解　説

Cプログラマからみた Ada†

中村英夫††　森本真一†††

1. はじめに

本稿は，C言語を知っているプログラマが Ada 言語でプログラミングしようとしたときにどうすれば良いのかまたどんな問題点があるのかをカーニハン，リッチーの「プログラミング言語C」[1]の例題のいくつかを用いて解説するものである．はじめに小さい例題でいくつかの言語仕様上の相違を解説し，その後にそれらを組み合わせた例題でプログラム構成上の相違と対処法を解説する．6. ではCと Ada の言語の性格のちがいや Ada を使用するときの留意点を解説する．もちろん少ない頁数で Ada の言語仕様をすべて説明することはできないので必要最少限に留める．なお，各機能のより詳細な内容については，この特集の他の記事または文法書[2]を参照されたい．

2. 文字列代入問題における比較

2.1 問　題

“文字列tの内容を文字列sにコピーする関数 strcpy を作成する” といった基本的な問題である．しかし，思いのほか相異がある．

2.2 Cプログラム

```
strcpy( s , t ) /* copy t to s*/
char *s , *t ;
{
        while( *s++ = *t++ ) ;
}
```

図-1　文字列代入問題のCコーディング例

2.3 Ada プログラム

```
procedure STRCPY(S:in out STRING; T:in STRING) is
    begin
        if T'LENGTH < S'LENGTH then
            S:=(S'FIRST..S'LAST=>' ');
            S(S'FIRST..S'FIRST+T'LENGTH-1)
              :=T(T'FIRST..T'LAST);
        else
            S(S'FIRST..S'LAST)
              :=T(T'FIRST..T'FIRST+S'LENGTH-1);
        end if;
end STRCPY;
```

図-2　文字列代入問題の Ada コーディング例

2.4 解　説

2.4.1 型の厳密性

Cは一般に型はそれほど厳密にはチェックしない．たとえば，int, char, bool を同一視したりする．また値も，0, null, '\0', false などは同じと見る．このための利点もあるが，大規模なプログラムになれば誤りの生じる可能性も高くなる．型について Ada はCとは逆に厳密なチェックを行っている．

2.4.2 配列の添字範囲

Ada の場合は部分配列の代入ができるのでそれを用いている．配列の添字範囲のチェックを行うためにCでは無視されていた添字範囲に対する配慮が必要である．代入される文字列Sの大きさが代入する文字列Tの大きさより小さいときでもCの場合は範囲を越えて実行されてしまう．もし，それを避けようとするとサイズに関する引数を与えて処理しなければならない．

したがってここでは，TとSのサイズの大小をT'LENGTH と S'LENGTH を用いて場合分けしている．

また，配列の添字範囲が Ada の場合は可変なので

† Ada for C Programmers by Hideo NAKAMURA (Toshiba Corporation, Systems and Software Engineering Division) and Shin-ichi MORIMOTO (NEC Corporation, Microcomputer Software Development Laboratory).

†† (株)東芝システム・ソフトウェア技術推進部
††† 日本電気(株)マイクロコンピュータ・ソフトウェア開発本部

それに対する配慮がしてある．そのために少し範囲の計算が煩雑になっている．Cの場合のように配列の添字は0から始まると仮定してしまえばずっと簡単になる．

2.4.3 文字列の終端

文字列の終りが '\0' (NUL コード) で終了しているといった制約は Ada にはない．

したがって文字列のコピーという機能に対してコピーされない部分に空白を埋めている(4行目)．ここでは最初にすべてに空白を埋めたあとで部分配列の代入によってコピーを行っている．この処理は必要はないかも知れないが空白の代入を行わないとコピーされた結果に以前はいっていたゴミの文字が残る可能性がある．しかし，そのために空白を含む文字列をうまく扱えない．空白以外の文字コードを適用する場合にも同様の注意が必要である．

ここでは文字列を単なる STRING 型としたが，可変長文字列は長さを判別子としたレコード型として定義した方が自然かもしれない．

3. 月名検索問題における比較

3.1 問 題

"月の番号を与えたときに月名文字列を返す関数 month_name を作成する" という簡単な問題である．

3.2 Cプログラム

```
char *month_name( n )
int n ;
{
        static char *name[] = {
                "illegal month" ,
                "January" ,
                "February" ,
                "March" ,
                "April" ,
                "May" ,
                "June" ,
                "July" ,
                "August" ,
                "September" ,
                "October" ,
                "November" ,
                "December"
        };
        return((n<1 || n>12)?name[0]:name[n]);
}
```

