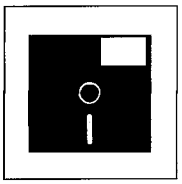


Viele Anwendungen von Juristen für Juristen sind in Clipper geschrieben worden. Zahlreiche in dBase konzipierte Programme dieser Art wurden mit Clipper für die Verteilung kompiliert. Wenn ein Programm eine derartige Verbreitung erreicht hat, rechtfertigt das auch einen etwas technischeren Blick auf die jeweiligen Entwicklungszustände. Der folgende Beitrag analysiert aus der Sicht des Programmierers die wichtigsten, mit Clipper Version 5.0 eingeführten Neuerungen. Für die Planung und Evaluation von Projekten ergeben sich aus dieser Analyse wichtige Beurteilungskriterien. (red.)

Offene Architektur: Clipper 5.0

Michael M. Zurek

Einführung: Interpreter und Compiler

Interpretieren

Ein mit dBASE entwickeltes Programm gelangt erst dann zur Ausführung, wenn es in einen dem Rechner verständlichen Code umgewandelt worden ist. Hierfür gibt es prinzipiell zwei Wege: Entweder geschieht die Übersetzung mittels eines Interpreters oder eines Compilers. Ein Interpreter (wie z.B. in dBASE enthalten) ist ein interaktives Programm, das vom Betriebssystem aus gestartet wird. Der Interpreter übersetzt das Programm in maschinenlesbare Form. Der Nachteil dieser Methode besteht darin, daß mit jedem erneuten Programmstart eine solche Übersetzung stattfindet, was die Ausführungsgeschwindigkeit erheblich reduziert.

Kompilieren

Im Gegensatz zum Interpreter übersetzt ein Compiler ein Programm nur ein einziges Mal in eine maschinenausführbare Form. Dieses compilierte Programm wird in einer Objekt-Datei (.OBJ) abgespeichert. Im letzten Schritt wird diese Objekt-Datei (von der es auch mehrere geben kann) zusammen mit einer (oder mehreren) sogenannten Bibliothek(en) mithilfe eines Linkers zu einer selbständig ausführbaren ".EXE-Datei" zusammengeführt. Diese ".EXE-Dateien" können direkt aus der Betriebssystem-Shell COMMAND.COM heraus ausgeführt werden. Für den Aufruf dieser Programme benötigt man jetzt weder einen Compiler noch dBASE. Darin liegt der entscheidende Vorteil eines Compilers gegenüber einem Interpreter.

Clipper 5.0-Funktionalität: Ein Überblick

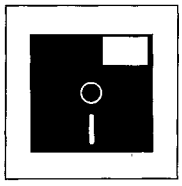
Einer der am weitesten verbreiteten dBASE-Compiler kommt aus dem Hause Nantucket: Clipper 5.0. Er bietet eine offene Architektur, mit der Anwender eigene Befehle definieren können. Im Unterschied zu den Vorversionen verfügt Clipper 5.0 über austauschbare Datenbanktreiber (Datenbank-Engines), eine verbesserte Speicherverwaltung, eine völlig überarbeitete Variablenverwaltung, zahlreiche neue Funktionen und eine, an die Programmiersprache C angelehnte Syntax. Außerdem gehören ein Editor, eine Online-Dokumentation, ein neuer Linker, ein neuer Quellcode-Debugger zum Lieferumfang. Dabei bleibt die jetzige Version mit Hilfe des mitgelieferten Pre-Prozessors kompatibel zu den Vorversionen. Eine vollständig objektorientierte Programmierung ("OOP") sowie die Unterstützung der graphischen Benutzeroberfläche von Windows wurde jedoch nicht integriert.

