorliegt, muß ein stellvertretendes Zeichen stehen. Die Prozedur TABELLE erfüllt diese Anforderungen.

```
to TABELLE :NAME :Z :S;Anlegen einer Tabelle
make :NAME []
repeat :S [make :NAME fput LEERSP :Z thing :NAME] end
```

```
to LEERSP :Z
local "SP make "SP []
repeat :Z [make "SP se ". :SP]
op :SP
end
```

Die Prozedur LEERSP liefert als Ergebnis eine Leerspalte; die ist wieder eine Liste, die aus Z Punkten besteht, wobei der Punkt hier als Stellvertreter für einen fehlenden Eintrag steht. Probieren Sie

```
LEERSP 10
```

In TABELLE werden S solcher Leerspalten zusammengesetzt zur Liste, deren Name als Wert von NAME übergeben wird.

Beachten Sie, daß der Name der Tabelle als Wert einer Größe übergeben wird, NAME enthält das Wort, mit dem die Tabelle bezeichnet werden soll. Bei make muß der Name nicht mit Anführungszeichen, sondern mit Doppelpunkt eingeleitet werden. Vergleichen Sie:

```
make "NAME ". make :NAME ".
```

Mit Anführungszeichen wird ein Name mit der Bezeichung NAME dem Punkt zugeordnet. Mit Doppelpunkt wird das Wort geholt, das unter dem Namen NAME abgelegt worden ist, in diesem Beispiel als Eingabewert für TABELLE. Dieses Wort gibt dann den tatsächlichen Namen.

Den Wert des so erzeugten Namens kann man nicht durch Vorsetzen eines weiteren Doppelpunktes bekommen. Statt dessen wird das LOGO-Wort thing benutzt.

```
thing :NAME anstelle von ::NAME
```

Grundoperationen für die Bearbeitung von Tabellen sind das Einsetzen sowie das Lesen eines Wertes an einer bestimmten Tabellenposition.

Einfach ist das Lesen eines Wertes, da man die LOGO-Vokabel item zweimal hintereinander benutzen kann.

```
to TABAUS :TAB :I :K;Tabellenposition lesen
op item :K item :I :TAB
end

to TABEIN :TAB :I :K :W;Tabellenposition setzen
if :k = 1 [op fput spein first :tab :I :W bf :TAB]
op fput first :TAB TABEIN bf :TAB :I :K - 1 :W
end

to SPEIN :SPALTE :I :W
if :I = 1 [op fput :W bf :SPALTE]
op fput first :SPALTE SPEIN bf :SPALTE :I-1 :W
end
```

Etwas schwieriger ist es, einen Wert in die Tabelle einzutragen, weil es in der vorliegenden LOGO-Version keine Umkehroperation zu item im Grundwortschatz gibt. Für diese Aufgabe wird die Prozedur TABEIN eingesetzt. Dabei wird die Hilfsprozedur SPEIN benutzt, die zum Eintragen eines Wertes in eine einzige Tabellenspalte verwendet wird. Sie werden sofort bemekren, daß TABEIN und SPEIN gleich aufgebaut sind, denn der Unterschied liegt nur in der zusätzlichen Eingabe für die Spaltennumer bei TABEIN.

Es reicht deshalb aus, die Funktionsweise von SPEIN genauer zu betrachten. Probieren Sie aus:

```
SPEIN [1 2 3 4 5] 3 "a
```

Ergebnis: `[1 2 a 4 5]`

Wenn das erste Element ersetzt werden soll, wird in der zweiten Zeile von SPEIN das vorhandene erste Listenelement durch bf :SPALTE abgetrennt und anschließend der gewünschte Wert mit fput davorgesetzt. Soll dagegen z.B. das dritte Element ersetzt werden, wird in der dritten Zeile von SPEIN das erste

Element mit first abgetrennt und der Rast mit bf :SPALTE beim rekursiven Aufruf als nunmehr zu behandelnde Spalte übergeben.

Ist der gewünschte Eintrag an der Spitze der jeweils verbliebenen Restspalte eingesetzt worden, dann kann die Rekursion abgebrochen werden. Bei der Rückkehr auf die Ausgangsebene werden dann mit fput die zuvor abgetrennten Elemente wieder an ihren ursprünglichen Platz gesetzt.

SPEIN kann übrigens auch verwendet werden, wenn die Elemente nicht Wörter, sondern Listen sind:

```
SPEIN [a b c d] 2 [5 [a d]]
```

Ergebnis: `[a [5 [a d]] c d]`

Die Prozedur TABEIN funktioniert analog, nur sind die Elemente jetzt ganze Spalten.

Eine weiter Grundfunktion dient der Ausgabe einer Tabelle.

```
to TABDRUCK :TAB :I :K :TPOS :TITEL;TABELLE AUSGEBEN
ct pr :TITEL
local "S make "S 1
repeat :K [SPDRUCK item :S :TAB :I item :S :TPOS 1 make "S :S+1]
end

to SPDRUCK :SP :I :H :V
local "Z make "Z 1
repeat :I [setcursor se :H :V + :Z pr item :Z :SP make "Z :Z+1]
end
```

TABDRUCK gibt die Tabelle TAB aus, I und K sind die Anzahl der Zeilen und Spalten, wobei man auch nur Teile der Tabelle ausgeben kann. Unter dem Namen TPOS wird eine Liste erwartet, in der die Druckbreite der einzelnen Spalten niedergelegt ist. Zusätzlich ist noch ein Titel vorgesehen, der als Überschrift den Tabellenkopf bildet.

TABDRUCK stützt sich auf die Prozedur SPDRUCK, die zur Ausgabe einer einzigen Spalte dient. Die Prozeduren für die Ausgabe sind zu Abwechslung einmal iterativ, d.h. ohne Rekursion formuliert. Einfach Wiederholungen müssen eben nicht unbedingt mit einer Rekursion programmiert werden.

Zum Abschluß der Tabellenfunktionen sei noch ein Eingabeprogramm vorgestellt, mit dem die Tabelle TAB eingegeben und verändert werden kann.

```
to EINGABE :TAB :TPOS :TITEL;Tabelle eingaben
(local "smax "lmax)
make "smax count thing :tab
make "lmax count first thing :TAB;Groesse von TAB
TABDRUCK thing :TAB :lmax :smax :TPOS :TITEL
label "ANFANG make "SN 1
catch "HOME [repeat :smax [TBEINGABE :SN make "SN :SN+1] EINZEILE 20]
go "ANFANG
end

to TBEINGABE :SN
SPEINGABE 1 item :SN :TPOS
end

to SPEINGABE :L :tp
if :L > :lmax [stop]
(local "pp "e "so "alt)
make "pp se :tp :l+1
REVERSE setcursor :pp
make "alt item :L item :SN thing :TAB type :alt
einzeile 20 make "e rq
if emptyp :e [go "fort] if :e=char 94 [DRUCK throw "HOME]
if not :e = char 92 [make "alt :e make :TAB TABEIN thing :TAB :L :SN :e]
label "fort
if :e = char 92 [stop] [SPEINGABE :L+1 :tp]
end
```

Bevor die Tabelle für Änderungen bereitsteht, wird sie ausgedruckt. Dann beginnt die Eingabe Spalte für Spalte in der Zeile von EINGABE, die mit

```
catch "HOME
```

beginnt. Das catch "HOME ermöglicht, die Tabelleneingabe vorzeitig abzubrechen, ohne dabei über alle Ebenen der Prozeduraufrufe zurück gehen zu müssen.

Die eigentliche Eingabe erfolgt im Programm SPEINGABE. Das gerade betrachtete Feld wird zunächst in der Variablen "alt aufgehoben und errneut auf dem Bildschirm ausgegeben. Davor wird das Programm REVERSE aufgerufen, das nur aus einer einzigen Anweisung besteht.

```
to REVERSE
type char 24;CPC
;type word char 27 "p;Joyce
end
```

Mit der Ausgabe des ASCII-Zeichen Nr. 24 wird beim Schneider CPC erreicht, daß bei nachfolgenden Ausgaben auf dem Bildschirm Hintergrund und Vordergrundfarbe miteinander vertauscht werden, was das aktuelle Feld optisch hervorhebt.

Beim Schneider Joyce-Computer kann die gewünschte Umschaltung nicht mit char 24 erreicht werden. Die inverse Bildschirmdarstellung wird dort mit der Escape-Folge ESC p ein- und mit ESC q abgeschaltet, wobei das ESC-Symbol mit char 27 erzeugt wird.

Wenn das Tabellenfeld verlassen wird, muß das Feld auf dem Bildschirm wieder das normale Aussehen bekommen. Das geschieht mit der kleinen Prozedur DRUCK.

```
to DRUCK
type char 24;CPC
;type word char 27 "q;Joyce
setcursor :pp type :alt
end
```

Die eigentliche Eingabe erfolgt in einem getrennten Eingabefeld, das mit der Prozedur EINZEILE gelöscht und angesteuert wird.

```
to EINZEILE :i
setcursor se Ø :i
repeat 11 [type char 32]
setcursor se Ø :i
end
```

Neben der Eingabe eines Tabellenwertes sind noch drei Sonderfälle vorgesehen:

- Eine leere Eingabe, d.h. Drücken der Returntaste ohne Eingabe, bewirkt den Übergang zum nächsten Feld

- Die Eingabe des Pfeilnachoben-Zeichens hat einen Sprung an den Tabellenanfang zur Folge.

- Der Backslash schließlich (char 92) zieht den Übergang zur nächsten Spalte nach sich.

Natürlich könnte man genausogut andere Sonderzeichen verwenden, insbesonder beim Joyce, bei dem char 92 mit der Kombination ALT+q erzeugt werden muß.

Beispiel:

```
make "TITEL "HAUSHALT
make "TPOS [Ø 1Ø 15 2Ø 25 3Ø 35]

TABELLE "HAUSH1.HJ 4 7
EINGABE "HAUSH1.HJ :TPOS :TITEL
```

Jetzt können Sie die Positionen besetzen, die zunächst nur als Punkte erscheinen.

.	JAN	FEB	MRZ	APR	MAI	JUN
ESSEN	380	410	.	.	.	.
BUECHER	40	62	.	.	.	.
AUTO	210	180	.	.	.	.

Damit ist die Tabelle mit Überschriften versehen und teilweise schon mit Zahlen besetzt. Wird danach wieder EINGABE aufgerufen, dann erscheint die Tabelle in der soweit fertiggestellten Form. Sie kann nun weiter gefüllt und auch verändert werden. Falls das Programm auf Diskette abgespeichert wird, werden die Tabelle und die benötigten Angaben TPOS und TITEL mit abgespeichert.

Beim Aufruf von EINGABE wird nicht der Wert, sondern der Name der Tabelle übergeben. Deswegen muß in den Programmen auf die Tabelle mit thing :TAB zugegrriffen werden. Der Grund dafür ist die Tabelleneingabe mit

```
make :TAB TABEIN ...
```

Würde man hier "TAB statt :TAB schreiben, so hätte die Tabelle immer den festen Namen TAB. Die hier gewählte Lösung erlaubt es dagegen, verschiedene Tabellen zu bearbeiten.

Das Programm wird in der nächsten Lektion noch in der Weise verbessert, daß auch Berechnung innerhalb der Tabelle vorgesehen werden können.

16.3 Baumstrukturen: Erraten von Tiernamen

Angenommen, Sie spielen Karten und Ihr Partner fordert Sie auf, sich eine Karte auszusuchen. Da er mit bloßem Raten nur schlechte Gewinnchancen hat, wird er Ihnen gezielte Fragen stellen.

Einfältig wäre es, die Karten der Reihe nach vorzulegen und die Frage "War es diese?" zu stellen. Die Kunst Ihres Partners besteht vielmehr darin, möglichst wenig Fragen zu stellen und dennoch zum Ziel zu kommen. Häufig versucht er auch, die verwendete Methode zu verschleiern. Bei einem fairen Spiel könnte die erste Frage lauten: Ist die Karte rot? Nach der Farbe können Zahl bzw. Bild noch abgefragt werden.

In einem anderen Spiel soll eine Sprache erraten werden. Der Dialog könnte dabei lauten:

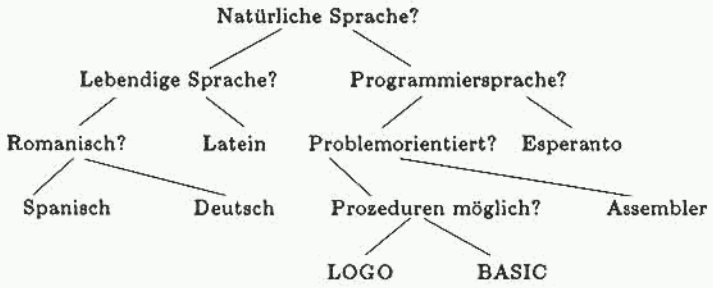
Ist die Sprache eine natürliche Sprache? - Nein. Ist es eine Programmiersprache? - Nein. Handelt es sich um Esperanto? - Ja.

Wird die Frage nach der Programmiersprache bejaht, könnte der Dialog wie folgt weiter gehen:

Ist sie problemorientiert? - Ja.
Sind Prozeduren möglich? - Ja.
Handelt es sich um LOGO? - Ja.

In diesen Dialogen durchläuft der Fragende einen Entscheidungsbaum; wenn nur wie hier Ja/Nein Antworten möglich sind, wird ein sogenannter binärer Baum verwendet. Nicht nur in spielerischen Situationen sind solche Bäume nützlich, gerade beim Abspeichern und Suchen von Daten in der EDV liefern Entscheidungsbäume effiziente Verfahren.

Grafisch kann die Struktur beim Aufsuchen einer Sprache dargestellt werden:



In einem Baum gibt es sogenannte Knoten, von denen aus Verzweigungen stattfinden können. Bei einem Baum des betrachteten binären Typs hat ein Knoten gerade zwei Nachfolger, je einen zur Antwort Ja bzw. Nein. Ein Nachfolger kann wieder ein Knoten mit Verzweigungen sein, zum Beispiel eine Frage oder auch ein Endknoten, z.B. der Name einer Sprache.

