

RUHR - UNIVERSITÄT BOCHUM

Arbeitsberichte

des

Rechenzentrums

Direktor: Prof. Dr. H. Ehlich

Nr. 7303

BO.CO.09

BODAT, ein schnelles und platzsparendes
System zur Datenmanipulation und -spei-
cherung in Algol 60 und Fortran

von

Rainerd Buchmann

Bochum, Juni 1973
Universitätsstraße 150, Gebäude NA

Inhaltsverzeichnis

	<u>Seite</u>
<u>O. Einleitung</u>	4
<u>I. Textorientierte E/A</u>	8
A. Grundsätzliches	10
B. Einzelbeschreibungen der Rahmenpro- zeduren (2. Stufe)	13
1. EIN Eingabe	13
2. AUS Ausgabe	15
3. AUSEIN spezielle Konsolprozeduren	17
4. EAFRAG Situationsabfragen	17
5. EAZAHL Kenngrößenabfrage	19
6. EASTEU Steuerung	21
7. Vergleiche mit Standard-E/A	26
C. Übersicht über den zusätzlichen ALGOL- RAHMEN (3. Stufe)	27
D. Einzelbeschreibungen der ALGOL-PROZEDU- REN für die 3. Stufe (formatierte E/A)	29
1. READ Lesen	30
Formatprozeduren zur Eingabesteuerung	
Beispiele	34
2. TRIM Hilfsprozedur zum Lesen	
3. WRITE	36
PRINT	
TYPE Schreiben	
LINE	
Formatprozeduren zur Ausgabesteuerung	38
4. STRINGHANDLING mit E/A-Prozeduren (mit Testmöglichkeiten)	41
5. Ver leiche mit Standard-Algol-E/A	42

Inhaltsverzeichnis (Fortsetzung)

Seite

II. Datenspeicherung in Gebieten und Physdateien

1. Einleitung	43	
2. Übersicht	45	
3. Funktionsweise	47	
4. Einzelbeschreibungen der Prozeduren	50	
A. EXTARR	Anfangsbehandlung	50
B. GEBHG	Speicherkreation im Gebiet	52
C. PHYSHG	Speicherkreation in Physdatei	59
D. AGET	Bringen	61
E. IAGET	Integer-Element bringen	61
F. APUT	Speichern	61
G. AGETS	Bringen sequentiell	61
H. IAGETS	Integer: Bringen sequentiell	61
I. APUTS	Speichern sequentiell	61
J. HGINF	Informieren über Speicher	64
K. ENDBEH	Endebehandlung	64
L. STRANS	schneller Massentransport	66
M. Hilfsprozeduren und Common-Zonen		67

Literaturverzeichnis

Für viele Anwendungen, bei denen der Algol- oder Fortran-Programmierer Alphatext bearbeiten muß, erweist sich die Standard-E/A für die jeweilige Sprache als unzulänglich, und zwar aus mehreren Gründen:

Auf der einen Seite bietet sie mit ziemlich variablen Formatmöglichkeiten einen Komfort, der manchmal unerwünscht ist, weil er bei reinen Textmanipulationen überflüssig ist und deswegen unnötigerweise die E/A-Geschwindigkeit herab- und den Kernspeicherbedarf heraufsetzt. Auf der anderen Seite kann die Standard-E/A, da für eine problemorientierte Sprache und damit maschinenunabhängig konzipiert, eine Vielzahl der speziellen Hardware- und Software-Möglichkeiten gerade des TR 440 nicht nutzen, muß also auf eine Reihe von sehr nützlichen Leistungen des Systems verzichten.

An speziellen Schwächen der E/A am TR 440 läßt sich im wesentlichen folgendes aufführen:

- 1.) In Algol 60 läßt sich Text nur sehr umständlich bearbeiten, da
 - a) bei INSYMBOL/OUTSYMBOL keine Zeilenwechsel erkennbar sind
 - b) im A-Format eingelesener Text nur schwer weiterverarbeitbar ist (Zeilenwechsel sind dabei auch nicht erkennbar)

- c) ganze Strings völlig unhandlich beim Einlesen sind (sie müssen immer von String-quotes eingeschlossen sein bei der Prozedur READ).
- d) die Prozeduren IN/OUT nur auf mühsamen Umwegen einen direkten Zugriff auf eine Datei erlauben
- e) viel Kernspeicherplatz benötigt wird, da das E/A-Paket wenig bzw. kaum segmentiert ist
- f) alle E/A-Prozeduren ausgesprochen langsam sind, insbesondere auch die Prozeduren zur formatfreien binären E/A auf Dateien.

2.) In Fortran treten ähnliche Mängel zutage:

- a) Das A-Format ist ebenfalls nur schlecht weiterverarbeitbar (Zeilenwechsel nicht erkennbar).
- b) Im String-format variable Satztlängen einzulesen ist nur auf hinterlistigen Umwegen erreichbar.
- c) Ein zeichenweises Einlesen mit Zeilenwechseln etc. ist nicht einfach durchführbar.
- d) Die binäre, formatfreie Datei-E/A ist auch hier extrem langsam durch mehrfaches Umpuffern.

Insgesamt läßt sich sagen, daß die Möglichkeiten, die die Sprachbezogene, maschinenunabhängige E/A für reine Textverarbeitung, binäre Zwischenspeicherung u.ä. bei vielen Aufgaben unrationell bzw. manchmal sogar unbrauchbar ist.

Zum Schließen all dieser Lücken dient ein Paket von Unterprogrammen, das von mir speziell konzipiert wurde, um

- 1.) extreme Geschwindigkeit bei allen E/A-Vorgängen zu erzielen
- 2.) alle Leistungen, die das TR 440-System bietet, und die normalerweise nur auf Maschinensprachen-ebene verfügbar sind, an die problemorientierten Sprachen Algol und Fortran weiterzureichen
- 3.) alle Möglichkeiten der Datenspeicherung mit ihren vielfältigen Vorteilen auszunutzen, also Bearbeitung von RAN-, RAM- und PHYS-dateien, von Gebieten und auch von Gemeinschaftsgebieten zur Kommunikation zwischen verschiedenen Jobs.

Der Komplex I beschreibt eine Reihe von Unterprogrammen, mit deren Hilfe es möglich ist, in sehr variabler und bequem steuerbarer Weise die Medien RAN-, RAM-Datei, Konsole und Fremdstring zu manipulieren.

Der Komplex II erläutert das Konzept der externen Arrays, das es ermöglicht, mit maximaler Geschwindigkeit und geringstem Kernspeicheraufwand große Datenmengen zwischenspeichern etc., indem die internen Speicherformen in PHYS-Dateien und Gebieten benutzt werden.

Dabei kam es mir bei der Programmierung insbesondere darauf an, möglichst alle Leistungen der entsprechenden

SSR-Befehle an die problemorientierte Ebene in Algol und Fortran durchzureichen und es dem Programmierer selbst zu überlassen, wieviel des möglichen Komforts, den die Unterprogramme bieten können, er in Anspruch nehmen und wieviel er selbst organisieren möchte. Daher sind alle Prozeduren auf verschieden hoch organisierten Stufen ansprechbar.

I.

Das Konzept der Text-orientierten E/A

Die unterste Ebene dieser E/A ist in 7 Teile segmentiert mit ca. 40 Basisprozeduren. Diese Basisprozeduren sind auf extreme Geschwindigkeit und möglichst wenig Komfort hin angelegt, erlauben aber dennoch, alle Leistungen des Programmiersystems zu benutzen. Sie sind im Grunde in 4 Blöcken angeordnet:

- 1.) Protokolldienste auf dem Schnelldrucker und der Konsole
- 2.) Eingabe von den Standardmedien Konsole und Fremdstring
- 3.) Manipulation von Texthaltungsdateien (Kapiteln) vom Organisationstyp RAN oder RAM
- 4.) Binäre Bearbeitung von beliebigen RAN- und RAM-Dateien.

Sie werden zusätzlich über eine Reihe von Common-Blöcken gesteuert.

Da sie auf jeden Anwendungskomfort bewußt verzichten und so vielfältige Möglichkeiten bieten, sind sie nicht einfach zu benutzen. Um dem Programmierer die Leistungen auf bequemere Weise zugänglich zu machen und damit auch einen höheren Dokumentationswert seiner Programme zu erzielen, habe ich zu den Basisprozeduren als zweite Ebene einen

globalen Rahmen geschrieben mit 6 Prozeduren, zu deren Steuerung kein Common-Block mehr benötigt wird, und die dennoch alle Möglichkeiten der Basisprozeduren enthalten (zur Vereinfachung noch einige mehr). Da die bequeme Handhabung meiner Meinung nach den geringen Zeitverlust durch die zusätzliche Organisation mehr als aufwiegt, möchte ich mich in diesem Bericht hauptsächlich auf eine detaillierte Beschreibung der Rahmenprozeduren beschränken. Eine ausführliche Beschreibung der Basisprozeduren wird in Form eines Nachtrages noch geliefert werden.

Diese 6 Prozeduren sind ebenfalls wie die Basisprozeduren nicht-rekursiv gebaut, um möglichst schnell sein zu können.

Das hat zur Folge, daß 1.) die Funktionsprozeduren auch nicht auf Parameterposition der eigentlichen Prozeduren stehen dürfen und 2.) in Algol keine formalen Prozedurparameter als Parameter an die E/A-Prozeduren durchgereicht werden dürfen.

Ist dies doch einmal nötig, so muß man sich der Prozedur AP bedienen (siehe [1] BOGOL-STRING).

Für den Algol-Benutzer dieser E/A steht dann noch eine dritte Ebene von Prozeduren zur Verfügung, die ihm alle Bequemlichkeiten einer formatierten und unformatierten E/A bietet, und zwar komfortabler und leichter zu handhaben als die sonstige Algol-EA.

