Teil 1

Strubs — ein Precompiler für Basic-Programme

In dieser und den folgenden Ausgaben des 64'er wollen wir Ihnen ein Programm für den Commodore 64 und VC 20 mit dem Namen »Strubs« — das steht für »strukturiertes Basic« — vorstellen. Es handelt sich bei dem Programm um einen sogenannten Precompiler, ein Programm, welches Programmtexte mit gewissen zusätzlichen Befehlen in normale, auf jedem Commodore 64 oder VC 20 ablauffähigen Basic-Programm übersetzt.

In der ersten Folge wollen wir zunächst beschreiben, was solch ein Precompiler (= Vorübersetzer) eigentlich macht und dessen Methode mit der Arbeitsweise von Interpretern auf der einen und Compilern auf der anderen Seite vergleichen. Anschließend wird dann das Programm »Strubs« kurz vorgestellt. Weitere Teile werden die Grundlagen der strukturierten Programmierung und die praktische Entwicklung von eigenen Programmen mit Hilfe von Strubs behandeln. Zu guter Letzt ist noch ein Teil vorgesehen, in welchem genauer auf den Aufbau des Programms, auf Änderungs- und Erweiterungsmöglichkeiten eingegangen wird. Das Programm Strubs wurde ursprünglich zu einer Zeit entwickelt, als Begriffe wie Forth oder Pascal noch Fremdworte für den C 64 waren. Der Zweck war die Entwicklung von Programmen übersichtlicher, effizienter und bequemer zu gestalten.

Strubs bietet neue Basicbefehle

Auf der einen Seite ermöglicht es Strubs, auf sanftem Weg, das heißt im Rahmen des gewohnten Basic (aber ohne auf unübersichtliche Klimmzüge innerhalb des Commodore Basic angewiesen zu sein), also ohne gleich eine neue Programmiersprache lernen zu müssen, mit der Technik strukturierter Programmierung vertraut zu werden. Auf der anderen Seite ermöglicht es Strubs, sich mit der Arbeit mit Compilern vertraut zu werden.

Schließlich bietet die Form des Precompilers noch erhebliche Geschwindigkeitsvorteile gegenüber vergleichbaren Interpretererweiterungen. Um diese letzten beiden Punkte zu verstehen, ist es angebracht, auf die unterschiedlichen Arbeitsweisen von Interpretern und Compilern einzugehen.

Bekanntlich versteht der eigentliche Computer, das heißt hier der Mikroprozessor, nur die sogenannte Maschinensprache. Da diese aber extrem problemfern und unübersichtlich ist, hat man verschiedene höhere Programmiersprachen erfunden, um dem Programmierer seine Arbeit zu erleichtern. Damit aber ein in einer solchen Sprache geschriebenes Programm vom Computer verarbeitet werden kann, muß zunächst eine Übertragung in die Maschinensprache des Computers stattfinden. Dabei wird diese Übertragung wiederum von Programmen vorgenommen und zwar von Compilern oder von Interpretern. Diese beiden Programmarten unterscheiden sich grundlegend in ihrer Arbeitsweise.

Ein Interpreter besteht im wesentlichen aus einer Reihe von in Maschinensprache geschriebenen Unterprogrammen, einer Tabelle welche die erlaubten Befehle und die Adresse des zu jedem Befehl gehörenden Unterprogramms enthält, schließlich der Variablenverwaltung sowie der sogenannten Interpreterschleife.

Diese Schleife geht den Programmtext Schritt für Schritt durch. Zu jedem Befehl sucht sie in der Tabelle die zugehörige Unterprogrammadresse, ruft dieses Unterprogramm auf, holt den nächsten Befehl und so weiter, bis das Programmende erreicht ist. Man sieht also, daß ein großer Teil der Arbeit eines Interpreters im Suchen besteht: Suchen in der Befehlstabelle, Suchen in der Variablentabelle und nicht zuletzt Suchen nach Sprungzielen im, zu interpretierenden, Programm.

