CHIP PROGRAMMIER-PRAXIS 84029/92001
29,– DM

Sources
auf Diskette
im Heft

Ausgabe 19

PASCAL

Turbo-Pascal

- Optimierung: Schnelle Assemblerroutinen
- Erfahrungsbericht: Stony-Brook-Pascal 6.0
- Toolbox: Netzwerk-Programmierung
- Grafik: Fenster und Sprites
- Grundlagen: Kopierschutz für eigene Programme
- Einsteigerseiten: Matrizenberechnung
- TP für Profis: Kampf den Speicherfressern

Markus Mück

Programme vor illegaler Verbreitung schützen

Wahrscheinlich hatten Sie schon irgendwann einmal mit einem Kopierschutz zu tun; sei es, daß Sie das neueste Spiel auf Ihrer Festplatte installieren oder einfach eine Sicherheitskopie eines teuren Textverarbeitungsprogrammes anlegen wollten. Früher oder später stellten Sie jedoch fest, daß eine oder mehrere Disketten nicht kopierbar waren. Sicherlich waren Sie in keinster Weise mit dem Vorgehen der Firmen einverstanden, ihre Programme auf diese Weise zu schützen.

Alle zu diesem Beitrag gehörenden Files finden Sie im SubDir PROTECT auf beiliegender Diskette

Haben die Firmen aber nicht recht, mit allen Mitteln eine illegale, für Sie gewinnlose Verbreitung eines teuren Produktes zu verhindern - schließlich stecken in einem solchen Programm viele Jahre Arbeit? Diese Frage existierte schon vor dem ersten Kopierschutz und noch keinem ist es gelungen, sie zur Zufriedenheit aller zu beantworten.

Doch wenn die großen Firmen Ihre Programme vor einer illegalen Verbreitung zu bewahren versuchen, warum soll dies dann nicht auch der weniger betuchte Programmierer tun? Natürlich ist es ihm gestattet, doch oft fehlt es am Know-how, einen Anwender "überreden" zu können, ein Programm nicht weiterzukopieren. Andere interessieren sich einfach dafür, wie ein Kopierschutz aufgebaut ist, auch wenn Sie ihn nicht bei eigenen Programmen anwenden wollen. Zu welcher Gruppe Sie nun auch gehören, die nächsten Zeilen werden Sie in die interessante Welt der Kopierschutzverfahren entführen.

Die geläufigsten Schutzverfahren

Eine alte Weisheit besagt, daß es nur einen hundertprozentigen Kopierschutz gibt: Ein so schlechtes Programm zu schreiben, daß es keiner kopieren will. Da es aber nicht unsere Absicht sein sollte, dem Anwender schlechte Programme anzudrehen, müssen wir uns etwas anderes überlegen. Folgendes hat sich in der Vergangenheit besonders bewährt:

- Der Anleitungsschutz: Dies ist ein sehr einfacher und besonders bei Computerspielen beliebter Schutz. Das Programm hat einige Worte aus der Anleitung und deren Stelle darin gespeichert, wählt bei jedem Start eines davon aus und läßt den Anwender die Anleitung danach durchforsten - alles andere als anwenderfreundlich aber besonders bei langen Anleitungen effektiv, da sie bei jeder illegalen Kopie mitkopiert werden muß.
- Ein Protection Sheet (deutsch: Schutz-Blatt): Beim Erwerb der Software bekommt der Anwender einen Bogen mit seltsamen Zeichen, Buchstaben oder ähnlichem mitgeliefert, die er beim Start eingeben muß. Solche Protection Sheets werden meist mit schwarzer Schrift auf dunkelrotes Papier gedruckt - für normale Kopierer ein unüberwindliches Hindernis.
- Die Keydisc (deutsch: Schlüsseldiskette): Diese Schutzmethode findet ebenfalls vor allem bei Spielprogrammen ihre Anwen-

dung, wurde jedoch in der Vergangenheit auch bei professioneller Software eingesetzt. Hierbei wird ein Teil einer Diskette beschädigt (softwaremäßig, durch ein Laserloch oder ähnliches). Diese Beschädigung kann kaum ein Kopierprogramm kopieren (worin läge auch der Sinn, eine Diskette zu zerstören?) und wird beim Start des Programms abgefragt. Keine schlechte Idee, da der Anwender nicht vor jedem Start Bücher über bestimmte Codes wälzen muß. Allerdings sind die geschützten Disketten auch für Datensicherungszwekke nicht kopierbar. Was geschieht also, wenn die Originaldisks verschwinden oder beschädigt werden sollten? Weiterhin muß bei jedem Start von der Festplatte die Originaldiskette in das Diskettenlaufwerk eingeschoben werden.

