Daten und Algorithmen

1 Grundbegriffe: Nachricht und Information

Fragen: Was ist "Information"?

Was sind "Nachrichten"?

Wie wird Information weitergegeben?

Wie wird Information dargestellt?

Neben der Energie (und der Materie) ist Information eine zweite Basisgröße von universeller Bedeutung. Ihre Eigenständigkeit wurde erst spät erkannt, da ihre Weitergabe immer an energetische oder materielle Träger gebunden ist. An Informationen gelangt man über Nachrichten. Die Definition von Information in der Informatik deckt sich nicht ganz mit dem umgangssprachlichen Gebrauch.

> Information ist die Kenntnis über Irgendetwas.
> Nachrichten dienen zur Darstellung von Information.

Anders formuliert:

> **Die abstrakte Information wird durch die konkrete Nachricht mitgeteilt.**

Nachrichten und Informationen sind nicht identisch, insbesondere kann die gleiche Nachricht (mit gleicher Information) auf verschiedene Empfänger unterschiedliche Wirkung haben. Es gibt aber auch Nachrichten, die subjektiv keine Information enthalten.

> Beispiel: Bei welcher Nachricht ist die Information größer?
> a) Am 1. Juli war die Temperatur größer als 25 Grad.
> b) Am 1. Juli betrug die Temperatur 29 Grad.

Bei a) gibt es nur zwei Möglichkeiten (kleiner/größer 25 Grad), bei b) sind theoretisch beliebig viele Möglichkeiten gegeben. Also ist bei b) die Information größer. Daraus folgt, dass Information mit der Zahl der Möglichkeiten zu tun hat.

> Beispiel: Wie komme ich zu meiner Freundin?
> Einfacher Weg: Sie wohnt in derselben Straße. Es gibt nur eine Entscheidung (nach rechts oder nach links gehen).
> Komplizierter Weg: Es gibt mehrere Abzweigungen; bei jeder Gabelung muss entschieden werden, ob man rechts oder links geht.

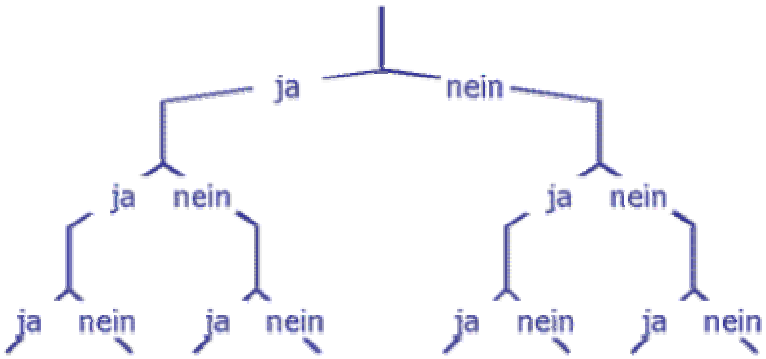
Der Informationsgehalt einer Nachricht ist also feststellbar und wird durch die Anzahl der (rechts-links) Entscheidungen bestimmt und wird in "bit" gemessen. 1 bit entspricht dabei einer Entscheidung:

Einfacher Weg: Information 1 bit

Komplizierter Weg: 2 Weggabelungen --> 4 Möglichkeiten, 2 bit
3 Weggabelungen --> 8 Möglichkeiten, 3 bit
4 Weggabelungen --> 16 Möglichkeiten, 4 bit
usw.

1 bit ist die Datenmenge, die mit der Antwort auf eine Ja-Nein-Frage übertragen wird.
Mit einem Binärzeichen wird nur dann 1 bit übertragen, wenn die beiden Zeichen des Zeichenvorrats gleichwahrscheinlich sind. In allen anderen Fällen wird weniger als 1 bit übertragen.

Beträgt die Zeichenzahl $N = 2^n$, so werden n bit übertragen. Man könnte dies als n aufeinander folgende Antworten auf jeweils eine Ja-Nein-Frage auffassen.



Jedes Astende (Blatt) entspricht dann einem Zeichen, also ergeben sich $2^3 = 8$ Zeichen, die hiermit übertragen werden können.

Darstellung von Information

Nach der abstrakten Information nun zu konkreten Darstellung der Information, der Nachricht. Dazu einige Definitionen:

Nachricht:
Zusammenstellung von Symbolen (Zeichen) zur Informationsübermittlung.

Symbol:
Element eines Symbol- oder Zeichenvorrates. Dieser Vorrat ist eine festgelegte endliche Menge von verschiedenen Symbolen (= Elemente der Menge). Der Unterschied zwischen Symbol und Zeichen ist recht subtil. Ein Symbol ist ein Zeichen mit bestimmter Bedeutung.

Alphabet:
Ein geordneter Vorrat von Symbolen.

Wort:
Folge von "zusammengehörigen" Zeichen, die in einem bestimmten Zusammenhang als Einheit betrachtet werden.

Beispiel:
Alphabet: A,B,C,D,E,F,...,X,Y,Z
Wort: DONALD
Nachricht: DONALD SUCHT DAISY

Nachrichten können durch Signale physikalisch dargestellt werden, z. B. Schallwellen, elektromagnetische Wellen, Ströme, Spannungen, Licht, etc. Dabei wird zwischen digitalen und analogen Signalen unterschieden.

Analoge (wertkontinuierliche) Signale:

- Kontinuierliche Zuordnung einer physikalischen Größe zum Informationsgehalt einer Nachricht, z. B. Temperatur <--> Höhe der Quecksilbersäule, Zeiger eines Messinstruments
- meist sehr anschaulich
- es sind beliebige Zwischenwerte darstellbar
- benachbarte Werte sind oft schwer zu unterscheiden

Digitale (wertdiskrete) Signale:
Hier gibt es nur eine endliche Zahl von möglichen Zuständen einer physikalischen Größe (im einfachsten Fall zwei), z. B. Ziffernanzeige eine Messinstruments, Folge von Ziffern. Da die meisten physikali-

schen Größen analog sind, wird oft eine Quantelung vorgenommen, z. B. eine Spannung zwischen 6,5 V und 7,5 V auf 7 V gequantelt.

Daher unterscheidet man zwischen Analogtechnik und Digitaltechnik. In den normalerweise verwendeten Digitalrechnern wird eine Digitaltechnik mit nur zwei möglichen Werten eingesetzt. Diese beiden Signale (meist durch "0" und "1" dargestellt) müssen in irgendeiner Form aus dem Signal erkennbar sein. Je nach verwendeter physikalischer Größe ergeben sich oft mehrere eindeutige Möglichkeiten für die "0" oder "1".

Informatik

Vereinfacht ausgedrückt kann man sagen: Die *Informatik* ist die Lehre von der Information.

Es stecken aber zahlreiche Gesichts- und Ansatzpunkte dahinter. Eine Information oder ein Datum ist etwas das uns eine Mitteilung macht. Diese können wir aufnehmen oder auch (als unwichtig) ignorieren. Oft sind die Datenmengen so immens, dass wir sie nur aufnehmen können, wenn sie entsprechend aufgearbeitet, z. B. reduziert sind. Daten sollen bei Bedarf abrufbar gespeichert sein und an beliebige Stellen transportiert werden können. Wir bedienen uns für diese Aufgaben oft der Hilfe von Maschinen und Computern. Diese müssen in der Lage sein, unsere Wünsche der Datenaufbereitung zu erfüllen. Das bedeutet aber, dass wir die Fähigkeit haben müssen, Ihnen unsere Wünsche (Anweisungen) auch mitzuteilen. Es muss eine Art Sprache zwischen Mensch und Maschine bestehen, die beide verstehen. Solche Anweisungen können nicht nur zur Verarbeitung von Daten dienen, sondern auch für zahlreiche Tätigkeiten auch im Sinne von Automaten und Robotern.

In diesen Bereichen Forschung zu betreiben und neue (bessere) Möglichkeiten zu finden ist die Aufgabe der Informatik. Sie hat daher einerseits mit Theorien zu tun wie das Verwalten von Daten und Entwickeln von Datenstrukturen, der Entwicklung von Sprachen und Abläufen (Algorithmen) für den Betrieb von virtuellen, simulierten und realen Maschinen und sich überhaupt mit der Lösung und Lösbarkeit gestellter Probleme zu befassen. Dies geht hin bis zu abstrakten Theorien. Andererseits hat die Informatik auch mit der Entwicklung praktischer Geräte, Maschinen, Prozessoren usw. zu tun und Kommunikationsmöglichkeiten (Schnittstellen) zu entwickeln. Hier grenzt sie an den Bereich der Elektrotechnik (Datentechnik), die letztendlich die gewünschten Schaltungsvarianten entwickeln und bereitstellen muss. Auch bestehen hier Verbindungen zum Maschinenbauer, der die Modelle von maschinellen Automaten und Robotern entwickelt, während der Informatiker die Steuerungsaufgaben (Programmierung) übernimmt.

Zu Beginn wollen wir uns vor allem mit einem praktischen Aspekt der Informatik auseinandersetzen. Sie lernen kennen, wie man Abläufe darstellen und werden auch eine Programmiersprache also ein Kommunikationsmittel kennen lernen. Hierzu gehört auch das Erfassen und Darstellen von Daten und zugehöriger Datenstrukturen und programmtechnischer Ablaufstrukturen.

2 Der Computer und seine Fähigkeiten

Es mag Sie erstaunen, dass im gesamten Skriptum kaum je von Computern gesprochen wird, obwohl es doch um das Programmieren von Computern geht. In der Regel braucht ein Programmierer heute kaum noch etwas darüber zu wissen, was im Detail im Computer vor sich geht. Eine modellhafte Vorstellung genügt zunächst.

Auch der normale Autofahrer braucht keine technischen Einzelheiten des Motors zu kennen. Doch sollte man einige grundlegende Dinge verstanden haben, wenn man mit dem Auto bzw. dem Computer sicher umgehen will.

In diesem Abschnitt werden einige Anwendungsfälle aufgezählt, die Arbeitsweise von Mensch und Computer verglichen und die Frage gestellt, wie der Computer zu seinen Fähigkeiten kommt.

Nehmen Sie als Beispiel moderne Kaufhauskassen. Sie schreiben jeden Betrag, mit Erklärung versehen, auf den Kassenbon. Das kann natürlich jede alte Registrierkasse auch. Was ist also Besonderes an den modernen Kassen? Die neuen Kassen können einem Computer mitteilen, wieviel von jedem Artikel im Lauf des Tages verkauft wurde. Wenn man dem Computer vorher mitgeteilt hatte, wie viele

Pakete Waschmittel im Regal standen, kann der Rechner am Abend nicht nur die Kassenabrechnung erledigen, sondern auch ausdrucken, ob das Regal aus dem Lager nachgefüllt werden muss. Das konnten die alten Kassen natürlich nicht. Zudem können diese Kassenterminals noch viel mehr, nämlich Tippfehler korrigieren (wenn z. B. Preis und Artikelnummer nicht zusammenpassen), Sonderangebote berücksichtigen (auch wenn der alte Preis noch auf der Ware steht), Preisfragen beantworten und Absatzstatistiken erstellen.

Kasse und Computer übernehmen also etliche Aufgaben, die vorher die Angestellten des Kaufhauses erledigen mussten. Eine weitere Computeranwendung war die Vorbereitung des Flugs und die Steuerung der Raumschiffe, die Menschen vor nunmehr 30 Jahren zum Mond und wieder heil zurück zur Erde brachten. Und die damaligen Computer waren verglichen mit heutigen Maschinen äußerst primitiv.

Am nächsten Beispiel, der Lohnabrechnung, werden Sie sehen, worin der Unterschied liegt zwischen dem, was ein Mensch tut, und dem, was der Computer tut. Schauen Sie sich zuerst die Arbeit eines Lohnbuchhalters an. Er stellt zuerst anhand der Stundenabrechnungen und Stempelkarten fest, wie viele Stunden der Arbeitnehmer gearbeitet hat. Die Stunden werden addiert um die Gesamtarbeitszeit zu erhalten. Diese Zeit wird mit dem Stundenlohn multipliziert, und dann werden Steuern und Sozialabgaben abgezogen. Dabei muss der Buchhalter einige Fakten aus der Personalkartei heraussuchen - Kinderzahl, Familienstand, Steuerklasse, Versicherungsnummer etc. Ist die Berechnung beendet, werden Lohnstreifen und Überweisungsformulare ausgefüllt und an die richtigen Stellen weitergeleitet.

Der Buchhalter erhält also gewisse Informationen, nämlich Arbeitszeit, Steuerklasse, Stundenlohn usw.; er verarbeitet dann diese Informationen nach einem bestimmten Schema und gibt schließlich neue Informationen, z. B. den Betrag des Nettolohns an andere weiter. Der Informationsfluss lässt sich also folgendermaßen skizzieren:

Eingabe - Verarbeitung - Ausgabe

Genauso arbeitet auch der Computer; er erhält aus einer Eingabestation die Daten über die Anwesenheitszeiten des Arbeitnehmers. Bereits das Addieren der einzelnen Zeiten wird "vom Programm" übernommen, das dazu ebenfalls eingegeben werden musste. Die eigentliche Verarbeitung erfolgt in der Zentraleinheit des Computers. Sie besteht im Wesentlichen aus drei Teilen, dem Arbeitsspeicher (dem Gedächtnis des Buchhalters entsprechend), das Rechenwerk und das Steuerwerk (am besten, wenn auch stark vereinfacht, den Koordinierungsfähigkeiten des Buchhalters vergleichbar). Die Personalkartei wird nicht im Arbeitsspeicher untergebracht, da dieser dafür zu klein ist (der Buchhalter kann ja auch nicht alle Kontonummern im Kopf behalten). Für die Personalkartei stehen externe Speicher zur Verfügung, meist Magnetplatten. Wenn das Rechenwerk mit seiner Arbeit fertig ist, wird das Ergebnis dann vom Drucker gleich an die richtigen Stellen in die entsprechenden Formulare eingetragen.

