Sprachen für Computer

Warum braucht man überhaupt eine Programmiersprache ? Welche Sprache ist für welchen Zweck geeignet ? Welche Sprachen gibt es für Commodore-Computer ? Der folgende Artikel gibt Auskunft.

Wer sich eine Stereo-Anlage oder einen Video-Recorder zulegt, muß in der Regel nur die Funktion einiger wichtiger Tasten kennenlernen, um damit umgehen zu können. Bei einem Computer sieht das schon etwas anders aus. Zwar muß der zukünftige Programmierer sich auch hier als erstes mit allen Tastenfunktionen vertraut machen, aber wenn er diese Phase mit Mühe und Not und schlechtem Handbuch hinter sich gebracht hat, beginnen erst die eigentlichen Probleme: Er muß eine Programmiersprache lernen, nämlich Basic.

Doch in vielen Fällen bleibt es nicht dabei. Nach einer Zeit der Euphorie, in der man glaubt, alles in Basic programmieren zu können, stellt man schnell einige Schwächen dieser Sprache fest: Basic-Programme sind langsam, der Aufruf von Unterprogrammen über Zeilennummern statt über Namen führt ebenso wie das Fehlen von Strukturbefehlen zur schlechten Lesbarkeit, und schließlich macht die Vielfalt der verschiedenen Basic-Dialekte es fast unmöglich, Programme von anderen Computersystemen ohne großen Änderungsaufwand zu übernehmen. Es bleibt also nur der Umstieg auf eine andere Progammiersprache.

Über diesem Punkt, beginnt die Programmierer-Gemeinde sich zu spalten: Die eine Gruppe fühlt sich vom Begriff »Maschinensprache« wie magisch angezogen und geht daran, die Schwächen von Basic in mühsamer Byte-für-Byte-Arbeit zu umgehen. Die andere Gruppe sieht sich nach echten Alternativen zu Basic um und stößt dabei mit einiger Sicherheit auf Begriffe wie »Pascal«, »Forth« oder »Logo«.

Wozu braucht man eine Programmiersprache?

Wenn wir uns mit anderen Menschen unterhalten wollen, benötigen wir eine Sprache. Genauso benötigt man auch für die Kommunikation mit — in einem gewissen Sinne — »intelligenten« Maschinen eine gemeinsame Basis, eine Sprache. Diese Sprache dient allerdings in erster Linie nicht zur Unterhaltung, sondern um dem Computer Anweisungen zu erteilen, um ihn zu programmieren.

Man unterscheidet grob zwischen den maschinennahen Programmiersprachen (Assembler) und den höheren Programmiersprachen wie Basic, Pascal oder Fortran. Während die Assemblersprachen jeweils eng an eine bestimmte CPU (Central Processing Unit, oder Zentralprozessor) gebunden sind, zeichnen sich die höheren Programmiersprachen neben ihrer besseren Lesbarkeit vor allem durch ihre Prozessorunabhängigkeit aus. Im Idealfall sollte es so sein, daß zum Beispiel ein auf einem Apple geschriebenes Basic-Programm auch ohne Änderungen auf einem C 64 laufen sollte.

