

Elementare Programmierkonzepte
zur Lösung algorithmischer Probleme
—→ ein kleiner Programmierkurs

H. P. Wolf*

WS 2003/04, Datei: `ideen.tex`, Formatierung: November 14, 2003

Übersicht

1	Gegenstand	2
2	Motivation	2
3	Rechner, Betriebssystem, Filesystem	3
4	Pascal-Einleitung	11
5	Basisstrukturen, Bausteine und fundamentale Regeln	13
6	Einfache Handlungsanweisungen	15
7	Einfache Datentypen	17
8	Variablen und Flusskontrolle: Abrechnungsprogramm	18
9	Richtig programmieren mit Unterprogrammen	22
10	Zugriffe auf gespeicherte Datenbestände – Dateien	26
11	Weitere Typfragen: array, record, set, ...	29
12	Praktische Hinweise	31
13	Literatur	33
14	Anhang: Zahleneinleseprogramm	34

*Statistik/Informatik, Fakultät für Wirtschaftswissenschaften, Universität Bielefeld
<http://www.wiwi.uni-bielefeld.de/~wolf/lehre/ws0304/progkurs/progkurs.html>

1 Gegenstand

Dieses Papier zeigt den roten Faden für den Programmierkurs, der ergänzend zur Einführung in die Informatik angeboten wird. Einige Diskussionspunkte sind nur skizziert und nicht in vollständigen Sätzen notiert. Dieses passt zur Angewohnheit in der Computerwelt, Dinge abzukürzen. Besonders sei auf die Abkürzungen *B:* und *F:* hingewiesen. Letzteres steht kurz für *Frage:*, während *B:* auf eine Beobachtung oder ein Beispiel aus der uns umgebenden Welt hinweisen soll.

Im Kurs wird zur Demonstration Turbo-Pascal verwendet. Trotzdem sollten die Programmierbeispiele im Prinzip mit wenigen Veränderungen auch mit anderen Produkten übersetzbar sein. Es sei erwähnt, dass Typen wie *Integer* unterschiedlich implementiert sein können, so dass sich bei zu großen Zahlen unterschiedliche Ergebnisse bzw. Effekte einstellen können. Auch ist zu beachten, dass die Steuerungs-Oberfläche von Turbo-Pascal zwar für den geübten Anwender die Arbeit erleichtert, gleichzeitig jedoch die Suche nach elementaren Prinzipien erschwert.

2 Motivation

F: Welche Motivation treibt einen Studierenden der Wirtschaftswissenschaften in einen Programmierkurs? Heute steht fast an jedem anspruchsvollen Arbeitsplatz ein Rechner und mit Hilfe dieses Gerätes ist ein großer Teil der Arbeit zu erledigen. Deshalb muß man sich einfach mal mit einem solchen Ding auseinandersetzen. Das klingt einleuchtend, aber warum muss denn gleich eine Einführung in das Programmieren her? Die Anwendungen sind doch dazu da, dass man es leicht hat und gerade nicht erst Programmieraufgaben zu lösen hat.

Fehler-
erkundung

→ Stimmt und stimmt auch nicht! Denn wenn alles klappt, dann muss man in der Tat genauso wenig über Interna nachdenken, wie immer vorgegeben wird. Doch sieht die (Computer-) Welt oft anders aus! Das Ding macht dann nicht das, was es tun soll. Das gedruckte Ergebnis sieht anders aus als vorher am Bildschirm suggeriert, das Anwendungs-Programm stellt sich stur und reagiert nicht auf die Mausclicks, es erscheint plötzlich eine unverständliche Warnung oder sogar Fehlermeldung. Was dann? In solchen Fällen hilft nur noch ein (teurer) Experte oder doch ein Eintauchen in die Welt oder besser das Meer der Rechner. Man muss dann eben unter die Oberfläche schauen.

F: Aber ist es dann nicht naheliegend, sich konkret mit Anwendungen zu beschäftigen, die – wie WORD, EXCEL, ... – im unternehmerischen Alltag wirklich eingesetzt werden? Noch dazu wird gemunkelt, dass angeblich sogar so seltsame Dinge wie MS-DOS angesprochen werden sollen?

Produkt-
unabhängigke

→ Eine gut formulierte Beschreibung auf einem Fertiggericht kann verhindern, dass das Gericht misslingt und man nicht verhungert. Doch wie kann man sich aus der Affäre ziehen, wenn man die Anweisungen auf einem neuen Fertiggericht nicht genau versteht? Will sagen, dass es für ein konkretes Problem natürlich die zeitlich effizienteste Lösung ist, wenn man genau zu dem Problem eine Antwort vorliegen oder erlernt hat. Jedoch ändern sich die Zeiten und damit die Dinge, die uns umgeben in rasender Geschwindigkeit. Und dann ist es schon mittelfristig eine bessere Strategie, einige ganz elementare Dinge verstanden zu haben. Manche dieser Elementarteilchen sind nicht nur zufälligerweise in sehr einfachen Lösungen aus vergangenen Computerzeiten anzutreffen und zwar in Reinkultur. Deshalb kann die Diskussion älterer Ansätze viel aktueller sein als jene der neusten Version von Produkt XYZ.

F: Vielleicht sind diese Antworten doch nicht so ganz überzeugend. Denn in großen Unternehmen gibt es ein Vielzahl von Leuten, deren Aufgabe darin besteht, die Rechner am Laufen zu halten und über rechnerspezifische Lösungen nachzudenken. Da liegt es doch auf der Hand, bei sich einstellenden Problemen genau diese qualifizierten Leute in Anspruch zu nehmen oder?

Konzept-
verständnis

→ Mag sein, dass diese Strategie klappt. Doch seien zwei Gegenfragen gestattet: Wer stellt eigentlich solche Experten ein und wer leitet sie gemäß Unternehmenszweck? Und wer fällt Entscheidungen, die etwas mit organisatorischen Abläufen und technologischer Infrastruktur zu tun haben? Es ist bei solchen Fragen schon sehr wichtig, elementare Konzepte und Prinzipien der Rechnerwelt verstanden zu haben. Denn da viele der elementaren Lösungsansätze in den unterschiedlichsten Gewändern wieder an die Oberfläche kommen, ist ihr Studium eine lohnenswerte Investition. Natürlich kann hier ein kurzer Programmierkurs nicht alle nötigen Weisheiten vermitteln, doch bevor man mit dem Laufenlernen beginnt, sollte man ein paar einfache Gehübungen machen. Und das ist auch der Grund, warum dieser Kurs, der sich besonders auch an Teilnehmer mit sehr geringen Vorkenntnissen richtet, angeboten wird.

F: Aber wieso soll ausgerechnet *Programmieren* den Anfang bilden? Ist das nicht nur etwas für Computerfreaks?

Erfahrungswissen

→ *Learning by doing!* Es fällt viel leichter, Schwierigkeiten, Lösungsansätze und Konzepte dadurch zu verstehen, dass man sich einige ganz einfache Probleme stellt und diese zu lösen versucht. Beim Fußball in der Bundesliga lassen sich strukturell verwandte Fehler und Erfolgsansätze wie in der Kreisklasse entdecken. Dieses gilt auch für die Rechnerwelt. Schwierigkeiten, die sich bei der Bearbeitung kleiner Probleme einstellen, sind auch bei großen Projekten anzutreffen und erklären Verhaltensweisen auf dem Markt erhältlichlicher Produkte. Zusammenfassend lassen sich für das Mittel *Programmieren* viele Pluspunkte anführen:

- die Teilnehmer lernen ein wenig zu programmieren und können hierdurch für eigene kleine Probleme Lösungen erstellen
- die Teilnehmer lernen fundamentale Lösungs-Konzepte kennen
- die Teilnehmer werden sich nach einem solchen Kurs viel schneller in eine andere Sprachwelt hineinfinden
- die Teilnehmer lernen Strategien kennen, Fehler zu suchen und zu vermeiden
- den Teilnehmern wird es später leichter fallen, verwendete Konzepte zu erkennen und Fehlersituationen zu beheben
- die Teilnehmer lernen allgemein Strategien zur Problemlösung kennen
- den Teilnehmern werden sogar Teile zukünftiger Prüfungen leichter fallen.

Vielleicht trägt diese Liste etwas dick auf, doch möge jeder die Überlegungen prüfen, hoffentlich mit dem Erfolg, dass das Vertrauen in den Kurs ansteigt und die Teilnehmer nun mit einer verstärkten Motivation zu Werke gehen.

3 Rechner, Betriebssystem, Filesystem

B: Damit jeder mitdenken kann, benötigen wir ein Problem, das sich jeder vorstellen kann, ein praktisches Problem: Stellen Sie sich vor, Sie haben einen wunderschönen Abend in einem Restaurant hinter sich und bitten die Bedienung, die Rechnung zu erstellen. Es ist oft ganz erstaunlich, wie schnell geübte Kellner die einzelnen Beträge addieren können. Hatten Sie nicht auch schon mal das Gefühl, dass eventuell das Ergebnis – der zu zahlende Betrag – falsch sein könnte? Da liegt der Wunsch nach einem Instrument auf der Hand, mit dem man leicht die Summe nachrechnen kann. Wir wünschen uns natürlich sofort einen Rechner mit einem Programm, mit dem wir den Job erledigen können – oder? Was spricht eigentlich für die Verwendung eines Rechners, was dagegen?

F: Welche Vor- und Nachteile hat ein Rechneinsatz?

- Bei der Verwendung einer Rechnerlösung haben wir uns mit verschiedenen Dingen herumzuschlagen: Geräte, Software, Oberflächen, Sprachen, Fehler, Breakdowns. Ein Gerät oder vornehm ausgedrückt die Hardware bildet die dingliche Voraussetzung für die computermäßige Unterstützung. Dass Geräte auch große Nachteile besitzen, musste der Autor kürzlich erfahren, als er bei Dunkelheit mit dem kleinen Zeh mit einer Personenwaage zusammenstieß, der kurz zuvor ihre neue Wirkungsstätte zugewiesen worden war. Man muss sich also über Geräte ärgern und über Programme, die heutzutage fast immer per graphischer Oberfläche bedient werden müssen. Dieser Umgang setzt zum Beispiel ein Verständnis für kleine Bildchen (Ikonen) voraus. Man muss die Bedeutungen der einzelnen Elemente bzw. der Vokabeln der Sprache zur Kommunikation kennen. Falls man abweichende Vorstellungen besitzt, wird man früher oder später seltsam anmutende Ergebnisse erhalten oder in Fehlersituationen geraten. Dann sitzt man fest, es hat sich ein Breakdown ereignet. Zur Fehlerbeseitigung müssen wir in der Regel unsere Vorstellungen korrigieren, denn unser Einfluss auf eine Anpassung von Hardware, Software, Oberflächenelemente usw. ist meist winzig – wir sind der vorgesetzten Konfiguration ausgeliefert.

Merke: Breakdowns offenbaren Differenzen zwischen der Realität und den Vorstellungen von der Realität, wie man sich am Beispiel des bargeldlosen Zahlungsverkehr überlegen kann.

B: Haben Sie schon mal ein fremdes Auto gefahren und dann festgestellt, dass es irgendwie nicht so wie gedacht funktioniert? Dabei habe ich stillschweigend unterstellt, dass Sie im Besitz einer Fahrerlaubnis sind. Oder haben Sie schon mal in einer fremden Küche neue Rezepte ausprobiert (nein? probieren Sie das mal!) und dann überlegt: *Da stimmt doch was nicht?* Man denke nur an Ausrufe, wie: *Was heißt eine Prise Salz?* Auch sei aus der Rechnerwelt an die karikierende Frage auf die Message: *press any key* erinnert: *Wo ist denn der Any-Key? Ich hab' so was nicht auf meiner Tastatur!*

F: Wie kann man den langfristigen Frust beim Umgang mit Rechnern klein halten?

