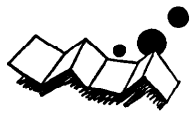
解　説



COBOLの標準化の動向†

今城　哲二††

1. はじめに

COBOLの特徴，歴史及びCODASYL COBOLの最近の動向については本誌第22巻第6号[1]に解説済みである．本稿では，第2章でCODASYLと規格の発展について触れ，第3章以降で1981年9月に提案された米国の第3次規格原案について詳細に紹介する．

2. CODASYLと規格の発展

COBOLはCODASYLが1959年12月に開発した．それから23年間コンピュータの進歩とともに発展してきており，その名前(Common Business Oriented Language)に背くことなく事務用共通言語としてのトップの地位を不動のものとしている．

CODASYLは今も休むことなく積極的に言語仕様の改良作業を行っており，COBOL言語仕様の正式報告としてCODASYL COBOL開発報告 (Journal of Development—JOD)[2]を1～3年ごとに発行している．各開発報告で追加された代表的な機能を**表-1**に示す．

CODASYLは複数のメーカとユーザで構成された公平な組織であるが，任意団体でしかない．また言語仕様の変更も多い．このため，仕様が固定した公式な標準が必要である．COBOLの標準化作業は米国規格協会が担当している．

表-1　CODASYL COBOLの発展

JOD発行年度	代表的な追加機能
1963	整列 (sort)，報告書作成 (report writer)
1965	大記憶ファイル(乱呼出し)，表操作
1968	プログラム間連絡
1969	通信，文字列操作
1970	デバッグ，併合 (merge)
1973	大記憶ファイルの大幅改訂 (相対，索引)
1976	データベース，ビット操作
1978	整構造プログラミング，プログラム間連絡の強化
1981	数学関数，2進・内部10進・浮動小数点データ

第1次規格は，1967年1月1日現在のCODASYL COBOLの完全な部分集合になっており，米国規格——ANS(1968年)，国際推薦規格——ISO(1972年)，日本工業規格——JIS(1972年)などになった．

第2次規格は，1971年12月31日現在のCODASYL COBOLの完全な部分集合になっており，米国規格(1974年)，国際推薦規格 (1978年)，日本工業規格(1980年)[3]などになった．

第3次規格は1980年制定を目標に，1977年に米国で作成作業が開始された．しかし，CODASYLの追加・変更機能をどこまで採用するか，第2次規格との互換性をどう保持するか，データベースの定義言語の規格化とどう調整するか，膨大な文書化作業の中で各々の記述の整合性をどう保つかなど多くの問題に時間がかかり，1981年9月にやっと第3次規格原案[4]が完成した．

その後半年間，世界各国のコンピュータ関連の規格委員会，メーカ，ユーザ，団体などが公式のレビュをし，約2,200通という予想外の大量な意見を米国規格協会に提出した．大部分は第2次規格と互換性のない点に対する批判的な意見であった．

米国規格協会は，更に半年かけてこれらの意見を整理し原案の変更を検討することになったが，1982年11月現在最終的な結論に至っていない．

3. 第3次規格原案の機能単位構成と水準

表-2に第3次規格原案の機能単位の構成と水準を示す．これは，第2次規格に対し次の点が変わった．

(1)　デバッグ，報告書作成，通信の三つの機能単位は利用者も少なく，重要度が低い．この三つを選択機能単位として，他の八つ（必要機能単位）よりもランクを落とした．これらを処理系（コンパイラ）で支援するかどうかは，処理系作成者の自由である．

(2)　必要機能単位の中の水準の組合せは，最高部

† The Trend of Standardization Activity of COBOL by Tetsuji IMAJO (Software Works, Hitachi Ltd.).

†† (株)日立製作所ソフトウェア工場

表-2 第3次規格原案の機能単位と水準

機能単位		水準		
		COBOL 部分集合		
		最低	中間	最高
必要	中核	水準1	水準1	水準2
	順ファイル	水準1	水準1	水準2
	相対ファイル	なし	水準1	水準2
	索引ファイル	なし	水準1	水準2
	プログラム間連絡	水準1	水準1	水準2
	区分化	なし	水準1	水準1
	整列併合	なし	水準1	水準1
	原始文操作	なし	水準1	水準1
選択	デバッグ	水準1		水準2
	報告書作成	水準1		
	通信	水準1		水準2

分集合，中間部分集合，最低部分集合の三つだけ許す．第2次規格では水準の組合せは処理系の自由だった．

（3） 第2次規格の表操作は中核に吸収された．

（4） 登録集(library)の名前が原始文操作(source text manipulation)に変わった．これは，REPLACE 命令が追加されたことによる．

