解　説

COBOL プログラマからみた Ada†

安原隆一†† 西山　茂†† 伊集院　正††
堀田博文†† 萩原茂夫††† 山田良史†††

1. はじめに

本稿は，COBOL を知っているまたは使っているプログラマが Ada 言語を理解しようとするのに役立つよう，簡単な例題を使って Ada プログラムを説明するものである．もとより少ない誌面で Ada 言語仕様のすべてを説明することはできないし，またそのつもりもない．ここでは，Ada 言語仕様の基本的な構文や特徴的な構文のいくつかを COBOL 言語と対応させて述べるにすぎない．はじめの例題では，プログラムの書き方，全体構造，代入文，if 文，loop 文など基本的な項目を中心に説明し，次の例題では，手続き，関数，パッケージ，例外など COBOL 言語仕様にない Ada 言語の特徴的な項目も含めて説明する．

2. 表検索問題における比較

2.1 問　題

氏名，職務給，振込銀行コードなどの項目番号からなる給与データの穿孔されているカードをカード読取り装置から読み，振込銀行ごとの職務給の合計を求め，印字するプログラムを作る。銀行コードは次の9つとし，該当する銀行コードがない場合は誤り表示をする．

114, 115, 116, 117, 201
202, 203, 220, 221

（a） カードデータ (CRFILE)

氏　名	職務給	振込銀行コード	その他
X(20)	9(6)	9(3)	X(51)

（b） 集計リスト (LPFILE)

GINKOU	SHOKUMUKYU
xxx	xxx, xxx, xxx
xxx	xxx, xxx, xxx

2.2 COBOL プログラム例

COBOL による記述例を**図-1** に示す．

2.3 Ada プログラム例

Ada による記述例を**図-2** に示す．

2.4 解　説

（1） 正 書 法

COBOL は正書法が決められているが，Ada にはこのような決りはなくどの字句要素も行のどこから書きはじめてもよい．

（2） プログラム構造

COBOL のプログラムは，Ada の手続き(procedure)に相当する．見出し部，環境部及びデータ部は，procedure から begin までの宣言部 (declarative part) に当る．手続き部は，begin から end までの文の列に当る．

（3） プログラム名

見出し部のプログラム名は，副プログラムの名前に当る．見出し部中の他の部分は，Ada 言語仕様に相当する部分がない．必要ならば注釈 (comment) として書く．

（4） 入出力ファイルの指定

入出力節中の FILE-CONTROL. に相当する部分は，宣言部にはない．Ada では，これらの情報は，実行するとき (COBOL の OPEN 命令に相当するところで) 指定する．これについては，後で述べる．

（5） データの型と宣言

Ada には，論理型，整数型，文字列型などいくつか

---

† Ada for COBOL Programmers by Ryuichi YASUHARA, Shigeru NISHIYAMA, Sei IJUIN and Hirofumi HOTTA (Programming Language Processing Section NTT Software Production Technology Laboratories, Nippon Telegraph and Telephone Corporation), Shigeo HAGIWARA and Ryoji YAMADA (Computer Systems Division Oki Electric Industry Co., Ltd.).

†† 日本電信電話(株) NTT ソフトウェア生産技術研究所プログラム言語研究室

††† 沖電気工業(株)コンピュータシステム開発本部ソフトウェア開発第一部

注） Ada は米国国防総省の登録商標である．

の既定の型 (predefined type) があり，型宣言なしに使うことができる．また Ada では，既定の型から同じ特性をもつ別の型を定義してデータの保全を図ったり（派生型や部分型），列挙型 (enumeration type) を用いてデータの読解性を向上させるなど，COBOL に比べ豊富な種類の型の使い分けが可能で，データの記述性に優れている．

データ部の作業場所節は，型や変数の宣言に対応する．プログラムで使用する変数は，COBOL も Ada もすべて宣言しなければならない．ただし，Ada では，for 繰返し規則をもつ loop 文のループパラメータは，そのループ内でしか有効でなく，宣言する必要がない．

（6） 初 期 値

COBOL のプログラムでは，配列に初期値を与えるために REDEFINES を使っているが，Ada では，配列やレコード型の変数に対しても直接初期値を与えることができるので COBOL のような特別な記述は不要である (①)．

