Technische Universität Chemnitz
Sonderforschungsbereich 393
Numerische Simulation auf massiv parallelen Rechnern

Dag Lohse

Ein Standard-File für 3D-Gebietsbeschreibungen - Datenbasis und Programmschnittstelle data_read

Preprint SFB393/98-17

Preprint-Reihe des Chemnitzer SFB 393

SFB393/98-17 **April 1998**

Author's addresses:

Dag Lohse
TU Chemnitz
Fakultät für Mathematik
D-09107 Chemnitz

mail: dlohse@mathematik.tu-chemnitz.de
http://www.tu-chemnitz.de/sfb393/

Inhaltsverzeichnis

1 Motivation

Die Standard–File–Struktur wurde entwickelt, um eine einheitliche Datenschnittstelle zwischen verschiedenen Programmen der Forschergruppe bereitzustellen. Ausgehend vom Aufbau des 2D-Standard–Files wurde eine Datenstruktur erarbeitet, die es ermöglicht, flexibel und plattformunabhängig die Topologie und Geometrie von geometrischen Körpern in Randrepräsentation[1] abzubilden und weiterhin Materialdaten und Randbedingungen für die Lösung von Differentialgleichungen auf diesen 3D–Gebieten aufzunehmen.

Die Standard–File–Struktur ist so konzipiert, daß sie jederzeit änder– und erweiterbar ist, so daß sie wechselnden Anforderungen gerecht werden kann. Um die dabei entstehenden Versionen unterscheiden zu können, wird vom Standard–File ein Versionskennung mitgeführt, an dem die Einleseroutinen erkennen können, wie die Datenzeilen zu interpretieren sind. Alle vorherigen Versionen behalten also Gültigkeit.

Neben der Definition eines Fileformates, die den Aufbau und die Eigenschaften eines Standard–Files beschreibt, gehören zum Umfang der Entwicklung einer einheitlichen Beschreibung der Eingabedaten auch eine Datenbasis, die in der Lage ist, den Inhalt eines Standard–Files im Speicher eines Rechners darzustellen sowie die zur Erzeugung und Manipulation dieser Datenbasis notwendigen Funktionen.

Durch den Einsatz unterschiedlicher Rechnerarchitekturen und mehrerer Programmiersprachen muß diese Datenbasis mit möglichst einfachen Mitteln auskommen. Einfache Mittel heißt in diesem Fall, daß weder strukturierte Datentypen, wie sie in C üblich sind, noch Zeiger, d.h. Speicheradressen, Verwendung finden durften. Da die F77-Programme, für deren Datenbereitstellung das Standard–File entwurfen wurde, grundsätzlich mit einem linearen Feld arbeiten, das den maximal verfügbaren freien Speicherplatz der Maschinen überdeckt, mußte bei dem Einsatz in einer F77–Umgebung auf eine dynamische Speicherverwaltung und auf den Einsatz von Funktionen verzichtet werden, die intern dynamischen Speicher anfordern, das umfaßt z.B. die gepufferten IO-Funktionen von C.

Da die Anwenderprogramme oft nur Teile der Datenbasis benötigen und an die Reihenfolge und den Aufbau der einzelnen Datenfelder oft spezielle Anforderungen stellen, wurde eine Schnittstelle geschaffen, die mit Hilfe einer Anforderungssprache flexibel die benötigten Informationen in dem gewünschten Format aus einem Standard–File extrahiert.

Zum Bearbeiten der Eigenschaften von Gebieten und der Visualisierung von Standard–Files stehen selbständige Programme zur Verfügung. Alle Programme und Funktionen zur Bearbeitung von Standard–Files benutzen dabei zunächst die Datenbasis und die darauf operierenden Funktionen. So sind Änderungen der Definition des Standard–Files schneller durchführbar.

[1] Randrepräsentation: Der Körper wird durch eine Hierarchie seiner begrenzenden Flächen, Kanten und Eckpunkte dargestellt. Diese Darstellung der Topologie des Objekts ist relativ unabhängig von dessen tatsächlichen geometrischen Dimensionen.

Dieses Dokument beschreibt die allgemeine Datenbasis und die Schnittstelle zur Erzeugung der nutzerdefinierten Datenstruktur. Die Definition des Fileformats ist dem Preprint SFB393/98-11 zu entnehmen.

2 Bezeichnungen

Folgende Datentypen werden pseudonym verwendet:

int	INTEGER	INTEGER*4
float	REAL	REAL*4

Um auf die Datensätze des Standard–Files effektiv zugreifen zu können, werden diese Daten in einem zusammenhängenden Arbeitsvektor abgelegt, der durch Bereiche, Blöcke, Felder und Teilfelder strukturiert ist. Die Namen, die im Standard–File (innerhalb eines Namensraumes) frei wählbar sind, werden dabei durch interne Nummern ersetzt, die von 1 aufsteigend vergeben werden. Diese Nummern dienen dem schnellen Zugriff auf die einzelenen Daten und werden deshalb direkt als Feldindizes benutzt. Damit ist immer, abweichend von der in C üblichen Praxis, das Element mit dem Index 1 das erste gültige Element eines Feldes.

Beim Erzeugen der Datenbasis entstehen zwar keine **negativen Nummern** für die Datensätze, das negative Vorzeichen kann jedoch dazu benutzt werden, spezielle Zustände anzuzeigen. Funktionen, die auf der Datenbasis arbeiten, sollten den **Absolutwert** der Nummer als Feldindex verwenden. Die 0 (Null) dient als Fehleranzeiger und kennzeichnet ungültige Daten. Negative Nummern haben folgende Bedeutung für spezielle Felder (siehe dazu auch Abschnitt 4.5):

`Name`–Feld	Duplikat des Datensatzes vorhanden
`Data`–Feld	Nach Einlesen – Unbekannter Name
	Nach Bearbeitung (Face) – Kante falsche Orientierung
`Type`–Feld	bei Topologiedaten – Unbekannter (Nutzerdefinierter) Typ

Soll ein Teilfeld mit einem **Offset** `k` in einem (Gesamt-)feld beginnen, so ist damit stets gemeint, daß das Element mit dem Index `k` im Gesamtfeld das Element mit dem Index 1 im Teilfeld darstellt.

Unter einem Offset in einem Feld eines bestimmten Types ist also der Abstand vom Beginn des Feldes, gemessen in Elementen des Feldes gemeint; **der wirkliche Abstand der Speicheradressen ist vom Typ des Feldes abhängig.**

3 Aufgaben und Eigenschaften der Datenbasis

Die Datenbasis ist das programminterne Äquivalent zum Standard–File. Alle Informationen, die ein Standard–File enthalten kann, sind in der Datenbasis darstellbar, alle Daten, die eine Datenbasis aufnehmen kann, haben eine Entsprechung im Standard–File.

Die Datenbasis dient in erster Linie dazu, allen Programmen, die das Standard–File einsetzen, einen direkten Zugriff auf den Inhalt des Standard–Files zu ermöglichen. Dies kann natürlich durch vorgelagerte Hilfsfunktionen geschehen, die die Daten entsprechend den Bedürfnissen des Anwenders anpassen und die Datenbasis in die benötigte Form konvertieren.

Ein weiterer Vorteil des Einsatzes der Datenbasis besteht darin, daß neue Features des Standard–Files, gekennzeichnet durch eine neue Version, relativ einfach zentral für alle Programme zugänglich gemacht werden können.

Alle Funktionen, die die Datenbasis manipulieren, sind in C geschrieben, ein Zugriff durch F77 ist dann möglich, wenn sich entsprechende Datentypen gefunden werden.

Die Datenbasis ist als ein Speicherblock (Arbeitsvektor) realisiert, der mittels Offset–Vektoren, die selbst Teil des Arbeitsvektors sind, in mehrere Teilbereiche zerlegt werden kann, die die Daten aufnehemen. Zugunsten von F77 wurde auf die Verwendung von Zeigern verzichtet. Als weitere Voraussetzung, die jedoch abgeschwächt werden kann, sei genannt, daß sowohl INTEGER als auch REAL den gleichen Speicherbedarf haben. Notwendig ist jedoch nur, daß Datentypen in C und der benutzenden Programmiersprache gefunden werden, die die gleiche interne Darstellung besitzen. Der Einsatz der Datenbasis ist damit nur bedingt portabel. Auch auf den Einsatz strukturierter Datentypen wurde zur Erhöhung der Portabilität verzichtet. Da sich C und FORTRAN jedoch auch bei der Behandlung von Feldgrenzen unterscheiden, existiert für die C–Umgebung eine Menge von Makros, die den Zugriff auf die Teilbereiche als Feldzugriffe ermöglichen, der Feldindex wird dabei entsprechend angepaßt. Diese Makros erlauben auch Änderungen an der Struktur der Datenbasis, ohne das dadurch Änderungen in den Nutzerprogrammen erforderlich würden. Der Einsatz der Makros in Funktionen ist jedoch an bestimmte Voraussetzungen gebunden, speziell muß die Datenbasis in der Funktion über den Namen `DataBase` ansprechbar sein.

Da diese Makros ständig die Feldgrenzen umrechnen, sind sie nicht sehr effektiv, soll die Datenbasis nur in C benutzt werden, kann durch Manipulation der Offset–Werte ein direkter Zugriff erfolgen. Diese Möglichkeit ist jedoch noch nicht implementiert.

Da eine Datenbasis ein zusammenhängender Speicherblock ist, ist die gleichzeitige Bearbeitung mehrerer Standard–Files möglich, zusätzliche Informationen zu den Datenbasen sind natürlich entsprechend extern zu verwalten.

4 Aufbau der Datenbasis

Die Datenbasis ist ein zusammenhängendes lineares Feld (Arbeitsvektor), das in mehrere logische Teilbereiche gegliedert ist. Der Typ des Arbeitsvektors ist `CHAR`. Da die einzelnen Teilfelder ohnehin mit ihrem eigenen (den enthaltenen Daten entsprechenden) Typ benutzt werden, ist der Typ des Gesamtfeldes nur insofern von Bedeutung, daß er zum Bestimmen des Beginns der Teilfelder aus den in Kopf gespeicherten Offsets und dem Beginn des Gesamtfeldes notwendig ist.

