

ある書式記述言語の設計と作成

A NEW FORMAT DESCRIPTION LANGUAGE AND ITS IMPLEMENTATION

住田 宏己 小松 清 荒木 俊郎 都倉 信樹
HIROKI SUMIDA KIYOSHI KOMATSU TOSHIRO ARAKI NOBUKI TOKURA
大 阪 大 学 基 礎 工 学 部

FACULTY OF ENGINEERING SCIENCE, OSAKA UNIVERSITY

1. まえがき

文献データ、原稿データなどのデータベースの内容を出力する際、普通、汎用の手続き向き言語に頼っている。それらは、1行単位の書式、若しくはその繰り返しを指定するので、配列や固定長レコードの一次元的な出力書式としては便利であるが、多くのデータの出力位置を二次元上で相対的に指定する書式を記述するのは困難である。そこで二次元的な枠組みとして、ブロックという考えを用いて柔軟な書式を記述する言語FDL(Format Description Language)を設計した。

FDLの環境としてPASCALのレコード構造のファイルをもつデータベースの存在を仮定している。このときレコードの各フィールドが、さらにファイル構造、レコード構造となっていてよい。例えば題名、著者名などがそれぞれ一つのフィールドを構成し、本文については、各段落からなるファイルとして構成されているものと考えられる。この環境のもとに、文献データ或いは原稿データの為のある程度実的な書式を、手軽に記述できることを目標とした。

FDLでは同じデータベースに対する多くの異なる書式を記述することができる。また可変長のストリングを扱うことができ、出力幅などがデータに依存してもよいなど柔軟な書式を記述できる。またALGOL型言語を基本として設計し、ブロック構造をなしており、トップダウン的に記述できる。

本稿では、設計した言語の書式記述方法、並びに、データベースとのインターフェース部を除いたALGOL60へのトランスレータ作成を主に報告する。尚、FDLの構文図を付録に示す。

2. 記述方法

FDLでは名前は参照に先だって宣言しておく。名前の有効範囲はALGOLなどのブロック構造による制限に従うが、再帰的な参照はできない。特に項目名、ブロック名を参照する場合は、各々<>、[]で囲み、他の名前と簡単に見分けられるようにして用いる。また、いわゆる式を扱うことができ、オペランドには整数値、ストリング値、論理値を許され、演算子には加減乗算、比較、NOT, AND, ORが使える、結果の値として整数値、ストリング値、論理値のいずれかの値をとる。

FDLで用いる名前及び文の種類を図2.1に示す。

- 1 item identifier
- 2 continue identifier
- 3 constant identifier
- 4 integer identifier
- 5 itemposition identifier
- 6 blockposition identifier
- 7 block identifier
- 8 cblock identifier
- 9 item 文
- 10 block 文
- 11 assign 文
- 12 next 文
- 13 for-while 文
- 14 selection 文

図2.1 FDLで用いる名前及び文の種類

FDLによる完全な記述全体が、一つのブロックを、宣言している。一つのブロックの宣言の中で②～⑧の名前を宣言し、⑨～⑭の文によって参照する。

先ず基本となる、項目名、ブロック名、及びitem文、block文を説明し、続いて、その他の幾つかの名前、文について説明する。尚、④及び⑧も、各々項目、ブロックを表わすが、継続型という特殊な属性を持っており、これらについては後半で説明する。さらにパラメータを持ったブロックも宣言でき、これについても説明する。最後に2.11において、FDL文法上の幾つかの特徴を述べる。

2.1 項目 (item) について

FDLでは基本となるデータ (ストリング、整数) を項目と呼び、名前だけでデータベースとの対応をとる。例えば、文献カードの日付けの部分、著者名の部分、登録番号、題名、出典を図2.2の様な書式で出力させようとする。ただし、文献カードデータベースは図2.3の様に定義されていると仮定する。

