論　文

階層状体系をもつプログラミング言語の考察*

有　澤　　誠**

Abstract

In order to design a programming language for use of structured programming, a single language seems not to be enough. Since "shell approach" (everything in all) nor "core approach" (extensible language) does not satisfy the present objectives, here we propose a third approach, a language system with hierarchy.

ESDL was one such experiment and is discussed briefly. The we establish desirable relationships between language levels, and consider what each level should be like. Throught the paper the arguements are made on general conceptual basis and no specific language is proposed.

1. まえがき

プログラミング言語は，その目的に応じて種々数多くが提案設計され，またインプリメント使用されている．ひとつの言語では，要求されるすべての機能をカバーしきれないか，または特定の目的のために広い機能をカバーする言語は効率等の面で適当でないか，または現実にそれぞれの言語を習得したあとでほかの言語にきりかえることを好まないか，原因はいろいろ考えられる．

しかし，ひとつの言語でできるだけ多くのプログラミングの仕事をカバーしようとする試みもまた，かなり以前からなされている．プログラムの融通性，ソフトウェア生産コストの低下，プログラミング教育の面など，たしかに言語の一本化によるメリットは大きいであろう．

従来の言語の統合化の試みは，大きく2つに分けることができる．第1は shell approach であり，第2は core approach である．

ここで shell approach というのは，ひとつの言語の中にすべての機能をとりこみ，言語の複雑化は可分離性 (modularity) や直交性 (orthogonality) によって救済しようとするものである．PL/I はこの代表的なものといえる．

また core approach とは，限られた基本成分と，言語に備わっている自己拡張機能とを用いて，ひとつの基本言語からユーザの目的別に拡張された言語を使用できるというものである．EL1[13] など拡張言語 (extensible languages) とよばれるのは，このグループに属する．

この2つのアプローチにはそれぞれ一長一短があって，現在どちらについても決定的なものはない．ただ PL/I はそれなりの成功をみており，また特に研究者間の小グループでは，拡張言語もある程度実用になっているようである．

ここ数年注目を集めている構造的プログラミング (structured programming)[1),7),8)] の立場からみると，この2つのアプローチはいずれも十分であるとはいえない．そこでこの論文では，第3のアプローチとして，階層状体系をもつ言語群を提唱する．これは，いくつかのコンパクトな目的別言語を，ひとつの階層状に配しておき，レベルの異なる言語間でもある統一性を保つよう設計しておくものである．

このような言語（群）のひとつである，システム記述言語 ESDL[10] について，**2.** でその設計思想などについて簡単に紹介する．

ESDL は，必ずしも構造的プログラミングの見地から理想的な階層状体系になってはいない．そこで **3.** では，ESDL のレベル分けをベースにした，より新しい階層状体系をもつ言語を提案する．ここでは言語の設計思想に関して考察し，具体的な言語仕様は提唱し

* Programming Languages with Hierarchical Structure by Makoto ARISAWA (Faculty of Engineering, Yamanashi University)
** 山梨大学工学部

ない．Hoareの「言語設計の一般論は，ひとつの言語仕様の設計とはきりはなすべきである」[4]という立場を本稿でもひきついでいる．

なお，「階層状」「レベル」などという概念は時と場合によっていくつかの異なった意味あいで用いられているようである．3.の議論では，それを整理分類して考察する．

2. ケース・スタディ：ESDL の階層状体系

ESDL* は，OSなどのシステム・プログラム記述用の言語として 1968 年に設計された．システムの総括的設計の過程からはじめて，個々のモジュールのコーディングに至るまでの一連の仕事を，ひとつの言語体系で処理することを目標としている．

言語の最も低いレベルには，ハードウェアを効率よく使用する要請から，マクロ・アセンブリ言語をとりあげる必要があった．OS の記述では，特権命令を含めたアセンブリ命令，EXCP レベルに至る I/O マクロ命令は必須である．

一方最も高いレベルとしては，全体的な設計図としてのフロー・チャートの使用が一般的であると考えられ，特に並列処理の同期を含めたフロー・チャートによる処理の流れを，このレベルで図の形に表現することをねらった．

その結果，この間にいくつか中間レベルを設けて，計6レベルからなる言語体系が考えられた．最も低いレベルをAレベル（アセンブリ言語レベルの意），以下B～Eレベル，最も高いレベルをFレベル（フロー・チャートレベルの意）と名づけている．