Clipper - dBASE: Ein (Kurz-) Vergleich

Eine ältere Verwandtschaft

Hauptsächlich erinnern die typischen dBASE-Strukturen in der Architektur von Clipper 5.0 an das "vergangene" Verwandtschaftsverhältnis. Denn mittlerweile hat sich Clipper vom dBASE-Sprachumfang emanzipiert und bietet zum Teil eigenständige Leistungen. (Daher ist grundsätzlich eine Kompatibilität zu dBASE IV nicht gegeben). Dies impliziert, daß Clipper 5.0 über die Funktion eines reinen Compilers weit hinausgeht. So können in Clipper beispielsweise mehr Variablen (dBASE III: 256; Clipper: 2048) und größere Variablen (dBASE III: je 255 Byte; Clipper: je 64 KByte) verwendet werden. Clipper bietet die Möglichkeit in 1.024 Felder mit je 32.000 Zeichen zu speichern. Dagegen kann dBASE IV lediglich in 256 Feldern je 255 Zeichen speichern.

Michael M. Zurek (Berlin) ist Verfasser des (in Clipper geschriebenen) juristischen Shareware-Programms TENSION II (Diskettenbeilage zu jur-pc 1/1991) und Autor eines Anwaltssystems.



Die Clipper 5.0-Sprache: Einige Details

Die Affinität zur C-Syntax

#include, #define, ++, :=

Auffallend ist eine partielle Übereinstimmung mit der C-Syntax, wie beispielsweise in den Anweisungen "#include" und "#define", neuen Operatoren wie "++" und ":=" und den Kommentarzeichen. Selbst die in der C-Syntax gewohnten Abkürzungen bei Rechenausdrücken (zusammengesetzte Operatoren) können in Clipper 5.0 verwendet werden. Zum Beispiel läßt sich die Anweisung x=x+2 jetzt auch so formulieren: x+=2. Sollen Variablen lediglich um den Wert 1 erhöht werden, so ersetzt ein kurzes x++ den Ausdruck x=x+1. Wichtig in diesem Zusammenhang ist auch, daß die von C gewohnten Post- und Pre-Inkrement- beziehungsweise -Dekrement-Operatoren implementiert sind. So hat etwa der Ausdruck a=x++ zur Folge, daß a der Wert von x zugeordnet wird und x erst anschließend inkrementiert wird. Dementsprechend wirkt a=++x wie bisher: a wird der Wert des um 1 erhöhten x zugewiesen. Wer sich mit dieser Notation nicht anfreunden will, kann seinen bisherigen Stil beibehalten: Die gewohnte Syntax wurde vollständig übernommen.

Der Pre-Prozessor

Auf der Suche nach UDCs

Der Pre-Prozessor wird als eigenständiges Programm mitgeliefert. Sein Nutzen liegt darin, daß der Anwender eigene Kommandos definieren kann. Der Pre-Prozessor durchsucht den Quellcode nach diesen speziellen Direktiven (UDCs = user defined commands) und übersetzt sie in einen gültigen Clipper 5.0-Quellcode. Das Ergebnis dieses Vorganges ist wieder eine reine Textdatei, die anschließend vom Compiler compiliert werden kann.
Der Pre-Prozessor übernimmt insgesamt folgende Funktionen: Er ermöglicht zum einen eine automatische Bearbeitung von Textmacros, zum anderen eine bedingte Compilierung von Programmen, und schließlich die Aufnahme von zusätzlichen Textmodulen (Include-Dateien) in den Quellcode.
Ein Manko der bisherigen Clipper-Versionen war, daß die Clipper-Sprache nicht vollständig zu dBASE III Plus kompatibel war. So waren Umsteiger von dBASE III Plus zu Clipper gezwungen, manchmal den unter dBASE III Plus erstellten Quellcode zu modifizieren. Befehle, die in Clipper nicht implementiert waren, mußten erst durch andere Code-Sequenzen ersetzt werden. Das ist zwar immer noch nötig, allerdings nur noch ein einziges Mal, wie die folgenden Beispiele zeigen.