図2.3のauthor, numberなどが項目を表わす名前である。cardfileは、number, date, title, source, member, authors をフィールドとする

```
*****
*
*                               77.9.1
* SINGER, A., HUERAS, J. & LEDGARD, H.
*
* *71      A BASIS FOR EXECUTING PASCAL
*           PROGRAMMERS
*
*           SIGPLAN NOTICES, Vol.12,
*           No.7, JULY 1977, PP.
*           101-105
*
*
*
*
*****
```

図2.2 文献カードの出力例

```
type      string = file of char ;
author = string ;
cardfile = file of record
number : string ;
date   : string ;
title  : string ;
source : string ;
member : integer ;
authors : file of author
end
```

図2.3 文献カードデータベース基本構造例

レコードからなるファイルとなっている。但し、memberは整数をもち、authorsはauthorのファイルであり、その他の項目はストリング値をもつとしている。

FDLでは用途から考え、扱う基本項目を、整数値、若しくは、アルファベット、空白、数字、記号からなるストリングの2種類としている。但し、それらをプログラム中の定数としても扱える。細かいストリング変換は、関数の形でのみ行ない、ストリング内の1文字ずつを直接には扱わないが、ストリング全体の比較は行なう。

2.2 ブロック, item文, block文

FDLではブロックと呼ぶ長方形を単位として書式を記述する。ブロックは入れ子構造をつくり、一つのブロックの中に他の幾つかのブロックを配置できる。外側のブロックからみて、その中に直接配置されるブロックを子ブロックと呼び、逆に、外側のブロックを親ブロックと呼ぶ。項目に対してブロックを割り当て、その大きさを指定することで、項目を出力する幅を指定する。またその位置を、既に配置が決まった他の同レベルのブロックや項目からの相対位置として指定できるし、親ブロック内での相対位置としても指定できる。

図2.2で示した文献カードの場合、文献によっては著者数、題名の長さなどに、大きな差があるが、FDLでは通常のどんな文献でも扱える書式を記述できる。図2.2の書式の記述例を一部省略して図2.4に示す。説明の為の行番号を添えておく。また、図2.4におけるブロックや幾つかの項目の配置の概略を図2.5に示す。但し図2.5では、ブロックを長方形で表わし、項目を<>で囲んで表わしている。

一つのブロックの宣言は、子ブロック等の宣言、ブロックの行数及び欄数の指定、並びに、ブロック内に項目や子ブロックを配置する文からなる。図2.4ではブロックcardは1枚のカードを表わしており、4～19行目、20～23行目、24～27行目で三つの子ブロックを宣言している。

block文は、既に宣言されているブロックを子ブロックとして配置する。図2.4では、33, 35, 36行目の三つのblock文が、各子ブロック、authorpart, titlepart, sourcepartを配置している。

```

1 item cardfile, number, date, member,
2   authors, author, title, source
3 block card
4   block authorpart
5   :
6   :
19  end { of author part }
20 block titlepart
21   begin size (flex * fix(30))
22   arrange( <title> )
23   end
24 block sourcepart
25   begin size (flex * fix(24))
26   arrange( <source> )
27   end
28 itemposition dt
29 blockposition ap,tp
30 begin { of card part }
31 size (fix(14) * fix(40))
32 (*+1, *+29)<date> = dt ;
33 (*+2, *+1) [authorpart] = ap;
34 (ap+1, *+2) "*" <number> ;
35 (ap,dt+1,*+9) [titlepart] = tp;
36 (tp,dt+1,*+15)[sourcepart]
37 end { of card part }

```

図2.4 文献カード書式の記述

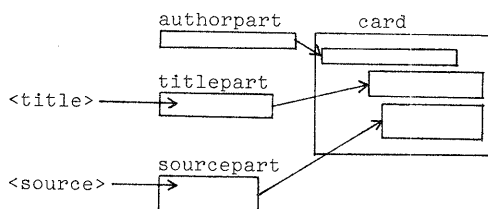


図2.5 図2.4の各ブロックの宣言と配置の概略

また、1～2行目で宣言している名前が項目名で、item文はこれらの項目及びストリング定数を配置する。図2.4では22、26、32、34行目でitem文を用い、各々、項目title、source、date、“*”及びnumberを配置している。尚、22、26行目のitem文はarrange関数を用いて配置しており、関数の働きによって、ストリングの空白部分で改行を行ない、端がそうように空白を配分して配置される。

