INTERAKTIVER BETRIEB

1. Einführung

Dieses Kapitel führt ein in das interaktive Arbeiten mit Rechnern der VAX Familie von DEC (Digital Equipment Corporation). Mit dem Graphik-Rechner DECMRG und dem Experiment-Rechner DECMRE stehen z.Zt. zwei Rechner VAX 11/750 für besondere Aufgaben zur Verfügung; sie werden voraussichtlich noch in diesem Jahr durch leistungsstärkere Rechner ersetzt; die beiden Rechner werden im folgenden kurz VAX genannt. Auf beiden VAXen wird (zur Zeit noch) das Betriebssystem VAX/VMS eingesetzt (der Einsatz von ULTRIX, d.h. UNIX von DEC, wird diskutiert).

Leistungen der VAX werden (innerhalb von Jobs) mit **Commands** angefordert. Im **interaktiven Betrieb** (syn. Dialog-Betrieb) wird i.a. jeweils ein Command (über die Tastatur des Terminals) eingegeben, von der VAX ausgeführt und mit einer Antwort (auf dem Bildschirm des Terminals) quittiert; danach kann ein neues Command eingegeben werden. Im Batch-Betrieb dagegen werden die Commands eines Jobs als Ganzes bei der Initiierung übergeben (in einem File); die Antworten werden gesammelt und nach Bearbeitung des gesamten Jobs z.B. an einem Drucker ausgegeben (vgl. BATCH-BETRIEB).

Im interaktiven Betrieb können **interaktive Jobs** (IBM: Sessions) geführt werden. Ein interaktiver Job wird durch spezielle Eingaben (Login) begonnen. Danach können **DCL Commands** eingegeben werden, wie z.B. EDIT zum Edieren eines Files, SUBMIT zum Initiieren eines Batch Jobs oder LOGOUT zum Beenden des Jobs.

Die Kommandosprache **DCL** (Digital Control Language) unterstützt das interaktive Arbeiten sowie die Formulierung von Batch Jobs gleichermaßen gut. (Abkürzungen, logische Namen und Symbole sparen Schreibarbeit; Online Informationen liefert das HELP Command; spezielle DCL Commands wie z.B. IF, READ, WRITE ermöglichen FORTRAN ähnliches Programmieren; vgl. KOMMANDOSPRACHE).

Dokumentationen:

[1] Introduction to VMS
(General User Subkit, Volume 2A Using VMS)

[2] Guide to Using VMS
(General User Subkit, Volume 2A Using VMS)

[3] Guide to VMS Text Processing
(General User Subkit, Volume 5A Processing Text)

[4] VMS DCL Dictionary
(General User Subkit, Volume 4 DCL Dictionary)

[5] VMS General User's Manual
(Base Set)

Als Einführungsliteratur werden die Dokumentationen [1], [2] und [3] empfohlen. [1] ist eine knappe, leicht verständliche Anleitung zum Arbeiten mit der VAX (Terminal Bedienung, Beginnen und Beenden von Jobs, Eingabe von DCL Commands, Arbeiten mit Files und Directories, Programmentwicklung, Arbeiten mit Command Procedures und Batch Jobs); leider fehlt die Textverarbeitung: es wird auf [3] verwiesen. [2] ist wesentlich ausführlicher als [1]; zusätzlich werden die Textverarbeitung (Edieren mit EDT und TPU, Formatieren mit DSR) und der Zugriffsschutz für Files (Protection) beschrieben; [2] enthält jedoch nichts zur Programmentwicklung. [4] dokumentiert Format und Leistung der DCL Commands.

[5] ist ein Taschenbuch, das neben leicht gekürzten Fassungen der Dokumentationen [2] und [4] auch stark gekürzte Fassungen der Manuals zu DSR (Text-Formatierung), EDT (Standard Editor), TPU (syn. EVE; Editor), Mail (Electronic Mail) und Sort/Merge enthält. Dieses Taschenbuch reicht einem Benutzer, der bereits erste Erfahrungen mit der VAX gesammelt hat, als Unterstützung vor Ort fast immer aus; seine Anschaffung wird trotz des hohen Preises (ca. 220 DM) dringend empfohlen.

Eine Liste aller verfügbaren Dokumentationen finden Sie zusammen mit den Bestellnummern in der Notiz DOKUMENTATIONEN. Als preiswerte deutschsprachige Einführungs-Literatur wird die folgende Druckschrift des HRZ Siegen empfohlen:

VMS - Handbuch
(VAX/VMS Version 5.0)

Diese Druckschrift ist im Sekretariat des HRZ erhältlich (vgl. Notiz DRUCKSCHRIFTEN). Sie enthält eine ausführliche Anleitung zum Arbeiten mit VAX/VMS (inklusiv EDT, Programmentwicklung und Mail). Im Gegensatz zum Benutzerhandbuch werden in dieser Druckschrift fast ausschließlich deutsche Begriffe verwandt (z.B. Datei statt File; Dateiverzeichnis statt Directory); zur Klärung der Begriffs-Welten plant das HRZ, der Druckschrift eine Synonym-Tabelle für die verwendeten Begriffe beizufügen. In diesem Zusammenhang sei noch darauf hingewiesen, daß die im VMS-Handbuch beschriebene LOcal Language Assistance (LOLA; stellt deutsche HELP Informationen bereit) auf den VAXen des HRZ nicht installiert ist.

Gebrauchsanweisung: Dieses Kapitel sollte parallel zu den Dokumentationen [1] und [3] durchgearbeitet werden (diese Dokumentationen können für ein bis zwei Tage ausgeliehen werden); Benutzer, die bereits Erfahrung im interaktiven Arbeiten mit Rechnern haben, können das Kapitel ggf. ohne diese Dokumentationen verstehen. Beginnen Sie bitte mit Abschnitt 2 dieses Kapitels und arbeiten Sie dann jeweils die angegebenen Chapters in den Dokumentationen durch. Die Beispiele in den Dokumentationen sollten am Terminal nachvollzogen werden, die Abschnitte dieses Kapitels als ergänzende Information gelesen werden.

Das Drücken einer Taste wird dargestellt durch <TASTE> (z.B. <RETURN> für Drücken der RETURN Taste). Beispiele für Eingaben sind `fett` gedruckt und durch <RETURN> abgeschlossen; Antworten der VAX sind originalgetreu angegeben. In den Erläuterungen zu den Beispielen sind wählbare Bezeichnungen (z.B. Filenames) *GROSS UND KURSIV*, feste Bezeichnungen (z.B. Command Names) GROSS geschrieben. Zur Darstellung von Formaten sind wählbare Bezeichnungen *klein und kursiv*, feste Bezeichnungen GROSS geschrieben; obligate Angaben sind ggf. unterstrichen.

Alphanumerische Zeichen sind Buchstaben (ohne Unterscheidung groß/klein), Ziffern und die Sonderzeichen $ und _ (Underscore); das Sonderzeichen $ darf nicht in Bezeichnungen verwendet werden, die vom Benutzer gewählt werden.

Übersicht: Abschnitt 2 beschreibt Eigenschaften von verschiedenen Terminals. Abschnitt 3 beschreibt die Funktionen für das interaktive Arbeiten mit der VAX; mit diesen Funktionen hat das Arbeiten im interaktiven Betrieb folgende grundsätzliche Struktur:

baue ggf. Verbindung auf (vgl. 3.1.1)
.
.
beginne interaktiven Job (Login vgl. 3.1.2)
.
.
führe DCL Command aus (vgl. 3.1.3)
.
.
führe Command Procedure aus
(@..., vgl. KOMMANDOSPRACHE)
.
.
führe Programm aus (RUN, vgl. 5.5)
.
.
beende interaktiven Job (Logout, vgl. 3.1.4)
.
.
baue ggf. Verbindung ab (vgl. 3.1.5)

Graphik- und Experiment-Rechner sind in das Marburger Netzwerk MR-DECNET integriert (Netzwerk-Software ist DECNET). Interaktive Jobs können sowohl an dem Rechner, mit dem das Terminal direkt, durch das DEVELNET oder durch einen Terminal Server verbunden ist (lokaler Knoten), als auch an einem entfernten Knoten des MR-DECNET geführt werden. Dafür gilt folgende grundsätzliche Struktur:

beginne interaktiven Job am lokalen Knoten
.
.
beginne interaktiven Job am entfernten Knoten
(SET HOST, vgl. 3.1.2)
.
.
beende interaktiven Job am entfernten Knoten
(LOGOUT, vgl. 3.1.4)
.
.
beende interaktiven Job am lokalen Knoten

Der Abschnitt 4 beschreibt das Arbeiten mit Files. Der Abschnitt 5 behandelt die Programmentwicklung (Edieren, Compilieren, Binden, Starten) in FORTRAN und PASCAL. Abschnitt 6 enthält Beispiele für interaktive Jobs zur Illustration der in Abschnitt 5 beschriebenen Funktionen.

2. Terminals

Interaktive Jobs können an **Terminals** vom Typ

- DEC VT100 — Standard Terminal
- TEK 41*XX*, CX41*XX* — Graphik Terminals
- QUME QVT201
- DEC VT220
- DEC VT320 — DEVELNET Terminals
- ...

geführt werden. Jedes Terminal trägt einen Aufkleber mit Typ (z.B. TEK 4107) und Anschlußbezeichnung (z.B. TXB2). Das Terminal VT100 ist als Standard Terminal anzusehen; seine Eigenschaften werden im Abschnitt 2.2 erläutert; zu den abweichenden Eigenschaften der anderen Terminals vgl. 2.3 ff. Für die Graphik Terminals gibt es Bedienungsanleitungen von TEKTRONIX, für die DEVELNET Terminals gibt es **Bedienungsanleitungen** des HRZ (sie liegen neben den Terminals). Demnächst wird es auch im HRZ Terminal Server geben, die es ermöglichen, von den angeschlossenen Terminals über das ETHERNET-LAN (vgl. MR-DECNET) auf die VAXen zuzugreifen.

