de Gruyter Lehrbuch

Hesse/Kirsten · Einführung in die Programmiersprache MUMPS

Stephan Hesse · Wolfgang Kirsten

Einführung in die Programmiersprache MUMPS

2., völlig neubearbeitete Auflage

Walter de Gruyter · Berlin · New York 1989

Dipl. inform. Stephan Hesse
Mitarbeiter der Firma Telenet Kommunikationssysteme, Geschäftsstelle Rhein-Main, Darmstadt

Dr. rer. med. Wolfgang Kirsten
Mitarbeiter der Abteilung für Dokumentation und Datenverarbeitung am Zentrum der Medizinischen Informatik des Klinikums der Johann Wolfgang Goethe-Universität, Frankfurt

CIP-Titelaufnahme der Deutschen Bibliothek

Hesse, Stephan:
Einführung in die Programmiersprache MUMPS / Stephan Hesse ; Wolfgang Kirsten. – 2., neubearb. Aufl. – Berlin ; New York : de Gruyter, 1989
(De-Gruyter-Lehrbuch)
ISBN 3-11-011598-0
NE: Kirsten, Wolfgang:

Vorwort

MUMPS ist eine der wenigen von der amerikanischen Normierungsbehörde ANSI standardisierten Programmiersprachen.

Die Abkürzung MUMPS steht für „Massachusetts General Hospital's Utility Multiprogramming System" und kennzeichnet so die Herkunft, jedoch in keiner Weise das ausschließliche Einsatzgebiet der Sprache.

Das vorliegende Buch stellt die zweite Auflage der 1983 erschienenen einführenden Darstellung in MUMPS dar. Seit dieser Zeit ist in weiten Fachkreisen eine zunehmende Kenntnis dieser in mancher Hinsicht ungewöhnlichen Programmiersprache zu verzeichnen, die sich auch in einer Verbreitung sowohl im traditionellen medizinischen Segment als auch — und hier vor allem — in den kommerziellen Anwendungen niedergeschlagen hat. Die Entscheidung bedeutender europäischer und amerikanischer Hersteller, MUMPS neben den traditionellen Programmiersprachen auf ihren Betriebssystemen anzubieten, hat und wird die weitere Verbreitung fördern. Zu beobachten ist ferner eine zunehmende Öffnung von MUMPS zu anderen Standards wie GKS oder SQL.

Sowohl die erste wie auch die zweite Auflage sind aus einführenden Vorlesungen zu MUMPS am Klinikum der J.W.Goethe-Universität in Frankfurt hervorgegangen. Hörer dort sind Studenten der Medizin und der Informatik. Hieraus leitet sich auch die Zielgruppe dieses Lehrbuchs ab: informierte Laien, die auf MUMPS basierende Anwendungen beruflich oder privat nutzen, sowie Programmierer anderer Programmiersprachen, die MUMPS als weitere Programmiersprache kennenlernen wollen.

Gegenüber der ersten Auflage wurde die zweite Auflage vollständig überarbeitet und erweitert. Insbesondere das in MUMPS zentrale Konzept der Dateibehandlung als integraler Bestandteil der Sprache wurde der Bedeutung gemäß beträchtlich erweitert und durch eine praxisbezogene Studie über das Design von globalen Variablen ergänzt.

Die konzeptionelle Überarbeitung hat auch zu einer neuen Gliederung geführt. Grundsatz des Stoffinhalts der einzelnen Kapitel ist die vollständige Einführung aller Sprachelemente, sowie die Darstellung ihrer Verwendungsart am Beispiel typischer Programmsegmente. Diese Programme können als Teil einer Gesamtanwendung aufgefaßt werden, so daß der Leser in die Lage versetzt wird, die wesentlichen Teile einer dialogorientierten Anwendung zu verstehen und in MUMPS zu realisieren.

Wie schon 1983, als die damalige Überarbeitung des Standards komplett berücksichtigt wurde, ist auch in der zweiten Auflage der zu erwartende neue Standard von 1989 vollständig eingearbeitet.

Da diese Spracherweiterungen zum Teil gravierender Natur sind, ist dieses Buch auch erfahrenen Programmierern von Nutzen, die sich über diese Änderungen informieren möchten.

Der Anhang enthält neben der Zusammenfassung der Sprachelemente eine Übersicht über die Änderungen und Erweiterungen des neuen Standards und kann daher als Nachschlagewerk benutzt werden. Zusätzlich aufgenommen wurden die formalen Definitionen von Befehlen, Funktionen und speziellen Variablen in der Backus-Naur-Form.

Interessierten Lesern kann von den Autoren eine preiswerte Implementierung von MUMPS für den Einbenutzerbetrieb vermittelt werden. Die Adresse findet sich im Anschluß an das Literaturverzeichnis.

Bei der Neufassung des Lehrbuches haben uns viele Kollegen, Mitarbeiter und Freunde durch fortgesetzten Rat geholfen. Allen möchten wir unseren herzlichen Dank aussprechen und die Bitte äußern, uns auch weiterhin zu unterstützen.

Frau Birgit Graf hat größere Teile des Manuskripts geschrieben und Frau Dr. Andrea Kilian hat die mühevolle Aufgabe der Korrektur des Textes übernommen. Auch Ihnen sei vielmals gedankt.