図2.4のブロックcardのサイズは、31行目で14行40欄と指定している。ブロックの行数については、キーワードflexで指定して可変指定とすることができ、その場合は、ブロック内の全ての配置が終了した時点で最も下に配置された項目や子ブロックの最終行に依存して決まる。図2.4の20～23行目のブロックtitlepartでは、

21行目によって、行数を可変指定しており、項目titleの内容を全て配置した後、その最終行によってブロックtitlepartの行数が決まる。

2.3 位置指定

item文、block文の出力開始位置は、他の項目或いはブロックからの相対的な位置として指定される。親ブロックの上端又は左端は“*”で示される。キーワードlastitem、lastblockは、各々、それまでに親ブロック内に直接出力した項目の中の最後の項目、又は最後の子ブロックを示す。キーワードmaximumも同様にして、それまでに出力した項目、ブロックのうちで最も下方でさらに最も右方にあるものを示す。

2節の初めに述べた名前のうち、項目、ブロックの場所に付ける名前で相対位置を指定できるが、その名前は参照の前に、他の項目、又はブロックの場所と対応づけて記述しておく。

位置指定の方法を図2.4の例を用いて説明する。図2.4の32～34行目の位置指定の概略を図2.6に示しておく。

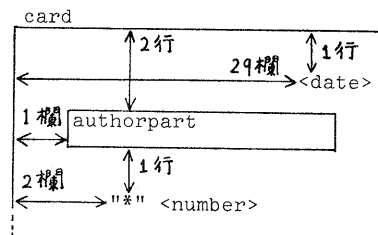


図2.6 図2.4のブロック、項目の相対位置指定の概略

図2.4の32行目のitem文は、親ブロックcardの上端から1行あけて、左端からは29欄あけて項目dateを配置する。33行目のblock文は、同様にして親ブロックの上端、左端からそれぞれ2行と1欄あけて、子ブロックauthorpartを配置する。また、dt、apは、各々28、29行目でitemposition、blockpositionの名前として宣言しており、項目、ブロックの配置された場所を参照する場合に用いる。32行目では項目dateを配置するitem文に続けて“=dt”と記述することによって、項目dateが配置された場所とdtとを対応づけられる。apも同様にして33行目でブロックauthorpartが配置された場所と対応づけられる。さらに34行目のitem文で、apが示している場所から1行あけて、親ブロックの

左端から2欄あけて“*”を配置している。また35行目では、2p, dtの示す場所のうちで、最も下の場所に対して1行あけてブロックtitlepartを配置している。

34行目の項目numberを配置するitem文、及び22, 26行目のitem文については、位置指定を省略している。item文については、このように位置指定を省略でき、それまでに出力した同レベルの項目と同じ行で、右隣の欄に続けて出力する。親ブロックの右端であれば改行される。また、22, 26行目の様に、それまでに項目を出力していなければ、親ブロックの左上の角から出力する。

2.4 next文、及びfor-while文

ファイル名に対してnext文が用意されている。この文の実行後は、ファイルの次の要素が扱われる。また、for-while文によって文の繰り返し記述される。

図2.4で省略したブロックauthorpartは、著者の数によって著者名の間にコンマや“&”を挿入する書式記述であり、図2.7に示して説明する。

図2.7の7~11行目が一つのfor-while文であり、whileに続く条件が真の間、コロンからendまでの文が繰り返し実行される。尚項目memberは著者数を表わす整数値をもっているとしている。9行目のitem文により、項目authorとコンマを出力している。next文は10行目の様にファイルの名前に対して記述し、ファイルの次の要素、この例ではauthorを読み込む働きをする。9~10行目が繰り返されるが8行目の条件により著者名のうち2人だけを残し、他はコンマを後に続けて出力される。