（7） 範囲の制約

Ada では，変数の値の範囲を限定できる (②)．これは，COBOL の ON SIZE ERROR よりきめ細かく，誤った値によるプログラムの誤動作を防止できる．

（8） 手続き部分の対応

COBOL の手続き部中に記載する処理は，Ada では，begin と end の間に書く．さらに，Ada は，制御構造を記述するのに容易な case 文，loop 文などをもっている．

（9） 代入と演算

COBOL の MOVE 命令に相当する Ada の機能は，代入文 (assignmentstatement) である．また COMPUTE 命令，ADD 命令など一連の計算をする命令も代入文と対応する (③)．MOVE 命令では自動的にデータの型の変換がなされるが，Ada では，型が一致しなければ代入はできない．これは，演算についても同様であり，思わぬ誤用を防ぐために

```
IDENTIFICATION DIVISION.
PROGRAM-ID.      GINKO-BUNRUI.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
     SELECT CRFILE ASSIGN TO READER.
     SELECT LPFILE ASSIGN TO PRINTER.
*
DATA DIVISION.
FILE SECTION.
FD CRFILE              LABEL RECORD IS OMITTED
                       DATA RECORD IS CR-REC.
01   CR-REC.
     02  FILLER       PICTURE     X(20).
     02  SHOKUMU-C    PICTURE     9(6).
     02  BANK-C       PICTURE     9(3).
     02  FILLER       PICTURE     X(51).
FD   LPFILE            LABEL RECORD IS OMITTED
01   LP-REC.           DATA RECORD IS LP-REC.
     02  FILLER       PICTURE     X(10).
     02  BANK-L       PICTURE     9(3).
     02  FILLER       PICTURE     X(5).
     02  SHOKUMU-L    PICTURE     ZZZ. ZZZ. ZZ9.
*
WORKING-STORAGE SECTION.
01   I                PICTURE     99.
01   MIDASHI          PICTURE     X(26)     VALUE
     "          GINKOU SHOKUMUKYU".
01   CODE-TABLE       PICTURE     X(27)     VALUE
     "114115117117201202203220221".
① 01 CODE-HYO         REDEFINES   CODE-TABLE.
     02  BANK         PICTURE     9(3)      OCCURS 9.
01   TOTAL-HYO        VALUE ZERO.
     02  TOTAL        PICTURE     9(9)      OCCURS 9.
PROCEDURE DIVISION.
HAJIME.
     OPEN INPUT   CRFILE.
     OPEN OUTPUT LPFILE.
YOMU.
     READ CRFILE AT END GO TO OWARI.
     MOVE 1 TO I.
TSUGI.
④    IF BANK (I)=BANK-C GO TO CODE-ARI.
⑦    COMPUTE I=I+1.
     IF I=10 DISPLAY "CODE ERROR" BANK-C GO TO YOMU.
     GO TO TSUGI.
CODE-ARI.
③    COMPUTE TOTAL (I)=TOTAL (I)+SHOKUMU-C.
     GO TO YOMU.
OWARI.
     MOVE MIDASHI TO LP-REC.
⑨    WRITE LP-REC AFTER ADVANCING 5 LINES.
     MOVE SPACE TO LP-REC.
     MOVE 1 TO I.
INJI.
     MOVE BANK (I) TO BANK-L.
     MOVE TOTAL (I) TO SHOKUMU-L.
     WRITE LP-REC AFTER ADVANCING 2 LINES.
     COMPUTE I=I+1.
     IF I=10 NEXT SENTENCE ELSE GO TO INJI.
     CLOSE CRFILE.
     CLOSE LPFILE.
     STOP RUN.
```

