ある書式記述言語の設計と処理ルーチン†

住田宏己†† 小松 清†††
荒木俊郎†† 都倉信樹††

文献データや原稿データなどのデータベースの内容を出力する際，汎用の手続き向き言語では，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しにくい．そこで新しい書式記述言語 FDL (Format Description Language) を設計した．FDL は二次元的な枠組みとして統一的に長方形ブロックを配置するやり方で，柔軟な書式を記述できる．FDL は ALGOL 型言語を基本として設計された．また FDL の環境として，PASCAL でいうレコード構造のファイルをもつデータベースを仮定している．現在，ALGOL 60 へのトランスレータとして小型計算機 NOVA 3 上に作成されている．本論文では出力例を通して FDL による書式記述方法を説明し，FORTRAN など既存の言語の書式記述方法と比較する．

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もよい．例えば題名，著者名などがそれぞれ1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，文献データあるいは原稿データのためのある程度実際的な書式を，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もでき，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．さらに ALGOL 型言語を基本として設計したので，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．

本論文では，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フェース部を除いた部分の ALGOL 60 へのトランスレータ作成を主に報告する．なお FDL の構文図を付録に示す．

2. 記述方法

FDL で用いられる名前には項目名，ブロック名などがあり，これらは参照に先だって宣言しなければならない．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．

以下，例を通して基本となる，項目名，ブロック名，item 文，block 文を説明し，続いて，その他のいくつかの名前，文について説明する．最後に，FDL の文法のいくつかの特徴を述べる．なお FDL におけるキーワードとその意味を**表1**に示す．

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1**のよう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**のように定義されていると仮定する．

図2の author, number などが項目を表わす名前で

---

† A New Format Description Language and Its Implementation by HIROKI SUMIDA (Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University), KIYOSHI KOMATSU (Hitachi Ltd.), TOSHIRO ARAKI, and NOBUKI TOKURA (Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University).

†† 大阪大学基礎工学部情報工学科
††† 日立製作所

表 1 キーワード表
Table 1 Keywords table.

キーワード	意味
ITEM	項目名の宣言.
CONTINUE	継続型項目名の宣言.
BLOCK	ブロックの宣言.
CBLOCK	継続型ブロックの宣言.
CONSTANT	定数の宣言.
INTEGER	整変数の宣言.
STRING	ストリング変数（パラメータのみ）の宣言.
ITEMPOSITION	項目に対する相対位置指定に使うために，項目が配置された場所をさす名前の宣言.
BLOCKPOSITION	ブロックに対する相対位置指定に使うために，ブロックが配置された場所をさす名前の宣言.
SIZE	ブロックのサイズ（行数，欄数）の宣言.
FIX	ブロックの行数，欄数を，式によって陽に指定.
FLEX	ブロックの行数を，そのブロック内の項目，子ブロックの配置に依存して決める指定.
BEGIN…END	ブロックの本体（そのブロック内の配置を指定）.
MAXIMUM	ブロック内にそれまでに配置した項目，ブロックのうちで最も下方でさらに最も右方に配置されたものの配置場所をさす.
LASTITEM, LASTBLOCK	ブロック内にそれまでに配置されたもののうち，それぞれ最後に配置された項目，ブロックの配置場所をさす.
FOR…STEP… WHILE…END	ループを表わす．FOR 句で制御変数の初期化を示し，STEP 句で制御変数のループごとの増加値を示し，WHILE 句でループを繰り返す条件を示す.
SELECTION…END	CASE 文に相当する条件分岐を表わす.
NEXT	ファイルに対し，現在読み込んだ要素の次の要素を読み込む.
SAVER…OF…	整数値を整変数に代入する.
NOT, AND, OR	論理演算子.
REST	継続型項目や継続型ブロックに対して，その配置が終了していれば偽，まだ残っていれば真を返す論理関数.
{…}	コメントを囲む.

ある．cardfile は，number, date, title, source, member, authors をフィールドとするレコードからなるファイルとなっている．ただし，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．ただし，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，関数の形でのみ行い，ストリング内の1文字ずつを直接には扱わない．

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，1つのブロックの中に他のいく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

```
**************************************
*                                    *
*                          '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 1 文献カードの出力例
Fig. 1 An output of a library card.