図-3 月名探索問題のCコーディング例

3.3 Ada プログラム

```
function MONTH_NAME(N:in INTEGER) return STRING is
        type NAME is
                  (ILLEGAL_MONTH ,
                   JANUARY       ,
                   FEBRUARY      ,
                   MARCH         ,
                   APRIL         ,
                   MAY           ,
                   JUNE          ,
                   JULY          ,
                   AUGUST        ,
                   SEPTEMBER     ,
                   OCTOBER       ,
                   NOVEMBER      ,
                   DECEMBER      );
        begin
                if N<1 or N>12 then
                        return NAME'IMAGE(NAME'VAL(0));
                   else return NAME'IMAGE(NAME'VAL(N));
                end if;
        end MONTH_NAME;
```

図-4 月名検索問題の Ada コーディング例

3.4 解 説

3.4.1 列 挙 型

Adaには，定まった値を書き並べて1つの型を作る機能がある．それを列挙型と言う．

たとえば，信号機を制御するプログラムで信号の色を type COLOR is (GREEN, YELLOW, RED); などと型宣言して用いることができる．この例の場合は各月の名前を列挙型として型宣言している．列挙型の要素には0からはじまる位置数が列挙の順につけられる．列挙型には既定の属性として位置数や順に関するものが備っており，それを扱う属性指示子がある．これを用いる場合には型変数名と属性指示子の間に ' をはさんで使う．

NAME'IMAGE や NAME'VAL はその例であり，この場合は NAME'VAL (N) で位置数Nの列挙値を取り出し，NAME'IMAGE (NAME'VAL (N)) でその列挙値の表記 (DECEMBER など) を文字列として取り出している．これにより関数 MONTH-NAME は月の名前の文字列を返す．Ada には多種類の型宣言があるので，問題に合ったものを選択することができる．

4. ソート問題における比較

4.1 問　題

"文字列を昇順に並べる関数 sort を作成する" という問題だがここでは要素比較関数 comp と要素交換関数 exch の取扱いが問題となる．

4.2 Cプログラム

```
sort( v , n , comp , exch )
char *v[];
int n;
int (*comp)() , (*exch)();
{
        int gap , i , j;
        for(gap=n/2 ; gap>0 ; gap /= 2)
                for(i=gap ; i<n ; i++)
                        for(j=i-gap ; j>=0 ;j-=gap){
                                if((*comp)(v[j] , v[j+gap]) <= 0)
                                        break;
                                (*exch)(&v[j] , &v[j+gap]);
                        }
}
```

図-5 ソート問題のCコーディング例

4.3 Ada プログラム

```
with TYPEDEFS; use TYPEDEFS;
generic
        with function COMP(S1,S2:STRING) return BOOLEAN;
        with procedure EXCH(S1,S2:in out STRING);
procedure SORT(V:in out STRARRAY);

procedure SORT(V:in out STRARRAY) is
        GAP,I,J,N,NO:INTEGER;
begin
        NO:=V'FIRST;
        N:=V'LENGTH;
        GAP:=N/2;
        while GAP>0 loop
                I:=GAP;
                while I<N loop
                        J:=I-GAP;
                        while j>=0 loop
                                exit when COMP(V(NO+J),V(NO+J+GAP));
                                EXCH(V(NO+J),V(NO+J+GAP));
                                J:=J-GAP;
                        end loop;
                        I:=I+1;
                end loop;
                GAP:=GAP/2;
        end loop;
end SORT;
```

図-6 ソート問題の Ada コーディング例

4.4 解 説

文字列の配列の型宣言は，パッケージ TYPEDEFS の中で次のように宣言されているとする．

```
type STRARRAY is
  array (INTEGER range<>) of STRING (1..20);
```

4.4.1 汎 用 体

Cプログラムでは，関数へのポインタを引数として与えることにより実行時に任意の関数を実行できる(*comp, *exch). Ada にはこの機能はないが，汎用体機能があるのでそれを用いれば類似の機能が実現できる．汎用体 (generic) は，プログラム中に関数などを丁度穴のように設定する機能である．穴開きプログラムを使うとき (new を用いる) になにか決まった関数などを指定すれば穴を埋めることができる．ただし汎用体を用いる場合には，翻訳時に相手が定まるので実行時に選択することはできない．

