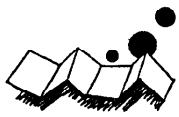
解　説



PL/I プログラマからみた Ada†

神　谷　刮　志††

1. はじめに

PL/I プログラマに対し，PL/I と Ada のプログラムを比較することにより Ada を紹介することを本稿の目的とする．

2章ではリスト処理を行う問題について，3章では同期処理と非同期処理を行う問題について，それぞれ PL/I と Ada のプログラムを対比させ，PL/I と Ada の相違を解説することにより Ada を概観することとする．

2. リスト処理を行うコーディング比較

2.1 問　題

部コード，社員名，および売上高をカード入力し，入力データリスト，および売上高が 1,000,000 円以上の社員リストを出力する．

（1） 入力形式

入力形式を図-1 に示す．

1 2 3	17 18 25		80
部コード ×(2)	社員名 ×(15)	売上高 ×(8)	未使用 ×(55)

図-1 入力形式

（2） 出力形式

出力形式を図-2 に示す．

（3） 処理内容

入力データを出力し，図-3 のように部別に入力情報をチェインして行く．売上高が 1,000,000 円以上の社員を部別に出力する．

2.2 PL/I プログラム例

図-4 に PL/I によるプログラム例を示す．

2.3 Ada プログラム例

図-5 に Ada によるプログラム例を示す．

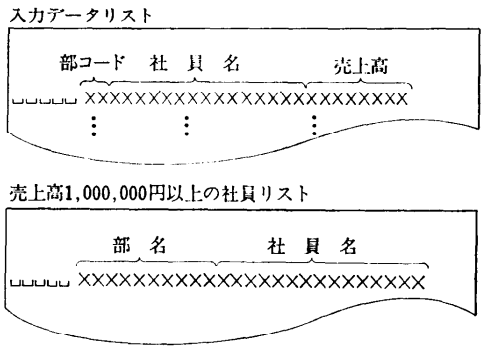


図-2 出力形式

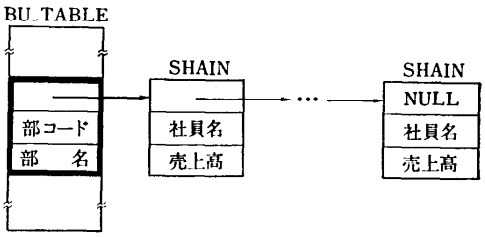


図-3 入力情報テーブルチェイン

2.4 解　説

（1） 表記法

Ada では英小文字の使用が許されており，英大文字と対応する英小文字は同じものとみなされる．したがって，読解性のよいプログラムが作成可能である．標準的な Ada の表記法は Ada 基準文法書[1]に記載されている例を参考にするとよい．

なお，Ada の文字セットは ISO 7 ビットコードを使用した文字セット（ASCII 図形記号）と規定されているので注意を要する．したがって，Ada ではプログラム中のコメントや文字列に日本語文字を書くことはできない．

Ada のコメント記述は　--　で始まり，行の終りまで続く形式であり，PL/I に比較して自然な書き方である（図-5(1)(2)(3)）．

Ada の数値リテラルは数字の間に下線を挿入する

† Ada for PL/I programmers by Katsushi KAMIYA (Fujitsu System Integration Laboratories Ltd.).

†† (株)富士通システム統合研究所

ことができ，位取り表示などに便利である（挿入した下線は数値リテラル値には影響しない）((57))．

（2） プログラム構造

PL/I の手続きに相当するものは，Ada では副プログラムである．副プログラムには手続き（procedure）と関数（function）があり，PL/I と同様である．副プログラムは副プログラム宣言（副プログラムの仕様を定義する）と副プログラム本体（(2)〜(65)）（副プログラムの実行を定義する）から成るが，副プログラム宣言は省略可能のため，本例題では省略した．

PL/I ではデータ宣言および内部手続きは有効範囲を考慮して，手続き内の任意の位置に置くことができるが，Ada では宣言部（begin の前）に置かなければならない．したがって，データ宣言（型宣言，算体宣言など），内部副プログラム宣言，内部副プログラム本体などを宣言部に置かねばならないため（(3)〜(34)），プログラムの作りによっては宣言部が大きくなってしまい，プログラム構造が分かりづらい場合がある．

Ada では文（動作の定義）は begin（必須）の後に置く（(36)〜(64)）．文の実行時に発生する例外を処理するため，文の列の後に例外ハンドラを置くことができる．PL/I でも ON 文で例外を扱うことができるが，Ada の exception の方が例外ハンドラとして機能が明確化され，固定した位置（副プログラム本体の末尾など）に置かれるため，プログラムの読解性に優れている．