- Es scheint langfristig nur eine Möglichkeit zu geben: Immer auf der Suche nach Grundkonzepten zu sein. Dazu ist es erforderlich, ein wenig die Kultur zu begreifen. Das gelingt zum Beispiel dadurch, dass man selbst Hand anlegt, also durch eigenes Tun. Aus diesem Grund werden bei uns für eine Fahrerlaubnis Fahrübungen bei einer Fahrschule sowie eine Führerscheinprüfung gefordert. Dabei soll der Klient theoretisches Wissen und praktische Erfahrungen anhäufen. In der Computerei müsste es auch so sein. Damit würden sich manche Breakdowns vermeiden lassen.

Merke: Fehlerbehebung erfordert oft konzeptionelles Verständnis und nicht nur Faktenwissen.

B: Eßkultur lernt man durch Blick in Küchen und Kochkurse kennen, übrigens erfordert das Maschinenbaustudium ein handwerkliches Praktikum.

F: Entsprechend können wir die Frage formulieren: *Wie sieht es eigentlich in der Küche der Programmentwicklung aus?* Wie kann man sich den Konzepten annähern?

- Gehe in einen Programmierkurs, lerne eine Sprache kennen und schreibe ein eigenes Kneipenabrechnungsüberprüfungsprogramm: KAUPP

B: Betrachten wir die Arbeit in einer Küche, es muss ja nicht gleich eine Großküche wie die Mensa sein. Kochen, wie geht das eigentlich? Überlege, wie Kochen funktioniert. Oder noch einfacher: kann man mir (der Autor ist Teetrinker und hat vom Kaffeekochen nur eine sehr diffuse Vorstellung) erklären, wie man Kaffee kocht? Geht Eierkochen genauso? Und was ist dabei zu beachten?

F: Für die Programmierung ergibt sich die große Frage: Welche Schritte beinhaltet der Weg von der Idee bis zum Programmeinsatz?

→ Bis zum fertigen Produkt ist es oft ein langer, nicht unbedingt gerader Weg, wie folgende Liste glauben machen könnte:

1. Ideen: zunächst muss für das gestellte Problem eine Idee, ein Ansatz her. Man muss schon wissen, in welcher Form man das Problem angehen will. Ist das geklärt, sollte man die Essenz seiner Gedanken unbedingt zu Papier bringen.
2. Algorithmus: hat man den Lösungsweg für sich verstanden, dann müssen die notwendigen Handlungsanweisungen in einer Form notiert werden, dass eine fremde Person die Chance hat, den Weg zu verstehen und zu prüfen.
3. Programm: wenn die einzelnen Schritte in ihrem Zweck geklärt sind, können die Anweisungen gemäß einer sprachlichen Verabredung formuliert werden, sprich: in eine Programmiersprache übersetzt werden.
4. Eintippen: bis einschließlich Punkt 3 konnte alles mit Bleistift und Papier erledigt werden, aber irgendwann kommt der Punkt, dass wir zum Beispiel unsere KAUPP-Lösung in den Rechner eingeben müssen. Wir brauchen also eine Stück Software, das uns gestattet, unser Programm einzugeben.
5. Speicherung: nach dem Eintippen, müssen wir unser Programm so abspeichern, dass es nicht verloren geht.
6. Übersetzen: da Programmiersprachen (wie wir sie hier im Auge haben) uns die Formulierung unserer Problemlösungsstrategie leicht machen sollen, müssen wir unser Programm in eine Darstellung oder Repräsentation übersetzen, die auf dem konkreten Rechner gestartet werden kann. Compiler heißen Programme, die solche Übersetzungsaufgaben erledigen.
7. Programmstart: falls die Übersetzung erfolgreich abgeschlossen ist, kann das übersetzte Programm getestet und gestartet werden.
8. Programmbedienung: für die Bedienung des neu geschriebenen Programms muss sein Anwender Regeln beachten, die in der Programmierphase durch den Programmierer festgeschrieben worden sind. Dieses gilt zum Beispiel für eine ordnungsgemäße
9. Programmbeendigung: beendet hoffentlich die Auseinandersetzung mit dem Programm nach erfolgreicher Arbeit.

Ein Anwender hat nur noch das Produkt in der Hand. Er kann über die verschiedenen Phasen Vermutungen anstellen. Und doch hat er das auszulöffeln, was in den einzelnen Phasen schief ging, also alles, was vorher eingebrockt worden ist. Sie sollten später einmal überlegen, ob Sie diese Schritte bei Ihren Problemlösungen identifizieren können? Haben Sie sie auch ordnungsgemäß absolviert?

Merke: Integrierte Lösungen wie Bedienungsflächen verdecken sehr oft Arbeitsphasen, Denkkonzepte und Abstraktionsebenen

B: In Fortführung der Küchenanalogie sei die Frage erlaubt: Was wäre ein Koch ohne eine anständige Küche? Ein Rezept allein reicht nicht aus, es muss schon eine Werkstatt her, in der der Koch sein Rezept umsetzen kann. Was muss ihm eine Küche bieten? Er braucht mit Sicherheit einen Herd. Wie ist sie strukturiert und aufgebaut? Etwas abstrakter formuliert: Welche Dienste muss eine Küche dem Koch bieten? Kühlung, Wasserversorgung, Eieruhr, Vorratsschränke, Beleuchtung fallen uns sofort ein.

F: Auch der Entwickler von programmtechnischen Lösungen muss eine Umgebung haben, in der er wirken kann. Was muss sein Rechner können? Welche Umgebungselemente setzt die Entwicklung von Programmen voraus und welche Funktionalitäten müssen im Angebot sein? Dem steht das Angebot moderner Rechner gegenüber. Wie sind diese aufgebaut?

- In unserer Aufzählung wurde herausgearbeitet, dass das eigene Programm dauerhaft gespeichert werden muss. Weiter muss das gespeicherte Programm (durch einen geeigneten Compiler) übersetzt werden können. Erst dann kann es vom Anwender gestartet werden. Schon in dieser kurzen Zusammenstellung lassen sich Aufgabenfelder erkennen, die nach ihrer Nähe zur Hardware gegliedert werden können:

Anwenderebene	Einsatz von Anwendungssoftware wie WORD, MAIL, EXPLORER
Problemebene	Erstellung von Anwendungssoftware wie KAUPP
Grenzschicht ↑ ↓	Angebot elementarer Programme wie Eingabeprogramme (Editoren), Compiler
Betriebssystemebene	elementare Verwaltung von Speicher, laufenden Programmen, diversen Geräten
Hardwareebene	Geräte wie Prozessor, CD-Laufwerk, Tastatur, Speicher

Abbildung 1: einfaches Schichtenmodell

Wenn keine weiteren Probleme bei der Arbeit mit Rechnern auftreten, sollte der Anwender sich wirklich nur auf der Anwenderebene aufhalten. Hier im Kurs wollen wir jedoch, wie versprochen, unter die Oberfläche schauen und bis zur Ebene des Betriebssystems vorstoßen.

- B:* Damit ein Koch seine Zutaten wiederfindet, muss er Ordnung halten. Doch welche Ordnung? Und wie kann man darüber sprechen?
- F:* Ob Bücher, CDs, Spiele oder auch Dinge, die in einem Rechner lagern – alles sollte so abgelegt sein, dass es wiederzufinden ist. Deshalb müssen wir, bevor wir uns an KAUPP wagen, hierzu ein paar Gedanken machen. Was lässt sich zur Speicherungs-Frage bemerken? Wo können wir unsere selbst geschriebenen Programme lassen?
- Wenn wir unser noch zu erfindenes Programm eingetippt haben, wollen wir es speichern. Ganz allgemein können wir zu speichernde Dinge als *Daten* bezeichnen. Damit wir die Ansammlung der eingetippten Zeichen, unser Programm, wiederfinden, müssen wir es als Einheit betrachten und mit einem Namen versehen. Solche unter einem Namen abgelegten Einheiten werden als *Dateien* (oder *files*) bezeichnet. Die abstrakten Orte, in denen Dateien gespeichert werden, heißen *Verzeichnisse* (*directories*) oder auch Ordner. Wie Städte zu Kreisen, Kreise zu Bundesländern, Bundesländer zu Staaten gehören, sind die Verzeichnisse hierarchisch angeordnet. Sie bilden einen Baum, der als *Dateisystem* (file system) bezeichnet wird.
- Der Behälter bzw. das Verzeichnis auf der obersten Ebene heißt *Wurzelverzeichnis* oder kurz: *Wurzel* (*root*) und wird oft mit einem Schrägstrich \ bezeichnet. Betrachten wir zur Anschauung einen Ausschnitt eines Dateibaumes, der nicht von unten nach oben, sondern aus zeichentechnischen Gründen von links bis rechts wächst:

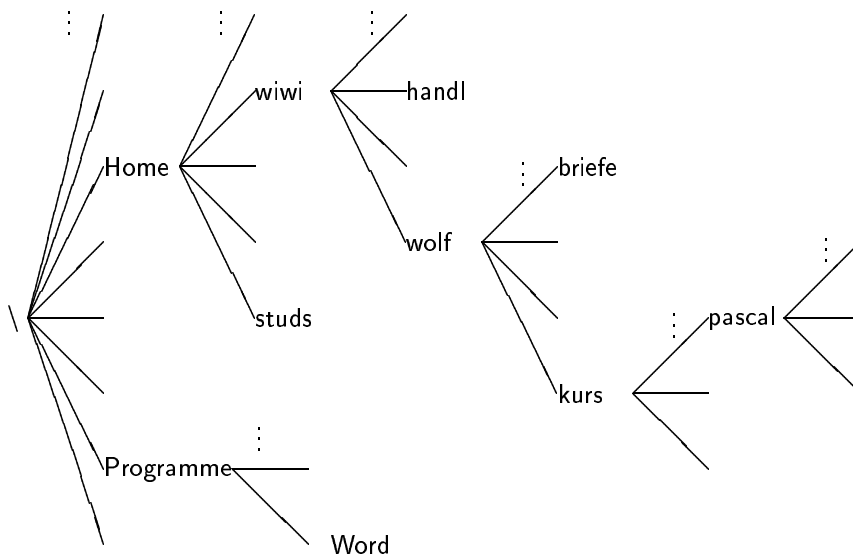


Abbildung 2: *Beispieldateibaum*

Wenn mein Programm KAUPP nur schon fertig wäre, könnte ich in dem Bild auf die Stelle des Dateibaumes zeigen, wo es hingehört:

Gehe von der Wurzel aus nach Home, dann nach wiwi, folge dem Pfad in Richtung wolf sowie Richtung kurs und wechsle zum Schluss in das Verzeichnis pascal. Dieser Weg- oder *Pfad* ist in absoluter Weise, ausgehend von der Wurzel beschrieben worden. Deshalb heißt er *absoluter Pfad*. Notiert wird er durch Aufzählung der besuchten Orte (oder Ordner), getrennt durch ein Trennzeichen. In der WINDOWS-Welt wurde als Trennzeichen der Rückstrich (*backslash*) (\) gewählt. Also lässt sich der Pfad kurz beschreiben durch:

```
\Home\wiwi\wolf\kurs\pascal
```

- Wenn Sie arbeiten, tun Sie dieses an einem speziellen Ort: am eigenen Schreibtisch, in der Bib, im Hörsaal usw. Sie befinden sich also irgendwo. Hier ist es entsprechend. Sie befinden sich an einer speziellen Stelle im Dateibaum und arbeiten dort. Wenn sich der Autor an seinem Rechner anmeldet, wird ihm automatisch der Ort

```
\Home\wiwi\wolf
```

zugewiesen. Oft ist es sehr vorteilhaft, die Welt von diesem Ort aus – also relativ zu diesem Ort – zu betrachten. Von diesem *Heimatverzeichnis* aus gesehen lassen sich die Kurs-Programme in zwei Schritten erreichen: Gehe nach kurs und dann nach pascal. Diese Pfadangabe ist relativ zu dem Ausgangspunkt zu interpretieren und der Pfad heißt *relativer Pfad*:

```
kurs\pascal
```

Befinden wir uns in dem Ordner pascal und wollen den Pfad zu den Briefen erklären, müssen wir im Baum nach oben steigen. Das Verzeichnis über dem Arbeitsverzeichnis wird abgekürzt mit zwei hinter einander stehenden Punkten: .. und wir erhalten:

```
..\..\briefe
```

Übrigens gezeichnet ein einzelner Punkt (.) den aktuellen Ordner.

