

BASIC解説書(A)

■目 次

まえがき

1 フローチャートとデシジョンテーブル	1
1. 1 フローチャートについて	1
1. 2 デシジョンテーブル	2

2 BASICのコマンドとステートメント

2. 1 コマンド	7
2. 2 コマンドの使い方	9
2. 2. 1 RUN	9
2. 2. 2 LIST	10
2. 2. 3 LLIST	10
2. 2. 4 AUTO	10
2. 2. 5 NEW	11
2. 2. 6 MOUNT	11
2. 2. 7 SAVE	11
2. 2. 8 FILES	12
2. 2. 9 LFILES	12
2. 2. 10 LOAD	12
2. 2. 11 REMOVE	13
2. 2. 12 RUN<ファイル名>	13
2. 2. 13 RENUM	13
2. 2. 14 NAME	14
2. 2. 15 KILL	14
2. 2. 16 DELETE	14
2. 2. 17 MERGE	15
2. 2. 18 CONT	15
2. 2. 19 CLOAD	15
2. 2. 20 CSAVE	16

本ノートは、慶應義塾大学経営管理研究科助教授柳原一夫が作成。編集したものである。BASICコマンド

およびステートメントは、PC9801F用のBASIC REFERENCE MANUAL(PC-9801 F-RM)を参照した。

「1982年3月作成」

2. 3 BASIC プログラムの作成と修正法	16
2. 3. 1 直接(ダイレクト)モードと間接モード	16
2. 3. 2 行と行番号	16
2. 3. 3 プログラムの修正	17
2. 3. 4 プログラムの実行順序	17
2. 3. 5 変数と定数	18
2. 4 BASIC の基本ステートメント	19
2. 4. 1 REM(注釈文)	19
2. 4. 2 入出力文	19
2. 4. 3 算術式と代入文	21
2. 4. 4 GOTO 文	22
2. 4. 5 IF 文	23
2. 4. 6 READ 文と DATA 文(初期値入力文)	25
2. 4. 7 END 文	25
2. 4. 8 FOR NEXT 文(繰返しのためのステートメント)	26
2. 4. 9 RESTORE 文	29
2. 5 基本ステートメントを使ったプログラム例	29
3 BASIC の関数	37
3. 1 数値関数	37
3. 1. 1 亂数	37
3. 1. 2 三角関数	37
3. 1. 3 指数関数	38
3. 1. 4 型変換に関する関数	39
3. 1. 5 符号に関する関数	40
3. 2 文字を扱う関数	40
3. 2. 1 年月日と時刻の表示	40
3. 2. 2 文字処理に関する関数	41
3. 3 その他の関数	44
4 附 錄	47
4. 1 キーボード	47
4. 2 キャラクター・コード表	48
4. 3 エラーメッセージ	49
4. 4 予約語	50
4. 5 漢字の使い方	51

は じ め に

コンピュータは既に多くの企業で広範に利用されているが、最近は大型、中型、小型のコンピュータ以外に、パーソナル・コンピュータの急速な普及傾向が見られる。

このパーソナル・コンピュータの普及はビジネスに従来とは若干異なる影響を与えるものと期待されている。その理由はパーソナル・コンピュータの持つ特性が大量のデータ処理、高速の演算処理に適すると言うよりはむしろ個人の好きなように自由にシステムを開発し、それをいつでも好きな時に利用できるというところに本来の良さがあるからである。すなわち、個人の能力を一層引き立てるツールとしてはパーソナル・コンピュータは大型コンピュータよりも優れた点が多いといえる。マネジメントの能力は本来個性的なものであるとするならば、パーソナル・コンピュータはマネジメント・サポートの強力な武器となってくれる筈である。

大型からパーソナルまでコンピュータの潜在的利用者層は拡大しているが、ビジネスにおいて、こうした利用層にコンピュータが有効に活用されているかといえば、実際は必ずしもそうではない。その理由はソフトウェア開発にある。コンピュータのハードウェアは大変な進歩を遂げたが、それを動かすプログラムがなければ、コンピュータはただの箱である。ところがそのソフトウェア開発が大変なネックになっているのである。その理由は第一にソフトウェアの専門家の不足である。こうした専門家は大型コンピュータの効率をより高め、経済性を追及するために時間を裂かれて、個別の細かいソフトウェア開発に手が回らないのが現状である。一方、市販のパッケージ・ソフトは大分充実して来たとはいえ、当然のことながら、標準的な汎用パッケージに限られ、個々の事情を反映した形のソフトウェアは提供してはくれない。ユーザはシステムに合わせて仕事をすることになる。

この事情は経営の意思決定を助ける情報システムを開発する場合には一層深刻な問題となる。すなわちマネジメント個人をサポートする情報システムは個別的であり、標準化、汎用化ができるにくいものである。その上、システムを個別的に開発しようとしても、そこで要求される能力はソフトウェアの高度なテクニックよりはマネジメントの知識や技量である。こうした分野のシステム開発は従ってソフト専門家にとって必ずしも得意とするものではない。ましてや市販のパッケージがマネジメントが望む細かな注文を満たしてくれることは少ないであろう。

外部のシステム・ハウスを活用する方策も有力であるが、開発の対象となる経営そのものを熟知していることが開発に最も重要な条件であるとすれば、この方策にもそれなりの限界があると思われる。

マネジメントの意思決定をサポートするシステムは意思決定者が自ら開発するのが最も望ましい姿である。しかしながら、これまでのマネジメントにとってコンピュータは比較的新しいツールであり、プログラミングに未経験だったため、ソフトウェアを自己開発することはな

かなか困難であった。しかし、これから若いマネジメントやスタッフにとって、システム開発の能力を身につけることは今述べた理由から極めて大事である。

システムを自己開発するメリットとして一般に次のような点が挙げられる。

まず第1に、既に述べてきたように、ニーズに合わせたシステムが開発できる。第2に、システムのメインテナンスが保証される。特に、意思決定は企業環境の変化に迅速に対応しなければならないが、そのことはそれをサポートするシステムも迅速に修正、改善されなければならぬことを意味している。第3に、修正、改善が迅速に行なえる場合には、初期のシステム開発が比較的短期間に、容易に行なえるという利点がある。このことはシステム開発を簡単なものからより複雑で有効なものへと段階的に進めることにも通じる。そして最後に、社内にシステム開発のできるマネジメント、あるいはスタッフを養成できる。このような人材が社内に確保されるようになれば、外部から購入したシステムを手直ししてつかうこともできるので、外部の開発スタッフを有効に活用することもできるようになる。

本解説書は初めてコンピュータ・プログラムを学習するビジネスマンのために作成されたBASIC(ベーシック)言語の解説書であり、最も基本的なものを中心に記述している。より実用的なプログラミングについては、解説書(B)を参照して頂きたい。機種としてはパーソナル・コンピュータのNEC-PC9800を前提としているが、他の機種ではキーボードやBASICステートメントが若干異なるのでその点注意が必要である。

第1章ではフローチャートとデシジョン・テーブルについて説明する。コンピュータで処理すべき仕事について、処理のタイプや手順、分岐とその条件などを視覚的なチャートで表示することはプログラム作成の上で手助けとなる。また、デシジョン・テーブルは複雑な条件によって、処理の内容が異なるときに、これらを論理的に整理し、理解するための手法である。

第2章ではBASIC(特に、N88-BASIC(86))についての解説である。プログラミングの学習ではコンピュータを使ってプログラムを作り、実行してみることがなによりも大切である。もし、プログラムに過ちがあれば、コンピュータがブザーで知らせてくるので、簡単に修正して完全なものに仕上げる事ができる。プログラミング学習での最良の先生はコンピュータであることを覚えておいて頂きたい。

なお、BASICは1963年ごろダートマス大学のケメニー教授とクルツ教授の二人により開発された言語である。Beginner's All Purpose Symbolic Instruction Codeの略語と言われており、もともとはタイムシェアリング・システム(TSS)に適した言語として利用されてきたが、現在ではパソコンの言語として大いに普及している。N88-BASICは米国マイクロソフト社のBASICに基づき、PC8800およびPC-9800シリーズ用にバージョンされたものである。

1 フロー チャートとデシジョンテーブル

1-1 フロー チャートについて

普段、私達がやっている仕事を注意してみると、仕事は幾つかの作業に分割できることが判る。更にそれらの作業には一定の順序があって、これを間違えると、とんでもない事になってしまう事が多い。

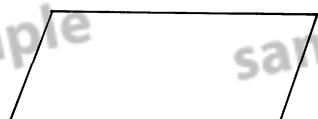
慣れた仕事は無意識のうちにやっているが、いざ他人に教えようと思うと、説明が体系的でなかったり、文章では判りにくい不便もある。新しい仕事では失敗なくやろうと思っても、考えることが断片的で全体が見えないという経験を持つ人も少なくない。更に、条件が異なれば、すべき作業も違うのが普通である。条件が複雑になれば、ますます頭が混乱してしまうことになる。そこで、この様な作業間の流れ（プロセスの流れ）を体系的に図表で示す技法が考え出された。このようなものの1つに、フロー チャートがある。フロー チャートは、プロセスを幾つかに細分化し、これを手順の流れに従って、記号で表現するもので、処理手順の明確化と論理性の検討に有効である。

フロー チャートのこの様な特徴は、電子計算機のプログラム作成（プログラミング）にも利用されている。

<フロー チャートの書き方>

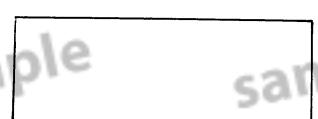
記号の説明

(1) 入出力記号



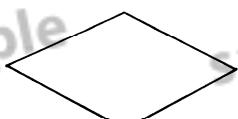
フロー チャートを作成するには、何らかの目的があるわけで、その目的を達成するのに必要な情報、又はデータを前もって規定してやらなければならない。また同様に、目的とする結果を表示する必要もある。この機能を持つものが入出力記号である。記号の中には必要な内容を文章で記述する。

(2) 処理記号



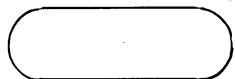
入力されたデータは、何らかの処理、又は加工（計算）される。処理記号はその内容を表示するために使われる。記号の中には、具体的な処理の内容を、文章、又は数式で表現する。

(3) 判断記号



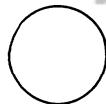
プロセスによっては、条件によってフローがジャンプしたり、分岐したりすることがある。この場合には、この判断記号が用いられる。記号の中には、文または数式で判断の基準が示されなければならない。

(4) 端子記号



これは、フローの始めと終りを示す記号である。記号の中には、始めまたは終り、と書かれる。

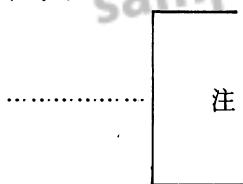
(5) 結合記号



記号間のフローが錯綜して見にくい場合、フローを分断して表示するために、この記号が使われる。本来つながっているフローを、この記号を使って分断した場合には、分断したフローの端にこの記号を書き（少なくとも 2 つの分