図-1 COBOL プログラムの例

```
with TEXT_IO: use TEXT_IO;

procedure GINKO_BUNRUI is

    package NEW_INT is new INTEGER_IO (INTEGER)         ; use NEW_INT;
    package NEW_LONG is new INTEGER_IO (LONG_INTEGER); use NEW_LONG;
    CR_FILE: FILE_TYPE;
    LP_FILE: FILE_TYPE;
    IS_CODE: BOOLEAN;
②   BANK_C: INTEGER range 1..999;
    KINGAKU: LONG_INTEGER range 0..999999;
①   CODE_HYO : array (1..9) of INTEGER : = (114, 115, 116, 117, 201, 202, 203, 220, 221);
    FURIKOMI_KINGAKU: array (1..9) of LONG_INTEGER: = (0, 0, 0, 0, 0, 0, 0, 0, 0);

begin
                            -- 給与データファイルを開く
    OPEN (FILE =⟩ CR_FILE, MODE =⟩ IN_FILE, NAME =⟩ "KYUYO_FILE");
                            -- ファイルの終りまで繰り返し
⑤   while not END_OF_FILE (CR_FILE) loop
⑥                           -- 1人分のデータの読込み
    SET_COL (CR_FILE, 21);
    GET     (CR_FILE, KINGAKU, 6);
    GET     (CR_FILE, BANK_C,  3);
    SKIP_LINE (CR_FILE);
                            -- 1記録の処理
    IS_CODE: = FALSE;
⑤⑧ for I in 1.. CODE_HYO' LAST loop
④     if CODE_HYO (I) = BANK_C then
③       FURIKOMI_KINGAKU (I) := FURIKOMI_KINGAKU (I)+KINGAKU;
        IS_CODE := TRUE;
⑦       exit;
      end if;
    end loop;
    if not IS_CODE then
       PUT      ("CODE ERROR");
       PUT      (BANK_C, 3);
       NEW_LINE;
      end if;
    end loop;
    CLOSE (CR_FIIE);
                            -- 銀行別振込み金額の印字
    CREATE (LP_FILE, OUT_FILE, "FURIKOMI_HYO");
                            -- タイトルの印刷
⑨   NEW_LINE (LP_FILE, 5);
    PUT      (LP_FILE, "GINKOU SHOKUMUKYU");
    NEW_LINE (LP_FILE, 2);
                            -- 振込金額の印刷
    for I in 1.. CODE_HYO' LAST loop
      PUT (LP_FILE, CODE_HYO (I)          , 10);
      PUT (LP_FILE, FURIKOMI_KINGAKU (I), 20);
      NEW_LINE (LP_FILE, 2);
    end loop;
                            -- ファイルを閉じる
    CLOSE (LP_FILE);

end GINKO_BUNRUI;
```

図-2 Ada プログラムの例

役立つ．このように厳密な型の運用は，一見不便ではあるが複数の開発担当者が共同でプログラムを作成するような場合に誤りを防ぐ上で有効である．

(10) 制御構造 (if 文と loop 文)

if は両言語にあり (if 命令及び if 文)，機能的にほぼ同じである(④)．

Ada の繰返しには，loop 文 (loop statement) があり，COBOL より大幅に簡単に書ける．繰返しには条件による繰返し (while 繰返し規則をもつ loop 文) と回数による繰返し (for 繰返し規則をもつ loop 文) があり，目的に応じて使い分けることができる(⑤)．AT END に相当する処理として，ファイルの終りを知るために END-OF-FILE 関数を呼出す(⑥)．END-OF-FILE は，論理型の関数値をもつ関数である．途中でループを抜け出すには，exit 文 (exit statement) を使うので，名札 (label) が不要となる(⑦)．

(11) 属性の利用

例題の loop 文で，1. CODE-HYO'LAST の部分は繰返しの範囲を指定する．ここで，CODE-HYO'LAST は，配列 CODE-HYO の最後の要素の添字を示す．1..9 と書いても同じであるが，銀行数の変化を考慮したものである．Ada には，LAST のほかにもこの種の属性 (attribute) が多数用意されている (⑧)．

(12) go to

go to は，両言語にある (go to 命令及び go to 文)．COBOL では必須の go to 命令も，Ada ではほとんど if 文，case 文，loop 文などに置き換えることができる．このため，COBOL の段落名に相当する Ada の名札もほとんど使用する必要がない．

