BASICによる構造化プログラミング

大久保　明　伸*

Structured Programing by BASIC

Akinobu Ōkubo

Abstract

In this paper, a method of structured programing by BASIC is described.
This method is suitable for small personal computer system which is not connected DISK or MT.
First, syntax of SBASIC which is a virtual language is defined.
Second, from SBASIC programes, BASIC programes are derived by mean of symple rules.
Finally, this method is adapted for some exaples.

1．ま　え　が　き

本校における基礎情報処理教育はFORTRANによるプログラミング教育が，その中心となっている．プログラムの例題や演習題の内容は，そのほとんどが，FORTRAN文法と数値的な計算が中心となっている．

本年度から，本校電子計算機室に導入されたパーソナルコンピュータを使用し，プログラミングの入門教育として，基礎情報処理教育の時間の一部を使用し，BASICの教育が実施されている．FŌRTRAN文法やBASIC文法の性質上，これらの教育に使用される例題は，数値計算が中心になっている．

ところで，卒業研究や実社会でコンピュータ関係に従事している卒業生の仕事の内容をみると，数値計算ばかりでなく，記号処理やデータ構造等の非数値的なプログラミング技術を必要としている．また，現状のようなマイクロコンピュータの各方面への普及は，ますます非数値的なプログラミング技術を必要とすることが推則される．

本校の学生も，個人的にパーソナルコンピュータを所有し，各種のプログラムを開発している．これ等の学生が希望しているのも非数値的なプログラミングの技法や大きなプログラムの作り方である．

非数値的なプログラムや大きなプログラムをより記述しやすくするための言語は多数発表されているが，これ等の言語の処理系の価格が高いため，ほとんどのパーソナルコンピュータの処理系はBASICとなっている．

プログラムの生産性からみると，BASICはFŌRTRANと同様か，それ以下である．しかし，BASICの急激な普及により，ますますBASIC人口は増え続けるものと思われる．

そこで本稿では，標準的なパーソナルコンピュータのBASICの中に仮想的なステートメントと技法を導入して非数値的なプログラムや大きなプログラムの製作をしやすくするための技法を提案する．

今までBASICでプログラムしにくかった分野の問題を，SBASICで記述することにより，比較的楽にプログラミングができる．

2．パーソナルコンピュータ用BASICの問題点

標準的なBASIC言語の処理系はインタプリタである．他の言語の処理系に比較して小型に作れるし，また文法が簡単なため初心者の入門用言語として適している．このため，ほとんどのパーソナルコンピュータの基本的な付属言語として定着している．また，新しく発表されたパーソナルコンピュータのBASICステートメントの機

*宇部工業高等専門学校　電気工学科

能は年々強力になりつつある．

しかし，アルゴリズムの記述能力の点から見ると，ステートメントの種類と機能が強化された割りには，あまり改良されているとはかぎらない．たとえば，FORTRANにある，サブルーチン副プログラムとメインプログラムの引数の受引しや変数，文番号の独立性に関してはインタプリタ型式の標準の BASIC では失なわれている．また，ALGOL や PASCAL 等の言語の特長であるブロックの概念はほとんどない．このため，大きなプログラムを製作する場合，プログラマは変数や文番号の管理に神経をつかわなければならない．そのためプログラムの分割製作はかなり困難になる．さらに，プログラムを構造化するためのステートメントやデータ構造の定義用ステートメントが少ないので，プログラマはかなり細い点まで考慮して，プログラムを製作しなければならない．

これらの欠点を解決する一方法として，BASIC ステートメントに，仮想的なステートメントを追加し，その記述能力を高めることを提案する．この意味で，拡張された BASIC を SBASIC ということにする．

SBASIC を解釈実行する方法は，(1)インタプリタを作る，(2)プリコンパイラを作る，(3)ハンドプリコンパイルをするか，いずれかである．インタプリタやプリコンパイラの方法は各方面で研究されているが，パーソナルコンピュータには高価な周辺装置を接続しなければならない．もし，十分な周辺装置を持つシステムであれば，PASCAL や ALGOL 等の処理系を用いることにより解決できる．

ここでは，(3)のハンドプリコンパイルの方法を公式化して，BASIC プログラムの構造化を試みることにする．SBASIC の拡張されたステートメントは主として，構造化プログラム言語として有名な PASCAL[(1)] の制御ステートメントとほぼ同じものを使用する．ただし，今回の報告では，データ構造や PASCAL の拡張されたデータ型については省略する．後述するように，BASIC ステートメントをわずかに拡張するだけで，構造化されたプログラム[(2)]の記述ができ，手続の再帰的な呼び出しが可能になる．構造化ステートメントを用いることによりプログラムの記述段階では不要な GOTO 文が，使用されることがなくなる．また，プログラムを，トップダウン的に製作することができる．このように SBASIC を用いて，プログラムを作れば構造化されたプログラムを作ることになる．構造化されたプログラムは，いわゆる流れ図よりも，読みやすいのが特長である．

3．SBASIC から BASIC への変換規則

ここで述べる SBASIC は，構造化あるいは系統的プログラミング用言語として評価の高い PASCAL のステートメントの一部を BASIC のステートメントに加え，アルゴリズムの記述能力を高めようとする仮想の言語である．

プログラマは図1の手順に従って，まず SBASIC によりアルゴリズムをコーディング用紙か，エディタを用いて記述する．SBASIC は構造化のためのステートメントを中心に構成されているので，系統的プログラミングの手法を用いてプログラミングが可能である．いわゆるフローチャートによるプログラミングよりは，プログラムは正確で早く完成するものとおもわれる．もし所有のパーソナルコンピュータに強力なエディタがあれば，さらに効率があがるであろう．次に，SBASIC は仮想の言語であるので，実際に実行しようとする場合には，目的のパーソナルコンピュータの BASIC へプリコンパイルしなければならない．この処理もエデタを用いて行う．プリコンパイルは後述する規則を用いて行う．SBASIC に導入されたステートメントの意味は PASCAL と同じである．以下 SBASIC から BASIC への変換規則を示す．