Diese Datenstruktur kann mit Listen unmittelbar beschrieben werden:

```
[[Natuerliche Sprache] [[Lebendige Sprache] [[Romanisch]
Spanisch Deutsch] Latein] [[Programmiersprache]
[[Problemorientiert] [[Prozeduren moeglich] LOGO BASIC]
Assembler] Esperanto]
```

Die Frage (Knoten mit Verzweigung) und die beiden Nachfolger (Ja-, Nein-Zweig) ergeben jeweils eine Liste mit drei Elementen. Prüfen Sie das bei der obigen Liste mit Hilfe von COUNT nach! Weil die Frage ein Satz aus mehreren Worten sein darf, ist sie als Element selbst eine einfache Liste. Sind die Nachfolger selbst Knoten mit Verzweigungen, setzen sich diese Listenelemente wieder selbst aus Listen zusammen. Entsprechend können auch Bäume dargestellt werden, die mehr als zwei Nachfolger haben.

Das Konzept der Liste eröffnet die Möglichkeit, eine Hierarchie ineinander geschachtelter Elemente aufzubauen, was gerade einer Baumstruktur entspricht. Auch die zuvor betrachteten Tabellen lassen sich als Baumstrukturen deuten. Beim ersten Knoten könnte eine der vorhandenen Spalten, in der zweiten Stufe jeweils eine Zeile ausgewählt werden. Die Baumstruktur ist aber vielseitiger verwendbar, weil Teile frühzeitig enden, während andere Teile dagegen weit verzweigen.

Die Baumstruktur ist eine effiziente Organisation, um eine gewünschte Information aufsuchen zu können. Ein Baum kann verändert und erweitert werden; daher bietet sich die Baumstruktur auch besonders dort an, wo die Informationsmenge sich dynamisch erweitern und verändern kann. Das folgende Programmbeispiel soll den Umgang mit dieser Datenstruktur verdeutlichen.

Erraten von Tiernamen

Ein beliebtes Programm, bei dem der Aufbau von Baumstrukturen gezeigt werden kann, ist ein Programm zum Erraten eines Begriffs. Als Begriffe werden gern Tiernamen genommen, weil die Tierwelt enorm vielfältig ist und deswegen weit verzweigte Bäume bildet.

Würde das Programm mit einem fest vorgegebenen Entscheidungsbaum arbeiten, wäre es sehr schnell langweilig. Viel aufregender sind Programme, die mit Hilfe von Antworten des Benutzers den Wissensbaum ständig erweitern können.

Die einfachste Form des Baums ist ein einziger Endknoten, hier also ein Tiername:

```
"Affe
```

Das Programm kann dann nur die Frage stellen:

Meinen Sie : Affe?

Bei der Antwort Nein, gibt sich das Programm damit nicht zufrieden und fragt nach dem nicht erratenen Begriff. Um für die nächste Runde besser gerüstet zu sein, wird der Benutzer aufgefordert, ein Frage anzugeben, die es gestattet, seinen Begriff vom vorgeschlagenen Begriff Affe zu unterscheiden, wobei man am einfachsten vorschreibt, ob der neue Begriff zum Ja- oder zum Nein-Zweig gehören soll. Danach hat die Liste beispielsweise die Struktur:

```
[[Kann es fliegen] Kolibri Affe]
```

Das Programm muß deshalb den Übergang:

von Affe (ein Wort)
zu [[Kann es fliegen?] Kolibri Affe] (eine Liste)

durchführen. Beidesmal ist das Wort Affe das letzte Element.

Hieraus wird schon das Verfahren zur Erweiterung des Wissens ablesbar. Das Wissen muß erweitert werden, wenn ein Endknoten erreicht und durch die Antwort NEIN verworfen worden ist. Dieser Endknoten, hier das Wort Affe, ist durch eine Liste aus drei Elementen, der Frage, dem neuen und dem alten Begriff, zu ersetzen.

Die neue Liste hat jetzt zwei Endknoten, Kolibri und Affe. Im nächsten Schritt der Erweiterung muß einer von beiden durch eine neue Liste ersetzt werden. Damit das bei einer kompliziert aufgebauten Liste möglich ist, muß erst einmal der zu ersetzende Begriff herausgesucht werden. Prinzipiell wurde dieses Problem schon beim Ersetzen von Wörtern in Texten betrachtet.

Bildung von Listen

Bevor die Prozeduren für die Erweiterung eines Wissensbaums betrachtet werden können, sollen die Möglichkeiten zum Aufbau von Listen in LOGO erweitert werden.

Listen werden durch die einschließenden eckigen Klammern erkenntlich. Aus Wörtern können Listen in der einfachen Form von Sätzen gebildet werden. Um einen Satz durch weitere Wörter zu ergänzen, ist die LOGO-Vokabel se geeignet:

```
se [Ein] [ganzer Satz] Antwort: [Ein ganzer Satz]
se "Ein [ganzer Satz] Antwort: [Ein ganzer Satz]
```

Beim zweiten Beispiel wird ein Wort, beim ersten dagegen eine Liste eingegeben, die nur aus einem Wort, besteht. se behandelt Wörter wie Listen, die aus einem Wort bestehen und faßt alle Eingaben zu einer Liste zusammen. Die Wirkungsweise von se läßt sich demnach so beschreiben:

- Die eckigen Klammern um die Eingaben werden, sofern vorhanden, zunächst einmal gestrichen. Anschließend wird alles zusammengefaßt und das Ergebnis wieder in eckige Klammern eingeschlossen.

Es werden aber nur die äußeren eckigen Klammern entfernt. Bei verschachtelten Listen bleiben die eckigen Klammern im Inneren erhalten.

```
se [Ein] [ganzer [Satz]] Antwort: [Ein ganzer [Satz]]
```

Als zweite Möglichkeit zum Listenaufbau kennen Sie die Vokabeln fput und lput. fput wirkt im allgemeinen anders als se.

- Zuerst werden die eckigen Klammern der zweiten Eingabe gestrichen. Dann wird die erste Eingabe unverändert davorgesetzt. Anschließend werden wieder eckige Klammern um das Ergebnis gesetzt.

```
fput "Ein [ganzer Satz] Antwort: [Ein ganzer Satz]
fput [Ein] [ganzer Satz] Antwort: [[Ein] ganzer Satz]
```

Bei fput macht es einen Unterschied, ob die erste Eingabe ein Wort oder eine aus einem Wort bestehende Liste ist. lput wirkt genauso wie fput, fügt aber die Eingabe am Ende an.

Neben se und fput (lput) gibt es eine weitere Möglichkeit zum Aufbau von Listen, die LOGO-Vokabel list.

```
list "Ein "ganzer Antwort [Ein ganzer]
list "Ein [ganzer] Antwort: [Ein [ganzer]]
list [Ein] [ganzer] Antwort: [[Ein] [ganzer]]
list [Ein] [ganzer [Tag]] Antwort: [[Ein] [ganzer [Tag]]]
```

Die Wirkung von list läßt sich daraus ablesen:

- Die Eingaben werden unverändert hintereinander und zum Schluß um das Ergebnis eckige Klammern gesetzt.

Mit dem Prüfwort wordp stellt man fest, ob ein Wort vorliegt. Als Gegenstück dazu gibt es auch das Prüfwort listp. listp antwortet mit TRUE, wenn die Eingabe eine Liste ist, sonst mit FALSE.

```
listp "Wort Antwort: FALSE
listp 5 Antwort: FALSE
listp [Liste] Antwort: TRUE
```

Prozeduren zur Erweiterung des Wissensbaums

Die Hauptaufgabe besteht in der Ersetzung eines Endknotens; der Knoten ist dabei ein Wort, während die Liste aus einem Fragesatz, dem neuen Begriff und dem alten Wort besteht. Dazu bietet sich die LOGO-Vokabel list an.

```
(list [Kann es fliegen] "Kolibri "Affe )
```

Dabei ist das Wort Affe schon vorhanden und beispielsweise unter dem Namen Knoten verfügbar. Die Frage kann als Liste mit rl und der neue Begriff mit rq eingelesen werden. Probieren Sie aus:

```
make "Knoten "Affe (list rl rq :Knoten)
```

Bei der Eingabe von der Tastatur müssen Sie dabei zuerst die Frage eingeben, diese mit Return abschließen und anschließend den neuen Begriff einschließlich Return eintippen.

Auf der nächsten Ebene wird es schwieriger, denn nun muß einer der beiden Endknoten, Kolibri oder Affe, durch eine Liste aus drei Elementen ersetzt werden. Dabei muß in der vorhandenen Liste das zweite Element ersetzt werden, wenn wir uns im Ja-Zweig befinden, d.h. wenn die Frage zuvor positiv beantwortet wurde. Im Nein-Zweig muß dagegen das dritte Element ersetzt werden.

```
make "Baum !
(list first :Baum (list rl rq item 2 :Baum) last :Baum)
```

Wenn Sie nun dabei eingeben:

Ist es gross (Return) Adler (Return)

so sollte Baum anschließend den folgenden Wert besitzen

```
[[Kann es fliegen] [[Ist es gross] Adler Kolibri] Affe]
```

Wenn im Nein-Zweig das dritte Element ersetzt werden soll, kann die Zeile

```
(list first :Baum item 2 :Baum (list rl rq item 2 :Baum))
```

oder etwas kürzer

```
lput (list rl rq last :Baum) piece 1 2 :Baum
```

benutzt werden.

Der Dialog zum Finden des Begriffs und gegebenenfalls zur Erweiterung des Wissensbaums müßte jetzt so durchgeführt werden:

- Ausgabe des ersten Elements der Liste, also der ersten Frage (Kann es Fliegen?).
- Je nach Antwort (Ja oder Nein) wird das zweite bzw. dritte Element als Endknoten genommen und gefragt, ob der gefundene Begriff stimmt.
- Bei negativer Antwort muß der verneinte Begriff durch die beschriebene Liste aus einer neuen Frage, einem neuen und dem abgelehnte Begriff ersetzt werden.
- Bei einer positiven Antwort wird der gesuchte Begriff als Ergebnis präsentiert.

```
to ERWEITERN :Baum
(type first :Baum "? )
if lc first rq = "j
[op (list first :Baum FRAGEN item 2 :Baum last :Baum)]!
[op lput FRAGEN last :Baum piece 1 2 :8aum]
end

to FRAGEN :Knoten
(type [Meinen Sie: ] :Knoten "?)
if lc first rq ="j [pr [Da habe ich also richtig vermutet ]!
op :Knoten]
type [Mein Irrtum! Welcher Begriff war gemeint?]
make "neu rq
(pr [Geben Sie eine Frage ein, die mit ja fuer] :neu!
[und mit nein bei] :Knoten [beantwortet werden muss!]
op (list rl :neu :Knoten)
end
```

Dort wo in der ersten Erarbeitung (list rl rq last :Baum) geschrieben wurde, wird nun das Ergebnis der Prozedur FRAGEN eingesetzt. Bei den Fragen soll der Benutzer schließlich nicht im Dunkeln tappen. Wenn der Begriff tatsächlich

schon gefunden wurde, liefert FRAGEN als Antwort einfach den eingegebenen Knoten zurück.

In diesem Fall wird mit ERWEITERN der Wissensbaum eigentlich nicht erweitert, da in der vorgeschlagenen Form der Knoten durch sich selbst ersetzt wird. Man könnte sich hier mit einer Überprüfung das Ersetzen ersparen, die dargestellte Formulierung läßt sich aber einfacher verallgemeinern.

Die Prozedur ERWEITERN funktioniert bis jetzt nur dann, wenn Baum gerade aus einer Frage und zwei Endknoten besteht. Auf den höheren Stufen müssen nach und nach Fragen gestellt werden, damit der Baum bis zu einer einzigen Frage und zwei Endknoten durchlaufen wird.

Wie Sie sicher schon vermutet haben, wird die Verallgemeinerung durch eine Rekursion erreicht. In ERWEITERN wird FRAGEN durch den Selbstaufruf, also durch ERWEITERN ersetzt. Die Rekursion muß dann abgebrochen werden, wenn ERWEITERN auf einen Endknoten trifft. Das läßt sich leicht mit dem Prüfwort wordp feststellen. Wenn tatsächlich ein Knoten erreicht ist, wird dieser an FRAGEN übergeben. Weil ja schließlich das Ergebnis von FRAGEN in den Baum eingesetzt werden soll, wird das Ergebnis von FRAGEN jetzt als Resultat von Erweitern auf der tiefsten Verschachtelungsebene zurückgegeben.

```
to ERWEITERN :Baum
if wordp :Baum [op FRAGEN :Baum]
(type first :Baum "? )
if lc first rq = "j
[op (list first :Baum!
ERWEITERN item 2 :Baum last :Baum)]!
[op lput ERWEITERN last :Baum piece 1 2 :Baum]
end
```

```
to TIERERATEN
(pr [Denken Sie sich einen Namen fuer ein] "Tier!
[aus. Ich will versuchen, ihn zu erraten ])
make "Wissen ERWEITERN :Wissen
TIERERATEN
end
```

TIERERATEN ist das Rahmenprogramm. In der ersten Begrüßungszeile ist hier Tier absichtlich aus der eckigen Klammer herausgezogen worden. Schließlich kann man auch andere Begriffstypen untersuchen. Mit POCALL "TIERERATEN wird der Ablauf deutlich:

```
Ø : TIERERATEN
1 : ...ERWEITERN
2 : ......FRAGEN
2 : ......ERWEITERN*
2 : ......ERWEITERN*
1 : ...TIERERATEN*
```

Bei jedem Durchlauf wird der Wissensbaum neu dem Namen Wissen zugeordnet. Der Baum läßt sich dementsprechend auch als Listenstruktur ausgeben und abspeichern. Wissen ist ein globaler Name, es wird hier angenommen, daß zumindest ein einziger Knoten bereits vorhanden ist. Beim Erraten von Tieren kann man etwa mit

```
make "Wissen "Affe TIERERATEN
```

starten. Es kann aber auch mit einem fortgeschrittenen Baum unter dem Namen Wissen begonnen werden, beispielsweise mit dem am Anfang des Abschnitts entwickelten Baum für die Sprachen.

Wenn man mit einem Startbegriff einen Baum aufbauen will, der schon im Entwurf vorhanden ist, muß jeweils der Begriff neu eingegeben werden, der auf der nächsten Ebene im Nein-Zweig vorkommen soll. Das Pogramm fordert nämlich immer eine neue Frage an, die für den neuen Begriff mit Ja beantwortet wird.