Im letzten Absatz wurden einige Begriffe erwähnt, so z. B. Eingabestation, Zentraleinheit, Arbeitsspeicher, Steuerwerk. Die Erklärungen dieser Begriffe sollen Sie nun erhalten.

Computer-Systeme

Computer sind Systeme, die aus Hardware und Software bestehen. Jede der beiden Komponenten ist ohne die andere nichts wert. Die von Ihnen geschriebenen Programme nehmen, damit sie ablaufen, sowohl die Hardware als auch die Software des Computers in Anspruch. Die Hardwarekomponenten umfassen alle "anfassbaren" Komponenten. Dazu zählt der Computer selbst und alle Geräte, die daran angeschlossen sind: Bildschirm, Drucker, Diskettenlaufwerk usw. Die Software ist "nicht anfassbar". Sie benötigt zum Transport, zur Speicherung und zum Funktionieren immer irgendwelche Hardware. Software lässt sich grob in das sog. Betriebsystem und die sog. Anwendersoftware unterteilen.

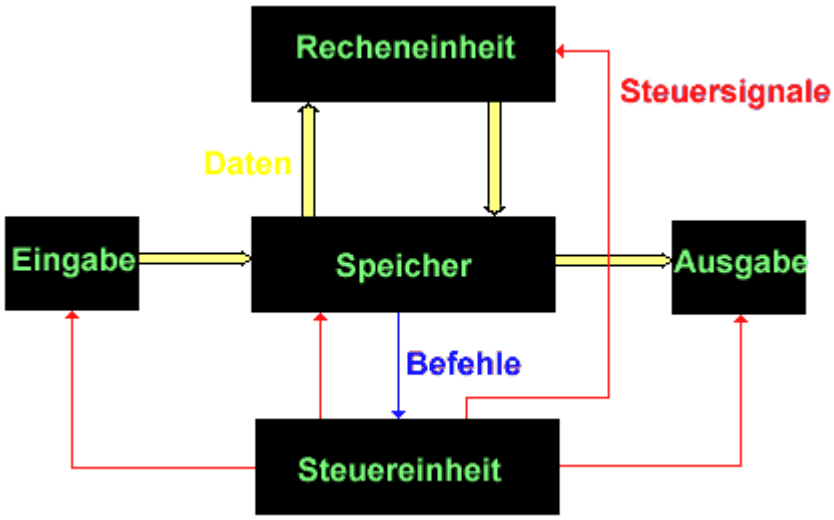
Trotz allem kann man die Software nicht von der Hardware trennen, da selbst die kleinsten Aufgaben beide in Anspruch nehmen. Man braucht Hardware, um Daten eingeben zu können, aber zugleich auch die Software, die den Datenaustausch zwischen Terminal und Computer steuert. Man braucht Hardware, um Daten ausdrucken zu können, aber auch die Software, die ein Zeichen nach dem anderen vom Computer zum Drucker überträgt. Man braucht Hardware, um mit Zahlen rechnen zu können,

aber auch die Software, die Zahlen für die Berechnung aufbereitet. Die beiden Komponenten eines Computersystems sind nicht voneinander zu trennen, trotzdem sollen sie nun nacheinander betrachtet werden.

Hardware

In der zuerst von J. v. Neumann 1947 angegebenen Modellvorstellung eines Rechners besteht dieser aus vier Funktionseinheiten:

- die Recheneinheit,
- die Steuereinheit,
- den Speicher (Memory) und
- die Ein- und Ausgabegeräte (E/A-Geräte engl. Input/Output Units oder I/O).



Recheneinheit und Steuereinheit werden oft als Zentraleinheit oder Central Processing Unit (CPU) zusammengefasst. Die CPU ist das "Herz" eines Computers. Sie bearbeitet Programme, führt Rechnungen aus und steuert die Vorgänge in den anderen Teilen.
Der Speicher nimmt alle Daten auf, die der Computer braucht. Das geht von Daten, die der nächste Programmschritt in einer Mikrosekunde benötigt, bis hin zu ganz selten gebrauchten Daten.
Schließlich stellen die E/A-Geräte die Verbindung her zwischen dem Computer und seinem Anwender.

Zentraleinheit

Die CPU besteht aus mehreren Teilen, die eng zusammenarbeiten: Recheneinheit und Steuereinheit. Sie enthält mehrere Register für die Aufnahme der Daten, die bei der Ausführung eines Programms verfügbar sein müssen. So nimmt z.B. das Befehlsregister den jeweils auszuführenden Befehl auf. Ein Register kann man sich als schnellen (Zwischen-) Speicher für ein einzelnes Datenwort vorstellen. Nur der anstehende Befehl steht im Befehlsregister, daher gibt es - von der Ausführung her gesehen - keinen Unterschied zwischen langen und kurzen Programmen oder zwischen schwierigen und einfachen. Die Befehle enthalten die Informationen über die Speicherplätze der benötigten Daten; bei der Ausführung des Befehls werden die Daten bereitgestellt und nach der Bearbeitung wieder abgelegt.

Recheneinheit

Die ALU (Arithmetic and Logic Unit) führt alle Berechnungen aus und trifft logische Entscheidungen. Ihre Hauptaufgabe ist es, Vergleiche zwischen Werten vorzunehmen und damit Bedingungen zu überprüfen. Die Rechenfähigkeit beschränkt sich auf die Addition, doch lassen sich daraus alle anderen arithmetischen Operationen aufbauen. Die komplizierten Rechnungen, die der Computer in kurzer Zeit ausführen kann, setzen sich aus einer Vielzahl kleinster Schritte in der ALU zusammen.

Steuereinheit

Die Register der Steuereinheit und die Recheneinheit arbeiten Hand in Hand, die Register der Steuereinheit liefern die Signale zur Steuerung der gewünschten ALU-Funktion. Die Verbindung zwischen ihnen (und von ihnen zu den übrigen Einheiten) wird durch die Steuereinheit hergestellt. Man kann die Steuereinheit als Schaltzentrale des Computers ansehen, die alle Datenströme lenkt.

Speicher

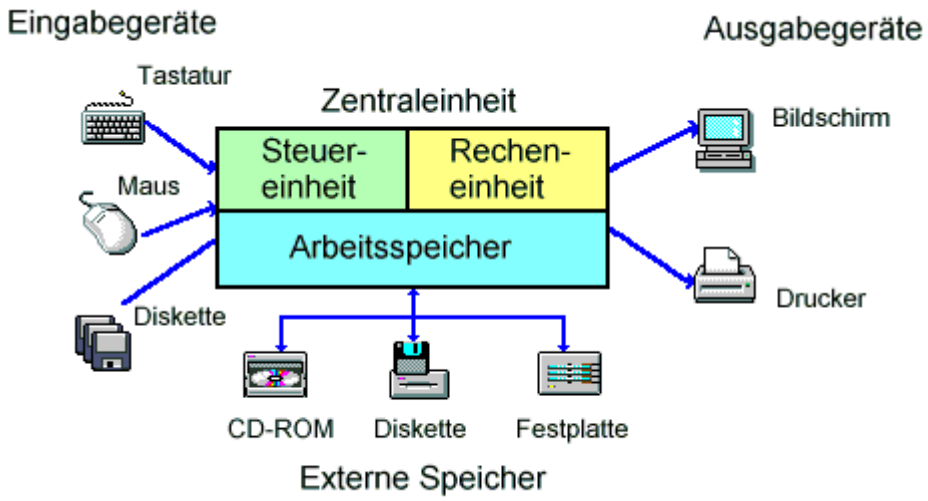
Der Speicher nimmt alle Daten auf, dazu gehören die Programme ebenso wie die von ihnen zu verarbeitenden Werte. Bei der Verarbeitung werden Zwischenergebnisse im Speicher abgelegt und schließlich die Endergebnisse. Man unterscheidet beim Speicher zwischen dem Hauptspeicher (oder Arbeitsspeicher) und dem externen Speicher. Der Hauptspeicher ist fest im Computer eingebaut, man kann ihn meist durch zusätzliche Speicherkarten erweitern. Im Hauptspeicher befindet sich das Programm, das Sie schreiben oder verändern wollen oder das der Computer bearbeiten soll. Auch die Daten, die beim Ablauf des Programms benötigt werden, werden dort abgelegt. Auf den Hauptspeicher kann die CPU direkt zugreifen, sie kann von dort neue Programmbefehle holen und kann Daten laden oder abspeichern.

Externe Speicher (Massenspeicher)

Im externen Speicher werden die Programme und Werte aufbewahrt, die man zurzeit nicht benötigt. Als Datenträger werden Festplatten und Disketten (Floppy- Disks) verwendet. Die Diskette wird ins Diskettenlaufwerk eingelegt und kann dann mit Daten "beschrieben" werden. Für das Abspeichern von Programmen vom Hauptspeicher auf die Festplatte oder Diskette gibt es besondere Kommandos des Betriebssystems, ebenso für das Laden von einem Massenspeicher in den Hauptspeicher.

Ein-/Ausgabe-Geräte

Die Eingabe- und Ausgabegeräte stellen die Verbindung her zwischen einem Computer und seinem Benutzer, aber auch zwischen einem Computer und anderen Geräten oder anderen Computern. Ohne Geräte zur Ein- und Ausgabe könnte man keine Programme eingeben und keine Daten an ein laufendes Programm liefern, man erhielte keine errechneten Ergebnisse. Ein Ausgabegerät wie der Drucker lässt sich einsetzen, um die Ergebnisse der Programmbearbeitung zu erhalten oder um die Werte auszugeben, die in einem Speicher abgelegt sind. Ein Eingabegerät wird komplementär eingesetzt, mit ihm versorgt man die CPU oder den Hauptspeicher durch die CPU mit neuen Daten.



Vernetzung

Zu den E/A-Geräten gehören auch die Netzwerk-Verbindungen, mit denen sich mehrere Computer zu einem Verbund zusammenschließen lassen. Sie erlauben einen sehr schnellen Datenaustausch zwischen verschiedenen Computersystemen. Ein Vorteil solcher Vernetzung liegt darin, dass mehrere Systeme gemeinsam auf periphere Geräte wie Drucker oder Festplatte zugreifen können. Damit erweitern sich die Fähigkeiten der einzelnen Maschinen ohne entsprechende zusätzliche Kosten.

Sichtweisen eines Rechners

Der Benutzer eines Rechners sieht von der Maschine je nach seinem Status einen mehr oder weniger eingeschränkten Teil.

- Der Anwender sieht den Rechner als das ihm zugängliche Anwendersystem, das genau auf seine Bedürfnisse zugeschnitten ist und mit dem er in einer für ihn bequemen Sprache Daten manipulieren kann.
- Der Anwendungsprogrammierer sieht den Rechner als das Betriebssystem, das ihm Compiler für verschiedene höhere Programmiersprachen zur Verfügung stellt und für ihn Dienstleistungen wie Editor und Dateiverwaltung bereithält. Wie das Betriebssystem realisiert ist, braucht auf dieser Stufe den Anwendungsprogrammierer nicht zu interessieren. Für ihn ist "der Rechner" das Betriebssystem mit den Compilern.
- Der Systemprogrammierer sieht vom Rechner schon sehr viel mehr. Er kann alle Möglichkeiten der Hardware ausnützen und Programme als Befehlsfolgen für das Leitwerk schreiben. Er schreibt diese Befehle aber i.A. in einer für ihn lesbaren Form (Assemblersprache, siehe später), die noch durch ein spezielles Programm, den Assembler, in eine Folge von 0 und 1 gewandelt werden muss, die allein das Leitwerk des Rechners interpretieren kann. Für ihn ist Rechner und Assembler fast synonym.
- Der Rechneringenieur beim Rechnerhersteller sieht die Hardware des Rechners in allen Einzelheiten und kennt den genauen Aufbau der Befehle, die sich z.T. aus Folgen einfachster Steuerkommandos (Mikroprogramm) zusammensetzen. Hier ist der interne Aufbau des Rechners von Interesse, und die technische Realisierung von Rechenwerk, Speicher und Leitwerk tritt in den Vordergrund.

Software

Damit kennen Sie nun in etwa den Hardwareaufbau eines Computers. Es stellt sich jetzt die Frage nach seinen Einsatzgebieten und nach der Herkunft seiner Fähigkeiten. Seine Universalität erklärt sich aus den sehr elementaren Maschinenbefehlen, der Rechnerarchitektur und der Tatsache, dass der Computer, bevor er überhaupt etwas tun kann, erst in einem Programm (Software) erklärt bekommen muss, was er zu tun hat. Es gibt eine riesige Vielfalt an Programmen, geschrieben für die unterschiedlichsten Aufgaben, die aber auf ein und derselben unveränderten Hardware lauffähig sind.

Die Software eines Computersystems kann grob in zwei Teile gegliedert werden: die Anwendersoftware und das Betriebssystem. Ein Anwenderprogramm ist für die Lösung einer ganz bestimmten Aufgabe geschrieben worden, der Anwender setzt es dann (und nur dann) ein, wenn er eine solche Aufgabe lösen will.

Dagegen ist das Betriebssystem eine anwendungsneutrale Software. Was die Infrastruktur einer Stadt mit Ihren Straßen, Buslinien, Trambahnen, Telefonnetz, Fernverkehrsanbindung für den Einzelnen und die Geschäftswelt ist, so ist das Betriebssystem die komfortable standardisierte Umgebung für die Anwenderprogramme. Möchte man in der Stadt von einem Ort zum anderen Waren transportieren, so benutzt man mit dem Auto die bestehenden Straßen und baut keine. Ähnlich ist es mit den Anwendungsprogrammen. Sie sind für eine spezielle Aufgabe geschaffen worden. Der Ersteller möchte sich aber nicht unbedingt mit der Infrastruktur, der Hardware, auseinandersetzten.