(13) 入 出 力

Ada には，COBOL の OPEN 命令，READ 命令などという特別な文はなく，すべて手続きや関数を呼び出すことより処理する．COBOL の OPEN 命令には，Ada の OPEN 手続きまたは CREATE 手続きが対応する．COBOL では，入出力節にファイルに関する情報を指定するが，Ada では，入出力手続きを呼出せばよい．すでに存在するファイルを開く場合は OPEN 手続き，新しくファイルを生成する場合は CREATE 手続きをそれぞれ呼出す．処理を終るとファイルを閉じる必要があるのは COBOL も Ada も同様である．これは，CLOSE 手続きを呼出す．ただし，標準入出力に対しては，OPEN 手続き，CREATE 手続き及び CLOSE 手続きを呼出す必要がない．

(14) PICTURE との対応

Ada には，PICTURE ZZZ, ZZZ, ZZ9 のような簡単な指定で整数値を3けたごとにコンマで区切って出力する機能はない．プログラムロジックで書くと繁雑になるので，Ada プログラムは PICTURE ZZZZ-ZZZZ9 に相当する略記コーディングにしてある．また，ファイルがすべて文字列であるということからパッケージ TEXT-IO を使っている．TEXT-IO は，テキストファイルの入出力の処理をするパッケージで，Ada 言語仕様に既定である (Ada 処理系に用意されている)．前出の END-OF-FILE 関数もこのパッケージ中にある．

(15) ファイル

ファイルを識別するには，変数を使う (CR-FILE 及び LP-FILE)．

COBOL では，1回の READ 命令で1行の文字列を読み込むが，Ada では1回の GET 手続きの実行で1つの値を読み込む．出力の場合も同様である．なお，Ada でもレコード (COBOL の集団項目に相当) 単位の入出力を行うこともできるので，実質的な差は少ない．AFTER ADVANCING に相当する処理は，NEWLINE 手続きの呼出しである (⑨)．

(16) 標準入出力

DISPLAY に相当するものは，標準出力である．これも，出力する値ごとに PUT や PUT-LINE 手続きを呼出す．

(17) プログラムの実行の終了

COBOL の STOP RUN に相当するものは，Ada にはない．主プログラム (main program) の実行文の列が終了したときにプログラムの実行が終わる．

3. 金種計算問題における比較

3.1 問 題

社員ごとに給料を記録したファイルを読み込んでその給料額から社員ごとの金種別表を作成するプログラムを作る．ただし，給料は，100 円単位とする．また，入力ファイルのレコードの形式は次の通りとする．

1カラム 128カラム

社員番号	氏 名	給 料	
X(6)	X(20)	X(9)	

3.2 COBOL プログラム例

COBOL による記述例を図-3 に示す．

3.3 Ada プログラム例

Ada による記述例を図-4 に示す．

3.4 解 説

（1） COBOL の CORRESPONDING 指定 COBOL の CORRESPONDINGに 相当するものは，Ada にはない．したがって，Ada では，操作対象１つずつについて演算や代入を行う必要があり，記述量がやや増加する(①)．

（2） 手続き，関数

Ada では，プログラムの機能を適当に分割し，それぞれを手続きあるいは関数として定義することができる．手続きは，COBOL の PERFORM 命令や CALL 命令により呼び出されるプログラム部分に相当すると考えてよい．関数は，値を返すプログラム部分である．Ada では，手続きや関数が，その内部にまた手続きや関数をもつことができる（②，③)．Ada でも，COBOL と同様，手続きや関数を別々にファイルに入れて別々に翻訳することができる．そのとき，Ada では，別々に翻訳したプログラム同士のパラメタの型チェックなどがなされるため，正しいプログラムが速く完成する．このため，Ada プログラム作成時には，読みやすさも考え，手続きや関数に分割してプログラムを作成する方がよい．

