ソフトウェア工学 40－14
（1985. 2. 8）

理解容易なプログラムの構造上の特性について

今泉 恵美子　　落水 浩一郎
（静岡大学大学院電子科学研究科　静岡大学工学部情報工学科）

1．はじめに　自分自身で作成したプログラムの変更と他人が作成したプログラムの変更では，後者の方が，変更に伴う波及効果を把握しにくい．このことから，他人が作成したプログラムは理解しにくいと考えられる．

ところで，プログラムテキストの書き方をどれほど改善しても，一読しただけで理解できるようにはできない．しかし，プログラムテキストをどのように編成すれば，必要な情報を自明な形で提示できるのかをまず明らかにすることは，このような問題を解決していく上での一つの基礎となる．　本論文では，プログラムを理解したということを，プログラムによって引き起こされる計算機の動作（計算）を正しく把握できることであるとし，理解に必要な要因について分析し，それを直接反映するようなテキストの記述法をさぐる．

2．問題点の分析　最初に理解という立場からプログラムの記述法を分析したのが，E.W.Dijkstra の構造化プログラミングである．　分析の対象は，1つのプログラム，並行制御を含まないという範囲に限定されてはいるが，システム設計レベルにも適用できる発展性に富む結果が得られている．構造化プログラミングの主旨は，①理解するための基本的作業は，1ステップずつ動作をおうこと，ループに対しては数学的帰納法を用いること　②理解しやすいとは，プログラムの長さに比例する程度の努力ですべてのパスの動作を把握できること，という前提のもとに，流れ制御を1入口，1出口に制限し，データの抽象化と組み合わせて，独立な流れの部分を分離することである．この上に，このような条件をみたすプログラムテキストの記述法(PearlとNecklace）を試作し，プログラムの作成，変更，実行状態の把握等への応用を示している．

ところで，独立な流れを分離し，レベルに分解することは主に設計段階における利点として議論されており，理解にあたっての効果は明らかにされていない．設計段階におけるレベル分けそのままでは，多くの場合，1ステップずつ動作を追跡できる範囲を限定し，理解にあたって逆効果を生みだす．たとえば，図1は文献［1］中にある，2を最初とする初めの 1000 個の素数を作成するプログラムで