この例で汎用体を利用するには次のようなプログラムを用いる．

```
with SORT, TYPEDEFS; use TYPEDEFS;
procedure MAIN is
VV: STRARRAY (1..100);
--その他の型宣言など
function MYCOMP (S1, S2: STRING) return BOOLEAN is
--MYCOMP 本体
end MYCOMP;
procedure MYEXCH (S1, S2: in out STRING) is
--MYEXCH 本体
end MYEXCH;
--汎用体の具体化 (次行)
procedure MYSORT is new SORT (MYCOMP, MYEXCH);
begin
  ...
  MYSORT (VV); --具体化された汎用体の呼び出し
  ...
end;
```

ただし，各 STRING 型変数の長さは一致しているものとする．

5. 最長行の決定問題における比較

5.1 問 題

ここでは

"入力データ中で最も長い行を出力する"

という問題をCと Ada でコーディングを行い，それに基づいてCと Ada の機能の違いを説明する．

両プログラムとも次のような構造になっている．

```
while (入力の終わりでない)
  if (読込んだ行が以前の最長の行より長い)
      その行の長さと内容を登録する
  if (読込んだ行がある)
      最長行の内容を出力する．
```

これらの操作を行うためにCプログラムでは行を読み込む getline，行のコピーを行う copy という関数を宣言している．Ada プログラムでは行を読み込む GET，行を出力する PUT-LINE，行の長さを比較する ">" という副プログラムを宣言している．

5.2 Cプログラム

次頁図-7 に掲載してある．

5.3 Ada プログラム

p. 288 図-8 に掲載してある．

5.4 解 説

この例題では，Cと Ada の機能のうち

- プログラムの構造
- 副プログラム (手続き，関数)
- パッケージ
- 型
- 入出力

について解説する．

5.4.1 プログラムの構造

Cでは全体のプログラムは関数や外部変数の宣言の並びという形式をしているが，Ada では副プログラム，パッケージの宣言の並びという形式をしている．(Ada では他にタスクというものがあるがCに対応するものがないので説明は省略する)．この例題ではCプログラムは，main, getline, copy という関数の宣言からなり，Ada プログラムは LINE-CLASS というパッケージの仕様と本体の宣言および MAIN という手続きの宣言からなっている．Ada では，これらの各宣言は翻訳単位と呼ばれ独立に翻訳できる．翻訳単位間の参照関係を表すために with 節が用いられる．with 節によってコンパイラは翻訳順序や再翻訳が必要な翻訳単位を決定できる．この例題ではたとえば MAIN の内部で LINE-CLASS, TEXT-IO 内で宣言された型や副プログラムを利用しているので，MAIN の前の with 節で LINE-CLASS, TEXT-IO を指定している．

5.4.2 副プログラム (手続き，関数)

Cでは副プログラムはすべて関数である (返す型を明示しなければ int 型を返すものとみなされる) が Ada では手続きと関数は厳密に区別される．たとえばCでは関数呼出しを単独で文とすることができ，同じ関数内に値を返す return 文と返さない return 文が

```
#include <stdio.h>
#define  MAXLINE   1000   /*  max input line size  */
main ()   /* find longest line */
{
        int  len, max;
        char  line[MAXLINE];     /*  current input line   */
        char  save[MAXLINE];     /*  longest line, saved  */
        max = 0;
        while ( (len = getline(line,MAXLINE)) > 0 )
                if (len > max)
                {
                        max = len;
                        copy(line,save);
                }
        if (max>0)     /*  there was a line  */
                printf("longest LINE is\n%s\n", save);
}

getline(s,lim)
char  s[];
int  lim;
{
        int  c, i;
        i = 0;
        while ( --lim>0 && (c=getchar()) != EOF && c != '\n' )
                s[i++] = c;
        if (c == '\n')
                s[i++] = c;
        s[i] = '\0';
        return i;
}

copy(s1,s2)      /*  copy s1 to s2  */
char  *s1, *s2;
{
        while (*(s2++) = *(s1++));
}
```

図-7 最長行の決定問題のCコーディング例

あってもよい．しかし Ada では手続き呼出ししか文にできず，関数内に値を返さない return 文があったり手続き内に値を返す return 文があったりしてはならない．