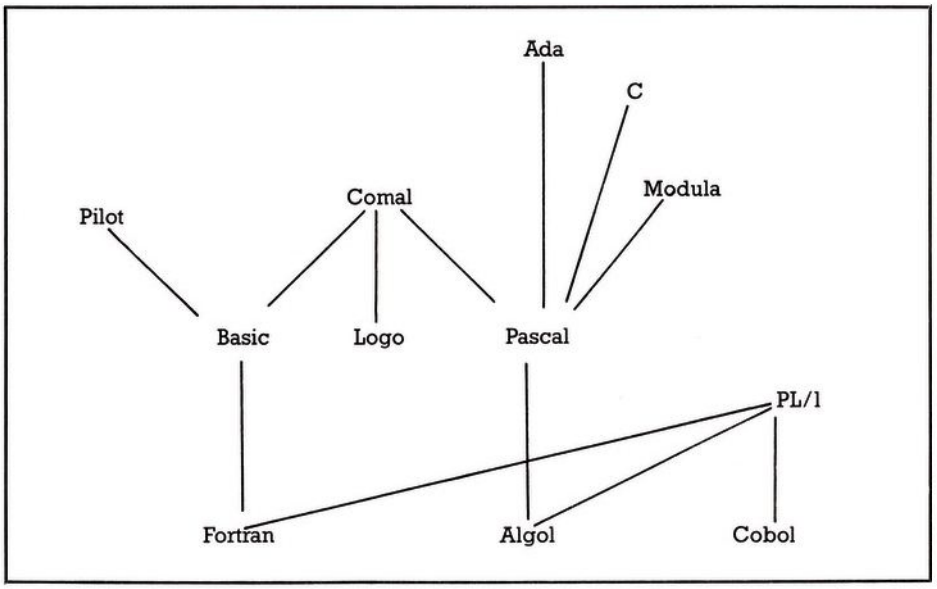
Natürlich funktioniert das in der Praxis nicht — zumindest bei Basic. Hierfür gibt es zwei Gründe. Zum einen existiert eine Vielzahl verschiedener Basic-Dialekte, das heißt vom Standard abweichende Basic-Versionen. Zum anderen gibt es in Basic die Möglichkeit, mittels PEEK, POKE oder SYS direkt auf die Maschinenebene zuzugreifen, was wegen unterschiedlicher Hardware der einzelnen Computer auch unterschiedliche Folgen zeigt. Gerade dieser Zugriff auf die Maschinenebene wird aber von den auch im engeren Sinne höheren Programmiersprachen tunlichst vermieden, damit die Programme »portabel«, also auf andere Computer übertragbar bleiben. Insofern ist Basic um einiges näher an der Maschinenebene als die meisten anderen Sprachen.

Compiler und Interpreter

Jeder Computer hat eine Maschinensprache, das ist die Sprache, die sein Zentralprozessor direkt versteht. Diese »Sprache« besteht unglücklicherweise nur aus Folgen von »0« und »1«, ist also für einen Menschen einigermaßen schwer verständlich. Eine Ebene höher stehen da schon die Assemblersprachen, bei denen es für jeden Befehl der Maschinensprache eine Klartext-Abkürzung gibt. Spezielle Programme, selbst auch »Assembler« genannt, übersetzen diese mit »Mnemonics« bezeichneten Abkürzungen in die Maschinensprache.

Doch auch in dieser Form ist das Programmieren noch ein sehr mühseliges Geschäft. Außerdem sind solche Assembler-Programme nicht oder nur sehr schwer auf andere Computersysteme übertragbar.

Anfang der fünfziger Jahre kam daher folgerichtig das Konzept der »höheren Programmiersprachen« (siehe Bild) auf. Eine Hochsprache unterscheidet sich in wesentlichen Punkten von der Assembler-Sprache. Die Programme sind maschinenunabhängig, können also problemlos auf den verschiedensten Computern laufen. Außerdem sind sie



Der »Stammbaum« der höheren Programmiersprachen

für einen Menschen wesentlich besser nachzuvollziehen und zu lesen, was die Fehleranfälligkeit drastisch senkt. Allerdings werden diese Vorteile mit einem Nachteil erkauft: Es gibt keinen Prozessor, der eine solche Hochsprache direkt ausführen könnte; die Sprache ist der Maschine einfach »zu hoch«.

Doch dieses Problem läßt sich leicht lösen. Alles was man braucht, ist ein Übersetzer, der die Hochsprache in die Maschinensprache übersetzt, und damit dem Computer zur Verarbeitung zugänglich macht. Der Übersetzungsvorgang selbst folgt dabei festen Regeln, was nichts anderes bedeutet, daß es einen Algorithmus (vereinfacht gesagt eine Rechenanleitung) dafür gibt. Jeder Algorithmus kann aber programmiert werden, und somit ist der Vorgang der Übersetzung automatisierbar. Die entsprechenden Übersetzungsprogramme heißen »Compiler«. Natürlich benötigt man für unterschiedliche Computersysteme (mit verschiedener Maschinensprache) auch unterschiedliche Compiler. Das Compiler-Programm selbst wie auch der erzeugte Maschinencode sind je nach CPU unterschiedlich, das übersetzte Programm tut jedoch völlig unabhängig vom Computersystem immer das gleiche, nämlich genau das, was durch das Hochsprachen-Programm vorgegeben ist.

