M E T A F O N T

(Version[1] 2.7 / 2.2)

Bedienungsanleitung

Lutz Birkhahn

April 1992

[1]Die erste Versionsnummer ist die offizielle Zählung von Donald E. Knuth, die zweite bezeichnet die aktuelle Implementation auf dem Atari ST

Sechste Auflage.
April 1992

Inhaltsverzeichnis

Kapitel 1

Einführung

Im fünfzehnten Jahrhundert wurde das erste mal versucht, Buchstaben auf mathematischem Wege zu erzeugen. Nach einer Blütezeit im sechzehnten und siebzehnten Jahrhundert wurde diese Methode im achtzehnten Jahrhundert aufgegeben, die Ergebnisse waren einfach zu schlecht. Erst seitdem in der heutigen Zeit Computer die Berechnungen schnell und exakt durchführen können (auch die Mathematik ist während dieser Zeit nicht stehen geblieben), scheint es möglich und vielleicht auch lohnend zu sein, die Mathematik zur Erzeugung von Buchstaben zu verwenden.

Der bekannte Mathematiker und Informatiker Donald E. Knuth hat sich seit 1977 mit dem weiten Gebiet der Typographie beschäftigt, nachdem er die ersten mit Computerhilfe hergestellten Drucke seiner Buchreihe „The Art of Computer Programming“ gesehen hatte. Die erste Auflage dieser Buchreihe war noch im Bleisatzverfahren hergestellt worden, und Knuth war nach einem Vergleich des damals noch neuen Computersatzes mit den herkömmlichen Druckverfahren so enttäuscht von der neuen Technik, daß er sich seine eigenen Gedanken über Typographie in Verbindung mit Computern machte. Ergebnisse dieser Forschungsarbeiten waren das Schriftsatzprogramm TEX78 und METAFONT79, ein Programm für den Entwurf von Schriften für rasterorientierte Ausgabegeräte mit mathematischen Mitteln. Nach einigen Jahren Erfahrung mit diesen beiden Programmen hat er 1982 eine neue Version von TEX und 1984 ein neues METAFONT geschrieben. METAFONT entwickelte er praktisch vollkommen neu, nachdem sich gezeigt hatte, daß der bisherige Ansatz erhebliche Schwächen aufwies.

Dieses neue METAFONT (und auch das neue TEX) hat Knuth in der von ihm erdachten, Pascal-ähnlichen Sprache WEB formuliert und in Buchform [2, 3] veröffentlicht. Das eigentlich neue an WEB ist die Kombination von Programm und Dokumentation in *einer* Datei, wofür Knuth den Begriff „Literarisches Programmieren“ eingeführt hat[5]. Und in der Tat sind die Programme fast so gut lesbar wie ein Roman, und sie verdeutlichen, was Knuth von der „Kunst, zu programmieren“ versteht. Mein Dank geht an Knuth, daß er seine Kunst nicht nur theoretisch in den Büchern „The Art of Computer Programming“ veröffentlicht, sondern sie auch praktisch in den Programmen TEX und METAFONT vorgeführt hat.

Angesichts dieser großen Leistungen war es für Stefan Lindner und mich – als wir im Frühling 1987 auf diese Programme stießen – klar, daß wir sie für den Atari ST implementieren wollten, und daß die angepaßten Programme als Shareware verbreitet werden sollten. Wegen der Form von WEB-Dateien und der Tatsache, daß wir beide keine großen Freunde

von Pascal sind, kam uns beiden sofort der Gedanke, die Programme in die Programmiersprache C zu übersetzen. Also besorgten wir uns die entsprechenden Bücher von Knuth, und tippten sie ab, wobei wir die Programme gleich in C übersetzten. Stefan implementierte TEX, während ich mich auf METAFONT stürzte. Nach über einem Jahr Tippfehlersuche und Ausprobieren verschiedener C-Compiler waren im Sommer 1988 erste brauchbare Ergebnisse zu sehen. Inzwischen hat sich Turbo C von Borland/Heimsoeth als der am besten geeignete Compiler herausgestellt, und es konnten noch einige Verbesserungen an der Benutzeroberfläche von TEX und METAFONT vorgenommen werden. Die vorliegenden Versionen von METAFONT und TEX sind das Ergebnis der langen Bemühungen, Knuths Programme für Besitzer von Atari ST Computern zugänglich zu machen. Beide sind `TRAP`- bzw. `TRIP`-getestet, haben also nachgewiesen, daß sie sich auch bei „unmöglichen“ Eingaben genau wie die Originale verhalten.

1.1 Der Autor

An dieser Stelle möchte ich mich ganz kurz vorstellen. Geboren bin ich anno 1963 (zufälligerweise erhielt Donald Knuth in diesem Jahr auch seinen Doktortitel in Mathematik vom California Institute of Technology), und entdeckte schon in der Schule mein Interesse für Computer, zunächst an einem Tischrechner von WANG, der noch mit einem richtigen Kernspeicher ausgestattet war, und bei dem man mit Hilfe einer teuren Zusatztastatur sogar Buchstaben (!) eingeben konnte, wenngleich diese auch nur selten richtig im Rechner ankamen. Nach diversen Zwischenschritten (CBM 3000, AIM-65, mein erster eigener Computer, und C-64) wurde ich 1985 schließlich stolzer Besitzer eines Atari ST. Neben METAFONT bin ich zur Zeit auch noch mit einem Informatik-Studium beschäftigt.

Meine derzeitige Adresse ist:

Lutz Birkhahn
Fürther Str. 6
W-8501 Cadolzburg 2
Deutschland

Telefon: 0 91 03 / 28 86

Wer seine Registrierungsgebühr zahlen möchte (Spenden werden natürlich auch angenommen), überweist den Betrag am besten auf eines der folgenden Konten:

Kontonr. 3062 25-852 beim Postgiroamt Nürnberg, BLZ 760 100 85

Kontonr. 533 45 37 bei der Vereinigten Sparkasse im Landkreis Fürth, Bankleitzahl 762 501 10

Mittels elektronischer Post (email) bin ich im Subnetz unter `lutz@bisun.nbg.sub.org` erreichbar. Da sämtliche Daten über private Telefonleitungen gehen und teilweise auch noch zusätzliche Übertragungsgebühren kosten (für Senden *und Empfangen*!), bitte keine Riesenbriefe schicken oder zumindest vorher bei mir anfragen.

1.2 Weitere Informationen

In großen Teilen des Z-Netzes und Fidonetzes sowie im Subnetz und Mausnetz gibt es ein TEX-Forum (Z-Netz: /T-NETZ/TEX, Fido: TEX.GER, Subnetz: sub.tex, Maus: Gruppe TeX), das sowohl von Entwicklern und Experten als auch von Benutzern (Anfänger und Fortgeschrittene) von TEX und METAFONT für den Informationsaustausch und Fragen genutzt wird. Wer einen Zugang zu einem der genannten Netze hat und auf dem laufenden bleiben will oder Fragen zu einem der Programme hat, sollte sich auf jeden Fall mal in diesem Forum umsehen. Dieses Forum ist natürlich nicht auf Atari-Benutzer beschränkt.

Wer Zugang zum Internet oder Bitnet (EARN etc.) hat, kommt noch an viel mehr Informationen ran, als Stichworte seien hier nur die TEXhax oder die deutsche Mailingliste tex-d-l@dearn genannt. Wer möchte, kann hier den ganzen Tag damit zubringen, Nachrichten über TEX zu lesen.

Neuerdings hat Stefan Lindner für Modem-Besitzer auch einen eigenen TEX-Server eingerichtet, der etliche Megabytes an TEX-Makros, Zeichensätzen für METAFONT und Programmen (hauptsächlich für den Atari ST) bietet. Die Telefonnummer ist 09 11 / 75 85 47, Parameter 1200–14400/8/N/1, Benutzername `gast`.

Von Stefan Lindner kann man zu ähnlichen Konditionen wie bei METAFONT das berühmte Schriftsatzprogramm TEX (ebenfalls von Donald Knuth entwickelt) zusammen mit DVI-Gerätetreibern für die gebräuchlichsten Drucker bekommen. Seine Adresse lautet:

Stefan Lindner
Iltisstraße 3
8510 Fürth

Telefon: 09 11 / 7 59 18 86

Ebenso sind TEX und METAFONT von der Firma Neumann-Seidel GbR erhältlich, näheres dazu im Abschnitt über die Shareware-Bedingungen auf Seite 5.

Kapitel 2

Shareware

2.1 Shareware-Bedingungen

Beide Disketten dürfen (und sollen) unter den folgenden Bedingungen beliebig kopiert und weitergegeben werden:

- Es werden *alle* Dateien auf der Diskette kopiert.
- Diese Anleitung und die Programme METAFONT und INIMF werden nicht verändert.
- Die Weitergabe erfolgt ausschließlich zu nichtkommerziellen Zwecken.

Kurz gesagt, ich möchte nicht, daß unvollständige Versionen kursieren, oder daß jemand mit diesen Programmen Geld verdient.

Ich kann natürlich für die gesamten Programme und die Daten keine Garantie geben, und auch keine eventuell auftretenden Schäden ersetzen (wenn also z.B. die Augen nicht mehr mitmachen, weil sie durch einen falschen Zeichensatz zu Tode erschreckt sind, dann bitte die Arztrechnung nicht an mich schicken). Ich habe jedenfalls versucht, alles so gut und richtig wie möglich zu machen.

Jeder, dem diese Implementation von METAFONT gefällt, und der die Programme öfter verwendet, wird gebeten, 50,– DM (außerhalb Deutschlands wegen des höheren Portos 55,– DM oder US-$ 30 bar oder Euroscheck oder Order-Scheck) an meine auf Seite 3 angegebene Adresse zu schicken.

Alternativ kann man eine registrierte Version von METAFONT auch von der Firma Neumann-Seidel GbR, Hafenstraße 16 in W-2305 Heikendorf beziehen. Das METAFONT zusammen mit den gedruckten Anleitungen zu METAFONT und zur TEX-Shell kostet dort DM 65,–. Wer Stefan Lindners TEX gleich mitbestellt, bekommt das Komplettpaket mit Anleitungen für DM 129,–.

Jeder registrierte Benutzer erhält folgenden Service:

- Die neueste Version (mit Seriennummer, dazu unten mehr) wird zugeschickt.
- Telefon-Hotline: kostenlose Beratung registrierter Benutzer[1].

[1]Das soll natürlich nicht heißen, daß nur registrierte Benutzer bei mir anrufen dürfen. Über Lob, Kritik, Verbesserungsvorschläge oder gar selbstgeschriebene Programme zu METAFONT oder Ideen dazu freue ich

- Registrierte Benutzer können bei mir für 30,– DM (oder US-$ 20) den kompletten Quellcode (in C) zu METAFONT anfordern, auch hier ist ein Update im Preis enthalten.
- Das erste Update wird kostenlos zugeschickt (wo kriegt man heute noch so einen Service?), alle weiteren Updates werden schriftlich bekanntgegeben, registrierte Benutzer erhalten diese dann gegen voraussichtlich ca. 10,– DM pro Diskette.