Mikrocomputer/PC's mit VT100 Emulation können ebenfalls über das DEVELNET oder über einen Terminal Server auf die VAX zugreifen (vgl. 2.4). Für IBM PC's und Kompatible stellt das HRZ mit KERMIT eine VT100 Emulation bereit, vgl. Einzelschrift KERMIT (MIKROCOMPUTER).

2.1 Datenübertragung

Die Datenübertragung ist zeichenorientiert. Die Eingabe von Zeichen ist i.a. nur möglich, wenn ein Programm eine Eingabe anfordert. Das Programm stellt dazu einen Eingabepuffer (Hauptspeicherbereich) bereit; jedes eingegebene Zeichen wird (vom Terminal Driver) zunächst in diesen Eingabepuffer eingetragen (die VAX hat keinen Vorrechner) und i.a. am Bildschirm protokolliert, dies wird als **Echo** bezeichnet. (Für Terminals am Terminal Server übernimmt der Terminal Server einige Aufgaben des Terminals Drivers, z.B. das Echo). Durch

<RETURN>

wird der Eingabepuffer dem Programm zur Bearbeitung übergeben. Vor dem <RETURN> kann in der **Eingabezeile** beliebig korrigiert werden (der Terminal Driver führt die Korrekturen sowohl im Eingabepuffer als auch auf dem Bildschirm aus). Nach dem <RETURN> kann die eingegebene Zeile - auch wenn sie noch auf dem Bildschirm steht - weder korrigiert noch erneut zur Bearbeitung übergeben werden. Ausnahme: Der Command Interpreter speichert die letzten 20 eingegebenen DCL Commands im sogenannten Recall Buffer ab; das letzte eingegebene DCL Command kann z.B. mit <↑> in die Eingabezeile "zurückgerufen", ggf. korrigiert und erneut zur Bearbeitung übergeben werden, vgl. 3.1.3.

2.2 DEC Terminal VT100

Das DEC Terminal VT100 wurde vom HRZ als Standard Terminal der VAX festgelegt.

Die **Tastatur** besteht aus Zeichentasten und Funktionstasten. Jede Zeichentaste realisiert 2 Zeichen (SHIFT Taste zum Umschalten), jede Funktionstaste 1 Funktion des VT100. Einige Zeichentasten werden außerdem zur Realisierung von Funktionen verwendet; dazu ist gleichzeitig die CTRL Taste zu drücken; im folgenden bedeutet z.B.

```
<CTRL/Z>
```

das gleichzeitige Drücken der Taste CTRL und der Taste Z.

Die Tasten des **Auxiliary Keypad** (außer PF1 bis PF4) können abwechselnd zur Eingabe von Ziffern, Punkt, Komma und Minus (NUMERIC_KEYPAD) und als anwendungsbezogene Funktionstasten (APPLICATION_KEYPAD) verwendet werden; die Umschaltung kann beim VT100 nur von einem Programm (z.B. dem Editor) oder einem Command (z.B. SET TERMINAL/ APPLICATION_KEYPAD, vgl. 3.1.3), nicht aber von der Tastatur erfolgen.

Zeichenvorrat: Zeichen werden von der VAX in 8-bit Bytes gespeichert (dezimale Werte 0-255).

Eingabe: Alle ASCII Schriftzeichen (dezimale Werte 32-126, vgl. CODES im Teil ALLGEMEINES) sind über die Tastatur eingebbar, die übrigen Zeichen sind nur mit Hilfe des Editors eingebbar (EDT Keypad Command SPECINS , vgl. 5.1).

Bei der Eingabe von DCL Commands hat das Zeichen - eine besondere Bedeutung: wenn es das letzte Zeichen einer Eingabe ist (oder nur noch Blanks folgen), wird es als Fortsetzungszeichen interpretiert, vgl. 3.1.3. Soll das Zeichen - nicht Fortsetzungszeichen sein, so muß -- eingegeben werden.

Ausgabe: Alle ASCII Schriftzeichen (s.o.) sind auf dem Bildschirm darstellbar; alle weiteren Zeichen (insbesondere mit dezimalen Wert ≥ 128) sollten nicht auf das Terminal ausgegeben werden (sie werden ggf. als Steuerzeichen für das Terminal oder die Datenübertragung interpretiert).

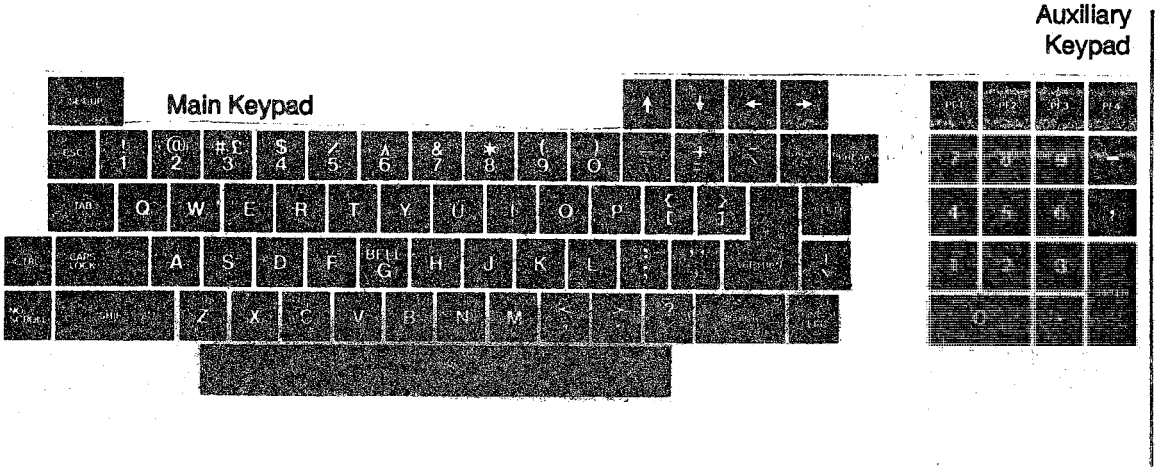

Abb. 2-1 VT100 Tastatur

Taste	Funktion
<SHIFT>	Umschalten (Zeichen)
<CTRL>	Umschalten (Zeichen → Funktion)
<RETURN>	Eingabezeile zur Bearbeitung übergeben
<NO SCROLL>	Ausgabe unterbrechen/fortsetzen (entspricht <CTRL/S> und <CTRL/Q>)
<CTRL/S>	Ausgabe unterbrechen
<CTRL/Q>	Ausgabe fortsetzen
<CTRL/O>	Ausgabe abbrechen
<CTRL/Z>	Eingabe von Zeilen beenden, vgl. 3.2.2
<CTRL/Y>	Programmausführung unterbrechen, vgl. 3.2.1
<CTRL/C>	entspricht im wesentlichen <CTRL/Y>
<CTRL/T>	informieren über verbrauchte Betriebsmittel auch während Programmausführung, vgl. 3.3.2
<→>,<←>	Cursor in Eingabezeile positionieren
<BACKSPACE>	Cursor an Anfang der Eingabezeile positionieren
<CTRL/E>	Cursor ans Ende der Eingabezeile positionieren
<TAB>	Cursor zum nächsten Tabulatorstop (alle 8 Zeichen) positionieren
<CTRL/A>	Umschalten einfügen/überschreiben, vgl. 3.1.3
<DELETE>	Zeichen vor Cursor löschen
<CTRL/U>	Eingabezeile löschen
<↑>,<↓>	Blättern im Recall Buffer, vgl. 2.1, 3.1.3
Tasten des Auxiliary Keypad	Anwendungsbezogene Funktionstasten, vgl. 3.1.3
<ESC>	kann von speziellen Programmen interpretiert werden

Tabelle 2-1: wichtige Funktionen des VT100

2.3 Graphik Terminals

Die Graphik Terminals TEK 41*XX* und TEK CX41*XX* sind i.a. direkt an die VAX angeschlossen; sie verhalten sich alphanumerisch im wesentlichen wie das Standard Terminal VT100; es sind lediglich einige Tasten anders beschriftet, die Zuordnung zu den VT100 Tasten wird durch Abb. 2-2 beschrieben. Das CX41*XX* hat neben dem asynchronen Anschluß auch einen CoaXial-Anschluß, der mit der IBM 4381 direkt verbunden ist; die Wahl des asynchronen Anschlusses erfolgt im Setup-Mode des Terminals mit

```
<Setup>
*host rs232 <RETURN>
<Setup>
```

Die graphischen Fähigkeiten dieser Terminals beschreibt die Notiz HARDWARE TEKTRONIX 4100. Graphiken können z.B. mit TEKGKS auf die Terminals ausgegeben werden (vgl. Notiz ANW.TEKGKS).

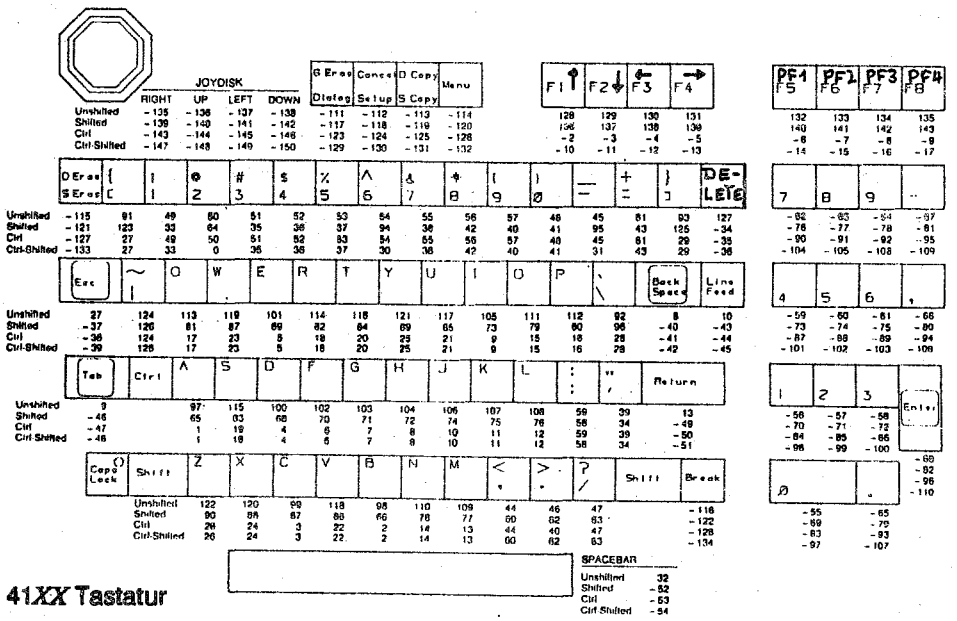

Abb. 2-2 TEK 41*XX* Tastatur

2.4 Terminals am DEVELNET

Ein Terminal QVT201, VT220, VT320 am DEVELNET kann wie das Standard Terminal VT100 an der VAX benutzt werden; das DEVELNET verbindet das Terminal mit der VAX. Die DEVELNET Terminals verhalten sich, wenn sie wie in der HRZ Bedienungsanleitung beschrieben eingestellt sind, wie das Standard Terminal VT100.