（3） パッケージ

図-5 中の(1)の文脈節（with 節，use 節）で指定した TEXT_IO は言語の一部としてシステムが提供するテキスト入出力用のパッケージである．入出力はシステムにより既定のライブラリパッケージ（ライブラリ単位であるパッケージ）として与えられるため，利用者は使用する入出力パッケージを with 節で指定しなければならない．パッケージはソフトウェアのパッケージ化（部品化）を可能とする Ada の優れた言語機能であり，データ宣言（型宣言，オブジェクト宣言）あるいは副プログラムをパッケージ内に置くことにより利用者はデータあるいは副プログラムを共用することができる．すなわち，共通テーブルあるいは共通ルーチンはパッケージで実現可能であり，さ

```
EX01: PROC OPTIONS (MAIN);
  DCL SHAIN_AREA AREA (3000);
  DCL 1  SHAIN          BASED (SHAIN_PTR),
         2 SHAIN_PTR    POINTER,
         2 SHAIN_NAME CHAR (15),
         2 URIAGE       FIXED DEC (8);
  DCL BU_MAX         FIXED BIN INIT (5);
  DCL 1  BU_TABLE (BU_MAX),
         2 SHAIN_PTR POINTER INIT (5 NULL),
         2 BU_CODE    CHAR (2) INIT
           ('01', '02', '03', '04', '05'),
         2 BU_NAME    CHAR (10) INIT
           ('EIGYO_1', 'EIGYO_2', 'EIGYO_3',
            'EIGYO_4', 'EIGYO_5');
  DCL BU_NO          FIXED BIN;
  DCL 1  SHAIN_IN,
         2 BU_CODE      CHAR (2),
         2 SHAIN_NAME CHAR (15),
         2 URIAGE       FIXED DEC (8);
  DCL ENDFLAG     CHAR (3) INIT (' ');
  ON ENDFILE (SYSIN) ENDFLAG='END';
  GET EDIT (SHAIN_IN) (A (2), A (15), F (8));
  PUT PAGE;
  DO WHILE (ENDFLAG=' ');
    PUT EDIT (SHAIN_IN) (X (5), 2A, F (8));
    PUT SKIP;
    BU_NO=SHAIN_IN. BU_CODE;
    ALLOC SHAIN IN (SHAIN_AREA);
    SHAIN=SHAIN_IN. BY NAME;
    SHAIN. SHAIN_PTR=BU_TABLE (BU_NO). SHAIN_PTR;
    BU_TABLE (BU_NO). SHAIN_PTR=SHAIN_PTR;
    GET EDIT (SHAIN_IN) (A (2), A (15), F (8)) SKIP;
  END;
  PUT PAGE;
  DO J=1 TO BU_MAX;
    IF BU_TABLE (J). SHAIN_PTR ¬=NULL
      THEN DO;
        PUT EDIT (BU_TABLE (J). BU_NAME) (X (5), A);
        PUT SKIP (0);
        SHAIN_PTR=BU_TABLE (J). SHAIN_PTR;
        DO WHILE (SHAIN_PTR ¬=NULL);
          IF SHAIN. URIAGE >= 1000000
            THEN DO;
              PUT EDIT (SHAIN. SHAIN_NAME) (X (15), A);
              PUT SKIP;
            END;
          SHAIN_PTR=SHAIN_PTR-> SHAIN. SHAIN_PTR;
        END;
      END;
  END;
END EX01;
```

図-4 PL/I プログラム例

```
(1)  with TEXT_IO; use TEXT_IO; --context clause
(2)  procedure EX01 is                --subprogram body
(3)    type SHAIN; --incomplete type declaration
(4)    type LINK      is access SHAIN;
(5)    type SHAIN     is
(6)      record
(7)        SHAIN_PTR:   LINK;
(8)        SHAIN_NAME: STRING (1..15);
```

らに不特定な利用者が利用できるソフトウェア部品をパッケージ化して提供可能である．ライブラリ単位であるライブラリパッケージを利用するには上記で述べたように with 節で指定する必要がある（パッケージの使用例については図-12 および 3.4(2)項を参照のこと）．

パッケージ INTEGER-IO は上記パッケージ TEXT-IO の中で定義された整数型入出力用パッケージであるが，汎用体パッケージであるため，その具体化が必要である ((30)(32))．汎用体とは，型や算体などをパラメタとする，副プログラムまたはパッケージの「ひな形」のことである．汎用体は，副プログラム仕様またはパッケージ仕様の前に generic の指定と汎用体仮パラメタの指定を置いて宣言する．汎用体を使用するには型や算体などの実パラメタを指定して汎用体の具体化を行わねばならない．汎用体は「ひな形」プログラムの再利用を可能とする Ada の優れた言語機能である．