Die Textverarbeitung dieses Buches wurde bei der Telenet GmbH, Geschäftsstelle Rhein-Main in Darmstadt besorgt. Für die Überlassung von Rechenleistung und die Unterstützung beim Ausdruck möchten wir uns bei Geschäftsstellenleiter Dr. Ian Nicholls ausdrücklich bedanken.

Besonderer Dank gilt Herrn Dr. Günter Schuller vom Rechenzentrum der Universität Würzburg, der das Manuskript sorgfältig und kritisch gelesen hat. Seine Vorschläge insbesondere zur Darstellung des Konzepts der globalen Variablen wurden dankbar übernommen.

Im Zusammenhang mit der Entstehungsgeschichte dieses Buches gilt ein besonderes Wort des Dankes Herrn Prof. Dr. Wolfgang Giere, Leiter der Abteilung für Dokumentation und Datenverarbeitung am Klinikum der J.W. Goethe-Universität in Frankfurt, der die Vorlesungsreihe über MUMPS bereits Ende der siebziger Jahre als festen Bestandteil des Lehrangebots im Bereich der Medizinischen Informatik etablierte.

Wir danken ihm insbesondere auch für die aktive Unterstützung, für den immerwährenden Meinungsaustausch und die detaillierte Durchsicht des Manuskripts mit vielen wertvollen Hinweisen.

Ferner danken wir dem Verlag de Gruyter für die fortgesetzte angenehme Zusammenarbeit.

Frankfurt, im Oktober 1988

Stephan Hesse
Wolfgang Kirsten

Inhalt

Kapitel 1

Grundlagen der Programmierung in MUMPS

1.1 Einführung

1.1.1 Programmiersprachen in der historischen Entwicklung — ein Überblick

Mit dem Aufkommen der programmgesteuerten Rechner, deren erste experimentelle Ausführungen Ende der dreißiger, Anfang der vierziger Jahre zur Verfügung standen, war den Menschen ein Mittel an die Hand gegeben, komplizierte, numerische Berechnungen schneller als mit den bisher eingesetzten manuell zu bedienenden Rechenmaschinen durchzuführen.

Es bereitete jedoch unendliche Mühe, zur Lösung eines mathematischen Problems den Rechner zu instruieren. Hierfür mußten lange binäre oder oktale Zahlenreihen eingegeben werden. Die Anweisungen, zwei Variablen a und b zu multiplizieren und das Ergebnis einer dritten Variablen z zuzuweisen, hätte etwa wie folgt aussehen können:

```
021 057 314
062 216 105
142 273 041
```

Bereits wenig später wurden jedoch die Grundlagen von Programmiersprachen im heutigen Sinne geschaffen, in dem man — Eselsbrücken gleich — Symbole für häufig gebrauchte Operationen einführte und anstatt numerischer Adressen im Rechner symbolische Namen benutzte. Damit wäre das Beispiel etwa wie folgt dargestellt worden:

```
LOAD A
MPY B
STORE Z
```

Die in dieser Form geschriebenen Programme wurden mittels eines eigenen Programms, das man Assembler nannte, wieder in einen maschinenlesbaren Code übersetzt. Diese Form der Notation nannte man daher eine Assemblersprache.

Obwohl diese Art der „Programmierung“ eines Rechners eine deutlich höhere Produktivität ermöglichte, war es immer noch sehr schwierig, Lösungsverfahren für komplizierte Probleme in einer auch für die Maschine verständlichen Form darzustellen.

Ein weiterer Entwicklungssprung fand Mitte der fünfziger Jahre statt, als eine Gruppe von Computerspezialisten unter der Leitung von John Backus ein Projekt zur automatischen Übersetzung von mathematischen Formeln in maschinenlesbare Instruktionen begann. Das Resultat dieser Bemühungen war FORTRAN (FORmular TRANslator), die erste höhere Programmiersprache in der Geschichte der EDV. In FORTRAN wurde das oben angeführte Beispiel als

```
Z=A*B
```

dargestellt. Man mußte dabei nicht mehr auf maschineninterne Register und Speicherzellen Bezug nehmen, sondern konnte beliebige Variablen mit symbolischen Namen ansprechen.

FORTRAN ist das bekannteste Beispiel von höheren Programmiersprachen, die Mitte der fünfziger Jahre entwickelt wurden. Gemeinsam ist ihnen allen, daß man sie zur Lösung von numerischen Problemen benutzte und daß sie in erster Linie im Hinblick auf leichte Übersetzbarkeit und nicht auf leichte Lesbarkeit entworfen wurden.

Dies änderte sich in den darauffolgenden Jahren, als mit der Entwicklung einer neuen Programmiersprache begonnen wurde, die das ursprüngliche Einsatzfeld der Datenverarbeitung beträchtlich erweiterte. Der Name der Programmiersprache ist COBOL (COmmon Business Oriented Language) und sie zielte auf kaufmännische Anwendungen, also die Verarbeitung von großen Mengen von Daten, wobei im allgemeinen nur einfache arithmetische Operationen benutzt werden. Hierunter fällt beispielsweise die Berechnung der monatlichen Lohn- und Gehaltszahlungen an die Mitarbeiter eines Unternehmens.

In COBOL werden in hohem Maße Worte der englischen Sprache direkt benutzt, so daß das Beispiel folgende Form annimmt:

```
MULTIPLY A BY B GIVING Z.
```

Ein anderer Aspekt, der zunehmend ins Bewußtsein rückte, war der, daß man höheren Wert auf die Unabhängigkeit von Sprachen von einem bestimmten Rechnerhersteller legte.

