A6-1　プログラム言語標準化の最近の動向（COBOL）

西村恕彦・植村俊亮（電子技術総合研究所）

1. アメリカ規格COBOLの改訂

１９６８年にアメリカ規格協会（American National Standards Institute；以下ANSIと略称する）は最初のアメリカ規格COBOL USA Standard COBOL，X3．23-1968を制定した．この規格は原案の段階でISOにも提出され，１９７２年にISO Recommendation R 1989 Information Processing－Programming Language COBOLとなった．

国内では，日本工業規格電子計算機プログラム用言語COBOL（ＪＩＳ Ｃ ６２０５）が１９７２年８月に制定された．これは上記のアメリカ規格や国際推薦規格案にもとづいた内容であるが，アメリカ規格の登場から約４年の時間遅れがあったことになる．

この間に，ANSI内でCOBOLを担当する技術委員会X3J4は１９６９年３月に「X3.23の保守を行なう」と作業目標を修正し，以後定期的に会合をつづけて改定にそなえてきた．当初X3.23－１９６８の改訂時期は１９７２年中ということであったが，すこし遅れて，１９７２年７月にX3J4段階での改訂が完成し，X3J4 内での投票を終った．この改訂案はすでにBEMAから出版され，１９７２年末日にX3における４ケ月の「コメント期間」を終る．さらにANSIにおける６週間の「出版コメント期間」（両者を並行して行なうらしい）を経て，１９７３年前半中に改訂版のアメリカ規格COBOLが制定されることになる．

本稿ではこの改定案の概要を紹介する．１９６８年の規格制定時にはＸ３の投票の段階で乱処理（random processing）機能が規格本文からはずされた（付録にまわった）．これと同様のことは今回も十分に起りうるので，今回の紹介はあくまでも改定案についてのものである．

なお現在のＪＩＳ COBOLには，参考としてCODASYL COBOL，１９６９との相違が注記してあり，これをたどると，以下に紹介する改定点がかなり示されていることがわかる．

2. 全般的にみた改訂点

全体として大幅の改訂である．規格の機能単位別の構成はこれまでどおりであるが，まったく新しい機能単位，拡張改訂された機能単位，水準分けに変更のあった機能単位，技術的内容はほぼ同じだが表現がすっかり書き改められた機能単位などさまざまであって，X3.23－１９６８とほとんどかわっていないのは区分化の機能単位だけぐらいなものである．図1に新しいアメリカ規格COBOL案の構造図を示す．

各機能単位別におもな変更点をひろってみると次のようになる．

中核――NOTE，REMARKSが削除された．そのかわりに注記行（第７桁に＊を書く）ができた．

ＳＩＧＮが追加された．

ＥＸＡＭＩＮＥ命令が拡張されてＩＮＳＰＥＣＴ命令になった．

ＡＣＣＥＰＴ命令に新しい書き方ができた．

ＳＴＲＩＮＧ命令，ＵＮＳＴＲＩＮＧ命令が追加された．

そのほかいろいろ．

<table>
<tr><td rowspan="2"></td><td colspan="10">機　　能　　単　　位</td></tr>
<tr><td>中　　核</td><td>表 操 作</td><td>順 編 成</td><td>相対編成</td><td>索引編成</td><td>分類組合せ</td><td>報告書作成</td><td>区 分 化</td><td>登 録 集</td><td>手 直 し</td><td>プログラム間連絡</td><td>通　　信</td></tr>
<tr><td rowspan="2">2中核1,2</td><td rowspan="2">2表操1,2</td><td rowspan="2">2順編1,2</td><td>2相対0,2</td><td>2イン0,2</td><td>2分併0,2</td><td>1報告0,1</td><td>2区分0,2</td><td>2登録0,2</td><td>2手直0,2</td><td>2連絡0,2</td><td>2通信0,2</td></tr>
<tr><td rowspan="2">1相対0,2</td><td rowspan="2">1イン0,2</td><td rowspan="2">1分併0,2</td><td rowspan="2">空</td><td rowspan="2">1区分0,2</td><td rowspan="2">1登録0,2</td><td rowspan="2">1手直0,2</td><td rowspan="2">1連絡0,2</td><td rowspan="2">1通信0,2</td></tr>
<tr><td rowspan="2">1中核1,2</td><td rowspan="2">1表操1,2</td><td rowspan="2">1順編1,2</td></tr>
<tr><td>空</td><td>空</td><td>空</td><td>空</td><td>空</td><td>空</td><td>空</td><td>空</td></tr>
</table>

図1　新しいアメリカ規格COBOL案の構造図

表操作――これまでの水準2，水準3がそれぞれ新しい水準1，水準2になった．表操作は2つの水準だけになったわけである．低いほうの水準でもSET命令のすべての書き方と3次元の表とを保障している．