2.2 Seriennummer

In jeder METAFONT-Version ist eine Seriennummer enthalten. Wenn jemand (im folgenden A genannt) den oben genannten Betrag bezahlt, erhält er dafür eine Version mit einer neuen Seriennummer. Diese Nummer wird bei mir registriert. Wenn er nun diese registrierte Version weitergibt (und das soll er ja), und einer der Empfänger dieser Kopie zahlt seinerseits die Sharewaregebühr, so erhält A davon 15,– DM (5,– DM ab 1.1.1993) als „Prämie“ für die Verbreitung des Programmes.

Wenn jemand allerdings mehr als dreißigmal die Belohnung kassiert, gehe ich davon aus, daß es sich um einen kommerziellen Software-Vertrieb handelt, und gebe an diesen keine Belohnungen mehr weiter.

Mit diesem Konzept will ich einerseits die Verbreitung von METAFONT etwas beschleunigen, und andererseits versuchen, mögliche Anfangsschwierigkeiten durch persönlichen Kontakt unter den Benutzern zu verringern (damit ein Benutzer, der weit entfernt von mir wohnt, bei Problemen mit METAFONT zunächst zum vermutlich näher gelegenen Bekannten gehen kann, von dem er die Kopie hat, bevor er bei mir anruft, um das Problem zu lösen. Vieles läßt sich eben bei einem persönlichen Gespräch viel leichter und billiger lösen als über eine lange Telefonleitung).

Damit ich die „Prämie“ weiterleiten kann, bitte bei der Registrierung unbedingt die Seriennummer der bisher benutzten Version (erscheint bei jedem Start von METAFONT in der zweiten Zeile auf dem Bildschirm) mit angeben. Falls man bisher noch keine Version besaß (ist mir im Moment allerdings noch unklar, wie man dann an diese Anleitung kommen kann), sollte man dies bitte deutlich vermerken. Am besten druckt man die Datei `FORMULAR.TXT` auf einem Drucker aus und macht die entsprechenden Eintragungen oder sendet das ausgefüllte Formular an die auf Seite 3 angebene email-Adresse.

---

mich immer. Nur kann es passieren, daß ich bei Fragen, zu deren Beantwortung ich selbst erstmal etwas Zeit investieren muß, zunächst frage, ob der Anrufer registriert ist.

Kapitel 3

Installation

Alle Hinweise in diesem Kapitel sind lediglich Vorschläge für Benutzer, die bis jetzt noch wenig Erfahrung mit METAFONT haben. Durch die Verwendung einer „Setup-Datei“ kann man die Programme und Daten fast beliebig auf seinen Speichermedien verteilen.

Da die kompletten Sourcecodes der Zeichensätze insgesamt über 700 KByte Umfang besitzen, war es nötig, einige Zeichensätze mit dem Programm LHARC zu komprimieren. Dieses Programm befindet sich ebenfalls auf Diskette 2. Bezüglich der Weitergabe dieses Programmes verweise ich auf die dortige Anleitung.

3.1 Installation auf Festplatte

Besitzer einer Festplatte sind (nicht nur) bei der Installation fein raus: Sie starten einfach das Programm `INSTALL.PRG` auf Diskette 1. Da dies ein allgemein verwendbares Programm ist, fragt es erst einmal nach einer Datei, in der die genauen Installationsanweisungen stehen. Für METAFONT wählt man hier die Datei `METAFONT.INS` aus. Nun kann man auswählen, ob man nur die Grundinstallation durchführen will oder auch die TEX-Shell installieren will, und ob man die Zeichensätze komprimiert auf der Platte stehen haben möchte oder lieber ausgepackt (schneller, aber dafür mehr Platzbedarf).

Wenn man nun den OK-Button anklickt, wird man noch nach dem Ziel gefragt, wo das METAFONT-System installiert werden soll (hier wird der Pfad angegeben, wo dann das Programm selbst und alle Unterordner stehen sollen). Dann kann man sich zurücklehnen und der Installation zuschauen, nur einmal wird man noch aufgefordert, die zweite Diskette einzulegen.

Man kann übrigens auch die Disketten in eine RAM-Disk oder auf die Festplatte kopieren, dann geht die Installation noch schneller. Man muß dabei nur beachten, daß die Installationsdatei `METAFONT.INS` auch von Platte bzw. RAM-Disk geladen wird. Die Aufforderung nach einem Diskettenwechsel beantwortet man einfach mit einem Tastendruck (es muß **nicht** die Festplatte oder RAM-Disk ausgewechselt werden!).

Wenn das Programm fertig ist, steht METAFONT gebrauchsfertig auf der Platte, die Setup-Datei ist auch bereits an die örtlichen Gegebenheiten angepaßt. Lediglich bei der TEX-Shell müssen noch die Pfade eingestellt werden (siehe dazu die Anleitung der TEX-Shell). Ein weiterer Vorteil des Installationsprogrammes ist die automatische Überprüfung

der Dateien mittels CRC-Prüfsummen, und zwar nicht nur beim Extrahieren (Auspacken) von Dateien aus LHARC- oder Larc-Dateien, sondern auch beim Kopieren.

3.2 Installation auf Disketten-Laufwerk

Für eigene Experimente mit METAFONT, bei denen die CMR-Sourcecodes nicht benötigt werden, kann die Programmdiskette (am besten natürlich eine Sicherheitskopie davon) ohne spezielle Installation verwendet werden. Benutzer, die nur einseitige Laufwerke besitzen, können sich (bei einem Freund mit zweiseitigem Laufwerk) eine Arbeitsdiskette zusammenstellen, auf der nur die Dateien `METAFONT.PRG`, `METAFONT.RSC`, `MFSETUP` und `PLAIN.BSE` stehen. Es bleiben dann noch etwas mehr als 100 KByte übrig, in denen Eingabedateien (`*.MF`) sowie LOG-, GF- und TFM-Dateien untergebracht werden können. Routinierte Diskjockeys oder Besitzer von zwei Laufwerken können in der Setup-Datei auch Laufwerk `B:\` angeben (z.B. bei den `inputpaths`). Wer noch etwas Platz im RAM hat, kann eine kleine Ramdisk (Minimum ca. 50 KByte) installieren, und in der Setup-Datei `logpath`, `gfpath` oder `tfmpath` (auf diese Dateien wird in der Regel während des gesamten Programmlaufes etwas geschrieben) auf die Ramdisk setzen.

Um komplette Zeichensätze für TEX zu erzeugen, muß man schon etwas mehr mit dem Diskettenplatz jonglieren. Nachdem es aber inzwischen möglich ist, auch komprimierte Dateien direkt mit METAFONT zu lesen, ist es durchaus möglich (wenn auch nicht komfortabel), mit nur einem einseitigen Laufwerk zu arbeiten. Wer tatsächlich so enge Platzverhältnisse hat und deswegen mit der Installation nicht klar kommt, möge sich bei mir melden, dann versuche ich, eine arbeitsfähige Verteilung der Dateien zu finden.

Insgesamt braucht man ca. 1 MByte Speicher auf Disketten und evtl. Ramdisk, um das Programm, die Base-Datei und alle Zeichensatz-Sourcen zu speichern. Es würde hier wohl zu weit führen, für jede mögliche Rechnerkonfiguration (davon gibt es schließlich sehr viele verschiedene) die beste Aufteilung anzugeben, schon allein deswegen, weil es auch auf den Anwendungsfall ankommt, was man als *beste* Aufteilung ansehen kann. Ich denke jedoch, daß es mit diesen Informationen jedem Benutzer möglich sein sollte, eine für seine Anwendung passende Verteilung der Daten zu finden (wenn nicht, gibt es ja immer noch die „Telefon-Hotline“ oder andere METAFONT-Benutzer, die solche Arbeiten schon hinter sich haben).

3.3 Wichtige Hinweise

Im folgenden ein paar Hinweise zum Betrieb des Programmes, die sich im Laufe der Zeit angesammelt und keinen besseren Platz in dieser Anleitung gefunden haben.

- Wenn man die Version 2.0 von Tempus verwendet und dort die Parameter abspeichern will, so sollte man das Programm dazu unbedingt vom Desktop aus starten. Wenn man Tempus aus der TEX-Shell (oder einer anderen Shell) startet und dann „`Param. & sichern...`“ aufruft, werden die Parameter *nicht* in Tempus, sondern in der *Shell* abgespeichert! Die Shell kann man nach so einer Aktion unbesorgt in den Desktop-Papierkorb schmeißen. Ab Tempus Version 2.05 ist dieser Fehler beseitigt worden.

- Mit einigen ‘mode’-Definitionen (z.B. `stlaser` für den Atari-Laserdrucker SLM804) in der Datei `ATARI.MF` gibt es manchmal Probleme mit CMR-Zeichensätzen: METAFONT

meldet den Fehler `"! Bad pos..."` und anschließend `"! Strange path..."`. Dieser Fehler liegt nicht an der METAFONT-Implementation, und auch die CMR-Sourcecodes sind korrekt. Eine genauere Untersuchung ergab, daß die negativen '`blacker`'-Werte in der Datei `ATARI.MF` (bzw. `MODES.MF`) schuld sind. Knuth warnt in seinem Buch „Computer Modern Typefaces“ (Computers & Typesetting, Volume E) auf Seite 7 ausdrücklich vor einem `blacker` < 0 :

> “Trouble might also arise if the device-specific parameter called *blacker* is made negative.”

Leider sehen auf manchen Geräten die Zeichensätze nun mal am besten aus, wenn '`blacker`' negativ ist. Deswegen wurde mittels der Datei `CMLOCAL.MF` eine Behandlung des Problems eingeführt. Es wird zwar noch ein Fehler gemeldet, aber zumindest ist der Zeichensatz noch verwendbar, wenn dieser Fehler aufgetaucht ist. Eine genauere Erklärung des Problems und der zur Lösung verwendeten Methode findet man in der Datei `DOC\BAD_POS.DOC`.

Kapitel 4

METAFONT auf dem Atari ST

METAFONT wurde auf dem ST als GEM-Programm realisiert. Sämtliche Textausgaben gehen in ein Text-Fenster, dort werden auch die Eingaben des Benutzers vorgenommen. Ein zweites Fenster dient der Ausgabe von Grafik. Dieses wird aber erst auf Anforderung geöffnet, z.B. mit den METAFONT-Befehlen `openit` oder `showit`. Wenn man bei der Erzeugung eines Zeichensatzes jedes Zeichen sehen will, bevor es in die GF-Datei geschrieben wird, kann man dies mit dem METAFONT-Befehl `screenchars` einschalten. Mit `screenstrokes` kann man jeden einzelnen „Pinselstrich“ auf dem Bildschirm sehen. Für die weiteren Grafikbefehle sei auf das METAFONTbook [1] verwiesen. Für Spezialanwendungen kann man die GEM-Umgebung in METAFONT auch deaktivieren, siehe dazu die Beschreibung der Environment-Variablen `MF_NOGEM` in Abschnitt 4.4.