```
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 文献カードデータベース基本構造例
Fig. 2 An example of the structure of library card data base.

幅を指定する．またその位置を，すでに同じブロック内に配置されている他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1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1の書式を記述した例を，1部省略して，説明のための行番号も添えて**図3**に示す．また，図3におけるブロックやいくつかの項目の配置の概略を**図4**に示す．ただし，図4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．

1つのブロックの宣言は，子ブロック等の宣言，ブロックの行数および欄数の指定，並びに，ブロック内に項目や子ブロックを配置するための item 文，block 文からなる．図3ではブロック card は1枚のカードを表わしており，4～16 行目，17～20 行目，21～24 行目で3つの子ブロックを宣言している．さらに 30, 32, 33 行目の3つの block 文でこれらの子ブロックを配置している．

また，1～2 行目で宣言している名前が項目名で，item 文はこれらの項目およびストリング定数を配置する．図3では 19, 23, 29, 31 行目で item 文を用

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

図 3　文献カード書式の記述

Fig. 3　The format of a library card in FDL.

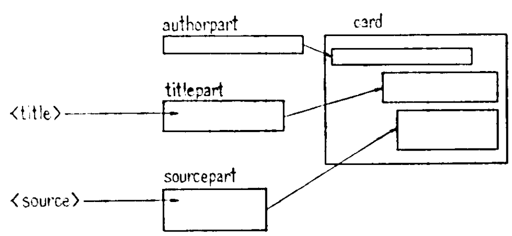


図 4　図3の各ブロックの配置の概略

Fig. 4　Block allocation for fig. 3.

い，各々，項目 title, source, date, number, およびストリング "*" を配置している．なお，19，23 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い，左端をそろえて配置する．

図3のブロック card のサイズは，28 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る．その場合，ブロックの行数はブロック内のすべ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3の 17～20 行目のブロック titlepart では，18 行目によって行数を可変指定しており，項目 title の内容をすべて配置した後，その最終行がブロック titlepart の最終行になる．

2.3 位置指定

item 文，block 文の出力開始位置は，他の項目，あるいはブロックからの相対的な位置として指定される．親ブロックの上端または左端は*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もののうち最後の項目，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．項目やブロックの配置場所をさす名前を使って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，またはブロックの配置場所と対応づけて記述しておく．

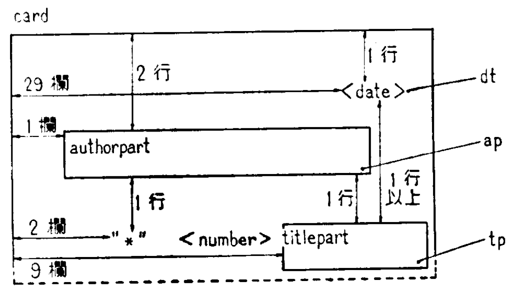


図 5　図3のブロック，項目の相対位置指定の概略

Fig. 5　Relatively specified positions of blocks and items for fig. 3.

位置指定の方法を図3の例を用いて説明する．図3の 29～32 行目の位置指定の概略を**図 5** に示しておく．図3の 29 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．また，dt, ap は各々 25, 26 行目で item-position, 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．29 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 とを対応づけている．ap も同様にして，30 行目でブロック authorpart が配置された場所と対応づけている．さらに 32 行目では，ap, 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．

31 行目の項目 number を配置する item 文，および，19，23 行目の item 文で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される．親ブロックの右端であれば改行される．また，19，23 行目のように，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される．

2.4 next 文，for-while 文および selection 文

next 文でファイルの要素を順に扱える．また，for-

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

図 6 著者名の書式記述

Fig. 6 The format of the block AUTHORPART of fig. 3 in FDL.

while 文で文の繰り返しが記述できる．図3で省略したブロック authorpart は，著者名の間にコンマや“＆”を挿入する書式記述であり，**図6**に示して説明する．

図6の 7～15 行目が1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．なお項目 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 を出力している．next 文は14行目のようにファイルの名前に対して記述し，ファイルの次の要素，この例では次の author を読み込む働きをする．8行目の条件により，9～15 行目が著者の数だけ繰り返される．