入出力関係の機能単位――これまでは「順呼出し」と「乱呼出し」とにわかれていたが，今回はファイル編成法によって「順編成（Sequential I-O）」，「相対編成（Relative I-O）」，「索引編成（Indexed I-O）」の3機能単位にわかれた．ファイル呼出し法には，順呼出し，乱呼出し，動的呼出しの3つがある．動的呼出しでは，順呼出しと乱呼出しの切替えを実行時に（ダイナミックに）行なえる．

分類組合せ――MERGE命令が追加された．

報告書作成――これまでの水準分けをけとめて1つの水準にしてしまった．すなわち空か水準1かだけになった．文法表現が全面的に改訂された．

区分化――優先番号（priority－number）という用語が区分番号（segment－number）に変った．オーバレイ可能な固定区分は論理的には固定区分と同じであることがはっきりした．

登録集――内容が拡張改訂されて全面的にかわった．

新しく追加された機能単位――プログラムの手直し（Debug）機能，プログラム間の連絡（Inter－program Communication）機能，通信（Communication）機能の3つが新たに追加された．入出力関係のうち，索引編成機能もまったく新しい機能単位といえよう．

また規格票の様式もかわった．これまでは各機能単位の各水準別に章をたてていたが，繰返しがおおくなるので，今回は各機能単位別にだけ章をわけている．1つの機能単位は1回だけ規定される．水準分けはわくでかこって示してある．わくでかこまれた部分をとばして読むと，水準1の規定になる．全部通して読むと水準2の規定になる．目をとじてしまうと空水準だと思われる．

以下の章ではすべての改訂点をつくすことはとうてい不可能であるから，おもな点を列記することにする．

3. 中核（Nucleus）のおもな改訂点

3.1　正　書　法

全体に制限がゆるくなった．終止符（．），コンマ（，），セミコロン，右かっこの前に空白を置いてもよい．左かっこのあとに空白を置いてもよい．表操作でも表の要素を呼ぶときに，添字や指標を並べた間にコンマを書いても書か

なくてもよい．左かっこの左に空白がなくてもよい．次の命令の書き方は正しいことになる．

```
MOVE  20000  TO  KYUUYO(  BU  KA  ) .
```

レベル番号７７の項目を作業場所節の先頭にまとめなくてもよい（０１記述項の間に７７があってもよい）．

NOTE，REMARKSが削除されて，かわりに注記行（Comment line）ができた．プログラムの第７桁目に星印（＊）を書くとその行は注記行になる．注記行は翻訳時に印刷されるだけで，翻訳はされない．プログラムの先頭と最後の行以外はどこにでも注記行を書けるので，環境部，データ部にも注をいれることが可能になった．注記行の特別の形として，第７桁目に斜線（／）を書いた行は，ページ送りをしてから印刷される．なお第７桁目にＤと書くと手直し行になる．これについてはのちにふれる．

節名だけからなる節，段落名だけからなる段落も書けるようになった．

3.2　データ記述項の改訂

数字項目の演算符号の取扱いを拡張するＳＩＧＮ句ができた．数字項目のＰＩＣＴＵＲＥの文字列にＳが含まれている場合，このＳＩＧＮ句を書かなければ（つまり従来どおりにしておけば），演算符号の存在は確保されるが，符号の位置や表現はすべて作成者まかせになる．桁数はかぞえないらしい．　ＳＩＧＮ　ＩＳ　ＬＥＡＤＩＮＧ　（あるいはＴＲＡＩＬＩＮＧ）句を書くと，その数字基本項目の左端（あるいは右端）の数字位置に符号が開連づけられる．符号をどう表現するかは作成者がきめるが，桁数にはかぞえない．ＬＥＡＤＩＮＧ（あるいはＴＲＡＩＬＩＮＧ）に加えてＳＥＰＡＲＡＴＥ　ＣＨＡＲＡＣＴＥＲＳと指定すると，数字位置の左側（あるいは右側）に演算符号の桁位置が１桁分できる．正号は＋，負号は－で表現され，もちろん桁数にかぞえる．

```
05  SUUZI  PIC  S9(4)  SIGN  LEADING  SEPARATE.
```

は５桁の数字基本項目である．＋３０００などと穿孔したカードを読み込めるようになったわけである．しかも計算機外部のデータ中の符号に関して互換性が保障されるのはこの場合だけになった（！）．計算機内部では，ＰＩＣＴＵＲＥの文字列にＳがありさえすれば，ほかの指定の有無にかかわらず必要な変換が自動的に行なわれる．