則を示す．

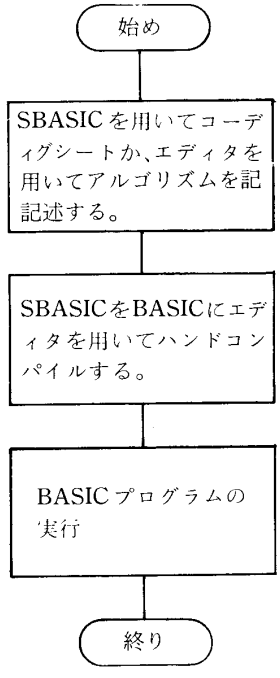


図1 プログラミングの手順

〔規則1〕 変数，定数およびデータの型

SBASIC中で使用した変数，定数およびデータの型はBASICの変数，定数および型に写像する．従ってSBASIC中で使用する変数や定数はBASICの規約を用いると便利である．

〔規則２〕　文番号

SBASICで用いる文番号はエディタを用いるときに必要であるのでBASICと同じ規則とする．

〔規則３〕　*begin*〈文〉*end*

規則（４）以降に示す制御文の実行の範囲を明確に示すためのもので，かならずしもBASICに変換する必要はない．ただし次のようにBASICプログラム中に入れておくとデバッグ時に便利である．

```
10  REM BEGIN
⋮     〈文〉
50  REM END
```

〔規則４〕　*if*〈論理式〉*then*〈文〉

```
10  IF〈論理式〉THEN〈文〉
```

又は

```
10  IF NOT〈論理式〉THEN GOTO50
15  REM BEGIN
⋮     〈文〉
40  REM END
50    〈次の文〉
```

〔規則５〕　*if*〈論理式〉*then*〈文１〉
else〈文２〉

```
10  IF〈論理式〉THEN〈文１〉: GO TO 50
15  REM BEGIN
⋮     〈文２〉
40  REM END
50  〈次の文〉
```

又は

```
10  IF〈論理式〉THEN GO TO 50
15  REM BEGIN
⋮   〈文２〉
30  REM END
45  GO TO 75
50  REM BEGIN
⋮   〈文１〉
70  REM END
75  〈次の文〉
```

〔規則６〕　*case*〈式〉*of*
〈定数１〉:〈文１〉
〈定数２〉:〈文２〉
〈定数３〉:〈文３〉
end

〈式〉の値は整数に写像する．その値は〈定数〉になり，それは文番号になる．

```
10  ON〈式〉GO TO 15, 40, 60
15  REM BEGIN
⋮     〈文１〉
30  REM END
35  GO TO 90
40  REM BEGIN
⋮     〈文２〉
50  REM END
55  GO TO 90
60  REM BEGIN
⋮     〈文３〉
80  REM END
85  GO TO 90
90  〈次の文〉
```

〔規則７〕*while*〈論理式〉*do*〈文〉

```
10  IF NOT〈論理式〉THEN GO TO 60
15  REM BEGIN
⋮     〈文〉
45  REM END
50  GO TO 10
60  〈次の文〉
```

〔規則８〕　*repeat*〈文〉*until*〈論理式〉

```
10  REM BEGIN
⋮     〈文〉
45  REM END
50  IF NOT〈論理式〉THEN GO TO 10
60  〈次の文〉
```

〔規則9〕 *for* 〈変数名〉＝〈式〉 *to* 〈式〉 *do* 〈文〉

```
10  FOR 〈変数名〉＝〈式〉 TO 〈式〉
15  REM BEGIN
⋮     〈文〉
50  REM END
55  NEXT 〈変数名〉
```

もし *to* が *downto* の場合は TO 〈式〉の後に STEP−1 を付ける

〔規則10〕 *procedure* 〈名前〉(〈引数〉, 〈引数〉)
又は *function* 〈名前〉(〈引数〉, 〈引数〉)

```
10  REM PROCEDURE 〈名前〉
20  {メインプログラムの値を引数に渡す}
30  {本体}
40  {メインプログラムに値をもどす}
50  RETURN
```

SBASIC でも BASIC でも変数はすべて大域変数である．従って引数等の管理はプログラマが責任をもつ必要がある．

また *procedure* は再帰的に呼び出されることもあるが，これは次の章で説明する．

4．再帰的な手続き

構造化プログラムの特長の一つはそのプログラム中にほとんど GO TO 文が現れないことである．その原因は SBASIC で示したような制御文があるために本質的に GO TO 文が消去されてしまうためである．GO TO 文はプログラムをもつれた糸のようにしてしまい，プログラミング中の思考やデバッグをかなり困難にしてしまう．結果としてプログラムの生産性は低下する．このくわしい議論は E．W．ダイクストラの「構造化プログラミング」[2] にある．これは「GO TO 文論争」とし有名である．構造化されたプログラミングを作る手法の一つが系統的プログラミング[3]である．さらに，このようにして作られたプログラムの「正当性」を検証する研究が進んでいる．これ等の研究成果から開発されたのが PASCAL である．PASCAL の手続き (PROCEDURE 又は FUNCTION) は再帰的な呼びだしを許るしている．こうすることによりプログラムを見通のよいものにする．いわゆる流れ図よりも，ずっと良い「文書性」を持たせることができる．

そこで SBASIC も形式的に手続きの再帰呼びだしを許すことにする．しかし SBASIC は BASIC に変換されなければならないので，その手順を明らかにしておく．

