資　料

Westran: Fortran をベースとした構造的言語とその処理系*

真野芳久** 杉藤芳雄** 鳥居宏次**

Abstract

A structured language called Westran (for **We**ll **S**tructured For**tran**) based on Fortran is introduced, and a preprocessor supporting the language, described. Three main features of the language are: (1) a large repertoire of additional control structures, (2) mechanisms for defining new data types similar to those of Pascal, and (3) unrestricted name lengths. Symbolic constants and in-line comments are also supported. A Westran-Fortran translator, described here, was written using bootstrap technique.

1. まえがき

Fortran に対しては，プログラミング方法論の観点からなされているものも含めて様々な批判がある．しかし，Fortran は現実的な面では多くの長所を備えている．たとえば，言語仕様の面で多少の差異はあるがほとんどの計算機システムに Fortran コンパイラが用意されていてかなりの移植性 (portability) が保証されていること，プログラムの蓄積が容易なこと，効率の良いオブジェクト・コードとなること，等であり，Fortran は最も広く使用されている言語の1つである．今後この状態が急激に変化するとは考えにくい．

この点から，Fortran の機能を整理・充実し，いわゆる構造的なプログラムを書き易くすることの重要性は広く認められているところである．

Fortran の改良・発展のための動きは，Fortran Development Newsletter[1] が発行されているように，近年盛んに議論されるようになった．その1つの動きは，Fortran の言語機能そのものの拡充である***．しかしこの方向は構造的なプログラムを作成するためではなく，従来の Fortran では不可能であった機能の追加が中心のようである．別の動きとしては，拡充された機能をプリプロセッサによって Fortran に変換する方法がある．その最大の利点として，既存コンパイラの利用と，その処理系作成の容易さがあげられる．

このプリプロセッサ方式によって構造的なプログラムを書き易くしようとする試みは既に数多く報告されている[2]．それらには，Fortran に if～then～else～と while～do～ を追加しただけのもの[3]，比較的豊富な制御要素を持っているもの[4]，制御要素の充実と共に記法上使いづらい点について考慮しているもの[5]，等が含まれる．これらはふつう Structured Fortran と呼ばれているものである．

これらの例からわかるように，従来のプリプロセッサ方式では，主として構造的な制御要素の導入によって構造的なプログラムの作成問題を解決しようとしてきた．しかし，プログラムにおける今1つの重要な要素であるデータを取り扱う方法についてはほとんど考慮されてこなかった．Fortran に用意されているデータをまとめる手段としては配列のみであり，データを構造的に扱うための手段についても考慮されなくては十分といえない．

以上の立場から，本稿では制御要素の充実の他に，関連あるデータを統一的に利用できるための機能について考察し，Fortran をベースとした新しい構造的言語 Westran (Well Structured Fortran) を提案する．更に，ブートストラップ手法によって Westran 自身を用いてインプリメントされている Westran プリプロセッサについても述べる．

2. Westran の設計方針と概略

2.1 制御要素の拡充

goto 文はその使われ方によっては危険な存在にな

---

* Westran: Fortran-based Well Structured Language and Its Implementation by Yoshihisa MANO, Yoshio SUGITO and Koji TORII (Computer Science Division, Electrotechnical Laboratory)

** 電子技術総合研究所ソフトウェア部

*** アメリカでは ANSI X3J3 委員会が公式に検討し，そこからの提案が SIGPLAN Notices, Vol. 11, No. 3 (1976) 発表にされた．

り得る．Westran では goto 文を使う必要が生じないよう，豊富で強力な制御要素を用意する．文献5）では，他の制御要素でさ程無理なく書ける制御要素は含めないで簡明性 (simplicity) を優先している．しかし，汎用性と有効性が確かめられている制御要素については，概念を可能な限り直接に表現できるようにするため含められるべきだと考えられる．Westran では，選択に関してはいわゆる if～then～ else～ 及び case ～の構文を，繰り返しに関してはいわゆる while～，repeat～ until，for～，doforever～の構文を用意し，更に内部手続きの定義と呼び出しのための構文を用意している．

なお，Westran での表記方法は原則として，Fortran 利用者が無理なく Westran を利用できるよう，Fortran に近い記法をとっている．例えば，行の終わりが通常，文の終わりを示し，継続行という考え方は Fortran と同じである．

また，新たに導入された制御要素の有効範囲を明瞭に示すための記法として，有効範囲の最後に対応するキーワードを置く．このキーワードの形は原則として“END 最初のキーワード”の形とする．