Diese ewige Sucherei führt nun dazu, daß Programme nur relativ langsam abgearbeitet werden. Eine Interpretererweiterung (wie etwa Siemens Basic) stellt nun einfach zusätzliche Befehlsroutinen zur Verfügung und erweitert die Befehlstabelle um die neuen Befehle und Adressen. Durch diese Erweiterung der Befehlstabellen wird jetzt aber leider auch der Zeitaufwand für das Suchen größer, so daß die Programme noch langsamer als bisher schon ablaufen. Simons Basic demonstriert dies sehr anschaulich. Ein Beispiel für eine Interpretererweiterung werden wir weiter unten besprechen.

Nehmen wir zur Illustration der Arbeitsweise eines Interpreters eine Programmzeile wie die folgende:

```
FOR I = 0 to 999: PRINT I: NEXT
```

Der Interpreter muß hier 1000-mal die Befehlstabellen nach dem Befehl PRINT und 1000mal die Variablentabelle nach der Variablen I durchsuchen.

Compiler kontra Interpreter

Völlig anders arbeitet der Compiler: Er übersetzt ein Programm, das in einer Sprache geschrieben ist, welche nur der Programmierer versteht — dieses Programm nennt man Quellprogramm — in ein äquivalentes Programm — das Objektprogramm —, das (meist nur noch) die Maschine versteht. Diese beiden Begriffe — Quellprogramm und Objektprogramm — sollten wir uns gut merken; sie werden noch öfter auftauchen.

Der größte Teil der Sucharbeit

kann nun ein für allemal bei der Übersetzung vom Compiler geleistet werden. Die benötigten Adressen der Befehlsroutinen, der Variablen und der Sprungziele sind für immer fest in das Objektprogramm eingebaut. Dadurch können compilierte Programme oft bis zu zehn- oder mehrmal schneller sein als entsprechende Interpreterprogramme.

Diesem beträchtlichen Gewinn an Geschwindigkeit steht allerdings ein nicht minder bedeutender Verlust an Bequemlichkeit gegenüber. Zum einen erfordert selbst die geringste Programmänderung eine vollständige Neuübersetzung des Programms. Dies allein kann bei umfangreichen Programmen erhebliche Zeit beanspruchen, zumal häufig auch noch diverse Zwischenschritte erforderlich sind, auf die wir hier nicht näher eingehen wollen. Zum anderen stellt das von einem einfachen Compiler erzeugte Objektprogramm für den Programmierer meist einen großen schwarzen Kasten dar, in den hineinzusehen ihm verwehrt bleibt. Er kann das Programm in der Regel nicht einfach unterbrechen, um sich bestimmte Variablenwerte anzusehen oder Variablen bestimmte Testwerte zuweisen, um damit dann einen kritischen Programmteil ausführen zu lassen, mal eben eine Zeile ändern und was der Annehmlichkeiten beim Programmtest mit einem Interpreter mehr sind. Bessere Compiler bieten zwar eine Reihe von Optionen und Hilfsprogrammen für die Fehlersuche und das Programmtesten an, jedoch bleibt auch hier, verglichen mit einem Interpreter, diese Arbeit reichlich unbequem. Ideal ist es sicherlich, äquivalente Interpreter und Compiler zur Verfügung zu haben. Auch gewisse Mischformen wie zum Beispiel bei der Sprache Forth sind hier interessant.

