≪論　文≫

構造化プログラミング用語 LSP (PL/I)*

小林　正**　青木恭太***　落水浩一郎****
豊田順一***　田中幸吉***

Abstract

This paper describes a programming language and it's processer which enables programming along the concept of "Structured Programming" proposed by E. W. Dijkstra.

First, the notion of E. W. dijkstra is briefly discussed.

Second, LSP(L) is briefly discussed.

Third, PL/I is chosen as a example of L and LSP(PL/I) is implemented over FACOM 230-45s (osII-ED11) written by PL/I.

Finally, a programming example of "magic square" is given and the usefulness of LSP (PL/I) is shown.

1. まえがき

E. W. Dijkstra らの提唱した構造化プログラミングの手法は[1)2)3)]，プログラムの作成時及び作成後のプログラムの論理構造の把握を容易にし，論理的な正しさの確認しやすいプログラムを作成するための方法である．彼はその方法として3つの基本的な分解規則—連結，選択，繰り返し—だけを用いて，トップ・ダウン・メソッドでプログラムを作成する方法を示した．

本論文では，この Dijkstra の方法を基盤として，既存のプログラム言語を対象として，その言語で書かれた構造化プログラムを組織的に作成するための LSP (PL/I)-Language for Structured Programming- について述べる．

プログラム構成法は，プログラムは可能な限り単純な命令系列とする．したがって複数の出口，入口を持つアクションの分解規則は採用せず，Dijkstra の1入口1出口のアクションの分解規則を採用する．この方法ではプログラミングは，アクションをサブ・アクションに分解し，最終的にはすべてのサブ・アクションが想定しているプログラム言語のステートメントとなった時点で，アクションをサブ・アクションで置き換える事によりプログラムが完成し，終了する．

実際にこの方法でプログラムを作成する事は，あるアクションをサブ・アクションに分解し，置き換える作業を繰り返す事に他ならない．特に置き換える作業は，非常に面倒な作業で手間がかかるが，機能的な面を見れば，きわめて機械的な作業である．そこで，アクションをサブ・アクションに分解する作業を書き換え規則の形で，すべて記録しておき，その書き換え規則の系列を，一定の形式で計算機への入力とする事を考える．このために，書き換え規則の書式を定める事が必要となる．そこでこの書式を定める事を言語の定義へと発展させ，その言語で記述した分解過程を入力として，目指す言語の構造化プログラムを出力するトランスレータを構成する．このトランスレータはアクションの書き換え規則から目的とするプログラムを作り出すので，プログラムの作成過程で成されたすべての意志決定の記録を一度は見ることになる．そこでこの機会を逃さず，すべての意志決定をコメントの形で表現する様にトランスレータを作成する．また，トランスレータは置き換える作業を実行するとともに，目的とするプログラムの形式を見易い，整った形式として出力する．プログラムを整った形式にする作業は人間にとっては忍耐強い作業が必要であったが，トランスレータはこの作業からも人間を解放する．

* Language for Structured Programming, by Tadashi KOBAYASHI, Kyota AOKI, Koichiro OCHIMIZU, Junichi TOYODA, and Kohkichi TANAKA

** 富士通（株）
*** 大阪大学基礎工学部情報工学科
**** 静岡大学工学部情報工学科

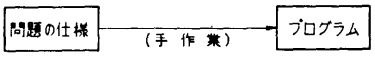

Fig. 1 An illustration of a process of a traditional programming method.

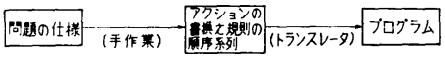

Fig. 2 An illustration of a process of a programming method which is discussed in this paper.

このトランスレータを使用したプログラミング・システムを，従来のシステムと比較すれば，従来のシステムでは，すべて手作業で一挙にプログラムを作成していたのに較べ，本システムでは，アクションのサブ・アクションへの分解を除いた大部分の機械的ではあるが大変な作業をトランスレータにまかせ，プログラマは，アクションをサブ・アクションに分解する，真に意志決定の必要とされる面のみに力を注げるという点が異なっている．