16.4 Verbundene Daten: Sortieren mit Schlüsseln

In einem Karteikasten sind auf einer Karteikarte meist mehrere Daten unterschiedlicher Art eingetragen. Bei einer Personaldatei sind dies z.B. neben dem Namen auch Wohnort, Postleitzahl, Straße, Geschlecht, Geburtsdatum und Geburtsort, Beruf, Gehaltsgruppe usw. Diese Daten sind teilweise Zeichenketten, Zahlen, manchmal auch Zahlen mit Struktur, wie beispielsweise das Geburtsdatum und können ein oder auch mehrere Elemente umfassen.

Alle Daten stehen auf einer Karteikarte und gehören zusammen. Man spricht deshalb von einem Verbund. Wird die Kartei benutzt, so wird manchmal die ganze Karteikarte betroffen, wenn sie z.B. gelöscht wird. Manchmal will man aber nicht den gesamten Verbund bearbeiten, sondern nur auf einzelne Teile zugreifen.

Das Listenkonzept ist flexibel genug, um mit solchen Datenstrukturen umgehen zu können. Daß die Teildaten von unterschiedlichem Typ und möglicherweise selbst aus mehreren Teilen aufgebaut sind, ist für eine Listenstruktur unerheblich. Die gesamte Karteikarte wird als Liste dargestellt, deren einzelnen Einträge Elemente der Liste sind und je nach Bedarf selbst wieder Listen sein können.

Auf die ganze Karteikarte wird mit dem Listennamen zugegriffen. Dazu muß die Liste geeignet zerlegt und wieder zusammengesetzt werden. Hierfür sind verschiedene Lösungen möglich, eine Variante soll am Beispiel des Sortierens mit einem Sortierschlüssel demonstriert werden.

Soll eine Kartei sortiert werden, kann das unter verschiedenen Aspekten geschehen. Bei einer Personenkartei wird am häufigsten eine alphabetische Sortierung nach dem Namen gewünscht; aber auch eine Sortierung nach dem Geburtsdatum oder anderen Personenkennzeichen von Interesse ist denkbar. Die Größe, die die Reihenfolge bestimmt, wird der Sortierschlüssel genannt. Beim Sortieren einer Zahlenkolonne in aufsteigender Reihen-

folge sind Daten und Schlüssel identisch. Bei der Personalkartei ist der Schlüssel nur ein Teil der Karteikarte. Trotzdem sollen natürlich beim Sortieren immer die ganzen Karteikarten umgestellt werden.

Um auf den Schlüssel zuzugreifen wird eine Prozedur SCHLUESSEL formuliert:

```
to SCHLUESSEL :l
op first :l
end
```

Hier soll der Schlüssel einfach das erste Element der Eingabe sein. In anderen Fällen kann das Ergebnis etwa mit Hilfe von ITEM aus der Eingabe herausgelöst werden. Sind Daten und Schlüssel identisch, wird die Eingabe unverändert als Ergebnis zurückgegeben.

Wenn diese Prozedur formuliert ist, können andere Programme, hier das Sortierprogramm, unter dem Namen SCHLUESSEL darauf zugreifen. Soll der Sortierschlüssel verändert werden, so ist davon nur die Prozedur SCHLUESSEL betroffen. Eine bestimmte Struktur der Daten wird bei dieser Methode durch solche Prozeduren dargestellt.

Ein Datenverbund kann in LOGO am einfachsten durch Eigenschaftslisten dargestellt werden, wobei die Eigenschaften als Schlüssel benutzt werden. Beim Wechsel der dafür verwendeten Eigenschaft wird auch der Zugriff auf die Daten entsprechend geändert. Die Prozedur SCHLUESSEL bekommt dann die folgende Form:

```
to SCHLUESSEL :l
op gprop :l :Begriff
end
```

Unter dem Namen Begriff muß dann die Schlüsseleigenschaft (Name, Wohnort usw.) entweder als globaler Name verfügbar sein, oder der Schlüsselbegriff wird hier direkt mit

```
op gprop :l "Name
```

angegeben.

Der Aufbau solcher Datenstrukturen mit Eigenschaftslisten wird in einer eigenen Lektion behandelt, die eine komplette Dateiverwaltung beinhaltet.

Sortieren

Über Sortiermethoden gibt es umfangreiche Literatur. Welches Verfahren als optimal anzusehen ist, hängt sowohl von der Größe der zu sortierenden Datenmenge als auch von deren vorliegender Anordnung ab. Hier soll nur eine einfache Methode eingesetzt werden, die als lineares Einfügen bezeichnet wird. Sehr große Datenmengen werden in LOGO-Programmen kaum vorkommen, weil die LOGO-Systeme dafür gegenwärtig noch zu langsam arbeiten. Das lineare Einfügen ist dann meist den komplizierteren Methoden überlegen, die darüberhinaus natürlich auch größeren Programmieraufwand erfordern.

Die folgende Prozedur LINSORT erwartet eine Liste, genauer einen Satz als Eingabe und gibt diese in aufsteigender Reihenfolge sortiert zurück.

```
to LINSORT :Kartei
op LIN (list first :Kartei) bf :Kartei
end

to LIN :v :h
if emptyp :h [op :v]
(pr :v "-::h);Ausgabe zum Testen
op LIN EIN :v first :h bf :h
end
```

```
to EIN :l :e
if emptyp :l [op (list :e)]
if first :l < first :e !
[op fput first :l EIN bf :l :e] [op fput :e :l]
end
```

Um das Verfahren zu verstehen, Probieren Sie es am besten erst einmal aus:

```
LINSORT [42 53 10 40 92 16 4 65]
```

In der dritten Zeile von LIN ist zu Testzwecken eine Zeile eingefügt, die den jeweiligen Zustand der zu sortierenden Liste ausgibt. Es sollte dann folgendes auf dem Bildschirm erscheinen:

```
LINSORT [42 53 10 40 92 16 4 65]
42 - 53 10 40 92 16 4 65
42 53 - 10 40 92 16 4 65
10 42 53 - 40 92 16 4 65
10 40 42 53 - 92 16 4 65
10 40 42 53 92 - 16 4 65
10 16 40 42 53 92 - 4 65
4 10 16 40 42 53 92 - 65
[4 10 16 40 42 53 65 92]
```

Die Prozedur LIN hat die zwei Einaben v und h, die für den vorderen bzw. hinteren Teil der Liste stehen. Der vordere Teil v ist für sich betrachtet bereits sortiert und wird schrittweise länger.

Beim linearen Einfügen wird jeweils das erste Element aus dem hinteren, unsortierten Teil entfernt und an die passende Stelle im bereits sortierten vorderen Teil eingefügt. Dieses Verfahren läßt sich direkt als Rekursion formulieren.

Das Einfügen des abgetrennten Elements geschieht in der Prozedur EIN. Auch dieser Schritt kann leicht rekursiv beschrieben werden: Das einzufügende Element wird mit dem ersten Element der Liste verglichen. Ist es kleiner, so kann es direkt an den Anfang gesetzt werden. Im anderen Fall werden von der Liste

solange die am Anfang stehenden Elemente abgetrennt, bis tatsächlich ein größeres gefunden oder die Liste erschöpft ist. Das einzufügende Element wird also stets an den Anfang der Restliste gesetzt, die möglicherweise leer sein kann. Anschließend wird die Liste in gewohnter Weise wieder zusammengesetzt.

Mit LINSORT können ebenso Wörter in der lexikalischen Ordnung sortiert werden:

```
LINSORT [Goedel Escher Bach]
```

Ergebnis: `[Bach Escher Goedel]`

Die Prozedur EIN ist bereits so formuliert, daß die zu sortierende Liste auch wieder richtig zusammengebaut wird, wenn die Elemente selbst Listen sind. Das ist natürlich nur dann sinnvoll, wenn ein Sortierschlüssel verwendet wird. Die dritte Zeile von EIN muß dann beginnen mit

```
if SCHLUESSEL first :l < SCHLUESSEL first :e
```

Als Beispiel soll eine Kartei mit Namen vorbereitet werden

```
make "Kartei [[Albert Einstein] [Otto Hahn] [Friedrich Hund] [Nils
Bohr]]
```

Wenn in SCHLUESSEL mit op last :l das letzte Element ausgegeben wird, dann kann nach dem Nachnamen, mit op first :l dagegen nach dem Vornamen sortiert werden.

```
op last :l in SCHLUESSEL:
[[Nils Bohr] [Albert Einstein] [Otto Hahn] [Friedrich Hund]]
op first :l in SCHLUESSEL:
[[Albert Einstein] [Friedrich Hund] [Nils Bohr] [Otto Hahn]]
```

Vielleicht wollen Sie das Geburtsjahr der großen Physiker einbeziehen, dann kann auch danach sortiert werden.

```
[[Otto Hahn 1879] [Albert Einstein 1879]!
[Nils Bohr 1885] [Friedrich Hund 1894]]
```

Übungen

1) Schreiben Sie ein Programm unter Verwendung von SIEB, das sogenannte Primzahlzwillinge aufsucht. Primzahlzwillinge sind benachbarte Primzahlen, deren Differenz nicht größer als Zwei ist.

2) Bei einem Test, z.B. einer Führerscheinprüfung, müssen eine Anzahl von Aufgaben gelöst werden. Die Bearbeitung jeder Aufgabe wird durch Punkte bewertet. Nutzen Sie das Tabellenprogramm aus Lektion 16, um einen solchen Test für 10 Teilnehmer auszuwerten.

3) Ein Ausdruck der Form A*(B+C) kann als Baumstruktur dargestellt werden. Die zugehörige Liste ist:

   ```
   [* [+ B C] A]
   ```

   Stellen Sie entsprechend den Ausdruck (A+B)*C+D*(A-E) dar. Versuchen Sie, das Programm Tiereraten für den Aufbau solcher Baumstrukturen zu nutzen.

Lektion 17: Programme als Listen

Neue Sprachelemente:
text, define, run, Listenstruktur von Programmen

Programme:
LOGOPLAN (rechnendes Tabellenprogramm mit Hilfsprogrammen), MASKENGENERATOR (Programmgenerator)

17.1 Programmtext

LOGO-Programme werden mit Hilfe von LOGO-Vokabeln formuliert, die entweder aus dem Grundwortschatz stammen oder vom Anwender selbst definiert werden. Die in Programmen vorkommenden Daten sind ebenso aus Wörtern aufgebaut; desgleichen sind Zahlen für LOGO Wörter. Programme und Daten haben also die gleichen Bestandteile.

Programme haben eine Struktur, wobei sich der Programmtext in Programmzeilen gliedert, die Sätzen in einem Text entsprechen. Es versteht sich fast von selbst, daß LOGO die Struktur von Programmen durch Listen darstellt.

Der Programmtext kann mit po ausgegeben werden und erscheint dann so, wie Sie ihn geschrieben haben, wenn auch möglicherweise noch Leerzeichen durch das LOGO-System hinzugefügt wurden. Probieren Sie folgende Zeile:

```
po "QUADRAT text "QUADRAT
```

Statt QUADRAT können Sie natürlich auch irgendeine andere Prozedur nehmen.

```
to QUADRAT :Seite
repeat 4 [fd :Seite rt 90]
end

[[Seite] [repeat 4 [fd :Seite rt 90]]]
```

Die LOGO-Prozedur text hat den Programmtext in der Form einer Liste ausgegeben, deren Elemente die Programmzeilen sind, die selbst wieder als Listen vorliegen. Der Prozedurname im Kopf der Prozedur kommt nicht vor, da er der Name ist, der dem Programmtext gegeben wurde.

Das erste Listenelement ist die Liste der Eingaben. Wenn eine Prozedur keine Eingaben hat, setzt LOGO die leere Liste an diese Stelle.

Wird bei text ein Prozedurname eingegeben, der nicht existiert, wird als Programmtext die leere Liste zurückgegeben.

```
text "Nixda Ergebnis: []
```

Der Programmtext kann mit einem Namen wie jede andere Liste versehen werden.

```
make "PROGRAMM text "QUADRAT
pr :PROGRAMM
```

Als Werte von PROGRAMM wird jetzt der Programmtext der Prozedur QUADRAT ausgegeben.

Die mit Hilfe von text erzeugte Liste kann wie jede andere bearbeitet werden.

```
first text "QUADRAT
```

Es ergibt sich die Liste der Eingaben der Prozedur QUADRAT

```
Antwort: [Seite]
```

Mit text können Prozeduren zunächst nur untersucht werden, der Programmtext kann aber als Liste auch manipuliert werden.

17.2 Programme schreiben Programme

text erzeugt aus einer vorhandenen Prozedur eine Liste mit dem zeilenweise gegliederten Programmtext. Umgekehrt kann aus einer Liste, die einen Programmtext enthält, eine Prozedur gemacht werden. Hierzu besitzt LOGO die Vokabel define.

```
make "PROGRAMM [[Seite] [repeat 4 [fd :Seite rt 90]]]

define "QUADRAT :PROGRAMM po "QUADRAT

to QUADRAT :Seite
repeat 4 [fd :Seite rt 90]
end
```

Aus dem Programmtext wurde mit Hilfe von define tatsächlich die entsprechende Prozedur QUADRAT erklärt. LOGO gibt dabei übrigens keine Meldung "QUADRAT defined" aus, wie das beim Schreiben von Programmen sonst erfolgt. Wenn im Programmablauf selbst Prozeduren definiert werden, soll die Bildschirmausgabe nicht durch Systemmeldungen gestört werden.

- define erzeugt eine LOGO-Prozedur. Es werden zwei Eingaben erwartet, wobei die erste Eingabe der Name der zu erklärenden Prozedur ist.

- Die zweite Eingabe muß eine zeilenweise gegliederte Liste sein. Jedes Element dieser Liste muß selbst wieder eine Liste sein, nämlich je eine Programmzeile. Das erste Element besteht aus der Liste der Prozedureingaben. Falls keine Eingaben vorgesehen sind, muß an dieser Stelle eine leere Liste stehen.

define funktioniert ähnlich wie das Schreiben von Programmen im Editiermodus, wie es durch to oder ed erreicht wird.