Man kann die Fenster jederzeit vergrößern, verkleinern oder verschieben, auch während METAFONT beschäftigt ist. Bei längeren Dateioperationen (z.B. Laden der Base-Datei) oder komplizierten Berechnungen kann es jedoch vorkommen, daß METAFONT nicht sofort auf die Aktivitäten des Benutzers reagiert, sodaß man entsprechend länger auf die Maustaste drücken muß. In den meisten Fällen zeigt METAFONT diesen Umstand mit einer „fleißigen Biene“ als Mauscursor an.

Die Fenstergrößen und -Positionen zu Beginn des Programmes können durch verschiedene Environment-Variablen eingestellt werden, siehe dazu Abschnitt 4.4.

Die Arbeit von METAFONT kann man durch gleichzeitiges Drücken der Tasten CONTROL und ALTERNATE unterbrechen, man landet dann in der normalen Fehlerbehandlungs-Routine, wo man z.B. Endlosschleifen untersuchen kann, Variablen abfragen oder das Programm beenden kann. Aus den gleichen Gründen wie im vorigen Absatz muß man die beiden Tasten eventuell etwas länger gedrückt halten. METAFONT unterbricht nämlich nur dann seine Arbeit, wenn es auch in der Lage ist, zusätzliche Eingaben des Benutzers zu verdauen. Trotzdem ist es zum Beispiel bei der Untersuchung von Variablen ratsam, die Aktivitäten vor METAFONT zu „verstecken“, damit man nichts durcheinanderbringt. Dies kann man durch das `hide`-Makro erreichen. Um z.B. den Inhalt der Variablen x anzuzeigen, gibt man folgendes ein:

```
I hide(show x)
```

Während METAFONT auf eine Eingabe wartet, kann man es durch Drücken der ESC-Taste abbrechen. Wenn man in der daraufhin erscheinenden Alarm-Box auf `abort` klickt,

riskiert man allerdings unvollständige GF- und TFM-Dateien!

4.1 Komprimierte Dateien

Wenn man wenig Speicherplatz zur Verfügung hat (und wer hat das nicht?) und dafür eine etwas geringere Geschwindigkeit in Kauf nehmen will, kann man die METAFONT-Eingabedateien in komprimierter Form speichern. METAFONT ist in der Lage, mit dem Programm Larc komprimierte Dateien (`*.LZS`) einzulesen. Dieses Dateiformat ist ähnlich dem bekannten `*.LZH`-Format von LHarc (genauer gesagt ist es ein Vorgänger davon, und neuere LHarc-Versionen sind auch in der Lage, dieses Format zu verwenden), mit dem wesentlichen Unterschied, daß es erheblich schneller expandiert werden kann als `LZH`-Dateien.

Um METAFONT mitzuteilen, welche Larc-Archive es durchsuchen soll, muß ihm lediglich im Suchpfad für die Eingabepfade (`inputpaths`, siehe dazu den Abschnitt 4.2) der Name der `LZS`-Dateien genannt werden.

4.2 Setup-Datei

In der Setup-Datei stehen sämtliche Pfade, auf die METAFONT zugreifen kann. Wenn in der Kommandozeile nichts anderes angegeben wurde, sucht METAFONT nach der Datei `MFSETUP` im aktuellen Ordner. Die Syntax für die Pfad-Definitionen ist bis auf die Bezeichnung der Schlüsselwörter genau die gleiche wie beim TEX von Stefan Lindner. Es sind also zwischen den einzelnen Wörtern (Token) beliebig viele Leerzeichen, Returns, Tabulatoren und Kommentare erlaubt. Kommentare werden mit einem `'%'` eingeleitet und gehen bis zum Zeilenende (wie bei TEX und METAFONT üblich). Auf der linken Seite vom `'='` dürfen folgende Schlüsselwörter stehen (in Klammern ist jeweils angegeben, welchen Pfad METAFONT verwendet, wenn in der Setup-Datei nichts definiert ist). Wenn nichts anderes angegeben ist, gelten die Pfade sowohl für METAFONT als auch für INIMF.

`poolfile` gibt an, von wo INIMF den String-Pool lädt. Dies muß ein gültiger Dateiname sein, er darf also insbesondere nicht mit einem Backslash (\) enden, und die Extension muß bei Bedarf explizit angegeben werden. Dieser Pfad gilt nur für INIMF; METAFONT ignoriert ihn. (`\MF\BASES\MF_POOL`)

`defaultbase` ist die Base-Datei, die METAFONT lädt, wenn der Benutzer nichts anderes angegeben hat. Auch dies muß genauso wie `poolfile` ein gültiger Dateiname sein. (`\MF\BASES\PLAIN.BSE`)

`basepaths` dort wird die Base-Datei gesucht, wenn eine angegeben wurde. Wenn keine angegeben wurde, lädt INIMF gar keine Base-Datei, während METAFONT die unter `defaultbase` genannte verwendet. (`\MF\BASES\`)

`inputpaths` besteht in der Regel aus einer Liste von Pfaden, in denen nach Input-Dateien (`*.MF`) gesucht werden soll. (`\MF\INPUTS\`)

`gfpath` in diesen Ordner werden die GF-Dateien geschrieben. (`\MF\`)

`tfmpath` ist der Ort, wo die TFM-Datei abgelegt wird. (`\MF\`)

`logpath` dorthin kommt die LOG-Datei. (`\MF\`)

`dumppath` ist der Ordner, in den INIMF die Base-Datei schreibt, wenn 'dump' befohlen wurde. (`\MF\BASES\`)

Auf der rechten Seite des `'='` steht jeweils der Pfad, in dem nach der entsprechenden Datei gesucht wird. Dieser Pfad sollte in der Regel absolut angegeben werden, damit METAFONT die Dateien auch dann findet, wenn es von einem anderen Directory oder Laufwerk aus gestartet wurde. Ein Punkt (`'.'`) ist auch ein gültiger Pfad, er bezeichnet das aktuelle Directory. Wenn dies nicht explizit in der Setup-Datei angegeben ist, sucht METAFONT *nicht* automatisch im aktuellen Directory nach einer Datei. Mit Ausnahme von `poolfile` und `defaultbase` (dies sind ja Dateinamen und keine Pfade) dürfen alle Pfade wahlweise mit oder ohne '\' am Ende angegeben werden. Bei `basepaths` und `inputpaths` dürfen mehrere Pfade, jeweils getrennt durch ein Komma, definiert werden. METAFONT durchsucht dann in der vorgegebenen Reihenfolge alle Pfade, bis es die Datei gefunden hat. Jede Pfad-Definition muß mit einem Strichpunkt abgeschlossen werden. Zwischen Groß- und Kleinschreibung wird nicht unterschieden.

Wie in Abschnitt 4.1 bereits angedeutet, kann man bei den `inputpaths` auch komprimierte Dateien (Larc-Archive) angeben, diese müssen die Extension `.LZS` besitzen. Im Prinzip werden die Larc-Archive von METAFONT wie ein Dateiverzeichnis angesehen, allerdings mit der Einschränkung, daß keine Unterverzeichnisse verwendet werden dürfen (im Larc-Archiv dürfen also nur Dateinamen, keine Pfade abgespeichert werden).

Um zu vermeiden, daß der Benutzer mit steigender Anzahl an METAFONT-Zeichensätzen eine undurchschaubar lange Liste an `inputpaths` angeben muß, wurde außerdem die Möglichkeit vorgesehen, sogenannte Wildcards zu verwenden. Dazu können in den `inputpaths` die GEMDOS-üblichen Zeichen Stern und Fragezeichen (`*`, `?`) mit der ebenso üblichen Bedeutung (beliebig viele Zeichen bzw. genau ein Zeichen, nur in letztem Pfad-Teil erlaubt, etc.) verwendet werden. Bei Pfaden mit Wildcards (und nur da) kommt es auch darauf an, ob der Pfad mit einem Backslash (\) abgeschlossen ist oder nicht: Pfade mit Backslash am Ende betreffen nämlich ausschließlich Directories, **keine** Archive. Damit ist es möglich, in einem Verzeichnis, das sowohl Unterverzeichnisse als auch LZS-Archive enthält, mit `...\*.*\` zuerst alle Unterverzeichnisse und dann mit `...\*.LZS` alle Archive auszuwählen, so daß zuerst die ausgepackten Dateien in den Verzeichnissen untersucht werden, und erst, wenn dort nichts gefunden wurde, auf die komprimierten Archive ausgewichen wird.

4.3 Kommandozeile

METAFONT wurde so implementiert, daß es als GEM-Programm vom Desktop aus möglichst einfach gestartet werden kann. Viele Benutzer verwenden jedoch lieber eine Shell, die ihnen manche Tipparbeit ersparen kann. Als erstes sei hier die hervorragende TEX-Shell von Klaus Heidrich, Robert Kießling und Reinhard Maluschka genannt, die dem Paket ebenfalls beiliegt. Mancher bevorzugt aber auch textorientierte Shells, die meist an UNIX oder MS-DOS angelehnt sind. Um die Verwendung solcher Shells (sowohl GEM- als auch textorientierte) zu ermöglichen, wurden einige Optionen definiert, die in der Kommandozeile angegeben werden können. Parameter (*Dateiname* bzw. *Zahl*) dürfen durch ein Leerzeichen von der Option getrennt werden. Verschiedene Optionen *müssen* durch ein Leerzeichen voneinander getrennt werden.

-j Wenn METAFONT seine Arbeit beendet hat, wird normalerweise ein akustisches Weck-Signal ausgegeben und gewartet, bis der Benutzer durch einen Tastendruck signalisiert, daß er wieder aufgewacht ist und alle wichtigen und unwichtigen Mitteilungen auf dem Bildschirm gelesen hat. Wenn man jedoch mit der TEX-Shell oder einer Batchdatei z.B. während der Nacht gleich mehrere Zeichensätze hintereinander erzeugen will, wäre es unpraktisch, wenn man alle zehn Minuten zum Computer rennen müsste, nur um auf eine Taste zu drücken. Das kann der Computer schließlich auch selber machen. Mit der Option `-j` kann man METAFONT mitteilen, daß es die Nacht- (oder Tag-) Ruhe nicht durch Gebimmel unterbrechen soll, und daß es am Ende nicht auf eine Taste warten soll, da sowieso niemand die Kommentare auf dem Bildschirm lesen will.