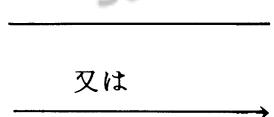
断点の端ができる）、記号の中には、同じ英数字を書く。これによって、同じ英数字を持つ結合記号が実はつながっていることを表わす。

(6) 注釈記号



記号の中に十分な記述がスペース上書けない時は、注釈記号を用いる。

(7) 流れ線



流れ線は原則として上から下へのフローを示す。しかし、分岐する時は必ずしもこの限りではない。判りにくい時は矢印をつける。

1-2 デシジョン・テーブル

条件によって、るべき作業が異なる場合、これが簡単なうちは直感で扱えるが、複雑な条件がつけられるような場合には、デシジョン・テーブルを利用すると便利である。デシジョン・テーブルでは、プロセスを条件とアクションに分け、その関係を図表で示すものである。

[例題 1] A と B の 2 人がジャンケンを繰り返している。開始後、最初に 3 回勝った人の名前を示せ。

ジャンケンについては誰でもそのルールについて知っているので、例題の文章を読んで直感でその内容を十分理解できると思うが、以下にデシジョン・テーブルとフローチャートを用いて、この例題に示されるプロセスを記述してみる。

デシジョン・テーブルでは、ジャンケンのルールを示すものと、勝ちが 3 回になったかどうかを示すものを別々のテーブルに分けて書いてある。

なお、テーブルの中に示した記号 Y は YES, N は NO, を示し、×印はそれに対応する条件の組合せのとき、そのアクションを取ることを示している。

—デシジョン・テーブル—

(1) ジャンケンの勝敗を示すデシジョン・テーブル

		ジャ ン ケ ン (1)	1	2	3	4	5	6	7	8	9		
条 件	A	が	グ	ー	Y	N	N	Y	N	N	Y	N	N
	A	が	チ	ヨ	K	N	Y	N	N	Y	N	N	Y
	A	が	パ	ー	N	N	Y	N	N	Y	N	N	Y
	B	が	グ	ー	Y	Y	Y	N	N	N	N	N	N
	B	が	チ	ヨ	K	N	N	N	Y	Y	Y	N	N
	B	が	パ	ー	N	N	N	N	N	N	Y	Y	Y
	A に勝点 1 を加える				X	X					X		
	B に勝点 1 を加える			X					X	X			
	引き分け		X				X				X		

A と B がジャンケンして、A の出した手と B の手の全ての組合せは 9 組であり、これが全て表に示されている。

これらの組合せの結果が A の勝、B の勝、引き分けのいずれになるかを X 印を使って示してある。

(2) 3 点先取による勝者を決めるテーブル

		勝 点 3 先 取	1	2	3	4	5	6	7	8	9
条 件	A に得点があったか	Y	N	Y	N						
	A の得点が 3 になったか	N	N	Y	N						
	B に得点があったか	N	Y	N	Y						
	B の得点が 3 になったか	N	N	N	Y						
	A の 勝				X						
	B の 勝					X					
	もう一度ジャンケンする		X	X							
	ゲーム終了				X	X					

第 2 のテーブルでは A か B に得点のあった時だけ、これまでの累積点が 3 になったかどうかをチェックする。

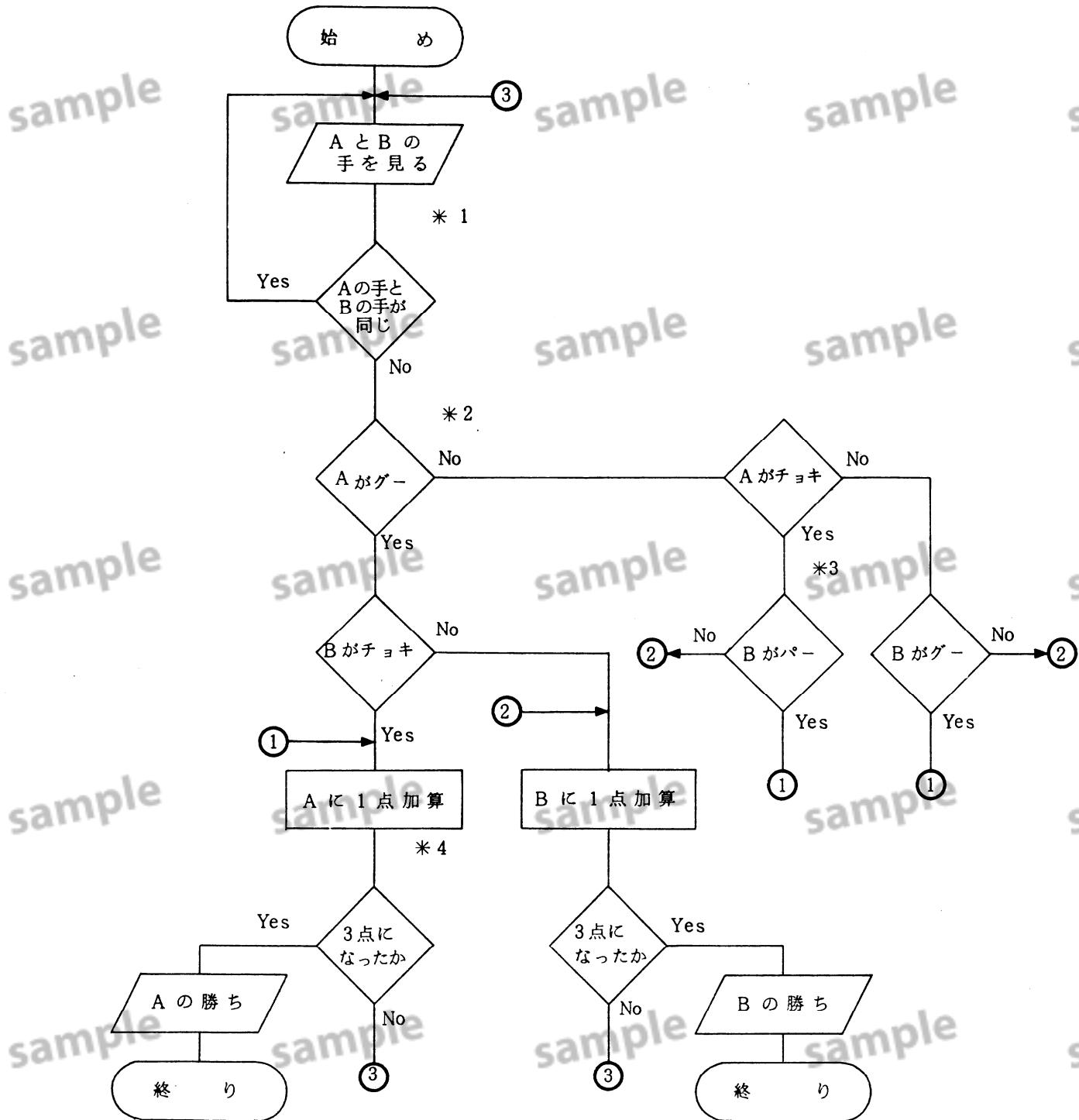
3 になれば、その時得点のあった人の勝ちとなり、その事を表示する。

そしてゲームは終了する。

例題についてのフロー・チャート

論理をデシジョン・テーブルで確かめながら前記のフローチャート記号を用いて例題のプロ

セスを示すと以下のようになる。



<説 明>

- * 1] 第 1 の デシジョンテーブルによれば、A と B が同じ手の場合はジャンケンしなおしである。従って以下の判断をする必要がない。すなわち、A, B の 勝負に 関する 判断は 不要。
- 2] ここのフローでは“あいこ”はないから必ず勝負がつく。そこで A が グー を 出したかどうかをチェックする。YES の 場合は 下に NO の 場合は 右へ 分岐する。
- * 3] A が チョキ を 出して、B が パー の 時は A の 得点すなわち 結合記号 ① を 通して A に 1 の 得

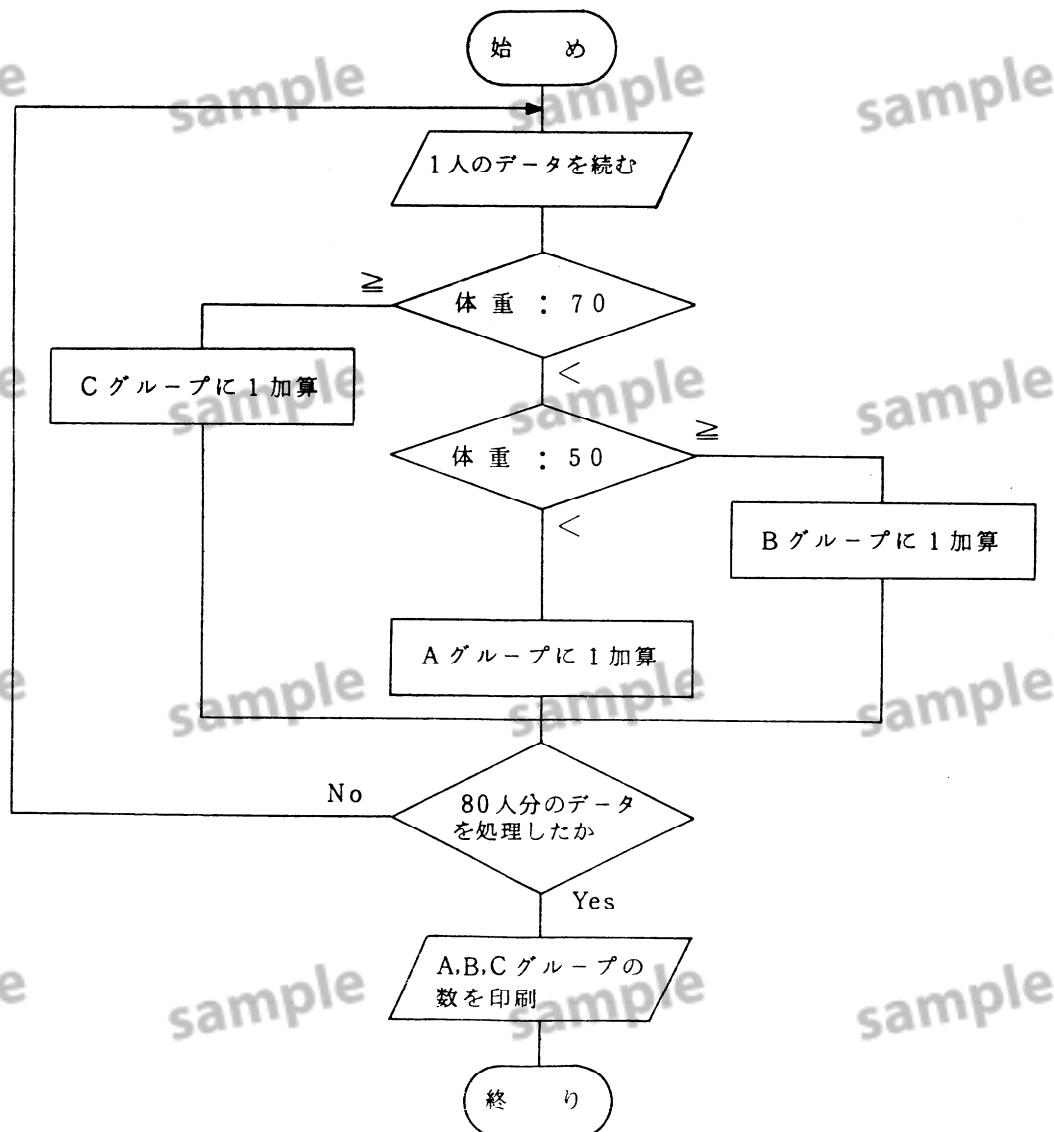
点が加算される。

なお A がチョキでなければ、あとは自動的にパーである。

- * 4) 第 2 のデシジョン・テーブルに従って、得点のあった時だけ 3 点になったかどうかをチェックし、3 点の時は勝ち、そうでない時はジャンケンを繰り返す。いづれかが勝てば、それを表示してゲームは終りになる。