2.5 Terminals an einem Terminal Server

Ein Terminal an einem Terminal Server kann i.a. wie das Standard Terminal VT100 an der VAX benutzt werden; das HRZ wird Bedienungsanleitungen für die Benutzung von Terminals an einem Terminal Server erstellen.

3. Interaktives Arbeiten

Die Benutzung der VAXen des HRZ muß beantragt werden, vgl. BENUTZUNG. Das HRZ teilt dem Benutzer einen **Username** und eine **Anfangs-Paßwort** zu; das Anfangs-Paßwort muß beim ersten Login geändert werden, vgl. 3.1.2 und BENUTZER-BERECHTIGUNGEN. Die VAXen des HRZ sind als Knoten (syn. Nodes) in das Marburger Netzwerk **MR-DECNET** eingebunden (Netzwerk Software ist DECNET). Wer über das Netzwerk an anderen VAXen arbeiten will, benötigt auch für diese VAXen einen gültigen Username.

Bitte arbeiten Sie jetzt in [1] das Chapter 1.1 (Interaction with the System) durch, um einen ersten Eindruck vom Arbeiten mit der VAX zu bekommen. Zur Vertiefung oder bei Fragen sollte dann der Abschnitt 3 dieses Kapitels fertiggelesen werden.

3.1 Interaktive Jobs

Interaktive Jobs können sowohl an direkt angeschlossenen Terminals (hauptsächlich Graphik Terminals von TEKTRONIX) als auch an Terminals am DEVELNET oder an einem Terminal Server (hauptsächlich alphanumerische Terminals verschiedener Hersteller) geführt werden. Außer von direkt angeschlossenen Terminals muß zunächst eine Verbindung zu einer der VAXen aufgebaut werden. Der Rechner, mit dem das Terminal (direkt bzw. über DEVELNET oder ETHERNET-LAN) verbunden ist, wird als **lokaler Knoten** (im MR-DECNET) bezeichnet, jeder andere Rechner im MR-DECNET wird als **entfernter Knoten** bezeichnet. Innerhalb eines interaktiven Jobs am lokalen Knoten kann ein interaktiver Job am entfernten Knoten geführt werden; der Rechner, der den Job bearbeitet, wird als **Host** bezeichnet. Zur grundsätzlichen Struktur, vgl. 1.

3.1.1 Verbindung aufbauen

Das Aufbauen einer Verbindung erfolgt an direkt angeschlossenen Terminals implizit. Von Terminals am DEVELNET oder an einem Terminal Server muß die Verbindung zu einer der angeschlossenen VAXen explizit aufgebaut werden. Im Auswahlmenü ist als Ziel der Knotenname des gewünschten Rechners anzugeben; näheres enthält die HRZ Bedienungsanleitung des Terminals (sie sollte neben dem Terminal liegen).

3.1.2 Interaktiven Job beginnen (Login)

Login am lokalen Knoten: Nach dem Aufbau einer Verbindung (vgl. 3.1.1) kann wie folgt ein interaktiver Job begonnen werden (Usernames und Paßwörter sind dabei durch eigene Usernames und Paßwörter für die entsprechenden VAXen zu ersetzen):

```
<RETURN>
*** A N N O U N C E ***

DECNET HRZ Uni Marburg, Knoten DECMRG (Graphik-Rechner)
 ... Nachrichten des HRZ ...
 ... vgl. NACHRICHTEN/MITTEILUNGEN ...

Username: meier_g <RETURN>                    (Username eingeben)
Password: hulhaz <RETURN>                     (Paßwort eingeben; kein Echo)
```

Ein ungültiger Username oder ein falsches Paßwort werden quittiert mit der Meldung "User authorization failure". Ein neuer Login kann wie oben beschrieben initiiert werden. Das Anfangs-Paßwort für neue Usernames ist verfallen; ein **verfallenes Paßwort** muß sofort geändert werden:

```
Your passwort has expired; you must set a new password to log in

Old password: hulhaz <RETURN>                 (altes Paßwort eingeben; kein Echo)
New password: geheim <RETURN>                 (neues Paßwort eingeben; kein Echo)
Verification: geheim <RETURN>                 (neues Paßwort eingeben; kein Echo)
```

Das Paßwort muß aus 6 bis 31 alphanumerischen Zeichen bestehen; es verfällt nach 90 Tagen, wenn es nicht mit dem DCL Command SET PASSWORD geändert wird (vgl. 3.4). Bei gültigem Username und Paßwort wird ausgeführt:

- das WELCOME (Begrüßung)
- die Command Procedure im **System Login Command File**
- die Command Procedure im **User Login Command File** (falls vorhanden; per Default **LOGIN.COM**, vgl. BENUTZER-BERECHTIGUNGEN)

In den User Login Command File können (mit Hilfe des Editors) DCL Commands eingetragen werden wie z.B. SET, DEFINE und SHOW Commands (vgl. 3.1.3); zum Arbeiten mit Command Procedures vgl. Chapter 5.2 in [1] und KOMMANDOSPRACHE. Nach erfolgreichem Login wird der DCL Prompt (vgl. 3.1.3) ausgegeben:

```
$
```

Login an einem entfernten Knoten: Ein Login an einem entfernten Knoten kann nur innerhalb eines interaktiven Jobs initiiert werden, und zwar mit dem **DCL Command SET HOST**, wobei der Name des Knotens (z.B. DEMRE) als Parameter anzugeben ist; z.B.

```
$ set host decmre <RETURN>
*** A N N O U N C E ***
DECNET HRZ Uni Marburg, Knoten DECMRE (Experiment-Rechner)
 ... Nachrichten des HRZ ...

Username: meier_e <RETURN>
Password: ...     <RETURN>
 ...
```

Der interaktive Job, der das SET HOST Command ausgeführt hat, bleibt erhalten.

Groß/Kleinschreibung: Kleine Buchstaben werden in große umgewandelt.

3.1.3 DCL Commands eingeben

Nach dem Login können **DCL Commands** eingegeben werden; eine Liste aller DCL Commands erhalten Sie mit HELP, vgl. 3.3.1. Eingaben werden i.a. durch die Ausgabe eines **Prompts** angefordert.

DCL Prompt: Der Command Interpreter fordert die Eingabe eines DCL Commands mit einer speziellen Zeichenfolge, im folgenden DCL Prompt genannt, an. Per Default besteht der DCL Prompt aus einem Dollar Zeichen gefolgt von einem Blank. Hinter dem Prompt beginnt die Eingabezeile (max. 78 Zeichen); es kann ein DCL Command eingegeben werden, z.B. ein Command zum Ändern des DCL Prompt:

```
$ set prompt = "GIB KOMMANDO: " <RETURN>
GIB KOMMANDO:
```

Von nun an werden DCL Commands mit GIB KOMMANDO: angefordert. In den folgenden Beispielen wird als DCL Prompt stets "$ " verwendet.

Korrigieren in der Eingabezeile (vor dem <RETURN>): Der Cursor kann mit <←> und <→> innerhalb der Eingabezeile positioniert werden; <BACKSPACE> bewegt den Cursor an den Anfang, <CTRL/E> an das Ende der Eingabezeile. Das Zeichen auf der Cursor-Position kann überschrieben werden; <CTRL/A> schaltet den Einfügmodus ein: es können Zeichen eingefügt werden; nochmaliges <CTRL/A> schaltet den Einfügmodus wieder ab. Die Korrektur erfolgt auf dem Bildschirm und im Eingabepuffer. Der Cursor muß vor dem <RETURN> nicht ans Ende der Eingabezeile positioniert werden. (Dieses "Edieren" der Eingabezeile kann mit SET TERMINAL/LINE_EDITING explizit ein- und mit SET TERMINAL/NOLINE_EDITING abgeschaltet werden; Default: LINE_EDITING).

DCL Commands zurückrufen: Der Command Interpreter speichert in einem Puffer - **Recall Buffer** genannt - die letzten 20 eingegebenen DCL Commands. Der Inhalt dieses Puffers kann am Terminal ausgegeben werden mit

```
$ recall/all <RETURN>
```

Zu jedem Command wird eine laufende Nummer n ($1 \leq n \leq 20$) ausgegeben, wobei das zuletzt eingegebene Command die Nummer 1 hat. Z.B. kann mit

```
$ recall 2 <RETURN>
```

das vorletzte Command in die Eingabezeile zurückgerufen, korrigiert (s.o.) und mit <RETURN> zur Bearbeitung übergeben werden. RECALL Commands selbst werden nicht in den Recall Buffer eingetragen. Noch bequemer ist es, mit <↑> und <↓> im Recall Buffer vorwärts bzw. rückwärts zu "blättern".

