# プログラミングの基礎

九州大学教授・理学博士
藤野精一著

基礎数学シリーズ 18

朝倉書店

基礎数学シリーズ 18

# プログラミングの基礎

藤野精一著

朝倉書店

**著者略歴**

昭和 2 年　松山市に生まれる
昭和 28 年　九州大学理学部卒業
現　在　九州大学教授・理学博士

基礎数学シリーズ 18
プログラミングの基礎

昭和 54 年 1 月 20 日　初版第1刷



著者　藤野精一（ふじのせいいち）

発行者　朝倉鑛造
東京都新宿区新小川町 2-10

印刷者　大久保絢史
東京都新宿区市ヶ谷本村町 27

発行所

株式会社 朝倉書店
東京都新宿区新小川町 2-10
郵便番号 162
電話 東京 (260) 0141 (代)
振替口座 東京 6-8673 番
自然科学書協会会員



3341-111348-0032

# 基礎数学シリーズ
# 編集のことば

近年における科学技術の発展は，極めてめざましいものがある．その発展の基盤には，数学の知識の応用もさることながら，数学的思考方法，数学的精神の浸透が大きい．理工学はじめ医学・農学・経済学など広汎な分野で，数学の知識のみならず基礎的な考え方の素養が必要なのである．近代数学の理念に接しなければ，知識の活用も多きを望めないであろう．

編者らは，このような事実を考慮し，数学の各分野における基本的知識を確実に伝えることを目的として本シリーズの刊行を企画したのである．

上の主旨にしたがって本シリーズでは，重要な基礎概念をとくに詳しく説明し，近代数学の考え方を平易に理解できるよう解説してある．高等学校の数学に直結して，数学の基本を悟り，更に進んで高等数学の理解への大道に容易にはいれるよう書かれてある．

これによって，高校の数学教育に携わる人たちや技術関係の人々の参考書として，また学生の入門書として，ひろく利用されることを念願としている．

このシリーズは，読者を数学という花壇へ招待し，それの観覚に資するとともに，つぎの段階にすゝむための力を養うに役立つことを意図したものである．

# 基礎数学シリーズ　全30巻

編　集

京都大学名誉教授　理学博士
小　堀　　憲
京都大学名誉教授　理学博士
小　松　醇　郎
東京大学名誉教授　理学博士
福　原　満　洲　雄

# まえがき

本書の前身として「電子計算機-プログラミング-」が刊行されたのは 1967 年(昭和 42 年)であった．そこでは ALGOL プログラミングの修得に重点を置き，プログラミングならびにコンパイリングの手法を，なるべく'自然'に会得できるように種々配慮した．このような形を何等かの意味で残したいという気持も若干あるにはあったが，今回，改訂を考えるに際し，思いきってその構成を一変することにした．それは次のような考えによる．

（1） 近年急速に発展したプログラミングの基礎概念を的確に把握できるような配列にする．

（2） プログラムの理論のみならず計算機科学の基礎として重要なカテゴリ論の知識を早い機会にできるかぎりわかりやすい形で導入し，それによって全体を可能なかぎり統一的に叙述する．

（3） 計算機科学，とくにその中心話題の一つである人工知能論はコンピュータの応用として理論的にも実際的にも非常に興味深い分野である．この分野でのプログラム作成への指針を与える．

上記の考えを達成するため，いわば改訂とは名ばかりで，実際には最初の一頁からまったく新しい書物を作ることにした．そのため，書名も「プログラミングの基礎」と改めた．本書は旧版の読者にも，新しい読者にも同様に新鮮なコンピュータプログラミングの理論的基礎ならびにその応用を与えるものである．