2.6 selection文

selection文を用いて、データに依存して、実行すべき文を選択できる。いわゆるif-then-elseとほぼ同じ制御構造をしている。図2.7では12~15行目、並びに16~18行目に、それぞれ一つのselection文を用いている。12行目のselection文により、著者数が1を越えていれば、即ち、あと2人残っていれば、13~14行目を実行して最後から2番目の著者名と“&”が出力される。16~18行目のselection文により、著者が1人でもいれば、17行目によって最後の著者名が出力される。

```

4  block authorpart
5      integer i
6      begin size ( flex * fix ( 38 ) )
7          for i=1 step 1
8              while i < <member> - 1 :
9                  <author> ; " , " ;
10                 next <authors>
11             end;
12
13         selection <member> > 1 :
14             <author> ; " & " ;
15             next <authors>
16         end;
17
18     selection <member> > 0 :
19         <author>
20     end
21 end { of author part }

```

図2.7 著者名の書式記述

```

1  item data
2  block dual
3      continue <data>
4
5      block page
6          block halfpage
7              begin
8                  size (fix(66) * fix(66))
9                  <data>
10                 end
11             begin { of page }
12                 size (fix(66) * fix(132))
13                 (*!0,*+0) [halfpage];
14                 (*!0,*+66) [halfpage]
15             end { of page }
16
17         begin { of dual }
18             size (flex * fix(132))
19             while rest( <data> ) :
20                 (lastblock+0,*+0) [page]
21             end
22         end { of dual }

```

図2.8 出力用紙を左右2頁とみなす書式記述

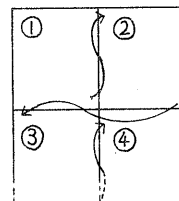


図2.9 図2.8の書式で出力する場合のラインプリンタ用紙上での出力順序

2.7 継続型項目

一つの項目は原則として親ブロックをはみ出さないように記述されるが、継続型として宣言することにより、二つ以上のブロックに継続して配置される。例えばラインプリンタ用紙1枚を左右2頁とみなして出力する書式を図2.8に

示し、ラインプリンタ用紙の上での出力順序を図2.9に示す。

図2.8の17～19行目によって、ブロック dual の中に、ブロック page を縦に繰り返して出力している。10～14行目によって、ブロック page の中に、ブロック half page を左右に並べて出力している。ブロック half page 自体は、5～9行目に示すように項目 data を出力するが、data は3行目で継続型項目として宣言されているので、以前に出力して残った部分から続いて出力される。data のストリング値を全て出力し終われば、17行目の rest 関数の値が偽となり、出力を完了する。

2.8 cblock

キーワード block で宣言される普通のブロックは親ブロックをはみだせないが、キーワード cblock で宣言されるものは、直感的には縦方向にかなり長いブロックとして、他の複数個のブロックに継続して配置される。但し、上下に幾つかのブロックに切断されるが、左右には切断されない。

単なるストリングでなく、論文データベースの内容を図2.9で示した様な順序で出力する場合を考える。FDLから見た論文データベースは、簡略して図2.10に示す基本構造をもっていると考えられる。即ち、データは、各要素が文章であったり数式であったりするようなファイルとする。要素の種類は flag という整数項目の値で区別される。図2.10のデータを出力する書式例を図2.11に示す。