Die Datenbasis hat folgenden Grundaufbau:

`<Kopf>`	`<Parameter>`	`<Counter>`	`<Datenbereich>`	`<Infos>`

Die einzelnen Teil**bereiche** haben folgende Bedeutung:

`<Kopf>`	Struktur und Zustand der Datenbasis – Offsets der Teilbereiche
`<Parameter>`	statische Größen der Datenbasis – direkte Parameter aus dem Infoteil und berechnete Parameter für Feldlängen
`<Counter>`	Belegungstabelle der Teilbereiche
`<Datenbereich>`	Der Inhalt des Datenteils des Standard–Files
`<Infos>`	Der Inhalt der Informationszeilen des Standard–Files kann bei Speichermangel freigegeben werden

4.1 Genereller Aufbau des Datenbereiches

Der Bereich `<Datenbereich>` ist in weitere **Blöcke** für die im Standard–File möglichen Datenblöcke unterteilt. Mit Ausnahme des Blockes für die `VERTEX`–Daten haben diese Blöcke eine prinzipiell gleiche Organisation:

`<Namen>`	`<Typen>`	`<Daten_Offsets>`	`<Daten>`

Das Feld `<Daten>` besteht wiederum aus vielen Teilfeldern, deren Zuordnung zu den entsprechenden Objekten über das Offsetfeld `<Daten_Offsets>` ermöglicht wird. Die einzeln Teilfelder eines solchen Blockes haben folgenden Inhalt:

`<Namen>`	Die externen Namen der Objekte
`<Typen>`	Die Typkennzeichen der Objekte
`<Daten_Offsets>`	Zuordnung der Teilfelder von `<Daten>`
`<Daten>`	Teilfelder zur Aufnahme der Daten

Alle zu einem Datensatz gehörenden Daten werden im `<Daten>` fortlaufend gespeichert. `<Daten_Offsets>` enthält für jeden Datensatz den Startindex in diesem Feld. Die Anzahl der Daten eines Datensatzes ergibt sich aus der Differenz des Startindex des aktuellen Datensatzes zum Startindex der Daten des nächsten Datensatzes. Da der Startindex vor dem Einlesen der Daten festgelegt wird, funktioniert diese Methode auch beim letzten Datensatz des Feldes.

Eine detailierte Beschreibung des Aufbaus dieser Bereiche und die genaue Beschreibung des Inhalts erfolgt in den speziellen Abschnitten.

Wird das lineare Feld in C mit dem Bezeichner `DataBase` benutzt, so existiert eine Menge von Makros, mit deren Hilfe das Feld bequem angesprochen werden kann. Dabei ist jedoch zu beachten das dieser Name nur innerhalb von Funktionen benutzt werden sollte, um globale Variablen zu vermeiden und es zu ermöglichen, mehrere Datenbasen gleichzeitig zu bearbeiten. `BASE` soll im weiteren als ein beispielhafter Name für ein solches lineares Feld in einer F77–Umgebung benutzt werden.

4.2 Kopfbereich

Der Kopfbereich beschreibt einerseits in Zustandsvariablen den Bearbeitungszustand (Einlesezustand, Konvertierung usw.) und die Version des zugeordneten Standard–Files und enthält andererseits die Offsets für den Beginn der einzelnen Bereiche, Blöcke und Felder der Datenbasis.

Obwohl es für die Übersichtlichkeit und Wartbarkeit zuträglicher wäre, den Kopf als einen strukturierter Typ in C zu implementieren, wurde jedoch ein Feld vom Typ `int` (`INTEGER*4`) gewählt, um einen direkten Zugriff von `FORTRAN`–Programmen zu gewährleisten. Die Offsets selbst sind die Abstände des ersten Elementes eines Teilfeldes vom Beginn des Gesamtfeldes gemessen in `CHAR`. Diese Maßeinheit kann durch die Konvertierungsfunktion in `INTEGER` umgerechnet werden, dies ist jedoch nur für `F77`–Umgebungen möglich.

In C kann der Kopfbereich direkt über das Makro `DB_KOPF` als Feld angesprochen werden. Die Anordnung der Elemente des Kopf–Feldes ist in den folgenden Tabellen dargestellt, wobei der Index für F77–Funktionen natürlich bei 1 beginnt. C–Funktionen können auf die einzelnen Werte mit Hilfe von vordefinierten Makros zugreifen. Da sich die Blöcke der Datenbasis in ihrer prinzipiellen Organisation gleichen, ergibt sich auch eine einheitliche Struktur der Offsets der einzelnen Felder der Blöcke. In C ist es deshalb möglich, auch auf alle Offsets eines Blockes mit einem Makro zuzugreifen, wobei der VERTEX–Block eine Ausnahme bildet. Während `DB_KOPF` tatsächlich ein Makro darstellt, wurde `KOPF` nur als ein beispielhafter Name eines F77–Feldes gewählt. `KOPF` fällt mit dem Beginn von `BASE` zusammen.

Steuerfelder:

F77	Element C	Makro	Beschreibung
KOPF(1)	DB_KOPF[0]	DB_PAR_O	Offset des Parameter
KOPF(2)	DB_KOPF[1]	DB_COUNT_O	Offset der Counter

VERTEX:

KOPF(3)	DB_KOPF[2]	DB_VN_O	Offset der VERTEX–Namen
KOPF(4)	DB_KOPF[3]	DB_VX_O	Offset der X–Koordinaten
KOPF(5)	DB_KOPF[4]	DB_VY_O	Offset der Y–Koordinaten
KOPF(6)	DB_KOPF[5]	DB_VZ_O	Offset der Z–Koordinaten

EDGE `DB_EDGE_O`:

KOPF(7)	DB_KOPF[6]	DB_EN_O	Offset der EDGE–Namen
KOPF(8)	DB_KOPF[7]	DB_ET_O	Offset der EDGE–Typen
KOPF(9)	DB_KOPF[8]	DB_EP_O	Offset EDGE–Daten–Offsets
KOPF(10]	DB_KOPF[9]	DB_ED_O	Offset der EDGE–Daten

FACE `DB_FACE_O`:

KOPF(11)	DB_KOPF[10]	DB_FN_O	Offset der FACE–Namen
KOPF(12)	DB_KOPF[11]	DB_FT_O	Offset der FACE–Typen
KOPF(13)	DB_KOPF[12]	DB_FP_O	Offset FACE–Daten–Offsets
KOPF(14)	DB_KOPF[13]	DB_FD_O	Offset der FACE–Daten

SOLID `DB_SOLID_O`:

KOPF(15)	DB_KOPF[14]	DB_SN_O	Offset der SOLID–Namen
KOPF(16)	DB_KOPF[15]	DB_ST_O	Offset der SOLID–Typen
KOPF(17)	DB_KOPF[16]	DB_SP_O	Offset SOLID–Daten–Offsets
KOPF(18)	DB_KOPF[17]	DB_SD_O	Offset der SOLID–Daten

REGION `DB_REGION_O`:

KOPF(19)	DB_KOPF[18]	DB_RN_O	Offset der REGION–Namen
KOPF(20)	DB_KOPF[19]	DB_RT_O	Offset der REGION–Typen
KOPF(21)	DB_KOPF[20]	DB_RP_O	Offset REG.–Daten–Offsets
KOPF(22)	DB_KOPF[21]	DB_RD_O	Offset der REG.–Daten

DIRICHLET `DB_DIR_O`:

KOPF(23)	DB_KOPF[22]	DB_DN_O	Offset der DIR.–Namen
KOPF(24)	DB_KOPF[23]	DB_DT_O	Offset der DIR.–Typen
KOPF(25)	DB_KOPF[24]	DB_DP_O	Offset DIR.–Daten–Offsets
KOPF(26)	DB_KOPF[25]	DB_DD_O	Offset der DIR.–Daten

NEUMANN DB_NEU_O:

KOPF(27)	DB_KOPF[26]	DB_NN_O	Offset der NEU.–Namen
KOPF(28)	DB_KOPF[27]	DB_NT_O	Offset der NEUMANN–Typen
KOPF(29)	DB_KOPF[28]	DB_NP_O	Offset NEU.–Daten–Offsets
KOPF(30)	DB_KOPF[29]	DB_ND_O	Offset der NEUMANN–Daten

MATERIAL DB_MAT_O:

KOPF(31)	DB_KOPF[30]	DB_MN_O	Offset der MAT.–Namen
KOPF(32)	DB_KOPF[31]	DB_MT_O	dummy
KOPF(33)	DB_KOPF[32]	DB_MP_O	Offset MAT.–Daten–Offsets
KOPF(34)	DB_KOPF[33]	DB_MD_O	Offset der MATERIAL–Daten

FACE_GEO DB_GEO_O:

KOPF(35)	DB_KOPF[34]	DB_GN_O	Offset der F_GEO–Namen
KOPF(36)	DB_KOPF[35]	DB_GT_O	Offset der FACE_GEO–Typen
KOPF(37)	DB_KOPF[36]	DB_GP_O	Offset F_GEO–Daten–Offsets
KOPF(38)	DB_KOPF[37]	DB_GD_O	Offset der FACE_GEO–Daten

Information:

KOPF(39)	DB_KOPF[38]	DB_INF_O	Offset des Info–Kopfes
KOPF(40)	DB_KOPF[39]	DB_INFD_O	Offset der Info–Daten

Zustandsvariable:

KOPF(41)	DB_KOPF[40]	DB_STATUS	Einlesezustand
KOPF(42)	DB_KOPF[41]	DB_VERSION	Version des Standard–Files
KOPF(43)	DB_KOPF[42]	DB_SPECIAL	reserviert
KOPF(44)	DB_KOPF[43]	DB_SIZE	Größe der Datenbasis
KOPF(45)	DB_KOPF[44]	DB_W_FLAGS	Schreibstatus

4.3 Parameterbereich

Der Parameterbereich enthält alle Größen, die in Parameterzeilen und im `#HEADER:`–Block des Standard–Files abgelegt sind sowie die daraus abgeleiteten Größen, die für die Dimensionierung der Blöcke und Felder notwendig sind. Die Parameter werden beim Einlesen des Datenteils des Standard–Files nicht mehr verändert, es handelt sich also um statisches Material.

Zur Motivation, statt eines übersichtlichen strukturierten Datentyps ein Feld zu wählen, gelten die Ausführungen im Abschnitt 4.2 entsprechend. Der Parameterbereich ist in C über das Makro `DB_PAR` als Feld ansprechbar, für die einzelnen Elemente gibt es naürlich wieder Makros. Auch innerhalb des Parameterfeldes schlägt sich die organisatorische Gleichheit der Blöcke in gleichen Parametersätzen für die Blöcke nieder, die wieder durch entsprechende Makros angesprochen werden können.