（5） プログラム間連絡機能の重要度が増したため，水準1はどの処理系でも支援が必要になった．

4. 規格の主要な変更点

4.1 第2次規格

第2次規格は，第1次規格と比べ188個の変更点があった．この中の88個は第1次規格に従って書かれたプログラムに影響しない機能追加・改善であり，残りの100個は既存プログラムに影響する機能削除・変更・追加である．主な変更点を次に示す．

（1） 乱ファイル機能単位を廃止し，相対ファイルと索引ファイルの二つの機能単位に置換

（2） 報告書作成と登録集機能単位を完全に書き直し

（3） デバッグ，プログラム間連絡及び通信機能単位を新設

（4） 文字列処理の INSPECT，STRING 及び UNSTRING 命令を追加

（5） その他多くの機能追加・変更・削除

（6） あいまいな仕様の明確化

4.2 第3次規格原案

第2次規格は，言語仕様が整理強化された反面，互換性の面では問題の多い改訂であった．この変化と比べ，第3次規格原案の変化はそれほど大きくない．すなわち，第2次規格は「革新」であったが，第3次規格原案は「改善」である．この差の原因は，第2次規格で COBOL の主要な機能がほぼ完成したことと，利用者が保有する COBOL プログラム財産が膨大な量になっているため大きな変化が許されなくなったことの2点である．

比較的規模が小さいとはいえ，第3次規格原案は第2次規格と比べ135個の変更点がある．この中の89個は第2次規格に従って書かれたプログラムに影響しない機能追加・改善であり，残りの46個は既存プログラムに影響する機能削除・変更・追加である．前者は第5章に，後者は第6章に詳述する．主な変更点を次に示す．

（1） 整構造プログラミング機能の導入——うち(inline) PERFORM 命令と多枝分岐する EVALUATE 命令の追加

（2） プログラム間連絡機能の大幅強化——入れ子原始プログラム，プログラム間でのデータとファイルの共用

（3） データの項類に従って値を初期化する INITIALIZE 命令の追加

（4） 数字編集項目から数字項目へ転記 (move) する逆編集機能の追加

（5） ALTER 命令，RERUN 句などの廃止

（6） 55個の予約語の追加

135個の変更点のほかに，第3次規格原案では14個の移行用 (transitional) 言語要素が定められている．これらは，第2次規格にあり，最新の CODASYL COBOL にはなくなった言語要素である．将来の第4次規格では削除されると予告されている．移行用言語

表-3 第3次規格原案の移行用言語要素一覧表

部	言語要素
環境	・ファイル管理記述項の ACCESS MODE, FILE STATUS, RECORD KEY 及び ALTERNATE RECORD KEY 句．ただし，これらはデータ部のファイル記述項に移る． ・MULTIPLE FILE TAPE 句．
データ	・ファイル記述項の BLOCK CONTAINS と CODE-SET 句．ただし，これらは環境部のファイル管理記述項に移る． ・DATA RECORDS, LABEL RECORDS 及び VALUE OF 句． ・RECORD CONTAINS 整数-4 TO 整数-5 CHARACTERS.ただし，可変長レコードはRECORD IS VARYING 句で指定する．
手続き	・INSPECT 命令での TALLYING と REPLACING との同時指定． ・STOP 命令の定数指定． ・デバッグ機能．ただし，デバッグ行は中核に移る．

要素を表-3 に示す.

5. 第3次規格原案の追加機能

本章では，既存プログラムに影響のない第3次規格原案の追加機能の大部分について述べる.

5.1 整構造プログラミング機能の導入

今までの COBOL は，PASCAL や PL/I などと比べ整構造プログラミング機能が弱体であった.

- 条件命令の制御の合流点は終止符しかない.
- IF 命令以外の条件命令を入れ子にできない.
- 他言語の do 文では繰返しの本体を do 文の直後に書けるのに，PERFORM 命令では繰返しの本体を外に書かなければならない.
- PERFORM UNTIL 命令は do-while 型（繰返し前に条件判定）である．repeat-until 型（繰返し後に条件判定）の書き方がない.
- case に相当する多枝分岐命令がない.

このような批判に対し，第3次規格原案では従来の COBOL 仕様と共存するよう細心の注意を払って整構造プログラミング機能を導入した.