Tasten belegen mit DCL Commands: Die Tasten des Auxiliary Keypad (vgl. Abb. 2-1) können je mit einer Zeichenfolge belegt werden; diese Belegung steht nur bei der Eingabe von DCL Commands zur Verfügung. Zum Arbeiten mit der Belegung muß das Terminal auf APPLICATION_KEYPAD (Default: NUMERIC_KEYPAD) geschaltet sein (nicht erforderlich für PF1 bis PF4):

```
$ set terminal/application_keypad <RETURN>
```

Beispiele:

```
$ define/key kp9 "link prog,disk0:[anw.cernlib]kernlib/library" <RETURN>
```

belegt die Taste 9 des Auxiliary Keypad mit der in " eingeschlossenen Zeichenfolge. Danach wird durch Drücken dieser Taste obiges LINK Command in die Eingabezeile kopiert (auf Cursor-Position), kann dort korrigiert (s.o.) und mit <RETURN> zur Bearbeitung übergeben werden. Tasten können auch so belegt werden, daß das Command sofort bearbeitet wird; nach

```
$ define/key pf3 "show quota"/terminate <RETURN>
```

wird durch <PF3> das DCL Command SHOW QUOTA ausgeführt.

```
$ delete/key pf3 <RETURN>
```

löscht die Belegung der Taste PF3. Auflisten aller Tastenbelegungen erfolgt mit

```
$ show key/all <RETURN>
```

Es wird empfohlen, entsprechende SET, DEFINE und SHOW Commands in den User Login Command File zu schreiben (z.B. mit EDIT LOGIN.COM, vgl. KOMMANDOSPRACHE).

Groß/Kleinschreibung: Kleine Buchstaben werden wie große interpretiert.

Abkürzungen: In DCL Commands können alle Schlüsselwörter (Command Name, Qualifiers, ...) abgekürzt werden, solange die Abkürzung eindeutig ist; die ersten 4 Zeichen sichern die Eindeutigkeit (vgl. KOMMANDOSPRACHE).

Fortsetzung: Zur Eingabe von DCL Commands können mehrere Eingabezeilen benutzt werden; das Zeichen - wird am Ende der Eingabezeile als Fortsetzungszeichen interpretiert (vgl. KOMMANDOSPRACHE; soll es nicht als Fortsetzungszeichen interpretiert werden, so muß -- eingegeben werden). Beispiel für Eingabe eines DCL Commands in 3 Eingabezeilen:

```
$ fortran/continuations=25 -<RETURN>
_$ a+b/list,c+d/list-<RETURN>
_$ /object=obl<RETURN>
```

3.1.4 Interaktiven Job beenden (Logout)

Ein Interaktiver Job wird mit dem DCL Command LOGOUT beendet. Logout Command Procedures (analog zu LOGIN.COM) gibt es nicht.

Logout am entfernten Knoten: LOGOUT führt in den Interaktiven Job, der das SET HOST Command ausgeführt hat, zurück:

```
$ logout <RETURN>
   ...
  MEIER_E  logged out at ...
   ...
%REM-S-END, control returned to node DECMRG::
```

Der Interaktive Job, der das SET HOST Command ausgeführt hat, kann jetzt fortgesetzt werden.

Auch mehrmaliges <CTRL/Y> (z.B. bei Systemfehlern an dem entfernten Knoten) führt zurück in den Interaktiven Job am lokalen Knoten.

Logout am lokalen Knoten:

```
$ logout <RETURN>
```

beendet den Interaktiven Job am lokalen Knoten.

3.1.5 Verbindung abbauen

Eine DEVELNET Verbindung wird beim Logout implizit abgebaut. Sie kann explizit abgebaut werden durch 3 mal <BREAK> oder durch Ausschalten des Terminals (vgl. HRZ Bedienungsanleitung); ein interaktiver Job wird dabei ggf. implizit beendet.

Bei den Terminals an einem Terminal Server hängt der Verbindungsauf/abbau vom Typ des Terminal Server ab; er wird in der HRZ Bedienungsanleitung des Terminals beschrieben.

Beim ersten Lesen des Kapitels können Sie die folgenden Abschnitte 3.2 bis 3.5 auslassen und mit Abschnitt 4. fortfahren.

3.2 Unterbrechen, Fortsetzen, Abbrechen

Das Arbeiten mit diesen Funktionen ist zweistufig; sie können entweder auf Ein/Ausgaben oder auf die Programme angewendet werden, die die Ein/Ausgaben anfordern/erzeugen:

Programm beginnen
.
.
Programm unterbrechen
Programm fortsetzen/abbrechen
.
.
Eingabe beginnen
.
.
Eingabe beenden/abbrechen
.
.
Ausgabe beginnen
.
.
Ausgabe unterbrechen
Ausgabe fortsetzen
.
.
Ausgabe beenden/abbrechen
.
.
Programm beenden

Alle Aussagen des Abschnitts 3.2 gelten nicht nur für Programme, sondern ganz analog auch für DCL Commands und Command Procedures.

3.2.1 Programme, DCL Commands, Command Procedures

Die Ausführung eines Programms (bzw. eines DCL Commands oder einer Command Procedure) wird

- unterbrochen mit <CTRL/Y>

Das unterbrochene Programm wird

- fortgesetzt mit $ **continue** <RETURN>
- abgebrochen mit $ **exit** <RETURN>

Nach der Unterbrechung können DCL Commands eingegeben werden (DCL Prompt). In der Regel wird zur Ausführung dieses DCL Commands das unterbrochene Programm abgebrochen (implizites EXIT; Fortsetzen ist nicht möglich). Nur DCL Commands, die vom Command Interpreter ausgeführt werden (vgl. Tabelle 3-1), erlauben das anschließende Fortsetzen des unterbrochenen Programms.

Bei der Unterbrechung eines Programms gehen Teile der letzten Ausgabezeile verloren. Die Möglichkeit der Programmunterbrechung mit <CTRL/Y> ist abschaltbar mit dem DCL Command SET NOCONTROL=Y.

=,==,:=,:==	ALLOCATE	ASSIGN
ATTACH	CALL	CANCEL
CLOSE	CONNECT	CONTINUE
CREATE/NAME_TABLE	DEALLOCATE	DEASSIGN
DEBUG	DECK	DEFINE
DEFINE/KEY	DELETE/KEY	DELETE/SYMBOL
DEPOSIT	DISCONNECT	ENDSUBROUTINE
EOD	EXAMINE	EXIT
GOSUB	GOTO	IF
INQUIRE	ON	OPEN
READ	RECALL	RETURN
SET CONTROL	SET DEFAULT	SET KEY
SET ON	SET OUTPUT_RATE	SET PROMPT
SET PROTECTION /DEFAULT	SET UIC	SET VERIFY
SHOW DEFAULT	SHOW KEY	SHOW QUOTA
SHOW PROTECTION	SHOW STATUS	SHOW SYMBOL
SHOW TIME	SHOW TRANSLATION	SPAWN
STOP	SUBROUTINE	WAIT
WRITE		

Tabelle 3-1: DCL Commands, die vom Command Interpreter ausgeführt werden

3.2.2 Ein/Ausgaben

Ausgaben auf das Terminal (z.B. vom TYPE Command) erfolgen in der Regel fortlaufend (Scrolling) und meistens schneller als sie gelesen werden können. Die Ausgabe von Zeilen wird

- unterbrochen mit <CTRL/S>
- fortgesetzt mit <CTRL/Q>
- abgebrochen mit <CTRL/O>

Die **Eingabe** von Zeilen (z.B. mit INSERT im EDT Line Mode, vgl. 5.1) wird

- abgebrochen mit <CTRL/Z>

<CTRL/Z> liefert beim Lesen vom Terminal auch die END-Bedingung im FORTRAN Programm bzw. die EOF-Bedingung im PASCAL Programm; Beispiele:

```
      READ (5,100,END=99) ...                                  (FORTRAN)

WILE NOT EOF (INPUT) DO                                        (PASCAL)
  BEGIN READ (INPUT), ...);...END                              (PASCAL)
```

3.2.3 Normieren des Terminals

Durch graphische Ausgaben, Ausgabe von Steuerzeichen oder Fehler bei der Datenübertragung können Terminal Parameter verändert werden, die das Arbeiten mit dem Terminal behindern (z.B. durch falsch formatierte Ausgaben) bis unmöglich machen (z.B. keine Eingabe möglich). Durch Aus- und wieder Einschalten wird das Terminal normiert. Bei direkt angeschlossenen Terminals bleibt der interaktive Job erhalten, bei DEVELNET Terminals wird implizit der Job beendet und die Verbindung abgebaut. Das Verhalten von Terminals an einem Terminal Server hängt u.a. vom Typ des Terminal Server ab, vgl. HRZ Bedienungsanleitung des Terminals.

3.3 Informieren

3.3.1 Leistung und Format von DCL Commands

Das **DCL Command HELP** informiert mehrstufig über Leistung und Format von DCL Commands, vgl. [1], Chapter 1.1.4 Beispiel:

```
$ help <RETURN>
```

gibt neben Erläuterungen zum Arbeiten mit HELP eine Liste aller DCL Command Names am Terminal aus (0. Stufe). Jeder Command Name kann als 'Topic' eingegeben werden, z.B.

```
Topic? show <RETURN>
```

gibt eine Liste aller Options von SHOW am Terminal aus (1. Stufe). Jede dieser Options kann als 'SHOW Subtopic' eingegeben werden, z.B.

```
SHOW Subtopic? quota <RETURN>
```

gibt eine Liste der Command Qualifiers von SHOW QUOTA aus (2. Stufe); jeder Command Qualifier kann als 'SHOW QUOTA Subtopic' eingegeben werden, z.B. informiert

```
SHOW QUOTA Subtopic? /disk <RETURN>
```

über den Command Qualifier /DISK des SHOW QUOTA Commands (3. Stufe). Eine leere Eingabe (nur <RETURN>) führt jeweils eine Stufe zurück; <CTRL/Z> beendet das HELP Command.