図2.11では4～24行目で継続型ブロックを宣言している。継続型のブロックは、その宣言によってブロック内の全ての配置を終了した後、全体を幾つかのブロックに切断して配置する。配置の順序は継続型項目に似ており、それを参照しているブロック内に一杯になるまで出力した後、残った部分を、次に参照された時に継続して出力する。図2.11では27～28行目でブロック page の中に左右に並べて継続型ブロック long を出力している。先ず左側に出力した後、残っていれば、続きを右側に出力するという処理が、継続型ブロック long を全て出力し終わるまで繰り返される。尚、32行目の rest 関数によって、継続型ブロック long を全て出力し終わったかどうかをチェックしている。

```
type string = file of char ;
datafile = file of record
case flag : integer kind of
1 : ( paragraph : string ) ;
2 : ( expression : string ) ;
end
```

図2.10 論文データベースの基本構造

```
1 item datafile, paragraph,
2 expression, flag
3 block dual
4 cblock long
5 block para
6 begin size(flex * fix(60))
7 (*+0, **+1) <paragraph>
8 end
9 block exp
10 begin size(flex * fix(50))
11 <expression>
12 end
13 begin { of long }
14 size (flex * fix(66))
15 while rest( <datafile> ) :
16 selection
17 <flag>=1:
18 (lastblock+1, **+0) [para]
19 <flag>=2:
20 (lastblock+1, **+5) [exp]
21 end;
22 next <datafile>
23 end
24 end { of long }
25 block page
26 begin size(fix(66) * fix(132))
27 (*+0, **+0) [long];
28 (*+0, **+66) [long]
29 end
30 begin { of dual }
31 size (flex * fix(132))
32 while rest( [long] ) :
33 (lastblock+0, **+0) [page]
34 end
35 end
```

図2.11 論文データの書式記述

2.9 パラメータ

ブロックの宣言時に仮引数を持つことが許されている。ブロックの大きさ、配置すべきストリング及び子ブロックなどを、実際の出力の段階で与えることができる。またパラメータ対応は名前による呼び出し (call by name) で行なわれる。尚、仮引数を持って宣言されているブロックを、他のブロックの参照時に実引数として与える場合は、参照する側で前もって、その実引数となるブロックにとって必要な実引数を与えて記述される。

パラメータの使用例として、図2.12に、大き

```

block a
  block frame ( block : x ;
                integer : i,j )
    integer k
    begin size (fix(i+2)*fix(j+2))
      (*+1,*+1) [x];
      for k=1 step 1
        while k<=2*(i+j+2) : "*"
          end
        end
    end
  block b ( integer : i, j )
    begin size
      (fix(i) * fix(j))
    end
  integer i,j
begin size (flex * fix(80))
:
(*+5,*+5)[frame([b(i,j)], i, j)]
:
end

```

```

[a]
[
  [frame]
  [*****]
  [ * [b] * ]
  [ *      * ]
  [ *      * ]
  [ *      * ]
  [*****]
]

```

図 2.12 “*”による枠どりとその書式記述

さの分かっているブロックのまわりを“*”で囲む書式を示す。写真スペースの枠どりなどに利用できるが、ブロック frame の参照時に実引数となっているブロック b に対しても、必要な実引数を与えている。

2.10 文法上の特徴

2.9 までに述べた事柄の他に、幾つかの文法上の特徴を挙げる。FDL ではある親ブロックと、その中に配置される項目、子ブロックの位置関係は、子供が親に対する相対位置を指定する場合に限られ、ある親ブロック内での同レベルの子ブロックや項目の位置関係は、先に配置されたブロックや項目に対する相対位置を指定する場合に限られる。このようにして、相対位置関係が互いに相手に対して指定されてそれぞれの位置が定まらない事態が避けられている。

FDL における全ての名前は参照に先立って宣言されており、1-パスコンパイルができる。

さらに、LL(1)文法として設計している⁽¹⁾。その為、式の構文では、演算子とオペランドの組み合わせのチェックは行なわず、タイプチェックの方に委ねている。

3. 他の言語との比較

FDL では各項目単位で扱い、データの内容に対する細かい記述は行なわない。また、全ての書式を記述できるものでもなく、第1節で述べた幾つかの用途を前提に設計している。この節では、FDL を他の言語による書式記述と比較・検討する。