（1） **範囲明示符**　END-IF など条件命令の範囲，すなわち制御の流れの終わりを明示する予約語が導入された．この予約語は「END-動詞」の形をしており，範囲明示符 (scope terminator) という．「IF…END-IF」のように範囲明示符で終わる命令を範囲明示命令 (delimited scope statement) という．範囲明示命令は無条件命令である.

```
例: IF A = B
    THEN  MOVE 1 TO X
    ELSE  READ M-FILE
          AT END
            IF T = HIGH-VALUE
            THEN MOVE 2 TO X
            ELSE  MOVE 3 TO X
            END-IF
          END-READ
    END-IF
```

この範囲明示の導入で，COBOL プログラム不良を誘発しがちな終止符はほとんど書かなくて済むようになった．すなわち，完結文（sentence：命令の最後に終止符空白を置いたもの）の概念は，既存プログラムの互換性のためだけのものとなり，終止符は段落の最後にだけ書けばよい.

（2） **CONTINUE 命令**　END-IF は完結文の終わりを意味しないので，IF…END-IF 中には NEXT SENTENCE が書けない．この代わりに，何も実行しない形式的な命令——CONTINUE 命令が新設された．この命令は非実行命令であり，次に実行可能な命令に制御を移す．なお，この命令はどこに書いてもよい.

（3） **条件命令の対称分岐（未採用）**　CODASYL では，IF 命令以外の条件命令でも対称的な分岐が可能である．例えば READ 命令の AT END 句や算術命令の ON SIZE ERROR 句に対応する NOT AT END 句や NOT ON SIZE ERROR 句が書ける.

```
例: ADD 1 TO A
    ON SIZE ERROR      MOVE 1 TO X
    NOT ON SIZE ERROR  MOVE 0 TO X
    END-ADD
```

しかし，第3次規格原案では「IF…THEN…ELSE」以外の対称的な分岐の言語仕様は採用されなかった．この理由は，次のように考えたからであると思われる.

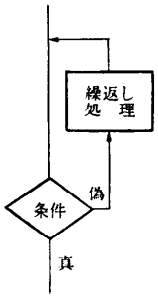
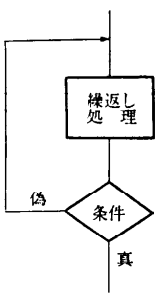
AT END 句やON SIZE ERROR 句に書く処理は例外処理である．一方，対称形の処理は一般に何度も実行する正常処理である．このように現実には対称的な処理にならないものを，形式的に対称形にするのは行き過ぎである.

（4） **うち PERFORM 命令**　今までの PERFORM 命令は繰返しの本体には手続き名を付けて外に書かなければならなかった（例-1）が，PERFORM 命令の直後に繰返しの本体を書けるようになった（例-2）．前者をそと (out-of-line) PERFORM 命令，後者をうち (in-line) PERFORM 命令という．うち PERFORM 命令の繰返し本体の最後には END-PERFORM と書く.

```
例-1. PERFORM LOOP-1
        VARYING I FROM 1 BY 1
        UNTIL I > 10.
          ⋮
      LOOP-1.
        MOVE A (I) TO B (I).
例-2. PERFORM VARYING I FROM 1 BY 1
                  UNTIL I > 10
                  MOVE A (I) TO B (I)
        END-PERFORM
```

（5） **EXIT PERFORM 命令（未採用）**　うち PERFORM 命令の繰返し本体の中には段落名は書けない．しかし，繰返しの本体の途中で繰返しを強制的に終了させたい場合がある．この目的の命令として，CODASYL では 1981 年に EXIT PERFORM 命令が追加されたが，第3次規格原案がほぼ完成していたため，原案には反映できなかった．この命令の書き方

表-4 PERFORM 命令の条件判定と繰返し処理の前後関係

書き方	do-while 型	repeat-until 型
フローチャート		
COBOL	PERFORM [WITH TEST BEFORE] UNTIL 条件 繰返し処理の命令群 END-PERFORM	PERFORM WITH TEST AFTER UNTIL 条件 繰返し処理の命令群 END-PERFORM

は次の二通りである.

- EXIT PERFORM : END-PERFORM の直後の実行命令に制御を移す.
- EXIT TO TEST OF PERFORM : その時点の繰返し処理を終わらせ, PERFORM 命令の繰返し条件判定の処理に制御を戻す.