Übrigens: Das HELP Command kann noch viel mehr (z.B. gibt HELP/OUTPUT die Informationen in einen File aus). Informieren Sie sich darüber mit HELP HELP.

3.3.2 Verbrauchte CPU-Zeit

Die vom Job verbrauchte CPU-Zeit (in Stunden:Minuten:Sekunden) kann abgefragt werden mit dem **DCL Command SHOW STATUS** oder mit <CTRL/T>. Für SHOW STATUS muß ggf. die Ausführung eines Programms mit <CTRL/Y> unterbrochen werden (vgl. 3.2.1). Danach gibt

```
$ show status <RETURN>
```

u.a. die vom Job verbrauchte CPU-Zeit (Elapsed CPU) am Terminal aus. Ein unterbrochenes Programm kann mit CONTINUE fortgesetzt werden (vgl. 3.2.1).

```
<CTRL/T>
```

kann auch während der Ausführung eines Programms verwendet werden; diese Funktion von <CTRL/T> ist abschaltbar mit dem DCL Command SET NOCONTROL=T.

3.3.3 Belegter Plattenspeicher

Das DCL Command **SHOW QUOTA** liefert neben der Größe des Plattenspeicher-Kontingents die Anzahl der belegten Blöcke (1 Block = 512 Bytes) zum Username des Jobs.

```
$ show quota <RETURN>
```

gibt die Informationen (für das Default Device) am Terminal aus. Ein mit <CTRL/Y> unterbrochenes Programm kann nach SHOW QUOTA mit CONTINUE fortgesetzt werden (vgl. 3.2.1).

3.3.4 Files des Default Directory

```
$ directory <RETURN>
```

listet alle Files des Default Directory am Terminal auf. Was das **DCL Command DIRECTORY** noch alles kann, erfragen Sie mit HELP DIRECTORY, vgl. 3.3.1.

3.3.5 Benutzer-Berechtigungen

Das HRZ legt Benutzer-Berechtigungen (Privilegien und Betriebsmittelschranken) durch Einträge im User-Authorization-File bzw. im Quota-File fest. Jedem interaktiven Job stehen die für den Username eingetragenen Berechtigungen zur Verfügung (vgl. BENUTZER-BERECHTIGUNGEN). Die folgenden SHOW Commands geben Informationen am Terminal aus.

Privilegien des Jobs:
```
$ show process/privileges <RETURN>
```

Betriebsmittelschranken des Jobs:
```
$ show process/quota <RETURN>
$ show working_set <RETURN>
```

Plattenspeicher-Kontingent (vgl. 3.3.3):
```
$ show quota <RETURN>
```

3.3.6 Nachrichten, Mitteilungen

Die **Nachrichten** des HRZ werden beim Login am Terminal ausgegeben; es ist nicht möglich, dies innerhalb eines Jobs zu wiederholen.

Die **Mitteilungen** des HRZ können mit Hilfe der Command Procedures NEU und INFO am Terminal ausgegeben werden (vgl. Notiz NACHRICHTEN/MITTEILUNGEN); erforderliche Angaben werden mit den entsprechenden Erläuterungen angefordert.

```
$ neu <RETURN>
$ info <RETURN>
```

3.4 Paßwort ändern

Das Paßwort wird mit dem DCL Command **SET PASSWORD** geändert, z.B.

```
$ set password <RETURN>
Old password:  pw-alt <RETURN>
New password:  pw-neu <RETURN>
Verification:  pw-neu <RETURN>
$
```

Zu den eingegebenen Paßwörtern gibt es kein Echo. Das Paßwort muß aus 6 bis 31 alphanumerischen Zeichen bestehen. Es verfällt nach 90 Tagen (vgl. 3.1.2), wenn es nicht geändert wird.

3.5 Drucken

Eine Liste aller System-Drucker (mit Name und Standort) gibt die Command Procedure **INFO_DRUCKER** am Terminal aus:

```
$ info_drucker <RETURN>
```

3.5.1 Bildschirminhalt drucken

Dies ist nur an einem an das Terminal angeschlossenen Gerät (Hardcopy, Matrixdrucker) möglich.

Graphik Terminal/Hardcopy: Das Hardcopygerät sollte nur zur Ausgabe einer Graphik verwendet werden. Hardcopygerät einschalten, Dialogbereich mit <DIALOG> ausblenden, dann <S COPY>.

DEVELNET Terminal/Matrixdrucker: Drucker ein- und auf ONLINE schalten, dann <Print Screen>.

3.5.2 Protokollieren eines interaktiven Jobs

DEVELNET Terminal/Matrixdrucker: Wenn an das Terminal ein Matrixdrucker angeschlossen ist, so können die Ein- und Ausgaben eines interaktiven Jobs gleichzeitig auf diesem Drucker ausgegeben werden; dies wird im folgenden **protokollieren** genannt.

Zunächst ist sicherzustellen, daß der Drucker ein- und auf ONLINE geschaltet ist; das Protokollieren wird innerhalb eines interaktiven Jobs

- begonnen mit `$ dr_ein <RETURN>`
- beendet mit `$ dr_aus <RETURN>`

Log File: Nicht empfehlenswert. Die DECNET Software bietet die Möglichkeit, alle Ausgaben eines mit SET HOST initiierten interaktiven Jobs gleichzeitig in einen File - Log File genannt - zu schreiben (SET HOST/LOG=*file-spec node;* dabei kann als *node* (Knoten) auch der Host angegeben werden, vgl. 3.1.2). Der Inhalt dieses Files kann i.a. nicht gedruckt werden, da er auch sämtliche Terminal-Steuerzeichen enthält.

3.5.3 File-Inhalt drucken

Gedruckt werden kann der Inhalt von **Standard Text Files** (dazu gehören z.B. vom Editor erzeugte Files, PASCAL Text Files oder von DCL Commands durch /LIST oder /OUTPUT Qualifier erzeugte Files) sowie der Inhalt von sequentiellen formatierten **FORTRAN Files**.

Mit Hilfe des **DCL Commands PRINT** kann i.a. nur auf die System-Drucker des Host zugegriffen werden. Dies wird im folgenden beschrieben; darüber hinaus kann mit Hilfe des Filetransfers (INITFT) auf weitere Drucker des HRZ zugegriffen werden, vgl. FILETRANSFER im Teil ALLGEMEINES. Beispiele:

```
$ print programm.lis <RETURN>
```

druckt den File-Inhalt von PROGRAMM.LIS am Default Printer des Host (am Graphik-Rechner: Matrixdrucker im Rechnerraum). Der File wird nach dem Drucken nicht gelöscht; soll der File gelöscht werden, so muß PRINT ersetzt werden durch PRINT/DELETE.

```
$ print/queue=mhrza3 programm.lis <RETURN>
```

druckt den File-Inhalt von PROGRAMM.LIS am System-Drucker MHRZA3 des Hosts (am Graphik-Rechner: Matrixdrucker des HRZ in der Außenstation in der Blegenstraße, Nr. 3); der File wird nicht gelöscht (s.o.). Bezeichnungen und Standorte der Drucker gibt die Command Procedure INFO_DRUCKER am Terminal aus (vgl. 3.5).

Vorschubsteuerung: Standard Text Files werden mit einzeiligem Vorschub gedruckt, bei FORTRAN Files wird das erste Zeichen jeder Zeile gemäß FORTRAN Konventionen zur Vorschubsteuerung verwendet.

Hinweis: Der File-Inhalt wird von PRINT nicht kopiert; PRINT trägt lediglich die File Spezifikation in die Printer Queue ein; der File darf weder überschrieben noch gelöscht werden, bevor das Drucken beendet ist.

4. Arbeiten mit Files

4.1 Plattenspeicher-Files

Programme und Daten werden in Files gehalten; wenn Sie Programmentwicklung an der VAX machen wollen, müssen Sie daher einiges über Files wissen. Zur Einführung arbeiten Sie bitte die Chapter 2 und 3 in [1] durch.

Jeder Benutzer (Username) besitzt ein **User File Directory** auf einem der Plattenspeicher; in diesem Directory hält er seine Files; es ist (im User Authorization File) als **Login Directory** eingetragen, d.h. es ist das **Default Directory** am Anfang jedes Jobs, vgl. BENUTZER-BERECHTIGUNGEN. Zur Strukturierung können in diesem Directory **Subdirectories kreiert** werden, z.B.

```
$ create/directory [.subdir] <RETURN>
$ set default [.subdir] <RETURN>
```

kreiert das Subdirectory *SUBDIR* und erklärt es zum neuen Default Directory. SHOW DEFAULT liefert die vollständige Bezeichnung des Default Directory; das Zurücksetzen auf das Login Directory erfolgt mit:

```
$ set default sys$login <RETURN>
```

Für einen File des Default Directory kann als **File Spezifikation** eines der folgenden Formate verwendet werden:

filename
filename.filetype
filename.filetype;version

(z.B. QUELLE.FOR;3). Der **Filename** ist frei wählbar (max. 39 alphanumerische Zeichen); für Filetype und Version gelten in der Regel Defaults. Statt Filename, Filetype oder Version kann in vielen DCL Commands ein * angegeben werden (syn. Wild Card), wenn alle Files mit beliebigem Filename, Filetype bzw. beliebiger Version bearbeitet werden sollen.

Der **Filetype** (max. 39 alphanumerische Zeichen) dient zur Klassifikation des File-Inhalts (z.B. FOR für FORTRAN Programme, PAS für PASCAL Programme). Es wird empfohlen, bei der Programmentwicklung ausschließlich die Default Filetypes (FOR bzw. PAS, LIS, OBJ, MAP, EXE) zu verwenden.

Die **Version** (pos. ganze Zahl ≤ 32767) dient zur Numerierung von Files mit gleichem *filename.filetype*. Der Default ist beim lesenden Zugriff die höchste existierende Version. Beim schreibenden Zugriff wird i.a. ein neuer File kreiert, entweder mit Version 1 oder höchster bereits existierender Version + 1; die explizite Angabe einer bereits existierenden Version führt i.a. zu einer Fehlermeldung. Alte Versions sollten zur Einsparung von Plattenspeicherplatz so bald wie möglich gelöscht werden, z.B. mit dem **DCL Command PURGE** (s.u.).