Cでは関数の内部で関数を宣言することはできないが，Ada では宣言できる．このためプログラムで用いる副プログラムを構造的に宣言できる．

Ada のパラメタの特徴はモードを指定できることである．モードは次の3種類があり指定を省略すると in モードになる．

- in モード　本体での値の読み出しだけ許す．
- out モード　本体での値の更新だけ許す．
- in out モード　本体での読み出しと更新を許す．

in モード，in out モードのパラメタの値は呼出し後に対応する実パラメタにコピーされる．Cのパラメタにはモードという概念はないが値呼び (call by value) という点では Ada の in モードに相当する．しかしCでもアドレスをパラメタとすることにより Ada の in out モードに相当する機能を実現できる．

Ada では既定義の演算子を再定義できる．このためプログラムで定義した型に対しても既存の演算子記号を用いた式が記述できるのでプログラムが読みやすくなる．このプログラムでは “>” という演算子を L_TYPE 型の要素に対して再定義している．

このこととも関連するが，Ada では複数の異なる副プログラムを同じ名前で宣言できる（これを多重定義という）．このような副プログラムが呼出された場合は，処理系がパラメータの型や返す型に基づきそのなかのどれが呼出されたかを判定する．この機能を用いると類似した機能の副プログラムは同じ名前にするこ

```
package LINE_CLASS is
  subtype SIZE_TYPE is INTEGER range 0..1000;
  type L_TYPE(SIZE : SIZE_TYPE:=132) is private;
  EMPTY : constant L_TYPE;
  function GET(LINE : in out L_TYPE) return BOOLEAN;
  procedure PUT_LINE(LINE : in L_TYPE);
  function ">"(X, Y : L_TYPE) return BOOLEAN;
private
  type L_TYPE(SIZE : SIZE_TYPE:=132) is
    record
      CONTENTS : STRING(1..SIZE);
      LEN    : SIZE_TYPE:=0 ;
    end record;
  EMPTY : constant L_TYPE
   :=( SIZE=>132, CONTENTS=>(OTHERS=>' '), LEN=>0 );
end LINE_CLASS;

with TEXT_IO, LINE_CLASS;
use  TEXT_IO, LINE_CLASS;
procedure MAIN is     --  find longest line
  MAXLINE : constant :=1000;  -- max input line size
  LINE : L_TYPE(MAXLINE);  --  current input line
  SAVE : L_TYPE;     --  longest line, saved
begin
  SAVE := EMPTY;
  while GET(LINE)  loop
    if LINE>SAVE
      then  SAVE:=LINE;
    end if;
  end loop;
  if SAVE /= EMPTY then     --  there was a line
    PUT_LINE("longest LINE is");
    PUT_LINE(SAVE);
  end if;
end MAIN;

with TEXT_IO;   use TEXT_IO;
package BODY LINE_CLASS is
  function GET(LINE : in out L_TYPE) return BOOLEAN is
    C : CHARACTER;
  begin
    LINE.LEN:=0;
    loop
      if END_OF_FILE then
        return FALSE;
      end if;
      exit when END_OF_LINE ;
      LINE.LEN:=LINE.LEN+1;
      GET(C);
      if LINE.LEN < LINE.SIZE then
        LINE.CONTENTS(LINE.LEN):=C;
      end if;
    end loop;
    SKIP_LINE;
    return TRUE;
  end  GET;
  procedure PUT_LINE(LINE : in L_TYPE) is
  begin
    PUT_LINE( LINE.CONTENTS(1..LINE.LEN) );
  end PUT_LINE;
  function ">"(X, Y : L_TYPE) return BOOLEAN is
  begin
    return X.LEN>Y.LEN;
  end ">";
end LINE_CLASS;
```

図-8 最長行の決定問題の Ada コーディング例

とができるのでプログラムが読みやすくなる．この例題でもCでは行を読み込む関数は getline，文字を読み込む関数は getchar と名前が異なっているが，Ada では両方とも GET と同じ名前にしている．

5.4.3 パッケージ

Ada ではパッケージという構成要素がある．パッケージは仕様と本体からなり，仕様はさらに可視部と密閉部に分かれる．本体や密閉部はない場合もある．仕様の中の可視部は大域的な変数，型，副プログラムなどの宣言からなり，密閉部は密閉型（これについては後述）に関する宣言からなる．また本体は局所的な型や変数の宣言からなる．この例題では LINE_CLASS というパッケージの可視部で L-TYPE という密閉型，EMPTY というその型の定数，GET，PUT_LINE という手続きや ">" という演算子（関数）の仕様を宣言し，密閉部で L-TYPE の実際の構造と EMPTY の実際の値を宣言している．また本体では仕様で宣言された各副プログラムの本体を宣言している．