2. 構造化プログラミング用言語

本章では，構造化プログラミング用言語 LSP (L) の定義を示す．

2.1 構造化プログラミングを本論文では，次のようにとらえる．

2.1.1 アクションをサブ・アクションに分解する分解規則を定める事から出発し．

2.1.2 アクションのサブ・アクションへの分解は，分解規則の厳密な適用だけで行なう．

2.1.1によって，プログラムの構造を規定し，2.1.2によって，それ以外の構造を持たない事を保証する．本論文では，アクションのサブ・アクションへの分解規則として，E. W. Dijkstra のあげる 3 つの規則を用いる．3つの分解規則とは次の(ⅰ)～(ⅲ)である．

(ⅰ) 連結 (concatenation)

(ⅱ) 選択 (selection)

(ⅲ) 繰り返し (repetition)

2.2 LSP(L) の定義

〔**定義1**〕 構造化プログラミングのための言語をLSP(L) で表わす．ここで言語 L で構造化プログラムが作成される．

〔**定義2**〕 LSP 構造化プログラムとは，LSP(L) 言語によるプログラムをLSP(L) のためのトランスレータに入力する事により得られる言語 L によるプログラムである．

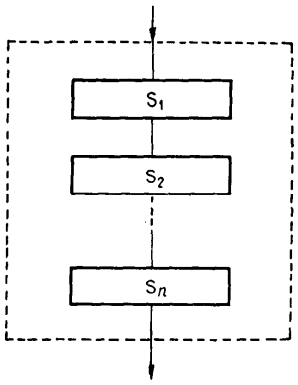

Fig. 3 An illustration of a refinement rule (i)

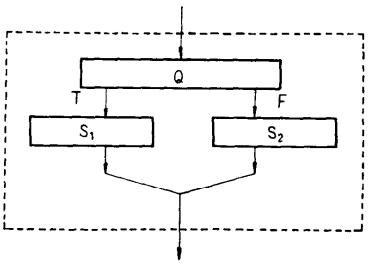

Fig. 4 An illustration of a refinement rule (ii)

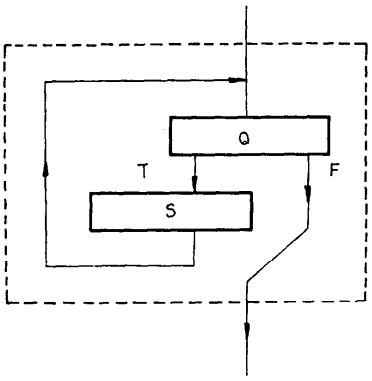

Fig. 5 An illustration of a refinement rule (iii)

先に示した (ⅰ)～(ⅲ) の分解規則を使って，1つのアクションを分解不可能な所まで分解する事と，生成文法において，書き換え規則を使って端記号に落ちるまで書き換えていく事とは，多くの類似性を持つ．そこで (ⅰ)～(ⅲ) の分解規則を書き換え規則の形で表現した生成文法を定義する．

〔**定義3**〕 LSP(L)言語を生成する生成文法を GSP (L) (Grammer for Structured Programming) とす

る．GSP(L) は次の様に定義される．

GSP(L)＝(V_N, V_T, P, S)

V_N：非終端記号の有限集合

V_T：端記号の集合

P：生成規則の有限集合

S：出発記号

$V_T \cap V_N = \phi$

実際には，

V_N＝{〈アクション〉，〈アクション名〉，〈アクション1〉，〈アクション2〉，〈アクション3〉}

V_T＝{SEL, WHILE, ·, ;, :=, @, 〔, 〕, {〈言語 L のステートメント〉}, {〈言語 L で注釈として使用可能な文字列〉}}

S＝{〈PROGRAM〉} が V_N, V_T, S の定義である，書き換え規則Pは次の様になる．

(1) 〈PROGRAM〉→〈アクション〉

(2) 〈アクション〉→〈アクション〉〈アクション〉

(3) 〈アクション〉→〔〈アクション1〉〕〈アクション〉

(4) 〈アクション〉→SEL〔〈アクション3〉，〈アクション2〉，〈アクション2〉，〈アクション〉

(5) 〈アクション〉→WHILE〔〈アクション3〉，〈アクション2〉〕〈アクション〉

(6) 〈アクション〉→ε

(7) 〈アクション1〉→〈アクション1〉；〈アクション1〉

(8) 〈アクション1〉→〈アクション2〉

(9) 〈アクション2〉→〈アクション名〉

(10) 〈アクション2〉→〈言語Lのステーナメント〉

(11) 〈アクション3〉→〈言語Lの論理式〉

(12) 〈アクション名〉→〈言語Lで注釈として使用可能な文字列〉

書き換え規則(3)が(ⅰ)の連結に，(4)が(ⅱ)の選択に，(5)が(ⅲ)の繰り返しに相当する．ところが，この形式では，アクションがサブ・アクションに分解された記録を残す事ができない．そこで(3)～(5)の右辺の左端に，"〈アクション名〉:=1 を付加する．従って，LSP(L) の構文は，定義3と書き換え規則の再定義により次の様になる．