Ausführen von Listen

Jede Liste kann nun umgekehrt als Programmtext gedeutet werden. So wie man eine Programmzeile schreiben und sofort ausführen kann, kann auch eine Liste als Programmtext unmittelbar mit Hilfe der LOGO-Vokabel run zur Ausführung gebracht weren.

```
repeat 5 [pr 1 + random 6]
```

Wird diese Zeile als Eingabezeile geschrieben, wird sie unmittelbar nach Drücken der Return-Taste ausgeführt. Bei einer Definition als Liste kann sie über die LOGO-Anweisung run wie eine Eingabezeile ausgeführt werden.

```
make "Zeile [repeat 5 [pr 1 + random 6]]
```

Ausführung: `run :Zeile`

- run nimmt eine Liste als Eingabe, deren Inhalt als Liste wie eine Eingabezeile im Direktbetrieb ausgeführt wird.

run entfernt die äußeren eckigen Klammern der Liste und macht daraus eine Eingabezeile, die anschließend ausgeführt wird. Die Eingabe für run kann natürlich auch als Ergebnis einer LOGO-Aktivität zustande kommen.

```
make "Aktion [pr 1 + random 6]
run ( list "repeat 4 :Aktion )
```

Damit ist es auch möglich, Prozedurnamen als Eingaben bei anderen Prozeduren zu übergeben. Mit run können diese Prozeduren dann benutzt werden.

```
to AUSFUEHREN :Prozedur
(pr [Ausgefuehrt wird die Prozedur] :Prozedur )
run :Prozedur
end
```

```
to WUERFELN
pr 1 +random 6
end
```

Aufruf: `AUSFUEHREN [WUERFELN]`

Wird in der Eingabeliste zu run eine Ergebnis erzeugt, muß auch die Ausgabe berücksichtigt werden. run [...] ist also genauso einzusetzen wie der Aufruf einer entsprechenden Prozedur.

```
to LOTTOZAHL
op 1 + random 49
end
```

Aufruf: `pr run [LOTTOZAHL]`

17.3 LOGOPLAN: Rechnendes Tabellenprogramm

Das Programmpaket zur Tabellenverwaltung aus der vorangegangenen Lektion soll professioneller werden. In Tabellenkalkulationsprogrammen kann, wie der Name schon sagt, gerechnet werden.

Für das Projekt LOGOPLAN soll die Möglichkeit geschaffen werden, innerhalb von Spalten bzw. Zeilen Summen zu bilden. Es soll neben der Eingabe von Tabellentext bzw. Tabellenwerten die Eingabe dreier Kommandos möglich sein:

- Z 2 5: Es wird die Summe der Tabellenwerte in der gerade angesprochenen Spalte, und zwar beginnend mit dem Element in der 2. Zeile bis zum Element in der 5.Zeile, berechnet und anstelle eines Eingabewertes eingetragen.
- S 3 6: Es wird entsprechend die Summe in der aktuellen Zeile für den angegebenen Spaltenbereich gebildet.
- ! (Ausrufezeichen): Es soll die bei früheren Eingaben definierte Summenbildung wieder verwendet werden.

In erster Linie geht es um die ersten beiden Erweiterungen, die dritte bedeutet nur eine zusätzliche Verwaltung der Formeln.

Die Auswertung einer Tabellenformel geschieht in der Prozedur TAB.FORMEL, die von der Prozedur WERT aufgerufen wird.

```
to TAB.FORMEL :S :Formel :I :Imax
;Auswertung einer Tabellenformel
if :I > :Imax [op :S]
op TAB.FORMEL run :Formel :Formel :I+1 :Imax
end

to WERT :E;Entschluesselung eines Kommandos
if first :E ="Z [op TAB.FORMEL ∅ !
[:S+item :I item :SN thing :TAB] item 2 :E last :E]
if first :E ="S [op TAB.FORMEL ∅ !
[:S+item :L item :I thing :TAB] item 2 :E last :E]
end
```

TAB.FORMEL ist allgemein gehalten. Das Ergebnis wird unter dem Namen S rekursiv erzeugt. Die rekursive Wiederholung läuft für I vom eingegebenen Wert bis zum Eingabewert Imax in Einerschritten. Für jeden Wert von I wird die eingegebene Formel ausgewertet.

Als Beispiel für TAB.FORMEL können Sie folgenden Aufruf ausprobieren:

```
TAB.FORMEL ∅ [:I+:S] 1 5
```

Ergebnis: 15

Wenn Sie den Ablauf im Detail verfolgen wollen, sollten Sie den Trace-Modus einschalten. Das Ergebnis ist die Summe der natürlichen Zahlen von eins bis fünf; die letzten beiden Eingaben definieren die Summationsgrenzen. Mit

```
TAB.FORMEL 1 [:I*:S] 1 5
```

Ergebnis: 120

wird gerade das Produkt der natürlichen Zahlen von eins bis fünf berechnet, was auch mit 5! bezeichnet wird. Wie an den Beispielen zu erkennen ist, bedeutet die erste Eingabe von TAB.FORMEL den Startwert für die rekursive Berechnung.

In WERT ist je ein Aufruf von TAB.FORMEL für den Fall vorgesehen, daß ein Kommando mit Z bzw. S beginnt. Der Unterschied liegt dabei nur in der Vertauschung von Zeilen mit Spalten. Die für den rekursiven Aufruf geeignete Formel ist bei Z:

```
:S+item :I item :SN thing :TAB
```

Das bedeutet, zum Wert von S wird das Tabellenelement der Zeile I und der Spalte SN addiert. Wird das Resultat wieder in TAB.FORMEL als Eingabe bei der Rekursion verwendet, so werden damit Elemente einer Tabellenspalte aufsummiert.

Wie erfolgt nun die Auswertung in TAB.FORMEL?

Die Formel, genauer gesagt die rekursive Formulierung der Formel, wird als Eingabewert übergeben. Darin kommt der Name I vor, der lokal bezüglich TAB.FORMEL ist, sowie die Namen SN bzw. L (aktuelle Spalten- bzw. Zeilennummer) verwendet, die hier bezüglich TAB.FORMEL global aus der rufenden Prozedur SPEINGABE kommen.

Die Auswertung geschieht nun mit Hilfe der LOGO-Vokabel run.

Die Prozedur WERT zur Formelauswertung wird im Zusammenhang mit der Eingabe von Tabellenwerten eingesetzt. Im bereits bekannten Programmpaket aus Abschnitt 16.2 geschieht das in der Prozedur SPEINGABE, die dementsprechend geändert werden muß.

```
to SPEINGABE :L :tp
if :L > :lmax [stop]
(local "pp "e "so "alt)
make "pp se :tp :L+1
reverse setcursor :pp
make "alt item :L item :SN thing :TAB type :alt
einzeile 20
setcursor [0 20] make "e rl
if emptyp :e [go "FORT] if :e= (se char 94) [DRUCK throw
"HOME]
if :e = [!] [make "e item :l item :SN :TABFORM]
if count :e > 1 [make "TABFORM TABEIN :TABFORM :L :SN :e
make "e WERT :e] [make "e first :e]
if not :e = char 92 [make "alt :e make :TAB TABEIN thing
:TAB :L :SN :e]
label "FORT
DRUCK
if :e = char 92 [stop] [SPEINGABE :L+1 :tp]
end
```

Die Eingabe muß jetzt mit rl erfolgen. Wird nur das Ausrufezeichen eingegeben, wird die Eingabe durch den entsprechenden Wert der Tabelle TABFORM ersetzt. Wenn die Eingabeliste dann mehr als ein Element enthält, handelt es sich um eine Formel, die in die Tabelle TABFORM eingesetzt wird. Anschließend wird die Formel mit WERT ausgewertet. Hat die Eingabeliste dagegen nur ein Element, so wird dieses mit first als Wort bereitgestellt.

Diese erweiterte Version arbeitet neben der eigentlichen Tabelle mit einer zweiten Tabelle TABFORM, die genauso erzeugt wird wie Tab, also gleichviele Elemente enthält. In dieser Tabelle werden nur eventuell eingegebene Kommandos eingetragen, um sie später verwenden zu können. TABFORM wird hier im

Gegensatz zur eigentlichen Tabelle als globaler Name angesehen. Das ist zwar inkonsequent, erspart uns aber die Auflistung der notwendigen Änderungen bei den rufenden Programmen.

Beispiel:

```
make "TITEL "Haushalt
make "TPOS [0 10 16 22 28 34]

TABELLE "Haus 4 6
TABELLE "TABFORM 4 6
EINGABE "Haus :TPOS :TITEL
```

Die Tabelle kann dann etwa folgendermaßen besetzt werden.

	1981	1982	1983	1984	SUMME
ESSEN	3200	3450	3600	3820	14070
KLEIDUNG	800	850	900	.	.
SUMME	4000	4300	4500	.	.

Die Summen in der letzten Zeile werden durch die Eingabe von Z 2 3 die in der letzten Spalte mit S 2 5 erzeugt.

Bei der nächsten Eingabe können Werte verändert werden. Damit in den Positionen für die Summen die Formeln neu ausgewertet werden, muß jeweils ! eingegeben werden. Es ist also nicht notwendig, die Kommandos bei jeder Ausführung neu zu definieren, was ja gerade der Sinn der Einführung der zusätzlichen Tabelle TABFORM ist.

17.4 Programmgeneratoren

Mit define können LOGO-Programme erzeugt werden, die anschließend für eine spätere Verwendung abgespeichert werden. Wer diese Programme dann benutzt, wird möglicherweise niemals erfahren, wie sie zustande gekommen sind. Es ist also letzten Endes unerheblich, ob Programme zuerst am Schreibtisch auf dem Papier entstehen oder mit Hilfe eigens dazu erdachter Programme entwickelt werden.

Nun kann man sich natürlich fragen, ob Programme denn bei der Erstellung anderer Programme eine wirkliche Hilfe darstellen. Schließlich muß die zündende Idee zum Programm irgendwo ihren Ursprung haben. Ohne damit alle Möglichkeiten von Programmmgeneratoren ins Blickfeld bringen zu können, sollen hier an einem einfachen Beispiel die betreffenden Fähigkeiten von LOGO betrachtet werden.

Neben define ist in diesem Zusammenhang auch die LOGO-Vokabel run wichtig. Man kann eine Folge von LOGO-Kommandos sowohl mit run direkt ausführen als auch mit define anschließend in ein Programm umwandeln. Eine Methode zur Erzeugung von Programmen sieht dann folgendermaßen aus:

- Es werden eine Reihe von Aktivitäten unter der Führung eines geeigneten Programms direkt am Rechner durchgeführt und zwar unter Zuhilfenahme von Eingabemedien wie Taststatur und Maus.

- Dabei können einzelne Schritte korrigiert bzw. sogar verworfen werden.

- Aus der Folge der akzeptierten Schritte wird ein separates Programm erzeugt.

Das entstande Programm führt dann alle Schritte automatisch durch, die bei der Erarbeitung am Rechner manuell hervorgerufen wurden. Mit dieser Methode können die erwünschten Programmaktivitäten direkt erarbeitet werden. Weil dieser Vorgang unter der möglichst komfortablen Führung eines geeigneten

Programms erfolgt, können sogar Programme von Anwendern erstellt werden, die eigentlich gar nicht programmieren können.

Voraussetzung für die beschriebene Methode ist die Möglichkeit, Programmschritte einerseits ausprobieren, Manipulationen daran durchführen, als auch andererseits die erfolgreichen Schritte so protokollieren zu können, daß daraus Programme entstehen. Alle genannten Anforderungen werden von LOGO erfüllt.

Maskengenerator zur Datenerfassung

Als Beispiel soll ein Programm dienen, mit dem Programme für Eingabemasken erzeugt werden. Wir wollen hier an die Programme zur Eingabe von Daten mit Bildschirmmasken aus Lektion 10 anknüpfen.

Eingabemasken sind Formulare, die auf dem Bildschirm erzeugt werden. Der Benutzer wird dann von einem zum nächsten Eingabefeld des Formulars geführt. In Lektion 10 haben die Programme MASKE und EINGABE diese Aufgabe erledigt. Dabei ist aber das Formular im Programm MASKE fest definiert. Wollen Sie ein anderes Formular benutzen, müssen Sie sich Ihr Formular auf dem Papier entwerfen und dann dementsprechend angepaßte Programme MASKE und EINGABE formulieren.

Hier soll nun ein Programm entwickelt werden, das diese Prozeduren automatisch erzeugt. Bei der Anwendung soll der Benutzer das gewünschte Formular direkt auf den Bildschirm schreiben. Daraus sollen die beiden Prozeduren MASKE und EINGABE erzeugt werden. Ist diese Aufgabe erledigt, können die beiden vom Programm her definierten Prozeduren abgespeichert und für künftige Zwecke verwendet werden.

An diesem Beispiel kann auch demonstriert werden, wie ein Anwender, der gar nicht programmieren kann, mit Hilfe eines speziellen Programms doch Programme erzeugen kann. Mit solchen Programmgeneratoren wird die Benutzung von Com-

putern um einen weiteren Schritt näher an den Anwender heran gebracht.

Die Analyse des Programmpakets ergibt:

```
Ø: MASKENGENERATOR
1: ...MEINGABE
2: ......STEUER
2: ......MPOS
3: .........WORT
4: ............CLINKS
2: ......MEINGABE*
1: ...MLISTE
2: ......MLISTE*
1: ...ELISTE
2: ......ELISTE*
```

MEINGABE dient der Eingabe der Maske durch den Anwender. MLISTE erzeugt den Programmtext für die zu erzeugende Prozedur MASKE als Liste. ELISTE erzeugt dann den Programmtext für die Prozedur EINGABE.

```
to MASKENGENERATOR
make "Pliste []
make "Posten []
ct
MEINGABE; Maskeneingabe
define "MASKE MLISTE :Posten [[] [ct ]]
define "EINGABE ELISTE :Posten :Pliste !
[[] [MASKE make "LA []]]
end

to MEINGABE
STEUER first cursor last cursor
MPOS if :AC = 252 [stop];clr·Taste
MEINGABE
end
```

```
to STEUER :SP :Z
type char 7;akustisches Signal
setcursor [0 24]
type [Naechstes Feld - Eingabe mit] char 94 [BEGINNEN]
label "Anfang setcursor se :SP :Z
make "C rc make "AC ascii :C
if :AC = 240 [if :Z > 0 [make "Z :Z -1]]
if :AC = 242 [make "Z :Z + 1]
if :AC = 250 [if :SP < 39 [make "SP :SP +1]]
if :AC = 254 [if :SP < 0 [make "SP :SP -1]]
if :AC = 243 [make "Z :Z +1 make "SP 0]
if :AC = 94 [stop]
go "Anfang
end

to MPOS
make "CP cursor type char 7
setcursor [0 24]
(type [Feld eingeben] char 94 [weiter, clr: Abbruch])
setcursor CP
make "Posten lput se :CP WORT :Posten
make "Pliste lput cursor :Pliste
end

to WORT
make "W "
label "Zeichen make "C rc make "AC ascii :C
if :AC = 248 [make "W bl :W CLINKS]
if or :AC = 94 :AC = 252 [op :W]
if and :AC > 31 :AC < 125 [type :C make "W word :W :C]
go "Zeichen
end

to CLINKS;Cursor nach links verschieben
type char 8;CPC-Steuerzeichen
end
```

Die Prozedur MEINGABE erzeugt nach Eingabe durch den Anwender zunächst zwei Listen, Posten und Pliste.