Während `DB_PAR` tatsächlich ein Makro darstellt, wurde `PAR` nur als ein beispielhafter Name eines F77–Feldes gewählt.

EDGE `DB_EDGE_P`:

F77	Element C	Makro	Beschreibung
PAR(1)	DB_PAR[0]	DB_E_MAX	Max. Anzahl EDGES
PAR(2)	DB_PAR[1]	DB_M_EP	Max. Anzahl E-Daten
PAR(3)	DB_PAR[2]	DB_M_ET	Max. EDGE–Typ
PAR(4)	DB_PAR[3]	DB_A_ED	Durchschn. Daten/EDGE
PAR(5)	DB_PAR[4]	DB_M_ED	Max. Anzahl Daten/EDGE

FACE `DB_FACE_P`:

PAR(6)	DB_PAR[5]	DB_F_MAX	Max. Anzahl FACES
PAR(7)	DB_PAR[6]	DB_M_FP	Max. Anzahl F-Daten
PAR(8)	DB_PAR[7]	DB_M_FT	Max. FACE–Typ
PAR(9)	DB_PAR[8]	DB_A_FD	Durchschn. Daten/FACE
PAR(10)	DB_PAR[9]	DB_M_FD	Max. Anzahl Daten/FACE

SOLID `DB_SOLID_P`:

PAR(11)	DB_PAR[10]	DB_S_MAX	Max. Anzahl SOLIDS
PAR(12)	DB_PAR[11]	DB_M_SP	Max. Anzahl S-Daten
PAR(13)	DB_PAR[12]	DB_M_ST	Max. SOLID–Typ
PAR(14)	DB_PAR[13]	DB_A_SD	Durchschn. Daten/SOLID
PAR(15)	DB_PAR[14]	DB_M_SD	Max. Anzahl Daten/SOLID

REGION `DB_REGION_P`:

PAR(16)	DB_PAR[15]	DB_R_MAX	Max. Anzahl REGIONS
PAR(17)	DB_PAR[16]	DB_M_RP	Max. Anzahl R-Daten
PAR(18)	DB_PAR[17]	DB_M_RT	Max. REGION–Typ
PAR(19)	DB_PAR[18]	DB_A_RD	Durchschn. Daten/REGION
PAR(20)	DB_PAR[19]	DB_M_RD	Max. Anzahl Daten/REGION

DIRICHLET `DB_DIR_P`:

PAR(21)	DB_PAR[20]	DB_D_MAX	Max. Anzahl DIR-RB'S
PAR(22)	DB_PAR[21]	DB_M_DP	Max. Anzahl Daten/FG
PAR(23)	DB_PAR[22]	DB_M_DT	Max. DIR–Typ
PAR(24)	DB_PAR[23]	DB_A_DD	Durchschn. Daten/FG
PAR(25)	DB_PAR[24]	DB_M_DD	Max. Anzahl Daten/DIR-RB

NEUMANN `DB_NEU_P`:

PAR(26)	DB_PAR[25]	DB_N_MAX	Max. Anzahl NEU-RB'S
PAR(27)	DB_PAR[26]	DB_M_NP	Max. Anzahl Daten/FG
PAR(28)	DB_PAR[27]	DB_M_NT	Max. NEU–Typ
PAR(29)	DB_PAR[28]	DB_A_ND	Durchschn. Daten/FG
PAR(30)	DB_PAR[29]	DB_M_ND	Max. Anzahl Daten/NEU-RB

MATERIAL `DB_MAT_P`:

PAR(31)	DB_PAR[30]	DB_M_MAX	Max. Anzahl MATERIALS
PAR(32)	DB_PAR[31]	DB_M_MP	Max. Anzahl M-Daten
PAR(33)	DB_PAR[32]	DB_M_MT	dummy
PAR(34)	DB_PAR[33]	DB_A_MD	Durchschn. Daten/MAT
PAR(35)	DB_PAR[34]	DB_M_MD	Max. Anzahl Daten/MAT

FACE_GEO `DB_GEO_P`:

PAR(36)	DB_PAR[35]	DB_G_MAX	Max. Anzahl FACE_GEOS
PAR(37)	DB_PAR[36]	DB_M_GP	Max. Anzahl G-Daten
PAR(38)	DB_PAR[37]	DB_M_GT	Max. FACE_GEO–Typ
PAR(39)	DB_PAR[38]	DB_A_GD	Durchschn. Daten/GEO
PAR(40)	DB_PAR[39]	DB_M_GD	Max. Anzahl Daten/GEO

Diverse `DB_DIV_P`:

PAR(41)	DB_PAR[40]	DB_V_MAX	Max. Anzahl VERTEX
PAR(42)	DB_PAR[41]	DB_DOF	Anzahl der Freiheitsgrade
PAR(43)	DB_PAR[42]	DB_A_DOFD	Durchschn. Daten/DOF
PAR(44)	DB_PAR[43]	DB_DOFD	reserviert
PAR(45)	DB_PAR[44]	DB_DOFN	reserviert
PAR(46)	DB_PAR[45]	DB_D_HS	Größe Dir.Heap
PAR(47)	DB_PAR[46]	DB_N_HS	Größe Neu.Heap
PAR(48)	DB_PAR[47]	DB_M_BT	Max. RB–Typ
PAR(49)	DB_PAR[48]	DB_M_BP	Max. Anzahl RB-Daten
PAR(50)	DB_PAR[49]	DB_M_BD	Max. Anzahl Daten/RB

4.4 Counterbereich

Der Counterbereich enthält die Anzahl der bereits eingelesenen Datensätze. In einer C-Umgebung ist mittels des Makros `DB_CNT` ein direkter Zugriff auf den Counterbereich als Feld möglich; `CNT` ist wiederum nur ein beispielhafter Name für ein F77-Feld. Der Counterbereich ist ein INTEGER–Feld mit folgendem Aufbau:

Counter:

F77	Element C	Makro	Beschreibung
CNT(1)	DB_CNT[0]	DB_V_CNT	Gelesene VERTEX
CNT(2)	DB_CNT[1]	DB_E_CNT	Gelesene EDGES
CNT(3)	DB_CNT[2]	DB_F_CNT	Gelesene FACES
CNT(4)	DB_CNT[3]	DB_S_CNT	Gelesene SOLIDS
CNT(5)	DB_CNT[4]	DB_R_CNT	Gelesene REGIONS
CNT(6)	DB_CNT[5]	DB_D_CNT	Gelesene DIRICHLET RB'S
CNT(7)	DB_CNT[6]	DB_N_CNT	Gelesene NEUMANN RB'S
CNT(8)	DB_CNT[7]	DB_M_CNT	Gelesene MATERIALS
CNT(9)	DB_CNT[8]	DB_G_CNT	Gelesene FACE_GEOS

4.5 Datenbereich

Der Datenbereich enthält die eigentlichen Informationen über das zu beschreibende Gebiet. Entsprechend dem Aufbau des Standard–Files enthält der Datenbereich für jede mögliche Art Datenblock eines Standard–Files **einen** Block, während das Standard–File mehrere gleichartige Datenblöcke enthalten darf. Jeder dieser Blöcke ist noch einmal entsprechend der Anforderungen in Felder untergliedert. Der Datenbereich besteht aus folgenden Blöcken:

`<VERTEX>`	`<EDGE>`	`<FACE>`	`<SOLID>`	`<REGION>`
`<DIRICHLET>`	`<NEUMANN>`	`<MATERIAL>`	`<FACE_GEO>`	

Der Aufbau dieser Blöcke ist inden folgenden Abschnitten erläutert.

4.5.1 VERTEX–Block

Der `VERTEX`–Block enthält alle Daten der `VERTEX`–Blöcke des Standard–Files. Entsprechend dem Aufbau einer `VERTEX`–Zeile ist dieser Block in vier Felder untergliedert:

`<VertexName>`	`<XCoord>`	`<YCoord>`	`<ZCoord>`

Diese Felder haben folgende Bedeutung:

`<VertexName>`	Namen der Eckpunkte
`<XCoord>`	
`<YCoord>`	Geometrische Koordinaten
`<ZCoord>`	

Sie lassen sich folgendermaßen ansprechen:

	Typ	F77 (erstes Element)	C (Feldbeginn)	C (erstes Element)
VertexName	INTEGER	BASE(KOPF(3))	DB_VertexName	DB_VertexName[1]
XCoord	REAL	BASE(KOPF(4))	DB_XCoord	DB_XCoord[1]
YCoord	REAL	BASE(KOPF(5))	DB_YCoord	DB_YCoord[1]
ZCoord	REAL	BASE(KOPF(6))	DB_ZCoord	DB_ZCoord[1]

Der F77-Zugriff beschreibt den Zugriff auf das erste Feldelement, der C-Zugriff beschreibt den Beginn des Feldes, der Zugriff auf das erste Element erfolgt über `DB_XCoord[1]` und nicht über `*DB_XCoord` wie in C sonst üblich.

Die Bezeichnungen `DB_VertexName`, `DB_XCoord`, `DB_YCoord` und `DB_ZCoord` sind als Makros definiert, die von einem Gesamtfeld mit dem Namen `DataBase` ausgehen. `DB_V_MAX` (`PAR(41)`) enthält den maximal zulässigen Index für die oben genannten Felder. `DB_V_CNT` (`CNT(1)`) gibt den letzten tatsächlich benutzten Index an.

4.5.2 EDGE–Block

Der `EDGE`–Block entspricht den `EDGE`–Blöcken des Standard–Files. Er ist in vier Felder untergliedert:

<EdgeName>	<EdgeType>	<EdgePtr>	<EdgeData>

Diese Teilfelder haben folgende Bedeutung:

<EdgeName>	Namen der Kante
<EdgeType>	Typ der Kante (ohne Bedeutung)
<EdgePtr>	Offsets der Teilfelder von <EdgeData>
<EdgeData>	Felder von `VERTEX`–Nummern der Kante

Da der Typ einer Kante für Erweiterungen vorgesehen ist, die beispielsweise Unterteilungspunkte einer Kante erfordern, kann die Datenbasis Kanten darstellen, die durch mehr als zwei Punkte definiert werden. Für jede Kante existiert ein Teilfeld des Feldes <EdgeData>, das die Nummern aller Punkte enthält, die zur Definition dieser Kante gehören. Die Länge eines solchen Teilfeldes kann natürlich von Kante zu Kante variieren, weshalb sich ein Hilfsfeld notwendig macht, daß die Offsets dieser Teilfelder im Feld <EdgeData> angibt.

