Gegenüberstellung

Compiler oder Interpreter? Compiler sind schneller. Interpreter brauchen weniger Platz. Wir zeigen Ihnen die Vorteile beider Methoden.

Beide Worte Compiler und Interpreter — kommen aus dem Englischen wie alles in der Computerei. Frei übersetzt bedeutet Compiler soviel wie Sammler oder Zusammensteller und Interpreter heißt Übersetzer. Damit ließe sich mit einiger Phantasie die Arbeitsweise schon erraten, aber wir wollen uns anhand zweier der bekanntesten Vertreter beider Gruppen ein genaues Bild über ihre Funktionsprinzipien machen. Diese beiden Vertreter sind zum einen der Basic-Interpreter von Microsoft, den wir in den meisten Heimcomputern finden, wie zum Beispiel im C 64, im Apple IIc, TRS-80 und so weiter. Aber auf vielen PCs und sogar Großrechnern läuft eine Version dieses Interpreters. Der Vertreter der Compiler ist das Turbo-Pascal, das sich mittlerweile zu einem Standard entwickelt hat. Da der Compiler unter CP/M läuft, ist es Voraussetzung, daß bei Heimcomputern eine Z80-CPU vorhanden ist. Natürlich finden wir auch hiervon Versionen auf allen anderen Rechnern.

Wie funktioniert ein Interpreter? Bestimmt haben Sie schon mit einem Monitorprogramm in den Speicher Ihres Computers geschaut und sich gewundert, daß Sie keine Basic-Wörter gefunden haben. Nun, sie sind schon vorhanden, wenn auch in verschlüsselter Form. Bekanntlich wird meist der ASCII-Code zur Darstellung von Zeichen verwendet. Dieser endet bei 127 ($7F). Mit 8 Bit Datenbreite, wie wir sie in den Heimcomputern finden, ist es aber möglich, 128 weitere Codes zu vergeben. Genau dies wird mit den Basic-Schlüsselwörtern gemacht. Der Interpreter übersetzt diese in sogenannte »Token«. Token heißt Kennzeichen. Diese Umwandlung hat einen entscheidenden Einfluß auf die Arbeitsgeschwindigkeit des Interpreters. Mit Hilfe des Codes für die einzelnen Basic-Befehle kann er schnell die Adresse der dazugehörigen Routine in einer Tabelle finden. Ein weiterer Vorteil ist, daß die zeitraubende Unterscheidung zwischen Variable und Befehl entfällt, da alle Codes größer 127 als Token und der Rest als Variable interpretiert wird. Nebenbei wird durch die Umwandlung in nur einen Wert für einen Befehl auch eine Menge Speicherplatz eingespart. Nun wissen wir, welche Aufgabe der Interpreter während der Programmeingabe hat: das Übersetzen von Schlüsselwörtern in Token und Kennzeichnen der Variablen. Um nun die Arbeitsweise während des Programmablaufes zu verstehen, wollen wir einmal eine Basic-Programmzeile analysieren.

Wenn wir folgendes eingeben:
`10 PRINT"BASIC":GOTO10` ⟨Return⟩
und uns dann mit dem Monitor das Programm anschauen, so entdecken wir folgende (hexadezimale) Zeichenfolge ab Adresse $0800:

```
0800:  00 12 08 0A 00 00 99 22 42 41 53
080B:  49 43 22 3A 89 31 30 00 00 00 00
```

Was haben diese Bytes zu bedeuten? Das erste Byte (00) kennzeichnet den Beginn einer Basic-Zeile, die beiden darauffolgenden Bytes (12 08), auch Linkpointer genannt, geben die Adresse der nächsten Basic-Zeile im Speicher an. Byte 3 und 4 stellen die Zeilennummer dar (0A 00 ergibt 10 Dezimal — das erklärt auch, warum die Zeilennummern 65535 nicht überschreiten können, da man mit 2 Bytes nicht mehr darstellen kann). Tatsächlich wird aber nur bis 63999 numeriert. Die nächste 00 ist ein Trennungszeichen; der Interpreter kann dadurch erkennen, daß hier der eigentliche Programmtext beginnt. Die 99 steht als Token des Print-Befehls, spart also 4 Bytes. Die 22 stellt das Anführungszeichen dar und die folgenden 5 Bytes ergeben das Wort Basic, geschlossen von einem weiteren Anführungszeichen (22). Der Doppelpunkt mit dem Wert 3A trennt das GOTO (89) ab. Es folgt nun wieder die Zeilennummer, zu der gesprungen werden soll — in diesem Falle (00 00 00), welches das Ende des Basic-Textes kennzeichnet (3 mal 00 = Textende). Während des Programmablaufes liest der Interpreter nun die im Speicher abgelegten Bytes der Reihe nach durch. Stößt er nun auf ein Token, so verzweigt er in die entsprechende Betriebssystemroutine (in der die Aufgabe dieses Befehls festgelegt ist), führt diese aus und liest das nächste Byte. Entdeckt er nun eine Variable, so versucht er, diese zuerst einmal im Speicher zu finden. Gelingt ihm dies nicht, so fügt er sie an eine eventuell bestehende Variablentabelle an oder er schafft sich mit Hilfe der Garbage Collection, wenn nötig, Platz dafür. Gleichzeitig muß der Variablentyp erkannt werden, das heißt ob es sich um Real, Integer oder um Arrays handelt. Wird das Programm editiert, so muß der Interpreter den Programmtext im Speicher verschieben, insofern etwas hinzukommt oder gelöscht wird. Dies erklärt auch, warum ein mit STOP oder BREAK unterbrochenes Programm, wenn es verändert wird, nicht wieder mit CONT fortgesetzt werden kann. Beim Verschieben des Programmtextes wird nämlich die Variablentabelle überschrieben, so daß der Interpreter seine Variablen nicht mehr findet. Eine weitere Aufgabe hat der Interpreter beim Listen. Er muß jetzt die Token wieder in Klartext zurückübersetzen, so daß sie vom Bediener gelesen werden.