4．1 再帰的な手続きや関数の例

(1) T(n)＝n＊T(n−1)；ただし T(0)＝1 である．(n≧0) これは階乗関数 T！である．

(2) F(n)＝F (n−1)＋F (n−1)；ただし F(1)＝F(2)＝1 である (n＞0)．これはフイボナッチ数として知られている．

(3) A：：＝x |(B)
B：：＝AC
C：：＝{＋A}
ただし x，(,)，＋は atom である．
これは atom 列 x，(x)，(x＋x)，((x))…を生成する言語の手続きである．

ここでは簡単な例を上げたが，この例を見てわかるように，あるものを定義するのに自分自身もしくは，間接的に自分自身を用いている．再帰が直接的か間接的かはプログラム上では同じ意味をもつ．

再帰的定義の特長は他の定義よりは「静的」である．このことはプログラム中に発生する再帰的手続は静的に記述できることを示している．

記号列の処理や試行錯誤により解決できるようなゲーム等を記述する場合には非常に重要な手法の一つである．

4．2 再帰的手続きのための道具

前節で示したフイボナッチ数の定義を例に必要な再帰のための道具を示す．フイボナッチ数を定義どおりに計算する手順を図にすると図−2のようになる．

F(5)＝F(4)＋F(3)である．しかし，F(4)と F(3)はまだ値が不確定である．そこで F(4)＝F(3)＋F(2)の計算をする必要がある．

F(5)＝F(4)＋F(3)＝{F(3)＋F(2)}＋{F(2)＋F(1)}
＝{{F(2)＋F(1)}＋F(2)}＋{F(2)＋F(1)}
＝5

再帰的手続の特長は，かならず再帰の計算(処理)を制御するための変数が必要である．この変数を再帰制御変数とよぶことにする．F(n)の例では n が再帰制御変数であり計算にも使用されている．プログラミングでは同じ処理を幾度も使用する場合は手続として独立したプログラムにすることが多い．この例の手続名は F でその引数は n である．ところが手続 F の中に手続 F を使用しているので通常のサブルーチンジャンプでは処理すること

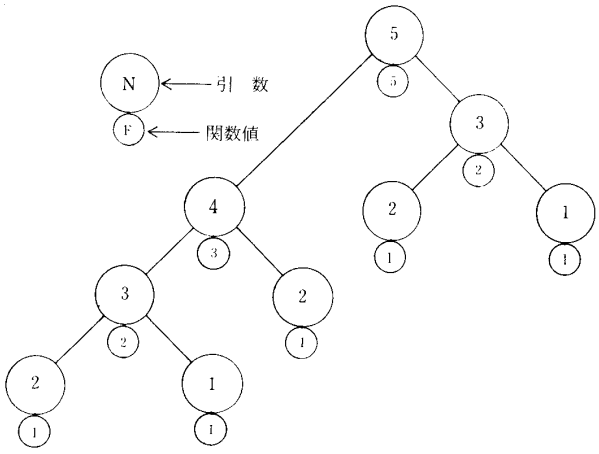


図２　フィボナッチ数Ｆ(5)の計算木

は不可能である．

FORTRAN 等ではある手続の中で自分自身を呼ぶことは禁止されている．ところでF(5)を計算する場合F(4)とF(3)の値が決定されていないので，さきにその計算をする必要がある．しかし，F(4)もF(3)もまだ値が決定されていない．そこで一時，これらを棚上げしておき計算できるところまで進む(図２参照)．一時棚上げしておく情報は自分自身の値がもとまった時点でもどる場所とｎの値である．幸なことに BASIC では，前者の処理はインタプリタが実行してくれるので，プログラマはｎの値を管理しさえすればよい．この棚上げのための記憶場所はスタッフという「棚」を用いればよい．概念図を図３に示す．

PUSH は棚に情報入れることであり，POP は棚から情報をとりだすことである．スタックは最後に入れたものがもっとも早くとりだせる記憶装置である．このような動作をするスタックを BASIC プログラムで実現すれば，次のようになる．

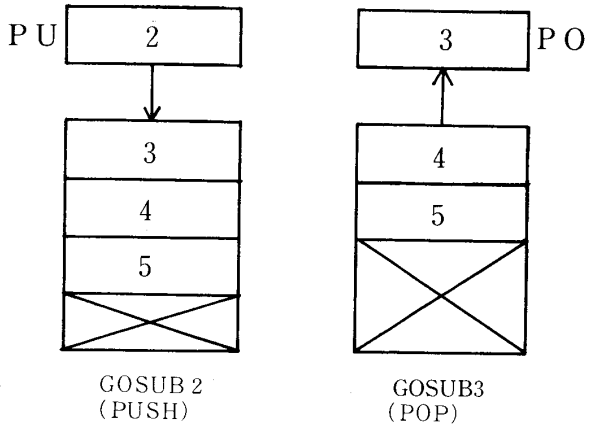


図３　スタックの概念図

```
1  DIM ST (200) :PI=0 : GO TO 1000
2  PI=PI+1 : ST (PI)=PU : RETURN : REM
       PUSH
3  PO=ST (PI ) : PI=PI−1 : RETURN : REM
       POP
```

以上，３行のプログラムでスタックを実現している．この３行の文番号，変数名，配列は予約することにして，他の目的には使用しないことにする．メインプログラムで，変数 PU に目的の値を代入し，文番号２へサブルーチンジャンプ (GO SUB 2) すれば PUSH の動作と等価になる．GO SUB 3 の後に変数 PO を参照すれば POP 動作と等価になる．ただし，スタック ST は数値用であるので，その他の型の場合はその属性をもったスタックを用意する必要がある．

また文番号１の GO TO 1000 はメインプログラムの START 番号としておく．従って手続を前にメインプログラムを後におく．再起プログラムが正しく終了するとスタックは完全に空になる．すなわち，変数 PI は０になる．

４．３　再帰的なプログラムの作り方

３．２で説明した再帰的手続のための道具を用いて，図２を参照して，フイボナッチ数を求めるプログラムを作る．まず SBASIC でアルゴリズムを記述し，そのあと，それを BASIC に翻訳する．SBASIC の手続は再帰を許すものとして次のようにかける．