Bレベルは，PL 360[14] や SAL[9] などの，構造をもった高級アセンブリ言語である** このレベルでは，あくまでマシン依存言語という性質は保持しながら，できるだけデータの構造を一般化し，コントロールの流れを構造化しておくものである．

その上のCレベルになると，機械への依存性のない一般的なアルゴリズム記述用言語になる．ALGOL 60 に近い形のものを想定している．

Cレベルでのデータ構造や演算をより抽象化したものがDレベルである．文献2) にみられる ALGOL-C および ALGOL-D に対応させて，Cレベルを拡張言語の基本言語，Dレベルを拡張機能によって拡張されたあとの目的別の言語，という形でとらえていた．

Eレベルは，コーディングとは関連を直接もたせずに，プログラムの設計のほうの立場から，Fレベルの具体化という形を考えている．シミュレーション言語のレベルとされ，Fレベルにある並列処理も，まだそのままの形で，並列にプロセスが走る形として残されている．SOL や SIMULA[6] など ALGOL 型のシミュレーション言語に似たものを想定していた．ただし SIMULA 67 のクラスの概念は考慮されていなかった．

以上の6レベルを図示したものが **Fig. 1** である．ここに示すように D→C へのレベル変換は，マクロ展開などの拡張機能によって，また B→A へのレベル変換は，コンパイラによって，それぞれ自動的に行うことを考えていた．一方 F→E, E→D, C→B については，人間による変換を想定しており，それだけレベル間のギャップが広いように考えられる．

ESDL は 1970 年までに実験システムと試作システムがインプリメントされた．この時には，必ずしも当初の言語設計の思想がインプリメントには反映されず

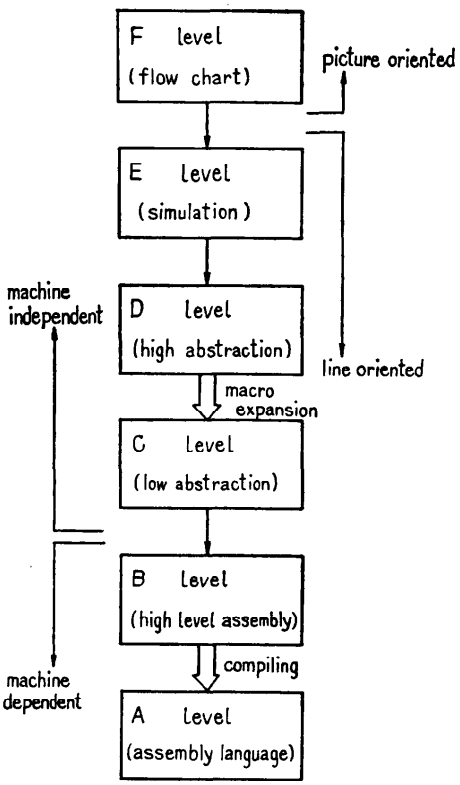

Fig. 1 Six level in ESDL

* ETL's System Description Language の意である．

** その後 PDP 用の BLISS[16),17)] が発表され，B の語呂もうまく合うようになった．

に，むしろ拡張言語を追求する姿勢が強くでてしまったようである．ESDLによるプログラミングは，本来次のようなものである．まずFレベルで概要を設計し，それをEレベルでプログラミングしてシミュレーションし，それぞれのモジュールごとに，必要な機能を定義したDレベルでプログラミングしなおし，それをCレベルにおとし，さらに能率の悪いところや機械に依存するところをBレベルでプログラミングしなおし，全体をコンパイラによってアセンブリ言語のレベルにおとして，最後のインタフェースの調整をすませて，機械語にする．

ESDLの実験は，言語の設計とインプリメントの2つのフェイズで，目標設定に変動があった．この論文でとりあげたいのは，前半の言語設計に関する部分である．ESDL全体については，文献10)を参照されたい．