(6) **repeat-until 型の PERFORM 命令**

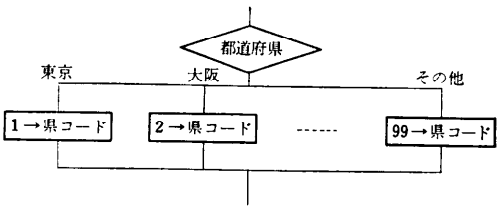
PERFORM 命令の条件判定の時期を繰返し処理の前又は後のいずれにするかの指定ができるようになった. これは, WITH TEST BEFORE/AFTER 句で指定する. WITH TEST BEFORE の場合は, 今までの PERFORM 命令と同じなので, 省略してもよい. この例を表-4 に示す.

(7) **繰返しネスト数の無制限化** PERFORM 命令の繰返しのネスト数は, 第2次規格では VARYING 句1個と AFTER 句2個の合計3個までだったが, 第3次規格原案では AFTER 句の個数の制限がなくなった.

(8) **EVALUATE 命令** case に相当し多枝分岐を可能とする EVALUATE 命令が追加された.

例-1 の TO-DO-FU-KEN を選択主体, WHEN のあとの定数を選択対象という. 選択主体には一意名, 定数, 式などが書け, 選択対象には一意名, 定数, 算術式, 条件式, 予約語 ANY, 値の範囲などが書ける. 選択主体は複数個書くこともでき, この場合には同じ数だけ選択対象も書かなければならない. このように EVA LUATE 命令は他の言語の case 構造の文と比べ, 非常に強力な命令となっている. 例えば, 複雑な決定表のコーディングも EVALUATE 命令を用いると簡単に書ける (例-2).

例-1: 単純な EVALUATE 命令のフローチャートとコーディング




```
EVALUATE TO-DO-FU-KEN
  WHEN "TOKYO" MOVE  1 TO KEN-CODE
  WHEN "OSAKA" MOVE  2 TO KEN-CODE
        ⋮
  WHEN OTHER   MOVE 99 TO KEN-CODE
END-EVALUATE
```

例-2: 複雑な EVALUATE 命令の決定表とコーディグ

条件	数学	5	5	3~4	4	1	1~5	その他
	国語	5	3~4	5	4	1~5	1	
判定	進路	自由	理科系	文科系	自由	落第	落第	浪人

```
EVALUATE SUGAKU KOKUGO
  WHEN    5        5      MOVE "JIYU"     TO SINRO
  WHEN    5    3 THRU 4   MOVE "RIKA"     TO SINRO
  WHEN 3 THRU 4    5      MOVE "BUNKA"    TO SINRO
  WHEN    4        4      MOVE "JIYU"     TO SINRO
  WHEN    1      ANY      MOVE "RAKUDAI"  TO SINRO
  WHEN  ANY        1      MOVE "RAKUDAI"  TO SINRO
  WHEN       OTHER        MOVE "RONIN"    TO SINRO
END-EVALUATE
```

5.2 プログラム間連絡機能の大幅強化

整構造プログラミング機能は一つのプログラムの中の制御構造の改善であるが, 第3次規格原案では複数のプログラム間の構造も改訂している.

(1) **入れ子プログラム** COBOL サブルーチンを書く方法として, ①プログラム内に書き PERFORM 命令で呼ぶ方法と, ②別の独立したプログラムを書き CALL 命令で呼ぶ方法の二つがあるが, 更に ③一つのプログラムの中に別プログラムを書き CALL 命令で呼ぶ方法 (**図-1**) が追加された. 図-1 の中のプログラム KO, MAGO, KYOTSU を入れ子プログラムという. 入れ子プログラムの中に入れ子プログラムを書くことができる.

入れ子プログラムからは, それが含む入れ子プログラム以外の入れ子プログラムを呼ぶことはできない. ただし, PROGRAM-ID 段落に COMMON (共用) と書いた入れ子プログラムは, 他の入れ子プログラムから呼ぶことができる.

入れ子プログラムの導入にともない, プログラムの最後を示す END PROGRAM も導入された.