Das Betriebssystem wird ständig benötigt, um die Vorgänge in der Hardware zu steuern und um die Wechselwirkung zwischen dem Computer und seiner Anwendungsprogramme zu koordinieren.

Anwenderprogramme

Wenn man allgemein von Software spricht, meint man meist Anwenderprogramme, also Programme, die eine spezielle Aufgabe lösen. Einige typische Anwenderprogramme sind

- Programme zur Textverarbeitung,
- Spielprogramme,
- Programme für Tabellenkalkulation,
- Lernprogramme und
- Programme für die Verwaltung von Dateien,
- Programme für die Verwaltung großer Datenmengen (Datenbanken),
- Programme zum Steuern von Maschinen,
- Programme zum Regeln industrieller Prozesse,
- Programmiersprachen-Compiler
- und vieles mehr.

Wenn Sie in eine Computerzeitschrift schauen, finden Sie viele weitere Beispiele für Anwenderprogramme. Und auch die Programme dieses Skriptums sind zu den Anwenderprogrammen zu zählen, jedes wird für die Lösung eines bestimmten Problems geschrieben.

Betriebssystem

Die "nackte" Hardware eines Computers hat nicht die Fähigkeiten, ein Anwenderprogramm zu bearbeiten. Auch wenn die CPU die Vorgänge in der Hardware auf einer niedrigen Ebene koordinieren kann, es muss eine Verbindung zwischen dem Anwenderprogramm und der jeweiligen Hardware hergestellt werden. Die Bearbeitung des Programms erfordert den Einsatz verschiedener Eingabe- und Ausgabegeräte. Es ist zunächst extern gespeichert und muss in den Hauptspeicher geladen werden. Diese und viele weitere Abläufe werden vorn Betriebssystem gesteuert. Es ist ein sehr umfangreiches Programm, das die Vorgänge in der Hardware steuert und koordiniert. Der Benutzer braucht sich nicht selbst darum zu kümmern, wie die einzelnen Schritte bei der Bearbeitung seines Programms intern ablaufen.

Wenn ein Computer nur ein einziges Programm zu bearbeiten hätte wie z.B. der Mikroprozessor in einer Waschmaschine, dann bräuchte er kein Betriebssystem. Die Aufgabe eines Betriebssystems ist es, eine Umgebung zu schaffen, in welcher verschiedene Anwenderprogramme ablaufen können. Es bietet verschiedene Möglichkeiten an, den Computer einzusetzen. Da das Betriebssystem unerlässlich ist, wird es vom Hersteller mitgeliefert und lässt sich kaum verändern.

Aufgaben des Betriebssystems

Von den vielen Aufgaben des Betriebssystems sollten Sie einige kennen. Zum Betriebssystem gehört z. B. ein Programm, das den Zugriff auf externe Speicher steuert und die gespeicherten Dateien verwaltet. Bei größeren Systemen und Netzwerken kontrolliert das Betriebssystem z.B. den Zugang zum Computer mit der Eingabe eines Passwortes und organisiert die gleichzeitige Arbeit, mehrerer Benutzer. Das Betriebssystem ist verantwortlich für das Speichern und Laden von Programmen und für die Zuteilung der benötigten Hilfsmittel. Aus der Sicht des Programmierers ist besonders wichtig, dass vom Betriebssystem her die Programmierumgebung geschaffen wird. Es liefert die Umgebung, in der Sie Programme schreiben können, und es steuert den Ablauf Ihrer Programme. Sie brauchen sich nicht um die Einzelheiten zu kümmern, das Betriebssystem nimmt Ihnen (fast) alles ab. Die Entwicklung immer leistungsfähigerer Betriebssysteme hat es ermöglicht, die Programmier-Arbeit immer komfortabler zu machen.

Fazit

Der Computer kann eine einmal programmierte Aufgabe beliebig oft ausführen. Er ist also für Routineaufgaben, aber auch für die Verarbeitung so riesiger Datenmengen geeignet, die der Mensch nicht mehr bewältigen könnte. Wenn es also darauf ankommt, ein Rechenergebnis unmittelbar nach Eintreffen der Daten zu erhalten, wird man einen Computer bemühen - oder, wenn Sie wissen wollen, ob die Zahl 94084005495481 eine Primzahl ist. Um dies nachzuprüfen, bräuchte man sehr lange; vom Com-

puter erfährt man schon nach Sekunden, dass es sich um keine Primzahl, sondern um das Quadrat der Primzahl 9699691 handelt. Die Datenverarbeitung umfasst also:

- Daten-Empfang ---------- lesen, eingeben
- Daten-Speicherung ---------- speichern
- Daten-Bearbeitung + /--------- umformen
  /-- nach der Form +
  / \--------- rechnen
  \
  \-- nach dem Inhalt --------- ordnen
  (sortieren, vergleichen,
  selektieren, mischen)
- Daten-Ausgabe ---------- schreiben, ausgeben
- Daten-Übertragung ---------- transportieren

3 Softwareerstellung (Programmierung)

Kennen Sie die Geschichte des Herrn X., der sich in eine astronomische Vorlesung verirrt hatte. Obwohl die dargestellten Gedankengänge fremd und neu für ihn waren, meinte er doch verstehen zu können, wie die Astronomen ihre Teleskope einsetzten, um die Entfernung der Gestirne von der Erde zu ermitteln. Es erschien ihm auch verständlich, dass sie die relative Positionen der Sterne und ihre Bewegungen voraussagen konnten. Was er aber gar nicht begreifen konnte: Wie zum Teufel konnten die Astronomen die Namen der Sterne und der Sternbilder herausbekommen?

Manche Leute haben das gleiche Problem bei den Programmiersprachen: Sie halten eine Sprache für ein kompliziertes mathematisches Begriffssystem, das von den ersten Informatikern mit viel Glück entdeckt worden ist. Es handelt sich aber nicht um einen Code, den es zu knacken galt, sondern um eine künstliche Sprache, die von Menschen aus ersten bescheidenen Anfängen heraus immer weiter entwickelt und verbessert worden ist.

Was sind Algorithmen?

Sie haben mit Sicherheit im Augenblick, in dem Sie diese Zeilen lesen, eine ganze Reihe von Problemen. Diese können Sie in verschiedene Kategorien einteilen, z. B.

- berufliche und private Probleme
- kurzfristige und langfristige Probleme
- bedeutende und banale Probleme
- usw. ...

Sie können sie aber auch unter einem Aspekt sehen, der uns ganz besonders interessiert, nämlich mit Computern lösbare bzw. nicht lösbare Probleme. Hier interessieren wir uns für die erste der beiden Gruppen. Eine klare Trennungslinie zwischen beiden lässt sich im Allgemeinen nicht ziehen, denn Probleme, die gestern noch als unlösbar galten, werden morgen von Computern vielleicht gelöst. Es gilt aber ein Grundsatz:

Jedes Problem, dessen Lösung durch einen Algorithmus beschrieben werden kann, ist im Prinzip durch einen Computer lösbar.

Aus dieser Aussage kann man zwei Schlüsse ziehen:

1. Ein Computer ist ein Werkzeug, welches Ihnen bei der Lösung gewisser Probleme hilft.
2. Ein Algorithmus ist eine Art Anleitung oder Vorschrift, wie man zu einem Problem eine Lösung findet.

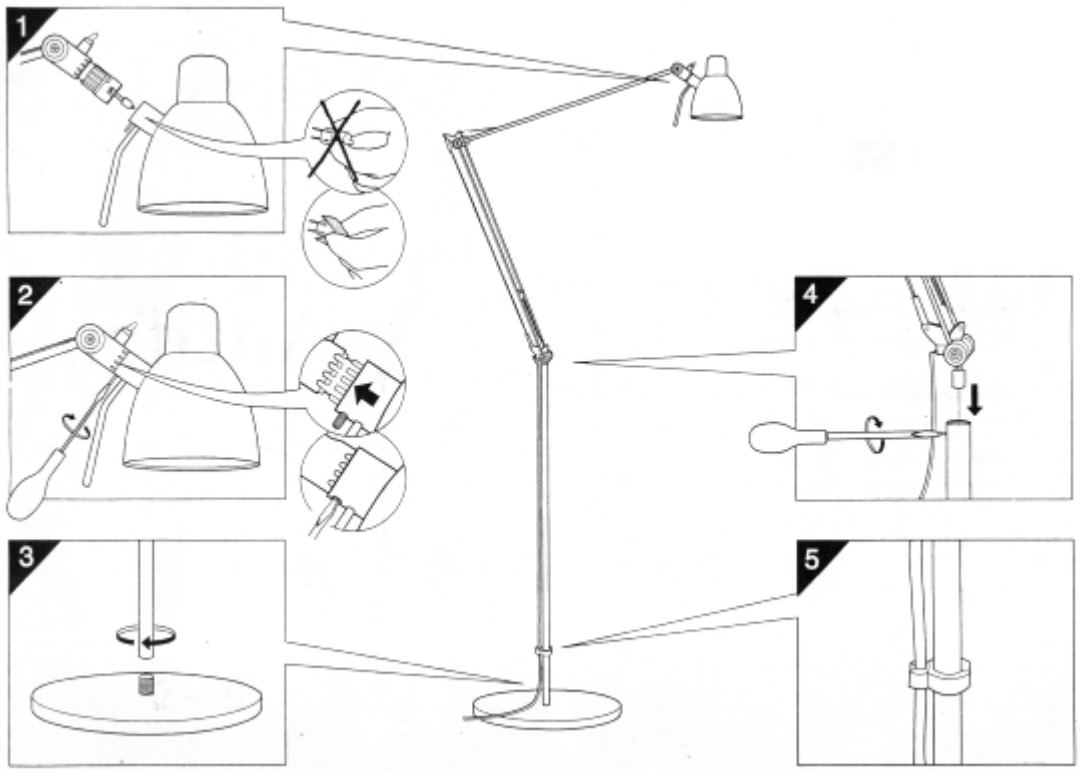
Das Wort "Algorithmus" wird im allgemeinen Sprachgebrauch kaum verwendet und ist Ihnen daher vermutlich auch nicht geläufig. Es geht zurück auf den arabischen Autor Al-Khowarizmi (ca. 825 n. Chr.), der ein Lehrbuch über Mathematik geschrieben hat. In der Informatik versteht man darunter eine Lösungsvorschrift. Wir wollen den Begriff "Algorithmus" folgendermaßen definieren:

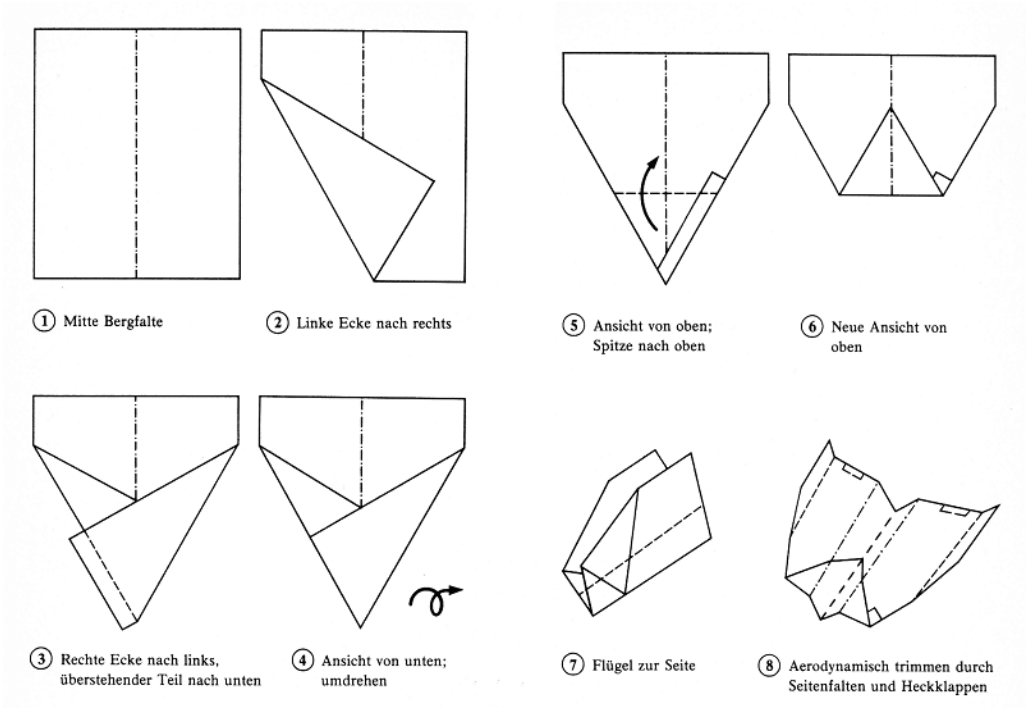
Ein Algorithmus ist eine eindeutige Beschreibung eines endlichen Verfahrens zur Lösung einer bestimmten Klasse von Problemen.

Zu viel auf einmal? Wir werden im nächsten Abschnitt auf die einzelnen Eigenschaften eines Algorithmus noch genauer eingehen. Zunächst wollen wir den Begriff des Algorithmus jedoch mit "Leben" füllen, damit Sie ganz konkrete Vorstellungen damit verbinden können.