(4) 宣言と型

Ada では算体宣言などにより，実体を表す名前を宣言する．算体は一定の型をもっており，算体宣言において固有の型を指定する ((17)〜(29))．型の宣言は型宣言にて行う ((3)〜(16))．このように Ada では算体にそれぞれ固有の型をもたせ，算体の使用誤りを型チェックにより検出できるようにしているため，翻訳時にプログラムのデバグ推進を可能としている．

型には列挙型，整数型，実数型，配列型，レコード型，アクセス型などがあり，PL/I の文字列，算術，配列，構造体，ポインタなどを表すことができる．STRING は文字列を表す既定の型である ((27) ほか)．(既定の型とはシステムが型宣言している型のことであり，利用者は型宣言なしに使用できる．) INTEGER (さらに処理系により SHORT-INTEGER, LONG-INTEGER) は既定の整数型である ((17)(25)(9))．NATURAL と POSITIVE も既定の型であり，それぞれ 0 以上の整数と 1 以上の整数を表す ((28)(29)(24))．

array は配列，record はレコード型，access はアクセス型を表す ((18)(6)(12)(4))．アクセス型は PL/I のポインタに相当するが，アクセス値の指す型を指定しなければならない ((4) の SHAIN)．割当て子 new は指定された型の算体を作り

```
(9)       URIAGE:       LONG-INTEGER;
(10)    end record;
(11)  type BU          is
(12)    record
(13)      SHAIN-PTR: LINK;
(14)      BU-CODE:   STRING (1..2);
(15)      BU-NAME:   STRING (1..10);
(16)    end record;
(17)  BU-MAX:   ITEGER := 5;
(18)  BU-TABLE: array (1..BU-MAX) of BU
(19)            := ((null, "01", "EIGYO-1   "),
(20)                (null, "02", "EIGYO-2   "),
(21)                (null, "03", "EIGYO-3   "),
(22)                (null, "04", "EIGYO-4   "),
(23)                (null, "05", "EIGYO-5   "));
(24)  BU-NO     : POSITIVE;
(25)  URIAGE    : LONG-INTEGER;
(26)  SHAIN-PTR: LINK;
(27)  CARD      : STRING (1..80);
(28)  NWK       : NATURAL;
(29)  PWK       : POSITIVE;
(30)  package INT-IO is new INTEGER-IO (INTEGER);
(31)  use INT-IO;
(32)  package INTL-IO is new INTEGER-IO
(33)          (LONG-INTEGER);
(34)  use INTL-IO;
(35)  begin
(36)  NEW-PAGE;
(37)  while not END-OF-FILE loop
(38)    GET-LINE (CARD, NWK);
(39)    SET-COL (6);
(40)    PUT-LINE (CARD (1..25));
(41)    GET (CARD (1..2), BU-NO, PWK);
(42)    GET (CARD (18..25), URIAGE, PWK);
(43)    SHAIN-PTR = new SHAIN;
(44)    SHAIN-PTR. SHAIN-NAME = CARD (3..17);
(45)    SHAIN-PTR. URIAGE = URIAGE;
(46)    SHAIN-PTR. SHAIN-PTR =
(47)        BU-TABLE (BU-NO). SHAIN-PTR;
(48)    BU-TABLE (BU-NO). SHAIN-PTR = SHAIN-PTR;
(49)  end loop;
(50)  NEW-PAGE;
(51)  for J in 1..BU-MAX loop
(52)    if BU-TABLE (J). SHAIN-PTR /= null then
(53)      SET-COL (6);
(54)      PUT (BU-TABLE (J). BU-NAME);
(55)      SHAIN-PTR = BU-TABLE (J). SHAIN-PTR;
(56)      while SHAIN-PTR /= null loop
(57)        if SHAIN-PTR. URIAGE >=1_000_000 then
(58)          SET-COL (16);
(59)          PUT-LINE (SHAIN-PTR. SHAIN-NAME);
(60)        end if;
(61)        SHAIN-PTR = SHAIN-PTR. SHAIN-PTR;
(62)      end loop;
(63)    end if;
(64)  end loop;
(65)  end EX01;
```

図-5 Ada プログラム例

出し，それを指すアクセス値を返す ((43)) (PL/I の BASED変数に割付けポインタ値を返すALLOCATE 文に相当する). アクセス型はそのアクセス値が指す型が固定されているため，PL/I のポインタのような柔軟性はない.

算体宣言で初期値を与えるには := 式 で指定する ((17)(19)).

算体宣言に constant を指定すると，宣言される算体は定数となる (図-12 参照).