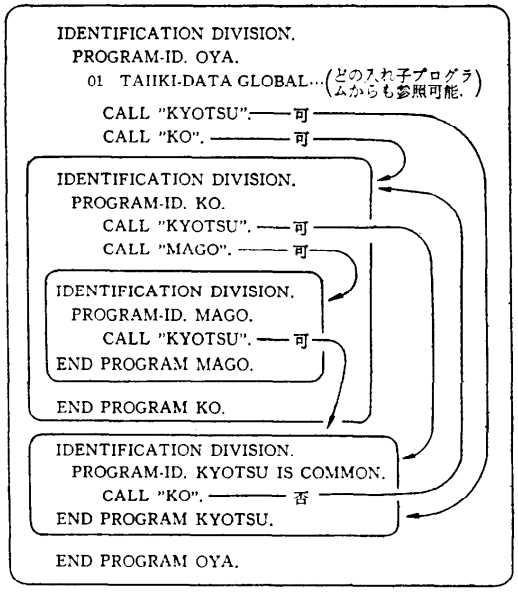


図-1　入れ子プログラムと各プログラム間の参照可否

（2）**プログラムの初期化**　プログラムが再度呼び出された時の状態は，CANCEL 命令でそれが取り消された直後を除き，前回そのプログラムが終了したときの状態のままである．この仕様のため，前回の実行結果に影響されないようにプログラムを作ること，例えば再使用可能 (reusable) プログラムを作ることは COBOL では簡単ではなかった．これを簡単にするために PROGRAM-ID 段落に INITIAL を書けるようにした．INITIAL と書いたプログラムは，呼ばれるたびに初期状態になっている．

例: PROGRAM-ID. プログラム名 IS INITIAL.

（3）**データの共用**　二つのプログラムでデータを共用する今までの方法は，両方のプログラムに同じデータを定義し，それを CALL 命令と呼ばれるプログラムの手続き部の見出しとの両方の USING 句に書くという方法であった．なお，呼ばれるプログラム側のデータ定義は連絡節に書く．第3次規格原案では新たに二つのデータ共用方法を追加した．

（a）**大域データと局所データ**　呼ぶプログラムの 01 レベルに GLOBAL (大域) と書くと，大域データとなる．大域データは，呼ばれる入れ子プログラム側に何も定義せずに参照可能となる（図-1 参照）．大域データ以外のものを局所 (local) データという．

（b）**外部データと内部データ**　別々のコンパイル単位のプログラム間でデータを共用するためには，各々の作業場所節で同一のデータを定義し，その 01 レベルに EXTERNAL (外部) と書けばよい．外部データは，USING 句にそのデータ名を書かずに共用できる．外部データ以外のデータを内部データという．なお，一つの 01 レベルに GLOBAL と EXTERNAL の両方を指定してもよい．

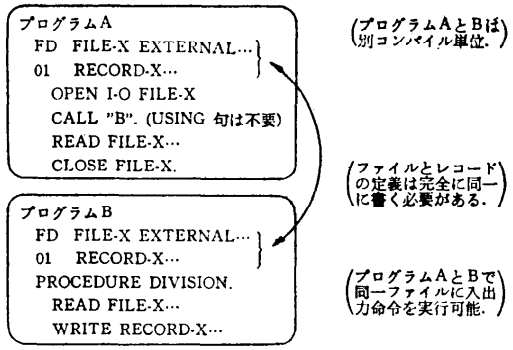


図-2　複数プログラム間でのファイルの共用

（4）**ファイルの共用**　今までは一つの実行単位内の複数プログラム間では，ファイル共用はできなかったが，FD 項に GLOBAL 又は EXTERNAL と書くことによってファイル共用が可能となった．図-2 に例を示す．GLOBAL と EXTERNAL の規則はデータ共用の場合と同じである．なお，異なる実行単位間でファイルの共用・排他制御をする言語仕様は CODASYL の将来検討項目となっており，第3次規格原案にも入っていない．

（5）**CALL 命令の機能拡張**　CALL 命令の USING 句に書いたデータは，呼ばれるプログラムで自由に変更可能である．このため，変更しては困るデータを誤って変更してしまうというプログラム不良が発生しがちであった．これを防止する書き方が CALL 命令に追加された．CALL 命令の USING 句で BY CONTENT を指定したデータ項目は，CALL 命令実行時に別の領域に転記され，後者が呼ばれるプログラムで参照される．この領域を誤って変更しても，呼ぶプログラム側のデータ領域と異なるため，呼ぶプログラムに影響を与えないで済む．今までどおり同一データ領域を共用する場合には，何も書かないか BY REFERENCE を書く．

例: CALL "B" USING BY CONTENT　X Y
　　BY REFERENCE Z.