Folgende Besonderheiten sind im File Handling an der VAX zu beachten:

- **Kreation**: Files werden i.a. implizit bei einem Schreibzugriff kreiert (z.B. durch Editor oder Compiler; neue Version, s.o.).
- **Lebensdauer**: Sie ist nicht von der Dauer eines Jobs abhängig; es gibt nur permanente Files. Files, die nicht mehr benötigt werden (z.B. alte Versions) müssen explizit gelöscht werden.
- **Zugriffsschutz**: Jeder File hat eine Protection, die festlegt, wer (System, Eigentümer, Gruppe, alle) wie (schreiben, lesen, ausführen, löschen) zugreifen darf. Per Default können nur Benutzer, die zu Ihrer Gruppe gehören, Ihre Files lesen und ausführen. Mit SET FILE/PROTECTION kann die Protection eines Files geändert werden. Zusätzlich können für Files und Directories Access Control Lists angelegt werden, den einzelnen Benutzern den Zugriff erlauben oder verbieten, vgl. [2], Chapter 7.2.2. Schlüssel oder Paßwörter für Files gibt es nicht.
- **Zugriffskoordination**: Explizites assignieren/freigeben (SPERRY) oder Link setzen/lösen (IBM) gibt es nicht. Falls mehrere Jobs auf den gleichen File zugreifen wollen, erfolgt die Koordination beim Öffnen/Schließen des Files im ausführbaren Programm.
- **File-Organisation**: mit FORTRAN und PASCAL Programmen können bearbeitet werden
  .sequentielle Files (nur sequentieller Zugriff)
  .direkte Files (Zugriff über relative Satznummer, feste Satzlänge)
  .indexsequentielle Files (Zugriff über Index)
  Der Editor EDT kann nur sequentielle (z.B. von FORTRAN oder PASCAL Programmen erzeugte) Files bearbeiten.

DCL Commands zum Arbeiten mit Files:

- **Kreieren**: implizit, vgl. 5.
- **Informieren**: DIRECTORY/FULL *file-spec*
- **Umbenennen**: RENAME
- **Kopieren**: COPY
- **Löschen**
  spezielle Version: DELETE
  alle Versions bis auf höchste: PURGE
- **File-Bearbeitung**: ASSIGN, vgl. 5.4

Dabei ist *file-spec* durch die entsprechende File Spezifikation zu ersetzen (z.B. *.FOR;*); RENAME, COPY und DELETE können ohne Parameter eingegeben werden, sie fordern die entsprechenden File Spezifikationen an; PURGE (ohne Parameter) löscht alle alten Versions der Files im Default Directory. Weitere Informationen liefert HELP, vgl. 3.3.1.

4.2 Standard Ein/Ausgabe Files

Der Standard Eingabe File wird in DCL mit dem logischen Namen **SYS$INPUT** bezeichnet, der Standard Ausgabe File mit **SYS$OUTPUT**. DCL Commands wie z.B. EDIT (Editor) oder CREATE (Eintragen von Zeilen in File) oder auch Programme (FORTRAN: READ(5,...)..., PASCAL: READ(INPUT,...)) lesen Eingaben von SYS$INPUT. DCL Commands wie z.B. SHOW oder HELP (ohne /OUTPUT Qualifier) oder auch Programme (FORTRAN: WRITE(6,...)..., PASCAL: WRITE(OUTPUT,...)) schreiben ihre Ausgaben nach SYS$OUTPUT. Beim Login wird den logischen Namen SYS$INPUT, SYS$OUTPUT und **SYS$COMMAND** das Terminal (genauer der Name des Terminal Ports, z.B. TXB2) zugeordnet. Innerhalb eines Jobs können SYS$INPUT und SYS$OUTPUT umdefiniert werden, z.B.

```
$ define sys$input editor.eingaben <RETURN>
```

ordnet SYS$INPUT den File *EDITOR.EINGABEN* zu; wird danach der Editor aufgerufen, so liest er die Eingaben aus dem File *EDITOR.EINGABEN* und nicht vom Terminal.

Der logische Name SYS$COMMAND sollte nicht umdefiniert werden, er dient zum Zurücksetzen von SYS$INPUT, z.B.

```
$ define sys$input sys$command <RETURN>
```

ordnet SYS$INPUT wieder das Terminal zu. Dies wird insbesondere für interaktive Command Procedures benötigt. Beim Start einer Command Procedure (@..., vgl. KOMMANDOSPRACHE) wird SYS$INPUT der Command File zugeordnet, d.h. Standard Eingabe File ist der Command File. Obiges DEFINE Command ermöglicht es, in einer Command Procedure Eingaben vom Terminal zu lesen.

Analog kann SYS$OUTPUT umdefiniert werden; das Zurücksetzen von SYS$OUTPUT auf das Terminal ist jedoch nur möglich mit:

```
$ deassign sys$output <RETURN>
```

5. Programmentwicklung in FORTRAN und PASCAL

Zur Programmentwicklung stehen die höheren Programmiersprachen **FORTRAN 77** und **PASCAL** zur Verfügung. Die Programmentwicklungsumgebungen sind für beide Programmiersprachen annähernd gleich; sie werden deshalb parallel beschrieben; PASCAL Programme können darüber hinaus FORTRAN Unterprogramme aufrufen, FORTRAN Programme können (separat compilierte) PASCAL Prozeduren aufrufen. Dieser Abschnitt ist lediglich als Einstieg konzipiert; weitergehende Informationen finden Sie in den Notizen FORTRAN und PASCAL.

Abb. 5-1 gibt einen Überblick über das Edieren, Compilieren, Binden und Starten eines FORTRAN bzw. PASCAL Programms in der einfachsten Form. Die nachfolgenden Abschnitte enthalten weitere Erläuterungen zu Abb. 5-1 (vgl. auch Chapter 4 in [1]).

Begriffe: Der **Source File** enthält die **Quellprogramme**. Der **Object File** enthält die vom Compiler erzeugten **Objektprogramme**; eine **Object Library** enthält i.a. viele Objektprogramme. Der **Image File** enthält das vom Binder erzeugte **ausführbare Programm**. Der **List File** enthält das vom Compiler, der **Map File** das vom Binder erzeugte Listing.

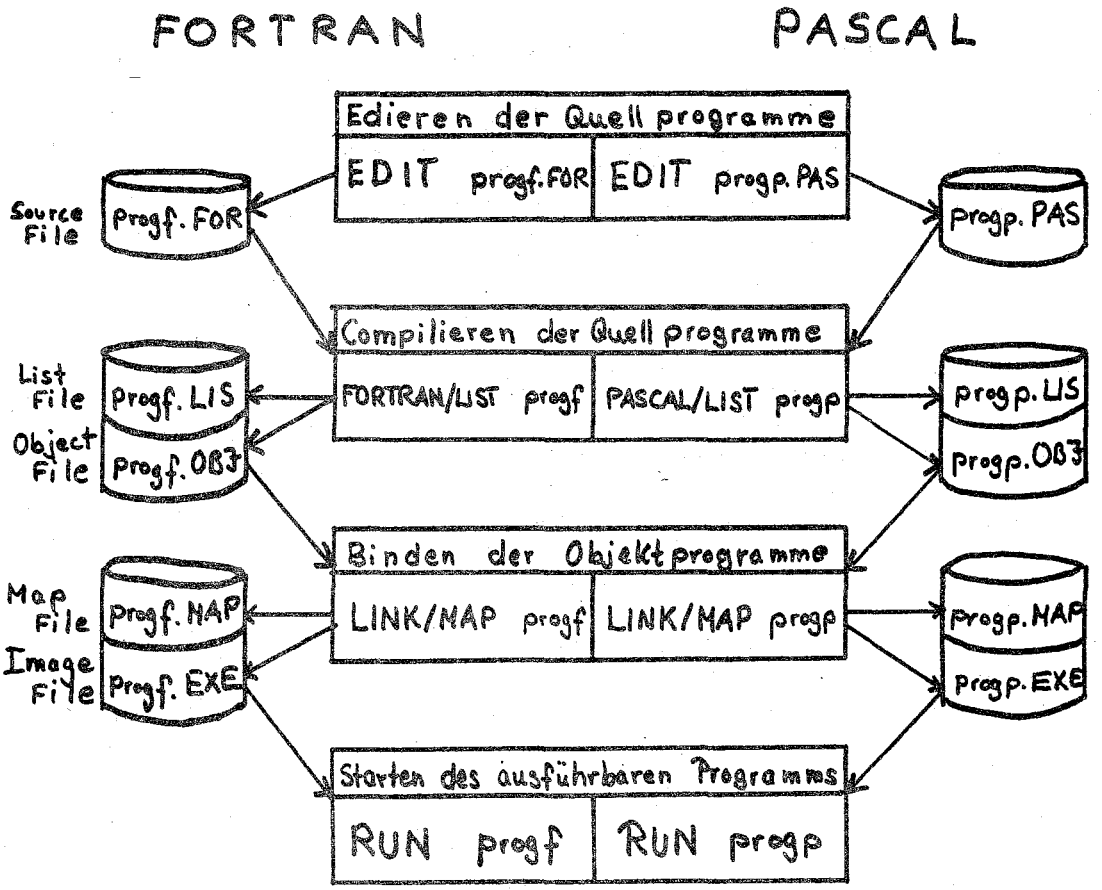

Abb. 5-1: Programmentwicklung

5.1 Edieren (Quellprogramme)

Wenn Sie noch wenig Erfahrung mit Editoren haben, arbeiten Sie bitte zuerst die Chapter 2.1 bis 2.3 von [3] durch. Es wird empfohlen, zunächst nur mit dem VAX Standard Editor EDT zu arbeiten. Der Editor TPU (syn. EVE, Extensible VAX Editor) hat einen EDT-kompatiblen Keypad Mode (s.u.) und wird denjenigen Benutzer empfohlen, die häufig komplexe Edieraufgaben zu bearbeiten haben; er bietet komfortable Möglichkeiten zur Erzeugung eigener Editor Commands.