Beispiel 1: Browse - DBedit

Wie man "Browse" nachbildet

Der in dBASE existierende Befehl "Browse" ist in Clipper nicht bekannt. Stattdessen gibt es dort die alternative "DBedit()"- Funktion. Mittels des Command-Schlüsselwortes wird nun der Befehl Browse angelegt. Die dahinterstehende Funktion DBedit() wird mit Hilfe des "daraus-folgt"-Pfeils = in folgender Form zugewiesen:

```
#command  BROWSE  =  DBedit()
```

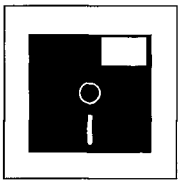
Beispiel 2: Schlüsselworte der alten Clipper-Version

Sicherung der "Abwärtskompatibilität"

Komplexere Strukturen, etwa mit Parameterübergabe, lassen sich ebenfalls realisieren:

```
#command  DEFINE  WINDOW  ,,,  = WOPEN(,,,)
oder
#command  ACTIVATE  WINDOW    =  WSELECT().
```

Mit der "Command"-Direktive lassen sich so die verschiedensten dBASE-Derivate emulieren. Diese Möglichkeit war der einzige Weg, um den Sprachumfang der Vorgängerversion Clipper (Sommer '87) vollständig zu unterstützen. Der komplette 87er Befehlssatz ist also als Pre-Prozessor-Anweisung implementiert. Um sich diese "Batch-Dateien" zugänglich zu machen, ist das Pre-Prozessor-Befehlswort "#include" eingeführt worden. Mit der #include-Anweisung lassen sich fertige Definitionen mit einer einzigen Zeile im Quelltext integrieren. So bewirkt #include , daß alle Schlüsselworte der letzten Clipper-Version richtig interpretiert werden. Die dafür nötigen Command-Statements sind in der Datei STD.CH enthalten. Include-Dateien lassen sich beim Compiler-Start mit angeben.



Auf Tastendrücke reagieren

Beispiel 3: Eigene Definitionen des Anwenders

Ein weiteres interessantes Schlüsselwort ist "#define". Wie auch in C weist der Programmierer damit einen nichtssagenden numerischen Wert einem symbolischen Ausdruck zu. Jeder dBASE- oder Clipper-Entwickler kennt eine Programmzeile wie die folgende:

```
if  lastkey()=27
```

Wenn also die Taste 27 betätigt wird, folgt eine Aktion. Nun bringt nicht jeder den Wert 27 auf Anhieb mit der Escape-Taste in Verbindung. Daher wäre wohl die bessere Schreibweise:

```
if  lastkey()=ESC
```

Die #define-Direktive erlaubt diese Abstraktion. In einer eigens erstellten Include-Datei kann nämlich folgende Zeile eingefügt werden:

```
#define  27  ESC
```

Damit ersetzt der Pre-Prozessor alle 27 durch ESC.

Schutz für Originalvariablen

"You can call **me** by value"

In Clipper 5.0 gibt es jetzt auch die Möglichkeit der Parameterübergabe per "value", das heißt, es wird nur der Wert der Variablen übergeben. Ein versehentlicher Zugriff auf die Originalvariable ist dabei nicht möglich. Hilfreich ist dies vor allem bei allgemeingültigen Programmteilen, da die Entscheidung, ob ein Variableninhalt verändert werden darf, ausschließlich beim aufrufenden Programmteil liegt. Muß dennoch innerhalb einer Prozedur oder Funktion auf eine Originalvariable zugegriffen werden, dann genügt es, beim Aufruf der betreffenden Variablen ein "§" voranzustellen. Wie bisher übergibt man für Prozeduren die Variablen per Aufruf mit "do" und "with" per Referenz. Die Übernahme der Parameter erfolgt nicht mehr mit dem Parameter- Befehl, sondern direkt als Liste von Argumenten. Ein Beispiel dieser Form der Übergabe per "value" lautet - inklusive dem in der neuen Version möglichen "Parameterskipping":

```
? Zeige ("Test",,,"Ende")
```

Die zugehörige Prozedur hat folgenden Code:

```
Procedure Zeige (Para1, Para2, Para3, Para4, Para5) ? Para1, Para2, Para3, Para4, Para5 Return
```

Als Ergebnis wird folgende Zeile angezeigt:

```
Test NIL NIL Ende NIL
```

Eine Kombination der Übergabeverfahren per "Referenz" und per "Value" ist nicht möglich.