(5) 名 前 と 式

名前は宣言された実体を表す. レコード要素，アクセス値によって指される算体などの選択要素の参照は 前置子. 選択子 の形式で指定する ((44)〜(48) ほか). パッケージで宣言された実体はパッケージ名で修飾して参照する. ライブラリパッケージで宣言された実体は，use 節 ((3) 項参照) でそのライブラリパッケージを指定することにより ((1))，パッケージ名で修飾せずに参照可能となる ((38) GET-LINE, (40) PUT-LINE ほか). 配列の要素は添字付きで表す ((47)) ほか. 1次元配列の連続する要素列は添字に 下限..上限 を指定することにより参照可能となるため，記述性および読解性に優れている ((40)(41)(42)(44)).

集成式は，指定されたいくつかと要素値をレコード型または配列型の要素と結合する. (19)〜(23) の例では BU-TABLE の初期値を集成式で与えている.

Ada では式で使用できる演算子と演算項の型の組合せが規定されている. また，代入文の右辺の式と左辺の変数の型は同一でなければならない. したがって，プログラマの不注意による使用誤りが翻訳時にチェックされる.

(6) 入 出 力

テキスト入出力はパッケージ TEXT-IO の中で定義されている手続き GET, PUT などを使用して行う. ファイルパラメタを省略すると，標準入力ファイルまたは標準出力ファイルが仮定される. GET の第1パラメタに文字列を指定すると，その文字列から読出しが行われる ((41)(42)). GET-LINE と PUT-LINE は行単位の入力と出力を行う ((38)(40)). TEXT-IO は CHARACTER 型，STRING 型，数値型，および列挙型の項目に対する入出力を行うものであり，PL/I の流れ入出力に相当する. Ada ではテキストファイルの読込み，または書出し位置を合わせるため SET-COL 手続きを使用して入出力の桁位置を設定しておかねばならない ((39)(53)(58)). Ada では PL/I の編集に従う入出力のような編集指定ができないため，記述性に劣る. また，PL/I のリストに従う入出力およびデータに従う入出力も Ada にはない. 出力ファイルの改ページ，改行を行うには手続き NEW-PAGE, NEW-LINE をそれぞれ使用する ((36)).

入力ファイルの読出しでファイルの終端は関数 END-OF-FILE の参照により知ることができる (END-OF-FILE はファイル終端子が次にある場合，TRUE を返す) ((37)).

Ada には PL/I のピクチャ指定 (流れ入出力，およびデータ属性指定に使用) がないため，編集機能が見劣りする.

3. 同期処理と非同期処理のコーディング比較

3.1 問 題

学生コードと数学，国語および英語の得点をカードから入力し，3科目の合計点の高い順に印刷する. また，合計点を 30 点間隔に分け，各相当する人数の合計をグラフに印刷する.

(1) 入 力 形 式

入力形式を**図-6** に示す.

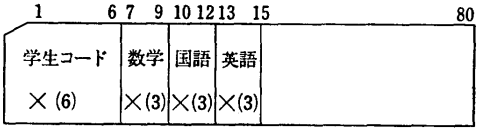


図-6 入力形式

(2) 出 力 形 式

出力形式を**図-7** に示す.

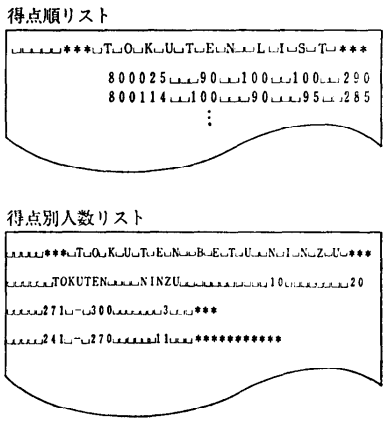


図-7 出力形式

(3) 処理内容

処理の流れを図-8 に示す.

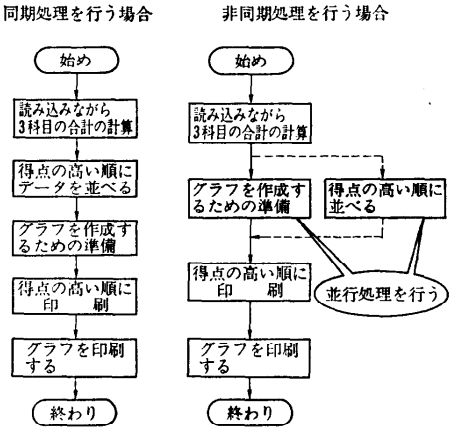


図-8 処理の流れ

3.2 PL/I プログラム例

(1) 同期処理を行う場合

同期処理の PL/I プログラム例を図-11 に示す.

(2) 非同期処理を行う場合

図-11 の同期処理プログラム例中の 〈1〉, 〈2〉, および 〈3〉 を以下のように変更する.

```
DCL SUBTSK TASK, EVT EVENT;      /*〈1〉*/
   CALL SORT(KEI, INNUM, SORTK)  /*〈2〉*/
     TASK(SUBTSK) EVENT(EVT);    /*〈2〉*/
   WAIT(EVT);                    /*〈3〉*/
```

3.3 Ada プログラム例

(1) 同期処理を行う場合

同期処理の Ada プログラム例を図-12 に示す.