ＰＩＣＴＵＲＥの文字列中で単純挿入記号として，斜線（／）をコンマ（，）やＢや０と同じように使える．なお数字項目の大きさは数字の桁数にして１ないし１８桁に制限された．

ＲＥＤＥＦＩＮＥＳ（再定義）句の制限がゆるくなって，次のような再定義ができることになった．

```
10  CARD.
    20  SEIGYO      ...
    20  HONTAI      ...
10  MOZI  REDEFINES  CARD  OCCURS  40.
    20  EIZI                          PIC  X(2).
    20  SUUZI  REDEFINES  EIZI        PIC  9(2).
```

ＳＵＵＺＩをＥＩＺＩに再定義している部分はこれまでは文法違反だった（ＭＯＺＩにＯＣＣＵＲＳ句があるから）が，これからは正しくなる．ただしＥＩＺＩの各反復要素に対して一律に再定義するのであるから，最後の行を

```
20  SUUZI  REDEFINES  EIZI  (7)  PIC  9(2).
```

などと書いてはいけない．このＲＥＤＥＦＩＮＥＳ句の作用対象には添字も指標もつけてはならない．

3.3　命令の改訂

中核には，通信機能の導入にしたがって，STRING命令，UNSTRING命令ができた．

```
UNSTRING   TUUSINBUN
           DELIMITED   BY   "*"   OR   "／"
           INTO   MIDASI
                  HONBUN   DELIMITER   IN   KUGIRI
                           COUNT   IN   ZISUU
           WITH   POINTER   KETA-ITI
           ON   OVERFLOW   GO   TO   AHURE.
     ⋮
STRING   MIDASI   "／"   HONBUN
         DELIMITED   BY   SIZE
         INTO   OKURIDASI-BUN.
```

EXAMINE命令はなくなり，拡張された形のINSPECT命令になった．INSPECT命令ではTALLYING句によってtallyカウンタを自前で宣言しなければならない．1字以上の長さの文字列で桁しらべができる．

```
INSPECT   MOZI-RETU
          TALLYING   TAL-A   FOR   LEADING   "0"
                     TAL-B   FOR   ALL   SUUZI
          REPLACING   LEADING   "0"   BY   "*".
```

算術演算関係の命令（COMPUTE，ADD，SUBTRACT，MULTIPLY，DIVIDE）の書き方で，結果を入れる項目を複数個指定できるようになった．

```
COMPUTE   KEKKA-A   KEKK-B   KEKKA-C   ...
          =   ...
```

ACCEPT命令には，システムから日付け，時間などを受け取る書き方ができた．GO　TO命令のTOや，EQUAL　TOのTOなどは書いても書かなくてもよい．

4．表操作（Table handling）の改訂点

まず水準分けがさきに述べたように変更された．表操作の水準1は最低水準COBOLに含まれなければならないから，この規格が制定されれば，最低水準のCOBOLでもSET命令のすべての書き方や第3次元の表（JIS規格での表操作水準2）が保障される．

DEPENDINGつきのOCCURS句の規則がだいぶ明確になった．OCCURS　S　TO　100　TIMESのようにTOを含むOCCURS句には，かならずDEPENDINGを書かなければならない．しかしDEPENDINGによって反復回数を可変にできる表は1つのレコードの最後の部分になければならない．まん中にこのような表を含むレコードは許されない．そして反復回数が可変の表を最後に含む集団項目を参照したときは，最大の大きさではなくて，そのとき実際に使われている部分だけが参照される．これによって可変長の表の転記の結果がかなりはっきりした．

5. 順編成，相対編成，索引編成

5.1 入出力関係に共通の改訂

CODASYLのプログラム言語委員会におけるラベルについてのCOBOL文法の改訂がおくれたので，新アメリカ規格（案）ではラベル関係の書き方がバッサリと整理されてしまった．すっきりしたが，これで大丈夫かという疑問も残る．まずLABEL RECORDはOMITTEDかSTANDARDかのどちらかだけになった．USE 命令でもLABELに関する書き方は削除されてしまった．VALUE OF句は，

VALUE OF 作成者のきめた名前 IS ...