Dies führte später zur Standardisierung von FORTRAN und COBOL durch unabhängige öffentliche Institutionen, wie etwa das American National Standard Institute (ANSI).

1.1.2 Die Entwicklung von MUMPS

Zu Beginn der sechziger Jahre setzte ein Prozeß ein, der die Struktur und Anwendungsvielfalt der damaligen Datenverarbeitung gründlich veränderte. Während bis dahin Datenverarbeitung fast ausschließlich auf großen Anlagen in noch größeren Rechenzentren im sogenannten Batchbetrieb ausgeführt wurde, konnten nun mehrere Benutzer gleichzeitig interaktiv am Rechner arbeiten. Dieses Prinzip der Benutzung einer Datenverarbeitungsanlage ist unter der Bezeichnung Timesharing-Betrieb bekannt geworden. Der Zugang zur Rechenleistung einer DV-Anlage war dadurch beträchtlich vereinfacht, sowohl für den Programmierer der unmittelbarer und direkter Programme schreiben konnte, als auch für den Nutzer einer Anwendung, der über eine schreibmaschinenähnliche Tastatur Daten von seinem Arbeitsplatz aus eingeben konnte.

Im Jahre 1967 begann eine Entwicklungsgruppe am Laboratory of Computer Science am Massachusetts General Hospital unter der Leitung von G. Octo Barnett und Neil Pappalardo für eine bis dahin gänzlich neue Benutzergruppe — medizinisch orientierte Anwender — Anforderungen an eine neue Programmiersprache zu definieren. Dies geschah in Anlehnung an frühere Arbeiten der Rand Corporation, die ein für den Timesharing-Betrieb geeignetes Betriebssystem entwickelt hatte.

Hauptziel war dabei weniger die mathematisch orientierte Anwendung, sondern die im Bereich der medizinischen Dokumentation notwendige Verarbeitung von textueller Information. Alle bis dahin entwickelten Sprachen taten sich außergewöhnlich schwer bei der Verarbeitung von variabel langen Zeichenketten. Als weiteres Entwicklungsziel war eine interaktive Sprache vorgegeben, die in einem Mehrbenutzerbetrieb operiert und auf den damals verfügbaren Kleinrechnern arbeiten sollte. Die dauerhafte Abspeicherung von Daten und die Wiedergewinnung von Information hieraus sollte in der Sprache integriert sein.

Der Name der daraus hervorgegangenen Programmiersprache ist MUMPS[1]. Dieses Kunstwort ist die etwas unglückliche Abkürzung von „Massachusetts General Hospital's Utility Multiprogramming System". Unglücklich deshalb, weil der Name eine hauptsächlich medizinische Nutzung suggeriert und vielleicht auch deshalb, weil der Name auf eine Sprache schließen läßt, die mehr experimentellen Charakter hat, aber für „ernsthafte" Anwendungen nicht bestimmt ist. Beide Folgerungen sind jedoch in keiner Weise begründet.

Tatsächlich hat MUMPS weiterhin im medizinischen Bereich eine große Verbreitung. Sehr erfolgreich operierende Krankenhausinformationssysteme sind in MUMPS geschrieben, große öffentliche Krankenhausträger in den

[1]MUMPS ist eingetragener Handelsname des Massachusetts General Hospital

USA, wie die Veterans Administration, benutzen ausschließlich auf MUMPS basierende Systeme. Der größte Einzelauftrag in der Geschichte von MUMPS wurde für die weltweit 750 Militärkrankenhäuser des amerikanischen Verteidigungsministeriums mit einem Auftragswert von über einer Milliarde Dollar Anfang 1988 vergeben.

Es wird jedoch angenommen, daß weltweit deutlich mehr als die Hälfte der MUMPS Anwendungen im kommerziellen Bereich anzutreffen sind. Darunter fallen Systeme aus der Lagerwirtschaft, Reservierungs- und Buchungssysteme, Schiffahrtssysteme, große Bankenanwendungen, Systeme zur Produktionssteuerung, mehrsprachige Textdokumentationssteme und viele andere Anwendungen mit zum Teil mehr als 1000 Bildschirmen.

Ein wichtiger Aspekt der Sprache ist die sehr weitgehende Standardisierung von MUMPS. Bereits 1977 wurde MUMPS als dritte Sprache nach FORTRAN und COBOL nach den Regeln des amerikanischen Standardisierungsbüros (abgekürzt ANSI, dem deutschen DIN-Institut vergleichbar) standardisiert und wird daher auch als ANS MUMPS bezeichnet. Da die Datenhaltung integraler Bestandteil der Sprache ist, gelingt es, vollkommen portable — also zwischen Rechnern verschiedener Hersteller übertragbare — Anwendungen zu schreiben. Die Prozeduren der Standardisierung und der Weiterentwicklung der Sprache sind im Anhang beschrieben.

Standardrevisionen erfolgten 1984 und 1989. Der vorliegende Text entspricht dem Standard von 1989 (ANSI X.11–1989). Einen vollständigen Überblick über die Erweiterungen des neuen Standards gegenüber dem von 1984 findet der Leser im Anhang.