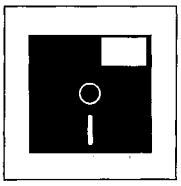
Kommentar mit //oder / ... */*

Vorteile des neuen Kommentarzeichens

Das neue Kommentarzeichen "//" ersetzt die bisherigen Symbole "*" und "&&". Es ist vielseitiger einsetzbar, da es sowohl am Anfang der Zeile wie auch in der Mitte stehen kann. Das Paar "/*" und "*/" wandelt alles, was zwischen diesen beiden Symbolen steht, zum Kommentar. Dies hat den Vorteil, daß man zu Testzwecken ganze Programmteile stillegen kann, ohne daß man an den Anfang einer jeden Programmzeile einen * (= Asterisk) setzen muß.

Variablen-Typen: Static, Private, Public, Local

Einerseits soll durch Clipper 5.0 die Variablenverwaltung deutlich verbessert werden. Andererseits müssen aus Gründen der Kompatibilität zu älteren Versionen die bereits bestehenden Variablen beibehalten werden. Man hat sich deshalb zu dem folgenden Kompromiß entschlossen: Mit der Deklaration "Static" wird einer Variablen ein fester Wert zugewiesen, der ab Zuweisungszeitpunkt gilt und den der Compiler bereits bei der Compilierung berücksichtigt. Wird eine Variable innerhalb einer Prozedur als "Static" deklariert, dann gilt sie nur innerhalb dieser Prozedur. Wird sie dagegen außerhalb einer Prozedur deklariert, gilt sie innerhalb der ganzen PRG-Datei.



Praktischer Variablen-Typ: Local

Die bisherigen Deklarationen "Private" und "Public" wurden durch einen dritten Typ "Local" ergänzt. "Private"-Variablen haben nämlich den Nachteil, daß sie ab dem Deklarationszeitpunkt in allen aufgerufenen Programmteilen gültig sind, und zwar so lange, bis die Programmebene der Deklaration nach oben verlassen wird. Noch mehr Kontrolle über die Variablen erhält man, wenn man die Bezeichner "Field-" für eine Feldvariable und "Memvar-" für eine Speichervariable voranstellt.

Multidimensionale Arrays

Die Größe der Zeichenvariablen wurde auf maximal 65520 Zeichen erweitert. Überarbeitet wurde auch die Verwaltung der "Arrays". Deren Multidimensionalität wird nur noch durch die Fähigkeiten des Programmierers begrenzt. Auch die Zuweisung eines kompletten Arrays als Element eines anderen Arrays bereitet allenfalls dem Programmierer Probleme, nicht aber Clipper.

Die Kompatibilitätsbefehle

In Zukunft nicht mehr empfohlen ...

Wer beim Umstieg auf die neue Version seine schon erstellten Programme verwenden möchte, wird im Handbuch darauf hingewiesen, daß dies in der vorliegenden Version noch möglich ist. Künftige Clipper-Versionen unterstützen nicht mehr alle von früheren Clipper-Versionen her bekannten Befehle. Die davon betroffenen Befehle werden daher als "Kompatibilitätsbefehle" bezeichnet und sind im Handbuch besonders gekennzeichnet. Dies bedeutet, daß diese Befehle für neue Applikationen nicht mehr empfohlen werden. Es werden allerdings Alternativen vorgeschlagen, die die Kompatibilität zu künftigen Clipper-Versionen sichern sollen.
"Kompatibilitätsbefehle" sind:

CALL, CANCEL, CLEAR ALL, DIR, FIND, NOTE, RESTORE SCREEN, SAVE SCREEN, SET COLOR, SET EXACT, SET EXCLUSIVE, SET FORMAT, SET PROCEDURE, SET UNIQUE, STORE, TEXT und WAIT.