Zur eigentlichen Ein- und Ausgabe dienen die Prozeduren

EIN

AUS und als Spezialprozedur

AUSEIN

Gesteuert wird die gesamte EA durch die Prozedur

EASTEU.

Zum Abfragen bestimmter Situationen dient die logische Funktionsprozedur

EAFRAG,

zum Abfragen gewisser Kenngrößen die Integer-Funktionsprozedur

EAZAHL.

Zusätzlich können mit der EA für Algol die an SNOBOL angelehnten Prozeduren TRIM, SUCC und FAIL kombiniert werden, die im Rechenzentrumsbereich BOGOL-STRING [1] beschrieben sind.

Um die Arbeitsweise einiger Prozeduren verstehen zu können, muß man noch folgendes wissen:

Zu einem aktuellen Zeitpunkt kennt dieses E/A-System immer eine sogenannte aktuelle Eingabedatei (bzw. eine aktuelle Ausgabedatei). Damit ist jeweils die Datei gemeint, aus der zuletzt gelesen wurde (bzw. in die zuletzt geschrieben wurde). Eine Steueranweisung oder eine Abfrage (Pro-

zedur EASTEU oder EAFRAG/EAZAHL) bezieht sich, wenn sie eine Datei anspricht, immer auf die jeweilige aktuelle Eingabe- bzw. Ausgabedatei, wenn nicht ausdrücklich das Ansprechen einer bestimmten symbolischen Gerätenummer ermöglicht wird.

Die Ein- und Ausgabe erfolgt grundsätzlich unformatted, und zwar mit jedem Aufruf wird immer genau ein Satz gelesen oder geschrieben, unabhängig vom Medium.

In dem E/A-Paket werden drei unterschiedliche Formen der Informationsdarstellung beim Lesen und Schreiben akzeptiert.

- 1.) Die erste ist die rein binäre Darstellung, wie sie in der Algol- und Fortran-E/A als umformatierte Datei-E/A möglich ist (in Algol z.B. durch die Prozedur PUT, GET etc.)
- 2.) Die zweite Art ist die der Strings. In Algol sind dies normale Strings, zusätzlich werden aber auch die String-formen akzeptiert, die durch string procedures etc. definiert werden, wie sie im Rechenzentrumsumsbericht BOGOL-STRING [1] beschrieben werden. In Fortran sind es die Strings im Sinne des String-handlings, die also in einem Element eines Feldes beginnen, und deren Ende intern verwaltet wird. Allerdings mit einer Abweichung: Beim Lesen beginnt der String nicht im angegebenen Feldelement (dort steht die Zeichenzahl!), sondern in dem dahinter.

3.) Die dritte Art ist eine spezielle, die in Algol durch die Prozedur IN/OUT bearbeitet wird, und die in Fortran nicht existiert. Sie wird im folgenden mit umgewandelt bezeichnet. Dabei steht in jedem Ganzwort (z.B. Feldelement) genau ein Zeichen, und zwar repräsentiert durch seinen Zentralcode-Wert. Das Zeichen "BLANK" wird z.B. durch die Zahl 175 dargestellt, das Fluchtsymbol "Kissen": ☒) durch die Zahl 124 etc. Die Felder, die zur Aufnahme dieser Art der Informationsdarstellung benötigt werden, müssen in Algol als integer array's deklariert werden, in Fortran als INTEGER*4-Felder.

Für die Eingabe muß speziell gesteuert werden, wenn die "umgewandelte" Form gewünscht wird, (Prozedur EASTEU), bei der Ausgabe wird nach dem internen Aufbau (Typenkennung) unterschieden. Ein solcher "umgewandelter" Text wird immer beendet durch eine Zahl <64, die dann als Vorschubsteuerzeichen ausgewertet wird.

Nun zur Beschreibung der Prozeduren:

1.) Die Prozedur EIN (INF ,SGNR ,SATZNR)

INF gibt an, wohin der gelesene Text gespeichert werden soll. Dabei kann INF ein Feldname oder ein Feldelement sein, in das als erstes Text gelesen wird.

Zusätzlich kann es aus Kompaktibilitätsgründen zu BOGOL-STRING auch ein Aufruf der Funktionsprozedur VAL sein.

Ist INF der einzige Parameter der Prozedur EIN, so wird vom Standard-eingabemedium gelesen. Das ist der Fremdstring, wenn er im Starte-Kommando unter DATEN=... angegeben ist (auch im Gespräch!), oder eine Datenanforderung von der Konsole im Dialog, wenn kein Fremdstring vorhanden ist.

Ist im Abschnitt kein Fremdstring vorhanden, so wird bei dieser Anfrageart mit 1 Parameter immer "leere Eingabe" übergeben.

SGNR Ist dieser Parameter vorhanden, so gibt er die symbolische Gerätenummer des Eingabemediums an. Diese wird genauso gehandhabt wie in der Standard -E/A, also 5 für Fremdstring, 8 für Konsol-eingabe, wobei DNUMMER-Angaben des Starte-Kommandos berücksichtigt werden. Sind nur die beiden ersten Parameter vorhanden, so wird aus Dateien sequentiell Satz für Satz gelesen.

SATZNR Ist dieser Parameter vorhanden, so gibt er bei Dateien vom Typ RAN oder RAM die Nummer des Satzes an, der gelesen werden soll.

Ein nachfolgender Aufruf von EIN ohne den Satznummer-Parameter bewirkt dann ein Lesen des nächsten Satzes hinter diesem im direkten Zugriff angesprochenen. Beim sequentiellen Lesen ist darauf zu achten, daß bei jedem Wechsel Lesen - Schreiben und umgekehrt wieder auf den ersten Satz der Datei eingestellt wird. Bei Eingabe aus Fremdstring oder von Konsole wird SATZNR ignoriert.

2.) Die Prozedur AUS (INF ,SGNR ,SATZNR)

INF gibt an, wo die auszugebende Information steht. Zusätzlich zu den Möglichkeiten der Parameterart wie bei der Prozedur EIN kann man hier einen Aufruf einer string procedure in Algol verwenden (siehe BOGOL-STRING [1]).

Außerdem bewirkt eine Integerzahl N ($16 \leq N \leq 23$) auf Parameterposition die Ausgabe einer Leerzeile mit N als Vorschubsteuerzeichen.

Ist INF der einzige Parameter, so wird das Standardausgabemedium genommen, das im Abschnitt der Schnelldrucker, im Gespräch die Konsole ist (bei eingeschaltetem Druckerprotokoll im Gespräch Ausdrucken auf Konsole und Schnelldrucker).

SGNR gibt, falls vorhanden, die symbolische Geräte- nummer des Ausgabemediums an, wiederum standard- mäßig

SGNR=9 auf Konsole, bei 6 auf dem Schnelldrucker (DNUMMER-Angaben werden berücksichtigt)

SATZNR gib' wiederum die Nummer des Satzes an, der geschrieben werden soll. Fehlt dieser Parameter, so wird ein sequentielles Schreiben in die RAM- oder RAN- datei simuliert, wobei steuerbar ist, mit welcher Schrittweite fortgeschrieben wird (Prozedur

EASTEU). Auch hierbei können Aufrufe mit und ohne Satznummerangabe gemischt werden.

- 3.) Die Prozedur AUSEIN (INFAUS, INFEIN) bewirkt ein Ausdrucken von INFAUS (Parametertyp siehe INF bei Prozedur AUS) auf der Konsole und sofortige Eingabeanforderung, die auf INFEIN abgelegt wird (Parametertyp siehe INF bei Prozedur EIN).

Im Abschnitt wirkt die Prozedur genau wie die Prozedur AUS mit einem Parameter, wobei zusätzlich "leere Eingabe" in INFEIN übergeben wird.

Eine leere Eingabe bzw. eine Leerzeile werden bei STRING-Eingabe als Leerstring, bei umgewandelter Eingabe als ein einzelnes Zeichen mit dem Wert 21 (bei Leerzeile) ("neue Zeile") übergeben (bzw. mit dem Wert 33 bei leerer Eingabe).

- 4.) Die Prozedur EAFRAG (NR) ist eine logische Funktionsprozedur, deren geforderte Leistung durch den Parameter NR gesteuert wird. ($1 \leq NR \leq 10$). Abhängig von NR liefert EAFRAG als Funktionswert:

NR	Funktionswert = <u>true</u> , wenn
1	Dateieingabeende
2	Konsoleingabeende
3	Fremdstringende
4	aktuelle Eingabedatei leer
5	Konsoleingabe leer
6	Fremdstring leer
7	vorrangige Kommandos in der Konsoleingabe waren
8	der letzte E/A-Befehl erfolgreich durchgeführt werden konnte
9	
10	noch nicht belegt

NR = 8 ist dabei identisch mit einem Aufruf der BOGOL-
STRING-Prozedur SUCC. (siehe [1]).

Bei NR = 4 kann zusätzlich noch ein zweiter Parameter
SGNR angegeben werden, und die Prozedur EAFRAG liefert
dann als logischen Funktionswert, ob die durch SGNR
identifizierte Datei leer ist.

5.) Die Prozedur EZAHL (NR) ist eine Integer-Funktionsprozedur, deren geforderte Leistung durch den Parameter NR gesteuert wird ($1 \leq NR \leq 10$).