1.1.3 Ein Kaleidoskop der Sprache — MUMPS im Überblick

Grundsätzlich unterscheidet sich MUMPS nicht von anderen prozeduralen Sprachen, wie etwa FORTRAN, COBOL oder C, wenngleich in der Anlage und im Detail beträchtliche Unterschiede sichtbar sind.

Eine der auffälligsten Eigenheiten ist der in MUMPS verwendete Datentyp der variabel langen Zeichenkette. Die meisten anderen bekannten Sprachen benutzen mehrere Datentypen (Integer, String, Real, ...), die entweder explizit durch spezielle Deklarationsanweisungen oder implizit durch die Form des Variablennamens einer Variablen zugeordnet sind. In MUMPS dagegen werden alle Informationen als Zeichenketten abgelegt. Umgekehrt kann jede Zeichenkette in MUMPS auch als Zahl oder logischer Wert interpretiert werden. Eine Deklaration ist daher überflüssig.

In konsequenter Fortentwicklung der Idee vom Wegfall aller Deklarationen ist es in MUMPS auch nicht notwendig, Felder zu deklarieren. Jede Variable kann ein Feld beliebiger Anzahl von Dimensionen und nur durch

den verfügbaren Speicher begrenzter Größe darstellen. Es belegen nur diejenigen Variablen Speicherplatz, die tatsächlich im Programm definiert wurden. Man erreicht durch diese Form der Speicherung eine optimale Ausnutzung des Speichers. Gewissermaßen in Fortführung dieses Konzeptes können in MUMPS beliebige Zeichenketten zur Indizierung von Feldern verwendet werden.

Das Prinzip der beliebig tief und mit textlicher Information indizierten Variablen wird in MUMPS auf permanente Datenstrukturen ausgeweitet. Auf diese „globale Variablen" genannten baumartigen Strukturen kann systemweit zugegriffen werden. Im Gegensatz dazu stehen die „lokalen Variablen", deren Lebensdauer durch die Prozeßdauer begrenzt ist.

Der Zugriff auf Variablen — gleich ob lokal oder global — wird durch mächtige Funktionen unterstützt, die MUMPS den Funktionsumfang eines Datenbanksystems geben.

Die Koordinierung der Zugriffe auf globale Variablen in Mehrplatzsystemen geschieht über den `LOCK`-Befehl.

Neben den schon erwähnten Funktionen zum Datenmanagement stehen weitere Funktionen insbesondere aus dem Bereich der Textmanipulation zur Verfügung. Außerdem kann der Programmierer eigene Funktionen definieren und wie Systemfunktionen verwenden.

Weiter existieren arithmetische, logische, stringbezogene und Vergleichsoperatoren, deren Argumente automatisch von MUMPS dem Kontext entsprechend numerisch, logisch oder als Zeichenketten interpretiert werden.

Der Programmfluß wird über die auch in anderen Programmiersprachen üblichen Methoden wie Unterprogrammaufruf, Verzweigung oder bedingte Ausführung gesteuert. Obwohl die Sprache grundsätzlich eine gemeinsame Menge von Variablen für Haupt- und alle Unterprogramme vorsieht, gibt es Sprachelemente wie die explizite Wertübergabe beim Unterprogrammaufruf und den `NEW`-Befehl, die den Sichtbarkeitsbereich von Variablen ausdrücklich begrenzen.

Neben dem `IF`-Befehl zur bedingten Programmausführung existiert noch die „Nachbedingung", die es erlaubt, praktisch jeden einzelnen Befehl oder in einigen Fällen auch Argumente von Befehlen ohne Verzweigung des Programmflusses bedingt auszuführen.

Ähnlich wie die in der Sprache integrierte Datenhaltung sind auch die Ein- und Ausgabeoperationen in der Sprachdefinition festgelegt. Durch dieses hohe Maß an Standardisierung ist es leicht möglich, vollkommen übertragbare Programme mit Bildschirmdialogen zu schreiben.

Schließlich sei noch auf die betriebssystemnahen Sprachelemente und Möglichkeiten verwiesen. Als Beispiel soll der JOB-Befehl genannt werden, der den Start von Hintergrundprozessen erlaubt.

1.2 Benutzerbereiche und das Einloggen

Um nun mit einem MUMPS-System arbeiten und die ersten Versuche mit der Sprache unternehmen zu können, ist es erforderlich, Zugriff zu dem System zu erlangen. Gerade hier unterscheiden sich die verschiedenen Systeme sehr stark. Unter anderem hängt die Vorgehensweise davon ab, ob man an einem Einbenutzer- oder an einem Mehrbenutzersystem arbeitet.

Während typische MUMPS-Systeme Mehrbenutzersysteme sind, bei denen also mehrere Benutzer gleichzeitig arbeiten können, ist es zum Erlernen der Sprache völlig ausreichend, über ein Einbenutzersystem zu verfügen. Solche Systeme, die häufig sogar den parallelen Betrieb mehrerer Prozesse erlauben, werden meistens auf Basis eines PCs unter PC/MS-DOS angeboten. Ein preiswertes Einbenutzersystem kann von den Autoren vermittelt werden.

Zusammen mit der Diskette eines Einbenutzersystems erhält man gewöhnlich eine Anleitung, die alle Informationen enthält, wie man MUMPS auf einen Personal Computer installiert. Hier soll nicht auf Einzelheiten eingegangen werden. Es sei nur noch darauf hinweisen, daß man nach erfolgreicher Installation das System im allgemeinen durch Eingabe des Wortes „MUMPS“ startet.