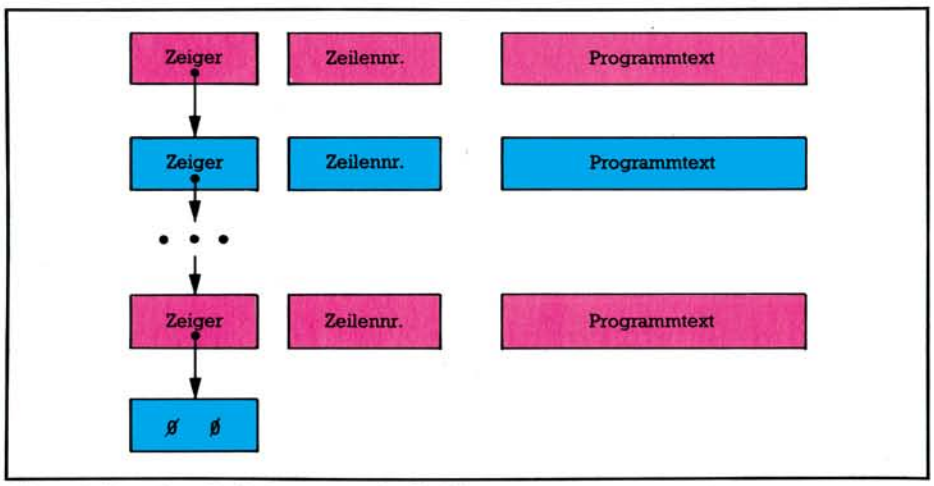

Bild 1. Aufbau eines Basic-Programms im Speicher

Strubs — eine Mischung von Interpreter und Compiler

Um nun aber auf das Programm Strubs zurückzukommen: Auch hier haben wir es in gewisser Hinsicht mit einer Mischform zu tun. Das selbst nicht lauffähige Quellprogramm, welches der Programmierer unter Benutzung der neuen Befehle erstellt, wird von Strubs nicht in Maschinensprache übersetzt, sondern in ein normales Basic-Programm, das dann wie bisher interpretiert wird. Dabei werden Programmteile, die keine Erweiterungen enthalten, mehr oder weniger unverändert übernommen. Dieses von Strubs erzeugte Objektprogramm kann nun wie jedes andere Basic-Programm — auch mit Hilfe von Toolkits — gelistet, ausgetestet und sogar geändert werden. Schließlich ist es dann noch möglich, dieses Objektprogramm mit Hilfe eines Basic-Compilers, wie zum Beispiel dem Austro Compiler, weiter zu übersetzen. Besonders hilfreich ist es, daß einander entsprechende Programmzeilen im Quellprogramm und im Objektprogramm gleiche Zeilennummern besitzen, so daß der Programmierer sich ohne Schwierigkeiten im Objektraum zurechtfinden kann. Gegenüber der Methode, den Basic-Interpreter zu erweitern, bietet dieses Verfahren Geschwindigkeitsvorteile: Diese ergeben sich einerseits aus der Tatsache, daß alle Kommentare und Leerzeichen gelöscht werden können, andererseits wird wie beim Compiler ein Teil der Sucharbeit während der Übersetzung erledigt. Dadurch werden zum Teil erst neue Anweisungen ermöglicht, deren Realisierung im Rahmen einer Interpretererweiterung zu aufwendig wäre.

Schon durch die Suche nach Sprungzielen wirkt der Basic-Interpreter langsam genug: Bei jedem Sprung wird das Programm von Anfang an durchsucht, bis die entsprechende Zeilennummer gefunden ist. Deshalb empfiehlt es sich auch, häufig aufgerufene Unterprogramme möglichst an den Programmanfang zu setzen, da sie dann schneller gefunden werden.

Wer sucht, der findet: aber wann?