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．これは，いわゆる case 文とほぼ同じ制御構造をしている．図6では，10～13 行目の selection 文により，最後と最後から2番目の著者名以外はコンマを付けて出力され，最後から2番目の著者名は“&”を付けて出力される．

2.5 継続型項目

1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，2つ以上のブロックに継続して配置される．例として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7**に示し，ラインプリンタ用紙の上での出力順序を**図8**に示す．

図7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 halfpage を左右に並べて出力している．ブロック halfpage 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

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

図 7 出力用紙を左右2頁とみなす書式記述

Fig. 7 The format of double column output in FDL.

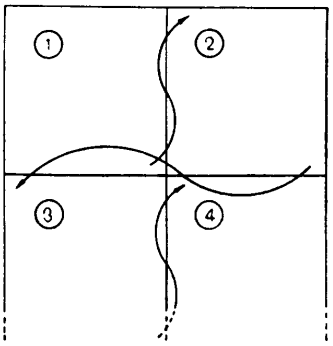


図 8 図7の書式で出力する場合のラインプリンタ用紙上での項目の継続順序

Fig. 8 The continuation of an item on the output by the format in fig. 7.

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 のストリング値をすべて出力し終われば，17 行目の rest 関数の値が偽となり，出力を完了する．

2.6 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考え，他の複数個のブロックに継続して配置される．ただし，上下には，いく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はなく，論文データベースの内容を図8で示したような順序で出力する場合を考える．ただし，FDL から見た論文データベースは，簡略して**図9**に示すように各要素が文章であったり数式

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

図 9 論文データベースの基本構造例

Fig. 9 An example of the structure of research report data base.

```
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 10 論文データの書式記述

Fig. 10 The format of a research report in FDL.

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9のデータを出力する書式記述例を図 10 に示す．

図 10 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すべての配置を終了した後に，全体をいくつかのブロックに切断して，図8に示した継続型項目の出力と同様な順序で配置されていく．

2.7 二重打ち禁止とスキップ

FDL では二重打ちはエラーとみなされる．すでに item 文や block 文が占めている場所を，同レベルの

```
      [frame]
**************************
*                        *
* [x]                    *
*                        *
*                        *
*                        *
*                        *
*                        *
*                        *
*                        *
**************************
   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
```

図 11 "*" による縁どりとその書式記述

Fig. 11 A block frame with "*" and its format in FDL.

他の block 文が占めようとするとエラーとみなされる．ただし，すでに 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，すで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あ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．図 11 のように，先に配置された写真スペースなどをスキップして項目を配置できる．

2.8 パラメータ

図 11 のようにブロックの宣言時に仮引数を持つことができる．したがって，ブロックの大きさ，配置すべきストリングおよ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．このパラメータ対応は名前による呼び出し (call by name) で行われる．なお，仮引数を持って宣言されているブロックを，他のブロックに実引数として与える場合は，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た形で，実引数リストに記述する．

2.9 文法の特徴

項目やブロックの位置は，親ブロックの左上角からの距離で指定するか，あるいは，先に配置された項目やブロックからの距離で指定するかの2通りの方法がある．したがって，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ような事態は避けられている．

FDL におけるすべ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[1]として設計している．そのため，構文解析が容易にできる．

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わない．また，すべての書式を記述できるものでもなく，第1節で述べたいくつかの用途を前提に設計している．この節では，FDL と他の言語とを比較，検討する．

3.1 記述方法の比較

FDL の守備範囲に含まれている1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 l で与えられるとする．この項目 S の内容を第4欄から横幅 i 欄で出力する書式について検討する．横幅 i もデータとして与えられるとする．

FDL では**図 12** に示すように，2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 i 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 13** のようになる．ただし，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が柔軟性に欠ける．ストリングの終わりでは i 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ため，図 13 のように出力並びに1行分を書くことが多い．

図 14，**図 15** に，PASCAL および PL/I で，同じ書式を記述した例を示す．特に PASCAL では，このよう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 の方が繰り返し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，書式記述は**図 16** に示すように 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 12〜図 15 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かなり短い．このような単純な1行分の書式を繰り返すだけであれば，制御構造を考案するのもたやすい．しかし，図8のようにさらに複雑になれば，例えば，二次元配列をとって，まず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，プログラマはそのような配列がすでにあるものとして記述できる．また COBOL などのように報告書作成機能のある言語もあるが，FDL の方がより柔軟に書式を記述できる．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，FDL の場合，相当短縮される．

```
item i, s
block x
    block y
        begin size (flex * fix (⟨i⟩))
            ⟨s⟩
        end
begin size (flex * fix (⟨i⟩+3))
    (*↑0, *← 3) [y]
end
```

図 12 FDL による書式記述例

Fig. 12 An example of FDL program.