Beispiel: Edieren eines FORTRAN Programms (Filetype FOR; für PASCAL Programme ist Filetype PAS zu verwenden)

```
                                                    PROG.FOR;1
                                                        .
                                                        .
$ edit prog.for <RETURN>                            PROG.FOR;8
                                                        ↓
... edieren ...                                       Puffer
                                                        ↓
*exit <RETURN>                                      PROG.FOR;9
```

Das **DCL Command EDIT** ruft (ohne Command Qualifier) den VAX Standard Editor EDT auf. Ediert werden können nur Texte im Puffer des Editors. Beim Aufruf des Editors wird - falls bereits Versions von *PROG.FOR* existieren - die höchste Version in den Puffer des Editors kopiert, andernfalls wird ein leerer Puffer bereitgestellt (Meldung: Input File does not exist). Beim EXIT wird eine neue Version von *PROG.FOR* kreiert.

Der EDT kann zeilenorientiert (Line Mode) und bildschirmorientiert (Keypad Mode) arbeiten; im **Line Mode** können nach dem Prompt *** EDT Line Commands** zum edieren des Texts eingegeben werden. Das Umschalten in den Keypad Mode erfolgt mit dem EDT Line Command CHANGE:

```
*change <RETURN>
```

Im **Keypad Mode** kann sofort Text eingegeben werden. Die eingegebenen Zeichen werden auf Cursor-Position eingefügt; der Cursor kann z.B. mit <↑>, <↓>, <←> oder <→> im Text positioniert werden; <RETURN> liefert eine neue Zeile, <DELETE> löscht das Zeichen vor dem Cursor. Darüber hinaus können **EDT Keypad Commands** mit Hilfe der Tasten des Auxiliary Keypad ausgeführt werden; die Belegung der Tasten beschreibt Abb. 5-2. Die meisten Tasten haben zwei Funktionen: das obere Command (z.B. DEL C) wird einfach durch Drücken dieser Taste ausgeführt; zur Ausführung des unteren Commands (z.B. UND C) ist zuerst <PF1> (GOLD) und danach die Taste zu drücken. Online HELP zur Funktion einer Taste kann nach

```
<PF2>
```

(HELP) durch Drücken dieser Taste angefordert werden. Das Umschalten in den Line Mode erfolgt mit

```
<CTRL/Z>
```

PF1 GOLD	PF2 HELP	PF3 FNDNXT FIND	PF4 DEL L UND L
7 PAGE COMMAND	8 SECT FILL	9 APPEND REPLACE	— DEL W UND W
4 ADVANCE BOTTOM	5 BACKUP TOP	6 CUT PASTE	9 DEL C UND C
1 WORD CHNGCASE	2 EOL DEL EOL	3 CHAR SPECINS	ENTER
0 LINE OPEN LINE		• SELECT RESET	SUBS

Abb. 5-2 EDT Keypad Commands

Zum Erfassen von Texten sowie für einfache Korrekturen sollte der Keypad Mode genutzt werden; für kompliziertere Edieraufgaben, wie z.B. das Austauschen von Zeichenfolgen, sollten zunächst EDT Line Commands genutzt werden. Für den Anfang kommen Sie mit folgender Teilmenge der EDT Line bzw. Keypad Commands aus:

vom Line in den Keypad Mode schalten:	EDT Line Command CHANGE
Text eingeben oder einfügen:	im Keypad Mode, s.o.
Zeichen löschen:	EDT Keypad Command DEL C
Zeile löschen:	EDT Keypad Command DEL L
Zeichenfolge suchen:	EDT Keypad Command FIND und FNDNXT,
vom Keypad in den Line Mode schalten:	<CTRL/Z>
Zeichenfolge austauschen:	EDT Line Command SUBSTITUTE
Edieren beenden:	EDT Line Command EXIT

Die Arbeitsweise des EDT kann über Parameter gesteuert werden, die mit Hilfe des EDT Line Command SET eingestellt und mit SHOW abgefragt werden. In diesem Kapitel wird vorausgesetzt, daß keiner der Parameter umgestellt wurde.

Tab. 5-1 enthält alle EDT Line Commands (außer SET und SHOW) mit ihrem Format; die unterstrichenen Angaben sind obligat.

CHANGE *range*
CLEAR *buffer*
COPY *range-1* TO *range-2* /QUERY/DUPLICATE:*n*
DEFINE MACRO *macroname*
DELETE range /QUERY
EXIT *file-spec* /SEQUENCE:*initial*:*increment*/SAVE
FIND *range*
HELP *topic subtopic*
INCLUDE *file-spec range*
INSERT *range*;*line-to-be-inserted*
MOVE *range-1* TO *range-2* /QUERY
PRINT *file-spec range*
QUIT/SAVE
REPLACE *range*;*line-to-be-inserted*
RESEQUENCE *range* /SEQUENCE:*initial*:*increment*
SUBSTITUTE /*string-1*/*string-2*/ *range* /BRIEF:*n*/QUERY/NOTYPE
SUBSTITUTE NEXT /*string-1*/*string-2*/
TYPE *range* /BRIEF:*n*/STAY
WRITE *file-spec range* /SEQUENCE:*initial*:*increment*

Tab. 5-1: EDT Line Commands

Die Leistung der aufgelisteten Commands wird größtenteils durch ihren Namen beschrieben; obiges HELP Command gibt Informationen über diese Commands am Terminal aus, z.B. informiert

***help find** <RETURN>

über Leistung und Format des EDT Line Editing Commands FIND.

Range		
Zeile	Zeilenbereich	
	zusammenhängend	nicht zusammenhängend
. +n -n n | n.m 'string' - 'string' BEGIN END LAST	Zeile-1:Zeile-2 Zeile#n BEFORE REST WHOLE	Zeile-1,Zeile-2,... ALL string zus.-Zeilenbereich ALL 'string'

Tab. 5-2: Range in EDT Line Commands

Der Range bezeichnet eine Menge von Zeilen im Puffer des Editors; er kann mit Hilfe von **Zeilennummern** (*n* bzw. *n.m*), Zeichenfolgen (*'string'* oder *-'string'*) oder relativ zur **aktuellen Zeile** (*+n* oder *-n*) angegeben werden, vgl. Tab. 5-2. Weitere Informationen liefert

```
*help range <RETURN>
```

Groß/Kleinschreibung: In EDT Commands sowie beim Suchen von Zeichenfolgen werden kleine Buchstaben wie große interpretiert; beim Eintragen von Zeichen werden genau die eingegebenen Zeichen eingetragen, z.B.

```
SUBSTITUTE /Hans/Otto/ WHOLE
```

tauscht auch die Zeichenfolgen *HANS,HANs, HAns,...* durch die Zeichenfolge *Otto* aus.

FORTRAN und PASCAL Compiler interpretieren kleine Buchstaben - außer in Character Konstanten - wie große. Vorsicht: der FORTRAN Compiler der IBM 4381 akzeptiert nur große Buchstaben.

Beispiele:

```
*find 'susi' <RETURN>
*change <RETURN>
```

FIND sucht, (beginnend hinter der aktuellen Zeile) die Zeichenfolge *SUSI* (s.o.); CHANGE schaltet vom Line Mode in den Keypad Mode um und positioniert den Cursor hinter die gefundene Zeichenfolge.

```
<CTRL/Z>
*substitute /susi/Sabine/ rest <RETURN>
```

<CTRL/Z> schaltet von Keypad Mode in den Line Mode um; SUBSTITUTE tauscht (beginnend in der Zeile, in welcher der Cursor stand) die Zeichenfolge *SUSI* (s.o.) durch die Zeichenfolge *Sabine* aus.

5.2 Compilieren (Quellprogramme)

Problem: Compiliert werden sollen die Quellprogramme aus dem Source File

PROG.FOR
bzw.
PROG.PAS

Lösung: Das **DCL Command FORTRAN** bzw. **PASCAL** ruft den **FORTRAN** bzw. **PASCAL Compiler** auf.

```
$ fortran/list prog <RETURN>
```

bzw.

```
$ pascal/list prog <RETURN>
```

compiliert die Quellprogramme des Source Files *PROG*.FOR bzw. *PROG*.PAS (höchste Version), schreibt die erzeugten **Objektprogramme** in den Object File *PROG*.OBJ (neue Version) und das **Compiler Listing** (/LIST Qualifier) in den List File *PROG*.LIS (neue Version); der List File kann entweder mit dem Editor gelesen (vgl. 5.1) oder mit PRINT gedruckt werden (vgl. 3.5.3); ohne /LIST Qualifier wird kein Compiler Listing erzeugt.

Die File Spezifikationen von Object File und List File können auch explizit angegeben werden (statt /LIST sind die Qualifiers/OBJECT=*file-spec-1*/LIST=*file-spec-2* zu verwenden); eine ausführliche Beschreibung des FORTRAN bzw. PASCAL Commands finden Sie in der Notiz FORTRAN bzw. PASCAL.

5.3 Binden (Objektprogramme)

Problem: Gebunden werden sollen die Objektprogramme (Hauptprogramm, ggf. mit Unterprogrammen) aus dem Object File

PROG.OBJ

Lösung: Das **DCL Command LINK** ruft den Binder auf.

```
$ link prog <RETURN>
```

bindet die Objektprogramme aus *PROG*.OBJ (höchste Version) mit Objektprogrammen aus der Run-Time Library zu einem **ausführbaren Programm** und schreibt dieses (als Image) in den Image File *PROG*.EXE (neue Version).