Wenn während des METAFONT-Laufes ein fataler Fehler oder Speichermangel auftritt, wird die Sache etwas komplizierter. Die folgende Beschreibung kann ohne Probleme übersprungen werden, wenn man beachtet, daß der `nonstopmode` (oder gar `batchmode`) nur dann eingeschaltet sein sollte, wenn METAFONT längere Zeit unbeaufsichtigt (z.B. über Nacht) laufen soll. Falls bei einem solchen „Batchlauf“ Probleme auftauchen, sollte der erste Schritt darin bestehen, den `nonstopmode` wieder auszuschalten (TEX-Shell: siehe Dialogbox im METAFONT-Teil unter „`P zus. Parm.`“)!

Bei fatalen Fehlern und Speicherproblemen (Rückgabewerte 3 und 6) wartet METAFONT auf einen Tastendruck, auch wenn die „Silent-Option“ `-j` aktiviert wurde. Dies läßt sich nur dadurch vermeiden, daß man METAFONT im `nonstopmode` oder `batchmode` laufen läßt (das sind eingebaute METAFONT-Befehle). Das kann zum Beispiel dann sinnvoll sein, wenn man den Rechner über Nacht laufen läßt, und die Shell auch bei fatalen Fehlern (z.B. mehr als 100 normale Fehler, capacity exceeded, aber auch Platte voll, nicht genügend Speicher!) weitermachen soll. Leider ist es bei manchen „fatalen Fehlern“ nicht möglich, die Fehlerursache in die Logdatei zu schreiben (z.B. wenn die Platte voll ist), so daß im `nonstopmode` oder `batchmode` der nur auf dem Bildschirm erscheinende Fehlerhinweis sofort wieder verschwindet. In diesem Fall sollte man (z.B. am nächsten Morgen) den entsprechenden Zeichensatz nochmal unter Aufsicht und ohne `nonstopmode` erzeugen, so daß man die Fehlermeldung auf dem Bildschirm betrachten kann.

Fatale Fehler, die bereits unmittelbar beim Programmstart auftreten (z.B. Fehler in Setup-Datei, Speichermangel) lassen sich jedoch auch mit einem `nonstopmode` oder `batchmode` nicht überspringen, weil die entsprechenden METAFONT-Befehle zu diesem Zeitpunkt noch gar nicht ausgewertet sind. In diesen Fällen ist es aber meistens sowieso nicht sehr sinnvoll, noch weiterarbeiten zu wollen, da sie auch bei jedem weiteren Programmstart wieder auftreten würden.

-s *Dateiname* METAFONT soll die Pfaddefinitionen aus der Datei *Dateiname* lesen. Ohne Angabe dieser Option sucht METAFONT nach der Datei `MFSETUP` im aktuellen Directory. Wenn eine Setup-Datei nicht gefunden wird, dann werden die im vorigen Abschnitt angegebenen Standardpfade verwendet. Bei Verwendung der TEX-Shell in der momentanen Version empfehle ich jedoch, immer die Datei `MFSETUP` zu verwenden, und dort auch alle Pfade zu definieren, sonst kann es zu Problemen im Zusammenspiel der verschiedenen Programme kommen (in zukünftigen Versionen der TEX-Shell wird es vielleicht ein Telepathie-Modul geben, das die Gedanken des Benutzers liest und daraus die gewünschten Pfade ableitet...).

-e *Zahl* Bei Verwendung der TEX-Shell ist es durch diese Option möglich, direkt aus der Fehlerbehandlung in METAFONT durch Eingabe von '`e`' einen Editor zu starten, der sofort die fehlerhafte Datei lädt und in die entsprechende Zeile springt (sofern der Editor ähnlich wie Tempus Dateiname und Zeilennummer in der Kommandozeile akzeptiert). Wie funktioniert nun das Ganze? METAFONT interpretiert die *Zahl* als Adresse im Hauptspeicher und erwartet dort einen Speicherbereich von mindestens 270 Zeichen, der mit dem nullterminierten String '`TEXSHELL`' vorbesetzt ist. Wenn der Benutzer nun eine Fehlermeldung mit '`e`' beantwortet, schreibt METAFONT den Namen der gerade bearbeiteten Datei und durch ein Leerzeichen davon getrennt die aktuelle Zeilennummer in diesen Speicherbereich (dabei wird die Zeichenfolge `"TEXSHELL"` überschrieben). Diese Informationen kann das aufrufende Programm auswerten und anschließend den Editor starten.

-w *Zahl*

-h *Zahl* Diese beiden Werte geben die Breite und Höhe des Grafik-Puffers in Pixeln an. Standard ist 640 mal 400 Pixel. Siehe dazu auch die Beschreibung der beiden Environment-Variablen MF_SCREENWIDTH und MF_SCREENHEIGHT in Abschnitt 4.4. Wenn sowohl Environment-Variable als auch die Kommandozeilen-Option angegeben sind, werden die Werte aus der Kommandozeile verwendet.

Wenn man eine Option eingibt, die METAFONT nicht kennt, wird eine Hilfsseite („Usage“) ausgegeben, in der nochmal die genaue Verwendung der Optionen erklärt ist.

Man kann METAFONT entweder als GEM-Programm starten oder durch Ändern der File-Extension auf `.TTP` oder `.TOS` als TOS-Programm, wobei dann die gesamte Text-Ein/Ausgabe über GEMDOS läuft und somit auch etwas schneller als über GEM. Da die Erkennung des eigenen Programmnamens nicht hundertprozentig sicher ist, kann man das auch über die Environment-Variable MF_NOGEM umschalten, mehr dazu in Abschnitt 4.4.

Zusätzlich zu den eben beschriebenen Optionen kann man bereits in der Kommandozeile eine Startzeile eingeben. Doch dazu erstmal eine Erklärung, was mit Startzeile gemeint ist. Wenn METAFONT vom Desktop aus gestartet wird, meldet es sich mit zwei Sternen ('**'). Damit läßt es erkennen, daß es auf die Eingabe der Startzeile wartet (die normale Eingabezeile beginnt mit einem einfachen Stern). Die Besonderheiten, die für die Startzeile gelten, werden im Kapitel 6 näher erläutert. Diese Startzeile kann man nun bereits in der Kommandozeile angeben, und zwar *nach* den Optionen. Es wäre zwar sehr ungewöhnlich, aber falls die Startzeile zufällig mit einem Minuszeichen ('-') beginnen sollte, muß man diese erstmal durch zwei Minuszeichen von den Optionen abtrennen. Eine Kommandozeile sieht also ganz allgemein so aus:

`METAFONT.PRG` [*Optionen*] [--] [*Startzeile*]

Falls man die Startzeile bereits in der Kommandozeile angibt, sollte man beachten, daß manche Shells zunächst alle Buchstaben in Großbuchstaben umwandeln, während die Kommandos in METAFONT in der Regel klein geschrieben sind (die bekannteste Shell mit diesem Verhalten ist wohl der GEM-Desktop, der allerdings für GEM-Programme normalerweise keine Kommandozeile vorsieht). Also entweder eine „vernünftige“ Shell verwenden oder die Startzeile erst in METAFONT (nach den zwei Sternen) eingeben.

Da in METAFONT der Backslash ('\') als Sonderzeichen definiert ist, darf er in Dateinamen in der Regel nicht verwendet werden (Ausnahmen: wenn METAFONT den Benutzer

explizit zur Eingabe eines Dateinamens auffordert sowie bei der '`-s`'-Option). Um Dateien trotzdem mit einem Pfad versehen zu können, wurde der einfache Schrägstrich ('/') als Ersatzzeichen gewählt (man ist damit auch dem Filesystem von UNIX ein kleines – wenn auch unbedeutendes – Stück näher). Vor dem Öffnen einer Datei wird dieses Zeichen wieder in einen Backslash zurückverwandelt, damit GEMDOS korrekt arbeiten kann. Generell läßt sich dazu aber sagen, daß man zumindest in den Eingabedateien auf Pfadangaben möglichst verzichten sollte, denn portabel ist sowas natürlich nicht. Schon der Nachbar hat auf seinem Atari ST mit sehr großer Wahrscheinlichkeit eine andere Directory-Struktur, ganz zu schweigen von anderen Computersystemen, wo Pfade möglicherweise ganz anders aufgebaut sind.

4.4 Environment-Variablen

Einige grundlegende Einstellungen von METAFONT kann man mit einer Reihe von Environment-Variablen tätigen. In den meisten Shells (z.B. in Gemini) kann man dies mit dem `setenv`-Befehl angeben, es gibt aber auch Programme, die bei jedem Einschalten des Rechners das Environment einstellen.

MF_RESOURCE *Dateiname* METAFONT sucht die Resource-Datei im AES-Standardpfad (meist das aktuelle Verzeichnis). Wenn sie dort nicht gefunden wurde, versucht METAFONT, seinen eigenen Standort (also das Verzeichnis, von dem aus das Programm geladen wurde) ausfindig zu machen und sucht dort nach einem `METAFONT.RSC`. Da es Situationen geben kann, in denen METAFONT seinen Programmnamen und Standort nicht findet, wurde die Möglichkeit geschaffen, den Ort der Resource-Datei über diese Environment-Variable anzugeben.

MF_NOGEM *Zahl* In manchen Fällen kann es sinnvoll oder zumindest wünschenswert sein, wenn METAFONT keine Ausgabe in GEM-Fenster macht, sondern nur die GEMDOS-Ein/Ausgabe verwendet. Wenn diese Environment-Variable auf `"1"` gesetzt ist, oder wenn die Extension des Programmnamens `.TTP` oder `.TOS` ist (und die Suche nach dem eigenen Namen geklappt hat), wird keine einzige AES- oder VDI-Funktion aufgerufen, sämtliche Textein- und ausgaben laufen über GEMDOS, dafür ist allerdings auch keine Grafikausgabe mehr möglich.

MF_TEXTWIND *x y Breite Höhe* Mit dieser Variable kann man die Fensterposition und -größe des Textfensters beim Programmstart angeben. Dabei wird eine Folge von maximal vier durch Leerzeichen voneinander getrennten Zahlen genannt, die nacheinander die gewünschte x- und y-Position sowie Breite und Höhe des Fenster-*Inneren* in Pixeln bedeuten. Werte, die zu klein, zu groß oder gar nicht angegeben sind, werden mit den Maximalwerten belegt: x und y so, daß die linke obere Ecke des Textfensters in der linken oberen Ecke des Desktops liegt, *Breite* und *Höhe* mit den Abmessungen des Desktops (abzüglich des Fensterrahmens), höchstens aber 25 Zeilen bzw. 80 Zeichen (es werden Zeichenbreite und -höhe des aktuellen Systemzeichensatzes zur Berechnung herangezogen).