(2) 非同期処理を行う場合

非同期処理の Ada プログラム例を図-13 に示す.

3.4 解 説

(1) 副プログラム呼出し

副プログラム呼出しは, PL/I 同様手続き呼出しまたは関数呼出しのいずれかであり, 実パラメタを指定できる. Ada の実パラメタ指定は位置による指定(図-12(108)) だけでなく, 名前による指定 ((108) SORT (SORTK 1 => SORTK, KEI 1 => KEI, INNUM 1 => INNUM);) も可能のため, パラメタ結合を明示することができる. 対応する実パラメタと仮パラメタの型は同一でなければならないため, PL/I のような暗黙の変換から生じる誤りを防止することができる.

呼出される副プログラムの仮パラメタにはモードが指定でき ((12)(13)), 仮パラメタの使用誤りを防止

```
EX02: PROC OPTIONS(MAIN);
  DCL INNUM       FIXED BIN;
  DCL (SUGAKU(150), KOKUGO(150),
       EIGO(150)) FIXED BIN;
  DCL G_NO(150)   CHAR(6);
  DCL KEI(150)    FIXED BIN;
  DCL SORTK(150) FIXED BIN;
  DCL NINZU(11)   FIXED BIN INIT((11)0);
  DCL AST  CHAR(80) INIT((80)'*');
  DCL TEN(11)  CHAR(10) INIT(
       '271 - 300 ', '241 - 270 ',
       '211 - 240 ', '181 - 210 ',
       '151 - 180 ', '121 - 150 ',
       ' 91 - 120 ', ' 61 -  90 ',
       ' 31 -  60 ', '  0 -  30 ',
       ' GOKEI    ');
  DCL TEN1(10)  FIXED BIN INIT(271,
    241,211,181,151,121,91,61,31,0);
  DCL SORT        ENTRY;
  DCL WK          FIXED BIN;
  DCL ENDFLAG     CHAR(3) INIT (' ');
                                  /*〈1〉*/
  ON ENDFILE(SYSIN) ENDFLAG='END';
  INNUM=0;
  GET EDIT(G_NO(1), SUGAKU(1),
    KOKUGO(1), EIGO(1)), (A(6), 3F(3));
  DO WHILE(ENDFLAG=' ');
    INNUM=INNUM+1;
    KEI(INNUM)=SUGAKU(INNUM)+
      KOKUGO(INNUM)+EIGO(INNUM);
    WK=INNUM+1;
    GET EDIT(G_NO(WK), SUGAKU(WK), KOKUGO(WK)
      , EIGO(WK)) (A(6), 3F(3)) SKIP;
  END;
  IF INNUM ¬=0
    THEN DO;
      CALL SORT(KEI, INNUM, SORTK); /*〈2〉*/
      DO J=1 TO INNUM;
        DO K=1 TO 10;
          IF KEI(J) > TEN1(K)
            THEN DO;
              NINZU(K)=NINZU(K)+1;
              NINZU(11)=NINZU(11)+1;
              K=10;
            END;
          END;
        END;
                                  /*〈3〉*/
      CALL PRINT1;
      CALL PRINT2;
    END;
PRINT1: PROC;
  PUT EDIT('     *** T O K U T ' ||
    'E N  J U N  L I S T ***') (A) PAGE;
  PUT SKIP;
  DO J=1 TO INNUM;
    WK=SORTK(J);
    PUT EDIT(G_NO(WK), SUGAKU(WK),
      KOKUGO(WK), EIGO(WK), KEI(WK))
      (X(11), A, 4(X(2), A(3))) SKIP;
  END;
```

することが可能である（in モードの仮パラメタ（(12)の KEI 1, INNUM 1）に値を設定したり，out モード仮パラメタ（(13)の SORTK 1）から値を読出すことはできない）．

（2） パッケージ

パッケージは，パッケージ仕様（(1)〜(10)）とパッケージ本体（(11)〜(32)）の2つの部分から成る．利用者がアクセスできるのはパッケージ仕様のみであり，パッケージ本体は利用者から隠ぺいされている．したがって，外部インタフェースがパッケージ仕様により明確化されており，インタフェースミスの防止に役立つ．また，パッケージ利用者はパッケージ仕様のみ存在すれば（翻訳されていれば），自らのプログラムを翻訳することができるためプログラムの並行開発が可能である（使用するパッケージ本体の翻訳完了を待つ必要はない）．

（3） タ　ス　ク

Ada にはタスク機能があり，プログラムの並列実行が可能である．タスクは，タスク仕様（**図-13** ③〜⑦）とタスク本体（⑧）から成る（separate については（4）項参照）．タスク仕様では他からの呼出しエントリを宣言する（④⑥）．