本書は次に示すような構成からなっている．第1章でコンピュータの数学的モデルを導入し，コンピュータのカテゴリ **Comp** を，計算機科学における主要なカテゴリの具体的例としてとりあげ，次章のカテゴリ論の基礎理論展開への前奏とする．第3章では前章で得たカテゴリ論の知識をオートマトンのカテゴリ **Aut** に応用し，プログラミングの重要な基礎であるオートマトン論の立脚点を確立する．ひきつづいて，第4章では，同様な進展をプログラムのカテ

ゴリ **Prog** でも期待できることを示す．その際，グラフのカテゴリ **Gph** の重要性について触れる．

第5章と第6章の2章では主要なプログラミング言語の特徴を論じ，とくに近年非常に注目された記号処理言語の特徴については，一章を設けてすこしくわしい説明を加える．

以上の知識を利用して，プログラミングの注意と心がけとを第7章にとりあげ，データ構造の新しい構成法ならびに構造的プログラミングの解説に及ぶ．最近ますます必要度の高まった再帰プログラミングとその取り扱いについては第8章で述べ，最後の2つの章ではコンピュータの人工知能的応用として定理の自動証明ならびにプログラムの自動改良と自動合成とをとりあげ，最近の話題の焦点を明らかにする．とくに，プログラムの正しさをデータ構造の形を基にして証明するために新しく導入された構造帰納法を第8章のはじめに例題を通じて解説し，理論的基礎の確立に意を注ぐ．

これら各章の叙述は，紙数の関係で簡単に述べざるをえない点もあったが，最初にかかげた著者の意図を達成したものである．プログラミングの数学的基礎に関心を抱く読者の参考になれば幸いである．

最後に，本書を出版するに当たり，著者の意向を十分汲みとり，編集・校正に全力を傾注していただいた朝倉書店の皆様に深く感謝の意を表す．

1978 年 12 月

藤野精一

# 目　　次

# 1. コンピュータ

本章ではコンピュータの数学的モデルを構成し，プログラミングに関する二，三の性質を調べる．コンピュータの数学的モデルを構成しようという動きは，ずいぶん昔からあった．A. M. Turing (1935) が計算理論の基礎を固めるため考えだした思考機械は，いまでは Turing 機械とよばれているが，現在の形のコンピュータが作られる10年も以前のことであり，理論的には満足すべきものではあっても，現在の時点からみると，あまりふさわしいモデルであるとはいえない．記憶装置への，プログラムの内蔵方式とか，制御装置の命令解読，およびその実行という機構の表現が，とくに不十分であった．これらを改善し，現実のコンピュータに近いモデルにしようと，各国で多くの人が活躍した．1960年前後は，この研究が絶頂を極めたといってもよい時代であった．ソ連では A. P. Ershov (1958) の研究，東ドイツでは H. Kaphengst (1959) の先駆的業績があり，つづいて同じく東ドイツの G. Asser (1959) は，マルコフアルゴリズムを演算の基礎におく面白いコンピュータモデルを完成し，プログラムの概念を明確にした．アメリカでは，H. Wang (1959) がはじめて，プログラムで動く思考機械の概念を導入し，Turing 機械の匂いの半ば残る形ではあるが，以後の研究への大いなる刺激を与えた．さらに，C. Y. Lee (1960) は Wang の考えを余すところなく発展させた．しかし，コンピュータのふさわしいモデルとしては，C. C. Elgot と J. A. Robinson (1964) による RASP とよばれる思考機械の創設を待たねばならなかった．見事に開花した RASP 流のモデルとほとんど同時に，イギリスでは，J. C. Shepherdson と H. E. Sturgis (1963) によるプログラム機械が案出された．いずれも，明確に，内蔵されたプログラムによる計算という考えが打ち出され，モデルとしては，一応満足すべき状態に近づいた．しかし，これらのモデルは，モデルの形を覚えるのが一苦労であった．かといって，モデルを極度に簡単化すると，コンピュータの像が浮かばない．また，モデルの構造を一度覚えても，それでプログラムを書こうとすると，基本的なことはできても，応用がきかない．