CODASYL では，BY CONTENT に定数や算術式も書くことができるが，第3次規格ではこの仕様は採用されなかった．

（6）**EXIT PROGRAM 命令の制限緩和**

この命令は段落名の直後にしか書けなかったが，どこに書いてもよいようになった．

5.3 中核の追加機能

（1） **中味のない部の見出しの省略** 環境部，データ部及び手続き部の見出しは，中味がない場合には省略可能になった．見出し部の見出しは省略できない．

（2） **INITIALIZE 命令の新設** この命令は数字にはゼロ，文字には空白など，基本項目の項類に従って項目を初期化する．これにより初期化コーディングが容易になった．

```
例-1:「INITIALIZE A」を実行すると，次のように値が入る．
 01  A.
   02  B              PIC X(5).——空白が入る．
   02  C              PIC A(2).——空白が入る．
   02  FILLER         PIC X.    ——何も入らない．
   02  D              PIC 9(5).——ゼロが入る．
   02  E REDEFINES D  PIC X(5).——再定義項目は無視．
   02  F OCCURS 5     PIC 9(2).——各要素にゼロが入る．
例-2: 例1の Aに対し次の命令を実行すると，数字項目 (D, F の各
  要素) だけに値1が入る．
  INITIALIZE A REPLACING NUMERIC DATA BY 1.
```

（3） **INSPECT 命令の拡張** この命令に CONVERTING 句が追加された．また BEFORE/AFTER INITIAL 句が複数個書けるようになり，文字列操作が便利になった．

```
例: INSPECT A CONVERING "ABCD" TO "1234"
      AFTER INITIAL "." BEFORE INITIAL ",".
    命令実行前のAの値      AB. DCBA, AB
    命令実行後のAの値      AB. 4321, AB
```

（4） **データの表現力の強化** データの定義や参照の機能が増え，上限値や各種制限が緩和された．

（a） データ項目の部分参照 (reference modification) 機能によって，データ項目の一部を，データ部で定義せずに，手続き部で直接参照可能とした．これはデータ名の後に(左端文字位置： 大きさ) と書く．左端文字位置と大きさには，データ名，定数又は算術式を書ける．

```
例: 01  A  PIC X(10).
    01  B  PIC X(3).
    MOVE  A (5: 3) TO B.
  Aの5文字目から3文字分をBに転記する．
```

（b）複数次元の表を参照する場合，添字付けと指標付けを混在して書けるようにした．更に，A(I+5) の書き方は，Iが指標の場合だけ許されたが，Iが添字の場合も書けるようにした．

（c） 表の次元数の上限は3次元であったが，48次元にした．この結果，第3次規格原案を満足する処理系を作る場合には，2^{48} 文字以上の記憶域をもつハードウェア，すなわちアドレス表現が48ビット以上のハードウェアが必要となる．このようなハードウェアは1980年代に普及する可能性は少なく，COBOL の応用範囲としても48次元を必要とするものは考えにくい，FORTRAN の規格同様，7次元とするのが規格として妥当であろう．

（d） OCCURS DEPENDING 句で指定する可変長の表の反復回数は1以上であったが，0も許されるようになった．この結果，大きさが0のデータの参照が可能となり，処理系作成者にとって面倒な問題が一つ増えた．

（e） 文字定数の上限値は120文字であったが，160文字に拡張された．

（f） データ名に付けることができる修飾語の個数の上限は5個以上であったが，50個に拡張された．

（g） 名前が不要なデータ項目につける FILLER は省略可能となった．

（h） REDEFINES 句の再定義項目と被再定義項目の大きさは等しくなければならなかったが，再定義項目の方が小さくてもよいとなった．なお,CODASYL では再定義項目の方が大きくてもよい．

（5） **SYMBOLIC CHARACTERS 句の新設** 環境部の特殊名段落に書き，利用者が任意に表意定数を指定する場合に用いる．

```
例: 漢字の空白をコード系の34番目のコード(16進数の21)が2つ並
  んだものとすると，これを次のように利用者表意数として指定でき
  る．
    SYMBOLIC CHARACTERS KANJI-SPACE IS 34.
    01  KANJI-RECORD PICTURE X(120)
                      VALUE ALL KANJI-SPACE.
```

（6） **逆編集機能の追加** MOVE 命令で数字編集項目から数字項目への転記が可能となり，数字データの入力が容易になった．

```
例: 01 A PIC−−−−, −−9.
    01 B PIC S9(6).
    MOVE A TO B.
  Aが −1,000 のとき，Bに負の 001000 が入る．
```

（7） **ACCEPT 命令と DISPLAY 命令の拡張** ACCEPT 命令で DAY-OF-WEEK を書くと，週の曜日番号（月曜が1，火曜が2，…，日曜が7）が求まる．DISPLAY 命令でWITH NO ADVANCING 句を書くと，改行せずに表示できる．

```
例: ACCEPT YOBI-BANGO FROM DAY-OF-WEEK.
    DISPLAY "ENTER DATA" WITH NO ANVANCING.
```