としか書けなくなった．つまり利用者独自のラベルは作れなくなった．

USE命令が水準1でも使えることになった．

USE AFTER ERROR PROCEDURE ON ファイル名－1

と宣言すると，ファイル名－1のファイルの入出力操作時に誤りが生じた場合のみならず，AT END 句のないREAD命令でAT ENDになったときにも，このUSE手続きが実行される．すなわち適当なUSE 手続きを宣言しておけば，READ命令のAT END句やINVALID KEY句は書いても書かなくてもよい．上のERROR PROCEDUREのかわりにEXCEPTION PROCEDUREと書いても意味は同じである．

入出力関係の命令を実行した結果の情報を自動的に収めるデータ項目をFILE STATUS句で指定できる．

文法の説明のために，現在レコード指示子（current record pointer）という概念上の項目ができたが，有効に役立っているとは思えない．

5.2 ファイル編成法と呼出し法

ファイルの編成法と呼出し法とは，SELECT記述項で指定する．

```
SELECT   ファイル名
         ASSIGN   TO   ...
         ORGANIZATION   IS  { SEQUENTIAL
                              RELATIVE
                              INDEXED    }
         ACCESS   MODE  IS  { SEQUENTIAL
                              RANDOM
                              DYNAMIC    }
         ...
```

ACCESS MODE句を省略すると，順（SEQUENTIAL）と想定される．ACCESS MODEが動的（DYNAMIC）の場合には，実行途中で順呼出しと乱呼出し（RANDOM）とを切り替えて使えるが，どちらの呼出し方式によっているかということは，READ命令にNEXT句を書くかどうかできまる．

```
READ   R－FILE   RECORD.          }
 ⋮                                 } ←―― 乱呼出し
READ   R－FILE   RECORD.          }
 ⋮
READ   R－FILE   NEXT   RECORD.   ←―― 順呼出し
 ⋮
```

```
READ    R−FILE    RECORD.          ←――― 乱呼出し
  ⋮
```

乱呼出しによって目的とするレコードをさがし出して，そのあとは順呼出しを行なうという仕事はいろいろあるから，便利な機能といえよう．

動的呼出し法を指定しても，WRITE命令については乱呼出し法とまったく同じ扱いをする．出力の場合には，キーの値にしたがって乱呼出し法で書き出していって，途中から順呼出し法に切り替えるなどということが，まったく無意味だからであろう．

5.3 順編成（Sequential I−O）

順編成ファイルは順呼出ししかできない．ファイルとその呼出し法の概念はこれまでの順呼出しの機能単位と同じである．もちろん大記憶装置をこの編成法で使ってもよい．

すでにあるファイルのうしろにレコードを追加していくためのEXTEND（拡張）という句ができた．

```
OPEN    EXTEND    MUKASI−NO−FILE.
```

ラベルつきのファイルをこの書き方で開くと，終りラベルを削除してくれて，そのあとへつぎつぎにレコードを追加していってくれる．順編成のファイルでありさえすればよいのだが，磁気テープ装置だと奮闘しなければならないのではなかろうか．

I−O（入力出力）指定でOPENした順編成ファイルに対してはREWRITE（書換え）命令が使える．REWRITE命令は，直前のREAD命令で読み込んだレコードの内容を書き換える．ファイル中でのそのレコードの値が新しくなる．

順編成ファイルのWRITE（出力）命令はラインプリンタへの印刷用にも使われる．印刷用のWRITE命令にADVANCING（行送り）句がないときは，AFTER 1とみなされることがはっきりした．規格案作成の最後の段階までもめたあげくに，WRITE命令のEND−OF−PAGE（ページの終り）句とFD記述項のLINAGE（行数）句とが含まれることになった．

```
FD    INSATU−YOO−FILE
      LABEL    RECORD    OMITTED
      LINAGE    54    LINES
        FOOTING    51
        TOP         6
        BOTTOM      6.
```

この機能にはいろいろ問題があって，このままで普及するとは考えにくい．

5.4 相対編成（Relative I−O）

相対編成のファイルでは，ファイル全体が論理レコード用の領域に1つずつあらかじめ分割されていて，各領域に先頭から1，2，…，nの番号がついていると概念上考えればよい．領域についている番号を相対レコード番号という．したがって，ファイルを開く時点で，このファイルに書き出す最大のレコードの大きさがきまってないといけない．

レコードが入っている領域もあるし，空の領域もある．いったんレコードが入った領域でも，DELETE（消去）命令によってそのレコードを削除すると空の領域になる．このファイルを順呼出し法で読み込んでいくと，空の領域は自動的にとばして読む．順呼出し法で書き出していくと，番号1，2，3，…の領域につぎつぎに書き出す．乱呼出

し法か動的呼出し法で処理するときは，ファイルのSELECT記述項にあらかじめ相対キー項目を宣言しておいて，このキー項目に相対レコード番号を与えることによって，読み書きするレコードを指定する．

```
SELECT   SOOTAI−FILE   ASSIGN   TO   ...
         RELATIVE   KEY   IS   BANGOO
           ⋮
```