もっと良い数学的モデルをという声は，その後もつづいた．ここでは RASP の考えを基礎におく，ポーランドの Z. Pawlak(1968) 一派が考案したモデルを，オートマトンの立場から見直すことにする．そして，後章との関連を考えながら，コンピュータに対する数学的考察が，このような形で可能であることを示そう．コンピュータ機構の簡単な説明から入り，各構成部分が集合と写像の組という数学的概念で，総合的にシステム表現化することが可能であることを示す．

## 1.1 コンピュータの数学的モデル

コンピュータは，入力装置を媒介にして，解こうとする問題のために作成したプログラムと，その問題に必要なデータ(数値，記号列その他の情報)とを，記憶装置に格納する．このプログラムとデータは，通常使用者が理解しやすい，コンピュータごとに定まったプログラミング言語(たとえば，フォートラン言語(FORTRAN))で許される規則に従って書かれ，一定の約束にしたがって，カードまたはテープに穿孔(さんこう)されて，入力装置にかけられる．場合によっては，タイプライタを使用して，コンピュータに直接入力することもある．

記憶されたプログラムは，入力装置にかけたプログラムとは，形は異なるが，全体として，同じ意味をもつ有限個の命令列に変換されて格納されている．このとき使用される各命令は，コンピュータのもつ機械固有の命令で，記憶装置の各記憶場所にそれぞれ格納される．また，データも，このコンピュータ固有の機械表現に変換されて，格納される．このような，外部で書かれたプログラムやデータの，内部表現への変換は，あらかじめ記憶装置に格納されている，編集プログラムとよばれる，特別な翻訳用のプログラムによって達成される．このプログラムを，**コンパイラ**(compiler)とよんでいる．コンパイラは，使用するコンピュータ固有の言語で書かれ，これを前もって，どのように記述，構成するかによって，このコンピュータの外部で使用できるプログラミング言語の種類，個数が決定され，同時に，固有の命令列への翻訳効率も定まる．コンパイラのないコンピュータでは，プログラムを，コンピュータ固有の命令列で直接入力しなければならない．これは，個々のコンピュータごとに種々変化す

る非常にやっかいな作業である．現在では，そんなことをしないですむように，コンパイラをあらかじめ備えつけたコンピュータが使用されている．ここでもコンパイラの存在は，はじめから仮定して，話を進める．

コンピュータは，制御装置をもっていて，記憶されたプログラム(これは前述したように，機械固有の命令列として記憶装置に格納されている)の各命令を，はじめから順に1つずつ制御装置にとりだして，その命令の意味を解読し，その内容を演算装置などを使用して実行する．制御装置には，とりだされた命令を一時収納し，解読の用意をするレジスタ(置数装置)，次の命令をとりだすために，とりだす場所の番地を一時記憶するレジスタ(これは，普通，命令カウンタとよばれている)，およびその他の(使用者にはあまり重要ではない)レジスタがあり，演算装置には，演算用の数個のレジスタがある．命令は，通常，格納されている順にとりだされるが，その順を飛び越して，とりださせるような命令(飛び越し命令とよんでいる)が入っている場合には，それが解読されて，通常の順を飛び越して，指定された場所(その位置を示す数値を記憶装置の番地(アドレス(address))とよんでいる)の命令を，制御装置にとりだすこともある．

このような手順を有限回くり返した後，プログラムの実行終了時，あるいは，計算実行途中の予定された時点において，必要な結果が記憶装置に収納される．出力装置を使用して，それらの結果を出力する．このとき，出力された結果が，使用者にとって容易に理解できる形に現れるように，最初から入力プログラムにその様式を指定しておく．……

以上が，コンピュータの主要な動作である．これら一連の動作をふくめて，コンピュータそのものを数学的に定義するには，各装置，各動作ごとに，それがどのように表現されるかを考えてみる必要がある．