Alltagsalgorithmen

Die Alltagswelt, die Sie umgibt, ist voller Algorithmen - Sie haben es bisher nur nicht gewusst. Wenn Sie z. B. eine Coladose öffnen wollen, führen Sie mit dem Objekt "Coladose" eine Folge koordinierter Bewegungsabläufe aus, die schließlich die gewünschte Lösung produzieren. Die Lösungsvorschrift hierzu haben Sie sich durch Lernen und Üben erworben und in Ihrem Gehirn abgespeichert. Sie tragen also den Algorithmus "Coladose öffnen" zusammen mit unzähligen anderen Algorithmen in Ihrem Kopf herum. Eine ganze Reihe von Alltagsalgorithmen ist schriftlich formuliert, z.B. Bedienungsanleitungen oder Kochrezepte, aber auch Wegbeschreibungen in einem Wanderführer oder Gesetze. Andere Algorithmen sind in Form eines Bildes formuliert, z.B. die Aufbauanleitung für eine Lampe oder die Anleitung zum Falten eines Papierfliegers:





Die Algorithmen haben alle einige Gemeinsamkeiten. Sie bestehen aus kurzen und knappen Formulierungen, welche dem "Ausführenden" gewisse Anweisungen geben, die zur Lösung seines Problems führen. Je komplizierter ein Problem ist, umso mehr Anweisungen wird man in der Regel benötigen. Da als Adressat dieser Anweisungen ein mit Vernunft begabter Mensch vorausgesetzt wird, ist es hinreichend, wenn man die Anweisungen in einem kurzen deutschen Satz formuliert. Verständnisschwierigkeiten bei deren Interpretation treten im Allgemeinen nicht auf. Der Mensch verfügt auch über ein gewisses Repertoire an Hilfsmaßnahmen, wenn unvorhergesehene Störungen auftreten. Dazu ein Beispiel:

Jeder von uns kann in einer Telefonzelle erfolgreich telefonieren. Nehmen wir an, wir hätten einen Gast von einem fernen Planeten, der perfekt deutsch spricht und liest, der aber noch niemals eine Telefonzelle benutzt hat (vermutlich, weil seinesgleichen per Telepathie kommuniziert). Er soll nun von uns auf einem Blatt Papier eine exakte Handlungsanweisung für die Benutzung dieser Telefonzelle erhalten, damit er uns nach dem Shopping anrufen kann. Es wird sich also um ein Ortsgespräch handeln, und wir unterstellen, dass unser Gast einige Groschen in der Tasche hat und unsere Telefonnummer kennt. Außerdem setzen wir einige Randbedingungen als bekannt voraus, obwohl gerade diese in der Praxis oft zu großen Realisierungsproblemen führen, hier z. B. die Frage: "Was ist der Hörer und wie dann wir herum wird er gehalten?".

Von Personen, die sich bisher mit der algorithmischen Lösung solcher Alltagsprobleme nicht beschäftigt haben, wird meist folgende Handlungsanweisung genannt:

1. Nimm Hörer ab;
2. Wirf 20 Cent ein;
3. Wähle ...

Bereits zwischen der ersten und zweiten Anweisung wurde übersehen, dass niemand von uns eine Münze einwirft, wenn er in dem zum Ohr geführten Hörer keinen Dauerton hört. Und schließlich würden wir nicht wählen, wenn eine der eingeworfenen Münzen durchgefallen wäre. Auch der jetzt nahe liegende Schritt, dass man diese Münze dem Rückgabefach entnimmt und es mit ihr nochmals probiert, muss nach wenigen gleichartigen Versuchen ersetzt werden durch das Austauschen der offensichtlich ungeeigneten Münze. Denn ein derart "falsch" programmierter Computer würde die gleiche durchgefallene Münze immer wieder erneut einwerfen, da er entgegen dem Menschen diese Handlungsweise niemals als unsinnig erkennen würde.

Tatsächlich lautet die korrekte Handlungsanweisung, nach der wir alle in der Praxis vorgehen:

1. Nimm Hörer ab;
2. Falls Dauerton vorhanden, dann wirf zwei Cent ein, sonst hänge Hörer ein und verlasse Telefonzelle;
3. Falls eine Münze durchfällt, dann prüfe, ob diese Münze schon einmal durchgefallen ist. Falls ja, tausche sie aus, sonst werfe sie erneut ein;
4. Wähle ...

Die Lesbarkeit dieser Handlungsanweisung lässt sich durch eine geeignete graphische Strukturierung verbessern:

1. Nimm Hörer ab;
2. Falls Dauerton vorhanden,

   dann wirf zwei Cent ein,
   sonst hänge Hörer ein und verlasse Telefonzelle;

3. Falls ein Cent durchfällt,

   falls die Münze schon einmal durchgefallen ist,

   dann wechsle sie aus,
   sonst werfe sie erneut ein,

   sonst wähle;

4. ...

Es ist offensichtlich, dass bei alltäglichen Handlungen, also auch bei berufsbezogenen Abläufen, ein großer Unterschied zwischen "tun können" und "exakt beschreiben können" besteht. Soll der Algorithmus jedoch von einer Maschine (einem Computer) ausgeführt werden, dann müssen diese Anweisungen viel präziser formuliert und alle möglichen Sonderfälle berücksichtigt werden. Auch wenn man selbst nicht programmiert, so ist diese Art zu denken eine wichtige Grundlage für eine sinnvolle Anwendung von Computern.

Eine weitere Gemeinsamkeit der aufgeführten Alltagsalgorithmen besteht darin, dass sie Anweisungen zur Manipulation von Objekten geben. Der Algorithmus oben beschreibt beispielsweise, wie die Objekte Hörer, Münze, Tastenfeld usw. zu manipulieren sind. Besonders deutlich tritt diese Eigenschaft bei Kochrezepten zum Vorschein. Als Beispiel betrachten wir die Zubereitung einer Gulaschsuppe.

Algorithmus "Gulaschsuppe zubereiten"

Zu manipulierende Objekte:
350 g Rindfleisch, 3 Zwiebeln, 50 g Schweineschmalz, 15 g Mehl, 1 kleine Dose Tomatenmark, 3/4 1 Wasser, Salz, Paprika, Majoran.

Hilfsobjekte:
Herd, Kochgeschirr

Anweisungen:

1. Fleisch und Zwiebeln würfeln
2. in Schmalz andünsten
3. mit Mehl bestäuben und kurz anrösten
4. Tomatenmark zugeben
5. mit Wasser auffüllen

6. garen
7. mit Salz, Paprika und Majoran abschmecken.

An diesem Beispiel wird deutlich, dass ein Algorithmus aus zwei Teilen besteht:

- einem Deklarationsteil, in dem die zu manipulierenden Objekte deklariert werden
- einem Aktionsteil, in dem die auszuführenden Aktionen in Form von Anweisungen beschrieben werden.

Eigenschaften von Algorithmen

In der Definition des Begriffs "Algorithmus" im vorausgegangenen Abschnitt kommen einige Adjektive vor, welche die Eigenschaften von Algorithmen beschreiben. Wir wollen Sie der Reihe nach betrachten.

- **Eindeutige Beschreibung**
  Bei der Verwendung einer natürlichen Sprache wie Deutsch, Englisch, Französisch usw. lässt es sich nicht vermeiden, dass gewisse Wörter und Sätze eine Mehrdeutigkeit beinhalten, d. h. dass man sie verschieden interpretieren kann. Denken Sie z. B. an folgende Wörter:

  - Stollen: Weihnachtsgebäck oder Gang im Bergwerk oder Teil eines Fußballschuhes.
  - Bund: Zusammenschluss oder Teil einer Hose.
  - Mutter: Weibliche Person mit Kind oder Teil einer Schraube.

  In einem Algorithmus dürfen solche mehrdeutigen Formulierungen nicht vorkommen. Der "Ausführende" - ganz gleich ob Mensch oder Maschine - darf nie im Zweifel darüber sein, wie eine bestimmte Anweisung zu interpretieren ist.

- **Endliches Verfahren**
  Ein Algorithmus muss ein endliches Verfahren beschreiben (das Gegenteil wäre ein unendliches). Man meint damit, dass das durch den Algorithmus beschriebene Verfahren nach einer endlichen Zahl von Schritten eine Lösung produziert. Die genaue Anzahl hängt natürlich davon ab, wer den Algorithmus ausführt. Man sagt auch: "Das Verfahren muss terminieren", also zu einem Abschluss kommen. Mit der Endlichkeit hängt ein weiterer Begriff zusammen: die Effizienz. Man sucht zur Lösung eines Problems einen möglichst effizienten Algorithmus (eigentlich müsste man genauer sagen: einen Algorithmus, der ein möglichst effizientes Verfahren beschreibt). Auf Computer bezogen bedeutet das zweierlei:

  1. Die Ausführungszeit soll möglichst kurz sein.
  2. Der Speicherplatzbedarf soll möglichst gering sein.

- **Bestimmte Problemklasse**
  Ein Algorithmus soll ein Lösungsverfahren für eine ganze Klasse von Problemen beschreiben und nicht nur für ein isoliertes Einzelproblem. Man sagt auch: "Das Verfahren soll allgemeingültig sein." Wir wollen uns diese Eigenschaft an einem ganz einfachen Beispiel klar machen. Nehmen wir einmal an, Sie sollen die Zahlen 7,12 und 16 addieren. Sie könnten das Lösungsverfahren folgendermaßen beschreiben:

  0. Addiere 7 und 12, nenne das Ergebnis SUMME1.
  1. Addiere 16 zu SUMME1, nenne das Ergebnis SUMME2.
  2. Schreibe als Lösung SUMME2 auf.

  Diese Lösungsvorschrift ist nach unserer Definition kein Algorithmus, da sie lediglich für ein ganz spezielles Teilproblem die Lösung liefert. Soll man beispielsweise die Zahlen 7, 12 und 17 addieren, dann kann man diesen Algorithmus nicht mehr verwenden. Ein erster Schritt zur Verallgemeinerung besteht darin, statt der Konstanten 7, 12 und 16 drei Variablen einzuführen, z. B. ZAHL1, ZAHL2 und ZAHL3. Das dadurch beschriebene Verfahren gilt dann für die Addition dreier beliebiger Zahlen. Das Problem, die Summe aus drei Zahlen zu berechnen, gehört aber zu einer allgemeineren Problemklasse, nämlich die Summe aus N Zahlen zu berechnen. Dabei ist N eine natürliche Zahl größer als 1. Unser spezielles Problem, die Zahlen

7, 12 und 16 zu addieren, ist also nur ein Spezialfall dieses allgemeinen Problems für N = 3. Die Lösungsvorschrift für dieses Problem ist dann ein Algorithmus im Sinn unserer Definition. Durch die Forderung nach Allgemeinheit erreicht man also, dass man wenige mächtige Algorithmen erhält und sich nicht in einer Vielzahl von Lösungsvorschriften für Sonderfälle verzettelt.

Weitere Punkte, die auch oft als Eigenschaften von Algorithmen gefordert werden, sich aber aus den obigen Eigenschaften ergeben, sind:

3. Vollständigkeit der Beschreibung
   Es muss eine komplette Anweisungsfolge vorliegen. Eine Bezugnahme auf unbekannte Information darf nicht enthalten sein.
4. Wiederholbarkeit des Algorithmus
   Im Sinne eines physikalischen Experiments muss man die Ausführungen eines Algorithmus beliebig oft wiederholen können und jede Wiederholung muss bei gleichen Eingabedaten das gleiche Resultat liefern (Reproduzierbarkeit).
   Ist ein Algorithmus endlich und definit so ergibt sich diese Forderung automatisch.
5. Korrektheit des Algorithmus
   Diese Forderung ist zwar selbstverständlich, aber in der Praxis ist die Korrektheit nur sehr schwer nachzuweisen. Man bedient sich daher Tests, bei denen für die vorgegebenen Eingabedaten das Ergebnis bereits bekannt ist, und vergleicht dann die erzeugten Ausgabedaten mit den Ergebnissen. (Ein solcher Test ist insofern problematisch, da alle möglichen Fälle abgedeckt werden müssen. Im Extremfall muss dieser Test sogar für jede mögliche Eingabe durchgeführt werden.) Der Begriff der Korrektheit nimmt zwar in der Literatur einen sehr großen Raum ein, da es nicht trivial ist, die Korrektheit nachzuweisen, er ist jedoch für die Definition eines Algorithmusbegriffes nicht erforderlich.

Neben diesen Eigenschaften gibt es noch weitere Eigenschaften von Algorithmen, die sich auf die Art und Weise der Ausführung des Algorithmus beziehen. Hier sind zu nennen:

- Die *Effizienz* der Beschreibung und der Ausführung (umständlich oder einfach).
- Die Art der Ausführung einzelner Anweisungen (sequentiell oder parallel).
- Die *Komplexität* bei der Ausführung. (Sie ist ein Maß für den Aufwand bei der Durchführung eines Algorithmus).

Zusammenfassen lassen sich alle diese Eigenschaften in einer kurzen aber präzisen Definition des Begriffs **Algorithmus**.

> Eine Bearbeitungsvorschrift heißt **Algorithmus**, wenn sie folgende Eigenschaften erfüllt:
> Die Vorschrift ist mit endlichen Mitteln beschreibbar.
> Sie liefert auf eine eindeutig festgelegte Weise zu einer vorgegebenen Eingabe in endlich vielen Schritten genau eine Ausgabe.

Da man nicht alle Bearbeitungsvorschriften durch eindeutige Folgen von Anweisungen als Algorithmus darstellen kann, wurde im Laufe der Zeit der Begriff ausgedehnt. Man unterscheidet heute zwischen **deterministischen** und **nicht-deterministischen** Algorithmen. Enthält ein Algorithmus elementare Anweisungen, deren Ergebnis durch einen Zufallsmechanismus beeinflusst wird, so heißt dieser Algorithmus **nicht-deterministisch**. Liefert er bei der gleichen Eingabe immer die gleiche Ausgabe, so heißt er **deterministisch**.