BANGOOは相対キー項目で，たとえばここに値８を入れてWRITE命令を実行すると，相対レコード番号８の領域にレコードが書き出される．相対キー項目はレコード記述の中にあってはいけない．空の領域の相対レコード番号を指定してREAD命令を実行したり，すでにレコードが入っている領域のレコード番号を指定してWRITE命令を実行したりすると，無効なキー（ INVALID KEY ）状態になる．

動的呼出し法では，乱呼出しを行なっている間は相対キー項目に相対レコード番号を入れてやらなければならないが，順呼出しに切り替えたときはその必要がない．

相対編成では次の索引編成と同様に，DELETE命令，REWRITE命令のほかに，START（位置ぎめ）命令がある．START命令はファイル中で入出力処理を開始する位置を指定できる．

```
START   SOOTAI−FILE
        KEY   IS   GREATER   BANGOO.
```

BANGOOは相対キー項目でなければならない．もしBANGOOの内容を１００にしておいてこの命令を実行すれば，相対レコード番号が１００より大で１００にいちばん近いレコードからファイルの処理が始まるように位置づけられる．

5.5　索引編成（ Indexed　I−O ）

索引編成のファイルでは，各レコードに１つまたは複数個のキー項目が含まれていて，このキー項目の値によってレコードを呼び出す．

```
SELECT   SAKUIN−FILE   ASSIGN   ...
         ...
         RECORD   KEY   SYU−KEY
         ALTERNATE   RECORD   KEY
           HUKU−KEY   DUPLICATE.
```

SYU−KEYを主キー（ prime record key ）といい，ファイル全体を通じてこの項目の値の同じレコードが複数個あってはいけない．またいちど与えた値をファイル中でみだりにかぞえてはいけない．

主キーに適当な値を設定しておいて，これを手がかりとしてレコードを識別するのが，標準の方法である．順呼出し法では，主キーの値の昇順にレコードが呼び出される．キーとレコードとは「索引（ index ）」を介して対応づけられると規定されている．

索引編成ファイルには，副キー（ alternate record key ）という画期的な機能が含まれることになった．SELECT記述のALTERNATE RECORD KEY句で指定する項目を副キーという．さきの例ではHUKU−KEYが副キーである．こちらは，主キーとはちがって，DUPLICATE 句を書いておきさえすれば，値の同じレコードがいくつあってもよい．そして副キーに適当な値を設定しておいて，これを手がかりとしてレコードを識別することもできるのである．副キーはいくつでも指定できるから，レコードを呼び出すキーが複数個あることになる．レコードを従業員番号をキーとして呼び出すことができるだけではなくて，生年月日をキーとして呼び出したり，給料をキーとして呼び出したりできるわけである（ただしキーは英数字項目でなければならない）．

主キーと副キーとがある場合に，実際にその時に呼出し用に使っているキーを参照キー（ key of reference ）と

いう．ふつうにREAD命令を書けば，主キーが参照キーになり，レコードは主キーを介して呼び出される．KEY句をつけてREAD命令を書けば，指定したキーが参照キーになり，レコードは指定したキーを介して呼び出される．

```
READ    SAKUIN−FILE    KEY    IS    HUKU−KEY.
```

ではHUKU−KEYが参照キーになる．動的呼出し法によって乱呼出しから順呼出しへ切り替えたときは，その時点での参照キーにしたがって順呼出しが行なわれる．順呼出し法では，キーの値の昇順にレコードが呼び出されるのだから，参照キーがちがえば呼出しの順もまったくちがうものになる．

はじめからACCESS SEQUENTIALと指定したファイルでは，主キーが参照キーである．またどんな呼出し法を指定した場合でも，WRITE命令は主キーを介して行なわれる．

システム作成の立場からみると，参照キーの切替えはREAD命令にKEY句を書くかどうかによって任意に行なえるから，それぞれのキーごとに独立した索引を作るだけでは不十分で，索引相互間の関係をなんらかの形でつけておかなければならないと考えられる．もちろん索引をかならず使わなければならないわけではない．リスト処理の技法は有力な候補であろう．実際に，「索引編成」機能単位の文法中に「索引（index）」という用語はわずかに3回現われるだけなのである．

6．分類組合せ

SORT（分類）命令とよく似た形式のMERGE（組合せ）命令ができた．

```
MERGE    ファイル名−1    ON { ASCENDING  } KEY  ...
                            { DESCENDING }
   USING    ファイル名−2，    ファイル名−3，    ...
   { GIVING    ファイル名−5                }
   { OUTPUT    PROCEDURE    ...            }
```