Der Zugriff auf die Teilfelder des des `EDGE`–Bereiches ist folgendermaßen möglich:

	Typ	F77 (erstes Element)	C (Feldbeginn)	C (erstes Element)
<EdgeName>	INTEGER	BASE(KOPF(7))	DB_EdgeName	DB_EdgeName[1]
<EdgeType>	INTEGER	BASE(KOPF(8))	DB_EdgeType	DB_EdgeType[1]
<EdgePtr>	INTEGER	BASE(KOPF(9))	DB_EdgePtr	DB_EdgePtr[1]
<EdgeData>	INTEGER	BASE(KOPF(10))	DB_EdgeData	DB_EdgeData[1]

Der Zugriff auf die einzelnen Elemente des Feldes DB_EdgeData ist jedoch nur in seltenen Fällen sinnvoll, da es sich bei diesem Feld eigentlich nur um die Zusammenfassung mehrerer Teilfelder handelt. Die Nummern der Eckpunkte, die zu einer Kante mit der Nummer k gehören befinden sich im Feld DB_EdgeData ab dem Index DB_EdgePtr[k]. Die Anzahl der zu dieser Kante gehörenden Eckpunkte ergibt sich aus der Subtraktion DB_EdgePtr[k+1] - DB_EdgePtr[k].

Soll das Datenfeld der Kante k an ein Unterprogramm übergeben werden, so ist dies in C durch DB_EdgeData + DB_EdgePtr[k] möglich. In diesem Unterprogramm beginnt das übergebene Feld dann jedoch mit dem Index 0.

Die Bezeichnungen DB_EdgeName, DB_EdgeType, DB_EdgePtr und DB_EdgeData sind als Makros definiert, die von einem Gesamtfeld mit dem Namen DataBase ausgehen. DB_E_MAX (PAR(1)) enthält den maximal zulässigen Index für die ersten drei Felder. Der Zähler DB_E_CNT (CNT(2)) enthält die Anzahl der eingelesenen Kanten–Datensätze und gibt damit den letzten tatsächlich benutzten Index an. DB_M_EP (PAR(2)) enthält die Größe des Feldes DB_EdgeData.

4.5.3 FACE–Block

Der FACE–Block entspricht den FACE–Blöcken des Standard–Files. Er ist in vier Felder untergliedert:

<FaceName>	<FaceType>	<FacePtr>	<FaceData>

Diese Teilfelder haben folgende Bedeutung:

<FaceName>	Namen der Fläche
<FaceType>	Geometrienummer der Fläche
<FacePtr>	Offsets der Teilfelder von <FaceData>
<FaceData>	Felder von EDGE–Nummern der Fläche

Das Teilfeld <FaceType> enthält die Nummern der Flächengeometrie–Beschreibung, falls solche vorhanden sind. Enthält das Standard–File keinen FACE_GEO:–Block, so werden diejenigen Nummern übernommen, die als Flächentypen im Standard–File eingertagen sind. Ist jedoch ein FACE_GEO:–Block vorhanden, der jedoch keinen Eintrag für den Flächentyp enthält, wird im Feld <FaceType> die Nummer aus dem Standard–File übernommen und zur Unterscheidung mit einem negativen Vorzeichen versehen.

Eine 0 (Null) als Eintrag im Feld <FaceType> bedeutet, daß die Fläche keine besonderen geometrischen Eigenschaften besitzt.

Der Zugriff auf die Felder des des FACE–Blockes ist folgendermaßen möglich:

	Typ	F77 (erstes Element)	C (Feldbeginn)	C (erstes Element)
<FaceName>	INTEGER	BASE(KOPF(11))	DB_FaceName	DB_FaceName[1]
<FaceType>	INTEGER	BASE(KOPF(12))	DB_FaceType	DB_FaceType[1]
<FacePtr>	INTEGER	BASE(KOPF(13))	DB_FacePtr	DB_FacePtr[1]
<FaceData>	INTEGER	BASE(KOPF(14))	DB_FaceData	DB_FaceData[1]

Der Zugriff auf das Feld `<FaceData>` erfolgt analog zu der in Abschnitt 4.5.2 beschriebenen Vorgehensweise.

Die Bezeichnungen `DB_FaceName`, `DB_FaceType`, `DB_FacePtr` und `DB_FaceData` sind als Makros definiert, die von einem Gesamtfeld mit dem Namen `DataBase` ausgehen. Der Wert `DB_F_MAX` (`PAR(6)`) enthält den maximal zulässigen Index für die ersten drei Felder. `DB_F_CNT` (`CNT(3)`) gibt den letzten tatsächlich benutzten Index an. `DB_M_FP` (`PAR(7)`) enthält die Größe des Feldes `DB_FaceData`.

4.5.4 SOLID–Block

Der `SOLID`–Block entspricht den `SOLID`–Blöcken des Standard–Files. Er ist in vier Felder untergliedert:

<SolidName>	<SolidType>	<SolidPtr>	<SolidData>

Diese Felder haben folgende Bedeutung:

<SolidName>	Namen des Körpers
<SolidType>	Materialnummer des Körpers
<SolidPtr>	Offsets der Teilfelder von <SolidData>
<SolidData>	Felder von FACE–Nummern des Körpers

Ist im Standard–File ein `MATERIAL`–Block vorhanden, so enthält das Feld `<SolidType>` die Nummern der Materialdatensätze, andernfalls werden die Nummern übernommen, die im Standard–File den Körpertyp bezeichnen. Sind `MATERIAL`–Blöcke vorhanden, die jedoch keine Beschreibung für den Körper–Typ enthalten, wird die Nummer aus dem Standard–File übernommen und mit einem negativen Vorzeichen versehen.

Der Zugriff auf die Felder des des `SOLID`–Bereiches ist folgendermaßen möglich:

	Typ	F77 (erstes Element)	C (Feldbeginn)	C (erstes Element)
<SolidName>	INTEGER	BASE(KOPF(15))	DB_SolidName	DB_SolidName[1]
<SolidType>	INTEGER	BASE(KOPF(16))	DB_SolidType	DB_SolidType[1]
<SolidPtr>	INTEGER	BASE(KOPF(17))	DB_SolidPtr	DB_SolidPtr[1]
<SolidData>	INTEGER	BASE(KOPF(18))	DB_SolidData	DB_SolidData[1]

Der Zugriff auf das Feld `<SolidData>` erfolgt analog zu der in Abschnitt 4.5.2 beschriebenen Vorgehensweise.

Die Bezeichnungen DB_SolidName, DB_SolidType, DB_SolidPtr und DB_SolidData sind als Makros definiert, die von einem Gesamtfeld mit dem Namen DataBase ausgehen. DB_S_MAX (PAR(11)) enthält den maximal zulässigen Index für die ersten drei Felder. DB_S_CNT (CNT(4)) gibt den letzten tatsächlich benutzten Index an. DB_M_SP (PAR(12)) enthält die Größe des Feldes DB_SolidData.

4.5.5 REGION–Block

Der REGION–Block entspricht den REGION–Blöcken des Standard–Files. Er ist in vier Felder untergliedert:

<RegionName>	<RegionType>	<RegionPtr>	<RegionData>

Diese Teilfelder haben folgende Bedeutung:

<RegionName>	Namen des Gebietes
<RegionType>	Typ des Gebietes
<RegionPtr>	Offsets der Teilfelder von <RegionData>
<RegionData>	Felder von SOLID–Nummern des Gebietes

Das Feld <RegionType> enthält die Nummern, die als Gebietstyp im Standard–File angegeben sind; für die Datenbasis selbst haben sie keine Bedeutung.

Der Zugriff auf die Felder des REGION–Blockes ist folgendermaßen möglich:

	Typ	F77 (erstes Element)	C (Feldbeginn)	C (erstes Element)
<RegionName>	INTEGER	BASE(KOPF(19))	DB_RegionName	DB_RegionName[1]
<RegionType>	INTEGER	BASE(KOPF(20))	DB_RegionType	DB_RegionType[1]
<RegionPtr>	INTEGER	BASE(KOPF(21))	DB_RegionPtr	DB_RegionPtr[1]
<RegionData>	INTEGER	BASE(KOPF(22))	DB_RegionData	DB_RegionData[1]

Der Zugriff auf das Feld <RegionData> erfolgt analog zu der in Abschnitt 4.5.2 beschriebenen Vorgehensweise.

Die Bezeichnungen DB_RegionName, DB_RegionType, DB_RegionPtr und DB_RegionData sind als Makros definiert, die von einem Gesamtfeld mit dem Namen DataBase ausgehen. DB_R_MAX (PAR(16)) enthält den maximal zulässigen Index für die ersten drei Felder. DB_R_CNT (CNT(5)) gibt den letzten tatsächlich benutzten Index an. DB_M_RP (PAR(17)) enthält die Größe des Feldes DB_RegionData.

4.5.6 MATERIAL–Block

Der MATERIAL–Block entspricht den MATERIAL–Blöcken des Standard–Files. Er ist in drei Felder untergliedert:

<MaterialName>	<MaterialPtr>	<MaterialData>

Diese Teilfelder haben folgende Bedeutung:

<MaterialName>	Namen des Materiales
<MaterialPtr>	Offsets der Teilfelder von <MaterialData>
<MaterialData>	Die REAL–Werte der Materialdaten

Für das Typ–Feld existieren zwar Makros und Offsets, um die Einheitlichkeit der Datenbereiche weitestgehend zu gewährleisten, es wird jedoch kein Feld <MaterialType> angelegt.

Der Zugriff auf die Felder des des MATERIAL–Bereiches ist folgendermaßen möglich:

	Typ	F77 (erstes Element)	C (Feldbeg.)	C (erstes Element)
<MaterialName>	INTEGER	BASE(KOPF(31))	DB_MatName	DB_MatName[1]
<MaterialPtr>	INTEGER	BASE(KOPF(33))	DB_MatPtr	DB_MatPtr[1]
<MaterialData>	REAL	BASE(KOPF(34))	DB_MatData	DB_MatData[1]

Der Zugriff auf das Feld <MaterialData> erfolgt analog zu der in Abschnitt 4.5.2 beschriebenen Vorgehensweise.