Bei Mehrbenutzersystemen muß man sich im allgemeinen zuerst durch Eingabe einer Benutzerkennung identifizieren (englisch: „User Code Identification“, UCI). Oft verlangt das System aus Sicherheitsgründen noch die Eingabe einer zweiten Kennung, des „Programmer's Access Code“ (abgekürzt: PAC).

Nach erfolgreich abgeschlossener Berechtigungsprüfung erlaubt das System die Eingabe von Befehlen. Das können bei Standalone-Systemen schon MUMPS-Befehle sein (das sind Systeme, bei denen die Sprache MUMPS in das Betriebssystem integriert ist und die daher auch nur diese eine Sprache verstehen). Andere Systeme, die mehrere Sprachen verstehen, müssen erst noch durch ein Kommando angewiesen werden, den Sprachprozessor für MUMPS zu starten.

Wenn schließlich das MUMPS-System bereit ist, hat es für den Benutzer einen eigenen Teil des Arbeitsspeichers reserviert, der oft auch „Partition“ genannt wird. Dort können Programme des Benutzers ablaufen und ihre Daten hinterlegen.

Weiterhin verwalten MUMPS-Systeme permanente Daten und Programme (solche, die auch nach dem Ende eines Programmlaufs erhalten bleiben sollen) auf dem Massenspeicher (Platte). Dabei sehen Mehrbenutzersysteme eigene Bereiche für jeden Benutzer oder für Benutzergruppen vor.

Dadurch wird gewährleistet, daß sich die Benutzer nicht gegenseitig behindern. Die Zuordnung, welcher Benutzer in welchem Bereich arbeiten soll,

geschieht über die eingegebene UCI. Ein einzelner Benutzer sieht von dem Mehrbenutzersystem nur den seiner UCI zugeordneten Bereich mit seinen Programmen und Daten, so daß das System für ihn den Anschein eines Einbenutzersystems hat. Auf Ausnahmen wird später noch hingewiesen werden.

1.3 Einfache syntaktische Regeln

Wie in jeder Programmiersprache gibt es auch in MUMPS eine Reihe von einfachen Absprachen, die den äußeren Aufbau einer Programmzeile festlegen.

Einleuchtend und fast selbstverständlich ist die Tatsache, daß eine Zeile von links nach rechts gelesen und abgearbeitet wird.

Die Zeilen selbst enthalten im allgemeinen einen oder mehrere Befehle. Sie bestehen aus Befehlsworten und Befehlsargumenten, die durch genau ein Leerzeichen getrennt werden. Es können mehrere Befehle in einer Programmzeile stehen, die dann mit einem oder mehreren Leerzeichen getrennt werden müssen.

Kommentare stehen am Ende einer Programmzeile und sind von dieser durch ein Semikolon (;) getrennt. Als Sonderfall ist es auch möglich, eine Programmzeile mit dem Semikolon beginnen zu lassen, so daß alle darauf folgenden Zeichen als Kommentar interpretiert werden.

Kommentare werden selbstverständlich nicht ausgeführt, sondern bei der Übersetzung eines MUMPS-Programms ignoriert. Sie sind nur für den menschlichen Leser von Bedeutung.

In der Sprachdefinition wird über die Anzahl der Befehle, die in einer Zeile höchstens stehen dürfen, nur eine indirekte Aussage gemacht: eine Programmzeile darf höchstens 255 Zeichen lang sein, natürlich einschließlich der Leerzeichen und Kommentare. Es sind also so viele Befehle erlaubt, wie in 255 Zeichen passen.

Normalerweise werden Programmzeilen über eine schreibmaschinenähnliche Tastatur eingeben, an die ein Bildschirm angeschlossen ist. Viele Bildschirme gestatten es, in eine Bildschirmzeile 80 Zeichen zu schreiben. Dann springt der Cursor — das ist eine Markierung auf dem Bildschirm, die die Position des nächsten einzugebenden Zeichens anzeigt — an den Anfang der nächsten Zeile. Damit muß keineswegs die Programmzeile abgeschlossen sein. Vielmehr muß man das MUMPS ausdrücklich durch Drücken einer besonderen Taste mitteilen. Diese Taste ist bei den einzelnen Herstellern nicht einheitlich definiert. Häufig wird dafür die „carriage return“ Taste benutzt. In den Beispielen wird das Betätigen dieser Taste mit <cr> symbolisiert.

Die für MUMPS-Anwendungen eingesetzten Bildschirme enthalten meistens 24 Zeilen, jede Zeile enthält 80 Zeichen.

Wenn also eine Bildschirmzeile 80 Zeichen, eine Programmzeile in MUMPS dagegen 255 Zeichen lang ist, dann kann man 3 Bildschirmzeilen und noch 15 Zeichen in der vierten Zeile für eine Programmzeile verwenden. Geht man darüber hinaus, kommt eine Fehlermeldung, die darauf hinweist, daß die Programmzeile zu lang ist und nicht mehr verarbeitet werden

kann. In der Praxis sind Programmzeilen solcher Länge allerdings selten anzutreffen, vielmehr wird man versuchen, die Befehle in einer Bildschirmzeile unterzubringen.