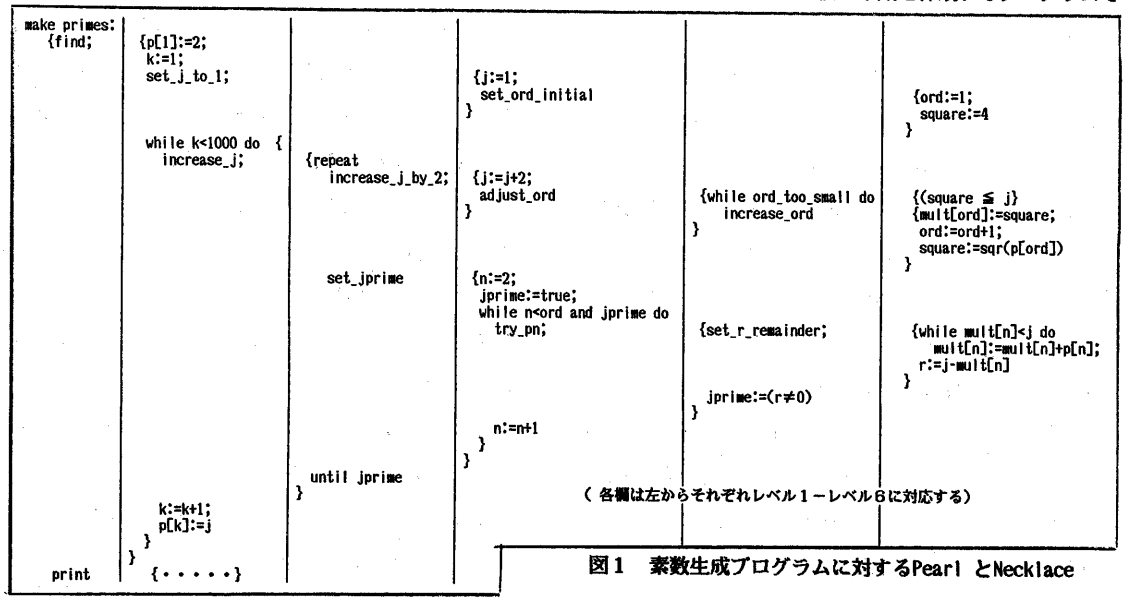

図1　素数生成プログラムに対するPearl とNecklace

ある．上位のレベルで実際の動作が記述されていない部分（例えば，set_jprime）は，その部分に対応する問題中の説明を純効果として補いつつ，レベル内の動作を１ステップずつ追跡することになるが，より下位レベルの動作の追跡と比べて動作把握が不十分となる．そこで，下位のレベルで定義される実際の動作を補って追跡しようとすると，手続きの詳細化の結果が散在するため，連続的に読みたい計算の流れ（変数の値変化や特定の時点における値の特性等）が理解しにくくなる逆効果を生じる．この原因は，独立な流れを分離するための基準が不十分であり，そのため，不適切なレベルの分解を生じていることにあると考えられる．本論文では，独立な流れに対するいくつかの基準を，主に多重ループの場合を中心にして考察する．

３．流れ制御に関する諸定義

３．１　計算とテキスト要素に関する定義　１つの計算が開始されてから停止するまでの時間の任意の時点は，記憶空間の値，入出力装置等の状態が変化しつつある点と，そのような変化が起っていない点に分けられるものとする．前者を動作点，後者を離散点とよぶ．各動作点での動作は，テキスト中の代入文，入出力文（これを動作文とよぶ）によって引き起こされる．また，離散点では，直前の動作点に対応する動作文から直後の動作点に対応する動作文に制御がうつされる．

１つのプログラムによって引き起される任意の計算の離散点において，直前の動作点に対応する動作文をａ，直後の動作点に対応する動作文をｂとするとき，ａ－ｂをその離散点に対応するテキスト区間といい，ａをこの離散点の先行点，ｂを後続点とよぶ．また，動作文ａと動作文ｂの間には，制御をａからｂに移す記述（；や条件分岐）が存在する．　先行点集合　$A=\{a_1, a_2, \cdots, a_n\}$ において，各先行点 a_i の後続点集合を $B_i=\{b_{i1}, b_{i2}, \cdots, b_{im_i}\}$ とし，$B=\bigcup_i B_i$ とするとき，テキスト区間 a_i-b_{ij}（$a_i \in A$，$b_{ij} \in B_i$）の和集合をテキスト領域とよび，Ａ－Ｂと表わす．また，ＢをＡの後続点集合とよぶ．　計算の再現性を以下のように定義する．

今，１つのプログラムから２つの計算が引き起こされ，次の条件が満たされているとする．　①離散点の数が等しい．　②一方の計算のｉ番目の離散点と，他方の計算のｉ番目の離散点には，同じテキスト区間が対応する．　③一方の計算のｉ番目の離散点と，他方の計算のｉ番目の離散点は，記憶空間，入出力装置等の状態が等しい．　このとき，この２つの計算は等しいとする．これより，あるプログラムから引き起こされる計算の集合（以後，計算の集合と略す）を，それによって引き起こされる互いに異なる計算の全体として定義する．以下，本論文でいうプログラムの理解とは，プログラムテキストを読むことにより，計算の集合を把握することであるとする．