- Die Diskussion über Orte der Speicherung ist jetzt schon fast abgeschlossen. Wir müssen nur noch verstehen, wie zwischen verschiedenen Speichermedien unterschieden werden kann. Schon aus Sicherheitsgründen sollten wir immer über Sicherungskonzepte, sprich: Sicherungskopien nachdenken. Bisher ist die Diskette das Hauptsicherungsmedium für kleine Dinge, wie zum Beispiel für selbst geschriebene Programmdateien. Solche Datenträger werden mit Hilfe von Diskettenlaufwerken beschrieben und gelesen. Aus diesem Grund hat sich eingebürgert, Orte von Geräten allgemein als *Laufwerke* zu bezeichnen. Als Namen werden einfallslos, wie die Welt nun mal ist, in der Regel einfache Buchstaben verwendet. Oft – also nicht immer – wird das Diskettenlaufwerk mit einem A, die eingebaute Festplatte mit einem C und das CD-Laufwerk mit einem D bezeichnet. Ortsangaben für Dateien lassen sich nun um die Angabe des Mediums erweitern, indem man den Laufwerksbuchstaben dem Pfad durch einen Doppelpunkt getrennt voranstellt:

```
C:\Home\wiwi\wolf\kurs
```

Falls unser Programm, was leider immer noch nicht vorliegt, den Dateinamen `kaupp.pas` bekommt, lässt sich Name mit Ortsangabe als Einheit schreiben:

```
C:\Home\wiwi\wolf\kurs\pascal\kaupp.pas
```

So, damit müsste eine vage Vorstellung von Dateien, Dateibaum, Pfaden und Laufwerken existieren.

- Die Speicherfrage lässt sich auf den verschiedenen, oben aufgeführten Ebenen betrachten. Auf der Hardware-Ebene können wir Materialien diskutieren und zum Beispiel fragen, wie lange die Medien halten. Auch auf dieser Ebene sind Überlegungen angesiedelt, wie es physikalisch gelingt, Informationen zu speichern. Das Betriebssystem ist dafür verantwortlich, dass man verschiedene Speichermedien ähnlich behandeln kann. Oberhalb der Betriebssystemebene kann man mit der einfachen Vision leben, dass auf diesen Medien Dateibäume untergebracht sind. Fügt man die Dateibäume gedanklich zusammen, ergibt sich für den Anwender ein großer gemeinsamer Speicher. Dennoch schlagen Unterschiede der Hardware auf den Anwender durch – der Autor erinnert sich noch mit Grausen an seine ersten Versuche, eine CD in ein CD-Laufwerk einzubringen.

Merke: In der Definition und Bezeichnung von Elementen, Klassen und Strukturen liegt der Schlüssel für den Aufbau beherrschbarer komplexer Systeme wie Informationsspeicher aber auch Programmsysteme.

- B:* Gehört das Vorwärmen von Schüsseln auch zum Kochen selbst? Wenn Kochen das Umsetzen eines Kochrezeptes ist, welche zusätzlichen Handlungen müssen im Rahmen des Kochens noch abgewickelt werden?
- F:* Für den Rechnerbetrieb ist ein Betriebssystem zur Verwaltung elementarer Einheiten und Prozesse unerlässlich. Doch was muss ein Anwender oder ein Programmierer von den Leistungen wissen? Und welches sind die aller wichtigsten Operationen?
- Dem Anwender von heute sind die Leistungen des Betriebssystems oft gar nicht mehr bewußt. Denn er kann seine Wünsche mittels Service-Programmen mit graphischen Oberflächen realisieren, die so hochtrabende Namen wie *Dateimanager* haben. Diese besitzen jedoch den Nachteil, dass nach einer Operation nicht mehr feststellbar ist, welche Anwenderaktion diese ausgelöst hat und was sich genau verändert hat. Der Programmentwickler sollte jedoch nicht nur eine rudimentäre Vorstellung von Dateibäumen, sondern auch von den elementaren Operationen haben, die mit dem Dateibaum zusammenhängen.

- Zum Beispiel muss der Entwickler Operationen kennen, um sein Programm für den Anwender angemessen zu platzieren. Diese Operationen lassen sich gut mit dem alten DOS (DOS=DiskOperatingSystem=diskettenbasiertes Betriebssystem) aus den 80-er Jahren üben. Wer begleitend am Rechner probieren will, muss zunächst eine sogenannte DOS-BOX öffnen. Klicke unter WINDOWS auf *Start*, dann auf *Programme*, dort ggf. auf *Zubehör* und zum Schluss auf den Eintrag mit der Ikone mit den Buchstaben MS-DOS. Es sollte sich ein schwarzes Fenster mit einer Eingabeaufforderung öffnen. Diese DOS-BOX ist nun unser Fenster zum Betriebssystem, in dem wir Aufträge in Form von Befehlen an das Betriebssystem übermitteln können. Ein Auftrag setzt sich zusammen aus dem Kommando und – wenn erforderlich – angehängte Parameter: $\langle \text{Kommando} \rangle \langle \text{Parameter}_1 \rangle \langle \text{Parameter}_2 \rangle$
- Neben dieser allgemeinen Syntax benötigt man nur noch eine Liste der Kommandos mit ihren Bedeutungen und der genauen Syntax. Übrigens hat jedes Betriebssystem eigene Kürzel für die elementaren Aufgaben. Für die ersten Schritte reichen aus:

Bedeutung	MS-DOS	Linux
Verzeichniswechsel	cd	cd
Dateien auflisten	dir	ls
Datei anzeigen	type	cat
Datei kopieren	copy	cp
Datei umbenennen	move oder rename	mv
Verzeichnis erstellen	md oder mkdir	mkdir
Datei löschen	del	rm
Laufwerk wechseln	x:	—
editieren	edit	ed

Tabelle 1: *einige Kommandos zum Umgang mit Dateibäumen*

- Hinweis: In der WINDOWS-Welt wird zwischen großen und kleinen Buchstaben nicht unterschieden, wohl jedoch in der Welt des Linux-Betriebssystems.

→ Für die Syntax dürften als Anschauung einige Beispiel genügen:

Kommando / Beispiele	Bemerkung
cd <i><Pfad></i>	change directory
cd	zeigt Arbeitsverzeichnis an
cd \Home\wiwi	definiert neue Arbeitsstelle: \Home\wiwi
cd ..\briefe	definiert neue Arbeitsstelle: einen nach oben und dann ins Verzeichnis briefe
dir <i><Pfad></i> <i><Parameter₂></i>	directory anzeigen
dir	zeigt Objekte des aktuellen Verzeichnisses
dir ..	zeigt Objekte des übergeordneten Verzeichnisses
dir kurs	zeigt Objekte des untergeordneten Verzeichnisses kurs
dir \Home	zeigt Objekte des Verzeichnisses Home unter der Wurzel
dir \Home /p	zeigt Objekte des Verzeichnisses Home seitenweise
copy <i><von></i> <i><nach></i>	kopiert Objekt
copy karl heinz	legt eine Kopie der Datei karl unter dem Namen heinz ab
del <i><Pfad></i> \(<i>Name</i>)	delete
del a:\test\muell.dat	löscht auf a: Datei muell.dat im Verzeichnis test
move <i><von></i> <i><nach></i>	verschiebt Objekt
rename <i><von></i> <i><nach></i>	benennt Objekt um
type <i><Datei></i>	gibt Dateiinhalt auf Bildschirm aus
type program.pas	gibt Dateiinhalt von program.pas auf Bildschirm aus
type program.pas more	gibt Dateiinhalt von program.pas seitenweise aus
mkdir <i><Pfad></i> \(<i>Name</i>)	make directory
mkdir neuverz	erstellt unter aktuellem Verzeichnis neuen Ordner neuverz
rmdir <i><Pfad></i>	remove directory
rmdir neuverz	löscht Verzeichnis neuverz
x:	wechselt auf Laufwerk <i><X></i>
a:	wechselt auf Diskettenlaufwerk
edit <i><Datei></i>	startet einfaches Editierprogramm

Tabelle 2: DOS-Kommandos mit Beispielen

Merke: Vokabeln und Regeln determinieren die Mächtigkeit von Sprachen und natürlich auch von Programmiersprachen.

B: Wieso wurden eigentlich Spül-, Kaffee- und andere Maschinen erfunden?

F: Was sind Batch- oder Skript-Programme?

→ Betriebssystemanweisungsfolgenausführungsprogramme. Hierbei handelt es sich um Dateien, deren Zeilen Betriebssystembefehle oder Programmaufträge enthalten und deren Name von der Form *<Datei>.bat* ist. Zum Beispiel könnte eine solche Datei *machmal.bat* heißen. Diese Datei wird gestartet durch Eintippen von *machmal* hinter der Eingabeaufforderung. Sie sollten am besten sofort Ihr erstes Batch-Programm schreiben. Nur zu!

Merke: Automatisierungen müssen vor der Nutzung formal beschrieben werden.

Merke: Unsere (Computer-) Technologie ist ein Segen und ein Fluch zugleich. Sie soll uns Werkzeuge zur Vereinfachung an die Hand geben, doch oft werden wir zu ihren (Be-) Dienern. Damit Menschen die Oberhand behalten, müssen sie klüger bleiben. Das geht nur durch

konzeptionelles Verständnis

– das zum Teil nur durch eigene Erfahrungen gebildet werden kann.

4 Pascal-Einleitung

F: Welche zentralen Punkte sind unter der Überschrift zu erwarten?

- Einleitung: Historie, Dialekte, Vorgehen, Beispiel
- Grundsätzliches: Programmaufbau, Regeln, elementare Bausteine
- Handlungsanweisungen: Zuweisungen, Operationen, Unterprogrammaufrufe
- Datentypen: vorhandene und definierbare
- Vereinbarungen: Variablen- und sonstige Deklarationen
- Spezialfragen: Dateizugriffe, Unterprogramme, sonstige Datenstrukturen

Merke: Ziel ist hier nicht die Ausbildung professioneller Programmierer, sondern

elementare Programmier-Konzepte

vorzustellen und zu verstehen. Für deren Vermittlung ist naheliegenderweise eine Sprache zur Beschreibung und zur Demonstration notwendig. Aus diesem Grund wird nicht der Schwerpunkt auf möglichst viele Extras gelegt, sondern auf die Herausstellung von Konzepten, die in vielen anderen Sprachwelten wiederkehren und dort zur Konstruktion und zur Fehlerbehebung unumgänglich sind.

F: Woher stammt Pascal?

- Hoare, Wirth, 1968–1970, problemorientiert, abstrakte Strukturen für Daten, Lern- und Lehrsprache, effizienter Code

F: Was zeichnet den Entwicklungsprozess eines Pascal-Programms aus?