3. 構造的プログラミングのための階層状体系

以上に，階層状体系をもつプログラミング言語を考察する上での背景を述べたので，これから本論にうつろう．

言語のもつ階層について，まず考えてみる．2つのレベル間での言語のもつ関係には，次のようなものが考えられる．

① 上のレベルで書かれたプログラムは，そのまま下のレベルのプログラムとしても通用する．

② 下のレベルで書かれたプログラムはそのまま上のレベルのプログラムとしても通用する．

③ 上のレベルのプログラムに許されるコントロール構造は，そのまま下のレベルのプログラムでも通用する．

④ 上のレベルのプログラムに許されるデータ構造およびデータ操作は*，そのまま下のレベルのプログラムでも通用する．

⑤ 上のレベルのプログラムに許されるコントロール構造の refinement が下のレベルのプログラムのコントロール構造になっている．

⑥ 上のレベルのプログラムに許されるデータ構造およびデータ操作の refinement が下のレベルのプログラムのデータ構造およびデータ操作になっている．

⑦ 上のレベルの言語の処理系を，下のレベルの言語でインプリメントできる．

⑧ 上のレベルのプログラムを，マクロ展開によって下のレベルのプログラムにおとすことができる．

⑨ 上のレベルのプログラムを，コンパイラ（またはトランスレータ）によって下のレベルのプログラムに変換できる．

以上である．これを詳しくみてゆく．①と②とは実質的に同じ内容であるが，①では下のレベルのプログラムの一部は上のレベルでは通用しないことを意味しており，また②では上のレベルのプログラムの一部は下のレベルでは通用しないことを意味している．

①の例としては，上のレベルが機械依存性のないマクロ命令の言語で，下のレベルが機械に依存する命令を含むような，マクロ・アセンブリ言語の場合である．また②の例としては，上のレベルがマクロ・アセンブリ言語で，下のレベルがアセンブリ言語の場合である．

③と④は自明であろう．⑤のコントロール構造の refinement とは，上のレベルにある if-then-else 文や while-do 文を，下のレベルでは branch (jump) 命令によって実現しているケースなどをさす．また⑥のデータ構造およびデータ操作の refinement とは，上のレベルにある行列構造のデータと行列演算を，下のレベルでは配列構造のデータと，スカラー演算をくりかえし文でコントロールする操作で実現しているケースなどをさす．

⑦のインプリメントとは，たとえば ALGOL-W[11] の処理系が PL 360 で作られている場合などをさす．また⑧のマクロ展開は，ALGOL-D と ALGOL-C の関係が一つの例である．さらに⑨は，コンパイラ自身がどの言語で書かれているかは問題でなく，要点は上のレベルから下のレベルへの変換が，第3のプログラム（コンパイラ）を介して自動的に行えることである．以上にあげたレベル間の関係は，プログラミングのレベルによって必要とされるものが変わってくる．なお2つ以上の関係がなりたつことも当然ありうる．

それでは次に，構造的プログラミングのための，言語のレベルの設定にうつる．レベルの種類はきっかり定まる性質のものではないので，ここでは前章に述べた ESDL の場合と並行させて，レベルを設定する．従って低いレベルAからはじまって，高いレベルFに

* データ操作はデータ構造と合わせて考慮してゆくべきであり，コントロール構造とは独立であるのが原則である．

いたる6段階を想定する．プログラミングの特殊性に応じて，A～F のレベルのいくつかを統合したり，どれかをさらに分離したほうが，プログラミングのために便利である場合が生じる可能性はあるけれども，階層状体系をもつ言語の一般的な議論では，6つのレベルで検討してよいと考えられる．

構造的プログラミングは，下降的 (top-down) 手法であるから，レベルの考察も高いレベルからはじめてゆく．最も高い F レベルは，構造的設計 (structured design)[12] や HIPO[5] などの，プログラムの全体的な構造を記述する言語レベルとすべきである．このレベルでは，問題の認識の記述が主であり，解を求めるためのアルゴリズムにはたちいらないか，または粗い手順の記述程度までにとどめておく．自然言語による注釈の形で書かれる部分のほうに，本質的な情報がまだ相当含まれている．

次のEレベルは，アルゴリズムが抽象的な形で記述されるレベルである．アルゴリズムのためのデータ構造やデータ操作は，一般のプログラミング言語では直接サポートされていない抽象的なものであってよい．また，使用するコントロール構造も，代入文，if 文および case 文，while 文および repeat 文，といった型にはまったものばかりでなく，その問題に合った多種のコントロール構造を許す．

プログラマが自由にデータ構造とデータ操作を選択できるためには，SIMULA 67[6] のクラスの概念をこのレベルにとりいれる必要がある．

3番目のDレベルでは，データ構造およびデータ操作に関する自由度は残したまま，コントロール構造を形の整った (well formed) いくつかの基本的なものに制限する．たとえば if 文と while 文以外の条件文やくりかえし文，およびそれと等価なコントロール構造を実現する goto 文などを禁止するなどである．従ってEレベルで定義したデータ構造の操作の記述についても，限られたコントロール構造だけで記述されることになる．

なお具体的にどのコントロール構造だけを許すかについては，現在の state of art では決定的なものはないように思われる*．どのコントロール構造についても一長一短があるので，それを論じることは言語の階層の議論とは別の機会になすべきであろう．