3.1 記述方法の比較

FDL の準備範囲に含まれている一つの書式を例にとり、他の言語による記述と比較検討する。項目 S にストリング値が入っており、ストリングの長さが必要と与えられるとする。この項目 S の内容を横幅と欄で出力する書式について検討する。横幅もデータとして与えられるとしている。

FDL では図 3.1 に示す様に、二つのブロックによって記述される。欄数、行数、ストリングの長さなどに柔軟性があり、S の長さが欄数以上の倍数になっているかどうかなどについて、プログラマは責任を持たなくてよい。いわばストリング出力用の書式記述になっている。

同じ書式を FORTRAN で記述すると、例えば、図 3.2 の様になる。但し、S は配列名として宣言されているとする。FORTRAN は行単位の書式には強いのだが柔軟性に欠ける。欄数がデータに依存しているので DO 文を用いている。ストリングの終わりでは欄より短くなる可能性があり、プログラマの責任で MIN 関数を用いて処理している。また、前の出力の次の位置を指定しにくい為、図 3.2 の様に出力並びに 1 行分を書くことが多い。

```

item i, s
block x
  block y
    begin size (flex * fix(<i>))
      <s>
    end
  begin size (flex * fix(<i>+3))
    (*+0,*+3) [y]
  end
end

```

図 3.1 FDL による書式記述例

```

10 FORMAT(4X,128A1)
DO 20 K=1, L, 1
  KK = MIN0 (K+1, L)
20 WRITE(6,10) (S(J),J=K, KK)

```

図 3.2 FORTRAN による書式記述例
(S は配列名)

```

for k:=1 to l do
if mod(k,i)>1 then write(s[k])
else if mod(k,i)=1 then write(' ':3,s[k])
else writeln(s[k])

```

図 3.3 PASCALによる書式記述例
(S は配列名)

```

PUT EDIT (S) (X(3),(I)A(1), SKIP);

```

図 3.4 PL/Iによる書式記述例
(S は配列名)

```

PUT EDIT ((SUBSTR(S,K,MIN(I,L-K+1))
DO K=1 TO L BY I))
(COLUMN(4), A(I));

```

図 3.5 PL/Iによる書式記述例
(S はストリング変数名)

図 3.3, 図 3.4に、同じく配列 S に対する書式を PASCAL, & PL/I で記述した例を示す。特に PASCAL では、この様な簡単な書式でも制御文を使わねばならない。もっとも、さらに複雑な書式になれば PL/I でも制御文が必要となり、それほど差は無くなる。いずれも、文字型配列を扱うのに便利な書式であり、PL/I の方が繰り返に強い表現を使えるし柔軟性も高い。

PL/I でもストリングを扱えるが、書式記述には図 3.5 に示す様に SUBSTR 関数を必要とし、配列を扱うほどには簡単ではない。改行、ストリングの終わりの行の処理など、プログラマが責任をもって細かく記述せねばならない。

3.2 記述の長さ並びにプログラミング時間の比較

図 3.1~3.5 の記述例からみれば、FDL で記述するより、PL/I などの方が記述の長さははるかに短い。この様な単純な 1 行分の書式を繰り返すだけか、それに多少の変化を付けた書式であれば、制御構造を考案するのにもたやすい。

しかし、さらに複雑になれば、例えば、二次元配列をとって一旦その中に並べた後に出力する方法をとることが多い。FDL では、プログラムはそのような配列が既にあるものとして記述できる。従って、記述の長さは必ずしも減るとは言えないが、プログラミングに要する時間は、他の言語による場合に比べ、大幅に短縮される。

4. トランスレータ作成

FDL では項目名で要求すれば、必要なデータを与えてくれるデータベースを想定しているが、本稿では、データの読み込み手続きを仮に作ることによってデータベースとの切り口とし、出力部分のみインプリメントしている。先ず短期間に作成することを第一目標とし、日本ミニコン NOVA3 エクステンディッド ALGOL へのトランスレータとして作成された。トランスレータ自身もこのエクステンディッド ALGOL によりプログラムされている。