Abhängig von NR liefert EZAHL als Funktionswert:

NR	Funktionswert (Integer-Zahl)
1	Nummer des letzten aus einer Datei gelesenen Satzes
2	Nummer des letzten in eine Datei geschriebenen Satzes
3	die symbolische Gerätenummer SGNR der aktuellen Eingabedatei
4	die symbolische Gerätenummer SGNR der aktuellen Ausgabedatei
5	Die Anzahl gelesener Ganzworte bei binärer Eingabe aus einer Datei Die Anzahl der gelesenen Zeichen sonst (ohne das Abschlußzeichen < 64 bei umgewandelter Eingabe)
6	Die niedrigste belegte Satzposition der Datei SGNR
7	Die höchste belegte Satzposition der Datei SGNR
8	Die Satzzahl der Datei SGNR
9	0, wenn kein SSR-Fehler bei dem letzten E/A-Aufruf aufgetreten ist, sonst den Fehlerschlüssel, den der SSR-Befehl übermittelte (siehe [3])
10	noch nicht belegt

Bei NR = 6,7 oder 8 ist also die symbolische Geräte-
nummer SGNR der angesprochenen Datei als zweiter Para-
meter anzugeben, bei allen anderen nicht.

Bei NR = 8 gilt für den Funktionswert Satzzahl zusätzlich:

Satzzahl = 0: Datei ist leer

-1: die symbolische Gerätenummer
SGNR ist durch eine Angabe im
Startekommando DNUMMER = n SGNR
nicht benutzbar

-2: Datei existiert nicht

-3: Es ist keine Zuordnung

DATEI = SGNR-Dateiname im Starte-
kommando angegeben (unter Berück-
sichtigung von DNUMMER).

6.) Die Prozedur EASTEU (NR ,GROESSE 1 , GROESSE 2)

Mit Hilfe dieser Prozedur werden sämtliche Steuerungen für die E/A-Vorgänge durchgeführt.

Der Parameter NR gibt wiederum an, welche Leistung erbracht werden soll ($-25 \leq NR \leq 25$, $NR \neq 0$).

GROESSE 1 und GROESSE 2 müssen nur angegeben werden, wenn sie in der Einzelbeschreibung aufgeführt werden, sonst können sie auch fortgelassen werden.

Abhängig vom Parameter NR können folgende Leistungen gefordert werden (ein Stern* vor dem Wert von NR bezeichnet die standardmäßig voreingestellte Leistung):

NR	zugehörige erbrachte Leistung
1	Die Eingaben sollen umgewandelt werden
* -1	Die Eingaben sollen nicht umgewandelt werden
2	Bei Lesen aus Dateien soll binär gelesen werden
* -2	Bei Lesen aus Dateien soll nicht mehr binär, sondern zeichenweise gelesen werden (maximal 160 Zeichen je Satz).
3	Datei-ausgaben sollen binär erfolgen
* -3	Datei-ausgaben sollen zeichenweise erfolgen (maximal 160 Zeichen je Satz)
4	Dateieingaben sollen mit Satznummer protokolliert werden (Eingabeprotokoll einschalten)
* -4	Eingabeprotokoll ausschalten

NR	zugehörige erbrachte Leistung
5	Dateiausgaben sollen mit Satznummer protokolliert werden (Ausgabeprotokoll einschalten)
* -5	Ausgabeprotokoll ausschalten
6	Datei-protokolldienste sollen auf dem Schnelldrucker ausgegeben werden (SSR 6 12)
* -6	Datei-protokolldienste sollen auf Konsole geleitet werden, bei eingeschaltetem Druckerprotokoll zusätzlich auf dem Drucker
	Bemerkung: die Dateiprotokolldienste sind nur wirksam bei zeichenweiser E/A, also nicht bei binärer E/A.
* 7	vorrangige Kommandos in einer Konsoleingabe sollen ausgeführt werden
-7	Kommandos in einer Konsoleingabe sollen nicht ausgeführt, sondern als Text gelesen werden (Fluchtsymbol = Zeichen 124)
* 8	es werden Texthaltungsdateien bearbeitet (mit Oktadenzähler im letzten Ganzwort)
-8	ein Oktadenzähler soll nicht hinzugefügt werden. (relevant nur für die Ausgabe)
9	Vorschubsteuerzeichen bei Konsoleingabe: = GROESSE 1 voreingestellt: 21 = "neue Zeile"
-9	Vorschubsteuerzeichen bei Konsolausgabe: = GROESSE 1 voreingestellt: 21 = "neue Zeile"
10	Alle Dateiausgaben werden nur auf dem Schnelldrucker simuliert (von den Protokolldiensten, die dabei implizit eingeschaltet werden)
* -10	echte Dateiausgabe

NR	zugehörige erbrachte Leistung
* 11	Vorschubsteuerzeichen der Datei-Protokolldienste: = GROESSE 1 voreingestellt 21 = einzeilige Protokollierung
-11	Die Ausgaben der Datei-Protokolldienste sollen schmalseitig erfolgen (auf der Konsole sind sie sowieso implizit schmalseitig)
12	Im Falle eines SSR-Fehlers (z.B. wenn bei Ausgabe die Datei zu klein ist etc.), wird auf das Label GROESSE 1 gesprungen (in Algol Integer-Label!)
* -12	Nach SSR-Fehler wird fortgefahren hinter dem Aufruf
13	der nächste sequentiell zu lesende Satz soll der mit der Satznummer GROESSE 1 sein (vorpositionieren)
-13	der nächste sequentiell zu schreibende Satz soll der mit der Satznummer GROESSE 1 sein (vorpositionieren)
14	Anzahl der Achtelseiten für Dateitransporte: = GROESSE 1 ($1 \leq \text{GROESSE } 1 \leq 32$)
-14	noch nicht belegt
15	Der Rest der Konsoleingabe soll überlesen werden
--15	Der Fremdstring soll auf den Anfang positioniert werden (um ihn noch einmal zu lesen)
16	maximal zu übertragende Satzlänge in Ganzworten bei binärer Eingabe: = GROESSE 1 (Der Rest wird ignoriert). voreingestellt: 30
-16	Satzlänge für den nächsten Schreibbefehl bei binärer Dateiausgabe in Anzahl Ganzworten: = GROESSE 1

NR	zugehörige erbrachte Leistung
17	Fortschaltungsgröße bei sequentieller Ausgabe := GROESSE 1. Das heißt, wurde zuletzt der Satz mit der Nummer N geschrieben, so werden als die nächsten die Sätze mit den Nummern $N + I * GROESSE 1$ ($I = 1, 2, \dots$) geschrieben, falls der Parameter SATZNR beim Aufruf der Prozedur AUS fehlt. voreingestellt: 10
-17	noch nicht belegt
18	Die Datei mit der symbolischen Geräte- nummer = GROESSE 1 soll von der Verar- beitung abgemeldet werden.
-18	noch nicht belegt
* 19	O - Datei-Ausgabe
-19	A - Datei-Ausgabe
	} ist nur relevant bei umzuwandelnder Aus- gabe - im Falle der A-Datei wird das er- ste Zeichen als Vor- schubsteuerzeichen und nicht als Satzende-Zeichen ausgewertet
20	Die Sätze mit den Nummern GROESSE 1 bis GROESSE 2 der aktuellen Ausgabedatei sollen gelöscht werden. Dabei muß gel- ten $GROESSE 1 \leq GROESSE 2$
-20	noch nicht belegt
21	Die Datei mit der symbolischen Geräte- nummer = GROESSE 1 soll zur aktuellen Ausgabedatei gemacht werden
-21	Die durch GROESSE 1 identifizierte Datei soll zur aktuellen Eingabedatei gemacht werden
* 22	Bei vorrangigen Kommandos, die ausgeführt wurden, soll danach die Anfrage wiederholt werden (relevant für Konsoleingabe)
-22	Nach vorrangigen Kommandos, die ausgeführt wurden wird die Anfrage nicht wiederholt, sondern "leere Eingabe" übergeben

NR	zugehörige erbrachte Leistung
23	Bei umzuwandelnder Konsoleingabe wird nicht nur jeweils ein Satz übertragen, sondern mit einem Aufruf der Prozeduren EIN bzw. AUSEIN wird die gesamte Eingabe übergeben. Das Ende der Eingabe ist durch das Zeichen mit dem Wert 33 kenntlich. Es ist darauf zu achten, daß das Eingabefeld lang genug ist.
* -23	Auch von der Konsoleingabe wird jeweils genau eine Zeile übertragen
24	noch nicht belegt
-24	
25	
-25	

Um einen groben Überblick über den Zeitbedarf dieser E/A-Unterprogramme zu geben, sollen noch folgende Beispiele gegeben werden:

- 1.) Ein Zeitvergleich der umzuwandelnden Eingabe aus einer Datei und wieder Ausgabe mit den Algol-prozeduren IN/OUT ergibt ein Rechenzeitverhältnis von ca 1:7 zugunsten EIN/AUS.
- 2.) Will man aus einem Fremdstring lesen, die Eingabe umwandeln, das Gelesene in eine Datei schreiben und protokollieren, was einem TEINTRAGE-Kommando mit Protokoll = -STD- und Numerierung = -STD- entspricht, so ergibt sich gegenüber TEINTRAGE ein Rechenzeitverhältnis von 1:1,5 zugunsten von EIN/AUS.
- 3.) Bei binärer E/A aus bzw. in Dateien ergibt sich gegenüber der binären Algol-E/A (GET/PUT) und der binären (formatfreien) Fortran-E/A ein Rechenzeitverhältnis von 1:20 zugunsten EIN/AUS.

Der Kernspeicherbedarf aller Prozeduren zusammen liegt bei der Benutzung der Rahmenprozeduren bei ca. 2,5 K Ganzworten gegenüber z.B. minimal 6 K für die Standard-Algol-E/A.

Die dritte Ebene, auf der die E/A für den Algol-Programmierer ansprechbar ist, ist speziell für String-Manipulationen mit Hilfe von BOGOL-STRING-Prozeduren [1] konzipiert und für beliebig variable formatierte und unformatierte EIN/Ausgabe.