Daneben gibt es, besonders auf Mikrocomputern verbreitet, eine zweite Art der Bearbeitung einer höheren Programmiersprache. Der C 64 beispielsweise hat als CPU einen 6510-Prozessor, der an sich weit davon entfernt ist, auch nur ein paar Brocken Basic zu beherrschen. Dennoch kann man mit dem Computer ganz normal in Basic arbeiten. Das Programm muß nicht erst übersetzt werden, sondern es wird so, wie es ist, einfach interpretiert. Damit ist nichts anderes gemeint, als daß der Basic-Interpreter des C 64 einfach einen Super-Prozessor simuliert, der Basic als Maschinensprache hat. Bei dieser Simulation wird fortwährend Zeichen für Zeichen so interpretiert, wie es der hypothetische Basic-Prozessor tun würde.

Natürlich ist dieses Vorgehen nicht besonders ökonomisch. Bei jedem Programmlauf muß der Interpreter ja wieder Zeichen für Zeichen durchsehen, was er damit anfangen kann. Der Compiler hingegen übersetzt das Programm nur einmal und wird anschließend nicht mehr gebraucht, da nun reiner Maschinencode vorhanden ist.

Zu Anfang des Computerzeitalters war Rechenzeit kostbar, und so ist es kein Wunder, wenn in den fünfziger Jahren ausschließlich Compiler entwickelt wurden. Interpreter kamen erst mit höheren Prozessorleistungen in Mode.

Der erste Schritt — Fortran

Aus dem Bedürfnis nach einfacher Programmierung und Übertragbarkeit der Programme entstand 1952 die erste höhere Programmiersprache der Welt, nämlich Fortran. 1954 gab es bereits eine in wesentlichen Punkten verbesserte und ausgereifte Version, Fortran II, das über lange Jahre Standardsprache auf Großrechenanlagen war und sich auch heute noch — inzwischen als Fortran V — regen Zuspruchs erfreut.

Der Name Fortran steht für **FOR**mula **TRAN**slator (Formelübersetzer), und genau dafür ist die Sprache auch entwickelt worden. Ihre Stärke liegt damit in der Programmierung mathematischer und naturwissenschaftlicher Formeln. Neben den bekannten Zahlentypen Integer (ganzzahlig) und Real (Fließkommazahlen) gibt es in Fortran daher noch den Typ Double Precision für das Rechnen mit doppelter Genauigkeit, sowie den vor allem in der Elektrotechnik sehr wichtigen Typ der »komplexen Zahlen« (Complex). Letzterer Datentyp steht aber wegen des intern zu treibenden Aufwands für komplexe Berechnungen in den meisten Mikrocomputer-Versionen nicht zur Verfügung.

Fortran ist eine streng zeilen- und formatorientierte Sprache und erlaubt in der Regel nur einen Befehl pro Zeile. Wie in Basic, das übrigens im Jahre 1965 aus Fortran hervorgegangen ist, muß viel mit dem Befehl GOTO hin- und hergesprungen werden, jedenfalls beim weitverbreiteten Fortran IV. Die neueste Version, Fortran V, nimmt der Sprache mit Strukturen wie IF...THEN... ELSE...ENDIF etwas von ihrer Ursprünglichkeit.

Fortran gibt es unter CP/M für fast alle Mikrocomputer mit Z80-CPU. Für den C 64 ist die Sprache derzeit noch nicht erhältlich.

Nur wenige Jahre nach der Entwicklung von Fortran kamen zwei weitere wichtige Programmiersprachen auf, Algol und Cobol.

Mit Algol (**ALGO**rythmic **L**anguage) sollte in erster Linie mathematisch-theoretische Probleme angegangen werden. Die Sprache hat nie auch nur im entferntesten die Verbreitung von Fortran gefunden, ist aber dennoch sehr bedeutsam als »Stammvater« einer ganzen Generation von hochentwickelten, blockorientierten Sprachen, deren vorläufige Krönung die Sprache Ada darstellt.

Cobol (**CO**mmon **B**usiness **O**riented **L**anguage) wurde im Jahre 1959 für kaufmännische Anwendungen entwickelt. Eine der Zielsetzungen für die Entwicklung war der Wunsch (oder besser die Fiktion), daß auch Nicht-EDV-Fachleute in der Lage sein sollten, die Programme wirklich lesen zu können.