このプログラム中の手続き *procedure fibo* (*f*, *n*) は再帰的であるので，スタックを用意しなければならない．まずメインプログラム中の変数Ｎを手続変数Ｎに引き渡さなければならないが，BASIC ではすべて全域変数なので，同じ変数名を用いればよい．従って，メインプログラムでは *procedure fibo* を呼び出すときは，GOSUB〈手続き *fibo* の文番号〉とすればよい．

次に，手続Ｆ本体で自分自身を呼びだすときは，まず

(1)引数や局所変数はスタック ST に積む (PUSH)
　すなわち PU＝〈変数〉: GO SUB 2

そして

(2)自分自身へ，サブルーチンジャンプを実行する．
　GOSUB〈手続 *fibo* の文番号〉

(3)手続を実行したあと，スタックから変数をとりだし (POP)，変数をもとの状態にもどしておく．
　GO SUB 3:〈変数名〉＝PO

ただしスタックは FIFO 構造をしているので，変

①

```
10   procedure fibo(f,n)
20   begin
30     if (n=1) or (n=2) then f=1
40         else begin fibo(f1,n-1):fibo(f2,n-2)
50                    f=f1+f2
60                end
1000 rem main
1010 for n=1 to 10 do
1020   begin
1030     fibo(f,n)
1040     print n,f
1050   end
```

プログラム1 －SBASIC

数を複数個スタックしたときは，順序が逆になるので注意がいる．この例は次章でのべる．

また，この例題では，まず F(N－1) を計算したあと F(N－2) の計算を行うので，F(N－1) の計算結果（図2では左の木の計算結果）も，スタックに一時格納しておく．再起の手順 (1)～(3) を BASIC プログラムに付加することにより，次の BASIC 表現を得る．

ここでプログラム1－BASIC のスタックへの PUSH, POP の記述だけを とりだしてみると次のようになる．

```
40  PU=N:GO SUB 2 ←┐
                   ├……fibo の呼び出し
60  GO SUB 3:N=PO ←┘
70  PU=F:GO SUB 2 ←┐
80  PU=N:GO SUB 2 ←┐  ……f の保存
100 GO SUB 3:N=PO ←┘  ……fiboの呼び出し
110 GO SUB 3:F=PO ←┘
```

このように，*PUSH* と *POP* はかならず入れ子になって使用される．もし入れ子の構造にならなければ，記述に誤りがある．ところで，マイクロコンピュータの *BASIC* のサブルーチンス用のリターンスタックは有限であり，*APPLE* では24回のネスティングしか許されていない．この例ではN＝24までしか計算することができない．そこで，システムのリターンスタックをシミュレートすることで，サブルーチンのネスティングを十分深くすることができる．原理は文番号に数値の目印を付けそれをたどることにより，*RETURN* と同じ動作をすることができる．

```
1    DIM ST(200):PI=0:GOTO 1000
2  PI=PI+1:ST(PI)=PU:RETURN
3  PO=ST(PI):PI=PI-1:RETURN
10 REM PROCEDURE FIBO(F,N)
20 REM BEGIN
30   IF(N=1) OR (N=2) THEN F=
       1:RETURN
40 PU=N:GOSUB 2
50 N=N-1:GOSUB 10
60   GOSUB 3:N=PO
70 PU=F:GOSUB 2
80 PU=N:GOSUB 2
90 N=N-2:GOSUB 10
100  GOSUB 3:N=PO
110 F2=F:GOSUB 3:F1=PO
120 F=F1+F2
130 RETURN
140   REM END FIBO
1000  REM MAIN
1010  FOR N=1 TO 10
1020  GOSUB 10
1030  PRINT N,F
1040  NEXT N
1050 END
```

プログラム 1－BASIC

```
JRUN
 1          1
 2          1
 3          2
 4          3
 5          5
 6          8
 7          13
 8          21
 9          34
 10         55
```

実行結果

る．

BASIC のプログラム中に次のような *GO SUB* ステートメントがあったとしよう．

1050　*GO SUB 10*

1080　*GO SUB 10*

この文番号1050と1080に数１，２の目印を付ける．そして

1050　*PU=1 : GO SUB 2 : GO TO 10*

1070　*PU=2 : GO SUB 2 : GO TO 10*

と変えればよい．次に文番号10番から始まるサブルーチンの *RETURN* ステートメントは次のように変えればよい．

1070 GO SUB 3 : K=PO : ON　K　GO TO 1050,

このようにすれば，サブルーチンのネスティングはスタック *ST* の大きさまで許されることになる．また，*FOR—NEXT* 文のネスティングも制限されているものが多いが，これは *IF* 文と *GO TO* 文により，簡単にこの制限を，とりのぞくことができる．付録に，*SBASIC* を用いて記述したプログラムを示す．

5　む　す　び

多くの高級言語が開発されたとは言え，パーソナルコンピュータのユーザの使用言語は，ほとんど *BASIC* である．*BASIC* は，*FORTRAN* や *COBOL* と同じように，広く使用されることになろう．一度，ある種の言語を学ぶと，プログラムの手法までもがその言語の影響を受ける．ここで述べた，ハンドコンパイル技法は簡単なものであるが，通常の *BASIC* や *FORTRAN* と違った発想を生みだすはずである．また，系統的なプログラミング手法により，構造化されたプログラムの開発ができる．再帰的技法は一時，実行の効率の悪さから使用しない立場の人がいたが，現在の新しい言語は再帰呼び出しを許すものが多い．特にコンパイラや *OS* の記述にはなくてはならない技法であるし，人口知能等の研究で開発されるプログラムでは本質的に再帰的にならざるをえないものが多い．以上，プログラム本体の記述を中心に述べたが，データ構造に関する技法は述べていない．データ構造も，*SBASIC* と重要な関係があるので，次の機会にこれに関する手法を発表する予定である．