Sie verarbeitet nicht die Speicherform ein ZC-Zeichen pro Ganzwort und auch nicht die Binärform.

Sie besteht aus folgenden eigentlichen Prozeduren:

- 1.) READ zum Lesen
- 2.) WRITE/PRINT/TYPE/LINE zum Schreiben
- 3.) READSGNR/WRITESGNR zum Einstellen der Geräte-
nummer
- 4.) READFEHL/WRITEFEHL zur Bekanntmachung von
Fehlermarken
- 5.) READEND zur Bekanntmachung eines Ende-Labels.

Dazu kommen noch folgende Format- und Steuerprozeduren, die als integer-procedure zu deklarieren sind und alle genau einen Parameter haben:

- | | | |
|------|-----------|---|
| 6.) | S ≡ SFORM | String-Format |
| 7.) | Z ≡ ZFORM | Zahlen-Format |
| 8.) | B ≡ BFORM | Blank-Format |
| 9.) | V ≡ VFORM | Vorschub-Format |
| 10.) | E ≡ EFORM | Ersetzungs-Format |
| 11.) | P ≡ PFORM | Positionierungs-Format |
| 12.) | SGNR | Einstellen der symbolischen
Gerätenummer |

- | | | |
|------|--------|---------------------------|
| 13.) | SATZNR | Einstellen der Satznummer |
| 14.) | END | Ende-Label |
| 15.) | ERR | Error-Label |

Da diese Funktionsprozeduren über Seiteneffekte wirken, müssen sie auf Parameterposition von READ oder WRITE/PRINT/TYPE stehen.

Ein Teil dieser Prozeduren ist selbst in Algol geschrieben unter Verwendung von BOGOL-STRING-Prozeduren (siehe [1]). Die Prozeduren READ, WRITE, PRINT, TYPE leisten alles, was die gleichnamigen ALCOR-Prozeduren leisten, darüber hinaus aber wesentlich mehr. Die Namen wurden so gewählt, um dem Algol-Programmierer einen leichten Übergang von Standard-E/A zu dieser E/A zu bieten, da er Programme, die die ALCOR-Prozeduren benutzen, ohne Änderung auf diese E/A umstellen kann, indem er nur die Prozeduren zu Programmbeginn mit dem Wortsymbol code; als Prozedurrumpf deklariert.

Die Leistungen der Prozeduren im einzelnen sind folgende:

1.) READ:

Mit dieser Prozedur können von einem beliebigen Medium bis zu 255 Zahlen und Strings pro Aufruf eingelesen werden.

a) formatfrei lesen:

Bis zum ersten Auftreten einer der Prozeduren S, SFORM, Z, ZFORM wird in jedem Aufruf von READ formatfrei gelesen, und zwar nach folgendem Schema:

Ist der Parameter eine einfache oder indizierte Variable, so wird eine Zahl gelesen bis zum nächsten Semikolon oder bis zum Satzende - die Syntax für interpretierbare Zahlen ist dabei:

$$\langle \text{Zeichen} \rangle ::= \text{Ziffer}$$

$$\langle \text{Zahl} \rangle ::= \{ \text{Ziffer} \}^{\infty} [\pm] [\langle \text{Zeichen} \rangle^{\infty}] [\langle \text{Punkt} \rangle] [\langle \text{Zeichen} \rangle^{\infty}] [\langle \text{Exponent} \rangle]$$

$$\langle \text{Punkt} \rangle ::= \text{.}$$

$$\langle \text{Exponent} \rangle ::= \langle \text{Exp} \rangle \{ \text{Ziffer} \}^{\infty} [\pm] [\langle \text{Zeichen} \rangle^{\infty}]$$

$$\langle \text{Exp} \rangle ::= E | D | \text{10}$$

Beispiele für richtige Zahlen:

$$10. = 10$$

$$0,3\text{-}1\text{-}4\text{-}E = 0,314$$

$$-.5_{10}\text{-}23 = -5 \cdot 10^{-24}$$

$$6D5 = 6 \cdot 10^5$$

$$-E2 = -10^2$$

Die Unterschiede zur Syntax der mit Standard-E/A formatfrei gelesenen Zahlen sind:

- a) Leerzeichen werden in beliebiger Menge überlesen, sie sind also keine Trennzeichen
- b) als Exponentenzeichen sind zugelassen: E D ' 10
- c) als Kommazeichen sind "Punkt" und "Komma" zugelassen
- d) unzulässige Zeichen werden nicht als Trennzeichen gewertet, sondern führen zu Fehlern: wurde schon einmal mittels der Prozeduren READFEHL oder ERR ein Integer-Label als Fehler-Label angegeben, so erfolgt ein Sprung auf dieses Label, sonst erfolgt der Ausdruck: DATENFEHLER: und dahinter der Rest des zu lesenden Satzes - anschließend wird das Unterprogramm READ beendet.

Wird ein String formatfrei gelesen, so wird er bis zum Satzende gelesen.

Ist in einem READ-Aufruf das Satzende erreicht, und es sind noch mehr Parameter vorhanden, so wird die nächste Zeile gelesen und ausgewertet.

War beim vorigen Aufruf von READ der Satz noch nicht zu Ende gelesen, so wird kein neuer Satz gelesen, sondern der Rest ausgewertet.

Es ist also sowohl möglich mit mehreren Aufrufen von READ aus einem Satz als auch mit einem Aufruf aus mehreren Sätzen zu lesen.

Zu Beginn der Prozedur READ ist immer der Zustand "formatfrei lesen" eingestellt. Vom ersten Auftreten einer der Formatprozeduren S, SFORM, Z, oder ZFORM an wird bis zum Ende dieses READ-Aufrufes nur noch formatgebunden gelesen. S \equiv SFORM bedeutet, daß die Prozedur unter beiden Namen benutzbar ist mit identischer Wirkung.

Die Formatprozeduren haben folgende Bedeutungen bei READ:

- a) S(n) \equiv SFORM(n) auf die folgenden beliebigen Parameter wird jeweils ein String der Länge n gelesen. Ist der Satz zu kurz, so erfolgt ein Fehler.
- b) Z(n) \equiv ZFORM(n) auf die folgenden Parameter wird jeweils eine Zahl gelesen, die aus n Zeichen besteht.
Diese Zahl kann beliebig nach der oben aufgeführten Syntax aufgebaut sein.
Ist der Parameter ein Feldname, so wird die Zahl auf das erste Feldelement gelesen.
- c) B(n) \equiv BFORM(n) die nächsten n Zeichen werden überlesen.

- d) $V(n) = VFORM(n)$ es werden n Zeilenvorschübe gemacht. $V(1)$ bedeutet z.B., daß der Rest des Satzes überlesen wird
- e) $SGNR(n)$ die weiteren Lesevorgänge erfolgen von dem Medium mit der symbolischen Gerätenummer n . War diese Gerätenummer schon eingestellt, so ist der Aufruf ohne Wirkung, sonst geht der Rest des Satzes immer verloren, wenn hinterher wieder von dem vorher eingestellten Medien gelesen werden soll. Voreingestellt ist 5, wenn ein Fremdstring vorhanden ist, sonst 8.
- f) $SATZNR(n)$ die folgenden Parameter werden vom Satz mit der Nummer n gelesen.
- g) $END (LABEL)$ von nun an soll beim Ende des Mediums (bei Konsole "leere Eingabe") auf das Label LABEL gesprungen werden.
- h) $ERR (LABEL)$ Bei einem Fehler - Satz zu kurz für angegebenes Format oder Zahlensyntax falsch - soll auf LABEL gesprungen werden, und zwar auch für alle folgenden Aufrufe von READ.

Bei g) und h) werden integer-Labels auf Parameterposition erwartet.

Beispiele:

als Programm sei gegeben:

```
array F1, F2 [1:10]; real VAR1, LAENGE;  
.  
.  
.  
READ (SGNR(8), END(4711), VAR1, LAENGE,  
      if VAR1 = 0 then Z (LAENGE) else S(LAENGE), F1);  
READ (F2);  
.  
.  
.
```

als Daten seien eingegeben:

a) 0; 11; 0,0341E-2 KOMMENTAR1 ...

bewirkt: VAR1 = 0

LAENGE = 11

F1 [1] = $3,41 \cdot 10^{-4}$

F2 = " KOMMENTAR1..."

1; 5; 47.13KOMMENTAR2...

bewirkt: VAR1 = 1

LAENGE = 5

F1 = "47.13" (und zwar als String!)

F2 = "KOMMENTAR2..."

Damit sind Lesemöglichkeiten in einer Variabilität

und Bequemlichkeit gegeben, wie sie bislang keine

E/A zu bieten vermochte!

Zu erwähnen ist noch eine besondere Prozedur, die an sich in das BOGOL-STRING-Konzept [1] gehört, wegen der engen Verknüpfung nur mit dieser E/A hier erwähnt sei:

Die Prozedur TRIM(n):

Sie hat einen Integer-Parameter n , durch den beim Lesen mit READ folgendes gesteuert wird:

- = 0 : Eingabezeilen bleiben unverändert
- = 1 : Bei jeder Eingabezeile werden zunächst alle Blanks am Anfang und am Ende entfernt
- = 2 : Blanks am Zeilenbeginn werden entfernt
- = 3 : Blanks am Zeilenende werden entfernt

Dadurch werden Zeilen, die nur aus Leerzeichen bestehen, eventuell zu Leerzeilen und als solche gelesen, während sonst jede Leerzeile übergangen wird bei READ.

2.) WRITE/PRINT/TYPE/LINE:

Mit diesen Prozeduren können Strings und Zahlen auf beliebige Medien ausgegeben werden.