（3） パッケージ

Ada には，パッケージというプログラム単位がある．これは，関連する型や変数や手続き，関数を１つにまとめて宣言し，パッケージ外から参照可能にするものである．パッケージは，通常，仕様と本体の２つから成る．package body から始まるのが本体（④）で body のついていないのが仕様（⑤）である．仕様では，パッケージ外から使う名前やパラメタを宣言している．本体では，仕様に書いた手続きや関数の実際の処理内容を記述している．本例では，金種の入出力を行うのに関係した型や手続きを定義した KINSYU-IO というパッケージを定義している．パッケージの仕様の部分だけ見ればそのプログラムの仕様がわかり，プログラムが読みやすい．また，複数のプログラムから共通に使用されるデータなどをこのパッケージを用いて定義することもできる．

```
IDENTIFICATION DIVISION.
PROGRAM-ID. KINSYU-KEISAN.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INPUT-F ASSIGN TO IF.
    SELECT PRINT-F ASSIGN TO SYSOUT.
DATA DIVISION.
FILE SECTION.
FD  INPUT-F LABEL RECORD OMITTED
    BLOCK CONTAINS 32 RECORDS
    RECORD CONTAINS 128 CHARACTERS
    DATA RECORD IS INPUT-REC.
01  INPUT-REC.
    02  BANGO   PICTURE X(6).
    02  NAMAE   PICTURE X(20).
    02  KYUYO   PICTURE 9(9).
    02  FILLER  PICTURE X(93).
FD  PRINT-F LSBEL RECORD OMITTED
    BLOCK CONTAINS 32 RECORDS
    RECORD CONTAINS 128 CHARACTERS
    DATA RECORD IS PRINT-REC.
01  PRINT-REC.
    02  BANGO   PICTURE X(6).
    02  NAMAE   PICTURE X(20).
    02  KYUYO   PICTURE Z(8)9.
    02  KINSYU-KOSU.
        03 MAN       PICTURE ZZZZ9P (4).
        03 GOSEN     PICTURE ZZZZ9.
        03 SEN       PICTURE ZZZZ9P (3).
        03 GOHYAKU PICTURE ZZZZ9.
        03 HYAKU     PICTURE ZZZZ9P (2).
    02  DUMMY     PICTURE X(68).
WORKING-STORAGE SECTION.
01  SYAIN.
    02  KINSYU-KOSU USAGE COMPUTATIONAL.
        03 MAN       PICTURE 9(5)P(4).
        03 GOSEN     PICTURE 9(5).
        03 SEN       PICTURE 9P(3).
        03 GOHYAKU PICTURE 9(5).
        03 HYAKU     PICTURE 9P(2).
    02  FILLER     PICTURE X(68) VALUE SPACE.
01  MIDASHI.
    02  BANGO PICTURE X(6) VALUE "BANGO".
    02  NAMAE PICTURE X(20)
                VALUE "   NAMAE       ".
    02  KYUYO PICTURE X(9) VALUE "KYUYO".
    02  KINSYU.
        03 MAN       PICTURE X(5) VALUE "10000".
        03 GOSEN     PICTURE X(5) VALUE " 5000".
        03 SEN       PICTURE X(5) VALUE " 1000".
        03 GOHYAKU PICTURE X(5) VALUE "  500".
        03 HYAKU     PICTURE X(5) VALUE "  100".
    02  FILLER PICTURE X(68) VALUE SPACE.
01  W-KYUYO    PICTURE 9(9).
PROCEDURE DIVISION.
HAJIME.
    OPEN  INPUT INPUT-F OUTPUT PRINT-F.
    WRITE PRINT-REC FROM MIDASHI.
YOMIKOMI.
```

（4） 例外処理

Adaで扱われる例外は，演算のあふれ，入出力の異常，変数値が定義された範囲にない，などの場合に発生する．Adaでは，COBOLとは異なり，例外発生への対処は，各処理を行う文個々の後ろには書かずに，手続きなどの最後（exceptionのところ）にまとめて書く．Adaの例外処理を用いれば，プログラム実行時に異常は自動的に発見され，exception whenのうしろの例外名に相当する処理に自動的に分岐するため，プログラムの通常の文の列の中には，正常処理のみを記述すればよく，プログラムの読解性がよい（⑥）．これは，COBOLの宣言部分（DECLARATIVES）に似ているが，その手続きで局所的に例外を処理するか，呼び出した手続きに例外の発生を伝えたりするかを選択できるしかけがAdaには用意されている．