Daß man überhaupt in einer Zeile mehr als jeweils einen Befehl schreibt, ist mehr als nur Konvention. Einige MUMPS-Befehle betrachten eine Programmzeile als ein einheitliches Ganzes.

Wenn im weiteren Verlauf Beispiele angegeben werden, geschieht das in Anlehnung an die übliche Arbeitsweise mit einem MUMPS-System. Hierbei gibt der Rechner immer dann, wenn er bereit ist, eine neue Eingabe zu verarbeiten, ein bestimmtes Zeichen aus, das „Prompt" genannt wird. Dieses Verhalten soll dadurch symbolisiert werden, daß in den Beispielen das Größerzeichen (>) vor jeder Anweisungszeile steht.

Eine Befehlszeile wird also wie folgt dargestellt:

```
>Befehlswort Argument Befehlswort Argument Befehlswort Argum
ent ; Kommentare <cr>
```

Dabei wurde bewußt die Befehlskette über die (Papier-) Zeile hinausgeschrieben und dabei der Beginn der Zeile durch den Prompt gekennzeichnet. Die Abwesenheit des Prompts in der zweiten Zeile zeigt, daß es ein Folgezeile ist und nicht etwa eine neue Befehlszeile.

1.4 Beispiele einfacher Sprachelemente

1.4.1 Der WRITE-Befehl

Einer der einfachsten und zugleich häufig gebrauchten Befehle ist der `WRITE`-Befehl.

Beispiel

```
>WRITE "Hallo"<cr>
Hallo
>
```

Der `WRITE`-Befehl wird dazu verwendet, Texte auf den Bildschirm zu schreiben. In Zusammenhang mit anderen Befehlen sei aber jetzt schon darauf hingewiesen, daß der `WRITE`-Befehl genauso benutzt wird, um auf Drucker oder andere Geräte zu schreiben.

Man beachte, daß zwischen dem Befehlswort `WRITE` und dem Argument `"Hallo"` genau ein Leerzeichen stehen muß!

Die Schreibweise des Argumentes `"Hallo"` zeigt noch eine weitere wichtige Festlegung von MUMPS: Texte, die als unveränderliche Zeichenfolgen im Programmtext stehen sollen, müssen von Anführungszeichen (") eingeschlossen sein.

Möchte man Zahlen schreiben, kann man auf die Anführungszeichen verzichten.

Beispiel

```
>WRITE 2<cr>
2
>WRITE 3+5<cr>
8
>
```

Im letzten Beispiel wird deutlich, daß das Argument des `WRITE`-Befehls zunächst errechnet wird und das Ergebnis dann geschrieben wird. Im `WRITE`-Befehl, aber auch bei (fast) allen anderen Befehlen in MUMPS, kann man mehrere durch Kommata getrennte Argumente angeben. Sie werden direkt nacheinander ohne trennende Leerzeichen oder andere Zeichen ausgegeben.

Beispiel

```
>WRITE "WRITE","mit","mehreren","Argumenten"<cr>
WRITEmitmehrerenArgumenten
>WRITE "WRITE ","mit ","mehreren ","Argumenten"<cr>
WRITE mit mehreren Argumenten
>
```

Der WRITE-Befehl kennt besondere Argumente zur Formatierung der Ausgabe. Hier soll zunächst der Zeilenvorschub erwähnt werden. Er wird durch ein Ausrufezeichen (!) erzeugt. Besteht ein Argument des WRITE Befehls aus einem oder mehreren Ausrufezeichen, so werden genau so viele Zeilenvorschübe erzeugt, wie Ausrufezeichen vorhanden sind.

Zeilenvorschübe werden in MUMPS nie automatisch erzeugt, sondern müssen immer explizit durch Ausrufezeichen in WRITE-Befehlen angefordert werden.

Beispiel

```
>WRITE "Eine",!,"Ausgabe mit",!!,"Zeilenvorschueben"<cr>
Eine
Ausgabe mit

Zeilenvorschueben
>
```

In ähnlicher Form wie für den Zeilenvorschub werden auch die Steuerzeichen für den Seitenvorschub und für die Positionierung innerhalb einer Zeile angegeben. Ein Vorschub auf den Beginn der nächsten Seite wird mit dem Nummernzeichen (#) erzwungen.

Die Positionierung innerhalb einer Zeile wird durch das Fragezeichen (?) eingeleitet. Dem Fragezeichen folgt die Position innerhalb der laufenden Zeile, ab der geschrieben werden soll.

Die zur Formatierung dienenden Argumente des WRITE-Befehls werden in der gleichen Weise wie die sonstigen Argumente angegeben. Jedoch können mehrere Formatierungszeichen ohne trennende Kommata aneinandergekettet werden. Hierbei ist jedoch die Reihenfolge zu beachten, daß zuerst Seitenvorschubszeichen, dann Zeilenvorschubszeichen und dann Tabulierungszeichen erscheinen. Eine andere Reihenfolge wäre schon von der Logik her nicht sinnvoll.

Die folgenden Beispiele sollen die Anwendung verdeutlichen.

Beispiele

```
WRITE #,"Neue Seite"
WRITE ?40,"Positionierung innerhalb einer Zeile"
WRITE #!!!?20,"Ueberschrift",!
```

Beim letzten Beispiel wird die Aneinanderkettung der Formatierungsoperatoren deutlich.