タスクの起動は PL/I では CALL 文によるが，Ada ではタスク本体の宣言部の確立で起る．異なるタスクの起動および実行は並列に進行する．タスク間の同期はエントリの呼出しを行うタスクと accept 文の実行によりエントリ呼出しを受付けるタスクとの間のランデブにより達成される（⑨と⑰，⑩と⑲）．

図-9 で①と①′，⑪と⑪′，⑱と⑱′はそれぞれ並列に実行され，同期⑦および同期⑭でそれぞれランデブする（同期点に早く到達したタスクの実行が待期状態となる）．

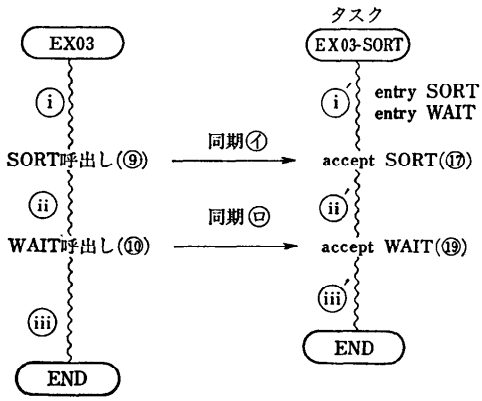


図-9 タスクの同期

```
END PRINT1;
PRINT2: PROC;
  PUT EDIT('      ＊＊＊ T O K U T E' ||
    ' N  B E T U  N I N Z U ＊＊＊')(A) PAGE;
  PUT EDIT('      TOKUTEN    NIN'    ||
    'ZU          10        20  '     ||
    '     30       40        5'      ||
    '0        60       70       '    ||
    '  80') (A) SKIP(2);
  DO J=1 TO 11;
    PUT EDIT(TEN(J), NINZU(J))
      (X(5), A, X(5), F(2)) SKIP(2);
    IF (NINZU(J) ¬=0) & (J ¬=11)
      THEN PUT EDIT(SUBSTR(AST, 1,
           NINZU(J)) (X(3), A);
  END;
END PRINT2;
END EX02;
SORT: PROC(KEI1, INNUM1, SORTK1);
  DCL (KEI1(150), SORTK1(150))  FIXED BIN;
  DCL INNUM1      FIXED BIN;
  DCL WK          FIXED BIN;
  DO J=1 TO INNUM1;
    SORTK1(J)=J;
  END;
  DO J=1 TO INNUM1-1;
    DO K=J+1 TO INNUM1;
      IF KEI1(SORTK1(J))<KEI1(SORTK1(K))
        THEN DO;
          WK=SORTK1(K);
          SORTK1(K)=SORTK1(J);
          SORTK1(J)=WK;
        END;
    END;
  END;
END SORT;
```

図-11 PL/I プログラム例（同期処理の例）

PL/I と Ada のタスク機能の相違を**図-10** に示す．

（4） 副　単　位

Ada プログラムの実本体（副プログラム本体，パッケージ本体，またはタスク本体）は本体受け（⑧）を指定することにより，副単位（⑫）として分離翻訳することができる．したがって，副単位を使用してプログラムを分割することにより，階層的なプログラム開