```
10 FORMAT (4X, 128 A 1)
   DO 20 K=1, L, I
      KK=MIN 0 (K+I−1, L)
20 WRITE (6, 10) (S(J), J=K, KK)
```

図 13 FORTRAN による書式記述例 (S は配列名)

Fig. 13 An example of FORTRAN statements equivalent to format specifications in fig. 12 (S is array).

```
for k: =1 to l do
  begin
    if (k−1) mod i=0 then
      write (' ': 3);
      write (s [k]);
    if k mod i=0 then
      writeln;
  end
```

図 14 PASCAL による書式記述例 (s は配列名)

Fig. 14 An example of PASCAL statements equivalent to format specifications in fig. 12 (s is array).

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

図 15 PL/I による書式記述例 (S は配列名)

Fig. 15 An example of PL/I statements equivalent to format specifications in fig. 12 (S is array).

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

図 16 PL/I による書式記述例 (S はストリング変数名)

Fig. 16 An example of PL/I statements equivalent to format specifications in fig. 12(S is character string variable).

4. トランスレータ作成

4.1 トランスレータの構成

短期間に作成することを第一目標とし，日本ミニコン NOVA 3 エクステンディッド ALGOL へのトランスレータを作成した．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ッセント (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も大変有効だった．

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，構文解析部が 800 行，コード生成部が 1,000 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．ただし，構文解析部は，いくつかのパターンの反復として多少の重複も許して作成している．主記憶 32K 語の NOVA 3 の 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にした．

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 1 つのファイルとして作り，まずそのファイル上へデータを転送した後，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 1 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 1 つの中間ファイルを作り，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 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 3，図 6 および縁取り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，すでに図 1 に示した．この書式の完全な記述は 78 行で，翻訳処理時間は 90 秒，出力時間は 30 秒だった．