Die Standard-Header-Datei

Eine weitere Verbesserung ist der interne Aufbau von Clipper 5.0. Ähnlich wie bei der Programmiersprache C liegen die Clipper Standard-Befehle als Textdatei im Source-Code vor. Während des Compilierens liest der Compiler diese Definitionen aus der Textdatei "STD.CH". Ist man mit der Ausführung eines Befehls, etwa der Fehlerbehandlung, nicht einverstanden, dann sind die betroffenen Stellen im Source-Code zu verändern. Da solche Änderungen u.U. kritisch sind, kann man beim Compiler-Aufruf per Schalter den Compiler anweisen, auf eine Kopie dieser Standard-Header-Datei zuzugreifen.

Erweiterungen, Tools, Optimierungen, Hilfen

Die Arbeitsbereiche

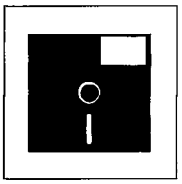
Nunmehr: 250 Arbeitsbereiche mit je 15 Indexdateien

Eine wichtige Erweiterung ist bei den Arbeitsbereichen zu verzeichnen: Standen bislang zehn Arbeitsbereiche mit maximal je sieben Indexdateien, zusammen aber höchstens fünfzehn gleichzeitig geöffnete Dateien zu Verfügung, so lassen die nunmehr möglichen 250 Arbeitsbereiche mit je 15 geöffneten Indexdateien die meisten Beschränkungen Vergangenheit werden. Für den Wechsel der Arbeitsbereiche empfiehlt das Handbuch, statt der Bereichsnummer nur den Aliasnamen anzugeben und beim Öffnen von neuen Dateien den zusätzlichen Parameter "New" anzugeben. Clipper öffnet die Datei dann automatisch im nächsten freien Arbeitsbereich.

Die Datenbank-Engines

Clipper als Datenbank-Toolkit

Eine weitere grundlegende Neuerung sind die austauschbaren "Datenbank-Engines". Dies ist eine Folge der Strategie des Hauses Nantucket, die dem Clipper-Compiler den Ruf des bloßen dBASE- Compilers nehmen soll. Daher wird zum einen dBASE IV grundsätzlich weder von der Syntax noch von der Dateistruktur unterstützt. Andererseits gibt es aber die Möglichkeit, jedes Dateiformat zu verwalten und Befehle selbst zu definieren. Daraus folgt, daß man Clipper 5.0 als Toolkit verwenden kann um einen dBASE IV-Compiler zu erstellen oder sogar Clones von dBASE IV oder FoxPro zu programmieren. Intern bleibt Clipper 5.0 dem dBASE-Sprachumfang sehr nahe. Die Syntax der Befehle basiert auf dBASE III, die Speichervariablen entsprechen weitgehend denen von dBASE III und das standardmäßig unterstützte Datenformat ist das von dBASE III Plus.



Dynamische Overlays

Der Linker .RTLink

Eine der gravierensten Neuerungen für Entwickler dürfte der neue Clipper-Linker ".RTLink" sein. Er kennt dynamische Overlays und erübrigt die bisherige Verwaltung von externen Overlays. Die Applikationen können nun größer sein als freier Speicherplatz im Arbeitsspeicher des jeweiligen Computers zur Verfügung steht. Dieser Linker fügt Overlays zu einem Overlayblock zusammen, der zum Kernprogramm in der EXE-Datei gefügt wird. Das fertige Programm besteht damit aus einer Vielzahl kleinerer Overlays. Beim Programmlauf wird dann neben dem Kernprogramm nur noch der Teil geladen, der aktuell benötigt wird. Je nach verfügbaren Speicherplatz werden öfter aufgerufene Programmteile im Arbeitsspeicher gehalten, so daß bei der praktischen Arbeit keine wesentlichen Verzögerungen durch das Nachladen anfallen. Für eine höhere Effizienz werden aber nie ganze Programmteile, sondern nur Teile davon geladen bzw. überschrieben. Dies ist das sogenannte "Paging-Ladeverfahren". Hierdurch wird das zeitraubende Konzipieren einer Overlay-Struktur weitgehend entbehrlich. Die einzige Ausnahme bilden die statischen Overlays, die noch für Objekt-Dateien von anderen Programmiersprachen benötigt werden, denn es ist nicht möglich C- oder Assembler-Module als dynamische Overlays einzubinden. Der neue Linker bietet gegenüber dem Linker der Vorgängerversion noch einen weiteren Pluspunkt. Er ist nämlich ein sogenannter "Incremental Linker": Nach einem einmaligen, kompletten Link-Lauf erzeugt der Linker eine Liste, die er für spätere Läufe heranzieht. Mit Hilfe dieser Liste stellt er fest, an welchen Stellen des ausführbaren Programms sich tatsächlich Änderungen ergeben. Nur an diesen Stellen wird der Code modifiziert. Dies erklärt, daß selbst bei großen Applikationen das Linken nur einige Sekunden dauert. Weiterhin erlaubt der Linker eine Anpassung des Codes in der Weise, daß die EXE-Dateien auf sogenannte "prelink libraries" (.PLL-Dateien) zugreifen. Der Anwender kann also schon bestehende Libraries in "vorgelinkte" umwandeln und diese während der Laufzeit dem Programm zur Verfügung stellen. Die Konsequenz ist, daß der Link- Lauf noch schneller vor sich geht, die resultierenden Dateien relativ klein sind und mehrere Programme gleichzeitig auf die "prelink-libraries" zugreifen, was sich zusätzlich platzsparend auswirkt.