例　１　図２に以上の例を図示する．

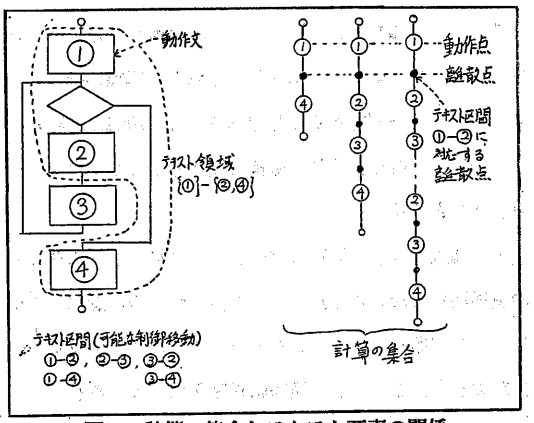

図２　計算の集合とテキスト要素の関係

３．２　条件分岐の意味と形式化　異なる離散点の系列を生みだすものは，制御移動記述のうち，条件分岐である．ここでは，流れ制御に関する種々の特性を考察するのに必要となる条件分岐の形式化を行なう．

定義　１　交差領域　先行点集合 $A=\{a_1, a_2, \cdots, a_n\}$，各先行点 a_i の後続点集合を $B_i=\{b_{i1}, b_{i2}, \cdots, b_{im_i}\}$，Ａの濃度をｎ，$B_i$ の濃度を m_i とするとき，テキスト領域Ａ－Ｂが次の２つの条件を満たしているとする．

①$n=1$ かつ $m_1 \geq 2$，または $n \geq 2$ かつ $m_i \geq 1$

②任意の B_i に対して，$B_i \cap B_j \neq \phi$（$i \neq j$）

このとき，テキスト領域から動作文を除いた最大のテキスト範囲を交差領域とよび，同様にＡ－Ｂと表わす．

例　2　図3に交差領域の例を図示する．

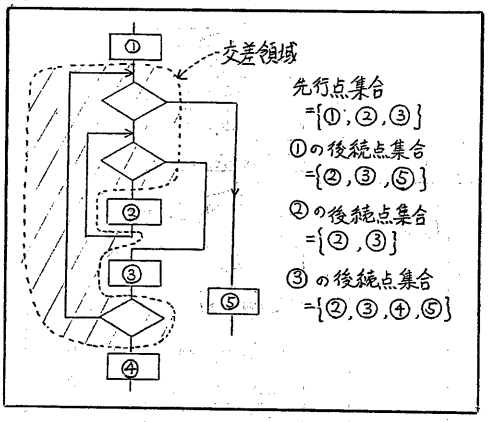

図3　交差領域の例

定義　2　合流領域　定義1に加えて　さらに，

$D = B_1 \cap \cdots \cap B_n \neq \phi \quad (n \geq 2)$

$A \cap D = \phi$

が満たされているとき，テキスト領域A－Dを合流領域とよぶ．

定義　3　分岐領域　先行点集合　$A = \{a\}$ に対し，$A \cap B = \phi$ を満たす後続点集合　$B = \{b_1, b_2, \cdots, b_m\}$ $(m \geq 2)$ が存在するとき，テキスト領域A－Bを分岐領域とよぶ．

例　3　図4に例を示す．

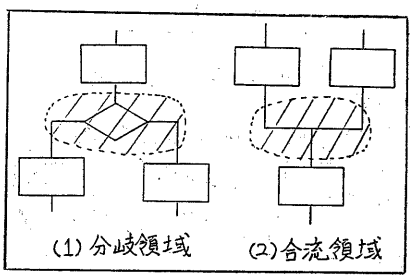

図4　分岐領域と合流領域

定義　4　ループ　先行点集合　$A_1 = \{a_{11}, a_{12}, \cdots, a_{1n_1}\}$ $(n_1 \geq 1)$ と各先行点 a_{1i} に対する後続点集合　B_{1i} $(|B_{1i}| \geq 1)$ から成るテキスト領域，および，先行点集合　$A_2 = \{a_{21}, a_{22}, \cdots, a_{2n_2}\}$ $(n_2 \geq 1)$ と各先行点 a_{2j} に対する後続点集合　B_{2j} $(|B_{2j}| \geq 1)$ から成るテキスト領域があり，以下の性質を満たすものとする．