第4のCレベルに至ると，データ構造とデータ操作も，標準的な組み込みのものだけに限られる．たとえば，配列，構造体 (structure)，スタック，系列 (sequence) または列 (string)，線形リスト (linear list) などの効率よい操作手法が広く知られている構造だけが許される．効率の面から，言語のもつダイナミックな性質はある程度制限される．PASCAL[3),15] はひとつの妥協点のめやすを与えている．

第5のBレベルではじめて，機械依存性がもちこまれる．コンピュータごとの特徴を生かすことのできるデータ構造やデータ操作を用意する．コントロール構造は，DレベルおよびCレベルで許されるものは，原則としてそのまま許し，それに加えて機械の特徴を反映したもの，たとえば割り込み機能のプログラマによるコントロールなどが機械の特性に応じて加わる．

BLISS[16),17] はこのレベルの言語の設計の，ひとつのモデルであろう．ただし BLISS の accessing path の概念と型のないデータという考え方は，必ずしもすべての機械に便利であるとは考えられない．IBM 360/370 の系列では，データ型に関しては PL 360 をモデルとすることも自然であろう．

最も低いAレベルは，直接機械命令に対応したマクロ・アセンブリ言語である．ここでは，いくつかの標準的なマクロ命令を用意して，Dレベルからひきつがれたコントロール構造を，マクロの形で残すようにしておけば，アセンブリ言語のレベルでもまだ構造化を保つことがある程度可能になると思われる．

これで，6つのレベルの特徴づけがおわった．次にこれらのレベル間で，前述した①～⑨のどの関係がなりたつべきかを考えてゆく．

まず①であるが，C→B でこれを満たすべきであろう．また②については，E→D および D→C があてはまる．③の関係は D→C および C→B で，④の関係は E→D および C→B で，それぞれ満たされる．⑤に関しては F→E および E→D であり，⑥に関しては F→E および D→C である．また広い意味で，refinement を解釈すれば，B→A は⑤と⑥の両方があてはまる．⑦については，F～B→A は明らかである．ブート・ストラップを考えれば A→A も含められる．また F～C→B も可能なようにBを設定すべきであろう．部分的に手をいれる前提をおけば F～C→C，あるいはレベルの高低は逆転するけれども，B→C も考えうる．⑧は D→C でなりたち，機械によっては B→A でもなりたつ．また，言語の設計の段階で厳しい制限を加えれば，F→E, E→D, C→B に関しても⑧を満

* たとえば，くりかえし文ひとつとっても文献 7) 等に新しい有望な試みがいくつもみられ，「while 文に限る」といった断定をすべき時期ではないと考えられる．

たすようにすることも考えられないことではない．⑨が実は階層状体系をなすための眼目である．⑧のマクロ展開も，マクロ・ジェネレータという一種のコンパイラによる自動変換であり，その意味から隣りあう二つのレベル間では，⑨を満たすことが不可能と断定できるものはない．しかし，すべてを自動的に変換できるように言語を設計することは，困難でもありまた使い易くもないであろう．

この問題を逆の立場からみれば，人間が介入することの利益が大きいレベル間は，どことどこかということになる．はじめにレベルを設けたときのいきさつからみれば，どのレベル間にも必要に応じて人間が介入する余地が与えられる．その必要が生じる可能性が高いのはどこかということがここで問題になるのである．

一般に下降法のプログラミングでは，重要な決断は高いレベルで行われる．そのために，高いレベルで行う必要のない決断は，できる限り先に，すなわち低いレベルまでもちこされる．低いレベルにもちこすことのできたことがらは，それだけ低レベル言語のきびしい言語仕様で記述可能なわけであり，人間の介入の必要性は少いはずである．低いレベルでの人間の介入は，主に効率の向上に関する要求から生じる．いいかえれば，低いレベルほどレベル間の自動変換が容易なように言語を設計しておくべきである．低いレベルほど言語仕様が厳密化していることを考えれば，これはまた当然であろう．

これまでは，主にあい隣り合う2つのレベル間の関係をみてきた．あまり大きくないプログラムの場合に

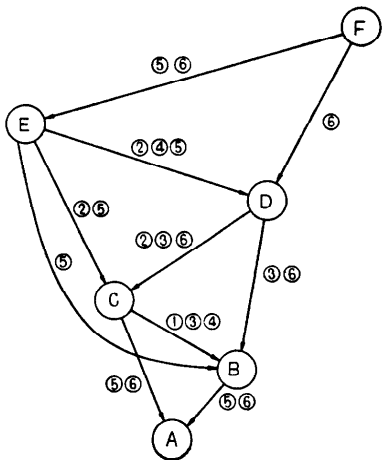

Fig. 2 Relations between the six levels.