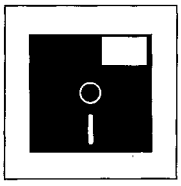
Der Source-Level-Debugger

Alle Programmzustände auf einen Blick

Die Programmierung unter Clipper wird nun durch den sogenannten "Source-Level-Debugger" erleichtert. "Source-Level" ist in etwa mit Quellcode gleichzusetzen. Dadurch ist die Aufgabe dieses Debuggers klar vorgegeben. In der 87er-Version mußte man den Debugger noch als Objekt-Datei hinzulinken. Beim Programmlauf verwies er dann meist nur auf eine fehlerhafte Zeile hin. Nun ist bei Clipper 5.0 ein Debugger vorhanden, der dem Anwender bei der Fehlerbeseitigung eine weitaus größere Hilfestellung leistet. Der Debugger läßt sich als Library integrieren oder er startet das zu untersuchende Programm als ein ausführbares. Damit wird das sogenannte "Real-Time-Debugging" möglich. In der unteren Bildschirmhälfte des Debuggers ist eine Befehlszeile zu sehen, die die Aufnahme von Steuerkommandos erlaubt. Für den weniger Geübten stehen Pulldown-Menüs zur Verfügung. In der Bildschirmmitte wird der momentan zu bearbeitende Quellcode angezeigt. Darüber sind die Variablen mit ihrem Inhalt zu sehen. Läßt man ein Programm im Einzelschrittmodus laufen, so hat man stets die aktuelle Variablenbelegung vor Augen. Die Ergebnisse von bedingten Ausdrücken erfährt der Anwender ebenfalls, da nach deren Ausführung sofort die Zeile bearbeitet wird, die dem Ausdruck zugeordnet ist. Zudem gestattet der Debugger den direkten Zugriff auf Variablen, die sich somit auch während der Programmausführung ändern lassen.

Mit VGA: 25 Zeilen Programm, 25 Zeilen Debugger

Besitzern einer VGA-Grafikkarte mit passendem Monitor bietet der Debugger mit dem S-Schalter eine zusätzliche Möglichkeit. Das S steht für "Shared Screen", also geteilter Bildschirm. In diesem Anzeigemodus versucht der Debugger, in den 50-Zeilen-Modus zu wechseln. Hierin dienen die ersten 25 Zeilen zur Anzeige der Programmausgaben, während die übrigen 25 Zeilen dem Debugger vorbehalten sind. Der Programmierer steuert also im unteren Bildschirmbereich den Debugger, während er im oberen Teil die Ausgaben des Programmes überwacht.