Die Elemente von Posten sind Listen aus drei Wörtern, sie stellen jeweils ein Eingabefeld dar. Die ersten beiden Elemente geben die Position des Feldes auf dem Bildschirm an, das dritte Element den zugehörigen Formulartext. Diese Liste enthält die für die Definition der Prozedur MASKE benötigten Eingaben.

In der Liste Pliste wird festgehalten, bei welcher Position der Formulartext des Feldes endet und das eigentliche Eingabefeld beginnt. Pliste besteht demnach aus zweielementigen Listen. Diese Angabe wird für die Prozedur EINGABE benötigt.

Beispiel:

```
[Ø 2 Wohnort:] [8 2]
```

Die erste Liste ist Teil von Posten, die zweite steht an entsprechender Stelle in Pliste. Damit soll am Anfang der dritten Bildschirmzeile (die erste Zeile hat die Nummer 0) der Text Wohnort: erzeugt werden, der Cursor soll bei der Eingabe dann auf die achte Spalte hinter den Text gebracht werden. Da der Beginn des nächsten Eingabefeldes bekannt ist, kann der gesamte Formulartext auch gegen Überschreiben durch den Benutzer geschützt werden.

Die Eingabeprozedur MEINGABE ruft zunächst die Prozedur STEUER auf. STEUER macht es möglich, während der Eingabe der Maske den Cursor zu steuern. Der Benutzer muß also zuerst den Cursor an die gewünschte Stelle auf dem Bildschirm bringen und dann mit der Eingabe des Formulartextes beginnen. In der Prozedur MPOS wird dann der aktuelle Posten, d.h. der Text des angesteuerten Feldes eingegeben.

Sowohl in STEUER als auch in MPOS wird eine erläuternde Zeile am unteren Bildschirmrand ausgegeben. Beginn und Ende des Textes im Formular wird durch ein spezielles Zeichen, hier wurde der Pfeil nach links gewählt, vom Benutzer angezeigt.

In STEUER gibt es eine Reihe von if-Zeilen vom Typ

```
if :AC = 242 [....]
```

Unter dem Namen AC ist der ASCII-Code des eingegebenen Zeichens verfügbar. Es bedeuten:

240:	Cursor nach oben
242:	Cursor nach unten
250:	Cursor nach rechts
254:	Cursor nach links
243:	Returntaste
94:	Pfeil nach oben
248:	DEL-Taste (verwendet in WORT)
252:	CLR-Taste (verwendet in WORT)

Man muß hier den ASCII-Code benutzen, weil die Steuertasten nicht als Zeichen abgefragt werden können.

Wenn der Cursor im Programm gesteuert werden soll, gibt es einmal die Möglichkeit, mit setcursor und cursor die gewünschte Position auf dem Bildschirm anzusteuern:

```
setcursor se (first cursor) - 1 last cursor
```

versetzt beispielsweise den Cursor um eine Spalte nach links. (Beachten Sie, daß hier die runden Klammern notwendig sind. Beim CPC-Computer kann der Cursor auch durch die Ausgabe der ASCII-Codes 8 bis 11 gesteuert werden, wovon in der Prozedur CLINKS Gebrauch gemacht wird. Diese gerätespezifische Lösung ist natürlich nicht ganz im Sinne der Philosophie von LOGO. CLINKS funktioniert übrigens auch beim Joyce-Computer, bei dem es zusätzlich eine ganze Reihe von sogenannten Escape-Sequenzen zur Bildschirmsteuerung gibt (siehe Anhang 3 im Handbuch).

```
to MLISTE :Posten :ml
if emptyp :Posten [op :ml]
make "l first :Posten
make "l (list "setcursor piece 1 2 :l !
"type ( list item 3 :l ))
op MLISTE bf :Posten lput :l :ml
end

to ELISTE :Posten :Pliste :ml
if emptyp :Posten [op lput [op :LA] :ml]
(local "l "ms "li) make "l first :Posten make "li first :Pliste
make "Posten bf :Posten
if emptyp :Posten !
[make "ms 40] [make "ms first first :Posten]
if :ms >0 [make "ms :ms -1] [make "ms 39]
make "l ( se "make ""la "lput "EIN :li :ms "" ":la)
op ELISTE :Posten bf :Pliste lput :l :ml
end
```

MLISTE und ELISTE bilden aus den beiden Listen Posten und Pliste, die bei der Eingabe der Bildschirmmaske erzeugt worden sind, diejenigen Listen, die den Programmtext für die als Endergebnis gewünschten Prozeduren MASKE und EINGABE darstellen. Wenn im Programmtext LOGO-Vokabeln eingebaut werden sollen, dann müssen sie in diesem Zusammenhang mit Anführungszeichen eingeleitet werden.

So soll beispielsweise in der vierten Zeile von MLISTE das LOGO-Wort setcursor in die entsprechende Zeile des Programmtextes von MASKE eingefügt werden. Würde hier das Anführungszeichen nicht stehen, würde die LOGO-Prozedur setcursor an dieser Stelle ausgeführt, was gar nicht beabsichtigt ist. Wenn in dem zu erzeugenden Programmtext selbst ein Anführungszeichen vorkommen soll, muß, z.B. in der siebten Zeile von ELISTE, das Anführungszeichen selbst mit einem Anführunszeichen eingeleitet werden.

In ELISTE wird auch die Prozedur EIN im Programmtext von EINGABE vorgesehen. Damit ist die in Lektion 10 behandelte Prozedur für die Eingabe der Daten im eigentlichen Eingabefeld

gemeint, die hier noch einmal zusammen mit der dort benutzten Prozedur ADRESSEN aufgeführt werden soll.

```
to ADRESSEN :liste
local "Person
label "ein make "Person EINGABE
if first :Person = "* [op :liste]
make "liste lput :Person :liste
go "ein
end

to EIN :S :Z :M :W
setcursor se :S :Z type char 32
setcursor se :S :Z
(local "c "ac)
label "taste make "c rc make "ac ascii :c
if :ac = 248 [op EIN :S-1 :Z :M bl :W]
if :ac = 243 [op :W]
if and :ac > 31 :ac < 123!
[make "W word :W :c type :c] [go "taste]
if :S = :M [op :W]
op EIN :S+1 :Z :M :W
end
```

ADRESSEN und EIN gehören nicht zu dem hier vorgestellten Programmpaket. Mit ADRESSEN können vielmehr die erzeugten Prozeduren MASKE und EINGABE zur Eingabe von Personaldaten verwendet werden.

Das Programmpaket wird mit

```
MASKENGENERATOR
```

in Gang gesetzt. Der Bildschirm ist danach gelöscht, der Cursor blinkt in der oberen linken Ecke. Danach wird die Eingabemaske entsprechend den Anweisungen der letzten Bildschirmzeile eingegeben. Ist der letzte Formulartext eingegeben, wird die Eingabe mit der CLR-Taste abgebrochen. LOGO arbeitet dann noch an der Erstellung der Prozeduren MASKE und EINGABE weiter.

Für die spätere Verwendung können die beiden erzeugten Prozeduren auf Diskette gespeichert werden. Zuvor sollte Überflüssiges gelöscht werden. Dazu gehören einmal alle Namen, die mit

```
ern glist ".APV
```

gelöscht werden. Ebenso können alle Prozeduren des Programmpakets mit er entfernt werden.

Das Löschen kann auch im Programm MASKENGENERATOR selbst programmiert werden, das für diesen Zweck um folgende Zeilen ergänzt werden muß:

```
...
ern glist ".APV
er [STEUER MPOS ELISTE MLISTE CLINKS]
er [MEINGABE WORT MASKENGENERATOR]
end
```

Mit dieser Änderung sind nach dem Aufruf nur noch die erzeugten Prozeduren MASKE und EINGABE vorhanden. Diese Version sollte natürlich erst dann in Gang gesetzt werden, wenn das Programmpaket einmal auf Diskette abgespeichert worden ist!

Nach der Eingabe einer einfachen Bildschirmmaske für die Erfassung von Name, Vorname und Wohnort erzeugt das Programm als Beispiel folgende zwei Prozeduren.

```
to MASKE
ct
setcursor [Ø Ø] type [Name:]
setcursor [22 Ø] type [Vorname:]
setcursor [Ø 3] type [Wohnort:]
end
```

```
to EINGABE
MASKE make "la []
make "la lput EIN 5 Ø 21 "'' :la
make "la lput EIN 3Ø Ø 39 "'' :la
make "la lput EIN 8 3 39 "'' :la
op :la
end
```

Lektion 18: Dateiverwaltung mit LOGO

Programme:
VERWALTUNG (mit Hilfsprogrammen)

18.1 Dateiverwaltung

Fast jeder muß heute Dateien verwalten, was z.B. beim eigenen Telefonverzeichnis anfängt. Das Telefonregister besteht aus einer Reihe von Eintragungen, wovon jede zumindest den Namen und die Telefonnumer des Fernsprechteilnehmers enthält. Aber auch weitere Merkmale wie die Adresse, Unterscheidungen in privat bzw. geschäftlich usw. können sinnvoll sein.

Wer viele Bücher, Schallplatten oder auch Computerprogramme besitzt, wird sich eine Kartei dazu anlegen. Neben dem allgemeinen Bedürfnis nach einer schriftlichen Dokumentation dient eine solche Kartei dem Wunsch, auf einzelne Objekte gezielt zugreifen zu können. In einer Bibliothek will man beispielsweise alle Titel aus einem Themenkreis heraussuchen und alphabetisch geordnet ausgeben.

Es soll aber meist nicht nur auf vorhandene Informationen zugegriffen werden. Ebenso wichtig ist es, den Inhalt der Kartei in komfortabler Weise immer wieder ergänzen und verändern zu können.

Obgleich die herkömmliche Bürotechnik zwar schon immer Hilfmittel dafür entwickelt hat, ist der Computer für solche Zwecke besonders geeignet. Nun wird man sicher nicht neben jedes Telefon einen eigenen Personalcomputer stellen, wohl aber bald auf den Schreibtisch jedes Sachbearbeiters. Es ist nicht verwunderlich, daß zur Dateiverwaltung mit Computern schon viele, leistungsfähige Programme auf dem Markt vorhanden sind.

In diesem Kapitel soll ein Paket von LOGO-Programmen für die Dateiverwaltung betrachtet werden, das natürlich nicht mit professionellen Softwareangeboten konkurrieren kann. Vielmehr sollen elementare Aktivitäten, die in diesem Zusammenhang auftreten, analysiert und zur vertieften Anwendung von LOGO in Prozeduren umgesetzt werden.

Zunächst muß eine für Karteien geeignete Datenstruktur ausgewählt werden. Bei DR LOGO bieten sich sofort die Eigenschaftslisten an. Eine Karteikarte wird als Objekt mit Eigenschaften dargestellt. Jedes Merkmal auf der Karteikarte ist eine Eigenschaft, der jeweilige Eintrag ist der zugehörige Eigenschaftswert. Die zu einer Kartei gehörenden Karteikarten sollen einmal einen gemeinsamen Namen haben, daneben sollen sie durchnummeriert werden. In LOGO ist es möglich, Namen jeweils aus verschiedenen Bestandteilen aufzubauen. Deswegen werden die Objekte mit einer Bezeichnung wie Buch1, Buch2 usw. versehen.

18.2 Das Verwaltungsmenü

Der Dateiname kennung wird als globaler Namen erwartet, der dann auch beim Abspeichern mit auf Diskette angelegt wird. Der Aufruf kann etwa mit

```
make "kennung "Buch VERWALTUNG
```

erfolgen. Anschließend wird folgendes Menü auf dem Bildschirm erzeugt.

Neu anlegen	1
Neu durchnummerieren	2
Weitere Datensaetze	3
Datensaetze bearbeiten	4
Sortieren	5
Speichern	6
Heraussuchen	7

Nach Auswahl einer der angebotenen Tätigkeiten mit Hilfe der Kennziffern, wird jeweils eine dafür vorgesehene Prozedur aufgerufen.

```
to VERWALTUNG
ct
(pr [Dateiname:] :kennung) setcursor [Ø 4]
type [Neu anlegen] setcursor [25 4] pr "1 pr "
type [Neu durchnummerieren] setcursor [25 6] pr "2 pr "
type [Weitere Datensaetze] setcursor [25 8] pr "3 pr "
type [Datensaetze bearbeiten] setcursor [25 1Ø] pr "4 pr "
type [Sortieren] setcursor [25 12] pr "5 pr "
type [Speichern] setcursor [25 14] pr "6 pr "
type [Heraussuchen] setcursor [25 16] pr "7 pr "
catch "error [run (list word "a rc)]
type [weiter mit beliebiger Taste!] pr rc VERWALTUNG
end
```

Nach der Ausgabe des Menüs wird mit word "a rc der Name der zum jeweiligen Menüpunkt gehörenden Prozedur, also A1, A2, ... A7, aufgebaut und das ausgewählte Programm dann mit der LOGO-Vokabel run in Gang gesetzt. Der Prozeduraufruf geschieht unter einem catch "error, was den Sprung auf die vorletzte Zeile des Programms VERWALTUNG bei einem auftretenden Fehler im Ablauf zur Folge hat.

Sind beispielsweise gewünschte Daten nicht vorhanden, was relativ leicht durch eine falsche Eingabe bei der Bedienung der Programme auftreten kann, erscheint die Meldung "weiter mit beliebiger Taste!" auf dem Bildschirm.

Das dargestellte Menü zeigt die Fähigkeiten des Programmpakets an. Diese sollen nun im einzelnen getrennt betrachtet werden.