SORT命令のINPUT PROCEDURE中でRELEASE命令を使ってレコードを引き渡してゆく手続きそのものが，つまりは組合せの操作に相当するから，MERGE命令にはINPUT PROCEDUREがない．OUTPUT PROCEDUREとRETURN命令はあり，組み合わせの途中でそのつどレコードを1つずつ受け取る．ファイル名−1のファイルは「組合せ用ファイル」であって，分類用ファイルと同様にSD項を使って記述するのだが，この「組合せ用ファイル」がなぜ必要なのかよくわからない．RETURN命令でファイル名−1を参照するぐらいのもので，実際には存在する必要のないファイルである．しかしこれもSELECT句でちゃんとどれかの装置にASSIGNしてやらなければならない．

なおSORT命令のUSING句にもファイル名を列記できることになった．GIVING句に書くファイル名のファイルに対するSELECT句ではASSIGN A OR Bという書き方があったが，これは削除された．

7．登録集（Library）機能の大拡張

CODASYL COBOLが登録集機能を大拡張したのにともなって，新しい規格案の登録集機能もあっという間に拡張されてしまった．

まず第1にCOPY命令をプログラム中のほとんどどこにでも書ける．ソースプログラム中で文字列（character string）か分離符（separator）の書けるところなら，どこに書いてもよい．文字列というのは，COBOLの語，定数，PICTUREの文字列，注記項（*）のいずれかであるから，

```
IDENTIFICATION    COPY    NOKORI.
```

と書いてもよいし，

```
MOVE   X   TO   Y   (Z   COPY   MIGI-KAKKO.   .
```

でも許される．ただ1行だけの

```
COPY   HITO-NO-PROGRAM.
```

でも1つのCOBOLプログラムらしい．

第2に登録集が複数個あってもよくなった．このときはCOPY命令に書く原文名（ text－name ）を登録集名（ library－name ）で「修飾」しなければならない．

```
      ⋮
COPY   MIDASI-BU   OF   MIDASI-TOOROKU-SYUU.
      ⋮
COPY   TETUZUKI-BU   OF   TETUZUKI-TOOROKU-SYUU.
      ⋮
```

第3に，これまでは定数，一意名，語のうちのどれかの置換えしかできなかったが，擬似原文（ pseudo－text ）によって文章の置換えが可能になった．文字列や分離符の列を擬似原文分離符（ // ）で区切ったものを擬似原文という．たとえば

```
COPY   KYUUYO-SYORI   OF   ZATUMU-TOOROKUSYUU
       REPLACING      //MOVE   70000   TO   TEATE//
       BY             //MOVE   77000   TO   TEATE//.
```

などと書ける．

以上のように拡張された登録集機能で，実際の複写は次のような制限のもとに行なわれる．まず登録集の原文はCOBOLの正書法の規則に従っていなければならない．そして置換え後の原文の語は正書法の規則に従ってソースプログラムに配置される．すべてのCOPY命令を完全に処理し終るまでは，そのプログラムが文法的に正しいかどうかを判定できない．COPY命令の中にCOPY命令があってはいけないし，COPY命令によって複写された原文の中にCOPY命令があってもいけない．

新しい文法であるから，なおいくつかあいまいな点が残っている．とくにCOPY命令を書く位置と複写される原文の置かれる位置との間の関係があいまいである．原文中には注記行や手直し行（ 9.参照 ）を書けるのだが，COPY 命令はどうがんばっても第7桁目からは書けない．複写が実際に始まるのはCOPY 命令のCの字を書いた位置からであろうか，それとも原文の内容によってつごうのよい位置からの複写が考慮されるのだろうか．B欄にCOPY命令を書いておいても，原文がA欄から始まっていればA欄から複写が行なわれるという説もある．この機能の拡張を担当したPLCの委員が，複写できる内容の自由化と，COPY命令を書ける位置の自由化とを混乱してしまったのでは ないかとも考えられる．

なお，擬似原文分離符は次のような場合に混乱を生じるので，ほかの記号に置きかえることが検討されている（ 等号2つ＝＝が候補にあがっているようである ）．

```
COPY   BUN   REPLACING   //PIC   XX//.
```

8．プログラム間の連絡（Inter－Program Communication）機能単位

ＪＩＳ規格原案作成の段落でこの機能を追加してほしいという意見が出され，議論をかさねたすえに追加を見合せたいきさつがある．規格には新しく追加されたわけであるが，これに相当する機能をもつコンパイラはかなりある．

1つの仕事を行なうプログラムをいくつかの小さな単位に分割して，それぞれを独立したプログラムとして作り，コンパイルし，修正する場合（いわゆるプログラムのモジュール化）にこの機能を使う．それぞれ独立してコンパイルされたプログラムの間で，データの受け渡し（データ領域の共有）と制御の移行を行なわせることが必要になる．