〔**定義4**〕 LSP(L) の構文を次の様に定義する．

〈PROGRAM〉→〈アクション〉

〈アクション〉→〈アクション〉〈アクション〉

〈アクション〉→〈アクション名〉：＝〔〈アクション1〉〕〈アクション〉

〈アクション〉→〈アクション名〉：＝SEL〔〈アクション3〉，〈アクション2〉，〈アクション2〉〕〈アクション〉

〈アクション〉→〈アクション名〉：＝WHILE〔〈アクション3〉，〈アクション2〉〕〈アクション〉

〈アクション〉→ε

〈アクション1〉→〈アクション1〉：〈アクション1〉

〈アクション1〉→〈アクション2〉

〈アクション2〉→〈アクション名〉

〈アクション2〉→〈言語Lのステートメント〉

〈アクション3〉→〈言語Lの論理式〉

〈アクション名〉→〈言語Lで注釈として使用可能な文字列〉

3. LSP(PL/I)

本章では，L＝PL/I と置いて，LSP(PL/I) なる言語を考える．

3.1 PL/I を L として選択した理由

PL/I が使用可能で，分解規則に対応する命令があり，書式に自由度が多く，構造を見易すく記述することが可能なことである．

3.2 命令の決定

LSP(PL/I) には，最低限度，3つの命令が必要である．(1)連結に対応する命令，(2)選択に対応する命令，(3)繰り返しに対応する命令．

(1)については，サブ・アクションを連結する機能が存在すればよい．PL/I では，サブ・アクションをまとめる手段として，次の3つが考えられる．

(A) S_1; S_2; …; S_n;

(B) DO; S_1; S_2; …; S_n; END;

(C) BEGIN; S_1; S_2; …; S_n; END;

(B)の DO グループは(A)で代用可能なので，作り出す命令を作る事はしない．(C)は必要である．BEGIN ブロック内では，宣言が可能であるから，これを(A)で代用することは不可能である．故に(1)より2つの命令を定義する．

〔一般形1〕 〈S_1; S_2; …; S_n〉

ここで S_1, S_2, S_3, …, S_n はサブ・アクションである．これは BEGIN ブロックでない命令の一次元の系列を作成する．

〔一般形2〕 BLK 〈S_1; S_2; …; S_n〉

これは BEGIN ブロックを作成する．

〔2〕については IF を選択するのが妥当である．

〔一般形3〕 SEL 〈C/S1/S2〉

ここで C は条件であり，C が真であるとき，S1 が C が偽であるとき，S が実行される．

(3)については WHILE～ という形式で十分であるが，DO 形の繰り返しもよく用いられる．そこで次の2つの形式を定義する．

〔一般形4〕 WHILE 〈C/S1〉

これは，DO WHILE ～ の形の文を作成する．

〔一般形5〕 REP 〈VAR/A/B/S1〉

これは，DO VAR=A TO B ～ という形の文を作成する．ここで VAR は変数名である．

3.3 プログラム要素の決定

LSP(PL/I) の仕様のうち，命令は3.2で決定した．本節では，プログラム要素の形を決定する．

LSP(PL/I) は書き換え規則を表現可能でなければならない．そこで，アクションとそのアクションを分解する5つの命令とを“：＝”で結びつける事によりアクションの分解過程を表現する．

〈例〉 A：＝〈A1；A2；A3〉— a
D：＝WHILE 〈P1=1/J〉— b

a はアクション A を3つのサブ・アクション，A1，A2，A3 の系列に分解する事を表わしており，b はアクション D を，P1=1 が成立する間，サブ・アクション J を実行するという下位構造に展開する事を，表わしている．またサブ・アクションがさらに分解可能なアクション名かどうかを明らかにするために，サブ・アクションの直前に“@”を付す．この様にして示されたサブ・アクションは，さらに分解される必要がある．