Daß sich die Suchzeit überhaupt in erträglichen Grenzen hält, liegt nun daran, daß der Programmtext selbst nicht durchsucht werden muß. Vielmehr braucht der Interpreter nur entlang der Kette aus Zeilennummern und Zeigern zur nächsten Zeile zu suchen, bis die gewünschte Zeilennummer gefunden ist (Bild 1). Sollte nun der Interpreter aber bei nicht erfüllter Bedingung in einer IF-Anweisung das zugehörige ELSE suchen, bei nicht erfüllter Eingangsbedingung einer

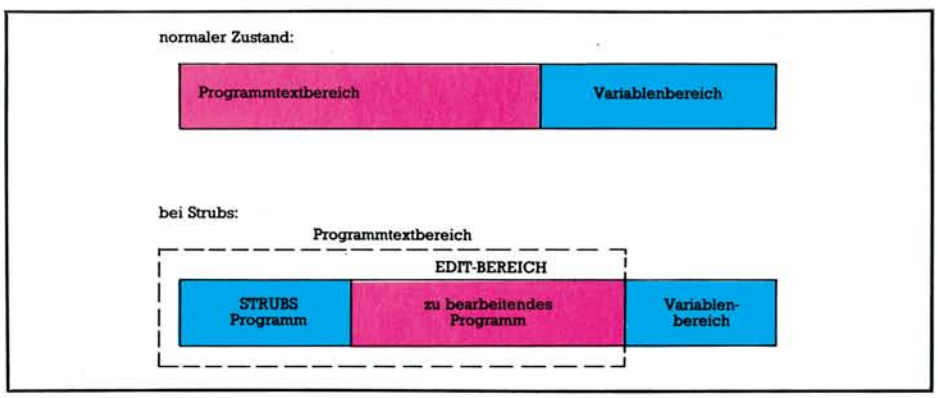

Bild 2. Aufteilung des Arbeitsspeichers

FOR-Schleife das zugehörige NEXT oder zu einem WHILE das END-WHILE, dann müßte der gesamte Programmtext selbst durchsucht werden.

Interpreter durchlaufen jede Schleife mindestens einmal

Deshalb arbeiten die Basic-Interpreter im allgemeinen so, daß solche Blöcke — wie die FOR-Schleife — mindestens einmal durchlaufen werden. Deshalb muß bei solchen Interpretern — sofern sie überhaupt ein ELSE kennen — dieses in der gleichen Programmzeile wie das zugehörige IF stehen. Deshalb kennt zum Beispiel Simons Basic die REPEAT-UNTIL-Anweisung, die immer mindestens einmal durchlaufen wird, nicht aber die WHILE-Anweisung. Ein Precompiler aber kann bei der Übersetzung den Abschlußbefehlen eines Blockes — wie ELSE oder END-WHILE — ihre Zeilennummern zuordnen, so daß beim Programmlauf nicht mehr der Programmtext selbst, sondern nur die Kette der Zeilennummern durchsucht werden muß.

Vorübersetzung nicht nur beim Precompiler

Die Methode der Vorübersetzung zur Erhöhung der Laufgeschwindigkeit benutzt im übrigen auch der Basic-Interpreter des Commodore 64. Und zwar findet sich die Übersetzungsfunktion im Editor: Sofort bei der Eingabe einer Zeile werden die Basic-Befehle, die aus mehreren Zeichen bestehen, in nur 1 Byte lange Zeichen, die sogenannten TOKENS, übersetzt. Eine Liste dieser Tokens findet sich zum Beispiel im Programmierhandbuch zum VC 20. Diese Vorübersetzung bringt zwar einen schönen Gewinn an Geschwindigkeit, hat allerdings den Nachteil, daß Programmtexte nicht mehr mit komfortableren Editor- beziehungsweise Textprogrammen erstellt werden können. Für uns ist jedoch vor allen Dingen wichtig, daß diese TOKENS berücksichtigt werden müssen, falls der Befehlsvorrat von Strubs erweitert werden soll, oder falls Programme für Interpretererweiterungen wie Simons Basic bearbeitet werden sollen. Aber auf diesen Punkt werden wir ein anderes Mal ausführlicher eingehen.

Wenn wir mit Strubs arbeiten, haben wir es — wie bei jedem Compiler — mit (mindestens) drei Programmen zu tun: Dem Übersetzungsprogramm, dem Quellcode (Quellprogramm) und dem lauffähigen Objektprogramm. Diese Programme müssen sich nun irgendwie den zur Verfügung stehenden Speicherplatz teilen. Daß das Übersetzungsprogramm, um arbeiten zu können, im Hauptspeicher stehen muß, versteht sich von selbst.