18.3 Anlegen einer Datei

Neu anlegen

Beim ersten Menüpunkt werden die gewünschten Eigenschaften, d.h. Merkmale auf einer Karteikarte, für eine neue Kartei, beschrieben und die vorliegenden Daten können erfaßt werden. Das angewählte Programm heißt A1, dieses ruft dann die Programme DATEN, ERFASSEN, SATZ und PROTOTYP.

```
to A1
ct (pr :kennung [neu anlegen])
DATEN :kennung
end

to DATEN :kennung
(pr [Datenerfassung mit Kennung:] :kennung)
pr [Bitte gewuenschte Merkmale aufzaehlen]
catch "schluss [ERFASSEN 1 PROTOTYP rl word :kennung "1]
end

to ERFASSEN :i :proto
local "name make "name word :kennung :i
ct (pr [Datensatz:] :name
SATZ :name :proto
make "imax :i;hoechste Datensatznummer
ERFASSEN :i+1 :proto
end

to SATZ :name :eigen
(local "e "w)
if emptyp :eigen [stop]
make "e last :eigen make "eigen bl :eigen
(type last :eigen ": :e) make "w rq
if numberp :w [make "w (+ :w)]
if :w = "eof [throw "schluss]
if emptyp :w [make "w :e]
pprop :name last :eigen :w
SATZ :name bl :eigen
end
```

```
to PROTOTYP :eig :name
if emptyp :eig [op plist :name]
pprop :name first :eig "
op PROTOTYP bf :eig :name
end
```

Das Programm A1 ruft unmittelbar die Prozedur DATEN; der Umweg über A1 wird nur gegangen, um auf die erklärende Bezeichung DATEN überzugehen. In DATEN wird der Benutzer aufgefordert, die gewünschten Eigenschaften aufzuzählen, was aber nicht ausschließt, daß im Laufe einer späteren Überarbeitung noch zusätzlich Eigenschaften hinzugefügt werden können. Mit einem catch "schluss erfolgt danach der Aufruf der Prozedur ERFASSEN. Das catch "schluss hat den Vorteil, daß nachfolgend aufgerufene Prozeduren vorzeitig abgebrochen werden können.

Im Programm ERFASSEN wird die Prozedur SATZ rekursiv gerufen, wobei jeweils die Kennummer erhöht wird. Die Datenerfassung wird beendet, indem für die erste Eigenschaft eines Objekts "eof" (für end of file) eingegeben wird. Die zugehörige Abfrage erfolgt in der Prozedur SATZ, mit throw "schluss wird dann direkt nach DATEN zurückgesprungen.

Die erste Eingabe für SATZ ist der Name des Datensatzes, etwa Buch1 oder Platte5, die zweite ist eine Liste von Eigenschaften und deren zugehörigen Werten. Beim Neuanlegen einer Kartei sind zwar keine Eintragungen vorhanden, weswegen die benötigte Liste zunächst mit Hilfe des Programms PROTOTYP beim Aufruf von ERFASSEN erzeugt werden muß. Das Verfahren erlaubt es, die Prozeduren ERFASSEN und SATZ auch bei anderen Menüpunkten zu benutzen, die der Pflege der Kartei dienen.

Im Programm SATZ wird nun die Liste der Merkmale rekursiv abgearbeitet. Falls bereits eine Eintragung vorhanden ist, erscheint diese auf dem Bildschirm. Ohne Benutzereingabe wird dann nach Drücken der Returntaste die dargestellte Eintragung übernommen. Auch dies ist im Hinblick auf die spätere Pflege

der Kartei so vorgesehen, wobei das entstehende Bildschirmlayout auch noch nicht optimal ist.

In SATZ wird mit numberp überprüft, ob eine Zahl eingegeben worden ist. In diesem Fall wird der Wert mit make "w (+ :w) neu definiert, was sicher zunächst seltsam erscheint, weil sich damit am Zahlenwert nichts ändert. Die Anweisung ist aus der Beobachtung heraus entstanden, daß ohne zusätzliche Maßnahme beim Vergleich von Eigenschaftswerten mit > bzw. < die lexikalische Ordnung verwendet wird, was etwa dazu führt, daß 1000 kleiner als 2 angenommen wird. Der benutzte Kunstgriff setzt offensichtlich in der internen Darstellung von DR LOGO einen Hinweis auf die numerische Bedeutung der Eigenschaft.

Das Programm PROTOTYP schließlich weist dem mit name übergebenen Objekt die vom Benutzer aufgezählten Merkmale als Eigenschaften zu, wobei als Wert das leere Wort genommen wird. Das Objekt hat danach formal die gewünschten Eigenschaften, die eigentliche Eingabe ist dann eine Veränderung der Eigenschaftswerte.

18.4 Aufräumen einer Datei

Neu durchnumerieren

Werden aus der Kartei Datensätze entfernt, z.B. Buch3 aus der Kartei Buch, so entsteht eine Lücke in der Numerierung. Um diese Lücke zu schließen, müssen alle nachfolgenden Datensätze um eine Position aufrücken, wofür es bei den Eigenschaftslisten keine direkte Möglichkeit gibt. Im Bild der Karteikarten ausgedrückt müssen die nachfolgenden Datensätze vollständig auf die jeweils vorangegangene Karteikarte übertragen werden, was recht aufwendig ist.

Deshalb werden hier die entstehenden Lücken zunächst einmal hingenommen. Der zweite Menüpunkt gestattet es, auf Wunsch die gesamte Kartei auf einmal neu durchzunumerieren, so daß dann alle entstandenen Lücken verschwinden.

```
to A2
ct (pr :kennung [durchnumerieren])
pr [Obere Grenze:]
make "imax NUMMER 1
end

to NUMMER :i
if :i > :imax [op :imax]
if not emptyp plist word :kennung :i [op NUMMER :i+1]
local "ne make "ne NACHF :i+1
if emptyp :ne [op :i-1]
VERSCHIEBE word :kennung :i :ne
op NUMMER :i+1
end

to NACHF :i
if :i > :imax [op "]
local "j make "j word :kennung :i
if emptyp plist :j [op NACHF :i+1] [op :j]
end

to VERSCHIEBE :h :k
local "sl make "sl plist :k
if emptyp :sl [stop]
pprop :h last bl :sl last :sl
remprop :k last bl :sl
VERSCHIEBE :h :k
end
```

Die vorhandenen Datensätze können Nummern zwischen eins und :imax besitzen. Die Buchhaltung über die größte vorkommende Nummer wird in der globalen Variablen imax geführt, welche auch beim Abspeichern mit auf Diskette abgelegt wird.

Ob ein Datensatz vorhanden ist, wird mit Hilfe von plist festgestellt; plist liefert nämlich die leere Liste als Ergebnis, wenn das Objekt keine Eigenschaften besitzt. Die Prozedur NACHF sucht den nachfolgenden Datensatz heraus, der tatsächlich Eintragungen enthält, während VERSCHIEBE schließlich die

Eigenschaften auf den freien Satz mit der kleinsten Nummer überträgt.

18.5 Erweitern und Pflegen der Datei

Weitere Datensätze

Die Aufgabenstellung ist bei diesem Menüpunkt klar: Die Datei (Kartei) soll um zusätzliche Datensätze erweitert werden.

```
to A3
ct (pr [Weitere Datensaetze fuer:] :kennung
(pr [beginnend mit der Nummer:] :imax+1)
catch "schluss [ERFASSEN :imax+1 plist word :kennung "1]
end
```

Die Ergänzung mit neuen Datensätzen geschieht analog zum ersten Menüpunkt mit der Prozedur ERFASSEN und SATZ. Die gewünschten Eigenschaften werden dem ersten Datenssatz entnommen, der als Prototyp benutzt wird.

Datensätze bearbeiten

Zur Pflege einer Datei sind hier folgende Möglichkeiten vorgesehen:

- einzelne Datensätze können gelöscht,
- vorhandene Eigenschaftswerte können verändert,
- zusätzliche Eigenschaften können hinzugefügt werden.

```
to A4
BEARBEITEN 1
end

to BEARBEITEN :i
local "name
if :i > :imax [stop]
make "name word :kennung :i
ct (pr [Datensatz:] :name)
SATZAUS plist :name setcursor [0 15]
pr [Naechster Datensatz] setcursor [23 15] pr "1
pr [Datensatz loeschen] setcursor [23 16] pr "2
pr [Eigenschaften aendern] setcursor [23 17] pr "3
pr [Eigenschaften anfuegen] setcursor [23 18] pr "4
run item rc [[] [LOESCHE :name] [SATZ :name plist :name] [ANFUEGEN
:name]]
BEARBEITEN :i+1
end

to SATZAUS :eliste
(local "e "w)
if emptyp :eliste [stop]
make "e last :eliste make "eliste bl :eliste
 (pr last :eliste ": :e)
SATZAUS bl :eliste
end

to LOESCHE :name
local "e make "e plist :name if emptyp :e [stop]
remprop :name first :e
LOESCHE :name
end
```

```
to ANFUEGEN :name
(local "e "w)
type [Gewuenschte Eigenschaft:]
make "e rq if emptyp :e [stop]
type [Wert:] make "w rq
if numberp :w [make "w (+:w)]
pprop :name :e :w
ANFUEGEN :name
end
```

Zu Beginn der Prozedur BEARBEITEN wird der Datensatz (Nummer i) ausgegeben. Dazu dient die Prozedur SATZAUS, die wie SATZ arbeitet, selbst aber keine Eingaben erwartet. Anschließend wird ein Untermenü, d.h. eine weitere Auswahl angeboten. Die Menütechnik ist zunächst wie im Hauptprogramm VERWALTUNG angelegt. Der Anwender gibt die gewünschte Kennziffer ein.

Die Ansteuerung der verschiedenen Möglichkeiten geschieht hier anders als im Hauptprogramm. Es werden keine Prozedurnamen aufgebaut, sondern die ausgelöste Aktivität wird mit item rc aus einer Liste von vier Elementen herausgelöst und anschließend mit run ausgeführt. Soll zum nächsten Datensatz übergegangen werden, braucht überhaupt nichts weiter zu geschehen, weswegen das erste Element in der Liste von Aktivitäten die leere Liste ist.

Zum Löschen eines Datensatzes wird die Prozedur LOESCHE eingesetzt, die sämtliche Eigenschaften des eingegebenen Objekts entfernt. Um Eigenschaften zu verändern, wird das bereits schon aus den Punkten eins und drei des Hauptmenüs bekannte Programm SATZ aufgerufen.

Im letzten Punkt des Untermenüs wird die Prozedur ANFUEGEN angesteuert, mit der zusätzliche Eigenschaften für den gerade aktuellen Datensatz angefügt werden können. Anfuegen ist ebenfalls nach dem Vorbild von SATZ aufgebaut, nur wird sie erst dann beendet, wenn die Frage nach der

gewünschten Eigenschaft ohne Eingabe mit der Returntaste abgeschlossen wird.

18.6 Sortierte Ausgabe

Zum Sortieren wurde in Lektion 16 das Programm LINSORT, d.h. Sortieren durch lineares Einfügen, entwickelt, das nun an dieser Stelle für die Dateiverwaltung eingesetzt werden kann. In der Kartei Buch sollen etwa alle Bücher bezogen auf die Autoren in alphabetischer Ordnung oder auch in der Reihenfolge des Erscheinungsjahrs, vielleicht auch nach dem Preis geordnet ausgegeben werden.

```
to A5
ct (pr [Sortierte Ausgabe fuer: ] :kennung)
pr [Sortierbegriff: ]
local "begriff make "begriff rq
ct (pr [Ausgabe nach] :begriff [sortiert]
SORTA SORTIERE :begriff
end

to SORTA :d
if emptyp :d [stop]
SATZAUS plist first :d
SORTA bf :d
end

to SORTIERE :begriff
local "kartei make "kartei glist :begriff
op LINSORT :kartei
end

to LINSORT :kartei
op LIN (list first :kartei) bf :kartei
end
to LIN :v :h
if emptyp :h [op :v]
op LIN EIN :v first :h bf :h
end
```

```
to EIN :l :e
if emptyp :l [op (list :e)]
if gprop first :l :begriff < gprop :e :begriff !
[op fput first :l EIN bf :l :e] [op fput :e :l]
end
```

Zunächst wird der Sortierbegriff erfragt, der natürlich eine in der Kartei vorhandene Eigenschaft sein muß. Der Begriff wird dann dem Programm SORTIERE übergeben, das die sortierte Liste von Datensätzen als Ergebnis liefert.

In der Prozedur SORTA werden die Datensätze dann mit der bereits bekannten Routine SATZAUS auf dem Bildschirm ausgegeben.

Das Programm SORTIERE trifft zunächst Vorbereitungen zum Sortieren, wobei dem Namen kartei mit Hilfe der LOGO-Vokabel glist die Liste aller Objekte zugewiesen wird, welchen die als Sortierbegriff eingegebene Eigenschaft zukommt. Das bereits bekannte Sortierprogramm LINSORT greift über diese Liste auf die Datensätze zu.

Die drei Prozeduren LINSORT, LIN, EIN sind hier nocheinmal wiedergegeben. Beachten Sie, daß der Zugriff über den Schlüssel direkt in der dritten Zeile von EIN mit gprop first :l :begriff eingebaut worden ist, ohne daß der Umweg über ein Programm SCHLUESSEL gegangen wird.

Das Sortieren äußert sich zum Schluß darin, daß eine Liste der Form [Buch7 Buch1 Buch3 ...] als Ergebnis entsteht. Die Eigenschaften der Objekte werden dann in der Reihenfolge Buch7, Buch1, Buch3, ... zunächst mit plist zusammengestellt und anschließend mit der Prozedur SATZAUS ausgegeben.

18.7 Suchen und Abspeichern

```
to A6
pr [Prozeduren vor dem Speichern loeschen? j/n]
if lc rc ="j [er glist ".DEF]
catch "error [save :kennung]
if emptyp error [stop]
(pr [File] :kennung [existiert bereits])
pr [Alten File loeschen? j/n]
if lc rc ="j [erasefile :kennung save :kennung] !
[pr [File nicht abgespeichert!]]
end
```

Es ist naheliegend, die unter dem globalen Namen kennung vorhandene Bezeichung ebenfalls als Filenamen beim Abspeichern zu verwenden. Beim Kommando save wird in LOGO grundsätzlich der gesamte Arbeitsspeicher auf Diskette übertragen. (Es gibt allerdings auch LOGO-Versionen, die es gestatten, Teile in sogenannte Packages zusammennzufassen, die dann getrennt verwaltet werden können.) Insbesondere werden alle Programmteile mit auf den Datenfile geschrieben. Das hat den Vorzug, daß für die Dateiverwaltung auch nur ein einziger Ladevorgang erforderlich ist.