以上見てきたようにフローチャートおよびデシジョン・テーブルはコンピュータプログラムに直接関係するものではなくとも、会社における事務手続きや日常生活での行動を記述できる。地震が起きた時、どのような行動をすべきかフローチャートで書き表わしておけば、いざというときに大いに役立つかも知れない。

[例題 2] 80人の体重を A グループ (50kg 以下), B グループ (50~70kg), C グループ (70kg 以上) に分類し、各グループの人数を求めたい。これを行なうための作業をフローチャートで示せ。



フローチャートは簡単なプログラムの作成には余り役に立たないかも知れない。フローチャートを書くよりも直接プログラムを書く方がむしろ容易であるともいえる。フローチャートが有益になるのは複雑なプログラムの作成時である。デシジョン・テーブルにも同様のことがいえる。フローチャートやデシジョン・テーブルは他人にプログラムを説明する際には非常に有用であり、従ってプログラマーがプログラムを作成する際にはドキュメンテーション（文書化）としてフローチャートを添付するのが普通である。

〔練習問題〕

3個の数A, B, Cを読んで小さい順に並べて書く作業を、フローチャートで示せ。

2 BASICのコマンドとステートメント

コンピュータの演算処理装置に命令を与えて、思い通りの演算を行なわせるのがプログラムの役割である。プログラムは人間にも機械にも理解できる言語で記述されなければならないが、実際コンピュータの演算処理装置が理解できるのは2進数で表わされた機械語だけである。しかし、人間が機械語でプログラムを作成するのは容易な事ではなく、一部の専門家だけが機械の構造を良く知った上で、特殊な用途のために機械語でプログラミングをする程度である。人間にとて、もっと容易にプログラムを作成できるよう工夫が重ねられた結果としてFORTRAN, COBOL, BASIC, PL/1等々の言語が開発されてきた。人間寄りのこうした言語を使ってどのようにして、機械語しか理解できないコンピュータに命令を伝えるかといえば、こうした言語で書かれたプログラム（ソース・プログラム）をあらかじめコンピュータに覚えさせておいた処理方法に従って、コンピュータ自身にプログラムから機械語への変換をまずさせて、その出来上った機械語（オブジェクト・プログラム）を用いて、再びコンピュータに演算を命令するという手続きをとることにしたのである。

人間寄りの言語で書かれたプログラムを機械語に変換するのに2つの方法が考えられている。その1つはインタープリターと呼ばれる方式でソース・プログラムを実行していく過程で遂時機械語に変換し、処理するもので、幾分外国語の通訳に類似している。BASICはこのインターパリター方式で実行される言語である。もう1つの方式はコンパイラと呼ばれるもので、プログラムの実行以前に全てのソース・プログラムをオブジェクト・プログラムに変換して、それから実行を行なうものである。通訳に対して、この方式は翻訳に類似している。FORTRANやCOBAL, PL/1などの言語はコンパイラ言語である。コンパイラ言語は何度も繰り返して実行するプログラムでは、処理時間が短かく効率が良い。

しかしながら、コンパイルには多くのメモリーが必要であり（ソース・プログラムとオブジェクト・プログラムの両方を記憶する必要がある）、パーソナル・コンピュータ向きとはいえない。しかしながら最近はメモリーが大規模化し、かつ安価になってきたので、パーソナル・コンピュータでも、コンパイラBASICが使えるようなものが出現している。

2-1 コマンド

```
list  
90 'EX-1  
100 INPUT A,B  
110 S=A*B  
120 PRINT S  
130 END  
Ok  
run  
? 12,15  
180  
Ok
```

これは、BASIC言語で書かれたプログラムの例である。INPUT, PRINT, ENDなどはプログラムがコンピュータに与える命令であり、ステートメント（文ともいう）と呼ばれる。

プログラムがコンピュータに命令を与え計算結果を出力するためには、プログラムが実行されなければならない。これにはキーボードからRUNと入力してやる必要がある。このRUNという命令によって、コンピュータははじめてプログラムの命令すなわちステートメントに従って処理を行なうのである。このRUNと云う命令はコマンドと呼ばれ、ステートメントとは区別される。LISTもまたコマンドである。コンピュータは、人間がプログラムを容易にかつ効率的に作成し、エラーを見出し、修正し、実行し、そして必要ならばプログラムを保存できるような機能を附加して作られている。こうした機能を果す役割を負っているのがオペレーティング・システム(OS)と呼ばれるソフトウェアである。コマンドとは実はOSに人間が命令を与えるための言葉である。OSがコンピュータ資源の利用をプログラムに与えたとき(この状態をプログラム・レベルと呼ぶ)はじめて、ステートメントによる命令が実行されるのである。従って、コンピュータに電源を入れると、まずははじめにOSの世話をになって、プログラム実行の準備をすることになる。(コマンドの入力待ちになっている状態をコマンド・レベルと呼ぶ)。

PC 9801 システムでは、コマンド待ちの状態にあるとき、通常画面にOKと表示される。

以下に示すのは、コマンドの一覧表であるが、最初のうちは特に重要なものだけ覚え、次第に利用の範囲を広げていくのが得策である。従って、この表では最も重要と思われるものから順に列挙してある。なお幾つかのコマンドはそれが頻繁に使われるという理由から、ファンクション・キーに登録されているので、簡単に利用できる。ファンクション・キーをf・n(ただし nは番号)とする。その内容は画面に表示されるので容易に判る。既に、例からも判るようにN88-BASICではコマンドを大文字でかいても小文字でかいてもかまわない。これはステートメントの場合も同様である。

コマンドの一覧表

1. RUN プログラムを実行する。 f・5
2. LIST プログラムのリストをスクリーンに表示する。 f・4
3. LLIST プログラムのリストをプリンターに表示する。
4. AUTO 自動的に行番号を発生する。 f・2
5. NEW 古いプログラムを消して、新たなプログラムの入力を待つ。

6. SAVE	プログラムをディスクに保存する（セーブする）。	f · 6
7. FILES	ディスク・ファイルのリストを表示する。	
8. LFILES	ディスク・ファイルのリストをプリンタに出力する。	
9. LOAD	ディスクに保存されているプログラムを本体に読み込む。（ロードする）	f · 1
10. RENUM	行番号をつけかえる。	
11. NAME	ディスクファイルのファイル名をつけかえる。	
12. KILL	ディスクからファイルを削除する。	
13. DELETE	行を削除する。	
14. MERGE	2つのプログラムを結合する。	
15. CONT	中断したプログラムの実行を再開する。	f · 10
16. EDIT	プログラムの画面修正を行なう。	f · 9

2-2 コマンドの使い方

コマンドを入力するには、あらかじめ定められた書式で入力する必要がある。書式については以下の約束で表示することにする。なお、コマンド入力後は必ず CR（キャリッジ・リターン）キーを押す必要がある。

書式の記法

1. アルファベットの大文字で記された項目はそのままの形で入力する。

この場合、シフト・モードで大文字にして入力する必要はなく小文字のまま入力してかまわない。ただし、引用符で囲まれた文字列のなかで使う場合（ファイル・ネーム等）には大文字と小文字は区別して入力しなければならない。

2. かぎカッコ<>で囲まれた項目は、ユーザーが指定する項目である。

角カッコ〔〕で囲まれた項目はオプションである。

省略した場合、必要に応じデフォルト値または以前に指定した値が適用される。デフォルト値とは BASIC があらかじめ定めた値である。

4. かぎカッコ及び角カッコ以外のコンマ、丸カッコ、セミコロン、ハイフン、等号などの記号は示された位置に正しく入力しなければならない。

5. 省略記号（…）の続く項目は、一行の許す長さ内で任意の回数繰り返すことができる。

2-2-1 RUN

書式：RUN [<行番号>]

目的：メモリにあるプログラムの実行を開始する。

解説：<行番号>を指定すると、その行から実行がはじまる。指定のない場合には、最

も若い行番号の行から実行がはじまる。プログラムの実行が終るとコマンド・レベルに戻る。

例：RUN

RUN 100

2-2-2 LIST

書式：LIST [<行番号>] [- [<行番号>]]

目的：メモリ内にあるプログラムの全部または一部をディスプレイにリストする。

説：LIST コマンドを実行し終わると、コマンド・レベルに戻る。

<行番号>が省略された場合は、最も若い行番号のプログラム行よりリストが始まる。（リストはプログラムの終り、またはストップキーの入力により終る。）

最初の行番号のみが指定された場合には、その指定された行だけがリストされる。

最初の行番号とそれに続くハイフン（マイナス記号）またはコンマ（,）までが指定された場合は、その行番号に始まり、それよりも大きい行番号の行すべてがリストされる。ハイフンまたは（,）とそれに続く2番目の行番号が指定された場合には、プログラムの始めからその行番号までの行がリストされる。両方の行番号が指定された場合には、その範囲のすべてがリストされる。

例：LIST メモリにあるプログラム全体をリストする。

LIST 500 500 行のみをリストする。

LIST 150- 150 行から最後までの行をリストする。

LIST -1000 最も若い行から 1000 行までをリストする。

LIST 150-1000 150 行から 1000 行までをリストする。（150 行及び 1000 行は含まれる。）

LIST 行番号にピリオド（小数点）を指定すると BASIC 内のポインタが示している現在の行を指す。（例えば、エラー時はエラーを起した行を示す。）

2-2-3 LLIST

書式：LLIST [<行番号>] [- [<行番号>]]

目的：メモリにあるプログラムの全部または一部をプリンタにリストする。

説：LLIST コマンドの実行が終るとコマンドレベルに戻る。

LLIST コマンドの使い方は LIST コマンドと同じである。

2-2-4 AUTO

書式：AUTO [<行番号>] [, <増分>]