パッケージの外部からは，パッケージの可視部で宣言されたものしか利用できない．このためプログラムの仕様とその実現法が分離でき，プログラムの保守性が向上する．

5.4.4 型

Cの構造型に対応して Ada ではレコード型が存在する．Ada ではレコード型の内容の一部を判別子と呼ばれるパラメタで指定できる．この例題では L-TYPE というレコード型の要素である CONTENTS という配列の大きさを SIZE という判別子で指定している．

Ada に特有な型として密閉型がある．これは型の構造や（既定義の）操作を隠蔽することにより型を抽象的に扱う場合に用いる．密閉型に対する既

定義の操作は代入と等価比較だけであり，それ以外の操作は新しく宣言しなければならない．このためプログラムが型の実際の構造に依存せず抽象的に記述できるので保守性が向上する．ここでは L-TYPE という密閉型を宣言している．この型は実際はCONTENTS, LEN という要素を持つレコード型であるが，パッケージの外部では具体的な構造を知ることができない．そこで L-TYPE 型の変数を用いる際も CONTENTS, LEN という要素を直接操作できない．

5.4.5 入 出 力

入出力関係の副プログラムは，Cと Ada でそれほど違わない．ただしCでは任意個のパラメタの値を含む文字列を printf という（1つの）手続きで出力することができるが，Ada では副プログラムのパラメタの個数や型はすべてあらかじめ決まっているのでこのような場合は複数の手続きに分けて出力しなければならない．

またCと Ada では入力の状態（入力の終わり，行の終わり）の判定法が異なる．Cでは getchar は入力の終わりに達すると EOF という値（具体的な値は処理系ごとに異なる）を返し，行の終わりに達すると '\n' という値を返すことになっている．そこで getchar で読み込んだ値が EOF や '\n' であるかをチェックすることにより入力状態を判定できる．これに対して Ada では入力の終わりを判定する END-OF-FILE, 行の終わりを判定する END-OF-LINE という関数が定義されていて，これらを用いて入力状態を判定する．

6. Ada 使用にあたって

6.1 Ada とCの違い

Ada とCは共にシステムソフトウェアや組み込みソフトウェアを主な記述対象とする言語であるが，両者の性格は異なっている．一言でいえば Ada は大規模なソフトウェアを多人数で組織的に開発するのに適した言語であり，Cは中規模の（高性能の）ソフトウェアを少人数で開発するのに適した言語である．

大規模なソフトウェアで最も重要なことは，そのソフトウェアの信頼性と保守性である．Ada ではプログラムの信頼性を向上させるための機能が多数備わっている．たとえば翻訳時や実行時に型の整合性や値の範囲に関するチェックを行うことによりプログラムの信頼性を向上させている．さらに大規模ソフトウェアの開発では分離翻訳を行うのが普通であるが，この場合にも全体を一括して翻訳した場合と同じ信頼性が得られるようになっている．

また Ada では保守性を向上させるために密閉型，限定型やパッケージのように仕様（インタフェース）と本体（実現部）を分離できる機能がある．これらを用いると仕様が変更しない限り本体を自由に変更できるので，プログラムの修正や改造が容易になる．さらにこれらによってプログラム間のインタフェースが明確になるのでプログラムの流用が容易になり生産性の向上につながる．

これに対して性能を重視する中規模以下のソフトウェアでは，プログラミング技術を駆使したコーディングを行うことが重要になる．Cではそのようなコーディングを行うための機能が多い．たとえばCには機械語の命令に対応した多数の演算子があり，出力コードを予想してコーディングを行うことができる．またCではポインタ型と配列型は同一視されポインタ型に対する加減算が定義されているので，配列の要素に対するアクセスが効率よく行える．

大規模なソフトウェアでも効率は重要であるが効率を上げるにはそれを考慮した設計を行うことや翻訳時に最適化を十分行うことが重要であり，コーディング時にそのための技術を駆使する必要は少ない．そこで Ada では効率よりも信頼性を重視した機能が多い．