- Die Seriennummer (englisch: serial number): Dies ist unter professioneller, teurer Software der am weitesten verbreitete Schutz. In eine Datei jedes verkauften Exemplares wird eine andere Nummer eingefügt (natürlich auf alle nur erdenklichen Arten verschlüsselt), die beim Weiterkopieren wohl oder übel mitkopiert werden muß. Nun liegt weiterhin jedem verkauften Produkt ein Zettel bei, der die Seriennummer und eine Notiz enthält, die besagt, daß der Anwender nur dann berechtigt ist, das Programm zu benutzen, wenn er diesen korrekt ausgefüllt an die Herstellerfirma zurückschickt. Schickt ein Anwender diesen Bogen nicht zurück, macht er sich beim Anwenden des Produkts strafbar; es gehört ihm ja nicht. Taucht nun doch einmal eine Kopie auf, weiß die Herstellerfirma sofort, aus wessen Quelle sie stammen muß - ein Risiko, das sich viele Käufer sicher zwei- oder dreimal überlegen.

Bevor Sie sich nun aber munter ans Schützen Ihrer eigenen Werke machen, noch folgendes: Ein Schutz, egal welcher Art, macht Ihr Produkt bestimmt nicht beliebter. "Kein Kopierschutz!" war schon immer ein schlagendes Werbeargument (Dieser Satz schließt allerdings nur das Versprechen ein, daß alle mitgelieferten Disketten kopiert werden können. Über etwaige Seriennummern oder Anleitungsabfragen sagt er nichts aus.) Überlegen Sie sich deshalb vorher, ob ein Kopierschutz Ihren Umsatz positiv oder negativ beeinflussen wird. Der nächste Abschnitt wird sich mit einem Zahlensystem befassen, das den Maschinenspracheprogrammierern sehr vertraut, dem Turbo-Pascal-Normalbenutzer aber unbekannt sein dürfte: Dem Hexadezimalsystem. Kenntnisse über dieses System werden in den darauffolgenden Abschnitten unbedingte Voraussetzung sein. Wer schon ein wenig Übung darin hat, kann die folgenden Zeilen ruhigen Gewissens überspringen.

Ein Kopierschutz stößt nicht immer auf Gegenliebe

Das Hexadezimalsystem

Das Dezimalsystem, ein Zahlensystem, dessen Ziffern sich aus den zehn Zahlen 0 bis 9 zusammensetzen, dürfte wohl das geläufigste überhaupt sein. Leider kann ein Computer mit diesem nicht sehr viel anfangen. Er versteht sich besser mit einem Zahlensystem, das seine Ziffern nicht aus zehn, sondern 16 verschiedenen Zahlen zusammensetzt: Dem Hexadezimalsystem. Folglich benötigt dieses System auch 16 verschiedene Zahlen. Es bedient sich dabei der zehn Zahlen unseres Zehnersystems (0 bis 9) und belegt die restlichen sechs Stellen mit den Buchstaben A bis F. Da eine Hexadezimalzahl aber immer noch mit einer Dezimalzahl verwechselt werden könnte (zum Beispiel 20 kann eine Hexadezimal- oder Dezimalzahl sein) hat man sich auf das Kennzeichen "$" für Hexadezimalzahlen geeinigt. Damit sind eventuelle Verwechslungen so gut wie ausgeschlossen. Einige einfache Beispiele sind in Tabelle 1 aufgeführt.

Wie Sie sicherlich schnell erkannt haben, hat die erste Ziffer einer zweistelligen Zahl im

Tabelle 1

Dezimalsystem	Hexadezimalsystem
5	$5
10	$A
15	$F
16	$10
255	$FF
256	$100
4096	$1000
65535	$FFFF

Hexadezimalsystem nicht den zehnfachen Wert, wie es im Zehnersystem der Fall ist (zum Beispiel 30), sondern den sechzehnfachen (dezimal 16 = $10). Die erste Ziffer einer dreistelligen Hexadezimalzahl hätte demnach nicht den 10*10 = 100-fachen, sondern 16*16 = 256-fachen Wert der ersten Stelle.

Der Anleitungs- und Protection-Sheet-Schutz

Wie schon im vorigen Teil erwähnt, muß der Hersteller jedem Anwender eine Anleitung beziehungsweise ein Blatt beim Erwerb seines Produkts mitliefern, woraus das Programm beim Start bestimmte Teile wissen möchte. Eine typische Aufforderung eines anleitungsgeschützen Programm könnte beispielsweise lauten:

```
"Geben Sie bitte das dritte Wort der vierten
Spalte auf Seite 3 der Anleitung ein."
```

Wenn schon Kopierschutz, dann aber richtig

Wie Sie sich aber leicht vorstellen können, werden manche Käufer versuchen, den Schutz aus Ihrem Programm zu entfernen. Um dies zu verhindern, sollten Sie mindestens folgende sechs Punkte beachten (Sie können die Auflistung selbstverständlich jederzeit erweitern):

- Niemals Worte, die abgefragt werden, unchiffriert in ein Programm einbauen.
- Niemals durchgehende Kontrollroutinen schreiben; Sie sollten zeitweise komplizierte, wenn auch sinnlose Rechenoperationen einfügen. Das macht das Programm fast unnachvollziehbar.
- Niemals Kontrollabfragen gleich nach dem Start des Programms durchführen. Erst sollten Sie ein Titelbild oder ähnliches zeigen.
- Niemals Kontrollabfragen in einer eigenen Prozedur ausführen lassen, da diese in Maschinensprache mit einem CALL-Befehl (Aufruf einer Unterroutine) aufgerufen wird. Ein Entschützer müßte also nur diesen einen CALL-Befehl entfernen und schon wäre Ihr Schutz umsonst gewesen.
- Niemals zu einfache Kontrollroutinen schreiben. Machen Sie einem Entschützer das Leben so schwer wie nur möglich!
- Niemals zu wenige Begriffe in ein Programm einbauen, die abgefragt werden können (mindestens 30 bis 50). Fragen Sie nur wenige Begriffe ab, stellen sich manche Anwender Listen zusammen, die alle Abfragemöglichkeiten enthalten.