Ob ein Parameter eine Zahl oder einen String darstellt, wird am binären Aussehen erkannt - an der Typenkennung. Soll eine undefinierte Variable ausgegeben werden (TK = 2 oder 3), so wird ihre Tetradendarstellung gedruckt, und zwar in folgender Form: `[[TKTTTTTTTT]]`

(TK = Typenkennung, T = Tetrade).

Strings werden immer formatfrei ausgegeben, also direkt anschließend an den vorigen Parameter, ohne Zwischenraum. Bis zum ersten Auftreten der Formatprozedur Z oder ZFORM werden auch Zahlen formatfrei ausgegeben, und zwar linksbündig alle relevanten Zeichen in der kürzest möglichen Darstellung - je nachdem als Integer- oder Gleitkommazahl. Die Prozeduren WRITE und TYPE sind völlig identisch und sind nur aus Kompatibilitätsgründen zu den ALCOR-Prozeduren beide vorhanden. Es wird zu Beginn dieser beiden Prozeduren kein Zeilenvorschub erzeugt, sondern fortlaufend hinter das letzte geschriebene Zeichen in dieselbe Zeile ausgegeben.

Wird eine Zeile zu lang (bei Ausgabe auf Konsole länger als 69 Zeichen, sonst länger als 160 Zeichen),

so wird kein Parameter aufgebrochen, sondern vor Beginn des Parameters, der über das Zeilenende hinausgehen würden, wird ein Zeilenvorschub eingeführt.

Die Prozedur PRINT ist genauso gebaut, nur wird zu Beginn ein Zeilenvorschub erzeugt.

Die Prozedur LINE hat keinen Parameter und bewirkt nur einen Zeilenvorschub.

Die Formatprozeduren bewirken folgendes:

- a) $Z(n) \equiv ZFORM(n)$ alle folgenden Zahl-Parameter werden mit einer Länge von n Zeichen ausgegeben. Dabei werden die Zahlen selbst in der kürzest möglichen Darstellung rechtsbündig in dieses n Zeichen lange Feld geschrieben. Wird eine Zahl mit den relevanten Zeichen länger als n Zeichen, so wird sie gerundet. Würde sie beim Runden zu sehr verstümmelt (Beispiel: $-5,6 \cdot 10^{-13}$ sollte in 5 Zeichen dargestellt werden), so wird sie formatfrei linksbündig ausgegeben. (Dagegen paßt $-5.67 \cdot 10^{-13}$ in ein 7 Zeichen langes Feld: $-6.'-13$). Das Format hat auf Strings keine Wirkung.
- b) $B(n) \equiv BFORM(n)$ Es werden n Leerzeichen eingefügt.
- c) $V(n) \equiv VFORM(n)$ Es werden n Vorschübe erzeugt.
- d) $P(n) \equiv PFORM(n)$ Ab dem nächsten Parameter wird von Spalten n an weitergeschrieben. Liegt Spalten n hinter dem bisherigen Satzende, so werden entsprechend Blanks eingefügt.
- e) $E(n) \equiv EFORM(n)$ Der nächste Parameter, der keine For-

matprozedur ist, wird von Spalte n ab in den Satz geschrieben und er-setzt dort das bisherige. Das bisherige Satzende wird dabei auf keinen Fall verändert. Der übernächste Parameter wird also wieder hinter das alte Satzende geschrieben. Damit ist es z.B. möglich, in eine standardisierte Tabellenzeile einen Wert einzufügen, ohne die Tabelle zu zerstören.

f) SGNR(n)

Die folgenden Parameter werden in das durch die symbolische Gerätenummer n identifizierte Medium geschrieben. (Voreinstellung: 9, d.h. auf Konsole und bei eingeschaltetem Druckerprotokoll auch auf dem Drucker). Ist der letzte geschriebene Satz noch nicht abgeschlossen, so wird dies vorher getan.

g) SATZNR(n)

Der letzte Satz wird abgeschlossen, die folgenden Parameter werden in den Satz mit der Nummer n geschrieben.

h) ERR(LABEL)

Bei einem auftretenden Fehler (z.B. wenn eine LF-Datei, in die geschrieben wird, zu klein ist usw.), soll auf LABEL gesprungen werden LABEL muß ein

integer-Label sein.

Mit Hilfe dieser Prozeduren ist es auf sehr bequeme Weise möglich, sich Tabellen zu erstellen, oder auch nur in möglichst kurzer Form Ergebnisse zu drucken, und zwar in einer Variabilität, wie sie sonst kaum erreichbar ist.

Wegen der Bequemlichkeit der Prozeduren habe ich zusätzlich die Möglichkeit geschaffen, sie für reines String-handling im BOGOL-STRING/BOGOL-TAS System zu benutzen.

Das wird erreicht, indem für das Schreiben eine symbolische Gerätenummer eingestellt wird, die größer als 99 ist. In diesem Fall wird nämlich nichts ausgegeben, sondern nur der interne Ausgabepuffer aufgebaut, der als echter Algol-String organisiert ist.

Deklariert man dann die Prozeduren WRITE/PRINT bzw. TYPE als integer-procedure's, so ist ihr Funktionswert gerade der aktuelle Ausgabe-string, der dann mit anderen BOGOL-STRING-Prozeduren weiterverarbeitet werden kann.

Das eröffnet zusätzlich die Möglichkeit eines bequemen TRACE-ausdruckes, in dem für den Testfall z.B. die Gerätenummer 9 oder 6 eingestellt wird und dann die string-handling-Operationen automatisch protokolliert werden.

Denn auch bei echter Ausgabe übergeben die Prozeduren PRINT/WRITE/TYPE den aktuellen Ausgabe-string immer als Funktionswert.

Es sei noch bemerkt, daß diese Prozeduren der dritten Ebene natürlich beliebig mit den Prozeduren EIN/AUS/EAFRAG/EAZAHL/EASTEU kombinierbar sind, sofern man beachtet, daß auf Strings eingestellt sein muß beim Lesen. (EASTEU(-1)), und daß bei den Prozeduren READ/WRITE etc. immer erst zwischengepuffert wird - daß ein Satz also erst, wenn er abgeschlossen wird, tatsächlich geschrieben wird, bzw. erst nach Abarbeiten eines Satzes der nächste tatsächlich gelesen wird.

Der Zeitbedarf und Speicherbedarf liegt natürlich erheblich höher als bei den Prozeduren der zweiten Ebene, da der Rahmen der dritten Ebene eine ganze Reihe von BOGOL-STRING-Prozeduren benutzt, die daher mit anmontiert werden müssen.

Von der Zeit und dem Speicherbedarf her ergibt sich gegenüber der Algol-Standard-E/A ungefähr nur noch ein Faktor 1:2 zugunsten meiner E/A, der sich allerdings weiter verbessert, wenn die BOGOL-STRING-Prozeduren sowieso schon benutzt werden sollen.

Dafür ist die Standard-E/A bei weitem nicht so variabel und bequem zu handhaben wie diese Prozeduren.

II.

Das Prinzip der externen Arrays

Ziel dieses Unterprogrammpaketes ist es, einen Prozedurenatz zur Verfügung zu stellen, der es dem Benutzer erlaubt, auf bequeme und extrem schnelle Art große Datenmengen zu verarbeiten, die nicht gleichzeitig im Kernspeicher gehalten werden können. Bisher war dies dem Programmierer in Algol und Fortran nur möglich, indem er selbst einen E/A-Algorithmus organisierte, wobei er Dateien benutzte. Dadurch wurde er erstens mit einer Arbeit belastet, die nichts mit seiner eigentlichen Problemstellung zu tun hatte, und zweitens ging ihm sehr viel Rechenzeit und E/A-Zeit verloren dadurch, daß er die Datenorganisation benutzen mußte und damit eine zusätzliche Dateistruktur über seine eigentliche Datenstruktur gelegt würde, die er in den meisten Fällen nicht benötigte, die manchmal sogar die Arbeit erschwerte.

Dieser ganze Organisations- und Zeitaufwand kann nun wegfallen, wenn man Zwischenspeicherung von Daten auf der physikalisch niedrigsten Stufe des Rechners benutzt.

Das sind beim TR 440 Gebiete und PHYS-Dateien, deren bequeme und schnelle Bearbeitung dem Benutzer probelemorientierter Sprachen mit dem Unterprogrammpaket RB&EXTARRAY ermöglicht wird.

Die Prozeduren belegen alle zusammen etwa 2 K Kernspeicher, dessen größter Teil aber im Datenspeicher liegt, den 16-Bit-Adressenraum also nicht belastet. Dazu kommt dann für jedes externe Array ein Kernspeicherbedarf zwischen 1K und 32K, dessen Größe vom Benutzer bestimmt wird (siehe Prozedur GEBHG), ebenfalls im Datenspeicher, und zwar ab der 32. Großseite, also ab Adresse '200000' sexdezimal. Das heißt aber nicht, daß soviel Kernspeicher benötigt wird, sondern nur, daß der belegte Adressenraum dort liegt.

Im folgenden soll die Benutzung des Prozedurensatzes im einzelnen beschrieben werden.

Die Prozeduren sind sämtlich nicht-rekursiv aus Geschwindigkeitsgründen, sodaß keine formalen Prozedurparameter durchgereicht werden dürfen.