(a) 記憶装置上にプログラムやデータを記憶すること　記憶装置は，格納場所の集まりであって，個々の格納場所を，それぞれ区別するために，その位置を示す番地をもっている．したがって，番地の集合と，格納される内容の集合を指定し，何番地に何が入っているかを表現する手段をもてば，上記のことは達成される．それには，番地の集合を$A$，記憶場所に収納することのできる内

容(これを**語**という)の集合を $B$ としたとき，$A$ 上で部分的に定義される関数 $s: A \to B$ を与えれば，その時点における記憶装置の内容が指定されたことになる．ここで，'部分的に定義される' とは，$A$ の部分集合で定義されることをさす．記憶装置の一部分には，その番地に何が入るか，まだ指定されていない部分があるのが普通である．そうでなければ，計算がつづけられない(これを作業用番地の部分とよんでいる)．$s$ が定義されないところがあるというのは，このことに相当する．以後，簡単に，このような関数を部分関数とよぶ．$A$, $B$ は，実際の場合には，いずれも有限集合であるが，話を理論的に進める便宜のため，ここでは無限集合も許すことにしよう．

$A, B$ を固定するとき，$A$ から $B$ への部分関数の全体を記号で，$\mathbf{Pfn}(A, B)$ と表す($\mathbf{Pfn}$ は **p**artial **fn** の略)．

$$\mathbf{Pfn}(A, B) \equiv \{s | s : A \to B(\text{部分的に定義される})\}$$

部分関数 $s: A \to B$ の定義域($s$ が定義される $A$ の要素の集合)を $\mathfrak{D}(s)$ で表す．記憶装置の状態は各 $s: A \to B$ で表現されるといったが，この $s$ のとる値は $B$ の，ある部分集合の値に限定されるのが普通である．たとえば，$Z_2$ を $\{0, 1\}$ ととり，$B \equiv Z_2^{36}$(0 または 1 を 36 個ならべた組の全体)のように定めると，$Z_2^{36}$ の，ある特定な部分集合がコンピュータの固有の命令や数値を表現するのに普通使用されている．したがって，$\mathbf{Pfn}(A, B)$ 全体をとらないで，その部分集合 $S \subset \mathbf{Pfn}(A, B)$ をとりあげ，この $S$ の要素 $s$ を**記憶状態**とよぶことにしよう．また，通常は，記憶装置の各記憶場所にのみ番地をつけているが，以後の考察を容易にするため，制御装置や演算装置のレジスタにもそれぞれ特別の番地をつけることにしよう．それには，はじめから，これらの番地を $A$ にふくめて，部分関数 $s: A \to B$ を考えればよい．以後，このようにする．さらに，$A \subset B$ と仮定しよう．この仮定は自然である．番地の表示が $B$ のある語で，それぞれ直接書けることを意味しているからである．

(b) コンピュータ固有の命令　これは，$B$ の特別の形の要素を指定して**命令**とよべばよいから，$B$ の部分集合を指定することによって定まる．ここでは，これを $I$ と書き，$I$ の要素を固有の命令とよぼう($I \subset B$)．

(c) 数値その他の記号の，データとしての表現　これも $B$ の特定の部分

集合$D$を指定し，$D$の要素が**データ**であるとすればよい．この部分集合$D$は$I$と共通部分があってもよい．$B$の要素をデータとみるか，命令と考えるかは，制御装置や演算装置におけるとらえ方いかんにかかっているからである．

（d） コンピュータ固有の命令の外部表現(記号化)　$I$と全単射(1対1，上への(onto)写像)に対応する記号列の集合$X$を指定し，$X$の要素がそれぞれの命令の**外部表現**であるとすればよい．このとき，$c(X)=\{c(x)|x\in X\}=I$なる$X$から$B$への単射 $c:X\to B$ を，命令の外部表現から内部表現への**符号化**(coding)という．$c$は単射であるから，$I$から$X$への単射 $c^{-1}:I\to X$ が定義される．これを，部分関数 $c^{-1}:B\to X$ (ただし，$\mathfrak{D}(c^{-1})=I$)と書き，**記号化**(decoding)という．