目的：キャリッジリターンの入力ごとに行番号を自動的に発生する。

説：AUTO は<行番号>に始まり、引き続く各行に<増分>ずつ増加させた行番号を付ける。どちらの数のデフォルト値も 10 である。<行番号>に引き続くコンマがあって、増分が指定されない場合には最後の AUTO コマンドの増分が適用される。AUTO は STOP キーを押すことにより終わる。STOP キーをタイプした行は格納

されない。STOPキー入力後はコマンドレベルに戻る。

例：AUTO 100, 50 100, 150, 200…のように行番号を発生する。

AUTO 10, 20, 30, …のように行番号を発生する。

2-2-5 NEW

書式：NEW

目的：メモリにあるプログラムを抹消し、すべての変数をクリアする。

解説：NEWコマンドは、コマンドレベルにあるとき新しいプログラムを入力する前に実行する。NEWコマンドの実行が終わると、コマンドレベルに戻る。

2-2-6 SAVE(ディスク)

書式：SAVE<ファイル名> [, A]

目的：<ファイル名>は引用符に囲まれた、ファイル名としての要件を満たした文字列である。もし<ファイル名>のファイルが既に存在している場合には、新しいファイルによって更新されることになる。（古いファイルは失なわれる。）

アスキーコードでファイルをセーブする場合はAオプションを指定する必要がある。指定のない場合にはファイルはバイナリ形式に圧縮してセーブする。アスキー形式でセーブするとより大きいディスクスペースを必要とする。アスキー形式でセーブする必要が起きるのは、例えば、MERGEコマンドを使用する場合である。MERGEコマンドはアスキー形式のファイルを必要とする。

キー形式のファイルを必要とする。

例：SAVE "COM2", A

SAVE "PROG1.N98"

SAVE "PROG"

2-2-7 FILES(ディスク)

書式：FILES [<ドライブ番号>]

目的：ディスクに入っているファイルの名前と大きさを表示する。

解説：FILESコマンドを実行すると、ディスクにはいっているファイルの名前とその大きさがクラスター単位で表示される。クラスターとはファイルが使用するディスクの最小単位で通常は8セクターである。OPEN文またはアスキーセーブで作られたファイル名がそのファイルタイプとの間を空白で区切ってリストされる。バイナリーセーブによって作られたバイナリファイルは、ファイルタイプとの間をピリオドで区切ってリストされる。

2-2-8 LFILES(ディスク)

書式：LFILES [<ドライブ番号>]

目的：プリンタに現在のディスクに納められているファイルの名前と大きさをプリントする。

説：出力がプリンタに行くことを除くと FILESと同じである。

2-2-9 LOAD(ディスク)

書式：LOAD<ファイル名> [, R]

目的：ディスクからプログラムをメモリーにロードする。

説：<ファイル名>は SAVE コマンドによりファイルをセーブした時に使用した名前である。

LOAD コマンドは、指定されたプログラムをロードする前に開いているファイルをすべて閉じ、メモリにあるすべての変数とプログラムを抹消する。しかし “R” オプションを LOAD コマンドと共に使用すると、それまで開いていたデータファイルはそのまま開かれたままで指定したプログラムはロードされた後自動的に実行がはじまる。したがって、 “R” オプションを伴なった LOAD コマンドは、使えるメモリサイズに比べてプログラムが大きすぎる場合、いくつかのプログラム（又は、同じプログラムのいくつかのセグメント）を連結して実行するのに使うことができる。プログラム間の情報は、これらのデータファイルを通じて渡すことができる。

例：LOAD "PG101.N98"

LOAD "TEST"

LOAD "STRTRK", R

2-2-10 RENUM

書式：RENUM [<新番号>] [, [<旧番号>] [, <増分>]]]

目的：プログラムの行番号をつけなおす。

説：<新番号>は、新しくつける行番号の最初の番号で、デフォルト値は 10 である。

<旧番号>は番号のつけ替えをはじめる現在のプログラムの行番号である。デフォルト値はプログラムの最初の番号である。<増分>は新しく付ける各行番号のあいだの増分で、デフォルト値は 10 である。

RENUM コマンドは、また、GOTO, GOSUB, IF~THEN~ELSE, ON~GOTO, ON~GOSUB 及び ERL 文などで参照している行番号に対応して変更する。これらの文が参照している行番号の行がもし存在しない場合には、"Undefined line xxxxx in yyyy" とエラーメッセージがプリントされる。この場合、誤った行番号 (xxxx) は RENUM コマンドによって変更されないが、行番号 yyyy は変更

される。

注 意：RENUMコマンドをプログラム行の順序を変えるのに使うことはできない。（たとえば、10, 20, 30の3つの行がある場合のRENUM 15, 30など），また65529以上の行番号を発生することもできない。このような場合には，“Undefind line call”エラーが起る。

例：RENUM プログラム全体の行番号を付け直す。新しい最初の行番号は10で，10ごとに増加して付けられる。

RENUM 300, 50 プログラム全体の行番号を付け直す。新しい最初の行番号は300で，50ごとに増加する。

RENUM 1000, 900, 20 900行からのプログラムの行の行番号を，1000から始まり20ずつ増える行番号に付け替える。

2-2-11 NAME

書 式：NAME <古いファイル名> AS <新しいファイル名>

目 的：ディスク・ファイルの名前を変える。

解 説：<古いファイル名>はディスク上に存在し，<新しいファイル名>は存在してはならない。そうでない場合にはエラーが発生する。NAMEコマンドの実行後は，そのファイルは同じディスクの同じ場所に新しい名前で存在する。

NAMEコマンドの実行が終わると，N-BASICはいつでもコマンドレベルに戻る。

例：NAME “ACCTS” AS “LEDGER”

OK

例：200 KILL “DATA1”

2-2-12 KILL

書 式：KILL <ファイル名>

目 的：ディスクからファイルを削除する。

解 説：もし現在オープン状態にあるファイルに対してKILL文を実行すると，“File already open”（ファイルは既に開かれている）エラーが起る。

KILL文はすべての型のディスク・ファイル，つまりプログラム・ファイル，ランダムデータ・ファイル，そしてシーケンシャルデータ・ファイルに有効である。

例：200 KILL “DATA1”

2-2-13 DELETE

書 式：DELETE [<行番号>] [-<行番号>]

目 的：プログラムの行を抹消する。

解 説：DELETEが実行されると，BASICはいつでもコマンドレベルに戻る。もし<行番号>がないか，あるいは該当する行がなければ“Illegal function call”エラ

ーが起こる。

- 例：DELETE 40 40行を抹消する。
DELETE 40-100 40, 及び100行を含み, その間のすべての行を抹消する。
DELETE-40 40行を含み, 40行までのすべての行を抹消する。

2-2-14 MERGE

- 書式：MERGE<ファイル名>
目的：メモリにあるプログラムに指定したディスクファイルのプログラムが混ぜられる。
説：<ファイル名>はそのファイルがSAVEコマンドによってセーブされたときの名前である。ただしファイルはアスキー形式でセーブされなければならない。（もしうでない場合には，“Bad file mode”エラーが起る）もしディスクファイルのプログラムがメモリにあるプログラムと同じ行番号を持っている場合には、ディスクのファイルの行が対応するメモリのプログラム行に置きかわる。（“混ぜる”とはディスクのプログラムをメモリにあるプログラムに“挿入する”と考えることもできる。）
MERGEコマンドの実行が終わると、コマンドレベルに戻る。
戻る。

2-2-15 CONT

- 書式：CONT
目的：STOP入力後、またはSTOPまたはENDステートメント実行後にプログラムの実行を再開する。
説：ブレークが発生した時点から実行が継続する。INPUTステートメントからのプロンプト出力後にブレークが起きた場合は、プロンプト（？または文）を再びプリントするところから実行を再開する。
CONTは通常デバッグのためにSTOP文と共に用いられる。プログラムの実行が停止したら、ダイレクトモードステートメントにより中間結果を調べたり変更したりすることができる。そして、CONTまたはGOTOによりプログラムの再開をすることができる。GOTOを使った場合は、実行再開するプログラムの行番号を指定することができる。
実行の中断中にプログラムが変更（編集）された場合は、CONTは無効となる。

2-2-16 EDIT

- 書式：EDIT<行番号>
目的：指定された行を画面上へ表示し、カーソルを移動する。
説：画面上での編集を容易に行なえる。ROLL-UP, ROLL-DOWNキーを使って

行送り、行の逆送りが自由にできる。

例：EDIT 100 100行画面に表示する。ROLL-UPで以後の行を表示できる。

EDIT 行番号にピリオドを指定すると BASIC内のポインタが示している現在の行を指す。

2-3 BASICプログラムの作成と修正法

2-3-1 直接(ダイレクト)モードと間接モード

スクリーンに“OK”とプリントされた状態はコマンド・レベルにあり、コマンドの入力待ちの状態になっている。この時点では BASIC は通常次の 2 つのモードすなわち、直接モードと間接モードのいずれかで利用することができる。間接モードでは次節で示すようにコマンドあるいはステートメントに行番号をつけるのに対し、直接モードではこうした行番号は使わず、その代り入力と同時に実行するという特徴がある。直接モードでは PRINT(3 * 25124 + 1213)
* 312 / 12 といった簡単な計算の答(1991210)を得るような場合には大変便利である。

一方、間接モードはプログラムを入力し、修正し、実行するのに使用され、プログラムの 1 行は行番号ではじまる必要がある。間接モードではプログラムはメモリに記憶され、RUN コマンドで実行される。

2-3-2 行と行番号

間接モードでプログラムが入力されるとき、BASIC の書式は次の通りである。

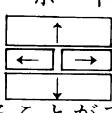
nnnnn BASIC ステートメント [: BASIC ステートメント ……] <CR>ただし、nnnnn は行番号で 0 から最大 65529 までの整数である。

また、CR はキャリッジリターンである。

1 行に複数のステートメントを入れることができるが、その場合、それぞれのステートメントは : (コロン) で区切られていなければならない。BASIC のプログラムは常に行番号ではじまり、キャリッジリターンで終わり、そこに使用される文字数は最大 255 文字までである。

AUTO コマンドを使えば、行番号を自動的に書き出すことができる。

2-3-3 プログラムの修正

キーボードから誤って入力がされてしまった場合には、テンキーボードの左横に配置されている  カーソル移動キーと **INS** **DEL** インサート、デリートキーを使って修正することができる。カーソル移動キーによってカーソルの位置を動かし、誤字の位置に止め、正しい文字を入力すれば新しい文字に置き替る。**DEL** キーを押せば、カーソルの左側の 1 文字が削除され、**INS** (シフトキーを併用) キーを押した後に新しい文字を入力すれば、文字の挿入ができる。挿入を止めたいときはもう一度 **INS** を押すかカーソル移動キーを押す。

既に存在する行番号と同じ行番号を使って、ステートメントを入力すれば、古い行は後で入

力された新しいステートメントに置きかわり、同じ行番号だけを入力すれば、古い行は削除される。なお、修正後は修正行ごとにCRを押す必要がある。さもないと何も修正されない。