最後に，*APPLE* の使用に関し，御便宜いただいた本校松井助教授に感謝する．

参　考　文　献

1) N. *Wirth et al.:PASCAL User Manual and Report, Springer- Verlag* (1974)

2) E.W. *Dijkstre et al.:Structured Programming, Academic Press* (1972)

3) 系統的プログラミング/入門, N. *Wirth*, 野下浩平他訳

4) N. *Wirth: Algorisms + Data Structines = Programes, Prentice-Hall* (1976)

〔付録１〕

ここで示す例題は，*Niklaus Wirth* が *PASCAL* により記述したものを *SBASIC* に翻訳したものである．

「最初のＮ個の素数を計算せよ」

〔*SBASIC*〕

```
10   dim p(100) ,v(10)
20   input n
30   p(1)=2:print "2": x=1:li=1:sq=4
40   for i=2 to n do
50     begin
60       repeat x=x+2
70         if sq<=x then
80             begin
90             v(lm)=sq:li=li+1:sq=p(li)*p(li)
100            end
110      k=2:pr=1
120      while (pr=1) and (k <li) do
130          begin if v(k) <x then
140            v(k)=v(k)+p(k)
150            if x< >v(k) then pr=1 else pr=0
160            k=k+1
170          end
180    until pr=1
190    p(i)=x:print x;' ';
200  end
210  end
```

プログラム２－SBASIC

〔*BASIC*〕

```
10    DIM P(100) ,V(10)
20    INPUT N
30  P(1)=2: PRINT "2": X=1:LI=1:SQ=4
40    FOR I=2 TO N
50    REM BEGIN
55  X=X+2
```

```
60    IF NOT (SQ<=X) THEN GOTO 80
70  V(LI)=SQ:LI=LI+1:SQ=P(LI)*P(LI)
80  K=2:PR=1
90    IF NOT (PR AND (K<LI)) THEN GOTO 150
100   IF NOT (V(K)<X) THEN GOTO 110
105  V(K)=V(K)+P(K)
110  PR=(X<  >V(K))
130  K=K+1
140    GOTO 90
150    IF NOT PR THEN GOTO 55
160  P(I)=X; PRINT X;" ";
170    NEXT I
180    END
```

プログラム2－BASIC

```
RUN
?50
 2   3   5   7   11   13   17   19   23   29   31
 37  41  43  47  53   59   61   67   71   73   79
 83  89  97  101 103  107  109  113  127
 131 137 139 149 151  157  163  167  173
 167 173 179 181 191  193  197  199  211
 223 227 229
```

実行結果

〔付録2〕

この例題は，各種ソーティングの中でもっとも早いアルゴリズムである．ソーティング本体は手続き*SORT*(*L*, *M*) である．ソートされる配列は配列Aである．この列では100個の整数乱数を，配列Aに発生させ，小さい順に整理するプログラムである．このソーティング法はクイックソートとして知られている．

手続き*SORT*の中で，自分自身を使用している(*SBASIC*プログラムの文番号120, 130)ので再帰的手続である．クイックソートのアルゴリズムは本質的に再帰的であるので，再帰的手続が記述できない言語では，プログラムが困難である．このプログラムを*SBASIC* と *BASIC* で示す．

```
10   procedure sort(l,m)
20   begin
30     i=l:j=m:x=a(int((l+r)/2))
40     repeat
50       while a(i)<x do i=i+1
60       while x<a(j) do j=j-1
70       if i<=j then
80         begin w=a(i):a(i)=a(j):a(j)=w
90           i=i+1:j=j-1
100        end
110  untile i>j
120  if l<j then sort(l,i)
130  if i<r then sort(i,r)
140end
1000  rem main
1010  input k:dim a(k)
1020  for i=1 to k:a(i)=int(100*red(1)): print
      a(i): :" ": :next
1030  print:print:print:print
1040  sort(1,k)
1050  for i=1 to k:print a(k):" ": :next i
1060  end
```

プログラム3－SBASIC

```
1    DIM ST(300):PI=O: GOTO 1000
2  PI=PI+1:ST(PI)=PU: RETURN
3  PO=ST(PI):PI=PI—1:RETURN
10    REN PROCEDURE SORT(L,M):
20    REM BEGIN
30  I=L:J=R:X=A (INT((L+R)/2))
40    REM REPEAT
50    IF A(I) <X THEN I=I+1:GOTO 50
60    IF X <A(J) THEN J=J-1:GOTO 60
70    IF I<=J THEN W=A(I):A(I)=A(J):A(J)=W:
      I=I+1:J=J-1
80    IF I<=J THEN GOTO 40
90    IF L<J THEN PU=L: GOSUB 2:PU=R:
      GOSUB 2:L=L:R=J: GOSUB 10: GOSUB 3:
      R=PO: GOSUB 3:L=PO
100   IF I<R THEN PU=L: GOSUB 2:PU=R:
      GOSUB 2:L=I:R=R: GOSUB 10: GOSUB 3:R=
      PO:GOSUB 3:L=PO
110    RETURN
1000    INPUT K: DIM A(K): FOR I= 1 TO K:A(I)=
INT (100*RND(1)): PRINT A(I);" "; :NEXT
1005    PRINT:PRINT:PRINT:PRINT
```

```
1010  L=1:R=K: GOSUB 10
1020  FOR I=1 TO K: PRINT A(I);" "; : NEXT I
```