Spielen wir nun noch ein Beispiel für eine Verfahrensbeschreibung durch. Beschreibung des Verfahrens "S":

(S1) Wenn keine Striche mehr in der Eingabe vorhanden sind, so höre auf.
(S2) Nimm einen Strich aus der Eingabe weg. Wenn bereits vier Striche ungebündelt in der

Ausgabe liegen, lege einen Querstrich über sie; andernfalls füge einen Strich in der Ausgabe hinzu. Dann gehe nach (S1).

Bei dieser Art der Darstellung eines Verfahrens gilt die Regel: Wenn man alle Handlungen eines Schritts durchgeführt hat und nicht ausdrücklich durch "höre auf" oder "gehe nach" etwas anderes gesagt bekommt, so geht man zum nächsten Schritt.

Sie haben zwar jetzt die Beschreibung eines Verfahrens erhalten, seinen Nutzen erkennen Sie aber erst, wenn Sie wissen, was in der EINGABE vorhanden ist und wie die AUSGABE aussieht. Denken Sie noch mal an den Informationsfluss. Das Kennzeichen der Datenverarbeitung wird Ihnen immer wieder begegnen: EINGABE - VERARBEITUNG - AUSGABE. Die Eingabe des Verfahrens "S" besteht aus einer Anzahl von Strichen, die irgendwie angeordnet sind, z. B. wie hier:

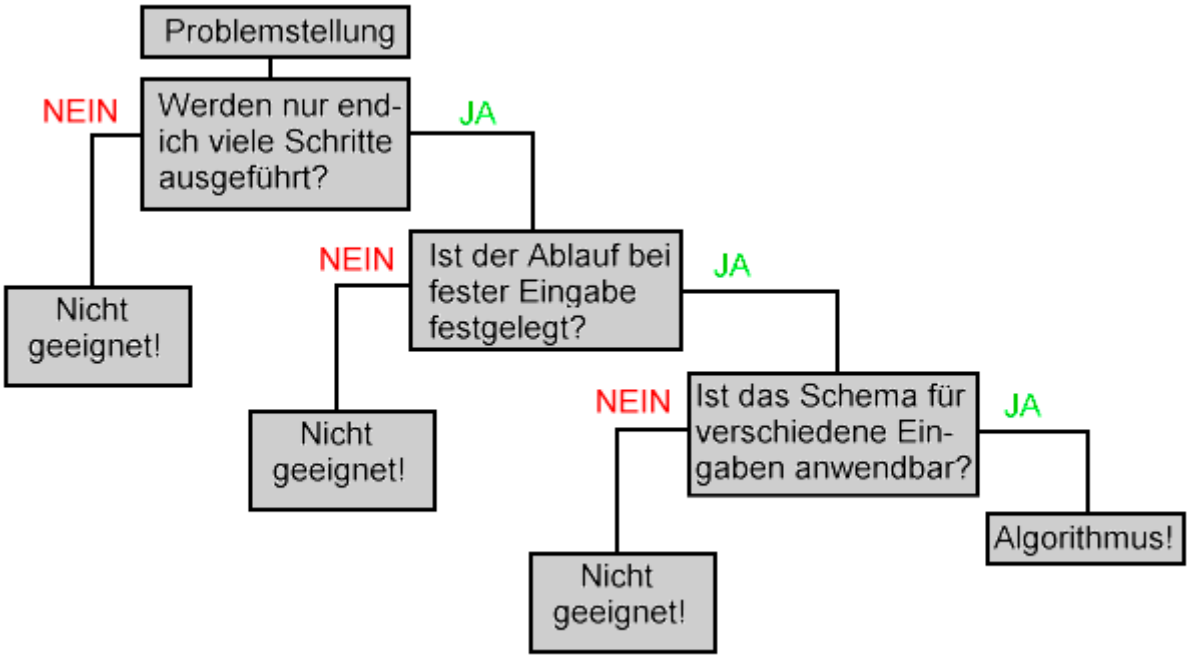
Führen Sie jetzt das Verfahren "S" durch. In (S2) ist eine Bedingung; wir müssen diese prüfen und, wenn sie erfüllt ist, die dahinter stehende Anweisung ausführen. Als Ausgabe erhält man was hier dargestellt ist.

Ausgabe:

Oder auch:

Gewünscht war natürlich das linke Beispiel. Also hat unsere Verfahrensbeschreibung noch etliche Schwachstellen. Sie müsste präzisiert werden, etwa durch: "Füge einen Strich von 8 mm Höhe parallel zu den anderen Strichen und parallel zur langen Kante des Papiers im Abstand von 0,5 mm von dem vorhergehenden auf derselben Grundlinie wie die anderen an. Man könnte der Beschreibung auch noch einen vernünftigen Namen geben: "Verfahren zur Bündelung von Strichen in Fünfergruppen".

Damit ein Verfahren einem Computer mittels einer Programmiersprache erklärt werden kann, muss es den folgenden Bedingungen genügen:



Der Begriff Algorithmus ist ein zentraler Begriff der Informatik. Er baut immer auf einer Menge von Grundoperationen auf.

Was ist Programmieren?

Das Verfahren "S" ist in deutscher Sprache geschrieben, die für ein und dieselbe Handlung oft mehrere Ausdrücke erlaubt. So lässt sich "wegnehmen" auch durch "entfernen" oder "streichen" ersetzen. Auch die Sätze lassen sich auf verschiedene Art konstruieren: "Wenn keine Striche mehr in der Eingabe sind, höre auf", "sind keine Striche mehr in der Eingabe, dann Ende", "Schluss, falls keine Striche mehr in der Eingabe sind", "Solange noch Striche vorhanden sind gehe zu S2, sonst höre auf." .
Da dem Computer alles genau erklärt werden muss, was er tun soll, müsste ihm erklärt werden, dass die obigen Konstruktionen und Worte die gleiche Bedeutung besitzen. Man muss dann aber auch klären, wann die Begriffe nicht das Gleiche bedeuten.

Die Schwierigkeiten, die dabei entstehen, sind der Grund, weshalb man dem Computer nur eine ganz bestimmte Sorte von Sprachen, so genannte normierte Sprachen, anbieten darf. Die Programmiersprachen sind normierte Sprachen, die der Beschreibung von Verarbeitungsvorschriften, Datenstrukturen sowie Ein- und Ausgabe dienen.

Programmieren bedeutet also, einen Computer dazu zu bringen, Algorithmen auszuführen. Dafür müssen wir den Algorithmus so formulieren, dass der Computer ihn versteht - in einer Programmiersprache. Mit dem, was wir bisher wissen, können wir definieren, was man unter einem Programm versteht:

Ein Programm ist die Formulierung eines Algorithmus in einer Programmiersprache.

Programme sind also Algorithmen, die in einer besonderen Sprache formuliert sind. Den Begriff "Programmieren" können wir somit wie folgt definieren:

Unter "Programmieren“ versteht man das Aufschreiben eines Algorithmus in einer bestimmten Programmiersprache.

Will man höhere Programmiersprache beherrschen, so muss man zuerst einmal ihr Vokabular und ihre Grammatik lernen. Zum Verständnis der Programmiersprache muss man zusätzlich auch deren Konzepte erlernen, um die man sich bei natürlichen Sprachen nicht kümmern muss. Bei imperativen Programmiersprachen wie Java sind dies z.B. Variablen, Anweisungen, Prozeduren. Alleine durch das Auswendiglernen von Vokabeln und Grammatikregeln schafft man es jedoch nicht, mit der Programmiersprache umgehen zu können. Man muss die Sprache konsequent einsetzen und durch ständiges Üben Erfahrungen sammeln.

Das Erlernen einer Programmiersprache ist nicht schwierig, da das Vokabular und die Grammatik nicht besonders umfangreich sind. Was sehr viel schwieriger ist, ist das Programmieren lernen, d.h. das Erlernen des Programmentwicklungsprozesses:

- Wie komme ich von einem gegebenen Problem hin zu einem Programm, das das Problem korrekt und vollständig löst.
- Wie finde ich eine Lösungsidee bzw. einen Algorithmus, der das Problem löst.
- Wie setze ich den Algorithmus in ein Programm um?

Während das Erlernen einer Programmiersprache ein eher mechanischer Prozess ist, bei dem die Verwendung des Compilers und das Hilfesystem helfen können, ist die Programmentwicklung ein kreativer Prozess, der Intelligenz voraussetzt. An dieser Stelle sind Sie gefragt! Programmieren lernen bedeutet in noch stärkerem Maße als das Erlernen einer Programmiersprache: üben und Erfahrungen sammeln.

- Schauen Sie sich die Programme anderer Programmierer an und überlegen Sie: Wieso hat der das Problem so gelöst?
- Denken Sie sich selbst Probleme aus und versuchen Sie, hierfür Programme zu entwickeln.
- Fangen Sie mit einfachen Aufgaben an und steigern Sie nach und nach den Schwierigkeitsgrad.
- Ganz wichtig ist: Versuchen Sie Programmierpartner zu gewinnen, mit denen Sie Probleme und Lösungsansätze diskutieren können.

Der Ausdruck "Programmieren" bezeichnet also im engen Sinn lediglich das Erstellen eines Programms. Oft bezeichnet man jedoch mit "Programmieren" alle Tätigkeiten eines Programmierers, die von der Problemstellung zur fertigen Lösung führen. Das Problemlösen mit Hilfe eines Computers umfasst folgende Stufen:

- Analyse der Problemstellung
- Entwurf des Algorithmus
- Erstellen des Programms
- Prüfen auf Korrektheit
- Dokumentation des Programms
- Anwendung des Programms

Ein Programm ist also eine eindeutige, logische Folge von (genormten) bekannten Bearbeitungsschritten endlicher Länge. Beim Computer richten sich diese Befehle an das Steuerwerk. Die Zahl der verschiedenen Befehle ist bei den Computern aus verständlichen Gründen beschränkt. Als Beispiel soll ein Algorithmus aus der Mathematik betrachtet werden: Die näherungsweise Berechnung des Kreisumfangs.

Algorithmus "Kreisumfang":
(K1) Nimm den Radius aus der Eingabe.
(K2) Multipliziere den Radius mit 2,0 und nenne das Ergebnis 'Durchmesser'.
(K3) Multipliziere den Durchmesser mit 3,1415926 und bringe das Ergebnis in die Ausgabe.

Dieser Algorithmus liefert beispielsweise zur Eingabe 7,0 die Ausgabe 43,982296. Wenn der Computer das Multiplizieren nicht beherrscht, dann müsste man ihm noch einen Algorithmus, ein Verfahren, für deren Durchführung geben.

Algorithmus "Multiplikation"

(M1) Schreibe die beiden Zahlen aus der Eingabe nebeneinander.
(M2) Nimm die erste Ziffer der zweiten Zahl.
(M3) Schreibe die erste Zahl sooft untereinander, wie die Ziffer der zweiten Zahl angibt, und zwar so, dass die letzte Ziffer unter der betrachteten Ziffer der zweiten Zahl steht. Für den Fall, dass besagte Ziffer 0 ist, schreibe 0.
(M4) Falls noch Ziffern in der zweiten Zahl vorhanden sind, nimm die nächste Ziffer und gehe nach (M3).
(M5) Addiere alle mit (M3) erzeugten Zahlen.
(M6) Zähle bei dem Ergebnis so viele Stellen von rechts ab, wie beide Eingabewerte zusammen an Stellen hinter dem Komma besitzen. Setze hier das Komma im Ergebnis.
(M7) Bringe dies Endergebnis in die Ausgabe.

Beispiel für die Eingaben 3,14 und 14,0:

```
 3,14 x 14,0
 -----------
    314
     314
     314
     314
     314
       0
 ----------
   43,960
```

Die Ausgabe ist 43,960. Dieser Algorithmus für die Multiplikation kann dann überall dort verwendet werden, wo "multipliziere" steht. Aus dieser Tatsache lassen sich zwei Erkenntnisse gewinnen:

- Wenn eine komplizierte Tätigkeit öfters benötigt wird, kann man für diese Tätigkeit einen eigenen Algorithmus definieren. Dieser wird dann Unteralgorithmus genannt.
- Man kann eine Programmiersprache auf eine bestimmte, begrenzte Menge von Operationen und Denkstrukturen beschränken.

Versuchen Sie doch einmal, den Algorithmus "Kreisumfang" so umzuformen, dass er überall dort, wo multipliziert wird, der Algorithmus "Multiplikation" als "Unteralgorithmus" aufgerufen wird. Auf dieser Idee beruht die Theorie der Softwareentwicklung. Komplizierte und komplexe Strukturen und Tätigkeiten werden durch bekannte Strukturen und einfachere Unteralgorithmen realisiert. Sie werden auch sehen, dass auch Programme auf entsprechend konstruierte "Unterprogramme" zurückgreifen können. Und es sei hier schon angemerkt, dass diese Möglichkeit die wichtigste und mächtigste Eigenschaft der Programmiersprachen ist. Die oben erwähnten komplizierten Strukturen sind genau jene, die der menschlichen Denkweise entsprechen. Zum Beispiel die Struktur:

"Wenn die Bedingung erfüllt ist, dann tue dies; sonst tue jenes."

Dass dabei eine ganze Reihe von Handlungen nacheinander erforderlich ist, wird dem Menschen gar nicht bewusst. Zuerst muss die Bedingung ausgewertet werden. Dazu müssen alle Informationen beschafft werden, die zu dieser Auswertung nötig sind. Danach müssen sie verarbeitet werden, bis festgestellt ist, ob die Bedingung erfüllt ist. Erst jetzt können die entsprechenden Tätigkeiten ausgeführt werden, was unter Umständen wieder recht kompliziert werden kann.
Die höheren Programmiersprachen bieten also schon als Grundoperationen recht komplizierte Dinge an. Ebenso werden komplizierte Denkstrukturen angeboten, die, wie gesagt, dem menschlichen Denken sehr verwandt sind.
Sie sind normierte Sprachen, die der Beschreibung von Algorithmen dienen. Genaueres über solche "algorithmische Sprachen" sollen Sie im folgenden Abschnitt erfahren. Durch die An-

passung an die menschliche Denkweise wird bei den Programmiersprachen das Schreiben von Algorithmen erleichtert; andererseits lässt sich aus einem Programm eine Beschreibung in natürlicher Sprache zurückgewinnen.