Werden FORTRAN Unterprogramme oder PASCAL Prozeduren aus Object Libraries (z.B. GKS Unterprogramme) aufgerufen, so muß die File Spezifikation dieser Object Library (im Beispiel DISK0:[ANW.TEKGKS]LIB) im LINK Command zusammen mit dem File Qualifier /LIBRARY explizit angegeben werden:

```
$ link prog,disk0:[anw.tekgks]lib/library <RETURN>
```

Die File Spezifikation des Image Files kann auch explizit angegeben werden (LINK/EXECUTABLE=*file-spec*...). Die Erstellung eines **Binder Listings** (Map File) muß explizit angefordert werden (LINK/MAP=*file-spec*...).

5.4 File-Bearbeitung

Zur File-Bearbeitung müssen FORTRAN bzw. PASCAL Programme die File Spezifikationen nicht kennen; sie können vielmehr logische Namen verwenden, denen mit Hilfe des **DCL Commands ASSIGN** die File Spezifikationen dann zuzuordnen sind (vgl. KOMMANDOSPRACHE). Im Interaktiven Job wird die Eingabe vom Terminal mit dem logischen Namen SYS$INPUT bezeichnet, die Ausgabe auf das Terminal mit dem logischen Namen SYS$OUTPUT, vgl. 4.2.

5.4.1 FORTRAN: Zuordnung von Unit Numbers zu Files

Es wird empfohlen, jeden File im FORTRAN Programm mit der OPEN Anweisung explizit zu öffnen und dabei der **Unit Number** einen logischen Namen zuzuordnen, z.B.

```
OPEN (UNIT=10,FILE='DATEI', ...)
```

Diesem logischen Namen (im Beispiel *DATEI*) ist vor dem Start des ausführbaren Programms die File Spezifikation des zu bearbeitenden Files (z.B. *PERSONAL.DATEN*) zuzuordnen:

```
$ assign  personal.daten  datei <RETURN>
```

Welche Version von *PERSONAL.DATEN* beim Öffnen zugeordnet wird, hängt vom STATUS Parameter der OPEN Anweisung ab. Bei STATUS='NEW' wird eine neue Version kreiert, bei STATUS='OLD' wird der File mit der höchsten Version zugeordnet.

Statt einer File Spezifikation kann auch ein logischer Name verwendet werden, dem bereits ein File zugeordnet ist, z.B. SYS$INPUT:

```
$ assign  sys$input  datei <RETURN>
```

Per Default gelten die Zuordnungen

```
UNIT = 5  SYS$INPUT      Terminal Eingabe
UNIT = 6  SYS$OUTPUT     Terminal Ausgabe
UNIT = nn FOR0nn         File FOR0nn.DAT
```

Weitere Informationen: Notiz FORTRAN.

5.4.2 PASCAL: Zuordnung von File Variablen zu Files

Eine **File Variable** des PASCAL Programms, z.B. *DATEI*:

```
PROGRAM XYZ (DATEI, ...);
VAR DATEI : FILE OF CHAR;
...
```

kann in DCL Commands als logischer Name verwendet werden. Diesem logischen Namen ist vor dem Start des ausführbaren Programms die File Spezifikation des zu bearbeitenden Files (z.B. *PERSONAL.DATEN*) zuzuordnen:

```
$ assign personal.daten datei <RETURN>
```

Welche Version von *PERSONAL.DATEN* zugeordnet wird, hängt von der Prozedur ab, mit der die File-Bearbeitung begonnen wird: REWRITE kreiert eine neue Version, RESET ordnet den File mit der höchsten Version zu.

Statt einer File Spezifikation kann auch ein logischer Name verwendet werden, dem bereits ein File zugeordnet ist, z.B. SYS$INPUT:

```
$ assign sys$input datei <RETURN>
```

Per Default gelten die Zuordnungen

```
INPUT     SYS$INPUT      Terminal Eingabe
OUTPUT    SYS$OUTPUT     Terminal Ausgabe
```

Weitere Informationen: Notiz PASCAL

5.5 Starten (ausführbares Programm)

Das **DCL Command RUN**

```
$ run  prog <RETURN>
```

startet das ausführbare Programm im Image File *PROG*.EXE (höchste Version).

6. Beispiele für interaktive Jobs

6.1 FORTRAN Programm

```
<RETURN>
*** A N N O U N C E ***

DECNET HRZ Uni Marburg, Knoten DECMRG (Graphik-Rechner)

+++ HRZ UNI MARBURG GRAPHIK-RECHNER DEC VAX 11/750++++++++++
+                                                          +
+ 01.11.1989 Es gibt neue Mitteilungen.                    +
+            Bitte NEU ausfuehren.                         +
+                                                          +
++++++ NACHRICHT ++++++ NACHRICHT ++++++ NACHRICHT +++++++++

Username: meier_g <RETURN>
Password: ...     <RETURN>

*** W E L C O M E ***

VAX/VMS 5.1

    Last interactive login on ...
    Last non-interactive login ...
*** S Y S T E M  L O G I N  C O M M A N D  F I L E ***

USERNAME        :MEIER_G
TERMINAL        :RTA1:
DATE            :1-NOV-1989 08:18:29.41

 ***  U S E R    L O G I N  C O M M A N D  F I L E ***


$ !Edieren: <RETURN>
$ edit  wurzel.for <RETURN>
Input file does not exist
[EOB]
*change <RETURN>
      program wurzel <RETURN>
10    read(5,*) r <RETURN>
      if(r.lt.0.) stop <RETURN>
      write(6,*) 'SQRT(',r,') = ',sqrt(r) <RETURN>
      goto 10 <RETURN>
      end
[EOB]
... Cursor in der 2. Zeile vor "read" positionieren und Anweisung

      write(6,*) 'Bitte REAL Zahl eingeben:' <RETURN>

... einfügen, dann "read (5,*) r" durch Eingabe von Blanks verschieben

<CTRL/Z>
```

```
*type whole <RETURN>
    1               program wurzel
    2         10    write(6,*) 'Bitte REAL Zahl eingeben:'
    2.1             read(5,*) r
    3               if(r.lt.0.) stop
    4               write(6,*) 'SQRT(',r,') = ',sqrt(r)
    5               goto 10
    6               end
[EOB]
*exit <RETURN>
DISK1:[MEIER_G]WURZEL.FOR;1 7 lines

$ !Compilieren: <RETURN>
$ fortran/list  wurzel <RETURN>

$ !Listing drucken: <RETURN>
$ print/delete  wurzel.lis <RETURN>
Job WURZEL (queue MHRZ4, entry 1123) started on MHRZ4

$ !Binden: <RETURN>
$ link  wurzel <RETURN>

$ !Starten: <RETURN>
$ run  wurzel <RETURN>
Bitte REAL Zahl eingeben:
49. <RETURN>
SQRT(    49.00000     ) =     7.000000
Bitte REAL Zahl eingeben:
123. <RETURN>
SQRT(    123.0000     ) =     11.09054
Bitte REAL Zahl eingeben:
-8. <RETURN>
FORTRAN STOP

$ !Job beenden: <RETURN>
$ logout <RETURN>
  MEIER_G     logged out at ...
```

6.2 PASCAL Programm

```
<RETURN>
*** A N N O U N C E ***

DECNET HRZ Uni Marburg, Knoten DECMRG (Graphik-Rechner)

+++ HRZ UNI MARBURG GRAPHIK-RECHNER DEC VAX 11/750++++++++++
+                                                          +
+ 01.11.1989 Es gibt neue Mitteilungen.                    +
+            Bitte NEU ausfuehren.                         +
+                                                          +
++++++ NACHRICHT ++++++ NACHRICHT ++++++ NACHRICHT +++++++++

Username: meier_g <RETURN>
Password: ...    <RETURN>

*** W E L C O M E ***

VAX/VMS 5.1

    Last interactive login on ...
    Last non-interactive login on ...
*** S Y S T E M  L O G I N  C O M M A N D  F I L E ***

USERNAME       :MEIER_G
TERMINAL       :RTA2:
DATE           :1-NOV-1989 10:33:45.32

 ***  U S E R   L O G I N  C O M M A N D  F I L E ***


$ !Edieren: <RETURN>
$ edit  wurzel.pas <RETURN>
Input file does not exist
[EOB]
*change <RETURN>
program wurzel(input,output); <RETURN>
var r : real; <RETURN>
begin <RETURN>
  repeat <RETURN>
    readln(input,r); <RETURN>
    if r >= 0.0 then <RETURN>
      writeln(output,'SQRT(',r,')=',sqrt(r)); <RETURN>
  until r < 0.0; <RETURN>
end.
[EOB]
...   Cursor in der 5. Zeile vor "readln" positionieren und Anweisung

      writeln(output,'Bitte REAL Zahl eingeben:') <RETURN>

...   einfügen, dann "readln(input,r);" durch Eingabe von Blanks
...   verschieben

<CTRL/Z>
```

```
*type whole <RETURN>
    1       program wurzel(input,output);
    2       var r : real;
    3       begin
    4         repeat
    5           writeln(output,'Bitte REAL Zahl eingeben:');
    5.1         readln(input,r);
    6           if r >= 0.0 then
    7             writeln(output,'SQRT(',r,')=',sqrt(r));
    8         until r < 0.0;
    9       end.
[EOB]
*exit <RETURN>
DISK1:[MEIER_G]WURZEL.PAS;1 10 lines

$ !Compilieren: <RETURN>
$ pascal/list  wurzel <RETURN>

$ !Listing drucken: <RETURN>
$ print/delete  wurzel.lis <RETURN>
Job WURZEL (queue MHRZ4, entry 1143) started on MHRZ4

$ !Binden: <RETURN>
$ link  wurzel <RETURN>

$ !Starten: <RETURN>
$ run  wurzel <RETURN>
Bitte REAL Zahl eingeben:
49. <RETURN>
SQRT( 4.90000E+01)= 7.00000E+00
Bitte REAL Zahl eingeben:
16. <RETURN>
SQRT( 1.60000E+01)= 4.00000E+00
Bitte REAL Zahl eingeben:
-8. <RETURN>

$ !Job beenden: <RETURN>
$ logout <RETURN>
  MEIER_G      logged out at ...
```