（5） ファイルの終了の判定

本Ada例では，入力ファイル終了の識別方法として図-2のプログラムとは異なる方法をとっている．COBOLでは，入力ファイルの終わりがくれば，INPUT-ENDというところへ分岐するのと同様に，Adaでは，ファイルの終わりに読込み時にEND-ERRORという例外が発生するため，ファイルの終わりになると，exception when END-ERROR =〉というところへ分岐し，そこに書かれた処理を実行する（⑦）．

（6） 名前の修飾

Adaでは，構造の中のもの個々にアクセスするときは，その構造の名前で修飾することが必要である．たとえば，本例では，KINSYU-IO中の手続きMAISU-KEISANを呼び出すときには，KINSYU-IO. MAISU-KEISANと書かねばならない．ただし，パッケージの場合，use KINSYU-IO；と書けばそのパッケージによる修飾は不要となる（⑧）．

（7） Adaでは，列挙型という型を定義することができる．ここでは，MAN, GOSEN, … のうちのいずれかの値をとりうる型KINSYUが定義されている（⑨）．COBOLでは，複数の値をもつ変数は，整数型として定義し，値と意味の対応は，プログラマが暗に意

```
⑦      READ INPUT-F AT END GO TO INPUT-END.
①      MOVE  CORRESPONDING  INPUT-REC TO PRINT-REC.
       PERFORM MAISU-KEISAN.
       MOVE CORRESPONDING SYAIN TO PRINT-REC.
       WRITE PRINT-REC.
       GO TO YOMIKOMI.
⑦  INPUT-END.
       CLOSE INPUT-F PRINT-F.
       STOP RUN.
   MAISU-KEISAN.
       MOVE   KYUYO IN INPUT-REC TO MAN IN SYAIN,
                  SEN IN SYAIN, HYAKU IN SYAIN.
       DIVIDE SEN IN SYAIN BY 5000
                  GIVING GOSEN IN SYAIN
                  REMAINDER SEN IN SYAIN.
       DIVIDE HYAKU IN SYAIN BY 500
                  GIVING GOHYAKU IN SYAIN
                  REMAINDER HYAKU IN SYAIN.
```

図-3 COBOLプログラムの例

```
   with TEXT_IO; USE TEXT_IO;
⑤  package KINSYU_IO is
⑨    type KINSYU is (MAN, GOSEN, SEN, GOHYAKU, HYAKU);
     subtype KOSU is LONG_INTEGER range 0..9999;
     subtype KINGAKU_TYPE is LONG_INTEGER
                                range 0..999999999;
     type KINSYU_KOSU_TYPE is array (KINSYU) of KOSU;
     type SYAIN_TYPE is
       record
         BANGO       : STRING (1..6);
         NAMAE       : STRING (1..20);
         KYUYO       : KINGAKU_TYPE;
         KINSYU_KOSU : KINSYU_KOSU_TYPE;
       end record;
     function GET (F: FILE_TYPE) return SYAIN_TYPE;
     procedure SYAIN_SYUTSURYOKU (S: SYAIN_TYPE);
   end KINSYU_IO;

   with TEXT_IO; use TEXT_IO;
④  package body KINSYU_IO is
     package LIO is new INTEGER_IO (LONG_INTEGER);
       use LIO;
     function GET (F: FILE_TYPE) return SYAIN_TYPE is
       S: SYAIN_TYPE;
     begin
       GET (F, S. BANGO);
       GET (F, S. NAMAE);
       GET (F, S. KYUYO, 9);
       SKIP_LINE (F);
       return S;
⑥    exception
       when DATA_ERROR =>
           PUT ("INPUT DATA ERROR"); raise;
       when CONSTRAINT_ERROR =>
           PUT ("OVERFLOW"); raise;
     end GET;
     procedure SYAIN_SYUTSURYOKU
                          (S: SYAIN_TYPE) is
     begin
       PUT (S. BANGO);
       PUT (S. NAMAE);
```