Zum Beispiel kann man mit `setenv MF_TEXTWIND "0 0 640 200"` angeben, daß das Textfenster anfänglich zwar links oben steht, aber auf dem normalen SM124-Schwarzweiß-Monitor von Atari nur die obere Hälfte des Bildschirms ausfüllt.

MF_GRAPHWIND *x y Breite Höhe x-Offset y-Offset* Für die ersten vier Parameter gilt das für MF_TEXTWIND gesagte, bezogen auf das Grafikfenster. Die Maximalgröße ist die Größe des Grafikpuffers (640 mal 400 bzw. die Werte der nächsten beiden Environment-Variablen).

Die beiden zusätzlichen Zahlen geben den anfänglichen Offset der linken oberen Ecke des Fensters von der linken oberen Ecke des Fensterinhaltes an. Will man beim Programmstart also den Grafik-Puffer ganz links oben sehen, hängt man noch `" 0 0"` oder gar nichts (weil das die Standard-Einstellung ist) an den Environment-String an. `" 100 200"` sagt, daß man die 100 Pixel am linken Rand und die 200 Pixelzeilen am oberen Pufferrand „überspringen“ (d.h. nicht im Fenster sehen) will. Wenn die Werte zu groß sind, wird der maximal mögliche Wert genommen. Man kann also z.B. bei einem 640×400-Puffer die Werte `" 700 500"` angeben, und hat dann auf jeden Fall die rechte untere Grafikpuffer-Ecke im Fenster.

MF_SCREENWIDTH *Zahl*

MF_SCREENHEIGHT *Zahl* Sämtliche Grafikausgaben gehen zunächst in einen internen Puffer, der im Normalfall 640×400 Pixel groß ist. Das Grafikfenster auf dem Bildschirm stellt einen Ausschnitt aus diesem Puffer dar. Wenn man einen Großmonitor verwendet oder oft sehr große METAFONT-Zeichen erzeugt, kann man den Grafikpuffer vergrößern, indem man die Environment-Variablen MF_SCREENWIDTH und MF_SCREENHEIGHT mit der gewünschten Zahl der Pixel (horizontal bzw. vertikal) belegt. Der Nachteil dieser Angelegenheit ist der größere Speicherbedarf (Breite $\times$ Höhe/8 Bytes).

Wenn zusätzlich in der Kommandozeile die Optionen `-w` oder `-h` verwendet wurden, so gelten die dort angegebenen Werte.

ACHTUNG: Mit diesen Variablen ändert man ausschließlich die Ausmaße des Grafikpuffers. Wieviel davon METAFONT für seine Grafikausgabe verwendet, ist eine andere Sache. Bei Benutzung der Plain-Makros müssen deshalb noch die Variablen `screen_cols` und `screen_rows` geändert werden (die Werte werden z.B. in `ATARI.MF` gesetzt), eventuell muß auch noch das `openit`-Makro geändert werden. Am besten liest man nach `ATARI.MF` noch eine weitere Datei, z.B. `LOCAL.MF` ein, in der solche Änderungen untergebracht werden. Letzten Endes sind die im „primitiven METAFONT-Befehl“ `openwindow` angegebenen Werte für METAFONTs Vorstellung von der Größe des Grafikpuffers verantwortlich.

Technischer Hinweis für „Insider“: Da `screen_cols` und `screen_rows` keine in METAFONT eingebauten Variablen sind, kann und vor allem darf METAFONT diese nicht von sich aus verändern. Darum ist es nicht möglich, hier automatisch die verwendete Grafikpuffer-Größe einzutragen. Eine Alternative wäre es, die METAFONT-Syntax um zwei weitere eingebaute Variablen zu erweitern, die bei Programmstart mit der Puffergröße initialisiert werden. Man könnte dann über `everyjob` die Plain-Variablen `screen_cols` und `screen_rows` verändern. Dummerweise gibt es aber keinen Variablennamen, der ausschließlich für METAFONT reserviert wäre. Es kann also bei jedem beliebigen Variablennamen passieren, daß ein Anwender genau diesen Namen schon in seinem Programm verwendet hat, womit die Kompatibilität nicht mehr gewährleistet wäre. Vorschläge hierzu sind sehr willkommen.

MF_EXTCHARSET *Zahl* Wenn die Variable MF_EXTCHARSET auf `"1"` gesetzt wird, dann stellt METAFONT die Umlaute und andere Zeichen oberhalb von 127 nicht mehr in der Ersatzdarstellung `^^`xx (z.B. `^^84` für „ä“) dar, sondern direkt als Umlaut bzw. das entsprechende Sonderzeichen. Diese speziellen Zeichen können allerdings nur in Strings, Kommentaren und in Dateinamen verwendet werden, in allen anderen Fällen beschwert sich METAFONT über ein „`invalid character`“. Achtung: MF_EXTCHARSET wird nur von INIMF ausgewertet und in die Base-Datei geschrieben. Um das Verhalten von METAFONT zu beeinflussen, muß mit INIMF eine entsprechende Base-Datei erzeugt werden.

Wer also die Umlaute auch in der Ausgabe als solche sehen möchte, sollte die Environment-Variable MF_EXTCHARSET auf `"1"` setzen, um dann mit der TEX-Shell durch Anwählen des Buttons „`IniMF`“ eine neue Base-Datei zu erzeugen. Für METAFONT selbst ist es dann völlig unerheblich, wie MF_EXTCHARSET gesetzt ist, solange die richtige Base-Datei verwendet wird.

Für die Experten hier noch eine etwas genauere Erklärung: Normalerweise untersucht METAFONT jedes einzelne eingelesene Zeichen, ob es gültig ist. Nur beim Einlesen von Strings, Dateinamen (nach `input`) und Kommentaren wird dieser Test außer Kraft gesetzt. In diesen drei Fällen kann also praktisch jedes beliebige Zeichen verwendet werden, unabhängig davon, ob mit der Environment-Variablen MF_EXTCHARSET ein erweiterter Zeichensatz aktiviert wurde oder nicht. Erst bei der Ausgabe dieser Zeichen tritt ein Unterschied zu Tage: Ein Standard-METAFONT gibt diese Zeichen in einer Ersatzdarstellung aus, z.B. ein „ä“ als „`^^84`“ (das ist der Hexadezimalcode mit zwei `^` davor). Durch Setzen von MF_EXTCHARSET kann diese Umwandlung unterdrückt werden, so daß auch ein ganz normales „ä“ wieder ausgegeben wird. Allerdings wird diese Ausgabe-Umwandlungstabelle (als Teil des Stringpools) in der Base-Datei abgespeichert, die von INIMF erzeugt wird. Ob also ein „ä“ in der Eingabe als „ä“ oder als „`^^84`“ in der Ausgabe erscheint, hängt einzig und allein davon ab, wie MF_EXTCHARSET gesetzt war, als die Base `PLAIN.BSE` mit INIMF (mit dem Kommando `dump`) erzeugt wurde.

Mutige können sich eine Base-Datei auch direkt mit einem Editor ansehen, um herauszufinden, ob sie Umlaute direkt oder in der Ersatzdarstellung definiert. Dazu muß man nur die 128 Zeichen vor dem String `pencircle` anschauen: enthalten sie die Zeichen „`Çüéâä` ... $\sqrt{}^{n23-}$“, sind die Umlaute aktiviert, ansonsten stehen dort die Ersatzdarstellungen „`^^c0^^c1` ... `^^fe^^ff`“.

ARGV Wird für die erweiterte Kommandozeile (ARGV-Verfahren) benutzt und sollte vom Benutzer nicht verändert werden. Eine genaue Beschreibung findet man in der Datei `EXARG.DOC`, „GEMDOS Extended Argument (ARGV) Specification“ im `DOC`-Ordner.

4.5 Rückgabewerte von METAFONT

Jedes Programm gibt an das aufrufende Programm eine Zahl zurück, den Rückgabewert. Mit Hilfe dieses Wertes kann man mit den meisten Shells eventuell aufgetretene Fehler oder besondere Ereignisse während des Programmlaufes erkennen und entsprechend darauf reagieren. Es ist üblich, daß bei einem fehlerfreien Lauf die Zahl Null zurückgegeben wird.

Da bei der Erzeugung eines Zeichensatzes eine ganze Menge ungewöhnliches passieren kann, wurden für METAFONT sechs weitere Werte definiert (übrigens gelten genau die gleichen Konventionen auch für das TEX von Stefan Lindner). Im folgenden also eine vollständige Liste der Werte, die METAFONT an das aufrufende Programm zurückgeben kann:

0 – no error Es lief alles zur vollsten Zufriedenheit von METAFONT.

1 – warning Während des Programmlaufes wurde mindestens eine Warnung ausgegeben, es wurden jedoch keine ernsthaften Fehler gefunden.

2 – error Mindestens ein Fehler ist aufgetreten, die Arbeit wurde aber korrekt beendet, die erzeugten Daten sind (eventuell beschränkt) verwendbar.

3 – fatal error METAFONT mußte wegen eines fatalen Fehlers abgebrochen werden, die erzeugten Dateien sind höchstwahrscheinlich unvollständig. Dies kann z.B. passieren, wenn METAFONT keine oder nur eine fehlerhafte Base-Datei finden konnte, wenn mehr als 100 normale Fehler aufgetreten sind, ein interner Fehler[1] entdeckt wurde oder der interne Speicher (siehe Speicherverwaltung) übergelaufen ist.

4 – edit Der Benutzer hat bei einem aufgetretenen Fehler oder einer Unterbrechung (Interrupt) ein '`e`' eingetippt. Wenn mittels der '`-e`'-Option ein Speicherbereich spezifiziert wurde, schreibt METAFONT in diesen den Namen der aktuellen Eingabedatei und die Zeilennummer.

5 – exit Wenn der Benutzer in der Fehlerbehandlung ein '`x`' eintippt oder während einer Eingabe die Escape-Taste betätigt, dann wird dieser Wert zurückgeliefert.

6 – low memory Zeigt an, daß der freie Hauptspeicher (RAM) nicht ausreicht, um METAFONT starten zu können (siehe Speicherverwaltung). Abhilfe schafft nur entweder das Entfernen nicht benötigter residenter Programme oder der Kauf von mehr Speicher.