Die Bezeichnungen DB_MatName, DB_MatPtr und DB_MatData sind als Makros definiert, die von einem Gesamtfeld mit dem Namen DataBase ausgehen. DB_M_MAX (PAR(31)) enthält den maximal zulässigen Index für die ersten drei Felder. DB_M_CNT (CNT(8)) gibt den letzten tatsächlich benutzten Index an. DB_M_MP (PAR(32)) enthält die Größe des Feldes DB_MatData.

4.5.7 FACE_GEO–Block

Der FACE_GEO–Block entspricht den FACE_GEO–Blöcken des Standard–Files. Er ist in vier Felder untergliedert:

<GeoName>	<GeoType>	<GeoPtr>	<GeoData>

Diese Felder haben folgende Bedeutung:

<GeoName>	Namen der Geometriebeschreibung
<GeoType>	beschreibender Gleichungstyp
<GeoPtr>	Offsets der Teilfelder von <GeoData>
<GeoData>	REAL Parameter für die beschreibende Gleichung

Das Feld `<GeoType>` verschlüsselt die Gleichung, die zur Beschreibung der Geometrie benutzt wird. Durch den Gleichungstyp wird festgelegt, wie die Parameter im Feld `<GeoData>` ausgewrtet werden. Folgende Gleichungstypen sind vordefiniert:

`<fg-name>`	`<fg-count>`	Beschreibung
1	6	(n_x, n_y, n_z),(x_0, y_0, z_0) Normalenvektor, Punkt auf Ebene
2	6	(x_0, y_0, z_0),(n_x, n_y, n_z) Punkt auf Ebene, Normalenvektor
11	7	(x_0, y_0, z_0),(x_1, y_1, z_1),r zwei Punkte auf Zylinderachse, Radius
21	4	(x_0, y_0, z_0),r Kugelmittelpunkt, Radius
31	7	(x_0, y_0, z_0),(x_1, y_1, z_1),r Kegelspitze, Punkt auf Kegelachse, Radius an dieser Stelle
41	8	(x_0, y_0, z_0),(x_1, y_1, z_1),A,B zwei Punkte auf Rotationsachse, Parameter der Gleichung der rotierenden Kurve $\pm\frac{\xi^2}{a^2} \pm \frac{\mu^2}{b^2} = 1$ mit $A = \pm a^2$, $B = \pm b^2$ Bem.: $B < 0$ einschaliges, $A < 0$ zweischaliges Hyperboloid
51	8	(x_0, y_0, z_0),(n_x, n_y, n_z),R,r Torusmittelpunkt, Normalenvektor der Grundebene, Rotationsradius (Abstand zum Mittelpunkt des rotierenden Kreises), Radius des rotierenden Kreises

Der Zugriff auf die Felder des des `FACE_GEO`–Bereiches ist folgendermaßen möglich:

	Typ	F77 (erstes Element)	C (Feldbeginn)	C (erstes Element)
`<GeoName>`	`INTEGER`	`BASE(KOPF(35))`	`DB_GeoName`	`DB_GeoName[1]`
`<GeoType>`	`INTEGER`	`BASE(KOPF(36))`	`DB_GeoType`	`DB_GeoType[1]`
`<GeoPtr>`	`INTEGER`	`BASE(KOPF(37))`	`DB_GeoPtr`	`DB_GeoPtr[1]`
`<GeoData>`	`INTEGER`	`BASE(KOPF(38))`	`DB_GeoData`	`DB_GeoData[1]`

Der Zugriff auf das Feld `<GeoData>` erfolgt analog zu der in Abschnitt 4.5.2 beschriebenen Vorgehensweise.

Die Bezeichnungen `DB_GeoName`, `DB_GeoType`, `DB_GeoPtr` und `DB_GeoData` sind als Makros definiert, die von einem Gesamtfeld mit dem Namen `DataBase` ausgehen. `DB_G_MAX` (`PAR(36)`) enthält den maximal zulässigen Index für die ersten drei Felder. `DB_G_CNT` (`CNT(9)`) gibt den letzten tatsächlich benutzten Index an. `DB_M_GP` (`PAR(37)`) enthält die Größe des Feldes `DB_GeoData`.

4.5.8 DIRICHLET–Block

Obwohl Randbedingungen im Standard–File durch die Benutzung von Unterblöcken eine komlexere Struktur haben, als die anderen Datenblöcke, werden Randbedingungen in der gleichen Art und Weise in der Datenbasis dargestellt, wie die anderen Datenblöcke. So ist der `DIRICHLET`–Block, der dem `DIRICHLET`–Blöcken des Standard–Files entspricht, in vier Felder gegliedert:

`<DirName>`	`<DirType>`	`<DirPtr>`	`<DirData>`

Diese Teilfelder haben folgende Bedeutung:

`<DirName>`	Nummer der zugehörenden Fläche
`<DirType>`	Typen der beschreibenden Gleichungen
`<DirPtr>`	Offsets der Teilfelder von `<DirData>`
`<DirData>`	REAL Parameter für die beschreibende Gleichungen

Im Gegensatz zu den einfachen Datenblöcken, in denen jedem Datensatz je ein Element im Namens–, Typen– und Offset–Feld zugeordnet ist, belegt jede Dirichlet–Randbedingung zwar genau einen Speicherplatz im Namensfeld, über das die Zuordnung zu den Flächen erfolgt, im Typen– und im Offset–Feld werden jedoch soviele Speicherplätze belegt, wie Freiheitsgrade vorhanden sind. Die Nummer der Randbedingung kann also nicht mehr als Index für diese Felder dienen, vielmehr wird der korrekte Index für die Felder des ersten Freiheitsgrades durch (Nummer - 1) * Anzahl der Freiheitsgrade ermittelt.

Das Feld `<DirType>` verschlüsselt die Gleichung, die zur Beschreibung der Randbedingung (eines Freiheitsgrades) benutzt wird. Durch den Gleichungstyp wird festgelegt, wie die Parameter im Feld `<DirData>` ausgewrtet werden. Folgende Gleichungstypen sind vordefiniert, wobei `rbdata` für das entsprechende Teilfeld von `<DirData>` steht:

- `rbtype=0` Dieser Freiheitsgrad stellt (für diese Fläche) eine offene Randbedingung dar, es werden keine `rbdata` benötigt.
- `rbtype=1` Dieser Freiheitsgrad ist für die gesamte Fläche konstant, der Parameter `rbdata(1)` enthält den Wert dieser Konstanten.
- `rbtype=2` Der Freiheitsgrad wird durch eine lineare Gleichung im *globalen* Koordinatensystem beschrieben. Der Wert $f(x, y, z)$ dieses Freiheitsgrades ergibt sich aus: $f(x, y, z) = \texttt{rbdata(1)} * x + \texttt{rbdata(2)} * y + \texttt{rbdata(3)} * z + \texttt{rbdata(4)}$.

Der Zugriff auf die Teilfelder des des `DIRICHLET`–Bereiches ist folgendermaßen möglich:

	Typ	F77 (erstes Element)	C (Feldbeg.)	C (erstes Element)
`<DirName>`	INTEGER	BASE(KOPF(23))	DB_DirName	DB_DirName[1]
`<DirType>`	INTEGER	BASE(KOPF(24))	DB_DirType	DB_DirType[1]
`<DirPtr>`	INTEGER	BASE(KOPF(25))	DB_DirPtr	DB_DirPtr[1]
`<DirData>`	REAL	BASE(KOPF(26))	DB_DirData	DB_DirData[1]

Der Zugriff auf das Feld <DirData> erfolgt analog zu der in Abschnitt 4.5.2 beschriebenen Vorgehensweise.

Die Bezeichnungen DB_DirName, DB_DirType DB_DirPtr und DB_DirData sind als Makros definiert, die von einem Gesamtfeld mit dem Namen DataBase ausgehen. DB_D_MAX (PAR(21)) enthält den maximal zulässigen Index für das Feld <DirName>. Der gültige Indexbereich für die Felder <DirType> und <DirPtr> beginnt bei 1 und endet bei DB_D_MAX * DB_DOF bzw. bei DB_D_MAX * DB_DOF + 1. Der Zähler DB_D_CNT (CNT(6)) gibt den letzten tatsächlich benutzten Index des Feldes <DirName> an. DB_D_HS (PAR(46)) enthält die maximale Anzahl von Werten, die das Feld <DirData> aufnehmen kann.

4.5.9 NEUMANN–Block

Die Darstellung von Neumann–Randbedingungen in der Datenbasis entspricht der Darstellung von Dirichlet–Bedingungen. Die Grundlegenden Bemerkungen aus dem Abschnitt 4.5.8 gelten deshalb für Neumann–Randbedingungen analog. Der NEUMANN–Block, der dem NEUMANN–Blöcken des Standard–Files entspricht, ist in vier Felder gegliedert:

<NeuName>	<NeuType>	<NeuPtr>	<NeuData>

Diese Teilfelder haben folgende Bedeutung:

<NeuName>	Nummer der zugehörenden Fläche
<NeuType>	Typen der beschreibenden Gleichungen
<NeuPtr>	Offsets der Teilfelder von <NeuData>
<NeuData>	REAL Parameter für die beschreibende Gleichungen

Die Gleichungstypen des Feldes <NeuType> entsprechen den Typen im DIRICHLET–Bereich (Abschnitt 4.5.8).

Der Zugriff auf die Teilfelder des des NEUMANN–Bereiches ist folgendermaßen möglich:

	Typ	F77 (erstes Element)	C (Feldbeg.)	C (erstes Element)
<NeuName>	INTEGER	BASE(KOPF(27))	DB_NeuName	DB_NeuName[1]
<NeuType>	INTEGER	BASE(KOPF(28))	DB_NeuType	DB_NeuType[1]
<NeuPtr>	INTEGER	BASE(KOPF(29))	DB_NeuPtr	DB_NeuPtr[1]
<NeuData>	REAL	BASE(KOPF(30))	DB_NeuData	DB_NeuData[1]

Der Zugriff auf das Feld <NeuData> erfolgt analog zu der in Abschnitt 4.5.2 beschriebenen Vorgehensweise.