識しているが，Adaでは，列挙型の使用により，プログラム上の変数の値のもつ意味がよくわかる．

4. Ada を使用するにあたって

（1） COBOL の強力な点は，次のようなところである．

① ソートマージ，報告書機能などが言語仕様上準備されている．

② 型変換が MOVE 命令により容易に行える．

③ 入出力の編集がしやすい．

これらは，Adaの言語仕様ではサポートされていない．しかし，Adaは，パッケージ機能などにより機能拡張が容易な言語仕様となっている．そこで，これらのCOBOL特有の機能をサポートするパッケージを作成し，それを共有することにより，Adaでも上記処理が行えるようになる．

（2） Adaでプログラムを作成する場合は，手続きや関数にこまかく分割するなどの構造的な設計を行ってから作成した方がよいプログラムがつくれる．

（3） Adaは，別のファイルに入っているプログラム単位間でもプログラムの翻訳時や実行時に手続き呼出しのパラメタ個数や型，値の範囲など，多くの誤りを見つけられるような機能を豊富に持っている．

（4） Adaは，列挙型や，loop，caseといった整備された制御構造のおかげで読みやすいプログラムが書ける．これは，保守時に大きな助けとなる．

（5） その他，本稿では触れなかったが，Adaは，タスク機能，マクロ機能の高級なものと考えられる汎用体，機械コード記述機能などを有しており，事務処理に限らず，種々の分野の記述に適用できる．

```
                PUT (S. KYUYO, 9);
                for I in KINSYU loop
                  PUT (S. KINSYU-KOSU (I), 5);
                end loop;
                NEW-LINE;
              end SYAIN-SYUTSURYOKU;
            end KINSYU-IO;
⑧           with KINSYU-IO, TEXT-IO; use KINSYU-IO, TEXT-IO;
            procedure KINSYU-KEISAN is
              SYAIN   : SYAIN-TYPE;
              INPUT-F: FILE-TYPE;
③             procedure MAISU-KEISAN
                    (KINGAKU      : in KINGAKU-TYPE;
                     KINSYU-KOSU: out KINSYU-KOSU-TYPE) is
                KINGAKU-TEMP: KINGAKU-TYPE;
                KINSYU-GAKU  : array (KINSYU) of KINGAKU-TYPE
                             := (10000, 5000, 1000, 500, 100);
                KOSU-TEMP    : KOSU;
              begin
                KINGAKU-TEMP := KINGAKU;
                for I in KINSYU loop
                  KOSU-TEMP := KINGAKU-TEMP/KINSYU-GAKU (I);
                  KINSYU-KOSU (I) := KOSU-TEMP;
                  KINGAKU-TEMP := KINGAKU-TEMP
                                - KOSU-TEMP * KINSYU-GAKU (I);
                end loop;
              exception
                when CONSTRAINT-ERROR =>
                  PUT ("OVERFLOW"); raise;
              end MAISU-KEISAN;
            begin
              OPEN (INPUT-F, IN-FILE, "INPUT-F");
              PUT-LINE ("BANGO    NAMAE          KYUYO"
                    & "10000  5000  1000  500  100)"
              loop
                SYAIN := GET (INPUT-F);
②⑧             MAISU-KEISAN(SYAIN. KYUYO, SYAIN. KINSYU-KOSU);
                SYAIN-SYUTSURYOKU (SYAIN);
              end loop;
⑥           exception
⑦             when END-ERROR =>
                CLOSE (INPUT-F);
              when others =>
                PUT ("ERROR OCCURS");
            end KINSYU-KEISAN;
```

図-4 Ada プログラムの例

5. おわりに

本稿では，COBOL プログラムと Ada プログラムを対比させ，Ada プログラムの記述方法を説明した．

COBOL は，事務処理用としてもっとも普及しているプログラミング言語であり，事務処理向きの機能と20年以上の実績による膨大な資産を有している．Adaにはその利点はないが，Ada言語使用の強みの1つであるパッケージを整備することにより，事務処理分野にも十分適用できるものと思われる．さらに，その読みやすさ，エラーチェックの強力さは，開発，保守を容易にするであろう．

参考文献

1) U. S. Department of Defense : Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815 A (1983)；邦訳—情報処理振興事業協会(編)：最新 Ada 基準文法書，bit 別冊，共立出版(1984)．

(昭和61年1月24日受付)