Viel mehr gibt es zum Prinzip eigentlich nicht zu sagen. Die Ausführung bleibt dem jeweiligen Programmierer überlassen. Bei Ihren ersten Versuchen greifen Sie sicher gerne auf ein fertiges Beispiel zurück. Dafür steht Ihnen das Programm ANLEITUNG.PAS zur Verfügung. Sie können es zur Lösung des Problems heranziehen. Das Programm fragt allerdings nicht nach Anleitungsseiten, sondern nur nach numerierten Antworten. Diese Antworten sind verschlüsselt im Programm verborgen: Der ASCII-Wert jedes Buchstaben wurde um 100 erhöht. Damit dürfte es fast unmöglich sein, die Antworten mit einem Diskmonitor zu entdecken. Die ASCII-Werte der vom User eingegebenen Buchstaben und Zahlen werden nun ebenfalls um 100 erhöht und mit den im Speicher befindlichen Werten verglichen. Zum besseren Verständnis ein Beispiel: Sie möchten von einem User entweder den Begriff "Hallo" oder "wieder" wissen. Die größte Todsünde wäre es, die Begriffe folgendermaßen in ein Programm einzubauen:

```
Antwort[1]:='Hallo';
Antwort[2]:='wieder';
```

Diese Begriffe könnte man mit einem einfachen Diskettenmonitor aufspüren und verändern. Eine bessere Lösung wäre:

```
Antwort[1]:=chr(ord('H')+100)+chr(ord('a')+100)+...
Antwort[2]:=chr(ord('w')+100)+chr(ord('i')+100)+...
```

Auf diese Weise werden auf der Diskette nicht die Worte "Hallo" und "wieder", sondern nur wirre Zeichen an ihrer Stelle gespeichert. Beim Vergleich mit der Eingabe des Users müssen nun, wie schon oben beschrieben, zuvor diese in dasselbe "Format der wirren Zeichen" verwandelt werden. In unserem Beispiel muß das Programm also den ASCII-Wert jedes Zeichens um 100 erhöhen. Die möglichen Antworten von ANLEITUNG.PAS:

1. Antwort der
2. Antwort ist
3. Antwort aber

Falls noch Fragen offen sein sollten, sollten Sie das eben genannte Programm genauer analysieren. Es ist gut dokumentiert und logisch aufgebaut.

Schutz durch Seriennummern

Fast ausschließlich große Firmen wenden dieses Verfahren an, um den illegalen Kreislauf Ihrer Software zu stoppen. Allerdings hat auch dieser Schutz einen gewaltigen Nachteil - für den Hersteller. Es liegt in der Natur der Sache, daß die codierte Seriennummer immer an der gleichen Stelle im Programm steht - andernfalls wären die Kosten für jede Diskette zu hoch. Was läge also näher, als sich zweimal dasselbe Programm zu besorgen und einen Diskettenvergleicher (zum Beispiel Diskcomp von DOS) nach Unterschieden zu befragen? Über das weitere Vorgehen brauchen wir wohl keine Worte zu verschwenden. Davon abgesehen hat dieses Verfahren auch einen materiellen Nachteil: Es ist teuer, jede verkaufte Diskette zu verändern.

Das erstere ist aber durch einige Sicherheitsmaßnahmen in den meisten Fällen verhinderbar. Erfinden Sie sich beispielsweise ein Schema für den Aufbau Ihrer Seriennummer, und prüfen Sie beim Start des Programms die Erfüllung desselben. So können Sie mit ziemlicher Sicherheit ein zu einfaches Entfernen der Seriennummer verhindern.

Doch nun genug dieser mißtrauischen Gedanken und zurück zum eigentlichen Thema. Da Sie sicherlich nicht vorhaben, die Seriennummer in jedes verkaufte Exemplar Ihres Programmes von Hand einzufügen und das Programm jedesmal von neuem zu compilieren, müssen Sie sich ein Programm entwerfen, das diesen Vorgang automatisch vornimmt. Doch zunächst geht es um den Aufbau einer Diskette. Dies ist zum weiteren Verständnis unbedingte Voraussetzung.

Beinahe jede Diskette ist heutzutage beidseitig beschreibbar. Demzufolge besitzt jedes Floppy-Laufwerk zwei Schreib/Leseköpfe; für jede Seite einen. Die einzelnen Seiten lassen sich nun wieder in Tracks (deutsch: Spuren) und diese in Sektoren unterteilen (eine auf 320 KByte formatierte Diskette hat acht, eine auf 360 KByte formatierte Diskette neun Sektoren je Track). Jeder dieser Sektoren faßt unter normalen Umständen 512 Bytes. Ihr Programm ist also über diese 512-Byte-Einheiten verteilt. Weitere Einzelheiten erfahren Sie später. Sie sind zum Verständnis an dieser Stelle nicht notwendig.

Eine Seriennummer schützt nur im Zusammenhang mit weiteren Verschlüsselungsmethoden

Ferner sollten Sie ein wenig über die Verwendung von Maschinen-spracheroutinen in Turbo-Pascal wissen:

In Maschinensprache gibt es unter anderem folgende Register (andere sind für uns nicht interessant), die sich ähnlich wie Variablen in Turbo-Pascal verwenden lassen : AX, BX, CX, DX und ES, wobei sich AX, BX, CX und DX wieder in AH, AL, BH, BL, CH, CL, DH und DL aufteilen lassen (H steht jeweils für High-Byte = höherwertiges Byte und L für Low-Byte = niederwertiges Byte). Werfen Sie zum besseren Verständnis einen Blick auf Tabelle 2. AX beispielsweise kann jede vierstellige Hexadezimalzahl speichern; demnach jeden Wert von $0000 bis $FFFF. Mit AH können Sie nun auf die ersten beiden, mit AL auf die letzten beiden Stellen einer in AX befindlichen vierstelligen Hexadezimalzahl zurückgreifen.

Doch wie lassen sich diese Register direkt von Turbo-Pascal aus beschreiben? Zu diesem Zweck stellt die Unit DOS den Record REGISTERS zur Verfügung. Mit ihm und einem weiteren Befehl können die einzelnen Register (daher der Name) direkt angesprochen werden.

Wir wissen jetzt, wie die Maschinenspracheregister von Turbo-Pascal aus direkt belegt werden können. Dies allein hat jedoch keinen großen Sinn. Wir benötigen weiterhin eine Routine, die etwas mit den übergebenen Werten anzufangen weiß. Andernfalls wäre es die ganze Arbeit nicht wert. Diese Routinen gibt es auch: Die Interrupts stellen sie zur Verfügung. Sie werden durch den Turbo-Pascal-Befehl INTR (Interruptnummer, Registervariable) aufgerufen. AH muß vor jedem Auf-

Tabelle 2

AX	AH (High-Byte)	AL (Low-Byte)
$1200	$12	$00
$F	$00	$0F
$FFDD	$FF	$DD
$0	$00	$00

ruf die Nummer der Funktion des jeweiligen Interrupts angeben, die wir benötigen. Mit den anderen Registern werden meist Werte über Daten oder ähnliches übergeben.

Beispiel: Wir wollen den Interrupt $13, Funktion $0D aufrufen, wobei DL $00 enthalten soll. Diese Funktion löst einen Diskreset auf dem von DL angegebenen Floppy-Laufwerk aus (DL: 0 = A, 1 = B etc.; bei Festplatten muß DL um 64 erhöht werden).

```
Program Register_DEMO;
        Uses DOS;
        var Reg : Registers;
        Begin;
          with Reg do begin;
            AH:=$0D;          { Funktion $0D }
            DL:=$00;          { Laufwerk A   }
            Intr($13, Reg);   { Interrupt $13 }
          end;
        end.
```

Doch nun wieder zurück zu unserem eigentlichen Problem: Dem schnellen und automatischen Einbinden von Seriennummern in Programme. Legen Sie die Seriennummer am besten in einer String-Variable in Form wirrer Zeichen ab. Doch wie können Sie ein Programm schreiben, das die jeweiligen Seriennummern auf Diskette und dabei genau in diese String-Variable schreiben kann, wenn Sie nicht einmal genau wissen, wo die Daten über die String-Variable auf der Diskette abgelegt sind?

Bei der Lösung dieses Problems können Sie wie folgt vorgehen:

Nehmen wir an, die String-Variable, welche die Seriennummer später enthalten soll, haben Sie NUMMER genannt. Schreiben Sie nun beispielsweise in den Hauptteil Ihres Programms:

```
NUMMER:='FindeDas';
```

Jetzt brauchen Sie nach der Compilierung nur noch einen Diskettenmonitor die Stelle auf der Diskette suchen zu lassen, welche das Wort "FindeDas" enthält. Und den haben Sie: Das Programm SUCHE.PAS durchsucht jede Diskette (keine Festplatte!) nach ihrem Inhalt. Bevor sich SUCHE.PAS jedoch an die Arbeit machen kann, benötigt es einige Werte (die Werte sind auf eine 360 KByte Diskette zugeschnitten; für andere Diskettenformate sollten Sie sich Tabelle 5 ansehen):

Begriff suchen	j (für ja)
Laufwerk	0=A, 1=B etc.
Suchbegriff	FindeDas
Sekt. in. anzeigen	j (für ja)
ab Seite durchs.	0 = ab 1. Seite, 1 = ab 2. Seite
wie viele Tracks	Einen Track von 0 bis 39 angeben
durchs. ab Sektor	1 eingeben (Suche bei Sektor 1 beginnen)
durchs. bis Sektor	9 eingeben (Suche bei Sektor 9 beenden)

Haben Sie diese Werte eingegeben, wird das Programm die Diskette in dem vorgegebenen Laufwerk nach FindeDas durchsuchen. Manchmal kann es allerdings vorkommen, daß das gesuchte Wort über mehrere Sektoren verteilt ist. In diesem Fall findet das automatische Suchprogramm den Begriff nicht. Sollte dieses Problem einmal auftauchen, können Sie SUCHE.PAS folgende Befehle geben:

Begriff suchen	n (für nein)
Laufwerk	0=A, 1=B etc.
Seite	0 = 1. Seite, 1 = 2. Seite
Track	Einen Track von 0 bis 39 angeben
von Sektor	1 eingeben (Suche bei Sektor 1 beginnen)
bis Sektor	9 eingeben (Suche bei Sektor 9 beenden)

Nach der Eingabe dieser Werte können Sie die Diskette mit den Tasten "Pfeil oben" und "Pfeil unten" durchforsten. Der Computer zeigt Ihnen im oberen Bereich des Bildschirms die Stelle auf der Diskette des gezeigten Ausschnittes. "Q" bricht das Programm ab.

Haben Sie damit schließlich den Begriff "FindeDas" gefunden, halten Sie sich die Daten über dessen Lage schriftlich fest. Vor einem weiteren Vorgehen sollten Sie einmal selbst das Programm SERIEN.PAS untersuchen. Es zeigt die Vorgehensweise, einen ge-

fundenen Begriff durch einen andern (oder durch Seriennummerdaten) zu ersetzen. Es greift dabei auf die Unit DISCFORM zurück und bedient sich folgender Funktionen (Sie liefern das Boolean-Ergebnis TRUE, wenn keine Komplikationen auftraten und FALSE, falls der Befehl nicht ausgeführt werden konnte.):

```
Function ReadSector(Drive,Head,Track,VielSectors,abSector :
Integer; OFSaddr, SEGaddr : LongInt) : Boolean;
Function WriteSector(Drive,Head,Track,VielSectors,abSector :
Integer; OFSaddr, SEGaddr : LongInt) : Boolean;
```

Beim Aufruf dieser Funktionen müssen einige Werte angegeben werden. Diese sollten folgende Angaben enthalten:

Drive	Nummer des Floppy-Laufwerks: 0=A, 1=B etc.
Head	Schreib/Lesekopf 0 oder 1
Track	Track zwischen 0 und 39
VielSectors	Wie viele Sectoren laden (meist 1)?
abSector	Mit welchem Sektor soll begonnen werden?
OFSaddr	Offsetadresse eines 512 Byte Puffers (siehe unten)
SEGaddr	Segmentadresse eines 512-Byte Puffers (siehe unten)

Wahrscheinlich werden Sie bei den letzten beiden Angaben nicht genau verstehen, um was es sich handelt. Es ist ganz einfach: Da Sie einen Sektor (der eine Größe von 512 Bytes hat) laden wollen, benötigen Sie auch einen reservierten Speicherbereich von 512 Byte. Definieren Sie sich dafür am besten eine Variable:

```
var  Buffer : Array[1..512] of Byte;
```

Nun wollen die obigen Funktionen aber die Anfangsadresse der Variable BUFFER wissen. Turbo-Pascal stellt dafür die beiden Befehle OFS(Variable) und SEG(Variable) zur Verfügung. Übergeben Sie beim Start der Funktion beispielsweise:

```
OFSaddr : OFS(Buffer[1])
SEGaddr : SEG(Buffer[1])
```

Die Unit DISCFORM stellt selbstverständlich mehr als nur diese beiden Befehle zur Verfügung. Die anderen werden aber erst später von Interesse sein und dann erklärt werden.

Sollten Sie auf die Idee gekommen sein, den Befehlen dieser Unit lag Zauberei und Magie zu Grunde, da sie nicht mit gewöhnlichen Turbo-Pascal-Befehlen realisierbar sind, müssen wir Sie leider enttäuschen. In beiden Fällen stand eine Maschinensprachеroutine Pate. Da Sie nach der vorhergehenden Einführung wissen, wie man solche Routinen in eigene Programme einbauen kann, geht es jetzt um deren genaue Beschreibung:

Interrupt $13, Funktion $02: Sektoren von Diskette lesen (siehe Tabelle 3)
Nach dem Aufruf der Funktion enthält AH den Status. Ist AH = 0, so ist kein Fehler vorhanden.

Interrupt $13, Funktion $03: Sektoren auf Diskette schreiben (siehe Tabelle $)
Nach dem Aufruf der Funktion enthält AH den Status. Ist AH = 0, so ist kein Fehler vorhanden.

Bevor Sie sich nun voller Vorfreude an Ihren Computer stürzen, geht es noch um ein kleines Problem, das bei der Arbeit mit diesen Funktionen auftauchte: Starten Sie ein Sektorenleseprogramm von der Festplatte, ohne vorher auf die Diskette zugegriffen zu haben, kann es vorkommen, daß das Programm den Sektor nicht findet. Ein bedauerliches Problem, zu dem es aber eine einfache Lösung gibt:

Sie kopieren die compilierte Version der Leseroutine als EXE-Datei auf die Diskette und rufen diese Routine von Ihrem Pro-

Tabelle 3 und 4

Register	High-Byte	Low-Byte
AX	$02	Anzahl Sektoren
CX	Nummer des Tracks	ab welchem Sektor
DX	Nummer des Kopfes	Nummer des Laufwerks
BX	Offset des Zeichenpuffers (512 Bytes/Sektor)	
ES	Segment des Zeichenpuffers (512 Bytes/Sektor)	

Register	High-Byte	Low-Byte
AX	$03	Anzahl Sektoren
CX	Nummer des Tracks	ab welchem Sektor
DX	Nummer des Kopfes (0-1)	Nummer des Laufwerks (0=A)
BX	Offset des Zeichenpuffers (512 Bytes/Sektor)	
ES	Segment des Zeichenpuffers (512 Bytes/Sektor)	

gramm aus auf. Zum Aufruf einer EXE-Datei mit einem Turbo-Pascal-Programm, stellt Turbo-Pascal den EXEC-Befehl bereit. Vor dessen Ausführung sollte jedoch der maximale Wert des HEAP begrenzt werden. Andernfalls ist der gesamte Speicher belegt und kein Programm kann mehr geladen werden. Haben Sie darin keinerlei Erfahrung, so fügen Sie nach dem PROGRAM-Befehl Ihres Programms folgende Zeile ein:

```
{$M $4000, 0, 0}
```

Diese Zeile setzt den maximalen HEAP-Wert auf Null herab. Andernfalls kann Ihr Programm, wie oben schon beschrieben, kein anderes aufrufen. Ein einfaches Beispiel zur Vorgehensweise beim EXEC-Befehl:

Angenommen Ihr Sektorenladeprogramm auf der Diskette heißt LADSEK.EXE. Demnach kann ein Aufruf dessen von jedem Laufwerk aus wie folgt geschehen:

```
SWAPVECTORS;                    { DOS-Vektoren }
EXEC('C:\COMMAND.COM','/C A:'); { A:           }
EXEC('A:\LADSEK.EXE','');       { LADSEK laden }
SWAPVECTORS;                    { TP-Vektoren  }
```

Mit diesem Wissen dürfte es Ihnen wohl kaum noch Schwierigkeiten bereiten, Ihre Programme schnell und unkompliziert mit Seriennummern zu schützen. Viele Anwender werden es sich dadurch mehrmals überlegen, ob Sie das Programm weiteren Personen kopieren werden oder nicht. Vergessen Sie nur nicht, Ihr Programm die Seriennummern auf Wahrscheinlichkeit untersuchen zu lassen! Andernfalls wird es früher oder später jemandem gelingen, das Programm von der Seriennummer zu "befreien".

Ebenfalls ist davon abzuraten, die Seriennummer in einen unbenutzten Sektor auf die Diskette zu schreiben und diesen bei jedem Start zu kontrollieren. In diesem Fall wäre es nicht möglich, das Programm auf Festplatte zu kopieren, ohne jedesmal die Keydisc in das Floppy-Laufwerk einzulegen.

Schutz durch eine unkopierbare Keydisc

Der Begriff Keydisc wurde schon zu Beginn dieses Artikels angesprochen. Er läßt sich wörtlich mit "Schlüsseldiskette" übersetzen. Und genau dies soll unser Ziel sein: Eine Diskette so zu verändern, daß sie nicht kopiert werden kann und als "Schlüssel" zu Ihrem Programm dient. In Tabelle 5 zuvor einige Daten über die geläufigsten Diskettenformate. Überdies ist eine weitere Maschinenspracheroutine für das Verständnis dieses Abschnittes notwendig:

Interrupt $13, Funktion $05: Track formatieren (siehe Tabelle 6)
Nach dem Aufruf der Funktion enthält AH den Status. Ist AH = 0, so ist kein Fehler vorhanden.

Diese Routine wird ebenfalls in die Unit DISCFORM eingebunden. Die Befehlssyntax lautet:

```
Function Format(Drive,Head,
Track,VielSectors,abSector : Integer;
OFSaddr, SEGaddr : LongInt) : Boolean;
```

Die Funktion liefert bei erfolgreicher Ausführung das Ergebnis TRUE, bei mißglückter FALSE. Zu dieser Maschinenspracheroutine dürfte wohl eine kleine Erklärung notwendig sein. Wie ihr Name schon vermuten läßt, kann die Routine einzelne Tracks formatieren. Dazu sind ähnliche Werte wie bei den in Abschnitt 4 aufgeführten Schreib-/Leseroutinen notwendig. Die Register BX und ES zeigen in diesem Fall allerdings nicht auf einen Datenpuffer, sondern auf ein 4 Byte Adreßfeld.

Vor einer weitergehenden Erklärung ein Beispiel: Sie möchten mit Schreib/Lesekopf

Tabelle 5

Disc-Größe	Tracks (Spuren)	Sektoren pro Track	Speicher
5 1/4 Zoll	40 (0 bis 39)	8	320 KByte
5 1/4 Zoll	40 (0 bis 39)	9	360 KByte
5 1/4 Zoll	80 (0 bis 79)	15	1.2 MByte
3 1/2 Zoll	80 (0 bis 79)	9	720 KByte
3 1/2 Zoll	80 (0 bis 79)	18	1.44 MByte

0 auf einer Diskette in Laufwerk A Track 39 mit 5 Sektoren, angefangen bei Sektor 1 formatieren. Dazu müßten Sie der Formatierroutine folgende Werte übergeben:

Anzahl Sektoren	5
Nummer des Tracks	39
ab welchem Sektor	1
Nummer des Kopfes	0
Nummer d. Laufwerkes	0 (0=A)
Adreßfeld	siehe unten

Für jeden dieser Sektoren ist nun ein wie oben beschriebenes 4 Byte Adreßfeld notwendig. Legen Sie dieses beispielsweise in der Variable ADRESSE ab, so muß sie wie folgt dimensioniert sein:

```
var ADRESSE : array[1..Sektoren, 1..4] of Byte;
```

Die vier Bytes des Adreßfeldes müssen unbedingt als letzteres definiert werden. Der Grund liegt in dem Turbo-Pascal spezifischen Aufzeichnungsverfahren. Turbo-Pascal speichert alle Bytes der zuletzt angegeben Dimension, in unserem Fall also die 4 Adreßbytes, so oft, wie es die Größe der ersten Dimension vorgibt; in unserem Beispiel die Anzahl der Sektoren. Definieren wir die Variable also wie angegeben, so speichert Turbo-Pascal so viele 4 Byte Adreßfelder wie angegebene Sektoren.

Bei der Angabe der Speicheradresse des Adreßfeldes ist also folgende Angabe denkbar:

```
BX : OFS(ADRESSE[1,1])
ES : OFS(ADRESSE[1,1])
```

Soweit zur Funktion selbst. Nun noch einige Worte über das Adreßfeld. Byte 1 des Adreßfeldes legt die Nummer des Tracks fest. In unserem Fall ist dies immer Track 39. Byte 2 legt die Nummer des Kopfes fest; bei uns immer 0. Soweit nichts Besonderes, doch das wird mit Byte 3 anders. Damit kann nämlich die logische Nummer des Sektors angegeben werden - ganz im Gegensatz zur physikalischen. Das war eine komplizierte Beschreibung einer im Prinzip einfachen Angelegenheit: Wir wollen Track 39 mit 5 Sektoren formatieren. Gewöhnlich und floppyintern haben diese die Nummern 1, 2, 3, 4 und 5. Dies ist die physikalische Anordnung der Sektoren auf der Diskette. Diese Sektoren können aber eine völlig andere logische Nummer von 0 bis 255 ($0 bis $FF) haben. Hat beispielsweise Sektor 1 die logische Nummer 47, so meldet die Leseroutine des Interrupts $13 bei einem Leseversuch von Sektor 1 auf Track 39 einen Fehler. Diese sucht nämlich nicht nach dem physikalischen, sondern dem logischen Sektor; in unserem Fall 47. Ebenso gehen fast alle Kopierprogramme vor - sie können in den meisten Fällen nur Sektoren mit den logischen Nummern 1 bis 9 kopieren.

Register	High-Byte	Low-Byte
AX	$05	Anzahl Sektoren
CX	Nummer des Tracks	ab welchem Sektor
DX	Nummer des Kopfes (0-1)	Nummer des Laufwerks (0=A)
BX	Offset des 4-Byte-Adreßfeldes	
ES	Segment des 4-Byte-Adreßfeldes	

Das 4-Byte-Adreßfeld

Byte	Wert betrifft	Werte möglich
1	Tracknummer	siehe Tabelle 5
2	Kopfnummer	siehe Tabelle 5
3	Sektornummer	siehe Tabelle 5
4	Bytes/Sektor	0 = 128, 1 = 256, 2 = 512, 3 = 1024

Tabelle 6 und 7

Byte 4 erlaubt, wie Byte 3 des Adreßfeldes, einigen Schabernack. Mit ihm kann die Anzahl der Bytes festgelegt werden, die in einen Sektor passen sollen. Normal ist hier der Wert 2 für 512 Bytes pro Sektor. Versucht Funktion 2 des Interrupts $13 einen Track zu lesen, dessen Größe nicht exakt 512 Bytes pro Sektor ist, so meldet der Status in AH nach dem Aufruf einen Lesefehler. Über die genaue Bedeutung der Statuswerte sollten Sie Tabelle 8 befragen. Diese zeigt Statuswerte für die Diskette (nicht für Festplatte), die nach dem Aufruf der beschriebenen Funktionen auftreten können.

Machen Sie sich bitte keine Vorwürfe über Ihr Unwissen, falls Sie nicht alle Fehlermeldungen verstehen sollten. In den meisten Fällen reicht es aus, zu wissen, daß alle Werte außer 0 auf Fehler hinweisen. Der einzige Wert, der hier beschrieben werden soll, ist Nummer $10: "CRC- oder ECC Error beim Lesen". CRC-Error ist eine Abkürzung und steht für Cyclical Redundancy Check Error; auf deutsch übersetzt: Prüfsummenfehler.

Statuswert	Beschreibung
$0	Kein Fehler
$1	ungültige Parameter (falsche Werte übergeben)
$2	Adreßmarkierung nicht gefunden
$3	Schreibschutz auf Diskette
$4	Sektor nicht gefunden
$6	Diskette nicht im Laufwerk
$8	DMA-Überlauf
$9	Überlauf der DMA-Segment-Grenze
$10	CRC- oder ECC Error beim Lesen
$20	Fehler des Controllers
$40	Track nicht gefunden
$80	Laufwerk zeigt keine Reaktion

Register	High-Byte	Low-Byte
AX	$04	Anzahl Sektoren
CX	Nummer des Tracks	ab welchem Sektor
DX	Nummer des Kopfes	Nummer des Laufwerks
BX	Offset des Zeichenpuffers (512 Bytes/Sektor)	
ES	Segment des Zeichenpuffers (512 Bytes/Sektor)	

Tabelle 8 und 9

Dieser Fehler passiert beim Versuch, einen Sektor zu lesen, der nicht auf 512 Bytes formatiert wurde.

Auf diese Weise können Sie auch schon einen einfachen Kopierschutz realisieren. Formatieren Sie einige Sektoren eines unbenutzten (!) Tracks beispielsweise mit 128 Bytes pro Sektor und fragen Sie bei jedem Programmstart ab, ob sich auf diesem Sektor ein Error befindet. Viele Kopierprogramme sind nicht in der Lage von einer so formatierten Diskette eine 1:1-Kopie anzulegen (zum Beispiel Diskcopy von DOS). Sie sollten aber auch hier darauf achten, das Leseprogramm möglichst von Diskette zu starten, da es den Sektor sonst unter Umständen nicht findet.

Ob ein Track mit Daten belegt ist, können Sie mit dem schon zuvor vorgestellten Programm SUCHE.PAS herausfinden. Alle Sektoren eines leeren Tracks müssen mit folgendem Zeichen gefüllt sein: ÷ (ASCII-Wert 246). Der Formatierbefehl füllt alle Sektoren automatisch damit. Um Sektoren auf ihre Fehlerhaftigkeit hin zu überprüfen, bietet DOS eine Sektoren-Check-Routine an. Sie ist sehr stark an die Lade-Funktion $02 des Interrupts $13 angelehnt:

Interrupt $13, Funktion $04: Sektoren prüfen (siehe Tabelle 9)
Anschließend enthält AH den Status und AL die Anzahl der überprüften Sektoren. Die Unit DiscForm stellt diese Routine in Form des folgenden Befehls zur Verfügung:

```
Function CheckSector(Drive,Head,Track,
VielSectors,abSector : Integer;
OFSaddr, SEGaddr : LongInt) : Boolean;
```

Die Funktion liefert bei erfolgreicher Ausführung das Ergebnis TRUE, bei mißglückter FALSE.

Der Nachteil dieser Methode liegt in ihrer unkomplizierten Form. Auf diese Weise geschützte Programme können von vielen speziellen Programmen problemlos kopiert werden, oder kopierte Disketten lassen sich nachträglich leicht modifizieren. Eine etwas komplexere, aber nur von den wenigsten Programmen überwindbare Lösung ist folgende:

Suchen Sie sich zuerst auf der späteren Originaldiskette einen leeren Track (mit SUCHE.PAS) und formatieren diesen ein wenig eigenwillig. Nehmen wir an, Sie haben herausgefunden, Track 39 ist unbenutzt. Formatieren Sie diesen beispielsweise mit 7 Sektoren der logischen Nummer 12, 17, 21, 98, 115, 167, 201 und dem Byte/Sektor Wert 2 (= 512 Bytes/Sektor). Um gar kein Risiko eingehen zu müssen, könnten Sie zum Beispiel in Sektor 167 eine kleine Nachricht hinterlegen. Was hielten Sie von "Bitte nicht weiterkopieren!" oder einigen wichtigen Daten, ohne die das Programm nicht lauffähig wäre?

Einen Lösungsvorschlag zeigt das Programm PROTECT.PAS. Es formatiert einen Track auf einer Diskette in Laufwerk A, schreibt eine Nachricht in einen Sektor und lädt diese wieder. Vergessen Sie aber hier nicht, falls Sie sich eine eigene Laderoutine schreiben, sie später von Laufwerk A starten zu lassen; andernfalls kann es vorkommen, daß das Programm diese Nachricht nicht findet. Diese Probleme treten beim Start von anderen Laufwerken (zum Beispiel Festplatte) zeitweise auf.

Falls Sie sich noch etwas tiefer in die Welt der Disketten einarbeiten möchten, empfiehlt sich das Public-Domain-Programm ULTRA UTILITIES. Diese Utilitysammlung stellt viele Routinen zum Erkunden und Verändern von Disketten und deren Inhalt bereit.