2.2 データの構造化と抽象化

データの取り扱い方法について最近急速に関心が持たれ研究が進められているが，互いに関連するデータを1箇所にまとめて統一的に取り扱えるようにすることは，互いに関連するコードを機能的にグループ化することと同様に，重要である．Westran では，データ型が異なっていてもよい要素を直積としてまとめあげ，それを新しいデータ型 (Cartesian product[6] あるいは record[7] として知られている）として定義できる機能を用意する．直積の成分が配列の形であることを許し，更に，このように定義されたデータ型の配列をも許すことにより，互いに関連するデータをまとめあげる機能は充実したものとなる．

また，構造を持たないいくつかの要素を扱う場合，それらの要素を明瞭に表わす名前がその定数値となる新しいデータ型（数えあげ (enumeration) 形式と呼ばれる）を定義することにより，それら要素をプログラム上の表現との対応が容易にできるようになる．Westran はこの機能を持つ．

まとめられたデータ（またはデータ型）を抽象化して取り扱うプログラミング方法の利点が指摘されている[8),9)]．その具体的手法としては，まとめられたデータの操作を，データ（型）の表現等と共に定義されているオペレーションを通してのみ行うものであり，データの実際の表現形式を知る必要をなくしている．Westran では，定義されるデータ型に対してこの手法を適用したプログラミングを行えるようにする．

2.3 名前とその意味との対応

概念とプログラム上での表現との対応が明瞭であることは，理解し易いプログラムであるための条件である．明瞭な対応を実現するための最も基本的な方法として，変数等の名前にその担っている役割・意味を十分反映させることがあげられる．しかし，Fortran の名前は JIS Fortran 7000 においても6文字以内と制限されており，上記要件を満たすように名前を付けることは困難である．Westran ではその長さの制限を事実上除き，役割・意味が十分反映される名前を付けられるようにする．

2.4 そ の 他

Weissman[10] は，プログラムの理解と保守を容易にあるいは困難にする要因をいくつかあげている．プログラムの形に関しては，コメントの有無とその書き方，インデンテーション，変数名の付け方，その値からは意味を推測しにくい数の使用，等をあげている．

Westran ではこれらの点を考慮して，

ⅰ）行の中で(ⅳ)で述べる文字定数の中以外での)！文字以降はコメントとみなされる．これにより妥当な長さのコメントを書き易くする．

ⅱ）定数を表わす名前の定義を可能とし，効率を下げることなく読み易くする．

ⅲ）**2.1** で述べた制御要素の形式は統一的なインデンテーションを行い易くしていると考えられる．また，プログラムの変更によるインデンテーションの狂いを修正するために，インデンテーションを行うプログラムを用意する．

この他の改良点として，

ⅳ）文字定数は，*nH* 形式 (Hollerith constant) の他に，引用符で囲んだものを許す．これにより見やすくし *n* を数える手間を省く．

Westran はそのプリプロセッサによって1回のパスで Fortran に変換され得るようになっている．即ち，1つの Westran 文が読み込まれて原則的に左から右にスキャンされ，いくつかの Fortran 文が出力される．1度出力された Fortran 文を後で書き変える必要はない．

3. Westran 言語仕様

Table 1 に Westran に用意されている制御要素について，**Table 2** に定数を表わす名前と新しいデータ型の定義について，それぞれの記述形式とその意味を示す．ここで，文は，空または1つ以上の Westran の文を表わす．Westran では Fortran と同様に，次が継続行でない行の終わりを文の終わりとする．ただし実行文については，セミコロンで区切ることによって同一行に複数個の文を書くことを認める．また，Westran は，文番号，継続行であることを示す文字，文のそれぞれの行中での位置を特に定めていない．文番号あるいは継続行であることを示す文字（"&"）が文（または文の一部）よりも前にあればよい．

名前，整数変数，整数式，論理式は，名前の長さが100以下である点を除いて，それぞれ JIS Fortran 7000 の名前，整数型の変数，整数型の算術式，論理式の規則に従う．なお，{A}，$\left\{\begin{matrix}B\\ \vdots\\ C\end{matrix}\right\}$ はそれぞれ，