5.4 その他の機能単位の機能拡張

（1） **可変長レコード処理の改善** 今までのCOBOLでは可変長レコードの入出力が可能であったが，入力時にレコード長を直接知る手段がなかった．また，各種の制限があったが，機能が拡張され取り扱いが容易になった．

（a） RECORD 句に可変長レコード用の VARYING 句が追加された．レコード長は整数-2 から整数-3 の範囲である．レコード入力時に，レコード長がデータ名-1 に入る．レコード出力前に，データ名-1 に値を設定すると，それが出力レコード長になる．

書き方: RECORD IS VARYING IN SIZE
[[FROM 整数-2] [TO 整数-3] CHARACTERS]
[DEPENDING, ON データ名-1]

（b） 固定長レコードファイルしか指定できなかった READ INTO 命令，RETURN INTO 命令，SORT USING/GIVING命令及び MERGE GIVING 命令で可変長レコードファイルの指定が可能となった．

（c） 相対及び索引ファイルの REWRITE 命令で異なった長さのレコードで更新が可能となった．なお，順ファイルに対しては相変わらず異なった長さのレコードによる更新はできない．

（d） ファイル管理記述項に書く RECORD DELIMITER 句が新設された．この句で外部媒体上の可変長レコードを識別する方法を指定する．

書き方: RECORD DELIMITER IS {STANDARD-1 / 作成者語}

（2） **環境部とデータ部に書く句の入替え** 物理的な性質を指定する句 (BLOCK CONTAINS 句と CODE-SET 句) をデータ部から環境部へ移し，論理的な性質を指定する句(ACCESS 句，FILE STATUS 句，RECORD KEY 句，ALTERNATE RECORD KEY 句)を環境部からデータ部へ移した．ただし，従来の部に書いてもよいことになっているが，これは移行用言語要素であり，将来の第4次規格では削除予定である．

（3） **ファイル編成の制限の緩和** 順ファイルだけに指定できた句 (SELECT OPTIONAL 句) と命令 (OPEN EXTEND 命令，MERGE GIVING 命令，SORT USING/GIVING 命令) が，相対ファイルと索引ファイルにも書けるようになった．

（4） **LABEL RECORDS 句が不要** この句を書かなくてもよくなった．書かないと標準ラベルが仮定される．なお，LABEL RECORDS 句は移行用言語要素であり，第4次規格では書けなくなる．

（5） **PADDING CHARACTER 句の新設** この句はファイル管理記述項に書き，順ファイルのブロックの埋字を指定する．

書き方: PADDING CHARACTER IS {データ名-1 / 定数-1}

（6） **整列併合機能の拡張** SORT 及び MERGE 命令の GIVING 句で複数のファイルを指定できるようになった．SORT 命令に DUPLICATES 句を書くと，同一整列キーの入力順序を保存して，整列する．

例: SORT SORT-FILE……
WITH DUPLICATES IN ORDER……

（7） **REPLACE 命令の新設** この命令は後続する原始文中の仮原文-1 を仮原文-2 で置き換える．この置換えは，次の REPLACE 命令まで有効である．

書き方: REPLACE {==仮原文-1== BY ==仮原文-2==}…
REPLACE OFF

（8） **トランザクション指向の通信機能** 一つの通信記述項で通信文の入力と出力の両方を可能とするI-O指定が追加された．これは，端末からトランザクション（入力通信文）を受信し，処理結果（出力通信文）をその端末に送信するという通常のオンライン業務に対応する機能である．

6. 既存プログラムに影響のある変更項目

第3次規格原案の中で46個の変更項目が，第2次規格に従って書かれた既存プログラムに影響する．若干のものを除き，これらの変更項目を**表-5** に示す．

削除された項目の中で最も問題なのは ALTER 命令である．「ALTER 命令使用プログラムの保守は困難．新規作り直しすべき．」といわれるほど悪名が高いが，スイッチとして一見便利なため昔から多用されている．

RERUN 句と，磁気テープを巻きもどさず(CLOSE NO REWIND)，逆読み (OPEN REVERSED) する処理も削除された．これらの代替手段はないので残すべきである．