Eine Möglichkeit wäre nun, daß das Übersetzungsprogramm das Quellprogramm von der Diskette einliest, und gleichzeitig das erzeugte Objektprogramm auf Diskette schreibt. Der Compiler zu Simons Basic arbeitet zum Beispiel nach dieser Methode. Da ein Compiler jedoch einen Programmtext in der Regel mindestens zweimal durchliest — man spricht in diesem Fall von 2-Pass-Compilern —, ist es günstiger, wenn das Quellprogramm sich ebenfalls im Hauptspeicher befindet. Diesen Weg gehen zum Beispiel Pascal 64 und Strubs. Um nun den zur Verfügung stehenden Platz aufzuteilen, benutzt zum Beispiel Pascal 64 eine sehr einfache und wirksame Methode: Der Compiler ist selbst in Basic geschrieben und enthält eine unsichtbare Zeile mit der Zeilennummer 0, die ihrerseits einen Sprung zum Übersetzungsprogramm enthält. Ein Pascalprogramm wird nun einfach mit den Zeilennummern zwischen 1 und 9999 in das Compilerprogramm eingefügt.

Dieses Verfahren hat allerdings den Nachteil, daß Programme immer nur zusammen mit dem Compiler abgespeichert und editiert werden können. Insbesondere ist es damit nicht möglich, Quellprogramme aus fertigen Bausteinen (Modulen) zusammenzusetzen.

Strubs geht andere Wege

Aus diesem Grund wurde für Strubs ein anderer Weg gewählt: Entsprechend Bild 2 wurde der Speicher des Commodore 64 in drei Bereiche aufgeteilt. Am Anfang des Arbeitsspeichers steht das Programm Strubs (Pointer in Zelle 43/44). Der Edit-Bereich für Quellprogramme beginnt bei (Wert der Variablen EA). Daran anschließend befindet sich der (gemeinsame) Variablenbereich (Pointer in Zelle 45/46). Um nun vom Edit-Bereich aus bequem in den anderen Speicherbereich umschalten und die Übersetzung starten zu können, benutzt Strubs selbst eine kleine Interpretererweiterung, die, wie versprochen, kurz vorgestellt werden soll.

Die Eingabe von »!« bewirkt nun dasselbe wie die Befehlsfolge »POKE 44,8: RUN«. Das entsprechende Assemblerlisting findet sich in Bild 3. Das kleine Programm »Erweiterung« holt zunächst den nächsten Befehl. Dann muß für die Routine »Befehl ausführen« der Status gerettet werden, da die CHARGET-Rou-

```
ERWEITERUNG:
02C0    207300    JSR 0073    ; Charget, nächstes Zeichen holen
02C3    08        PHP         ; Status retten
02C4    C921      CMP __21    ; »!«, neuer Befehl?
02C6    F004      BEQ 02CC
02C8    28        PLP         ; nein, dann Status wiederherstellen
02C9    4CE7A7    JMP A7E7    ; und normalen Befehl ausführen
02CC    28        PLP         ;
02CD    A908      LDA __08    ; Erweiterungsroutine:
02CF    852C      STA 2C      ; entspricht Poke 44,8: RUN
02D1    A98A      LDA __8A    ; RUN-TOKEN
02D3    4CE7A7    JMP A7E7    ; Befehl ausführen

INIT:
02EE    A9C0      LDA __C0    ; Erweiterung, Low Byte
02F0    8D0803    STA 0308
02F3    A902      LDA __02    ; Erweiterung, High Byte
02F5    8D0903    STA 0309
02F8    60        RTS
```