4.6 Speicherverwaltung

Beim Programmstart fordert METAFONT einen konstanten Speicherbereich („interner Speicher“) vom Betriebssystem an, in den es alle Variablen speichern kann. Dieser Speicher ist in verschiedene Bereiche aufgeteilt (z.B. für Strings, Eingabepuffer, verschiedene Kernings, „main memory“ für Zahlen, Pfade, Pens, Transformationsmatrizen usw.). Wenn das Betriebssystem diesen Speicher nicht zur Verfügung stellen kann (wegen residenter Programme, RAM-Disk o.ä.), wird der Rückgabewert 6 zurückgeliefert. Wenn METAFONT jedoch während der Arbeit feststellt, daß einer der Bereiche im internen Speicher zu klein ist (z.B. bei endlos rekursiven Makros, zu vielen Strings, zu langen Zeilen, ...), so meldet es „`capacity exceeded`“ zusammen mit der Angabe, welcher Bereich gesprengt wurde. In diesem Fall kann der Benutzer nur versuchen, sparsamer mit den Ressourcen umzugehen (das sagt sich natürlich leicht, aber es ist kaum möglich, hier konkretere Hinweise zu geben; ein paar Tips kann man noch im METAFONTbook[1] von Knuth finden). Im äußersten Notfall kann man auch bei mir anfragen, ob es möglich ist, den entsprechenden Bereich zu

[1] ist zwar unwahrscheinlich, aber nie völlig auszuschließen! In diesem Fall bitte ich um sofortige Benachrichtigung zusammen mit einer möglichst genauen Fehlerbeschreibung

vergrößern; das hätte jedoch auf jeden Fall eine Änderung des Programmes zur Folge, so daß man auf diese Möglichkeit nur zurückgreifen sollte, wenn man sich sicher ist, daß eine andere Lösung nicht möglich oder nicht vertretbar ist.

Für „Insider“ sind in Tabelle 4.1 die Werte der wichtigsten Konstanten von METAFONT angegeben, so wie sie in dieser Implementation gewählt wurden. Eine ausführliche Erklärung dieser Variablen findet man in [2].

main memory size	*mem_max* = *mem_top*	65 534
	mem_min = *mem_bot*	0
number of internals	*max_internal*	100
buffer size	*buf_size*	2000
error messages	*error_line*	72
	half_error_line	42
text output width	*max_print_line*	79
graphics screen size	*screen_width*	variabel (640)
	screen_depth	variabel (400)
input stack	*stack_size*	30
number of strings	*max_strings*	2 500
string pool size	*pool_size*	40 000
	string_vacancies	8 000
GF file buffer	*gf_buf_size*	800
file names	*file_name_size*	250
TFM header words	*header_size*	100
ligature/kern steps	*lig_table_size*	5000
distinct kern amounts	*max_kerns*	500
fontdimen parameters	*max_font_dimen*	50
symbolic tokens	*hash_size*	2 100
input files	*max_in_open*	12
macro parameters	*param_size*	150

Tabelle 4.1: METAFONT-Konstanten

Kapitel 5

INIMF

Auf der Programmdiskette befinden sich zwei unterschiedliche Versionen von METAFONT: `METAFONT.PRG` und `INIMF.PRG`. Dabei ist ersteres für die tägliche Arbeit gedacht; INIMF wird im allgemeinen nur zur Erzeugung von Base-Dateien (`.BSE`) benötigt. Wer bereits die Format-Dateien (`.FMT`) von TEX kennt, kann den nächsten Absatz überspringen, denn die Base-Dateien bei METAFONT entsprechen genau den Format-Dateien bei TEX.

Beim Programmstart müssen sehr viele Variablen initialisiert und der sogenannte „String-Pool“ (Datei `MF_POOL`) mit allen vordefinierten Strings eingelesen werden. Außerdem wird fast jeder Benutzer zumindest die Makros aus der Datei `PLAIN.MF` verwenden, da die sogenannten „primitives“ – also die Befehle des „nackten“ METAFONT ohne Plain-Makros – so primitiv sind (wie der Name schon sagt), daß man sehr viel tippen müßte, um nur ein paar einfache Sachen auf den Bildschirm zu bringen, geschweige denn einen ganzen Zeichensatz. Die „primitives“ könnte man auch als die „Maschinensprache“ von METAFONT bezeichnen. Wenn METAFONT bei jedem Programmstart erst diese ganzen Initialisierungen ausführen und Makrodefinitionen einlesen müßte, würde das recht lange dauern. Darum hat Knuth eine Möglichkeit vorgesehen, diese Initialisierungen und Definitionen nach einmaliger Ausführung in kompakter Form auf Diskette bzw. Platte abzuspeichern. Diese Datei wird Base-Datei genannt. Wenn man das einmal erledigt hat, braucht das Programm in Zukunft nur noch die Base-Datei einzulesen, und ist danach sofort startbereit.

Der Unterschied zwischen INIMF und METAFONT besteht im wesentlichen darin, daß bei METAFONT die komplette Variablen-Initialisierung und das Lesen des String-Pools entfernt wurde, wodurch es nicht nur schneller, sondern auch um einiges kürzer als INIMF ist. Aus diesem Grund braucht es aber unbedingt eine Base-Datei, um arbeiten zu können. Und zur Erzeugung so einer Base-Datei braucht man eben INIMF, auch deshalb, weil der Befehl zum Schreiben der Base-Datei (`dump`) im normalen METAFONT gar nicht vorhanden ist.

Ein weiterer Unterschied besteht in der Berechnung einiger „statistischer“ Werte während der Arbeit. Wenn man z.B. die Variable *tracingstats* auf einen positiven Wert setzt, so erhält man am Ende der LOG-Datei einige Informationen darüber, wie stark der interne Speicher ausgenutzt wurde. Das Mitführen und Aktualisieren solcher Zusatzinformationen kostet jedoch Zeit und Speicherplatz, deshalb wurden die entsprechenden Code-Teile in `METAFONT.PRG` entfernt. Wenn man diese Informationen benötigt, muß man auf INIMF zurückgreifen, muß dafür aber etwas mehr Zeit und Speicher einkalkulieren. An dieser Stelle sei nochmal betont, daß man im Prinzip auch mit INIMF allein auskäme; alles, was man mit

METAFONT machen kann, funktioniert auch mit INIMF (vorausgesetzt, man hat genügend Speicher zur Verfügung). Insbesondere kann man auch bei INIMF eine Base-Datei laden (z.B. durch Eingabe von '`&plain`' in der Startzeile). Lediglich wenn man keine Base-Datei in der Startzeile angibt, lädt METAFONT die in der Setup-Datei angegebene `defaultbase`, während INIMF in diesem Fall gar keine Base-Datei lädt, sondern nur seine Variablen initialisiert und den String-Pool liest (nicht aber die Plain-Makros!).

Abschließend nochmal eine kurze Zusammenfassung der Besonderheiten von INIMF:

- alle Variablen werden beim Programmstart initialisiert;
- der String-Pool wird gelesen (der Dateiname steht in der Setup-Datei unter `poolfile`);
- der Befehl `dump` erzeugt eine Base-Datei;
- wenn in der Startzeile keine Base-Datei angegeben wird, wird auch keine geladen;
- es werden zusätzliche Statistiken geführt. Diese erhält man, wenn *tracingstats* > 0 bzw. *tracingedges* > 1;
- es wird mehr Hauptspeicher benötigt (für zusätzlichen Programmcode und die statistischen Informationen);
- es wird mehr Rechenzeit „verbraten“.

Kapitel 6

Erste Schritte mit METAFONT

„Aller Anfang ist schwer“, noch dazu, wenn man so komplexe und flexible Programme wie TEX oder METAFONT vor sich hat. Um dem Anfänger dennoch einen ganz kleinen Ausschnitt der Möglichkeiten von METAFONT zu zeigen und ihm gleich am Anfang ein paar kleine Erfolgserlebnisse zu ermöglichen, sind in diesem Kapitel ein paar Beispiele aufgeführt, die am besten gleich am Rechner ausprobiert werden sollten. Für die ersten beiden Beispiele kann METAFONT direkt von der Programmdiskette aus gestartet werden, das dritte Beispiel ist am einfachsten mit einem fertig installierten METAFONT durchzuführen. Bevor ich's vergesse: beenden kann man das Programm durch Eingabe von '`end`' oder '`bye`', genauso wie TEX also (wenn man mal davon absieht, daß man in METAFONT kein '\' vor die Befehle setzen muß).

6.1 Beispiel 1: einfache Grafik

Im ersten Beispiel sollen ein paar Pinselstriche auf den Bildschirm gebracht werden. Zunächst muß METAFONT gestartet werden, z.B. durch Anklicken von `METAFONT.PRG` auf dem Desktop (keine Angst, ich erkläre jetzt nicht, wie man mit GEM umgeht). Nach der üblichen Titelzeile meldet sich das Programm mit zwei Sternchen. Normalerweise kann man hier angeben, welchen Zeichensatz man erzeugen möchte, und noch viele andere Dinge. Für dieses Beispiel jedoch kann sich METAFONT „ganz entspannt zurücklehnen“ und auf die Eingabe warten, also tippt man

```
\ relax
```

ein. Zunächst muß METAFONT mitgeteilt werden, daß man die gezeichneten Striche gleich auf dem Bildschirm sehen möchte. Dies erreicht man durch Eingabe von

```
screenstrokes;
```

(den Strichpunkt nicht vergessen!). Zum Zeichnen einer Linie ist die Angabe von zwei Punkten nötig, wobei die Koordinaten eines Punktes als `(x,y)` eingegeben werden können:

```
draw (10,10)..(50,30);
```

Wenn man mehr als zwei Punkte angibt, wird eine Kurve durch alle Punkte gezogen (sogenannte Bezier-Kurven), z.B.

```
draw (10, 100)..(50,80)..(90,100);
```

Das ist natürlich nur die einfachste Methode, zwei oder mehr Punkte mit einer Kurve zu verbinden. METAFONT kann noch sehr viel mehr, z.B. kann man für bestimmte Punkte der Kurve eine Richtung vorgeben, mit „Kontrollpunkten“ die Form der Kurve fast beliebig verändern, Schnittpunkte mit anderen Kurven definieren usw. Experimentieren kann man auch mit speziellen Verbindungen, indem man die zwei Punkte '..' durch '...' oder '--' ersetzt. Um das Grafikfenster zu löschen, kann man

```
clearit; showit;
```

eingeben. `clearit` löscht dabei den Bildpuffer, `showit` bringt den gelöschten Puffer dann auf den Bildschirm.