Table 1 Control primitives in Westran

(a) 選 択

```
IF      (論理式1)
  THEN 文1
ELSEIF (論理式2)
  THEN 文2
    ⋮
ELSEIF (論理式n)
  THEN 文n
 {ELSE 文n+1}      (n≧1)
ENDIF
(通常の意味での if～then～else if～ を実現している．)
```

```
CASE    整数式
  WHEN 範囲11, …, 範囲1k1  =>  文1
    ⋮
  WHEN 範囲n1, …, 範囲 nkn =>  文n
 {OTHERWISE                =>  文n+1}
ENDCASE          (n≧1, ki≧1)
  ここで範囲ij は次のいずれかの形である
                整数式ij
               <整数式ij
                整数式ij<
          整数式ij1<整数式ij2
整数式が評価されその値を含む範囲ij があれば文i が実行される．そのような範囲ij がなく OTHERWISE 節があれば文n+1 が実行される．ここで範囲ij の形の意味はそれぞれ
        整数式ij
        整数式ij 以下
        整数式ij 以上
        整数式ij1 以上整数式ij2 以下
である
```

(b) 繰り返し

```
WHILE (論理式)
  文
ENDWHILE        (理論式が .TRUE. の間文が繰り返し(0回以上) 実行される．)
```

```
REPEAT
  文
UNTIL (論理式)   (論理式が .TRUE. になるまで文が繰り返し(1回以上) 実行される．)
```

```
DOFOREVER
  文
EXIT             (EXIT 文が実行されるまで，DOFOREVER と ENDDOFOREVER で囲まれた文が繰り返し実行される．)
  文
ENDDOFOREVER
```

```
IFOR 整変数=整数式1, 整数式2 {, 整数式3}
  文
ENDIFOR          ((Incremental FOR) Fortran の DO 文とほぼ同様である．文の繰り返しは0回以上．)
```

```
IFOR 整変数=整数式1, 整数式2, {整数式3}, {WHILE|UNTIL|SUCHTHAT} (論理式)
  文
ENDIFOR          (付加される条件はそれぞれ，論理式が，.TRUE. の間だけループを実行，.TRUE. になるまでループを実行，.FALSE. のとき1回ループをスキップを表わす，)
```

```
DFOR 整変数=整数式1, 整数式2, {整数式3}
  文
ENDDFOR
```

```
DFOR 整変数=整数式1, 整数式2, {整数式3}, {WHILE|UNTIL|SUCHTHAT} (論理式)
  文
ENDDFOR          ((Decremental FOR) 整変数が整数式3の値(省略されているときは1)だけ減じられていく点を除いて，IFOR に同じ．)
```

(c) 内部手続き

```
PENTRY 名前
  文
  PEXIT
  文
ENDP
(名前で示される内部手続きを定義する．内部手続きは同一プログラム単位内から呼び出され，PEXIT 文または ENDP 文の実行により制御が呼び出しの次に移る．)
```

```
PERFORM 名前
(名前で示される内部手続きをサブルーチン的に呼び出す．)
```

Table 2 Definition of symbolic constants and new data tydes

(a) 定数名の定義

```
GCONSTANT  名前1=整数式1, …, 名前n=整数式n (n≧1)
(Westran 処理系による1回の変換全体で有効な定数を表わす名前を定義する，ここで整数式i は括弧を含まず左から右に計算される．そのオペランドは整定数または GCONSTANT 文で既に定義されている名前である．)
```

```
CONSTANT  名前1=整数式1, …, 名前n=整数式n (n≧1)
(この文を含むプログラム単位内で有効な定数を表わす名前を定義する．整数式のオペランドは整定数または GCONSTANT 文，CONSTANT 文で既に定義されている名前である．)
```

(b) 新しいデータ型の定義

```
ETYPE  名前0=(名前1, …, 名前n)   (n≧1)
(名前1, …, 名前n がその定数値となる数え上げ形式のデータ型名前0 を新たに定義する．)
```

```
CTYPE  名前0=(型宣言1; …; 型宣言n)   (n≧1)
  ここで型宣言i は Fortran の型宣言文と同じ形式である．
(型宣言1, …, 型宣言n で宣言されている名前をその成分名とする直積形式のデータ型名前0 を新たに定義する．型宣言i に現われるデータ型は Fortran 既存のデータ型または ETYPE 文で定義されたデータ型である．)
```

Aが省略されても，B,…，C のいずれが選ばれても構文的に正しいものであることを示す．

新しく定義されるデータ型は Westran プリプロセッサによる1回の変換全体で有効である．これら新しいデータ型の変数・配列を宣言する方法は，Fortran の型宣言文と同じ形式であり，新しいデータ型の名前の後に変数名または配列宣言子を置く．ただし，新たに定義されたデータ型の関数を宣言することはできない．