Die fest vorgeschriebenen Teile der Programmiersprachen orientieren sich meist am Englischen. So wird aus der Struktur

"Wenn ... dann ... sonst ..."

in der Programmiersprache C zu

"if (..) { ... } else { ... }

Allerdings wurden bei der Konstruktion der höheren Programmiersprachen die Ausdrucksmöglichkeiten so ausgewählt, dass Doppeldeutigkeiten unmöglich sind. Dazu wird die Form der Ausdrucksmöglichkeiten vorgeschrieben und ihre Bedeutung eindeutig und genau erklärt. Der entscheidende Punkt bei den höheren Programmiersprachen ist aber folgender. Dadurch, dass die Form der Programme, die in höheren Programmiersprachen geschrieben sind, bestimmten, festen Regeln genügt, können diese Regeln automatisch, also auch von einem Computer (mit einem entsprechenden Programm) analysiert werden. Da zu jeder erkannten Form die zugehörige Bedeutung genau festgelegt ist, kann das Programm automatisch (d. h. wieder von Computer) auf eine andere Form mit der gleichen Bedeutung, jedoch mit einfacheren Grundoperationen und Grundstrukturen umgewandelt werden.
Die Umwandlung kann so erfolgen, dass sich die Form völlig, die Bedeutung aber überhaupt nicht ändert. Wenn nun diese neue Form in einer, von dem Computer ausführbaren Sprache (Maschinensprache) besteht, so wurde eine höhere Programmiersprache (die der Mensch versteht) in die Maschinensprache (die der Rechner versteht) übersetzt.

Befehlssatz

Die einfachste Programmiersprache besteht aus einem Satz von Befehlen. Dies sind die eingebauten Kommandos, sie sind in ihrer Wirkung vergleichbar mit dem, was Sie mit den Tasten Ihres Taschenrechners auslösen. Mit diesen Befehlen lassen sich u.a. arithmetische Operationen ausführen und Werte für die Dauer der Rechnung speichern. Auch gibt es Befehle, mit denen man Werte vergleicht und damit entscheidet, was als Nächstes gemacht wird. Ein Teil der Befehle wird benötigt, wenn man Daten im Speicher ablegen oder von dort zurückholen will. Besondere Befehle befassen sich damit, in einem Programm Sprünge auszuführen, d.h. Programmteile, die im Moment nicht ausgeführt werden sollen, zu überspringen, oder auch um an eine bereits ausgeführte Stelle im Programm zurückzuspringen und sie so zu wiederholen.

Maschinensprache

Wenn man die Befehle durchnumeriert, erhält man eine Maschinensprache; das ist der einfachste Programmiercode. in diesem Code kann die Nummer eines Befehls als sein Name verwendet werden. Ein Maschinensprachenprogramm ist nichts weiter als eine lange Folge von solchen Codewörtern.
In einer Maschinensprache zu programmieren, ist nicht schwer, aber unglaublich mühsam. Zum Glück hatte einer der frühen Programmierer eine brillante Idee, wie man sich die Arbeit erleichtern kann. Wenn man ein Maschinenprogramm schreibt, das kurze Buchstabenfolgen erkennen und in zugehörige Maschinenbefehle übersetzen kann, dann braucht der Programmierer nicht die Codierung der Befehle in der Maschinensprache zu lernen. Stattdessen kann er jeden Maschinenbefehl als "Klartext-Kürzel" (engl. mnemonic) hinschreiben. Die entsprechenden Übersetzungsprogramme, man nennt sie Assembler, wurden bald für alle Computer entwickelt.

Das Programmieren in einer Assemblersprache ist etwas weniger mühsam. Ein Programm ist eine Folge von Kommandos, die jeweils aus zwei bis vier Buchstaben bestehen und denen Adressen von Speicherplätzen angefügt werden.

Zum Beispiel bedeutet das Kommando „ADD R2, R6“ :

Addiere den Inhalt des Speicherplatzes R2 zum Inhalt von R6 und speichere die Summe in R6. In welche Befehle dieses Kommando übersetzt wird, können Sie sich verdeutlichen, wenn Sie die Operation an einem Taschenrechner (der einen Speicher hat) mit einer Tastenfolge ausführen.

Noch bis zum Ende der fünfziger Jahre bestand das Programmieren tatsächlich aus der minutiösen Übersetzung von Befehlen in binär, oktal oder sexadezimal dargestellte Zahlen und deren Aneinanderreihung zu einem sinnvollen Programm. Man bezeichnet diese Tätigkeit mit **Codieren**. Die Unzulänglichkeiten dieses Verfahrens traten aber mit dem Schneller- und Größerwerden der Computer mehr und mehr hervor:

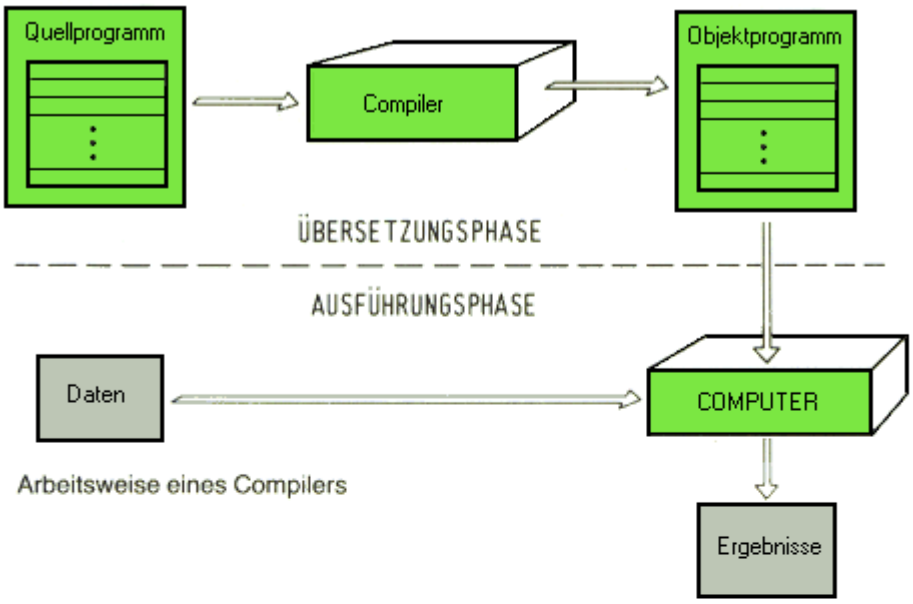
- Der Codierer war gezwungen, sein Programm auf die Eigenheiten des spezifischen Computermodells auszurichten. Er benötigte dazu genaueste Kenntnisse aller Details dieser Maschine und deren Befehlssatz. Der Austausch von Programmen zwischen verschiedenen Maschinen wurde dadurch unmöglich, und Kenntnisse der Codiermethoden eines Computers waren oft wertlos für die Codierung eines anderen. Jeder erstellte eigene Programme und war gezwungen, bei der Neuanschaffung eines Computers diese aufzugeben und die Codierarbeit von vorne zu beginnen. Es wurde klar, dass das gezielte Ausrichten von Algorithmen auf die merkwürdigsten Eigenheiten eines bestimmten Computers eine schlechte Verwendung des menschlichen Intellektes war.
- Der Programmierer wurde durch seine enge Bindung an eine Rechenanlage und die extrem begrenzten Hardwaremöglichkeiten - langsam, wenig Arbeitsspeicher, extrem teuer (mehrere MegaDMs/Rechner)- nicht nur befähigt, sondern sogar ermuntert, alle möglichen Tricks zu erfinden, um aus den Eigenheiten des Computers ein Maximum herauszuwirtschaften. Zu dieser Zeit galten die Programmierer noch als Beigabe des Hardwareherstellers, denn im Vergleich zu den Hardwarekosten, kosteten die Arbeitskräfte kaum was. Als die verzwickte Programmierung in Mode war, verwendeten Programmierer nicht nur viel Zeit zur Erstellung "optimaler" Programme, auch deren Tests stellte sich als äußerst schwierig dar. Es war für einen Mitarbeiter beinahe unmöglich, die Funktionsweise eines fremden Programms herauszubekommen (und oft war es sogar schwierig das eigene zu überblicken). Heute ist das Teuerste die Erstellung von Anwendersoftware. Deshalb vermeiden heutige Programmierer die Anwendung von Tricks um jeden Preis.
- Der so genannte Maschinencode enthielt sehr wenig Redundanz, auf Grund deren ein Fehler hätte entdeckt werden können. Bereits kleine Schreibfehler verändern einen Maschinencode in einen anderen gültigen Maschinencode, der aber völlig andere Bedeutung und somit Auswirkungen auf die Funktion des Programms hat. Solche Fehler sind nur sehr schwer zu entdecken, obwohl sie bei der Ausführung des Programms verheerende Folgen haben konnten.
- Die Darstellung eines Vorganges als unstrukturierte, lineare, monotone Sequenz von Befehlen ist eine für den Menschen ungeeignete Form, komplizierte Prozesse zu beschreiben und zu überblicken. Die Präsenz von hierarchischen Beschreibungsstrukturen, die eine Sicht auf die Funktionen in unterschiedlicher Detailierungstufe ermöglichen, ist das hauptsächlichste Hilfsmittel, um den Überblick zu wahren und Programme systematisch zu erstellen.

Diese Unzulänglichkeiten führten unter Anderem zur Entwicklung so genannter "höherer Programmiersprachen", die zum Einen nach den Gewohnheiten und Fähigkeiten des Menschen im Ausdruck seiner Gedanken ausgerichtet sind und oftmals in ihrer Art und im Umfang am Anwendungsgebiet ausgerichtet sind (und nicht an der unterlegten Hardware).

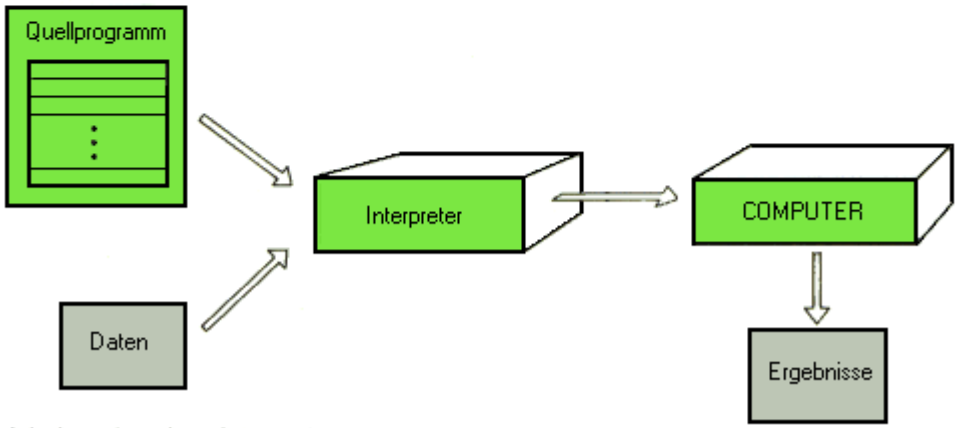
Interpreter, Compiler

Mit den Kommandowörtern der Assemblersprache war ein erster Schritt getan, die Programme für den Menschen verständlich zu schreiben. Man wählte Bezeichnungen, die wie ADD auf die Operation hinwiesen. Doch warum sollte man sich auf Wörter mit drei Buchstaben beschränken? Es wäre noch leichter, Programme zu schreiben, wenn man näher an der Umgangssprache formulieren könnte. Nun wiederholte sich der gleiche Schritt, der von Maschi-

nensprachen zu Assemblersprachen geführt hatte. Man entwickelte (kompliziertere) Übersetzungsprogramme, sie heißen Interpreter und Compiler. Diese neuen Programme übersetzten zunehmend komplexe Buchstabenfolgen in eine Form, die der Computer verstehen kann. Damit konnten die Programme in einer Sprache geschrieben werden, die aus Wörtern der Umgangssprache bestand.



Arbeitsweise eines Compilers



Arbeitsweise eines Interpreters

Meist tun diese Programme noch mehr, als nur zu übersetzen: Es meldet dem Programmierer auch Stellen in seinem Programm, die nicht den Konventionen der Sprache genügen - Programmierfehler also. Ein Maschinenprogramm lässt sich nur noch unter den größten Schwierigkeiten verstehen. Ein Beispiel soll das verdeutlichen:

a) In natürlicher Sprache: Schreibe das Wort "PASCAL"
b) In der höheren Sprache C: printf ("PASCAL");
c) In der Maschinensprache des programmierbaren Taschenrechners TI-59:
69 00 03 03 01 03 03 06 01 05 01 03 69 01
02 07 00 00 00 00 00 00 00 00 69 02 69 05

Wie Sie sehen, bieten die höheren Programmiersprachen ganz entscheidende Vorteile. Wir habe also kennen gelernt:

- Assemblersprache (kurz: Assembler, maschinenorientiert) Hier wird jeder Maschinenbefehl durch eine mnemotechnische Abkürzung bezeichnet. Speicheradressen können mit symbolischen Namen versehen werden. Die Zuordnung von Assemblerbefehl zu Maschinenbefehl ist 1:1, d. h. für jeden Maschinenbefehl wird ein Assemblerbefehl benötigt. Die Assemblersprache ist daher extrem stark an den jeweiligen Prozessor gebunden. Ein Transport des Programms auf einen anderen, nicht kompatiblen Prozessor ist nicht möglich.
- Höhere Programmiersprachen (problemorientiert) Hier wird die Programmierung in einer eigens entwickelten problemorientierten Sprache vorgenommen (algorithmische Sprache). Die Zuordnung der Befehle ist nicht mehr 1:1, ein Befehl in einer höheren Sprache hat in der Regel eine ganze Folge von Maschinenbefehlen als Ergebnis. Der Vorteil einer problemorientierten Sprache liegt auch darin, dass sie maschinenunabhängig ist. Ein Transfer der Programme auf andere Prozessoren ist mit wenig Aufwand möglich.