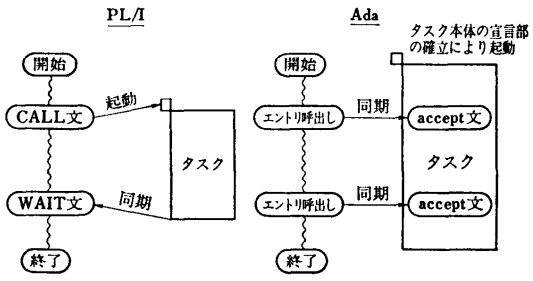


図-10 PL/I と Ada のタスク機能の相違

```
( 1 )package EX02_SORT is
( 2 )  MAXNINZU     : constant := 150;
( 3 )  MAXTOKUTEN: constant := 300;
( 4 )  type TOTAL is array(1..MAXNINZU) of
( 5 )        INTEGER range 0..MAXTOKUTEN;
( 6 )  type SUFFIX is array(1..MAXNINZU) of
( 7 )        INTEGER range 1..MAXNINZU;
( 8 )  procedure SORT(KEI1: in TOTAL; INNUM1
( 9 )     : in INTEGER; SORTK1: out SUFFIX);
(10)end EX02_SORT;
(11)package body EX02_SORT is
(12)  procedure SORT(KEI1: in TOTAL; INNUM1:
(13)    in INTEGER; SORTK1: out STFFIX) is
(14)    WK     : INTEGER;
(15)    SORTW: SUFFIX;
(16)  begin
(17)    for J in 1..INNUM1 loop
(18)      SORTW(J) := J;
(19)    end loop;
(20)    for J in 1..INNUM1-1 loop
(21)      for K in J+1..INNUM1 loop
(22)        if KEI1(SORTW(J)) <
(23)          KEI1(SORTW(K)) then
(24)          WK := SORTW(K);
(25)          SORTW(K) := SORTW(J);
(26)          SORTW(J) := WK;
(27)        end if;
(28)      end loop;
(29)    end loop;
(30)    SORTK1 := SORTW;
(31)  end SORT;
(32)end EX02_SORT;
(33)with TEXT_IO; use TEXT_IO;
(34)use EX02_SORT;
(35)procedure EX02 is
(36)  CARD  : STRING(1..80);
(37)  INNUM: INTEGER;
(38)  SUGAKU, KOKUGO, EIGO: array(1..MAXNINZU)
(39)              of INTEGER range 0..100;
(40)  G_NO: array(1..MAXNINZU) of STRING(1..6);
(41)  KEI    : TOTAL;
(42)  SORTK: SUFFIX;
(43)  NINZU : array(1..11) of INTEGER
(44)      range 0..MAXNINZU := (1..11=> 0);
(45)  AST    : STRING(1..80):=(1..80=>'*');
(46)  TEN    : array(1..11) of STRING(1..10) :=
(47)    ("271 - 300 ", "241 - 270 ", "211 - 240 "
(48)    , "181 - 210 ", "151 - 180 ", "121 - 150 "
(49)    , " 91 - 120 ", " 61 -  90 ", " 31 -  60 "
(50)    , "  0 -  30 ", "  GOKEI   ");
(51)  TEN1    : array(1..10) of INTEGER:=(271
(52)    ,241,211,181,151,121,91,61,31,0);
(53)  NWK    : NATURAL;
(54)  PWK    : POSITIVE;
(55)  package INT_IO is new
(56)    INTEGER_IO(INTEGER);
(57)  use INT_IO;
(58)  procedure PRINT1 is
(59)    WK : INTEGER;
(60)  begin
(61)    NEW_PAGE;
(62)    PUT_LINE("     *** T O K U T " &
(63)      "E N  J U N  L I S T ***");
(64)    NEW_LINE;
(65)    for J in 1..INNUM loop
(66)      WK := SORTK(J);
(67)      NEW_LINE;
(68)      SET_COL(12); PUT(G_NO(WK));
(69)      SET_COL(20); PUT(SUGAKU(WK), 3);
(70)      SET_COL(25); PUT(KOKUGO(WK), 3);
(71)      SET_COL(30); PUT(EIGO(WK), 3);
(72)      SET_COL(35); PUT(KEI(WK), 3);
(73)    end loop;
(74)  end PRINT1;
(75)  procedure PRINT2 is
(76)  begin
(77)    NEW_PAGE;
(78)    PUT_LINE("     *** T O K U T " &
(79)      "E N  B E T U  N I N Z U ***");
(80)    NEW_LINE;
(81)    PUT_LINE("      TOKUTEN    NINZU   "
(82)      & "       10       20       30"
(83)      & "       40       50       60"
(84)      & "       70       80");
(85)    NEW_LINE;
(86)    for J in 1..11 loop
(87)      SET_COL(6);  PUT(TEN(J));
(88)      SET_COL(21); PUT(NINZU(J), 2);
(89)      if (NINZU(J) /=0) & (J /=11) then
(90)        SET_COL(26); PUT(AST(1..NINZU(J));
(91)      end if;
(92)      NEW_LINE(2);
(93)    end loop;
(94)  end PRINT2;
(95)begin
(96)  INNUM := 0;
(97)  while not END_OF_FILE loop
(98)    GET_LINE(CARD, NWK);
(99)    INNUM := INNUM+1;
(100)   G_NO(INNUM) := CARD(1..6);
(101)   GET(CARD(7..9), SUGAKU(INNUM), PWK);
(102)   GET(CARD(10..12), KOKUGO(INNUM), PWK);
(103)   GET(CARD(13..15), EIGO(INNUM), PWK);
(104)   KEI(INNUM) := SUGAKU(INNUM)+
(105)       KOKUGO(INNUM)+EIGO(INNUM);
(106) end loop;
(107) if INNUM /=0 then
(108)   SORT(KEI, INNUM, SORTK);
(109)   for J in 1..INNUM loop
(110)     for K in 1..10 loop
(111)       if KEI(J) > TEN1(K) then
(112)         NINZU(K) := NINZU(K)+1;
(113)         NINZU(11) := NINZU(11)+1;
(114)         exit;
(115)       end if;
(116)     end loop;
(117)   end loop;
(118)   PRINT1;
(119)   PRINT2;
(120) end if;
(121)end EX02;
```