$A_1 \cap A_2 = \phi$

$B_1 = B_{11} \cap \cdots \cap B_{1n_1} \neq \phi$

$B_2 = B_{21} \cap \cdots \cap B_{2n_2} \neq \phi$

$B = B_1 \cap B_2 \neq \phi$

ここで，以下の3条件を満たす $B' \subseteq B$ が存在すれば，$A_2 - B'$ をループ区間とよぶ．

① $b \in B'$ は，A_1 のどの要素も通過せずに，A_2 のいずれかの要素に到達可能である．

② $a_{2j} \in A_2$ は，A_1 のどの要素も通過せずに，B' のいずれかの要素から到達可能である．

③ $b \in B$ かつ $b \notin B'$ を満たす任意のbは，①または②を満たさない．

さらに，$A_1 - B'$ を入口区間，A_1 を入口点集合，B' を開始点集合，A_2 を繰り返し点集合とよぶ．ループ点と①の通過点を合わせたものを $A_1 - B'$，$A_2 - B'$ で定義されるループとよぶ．Cを $A_1 - B'$，$A_2 - B'$ で定義されるループの動作文とするとき，$c \in C$ を先行点とし，$d \notin C$ を後続点とするテキスト区間において，cを脱出点，dを出口点と呼ぶ．

例　4　図5にループの例を図示する．

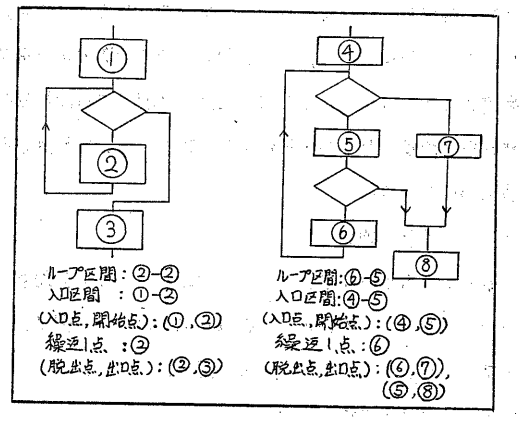

図5　ループの例

定義　5　連結点　動作文aの後続点は，動作文bただ一つであり，かつ，bの先行点はaただ一つであるとき，テキスト区間a－bを連結点という．

例　5　連結点

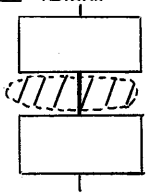

図6　連結点

図1で示したプログラムのフローチャートによる表現を図7に示す．また図8は，本章で述べた定義に基づいて，図7を位相的に変形したものである．図8から明らかなように，我々の定式化は，交差領域を，動作文間の構造を持つ区切り記号（すなわち，コンパイラ言語の；と同じ役割を果たすもの）としてとりあつかうところに特徴がある．

4．計算の組織化と独立な流れの関係

計算の集合の要素である各計算の構造上の差は，図8における交差領域によって生みだされる．　すなわち，動作文により一連の値の参照・設定が行なわれた後，変数のある特定の値状態で交差領域に入る．その状態に依存して，交差領域内のテキスト区間を通過した後，交差領域からでて再び動作文による状態の変化が起こる．このとき，交差領域内の条件分岐に付随する分岐領域，合流領域，ループ等の各特性は，計算の組織化をおこないテキスト上の選択構造，繰り返し構造，異常脱出等の制御構造を編成するための基本要素となる．プログラムの実行によりどのような計算が生成されるかを，変数の値を変化させる動作文と動作文の選択を行なう交差領域を基にして，どのように認識しテキスト表現すれば，より少ない手間で計算の集合を把握できるだろうか．ここでは，独立に理解できる流れ制御部分を定めることに関する若干の考察を，主に繰り返し構造について試みる．