CALL命令によって，あるプログラムから別のプログラムへ制御を移す．CALL命令を含むプログラムを「呼ぶプログラム」，CALL命令の対象になるプログラムを「呼ばれるプログラム」という．いったん呼んだプログラムはCANCEL命令で取り消すまでは，呼ぶプログラムと論理的なつながりをたもっている．むろん呼ばれるプログラムCOBOLで書かれているとはかぎらない．

CALL命令では，PROGRAM－ID段落に書いたプログラム名を使うのだが，文字定数として直接書く方法と，一意名を書く方法とがある．

```
  ⋮
CALL   “HUKU－PROGRAM”.
  ⋮
CALL   YOBIDASI－MEI.
  ⋮
```

後者のCALL命令では，実行直前までにデータ項目YOBIDASI－MEIに，呼びたいプログラムの名前を転記しておけばよい．呼びたいプログラム名を外部のデータ（たとえば穿孔カード）から読み込むことができるわけである．

データの受け渡しはLINKAGE SECTIONを介して行なう．呼ぶプログラム中でレベル番号が77または01（報告書節はのぞく）のデータ項目はどれでも呼ばれるプログラム中で使える．まず呼ぶプログラム中のCALL命令に，

CALL プログラム名 USING A B C.

というふうに，引き渡すパラメータを書く．A，B，Cは呼ぶプログラム中のデータ項目である．呼ばれるプログラムの手続き部の見出しにもUSING句を書いて，

PROCEDURE DIVISION USING X Y Z.

とすると，両方のプログラムの間でAとX，BとY，CとZがそれぞれ対応づけられる．X，Y，Zは呼ばれるプログラム中のLINKAGE SECTIONに記述しておく．この節に記述した項目は，そのプログラム中には実体がなくて，実際に呼ばれたときにはじめて対応する項目の記憶場所に割り付けられる．パラメータについてはいくつかの制限がある．両方のUSING句のパラメータの個数は同じでなければならない．AとXの例のように，対応するデータ項目の名前は異なっていてもよいし，それぞれのデータ構造が異なっていてもよいが，同じ大きさでなければならない．CALL命令のUSING句中には，同じデータ項目をなん度でも指定できるが，手続き部見出しのUSING句中では，同じデータ項目を繰り返し書いてはいけない．

この機能によってプログラムのモジュール化がきわめて容易になった．区分化機能と併用すると，大混乱が生じる可能性がある．

9. プログラムの手直し（Debug）機能

新しく追加された機能である．コンパイルは無事に終了したが思いどおりに働いてくれないプログラムの論理的なまちがいを手直しするときに助けになる．いわゆる追跡ルーチン（tracer）が文法の一部に組み込まれたものと考えられる。手直し行（Debugging line），手直し節（Debugging Section），手直し機能を動作させるためのスイッチなどがある．

ソースプログラムの第7桁目にDと書いた行は手直し行になる．たとえばSUUZIというデータ項目の内容が絶対に0にならないはずなのに，計算結果を見るとどうも0になることがあるらしい，それを確かめてみたいとしよう．DISPLAY SUUZIなどという行を適当な場所に挿入するとよいが，挿入する場所がおおいときは，結果を確かめたあとで各行を削除するのがたいへんめんどうである．そこで挿入したい各行の第7桁目にあらかじめDと書いて，手直し行にしておく．手直し行は，SOURCE-COMPUTER段落にWITH DEBUGGING MODE句が書いてあれば，ふつうのプログラムの行とみなされてコンパイル（アセンブル）される。WITH DEBUGGING MODE句がないときは，手直し行は注記行とみなされる．だから手直しが完了すれば，WITH DEBUGGING MODE句をけずるだけでよい．実際に各行を削除しなくてもよい．

```
SOURCE-COMPUTER.    ANS1973  WITH  DEBUGGING  MODE.
```

手直し行はOBJECT-COMPUTER段落よりはあとにだけ書ける．ふつうの命令の行のみならず，データ項目を記述した行でも，手続き名の行でも，なんでも手直し行にできる．ただし手直し行を注記行とみなしてもみなさなくても，つねに文法的に正しいプログラムになるようにしなければならない．

USE命令を手直し用に使う機能もある．

```
PROCEDURE  DIVISION.
DECLARATIVES.
TENAOSI  SECTION.
   USE  FOR  DEBUGGING  ON  SUUZI.
SUUZI-TUISEKI.
   IF  SUUZI  =  0  DISPLAY  DEBUG-ITEM.
END  DECLARATIVES.
SYORI-KAISI  SECTION.
SYKS.
   ⋮
```

