1 Motivation

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.

2 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.

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 gewünschten Datenstruktur beschreibt. Der Aufbau des Format–Strings ist im Abschnitt 3 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.

3 Format–Syntax

3.1 Beschreibung

Der Formatstring beschreibt den Inhalt, den Aufbau und damit auch die Länge der Datenfelder und 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 etc). 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	Vertex Count
E	EDGE	Topologiedaten	Edge Count
F	FACE	Topologiedaten	Face Count
S	SOLID	Topologiedaten	Solid Count
R	REGION	Topologiedaten	Region Count
D	DIRICHLET	Randbedingungen	Dirichlet Count
N	NEUMANN	Randbedingungen	Neumann Count
M	MATERIAL	Materialdaten	Material Count
G	FACE_GEO	Flächengeometrie	FGeo Count

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: $(\mathtt{x}_1\ ,\ \ldots,\ \mathtt{x}_n)(\mathtt{y}_1\ ,\ \ldots,\ \mathtt{y}_n)(\mathtt{z}_1\ ,\ \ldots,\ \mathtt{z}_n)$.

`V(x+y+z)` dagegen liefert **ein** Feld, in dem die Koordinaten als Tripel angeordnet sind: $(\mathtt{x}_1\mathtt{y}_1\mathtt{z}_1\ ,\ \ldots,\mathtt{x}_n\mathtt{y}_n\mathtt{z}_n)$.

3.1.1 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).

3.1.2 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`.

3.1.3 Topologiedaten

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

`dummy[:n]` siehe Abschnitt 3.1.2 (Geometriedaten)

`name[:n]` siehe Abschnitt 3.1.2 (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`.

3.1.4 Materialdaten

`dummy[:n]` siehe Abschnitt 3.1.2 (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`.

3.1.5 Flächengeometrie

`dummy[:n]` siehe Abschnitt 3.1.2 (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`.

3.1.6 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 3.1.2 (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 3.1.2 (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)))
```

3.2 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>
```

4 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
  - pro 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.