6.2 Beispiel 2: METAFONT als „Rechner“

Im zweiten Beispiel sollen die mathematischen Fähigkeiten von METAFONT etwas ausprobiert werden. Zunächst kann man in METAFONT (wie in fast jeder anderen Programmiersprache auch) Variablen definieren. Man hätte das erste Beispiel auch so eingeben können:

```
x1 = 10;  y1 = 10;  x2 = 50;  y2 = 30;
draw (x1, y1)..(x2, y2);
```

Dabei steht das Gleichheitszeichen nicht für eine Zuweisung, sondern für eine Gleichung. Der Unterschied wird klar, wenn man z.B. den Wert von $x1$ um eins erhöhen will. Schreibt man

```
x1 = x1 + 1;
```

so beschwert sich METAFONT zu Recht, daß diese Gleichung „inkonsistent“ (d.h. nicht lösbar) ist, denn es gibt nun mal keine reelle Zahl, deren Wert bei Addition von eins unverändert bleibt. Will man eine Zuweisung erreichen, muß man ':=' eingeben. Richtig wäre also in diesem Fall:

```
x1 := x1 + 1;
```

Interessanter ist aber wohl die Verwendung von Gleichungen. Lineare Gleichungssysteme sind für METAFONT was ganz alltägliches. Man hätte die obigen Werte also auch so definieren können:

```
 x1 +  x2 = 60;
3x1 + 2x2 = 130;
```

Man kann leicht nachprüfen, daß METAFONT dieses Gleichungssystem korrekt gelöst hat, wenn man

```
show x1, x2;
```

eingibt. Interessant ist auch, daß man nicht '`3*x1`' schreiben muß, sondern den Malpunkt einfach weglassen kann, so wie es wohl (nicht nur) jeder Mathematiker gewohnt ist. Es gibt nur sehr wenige andere Programmiersprachen, in denen solche Selbstverständlichkeiten möglich sind. Es bedarf wohl keiner besonderen Erwähnung, daß für METAFONT auch die Rechenregel „Punkt vor Strich" nichts Unbekanntes ist. Ansonsten bietet METAFONT unter anderem noch trigonometrische Funktionen, approximative (d.h. näherungsweise) Lösung von nichtlinearen Gleichungssystemen, Vektor- und Matrizenrechnung. Durch die Möglichkeit, Makros zu definieren (was im Prinzip den Unterprogrammen oder Prozeduren herkömmlicher Programmiersprachen entspricht), kann man natürlich noch beliebig viele andere mathematische Rechenverfahren implementieren.

Wer mehr über METAFONT erfahren möchte, sollte sich eines der im Literaturverzeichnis genannten Bücher zu Gemüte führen. Auch das genaue Studium fertiger Zeichensätze oder Logos ist wichtig, anfangen sollte man mit einfacheren Dingen wie z.B. das METAFONT-Logo im Verzeichnis `MF_LOGO` oder das DFF-Logo von Jürgen E. Günther, `DFF.MF`. Auch die vielen anderen kleinen Beispiele im `INPUTS`-Ordner sind in der Regel relativ gut verständlich. An das Studium der CMR-Zeichensätze sollte man sich erst später heranwagen, denn diese Zeichensätze sind in einem Zeitraum von etlichen Jahren immer wieder verbessert und verfeinert worden, so daß der Blick meistens durch viele Details verstellt wird.

6.3 Beispiel 3: Erzeugung eines Zeichensatzes

Zum Schluß dieses Kapitels soll noch erklärt werden, wie man mit METAFONT komplette „Computer Modern Roman" Zeichensätze erzeugen kann. Das ist wahrscheinlich eine der häufigsten Aufgaben dieses Programmes, schließlich wollen (und können) nicht alle Benutzer ihre eigenen Zeichen entwerfen, darum greift man in der Regel auf die fertigen Zeichensätze von Knuth zurück, die ja auch eine lange Entwicklungszeit hinter sich haben und deshalb wenigstens halbwegs professionell aussehen (dennoch hat Knuth nie behauptet, ein Typograph zu sein, er hat sich aber bei der Entwicklung der Zeichensätze von einigen Experten dieses Faches beraten lassen).

Als erstes sollte man nachschauen, ob in der Datei `ATARI.MF` ein 'mode_def' für den verwendeten Drucker vorhanden ist und wie dieser heißt. Wenn dort der gewünschte Drukker nicht angegeben ist, sollte man nach einem 'mode_def' suchen, dessen Auflösung der gewünschten möglichst nahe kommt oder gleich ist. In diesem Fall wird man wohl früher oder später nicht darum herum kommen, sich einen eigenen Parametersatz zu definieren. Vorher sollte man aber bei mir oder Stefan Lindner anrufen, ob vielleicht inzwischen schon fertige Parameter existieren, es kommen fast täglich neue hinzu.

Als Beispiel soll hier der Zeichensatz '`cmr10`' in 1,44-facher Vergrößerung für einen Laserdrucker erzeugt werden, sodaß er in einem TEX-Dokument mit

```
\font\bigtenrm = cmr10 scaled \magstep2
```

angesprochen werden kann. Zum Üben oder schnellen Ausprobieren kann man alternativ auch einen kleinen Zeichensatz wie '`logo10`' erzeugen, dazu muß im folgenden Text nur `cmr` durch `logo` ersetzt werden.

Erzeugen der Base-Datei

Zunächst muß man dafür sorgen, daß eine geeignete Base-Datei vorhanden ist. Für dieses Beispiel genügen die Plain-Makros, also wird die Datei `PLAIN.BSE` benötigt. Es ist zwar auf der Programmdiskette schon eine fertige Datei vorhanden, dennoch soll hier kurz erläutert werden, wie diese Datei erzeugt wurde (schließlich fallen solche Dateien ja nur äußerst selten vom Himmel).

Wie in Kapitel 5 bereits erwähnt wurde, benötigt man für diese Aufgabe das Programm INIMF. Man startet also `INIMF.PRG` und gibt in der Startzeile (durch '**' gekennzeichnet)

```
plain
```

ein. Daraufhin werden die Plain-Makros geladen und im Speicher abgelegt. Wenn wieder ein '*' erscheint, ist METAFONT zu weiteren Untaten bereit. Jetzt kann man die eigenen Dateien laden, die damit bei jedem Start von METAFONT automatisch mitgeladen werden. Für den Anfang also mal

```
input atari
```

um einige Werte wie Auflösung, Strichstärke etc. für ein paar übliche Ausgabegeräte (z.B. `stscreen`, `stlaser` und `starnl`) zu definieren. Für den Anfang genügt das mal, man kann INIMF mitteilen, daß es nun seinen Speicher „dumpen" soll:

```
dump
```

Zur Belohnung erhält man ein paar neue Dateien:

`PLAIN.BSE` in dem Ordner, der in der Setup-Datei unter `dumppath` angegeben wurde. Das ist die Base-Datei, in der alle wichtigen Daten für `METAFONT.PRG` enthalten sind.

`PLAIN.LOG` im `logpath`-Ordner. In diese Datei hat INIMF sämtliche Bildschirm-Ausgaben und noch einiges mehr mitprotokolliert, damit man sich auch später noch ansehen kann, was das Programm alles angestellt hat.

Die Startzeile

Diese Base-Datei kann man nun mit jedem der beiden Programme (INIMF und METAFONT) ruckzuck einladen, indem man in die Startzeile

```
&plain
```

(man beachte das '`&`'!) eintippt. Dadurch wird die Datei `PLAIN.BSE` geladen, und man hat alles wieder so, wie es vor dem Eintippen von `dump` war.

Jetzt wird es aber doch allerhöchste Zeit, daß genau erklärt wird, was die Startzeile ist, und welche Besonderheiten sie aufweist. Die Startzeile ist die erste Zeile, die man für METAFONT eingibt. Eine Möglichkeit besteht bereits in der Kommandozeile *nach* den Optionen. Wenn dort nichts eingegeben wurde, meldet sich das Programm mit zwei Sternchen ('**') am Zeilenanfang, um anzudeuten, daß es die Startzeile erwartet (in den normalen Zeilen steht nur *ein* Sternchen am Zeilenanfang). In dieser Startzeile muß nun angegeben werden, welche Base-Datei geladen werden soll (bei INIMF optional), außerdem leitet METAFONT aus dieser Zeile den „Jobname" ab, nach dem sämtliche Ausgabedateien (GF-, TFM- und LOG-Datei) benannt werden. Die Besonderheiten der Startzeile sind:

- Eine Base-Datei läßt sich durch Voranstellen eines '`&`' vor dem Dateinamen laden (z.B. '`&plain`' lädt die Base-Datei '`PLAIN.BSE`'). Wenn eine Base-Datei angegeben ist, muß diese auf jeden Fall am Anfang der Startzeile stehen.

- Eingabedateien (`*.MF`) kann man einfach durch Angabe des Dateinamens laden, während man in den normalen METAFONT-Eingabezeilen '`input` *name*' schreiben müsste (z.B. '`cmr10`' lädt die Eingabedatei '`CMR10.MF`').

- Wenn man in dieser Zeile andere Sachen eingeben möchte, z.B. Initialisierungen irgendwelcher Variablen, bevor eine Datei gelesen wird, muß man mit einem '`\`' auf den normalen Eingabemodus umschalten. Danach kann man alles machen, was man normalerweise (in '`*.MF`'-Eingabedateien oder in Zeilen mit nur einem '$*$' am Anfang) machen kann. Üblicherweise gibt man hier das Ausgabegerät und eventuelle Vergrößerungen an, z.B.

  ```
  \ mode=stlaser; mag=1.44;
  ```

  setzt Auflösung etc. auf die Werte, die im '`mode_def stlaser`' in der Datei `ATARI.MF` angegeben wurden, und vergrößert den Zeichensatz auf 1.44-fache Größe. Will man mit diesen Werten einen Zeichensatz erzeugen, so müssen die Zuweisungen natürlich *vor* dem Laden der Sourcedatei ausgeführt werden (es nützt nichts, wenn METAFONT zuerst den ganzen Zeichensatz erzeugt und erst hinterher erfährt, daß dieser vergrößert werden sollte). Da man nach Eingabe des `\` nicht mehr im „Startzeilen-Modus“ ist, muß man die Sourcedatei nun (wie üblich) mit `input` laden.

- Anhand der Startzeile bestimmt METAFONT den Jobnamen. Dabei richtet sich METAFONT nach der ersten Datei, die eingelesen wird (ohne Beachtung einer eventuell angegebenen Base-Datei), also entweder der erste Dateiname ohne einem '`&`' davor, oder (falls vor dem '`\`' keine `MF`-Datei eingelesen wird) die erste Datei, die nach einem '`\`' mit `input` eingelesen wird. Wenn mit den Kommandos in dieser Zeile überhaupt keine `MF`-Datei eingelesen wird, dann setzt METAFONT den Jobname auf '`MFPUT`'. Wenn man also

  ```
  &plain f1 \ input f2
  input f3
  ```

  eingibt, heißen die Ausgabedateien '`F1.`xxx`GF`', '`F1.LOG`' und '`F1.TFM`', weil '`F1.MF`' die erste Datei ist, die gelesen wird (die Base-Datei wird nicht mitgerechnet, sonst hieße ja alles '`PLAIN.`xxx').