4．1　流れの独立性

構文上2重ループであっても，定義4に従って交差領域を分析すると，1重のループになる場合がある．図7のループL_1，L_2を用いて例示する．図7において，

以下のテキスト領域

先行点集合　A_1= {⑤}

後続点集合　B_1= {⑥，停止点}

および，

先行点集合　　A_2= {⑪，⑮　⑰}

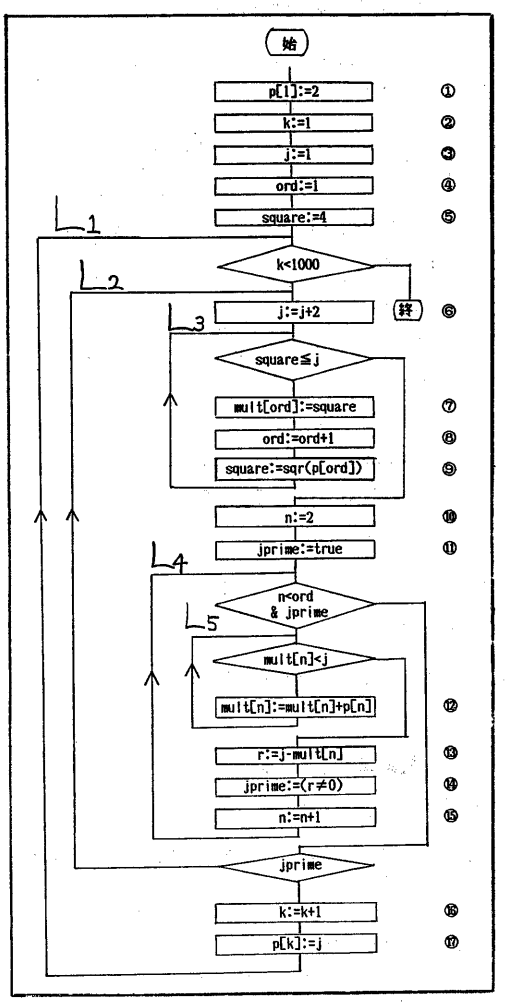

図　7　素数生成のプログラム

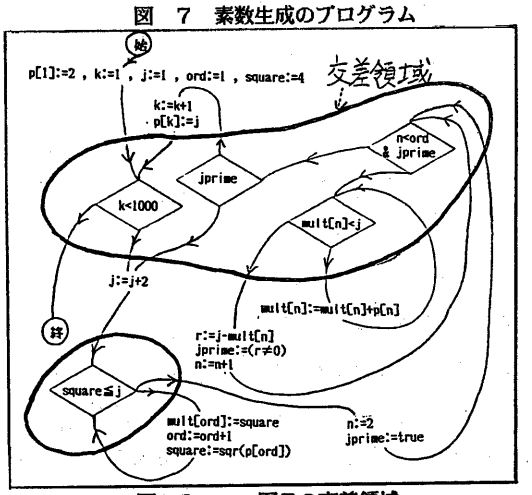

図　8　　図7の交差領域

⑪の後続点集合　B_{21}＝{⑥，⑮，⑬，⑫}

⑮の後続点集合　B_{22}＝{⑥，⑮，⑬，⑫}

⑰の後続点集合　B_{23}＝{⑥，停止点}

に対して，

$B_1=B_{11}\neq\phi$

$B_2=B_{21}\cap B_{22}\cap B_{23}=\{⑥\}\neq\phi$

$B=B_1\cap B_2=\{⑥\}\neq\phi$

⑥からは，A_2 のすべての要素に到達可能．

したがって，A_2-B はループ区間であり，⑤が入口点，⑥が開始点，⑪，⑮，⑰が繰り返し点である．したがって，⑯，⑰は，L'_4 の脱出状態によって条件的に実行される動作文である．この脱出状態をテキスト区間⑮－⑯，⑮－⑥に対応する離散点について検討してみる．