Die Bezeichnungen DB_NeuName, DB_NeuType DB_NeuPtr und DB_NeuData sind als Makros definiert, die von einem Gesamtfeld mit dem Namen DataBase ausgehen. DB_N_MAX (PAR(26)) enthält den maximal zulässigen Index für das Feld <NeuName>. Der gültige Indexbereich für die Felder <NeuType> und <NeuPtr> beginnt bei 1 und endet bei DB_N_MAX

* DB_DOF bzw. bei DB_N_MAX * DB_DOF + 1. Der Zähler DB_N_CNT (CNT(7)) gibt den letzten tatsächlich benutzten Index des Feldes <NeuName> an. DB_N_HSi (PAR(46)) enthält die maximale Anzahl von Werten, die das Feld <NeuData> aufnehmen kann.

4.6 Info–Bereich

Der Info–Bereich enthält den Inhalt der Informations–Zeilen des Standard–Files. Er besteht aus zwei Feldern:

<InfoKopf>	<InfoHeap>

Diese Felder haben folgende Bedeutung:

<InfoKopf>	Parameter und Offsets von <InfoHeap>
<InfoHeap>	Daten der Informations–Zeilen

Das INTEGER–Feld <InfoKopf> enthält neben den Offsets der im Felde <InfoHeap> gespeicherten Zeichenketten auch Informationen über die Größe und den Beginn des nächsten freien Speicherbereiches des CHAR–Feldes <InfoHeap>.

In einer C–Umgebung kann auf die beiden Felder direkt durch die Makros DB_INF und DB_INF_HEAP Zugegriffen werden, INF steht symbolisch für ein F77-Feld, das <InfoKopf> enthält.

<InfoKopf>:

F77	Element C	Makro	Beschreibung
INF(1)	DB_INF[0]	DB_Desc_O	Offset DESCRIPTION
INF(2)	DB_INF[1]	DB_Date_O	Offset DATE
INF(3)	DB_INF[2]	DB_User_O	Offset USER
INF(4)	DB_INF[3]	DB_E_Type_O	Offset EQN_TYPE
INF(5)	DB_INF[4]	DB_Dim_O	Offset DIMENSION
INF(6)	DB_INF[5]	DB_Prog_O	Offset PROGRAM
INF(7)	DB_INF[6]	DB_INF_LEN	Länge von <InfoHeap>
INF(8)	DB_INF[7]	DB_NEXT_I_DATA	Nächster freie Bereich in <InfoHeap>
INF(9)	DB_INF[8]	DB_INFO_FREE	verbleibender Speicherplatz in <InfoHeap>

5 Grundlegende Funktionen auf der Datenbasis

5.1 DB_Dialog() und DB_No_Dialog()

Die beiden Funktionen DB_Dialog() und DB_No_Dialog() steuern das Verhalten der andern auf der Datenbasis arbeitenden Funktionen bei der Fehlerbehandlung.

Standardmäßig werden vier Klassen von Fehlern unterschieden:

Kritisch	sofortiger Programmabbruch
Schwer	Nutzerdialog – meist Programmabbruch
Unkritisch	Nutzerdialog – meist Programmfortsetzung
Warnung	Programmfortsetzung

Nach dem Aufruf von DB_No_Dialog() entfällt der Nutzerdialog und schwere Fehler werden wie kritische Fehler behandelt, unkritische Fehler rufen dagegen nur noch Warnungen hervor. Die Funktion DB_Dialog() stellt den Defaultzustand wieder her.

Beide Funktionen erwarten keine Argumente und liefern keinen Rückgabewert.

5.2 DB_Create() und DB_CreateF()

Die beiden Funktionen DB_Create() und DB_CreateF() erzeugen aus einem Standard–File eine Datenbasis. Sie Unterscheiden sich jedoch darin, daß DB_CreateF() auf einem vorgegebenen Speicherbereich arbeitet, während DB_Create() selbständig Speicher vom System anfordert. DB_Create() ist außerdem in der Lage, während des Einlesens entstehende Hilfsfelder zur weiteren Nutzung bereitzustellen. Die Nutzung von DB_Create() ist nur aus einer C–Umgebung möglich.

```
DB_Type * DB_Create(const char * const filename,NSLp help[])
```
liefert als Rückgabewert einen Zeiger auf eine Datenbasis, die den Inhalt des Standard–Files `filename` enthält. Im Fehlerfall liefert `DB_Create()` `NULL`. Während des Einlesens werden Hilfsfelder erzeugt, die es ermöglichen, aus den Namen eines Elementes sehr schnell dessen interne Nummer zu ermitteln. Wird an `help` die Adresse einer Struktur übergeben, die in der Lage ist, die Startadressen dieser Hilfsfelder aufzunehmen, so werden diese (eigentlich temporären) Felder nicht gelöscht.

```
int DB_CreateF(const char *fname, char *DataBase, int *len, \
                                                  int *errnr, int nlen)
```
erzeugt auf dem Feld `DataBase`, dessen Länge in `len` übergeben wird, eines Datenbasis, die den Inhalt von `fname` repräsentiert. In `len` wird die nach dem Erzeugen der Datenbasis verbleibende Restlänge von `DataBase` und in `errnr` ein Fehlercode zurückgegeben.

Ein fehlerloses Einlesen des Files wird durch den Fehlercode 0 (Null) angezeigt. Wird diese Funktion nicht von F77 aufgerufen, so muß in `nlen` die Länge des Filenamens `fname` übergeben werden, wird die Funktion durch ein F77–Programm gerufen, darf dieser Wert nicht benutzt werden, da er durch FORTRAN automatisch benutzt wird.

5.3 DB_Delete()

Die Funktion `void DB_Delete(DB_Type *DataBase)` entfernt eine Datenbasis aus dem Speicher, die mir `DB_Create()` erzeugt wurde, für das entfernen einer Datenbasis, die mit `DB_CreateF()` erzeugt wurde, ist das rufende F77–Programm verantwortlich.

5.4 DB_Write() und DB_WriteF()

Die beiden Funktionen `DB_Write() und DB_WriteF()` erzeugen ein Standard–File und speichern darin den Inhalt der Datenbasis. Sie unterscheiden sich in ihrer Rufzeile, da die Behandlung von Zeichenketten in F77 und in C unterschiedlich erfolgt.

`DB_Write( char *outname,DB_Type *DataBase)`
öffnet eine Datei mit dem Namen, der durch `outname` bestimmt ist und speichert den Inhalt der Datenbasis `DataBase` als Standard–File in dieser Datei.

`DB_WriteF( char *outname,DB_Type *DataBase,int nlen)`
ist die entsprechende F77–Funktion, wird sie aus einer C–Umgebung aufgerufen, muß ihr in `nlen` die Länge der Zeichenkette `outname` übergeben werden, in einer F77–Umgebung darf dieser Wert nicht belegt werden, da er intern von FORTRAN benutzt wird.

5.5 DB_WriteP()

Die Funktion `DB_WriteP()` schreibt Teile der Datenbasis im Standard–File–Format in eine Datei, dabei kann eine Liste von Permutationsvektoren angegeben werden, die die Reihenfolge der Einträge innerhalb eines Blockes festlegt.

`DB_WriteP(FILE *outfile,DB_Type *DataBase,int part,DBPermVec plist)`
erwartet ein File `outfile`, das zum Schreiben geöffnet wurde. `part` gibt an, welcher Teil

der Datenbasis `DataBase` geschrieben werden soll. Folgende Werte sind möglich:

DB_STAT_INFO	1	Informationsteil
DB_STAT_HEADER	2	Header
DB_STAT_DATA	4	Datenteil
DB_STAT_END	8	Endemarkierung

Durch die bitweise oder–Verknüpfung lassen sich gleichzeitig mehrere Teile der Datenbasis schreiben.

`plist` kann eine Struktur enthalten, die Permutationsvektoren für die einzelnen Blöcke des Datenteils enthält. Wird als `plist` ein leerer Zeiger `NULL` übergeben, werden alle Datenblöcke in der Reihenfolge ihrer internen Nummern geschrieben. Kommt innerhalb der Struktur von Permutationsvektoren ein leerer Zeiger vor, so wird der entsprechende Datenblock in seiner internen Reihenfolge geschrieben. Die Benutzung von umgeordneten Datenblöcken kann zu einer Verbesserung der Eigenschaften von Netzen führen.

6 Die Programmschnittstelle data_read

Der zunehmende Einsatz der Standard–File–Struktur in Programmen der Forschergruppe führte zu einer Vielzahl von Unterprogrammen, die aus den Eingabefiles anwendungsspezifische Datenstrukturen erzeugen. All diese Routinen unterscheiden sich jedoch nur in wenigen Punkten, so daß es nahe liegt, ein Unterprogramm zu entwickeln, welches flexibel die verschiedenen Anforderungen der Anwendungen erfüllt, ohne daß Änderungen im Quelltext des Einleseprogramms nötig sind; der Routine wird lediglich ein Format–String übergeben, in dem die gewünschte Struktur der Daten beschrieben wird.

6.1 Grundlegende Arbeitsweise und Rufzeile

Das Unterprogramm `data_read` öffnet das Eingabefile und erzeugt auf einem Arbeitsvektor eine Datenbasis, die alle Infomationen des Eingabefiles enthält. Erst nach dem vollständigen Aufbau der Datenbasis wird der Format–String ausgewertet und die gewünschte Datenstruktur erzeugt. Der Arbeitsvektor muß also sowohl die Datenbasis als auch die gewünschte Datenstruktur aufnehmen können. Alle in der Datenbasis durch negative Vorzeichen markierten Nummern werden durch positive Nummern ersetzt. Damit ist es möglich, nutzerdefiniertre Typen zu benutzen. Traten beim Einlesen Datenfehler auf, so bricht die `data_read` ohnehin ab.

Die Rufzeile von `data_read` hat folgenden Aufbau:

```
data_read(filename,format,vektor,offset,counter,vlength,maxtoken,err
                                                   [,nlength,flength])
```

Die Parameter haben folgende Bedeutung:

`filename` ist ein Zeichenkette, die den Namen des Input–Files enthält. Wird `data_read` von einer C–Routine gerufen, so muß in `nlength` die Länge von `filename` als `int`–Wert übergeben werden, in FORTRAN **darf** dieser Wert **nicht** belegt werden, da dies automatisch geschieht.

`format` ist eine Zeichenkette, die den Aufbau der nutzerdefinierten Datenstruktur beschreibt. Der Aufbau des Format–Strings ist im Abschnitt 6.2 beschrieben. Wie bei `filename` muß die Länge der Zeichenkette beim Aufruf von `data_read` aus einer C–Routine separat als `int`–Wert in `flength` übergeben werden. Beim Aufruf aus einem FORTRAN–Programm darf dieser Wert natürlich **nicht** belegt werden.