Aus dieser Zielsetzung entstand die wahrscheinlich schlechteste Programmiersprache aller Zeiten, die zu allem Unglück auch noch die am meisten verbreitete für den kommerziellen Einsatz wurde, und der erst in neuester Zeit mit der Verbreitung der Heimcomputer in Basic ein Konkurrent erwuchs.

Cobol-Programme simulieren natürliche (englische) Sprache. Die Basic-Zeile »IF X>0 THEN A=B/X« sieht in Cobol folgendermaßen aus: IF X GREATER THAN 0 DIVIDE B BY X GIVING A

Bei so einer Sprache wird natürlich aus jeder mathematischen Formel ein völlig unüberschaubares Gebilde. Hinzu kommt noch die merkwürdige Eigenschaft von Cobol, den Programmierer als Precompiler einzusetzen. Da gibt es als Vorspann zum eigentlichen Programm, der »Procedure Division«, eine sogenannte »Data Division«, in der vom Programmierer (!) festgelegt werden muß, wieviel Speicherplatz für eine Variable verwendet werden soll. Auch das Umcodieren von der internen Darstellung einer Variablen zur Ausgabe auf Bildschirm oder Drucker muß hier genau bestimmt werden.

Wenn Cobol dennoch eine so weite Verbreitung gefunden hat, dann liegt das in erster Linie an der ausgeprägten Fähigkeit, große Datenmengen in Dateien zu organisieren und zu verwalten, eine im kaufmännischen Bereich sehr wichtige Aufgabe. Für alle anderen Anwendungsbereiche muß die Sprache allerdings als hoffnungslos ungeeignet bezeichnet werden.

Wer trotz aller Warnungen in Cobol einsteigen möchte, der hat im Kleincomputerbereich derzeit nur unter CP/M die Möglichkeit dazu.

Aus dem Bestreben, eine einheitliche Sprache zu entwickeln, welche alle Vorzüge der damals wichtigsten Sprachen Fortran, Algol und Cobol

in sich vereinen sollte, entstand Mitte der sechziger Jahre die Sprache PL/1 (Programming Language 1). Leider wurde das Ziel nicht vollständig erreicht. Zwar vereinigt PL/1 Eigenschaften aller drei Sprachen in sich, aber nicht unbedingt die besten. PL/1-Programme wirken in der Regel aufgebläht und dennoch merkwürdig unzusammenhängend und unstrukturiert. Je nach persönlichem Programmierstil kann ein PL/1-Programm wie in Fortran, Algol oder Cobol geschrieben aussehen — im schlimmsten Falle auch wie ein Gemisch aus allen dreien. Eine fast unüberschaubare Anzahl an Schlüsselworten trägt zur Verwirrung jedes Programmierers bei. Böse Zungen ordnen die Sprache denn auch eher den Problemen als den Lösungen zu. PL/1 ist unter dem Betriebssystem CP/M für Heimcomputer verfügbar.

Pascal

Nach der Vorstellung dieser altehrwürdigen, aber immer noch weitverbreiteten Sprachen kommen wir nun zu einem modernen Klassiker unter den Programmiersprachen. Die Rede ist von Pascal, das 1971 von Niklaus Wirth an der ETH Zürich auf der Grundlage von Algol entwickelt worden ist. Pascal ist übrigens ausnahmsweise keine Abkürzung, sondern die Sprache erhielt ihren Namen zu Ehren des französischen Mathematikers und Philosophen Blaise Pascal, der im 18. Jahrhundert lebte.

Wie der Vorgänger Algol ist auch Pascal eine stark standardisierte Sprache. Der ursprüngliche »Wirth-Standard«, der noch immer den meisten Pascal-Versionen zugrunde liegt, regelt sehr genau, welche Sprachelemente erlaubt sind, und was sie bewirken.