- Oben hatten wir einen 9 Punkte umfassenden Plan aufgestellt. Unter der Annahme, dass die Ideenphase fruchtbar war und schon ein Algorithmus vorliegt, müssen weiter folgen:

0. ggf. *Installation* der notwendiger Hilfsmittel wie Compiler
1. Reformulierung des Algorithmus in der zu ausgewählten *Programmiersprache* (sowie Eintippen und Speicherung)
2. *Übersetzen* in ein auf der konkreten Hardware lauffähiges Programm
3. *Ausführen* des übersetzten Programms

→ Als Abbildung dargestellt ist folgender Prozess zu durchlaufen:

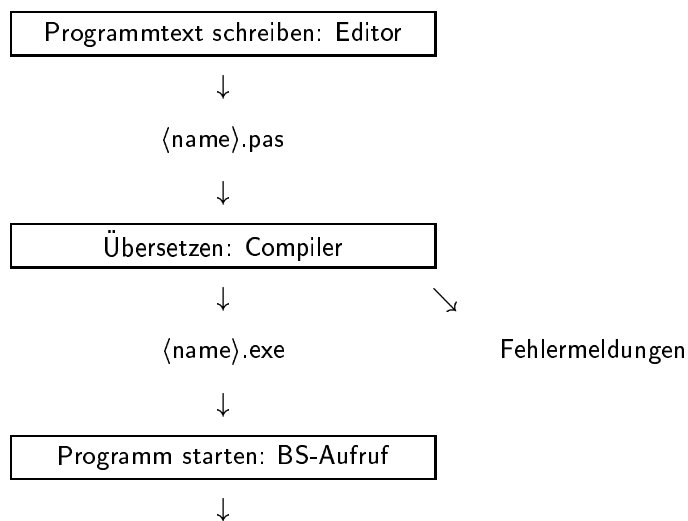


Abbildung 3: *Prozess von der Programmentwicklung zur Nutzung*

Nicht zu vergessen sind: testen, ablegen und verwenden

F: Wie könnte ein einfaches Pascal-Programm, das ein paar Zahlen zusammenzählt, aussehen?

```
program SummeN (Input,Output);
{
  Programm zur Berechnung der Summe der ersten n Zahlen
  pw 06.10.2003
}{ uses winCRT; }{ fuer Turbo-Pascal notwendig }
var
  n,i : integer;
  sum : integer;
begin
  write('Berechnung der Summe ');
  writeln('der ersten n natuerlichen Zahlen');
  write('Bitte geben Sie n ein: n=');
  read(n);
  sum:=0;
  for i:=1 to n do
    begin
      sum:=sum+i;
      { writeln(sum); }
    end;
  writeln('Die Summe der Zahlen von 1 bis n=',n);
  writeln('ist: ', sum);
end.
```

p 1

Programm-Beispiel 1: *Programm zur Summierung ganzer Zahlen*

F: Lassen sich wirklich so einfach Programme schreiben?

→ Ja und nein. Das hängt vom eigenen Wissensstand ab. Zum Beispiel wurden auch bei dem Entwurf des einfachen Programms einige (Flüchtigkeits-) Fehler gemacht:

```
Fehler beim Übersetzen mittels >ppc386 first.pas:
i nicht deklariert,
; vergessen,
string mit " statt ' geschrieben, ein ' vergessen,
falsches Objekt (semantischer Fehler): +1 statt +i
```

Merke: Empfehlenswert ist es, in Fehlersituationen über Fehlerentstehung und Fehlervermeidung nachzudenken!

Merke: Sprachen werden für bestimmte Zwecke entwickelt und ihr Entwicklungsprozess ist im Kontext mit dem zugehörigen Umfeld zu sehen. Pascal ist ein Kompromiss für die Ausbildung, er vereinigt Problemorientierung mit Einfachheit, erlaubt den Einsatz von Unterprogrammtechniken wie auch die Konstruktion problemgerechter, abstrakterer Datenstrukturen. Pascal enthält wichtige elementare Strukturen, die in vielen anderen Sprachen anzutreffen sind. Deshalb ist eine Auseinandersetzung mit den Konzepten von Pascal als Musterbeispiel immer noch empfehlenswert.

5 Basisstrukturen, Bausteine und fundamentale Regeln

F: Welche Grundstruktur besitzen Pascal-Programme?

→ Ein Pascal-Programm besteht aus einer fest definierten Reihenfolge von Teilen. Die Struktur eines Programms ergibt sich aus folgender Abbildung:

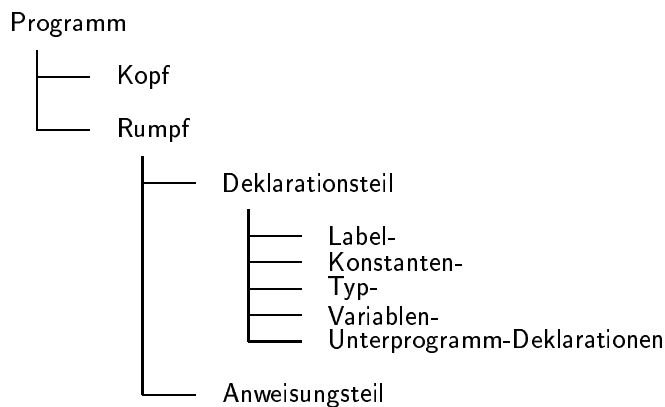


Abbildung 4: *logische Struktur eines Pascal-Programms*

→ Die einzelnen Teile der logischen Struktur werden vom Compiler anhand von Schlüsselwörtern erkannt, wie aus folgender Darstellung zu ersehen ist. Die fett geschriebenen Wörter sind Vokabeln der Sprache von Pascal. Drei Punkte deuten an, dass noch weitere Eintragungen gleicher Art folgen können. Die Bezeichnungen in den spitzen Klammern müssen für ein konkretes Programm durch konkrete Bezeichnungen ersetzt werden.

```

program <Programmname> (<DateiNamen>,...);
label
    <Markenname>, ...;
const
    <Konstantenname>=<konstanter Wert>;...;
type
    <Typname>=<Datentyp>;...;
var
    <Variablenname>:<Datentyp>;...;
begin
    <Handlungsanweisung>;
    <weitere Handlungsanweisungen>
end.

```

Abbildung 5: Aufbau eines Pascal-Programms

F: Welche atomistischen Elemente (Vokabeln) kennt Pascal?

→ Pascal-Programme bestehen aus der Aneinanderreihung von Symbolen, die sich in verschiedene Klassen einordnen lassen.

- Spezialsymbole:

+ - * / = < > <> <= >= . , ; ' () { } [] := .. ^

- Wortsymbole:

begin end program procedure function
and or not if then else case
for to do downto while repeat until
const type var array of file record set
div mod nil in goto label packed with

- Namen / Bezeichner / Identifier:

i, x, KontoNr, Vektor10, Betrag13, ...

- Zahlen:

4 007 +1234 -13 0.1 -0.33333 +31.414E-1
.1 oder 1. sind übrigens unzulässig

- Strings / Zeichenreihen:

'Hello world!' 'A' 'don't worry be happy'

- Kommentare:

{ dies ist ein Pascal-Kommentar }

- Trennzeichen:

zwischen Pascal-Symbolen: "␣", *newline* zwischen Anweisungen: ";" und zwischen Parametern ", "

F: Welche fundamentalen Regeln sind zu beachten?

→ Es gibt wenige unbedingt zu beachtende Regeln:

- Programmbeginn: ein Pascal-Programm beginnt mit Schlüsselwort `program`
- Anweisungsteil: der Teil mit den Handlungsanweisungen beginnt mit `begin` und das Programm endet mit `end`.
- Kommentare: `{` und `}` schließen Kommentartexte ein.
- Große und kleine Buchstaben: haben identische Wirkung.
- `;` trennt eine Anweisung von der nächsten.
- `;` beendet Deklarationsanweisungen.
- Leerzeichen ist wie Newline-Zeichen Trennzeichen.
- Namen müssen mit Buchstaben beginnen und können Ziffern enthalten.
- Wortsymbole sind nicht veränderbar.
- Deklarationen: Vor der Verwendung von Dingen sind diese im Programm exakt einzuführen. Deshalb müssen Variablen oder Unterprogramme vor ihrer Benutzung deklariert werden. Dieser strenge Ansatz erleichtert die Übersetzung, gleichzeitig werden Fehler verhindert, indem zum Beispiel Ergebnisse nur typengerecht abgelegt werden können.

Merke: Exakte Regeln müssen sein. Andernfalls kann eine Maschine Aufträge nicht erledigen. Schwierig ist es, für das

Regelwerk einer Sprache

den richtigen Kompromiss zu finden zwischen der Eignung für das menschliche Denken und für die maschinelle Abwicklung. Andere Sprache besitzen andere syntaktische Regeln als Pascal. Je weniger streng die Regeln einer Sprache sind, desto lockerer lassen sich Programme schreiben, doch wachsen gleichzeitig die Gefahren für (versehentlichen) Fehlgebrauch. In der Regel werden sich angesprochene Fragen wiederholen: Programmstruktur, Elementar-Bausteine, Verknüpfung von Bausteinen, Modellierungsmöglichkeiten von Handlungs- und Datenseite, Strukturierungsregeln.

6 Einfache Handlungsanweisungen

B: Ein Koch hat verschiedene kleine Fingerfertigkeiten – wie *Kneten* oder *Unterheben* – zu lernen. Zählen Sie einige weitere auf.

F: Welche Operationen erkennen wir im Handlungsteil des Beispielprogramm?

→ Addition, `sum+i`. Weiter werden unterstützt:

Operator	Bedeutung
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
div	ganzzahlige Division
mod	modulo: ganzzahliger Rest
=	rechter gleich linkem Operand?
<>	Operatoren verschieden?
<	linker kleiner?
<=	linker kleiner gleich?
>=	linker größer gleich?
>	linker größer?
and	gilt linke und rechte Aussage
or	gilt linke oder rechte Aussage

Tabelle 3: Operatoren von Pascal

→ beachte Prioritätsklassen:

Priorität	Operatoren
sehr hoch	not
hoch	*, /, mod, div, and
mittel	+, -, or
gering	=, <>, <=, >=, <, >

Tabelle 4: Operatoren nach Prioritäten klassifiziert

→ Zuweisung: durch `:=` wird der Inhalt, der sich rechts ergibt, auf der Variablen, die links von `:=` steht, zugewiesen.

→ Schreibbefehle und Lesebefehl: `writeln`, `write`, `read` sind Aufrufe von Prozeduren. Syntax von Prozedur-Aufrufen, spezielle Semantik: `write` gibt (in einfacher Form) die angegebenen Dinge aus, `writeln` erzeugt nach Ausgabe Zeilenvorschub. `read` liest nächstes Objekt, `readln` würde zudem noch Zeilenendezeichen einlesen. Später wird erklärt, dass Pascal neben Prozeduren auch noch Funktionen – wie in der Mathematik gebräuchlich – anbietet, zum Beispiel: `sin(x)`.

→ Die Anweisung `for i:=1 to n do` heißt `for`-Schleife: frei übersetzt bedeutet dieses: erledige die Anweisung(en) hinter `do` zunächst für `i=1`, dann für `i=2` bis `i=n`. Es wird also `n`-mal dasselbe erledigt, jedoch mit veränderten `i`. `i` und `n` sind die Namen von Variablen. Statt `i` kann natürlich auch ein anderer zulässiger Name verwendet werden. Statt `n` kann auch ein auswertbarer Ausdruck stehen. Solche Konstruktionen zur Wiederholung bestimmter Anweisung(s)folgen heißen Schleifen. Neben diesen werden weiter unten noch die Geschwister `while`- und `repeat-until`-Schleife vorgestellt. In diesem Atemzug lassen sich auch noch die Vettern `if-then` und `if-then-else` zur bedingten Ausführung von Anweisungen nennen.

Merke: Konzepte wie Operatoren, auswertbare Ausdrücke, Prioritätsregeln, Variablen, Zuweisungen und Unterprogramme finden wir überall in der Computerwelt. Glücklicherweise gleichen sich oft die vorgeschlagenen Sprachvorschläge. Unglücklicherweise sind Automaten Pedanten. Schon winzige Kleinigkeiten können zeitraubende

Fehlersuchen

auslösen. Zum Beispiel ist in manchen Sprachen = der Zuweisungs- und == der Vergleichsoperator, manchmal kommt es im Gegensatz zu Pascal sehr wohl auf Gross- und Kleinschreibung an. Übrigens muss jeder mindestens einmal im Leben drei Stunden opfern, um dann festzustellen, dass o und 0 sowie 0 völlig verschiedene Zeichen sind.