直積データ型の変数・配列で，その成分を表わすには“そのデータ型の1つの要素．そのデータ型の成分名”（成分が配列であれば更に添字が続く）の記法を用いる．例えば VAR . COMP, ARRAY(I) . COMP, ARRAY(I) . ARRAYCOMP(J) 等である．

新たに定義されたデータ型を抽象化して取り扱い，その型のデータへの操作は専用のルーチンを通してのみ行うようにするためには，それらデータをパラメータとして他の副プログラムに渡したり，それらデータを共通領域中に置くことが必要となる．これは，型宣言文と同じ形式のデータの宣言を仮引数リスト中あるいは COMMON 文中に（前あるいは後に区切り記号が必要ならばセミコロンを用いて）置くことによりなされる．

いくつかの Fortran 言語機能が Westran に含まれていない．これらは，論理 IF 文，算術 IF 文，DO 文，である．しかし，これらと同等あるいは同等以上の機能が Westran に用意されている．一般にはもとの言語にある機能はすべて含まれることが望ましい．しかし，それにより言語の全体的な仕様が不自然あるいは不統一になる場合には，それらもとの言語の機能は，新しい言語の機能としてではなく，プリプロセッサへの指示（変換処理の必要・不要）という形で実現させるべきであると考えられる．インプリメントされている Westran のプリプロセッサでは，変換処理の一時中断や再開を注釈行中に指定できるようになっている．

4. Westran プリプロセッサ

Westran プリプロセッサは Westran 自身で書かれている．それは次のようにブートストラップ手法を用いてインプリメントされた．

まず Westran の持っている制御要素の簡略化されたサブセットのみを扱うプリプロセッサが Fortran で書かれた．そこでは，新たに導入されたキーワードの前後には空白を置く，キーワードで始まる文は文番号を持たず2行以上にならない，等の制限が付けられ，これらの制限によりプリプロセッサは極めて容易に作成された．Westran プリプロセッサは，このサブセットを用いて，途中からは自分自身を利用して，一切 GOTO 文を用いずにインプリメントされている．

Westran プリプロセッサは最初 TOSBAC 5600 の TSS（時分割システム）下で利用できるように開発された．このプリプロセッサは他の1機種にも移植された*が，以下は TOSBAC 5600 上のプリプロセッサについての記述である．変換の対象となる Westran プログラム及び変換されて得られる Fortran プログラムは，TSS が管理しているファイル（それぞれファイル1，ファイル2とする）中に置かれる．利用者は Westran プログラムが格納されているファイル1を入力として，Westran プリプロセッサを起動させる．プリプロセッサは Westran プログラムを処理して一定書式の Fortran プログラムに変換し，ファイル2に出力する．変換終了後，ファイル2を入力として Fortran コンパイラを起動させることになる．この間のコマンドは簡単であり，実行時間もわずかである．この変換はプログラム全体に対して一括してなされることを原則としているが，次の点が満たされていればプログラム単位でなされてもよい．即ち，外部手続き名，共通領域名が6文字以内，及び新たに定義されるデータ型が使われていれば，その定義は変換の単位ごとに付けられていることである．

この種のプリプロセッサの場合，変換される前のプログラム上でのデバッグが容易か否かは，その有用性を左右する重要な問題である．TOSBAC 5600 の Fortran コンパイラは，プログラムの各行に行番号を持つことを許し，コンパイル時のエラーはこの行番号を添えて示す．Westran プログラムもまた行番号を持つこととし，同じ行番号を持つように変換して，Fortran コンパイラの上記性質を利用している．また．変換時に見つけられるエラーもこの行番号を添えて示す．こうして Westran プログラムの構文に関するエラーは容易に Westran プログラム上で修正され得る．

プリプロセッサでは，新たに導入したキーワードや Fortran の既存キーワードについて次の制限を設定して，入力文の完全な解析を避けている．

- キーワードの前後は英数字以外，

* HITAC 8350 の TSS で動く．

- キーワードの途中に空白を含まない（ただし，BLOCK DATA と DOUBLE PRECISION では途中に1つ空白を含む形をキーワードとする），
- 前2条件を満たすキーワードと同じ文字列があれば，（そのキーワードが明らかに現われ得ない状況である場合を除き）キーワードとみなす．

新たに導入されるデータ型の名前についても，同様の制限を付けている．

この他の制限としては，

- 内部手続きの名前には，戻り先(を示す整数値)を格納するので，整数型でなければならない．しかし逆に，その名前に戻り先が与えられる点を利用して，利用者の責任でスタックに戻り先や局所変数の保存・回復を行うことにすれば，比較的容易に再帰呼び出しを実現できる．
- ある種の名前，文番号はプリプロセッサにより生成されるので利用者が使うことはできない．