Wenn man in der ersten Zeile kein Ausgabegerät spezifiziert, dann verwendet METAFONT '`mode = proof;`', was zwar zum Ausprobieren ganz nett aussieht (eigentlich sollte sich dieses Schauspiel niemand entgehen lassen, man sieht die Zeichen mal richtig groß und in hervorragender Schärfe auf dem Monitor), aber für einen Gerätetreiber in der Regel nicht so ideal ist (schon allein wegen der Auflösung von 2601.72 dpi). Also muß man das gewünschte Gerät explizit angeben, indem man in der ersten Zeile vor dem Namen der Eingabedatei '`mode=stlaser;`' angibt. Damit nun aber METAFONT nicht nach der Eingabedatei '`mode`' sucht, ist vor der Gleichung noch ein '`\`' nötig. Wenn keine Vergrößerung angegeben wird, ist sie automatisch 1. Allgemein wird die erste Zeile also ungefähr so aussehen:

```
&<Base> \ mode=<Geraet>; mag=<Vergroesserung>; input <Datei>;
```

wobei für die Namen in spitzen Klammern entsprechende Werte einzusetzen sind (die Klammern tippt man natürlich nicht mit ein).

Die „Schöpfung“

Jetzt steht alles bereit, um den Zeichensatz zu erzeugen. Man starte also das Programm `METAFONT.PRG` und gebe als Startzeile

```
&plain \ mode=stlaser; mag=1.44; input cmr10
```

ein. Nach einigen Minuten ist METAFONT fertig, und man hat ein paar weitere Dateien auf der Diskette / Platte:

`CMR10.432` sollte eigentlich `CMR10.432GF` heißen, aber TOS kann halt leider nur drei Zeichen in der Extension speichern. Die „krumme“ Zahl 432 erhält man, wenn man die Geräte-Auflösung von 300 dpi (dots per inch) mit der Vergrößerung (in diesem Fall 1.44) multipliziert. Das ist also nun die GF-Datei, in der die Beschreibung der vielen Zeichen drin ist. Wenn man einen Zeichensatz für ein anderes Gerät generiert hat, steht natürlich nicht 432, sondern die gewählte Auflösung (Vergrößerung nicht zu vergessen) in der Extension, z.B. `CMR10.96G` (‘`.96GF`’ auf 3 Zeichen verkürzt!) für eine GF-Datei mit 96 dpi und Vergrößerung 1.

`CMR10.LOG` auch hier hat METAFONT wieder fein säuberlich mitprotokolliert, was es alles gemacht hat.

`CMR10.TFM` diese Datei benötigt TEX, um zu erfahren, wie groß die einzelnen Zeichen sind (wie sie tatsächlich aussehen, interessiert TEX gar nicht, das geht einzig und allein den Druckertreiber etwas an). TFM steht für „TEX Font Metric File“.

Da die meisten Gerätetreiber sogenannte PK-Files („Packed Font Files“) bevorzugen, muß man nun noch die GF-Datei in eine PK-Datei umwandeln (man könnte auch komprimieren sagen). Man startet also `GFTOPK.TTP` und gibt diesem den Namen der GF-Datei und der gewünschten PK-Datei, also z.B.

```
cmr10.432 \prtfonts\res300.slm\mag____1.440\cmr10.pk
```

als Kommandozeile mit auf den Weg (zum Zwecke der Demonstration sei hier mal davon ausgegangen, daß die GF-Datei im aktuellen Ordner und die Zeichensätze für die Druckertreiber im Ordner `\prtfonts` stehen, wobei letzterer entsprechend den Konventionen von Stefan Lindner’s Druckertreiber-Familie aufgebaut ist). Damit ist der Zeichensatz fertig und steht zum Ausdrucken bzw. Betrachten auf dem Bildschirm bereit.

Kapitel 7

Zukunftsmusik

Die Arbeit an METAFONT ist noch nicht abgeschlossen, da gibt es noch viel zu viele Ideen, was man besser und schöner machen könnte, und der eine oder andere Fehler läßt sich vermutlich auch noch in der Benutzeroberfläche finden. Sogar im METAFONT selbst kann man heute noch Fehler finden, man versuche nur mal, in einer Feld-, Wald- und Wiesen-Implementation das INIMF zu starten, keine Base-Datei zu laden, und `\showstats` einzugeben. In 90% aller Versionen (natürlich nicht in dieser) wird man „`Memory usage 23&-1`“ zur Antwort bekommen. Und -1 verbrauchte Speicherzellen ist schon etwas ungewöhnlich (keine Sorge, der Fehler ist inzwischen an den „Chef“ weitergegeben).

Daß bei der Texteingabe kein Cursor zu sehen ist, stört manchmal schon gewaltig. Irgendwann wird auch das sicherlich in den Griff zu bekommen sein. Die Gedanken kreisen da allerdings viel weiter: wenn man die Eingabe über allgemeine Textfenster (wie sie heutzutage in jeder besseren grafischen Benutzeroberfläche enthalten sind, z.B. in X oder in SunView) gestalten würde, hätte man enorm viel mehr Möglichkeiten, angefangen bei einfachen Editorkommandos (Cursor-Bewegung, Zeichen löschen) bis hin zu Cut/Paste-Operationen über mehrere Fenster (und damit mehrere Dateien) hinweg. Eventuell könnte man das auch über höhere XACC-Levels in Zusammenhang mit einem einfachen Accessory-Editor, der auch das XACC-Protokoll versteht, erledigen. Wer einen solchen Editor kennt, der noch dazu frei kopierbar sein sollte (PD- oder Shareware), möge mich bitte gleich benachrichtigen.

Schön wäre es auch, wenn man die Setup-Datei von METAFONT aus ändern könnte. Ich stelle mir da einen Eintrag in der Menüleiste vor, ähnlich wie bei vielen anderen GEM-Programmen, wo man die Standard-Pfade angeben und anschließend abspeichern kann.

Wenn die Larc-Bibliothek mal ausgereift ist, dann wird sicherlich auch in METAFONT die Behandlung der archivierten Dateien konsistenter und natürlicher werden. Letzten Endes ist ein Larc-Archiv ja nichts anderes als ein Directory, komplett mit Unterverzeichnissen und Dateien, und so sollte auch der Benutzer keinen Unterschied machen müssen (leider läßt sich das nicht so einfach auf den Desktop ausweiten, denn da muß der Benutzer immer noch selbst die entsprechenden Archivier-Programme aufrufen, anstatt sie wie normale Directories über Desktop-Fenster zu behandeln). In diesem Zusammenhang ist auch nicht einzusehen, warum METAFONT keine mit Pfad abgespeicherten Dateien in den Archiven duldet.

Ein sicher sehr interessantes, aber auch aufwendiges Projekt wäre, METAFONT mit der hervorragenden Benutzeroberfläche von SMALLTALK-80 zu verbinden. Ein erster Ansatzpunkt dazu ist die Möglichkeit, C-Funktionen als „primitive“ Methoden zu SMALLTALK-80

dazulinken zu können (und im Prinzip ist METAFONT nichts anderes als eine C-Funktion namens `main`). Benutzern mit weniger als 4 MByte RAM würde das allerdings vermutlich nicht sehr viel bringen (höchstens den Wunsch nach mehr Speicher).

Literaturverzeichnis

[1] Knuth, Donald E. *The METAFONTbook*, Computers and Typesetting Vol. C, Addison-Wesley, Reading (Massachussets) 1986. 361 Seiten.

Die definitive Anleitung zu METAFONT. Auf dieses Buch kann wohl nur verzichten, wer ausschließlich „Computer Modern Roman“-Zeichensätze für verschiedene Ausgabegeräte erzeugt. Wer mit den Parametern etwas experimentieren will oder eigene Grafiken oder gar Zeichensätze erzeugen will, sollte sich das METAFONTbook zulegen, denn da steht praktisch alles drin, was der Benutzer wissen muß.

[2] Knuth, Donald E. *METAFONT: The Program*, Computers and Typesetting Vol. D, Addison-Wesley, Reading (Massachussets) 1986. 560 Seiten.

Wer es ganz genau wissen will, kann hier den Original-Sourcecode in WEB nachlesen. Dieses Buch ist die Grundlage meiner C-Version von METAFONT.

[3] Knuth, Donald E. *TEX: The Program*, Computers and Typesetting Vol. B, Addison-Wesley, Reading (Massachussets) 1984. 594 Seiten.

Der Sourcecode von TEX für alle, die auch dort hinter die Kulissen schauen wollen. Ebenfalls in der Programmier- und Dokumentationssprache WEB geschrieben.

[4] Knuth, Donald E. *Computer Modern Typefaces*, Computers and Typesetting Vol. E, Addison-Wesley, Reading (Massachussets) 1986. 590 Seiten.

Hier sind die ganzen METAFONT-Programme für die komplette Computer Modern Schriftenfamilie dokumentiert, zusammen mit vielen Probeausdrücken (proofs). Man kann dieses Buch also als Nachschlagewerk, als Bilderbuch oder auch als Lehrbuch für beispielhafte METAFONT-Programmierung ansehen.

[5] Knuth, Donald E. *The WEB System of Structured Documentation*, Stanford Computer Science Report No. 980, Stanford University, Department of Computer Science, Stanford (California) 1983. 206 Seiten.

Anleitung zu WEB und Sourcecode zu WEAVE und TANGLE, die zusammen das WEB-System bilden.

[6] Billawala, Nazneen N. *Metamarks: Preliminary studies for a Pandora's Box of shapes*, Stanford Computer Science Report No. 1256, Stanford University, Department of Computer Science, Stanford (California) 1989. Erhältlich von der TEX Users Group.

Das Buch zu den Pandora-Fonts. Dieses Buch ist **keine** Einführung in METAFONT, sondern mehr ein Bilderbuch, das die Ergebnisse einiger Experimente mit METAFONT zeigt.

Das Herausragende an den Pandora-Fonts ist, daß sie von Anfang an für und vor allem mit METAFONT entwickelt und nicht wie die meisten anderen Zeichensätze (cmr eingeschlossen) erst nachträglich für METAFONT zurechtgebogen wurden. Wer die Pandora-Fonts genauer studieren will, findet in diesem Buch einige Hinweise und die grundlegenden Elemente (Serifen, Kreise, Bögen, Bowls), aus denen die Zeichen aufgebaut sind. Den Hauptteil des Buches machen Parametervariationen und ihre Auswirkungen auf das Aussehen der Zeichenelemente aus.

[7] Kopka, Helmut. LaTeX – *Erweiterungsmöglichkeiten*, 2. Aufl., Addison-Wesley, Bonn 1991.

Obwohl es in diesem Buch hauptsächlich um LaTeX geht, enthält es auch eine knapp hundertseitige Kurzeinführung in METAFONT. Momentan ist dies das einzige deutsche Buch, das METAFONT behandelt. Angefangen bei den einzelnen Elementen des METAFONT-Systems (Programm-Umgebung) über die wichtigsten Grundlagen von METAFONT und seiner Programmiersprache bis hin zur Erzeugung eines Firmenlogos vermittelt Kopka das nötige Wissen, um fertige METAFONT-Programme in ihrer Grundstruktur erfassen und eigene kleinere Arbeiten selbst erledigen zu können. Die ideale Vorbereitung für das Werk des „Meisters“, das METAFONTbook.