機能追加の結果，56個の予約語が増えた．これはCOBOLの宿命であり，これらの予約語を利用者語として使っているプログラムの変更が必要となる．

表-5 既存プログラムに影響のある第3次規格原案の変更項目（分類中の番号の項目が特に影響大）

分類	項目
削除（①～④）	①ALTER 命令 ②RERUN 句 ③OPEN REVERSED 命令 ④CLOSE NO REWIND 命令 ⑤ENABLE/DISABE 命令の KEY 指定 ⑥ENTER 命令 ⑦MEMORY SIZE 句
無意味な仕様の禁止	①ADVANCING PAGE 句と END-OF-PAGE 句を WRITE 命令に同時指定 ②LINAGE 句指定ファイルに対する OPEN EXTEND 命令 ③相対レコードキー項目の PICTURE 中の P
制限事項の解除（①）	①AT END 条件成立後の READ 命令（再度 AT END 条件になる） ②副プログラムの最後に次の実行命令なし（EXIT PROGRAM 命令を仮定）
第2次規格未規定仕様の明確化（⑤）	①DIVIDE 命令の REMAINDER 句の添字評価時期 ②INSPECT, STRING, UNSTRING 命令の添字評価時期 ③CANCEL 及び STOP RUN 実行時の未クローズファイルの処置 ④副プログラム終了時の未完了 PERFORM 命令の処置 ⑤ファイル及び通信機能の各種エラーケースの状態キーの値
仕様変更（①～③）	①英小文字に対する ALPHABETIC 条件は真になる（第2次規格では偽） ②OPEN I-O, 最小キーより小さいレコードを WRITE し，続いて READ NEXT したときの入力レコード ③反復回数を自分自身に含む可変長レコードの長さ
その他（①）	①56個の予約語の増加

7. 第3次規格原案と CODASYL との相違

CODASYL にあり，第3次規格原案に反映されていない項目はかなり多い．主要なものを表-6 に示す．データベースは当初含む予定であったが，CODASYL のデータ定義言語の規格化が延期になったため，COBOL の規格に含めないことになった．その他の項目は，互換性又は汎用性の面で問題があり，規格化の対象から外された．また，原案がほぼ完成した後の CODASYL 変更点も反映していない．

8. おわりに

本稿では，米国の COBOL 第3次規格原案を中心に COBOL の標準化の動向を紹介した．この中の新機能の中で整構造プログラミングと INITIALIZE 命令はプログラムの生産性・保守性向上に大きく寄与するものであり，既に支援している処理系も増えつつある．一方，プログラム間連絡機能の追加機能を支援している処理系はないようである．これは，この機能に対する COBOL 利用者のニーズが CODASYL が重要視するほどないことが原因であろう．

表-6 第3次規格原案で不採用の CODASYL COBOL の項目

分類	項目
大機能	データベース，ビット操作，数学関数
中核	EXIT PERFORM 命令，条件命令の対称形の分岐，浮動小数点データ，USAGE 句の BINARY と PACK-DECIMAL，添字中の算術式，REDEFINES 句の長さの制限事項解除，表の要素に対する初期値設定，利用者語中の英小文字，77レベルの廃止，CORRESPONDING の廃止
ファイル	DELETE FILE 命令，WRITE ADVANCING 命令の絶対行指定，DATA RECORDS 句と LABEL RECORDS 句の廃止
連絡	CALL 命令の USING 句で定数と算術式を指定
デバッグ	USE DEBUGGING 命令の廃止
報告書作成	COLUMN 句の相対文字位置指定，PRESENT WHEN 句の条件で行又は項目の印刷を抑止
正書法	全面改訂（自由形式化，標式領域の廃止，文字定数の継続方法の変更）

第3次規格は，第2次規格からの互換性が一部ないため利用者の反対が多く，当初予定と比べ制定が大幅に遅れている．これについては，影響度の大きいものを互換機能として残すことで解決し，規格の正式制定を急ぐべきと考える．

参考文献

1) 今城哲二：COBOL，情報処理，Vol. 22, No. 6, pp. 457-460 (1981).
2) CODASYL COBOL Committee Journal of Development 1981, the Secretariat of the Canadian Government EDP Standards Committee (1981).
3) 日本工業規格（JIS）電子計算機プログラム言語 COBOL, JIS C 6205-1980, p. 383, 日本規格協会 (1980).
4) Draft Proposed Revised X 3.23 American National Standard Programming Language COBOL, p. 750, X3 Secretariat of the Computer and Business Equipment Manufacturers Association (1981).
5) 植村俊亮，真野芳久：整構造 COBOL（1）～（3），bit, Vol. 10, No. 11, pp. 68-75, No. 12, pp. 51-58, No. 14, pp. 68-76, 共立出版 (1978).

（昭和58年1月15日受付）