計算内で，⑮－⑯に対応する離散点が生起するときには，⑮の終了時点で n≧ord ∧ jprime=true が成立している．同様に，⑮，⑥に対応する離散点が生起するときには，⑮の終了時点で jrime=false が成立している．jprime の値に寄与している実行文は⑭であり，⑮は寄与していない．したがって，⑥に制御が移ることは⑭の終了時点で定まる．また，L_4 はループの入口でｎの値が新規代入され（⑩），L_4 を脱出するときに⑮が実行されたか否かは，次に L_4 に制御が移されたときの流れに影響を与えない． jprime=false が成立するとき，⑭－⑥に対応する離散点が成立するように⑭から⑮の流れ制御を変更すると，⑮に制御が移ったとき，jprime=true が成立している．したがって，⑮－⑯に対応する離散点が生起するためには，　n ≧ord が成立すればよい．これより，L_4 の脱出状態の違いを明らかにすると，図９のようになる．

以上により，<u>構文的に多重ループであっても，内側のループの脱出状態が外側のループの流れに影響を与える場合は，それぞれが独立な流れではないことがわかる</u>．したがって，図７の L_1，L_2，L_4 は，図１において３レベルに分けて記述されているが，１レベルに記述されるべきである．

４．２　変数の値変化系列とループ

ループにより引き起こされる計算に関して，値変化の時

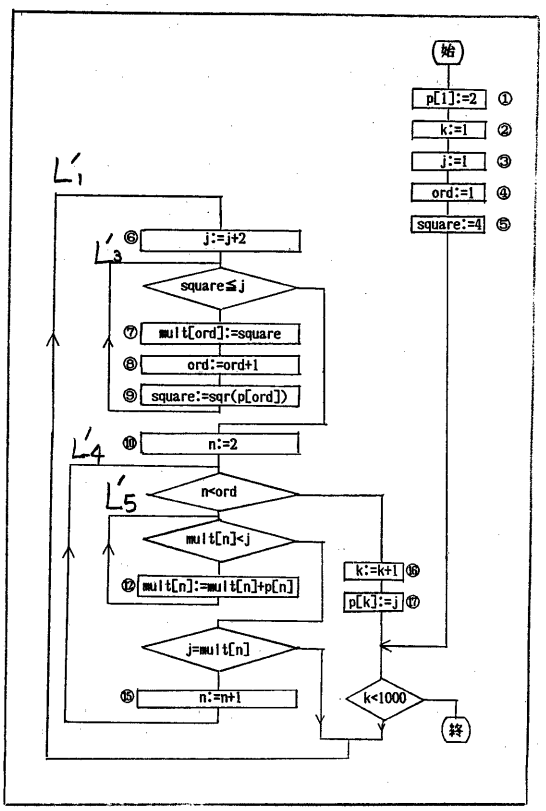

図９　多重ループの再編成

系列構造が類似した変数がある．このような変数に値を設定する代入文が，同一のレベルで記述されることは，レベル分解にあたっての一つの基準であると考えられる．以下，図９の例を用いて説明する．

図９において，ｎの値は L'_4 に入るときに２に設定され，L'_4 が継続する間，⑮によって値が更新される．L'_1 が１回実行されるたびに，この一連の変化は同様に起こる．したがって，ｎの値変化の系列は，L'_4 により定まる．

ｊの値は，L'_1 が１回実行されるたびに，⑥により変化し，値変化系列は L'_1 により定まる．

ord は，L'_1 の外側で初期設定され，L'_1 が継続する間，値が継続している．値を変化させる文（⑧）は，L'_3 に含まれており，値の変化系列は，L'_1，L'_3 により定まる．

mult[ord]，mult[n] は，L'_3 で ord の値が選択され，初期設定される．さらに，L'_4 で ｎ の値が選択されて，L'_5 の中で更新され，L'_1 を通じて値が継続される．した