Bei der Benutzung des Tabulierungsoperators ist es wichtig zu wissen, daß in MUMPS die erste Spalte die Nummer 0 hat. Daher bewirkt die Angabe von ?1 eine Positionierung in die zweite Spalte.

1.4.2 Variablen in MUMPS und der SET-Befehl

Jetzt soll ein weiteres wichtiges Element von (fast) allen Programmiersprachen behandelt werden: die Variablen.

Variablen stellen das „Gedächtnis“ der Programme dar. Es sind Speicher, die beliebige Werte, z.B. Ergebnisse von Berechnungen, aufnehmen und aufbewahren können. Jede dieser Variablen hat einen eindeutigen Namen, über den auf den Inhalt (d.h. Wert) zugegriffen werden kann. Dies geschieht einfach in der Weise, daß man den Namen der Variablen an die Stelle eines Ausdrucks oder eines Befehls schreibt, an der der Wert benötigt wird.

Die Namen von Variablen in MUMPS müssen entweder mit einem Buchstaben oder einem Prozentzeichen (%) beginnen. Die folgenden Zeichen müssen Buchstaben oder Ziffern sein. Buchstaben dürfen in Groß- oder Kleinschreibweise eingegeben werden. Beispiele richtiger Variablennamen sind:

```
ABC    abc    A123    A1b2c3    %Var    %1    %
```

Nicht erlaubt sind:

```
1AZ    123    Name-1    Var.X
```

Auch wenn die Länge von Namen nicht begrenzt ist, werden nur die ersten acht Zeichen zur eindeutigen Kennzeichnung herangezogen.

Obwohl der Name von Variablen in gemischter Groß- und Kleinschreibung erlaubt ist, ist dringend zu empfehlen, eine einheitliche Schreibweise von Anfang an zu wählen. In vielen (aber nicht in allen) MUMPS-Systemen wird zwischen Variablennamen, die sonst gleich sind, aber unterschiedliche Groß- und Kleinschreibung verwenden, unterschieden.

Daher stellen in solchen Systemen beispielsweise XX, Xx, xX und xx vier unterschiedliche Namen von Variablen dar.

Zur Verbesserung der Übersichtlichkeit von Programmen ist es sinnvoll, Befehle groß und die Namen von Variablen gemischt (bei langen, sprechenden Namen wie `KundenNummer`) oder klein zu schreiben.

Die Zuweisung eines neuen Wertes an eine Variable wird in MUMPS durch den Befehl `SET` erreicht.

Beispiel

```
>SET abc=5<cr>
>
```

In diesem Beispiel ist der Name der Variablen `abc` und ihr wird der Wert `5` zugewiesen. Das Gleichheitszeichen (=) zeigt die Zuweisung an. Es wird jeweils der Wert auf der rechten Seite des Gleichheitszeichens der Variablen auf seiner linken Seite zugewiesen.

Man beachte hierbei den Unterschied zwischen einer Zuweisung, die durch das Gleichheitszeichen angezeigt wird, und der mathematischen Aussage, daß zwei Werte einander gleich sind. So ist die Aussage `i=i+1` im mathematischen Sinne falsch, aber die Anweisung `SET i=i+1` stellt einen gültigen Befehl dar und bewirkt, daß der Wert der Variablen `i` um eins erhöht wird.

Wie das `WRITE` und fast alle anderen Befehle der Sprache erlaubt auch das `SET` die Angabe von mehreren Argumenten, die durch Kommata getrennt sind. Bei diesen Befehlen gilt:

```
<Befehlswort> <arg1>,<arg2>,...
```

ist äquivalent zu

```
<Befehlswort> <arg1> <Befehlswort> <arg2> ...
```

Beispiel

```
>SET abc=5,x="Hallo"<cr>
>WRITE "Kontrolle:",!,abc,!,x<cr>
Kontrolle:
5
Hallo
>
```

Im obigen Beispiel wird gleichzeitig deutlich, wie die Namen von Variablen benutzt werden, um ihren Inhalt auszugeben. Man beachte dabei, daß Textkonstanten (`"Hallo"`, `"Kontrolle:"`) in Anführungszeichen eingeschlossen sind, während Variablennamen (`abc`, `x`) direkt angegeben werden.

Bei der Zuweisung eines Wertes an mehrere Variablen kann man die Klammerform des Befehls benutzen. Sollen den Variablen `a`, `b`, `c` jeweils der Wert `1` zugeordnet werden, schreibt man abkürzend:

Beispiel

```
>SET (a,b,c)=1<cr>
>
```

Bei der Wertzuweisung durch einen SET-Befehl wird die Seite rechts des Gleichheitszeichens zunächst ausgewertet und dann das Ergebnis den auf der linken Seite angegebenen Variablen zugewiesen. Damit ist auch das Ergebnis von Zuweisungen, bei denen die gleiche Variable rechts und links des Gleichheitszeichens auftritt, wohl definiert:

Beispiel

```
>SET i=5<cr>
>SET i=i+1<cr>
>WRITE i<cr>
6
>
```

Die mit dem SET-Befehl definierten Variablen werden im Arbeitsspeicher abgelegt und können innerhalb einer Sitzung beliebig benutzt und verändert werden. Für andere Benutzer des gleichen Rechners sind diese Variablen nicht sichtbar, können also weder gelesen noch geändert oder gelöscht werden. Die so definierten Variablen sind also jedem Benutzer (genauer: jedem Prozeß) lokal zugeordnet und heißen daher „lokale Variablen".