7 Einfache Datentypen

B: Zutaten. Welche Kochzutaten kennen Sie? Wie würden Sie diese klassifizieren? In welchen Mengengrößen werden Zutaten benötigt, zu welchen Mengen lassen sich verschiedenartige Zutaten zusammenfügen?

F: Welche Daten benötigt das Beispielprogramm in welcher Form?

→ Daten werden auf Variablen gespeichert. Je nach Qualität werden verschiedene Datentypen unterschieden. Der Typ legt den Bereich der Werte fest, die eine Variable annehmen kann. Im Beispielprogramm sind uns nur integer-Variablen begegnet. Erforderlich waren die Variablen `n`, `i`, `sum`.

F: Gibt es noch weitere einfache Datentypen?

→ Als ganz elementare oder skalare Datentypen kennt Pascal:

Datentyp	Bedeutung
Integer	Ganzzahlen
Real	reelle Zahlen
Boolean	Wahrheitswerte
Char	Zeichen
String	Zeichenketten
Longint	Ganzzahlen, sehr viel mehr als bei integer

Tabelle 5: *elementare Datentypen*

→ Pascal kennt einige in der Wirkung ihrem Namen entsprechende Standardfunktionen wie `sin`, `cos`, `exp`, `sqrt`, `abs`, `round`. Deren Ergebnisse sind von einem angemessenen Typ.

Funktion	Typ des Resultats	Bedeutung / Input
<code>sin(x)</code>	<code>real</code>	$\sin(x)$ – Sinusfunktion, x <code>real</code>
<code>cos(x)</code>	<code>real</code>	$\cos(x)$ – Cosinusfunktion, x <code>real</code>
<code>sqrt(x)</code>	<code>real</code>	\sqrt{x} – Wurzelfunktion, x <code>real</code>
<code>exp(x)</code>	<code>real</code>	e^x – Exponentialfunktion, x <code>real</code>
<code>abs(i)</code>	<code>integer</code>	absoluter Betrag von Integer i
<code>trunc(x)</code>	<code>integer</code>	Abschneiden der reellen Zahl x
<code>round(x)</code>	<code>integer</code>	Runden der reellen Zahl x

Tabelle 6: *Standardfunktionen mit Ergebnistyp*

`MaxInt` ist übrigens eine vordefinierte Konstante, die die größte Zahl von Typ `integer` zeigt.

→ Für Objekte vom Typ `Boolean` sind die Operationen `and`, `or`, `not` definiert.

Merke: Auswertbare Ausdrücke gibt es in allen vergleichbaren Programmierumgebungen.

Zentrale Fragen an Programmiersprachen

sind:

- Welche verschiedenen Objektarten (Typen) gibt es?
- Welche Operationen (Operatoren, Funktionen) gibt es für diese Typen?
- In welcher Syntax sind die Handlungen zu notieren?
- Welche Auswertungsregeln – vgl. Klammern und Prioritätsregeln – kommen zur Anwendung?
- Wie werden die Auswertungsergebnisse gespeichert, zugewiesen oder weiterverarbeitet?

8 Variablen und Flusskontrolle: Abrechnungsprogramm

B: Wir wollen bei der Bezahlung im Restaurant großzügig sein und die Bezahlung der Getränke übernehmen. Deshalb wird eine veränderte Software benötigt. Listen Sie wesentliche Eigenschaften auf!

F: Lässt sich nicht wieder elegant ein Programm in Pascal schreiben, das die Umsätze getrennt nach ihrer Qualität zusammenzählt?

→ Klar! Hier ist es:

```

program EssenUndTrinken (Input,Output);
{
  Programm der Getränke-Umsatzsumme
  pw 08.10.2003
}{ uses wincrt; }{ fuer Turbo-Pascal notwendig }

var
  Betrag,   { für Betragseingabe }
  Speisen,  { für den Speiseumsatz }
  Getraenke { für den Getränkeumsatz }
            : real;
  i : integer; { Variable zum Zählen }
  Art : char; { zeigt Art des Verzehrs an }

begin
  { Initialisierung }
  Speisen:=0.0; Getraenke:=0.0; i:=0;
  writeln('Getränksummenberechnungsprogramm');
  { Vorbereitung der Einlese-Verarbeitungsschleife }
  write('Umsatzart eingeben: g=Getränk, s=Speise, x=Exit: ');
  readln(Art);
  { Einlese-Verarbeitungsschleife }
  while Art <> 'x' do
    begin;
      if (Art='s') or (Art='g') then
        begin
          i:=i+1;
          write('Betrag eingeben:'); readln(Betrag);
          { Verbuchen gemäß Art und Betrag }
          if Art='g' then Getraenke:=Getraenke + Betrag;
          if Art='s' then Speisen :=Speisen  + Betrag;
        end
      else
        { falsch eingegebene Verzehrart bearbeiten }
        writeln('Falsche Eingabe');
        write('Umsatzart eingeben: g=Getränk, s=Speise, x=Exit: ');
        readln(Art);
      end;
    { Endauswertung }
    writeln('Es wurden ',i,' Positionen eingegeben. ');
    writeln('Die Summe der Speisen ist',  Speisen:10:2);
    writeln('Die Summe der Getraenke ist',Getraenke:10:2);
    writeln('Hoffentlich bekommt es auch gut!');
  end.

```

Programm-Beispiel 2: *Programm zur Ermittlung von Getränke- und Speisekosten*

F: Und was gibt es über Fehler und zur Entstehung zu berichten?

```

Fehler:
  statt ' Doppeltüddelchen " geschrieben
  2 * then vergessen
  statt Speisen nur Speise eingetippt
  identische Bedingungen in if-Anweisungen durch copy-and-paste
Verbesserungen:
  Anzahl Eingaben, Layout, Formate, Bezeichnungen

```

F: Was zeigt uns das Beispiel zum Thema Variablen und Typen?

→ Wir sehen Verwendungen des Typs real für Beträge. Zwar könnte man sich auch mit einer Integer-Cent-Rechnung behelfen, jedoch würden wir bei der Berechnung von Umsatzsteuern auf Probleme stoßen.

Übrigens werden Integer-Variablen oft mit Namen wie i, j, k, n, m bezeichnet.

Weiter wird anhand einer char-Variablen der Typ des Verzehrs verwaltet.

→ Für Variablen gilt die Reihenfolge: Deklaration, Initialisierung, Gebrauch

Merke: Der für einen bestimmten Zweck gewählte Datentyp muß alle die Werte zulassen, die aufgrund der Problemlage (inklusive zu erwartenden Modifikationswünschen) theoretisch gebraucht werden könnten. Andererseits sollte die

Menge der durch den Datentyp zugelassenen Werte

so klein wie möglich sein. Denn hierdurch werden viele falsche, semantisch unzulässige Inhalte ausgeschlossen.

→ Zuweisung: Eine Zuweisung kann für Pascal beschrieben durch:

`<variable> := <auswertbarer ausdrück>;`

Objekte, die sich nicht ändern, sollten als Konstanten modelliert werden.

Merke: Der Ablauf eines Programmes mit den sich verändernden Speicherzuständen wird

Prozess

genannt. Er zeichnet sich dadurch aus, dass Variablen mit dem Zeitablauf ihren Wert aufgrund von Zuweisungen ändern.

F: Was zeigt uns das Beispiel zur Schleifenfrage?

→ while-Schleife.

In dem Programm SummeN ist uns eine for-Schleife begegnet. Dabei stand vorher fest, wie oft der Rumpf der Schleife durchschritten werden muss. Ist das vorher nicht klar, wird besser eine while-Schleife verwendet. Der Rumpf der Schleife wird dann wieder und wieder durchlaufen, sofern vor dem Durchlauf die Schleifenbedingung wahr ist. Die allgemeine Struktur lautet:

```
while <bedingung> do
  begin
    <auswertbare ausdrücke>
  end;
```

→ repeat-Schleife.

Die repeat-until-Schleife ist der dritte Typ einer Schleife. Im Unterschied zur while-Schleife wird die Schleifenbedingungen erst nach Durchlauf des Schleifenrumpfes betrachtet, so dass diese Schleife mindestens einmal durchlaufen wird. Hier ist die allgemeine Struktur:

```
repeat
  <auswertbare ausdrücke>
until <bedingung>;
```

→ for-Schleife.

Der Vollständigkeit halber sei die for-Schleife noch in ihren zwei Varianten präsentiert. In der Form, bei der die Schleifenvariable nach jedem Durchlauf herauf gezählt (inkrementiert) wird, hat sie die allgemeine Gestalt:

```
for <start der laufvariablen> to <maximaler Wert der laufvariablen> do
  begin
    <auswertbare ausdrücke>
  end;
```

In der anderen Form wird nach jedem Durchlauf die Laufvariable um 1 vermindert (dekrementiert):

```
for <start der laufvariablen> downto <minimaler wert der laufvariablen> do
  begin
    <auswertbare ausdrücke>
  end;
```

Merke: Bei Schleifen sind

erster und letzter Schleifendurchlauf

die Durchläufe mit einer hohen Fehlerrate. Man sollte unbedingt nach Fertigstellung diese beiden Durchläufe nochmals überprüfen.

F: Was für Regeln gelten für Anweisungen, die nur unter bestimmten Bedingungen ausgeführt werden sollten?

→ *if*-Anweisungen erlauben die Definition von Bedingungen für Handlungsaufträge. Es sind zwei Formen anzutreffen.

```
if <bedingung> then
  begin
    <auswertbare ausdrücke>
  end;
```

... hat die Bedeutung: Sofern die angegebene Bedingung erfüllt ist, soll die Anweisung bzw. der Anweisungsblock hinter dem *then* ausgeführt werden. Ein Entweder-Oder wird hingegen von der *if-then-else*-Konstruktion umgesetzt:

```
if <bedingung> then
  begin
    <auswertbare ausdrücke>
  end { kein Semikolon! }
else
  begin
    <auswertbare ausdrücke>
  end;
```

→ Beachte: ein *else* gehört immer zu dem unmittelbar davor stehenden *if*, und vor einem *else* darf nie ein *;* stehen!

→ Für die aufgeführten Beispiele gilt, dass ein Anweisungsblock, der aus einer einzigen Anweisung besteht, nicht durch eine *begin-end*-Klammer verpackt werden muss. Andererseits können jedoch beliebig viele Anweisungen einer Sequenz in mittels *begin* und *end* zusammengefasst werden. Zur Umrahmung des eigentlichen Programms darf das *begin-end*-Paar nicht weggelassen werden.

Merke: Konstruktionen wie die vorgestellten Schleifen und *if*-Konstruktionen heißen

Kontroll-Strukturen.

Die einfachste Struktur ist die Sequenz, die bisher nur implizit vorgestellt worden ist. Der Name geht darauf zurück, dass durch diese Strukturen der Fluss durch das Programm während der Ausführung gesteuert oder kontrolliert wird. Die Typen: Sequenz-, Auswahl- und Wiederholungsstrukturen findet man in den verschiedensten Sprachwelten und sind deshalb essentiell. Für ein konkretes Problem resultiert die Frage, wie sich ein algorithmischer Lösungsvorschlag unter Verwendung der erlaubten Kontrollstrukturen so hinschreiben lässt, dass die Niederschrift möglichst kurz wird und doch verständlich bleibt. Dieses erfordert zum Beispiel, Strukturparallelitäten zu erkennen und mittels Schleifen zu formulieren.

→ Für Interessierte: Pascal bietet auch noch eine *case*-Anweisung an. Was es hiermit auf sich hat und wie sie syntaktisch korrekt eingesetzt wird, möge jeder selbst nachschlagen.