Zunächst eine kurze Übersicht über den Prozedurensatz:

- A) EXTARR (Label) Sprachinitialisierung und Anmelden eines Fehlerlabels
- B) GEBHG (SPNR, LÄNGE, KENNUNG, TRÄGER, ART, BEREICHSLÄNGE, ANZAHL, VORBESETZUNG)
Kreation eines externen Arrays auf einem Hintergrundgebiet und dynamisches Verändern von spezifischen Kenngrößen
- C) PHYSHG (SPNR, LÄNGE, KENNUNG, BEREICHSLÄNGE, ANZAHL, VORBESETZUNG) Kreation eines externen Arrays auf einer Physdatei (LFD) und dynamisches Verändern von Kenngrößen
- D) AGET (SPNR, INDEX) Bringen eines externen Feldelementes
- E) IAGET (SPNR, INDEX) wie AGET - für Integer-Elemente
- F) APUT (SPNR, INDEX, WERT)
Speichern in ein externes Feldelement
- G) AGETS
Sequentielles Bringen des nächsten externen Feldelementes
- H) IAGETS
wie AGETS - für Integer-Elemente
- I) APUTS (WERT)
sequentielles Speichern in das nächste externe Feldelement
- J) HGINF (SPNR)
Informieren über die Länge eines externen Arrays

- K) ENDBEH(SPNR) Endebehandlung für ein externes Array
durchführen
- L) STRANS(SPNR, KSPADR, ACHTNR, ANZAHL)
extrem schneller Transport von Achtel-
seiten bzw. Blöcken vom Kernspeicher in
ein externes Array und umgekehrt.
- M) Beschaffung von Feldelementen, die auf Achtelseiten-
anfang liegen, für die Prozedur STRANS:
IFACHT(FELDNAME)
Bringen des Indexes des ersten Feldele-
mentes, das auf Achtelseitenanfang liegt
- ARACHT(FELDNAME)
Verschieben des Feldanfangs eines Fel-
des auf Achtelseitenanfang
- CACHTO, ..., CACHT9
sind Namen von Common-Zonen, die auf
Achtelseitengrenze beginnen.

a) Funktionsweise

Die einzelnen Prozeduren erlauben es, jedes Ganzwort eines Gebietes oder einer Physdatei direkt zu adressieren. Man kann also damit so programmieren, als ob man Felder zur Verfügung hätte, die mehrere Tausend K Ganzworte Information enthalten können, mit einem tatsächlichen Kernspeicheraufwand zwischen 1K und 32K Ganzworten je Feld.

Die Größe der Felder auf dem Hintergrund ist dabei nur durch die Größe des zur Verfügung stehenden Hintergrundspeicherplatzes (Platte und Trommel) beschränkt.

Wird ein Element eines "Feldes" angesprochen, so wird zunächst nachgesehen, ob schon ein Stellvertreter dieses "Feldelementes" im Kernspeicher vorhanden ist, wenn nicht, wird eine gewisse Umgebung des Elementes, also ein gewisser Bereich des "Feldes", vom Hintergrund in den Kernspeicher transportiert und dort weiter verarbeitet.

Um möglichst variabel und sowohl Zeit- als auch kernspeicher-optimal arbeiten zu können, hat der Benutzer die Möglichkeit, die Größe eines solchen Bereiches und die Anzahl der Bereiche, die im Kernspeicher gleichzeitig sein sollen, für jedes externe Array getrennt frei zu wählen und dynamisch zu verändern.

Es sind maximal 15 gleichzeitig ansprechbare Felder möglich, nacheinander durch "Abmelden" von Feldern aber beliebig viele. Auf die Information eines abgemeldeten Feldes

kann durch erneutes Ansprechen aber wieder zugegriffen werden.

Wird ein Feldelement angesprochen in einem Bereich, der noch nicht im Kernspeicher liegt, die maximale gewählte Anzahl von Bereichen (obere Grenze: 32) je Feld aber schon im Kernspeicher liegt, so wird automatisch ein entsprechender ROLLOUT-ROLLIN-Algorithmus angesprochen, der nach dem FIFO-Prinzip (First in - First out) dafür sorgt, daß ein Bereich aus dem Kernspeicher ausgelagert wird und somit Platz für einen anderen Bereich schafft.

Im Fehlerfall und bei Programmende wird außerdem dafür gesorgt, daß die gesamte Information, die noch im Kernspeicher liegt, auf den Hintergrund transportiert wird, und später evtl. weiterverarbeitet werden kann.

Die Identifikation der 15 einzelnen Felder geschieht immer durch Angabe einer Speichernummer (SPNR): $1 \leq \text{SPNR} \leq 15$

Die Programme können von ALGOL, FORTRAN und TAS her benutzt werden.

Der Aufruf durch ein TAS-Programm muß folgenden Konventionen genügen:

- 1.) Die Kontrollprozedur S&CC muß mit dem relativen Eingang 1 (Anfangsaufruf) angesprungen worden sein - das geschieht z.B. implizit bei der Übersetzung eines Tas-Programmes mit der Sprachangabe TASR, oder wenn das Hauptprogramm in Algol oder Fortran geschrieben wird.

2.) Aufruf durch SFB

3.) Versorgung: in RA die Adresse eines Versorgungs-
blockes, der auf Ganzwortgrenze beginnen
und folgendermaßen aufgebaut sein muß:

1. Halbwort: Fehleradresse
2. Halbwort: Parameterzahl
3. Halbwort: Adresse des ersten
Parameters

.

.

.

$n+2$. Halbwort: Adresse des n -ten
Parameters

Ein Ansprung der Prozedur GEBHG könnte z.B. durch fol-
gende Tas.-Sequenz erfolgen:

BA(FAD/AG, 2/H, VAR1/A, VAR2/A),

SFB GEBHG,

wobei in VAR1 die Speichernummer,

in VAR2 die Anzahl Ganzworte steht.

4) Beschreibung der Einzelprozeduren

Der gesamte Komplex der externen Arrays muß zunächst initialisiert werden, um die Art der Parameterübergabe etc. auf die Sprachkonventionen des aufrufenden Programmes einzustellen usw.

Dazu dient die Prozedur

A.) EXTARR (Externe Arrays sollen benutzt werden) mit einem Parameter. Dieser Parameter muß ein Label (in Algol: Integer-Label!) sein, das angesprungen wird, wenn

- 1.) ein Fehler beim Zugriff auf eine Physdatei, oder ein Gemeinschaftsgebiet die als Hintergrundspeicher dienen, auftritt (z.B. ein Bereich angesprochen wird, der nicht mehr im Datei-Bereich liegt) oder
- 2.) bei einem Gebiet ein "Feldelement" geholt werden soll, das nicht existiert (beim Speichern eines "Feldelementes", das noch nicht existiert, wird das Gebiet implizit verlängert) oder beim Verlängern eines Gebietes ein Fehler auftritt, (z.B. Speichermangel etc.)

Die Prozedur EXTARR muß jedesmal aufgerufen werden, wenn die aufrufende Sprache sich ändert, d.h. wenn die externen Arrays von verschiedenen

Sprachen her wechselweise angesprochen werden sollen. Dann müssen die "Felder" kreiert werden. Dazu dienen die Prozeduren GEBHG und PHYSHG, mit denen auch dynamische Längenänderungen durchgeführt werden.

B) GEBHG (ein Gebiet soll als Hintergrundspeicher benutzt werden) hat folgende Parameter, von denen nur der erste obligat ist, alle anderen können fortgelassen werden, was die gleiche Wirkung hat, als ob die fortgelassenen den Wert 0 hätten:

1.) SPNR die Speichernummer des Gebietes
($1 \leq \text{SPNR} \leq 15$)

2.) LÄNGE gibt die einzustellende neue Länge des externen Feldes auf dem Hintergrund an, und zwar in Elementen (Anzahl Ganzworte). Dabei wird auf Seitengrenze aufgerundet (Eine Seite = 1024 Ganzworte = 1K).

= 0 Das Gebiet wird "abgemeldet", d.h. Kernspeicher wird freigegeben - Die Information bleibt auf dem Hintergrund erhalten.

=-1 Ist das Hintergrundgebiet schon vorhanden (z.B. neues Ansprechen eines "abgemeldeten" externen Feldes), so wird seine Länge unverändert übernommen, sonst wird die Voreinstellung von 10K Länge genommen.

=-2 Die Information, d.h. das Gebiet, wird auch auf dem Hintergrund gelöscht.

3.) KENNUNG bezeichnet den Namen des angesprochenen Hintergrundgebietes. Der Parameter kann ein Algolstring, ein Fortranstring im Sinne des Stringhandlings, oder eine Variable sein, auf die im A-Format eingelesen wurde.

= 0 Es wird entweder nur etwas an dem angegebenen Gebiet geändert, das schon einmal unter der Speichernummer angesprochen wurde, oder (beim ersten Ansprechen der Speichernummer) es wird die Voreinstellung genommen ("GEB*1", ..., "GEB*15").

Der Parameter KENNUNG darf nur beim ersten Ansprechen der jeweiligen Speichernummer $\neq 0$ sein, oder beim ersten Ansprechen nach dem "Abmelden" der Speichernummer!

4.) TRAEGER Kennzeichnet den Träger des Gebietes:

= 0 Gebiet lagert auf der Platte
= 1 Gebiet lagert auf der Trommel
(existiert das Gebiet bereits, so wird dieser Parameter ignoriert).

5.) ART charakterisiert das Gebiet näher:

- = 0 das Gebiet ist ein Laufzeitgebiet, d.h. es wird implizit bei Programmende gelöscht
- = 1 das Gebiet ist ein Dauergebiet, d.h. es wird implizit bei Abschnittsende gelöscht
- = 2 das Gebiet ist ein Gemeinschaftsgebiet. Existiert das Gemeinschaftsgebiet bereits, so wird die vorhandene Länge übernommen.
- = 3 muß angegeben werden, wenn das Gebiet ein Gemeinschaftsgebiet ist, dieses aber schon einmal im Abschnitt angesprochen wurde.