Als Alternative existiert seit einiger Zeit das sogenannte USCD-Pascal, entwickelt in Kalifornien, das den im Wirth-Standard nicht vorgesehenen Datentyp »String« sowie entsprechende Funktionen dazu vorsieht. Der Unterschied zwischen Wirth- und USCD-Pascal ist ansonsten minimal. Die Übertragbarkeit von Pascal-Programmen wird dadurch jedenfalls kaum berührt, weil man sich in Pascal bei Bedarf einfach neue Datentypen definieren kann. Hat man keinen Datentyp »String« zur Verfügung, dann schreibt man einfach:

TYPE STRING = ARRAY [1..80] OF CHAR;

Der Typ »Char« ist in Pascal vordefiniert und repräsentiert jeweils ein einzelnes Zeichen. Mit der obigen Pascal-Zeile wurde der (neue) Datentyp »String« als Feld aus maximal 80 »Char«-Elementen definiert. Man kann jetzt in diesem Sinne weitermachen und mit Stringvariablen arbeiten. Das geht über die — in Pascal immer obligatorische — Variablenvereinbarung:

VAR ALPHA: STRING;

Damit wurde eine Variable Alpha vereinbart, die vom Typ String sein soll.

Auf diese Art und Weise kann der Vorrat an Datentypen fast beliebig erweitert werden, ein Konzept, das in Pascal zum ersten Mal konsequent realisiert wurde und seither aus modernen Programmiersprachen nicht mehr wegzudenken ist.

Pascal ist wegen seiner Leistungsfähigkeit und wegen seines logischen und konsistenten Aufbaus als Ausbildungssprache für angehende Programmierer oder Informatiker sehr beliebt.

Die Sprache erzieht zum strukturierten Programmieren und, mehr noch, zum strukturierten Denkansatz bei der Problemlösung. Ein solcher Ansatz ist bei der Entwicklung größerer Programme ein nicht zu unterschätzender Vorteil.

Pascal gibt es in mehreren Versionen für den C 64 und unter CP/M.

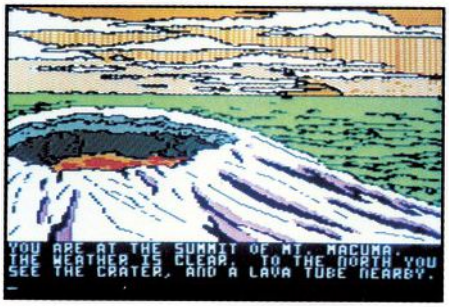
Eine Marktübersicht finden Sie in einer der nächsten Ausgaben. (ev)

Amazon — Das besondere Adventure

Amazon ist das erste aus einer Reihe von Abenteuerspielen, die gemeinsam mit bekannten Romanautoren produziert wurden. Was dahinter steckt, ist beachtlich.

Michael Crichton (Autor von Romanen wie »The Andromeda Strain«, »Congo« und »The first great Train Robbery 1855«), der normalerweise seine Bücher immer gleich verfilmt, tat diesmal etwas ganz anderes. Er schrieb weder ein Buch noch drehte er einen Film: Er setzte sich mit dem Programmierer Stephen Warady und dem Film-Grafik-Illustrator David Durand zusammen. Nach zwei Jahren Arbeit kam dann das Adventure Amazon (99 Mark) dabei heraus.

Ziel des Spiels für den C 64 ist es, die verlorene Stadt »Chak« zu finden, dort einen sagenumwobenen Schatz herauszuholen und damit wieder heimzukommen. Das Ganze spielt sich im Amazonasgebiet ab, das von wilden Eingeborenen und



Noch ist der Vulkan ruhig

tückischen Guerrilleros besiedelt ist.

Amazon kann auf drei verschiedenen Schwierigkeitsgraden gespielt werden, was bei Abenteuerspielen unüblich ist. Da das Spiel mit guter Grafik, Soundeffekten, Rätselaufgaben und auch joystickgesteuerter Action vollgepackt ist, benötigt es vier Diskettenseiten. Bemerkenswert ist die aufwendige Verpackung mit ihren vielen kleinen Beilagen, darunter auch durcheinandergewürfelte Lösungshinweise für Anfänger, die man mit Hilfe einer Decodierungstabelle erst entschlüsseln muß — damit man nicht aus Versehen schummelt. Um unnötige Raterei zu vermeiden, legt Tellarium (früher Trillium) übrigens zu jedem ihrer Produkte eine Wortschatzliste bei. Wer noch Näheres über die Tellarium-Adventures erfahren will, der sollte die Mai-Ausgabe von Happy-Computer lesen.

(M. Kohlen/rg)

Quelle: Fun-Tastic Videospielversand, Tannhäuserplatz 22, 8000 München 81, Preis: 99 Mark