Eine "Wissensbank" als Teil der Arbeitsumgebung

Die Norton-Guides
In Clipper 5.0 findet man nun auch eine On-Line-Dokumentation; die "Norton Guides" (The Guide to Clipper). Ohne jedesmal das Handbuch zur Hand nehmen zu müssen, kann man auf Tastendruck Erklärungen in deutscher Sprache für alle Befehle, Funktionen, Direktiven und sonstigen Programmaufrufen abrufen. Auf diese "Wissensbank" wird über ein speicherresidentes Programm, die "Norton Instant Access Engine", zugegriffen.

Die Optimierung der EXE-Dateien
Im praktischen Einsatz können bestehende Clipper-Programme ohne Änderungen mit dem neuen Clipper 5.0 compiliert werden, lediglich die bestehenden MAK- und LNK-Dateien müssen geändert werden. Clipper 5.0 verfügt über ein neues RMAKE-Utility mit einer abweichenden Syntax, und die LNK-Anweisungen müssen keine Overlay-Anweisungen mehr enthalten.

Ein Beispiel: 91 K gespart.

Ein Programm, das zuvor 393 KByte EXE-Größe und 21 Overlays mit zusammen 128 KByte hatte, war nach dem compilieren mit Clipper 5.0 nur noch 430 KByte groß. Diese neue EXE-Datei lief problemlos und vor allen Dingen schneller als zuvor.
Ein Programm wie etwa:

```
? "Dies ist ein Test"
```

das unter der alten Clipper-Version noch eine EXE-Größe von etwa 162 KByte hatte, ist unter Clipper 5.0 nur noch 134 KByte groß. Dies ist ein Indiz dafür, daß der Compiler den endgültigen Code effektiver zusammenstellt als es bisher der Fall war.

Der Programmeditor PE.EXE
Clipper 5.0 wurde ein vollständig in Clipper geschriebener Programmeditor hinzugefügt. Der Quellcode dieses Programmes wird, wie auch der aller anderen in Clipper geschriebenen Hilfsprogramme, mit dem Clipper-System geliefert. Mit Hilfe diese Editors kann man kleinere Quellprogramme (.PRG) erstellen. Er ist aber keinesfalls geeignet, professionelle Editoren zu ersetzen.

Die Dokumentation
Mit dem Programm werden sieben umfangreiche Handbücher in deutscher Sprache mitgeliefert. Der Einsteiger wird gerade noch die Einführung verstehen, wird aber an den anderen Dokumentationen scheitern, da sich diese in erster Linie an den versierten Anwender richten. Inhaltlich werden alle Programmbestandteile mit der nötigen Ausführlichkeit beschrieben. Allerdings vermißt man eine größere Zahl von Programm-Beispielen.

Gute Dokumentation der Fehlermeldungen

Alle eventuellen Fehler werden mit Fehlercode und entsprechender Fehlermeldung erläutert. Die Dokumentation hält zudem eine genaue Beschreibung der Fehlerursache und Lösungsvorschläge parat.

Fazit

Los von dBASE zu neuen Ufern?

Zusammenfassend läßt sich sagen, daß Clipper 5.0 noch besser und schneller geworden ist. Er bleibt der beste dBASE III-Compiler, auch wenn die Herstellerfirma Nantucket dieses Image ändern möchte. Die Programmiersprache überzeugt ebenfalls, ohne aber mit fortgeschrittenen Eigenschaften anderer Hochsprachen aufzuwarten. Mit dem Pre-Prozessor ist Clipper 5.0 sinnvoll erweitert worden. Obwohl er nichts als ein einfacher Textersetzer ist, vereinfacht er durch die Aufnahme symbolischer Konstanten die Lesbarkeit und Wartung eines Programmes. Clipper 5.0 fordert vom Anwender immer noch ein gewisses Geschick und die Bereitschaft, sich genauestens mit dem Befehlsumfang auseinanderzusetzen. Die Hilfsmittel, die das Programm bereithält, sind nicht gerade zahlreich. Dies ist eine Folge der relativ unkomfortablen Entwicklungsumgebung. Versierte Anwender bekommen aber mit Clipper 5.0 eine überaus leistungsfähige Programmiersprache.