Merke: Stilhinweise:

Man sollte für einen menschlichen Leser nur

sinnvolle Namen

verwenden, damit der Leser inhaltliche Zuordnungsfehler möglichst im Ansatz erkennt. Weiterhin sollten Dinge mit großer Verwechslungsgefahr deutlich unterschiedliche Namen bekommen. Es ist zweckmäßig, sich bestimmte einprägsame Namenskonventionen vorzugeben, damit genügend Kapazität zur Problemlösung verbleibt. Trotzdem bleibt die Erfindung geeigneter Namen eins der schwierigsten Probleme in der Informatik.

Für die

Lesbarkeit von Programmen

ist weiter dringend zu empfehlen, Klammer- und andere Strukturen, so hinzuschreiben, dass mit einem Blick die syntaktische Richtigkeit festgestellt werden kann und Beziehungen zwischen Elementen deutlich werden.

9 Richtig programmieren mit Unterprogrammen

B: Haben Sie schon einmal eine größere Aufgabe erledigt? Stellen Sie sich vor, Sie wollten ein 5 Gänge-Menü für 10 Personen servieren? Wie würden Sie das machen? Die Lösung ist ganz einfach: Für den Nachtisch schicken Sie ihren Gehilfen zur Eisdiele, um Eis zu kaufen. Als eine Vorspeise kann eine Party-Pizza des nahen Pizza-Service dienen. Der Hauptgang kann mit Leichtigkeit von McKing oder einer Pommes-Bude bereitgestellt werden und als zweite Vorspeise zerschneiden Sie selbst ein paar Tomaten. Zum Abschluss gibts Brot und Käse, natürlich auch nicht aus eigener Herstellung. Ihre Aufgabe besteht also darin, die Gänge kreativ auszuwählen, verschiedene Dinge zu strukturieren und das ganze Essen zu managen.

F: Wie löst man in Pascal komplizierte Aufgaben und kommt zu vorzeigbaren Programmen?

→ Verwende Funktionen und Prozeduren! In Pascal lassen sich zusammenhängende Blöcke mit Teil-Lösungen zu Funktionen und Prozeduren zusammenschnüren. Hierdurch können diese kleinen Lösungen einfach in zukünftigen Lösungsvorschlägen integriert werden. Außerdem wird durch solche Strukturierungsinstrumente die Lösung übersichtlicher, lesbarer und damit verständlicher.

Funktionen wie `sin(x)` wurden oben angesprochen. Die Prozeduren `Readln`, `Writeln`, `Read`, `Write` haben wir schon im Einsatz gesehen. Sie erledigen Einlese- und Ausgabeprozesse, ohne dem Anwender zu offenbaren, wie sie das eigentlich anstellen.

→ Betrachten wir zunächst Prozeduren etwas näher. Zur Erinnerung hier eine Prozedur-Beispielanwendung:

```
writeln('Die Summe der Zahlen von 1 bis n=',n);
```

`writeln` ist der Name der Prozedur. Innerhalb der Klammern werden der Prozedur Informationen übergeben, im Beispiel, die zu druckenden Dinge durch Kommata getrennt.

→ Aufruf ohne Parameter – Syntax:

```
<prozedurname>;
```

Aufruf mit Parametern – Syntax:

```
<prozedurname>(<parameter 1>, <parameter 2>, ... );
```

Die Parameter einer Parameterliste werden also mittels Semikola voneinander getrennt. Übrigens werden in der Regel nur Prozeduren mit Informationsaustausch verwendet.

Merke: *Teile und herrsche* ist ein altes Sprichwort, das besonders in der Computerwelt eine neue Relevanz erfahren hat.

Verständlichkeit und Wiederverwendbarkeit

sind die Ziele. Modularität charakterisiert den Weg zum Ziel. Hiernach werden gelöste Dinge in abgeschlossenen Einheiten versteckt, deren innere Struktur für den Anwender völlig belanglos ist, und er nur die Regeln für den Informationsaustausch – die Schnittstellendefinition – kennen muss.

B: Schön wenn es zum Beispiel einen Pizza-Service gibt, doch wie lässt sich ein solcher aufbauen. Das erfordert natürlich ein paar Schritte...

F: Wie kann man zum Beispiel das Schreiben von Meldungen in eine Prozedur sperren?

→ Ganz einfach, hier ist die Definition der Prozedur *ZeigeMeldung*:

```
procedure ZeigeMeldung ( meldung : string; typ : char );
begin
  if (typ='e') or (typ='E') then writeln('!!!Fehler!!!');
  if (typ='w') or (typ='W') then writeln('Warnung:');
  if (typ='h') or (typ='H') then writeln('Hinweis:');
  writeln(meldung);
end;
```

Es ist also das Erkennungswort *procedure* gefolgt vom gewählten Namen sowie einer Liste von Variablen im Kopf der Prozedur-Deklaration anzugeben. Die Liste besteht aus den Variablennamen, die intern verwendet werden, sowie der Angabe des Typs. Achtung: Der Typ des sogenannten *Formalparameters* muss mit dem Typ der übergebenen Information beim Aufruf übereinstimmen!

F: Das hilft noch nicht weiter, wenn das Unterprogramm ein Ergebnis erarbeitet, das wie an das aufrufende Programm zurückgeliefert werden soll. So wie bei der Auswertung einer Funktion.

→ Neben Prozeduren gibt es auch Funktionen. Betrachten wir eine Funktion zur Umsatzsteuerberechnung mit dem Namen: *UmsatzSteuer*. Deren Einsatz könnte so aussehen:

```
Rechnungsbetrag := ...
Steuer := UmsatzSteuer(Rechnungsbetrag);
Rechnungsbetrag := Rechnungsbetrag + Steuer;
```

→ Und hier die Prozedur-Definition in einem kompletten Programm:

```
program ReBetragFunc;
{ uses wincrt; }{ fuer Turbo-Pascal notwendig }
const      UMSTEUERSATZ = 0.16;
var        Rechnungsbetrag, Steuer : real;
{ ----- Beginn der Deklaration der Funktion UmsatzSteuer ----- }
function UmsatzSteuer( betrag : real ) : real;
var
  x : real;
begin
  x := betrag * UMSTAZSTEUERSATZ;
  UmsatzSteuer := x;
end;
{ ----- Ende der Deklaration der Funktion UmsatzSteuer ----- }
begin
  write('Bitte Rechnungs-Betrag ohne Umsatzsteuer eingeben: ');
  readln(Rechnungsbetrag);
  Steuer := UmsatzSteuer(Rechnungsbetrag);
  Rechnungsbetrag := Rechnungsbetrag + Steuer;
  writeln('Rechnungsbetrag mit Umsatzsteuer: ',Rechnungsbetrag:7:2);
end.
```

p 3

Programm-Beispiel 3: *Programm zur Umsatzsteuer mit Funktiondefinition*

- Es sind sicher einige Bemerkungen erforderlich.
Ergebnistyp: Das Ausgabe-Ergebnis einer Funktion ist der Inhalt, den die Variable annimmt, deren Name mit dem der Funktion selbst identisch ist. Hierfür muss die Funktion in ihrer Kopfzeile mit einer Typvereinbarung versehen werden.
Struktur: Funktionen und Prozeduren können zwischen ihrem Kopf und dem Handlungsteil einen Vereinbarungsteil zur Definition lokaler Variablen enthalten. Damit besitzt die Deklaration eines Unterprogramms dieselbe Struktur wie das gesamte Pascal-Programm.
Lokalität: Größen, die außerhalb und vorher definiert worden sind, sind aus Sicht der Funktion (und auch einer Prozedur) globale Größen und können unter ihrem Namen lesend verwendet werden. Unterprogramm-Objekte, die im Kopf einer Funktion oder einer Prozedur oder im Vereinbarungsteil eines Unterprogramms lokal eingeführt worden sind, verdecken gleichnamige Objekte, die außerhalb bekannt sind. Lokal eingeführte Objekte sind nach Beendigung des Unterprogramms auch nicht mehr zugreifbar.

Merke: Wer von wo aus unter welchem Namen wann

Speicherzugriffe

in ausführender, lesender oder verändernder Weise tätigen darf, ist in der Informatik eine zentrale Fragestellung, die immer wieder zu schwierigen und folgeträchtigen Entscheidungen führt.

- F:* Wie lassen sich in einem Unterprogramm auch kompliziertere Übergaben realisieren? Zum Beispiel könnte ja eine Funktion eine ganze Rechnung mit sehr vielen Detail-Infos erarbeiten.
- Für komplizierte Übergaben können im Kopf eines Unterprogramms Variablen mit dem Zauberwort `var` versehen werden. Dieses hat zur Wirkung, dass beim Aufruf nicht eine Kopie der übergebenden Information verwendet wird. Sondern es werden die Speicherplätze der Variablen verwendet, die beim Unterprogrammaufruf genannt werden.
Beispiel in Parallelität zum letzten Beispiel:

```
program ReBetragProc;
{ uses wincrt; }{ fuer Turbo-Pascal notwendig }
const    UMSTEUERSATZ = 0.16;
var      Rechnungsbetrag : real;
procedure UmsatzSteuerAddieren( var betrag : real );
begin
    betrag := betrag + betrag * UMSTEUERSATZ;
end;

begin
    write('Bitte Rechnungs-Betrag ohne Umsatzsteuer eingeben: ');
    readln(Rechnungsbetrag);
    UmsatzSteuerAddieren(Rechnungsbetrag);
    { Variable Rechnungsbetrag wird in UmsatzSteuerAddieren angepasst! }
    writeln('Rechnungsbetrag mit Umsatzsteuer: ',Rechnungsbetrag:7:2);
end.
```

p 4

Programm-Beispiel 4: *Programm zur Umsatzsteuer mit Prozedur-Definition*

- Es sei darauf hingewiesen, dass durch die `var`-Parameter-Übergabe – besonders bei Funktionen – verdeckte Nebeneffekte erzielbar sind, die das Verstehen des Programm extrem erschweren können. Deshalb sind solche Konstruktionen nur mit ausreichenden Kommentaren einzusetzen. Die verschiedenen Möglichkeiten mit ihren Auswirkungen werden in folgendem Demonstrationsprogramm vorgeführt:

```

program paratest (input,output);
{ uses wincrt; }{ fuer Turbo-Pascal notwendig }
var   i_global, i_in, i_inout : integer;
procedure vartest(   proc_i_in   : integer;
                   var proc_i_inout : integer);
var   proc_i_local : integer;
begin
  proc_i_local:=4;
  proc_i_local:=proc_i_local * 10;
  i_global   := i_global   * 10;
  proc_i_in  := proc_i_in  * 10;
  proc_i_inout:= proc_i_inout* 10;
  writeln('IN der Prozedur:');
  writeln('i_global:',   i_global);
  writeln('proc_i_in:',   proc_i_in);
  writeln('proc_i_inout:',proc_i_inout);
  writeln('proc_i_local:',proc_i_local);
end;

begin
  i_global := 1; i_in := 2; i_inout := 3;
  writeln('VOR der Prozedur:');
  writeln('i_global:',i_global);
  writeln('i_in:',   i_in);
  writeln('i_inout:', i_inout);
  { Prozeduraufruf: } vartest(i_in, i_inout);
  writeln('NACH der Prozedur:');
  writeln('i_global:',i_global);
  writeln('i_in:',   i_in);
  writeln('i_inout:', i_inout);
end.

```

```

Ausgabe:  VOR der Prozedur:
          i_global:1
          i_in:2
          i_inout:3
          IN der Prozedur:
          i_global:10
          proc_i_in:20
          proc_i_inout:30
          proc_i_local:40
          NACH der Prozedur:
          i_global:10
          i_in:2
          i_inout:30

```

→ Aus Sicht des Unterprogramms lassen sich folgende fünf Arten des Umgangs mit Variablen unterscheiden.