たとえばCのポインタの機能（スタック領域の変数を指せる，加減算や大小比較ができる）はプログラムの効率を上げるには有効だが，エラーの原因にもなりやすい．そこで Ada ではポインタ型（アクセス型）の機能に大きな制限（ヒープ領域しか指せない，代入と等号比較しかできないなど）をくわえている．

6.2 設計上の留意点

6.1 で述べたようにCと Ada では言語の設計思想が異なっている．そこで Ada らしい（信頼性や保守性を重視した）プログラムを開発するのは設計段階からそのための配慮が必要になる．たとえば次の点に留意する必要がある．

6.2.1 処理系の行うチェックの利用

処理系が翻訳時や実行時に行うチェックを利用して信頼性の向上をはかる．たとえば配列の添字を表す変数の型を配列の添字範囲を制約とする部分型にすることにより，その値が常に添字範囲内であることを保証できる．

6.2.2 読解性の向上

多重定義や例外処理機能を用いてプログラムをわかりやすくすることにより保守性の向上をはかる．たと

えばプログラム中で定義した型に対してもシステム定義の型と同様の演算子を用いることによりプログラム中の式を読みやすくできる．また例外処理機能によって通常の処理とエラー処理をプログラム文面上で分離できる．

6.2.3 インタフェースの明確化

仕様と本体を分離して外部とのインタフェースを明確化することによって保守性や汎用性の向上をはかる．たとえばデータ型を密閉型，限定型として宣言することにより，その型の操作をそこで定義された副プログラムだけに制限できる．これによってそのデータ型の内部構造の変更あるいはそのデータ型の他のプログラムへの流用が容易になる．

6.3 機能上の留意点

Cではよく用いられる機能でも Ada では対応する機能がない場合がある．たとえば次の機能に関しては注意が必要である．

6.3.1 static 属性

Ada にはこの属性は存在しない．便利な属性ではあるが，プログラムを不明確にする原因の1つとなる．そこで対応する変数の宣言を，その宣言の有効範囲がプログラム上必要な範囲になるように設定するか，あるいはパッケージの中に設定してパッケージを必要な個所で参照するようにするかして，その変数の有効範囲を局所的にするのが望ましい．

6.3.2 単項演算子 &

Ada では類似の機能としてアクセス型が用意されているのでそれを用いるようにする．& に直接対応するものとして Address 属性があるがプログラムの読解性や保守性の点からこれはなるべく用いない方がよい．

6.3.3 整数型と論理型

Ada では整数型と論理型はまったく異なる型なので両者を混合した使用は避けるべきである．

6.3.4 関数パラメタ

Ada では汎用体を用いることにより，ある程度のパラメタ化は可能である．それ以上はプログラムの読解性の点から避けることが望ましい．

6.3.5 書式変換

Ada では可変個で動的に定まるパラメタの記述はできない．書式付き入出力パッケージを用意しても可変個のパラメタを記述できない点は要注意である．

6.3.6 非局所 GOTO (setjmp, longjmp)

例外処理の機能を用いて Ada でも記述できる．

7. おわりに

本稿では，C言語によるプログラミングに慣れたプログラマのために，Cと Ada によるプログラムを対比させて解説した．今後のソフトウェアは効率だけでなく保守性がより重視され，また開発や保守の支援システムを利用して行うことが多くなると予想される．こういった環境に適した機能を持つ Ada はますます重要になると思われる．C言語に慣れたプログラマが Ada プログラムを作成する場合には，Cの場合に類似したパッケージライブラリ[3]を用意することによりさらに使いやすい環境になると考えられる．

謝辞 本解説の原稿に対し多くのコメントをいただいた NTT 福山調査役をはじめとする Ada 分科会のメンバ各氏に感謝いたします．

参考文献

1) Kernighan, B. and Ritchie, D.: The C Programming Language, Prentice-Hall, (1978), 邦訳：石田晴久訳：プログラミング言語C，共立出版 (1981).
2) U. S. Department of Defense: Reference Manual for the Ada Programming Language, (ANSI/MIL-STD-1815A) (Jan. 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).
3) 木下，落合，中村，田中，浅井：Ada 利用に当っての標準パッケージ定義 (1)～(2)，情報処理学会第 31 回全国大会論文集，3E-3, 4, pp. 325-328 (1985).

(昭和 60 年 11 月 5 日受付)