（e） 制御装置の命令カウンタ　命令カウンタには，実行する命令の入っている番地が格納されている．これを表現するには，$A$ の中に特別な番地(これを**1**とする)を指定し，$s(\mathbf{1})\in A$ (このとき，$s(\mathbf{1})$ はある番地を意味する)なるような記憶状態 $s$ のみを考えれば，命令カウンタとして動作していることになる．

（f） 各時点における実行命令の指定　記憶状態が $s$ のとき，現在実行しようとする命令は $s(\mathbf{1})\in A, s(s(\mathbf{1}))\in I$ のとき定義されて $c^{-1}(s(s(\mathbf{1})))(\in X)$ で表される．このとき，$S$ から $X$ への部分関数 $\tau:S\to X$ を

$$\tau(s)=c^{-1}(s(s(\mathbf{1})))$$

で定義すると，$\tau$ は記憶状態 $s$ のときの実行命令(の外部表現)をとりだす．

（g） 1つの命令を実行後，次の命令をとりだす準備　それには，記憶状態 $s$ で，1つの命令 ($\tau(s)$ である)を実行したとき，それと同時に，次の命令の入っている番地を命令カウンタにとりだすように，命令カウンタの値を変更すればよい．このことは，命令カウンタの内容のみ一意に変更し，他の番地の内容は変更しない変換 $\nu:S\to S$ を新しく定義すればよい，すなわち，$\nu$ は，$s_1, s_2\in S$ に対して，

$$s_1(\mathbf{1})=s_2(\mathbf{1}) \quad \text{ならば，} \quad \nu(s_1(\mathbf{1}))=\nu(s_2(\mathbf{1}))$$

であり，かつ各 $s\in S$ に対して，

$$\nu(s)|_{A-\{\mathbf{1}\}}=s|_{A-\{\mathbf{1}\}}$$

なるように指定すればよい.

（h）（記号列としてみた）命令の外部表現　外部で使用するアルファベットをはじめに決定し，これを使用して，一定の記号列を構成する手順（**構文則**(syntax)ともいう）を与え，それによって構成された記号列の集合の中から，ある部分集合をとりだし，それを命令の外部表現に使用すればよい．いろいろな仕方があるが，ここでは次のように定めよう：アルファベットを

$$A\cup\{f_1, f_2, \cdots, f_n, [\ ,\ ],\ ;\ , \alpha, \rightarrow\}$$

ととる．ここに，$f_i(i=1,2,\cdots,n)$は $B$ 上の2項演算記号，すなわち，積集合 $B\times B$ から $B$ への部分関数につけられる名前である．$f_i$ の実際の作用（関数）を $\bar{f}_i$ と表そう．セミコロン（；）は記号列間の分離記号，$\alpha$ は後述するような，ある特別な意味をもつ記号である．