Ein Gemeinschaftsgebiet wird immer als Dauergebiet kreiert, das aber bei Abschnittsende nicht gelöscht wird, sondern explizit mit dem Parameter LÄNGE = -2 gelöscht werden muß.

Die Koordination der Gemeinschaftsgebiete, auf die ja von verschiedenen Abschnitten her zugegriffen werden kann, wird automatisch durchgeführt. Möchte man eine eigene Koordinierung benutzen, so muß man zunächst das Gebiet mit ART = 2 kreieren,

dann das Gebiet abmelden mit LÄNGE = 0
und erneut mit ART = 1 und LÄNGE = -1
die Prozedur GEBHG aufrufen.

6.) BEREICHSLÄNGE

gibt die Länge eines Bereiches an, der
jeweils in den Kernspeicher transpor-
tiert werden soll, und zwar die Länge
in Achtelseiten (eine Achtelseite = 128
Ganzworte)

= 0 Die Voreinstellung von 4 Achtelsei-
ten je Bereich soll benutzt werden.

Aus zeitlichen Gründen, die vom inter-
nen Aufbau der Unterprogramme abhän-
gen, und zur optimalen Kernspeicheraus-
nutzung sind für BEREICHSLÄNGE nur 2-er
Potenzen zwischen 1 und 256 zugelassen.

7.) ANZAHL gibt die Anzahl der Bereiche an, die
maximal gleichzeitig im Kernspeicher
liegen sollen. Dabei darf das Produkt
aus ANZAHL und BEREICHSLÄNGE die Zahl
256 (32K Ganzworte) nicht übersteigen.
Die Optimierungsstrategie rundet in-
tern die Zahl ANZAHL so auf, daß das
Produkt ANZAHL·BEREICHSLÄNGE ein ganz-
zahliges Vielfaches von 8 (also einer
Seite) ergibt.

= 0 die Voreinstellung von 2 Bereichen wird genommen. Das Produkt aus ANZAHL und BEREICHSLÄNGE ergibt die Größe des tatsächlichen Kernspeicherbedarfes, den man natürlich möglichst niedrig halten möchte, also minimal $2 \cdot 4 = 8$ Achtelseiten = 1K Ganzworte. Auf der anderen Seite wird der Zeitbedarf um so geringer, je größer die Bereiche sind. Dabei wird vorausgesetzt, daß der Zugriff auf das externe Feld in Abschnitten ziemlich sequentiell ist, da sonst im Extremfall für jeden Feldzugriff ein Hintergrundtransport nötig ist. Um die Zahl der Verdrängungen von Bereichen möglichst klein zu halten, empfiehlt sich auch eine möglichst große Anzahl von Bereichen. Wenn man z.B. hauptsächlich an 4 Stellen des externen Arrays gleichzeitig zugreifen will, sollte man ANZAHL = 4 wählen. Ist ANZAHL = 0, so werden keine Stellvertreter der Feldelemente im Kernspeicher gewünscht. Auf das externe Array kann dann nur mit der Prozedur STRANS zugegriffen werden.

8.) VORBESETZUNG Ist dieser Parameter vorhanden, so gibt er an, mit welchem Inhalt das Gebiet vorzubesetzen ist (d.h. der Wert 0 hat in diesem Fall eine andere Bedeutung als ein Fehlen des Parameters). Ist eine Vorbesetzung gewünscht und das Gebiet ein Gemeinschaftsgebiet (ART=2), so wird es nur dann vorbesetzt, wenn es kreiert wird. Wurde es in einem anderen Abschnitt bereits kreiert, so ist die Angabe einer VORBESETZUNG ohne Wirkung. VORBESETZUNG muß eine Variable sein, deren Inhalt als Vorbesetzung des Gebietes genommen wird. Bei anderen Gebieten (ART=2) muß der Benutzer selbst dafür sorgen, daß sie nur bei der Kreation oder wenn dies wirklich gewünscht wird, vorbesetzt werden und damit die vorige Information gelöscht wird.

Jede dieser für ein externes Array spezifischen Größen kann dynamisch im Programmlauf geändert werden (außer TRÄGER und ART - diese beiden nur durch vorheriges explizites Löschen

des Gebietes: Länge = -2). Wird ein Gebiet irgendwann angesprochen, das bereits existiert, so wird dieses immer übernommen, eventuell unter Änderung seiner Länge (zu beachten ist dabei, daß die Länge eines Gemeinschaftsgebietes nicht verändert werden kann), wenn diese nicht mit -1 angegeben ist.

C.) Die Prozedur PHYSHG (eine Physdatei soll als Hinter-
grundspeicher genommen werden)

kreiert ebenfalls ein externes Array,
dessen echte Information aber in einer
Phys-Datei liegt. Das ist zum Beispiel
sinnvoll, wenn diese Physdatei eine LF-
datei ist, deren Information nicht ver-
lorengehen soll nach Abschnittsende.

Diese Physdatei muß beim Aufruf von
PHYSHG bereits existieren. Sie wird nicht
implizit verlängert wie ein Gebiet beim
Speichern, und ein Löschen (LÄNGE = -2)
bewirkt ebenfalls nur ein Abmelden wie
LÄNGE = 0. Kreation, Löschen und Längen-
änderung muß der Benutzer also selbst auf
Kommandoebene durchführen.

PHYSHG hat folgende Parameter:

- 1.) SPNR die Speichernummer ($1 \leq \text{SPNR} \leq 15$)
- 2.) LÄNGE (siehe Prozedur GEBHG)
- 3.) KENNUNG ist eine Zahl, die der Datei im
Starte-Kommando (Spezifikation DATEI=...)
als symbolische Gerätenummer zuge-
ordnet wurde ($1 \leq \text{KENNUNG} \leq 99$)..
Die DNUMMER-Spezifikation des
Starte-Kommandos wird dabei mit
berücksichtigt.

= 0 bedeutet, daß die Datei schon einmal durch einen Aufruf von PHYSHG angesprochen wurde und daß nur andere Parameter des externen Arrays geändert werden sollen. Der Parameter darf nur beim ersten Ansprechen der Physdatei bzw. beim ersten Ansprechen nach einem eventuellen Abmelden ungleich 0 sein

- | | | |
|-------------------|---|--|
| 4.) BEREICHSLÄNGE | } | für diese Parameter gilt das gleiche wie für die entsprechenden Parameter der Prozedur GEBHG |
| 5.) ANZAHL | | |
| 6.) VORBESETZUNG | | |

Die Feldelemente eines externen Arrays werden nun durch folgende Prozeduren angesprochen:

- D) AGET (Array: GET) Bringen eines Feldelementes
- E) IAGET (Integerarray: GET) Bringen eines Integer-elementes eines Feldes
- F) APUT (Array: PUT) Speichern eines Feldelementes
- G) AGETS (Array: GET sequentiell) Sequentielles Bringen des nächsten Feldelementes
- H) IAGETS (Integerarray: GET sequentiell) Sequentielles Bringen des nächsten Integer-Feldelementes
- I) APUTS (Array: PUT sequentiell) Sequentielles Speichern des nächsten Feldelementes

AGET, IAGET, AGETS und IAGETS sind Funktionsprozeduren, APUT und APUTS echte Prozeduren.

Funktionsweise und Parameterübergabe dieser Prozeduren:

AGET und IAGET haben 2 Parameter:

- 1.) SPNR die Speichernummer des externen Arrays, das angesprochen werden soll
- 2.) INDEX der Index des zu bringenden Feldelementes ($0 \leq \text{INDEX} \leq \text{Maximalgröße}$).

Der Inhalt des durch SPNR und INDEX identifizierten Elementes des externen arrays wird als Funktionswert übergeben, wobei IAGET dazu dient, auch Integer-Inhalte ansprechen zu können.

APUT hat 3 Parameter:

- 1.) SPNR die Speichernummer des externen Arrays
- 2.) INDEX der Index des Elementes, in das gespeichert werden soll
- 3.) WERT gibt den Ganzwortinhalt an, den das durch SPNR und INDEX identifizierte Element des externen Arrays erhalten soll.

AGETS und IAGETS

haben keinen Parameter.

Sie übergeben als Funktionswert den Inhalt desjenigen "Feldelementes", das durch J und I+1 identifiziert wird, wenn J die Speichernummer und I der Index ist, die durch den letzten Aufruf von AGET, IAGET, AGETS oder IAGETS angesprochen wurden. Durch eine Kette von Aufrufen von AGETS oder IAGETS werden also sequentiell die hintereinanderliegenden Feldelemente des zuletzt "bringend" angesprochenen externen Arrays als Funktionswert übergeben. IAGETS ist wiederum für Integer-Werte gedacht.

APUTS hat einen Parameter:

- 1.) WERT gibt den Inhalt an, den dasjenige externe Feldelement bekommen soll, das durch J und I+1 identifiziert wird, wenn J die Speichernummer und I der Index ist, die durch den letzten Aufruf von APUT oder APUTS angesprochen wurden.

Eine Kette von Aufrufen von APUTS erlauben es also, sequentiell Werte in die aufeinanderfolgenden Elemente desjenigen externen Arrays zu speichern, dessen Speichernummer zuletzt "speichernd" angesprochen wurde.

Wann immer möglich, sollte man diese sequentiellen Prozeduren AGETS, IAGETS und APUTS benutzen, da sie noch erheblich schneller sind als die entsprechenden nicht-sequentiellen AGET, IAGET und APUT.

Die Prozedur:

J) HGINF (über Hintergrundlänge informieren) ist eine Funktionsprozedur, die als Funktionswert die Anzahl der Elemente (Ganzworte) eines externen Arrays auf dem Hintergrundspeicher übergibt. Sie hat einen Parameter:

- 1.) SPNR die Speichernummer eines externen Arrays, über dessen Länge man sich informieren will.