長いプログラムの修正には EDIT コマンドが便利である。

2-3-4 プログラム実行順序

一連のステートメントから成るプログラムは原則として、プログラムの上の行から順に下の方へと実行される。

三角形の面積を計算するためのプログラムが次に示されている。

```
90 'EX-2
100 REM SANKAKUKEI NO MENSEKI
110 A=10
120 B=15
130 S=A*B/2
140 PRINT S,A,B
```

この例では三角形の底辺を A (= 10), 高さを B (= 15) とした場合の面積 S を $S = A \times B \div 2$ によって計算し、その結果 S と A, B を並べて一行に印刷する。

このプログラムの処理は行番号 100 番から順に 140 番まで処理される。(RUNと入力することによって、このプログラムは実行される。)

行番号のつけ方は大小関係だけが意味をもち、分岐を指示するステートメントに出会うまでは、行番号の小さい方から順に実行される。従って行番号の開始番号と間隔は定められた範囲内の整数であれば全て自由である。

プログラムを入力する場合、そのプログラムは入力の順序に関係なく自動的に行番号の若い順に編集される。

行番号 100 は REM 文と呼ばれ、行番号の次に書かれた REMあるいはアポストロフィ(‘)によって識別される。(従って、90 も REM 文である)。

REM 文はプログラムの実行に何ら関与せず、プログラム作成者がメモとしてこれを利用する。110 番、120 番は変数 A, 変数 B に値をセットするステートメントである。130 番で S が計算される。

BASICでは乗算記号に * (アステリスク), 除算記号に / (スラッシュ) が用いられる。140 番では変数 S, A, B の値を印刷(出力)する。

この例からも判るように、プログラムでは普通、変数に値を入力するステートメント、計算するステートメント、出力するステートメントが必要である。これらについては後に詳しく説明する。

2-3-5 変数と定数

先の三角形の面積を求めるプログラムを例にとると A, B, S は変数と呼ばれる。この例では変数 A には 10 という定数が代入された。

変数はコンピュータ内の記憶場所を表わしていると考えることができる。すなわち記憶場所

Aには10が、Bには15が、そして記憶場所Sには計算結果の75が記憶される。

定数にはこの他に、3.14のように小数点のもつものや、 2.4×10^{-3} (=2.4×10⁻³)または 6.5×10^5 (=6.5×10⁵)など指数付きのものがある。(EとDの違いについては後述)

BASICが数値を扱うときには、状況に応じて異なる3種の取扱い方をする。(1)もし、定数が8桁かそれ以上の桁数である場合、あるいは指数がD表示になっている場合、または定数の後尾に#が付されている場合、倍精度で記憶する。倍精度のもとでは有効桁数が17桁であるが、記憶に必要なメモリの容量は8バイト必要である。(2)定数が倍精度でなく、またそれが-32768から+32767の範囲外である場合、あるいは小数点が含まれるか、または指数がE表示になっているものは単精度で記憶する。後尾に#が付された定数も単精度扱いとなる。単精度の定数はメモリ4バイトに記憶される。(3)以上の(1)および(2)に該当しない定数は整数として2バイトのメモリに記憶される。なお、バイトとは8ビットで構成される記憶容量であり、その大きさは $2^8 = 256$ つまり整数ならば1から256までの数を記憶できる。

コンピュータの内部では定数のタイプによって異なった記憶の仕方をしているが、通常の場合、これについて特に意識する必要はない。

次に変数についてであるが、BASICでは変数は最初の文字が英字で始まる必要がある。字数は最大40字で、字の種類は英数字およびピリオドであり、その全てを区別する。ただし、FNで始まる変数は許されないし、予約語(付録参照)そのものを変数にすることもできない。

変数の後尾に#記号をつければ倍精度の数値を代入できる変数になる。(例; A#=12345678)また、#に代りに%記号をつければ整数形式の数値を代入するための変数となる。小数点を有する数値を整数形式の変数に代入すると小数点以下が切捨てされる。

例:

```
90 'EX-3
100 A% = 12.34
110 PRINT A%
Ok
run
12
```

また、-32768から32767までの範囲を超える数値を代入するとオーバーフローのメッセージが出される。

例:

```
90 'EX-4
100 A% = 1.23457E+06
110 PRINT A%
Ok
run
Overflow in 100
Ok
```

後尾に#記号をつけるか、あるいは何も記号をつけない場合には、その変数に単精度の数値を代入することができる。特別な計算をしない限りは、通常、単精度の変数で十分である。変

数についても、精度によって使用されるメモリーの容量は異なり、定数の場合と同一である。

以上、定数と変数について説明してきたが、これまで全て数値に関するものに限られていた。

BASICではこれ以外にも文字定数と文字変数が利用できる。文字定数は引用符で囲まれた英数字の列で、文字列の長さは最大255文字までである。文字変数は変数の規則に従がい、後尾に\$記号をつけたものである。以下に文字変数と文字定数の例を示す。

文字定数の例

" K B S "

" H e l l o "

" イ ロ ハ "

" \$ 153,000 "

文字変数の例

A \$

X 3 \$

K B S \$

S S 3 2 \$

BASICは型が異なれば異なる変数として識別するので、A%とAは別の変数として扱かわれる。

2-4 BASICの基本ステートメント

2-4-1 REM

プログラマーが変数の説明や注意書きを自分のため、あるいはユーザのためにプログラム中に書き印すときに、REM文を使うことができる。行番号に続いてREM又は'(アポストロフィ)を書けば注釈行とされる。プログラムの実行には何ら影響を与えない。REM文の後に続いて:(コロン)で区切って、他の文を書くことはできない。

例

```
90 'EX-5
100 REM ***ウリアケ トケイ プログラム ***
110 ' S=ウリアケ タ カ
120 ' Q=1ンバ イリョウ
130 ' P=カカク
```

2-4-2 入出力

フローチャートの□記号に相当するBASIC文はINPUTとPRINT文である。

形式は INPUT<変数名> [, <変数名>……] および PRINT<変数名> [, あるいは; <変数名>……] である。

まずINPUT文はキーボードから変数にデータ値を入力する機能を持っている。この文が実行されるとスクリーン上に?印が表示され、データの入力をうながす。複数の変数にデータを入力する場合、INPUT文の変数は、(コンマ)で区切る必要がある。このINPUT文が実行された場合、キーボードからの入力はデータ間を、(コンマ)で区切る必要がある。

データの個数が変数のそれに満たない場合、あるいは、変数よりもデータの数が多かった場合、または入力データに問題があって、BASICが受け入れられない場合には、Rodo from Startと表示してくるので、当該INPUT文のデータ入力を最初からやりなおす必要がある。

INPUT文の実行は単に?印の表示だけで、ユーザに知らされるので、どの変数への入力なの

か判りにくい。そこでユーザが間違えずにデータ入力できるようにBASICでは次の形式のINPUT文を用意してある。

INPUT "プロンプト文";<変数> [, <変数>……] プロンプト文とは注意書きの事で、ユーザはこれによって、現在どの変数へのデータ入力が行なわれているかが容易に判る。

データをキーインしてCRを押すと、はじめて変数へのデータ入力が完了する。

PRINT<変数>は変数の値をスクリーンにアウトプットする。変数は数値型に限らず文字型でも良いし、式でも良い。何も変数を書かずに単にPRINTとすると1行だけ行送りになる。従って、3行空きにしたければ、PRINT:PRINT:PRINTのようにPRINT文を3つ使えば良い。なお、このようにコロン(:)を用いて文をつなぐことをマルチステートメントという。

所で、次の例のように変数間をコンマ(,)で区切る方法とセミコロン(;)で区切る方法ではプリントの状況は多少異なる。

```
list  
90 'EX-6  
100 A=10:B=20:C=30:D=40  
110 PRINT A,B  
120 PRINT:PRINT  
130 PRINT C;D  
Ok  
run  
10          20
```

30 40

すなわち、(コンマ)で区切った場合はデータ間にすき間があるのに対し、;(セミコロン)の場合は前のデータに続けて、後続のデータを出力する。

コンマで区切った場合、1行を14文字づつの領域に分け、1つのデータを1つの領域に出力する事になっている。もし、PRINT文の最後の変数の後ろが(コンマ)または;で終わっている場合には、次のプリント文は同じ行に上で述べたと同様の間隔で出力される。

以上、入出力文の例を示すと次のようである。

```
90 'EX-7  
100 INPUT B0  
110 INPUT "INP B1,B2";B1,B2  
120 INPUT "INP C1$,C2$,C3$";C1$,C2$,C3$  
130 PRINT C1$,B0  
140 PRINT C2$;B1  
150 PRINT C3$;  
160 PRINT B2  
Ok  
run  
? 88  
INP B1,B2? 62,90  
INP C1$,C2$,C3$? B=,W=,H=  
B=          88  
W= 62  
H= 90
```

また、プリンターへ出力したい場合には、PRINT文を単にLPRINTに変えるだけで良い。

プリンターへは1行80文字（紙巾によって異なる）までの出力が可能である。

例

```
90 'EX-8
100 LPRINT"*****"
110 LPRINT"*** セイキウショ ***
120 LPRINT"*****"
130 INPUT"キンカ" K =";K
140 LPRINT"セイキウキンカ" K=";K;" 円"
*****"
*** セイキウショ ***
*****"
セイキウキンカ" K= 300000 円
```

なお、確認であるが、コロンはマルチステートメントに用いられ、セミコロンはPRINT文で変数の区切りに用いられる。両者は全く異なることに注意して頂きたい。

2-4-3 算術式と代入文

BASICで使える演算記号および演算の優先順位は次の通りである。

順位	記号	使用例	演算
(1)	^	X ^ 3	X^3 べき乗計算
(2)	* , /	3 * 2 / 4	*は乗算 /は割算
(3)	+ , -	X + Y - Z	+は加算 -は減算

優先順位の変更のために() (カッコ)が使える。

この他に特殊な演算として、以下の記号が使える。

(4)	¥	17 ¥ 4	整型の除算 (例の結果は4となる)
(5)	MOD	17 MOD 4	剰余の計算 (例の結果は1となる)

¥, MODの処理の優先順位は(2)と(3)の間にに入る。

例として

```
90 'EX-9
100 A=10:B=20:C=30
110 PRINT (A+B)*C/10^2
120 PRINT A+B*3-C-A*2
130 X=((A+B)/(B+C))^.5
140 PRINT X
Ok
run
9
20
.774597
```

代入文は変数と式または変数と変数を= (等号)で結んだものである。この場合、左辺には必ずしも変数がなければならない。すなわち

変数 = 式 または 変数

左辺の変数の型は数値型および文字型のいずれでも良いが、対応する式または変数は左辺と同じ型でなければならない。

数値型変数には整数、単精度および倍精度があるが、左辺と右辺との型の不一致には特別の意味がある。それは、右辺を左辺の型に変換するということである。前にも一寸触れたように $B \% = A$ という式は単精度の変数の値を整数に変換する（その結果、小数点以下切棄てが起きる）ことになる。