作成には、いわゆるリカーシブディセセント (recursive descent) な方法を用いた。初めに拡張構文図を作成し、後に ALGOL で書き直したが、拡張構文図は論理上の虫をデバッグするのに大変有効であった。本節では、トランスレータ作成に当たっての留意点、並びに、本トランスレータの使用状況を述べる。

4.1 ブロック及びパラメータの実現

ブロック宣言を一つの手続き (procedure) の宣言と考え、block 文によってその手続きを呼び出すように作成された。また、名前による呼び出し (call by name) は、thunk⁽²⁾ を用いて実現された。

4.2 バッファリング

プログラムの最外部のブロックが表わしている領域を、実際に一つのファイルとして作り、一旦そのファイル上へデータを転送し、最後にファイルの内容をラインプリンタに出力している。実際、プログラム中の item 文のみが、データベースからの出力を要請しているので、各 item 文を、データベースから作業用ファイルへの転送手続きの呼び出しとして実現している。また、継続型ブロックも一つの書式記述プログラムに相当すると考えられ、それに対して一つの間中ファイルを作り、参照する block 文を、中間ファイルからプログラム全体を表わす作業用ファイルへ転送する手続きの呼び出しとして実現している。

4.3 二重打ち検出方法

FDL では二重打ちはエラーとみなされる。例えば、既に他の item 文が占めている場所を、同じレベルの他の item 文や block 文が占めようとしたり、既に block 文が占めている場所を、

同レベルの他の block 文が占めようとするエラーとみなされる。但し、既に block 文が占めている場所を、同レベルの item 文が占めようとした場合は、既にあるブロックの場所を飛び越し、親ブロックを出ない範囲で、次に空いている場所から出力されることになり、エラーとはみなされない。

このトランスレータでは、簡単のため実行時に、データを転送する各欄毎に、既に文字が書かれているかどうか、もし書かれていれば、同レベルの block 文によるものか、親ブロックより外の浅いレベルの block 文によるものか、同レベルの item 文によるものなのかを検査する。そこで各欄毎に、書かれた時の block 文又は、item 文のレベル数を保持している。この為、各欄について、ブロックの入れ子の深さの2倍のビットを必要とした。item 文、block 文の解釈について、上の制限を緩め、同レベルの item 文どうしも互に飛び越すとみなすなら、二重打ち検出の為に必要なビット数が半分で済む。少し複雑になるが、ここでさらにブロックの頭で二重打ちを検査するなら、各要素についてブロックの入れ子の深さに無関係に1ビットだけで済む。ASCII コード7ビットに対し、NOVA3の1バイトが8ビットであるから、メモリ節約には非常に有効な方法である。

さらに二重打ちはエラーでないと解釈し、積極的に利用することも考えられるが、禁止する場合の実現方法とは、大きな差がある。

4.4 トランスレータの構成

トランスレータの大きさは、ALGOL ソースプログラムの長さでみると、構文解析部が800行、コード生成部が1000行、ALGOL で書かれたランタイムルーチンが600行で構成されている。但し、構文解析部は、幾つかのパタンの反復として多少の重複も許して作成されている。主記憶32K語のNOVA3 RDOS下では大きなプログラムは走らないので、構文解析部とコード生成部とは別のステップとなっている。

```

item cardfile, number, date, member,
  authors, author, title, source
block list
  block unit
    integer i
    begin size (flex * fix(79))
      for i=1 step 1
        while i < <member>-1 :
          <author>; " , ";
          next <author>
        end;

        selection <member> > 1 :
          <author>; " & ";
          next <author>
        end;

        selection <member> > 0 :
          <author>
        end;
        " : ";
        arrange( <title> ); " , ";
        arrange( <source> )
      end
    integer i
    begin {of list} size(flex * fix(82))
      for i=1 step 1
        while rest( <cardfile> ) :
          (lastblock+1,*+0) char(1); " ) ";
          (lastblock+1,*+3) [unit];
          next <cardfile>
        end
      end { of list }