このアルファベット上に，集合 $T$ を，次のように帰納的に定義する（$T$ の要素を**項**(term)とよぶ）；

（1）$A$ の各要素は，すべて $T$ の項である，

（2）$t$ が $T$ の項であるならば，記号列 $\alpha[t]$ は $T$ の項である，

（3）$t, t'$ が $T$ の項であるならば，記号列 $f_i[t\ ;\ t']$ は $T$ の項である（$i=1,2,\cdots,n$），

（4）以上の規則で構成されたもののみが $T$ の項である．

$t, t'$ が $T$ の項である記号列 $t\rightarrow t'$ の全体を $\sum$ と書く；

$$\sum=\{\sigma|\sigma=t\rightarrow t',\ t, t'\in T\}$$

このとき，所要の $X$ を，$\sum$ の部分集合で，$I$ と全単射に対応がつくようにとる．$X$の要素が，1つのコンピュータの固有の命令の外部表現と解釈されよう．コンピュータによっては，$X$ に {**stop**} を加えて，$X_1=X\cup\{\textbf{stop}\}$ をあらためて $X$ と考える場合もある．記号列に，それぞれの意味づけを与えよう；まず，$T$ の要素の（$B$ による）**意味づけ**(semantics)は，写像 $v: S\times T\rightarrow B$ を与えることによって定まる．$v(s,t)(s\in S,\ t\in T)$ は記憶状態 $s\in S$ のときの項 $t\in T$ の（$B$ による）意味づけを表すから，この $v(s,t)$ を，記憶状態 $s$ のときの $t$ の値（$\in B$）にとるように，$v$ を定めればよい，ということは，$v(s,t)$ を，$v_s(t)$ と書いて，$v_s: T\rightarrow B$ を考えることが重要であることを示している．

$v_s$ を次のように定めよう:

(1) $v_s(a)=a \quad (a\in A)$,

(2) $v_s(\alpha[t])=s(v_s(t)) \quad (t\in T)$,

(「項 $t$ に対する番地の内容」が，$\alpha[t]$ の意味である)

(3) $v_s(f_i[t\,;t'])=\bar{f}_i(v_s(t),v_s(t')) \quad (i=1,2,\cdots,n\,;\,t,t'\in T)$

$X$ の記号列 $t\to t'$ を $x$ と表すとき，定義から $c(x)\in I(\subset B)$ である．この $x$ の意味を，次項(i)で説明する．

(i) 命令の実行による記憶状態の変換　記憶状態が $s\in S$ のとき，実行命令は $\tau(s)$ である．このとき，$s(s(1))$ が $c(x)(\in I)$ に等しいとき，命令 $x$ (これを $t\to t'$ とする)$\in X$ が実行される．記憶状態 $s$ は，この命令 $x$ により，$t$ の意味する内容 $v_s(t)$ を，$t'$ の意味する番地 $v_s(t')$ に収納し，その他は不変とするような記憶状態 $s'$ に変化する；すなわち，

$$s'(a)=\begin{cases} v_s(t) & (a=v_s(t') \quad \text{かつ} \quad v_s(t')\in A \text{ のとき}) \\ s(a) & (a\neq v_s(t') \text{ のとき}) \end{cases}$$

で定まる $s'\in S$ に移る．組 $\langle s,x\rangle (s\in S, x\in X)$ に，この $s'$ を対応させる部分関数を $\delta: S\times X\to S$ とする．$\delta_x(s)$ を $\delta(s,x)$ で定義して，

$$\delta_x: S\to S$$

が定まる．これは，命令 $x\in X$ による記憶状態の変換を表現する関数である．ただし，$x=$**stop** のときは，任意の $s\in S$ に対して，$\delta_{\mathbf{stop}}(s)$ は定義されないものとする．したがって，$\delta_{\tau(s)}(\nu(s))(s\in S)$ は，現在の記憶状態 $s$ から実行命令 $\tau(s)$ がとりだされ，命令カウンタの値(次の実行番地)のみを変更させた状態 $\nu(s)$ に $\delta_{\tau(s)}$ を実行して得られる，命令 $\tau(s)$ による新しい記憶状態を意味する．

$\Pi(s)\equiv\delta_{\tau(s)}(\nu(s))(s\in S)$ で定まる部分関数 $\Pi: S\to S$ を(記憶状態の)**制御** (control)という．あきらかに，$\Pi$ は，普通のコンピュータでの制御装置の作用を表している．

以上のことから，コンピュータは，次のように定義される:

**定義 1.1** $\langle A, B\rangle$ 上の**コンピュータ**とは，5つ組 $C=\langle S, X, \delta, \tau, \nu\rangle$ である．ここに，$A$ は番地とよばれる要素の集合，$B$ は語の集合，$S(\subset \mathbf{Pfn}(A,$