代入文における等号はむしろ←（左向矢印）として、右辺の値を左辺の変数に代入すると考える方がより実際的である。

2-4-4 GOTO 文

GOTO文の書式は

GOTO＜行番号＞で、このステートメントを実行すると処理の順序が指定された行番号へジャンプする。

例として

```
90 'EX-10
100 INPUT A
110 S=S+A
120 PRINT S
130 GOTO 100
```

このプログラムは 130 行の GOTO 100 を実行すると再び 100 番に戻り、それから行番号順に処理を続けるため、いつまでも終りのないプログラムとなる。プログラムの実行を停止しコマンドレベルに戻るために STOP キーを押せば良い。

なお、このプログラムの PRINT S によって得られる結果は入力データの累積値となっていことに注目されたい。合計値を計算するときに BASIC で良く使われる手法である。

2-4-5 IF 文

フローチャートの判断記号  に相当する BASIC ステートメントが IF 文である。

IF文の書式：

IF＜論理式＞ THEN ＜文＞ または <行番号> [ELSE <文> または <行番号>]
あるいは

IF <論理式> GOTO <行番号>

論理式が真ならば THEN文またはGOTO文が実行され、偽であれば ELSE文が実行される。
ELSE文がない場合には、次の実行文へプログラムの処理が移る。

論理式に用いられる関係演算子は = (等号), > (より大きい), < (より小さい) の 3種があり、次のような関係表示に用いられる。

演算子

関係(条件)

例

(1) =

等しい

A = B

(2) <>, ><	等しくない	A <> B
(3) >	大きい	A > B
(4) <	小さい	A < B
(5) >=, =>	等しいか又は大きい	A >= B
(6) <=, =<	等しいか又は小さい	A <= B

この例で A および B は式であっても良い。その場合には式の演算を先に行ない、その結果についての関係を表わすものである。

論理式は文字数あるいは文字変数の関係にも使える。例をあげると次の通りである。

```
" AAN " < " AB "
" FILENAME " = " FILE " + " NAME "
" CL" > " CL "
" kg " > " KG "
```

この例からも判るように、アルファベット順に大きくなつて行き、大文字の方が小文字よりも小さい。文字列の長さが異なる場合は、短かい方の列尾に空白を追加して長さをそろえてから、関係を見定めるようになる。アルファベット以外の文字についても大小関係があって、論理式が使える。文字の大小については附録 4-2 のキャラクタ・コード表を参照のこと。

IF 文の例を示せば

```
90 'EX-11
100 INPUT A
110 INPUT B
120 IF A>B THEN PRINT "A>B":GOTO 100
130 IF A=B THEN PRINT "A=B":GOTO 100
140 PRINT "A<B":GOTO 100
```

なお、これは IF 文でマルチステートメントを使った例であるが、THENあるいは ELSE 文の後にマルチステートメントを使えば、それぞれの条件下でマルチステートメントが実行される。すなわち、120 では A>B ならば PRINT "A>B" を実行後、GOTO 100 を実行する。

sample

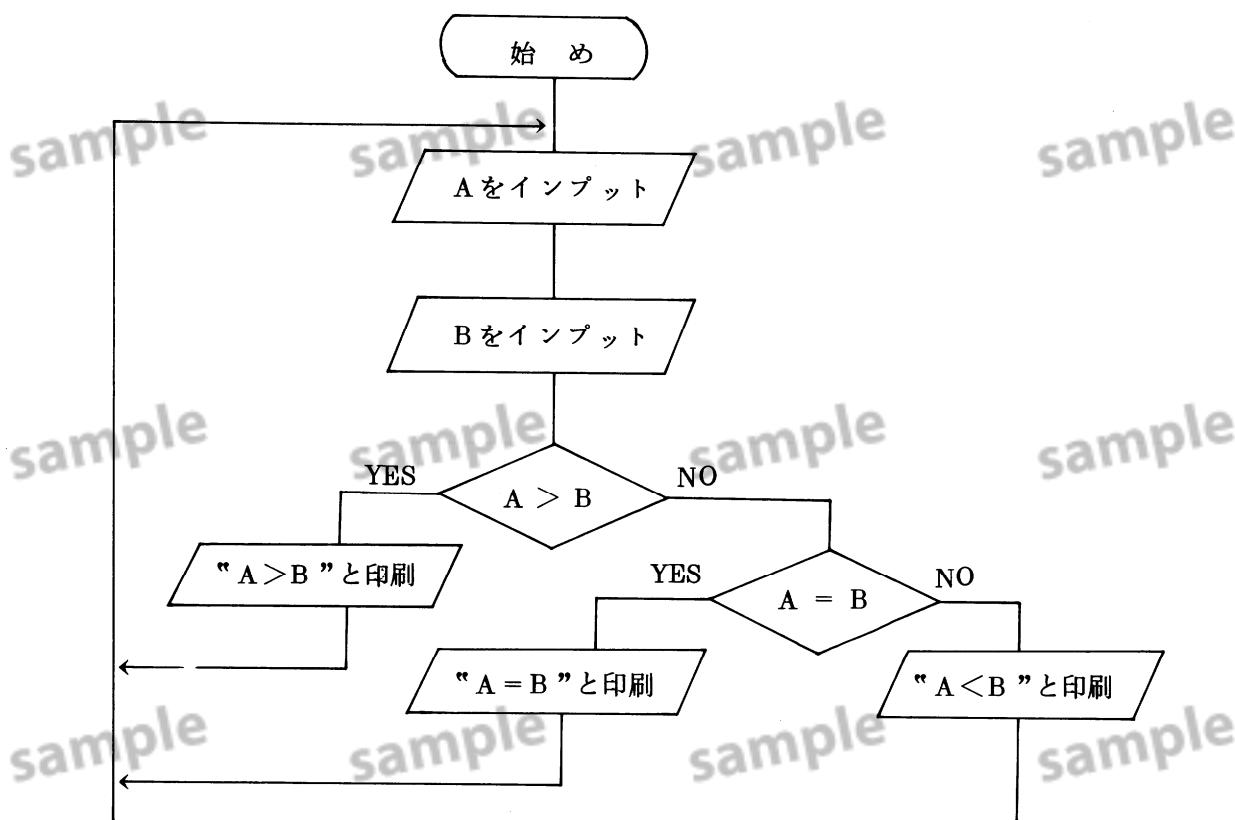
sample

sample

sample

sample

これをフローチャートで示すと次の様になる。



これはまた、次のようにもプログラミングできる。これは IF 文の多重化(ネスティング)の例である。

```
90 'EX-12
100 INPUT A
110 INPUT B
120 IF A>B THEN PRINT "A>B" ELSE IF A=B THEN PRINT "A=B" ELSE PRINT "A<B"
130 GOTO 100
```

Ok

run

```
? 1
? 2
A<B
? 2
? 0.5
A>B
? 85
? 85
A=B
?
```

文字型の変数を含んだ IF 文の例である。

```
9 'EX-13
10 INPUT B$
20 INPUT C$
30 A$="filename"
40 IF A$=B$+C$ THEN PRINT "YES":GOTO 10
50 PRINT A$, "and", B$+C$
60 GOTO 10
Ok
run
? file
? name
YES
? file
? name
YES
? "file"
? name
filename      and      filename
?
```

この例で判るように INPUT 文で入力されるデータは前後の空白が無視される。もし、空白を入力データの一として入力するためには文字データを引用符で囲む必要がある。

2-4-6 READ 文と DATA 文

変数にデータを入力するステートメントとして INPUT 文は既に説明してきた。READ 文は DATA 文と共に変数に初期値（数値でも文字でもよい）を設定するためのステートメントである。

READ 文の書式

READ <変数> [, <変数>]

例をあげれば次の通りである。

```
list
90 'EX-14
100 READ A1,A2,A3,B$
110 DATA 10,20,30.5,X-30
120 PRINT A1;A2;A3;B$
Ok
run
10 20 30.5 X-30
```

READ 文の変数と DATA 文のデータとは数値型か文字型かで型が一致していなくてはならない。

2-3-13 END 文

END 文はプログラムの実行を終了させる。END 文はプログラムのどこに置いても、また幾つおいてもかまわない。プログラムの最後に END 文はなくても、あったと同様にプログラムは終了する。

END文を使った例

```
90 'EX-15
100 INPUT A
110 IF A<=0 THEN END
120 N=N+1
130 S=S+A
140 PRINT N;S
150 GOTO 100
Ok
run
? 1
1 1
? 2
2 3
? 3
3 6
? 4
4 10
? 0
```

?マークに続いてデータを入力するとNとSの値をプリントする。この場合、Nはプリント（あるいは入力）の回数、Sは入力データの合計である。入力データが零または負のときはプログラムは終了する。

2-3-14 FOR NEXT文

FOR NEXT文はFOR文とNEXT文にはさまれたプログラムを何回か繰返して実行する。

FOR～NEXT文の書式

FOR <変数> = X TO Y [STEP Z]

:

NEXT [<変数>], <変数>………]

変数はカウンタとして利用され、ループの制御変数と呼ばれる。X, Y, Zは変数でも定数でも良い。制御変数は初期値Xから、Zづつ増加し、Y（終値と呼ぶ）を超えた所でループを中止する。STEP Zを省略すると増分は1となる。もしZが負のときは、制御変数がYより小さくなつたときループを中止する。

FOR～NEXTの例

[例1]

```
90 'EX-16
100 FOR I=1 TO 5
110 PRINT I;
120 NEXT I
Ok
run
1 2 3 4 5
```

[例 2]

```
90 'EX-17
100 FOR J=1 TO 10 STEP 2
110 PRINT J;
120 NEXT J
Ok
run
1 3 5 7 9
```

[例 3]

```
90 'EX-18
100 FOR I=10 TO 5 STEP-.6
110 PRINT I;
120 NEXT I
Ok
run
10 9.4 8.8 8.2 7.6 7 6.4 5.8 5.2
```

FOR～NEXT文は多重にして使用することもできる。（多重ループ）

[例 4]

```
90 'EX-19
100 FOR I=1 TO 4
110 FOR J=2 TO 3
120 PRINT I,J
130 NEXT J,I
Ok
run
1 2
1 3
2 2
2 3
3 2
3 3
4 2
4 3
```

この例からも判るように内側のFOR～NEXT文（この場合FOR J = 2 TO 3～NEXT J）が全部で4回実行されていることが判る。多重ループでは常に内側のループが早く回転する。

[例 5]

```
list  
90 'EX-20  
100 FOR I=0 TO 3  
110 FOR J=1 TO 3  
120 IF I+J>=4 THEN 150  
130 PRINT I,J  
140 NEXT J  
150 NEXT I  
Ok  
run  
0 1  
0 2  
0 3  
1 1  
1 2  
2 1
```