Als Alternative dazu kann man sämtliche Prozeduren vor dem Abspeichern löschen, wobei dann weniger Platz für eine Kartei auf der Diskette beansprucht wird. Die Prozedur A6 bietet diese Möglichkeit an. Gelöscht werden sämtliche Prozeduren mit der Anweisung er glist ".DEF, was sich übrigens auch auf die gerade in Gang befindliche Prozedur A6 bezieht. Bei dieser Variante müssen die Prozeduren dann wieder nachgeladen werden, wenn die Dateiverwaltung fortgesetzt werden soll. Bei dieser Methode kann man mit Hilfe des Standardfilenamens STARTUP auf der LOGO-Systemdiskette das automatische Laden der Programme veranlassen.

Falls beim Abspeichern der Filename bereits existiert, was ja bei einer echten Dateiverwaltung nach dem ersten Anlegen sogar die Regel ist, wird das Löschen des vorhandenen Files angeboten. Wenn das Kommando changef richtig funktioniert, dann würde

man an dieser Stelle den alten File nicht löschen, sondern umbenennen.

Heraussuchen

Der zum Schluß behandelte Menüpunkt ist bei Dateiverwaltungsprogrammen meist die am häufigsten benötigte Aktivität. Die effiziente Organisation eines gezielten Zugriffs auf Daten ist ein wichtiges und zugleich schwieriges Problem in der EDV. Hier wird nur ein einfaches Durchmustern der Datensätze mit einem Suchkriterium betrachtet.

```
to A7
(local "such "vergleich)
ct pr [Suchbegriff: ] make "such rq
pr " pr [Gewuenschter Vergleich:]
make "vergleich rl
SUCHEN 1
end

to SUCHEN :i
if :i > :imax [stop]
(local "name "wert)
make "name word :kennung :i
make "wert gprop :name :such
if emptyp :wert [go "weiter]
if run se ":wert :vergleich [SATZAUS plist :name]
label "weiter suchen :i+1
end
```

Als Suchbegriff ist eine vorhandene Eigenschaft einzugeben. In der Prozedur SUCHEN werden alle Datensätze daraufhin durchgesehen, ob der Wert, der dieser Eigenschaft zukommt, eine vom Anwender verlangte Bedingung erfüllt.

In der Kartei Buch z.B. könnte als Suchbegriff das Erscheinungsjahr genommen und als Bedingung "> 1980" verlangt werden. In der drittletzten Zeile von SUCHEN wird mit

```
run se ":wert :vergleich
```

die Bedingung überprüft. Im genannten Beispiel würde etwa unter :wert die Zahl 1986 verfügbar sein. Mit se ":wert :vergleich wird dann die Liste

```
[:wert > 1980]
```

gebildet, die mit run ausgeführt und einen Wahrheitswert, also TRUE oder FALSE, ergibt. Wenn die Bedingung erfüllt ist, wird der Datensatz ausgegeben.

Die erläuterte Methode ist schon recht flexibel, um auf die gewünschten Datensätze zugreifen zu können. Sollen etwa alle Titel eines Autors gesucht werden, so gibt man "Autor" als Suchbegriff und anschließend beispielsweise "= "Sauer" als Vergleich ein. (Beachten Sie, daß die Anführungszeichen vor dem Namen notwendig sind.)

Wenn Sie kompliziertere Suchkriterien verwenden wollen, werden Sie am besten die drittletzte Zeile von SUCHEN ersetzen durch

```
if run :vergleich [SATZAUS plist :name]
```

Der Wert der betrachteten Sucheigenschaft muß in dieser Variante explizit in das Vergleichskriterium einbezogen werden. Beim Beispiel des Erscheinungsjahrs müßte etwa eingegeben werden:

```
:wert > 1980 statt > 1980
```

Dafür können dann auch logische Verknüpfungen verwendet werden. Beim Suchen nach Autorennamen beispielsweise:

```
or :wert = "Sauer :wert = "Einstein
```

Damit werden alle Titel beider Autoren herausgesucht. Beim Erscheinungsjahr könnte ein Zeitraum mit Hilfe von

```
and :wert > 1920 :wert < 1950
```

für die Suche benutzt werden.

Lektion 19: Farbe und Töne

Neue Sprachelemente:
setbg, setpc, setpal, pal, sound, env, ent, release

Programme:
KETTE, FARBWECHSEL, NUM, PERIODE

In der letzten Lektion soll auf die Programmierung der Farben wie auch die der Tongeneratoren eingegangen werden. Natürlich sind die Farben nur für die Besitzer von Schneider-CPC-Computern mit einem Farbmonitor interessant. Tongeneratoren gibt es bei allen CPC-Modellen, jedoch nicht beim Schneider Joyce.

19.1 Die Grundfarben Rot, Grün und Blau

Das farbige Bild wird bei Farbmonitoren wie bei Farbfernsehern durch drei Bilder in den Grundfarben Rot, Grün und Blau erzeugt (RGB), wobei der tatsächliche Farbeindruck als Überlagerung der drei Bilder entsteht, was die Physiker auch als additive Farbmischung bezeichnen.

Für jede Grundfarbe gibt es eigene "Farbkanonen", die getrennt angesteuert werden. Durch Veränderung der Intensität, d.h. der Helligkeit der drei Grundfarben an jedem Bildpunkt, werden beliebige Farbschattierungen erzeugt. Bei einem Computerbild kann die Intensität aber nur über bestimmte, getrennte Stufen hinweg variieren; die Intensität wird digital gesteuert.

Beim Schneider-CPC gibt es für jede der drei Grundfarben nur die drei Helligkeitsstufen

0, 1 oder 2

und dementsprechend können

3*3*3 = 27

verschiedene Farbtöne erzeugt werden. Deswegen finden Sie rechts auf dem Gehäuse über dem eingebauten Laufwerk eine Tabelle mit den Farbcodes 0 bis 26. Diese Farbnummern werden allerdings in LOGO nicht in dieser Form verwendet.

Tatsächlich kann man nicht gleichzeitig über alle 27 mögliche Farben verfügen; in LOGO gibt es sozusagen jeweils vier Farbtöpfe auf einmal. In jedem Farbtopf kann aber nach Belieben eine der 27 Farbschattierungen vorhanden sein, was vielleicht zunächst etwas verwirrend klingt.

Praktisch heißt das: Sie haben drei verschiedene Farbstifte zur Auswahl und eine weitere Wahl für das Zeichenblatt, d.h. den Hintergrund. Die tatsächlich auf dem Monitor sichtbare Farbe ist bei jedem Stift eine der 27 möglichen rot-grün-blau Kombinationen. Diese Zuordnung kann laufend, auch programmgesteuert, verändert werden, wobei aber, und das ist entscheidend, alle Punkte, die mit demselben, gerade angesprochenen Farbstift gezeichnet worden sind, gleichzeitig die Bildschirmfarbe wechseln.

19.2 Festlegung der Farben

Auswahl der Farbstifte

Auf dem Textfenster haben Sie hier überhaupt keine Wahl. Der Hintergrund gehört stets zum Farbtopf mit der Nummer Null, die Schrift zu dem mit Nummer Eins.

Beim Grafikfenster wird die Farbe des Hintergrundes mit der LOGO-Vokabel setbg, die des Zeichenstift - gleichzeitig auch die der Turtle - mit dem Kommando setpc gesetzt.

```
setbg 1 setpc 0
```

Eine Änderung der Hintergrundnummer macht sich erst bemerkbar, wenn das Grafikfenster mit cs oder clean gelöscht wird.

Mit den Funktion sf (screenfacts) und tf (turtlefacts) können die gerade aktuellen Werte der Farbtöpfe erfragt werden.

```
sf tf
```

Antwort: z.B. `[1 SS 5 WINDOW 1] [Ø Ø Ø PD Ø TRUE]`

In der Antwortliste von sf gibt das erste Element die Hintergrundfarbe an, in der von tf das vorletzte ELement die Farbe des Zeichenstifts.

Zulässige Werte sind jeweils 0, 1, 2, 3 bei Geräten mit Farbmonitor; bei monochromem Bildschirm gibt es nur die Farbkennzeichnungen 0 bzw. 1.

Auswahl der Bildschirmfarbe

Die zu jedem Farbtopf gehörende Farbe wird durch die Angabe der Intensitäten für die drei Grundfarben festgelegt.

```
setbg Ø setpal Ø [2 Ø Ø]
```

Hier wird im Farbtopf nur die Farbe rot mit der Intensität 2 ausgewählt, die beiden anderen Grundfarben fehlen. Der Bildschirm sollte danach einen roten Hintergrund zeigen. In jedem Fall sollte damit der Hintergrund des Textfensters erröten.

setpal hat zwei Eingaben: die angesprochene Farbnummer und eine Liste aus drei Elementen, welche die zugehörige Farbzusammensetzung angibt. Probieren Sie aus:

```
repeat 1ØØ !
[setpal Ø (se random 3 random 3 random 3) wait 1Ø]
```

Damit wird die Hintergrundfarbe durch Würfeln ständig verändert. Betrachtet man dabei den Farbeindruck, den eine Grafik oder ein Text bei wechselndem Hintergrund erweckt, so fällt auf, daß sich ohne eine Änderung des Farbtopfs bzw. der

zugehörigen Farbschattierung ein Farbwechsel allein durch den veränderten Farbkontrast zum Hintergrund ergibt. Insbesondere bei dünnen Linien kann man hinsichtlich der Farbwirkung Überraschungen erleben. Bei der Farbgestaltung muß letztlich viel durch Experimentieren in die endgültige Form gebracht werden.

Mit setpal können Sie natürlich auch einfache Experimente zur additiven Farbmischung anstellen.

```
ts ct setpal 0 [2 0 0] wait 50 setpal 0 [0 2 0] wait 50
setpal 0 [2 2 0]
```

Zunächst sollten die beiden Grundfarben rot und grün und anschließend die Addition beider als gelb im Hintergrund erscheinen.

Die aktuelle Farbpalette, die zu einem Farbtopf gehört, wird mit der Funktion pal festgesetellt.

```
pal 0 Antwort: [2 2 0]
```

Damit kann auch ein Austausch von Farben in verschiedenen Farbtöpfen geschehen.

```
setpal 0 pal 1
```

Die Farbe aus Topf 1 wird in den mit Nummer 0 geschüttet, wodurch auf dem Textschirm der Text unsichtbar wird, weil Zeichen und Hintergrund jetzt die gleiche Farbe bekommen.

Bewegungseffekte

Durch einen Wechsel der Farbpalette können Teile einer Grafik sichtbar wie unsichtbar gemacht sowie in ihrem Farbeindruck wesentlich verändert werden. Bei einem raschen Wechsel kann damit der Eindruck einer Bewegung entstehen, ohne daß sich die eigentliche Grafik überhaupt zeitlich verändert.

Folgendes Beispiel soll einen gewissen Eindruck davon geben.

```
to Q :l;ausgefuelltes Quadrat
pd
repeat 5 [seth 0 fd :l rt 90 fd 2 lt 90 bk :l rt 90 fd 2]
pu fd :l
end

to KETTE :l
setpal 0 [2 2 2]
setpal 3 pal 0
setpal 2 pal 0
setpal 1 pal [2 0 0]
c pu setx -320 KE :l 1
end

to KE :l :i
if first tf >320 [stop]
setpc :i q :l
if i=3 [KE :l 1] [KE :l :i+1]
end

to FARBWECHSEL
setpal 1 [2 2 2]
setpal 2 [2 0 0]
wait 10
setpal 2 [2 2 2]
setpal 3 [2 0 0]
wait 10
setpal 3 [2 2 2]
setpal 1 [2 0 0]
wait 10
FARBWECHSEL
end
```

Aufruf: `fs cs KETTE 20 FARBWECHSEL`

Mit Q werden ausgefüllte Quadrate gezeichnet, die mit Kette waagerecht aneinandergereiht werden. Zunächst wird jedes dritte Kästchen mit roter Farbe versehen, dazwischen kommen zwei, deren Farbe die Hintergrundfarbe des Grafikschirm ist (hier als weiß gewählt).

In der Prozedur Farbwechsel werden nun jeweils die gefärbten Kästchen gelöscht und anschließend die rechts daneben liegenden rot eingefärbt. Durch den periodischen Wechsel scheint eine Kette aus roten Quadraten von links nach rechts über den Bildschirm zu laufen.

Weil die quadratischen Kästchen alle gleiche Gestalt haben, ist der Bewegungseffekt natürlich nicht sehr beeindruckend. Versuchen Sie beispielsweise, mehrere Phasen der Bewegung einer Figur mit teilweise verschiedenen Farbstiften zu zeichnen und anschließend mit der Technik des Programms FARBWECHSEL in periodischem Wechel darzustellen! Die ganze Technik würde natürlich noch eindrucksvollere Effekte erzielen können, wenn mehr verschiedene Farben gleichzeitig dargestellt werden könnten.

Als letzte Anregung zum Spiel mit Farben sei auf das Programm KIPPBILD aus Lektion 14 zurückgegriffen. Die dort verwendete Prozedur FELD kann wie folgt abgeändert werden, um Farbe ins Spiel zu bringen.

```
to FELD :y
if :y < -180 [stop]
pu setx 320 sety :y seth 0 pd make "i 2
repeat 8 [setpc 1 par 120 2 12 setpc :i para 60 12 2 !
make "i nach :i]
pu setx 320 sety :y - :ns - :l seth 0 pd
repeat 8 [setpc 1 para 60 12 2 make "i nach i setpc 1 !
par 120 2 12]
FELD :y - :ns -:l -:l
end
```

```
to nach :i
op item :i [2 3 1]
end
to FARBWECHSEL
make "k pal Ø
setpal Ø pal 1 setpal 1 pal 2
setpal 2 pal 3 setpal 3 :k wait 3Ø
FARBWECHSEL
end
```

Nach dem Aufruf von KIPPBILD (Lektion 14) kann dann mit FARBWECHSEL ein buntes Spiel in Gang gesetzt werden, wobei die Farben am Anfang noch nach Belieben festgesetzt werden können.