Eine weitere Prozedur ist:

K) ENDBEH (Endebehandlung durchführen)

Durch den Aufruf dieser Prozedur wird der Inhalt der Bereiche, die im Kernspeicher liegen, auf den Hintergrund transportiert, also in das Hintergrundgebiet bzw. die Physdatei.

Diese Prozedur kann ohne Schaden immer aufgerufen werden, wenn man sichergehen will, daß alle in das externe Feld mittels APUT oder APUTS gespeicherte Information auch in dem Hintergrund-"Array" steht. Sie muß aufgerufen werden, wenn unter derselben Speichernummer ein anderes Gebiet oder eine andere Physdatei angesprochen werden soll, ohne daß vorher das bisherige externe Array "abgemeldet" wird (siehe GEBHG, LÄNGE = 0).

Beim "Abmelden" eines externen Arrays, bei Operatorlaufende und bei Fehlerabbruch wird sie implizit intern angesprungen (von der Kontrollprozedur S&CC), nicht jedoch bei Anspringen des in EXTARR angegebenen Fehlerlabels.

Die Prozedur hat einen Parameter:

- 1.) SPNR die Speichernummer des externen Arrays, für das die Endebehandlung durchgeführt werden soll

= 0: Für alle angesprochenen und noch nicht "abgemeldeten" externen Arrays soll die Endebehandlung durchgeführt werden.

Die Prozedur:

L) STRANS ermöglicht einen extrem schnellen Daten-
transport zwischen Kernspeicher und Hinter-
grundgebiet (bzw. Physdatei).

Es ist möglich, sowohl nur mit der Prozedur STRANS auf ein externes Array zuzugreifen, da dies der schnellste überhaupt mögliche Daten-transport ist, als auch gemischt mit den anderen Prozeduren für externe Arrays. Arbeitet man nur mit der Prozedur STRANS, so wählt man zweckmäßigerweise, um Kernspeicherplatz zu sparen, ANZAHL=0 beim Aufruf von GEBHG bzw. PHYSHG, die auf jeden Fall vorher aufgerufen werden müssen, da auch bei STRANS das externe Array über die Speichernummer identifiziert wird.

Werden auch die anderen Prozeduren für externe Arrays benutzt, so muß vor einem Wechsel von anderen Prozeduren zu STRANS die Endbehandlung für die entsprechende Speichernummer durchgeführt werden, damit die Information richtig übergeben und weiterverarbeitet werden kann. Eine Ausnahme davon bilden Transporte mit Hilfe von STRANS, bei denen man sicher sein kann, daß keiner der dadurch angesprochenen Bereiche zum Zeitpunkt des Aufrufes von STRANS

im Kernspeicher liegt - vor solchen Transportbefehlen ist eine Endbehandlung nicht notwendig.

M) Die Prozedur STRANS arbeitet direkt mit Achtelseitentransport (SSR 3 68) bzw. Blocktransporten (SSR 253 23) bei Physdateien. Deswegen wird als Quelladresse im Kernspeicher (bzw. Zieladresse) immer eine Achtelseitenadresse verlangt. Um in Algol oder Fortran solche Achtelseitenadressen zu bekommen, gibt es folgende Möglichkeiten:

- 1.) Man kann Fehler in 10 eigens dafür vorgesehenen Common-Zonen deklarieren, die so gelegt sind, daß sie immer auf Achtelseitengrenze beginnen. Die Namen dieser Common-Zonen sind: CACHT0, ..., CACHT9
- 2.) Man kann sich von einem vorhandenen ein-dimensionalen Feld den Index des ersten Elementes besorgen, daß auf Achtelseitengrenze liegt. Dazu dient die Prozedur IFACHT (Index des ersten Feldelementes, das auf Achtelseitenanfang liegt).

Der Funktionswert der Integer-Funktionsprozedur IFACTH ist dieser Index.

Als Beispiel in Algol:

```
J: = IFACHT(A);
```

A ist dabei der Name eines eindimensionalen arrays. (In Fortran muß A ein INTEGER*4 oder REAL*4 - FELD sein!)

Das Feldelement A [J] liegt dann auf Achtelseitenanfang und kann in STRANS als Kernspeicheradresse benutzt werden (Ebenso natürlich

```
A [J+I*128] . (I = 1,2,.....)
```

- 3.) Man kann sich ein beliebiges (!) Feld so ändern lassen, daß das erste existierende Feldelement auf Achtelseitenanfang zu liegen kommt. Dazu dient die (echte) Prozedur ARACHT (Array auf Achtseitengrenze beginnen lassen).

Beispiel in Fortran:

```
Dimension J1 (1000,10)
```

```
 .
```

```
 .
```

```
 .
```

```
CALL ARACHT(J1)
```

```
 .
```

```
 .
```

```
 .
```

Danach liegt J1 (1,1) auf Achtelseitenanfang.

Dabei ist allerdings zu berücksichtigen, daß einige Ganzworte eventuell dadurch verlorengehen, daß der Feldanfang nach hinten verschoben wird: Im ungünstigsten Fall 127 Ganzworte.

Nach diesen vorbereitenden Erklärungen nun die Beschreibung der Prozedur:

STRANS hat 4 Parameter:

- 1.) SPNR Die Speichernummer des externen Arrays (siehe Prozedur GEBHG und PHYSHG).
- 2.) KSPADR Die Kernspeicheradresse für den Transport. KSPADR muß eine (indizierte) Variable sein, die auf Achtelseitengrenze liegt (siehe Ausführungen weiter oben: "IFACHT" und "ARACHT").
- 3.) ACHTNR erste Achtelseitennummer (bzw. Blocknummer bei Physdateien), in die bzw. aus der der Transport erfolgen soll. (Die Achtelseitennummern bzw. Blocknummern werden von 0 an durchnumeriert). Ein Block hat genau wie eine Achtelseite 128 Ganzworte.

4.) ANZAHL |ANZAHL| gibt die Anzahl der zu transportierenden Achtelseiten bzw. Blöcke an.

Ist $ANZAHL > 0$, so erfolgt der Transport vom Kernspeicher auf den Hintergrund, also "Schreiben" in Gebiet bzw. Physdatei

Ist $ANZAHL < 0$, wird vom Gebiet bzw. aus der Physdatei gelesen.

Literaturverzeichnis:

- 1 Arbeitsbericht des Rechenzentrums der Ruhr-Universi-
tät Bochum
Nr. 7210 BO.E6.03
BOGOL-STRING, eine flexible Zeichenkettenverarbeitung
in Algol 60

- 2 Arbeitsbericht des Rechenzentrums der Ruhr-Universi-
tät Bochum
Nr. 7206 BO.E2.06
BOGOL-TAS, ein Weg zur systematischen Programmierung
in Algol am TR 440

- 3 TR 440
BS3 Systemdienste Unterlagensammlung

Bisher erschienene Arbeitsberichte des Rechenzentrums
der Ruhr-Universität Bochum

- Nr. 7101: K.-H. Mohn, M. Rosendahl, H. Zoller
AIDA, eine Dialogsprache für den TR 440
- Nr. 7102: K.-H. Mohn, M. Rosendahl, H. Zoller
AIDA, ein Dialogsystem und seine Implementierung in ALGOL
- Nr. 7103: K.-H. Mohn, M. Rosendahl, H. Zoller
AIDA, Manual für den Benutzer
- Nr. 7104: 4. Jahresbericht des Rechenzentrums (Juni 1970 bis Juni 1971)
- Nr. 7105: H. Wupper
WR MBO2 - Ein einfaches Band-Betriebssystem für einen mittleren Rechner
- Nr. 7201: H. Windauer
Existenzsätze zur $(0, 1, \dots, R-2, R)$ -Interpolation
- Nr. 7202: W. Schelongowski
DIATRACE, Ein System zur interaktiven Assemblerprogrammierung
- Nr. 7203: M. Jäger, M. Rosendahl, R. Staake
Einführung in die Listenverarbeitung anhand der Dialogsprache AIDA
- Nr. 7204: R. Mannshardt, P. Pottinger
Einführung in die Benutzung des Teilnehmer-Rechensystems TR 440
in der RUB
- Nr. 7205: 5. Jahresbericht des Rechenzentrums (1.7.1971 bis 30.6.1972)
- Nr. 7206: M. Rosendahl
BOGOL-TAS, ein Weg zur systemnahen Programmierung in ALGOL am TR 440
- Nr. 7207: W. Stark
ILW, Programmsystem zur Berechnung des Instationären Ladungs-
wechsels von Verbrennungskraftmaschinen (Modulbeschreibung und
Eingabekonventionen)
- Nr. 7208: W. Stark
ILW, Programmsystem zur Berechnung des Instationären Ladungs-
wechsels von Verbrennungskraftmaschinen (Regelmechanismus und
Berechnung der Rohrströmung)
- Nr. 7209: H. Ehlich
Anregungen und Kritik zum Betriebs- und Programmiersystem der TR 440

- Nr. 7210: M. Rosendahl
BOGOL-STRING, eine flexible Zeichenkettenverarbeitung in ALGOL 60
- Nr. 7211: H. Camici, H. Claus, H. Ehlich, D. Kipp
Arbeitsbericht über ein Programm zur Haushaltsführung
- Nr. 7301: R. Mannshardt, K.-H. Mohn, H.J. Münch, P. Pottinger
Einführung in die Benutzung des Teilnehmer Rechensystems TR 440
2. geänderte Auflage
- Nr. 7302: K.-H. Mohn
Über einige Anwendungen des Computers in der Medizin
- Nr. 7303: R. Buchmann
BODAT, ein schnelles und platzsparendes System zur Datenmanipulation und -speicherung in Algol 60 und Fortran