この例からも判るように、ループの外へ飛び出すことは可能である。しかしながらループの中へ飛び込むことは出来ない。ループは常に FOR 文から入る必要がある。

上の例では 120 行で $i + j$ が 4 以上になると FOR $J = 1 \text{ TO } 3 \sim \text{NEXT } J$ 文から飛び出しが、再び 110 行の FOR J 文から入るので、何ら問題はない。

FOR NEXT 文では次のような使い方はできない。

[例 6] 誤った FOR 文の使い方

(I) → 100 FOR i = 1 TO 5
110
120
130 ステートメント ←
.....
160 NEXT i
.....
200 GOTO 130 —

ループの中へ飛び込んでいる。

(II) → 100 FOR i = 1 TO 5
→ 110 FOR j = 1 TO 5
.....
.....
150 NEXT i
160 NEXT j

内側の FOR 文は入子になつていなければならない。すなわちループは交錯してはいけない。

(Ⅲ)

```
100 FOR i = 1 TO 5  
110 .....  
120 i = 30  
.....  
150 NEXT i
```

ループ内で制御変数を変えてはいけない。

2-4-9 RESTORE 文

同じデータ文を何回か繰返して使いたい場合に、便利なステートメントである。

RESTORE文の書式：

RESTORE [<行番号>]

この文が実行されると、次の READ文はプログラム中の最初の DATA文からデータを読む。もし行番号が指定されれば、次の READ文は指定された行の DATA文から順次データを入力する。

[例]

```
list  
90 'EX-21  
100 READ A,B,C  
110 READ D,E,F  
130 RESTORE 160  
140 READ G,H,I  
150 DATA 1,2,3  
160 DATA 4,5,6  
170 PRINT A;B;C  
180 PRINT D;E;F  
190 PRINT G;H;I  
Ok  
run  
1 2 3  
4 5 6  
4 5 6
```

2-5 基本ステートメントを使ったプログラム例

これまで説明してきた基本ステートメントだけを使って、いろいろなプログラムを作成できる。以下に幾つかの例を示すことにする。なお参考として、一部にフローチャートも示すこととした。

[例題 1] いろいろな大きさの円について面積と円周の長さを計算したい。円の大きさを半径で示し、キーボードから入力し、結果をスクリーンに示せ。

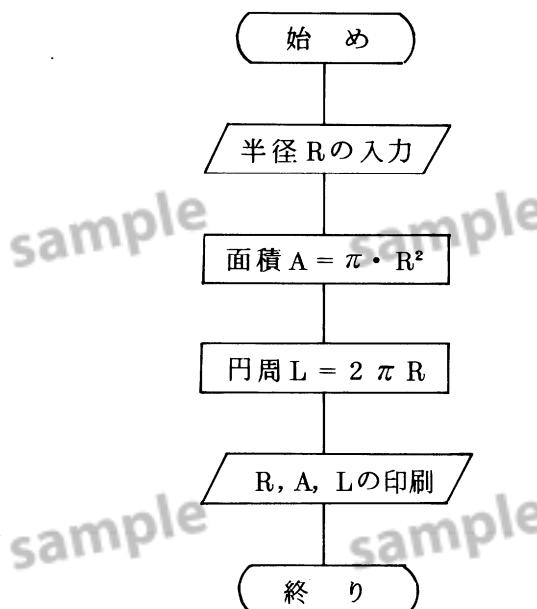
sample

sample

sample

sample

sample



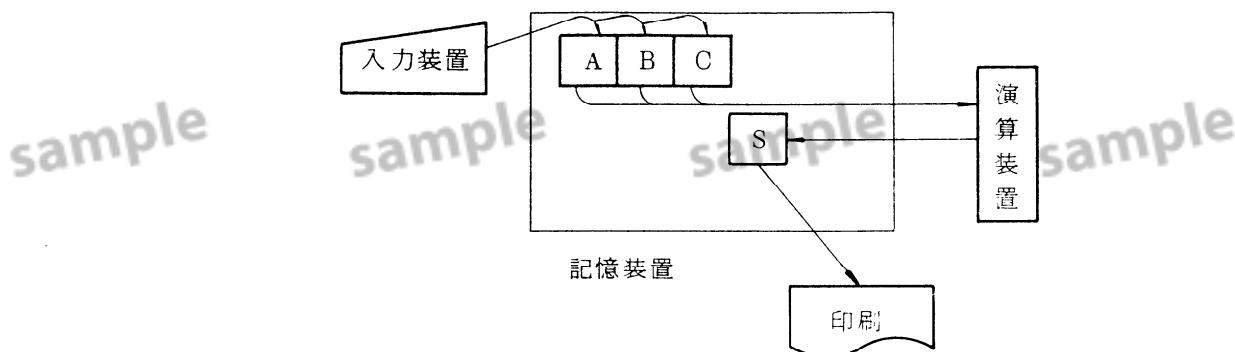
```

9 'EX-22
10 REM インノメンセキトエンシュウノケイサン
20 REM R=バンケイ
30 REM A=メンセキ
40 REM L=エンシュウ
50 PAI=3.1415
60 INPUT"バンケイ=";R
70 A=PAI*R*R
80 L=2*PAI*R
90 PRINT"バンケイ=";R
100 PRINT"メンセキ=";A
110 PRINT"エンシュウ=";L
120 PRINT:PRINT:PRINT
130 GOTO 60
  
```

[解説] 計算のしくみ

$S = A + B + C$ というステートメントをコンピュータが計算する場合、その計算ステップを少し詳しくみてみよう。A, B, C の変数には、前もって READ 文でデータが与えられているものとする。これらのデータはコンピュータの記憶装置に記憶されているが、変数名、A, B, C はこれに対応するデータ記憶装置における記憶場所を指定しているものと見ることができる。従って、 $S = A + B + C$ のステートメントをコンピュータが計算するときに、まず、記憶装置内の A で示される場所からデータを持ってきて、これを演算装置と呼ばれる装置に入力する。次に記憶装置内の場所 B からデータを持ってきて、同じく演算装置に入力すると、そこで A + B の値が計算される。そしてその計算値が A に代って新たに、演算装置の内容となる。続けて、

第 1 図 計算の仕組



同様に C からデータが取り出され、これを演算装置に入力すると、 $A + B + C$ が計算され、その結果がやはり演算装置に残される。この計算結果を記憶装置内の場所 S に貯蔵する命令が代入文の等号である。すなわち $S = A + B + C$ のステートメントは $S \leftarrow A + B + C$ のように等号

を矢印に替えて考えると、その意味が理解しやすい。このように1つの数式の演算は幾つかの時間的に異なるステップで遂次計算される。従って、 $S = S + A$ のような数式は一見、無意味なものと考えられるかも知れないが、BASICでは立派なステートメントとして非常に良く使われる。この場合、右辺の $S + A$ における S の値と左辺の S の値は一般には等しくなく（ $A = 0$ のときだけ両者の S の値は同じである）左辺の S の値は右辺の S の値に A の値を加えたものに等しい。このように変数の値が異なるのは変数の値の記憶されるタイミングが異なるからである。同一の数式においても、前に述べたように計算は同時に進行なわれる訳でなく、非常にわずかの時間であるが、タイミングは異なる点に注意されたい。一つの変数に2つの異なるデータが記憶された場合、常に時間的に後で記憶された方の値が、前に記憶された値にとって代る。

以上の点に注意して、次のプログラムを見てみよう。

```
90 'EX-23
100 A=10
110 S=0
120 S=S+A
130 S=S+A
140 S=S+A
150 PRINT S
Ok
run
30
```

このプログラムによって、計算され、印刷される S の値はどうなるであろうか。

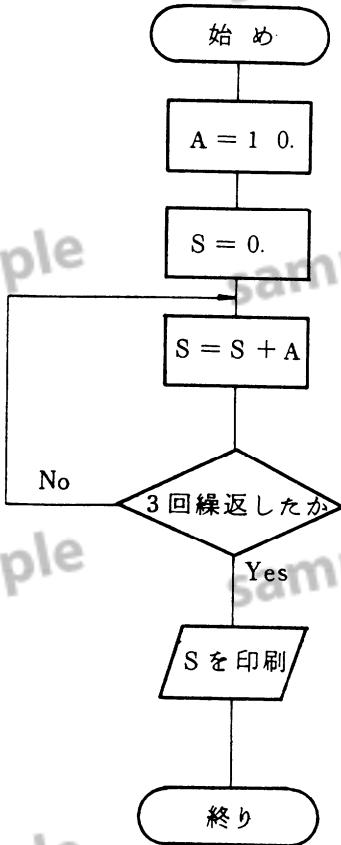
110行で S の値は0となり以下行を進むごとに S の値は10ずつ増え、結局140行の計算が終了した時点で S の値は30になる。従って、答は30である。これは A を3回累積した値になっている。

所で、上のプログラムでは120行から140行までのステートメントが全く同じである。この様な場合、BASICでは次の様に繰返しのステートメントを使って表現できる。

```
90 'EX-24
100 A=10
110 S=0
120 FOR I=1 TO 3
130 S=S+A
140 NEXT I
150 PRINT S
160 END
Ok
run
30
```

FOR 1 = 1 TO 3というFORステートメントは下の140行のNEXT Iとペアになって用いられ、この二つのステートメントで囲まれるステートメント（複数でも良い）を3回繰返して実行することを意味する。

当然であるが、このプログラムの計算結果は前と同様30である。また、このプログラムをフローチャートで示せば次のようになる。



〔例題 2〕 一般に N 人分の給料を読み込んでその合計を計算するプログラムを作成せよ。

方針) いかにコンピュータといえども、何人分のデータが読み込まれるのかを知らなければ、この問題について、結果を出すことはできない。この N の与え方については普通以下に示される二つの方法が良く用いられる。

- 1) N を前もって、入力データとして読み込んでおく。
- 2) 給料についてのデータを入力し終ったら、読み込みの終了を知らせる合言葉で、このことをコンピュータに合図する。なおこの合言葉は姿意的に決めてよいが、データと決して混同しないようなものを設定する必要があり、例えば、この問題では負値や 99999999 などが考えられる。