Bild 3. Interpretererweiterung

tine damit wichtige Informationen übermittelt. (Dies ist wichtig und wurde in dem unten erwähnten Buch übersehen.) Nachdem verglichen wurde, ob ein neuer Befehl vorliegt, wird dann entsprechend zum normalen Programmverlauf oder zur Erweiterungsroutine verzweigt. Für eigene Versuche mit Interpretererweiterungen können an dieser Stelle beliebige Maschinenprogramme (gegebenenfalls mit weiteren Decodierungen) gesetzt werden. Nur sollte zum Abschluß — anders als hier, wo ein Basic-Befehl aufgerufen wird — ein Sprung zur Interpreterroutine ² A7E4 erfolgen, wo dann der nächste Befehl bearbeitet wird. Um nun die Erweiterung in den Basic-Interpreter einzubinden, benötigen wir dann nur noch eine kurze Initialisierungsroutine, die den Zeiger in ² 0308 auf den Anfang der Erweiterung setzt.

Wer selbst solche Erweiterungen entwickeln möchte, findet weitere Informationen und viele Anregungen in dem Buch »64 Intern« von Data Becker. Für weitergehend Interessierte empfiehlt sich die gut verständliche Einführung »Compilerbau« von N. Wirth, Teubner, Stuttgart 1981.

Strubs stellt sich vor

Abschließend wollen wir nun das Programm Strubs kurz vorstellen. Am Anfang der Programmentwicklung standen folgende Vorstellungen, die durch das Programm erfüllt werden sollten:

1. Unabhängigkeit von Zeilennummern
2. Unterstützung strukturierter Programmierung
3. Unterstützung modularer Programmentwicklung
4. Erweiterung der Dokumentationsfähigkeit des Programmtextes.

Dabei sollte das Programm

5. einfache Handhabung gewährleisten und
6. effiziente Fehlersuche ermöglichen.

Die Unabhängigkeit von Zeilennummern wird erreicht durch die Verwendung beliebig langer Labels oder relativer Sprünge anstelle von Zeilennummern.

Die wichtigsten Kontrollstrukturen höherer Programmiersprachen werden von Strubs zur Verfügung gestellt:

IF — THEN — FI
IF — THEN — ELSE — FI
WHILE — EWHILE
REPEAT — UNTIL
LOOP — EXIT (beliebig oft) — ELOOP
CASEOF — OF (beliebig oft) — ELSE (optional) — ECASE

Durch die Unabhängigkeit von Zeilennummern und eine EXTERN-DECLARATION wird das Anlegen einer Modulbibliothek — sowohl auf Quellprogramm- als auch auf Objektprogrammebene — unterstützt.

Mit Strubs werden Sie ein vielseitiges Werkzeug in Händen halten

Der Dokumentationsfähigkeit des Programmtextes dienen neben den bereits erwähnten Marken und Kontrollstrukturen ein Tabulator und Kommentare an beliebiger Stelle auch innerhalb einer Zeile, ja selbst innerhalb eines Variablennamens (zum Beispiel A'US'G'ABE'% = AG%).

Programmtexte können wie gewohnt mit dem normalen Basic-Editor geschrieben werden.

Schließlich werden wir zur Illustration der Erweiterung des Befehlssatzes von Strubs noch eine MAKRO-Funktion implementieren. Von besonderer Bedeutung ist, daß das Programm von Anfang an unter dem Aspekt möglichst einfacher Erweiterbarkeit konzipiert wurde. Damit konnte das Programm im Bootstrapping-Verfahren entwickelt werden, so daß es jetzt selbst sowohl als Quellprogramm als auch als Objektprogramm vorliegt. Wem es Spaß macht, der mag Strubs einfach auch als ein generelles Übersetzungsprogramm zur Aufbereitung von Programmtexten auffassen und seine gegenwärtigen Features als Beispiel möglicher Implementationen. In der nächsten Ausgabe werden wir das komplette Objektprogramm von Strubs abdrucken. (Matthias Törk)