- globale Variablen:
lesender und schreibender Zugriff auf externe Variablen
kein Eintrag im Unterprogrammkopf oder UP-Deklarationsteil
gefährlich!
- lokale Variablen:
Verwendung innerhalb des UPs ohne Außenwirkung
Deklaration im UP-Deklarationsteil
ungefährlich und nützlich
- Wertübergabe an Unterprogramm:
Deklaration im Unterprogrammkopf **ohne var**
für rein lesende Zugriffe zu empfehlen
- Variablen(adressen)übergabe:
Deklaration im Unterprogrammkopf **mit var**
für auch schreibende Zugriffe notwendig,
möglichst sparsam zu verwenden
- Funktionswertrückgabe (nur bei Funktionen):
Deklaration der Funktion gemäß Ergebnistyp,
Zuweisung des Ergebnisses auf Namen der Funktion
empfehlenswert

Merke: In der Informatik werden die

Parameter-Übergabearten

call by value und *call by reference* unterschieden.

→ Übrigens erlaubt Pascal rekursive und wechselseitige Funktionsaufrufe. Diese Themen gehören jedoch in einen Fortgeschrittenenkurs.

10 Zugriffe auf gespeicherte Datenbestände – Dateien

B: Was müssen Sie eigentlich genau tun, wenn Sie als Gehilfe für einen Freund aus dessen Gewürzschrank ein spezielles Gewürz holen sollen? Wie wäre es mit folgendem systematischen Vorgehen:

1. Schrank zeigen lassen
2. Schrank öffnen
3. Gewürze im Schrank durchgehen
4. gewünschtes Gewürz herausnehmen
5. Schrank schließen

Überlegen Sie nun, ob es Gemeinsamkeiten zu der Suche eines speziellen Rezeptes in einem Küchen-ABC-Buch gibt.

F: Nehmen wir an, dass in einer Datei verschiedene Dinge, zum Beispiel Kochrezepte oder hier passender Verbrauchsmengen von Lebensmitteln, nacheinander abgelegt sind. Was ist wohl zu tun, um diese durchzugehen und zum Beispiel die Summe der Verbrauchsmengen zu ermitteln? Wir gehen davon aus, dass die Verbrauchsmengen in einer Datei stehen, deren Elemente vom Typ `longint` sind.

→ Halbformal funktioniert das Summieren wie folgt:

1. Dateinamen feststellen
2. Datei öffnen
3. Datei bis zum Ende Schritt für Schritt durchgehen und
4. den gefundenen Betrag zur bisherigen Summe hinzu addieren
5. Datei schließen und Summe ausgeben

→ Eine Pascal-Lösung sieht sehr ähnlich aus:

```
program SummeVonZahlenAusDatei;
{
  Programm zur Addition von Zahlen vom Typ longint,
  die in einer Datei abgelegt sind.
  pw 14.10.2003
}{ uses wincrt; }{ fuer Turbo-Pascal notwendig }
var
  Dateiname : string;
  zahl      : longint;
  sum       : real;
  i         : integer;
  FileId    : file of longint;
begin
  writeln('Wie lautet der Dateiname mit den zu addierenden Zahlen?');
  readln(Dateiname);
  { Namen der Datei mitteilen: }
  assign(FileId,Dateiname);
  { Datei zum Lesen oeffnen: }
  reset(FileId);
  { erstes Element aus der Datei lesen: }
  read(FileId,zahl);
  sum := 0;
  while not Eof(FileId) do
    begin
      sum := sum+zahl;
      i   := i+1;
      { naechstes Element aus der Datei lesen }
      read(FileId,zahl);
      writeln('eingelesene Zahl: ',zahl);
    end;
  { Datei schliessen: }
  close(FileId);
  writeln('Es wurden ',i,' Werte in der Datei gefunden. ');
  writeln('Die Summe der Werte ist',sum:10:2);
end.
```

p 5

Programm-Beispiel 5: *Programm zur Summierung ganzer Zahlen aus Datei*

- Übersicht über die Dateioperationen: `FileId` bezeichnet dabei den Namen der Variablen, über den die Datei angesprochen werden soll.

Befehl	Bedeutung
<code>reset(FileId)</code>	Datei zum Lesen öffnen
<code>rewrite(FileId)</code>	Datei zum Schreiben öffnen
<code>read(FileId, <var-name>)</code>	Dateisatz lesen und auf Variablen ablegen
<code>write(FileId, <var-name>)</code>	Variableninhalt an Datei anfügen
<code>close(FileId)</code>	Datei schließen
<code>eoln(FileId)</code>	folgt als nächstes ein Zeilenende?
<code>eof(FileId)</code>	folgt als nächstes das Dateiende?

Tabelle 7: *elementare Dateioperationen*

Merke: Die im Umgang mit sequentiellen Dateien vorgestellten Prozeduren und Funktionen trifft man überall in der Informatik an. Sie bilden den Grundstock von

Dateizugriffsoperationen.

Ergänzen ließe sich die Diskussion noch um Zugriffe, die einen wahlweisen Zugriff auf Datensätze erlauben, und um die Verwendung von Zeigern, mit denen Speicherplätze über Adressen angesprochen werden können.

- Für eine Datei mit Elementen vom Typ `longint` muss vorher die Deklaration fixiert worden sein:

```
FileId : file of longint;
```

Das bedeutet, zum Lesen einer Datei, die mit Pascal geschrieben worden ist, ist die genaue Kenntnis des Typs erforderlich. Anderfalls werden die abgelegten Inhalte falsch interpretiert.

Merke: Aus tiefster Sicht sind Dateien nichts anders als eine Abfolge von Bits, Bytes oder Zeichen. Für die richtige Interpretation muss deshalb stets mitgeteilt werden, wie man

Zeichen zu semantischen Einheiten zusammenfassen

muss. Bei Text-Dateien erkennt der menschliche Betrachter sofort Wörter und Zahlen, da er unbewusst die passenden Interpretationsregeln anwendet. Einem Programm müssen jedoch diese Regeln genau einverleibt werden. Im Gegenzug haben wir (Menschen) kaum eine Chance, Informationen, die in internen Formaten abgelegt sind, zu verstehen. Wir benötigen dann ein passendes Übersetzungsprogramm – zu neudeutsch: einen Browser.

- Man kann auch Dateien lesen, die aus für uns lesbaren Zeichen bestehen. Ein längeres Beispiel ist dazu im Anhang zu finden. Der Typ des File-Identifiers muss dann `Text` lauten. Eine Anwendung zeigt folgendes kleine Programm, das nur Zahlen einliest und wieder ausgibt:

```

program ZahlenAusDateiEinlesen;
{
  Programm zum Einlesen von Zahlen vom Typ longint
  pw 14.10.2003
}{ uses wincrt; }{ fuer Turbo-Pascal notwendig }
var
  Dateiname : string;
  FileId    : Text;
  zahl      : longint;
begin
  writeln('Wie lautet der Name der Zahlen-Datei?');
  readln(Dateiname);
  assign(FileId,Dateiname);
  reset(FileId);
  writeln('Folgende Zahlen wurden gefunden:');
  while not eof(FileId) do
    begin
      read(FileId,zahl);
      writeln(zahl);
    end;
end.

```

Programm-Beispiel 6: *Programm zum Lesen von Text-Dateien*

- Dateien können auch nur während der Programmlaufzeit verwendet werden, so dass sie nach Beendigung im Dateisystem nirgends zu finden sind. Da sich gerade bezüglich des Umgangs mit Dateien verschiedene Pascal-Compiler unterscheiden, wird hier im Grundkurs der Umgang mit Datei nicht weiter vertieft.

Merke: Gemeinsam ist allen Kommunikationen, sei es mit Dateien oder mit Geräten, dass immer zwei unterschiedliche Qualitäten von Informationen ausgetauscht werden müssen:

Steuerinformationen und Dateninformationen

Wenn Probleme auftreten, ist zunächst Steuerinformationsaustausch zu überprüfen und erst als zweites die Datenseite.

11 Weitere Typfragen: array, record, set, ...

- B:* Küchen werden zurecht so eingerichtet, dass Kinder an gefährliche Dinge nicht herankommen, nach dem Motto: Sicherheit durch Beschränkung.
- F:* Oft benötigt man nur Teilmengen der Wertebereiche einfacher Typen. Zum Beispiel werden bis jetzt bei Postleitzahlen keine Zahlen größer als 99999 verwendet. Was bietet Pascal für solche Beschränkungen an?
- *Unterbereiche* heißt das Stichwort. Für Postleitzahlen lässt sich der Bereich durch eine Variablendeklaration der Form

```

const
  MaxPLZ = 99999;
var
  PLZ : 1..MaxPLZ;

```

umsetzen.

B: Wie berichten wir elegant von unserem letzten Lebensmitteleinkauf? Ganz einfach: wir halten unseren Kassenbon hoch, der den Einkauf als Einheit beschreibt.

F: Wie lassen sich in Pascal Dinge zusammenfügen?

→ Zum Beispiel lassen sich Geldbeträge in einem sogenannten Vektor (array) aus Zahlen zusammenfassen und unter einem Namen, zum Beispiel: `einkaufsbetraege`, ansprechen. Hierzu benötigt man die Deklaration

```
const
  MaxAnzahl = 100;
var
  einkaufsbetraege : array[1..MaxAnzahl] of real;
```

Damit haben wir einen Vektor für reelle Zahlen der Länge 100 beantragt. Nun können wir beispielsweise das 37-ste Element per Zuweisung verändern, indem wir dort 3.99 ablegen:

```
einkaufsbetraege[37] := 3.99;
```

Die eckigen Klammern beschreiben ein Zugriff auf Position oder Index 37, deshalb heißen sie auch Indexklammern und der Zugriff Indexzugriff. Entsprechend verläuft ein lesender Zugriff:

```
x := einkaufsbetraege[37];
```

Als Anwendungsbeispiel folgt mal wieder ein Additionsprogramm:

```
program SummeVonZahlenMitTastatur;
{
  Programm zur Addition von Zahlen,
  die per Hand einzugeben sind.
  pw 14.10.2003
}{ uses wincrt; }{ fuer Turbo-Pascal notwendig }
const  MaxAnz = 100;
var
  betraege  : array [1..MaxAnz] of real;
  sum       : real;
  i         : integer;
begin
  i:=0;
  repeat
    begin
      i:=i+1;
      writeln(i,'-ten Betrag eingeben (0=Ende):');
      read(betraege[i]);
    end;
  until (i > (MaxAnz-1)) or (0=betaege[i]) ;
  sum := 0;
  write('Die Summe der ',i);
  while i>0 do
    begin
      sum := sum+betaege[i];
      i  := i-1;
    end;
  writeln(' eingegebenen Zahlen lautet: ',sum:10:2);
end.
```

p 7

Programm-Beispiel 7: *Programm zur Summierung unter Verwendung eines array*

- Eigene Typen lassen sich im Vereinbarungsteil hinter dem Schlüsselwort `type` definieren. Die allgemeine Struktur lautet:

$\langle \text{neuer-Typname} \rangle = \langle \text{Typkonstruktion} \rangle;$

Dieses macht erst für kompliziertere Typen Sinn. Zum Beispiel ließe sich der Deklarationskopf oben umformulieren zu (ergänzt um eine alternative PLZ-Begrenzung):

```
const MaxPLZ;
type
  MYVECTOR = array [1..MaxAnz] of real;
  PLZTYPE = 1..MaxPLZ;
var
  betraege : MYVECTOR;
  plz      : PLZTYPE;
```

- Zum `record`-Typ sei nur so viel gesagt, dass mit ihm im Gegensatz zu einem `array` auch verschiedene Typen zu einer Einheit zusammengebunden werden können. Nähere Diskussionen erfordern einen F-Kurs. Besonders im Zusammenhang mit dynamischen Strukturen werden diese Typen interessant. Klingt spannend, nicht wahr?
- Wer aus der Mathematik mit Mengen vertraut ist, dem wird die Möglichkeit erfreuen, Typen durch Aufzählung festlegen zu können:

```
type Tag = (Mo, Di, Mi, Fr, Sa, So);
var heute : Tag;
```

Erkunden Sie, was es in Pascal mit `set` auf sich hat. Leider werden hier diese Hinweise nur wie Knochen dem Hund hingeworfen. Doch wer erst einmal Appetit entwickelt hat, wird selbständig seine Nase in ein Buch stecken.