PL/I に用いられているラベルの処置については別に考察する必要がある．なぜならラベルは，それ自体，プログラムの構造に関する情報を含んでいる．そこで，ラベルはサブ・アクションの中に書き込まずに表現する方法を検討する．PL/I のラベルの種類を考えると，次の3つが存在する．

(A) 外部手続きのラベル
(B) 内部手続きのラベル
(C) 単なる名札としてのラベル

特に外部手続きの直前には，コンパイラ制御文が必要である．そこで LSP(PL/I) で外部手続きか，内部手続きかの情報を表現するためと，プログラムの見易さも考えて，ラベルはプログラム要素の頭部に次の様な形で付ける．

(A) (((L0))) (3重カッコ)
(B) ((L1)) (2重カッコ)
(C) (L2) (1重カッコ)

ここで，L0，L1，L2 はそれぞれのラベルである．(A)，(B)，(C)の書式が3種のラベル(A)，(B)，(C)に対応している．(A)の形においてはラベルは付けるとともに，コンパイラ制御文の必要な外部手続のラベルであることを表現する．(B)と(C)の形は単なるラベルを付けることが必要であることを表現する．

3.4 プログラム要素の書式

3.4.1 ラベル

ラベルは PL/I のラベルの書式に一致させる必要がある．

3.4.2 アクション名

アクション名は，PL/I のコメントとして使用可能な文字列で，サブ・アクションとして用いる時は，“@”を付加する．

3.4.3 アクション名以外のサブ・アクション

これは PL/I の文を表現していなければならない．

3.4.4 空 白

I) 一般に空白は意味を持つ．

II) 命令語の後に続く空白は PL/I のステートメントか“@”が現われるまで無視される．

III) 左辺側のアクション名は，ラベルがあれば，ラベルに関係ある文字とそれに続く空白を無視する．

IV) “；”，“/”の後に続く空白は PL/I のステートメントか“@”が現われるまで無視される．

V) “>”の右側は無視される．

3.4.5 サブ・アクションの区切り

サブ・アクションの終わりは“；”または“/”または“>”で識別される．ここで / は改行記号を兼ねる．

3.4.6 手掛かり語

“；”，“/”，“<”，“>”は手掛かり語である．手掛かり語を手掛かり語以外の目的で書くには，2度同じ記号を繰り返して書けばよい．

3.5 LSP(PL/I) の構文

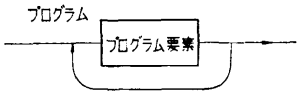

Fig. 6 An illustration of a syntax of a program

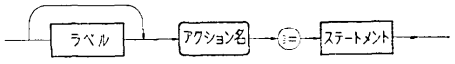

Fig. 7 An illustration of a syntax of a program unit

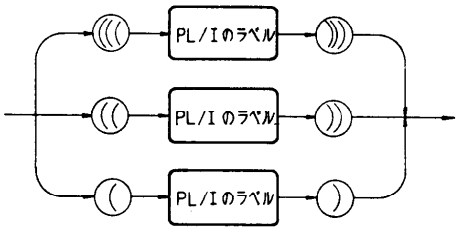

Fig. 8 An illustration of a syntax of labels

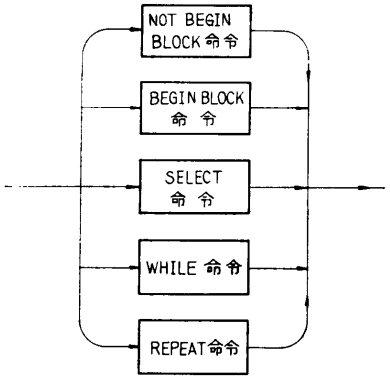

Fig. 9 An illustration of a syntax of statements

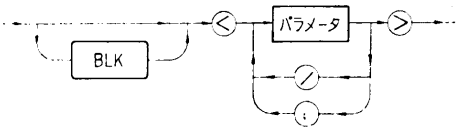

Fig. 10 An illustration of a syntax of a NOT BEGIN BLOCK command and a BEGIN BLOCK command

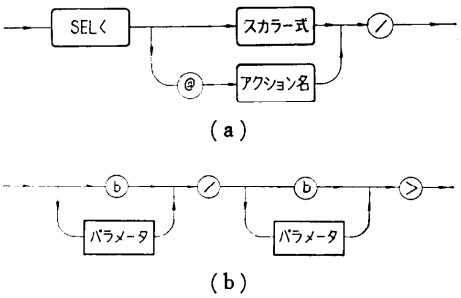

Fig. 11 An illustration of a syntax of a SELECT command

3.6 LSP(PL/I) トランスレータ

このトランスレータは，LSP(PL/I) により記述された分解過程を入力として，PL/I の構造化プログラムを出力とする．このトランスレータは FACOM 230-OSII-PL/I によりコーディングした．ステートメント数は約 2000 である．