なお，Westran プリプロセッサの大きさは約 2200 行であり，そのうち注釈行と実行文でない行がそれぞれ約2割を占める．これを自分自身で Fortran に変換したとき約 2500 行（注釈行は除かれる）になる．

```
010 GCONSTANT MAXNODE=40,MAXN1=MAXNODE-1
020 &        ,STACKSIZE=MAXNODE
030 CTYPE NODETYPE =(INTEGER ORD,DEGO,ADJ(MAXN1))
040 CTYPE STACKTYPE=(INTEGER TOP,BODY(STACKSIZE))
050 NODETYPE NODES(MAXNODE)
060  :
070  :
080 CALL TOPSORT(NUMBEROFNODES,NODES)
090  :
100  :
110 STOP
120 END
130C
140 SUBROUTINE TOPSORT(NUMBEROFNODES;NODETYPE NODES(*))
150 STACKTYPE STACK
160 INTEGER POP
170 LOGICAL EMPTY
180 CALL STACKINIT(STACK)
190 IFOR I=1,NUMBEROFNODES,,CHECKVAL(NODES(I).INDG.EQ.0)
200   CALL PUSH(STACK,I)
210 ENDIFOR
220 ORDER=1
230 WHILE(.NOT.EMPTY(STACK))
240   I=POP(STACK)
250   NODES(I).ORD=ORDER
260   ORDER=ORDER+1
270   IFOR J=1,NODES(I).DEGO
280     K=NODES(I).ADJ(J)
290     NODES(K).DEGO=NODES(K).DEGO-1
300     IF(NODES(K).DEGO.EQ.0)
310       THEN CALL PUSH(STACK,K)
320       ENDIF
330   ENDIFOR
340 ENDWHILE
350 RETURN
360 END
370C
380 SUBROUTINE STACKINIT(STACKTYPE STACK)
390 STACK.TOP=0
400 RETURN
410 END
420C
430 SUBROUTINE PUSH(STACKTYPE STACK;I)
440 STACK.TOP=STACK.TOP+1
450 IF(STACK.TOP.LE.STACKSIZE)
460   THEN STACK.BODY(STACK.TOP)=I
470   ELSE CALL ERROR
480 ENDIF
490 RETURN
500 END
510C
520 INTEGER FUNCTION POP(STACKTYPE STACK)
530  :
540  :
550 END
```

Fig. 1 An example of Westran program

5. Westran プログラム例

Fig. 1 に Westran プログラム例を示す．そこに示されているサブルーチン TOPSORT は，サイクルを含まない有向グラフの節点に，すべての有向枝の終点は始点より常に大きな番号が与えられるように，1から順に番号を割り付ける，いわゆる topological sort を行うものである．

この例ではグラフの節点を，付けられるべき番号(ORD)・入次数 (DEGI)・出次数 (DEGO)・その節点を始点とする有向枝のすべての終点を含んでいる配列(ADJ)，という要素を持つ4ツ組として表現しており，そのための直積型のデータ型 NODETYPE を定義している．

また，STACKTYPE はスタックを表現している直積型のデータ型であり，STACK はそのデータ型の変数である．その成分へのアクセスは任意の時点で可能ではあるが，スタックを操作する種々のオペレーションを用意し，他の副プログラムへパラメータとして渡せる機能を利用して，STACKTYPE なるデータ型を抽象的なデータ型であるかのように使用している．

Fig. 2 は，変換されて得られる Fortran プログラ

```
140       SUBROUTINE T9996(N9994,N9993,N9992,N9991)
140       INTEGER N9993(40,39)
140       INTEGER N9992(40)
140       INTEGER N9991(40)
150       INTEGER S9990(40)
150       INTEGER S9989
160       INTEGER POP
170       LOGICAL EMPTY
180       CALL S9988(S9990,S9989)
190       I=1
190       I0001=N9994
190       GOTO 9985
190 9986  I=I+1
190 9985  IF(I.GT.I0001)GOTO 9987
190       IF(.NOT.(N9992(I).EQ.0)
190      & )GOTO 9986
200       CALL PUSH(S9990,S9989,I)
210       GOTO 9986
210 9987  CONTINUE
220       ORDER=1
230 9983  IF(.NOT.(.NOT.EMPTY(S9990,S9989))
230      & )GOTO 9984
240       I=POP(S9990,S9989)
250       N9991(I)=ORDER
260       ORDER=ORDER+1
270       J=1
270       I0002=N9992(I)
270       GOTO 9980
270 9981  J=J+1
270 9980  IF(J.GT.I0002)GOTO 9982
280       K=N9993(I,J)
290       N9992(K)=N9992(K)-1
300       IF(.NOT.(N9992(K).EQ.0)
300      & )GOTO 9979
310       CALL PUSH(S9990,S9989,K)
320 9979  CONTINUE
330       GOTO 9981
330 9982  CONTINUE
340       GOTO 9983
340 9984  CONTINUE
350       RETURN
360       END
```