がって，値の変化系列は，L'_1,L'_5により定まる．

値の変化系列を定めているループが共通である変数に対する代入文は，同一のレベルで記述することが，繰り返し構造によって生じる一連の計算（変数の値変化や特定の時点における値の特性等）を理解するには必要であると思われる．他の変数についても，値の変化系列を定めているループを調べ，４．１の結果と合わせて，図１のプログラムに対してレベルを与えると，図10のようになる．

```
   {init_k;                  k:=1
                             p[k]:=2;

    init_j;                                  {j:=1;
                                              ord:=1;
                                              square=4;
                                             }
    goto D;
A:  increacse_j;                             {j:=j+2;
                                              while square≦j do {
                                                mult[ord]:=square;
                                                ord:=ord+1;
                                                square:=sqr(p[ord]);
                                              }
                                             }
    n:=2;
B:  if n>ord goto C;                         {whilw mult[n]<j do
      increase_m;                               mult[n]:=mult[n]+p[n];
                                             }
    if j=mult[n] goto A;
      n:=n+1;
      goto B;
C:  set_p;                    {k:=k+1
                               p[k]:=j
                              }
D:  if k<1000 goto A;
```

図　１０　素数生成プログラムの再レベル化

本章で述べたレベル分けの基準は，他のいくつかのプログラムにも適用した結果，かなり一般性をもつ規則であるように思える．これを形式化し，段階的詳細化における明示的規則とすることは，今後の課題である．

また，本論文では，分岐・合流領域は通常の意味でのforkとjoinとして形式化したが分岐領域中の条件分岐が，その影響を交差領域のどの範囲に与えるのかという観点から分析すると，対応する合流点迄であるという直観的予想は必ずしも十分ではない．同様に今後の課題である．

レベル分けの有効な基準を見い出すためには，流れ制御に関する問題としては，次の３点を考察する必要がある．

①その動作を独立に理解できる流れ制御部分

②流れ制御に依存する変数，動作文

③条件分岐の影響範囲

５．おわりに

プログラム理解に必要な情報を自明な形で示しうるテキストの編成方法を明らかにするためには，次の４つの点を考察する必要かある．

１．プログラムを理解するとは，それによって引き起される計算を理解することであるとして，計算をどのように特徴づけるか．

２．計算の集合をテキストの長さに比例する量で把握するために望ましいテキストの構造上の特性は何か

３．２に対して理解のし易さの評価をどのように行なうか，どのような基準を設けるか．

４．３のような評価方法を直接反映するようなテキストの書き方があるか．

さらに，４のようなプログラムテキストを書くとしたら，プログラム設計上の要素は何か（コンパイラに入力するものがプログラムテキストであるときは，設計の要素は構文則である）というように展開することによって，変更（再利用）容易なプログラム設計法［２］開発の一つの基礎を築ける．　本論文では，主に１，２について基本的かつ基礎的な考察を行なった．　また，

①プログラム中における独立な流れを定義できること

②プログラムテキストをレベルに分解できること

は量的問題が本質であるこの種の問題に対して解決されるべき要件であるが，本文中では流れ制御の問題に関して若干の基礎的考察をおこなった．レベル分けの基準にはデータの抽象によるものがある．すなわち，配列のような構造をもつ変数を使うプログラムにおいて，配列の添字を認識する順番をレベル分けの基準とするものである．このとき，レベル間の独立性を保証するには，作成段階においてデータ型の導入，詳細化に制限を設ける必要があり，今後の課題である．

参考文献　［1］E.W.Dijkstra,"Notes On Structured Programming",「Structured Programming」,ACADEMIC PRESS (1972)．［2］落水，今泉，"ソフトウェア開発と保守作業の形態の同質性について" 本研究会資料34-5(1984)．