Die Programmiersprachen sind in erster Linie dazu da die Lösung einer Aufgabe in computergerechter Form zu formulieren. Die Lösung einer hier betrachteten Aufgabe soll also durch einen Algorithmus dargestellt in einer Programmiersprache beschrieben sein. Wie muss eine solche Programmiersprache (Algorithmische Sprache) nun aufgebaut sein damit sie geeignet ist beliebige Algorithmen darzustellen?

4 Algorithmische Sprachen

In diesem Abschnitt wollen wir untersuchen, welche Eigenschaften die Sprachen besitzen müssen, die zur Beschreibung von Algorithmen verwendet werden. Im letzten Abschnitt haben wir die wesentlichen Merkmale der algorithmischen Sprachen bereits kennen gelernt. Bei ihnen handelt es sich um normierte Sprachen, die der Beschreibung von Algorithmen dienen. In diesem Abschnitt wollen wir versuchen, zwei Kernfragen zu beantworten:

- Wie kann man eine Sprache normieren?
- Was ist zur Beschreibung von Algorithmen erforderlich?

Fangen wir gleich mit dem Problem der Normierung an. In jeder beliebigen Sprache, sei es die deutsche, englische, italienische, griechische oder japanische Sprache, wird ein Text als Folge von Zeichen niedergeschrieben. Die Menge dieser Zeichen ist bei den verschiedenen Sprachen nicht immer die gleiche, es gibt auch nicht immer gleich viel Zeichen. Aber für jede Sprache existiert eine endliche Menge von Zeichen, die zur Niederschrift eines Textes verwendet werden dürfen.
Wenn man diese Menge von Zeichen in einer festgelegten Reihenfolge ordnen kann, spricht man von einem Alphabet. Es ist aber noch nicht möglich, einen Text zu schreiben, wenn man nur das Alphabet kennt. Wenn Sie zum Beispiel unser Alphabet betrachten, dann ergibt: "qwesdfcvxwcbvhgnrzrrdegfbcvdgftejsmbljouri" noch keinen Text.
Was man benötigt, sind Elemente der Sprache, gebildet aus den Zeichen des Alphabets. Gebilde, die durch aneinanderreihen von Zeichen eines Alphabets nach irgendwelchen Regeln entstehen, heißen Worte. Worte sind also Zeichenreihen. Ein Wort unterliegt gewissen Bildungsregeln. "3az45" ist kein Wort der deutschen Sprache, da es gegen die Regeln der deutschen Wortbildung verstößt. Das einzelne Wort lebt wiederum von der Kombination mit anderen Worten. Eine Kombination von Worten heißt Satz.

Zeichen + Regeln --> Wort
Wort + Regeln --> Satz

Aus einer festgelegten Menge von Zeichen werden nach bestimmten Regeln Worte gebildet, die selbst wieder nach festen Regeln zu Sätzen vereinigt werden. Ein Text besteht dann aus einer Folge von Sätzen. Dieser formale Aufbau einer Sprache heißt Grammatik oder **Syntax** der Sprache. Die Syntax beschreibt im Wesentlichen die Regeln, die Sie beachten müssen, damit die Form des Textes richtig ist.

Die Syntax ist auch bei Worten wie BARTSCHE, DRAUDELN, GRUBBEND oder ZERWANKELMÜTIGKEIT und Sätzen folgender Art völlig in Ordnung: DER COMPUTER BETRITT

DEN RASEN. DER HIMMEL WEIGERT SICH, GEMÄß VERORDNUNG 25A DEN DUSCHVORHANG ZU WABELN. ES IST PHANTASTISCH, WIE UNMÖGLICH ES IST, DAß EINE MENGE VON WÖRTERN KEINEN SINN ERGIBT. Aber diese Worte und Sätze, die von der Syntax her richtig (also syntaktisch korrekt) sind, ergeben keinen Sinn, keinen Inhalt, sie bedeuten nichts. Der Sinngehalt eines Satzes oder Wortes muss also ebenfalls festgelegt werden. Das bedeutet, wenn Sie ein Wort schreiben, müssen Sie die Bedeutung des Wortes kennen. Wer MAUS schreibt und dabei an einen großen, grauen Dickhäuter denkt, verstößt gegen jene Regeln der Sprache, die für den Sinngehalt wesentlich sind, die Regeln der **Semantik**.

Mit der Erkenntnis, dass die Syntax einer Sprache ohne eine ausreichende Definition der Semantik nicht für die Normierung der Sprache ausreicht, ist die erste der beiden anfangs gestellten Fragen beantwortet. Es stellt sich jetzt die Frage, was in einer normierten Sprache alles vorhanden sein muss, um Algorithmen zu beschreiben.
Also benötigt eine algorithmische Sprache Möglichkeiten zur Beschreibung der Eingabe- und Ausgabedaten und zur Beschreibung der Verarbeitungstätigkeit. Die Beschreibung der Daten muss die Art der Daten, (z.B.: Zahlen, Striche, Texte, usw.) sowie die Anzahl der Daten enthalten. Die Beschreibung der Tätigkeiten soll der menschlichen Denkweise möglichst gut angepasst sein, muss aber trotzdem normiert sein. Dazu ein Beispiel:

- Wenn es regnet, nehme ich den Schirm mit, sonst nicht.
- Regnet es, so nehme ich den Schirm mit, sonst halt nicht.
- Ich nehme den Schirm nur dann mit, wenn es regnet.
- Nur falls es regnet, nehme ich den Schirm mit.
- Ich nehme, sollte es regnen, den Schirm mit, sonst nicht.
- Dann und nur dann, wenn es regnet, nehme ich den Schirm mit.

Eine Normierung tut not, denn alle diese Sätze haben dieselbe Bedeutung. Also entwerfen wir die Syntaxregel:

> **WENN** <Bedingung> **DANN** <Erfüllt-Aktion> **SONST** <Nicht-erfüllt-Aktion>

Dann lautet das Beispiel:

> **WENN** es regnet **DANN** ich nehme Schirm mit **SONST** ich lasse Schirm zu Hause;

Es gibt nur eine Möglichkeit, das ganze noch anders darzustellen, und zwar:

> **WENN** es regnet nicht **DANN** ich lasse Schirm zu Hause **SONST** ich nehme Schirm mit;

Aufgrund der Normierung der algorithmischen Sprache bleibt von den vielen Ausdrucksvariationen der natürlichen Sprache nur noch eine einzige Version übrig. Abkürzungen und Umstellungen sind nicht mehr zulässig. Die Sätze müssen immer in der vorgeschriebenen Reihenfolge aufgeschrieben und die Wörter immer ausgeschrieben werden. Wie in der natürlichen Sprache gibt es auch in einer algorithmischen Sprache zahlreiche festgelegte Worte, die sich nicht auf Gegenstände oder Tätigkeiten beziehen, sondern der Verbindung der Worte zu Sätzen dienen. Im Deutschen sind dies zum Beispiel die Präpositionen oder die Konjunktionen; im letzten Beispiel waren es die Worte WENN, DANN und SONST. Diese Worte dienten der Verbindung der Sätze: es regnet ich nehme Schirm mit ich lasse Schirm zu Hause Die drei "Schlüsselworte" verbinden die genannten Sätze in festgelegter Weise; man nennt sie daher auch Wortsymbole. Die im Beispiel gewählte Syntax:

> **WENN** <Bedingung> **DANN** <Erfüllt-Aktion> **SONST** <Nicht-erfüllt-Aktion>

wurde so gewählt, dass auf Grund der Kenntnis der deutschen Sprache der Sinn des "Gebildes" ungefähr klar wird. Bedingung steht für einen Satz, der nach anderen Regeln gebildet werden muss, wie die Sätze hinter den Worten DANN und SONST und der, wie es von einer Bedingung erwartet wird, entweder erfüllt oder nicht erfüllt sein muss. Ist die Bedingung erfüllt,

wird die Tätigkeit: <Erfüllt-Aktion> ausgeführt, im anderen Fall wird die mit <Nicht-Erfüllt-Aktion> bezeichnete Aktion ausgeführt. Somit hätten wir die Syntax und die Semantik unseres Beispiels definiert. Natürlich muss auch die Reihenfolge der Bearbeitung festgelegt werden: Prüfe zuerst die Bedingung und führe dann die entsprechende Tätigkeit aus. Die Wortsymbole sind also festgelegte Worte mit fest definierter Bedeutung. Sie werden in den algorithmischen Sprachen meist durch Symbolzeichen wie "+" für die Addition oder "/" für die Division ergänzt. Auch die Bedeutung dieser Zeichen ist festgelegt und nicht zu ändern. Die Menge aller Wort- und Zeichensymbole bildet zusammen mit der Beschreibung von Syntax und Semantik die Definition einer algorithmischen Sprache. Die Symbole dürfen auch in keiner anderen Bedeutung verwendet werden als in der festgelegten. Die Zahl der Symbole sollte so klein wie möglich, aber trotzdem so universell wie möglich sein. Die Vorteile von wenigen, der menschlichen Denkweise angepassten Symbolen und Zeichen liegt auf der Hand:

- Wenig Symbole erfordern geringen Lernaufwand sowohl für Syntaxregeln als auch für die Semantik.
- Eine geringe Anzahl von Symbolen kann auch durch ein relativ kurzes und effektives Übersetzungsprogramm erkannt und ausgewertet werden.
- Die Universalität der Symbole soll verhindern, dass ein Programmierer "um die Ecke denkt".

Mit den Symbolen alleine kommt die algorithmische Sprache aber immer noch nicht aus, sie möchten ja den Sätzen einen Inhalt geben und den Dingen die Sie behandeln und den Tätigkeiten einen Namen geben. Erinnern Sie sich noch an das Verfahren "S". "S" war dabei ein völlig willkürlich gewählter Name für das Verfahren. Anschließend wurde für das gleiche Verfahren ein anderer, ebenfalls willkürlicher, aber einleuchtenderer Name gewählt: "Verfahren zur Bündelung von Strichen in Fünfergruppen". So wie dort die Namen der Verfahren frei gewählt wurden, können auch die Namen für Dinge, Daten, Personen, Verfahren und Tätigkeiten in einer algorithmischen Sprache frei gewählt werden. Damit Sie sich bei den frei gewählten Bezeichnungen später auch auskennen, sei Ihnen hier erklärt, was mit einer solchen Bezeichnung eigentlich benannt wird:
Der Gegenstand der Benamselung wird zuerst durch seinen Namen und seine Art bekannt gemacht (z. B.: Zahl, Tätigkeit, Person, Menge, Buchstabe, Operation, Unteralgorithmus, usw.). Ferner wird er charakterisiert durch die Tätigkeiten, die auf ihn angewendet werden. Steht zum Beispiel irgendwo "X/12" und jemand fragt: "Was ist X?", können Sie nur antworten: "Es handelt sich um irgendetwas, das durch 12 geteilt wird". So eine Antwort ist aber recht unbefriedigend. Als Lehre sollten Sie daraus ziehen, alle Dinge sinnvoll bezeichnen und zusätzliche Erklärungen und Kommentare für den Leser hinzufügen. Beispiel:

Schlechter Stil: Y = X/12

Guter Stil: EIERKARTONZAHL = EIERZAHL/12

Das Beispiel enthält die frei gewählten Bezeichnungen: X, Y, EIERKARTONZAHL, EIERZAHL. Es fehlt nur noch die Einordnung der Zahl 12. Es handelt sich hierbei um eine allgemein übliche und allseits bekannte Bezeichnung. Sie wird daher auch in algorithmischen Sprachen als allgemein üblich betrachtet und Standardbezeichnung genannt. Die Ausdrucksmöglichkeiten einer algorithmischen Sprache werden also durch Wortsymbole, Symbolzeichen, Standardbezeichnungen und frei wählbare Bezeichnungen im Rahmen einer (computerübersetzbaren) Syntax festgelegt. Die Semantik der Symbole und Standardbezeichnungen ist vordefiniert und wird vom "Compiler" schrittweise in eine Maschinensprache übersetzt, während die Semantik der frei wählbaren Bezeichnungen von der durch die Syntax festgelegten Deklarationen (Bekanntmachungen) festgelegt wird.

Ideal ist es, wenn eine solche, syntaktisch festgelegte Form mit ihrer Semantik dem Menschen, der die Verarbeitungsvorschrift verfaßsst, leicht verständlich und zugänglich ist. Die erzeugten Verarbeitungsvorschriften müssen lesbar und der menschlichen Denk- und Ausdrucksweise angepasst sein und trotzdem normiert und computerübersetzbar. Seit Mitte der sechziger Jahre werden Programmiersprachen (= algorithmische Sprachen) entwickelt.