Fig. 2 The Fortran program derived from the Westran program of Fig. 1

ムのうち，サブルーチン TOPSORT に関する部分である．

6. むすび

プログラム作成時の人間の考え方，制御だけでなくデータに関しても不必要に歪められることなくプログラム・コード上に表現できることを目的とする Fortran をベースとした構造的言語 Westran とそのプリプロセッサについて述べた．Westran プリプロセッサは Westran 自身で書かれており，この経験から，Fortran よりはるかに理解し易いプログラムが容易に得られることが実証された．

しかし，Westran 及びそのプリプロセッサについての問題点や残された課題をいくつか挙げることができる．

まず，新しいデータ型を導入する利点の1つに，タイプチェックを計算機にさせることがあげられる．しかし Westran プリプロセッサはこの種のサービスを行っていない．

第2に，プリプロセッサの作成を容易にするために，不統一な構文になっている部分がある．即ち，COMMON 文中の変数・配列リストや，仮引数リストの中に現われる新たに導入されたデータ型の変数・配列の宣言に関してである．

第3に，Fortran では1つの論理 IF 文で書けることが，Westran では少なくとも3つの文になる，というようにコードが長くなってしまう場合がある．

最後に，入出力形式の指定方法は Fortran で最も理解し難い部分であると考えられるが，Westran では特に取り扱っていない．プリプロセッサで処理できる範囲外だと考えられるからである．

現在 Westran は，特に TOSBAC 5600 上の版では，TSS 及びファイル・システムの下で動いていることによっていくつかの利点・使い易さを得ているが，他の環境の下でも適当な入出力装置・入出力方法を考慮することにより，この種のプリプロセッサは容易にインプリメントできる極めて有益なプログラミングの道具になり得ると考えられる．それを実現する最も簡単な方法は，既に1異機種に対してなされたように，Westran プリプロセッサを移植することである．入出力関係で多少の作業を必要とするが，理解し易い Westran 版と自分自身を変換して得られる Fortran 版を利用することによって容易に行われ得るであろう．

末筆ながら，本研究の機会を与えて頂いた当研究所石井治ソフトウェア部長に感謝する．また，有益な助言をして頂いた方々，プリプロセッサ等の作成を手伝って頂いた方々に感謝する．

参考文献

1) FOR-WORD Fortran Development Newsletter, Ad Hoc Comittee on Fortran Development, ACM-SIGPLAN, No. 1 (Feb. 1975)～No. 5 (Oct. 1975), Vol. 2, No. 1 (Feb. 1976)～
2) D. J. Reifer & L. P. Meissner: Structured Fortran Preprocessor Survey, Lawrence Berkeley Laboratory, University of California, UCID-3793 (Nov. 1975)
3) E. F. Miller, Jr.: Extensions to Fortran to Support Structured Programming, SIGPLAN Notices, Vol. 8, No. 6 (1973)
4) D. S. Higgins: A Structured Fortran Translator, SIGPLAN Notices, Vol. 10, No. 2 (1975)
5) B. W. Kernighan: RATFOR-A Preprocessor for a Rational Fortran, Software-Practice & Experience, Vol. 5, No. 4 (1975)
6) C. A. R. Hoare: Notes on Data Structuring, in "Structured Programming", Academic Press (1972)
7) N. Wirth: The Programming Language Pascal, Acta Informatica, Vol. 1, No. 1 (1971)
8) B. H. Liskov & S. N. Zilles: Programming with Abstract Data Types, SIGPLAN Notices, Vol. 9, No. 4 (1974)
9) O. Shigo, T. Shimomura, K. Iwamoto & T. Maejima: Implementing the Abstraction Technique in Software Development, 2nd USA-Japan Comp. Conf. (1975)
10) L. Weissman: Psychological Complexity of Computer Programs: An Experimental Methodology, SIGPLAN Notices, Vol. 9, No. 6 (1974)

(昭和51年3月19日受付)
(昭和51年8月21日再受付)