プログラム3 －BASIC

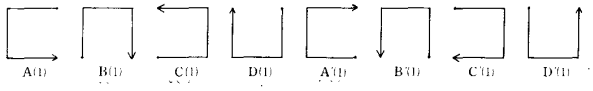
```
RUN
?50
 53   45   33   5   98   99   46   39   50   67   94
 72   82   14   3   65   32   7   74   37   82   59
 11   43   7   80   21   31   69   18   27   68   32
 74   63   55   38   36   22   25   88   63   39   74
 47   67   10   57   27   36

 3   5   7   7   10   11   14   18   21   22   25
 27   27   31   32   32   33   36   3   37   38   39
 39   43   45   46   47   50   53   55   57   59   63
 63   65   67   67   68   69   7   74   74   74   80
 82   82   88   94   98   99
```

実行結果

〔付録3〕

もう一つの再帰手続の例として，有名な *Hilbert* 曲線の例をあげる．プログラムの処理結果は図形である．図形は次のようにして構成されている．図4は，ある正方形の平面を4つの同じ正方形に分割し，各々の正方形の中心を直線で接続する．次に図5に示すようにさらに各々の正方形の中で同様に各要素を直線で結ぶ．このようにしてh(3)，h(4)，h(5)を次々と作図することができる．これを3次，4次，5次のヒルベルト曲線という．これは，数列2，4，8，16があたえられたとき，100項目は 2^{100} と推定することと同じである．そこでh(1)とh(2)の関係からh(n)を推定してみる．h(1)には次のように接続との順序と通過する面の順序に対して次の8種類がある．



$h(1)$＝A(1)である．

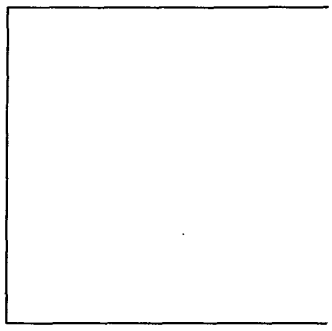


図4　1次のヒルベルト曲線

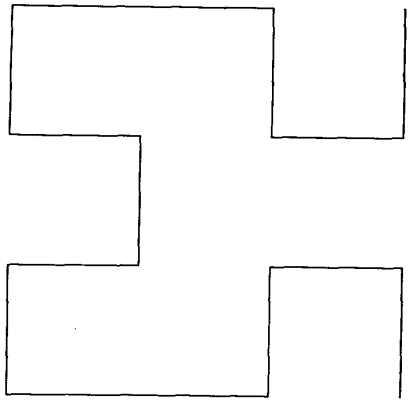
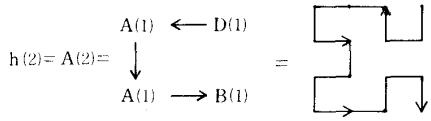
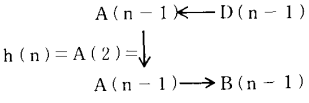


図5　2次のヒルベルト曲線

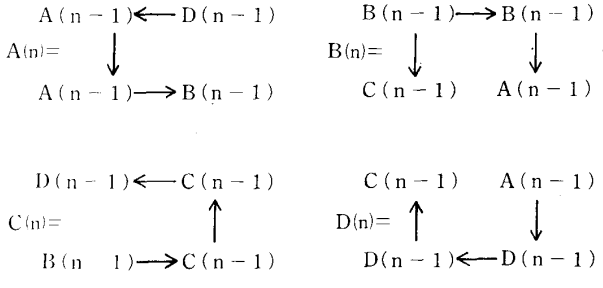
次にh(2)は次のように構成されている．



従ってh(n)は次のように推定できる．



ただし，A(0)，B(0)，C(0)，D(0)の曲線は空とする．また，h(n)＝A(n)であるから，A(n)の曲線をプロットすればn次の曲線ができる．以上の定義のしかたをB(n)，C(n)，D(n)にあてはめると



手続A(n), B(n), C(n), D(n)は一つ下のn－1次の自分自身を直接，間接に使用している．間接再帰の場合は，プログラマが *SBASIC* から *BASIC* に変換するときに注意しなければならないが，*BASIC* の再帰手順は直接再起と同じである．*SBASIC*, *BASIC* によるプログラム，結果を次に示す．*APPLE* 以外のコンピュータに使用するには，*PLOT*, *STARTPLOT*, *SETPLOT* を作り直せばよい．

(昭和57年9月16日受理)

```
10  procedure plot
20    begin
30    moveto(x,y)
40    end

50  procedure startplot
60    begin
70      initturtle
80      pencolor(none)
90      moveto(o,o)
100   end

110 procedure setplot
120   begin
130     pencolor(none)
140     moveto(x,y)
150     pencolor(white)
160   end

170 procedure a(i)
180   begin if i>o then
190     begin d(i-1):x=x-h:plot
200       a(i-1):y=y-h:plot
210       a(i-1):x=x+h:plot
220       b(i-1)
230     end
240   end

250 procedure b(1)
260   begin if i>o then
270     begin c(i-1):y=y+h:plot
280       b(i-1):x:=x+h:plot
290       b(i-1):y=y-h:plot
300       a(i-1)
310     end
320   end

330 procedure c(i)
340   begin if i>o then
350     begin b(i-1):x=x+h:plot
360       c(I-1):y=y+h:plot
370       c(i-1):x=x-h:plot
380       d(i-1)

390     end
400   end

410 procedure d(i)
420   begin if i>o then
430     begin a(i-1):y=y-h:plot
440       d(i-1):x=x-h:plot
450       d(i-1):y=y+h:plot
460       c(i-1)
470     end
480   end

1000  rem main
1010  n=5:  ho=2600:  i=o:h=ho:  xo=int(h/2):
      yo=xo
1020  startplot
1030  repeat
1040    i=i+1:  h=int(h/2)
1050    xo=xo+int(h/2):  yo=yo+int(h/2)
1060    a(i)
1070  until i=n
1080  end
```