```

図4.1 参考文献書式の記述

4.5 データの出力例

実際にFDLで記述した文献カードの連続書式(図2.4、図2.7及び棒打ちをしてカード出力を繰り返す記述の組み合わせ)によってラインプリンタに出力したものを、既に図2.2に示した。この書式の完全な記述は61行で、翻訳処理時間は、約1分半であった。

同じデータベースに対し、異なる書式をFDLで記述して出力することもできる。図2.3に示した文献データベースに対し、図2.4とは異なる書式を図4.1に示し、それによる実際の出力例を図4.2に示す。

さらに、幾つか特定のカードのみ出力させれば、整数型項目、或いはストリング型項目の幾つかをコンソール入力に対応づけておき、

- 1) SINGER, A., HUERAS, J. & LEDGARD, H.: A BASIS FOR EXECUTING PASCAL PROGRAMMERS, SIGPLAN NOTICES, Vol.12, No.7, JULY 1977, PP. 101-105
- 2) AMMANN, U.: ON CODE GENERATION IN A PASCAL COMPILER, SOFTWARE PRACTICE AND EXPERIENCE, Vol.7, No.3, JUNE-JULY 1977, PP. 391-423

図4.2 図4.1の書式による出力例

記述中で、selection 文によって特定の登録番号、著者名、題名をもつ文献データを選択すればよい。或いは、データベース側からのコマンドによって、特定のデータをFDLプログラムに与えてやればよい。

5. あとがき

FDLの利用価値を高める為に、様々なストリング関数が考えられる。現在のインプリメントでも、arrange 関数(空白部分でのみ改行されるようにし、さらに、両端をそろえる様に空白を配分する)を組み込んでおり、第2節でその出力例も示した。その他にも、例えば下線を付けたり、1行の中央に出力させたり、字体を変えたり、拡張ローマ字カタカナ変換などの関数が考えられる。

FDLでは細かい制御は記述しない方針なので、脚注を含む書式などは記述しにくい。最初の目的は満たされ、統一的に長方形ブロックを配置するやり方で、あまり複雑でない論文書式などを、書式自体に柔軟性を持たせて手軽に記述できるものになった。言語設計及びインプリメントに、1.5人約1年を要したが、比較的早くインプリメントできたのは、ALGOLという強力な言語へのトランスレータという形式、並びに拡張構文図を利用したことによる。

但しこのインプリメントはオブジェクトのエクステンディッドALGOLプログラムにおいて、

固有の出力手続き及びファイル操作手続きを多く用いている。またアセンブリ言語など低級言語へ翻訳するインプリメントを考える時、FDL文法上の幾つかの点によって翻訳全体が複雑になるような箇所までは検討できていない。

データベースについては、我々の研究室でインプリメントされている他の言語(PASCAL)からも使える様な柔軟なシステム作りが進んでいる段階である。

謝辞 日頃より熱心に御討論いただいた本研究室諸氏、並びに、大阪府立大学の白川友紀博士に深く感謝致します。

参考文献

- (1) A.V.Aho and J.D.Ullmann: "The Theory of Parsing, Translation, and Compiling", Vol. I, Prentice-Hall (1972).
- (2) M.Griffiths: "Runtime Storage Management", in Compiler Construction (Eds. F.L.Bauer and J.Eickel) 2nd Edition, Springer-Verlag (1976).
- (3) 小松, 荒木, 白川, 都倉: "文献リストのフォーマット記述用言語", 昭52情報処理学会全国大会, 1 (昭52-10).
- (4) 住田: "ある書式記述言語の設計と作成", 阪大基礎工学部情報工学科特別研究報告 (昭53-3).

付録 FDLの構文図を次頁にまとめて示す。

FDL 構文図