`vector` ist ein `INTEGER*4`–Feld, das als Arbeitsvektor dient. Er nimmt sowohl die temporären Daten als auch die zu erzeugende Datenstruktur auf. Die verfügbare Feldlänge (gemessen in 4–Byte Blöcken) ist in `vlength` abgelegt.

`offset` ist ein `INTEGER*4`–Feld, daß den Beginn der Teilfelder der Datenstruktur im Abeitsvektor angibt. Der letzte aktive Eintrag des `offset`–Feldes bezeichnet den Beginn des ungenutzten Teiles des Arbeitsvektors.

`counter` ist ein `INTEGER*4`–Feld, in dem die Anzahl der Feldelemente der Teilfelder abgelegt wird. Der letzte aktive Eintrag des `counter`–Feldes enthält die Anzahl der Freiheitsgrade (`DOF`).

`vlength` ist eine `INTEGER*4`–Variable (Adresse) und gibt die Länge des Arbeitsvektors (in 4–Byte Blöcken) an. `data_read` belegt diese Variable mit der Länge des ungenutzten Arbeitsvektors.

`maxtoken` ist eine `INTEGER*4`–Variable (Adresse), die angibt, wieviele Token in `format` vorkommen dürfen. Sowohl das `offset`– als auch das `counter`–Feld muß jeweils `maxtoken`+1 Elemente aufnehmen können.
`data_read` belegt `maxtoken` mit der Anzahl der tatsächlich im `format`–String gelesenen Token plus eins. Damit ist eine einfache Adressierung des freien Arbeitsvektors über das `offset`–Feld möglich. Die Anzahl der Freiheitsgrade befindet sich entsprechend an der Stelle `maxtoken` im `counter`–Feld.

`err` ist eine `INTEGER*4`–Variable (Adresse), in der Fehlerzustände beim Einlesen und Konvertieren angezeigt werden. Fehlerfreie Bearbeitung liefert Null.

6.2 Format–Syntax

6.2.1 Beschreibung

Der Formatstring beschreibt den Inhalt, den Aufbau und damit auch die Länge der Datenfelder sowie ihre Anordnung auf dem Arbeitsvektor. Der Formatstring besteht aus einer Folge von Token, die jeweils mit einem Teilfeld auf dem Arbeitsvektor korrespondieren. Token werden durch ein Leerzeichen oder ein Komma voneinander getrennt. Jedes Token des Formatstrings bezeichnet genau ein Teilfeld, dessen Beginn relativ zum Arbeitsvektor im `offset`–feld abgelegt wird. Als Zähleinheit dient ein 4–Byte–Block (`INTEGER*4`, `REAL*4`).

Ein Token hat grundsätzlich die Form `<Bezug><Inhalt>[:<counter>]`. `<Bezug>` gibt dabei an, auf welchen Bereich des Eingabefiles sich `<Inhalt>` bezieht; die für `<Inhalt>` zur Verfügung stehenden Bezeichner sind also von `<Bezug>` abhängig. Token werden grundsätzlich von links nach rechts ausgewertet. Die Anzahl der Feldelemente in den einzelenen Teilfeldern entspricht der Anzahl der zugehörigen Datensätze im Eingabefile (`VertexCounter` usw.). Diese realen Feldlängen werden im `counter`–Feld abgelegt. Durch die Angabe von `:<counter>` kann eine fixe Feldlänge von `<counter>` Elementen erzwungen werden. Die Werte im `counter`–Feld werden dabei jedoch nicht verändert.

Für `U` (Freier Bereich) ist die Angabe von `:<counter>` notwendig, für alle anderen Token ist die Angabe optional.

Folgende Werte sind für `<Bezug>` möglich:

`<Bezug>`	im Std-File	Datenart	Standardlänge
U	User defined	Freier Bereich	—
V	VERTEX	Geometriedaten	DB_V_CNT
E	EDGE	Topologiedaten	DB_E_CNT
F	FACE	Topologiedaten	DB_F_CNT
S	SOLID	Topologiedaten	DB_S_CNT
R	REGION	Topologiedaten	DB_R_CNT
D	DIRICHLET	Randbedingungen	DB_D_CNT
N	NEUMANN	Randbedingungen	DB_N_CNT
M	MATERIAL	Materialdaten	DB_M_CNT
G	FACE_GEO	Flächengeometrie	DB_G_CNT

Den Aufbau eines Elementes des Ausgabefeldes legt `<Inhalt>` fest; `<Inhalt>` besteht entweder aus einem einzelnen Bezeichner oder einer in Klammern eingeschlossenen Bezeichnerliste. In einer Bezeichnerliste werden mehrere Bezeichner durch ein `+`–Zeichen veknüpft: `(bez1+bez2...)`. Mit Hilfe dieser Bezeichnerlisten ist es möglich, Felder zu erzeugen, deren Elemente komplexe Strukturen aufweisen. Der Unterschied zwischen einem einzelnen Bezeichner und einer Bezeichnerliste soll mit einem einfachen Beispiel verdeutlicht werden:

`Vx Vy Vz` liefert **drei** Felder, wobei das erste Feld die x–Koordinaten, das zweite die y–Koordinaten und das dritte entsprechend die z–Koordinaten enthält: $(x_1, \ldots, x_n)(y_1, \ldots, y_n)(z_1, \ldots, z_n)$.

`V(x+y+z)` dagegen liefert **ein** Feld, in dem die Koordinaten als Tripel angeordnet sind: $(x_1y_1z_1, \ldots, x_ny_nz_n)$.

6.2.2 Freier Bereich

Aufruf: `U:n[:m]`

Es wird ein Feld mit `n*m`–Elementen angelegt und initialisiert. Die Angabe von `:n` ist zwingend, `:m` ist optional (Default: 1).

6.2.3 Geometriedaten

Für Geometriedaten stehen folgende Bezeichner zur Verfügung:

`dummy[:n]` erzeugt einen freien Bereich von `n` 4–Byte–Blöcken. Der Parameter `:n` ist optional, default: `n=1`.

`name[:n]` erzeugt einen Eintrag für VERTEX–Namen, wobei jedem Namen ein freier Bereich von `n` 4–Byte–Blöcken folgt. `:n` ist optional, default: `n=0`.

`x|y|z[:n]` erzeugt einen Eintrag für `x`–,`y`– bzw. `z`–Koordinaten. Jedem Element folgt ein freier Bereich von `n` 4–Byte–Blöcken. `:n` ist optional, default: `n=0`.

`datab:k[:n]` erzeugt als Eintrag ein Feld mit der fixen Länge von `k` 4–Byte–Blöcken, die mit denjenigen Nummern der EDGES belegt werden, in denen das VERTEX–Element zur Definition benutzt wird. Nicht benötigte Feldelemente werden mit -1 markiert. Zu kleine Datenfelder führen zu einem Fehler und dem Abbruch der Konvertierung. Dem Datenfeld folgt ein freier Bereich von `n` 4–Byte–Blöcken. Die Angabe von `:k` ist notwendig; `:n` ist optional, default: `n=0`.

6.2.4 Topologiedaten

Die Datensätze der Topologieelemente (EDGE, FACE, SOLID und REGION) sind im Standard–File prinzipiell gleichartig aufgebaut. Diese Gleichartigkeit ermöglicht es, diese Daten mit Hilfe der gleichen Bezeichnersyntax aus der Datenbasis zu erhalten.

`dummy[:n]` siehe Abschnitt 6.2.3 (Geometriedaten)

`name[:n]` siehe Abschnitt 6.2.3 (Geometriedaten)

`type[:n]` erzeugt einen Eintrag mit dem Typ des Topologieelementes, gefolgt von einem freien Bereich von `n` 4–Byte–Blöcken. `:n` ist optional, default: `n=0`. Bei FACES stellt `type` einen Zeiger in das Feld der Flächengeometrien (FACE_GEO) dar.

`dataf:k[:n]` erzeugt als Eintrag ein Feld mit der fixen Länge von `k` 4–Byte–Blöcken, die mit den Datensätzen (d.h. VERTEX–, EDGE–, FACE– bzw. SOLID–**Nummern**) belegt werden, durch die das Topologieelement definiert wird. Nicht benötigte Feldelemente werden mit -1 markiert. Zu kleine Datenfelder führen zu einem Fehler und dem Abbruch der Konvertierung. Dem Datenfeld folgt ein freier Bereich von `n` 4–Byte–Blöcken. Die Angabe von `:k` ist notwendig; `:n` ist optional, default: `n=0`.

`datab:k[:n]` erzeugt als Eintrag ein Feld mit der fixen Länge von `k` 4–Byte–Blöcken, die mit denjenigen Nummern der FACEs, SOLIDs bzw. REGIONs belegt werden, in denen das Topologieelement zur Definition benutzt wird. Nicht benötigte Feldelemente werden mit -1 markiert. Zu kleine Datenfelder führen zu einem Fehler und dem Abbruch der Konvertierung. Dem Datenfeld folgt ein freier Bereich von `n` 4–Byte–Blöcken. Die Angabe von `:k` ist notwendig; `:n` ist optional, default: `n=0`.

6.2.5 Materialdaten

`dummy[:n]` siehe Abschnitt 6.2.3 (Geometriedaten)

`name[:n]` erzeugt einen Eintrag für den Namen der REGION, gefolgt von einem freien Bereich von `n` 4–Byte–Blöcken. `:n` ist optional, default: `n=0`.

`count[:n]` erzeugt einen Eintrag für die zur Verfügung stehende Anzahl von `FLOAT`–Daten zur Materialbeschreibung, gefolgt von einem freien Bereich von `n` 4–Byte–Blöcken. `:n` ist optional, default: `n=0`. `count` gibt also an, wieviele Daten aus `dataf` jeweils gültig sind.

`dataf:k[:n]` erzeugt als Eintrag ein Feld mit der fixen Länge von `k` 4–Byte–Blöcken, die mit den `FLOAT`–Werten der Materialdaten belegt werden. Zu kleine Datenfelder führen zum Abbruch der Routine mit einem Fehlerstatus. Sind nicht genügend Materialdaten im Ausgangsfile vorhanden, so werden die freien Stellen mit 0 belegt. Dem Datenfeld folgt ein freier Bereich von `n` 4–Byte–Blöcken. Die Angabe von `:k` ist notwendig; `:n` ist optional, default: `n=0`.