図-12 Ada プログラム例 (同期処理の例)

発が可能となる．これは Ada の優れた言語機能の１つである．

（5） 翻 訳 単 位

Ada の翻訳単位は副プログラム宣言，パッケージ宣言，汎用体宣言，汎用体具体化，副プログラム本体，パッケージ本体，副単位から成り，各翻訳単位は図-12 のように一度に翻訳してもよいし，いくつかに分けて翻訳してもよい．すなわち，１つのソースファイル中に複数の翻訳単位を置き，一度に翻訳することができるため，PL/I のように外部手続き単位の翻訳を意識する必要はない．

主プログラム(初期起動プログラム)として，翻訳単位である副プログラムが必ず存在しなければならない．

Ada では翻訳順序が規定されており，翻訳単位は文脈節に指定されたすべてのライブラリ単位のあとで翻訳しなければならない．副プログラム本体またはパッケージ本体は副プログラム宣言（もし存在すれば），またはパッケージ宣言のあとで翻訳しなければならない．副単位は，親の翻訳のあとで翻訳しなければならない．このような翻訳順序はコンパイラにより自動的にチェックされるため，特にプログラム修正時，依存関係にある翻訳単位の翻訳もれを防ぐことができる．これは Ada の優れた機能の１つである．

4. Ada 使用にあたって

PL/I プログラマにとって，Ada の型宣言，入出力などは冗長で不便に思われるかもしれない．また，PL/I の PICTURE，省略時解釈などに相当する機能がないことは非常に使いづらく感じられるかもしれない．しかし，ソフトウェア開発の大きなテーマである，信頼性の向上と保守の容易性，生産性の向上とプログラム開発の容易性などに関しては PL/I より Ada の方がより強く考慮されている．そのため，Ada は PL/I より言語仕様が厳しくなっており，プログラムの翻訳時に多くの誤りを検出し，修正することが可能である．本稿で示した PL/I と Ada のプログラム比較は PL/I プログラムを主体にして説明しているが，今後はさらに Ada の言語機能そのものを体系的に理解することを念願する．

5. おわりに

Ada はまだ若い言語であるため，目標とする Ada プログラム開発支援環境が十分に構築されているとは言い難い．今後，支援ツール類の充実，コンパイラの性能向上，パッケージを利用した Ada ソフトウェア部品の充実などがはかられれば，Ada の適用範囲が拡大してゆくものと予想される．Ada は現在，最も注目されているプログラム言語の１つであり，今後の動向を注意深く見守る必要がある．

```
①  with TEXT_IO; use TEXT_IO;
②  procedure EX03 is
       MAXNINZU: constant = 150;            ( 2 )
            [ 宣    言 ]                       ≀
            INTEGER range 1..MAXNIZU; ( 7 )
③    task EX03_SORT is
④      entry SORT(KEI1: in TOTAL;
⑤                INNUM1: in INTEGER);
⑥      entry WAIT(SORTK2: out SUFEIX);
⑦    end;
⑧    task body EX03_SORT is separate;
     CARD: STRING(1..80); ( 36 )
            [ 宣    言 ]
          [ PRINT1  手続き ]       同期処理を行なう
          [ PRINT2  手続き ]       場合のプログラム
            [ 文  の  列 ]         例と同じ.
     if INNUM /=0 then         (107)
⑨      EX03_SORT. SORT(KEI, INNUM); --SORT call
       for J in 1..INNUM loop  (109)
          [ 文  の  列 ]           ≀
       end loop;               (117)
⑩      EX03_SORT. WAIT(SORTK); --WAIT call
       PRINT1;                 (118)
       PRINT2;                 (119)
     end if;                   (120)
⑪  end EX03;
⑫  separate(EX03)
⑬  task body EX03_SORT is
⑭    SORTK1: TOTAL;
⑮    WK     : INTEGER;
⑯  begin
⑰    accept SORT(KEI1: in TOTAL; INNUM1
⑱             : in INTEGER); --rendezvous
       for J in 1..INNUM1 loop ( 17 )
          [ 文  の  列 ]           ≀
       end loop;               ( 29 )
⑲    accept WAIT(SORTK2: out SUFFIX) do
⑳      SORTK2 = SORTK1;
㉑    end;                        --rendezvous
㉒  end EX03_SORT;
```

図-13 Ada プログラム例（非同期処理の例）

謝辞 本稿の執筆にあたり Ada 分科会主査 NTT 福山峻一調査役はじめ Ada 分科会メンバ各位のご指導，ご支援に深く感謝致します．

参 考 文 献

1) U. S. Department of Defense: Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815 A (1983)；邦訳一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 60 年 2 月 7 日受付）