プログラム4 －SBASIC

```
1    DIM ST (200) : PI=0 : GOTO 1000
2  PI=PI+1 : ST (PI)=PU: RETURN
3  PO=ST (PI) : PI=PI-1 : RETURN
10   REM PROCEDURE A (I)
30   IF I>0 THEN GOTO 50
40   RETURN
50 PU=I : GOSUB 2
60 I=I-1
70   GOSUB 400 : GOSUB 3 : I=PO : X=
```

```
      X−H : GOSUB 6000
75  PU=I : GOSUB 2 : I=I−1 : GOSUB
      10 : GOSUB 3 :I=PO : Y=Y−H
      : GOSUB 6000
80  PU=I : GOSUB 2 : I=I−1 : GOSUB
      10 : GOSUB 3 : I=PO : X=X+H
      : GOSUB 6000
90  PU=I : GOSUB 2 : I=I−1 : GOSUB
      100 : GOSUB 3 : I=PO
95    RETURN
100   REM PROCEDURE B (I)
120   IF I>0 THEN GOTO 140
130 RETURN
140 PU=I : GOSUB 2 : I=I−1 : GOSUB
      200 : GOSUB 3 : I=PO : Y=Y+
      H : GOSUB 6000
150 PU=I : GOSUB 2 : I=I−1 : GOSUB
      100 : GOSUB 3 : I=PO : X=X+
      H : GOSUB 6000
160 PU=I :GOSUB 2 : I=I−1 : GOSUB
      100 : GOSUB 3 : I=PO : Y=Y−
      H : GOSUB 6000
170 PU=I : GOSUB 2 : I=I−1 : GOSUB
      10 : GOSUB 3 : I=PO
180   RETURN
200   REM PROCEDURE C (I)
220   IF I>0 THEN GOTO 240
230   RETURN
240 PU=I : GOSUB 2 : I=I−1 : GOSUB
      100 : GOSUB 3 : I=PO : X=X+
      H : GOSUB 6000
250 PU=I : GOSUB 2 : I=I−1 : GOSUB
      200 : GOSUB 3 : I=PO : Y=Y+
      H : GOSUB 6000
260 PU=I : GOSUB 2 : I=I−1 : GOSUB
      200 : GOSUB 3 : I=PO : X=X−
      H : GOSUB 6000
270 PU=I : GOSUB 2 : I=I−1 : GOSUB
      400 : GOSUB 3 : I=PO
280   RETURN
400   REM PROCEDURE D (I)
420   IF I>0 THEN GOTO 440
430   RETURN
440 PU=I : GOSUB 2 : I=I−1 : GOSUB
      10 : GOSUB 3 : I=PO : Y=Y−H
      : GOSUB 6000
450 PU=I : GOSUB 2 : I=I−1 : GOSUB
      400 : GOSUB 3 : I=PO : X=X−
      H : GOSUB 6000
460 PU=I : GOSUB 2 : I=I−1 : GOSUB
      400 : GOSUB 3 : I=PO : Y=Y+
      H : GOSUB 6000
470 PU=I : GOSUB 2 : I=I−1 : GOSUB
      200 : GOSUB 3 : I=PO
480   RETURN
1000  N=4 : H0=128: I=0 : H=H0 :
      X0=INT (H/2) : Y0=X0
1005    GOSUB 5000
1010    REM REPEAT
1020  I=I+1 : H=INT (H/2)
1030  X0=X0+INT (H/2) : Y0=
      Y0+INT (H/2)
1040  X=X0 : Y=Y0 : GOSUB 7000
1050    GOSUB 10
1060    IF I<N THEN GOTO 1010
1070    END
5000    REM STARTPLOT
5010    HGR2 : HCOLOR=3 : RETURN
6000    REM PLOT
6010    HPLOT TO X, Y: RETURN
7000    REM SET PLOT
7010    HPLOT X0, Y0
7020    RETURN
```