6.2.6 Flächengeometrie

`dummy[:n]` siehe Abschnitt 6.2.3 (Geometriedaten)

`name[:n]` erzeugt einen Eintrag für den Namen des Datensatzes, gefolgt von einem freien Bereich von `n` 4–Byte–Blöcken.

`type[:n]` erzeugt einen Eintrag für den Typ der zu beschreibenden geometrischen Form. Der Typ legt die Bedeutung (und damit die Anzahl) der Beschreibungdaten fest. Dem Typeintrag kann noch ein freier Bereich von `n` 4–Byte–Blöcken folgen.

`count[:n]` gibt an, wieviele Daten aus `dataf` jeweils gültig sind.

`dataf:k[:n]` erzeugt als Eintrag ein Feld mit der fixen Länge von `k` 4–Byte–Blöcken, die mit den `FLOAT`–Werten der Geometriebeschreibung belegt werden. Zu kleine Datenfelder führen zum Abbruch der Routine mit einem Fehlerstatus. Sind nicht genügend Daten im Ausgangsfile vorhanden, so werden die freien Stellen mit 0 belegt. Dem Datenfeld folgt ein freier Bereich von `n` 4–Byte–Blöcken. Die Angabe von `:k` ist notwendig; `:n` ist optional, default: `n=0`.

6.2.7 Randbedingungen

Zu jeder Fläche, auf der Randbedingungen definiert sind, existieren `DOF` Datensätze, wobei `DOF` die Anzahl der Freiheitsgrade angibt. Diese Datensätze enthalten einerseits den Typ der Randbedingung und andererseits eine (vom Typ abhängige) Anzahl von `FLOAT`–Daten. Der Zugriff auf diese Datensätze erfolgt über den Bezeichner `info`, der seinerseits wieder den Aufbau `info<RBinhalt>[:count]` besitzt. `<RBinhalt>` ist entweder ein einzelner Bezeichner oder eine Bezeichnerliste. `:count` gibt an, für wieviele Freiheitsgrade Platz reserviert werden soll. Fehlt diese optionale Angabe, wird der Wert von `DOF` aus dem Standard–File benutzt.

`dummy[:n]` siehe Abschnitt 6.2.3 (Geometriedaten)

`name[:n]` erzeugt einen Eintrag für den Namen der Fläche, gefolgt von einem freien Bereich von `n` 4–Byte–Blöcken. `:n` ist optional, default: `n=0`.

`info<RBInhalt>[:n]` Steuerung des Zugriffes auf die Datensätze der Randbedingungen. Es werden `n` Felder mit dem in `<RBInhalt>` festgelegten Inhalt erzeugt. Die Angabe von `:n` ist optional, der Defaultwert ist `DOF`. Ist `n` größer als `DOF`, werden nicht belegte `type`–Einträge mit 0 belegt.

Für `<RBInhalt>` stehen folgende Bezeichner zur Verfügung:

`dummy[:n]` siehe Abschnitt 6.2.3 (Geometriedaten)

`type[:n]` erzeugt einen Eintrag mit dem Typ der Randbedingung, gefolgt von einem freien Bereich von `n` 4–Byte–Blöcken. `:n` ist optional, default: `n=0`.

`dataf:k[:n]` erzeugt als Eintrag ein Feld mit der fixen Länge von `k` 4–Byte–Blöcken, die mit den `FLOAT`–Daten der Randbedingungen belegt werden. Die Anzahl der signifikanten Daten ist implizit durch `typ` gegeben. Zu kleine Datenfelder führen zum Abbruch der Routine mit einem Fehlerstatus. Dem Datenfeld folgt ein freier Bereich von `n` 4–Byte–Blöcken. Die Angabe von `:k` ist notwendig; `:n` ist optional, default: `n=0`.

Beispiele:
Sei `DOF=`2 und seien weiterhin Dirichlet–Randbedingungen für 3 Flächen gegeben.

`D(name+info(type+dataf:4))` liefert folgendes Feld:

```
                        (n (t (d d d d) t (d d d d))
                         n (t (d d d d) t (d d d d))
                         n (t (d d d d) t (d d d d)))
```

`D(name+info(type+dataf:4)):4` dagegen:

```
                        ( n (t (d d d d) t (d d d d))
                          n (t (d d d d) t (d d d d))
                          n (t (d d d d) t (d d d d))
                         -1 (0 (0 0 0 0) 0 (0 0 0 0)))
```

`D(name+info(type+dataf:4):3)` jedoch:

```
                        (n (t (d d d d) t (d d d d) -1 (0 0 0 0))
                         n (t (d d d d) t (d d d d) -1 (0 0 0 0))
                         n (t (d d d d) t (d d d d) -1 (0 0 0 0)))
```

`D(name+info(type+dataf:4:1))` erzeugt:

```
                        (n (t (d d d d) 0 t (d d d d) 0)
                         n (t (d d d d) 0 t (d d d d) 0)
                         n (t (d d d d) 0 t (d d d d) 0))
```

```
D(name:1+info(type+dataf:4)) (n 0 (t (d d d d) t (d d d d))
                         n 0 (t (d d d d) t (d d d d))
                         n 0 (t (d d d d) t (d d d d)))
```

6.2.8 Syntax Struktur

```
<Format>        := <Element> | <Element> <FTrenner> <Format>
<FTrenner>      := " " | ","
```

```
<Element>       := "U"<counter> | "V"<GArray>  | "E"<TArray> |
                   "F"<TArray>  | "S"<TArray>  | "R"<TArray> |
                   "D"<RBArray> | "N"<RBArray> | "M"<MArray> |
                   "G"<FGArray>

<GArray>        := <GElement><counter> | "(" <GList> ")"<counter>
<GList>         := <GElement>          | <GElement> "+" <GList>
<GElement>      := "dummy"<counter>    | "name"<dummy> |
                   "x"<dummy> | "y"<dummy> | "z"<dummy> |
                   "datab"<counter><dummy>

<TArray>        := <TElement><counter>   | "(" <TList> ")"<counter>
<TList>         := <TElement>            | <TElement> "+" <TList>
<TElement>      := "dummy"<counter>      | "name"<dummy> |
                   "type"<dummy>         | "dataf"<counter><dummy>
                   "datab"<counter><dummy>

<RBArray>       := <RBElement><counter>  | "(" <RBList> ")"<counter>
<RBList>        := <RBElement>           | <RBElement> "+" <RBList>
<RBElement>     := "dummy"<counter>      | "info"<IArray><dummy>

<IArray>        := <IElement><counter>   | "(" <IList> ")<counter>"
<IList>         := <IElement>            | <IElement> "+" <IList>
<IElement>      := "dummy"<counter>      | "name"<dummy> |
                   "dataf"<counter><dummy>

<MArray>        := <MElement>       | "(" <MList> ")"
<MList>         := <MElement>       | <MElement> "+" <MList>
<MElement>      := "dummy"<counter> | "name"<dummy> |
                   "dataf"<counter><dummy>

<FGArray>       := <FGElement>      | "(" <FGList> ")"
<FGList>        := <FGElement>      | <FGElement>"+" <FGList>
<FGElement>     := "dummy"<counter> | "name"<dummy>  |
                   "type"<dummy>    | "count"<dummy> |
                   "dataf"<counter><dummy>

<counter>       := ":"n
<dummy>         := <> |<counter>
```

6.3 Beispiele

Das Format

```
V(x+y+z) D(name+info(type+dataf:4)) N(name+info(type+dataf:4))\
                            Sdataf:4 Sdatab:1  Edataf:2:4 Fdataf:3:1
```

liefert sechs aufeinanderfolgende Teilfelder:

- Koordinatentripel;
- Dirichlet–Randbedingungen bestehend aus
  - FACE-Nummer gefolgt von einem Feld, daß
  - je Freiheitsgrad den Gleichungstyp und ein Datenfeld mit vier Einträgen enthält;
- Neumann–Randbedingungen mit gleichem Aufbau wie die Dirichlet–Randbedingungen;
- SOLID-Daten, bestehend aus vier FACE-Nummern;
- den REGION-Nummern der SOLIDS;
- EDGE-Daten, bestehend aus zwei VERTEX-Nummern und vier freien Worten.
- FACE-Daten, bestehend aus drei EDGE-Nummern und einem freien Wort

Die Länge der Teilfelder ergibt sich aus der Anzahl der im Eingabefile vorhandenen Daten und ist im `counter`–Feld der Rufzeile abgespeichert.

Der Formatstring

```
G(type+dataf:6)
```

Erzeugt ein Feld der Flächengeometriedaten (FACE_GEO), dessen Elemente aus dem Typschlüssel der Geometrie (Ebene, Zylinder etc.) und einem Datenfeld, welches bis zu sechs REAL*4–Zahlen aufnehmen kann, bestehen.

7 Sichtbare Namen

Folgende Namen sind global sichtbar:

Modul `data_read`:
`CreateStuct`, `data_read`

Modul `db2`:
`DB_load_head`, `DB_check_ver`, `DB_Recmpt_AVG`, `DB_cmpt_size`, `DB_Write`, `DB_WriteP`,

DB_execute_line, DB_BCTL, DB_ETL, DB_FTL, DB_load_dir, DB_load_edge, DB_load_face, DB_load_geo, DB_load_mat, DB_load_neu, DB_load_region, DB_Create, DB_load_solid, DB_load_vertex, DB_dump, DB_Dialog, DB_Error, DB_NoDialog, DB_use_conErr, NSL_free, DB_rename

Hilfmodule:
DL_fclose, DL_fflush, DL_fgets, DL_fopen, DL_fputs, bf_EOF, bf_EOL, bf_GetIntVec, bf_GetRealVec, bf_GetlongIntVec, bf_buf_free, bf_buf_used, bf_fprintf, bf_nputs, bf_stderr, bf_stdin, bf_stdout, bf_write_buff, close_bf, flush_bf, get_line_bf, get_log_line_bf, open_bf, show_bf, show_short_Bf, bf_vfprintf, check_mem, cpy_mem, enlarge_mem, free_mem, get_mem_heap, get_mem_stack, init_mem, release_mem, save_mem, show_mem, switch_mem, use_mem, ex_f77_open, ex_f77_fclose, ex_f77_fgets, ex_f77_fputs

Index