5. むすび

FDL の利用価値を高めるために，様々なストリング関数が考えられる．現在，arrange 関数（空白部分でのみ改行されるようにし，左端をそろえて出力する）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，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が，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．言語設計およびインプリメントが，1.5 人約 1 年という比較的短期間でできたのは，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．

ただし，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，固有の出力手続きおよびファイル操作手続きを多く用いている．

文献データベースは，本研究室で PASCAL からも使えるような柔軟なシステムの上で，実験的に作成されている．

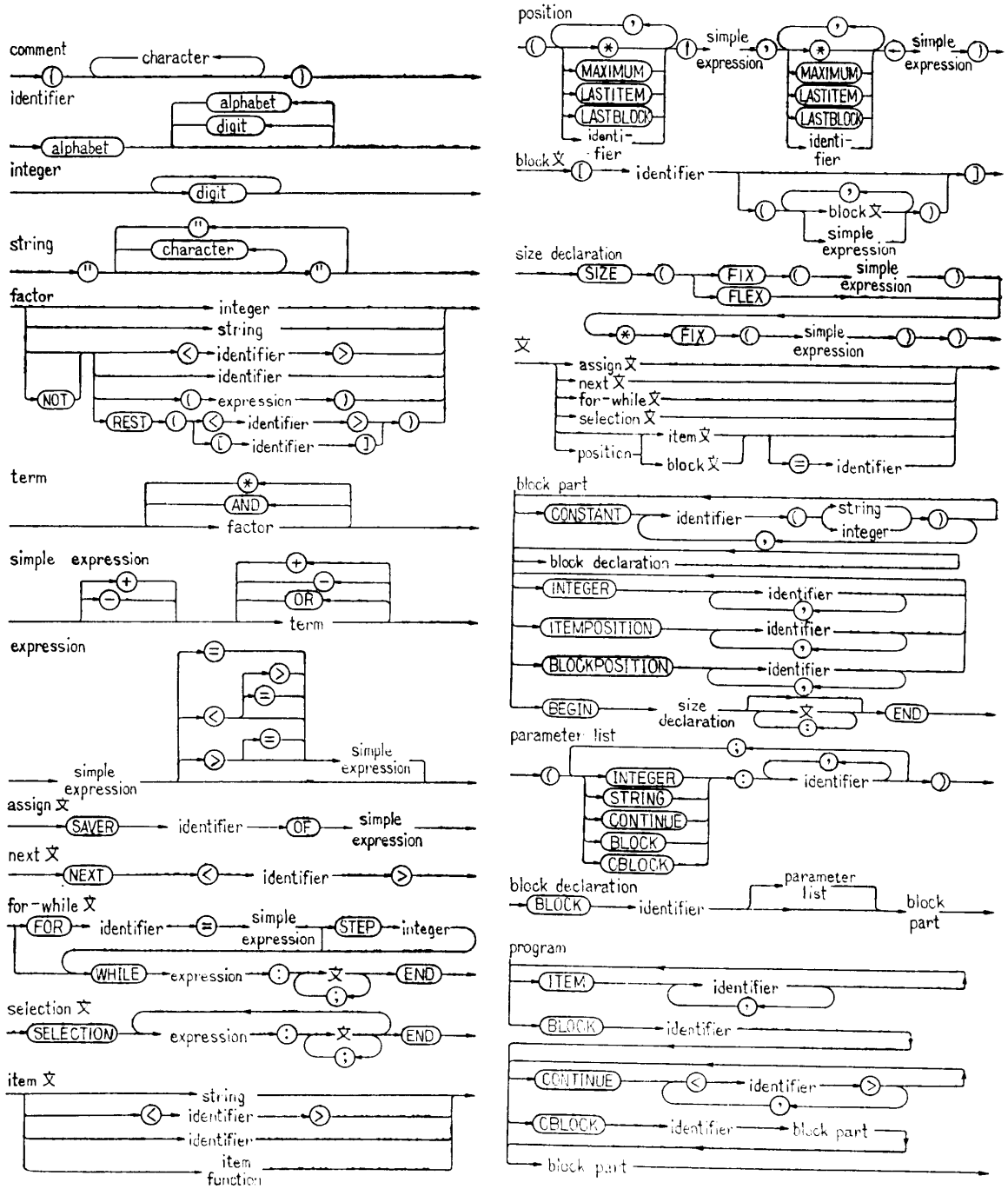
謝辞 日頃より熱心に御討論いただいた本研究室諸氏，並びに，大阪府立大学の白川友紀博士に深く感謝致します．

参考文献

1) Aho, A. V. and Ullmann, J. D.: The Theory of Parsing, Translation, and Compiling, Vol. 1, p. 542, Prentice-Hall (1972).
2) Griffiths, M.: Runtime Storage Management, in Compiler Construction (Eds. Bauer, F. L. and Eickel, J.) 2nd Edition, pp. 195-221, 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 情報処理全大，1 (1977-10).
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 (1978-03).
5) 住田，小松，荒木，都倉：ある書式記述言語の設計と作成，信学技報 EC 78-41，情報処理学会計算機アーキテクチャ研究会資料 33-12(1978-11).

(昭和 54 年 7 月 16 日受付)
(昭和 54 年 9 月 20 日採録)

付　録　FDL 構文図


comment
character
identifier
alphabet
digit
integer
string
factor
NOT
REST
expression
term
AND
simple expression
OR
assign 文
SAVER
OF
next 文
NEXT
for-while 文
FOR
STEP
WHILE
selection 文
SELECTION
item 文
item function
position
MAXIMUM
LASTITEM
LASTBLOCK
identifier
block 文
size declaration
SIZE
FIX
FLEX
文
block part
CONSTANT
block declaration
INTEGER
ITEMPOSITION
BLOCKPOSITION
BEGIN
END
parameter list
STRING
CONTINUE
BLOCK
CBLOCK
program
ITEM