5 Programmiersprachen

Diese "höheren" Programmiersprachen wie z. B. BASIC, FORTRAN, Pascal oder C wurden entwickelt, um Probleme leichter lösen zu können. Dagegen waren die Maschinensprachen und auch die Assemblersprachen eher dafür bestimmt, die internen Abläufe im Computer zu steuern. Wenn man eine höhere Programmiersprache verwendet, braucht man sich nicht mehr darum zu kümmern, wie man die Befehle erhält und auf welche Speicherplätze man zugreifen kann. Einige Programmiersprachen haben sich an den Bedürfnissen bestimmter Anwenderbereiche orientiert. Ebenso wie verschiedene Typen von Taschenrechnern etwa für statistische, für kommerzielle oder für wissenschaftliche Berechnungen entwickelt wurden, gibt es anwendungsspezifische Programmiersprachen. Man kann die meisten Programme in jeder Programmiersprache schreiben. Man könnte auch ein kommerzielles Problem mit einem Taschenrechner lösen, der speziell auf statistische Berechnungen zugeschnitten ist. Doch es ist sicher vernünftiger, das besser angepasste Werkzeug zu verwenden. Der Anwendungsbereich der einzelnen Programmiersprachen lässt sich meist schon aus dem Namen erkennen:

FORTRAN (FORmula TRANSlator) ist eine der ersten und am weitesten verbreiteten Sprachen. Sie ist hauptsächlich für wissenschaftlich-technische Anwendungen bestimmt.

ALGOL (ALGOrithmic Language) ist für die Bearbeitung mathematischer Probleme und für wissenschaftliche Anwendungen besonders geeignet. Die Sprache wurde ständig weiter ausgebaut und verbessert (ALOGOL 60, ALGOL 68).

COBOL (COmmon Business Oriented Language) wurde entwickelt als Standardsprache für kommerzielle Berechnungen. Manche Anweisungen beziehen sich unmittelbar auf Lohnabrechnung oder Rechnungswesen.

BASIC (Beginners All-purpose Symbolic Instruction Code) ist eine einfache Sprache, die benutzt wird, um in Computer-Programmierung einzuführen. Sie ist einfach zu erlernen, geht aber nicht sehr weit. Ihre Schwächen sind Grund dafür, dass sie kaum noch im Unterricht eingesetzt wird.

LISP (LIST Processing language) wird weithin benutzt in Programmen, mit denen Zeichen von mathematischen Symbolen bis hin zu den Zeichen der Umgangssprache zu verarbeiten sind. Eine Hauptanwendung liegt im Bereich der Untersuchungen zur künstlichen Intelligenz.

Pascal (benannt nach dem Mathematiker und Philosophen Blaise Pascal) wurde für den Einsatz im Informatikunterricht entwickelt. Diese Sprache gehört zur Algol-Familie und eignet sich besonders für wissenschaftliche Probleme. Da Pascal als Lehr-Programmiersprache gedacht war, soll die Entwicklung dieser Sprache etwas genauer dargestellt werden. Niklaus Wirth verfolgte dabei u.a. die folgenden Ziele:

- Er wollte eine Sprache für den Unterricht schaffen, die einerseits die in allen Sprachen durchgängig vorkommenden Konzepte enthielt und andererseits Inkonsistenzen und unnötige Details vermied.
- Er wollte eine Standardsprache entwickeln, die preiswert war und sich leicht auf jedem Computer implementieren ließ.

In diesem Sinne ist Pascal so etwas wie eine allgemein verständliche Sprache für das Programmieren geworden. Sie lässt sich leicht erlernen und bildet eine gute Grundlage für die Aneignung anderer Sprachen. Man kann sagen: Wer Pascal beherrscht, lernt BASIC an einem Nachmittag. Und für FORTRAN braucht er nicht mehr als ein bis zwei Wochen.
Einer der Vorzüge von Pascal gegenüber den früher entwickelten Sprachen besteht darin, dass man die Programme fast mit den gleichen Formulierungen wie die Lösung des Problems schreiben kann. Die reservierten Wörter, sie bilden das Gerüst von Pascal, sind der englischen Umgangssprache entnommen. Wer Englisch kann, liest ein geschickt gemachtes Pascal-Programm wie eine umgangssprachliche Beschreibung des Weges, der das gestellte Problem löst. Und wenn man ein Problem aufbereitet, gelangt man sehr schnell von der umgangssprachlichen Formulierung zum Pascal-Programm.

C Die Entwicklung der Programmiersprache C ist eng mit der des Betriebssystems UNIX verknüpft. Nachdem die erste UNIX-Version noch in Assembler erstellt worden war (1969), entwickelte Ken Thomson 1970 die Sprache B zur Implementierung eines UNIX-Systems für eine PDP-7-Maschine. Aus der mit zu vielen Schwächen behafteten Sprache B entwickelte Dennis Ritchie 1972 C. Seit 1973 ist das Betriebssystem UNIX fast vollständig in C geschrieben. Zunächst gab es keinen offiziellen Sprachstandard. Stattdessen erreichte die Sprachdarstellung in einem Lehrbuch - deutsch: Kemighan, Ritchie; Programmieren in C, Hanser Verlag 1983 - den Status eines Quasi-Standards (Kernighan-Ritchie-Standard). Kleinere Erweiterungen und Verbesserungen führten zum ANSI-Standard. Die Sprache C++ wurde Anfangs der 80er Jahre von Bjarne Stroustrup an den Bell Laboratories entwickelt. Es handelt sich dabei um einen Zusatz für C.
C ist eine Sprache der 3. Generation (strukturierte Sprache) und gehört neben FORTRAN und Pascal zu den wichtigsten Höheren Programmiersprachen im Ingenieurbereich. Die wesentlichen Merkmale der Sprache sind:

- breites Anwendungsspektrum
- knappe Befehle (short is beautiful)
- sehr klares Sprachkonzept.

Was oben als Vorteil erscheint, erweist sich als Nachteil bezüglich der Erlernbarkeit als "Erstsprache". Die "kappen Befehle" könnte man etwas böswillig auch als kryptisch bezeichnen und das "klare Sprachkonzept" verlangt vom Programmierer Grundkenntnisse über Aufbau und Arbeitsweise von Computern, deutlich mehr als Pascal und FORTRAN. Allerdings steigen diese Grundkenntnisse bei der jungen Generation von Jahrgang zu Jahrgang. Fast jeder (interessierte) kann mit einem PC umgehen. Viele haben bereits gelernt, kleinere Probleme in Basic oder Pascal zu lösen. Und so kann man es heute wagen, ernsthaft mit C zu starten. Die Mühe lohnt sich!

Mittlerweile gibt es neben den vorgenannten aufschreibungsorientierten Programmiersprachen auch graphische Programmiersprachen. Populäre Firmenübergreifende Standards sind mir unbekannt. Idee ist es mittels graphischer Verknüpfung von Piktogrammen, die stellvertretend jeweils unterschiedliche Grundfunktionen darstellen, ein Programm ähnlich einer Zeichnung zu entwerfen. LabView ist ein solches Produkt mit einer solchen Programmierumgebung. LabView wird in der Messtechnik häufig eingesetzt. Virtuelle Messgeräte (Softwarefunktionen), wie Voltmeter, Ampèrmeter, Spektrumsanalysator etc werden als Piktogramme dargestellt und der Messaufbau, die Verkabelung, durch graphische Linien, die die Geräte an den unterschiedlichen Buchsen verbinden. Messvorgänge werden per Knopfdruck in der Graphik ausgelöst, und die Ergebnisse graphisch dargestellt. Vorteil dieser Art der Programmierung ist, dass sich der Anwender voll auf seine 'Welt' konzentrieren kann, ohne eine Programmiersprache lernen zu müssen. Ob sich diese Art der Programmierung auch in anderen Bereichen durchsetzt bleibt abzuwarten.

6 Algorithmen und das Lösen von Aufgaben

Bevor man ein Programm schreiben kann, muss man einen Algorithmus für die Lösung der (hier vorgegebenen!) Aufgabe entwerfen. Ein Algorithmus fasst alle Schritte zusammen, die man auf dem Wege zur Lösung gehen muss. Er ist im Allgemeinen so detailliert dargestellt, dass er die Grundlage für ein Programm bilden kann, aber noch nicht in einer Programmiersprache formuliert. Ein guter Programmierer wird einen Algorithmus ohne Schwierigkeiten aus der Umgangssprache in jede Programmiersprache übertragen können. Für den Anfänger stellt dagegen die "Unschärfe" der Umgangssprache häufig ein Problem dar.

Beispiel: Vertauschen von Zahlen

Beim Algorithmus für die Division durch einen Bruch muss der Kehrwert gebildet werden, d. h. der Zähler ist mit dem Nenner zu vertauschen. Man ist versucht, das auf folgende Weise zu machen:

Nimm den Zähler und den Nenner auf.
Gib dem Zähler den Wert des Nenners.
Gib dem Nenner den Wert des Zählers.

Bemerken Sie den Fehler, der auftreten würde, wenn der Computer diese Anweisungsfolge pedantisch ausführt (und das tut er tatsächlich!)? Beide Variablen, die für den Zähler und die für den Nenner, hätten zum Schluss den gleichen Wert, den des ursprünglichen Nenners. Das liegt daran, dass man den Wert des ursprünglichen Zählers nicht zwischengespeichert hat. Ein Algorithmus für den Computer muss das ausdrücklich vorsehen:

Nimm den Zähler und den Nenner auf.
Speichere den Zähler.
Gib dem Zähler den Wert des Nenners.
Gib dem Nenner den gespeicherten Wert.

Beispiel: Fallentscheidung

"Gegeben ist eine Variable X. Wenn X den Wert 2 hat, soll X den Wert 1 erhalten und umgekehrt". Ein an sich klarer und einfacher Algorithmus. Doch bei der Umsetzung gibt es ein paar Fallen.

Lösung 1:
Prüfe, ob X den Wert 2 besitzt.
Falls ja, setze X auf 1.
Setze X auf 2.

Lösung 2:
Prüfe, ob X den Wert 2 besitzt.
Falls ja, setze X auf 1.
Sonst setze X auf 2.

Lösung 3:
Setze X auf das Ergebnis der Rechnung 3 - X.

Welche Lösung ist richtig? Erstaunlicherweise keine! Bei der ersten Lösung hat X nach Ablauf immer den Wert 2. Die beiden anderen Lösungen scheinen beide richtig, wobei die dritte zwar elegant erscheint, aber für den Leser des Algorithmus schwerer verständlich ist. Sie sind aber deshalb falsch, weil nicht überprüft wird, ob X einen der beiden zulässigen Werte (1,2) besitzt. Richtig wäre also:

Lösung 2:
Prüfe, ob X den Wert 1 oder den Wert 2 besitzt.
Falls nein, melde den Fehler und beende die Arbeit.
Prüfe, ob X den Wert 2 besitzt.
Falls ja, setze X auf 1.
Sonst setze X auf 2.

Denken und Arbeitsweisen schulen

Das Denkvermögen des Menschen lässt sich ebenso trainieren wie seine körperliche Leistungsfähigkeit. In den Kapiteln dieses Skriptums werden Sie eine Vielfalt von Ansätzen zur Lösung von Problemen kennen lernen. Auch Vorgehensweisen wie die schrittweise Verfeinerung eines Problems und das Top-down-Verfahren beim Erstellen von Programmen werden an mehreren Beispielen ausführlich dargestellt. Es wird untersucht, wie sich die Eleganz eines Algorithmus auf das Programm und seine Effizienz auswirkt. Wenn man herausgefunden hat, wie ein Problem zu lösen ist, und einen Algorithmus auf dem Papier (oder im Kopf) hat, dann ist es an der Zeit, ein Programm zu entwerfen. Dazu muss gesagt werden: In den letzten Jahren hat es gewaltige Veränderungen beim Softwareerstellen gegeben.

Im Gegensatz zu den Anfangsjahren ist jetzt die Software und nicht mehr die Hardware der Faktor, der die größten Kosten verursacht. Untersuchungen haben gezeigt, dass beträchtlich mehr Zeit in die Überarbeitung und die Anpassung vorhandener Programme investiert wird als in die Entwicklung neuer Software. Das Interesse konzentriert sich daher auf Verfahren, wie man Programme schreiben kann, die nicht nur korrekt arbeiten, sondern auch von anderen verstanden werden können. Der neue Bereich der Softwareerstellung, den man Software-Engineering (Softwaretechnik) nennt, befindet sich in einer stürmischen Entwicklung. Es darf die systematische Arbeitsweise (Arbeitsdisziplin) nicht vergessen werden: Lernen und Verbessern erfordert einen Rückkopplungsprozess. Allgemein: Dazu definiert sich der Ingenieur ein quantitativ erfassbares Ziel und versucht es zu erreichen. Dabei ist es wichtig ein objektives Maß (Metrik) dafür zu haben wie nahe er dem Ziel ist. Aus der Abweichung ergibt sich Steuerungsbedarf im Bestreben dem Ziel näher zu kommen. Dieser interaktive Prozess der Zielannäherung kann nur verbessert (Optimierung) und an Andere weitergegeben (Erfahrungsaustausch) werden, wenn er dokumentiert und objektiv nachvollziehbar (entspricht in der Mathematik: 'nachrechenbar') ist. Für die Softwareentwicklung bedeutet dies: Alle Arbeitsschritte, Metriken und Denkprozesse müssen nachvollziehbar (schriftlich) dokumentiert sein, inklusive der getroffenen technischen Entscheidungen. Nur so kann man selbst und alle zusammen an Können gewinnen.

Programmierstil und Softwarequalität

Auch der Programmierstil lässt sich durch Training verbessern. Früher meinte man, zum Programmieren gehöre ein angebotenes Talent, der eine besitze es, der andere nicht. Doch das Programmieren lässt sich wie das Denken schulen und verbessern. Viele Probleme lassen sich auf unterschiedlichen Wegen lösen. Im Interesse derjenigen, die ein Programm lesen (und verstehen!) müssen, sollte man ein Gespür dafür entwickeln, welcher Lösungsweg gut ist und welcher nicht. Die Problemlösung läuft also meist nach folgendem Schema ab: