Schlankheitskur

Schmalschrift mit QuickBASIC

Harald Zoschke

In vielen Anwendungen kann sich eine schmalere Schrift als äußerst nützlich erweisen - um interessante Programmiertechniken vorzustellen, enthält dieses Programm gleich drei Varianten eines solchen Fonts.

Es gibt nämlich eine Reihe von Einsatzgebieten, in denen die herkömmlichen Textzeichen zu groß (insbesondere zu breit) sind - zum Beispiel, wenn es bei der Beschriftung angezeigter Kurvenskalen oder Skizzen eng wird oder viel Text etwa auf einer Seitenvorschau angezeigt werden soll.

Hier kommt die Routine `SchmalPrint´ zum Einsatz, die bei Bedarf einfach anstelle von PRINT verwendet wird. Extrem schmale Zeichen eines speziellen Zeichensatzes ermöglichen die Anzeige von bis zu 160 Zeichen pro Zeile auf EGA/VGA- und bis zu 180 Zeichen auf Hercules-Bildschirmen. Jedem Zeichen steht ein Raster von acht Punkten Höhe und vier Punkten Breite zur Verfügung. Der neue Font enthält sämtliche ASCII-Zeichen von 32 bis 255, inklusive Sonder- und Grafikzeichen.

Nebenbei werden in der Schmalschrift-Anzeigeroutine einige interessante Programmiertechniken vorgestellt. Unter anderem wird demonstriert, welch unterschiedliche Ausführungsgeschwindigkeiten die Wahl eines bestimmten Algorithmus zur Folge haben kann. Da die Routinen keine besonderen Interna der Microsoft-BASIC-Compiler verwenden, läßt sich das Programm auch ohne größere Hürden auf Entwicklungssysteme anderer Anbieter portieren.

Die Zeichenmatrix zeigt den Buchstaben `A´ des Schmalschrift-Zeichensatzes in `Großaufnahme´. Hier wird der Aufbau deutlich: Jedes Zeichen besteht aus vier Spalten mit je acht Punkten, die für BASIC in einem Datenfeld abgelegt werden müssen. Normale Zeichen belegen drei, Semigrafik-Zeichen alle vier Spalten. Acht Punkte - das läßt bereits ahnen, daß sich im Zeichensatz zu deren Unterbringung ein Byte anbietet.

Da die Elemente von BASICs kleinstem Datentyp Integer aus zwei Bytes bestehen, wir aber nur ein Byte benötigen, `verschwenden´ wir jedesmal ein Byte. Ohne Platzvergeudung kommt man jedoch zurecht, wenn man für jede Achter-Punktreihe ein Byte eines Strings verwendet. Diesem Zweck dient im vorgestellten Programm die Variable *Font$*: Je vier Bytes enthalten das Bitmuster eines Zeichens. Für 224 Zeichen werden daher 896 Bytes benötigt.

Zum ersten ...

Wie bekommen wir nun die Zeichensatz-Bytes in den String? Am einfachsten wäre es, die Daten

befänden sich in einer Binärdatei. Mit einem Befehl wie *GET #1, FONT$* könnte man - sofern die Variable mittels *FONT$ = SPACE$(896)* vorher auf die entsprechende Größe eingestellt wurde - den kompletten Datensatz mit einem Arbeitsschritt einlesen. Da sich eine solche Datei drucktechnisch schlecht darstellen läßt, sind die Bytes im Hexadezimal-Format in DATA-Zeilen abgelegt.

Beim Start des Programms werden daher zunächst die DATA-Zeilen gelesen und die darin enthaltenen Hex-Bytes in ASCII-Codes umgewandelt; diese werden dann an der entsprechenden Stelle im String `Font$´ eingesetzt. Den Zeiger dafür stellt der mittlere Parameter des MID$-Befehls dar, der grundsätzlich zur Positionsangabe innerhalb des behandelten Strings dient. Anstelle der Voreinstellung der String-Variablen auf die erforderliche Länge und der anschließenden Zeichen-Zuweisung mittels *MID$* hätte man den String innerhalb der Schleife auch mit Hilfe einer Befehlszeile wie *FONT$ = FONT$ + NaechstesZeichen$* `wachsen´ lassen können. Da in diesem Fall jedoch für die hinzukommenden Zeichen BASIC-intern laufend neuer Platz durch Verschieben anderer Daten geschaffen werden muß, ist die erste Methode schneller.

Befindet sich der Zeichensatz in der String-Variablen `Font$´, kommt die Zeichen-Anzeigeroutine zum Zuge. Das Prinzip ist einfach: Je nach ASCII-Zeichen wird ein Offset (Abstand) in den String ermittelt, das heißt jene Position, ab der die zu diesem Zeichen gehörigen Bytes zu finden sind. Da dies stets vier Bytes sind, ergibt sich der Offset als ASCII-Code mal vier minus 32 (der Font beginnt bei ASCII-Code 32, da es sich bei den Codes darunter um Steuerzeichen handelt).

Abschließend müssen dann die vier Bytes so auf den Bildschirm gebracht werden, daß sie das Zeichen anzeigen. Um zu prüfen, wie dies am schnellsten geschehen kann, können drei verschiedene Möglichkeiten verwendet werden. Die erste Methode ist wohl die naheliegendste - da BASIC die Möglichkeit zur Potenzierung bietet, wird sie nach folgendem Prinzip in einer Schleife zum bitweisen Auslesen eines jeden Bytes genutzt:

```
FOR Bit = 0 TO 7
  IF Byte AND 2 ^ Bit THEN PSET ...
NEXT Bit
```

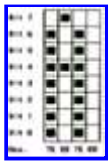
Anhand eines jeden Bits wird dann - je nachdem, ob es den Wert 1 oder 0 hat - ein Punkt in der Vordergrund- oder Hintergrundfarbe geplottet. Die Bildschirmposition eines jeden Punktes ergibt sich aus den angegebenen x- und y-Koordinaten sowie der Position des Punktes innerhalb des gerade behandelten Bytes. Die Routine wurde als SUB-Unterprogramm ausgelegt, die folgendermaßen aufgerufen wird:

```
CALL SchmalPrint (Zeichen, Y, X)
```

Statt *Zeichen* wird der Routine (zu finden im zweiten Listing) der ASCII-Code des anzuzeigenden Zeichens übergeben. Im Gegensatz zu BASICs PRINT-Befehl spart man sich hier die vorherige Koordinatenangabe mit LOCATE durch die direkte Angabe beim Aufruf. Nach Deklaration der Routine läßt sie sich sogar ohne CALL-Anweisung aufrufen und wirkt somit wie ein selbstentwickelter neuer BASIC-Befehl. Die Syntax beim Aufruf hieße dann:

```
SchmalPrint Zeichen, Y, X
```

Daß jedoch die Potenz-Berechnung unter BASIC recht langsam ist, zeigt sich in der Print-Routine, die nach der ersten Methode arbeitet: Die niedrige Anzeigegeschwindigkeit macht sie nahezu unbrauchbar. Grund hierfür ist, daß BASIC bei der Abarbeitung von Potenzen - auch bei Integerwerten - auf Fließkomma-Routinen der Sprachbibliothek zurückgreifen muß.



Der Aufbau eines Zeichens (acht Bits hoch, vier Bits breit) in dem neuen Font

In diesem Zusammenhang sei darauf hingewiesen, daß wohl die wichtigste Maßnahme zur Optimierung eines BASIC-Programms darin besteht, dem Listing ein *DEFINT A-Z* voranzustellen, um so weit wie möglich mit Integer-Variablen zu arbeiten. Auf Long-Integers sollte man erst dann zurückgreifen, wenn der Wertebereich der Integerzahlen nicht ausreicht. QuickBASIC muß nämlich bei den Grundrechenarten - im Gegensatz zu den Integerwerten - bei Operationen mit einfacher oder doppelter Genauigkeit spezielle Runtime-Routinen für Fließkommaberechnung verwenden.

Zum zweiten ...

Versuchsweise arbeitet unsere zweite Methode daher mit sequentiellen Befehlsfolgen, um auf die Potenzierung verzichten zu können.

```
IF Byte AND 128 THEN PSET ...
IF Byte AND 64  THEN PSET ...
...
```

Was auf den ersten Blick wie schlechter Programmierstil aussieht (`Hat denn dieser Mensch noch nichts von Schleifen gehört?´), erweist sich in der Praxis als drastische Verbesserung der Ausführungsgeschwindigkeit: Die Routine arbeitet rund 18mal schneller als das Programm der ersten Methode. Die Syntax zu dieser Routine (zu finden im dritten Programmlisting) lautet gleich wie die der ersten.

Und zum ...

Trotzdem habe ich weitergeforscht, ob sich die Anzeigegeschwindigkeit nicht noch weiter erhöhen läßt. Und siehe da, es geht. Der dritten Methode liegt die Überlegung zugrunde, dem BASIC-Befehl LINE die Arbeit zu überlassen, ein Byte auf dem Bildschirm als Bitmuster abzubilden. Möglich wird dies durch den Struktur-Parameter dieses Befehls. Sehen wir uns dazu die LINE-Syntax mit den für uns relevanten Parametern an:

```
LINE (x1, y1) - (x2, y2), Farbe, , Struktur
```

Die Parameter `x1, y1´ und `x2, y2´ geben Start- und Endpunkt der von LINE zu zeichnenden Linie an. `Farbe´ bestimmt die Farbe der Punkte, aus denen sich die Linie zusammensetzt. Beim optionalen (nur bei Bedarf zu verwendenden) Parameter `Struktur´ handelt es sich um einen 16-Bit-Wert, mit dessen Bitmuster sich die Struktur der Linie bestimmen läßt.

Der Hauptzweck des letzten Parameters liegt in der Möglichkeit, punktierte, gestrichelte und strichpunktierte Linien zeichnen zu können, zum Beispiel zur Unterscheidung verschiedener Kurven in Diagrammen. Wir wollen ihn nun jedoch dazu `mißbrauchen´, die Punktreihen unserer Zeichensatz-Bytes auszugeben, um uns das Auflösen der Bytes und das Umsetzen in PSET-Befehle zu ersparen:

```
LINE (X + Spalte, Y )-(X + Spalte, Y + 7), Farbe, , Struktur
```

Die Bildschirmposition jeder Linie ergibt sich aus den angegebenen x- und y-Koordinaten zuzüglich der Punktposition innerhalb des gerade behandelten Bytes (Spalte). Die komplette Routine finden Sie im dritten Listing. Die Aufruf-Syntax ist wiederum dieselbe wie bei den Unterprogrammen der Methoden Nummer 1 und 2.

Bei allen drei Anzeige-Unterprogrammen lassen sich sowohl die Vorder- als auch die Hintergrundfarbe der angezeigten Schmalschrift-Zeichen einstellen (FrbVG und FrbHG); durch Vertauschen dieser Werte läßt sich die Schrift negativ darstellen (Anmerkungen in den Listings beachten). Auch Zeilen- und Zeichenabstand sind frei wählbar. Die x-/y-Koordinaten sind zeichenorientiert; sie beziehen sich auf die

Zeichen- und Zeilenabstände, die Sie mit X1 und Y1 eingestellt haben (bei Bedarf lassen sich die Routinen mit wenig Aufwand auf punktweise Positionierung umschreiben).

Um eine möglichst schnelle Anzeige zu erzielen, werden diese Parameter - genau wie der Zeichensatz-String - nicht als CALL-Parameter an die Anzeigeroutinen übergeben (was zudem den Stack belasten würde), sondern als SHARED-Variable deklariert. Hiermit stehen die Inhalte der betroffenen Variablen auch den Unterprogrammen zur Verfügung.

Außer der in den drei Varianten vorgestellten Routine SchmalPrint zur Anzeige eines Zeichens wurde mit `SPrintString´ auch ein Unterprogramm zur Ausgabe von ganzen Strings vorgesehen. Die Syntax dafür lautet:

```
CALL SPrintString(Text$, Y, X)
```

In Text$ wird der anzuzeigende String angegeben. Da SPrintString zur Anzeige das Unterprogramm SchmalPrint benutzt, werden alle ASCII-Zeichen ab 32 (Leerzeichen) berücksichtigt. Y und X bestimmen die Bildschirmkoordinate des ersten Zeichens. Die Angabe des Zeichen- und Zeilenabstands sowie der Vorder- und Hintergrundfarben geschieht wie bei SchmalPrint. Hier ein Aufruf-Beispiel:

```
Y1 = 8  ´Schmalschrift-Zeilenabstand
X1 = 4  ´Schmalschrift-Zeichenabstand
FrbHG = 15      ´Farbe der nicht gesetzten Pixel
FrbVG = 0       ´Farbe der gesetzten Pixel
X = 100 : Y = 20        ´Position des ersten Zeichens
CALL SPrintString(" Weiter mit beliebiger Taste... ", Y, X)
```

Vielfältige Möglichkeiten erschließen sich dem QuickBASIC-Programmierer durch schmalere Schriften. Drei Programmiermethoden haben verschiedene Vorteile.

Zu den Beispielen im Hauptprogramm:

Alle im ersten Listing enthaltenen Beispiele lassen sich wahlweise mit den Grafikmodi von EGA, VGA, CGA oder Hercules betreiben; die Grafikstufen-Einstellung erfolgt wie gewohnt mit SCREEN. Es ist lediglich auf die mit der jeweiligen Karte verwendbaren Farben und die maximale Auflösung der gewählten Grafikstufe zu achten. Während zum Beispiel mit EGA (SCREEN 9) und kleinstem Zeichenabstand bis zu 160 Zeichen in eine Zeile passen, lassen sich bei Hercules (SCREEN 3) bis zu 180 Zeichen darin unterbringen.

Beispiel 1 sorgt für eine formatierte Anzeige aller ASCII-Zeichen von 32 bis 255. Beispiel 2 demonstriert, wie sich mit Hilfe der Variablen X1 und Y2 die Zeichen- und Zeilenabstände variieren lassen. Soll Blockgrafik angezeigt werden, ist der kleinstmögliche Zeichenabstand (X1 = 4) zu verwenden.

Beispiel 3 ist eine kleine Uhrzeitanzeige, die im Grunde nichts weiter macht, als TIME$ mittels SPrintString auf dem Bildschirm anzuzeigen. Die Programmschleife zur Aktualisierung der Anzeige wird durchlaufen, bis irgendeine Taste gedrückt wird. Hier wird übrigens ganz gezielt LOOP UNTIL LEN(INKEY$) verwendet. Gegenüber ebenfalls zulässigen Befehlen wie LOOP WHILE INKEY$ = "" kommt die LEN-Abfrage auf Leerstring-Bedingung im kompilierten Programm mit weniger Bytes aus.

Mit kleinem Aufwand läßt sich auch eine Textdatei lesen und in Schmalschrift auf dem Bildschirm anzeigen. Prinzipiell geht man dabei wie folgt vor:

```
LinkerRand = 5
OPEN TextDatei$ FOR INPUT AS 1
WHILE NOT EOF(1)
   LINE INPUT #1, Text$
   SPrintString Text$, y, LinkerRand
   y = y + 1
WEND
CLOSE #1
```

Umfaßt eine Textdatei mehr Zeilen, als auf den Bildschirm passen, muß die Anzeige portionsweise geschehen; ein Rollen der Anzeige (Scrolling) ist noch nicht vorgesehen. Erfolgt die Anzeige in einem Bildschirmausschnitt, läßt sich das Textfenster mit Hilfe des LINE-Befehls unter Angabe des optionalen Parameters BF (gefüllte Box) einfärben und löschen; alternativ läßt sich dies auch mit PAINT bewerkstelligen.

Wie schon die drei im Vergleich vorgestellten Anzeige-Methoden zeigen, gibt es meist mehrere Wege, ein Problem zu lösen, von denen sich einer als der am besten geeignete herausstellt. Mit Sicherheit gibt es im vorgestellten Programm weitere Ansatzpunkte, die Anzeigetechnik zu optimieren und ihre Möglichkeiten zu erweitern. (se)

Kasten 1

Das Hauptprogramm des Font-Utility für QuickBASIC beziehungsweise BASIC-PDS

```
 1 ´ ********************************************
 2 ´ * SCHMAL.BAS: Schmalschrift-Anzeige für BASIC *
 3 ´ *                                            *
 4 ´ * (c) c't, Harald Zoschke                    *
 5 ´ ********************************************
 6
 7 DEFINT A-Z
 8
 9 DECLARE SUB SPrintString (Text$, Y, X)
10 DECLARE FUNCTION Hex2Dez (H$)
11 DIM SHARED Font$, X1, Y1, FrbHG, FrbVG
12
13 ´*** Laden des Zeichensatzes in Font$
14 ´*** (vor Benutzung der Schmalschrift einmal aufrufen)
15
16 Font$ = SPACE$(900)
17
18 RESTORE FontData
19 FOR i = 0 TO 44
20    READ HexZeile$
21    FOR j = 1 TO 20
22       MID$(Font$,i*20+j,1) = CHR$(Hex2Dez(MID$(HexZeile$,(j-1)*3+1,2)))
23    NEXT j
24 NEXT i
25
26 ´*** Verwendung der Schmalschrift (Beispiele)
27
28 SCREEN 9      ´EGA 640 x 480  (zum Beispiel)
29 ´ SCREEN 3    ´Hercules (zuvor MSHERC.COM bzw. QBHERC.COM laden!)
30 CLS
31
32 ´*** Beispiel 1:
33
34 LOCATE 1, 3: PRINT "Alle ASCII-Zeichen von 32 bis 255:"
35 FrbHG = 0     ´Farbe nicht gesetzter Pixel
36 FrbVG = 14    ´Farbe gesetzter Pixel
37 X = 0         ´Startposition der Anzeige
38 Y = 1
39 Y1 = 10       ´Schmalschrift-Zeilenabstand
```

```
 40 X1 = 6        ´Schmalschrift-Zeichenabstand
 41
 42 FOR Zeile = 1 TO 7          ´ASCII-Zeichensatz (32-255) anzeigen
 43    FOR Stelle = 0 TO 31
 44       Zeichen = Zeile * 32 + Stelle
 45       X = X + 2
 46       CALL SchmalPrint3(Zeichen, Y + Zeile, X)
 47    NEXT Stelle
 48    X = 0
 49 NEXT Zeile
 50
 51 ´*** Beispiel 2: Der Zeichenabstand läßt sich variieren
 52
 53 FrbHG = 0     ´Farbe nicht gesetzter Pixel
 54 FrbVG = 15    ´Farbe gesetzter Pixel
 55 Y1 = 8
 56 X1 = 8        ´großer Zeichenabstand
 57 CALL SPrintString("Großer Zeichenabstand", 16, 1)
 58 X1 = 5        ´kleinerer Zeichenabstand
 59 CALL SPrintString("Kleinerer Zeichenabstand", 18, 2)
 60 X1 = 4        ´kleinster Zeichenabstand (Normal-Einstellung)
 61 CALL SPrintString(" Kleinster Zeichenabstand", 20, 2)
 62 SPrintString STRING$(160, "A"), 13, 2
 63
 64 ´Invertieren (Negativ-Schrift) durch Tauschen der Farben:
 65 FrbHG = 15     ´Farbe eines nicht gesetzten Pixels
 66 FrbVG = 0      ´Farbe eines gesetzten Pixels
 67
 68 ´ (wie man sieht, kann der Text natürlich auch eine Variable sein):
 69 Text$ = " Negativ-Schrift "
 70 CALL SPrintString(Text$, 20, 35)
 71
 72 ´*** Beispiel 3: Zeitanzeige in einer simulierten LED-Uhr
 73
 74 CIRCLE (511, 54), 30, 7         ´Uhren-Gehäuse zeichnen
 75 CIRCLE (509, 52), 30, 3
 76 PAINT (509, 52), 3
 77 Y1 = 8
 78 X1 = 4
 79 FrbHG = 3
 80 FrbVG = 0
 81 CALL SPrintString("+----------+", 5, 120)
 82 CALL SPrintString("|          |", 6, 120)
 83 CALL SPrintString("+----------+", 7, 120)
 84
 85 FrbHG = 0                      ´Zeit anzeigen...
 86 FrbVG = 12
 87 DO                             ´...bis eine Taste gedrückt wird
 88    CALL SPrintString(" " + TIME$ + " ", 6, 121)
 89 LOOP UNTIL LEN(INKEY$)
 90 SCREEN 0                       ´zurück zum Textbildschirm
 91
 92 ´*** Schmalschrift-Zeichensatztabelle
 93
 94 FontData:
 95 DATA "41 00 00 00 00 00 FA 00 00 E0 00 E0 00 FF 24 FF 00 24 D6 48"
 96 DATA "00 86 38 C2 00 FE CC 14 00 20 40 80 00 38 44 82 00 82 44 38"
 97 DATA "00 54 38 54 00 10 7C 10 00 05 06 00 00 10 10 10 00 06 06 00"
 98 DATA "00 06 38 C0 00 7C 82 7C 00 20 40 FE 00 CE 92 72 00 44 92 6C"
 99 DATA "00 1E 62 87 00 E2 A2 9C 00 7C 92 8C 00 86 88 F0 00 6C 92 6C"
100 DATA "00 62 92 7C 00 00 24 00 00 05 16 00 00 10 28 44 00 28 28 28"
101 DATA "00 44 28 10 00 40 8A 70 00 3A EE B8 00 7E 90 7E 00 FE 92 6C"
102 DATA "00 7C 82 82 00 FE 82 7C 00 FE 92 82 00 FE 90 80 00 7C 92 5C"
103 DATA "00 FE 10 FE 00 82 FE 82 00 02 01 FE 00 FE 10 EE 00 FE 02 02"
104 DATA "00 FE 60 FE 00 FE 38 FE 00 FE 82 FE 00 FE 90 60 00 7C 86 7A"
105 DATA "00 FE 90 6E 00 64 92 4C 00 80 FE 80 00 FE 02 FE 00 FC 02 FC"
106 DATA "00 FE 0C FE 00 EE 10 EE 00 F0 0E F0 00 8E 92 E2 00 00 FE 82"
107 DATA "00 C0 38 06 00 82 FE 00 00 40 80 40 00 02 02 02 00 80 40 20"
```

```
108 DATA "00 2E 2A 1E 00 FE 22 1C 00 1C 22 22 00 1C 22 FE 00 1C 2A 1A"
109 DATA "00 10 7E 90 00 19 25 3E 00 FE 20 1E 00 10 5E 00 00 02 11 5E"
110 DATA "00 FE 08 16 00 80 FE 00 00 3E 38 1E 00 3E 1E 00 00 1C 22 1C"
111 DATA "00 3F 24 18 00 18 24 3F 00 3E 20 10 00 12 2A 24 00 FC 22 22"
112 DATA "00 3E 02 3E 00 3C 02 3C 00 3E 0C 3E 00 36 08 36 00 39 05 3E"
113 DATA "00 26 2A 32 00 10 6C 82 00 00 EE 00 00 82 6C 10 00 18 10 30"
114 DATA "00 0E 12 12 0E 7C 82 83 00 5E 02 5E 00 1C 2A 5A 80 6E AA 5E"
115 DATA "00 AE 2A 9E 00 AE AA 1E 00 2E EA 1E 00 1C 22 23 00 5C AA 5A"
116 DATA "00 5C 2A 5A 00 5C 6A 1A 00 50 1E 40 00 50 9E 40 00 90 5E 00"
117 DATA "00 BE 50 BE 00 1E A8 1E 00 3E 6A A2 00 3A 1C 2E 00 3E 50 7E"
118 DATA "52 5C A2 5C 00 9C 22 9C 00 9C 62 1C 00 5E 82 5E 00 9E 42 1E"
119 DATA "00 B9 05 BE 00 BE 22 BE 00 BE 02 BE 00 3C E7 24 00 FE D2 06"
120 DATA "00 D4 3E D4 00 FE A0 5E 0A 11 FF 90 00 2E 6A 9E 00 10 5E 80"
121 DATA "00 1C 62 9C 00 1E 42 9E 00 BE 9E 00 00 BE 9C BE 00 E8 E8 28"
122 DATA "00 E8 A8 E8 00 0C B2 04 00 1C 10 10 00 10 10 1C 00 66 38 CC"
123 DATA "04 66 38 E2 06 00 BE 00 00 28 10 28 10 10 28 10 28 AA 55 AA"
124 DATA "55 AA AA 55 55 FF 55 FF 55 00 FF 00 00 08 FF 00 00 18 FF 00"
125 DATA "00 08 FF FF 00 08 0F 0F 00 18 1F 00 00 18 FF FF 00 00 FF FF"
126 DATA "00 18 1F 1F 00 18 F8 F8 00 08 F8 F8 00 18 F8 00 00 08 0F 00"
127 DATA "00 00 F8 08 08 08 F8 08 08 08 0F 08 08 00 FF 08 08 08 08 08"
128 DATA "08 08 FF 08 08 00 FF 18 18 00 FF FF 08 00 F8 F8 18 00 1F 1F"
129 DATA "18 18 F8 F8 18 18 1F 1F 18 00 FF FF 18 18 18 18 18 18 FF FF"
130 DATA "18 18 F8 18 18 08 F8 F8 08 18 1F 18 18 08 0F 0F 08 00 F8 F8"
131 DATA "08 00 F8 18 18 00 1F 18 18 00 0F 0F 08 08 FF FF 08 18 FF 18"
132 DATA "18 08 F8 00 00 00 0F 08 08 FF FF FF FF 0F 0F 0F 0F FF FF 00"
133 DATA "00 00 00 FF FF F0 F0 F0 F0 3E 3E 1C 22 3F 54 28 00 7E 40 60"
134 DATA "00 3E 20 3E 20 C6 BA 92 00 1C 22 3C 20 3F 08 38 00 20 1E 20"
135 DATA "00 BA C6 BA 00 7C 92 7C 00 F6 88 F6 00 CE AA 9E 00 0C 0C 0C"
136 DATA "00 1D 36 5C 00 7C 92 82 00 7C 40 7C 00 54 54 54 00 24 74 24"
137 DATA "00 54 24 24 00 24 24 54 00 00 7F 60 00 06 FE 00 00 08 2A 08"
138 DATA "00 14 28 14 28 40 A0 40 00 00 18 00 00 00 08 00 00 18 0E F0"
139 DATA "80 F0 80 70 00 B0 90 50 00 00 18 18 00 00 00 00 00 00 00 00"
140 ´Die letzten drei "00" dienen nur zum Auffüllen (siehe Text)
141
142 ´*** Hexadezimalzahl (String) in Dezimalwert wandeln
143 FUNCTION Hex2Dez (H$)
144 D = 0
145 FOR i = 1 TO LEN(H$)
146    z = ASC(MID$(H$, i))
147    z = z - 48 - 7 * INT((z - 48) / 17)
148    D = D * 16 + z
149 NEXT i
150 Hex2Dez = D
151 END FUNCTION
152
153 ´*** Textstring als Schmalschrift-Zeichen auf dem Bildschirm ausgeben
154 SUB SPrintString (Text$, Y, X)
155 X2 = X ´Um den Wert von X zu bewahren
156 FOR Stelle = 1 TO LEN(Text$)
157   Zeichen = ASC(MID$(Text$, Stelle, 1))
158   X2 = X2 + 1
159   CALL SchmalPrint2(Zeichen, Y, X2)
160 NEXT Stelle
161 END SUB
```

Kasten 2

Das erste Verfahren der Font-Darstellung verarbeitet die acht Spalten-Bits in einer Schleife.

```
 1 DEFINT A-Z
 2 SUB SchmalPrint1 (Zeichen, Y, X)
 3 ´*** Ein Schmalschrift-Zeichen auf dem Bildschirm ausgeben
 4 ´*** Methode 1:
 5 ´*** Verarbeitung der acht Spalten-Bits in einer Schleife
 6 Zeichen = Zeichen - 32           ´nur ASCII > 32 verwenden
 7 ZE = Zeichen * 4 + 1             ´Offset in den String Font: Ab hier
```

```
 8                                  ´bilden 4 Spalten-Bytes ein Zeichen
 9 ´Die vier Pixelreihen eines Zeichens ermitteln und ausgeben
10 FOR Byte = 1 TO 4
11    A = ASC(MID$(Font$, ZE + Byte, 1))
12    ´A = 255 - A                 ´hiermit wäre Invertieren möglich
13                                 ´ (ggf. REM-Zeichen entfernen)
14    FOR Bit = 0 TO 7
15       IF (A AND 2 ^ Bit) <> 0 THEN Frb = FrbVG ELSE Frb = FrbHG
16       Spalte = X * X1 + Byte: Zeile = Y * Y1 + 7 - Bit
17       PSET (Spalte, Zeile), Frb
18    NEXT Bit
19 NEXT Byte
20 END SUB
```

Kasten 3

Das zweite Verfahren der Font-Darstellung verarbeitet für jedes Bit einer Zeichenspalte eine eigene Befehlsfolge.

```
 1 DEFINT A-Z
 2 SUB SchmalPrint2 (Zeichen, Y, X)
 3 ´*** Ein Schmalschrift-Zeichen auf dem Bildschirm ausgeben
 4 ´*** Methode 2:
 5 ´*** Für jedes Bit einer Zeichen-Spalte eine eigene Befehlsfolge
 6 Zeichen = Zeichen - 32           ´nur ASCII > 32 verwenden
 7 ZE = Zeichen * 4 + 1             ´Offset in den String Font$: Ab hier
 8                                  ´bilden 4 Spalten-Bytes ein Zeichen
 9 ´Die vier Pixelreihen eines Zeichens ermitteln und ausgeben
10 FOR Byte = 1 TO 4
11    A = ASC(MID$(Font$, ZE + Byte, 1))
12    ´A = 255 - A                 ´hiermit wäre Invertieren möglich
13                                ´ (ggf. REM-Zeichen entfernen)
14    IF (A AND 128) <> 0 THEN Frb = FrbVG ELSE Frb = FrbHG    ´Bit 7
15    Spalte = X * X1 + Byte: Zeile = Y * Y1
16    PSET (Spalte, Zeile), Frb
17    IF (A AND 64) <> 0 THEN Frb = FrbVG ELSE Frb = FrbHG     ´Bit 6
18    Spalte = X * X1 + Byte: Zeile = Y * Y1 + 1
19    PSET (Spalte, Zeile), Frb
20    IF (A AND 32) <> 0 THEN Frb = FrbVG ELSE Frb = FrbHG     ´Bit 5
21    Spalte = X * X1 + Byte: Zeile = Y * Y1 + 2
22    PSET (Spalte, Zeile), Frb
23    IF (A AND 16) <> 0 THEN Frb = FrbVG ELSE Frb = FrbHG     ´Bit 4
24    Spalte = X * X1 + Byte: Zeile = Y * Y1 + 3
25    PSET (Spalte, Zeile), Frb
26    IF (A AND 8) <> 0 THEN Frb = FrbVG ELSE Frb = FrbHG      ´Bit 3
27    Spalte = X * X1 + Byte: Zeile = Y * Y1 + 4
28    PSET (Spalte, Zeile), Frb
29    IF (A AND 4) <> 0 THEN Frb = FrbVG ELSE Frb = FrbHG      ´Bit 2
30    Spalte = X * X1 + Byte: Zeile = Y * Y1 + 5
31    PSET (Spalte, Zeile), Frb
32    IF (A AND 2) <> 0 THEN Frb = FrbVG ELSE Frb = FrbHG      ´Bit 1
33    Spalte = X * X1 + Byte: Zeile = Y * Y1 + 6
34    PSET (Spalte, Zeile), Frb
35    IF (A AND 1) <> 0 THEN Frb = FrbVG ELSE Frb = FrbHG      ´Bit 0
36    Spalte = X * X1 + Byte: Zeile = Y * Y1 + 7
37    PSET (Spalte, Zeile), Frb
38 NEXT Byte
39 END SUB
```

Kasten 4

Das dritte Verfahren der Font-Darstellung nutzt BASICs LINE-Befehl für die Darstellung der Zeichen.

```
 1 DEFINT A-Z
 2 SUB SchmalPrint3 (Zeichen, Y, X)
 3 ´*** Ein Schmalschrift-Zeichen auf dem Bildschirm ausgeben
 4 ´*** Methode 3:
 5 ´*** Nutzung des LINE-Befehls zur Bitmuster-Ausgabe
 6 Zeichen = Zeichen - 32           ´nur ASCII > 32 verwenden
 7 ZE = Zeichen * 4 + 1             ´Offset in den String Font$: Ab hier
 8                                  ´bilden 4 Bytes ein Zeichen)
 9 ´Die vier Pixelreihen eines Zeichens ermitteln und ausgeben
10 FOR Byte = 1 TO 4
11    A& = ASC(MID$(Font$, ZE + Byte, 1))
12    ´ A& = 255 - A&               ´hiermit wäre Invertieren möglich
13                                   ´ (ggf. REM-Zeichen entfernen)
14    Muster& = 256 * A&             ´Bitmuster in die höhere Hälfte der
15    IF Muster& > 32767 THEN        ´Variablen, die wir als Struktur-
16       M = Muster& - 65536         ´Parameter für LINE verwenden
17    ELSE
18       M = Muster&
19    END IF
20    ´Zeile nur freischalten, wenn der Hintergrund berücksichtigt werden
21    ´muß, z.B. wenn bereits vorhand. Schmalschrift überschrieben werden soll
22    ´LINE (x * X1 + Byte, y * Y1)-(x * X1 + Byte, y * Y1 + 7), FrbHG
23    LINE (X * X1 + Byte, Y * Y1)-(X * X1 + Byte, Y * Y1 + 7), FrbVG, , M
24 NEXT Byte
25 END SUB
```