Merke: Jedes Produkt, das vorgibt, eine Programmiersprache umzusetzen, bietet neben den elementaren Kernfunktionalitäten ein reichhaltiges Angebot an Sonderfähigkeiten, Zusatzroutinen, Unterprogrammibliotheken usw. an. Das ist gut, weil man vor der selbständigen Lösung schauen kann, ob nicht schon eine Lösung existiert. Es ist aber auch schlecht, weil man in der Angebotsfülle ertrinken kann und man dann den Wald vor lauter Bäumen nicht mehr sieht. Deshalb lautet hier die Empfehlung: Nicht einwühlen, sondern immer nach elementaren Prinzipien und Strukturierungsregeln Ausschau halten, sprich:

suche nach Konzepten!

12 Praktische Hinweise

F: Und wie sollte man vorgehen auf dem Weg vom Problem zur Lösung?

- Leider gibt es hier unendlich viele Antworten: Sammele, ordne, strukturiere, baue Bausteine, beschreibe, zerlege das Problem, füge Bausteine zusammen, berücksichtige schon bei der Konstruktion die Behebung von möglichen Fehlern und

Merke: vergesse niemals zu dokumentieren

— **niemals!**

- Das Ergebnis muss auch später noch verständlich sein, die Bausteine sollten in sich abgeschlossen sein, Namen müssen zweckmässig gewählt sein, Teillösungen müssen isoliert prüfbar sein, Schnittstellen müssen klar sein.

- Viel lässt sich aus Fehlersituationen lernen. Besonders, wenn man richtig feststellt. Versuche Fehler einzukreisen. Bis zu welcher Stelle ist alles ok? Welche Teile sind in Ordnung, wie lässt sich das überprüfen? Welche Bedingungen müssen vor einem Block herrschen? Überprüfen Sie diese! Was soll die Einheit eigentlich erarbeiten? Lassen Sie sich Zwischenergebnisse ausgeben! Welcher Zustand muss nach Blockbeendigung eingetreten sein? Wie unterscheidet sich der erwartete vom beobachteten Zustand? Machen Sie wohlüberlegte und systematische Fehlersuchexperimente! Werden Sie im Zweifelsfall bei einem Fehler nicht hektisch, sondern legen eine kurze Gedankenpause (Kaffee?!) ein. Dann gehen Sie die Stellen vor der mutmaßlichen Fehlerstelle durch. Bedenken Sie, dass sehr oft auf die Frage: "Ich habe genau hier einen Fehler, aber ich finde ihn nicht!" zunächst von erfahrenden Leuten die Gegenfrage gestellt wird: "Wo ist denn Ihrer Meinung nach der Fehler auf keinen Fall zu finden? Dort sollten wir mit der Suche beginnen...!"
- Dem Autor sind bei der Erstellung dieses Skripts auch viele Fehler unterlaufen, hier die häufigsten:
 1. ; vergessen (Syntaxfehler)
 2. Identifier (Name) falsch geschrieben (Tippfehler)
 3. Variablen falsch deklariert (semantischer Fehler)
 4. . am Programm-Ende oder) vergessen (Flüchtigkeitsfehler)
 5. Typ einer Funktionsdefinition vergessen (Verwechslungsfehler)
 6. zusammengesetzte Bedingungen falsch geklammert (Prioritätsregelfehler)
 7. ! statt not zur Verneinung verwendet (Vokabelfehler)
 8. <> statt = bei Bedingung (Logikfehler)
 9. readLN statt read für getypte Datei (Verständnisfehler)
 10. i falsch gesetzt (Initialisierungsfehler)

Merke: Fehler können auf fundamentale

Missverständnisse

hinweisen. Beheben Sie nicht nur den Fehler, sondern arbeiten Sie an Ihrem Weltbild. Dann ist das Lehrgeld gewinnbringend angelegt.

- Und beachten Sie, dass Anfänger wirklich sehr viel länger an kleinen Problemen sitzen als sie vermuten. Das ist wirklich normal! Doch glauben Sie, es zahlt sich später einmal aus: Ohne Fleiß kein Preis.

Merke: Konstruieren Sie ihre Lösung so, dass diese aus

leicht verstehbaren Bausteinen in einer einfachen, kontrollierbaren Architektur

zusammengesetzt ist. Dann haben Sie eine große Chance, dass Ihr Ansatz im Prinzip sofort läuft und nur Kleinigkeiten repariert werden müssen.

- Beachte:

Wir können die Technik nicht aus unserem Leben verbannen, die Verkehrsmittel nicht und inzwischen auch die Computer nicht. Umso wichtiger aber ist es, daß wir darüber nachdenken, wie wir mit den Errungenschaften der Technik in Zukunft umgehen sollen und wollen.

Prof. Dr. Josef Weizenbaum

13 Literatur

- Im Internet finden sich eine grosse Zahl von Kursmaterialien, so dass ein Suchender schnell fündig wird. Siehe zum Beispiel:
<http://www.uni-koblenz.de/~schulze/pascal/files/anhang.pdf>
Suchen und finden Sie!
- Weiter ist ein kurzer Umdruck im HRZ für wenig Geld erhältlich:
RRZN (2001): Pascal – ISO-Standard 7185, Uni Hannover.
- Zu Ehren von Wirth sei hingewiesen auf:
Niklaus Wirth (1983): Algorithmen und Datenstrukturen.
- die erwähnten Arbeitsmaterialien (AM-1 und AM-2) sind zu finden unter
<http://www.wiwi.uni-bielefeld.de/~spitta/download.html>
- zu dem Kurs gibt es eine allgemeine Kursseite
<http://www.wiwi.uni-bielefeld.de/~spitta/lehre/GrundkursProg.html>
- und eine spezielle für den Kurs WS 2003/04:
<http://www.wiwi.uni-bielefeld.de/~wolf/lehre/ws0304/progkurs/progkurs.html>

14 Anhang: Zahleneinleseprogramm

Das im folgenden abgedruckte Programm liest Zahlen aus einer Textdatei ein und schreibt sie in eine Output-Datei vom Typ longint. Ein solches Programm kann als Startpunkt hilfreich sein, wenn man dem Input-Device nicht trauen kann und die Eingaben erst überprüft werden müssen.

p 8

```
program InputToFile;
{
  Programm zum Einlesen von Integer-Werte aus einer Datei und
  deren Abspeicherung in einer Datei im Pascal-internen Format.
  pw 14.10.2003
  Syntax: Programm starten, dann den Namen der Input-Datei,
         dann den der Output-Datei angeben und drittens,
         zur Sicherheit eine Obergrenze fuer die einzulesenden Zeichen
  unter linux ist zur Uebersetzung der Befehl notwendig: pcc386 atopas.pas
  mit Turbo-Pascal ist erforderlich die Anweisung:
}{ uses wincrt; }{ fuer Turbo-Pascal notwendig }
var
  zeichen      : char;           { merkt eingelesene Zeichen }
  zahl         : longint;        { zahl merkt gefundene Zahl }
  ziffer,i     : integer;        { ziffer merkt gefundene Ziffer,
                                 i zaehlt Schleifendurchlaeufe }
  InputFile    : Text;           { Variable zum Lesen der Input-Datei }
  OutputFile   : file of longint; { Variable zum Schreiben der Output-Datei }
  fertig       : boolean;        { zeigt gefundenes Dateiende an }
  teilzahlgelesen : boolean;     { zeigt das Lesen einer Zahl an }
{ die Prozedur DebugHilfe ist entworfen worden, um waehrend der Arbeit
  Informationen ueber den Gang der Dinge auszugeben }
procedure DebugHilfe (typ      : char;      message : string;
                     zeichen : char;      int      : longint;      nl : Boolean);
begin
  if (typ = 'e') or (typ = 'E') then write('ERROR:', message);
  if (typ = 'z') or (typ = 'Z') then write('Zeichen:', zeichen);
  if (typ = 'm') or (typ = 'M') then write('Meldung:', message);
  if (typ = 'i') or (typ = 'I') then write('Integer:', int);
  if nl then writeln();
end;
{ die Prozedur DateienOeffnen leistet die Initialisierungsarbeit bzgl. Dateien }
procedure DateienOeffnen ( var InFile : Text; var OutFile : file of longint );
var
  Ascii, Pascal : string;
begin
  { Dateien oeffnen }
  writeln('Wie lautet der Dateiname der ascii-Datei?'); readln(Ascii);
  writeln('Wie lautet der Dateiname der Output-Datei?'); readln(Pascal);
  assign(InFile,Ascii); assign(OutFile,Pascal);
  reset(InFile); rewrite(OutFile);
end;
{ die Prozedur FindeZiffer ermittelt, welche Ziffer hinter einem Zeichen steckt }
function FindeZiffer (zeichen : char) : Integer;
begin
  FindeZiffer := ord(zeichen)-ord('0');
end;
{ die Prozedur IstZeichenZiffer stellt fest, ob ein Zeichen eine Ziffer ist }
function IstZeichenZiffer (zeichen : char) : Boolean;
begin
  IstZeichenZiffer := (ord(zeichen) >= ord('0')) and (ord(zeichen) <= ord('9'))
end;
{ Beginn des Hauptprogramms }
begin
  { Initialisierungen }
  DateienOeffnen(InputFile,OutputFile);
  zahl := 0;
  fertig := FALSE;
  teilzahlgelesen := FALSE;
  writeln('Anzahl maximl zu lesender Zeichen: '); readln(i);
```

```

{ Einleseschleife }
while not fertig and (i>0) do
  begin
    i:=i-1;
    read(InputFile,zeichen);
    DebugHilfe('m','next char','x',0,TRUE);
    DebugHilfe('z',' ',zeichen,0,TRUE);
    { handelt es sich um eine Ziffer? }
    if IstZeichenZiffer(zeichen) then
      begin
        ziffer := FindeZiffer(zeichen);
        zahl := zahl*10 + ziffer;
        teilzahlgelesen := TRUE;
      end;
    { handelt es sich um ein Leerzeichen? }
    if zeichen = ' ' then
      if teilzahlgelesen then
        begin
          DebugHilfe ('m','Zahl identifiziert','x',zahl,TRUE);
          DebugHilfe ('i',' ','x',zahl,TRUE);
          write(OutputFile,zahl);
          zahl:=0; teilzahlgelesen:=FALSE;
        end;
      { handelt es sich um ein Zeilenende? }
      if EoLn(InputFile) then
        begin
          DebugHilfe ('m','EOLN gefunden','x',0,TRUE);
          if teilzahlgelesen then
            begin
              DebugHilfe ('m','Zahl identifiziert','x',zahl,TRUE);
              DebugHilfe ('i',' ','x',zahl,TRUE);
              write(OutputFile,zahl);
              zahl:=0; teilzahlgelesen:=FALSE;
            end;
          read(InputFile,zeichen);
        end;
      { handelt es sich um das Dateiende? }
      if EoF(InputFile) then
        begin
          DebugHilfe ('m','EOF gefunden','x',0,TRUE);
          Fertig := TRUE;
        end;
      end;
    { Dateien schliessen }
    close(InputFile); close(OutputFile);
    DebugHilfe ('m','Dateierstellen fertig','x',0,TRUE);
    { zur Kontrolle noch einmal lesen und zeigen }
    DebugHilfe ('m','Dateiausdruck zur Kontrolle','x',0,TRUE);
    reset(OutputFile);
    while not EoF(OutputFile) do
      begin
        read(OutputFile,zahl); writeln(zahl);
      end;
    close(OutputFile);
    DebugHilfe ('m','Programm fertig','x',0,TRUE);
  end.

```