USE FOR DEBUGGING ... と宣言した節を手直し節という．

プログラム実行中に，SUUZIが陽に参照されてその内容がかわった（replaced）ときに，TENAOSI SECTIONが実行される．USE命令のほかはどんな手続きを書いてもよいが，上の例ではSUUZIの内容をしらべてその値が0のときにDEBUG-ITEMを印刷させている．DEBUG-ITEMは手直し節中でだけ呼べる特殊レジスタで，その手直し節（上の例ではTENAOSI SECTION）を実行させる原因となった行を知るてだて，データ名，データ項目の値などが自動的に入っている．

USE FOR DEBUGGING命令では，一意名のほかにファイル名，手続き名，通信記述領域名を指定できる．指定したものが「参照」されたときに手直し節が実行される．「参照」の意味はかなりこまかく規定されている．SOURCE-COMPUTER段落のWITH DEBUGGING MODE句は手直し節に対してもまったく同じ働きをする．

COBOLプログラムのそとには実行用スイッチが用意されている．WITH DEBUGGING MODE句とともにコンパイルされたプログラムを実行するときに，実行用スイッチがオンであれば，プログラムはそのまま実行される．オフであれば，実行時に手直し節の部分の効果が抑制される．手直し行の効果はそのままである．

全体にかなりまとまった機能単位で，あまりおおきな騒動はまき起さないと思われる．USE命令による手直しがどの程度普及するかには疑問が残る．WITH DEBUGGING MODE句の挿入削除はいささかめんどうである．

10. おわりに

改訂作業の途中で，COBOL文法をきめる組織であるCODASYLのプログラム言語委員会に，「報告書作成機能をCOBOLから削除せよ」という提案が出されておおきな反響を呼んだが，審議に入るまえに提案そのものが取りさげられてしまって，結局もとのままになった．新アメリカ規格では，CODASYL COBOLにしたがって報告書作成機能の文法表現を全面的に改訂したうえで，全体を1つの水準にまとめている．おもな改訂内容については，「コボルPLCの活動」（植村），情報処理学会誌，1971年2月号に紹介ずみである．

通信機能は，この機能をCOBOLに組み込んだ張本人であるHISIのHamさん（現PLC議長）が 規格化にあまり乗り気でないと伝えられていたが，あっさりと規格（案）に含まれてしまった．この機能全体については「データ通信とCOBOL」（西村），bit，1970年2月号に紹介がある．

（参 考 資 料）

Draft Proposed Revised X3.23 American National Standard Specifications for COBOL

BEMA／STANDARDS

1828 “L” Street，N.W.

Washington，D.C. 20036

（価格6ドル，マイクロフィッシュなら3ドル）

本 PDF ファイルは 1965 年発行の「第 6 回プログラミング―シンポジウム報告集」をスキャンし、項目ごとに整理して、情報処理学会電子図書館「情報学広場」に掲載するものです。

この出版物は情報処理学会への著作権譲渡がなされていませんが、情報処理学会公式 Web サイトの https://www.ipsj.or.jp/topics/Past_reports.html に下記「過去のプログラミング・シンポジウム報告集の利用許諾について」を掲載して、権利者の捜索をおこないました。そのうえで同意をいただいたもの、お申し出のなかったものを掲載しています。

過去のプログラミング・シンポジウム報告集の利用許諾について

情報処理学会発行の出版物著作権は平成 12 年から情報処理学会著作権規程に従い、学会に帰属することになっています。

プログラミング・シンポジウムの報告集は、情報処理学会と設立の事情が異なるため、この改訂がシンポジウム内部で徹底しておらず、情報処理学会の他の出版物が情報学広場 (=情報処理学会電子図書館) で公開されているにも拘らず、古い報告集には公開されていないものが少からずありました。

プログラミング・シンポジウムは昭和 59 年に情報処理学会の一部門になりましたが、それ以前の報告集も含め、この度学会の他の出版物と同様の扱いにしたいと考えます。過去のすべての報告集の論文について、著作権者（論文を執筆された故人の相続人）を探し出して利用許諾に関する同意を頂くことは困難ですので、一定期間の権利者捜索の努力をしたうえで、著作権者が見つからない場合も論文を情報学広場に掲載させていただきたいと思います。その後、著作権者が発見され、情報学広場への掲載の継続に同意が得られなかった場合には、当該論文については、掲載を停止致します。

この措置にご意見のある方は、プログラミング・シンポジウムの辻尚史運営委員長 (tsuji@math.s.chiba-u.ac.jp) までお申し出ください。

加えて、著作権者について情報をお持ちの方は事務局まで情報をお寄せくださいますようお願い申し上げます。

期間：2020 年 12 月 18 日 〜 2021 年 3 月 19 日
掲載日：2020 年 12 月 18 日

プログラミング・シンポジウム委員会

情報処理学会著作権規程
https://www.ipsj.or.jp/copyright/ronbun/copyright.html