は，この6つのレベルが全て必要ではなく，一部のレベルだけで十分な場合も多い．省略される可能性のあるレベルは，E, D およびBであろう．その場合Aレベルは，自動変換が適用されよう．

Fig. 2 には，こうした隣りあわないレベル間も含めて①〜⑥の関係がどう成り立つべきかを図示した．図をみやすくするために，⑦〜⑨は含めていない．この図の中で，③→③は③に等価，④→④は④に等価，③→⑤および⑤→③は⑤に等価，そして④→⑥および⑥→④は⑥に等価という性質を使用している．

いくつかのレベルの分離や統合を行う場合にも，Fig. 2 を描きなおして，各レベル間のバランスを考えてゆけばよいと考えられる．

4. むすび

ひとつの言語で，プログラミングの過程に生じるいろいろのレベルの概念を処理しようとせずに，いくつかの（ここでは仮に6つとした）レベルを設け，それぞれのレベルごとに閉じた言語を用意して，各言語間の連絡をうまくとりながらプログラミングをする．という方式について述べた．

ここでは，それぞれの言語の具体的な仕様にはふれず，コントロール構造と，データ構造およびデータ操作とに粗く2分し，レベル間の関連性から望ましい言語の位置づけを行った．

階層状体系をもつプログラミング言語を実際に設計するためには，Hoare[4] の示唆にもあるように，言語のバランスに注意を払うこと，特にこの場合は，隣り合うレベル間とのバランスにも留意することが大切であろう．SIMULA 67, PASCAL, ALGOL-W, BLISS などの言語仕様は，そのような前提の上で十分に参考にすべきと考えられる．

構造的プログラミングのための言語は，プログラムの構造化だけでなく，プログラム作成過程の構造化を考えて設計されねばならない．その意味から，階層状体系を言語（群）に与えておくことは，ひとつの解決法であると考えられる．

謝辞 この研究を側面からご援助いただいた，鳥居宏次，斎藤信男，古川康一，杉藤芳雄および真野芳久の諸氏に感謝する．

参考文献

1) O. J. Dahl, E. W. Dijkstra, C. A. R. Hoare: Structured Programming, Academic Press

(1972).
2) B. Galler, A. J. Perlis: A proposal for definition in ALGOL, CACM 10-4, pp. 204～219 (1967).
3) A. N. Habermann: Critical comments on the programming language PASCAL, Acta Informatica 3, pp. 47～57 (1973).
4) C. A. R. Hoare: Hints on programming language design, STAN-CS-73-403 (1973).
5) IBM: Management overview, Internal Report (1973).
6) J. D. Ichbiah, S. P. Morse: General concepts of the SIMULA 67 programming language, Annual Review in Automatic Programming 7, pp. 65～93 (1973).
7) D. E. Knuth: Structured programming with goto statements, Computing Surveys 6-4, pp. 261～301 (1974).
8) D. E. Knuth: A review of structured programming, STAN-CS-73-371 (1973).
9) C. Lang: SAL- system assembly languages, SJCC, pp. 543～555 (1969).
10) 斎藤信男, 杉藤芳雄, 有澤誠, 宮川正弘, 佐藤光昭: ESDL (ETL's System Description Language) について, 電気試験所彙報 34-5/6 ESDL 特集号 (1970).
11) R. L. Sites: ALGOL-W reference manual, (改訂版) STAN-CS-71-230 改 (1972).
12) W. P. Stevens, G. J. Myers, L. L. Constantine: Structured design, IBM System Journal 1974-2, pp. 115～139 (1974).
13) B. Wegbreit: The ECL programming system, FJCC pp. 253～262 (1971).
14) N. Wirth: PL 360- a programming language for the 360 computers, JACM 15-1, pp. 37～74 (1968).
15) N. Wirth: The programming language PASCAL, Acta Informatica 1, pp. 35～63 (1971).
16) W. A. Wulf: Programming without the GOTO, Proc. IFIP 71,, pp. 408～413 (1972).
17) W. A. Wulf, D. B. Russell, A. N. Habermann: BLISS- a language for system programming, CACM 14-12, pp. 780～790 (1971).

(昭和50年8月8日受付)