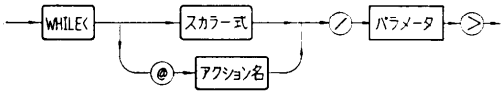

Fig. 12 An illustration of a syntax of a WHILE command

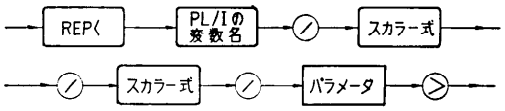

Fig. 13 An illustration of a syntax of a REPEAT command

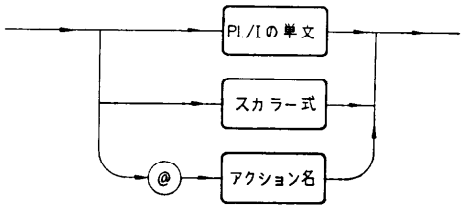

Fig. 14 An illustration of a syntax of a parameter

4. 実行例

本章では，簡単な構造化プログラミングの例を示す．

4.1 問題

奇数次の魔法陣を作れ．

4.2 アルゴリズム

魔法陣の次数を n とする．魔法陣の第 i 行，第 j 列を M(i, j) で表わす．すると次のアルゴリズムで n が奇数の場合の魔法陣を求めることが可能である．M((n+1)/2, n) は 1 である．