プログラム５—BASIC

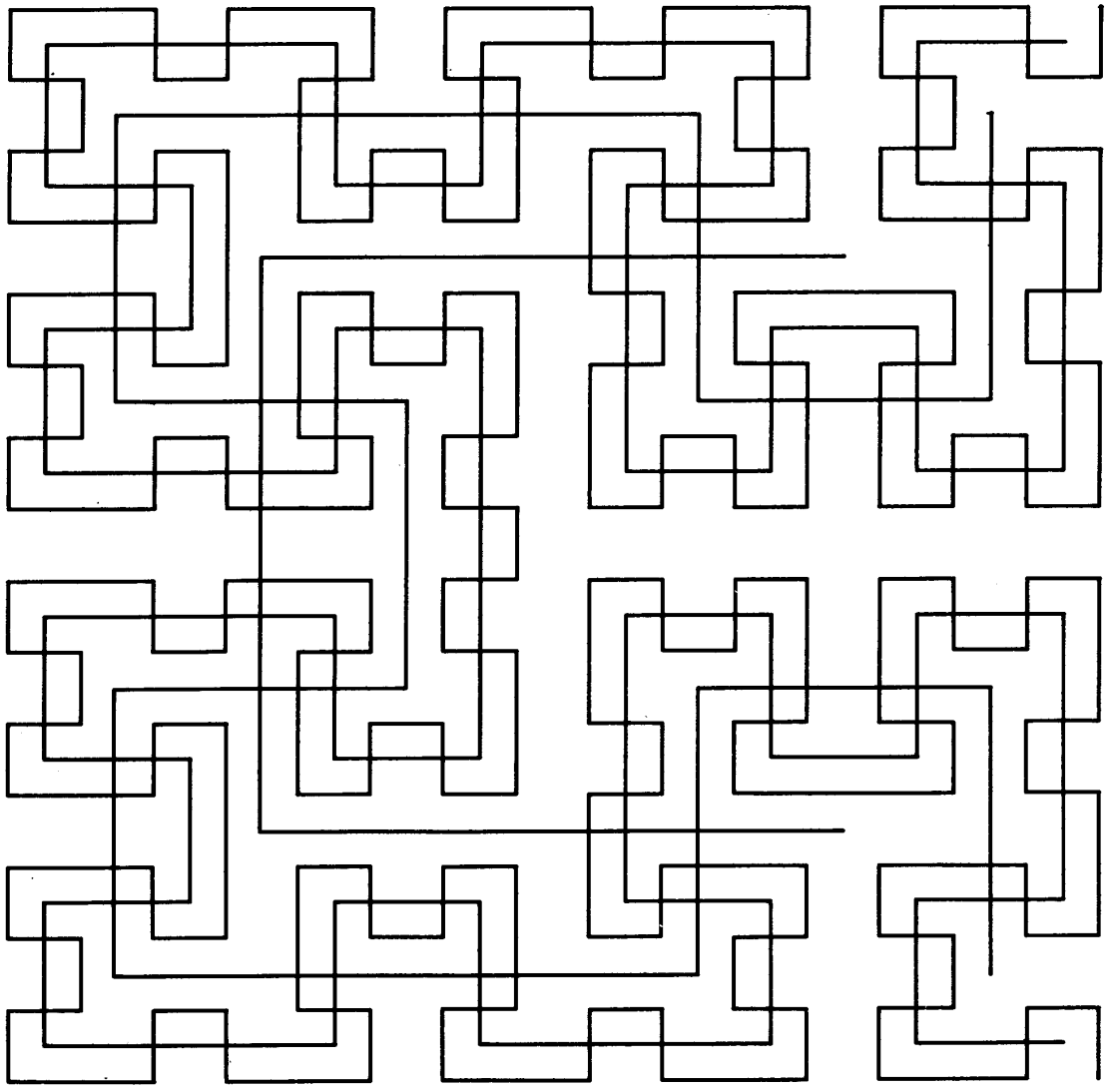


図6 実行結果 h(1)、h(2)、h(3)、h(4)

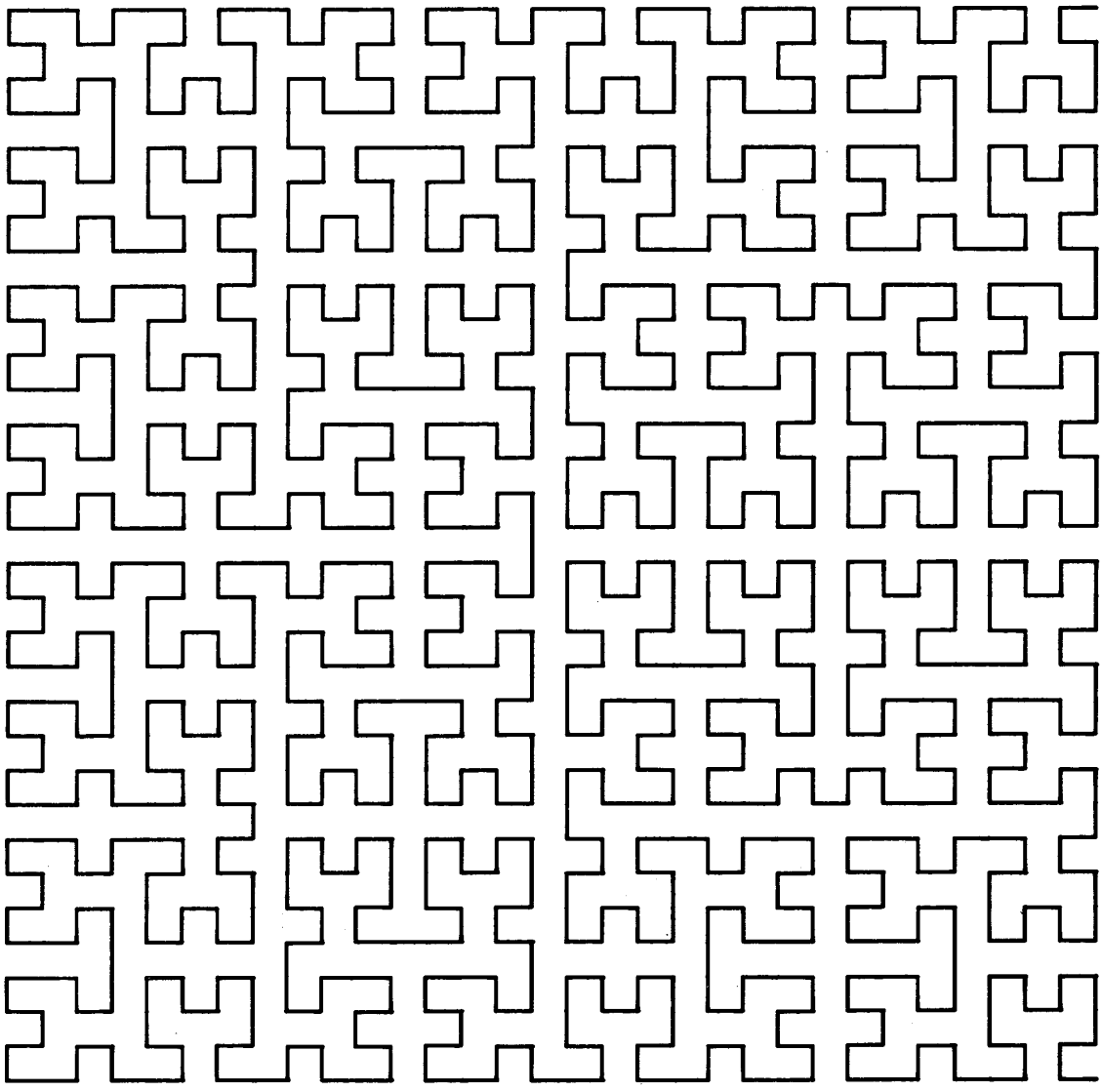


図７　h(5) のヒルベルト曲線