19.3 Programmierung der Tongeneratoren

Die Schneider-CPC-Computer besitzen einen leistungsfähigen, dreistimmigen Tongenerator. Auf dessen Programmierung wird hier aus mehreren Gründen nur kurz eingegangen. Um die Möglichkeiten der Tonerzeugung hinreichend deutlich zu erklären, wäre schon ein recht umfangreiches Kapitel erforderlich. Zu diesem Zweck sollten Sie sich den Abschnitt "Der Ton macht die Musik" in Kapitel 9 ihres Benutzerhandbuchs vornehmen.

Dort finden Sie Kommandos, die zunächst zur Programmiersprache BASIC gehören. In DR LOGO hat man nun diese Kommandos einfach einschließlich der Bezeichnung übernommen. Notwendig war lediglich die relativ geringfügige Anpassung an die Syntax der Sprache LOGO. Bei der Übertragung sind leider auch Fehler bzw. Ungereimtheiten entstanden.

Der Grundbefehl ist das Kommando sound:

```
sound [1 142 50]
```

Das erste Element der Eingabeliste gibt den gewünschten Tonkanal an, das zweite ist die sogenannte Tonperiode, die mit der Frequenz des Tons durch

Tonperiode: round 62500/Frequenz

zusammenhängt; beim betrachteten Beispiel handelt es sich um den Kammerton A. Das dritte Listenelement ist ein Maß für die Tondauer.

In BASIC hat derselbe Befehl die Form

SOUND 1,142,50,

wie er ausführlich im genannten Abschnitt von Kapitel 9 des Benutzerhandbuchs erläutert wird.

Eine chromatische Tonleiter kann wie folgt angesprochen werden:

```
to NUM :ton
if memberp :ton [c cis d dis e f fis g gis a b h] ! [op
item where [3822 3608 3405 3214 3034 2863 2703 !
2551 2408 2273 2145 2025] [op 0] end

to PERIODE :ton :oktave
op round (NUM :ton)/ item :oktave [1 2 4 8 16 32 64 128]
end
```

Die Funktion PERIODE liefert den Wert der im sound-Befehl verlangten Tonperiode, wobei die erste Eingabe die Tonbezeichung ist, die zweite eine Angabe über die Oktave. In der gegebenen Form von PERIODE wird der Kammerton A erzeugt mit

```
sound se 1 PERIODE "a 5
```

Die Oktavennummern laufen dabei von 1 bis 8, wobei man die Abzählung natürlich auch leicht verändern kann.

Der Musiker wird vielleicht die Notenbezeichnung anders wählen (eingestrichenes c usw.) und die absolute Tonhöhe auch im Symbol unterbringen. Bei der Übertragung von Notenschrift in eine vom Computer leicht benutzbare Form kann man beispielsweise auch so vorgehen, daß nur die Übergänge in andere Oktaven notiert werden. In diesem Zusammenhang bietet es sich an, mit Hilfe der Listenverarbeitung in LOGO komfortable Hilfsprozeduren zu entwickeln, die eine möglichst einfache und an der üblichen Notenschrift orientierte Eingabe erlauben. Dazu ist hier aber kein Raum mehr.

Bisher wurde nur mit einem Tonkanal gearbeitet, die beiden anderen werden mit den Nummern 2 bzw. 4 angesteuert. Tatsächlich ist der erste Eingabewert für sound eine Zahl zwischen null und 255, was für den Computer intern eine 8-Bit-Binärzahl ist. Die einzelnen Bits werden im sound-Befehl verschieden interpretiert. Die letzten drei Bits geben die angesteuerten Tongeneratoren an, wobei z.B. mit 7 alle drei gleichzeitig gesetzt werden.

Die Bedeutung der übrigen Bits, die die sogenannte Rendezvous- und Warteschlangentechnik steuern, entnehmen Sie am besten Ihrem Handbuch.

Neben der Dauer können wahlweise auch noch Angaben über die Lautstärke und die sogenannten Lautstärke- bzw. Tonhüllenkurven sowie zur Erzeugung von Geräuscheffekten als Parameter für sound angegeben werden.

Die Lautstärkehüllkurve bietet die Möglichkeit, die Lauststärke eines Tons stufenweise zu verändern, typischerweise also zuerst anschwellen und dann ausklingen zu lassen. Die Tonhüllenkurve erlaubt darüber hinaus, die Tonperiode stufenweise zu verändern, ähnlich wie auf Saiteninstrumenten ein Vibrato erzeugt werden kann. Auch hierüber lesen Sie am besten im Abschnitt "Der Ton macht die Musik" im Benutzerhandbuch nach.

Die Vorgabe von Hüllkurven geschieht mit den LOGO-Vokabeln env und ent, die in der Bezeichung mit den im Handbuch beschriebenen BASIC-Befehlen übereinstimmen, wobei die Eingabwerte aber nicht durch Kommata getrennt, sondern als Liste zusammengefaßt werden müssen. Hier wird nun leider beobachtet, daß DR LOGO negative Zahlen als Parameterwerte zurückweist, obwohl diese von der Funktionsweise her auch benötigt werden.

In der Lautstärkehüllkurve werden beispielsweise negative Zahlen benutzt, um ein Abschwellen zu erreichen. In diesem Fall kann man sich tatsächlich helfen, denn der Wert 15 hat dieselbe Wirkung, wie sie von (-1) erwartet wird, 14 dementsprechend wie von (-2) usw.

In der Tonhüllenkurve wird mit einer negativen Hüllkurvennummer eine periodische Wiederholung der Hüllkurve erreicht. Es ist dem Autor leider nicht gelungen, an dieser Stelle die Zurückweisung negativer Werte durch DR LOGO zu überlisten.

Als letzter Befehl ist noch release zu erwähnen, womit Tonkanäle freigegeben werden. Dieser Befehl funktioniert auch in LOGO wie im Handbuch als BASIC-Kommando beschrieben.

Endlich CP/M beherrschen! Von grundsätzlichen Erklärungen zur Speicherung von Zahlen, Schreibschutz oder ASCII über Anwendung von CP/M-Hilfsprogrammen bis zu „CP/M intern“ für Fortgeschrittene findet hier jeder CPC-Anwender schnell die notwendigen Hilfen und Informationen zur Arbeit mit CP/M. Dieses Buch berücksichtigt die Versionen CP/M 2.2 und CP/M PLUS (3.0) für Schneider CPC 464, CPC 664 und CPC 6128.

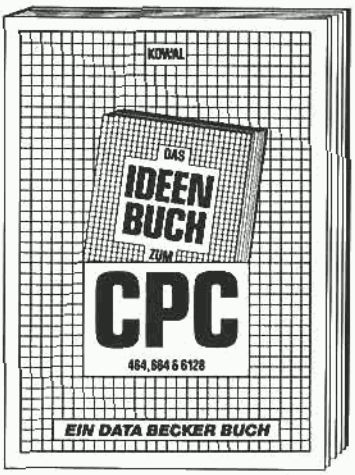
Aus dem Inhalt:

- Die Aufgabe von CP/M
- Die System-Diskette
- Regeln für Dateinamen
- Eingebaute Befehle USER, DIR, ERASE
- Transiente Befehle SET, PROTECT, SHOW, SUBMIT
- Alles über PIP
- Mehrere Dateien hintereinander drucken
- Fremde Diskettenformate lesen
- Unterschiede der CPC-Rechner und vieles mehr

Schieb, Weiler
Das CP/M-Trainingsbuch zum CPC
268 Seiten, DM 49,–
ISBN 3-89011-089-4

Bücher zum CPC

Sie haben sich Ihren CPC Schneider nicht nur zum Spielen gekauft? Dann ist dies das richtige Buch für Sie. Fast alles, was man mit den CPCs machen kann, ist in diesem Buch beschrieben. Es ist nicht nur interessant geschrieben, sondern enthält neben nützlichen Programmlistings vor allem viele, viele Anwendungsmöglichkeiten. Dabei wurde besonders Wert darauf gelegt, daß das Buch auch für den Einsteiger leicht verständlich ist.



Aus dem Inhalt:

- Schaufensterwerbung
- Auto und Computer
- Fahrtstreckenoptimierung
- Autokosten fest im Griff
- Geld, Kredit und Computer
- Zinseszinsberechnung
- Texten und Drucken
- Textverarbeitung
- Der Staat und das Geld
- Rentenberechnung
- Lohnsteuerjahresausgleich
- Haushalt und Gesundheit
- Elektronischer Kalorienzähler
- Malen, Zeichnen und die Erstellung von Grafiken
- Musik
- Fußballbundesliga
- Vokabeltrainer

Kowal
Das Ideen-Buch zum CPC
302 Seiten, DM 39,–
ISBN 3-89011-101-7

Bücher zum CPC

DFÜ für jedermann mit dem CPC bietet eine ausführliche und leichtverständliche Einführung in das Gebiet der Datenfernübertragung. Ausgelegt für die drei CPC-Rechner 464, 664 und 6128 bietet es sowohl dem Einsteiger als auch dem Profi neueste Informationen für die effektive Nutzung der vorhandenen Kommunikationsnetze.

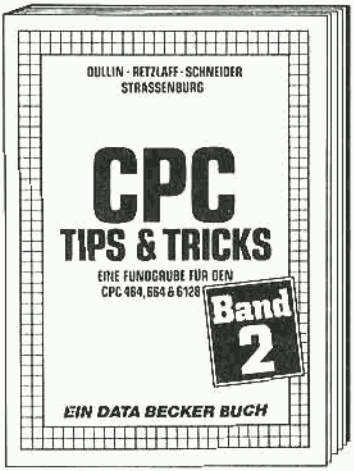
Aus dem Inhalt:

- Was ist DFÜ?
- Die Netze der Post
- Wichtige Postbestimmungen und Gebühren
- DATEX-P
- BTX
- Alles über Akustikkoppler und MODEMs
- Einrichtung und Benutzung von Mailboxen
- Der Zugriff auf Datenbanken
- Begriffserklärung: Originate, Answer, Half-Duplex usw.
- Serielle Schnittstelle selbstgebaut und vieles mehr

Severin, Schulwitz
DFÜ für jedermann zum CPC
306 Seiten, DM 49,–
ISBN 3-89011-140-8

Bücher zum CPC

Der 2. Band CPC Tips & Tricks ist für alle CPC-Besitzer interessant, die einen 464, 664 oder 6128 besitzen und Insider-Tips suchen zum BASIC, für Befehlserweiterungen und zur Maschinensprache. Viele effektive Hilfsroutinen!



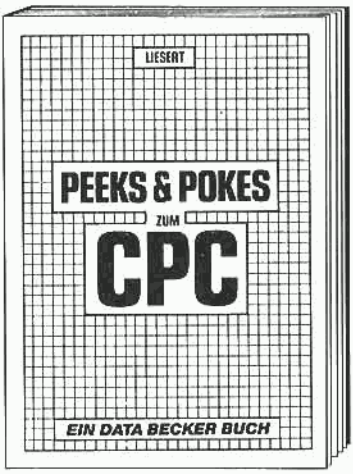
Aus dem Inhalt:

- Sortierverfahren
- 3-D-Grafik
- Menügenerator
- Eingabemaskengenerator
- Schützen eigener Programme
- Variablendump
- Grafik-Hardcopy
- BASIC-Zeile von BASIC aus erzeugen
- Wichtige Facts zur Programmierung in Maschinensprache
- Soft-Scrolling
- Schnittstelle von BASIC zu den Z80-Registern
- Insiderroutinen des Interpreters und des Betriebssystems
- Kompatibilität zwischen den 3 CPC-Rechnern
- Relokative Maschinenprogramme

Dullin, Retzlaff, Schneider, Straßenburg
CPC Tips & Tricks Band 2
256 Seiten, DM 39,–
ISBN 3-89011-131-9

Bücher zum CPC

PEEKs, POKEs und CALLs werden zum Anlaß genommen, eine wirklich leichtverständliche Einführung in Betriebssystem und Maschinensprache des CPC zu geben. Daß sich dabei viele interessante Programmier- und Anwendungsmöglichkeiten des Schneider-Computers ergeben, ist ein praktischer Nebeneffekt.



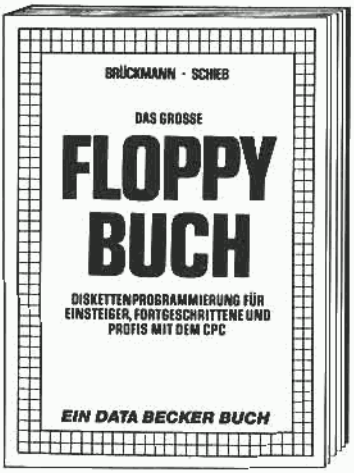
Aus dem Inhalt:

- Hardwareaufbau des CPC
- Betriebssystem und Interpreter
- PEEK & POKE – CALL
- Binärarithmetik
- Speicher schützen
- Bankswitching – ROM auslesen
- Video-RAM – Grafik – Scrolling
- BASIC-Interrupt
- Speicherung von BASIC-Zeilen
- Garbage Collection
- Aufbau und Funktionsweise des Z80
- Adressierungsmöglichkeiten
- Nützliche Maschinenroutinen

Liesert
Peeks & Pokes zum CPC
182 Seiten, DM 29,–
ISBN 3-89011-092-4

Bücher zum CPC

Das große FLOPPYBUCH zum CPC erklärt alles über Diskettenprogrammierung und Dateiverwaltung mit der Floppy DDI-1 des CPC 464, CPC 664 und CPC 6128. Sowohl für Anfänger der Diskettenprogrammierung als auch für ausgefuchste Assembler-Programmierer stellt dieses umfangreiche Standardwerk eine unentbehrliche Hilfe dar. Besonders interessant sind das ausführlich dokumentierte DOS-Listing und die vielen Beispielprogramme, darunter ein komplettes Dateiverwaltungspaket.



Aus dem Inhalt:

- Alles über sequentielle und relative Dateien
- Listing einer Dateiverwaltung
- Abfangen von Fehlermeldungen
- Relative Dateiverwaltung (!)
- Dokumentiertes DOS-Listing
- Programmierung des Controllers
- Disk-Monitor
- Diskettenmanager
- Sieben neue Diskettenbefehle
- Direktzugriff
- Grundlagen zu CP/M
- Mit zahlreichen Abbildungen und vieles mehr

Brückmann, Schieb
Das große Floppy-Buch
424 Seiten, DM 49,–
ISBN 3-89011-093-2