I, J, k を (n+1)/2, n, 1 とおく．

I, J を 1 ずつ MOD. n で増加する．n 回目ごとには，J を 1 へらし，I はそのままとする．

この M(I, J) の系列に順次 2, 3, 4, …, n^2 を与えればよい．

4.3 詳細化の過程

4.4 トランスレータの入力と出力

5. むすび

LSP(PL/I) を用いたプログラミングシステムは構造化プログラミングにおける機械的，形式的な作業の大部分からプログラマーを解放し，その結果として，プ

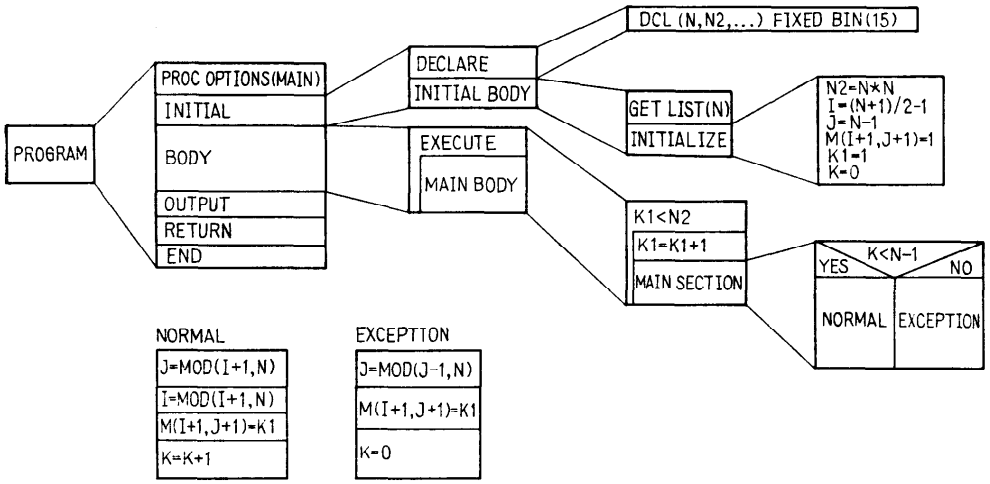

Fig. 15 An illustration of an example of a refining process

```
1  (((MAGIC))) PROC:=<PROC OPTIONS(MAIN)/ @INITIAL/@BODY/@OUTPUT/
2                    RETURN>
3          INITIAL:=<@DECLARE/@INITIAL BODY>
4               DCL:=< DCL (N,N2,K,K1,I,J,M(50,50)) FIXED BIN(15)>
5               INITIAL BODY:=< GET LIST(N)/@INITIALIZE>
6                    INITIALIZE:=<N2=N*N;I=(N+1)//2-1;J=N-1;
7                                M(I+1,J+1)=1/K1=1/K=0>
8          BODY:=WHILE< @EXECUTE/@MAIN BODY>
9               EXECUTE:=<K1<N2>
10              (KYOTA) MAIN BODY:=<K1=K1+1/@MAIN SECTION>
11                   MAIN SECTION:=SEL<<<N-1/@NORMAL/@EXCEPTION>
12                        NORMAL:=<J=MOD(J+1,N)/I=MOD(I+1,N)/
13                                M(I+1,J+1)=K1/K=K+1>
14                        EXCEPTION:=<J=MOD(J-1,N)/M(I+1,J+1)=K1/K=0>
15         OUTPUT:=<PUT LIST ((((M(I,J)) DO I=1 TO N) DO J=1 TO N)>>>
```

Fig. 16 An example of an input for the translater

```
                    TRANSLATOR FOR LSP(PL/I) PROGRAM

STRUCTURED PROGRAMMING LIST

  1 MAGIC:PROC OPTIONS(MAIN);
                                                              /*
          << INITIAL >>                                       */
                                                              /*
          << DECLARE >>                                       */
  2       DCL (N,N2,K,K1,I,J,M(50,50)) FIXED BIN(15);
                                                              /*
          << INITIAL BODY >>                                  */
  3       GET LIST(N);
                                                              /*
          << INITIALIZE >>                                    */
  4       N2=N*N;I=(N+1)/2-1;J=N-1;M(I+1,J+1)=1;
  8       K1=1;
  9       K=0;


                                                              /*
          << BODY >>                                          */
 10       DO WHILE(
                                                              /*
               << EXECUTE >>                                  */
               K1<N2);
                                                              /*
               << MAIN BODY >>                                */
 11       KYOTA:K1=K1+1;
                                                              /*
               << MAIN SECTION >>                             */
 12            IF K<N-1
                                                              /*
                    << NORMAL >>                              */
 13                 THEN DO;
 14                      J=MOD(J+1,N);
 15                      I=MOD(I+1,N);
 16                      M(I+1,J+1)=K1;
 17                      K=K+1;
 18                 END;
                                                              /*
                    << EXCEPTION >>                           */
 19                 ELSE DO;
 20                      J=MOD(J-1,N);
 21                      M(I+1,J+1)=K1;
 22                      K=0;
 23                 END;

 24       END;
                                                              /*
          << OUTPUT >>                                        */
 25       PUT LIST ((((M(I,J)) DO I=1 TO N) DO J=1 TO N));
 26       RETURN;
 27  END;
```

Fig. 17 An example of an output by the translater

ログラマーにより多い問題解決のための時間を与えてくれる. またLSP(PL/I)を用いる事は必然的にプログラムの形を制限し, 自然に構造化プログラミングを可能とする. 例題の場合には, 15行の入力により52行のプログラムが作り出されている, しかも作り出されているプログラムは紙幅136桁を十分に活用している. この様にカードにより構造化プログラムを直接 PL/I コンパイラに入力する場合には不可能であったより複雑な構造を持つプログラムも表現可能となった.

今後の問題点としては, 本論文では PL/I に制限されている目的言語を拡張する事が考えられる. 特にアセンブラの様に言語レベルの低い場合には, プログラムの制御を簡単に記述でき, かつ複雑にならざるを得ない構造を見易すく表現する上で大きな助けになると考えられる. また別の問題として, 構造化プログラムの出力形式をより見易い形式に改良する事が考えられる. このためには多数の人による使用経験を検討する必要があろう.

最後に, 本研究において御協力をいただいた田中研究室の諸氏に感謝の意を表します.

参 考 文 献

1) O. J. Dahl, E.W. Dijkstra, and C. A. R. Hoare; "Structured Programming", p. 220, Academic Press, London, (1972)

2) E. W. Dijkstra: "Go to Statement Considered Harmful (Letters to the Editor)", Comm, ACM, Vol. 11, pp. 147-148 (1968)
3) Donald E. Knuth: "Structured Programming with GO TO Statements", National Science Foundation; IBM Corporation,. (1974)
4) Niklaus Wirth: "Systematic Programming, An Introduction", Prentice-Hall, Inc., Englewood Cliffs, N. J., (1973)

(昭和49年7月5日受付)
(昭和49年9月2日再受付)