Compiler geben Gas

Compiler kann man als direkte Schnittstelle einer Hochsprache zur niedersten Ebene des Computers, der Maschinensprache, betrachten. Worin liegt nun der große Unterschied zum Interpreter? Es gibt zwei Gruppen von Compilern. Die einen erzeugen einen Zwischencode, den sogenannten P-Code und arbeiten somit in entferntem Sinne ähnlich wie ein Interpreter. Dieser P-Code hat den Vorteil, daß er relativ platzsparend ist, andererseits ist aber, durch die interpreterähnliche Struktur bedingt, der Geschwindigkeitsvorteil nicht überragend hoch. Es lassen sich hierbei Zeitvorteile von bis zu 40 Prozent gegenüber einem Interpreter erreichen. Beispiele hierfür sind der bekannte UCSD-Compiler oder der Austro-Compiler. Die zweite Compiler-Art erzeugt direkt Maschinencode. Da wäre zum einen der Aztek-C-Compiler, der Assembler-Quellcode erzeugt oder Turbo-Pascal, das direkt Maschinencode im Speicher ablegt. Doch wie läuft nun eine Programmausführung mit einem Compiler ab? Bereits bei der Eingabe bemerken wir den ersten Unterschied: Wir können den Programmtext (Quell-

code) mit Hilfe eines beliebigen Editors erstellen. Das kann zum Beispiel ein Textverarbeitungsprogramm wie Wordstar sein. Zwar haben die meisten Compiler einen Editor eingebaut, aber die Eingabe gestaltet sich über eine Textverarbeitung um einiges komfortabler. Die eingebauten Editoren sind meist dazu da, eventuell auftretende Fehler rasch zu beseitigen. Mit den Fehlern kommen wir zum zweiten großen Unterschied: Erst nach der vollständigen Eingabe wird das Programm compiliert. Das geschieht in den sogenannten Passes (Durchgänge). Beim ersten Paß wird die Syntax überprüft, im zweiten wird dann das Programm übersetzt, alle erforderlichen Tabellen errechnet und in das Programm eingebracht. Es gibt Compiler, die nur einen Durchlauf brauchen (zum Beispiel Turbo-Pascal). Andere können sogar auf vier Durchgänge kommen. Tritt während des Übersetzens ein Fehler auf, so wird das Compilieren abgebrochen und eine Meldung ausgegeben, die die Art des Fehlers und die Stelle, an der er auftrat, mitteilt. Nun wird der Fehler vom Programmierer korrigiert. Dieser Vorgang wiederholt sich so oft, bis das Programm fehlerfrei ist. Der Vorteil dabei ist, daß auch Programmteile, die selten aufgerufen werden, fehlerfrei sind und das Programm als solches von der Syntax her in Ordnung ist. Der Nachteil an der ganzen Sache ist, daß es oft sehr aufwendig sein kann, den Quelltext zu ändern. Dies trifft zum Glück nur noch für ältere Compiler zu; bei den neueren Versionen kann man teilweise bis zu 30 Kilobyte Quelltext auf einmal im Speicher halten und kann zusätzlich noch das Programm im Speicher compilieren lassen. Damit sind wir bei den verschiedenen Optionen, die so ein Compiler zu bieten hat.

Die verschiedenen Compiler-Optionen

Da wäre zum einen die Möglichkeit, das Programm fix und fertig auf die Diskette compilieren zu lassen. Man braucht es nur noch einzuladen und zu starten. Allerdings setzt der Compiler noch seine Run-Time-Routinen vor das Programm. Das ist sozusagen eine Bibliothek, die die Fehlermeldungen und einige wichtige Routinen (zum Beispiel schnelle Arithmetik oder Bildschirmverwaltung) beinhaltet. Dadurch wird das Programm je nach Compiler zwar um 4 bis 40 KByte länger, aber es ist absolut unabhängig vom Compiler lauffähig. Eine zweite Möglichkeit besteht darin, das Programm in den Speicher compilieren zu lassen, um es dort auszutesten und zu optimieren. Als drittes kann man das Programm ohne Run-Time-Routinen auf Diskette compilieren lassen, so daß es nur in Verbindung mit dem Compiler, quasi als Overlay, lauffähig ist.

Desweiteren bieten die meisten Compiler dem Programmierer sogenannte Switches; das sind Optionen, die im Quellcode eingestellt werden und die über Komfort und Schnelligkeit entscheiden. So kann man zum Beispiel Fehlermeldungen abfangen oder man kann entscheiden, ob Arrays möglichst schnell oder möglichst platzsparend behandelt werden sollen. Grundsätzlich ist das Arbeiten mit einem Compiler kein Kunststück, denn etwa die Hälfte der dem Markt erhältlichen Compiler sind menügesteuert und geben dem Bediener jederzeit Auskunft über noch verfügbaren Speicherplatz oder die Art des aufgetretenen Fehlers. Durch die vom Compiler unabhängige Erzeugung des Quellcodes erreicht man ein Höchstmaß an Eingabekomfort, denn welcher Editor ist schon so komfortabel wie ein Textverarbeitungssystem? Der größte Vorteil des Compilers einem Interpreter gegenüber ist aber die Zeitersparnis beim Programmablauf.

(U. Reetz/cg/dm)