Die definierten lokalen Variablen können in allen MUMPS-Systemen durch einen einfachen Befehl angezeigt oder ausgedruckt werden. In vielen Fällen wird dazu der WRITE-Befehl ohne Argument verwendet, der in dieser Form allerdings nicht standardisiert ist.

Beispiel

```
>WRITE                  ; WRITE-Befehl ohne Argument<cr>
  a           1
  abc         5
  b           1
  c           1
  i           6
  x           "Hallo"
>
```

Hier werden die in den vorangegangenen Beispielen gesetzten Variablen angezeigt.

Eine Bemerkung wert ist noch die alphabetische Sortierung der Ausgabe, die insbesondere bei großen Variablenlisten die Übersichtlichkeit wesentlich erhöht.

Gerade bei der Programmentwicklung ist die Möglichkeit, jederzeit die aktuellen Werte von Variablen einfach anzeigen lassen zu können, sehr hilfreich. Fehler durch zum Beispiel falsche Zuweisung an Variablen können damit leicht entdeckt werden.

1.4.3 Löschen von Variablen mit dem KILL-Befehl

Das natürliche Gegenstück des SET-Befehls ist der KILL-Befehl. Durch ihn wird man in die Lage versetzt, aus der Gruppe der definierten lokalen Variablen einzelne (oder alle) zu entfernen. In diesem Abschnitt sollen einige Aspekte dieses Befehls behandelt werden. Weitere Details werden im Zusammenhang der indizierten Variablen beschrieben.

Das Argument des Befehls ist normalerweise der Name einer Variablen, die man löschen möchte. Es gibt den Befehl in drei verschiedenen Formen.

`KILL a` bedeutet, daß die Variable `a` aus der Liste der definierten Variablen entfernt wird. War diese Variable nicht definiert, hat der KILL-Befehl keine weiteren Auswirkungen. Entsprechend löscht man mit dem Befehl `KILL a,b,c` die Variablen `a`, `b` und `c`.

KILL ohne Argument bedeutet, daß alle lokalen Variablen ohne Ausnahme gelöscht werden.

In der dritten Form des KILL-Befehls werden Klammern benutzt, um mitzuteilen, welche Variablen nicht gelöscht werden sollen. `KILL (k,l,def)` heißt also: lösche alle sichtbaren Variablen (siehe dazu auch Abschnitt 2.5) bis auf `k`, `l`, `def`.

Anhand der argumentlosen Form des KILL-Befehls soll auf die besondere Schreibweise von argumentlosen Befehlenin MUMPS hingewiesen werden:

dem Befehl müssen zwei Leerzeichen folgen, bevor ein weiterer Befehl auf der gleichen Zeile geschrieben werden kann.

Beispiel

```
KILL␣␣SET a=1
```

In diesem Beispiel sind die beiden Leerzeichen durch ␣␣ dargestellt worden.

Der Vorteil des KILL-Befehls wird besonders einsichtig, wenn man sich vor Augen hält, daß in jedem MUMPS-System der Speicherplatz, der zur Abspeicherung von Variablen zur Verfügung steht, begrenzt ist. Durch das Löschen einer Variablen wird der von ihr belegte Speicherplatz wieder frei. Es empfiehlt sich, Variablen dann zu löschen, wenn sie in der weiteren Programmausführung nicht mehr benötigt werden. Im übrigen zeugt es von einem guten Programmierstil, wenn alle Variablen, die in einem Programm oder Unterprogramm für interne Zwecke benutzt werden, an dessen Ende wieder gelöscht werden.

Wie aber kann man feststellen, wieviel Speicherplatz man noch im Arbeitsspeicher zur Verfügung hat? Ehe diese Frage beantwortet werden wird, soll noch einmal auf den Aufbau einer Partition eingegangen werden.

Die beiden Hauptobjekte einer Partition sind die definierten lokalen Variablen sowie das geladene MUMPS-Programm. Die Größe einer Partition kann in vielen Fällen vom Systemverwalter eingestellt werden. In Abhängigkeit von der Größe des geladenen Programms und der Anzahl der definierten lokalen Variablen kann der aktuell zur Verfügung stehende freie Platz in der Partition mit einer sogenannten Systemvariablen angezeigt werden.

Systemvariablen sind spezielle Sprachelemente in MUMPS, die Auskunft über gewisse systemnahe Parameter geben. In Anlehnung an die englische Bezeichnung werden die Systemvariablen auch „Spezielle Variablen" genannt. Jede Systemvariable (von denen es in MUMPS sieben gibt), hat einen eindeutigen Namen, der stets mit einem Dollarzeichen ($) beginnt. Man kann eine Systemvariable wie eine normale Variable verwenden, ihr aber (bis auf Ausnahmen) keinen Wert zuweisen.

Als Beispiel und zur Beantwortung der oben gestellten Frage soll die Systemvariable **$STORAGE** eingeführt werden. Sie gibt die Anzahl der noch freien Zeichen in einer Partition an.

Der Wert von **$STORAGE** ist natürlich in hohem Masse abhängig vom verwendeten MUMPS-System. Er ist außerdem abhängig von der aktuellen Programmgröße und von der Größe des Speicherplatzes, der für zusätzliche Verwaltungsinformation benötigt wird.