N の入力によるプログラムの例

```

90 'EX-25
100 INPUT "n=";N
110 S=0
120 FOR I=1 TO N
130 INPUT "a=";A
140 S=S+A
150 NEXT I
160 PRINT "s=";S
170 END
Ok
run
n=? 4
a=? 1
a=? 2
a=? 3
a=? 4
s= 10

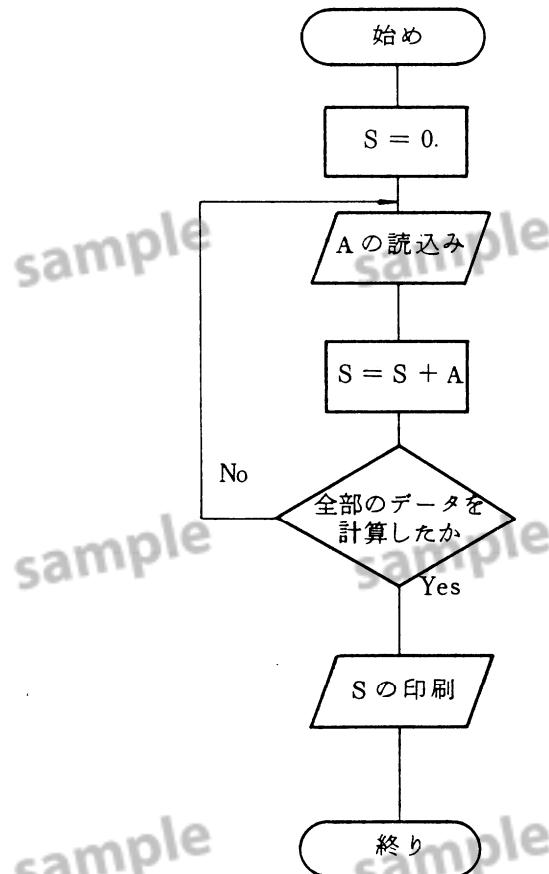
```

合言葉によるプログラムの例（註 1）

```

90 'EX-26
100 S=0
110 INPUT "a=";A
120 IF " " THEN 150
130 S=S+A
140 GOTO 110
150 PRINT "s=";S
160 END
Ok
run
a=? 1
a=? 2
a=? 3
a=? 4
a=? 0
s= 10

```



[例題 3] 3つのデータを読み込んで、その積を求めるプログラムを繰返しの計算法を用いて作成せよ。

```

90 'EX-27
100 S=1
110 FOR I=1 TO 3
120 PRINT I;"×" / data=?";
130 INPUT A
140 S=S*A
150 NEXT I
160 PRINT "s=";S
170 END
Ok
run
1 × data=? 12
2 × data=? 8
3 × data=? 5
s= 480

```

[例 4] 正の実数 (< 32767) を読み込んで、これの小数点以下を 4 捨 5 入せよ。

```

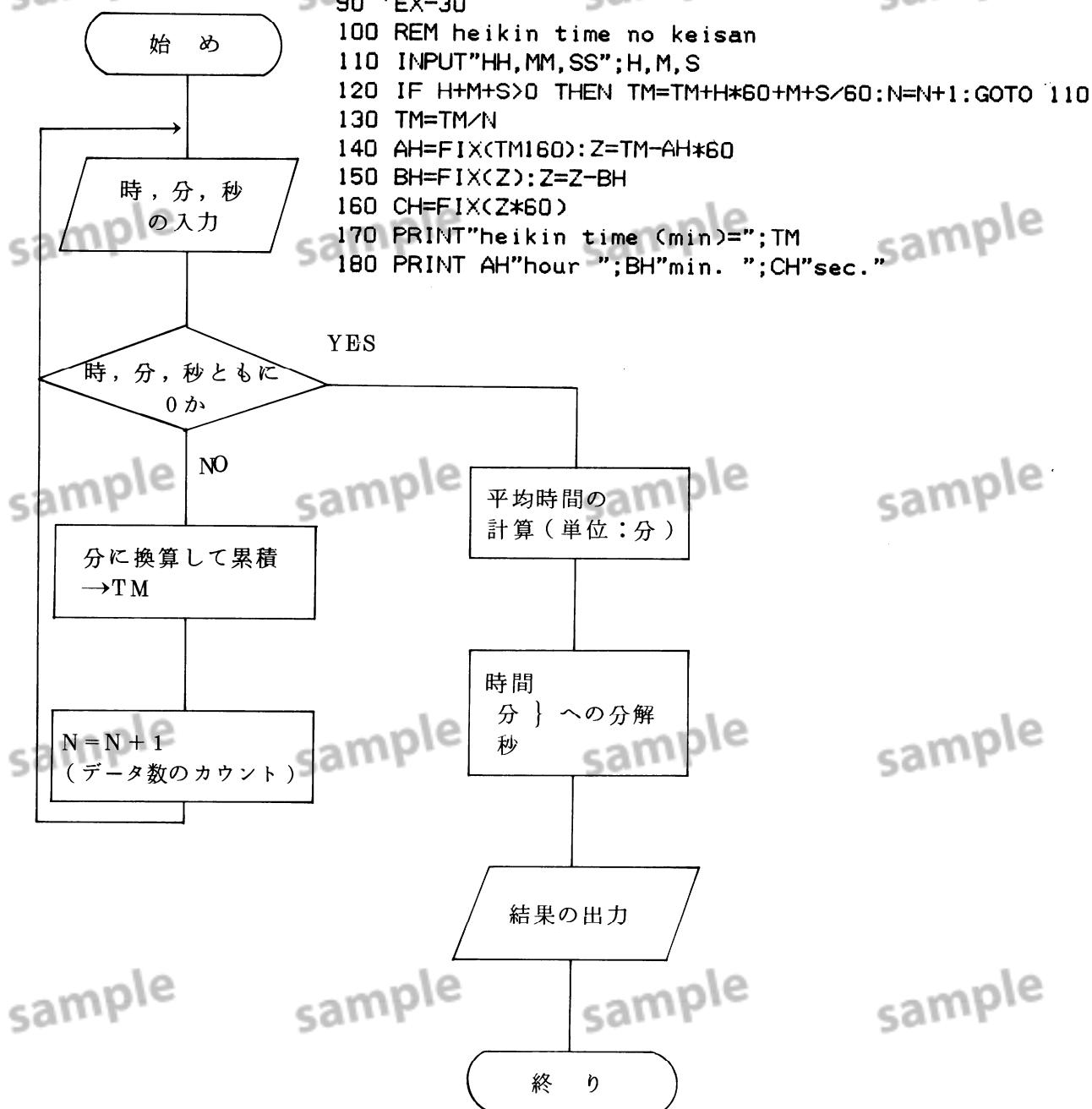
90 'EX-28
100 REM シャゴ ニュウ
110 INPUT "inp a number ";N
120 L% = N+.5
130 PRINT L%
140 GOTO 100

```

[例 5] N人の給料を読み込み、合計金額、平均給与を計算せよ。

```
90 'EX-29
100 INPUT "キュウリョウ="; KY
110 IF KY<=0 THEN GOTO 140
115 N=N+1
120 S=S+KY
130 GOTO 100
140 PRINT "コ" ウケイ=""; S
150 PRINT "ソウスウ="; N
160 PRINT "ハイキン キュウヨ="; S/N
Ok
```

[例 6] 校内マラソンの記録をもとに、平均タイムを計算したい。個々の記録は X X時間、Y Y分、Z Z秒の形で記録されている。



[例 7] RND関数を使って、0から100までの整乱数を50個発生し、これを表示せよ。

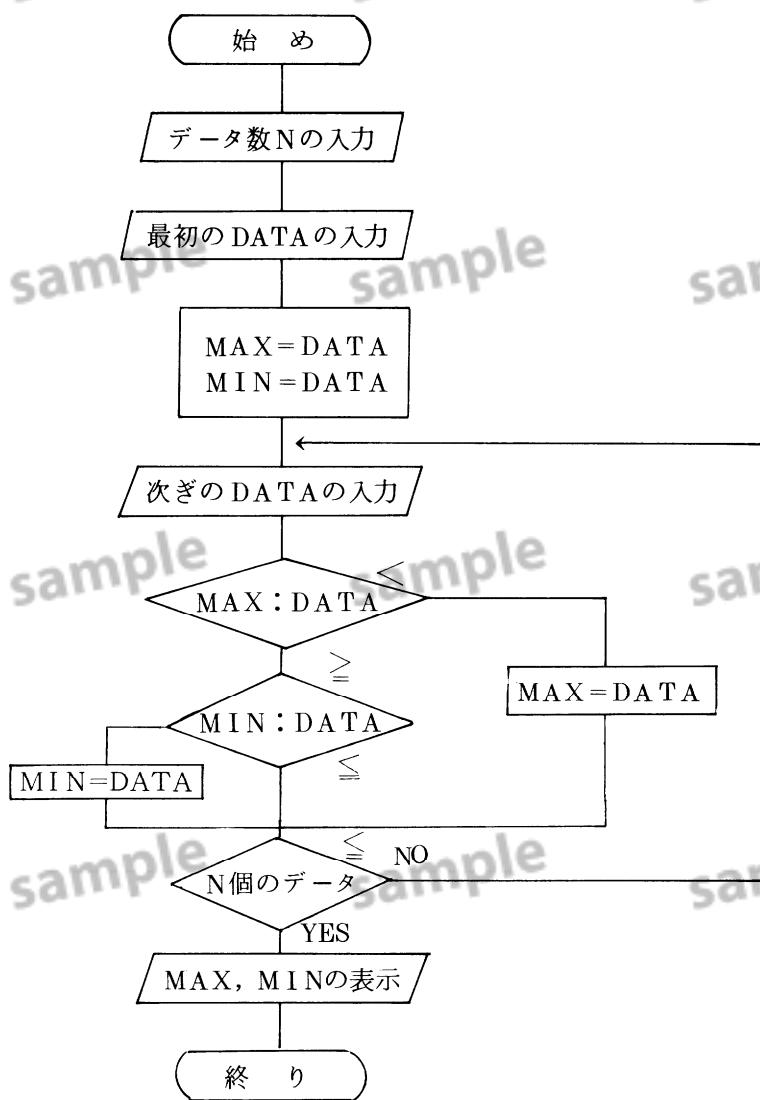
```

list
90 'EX-31
100 FOR I=1 TO 50
110 X%=RND(1)*100
115 PRINT X%;
120 NEXT I
130 END
Ok
run
 3 61 96 97 39 85 86 80 92 37 14 12 6 61 25 38 25 49 12 51
 42 48 20 45 1 100 42 15 96 98 21 4 71 37 2 55 70 11 78 11
 85 12 94 18 11 85 92 69 42 74

```

[注] RND(1)によって発生する一様乱数Rは $0 \leq R < 1$ の範囲で発生するので、これを100倍すると $0 \leq 100 \times R < 100$ の範囲の一様整乱数となる。

[例 8] n個の数値データを読み込んで、最大値と最小値を表示せよ。



```

90 'EX-32
100 INPUT "no.of data?";N
110 INPUT "data 1?";X
120 MAX=X:MIN=X
130 FOR I=2 TO N
140 PRINT "data";I;"=";
150 INPUT X
160 IF MAX>X THEN MAX=X:GOTO 180
170 IF MIN>X THEN MIN=X
180 NEXT I
190 PRINT "Max=";MAX
200 PRINT "Min=";MIN
210 END

```

```

run
no.of data=? 12
data 1=? 23
data 2=? 25
data 3=? 10
data 4=? 48
data 5=? 69
data 6=? 12
data 7=? 50
data 8=? 71
data 9=? 22
data 10=? 30
data 11=? 12
data 12=? 59
Max= 71
Min= 10

```

[例 9] 下に示されるように製品売上高が与えられた場合、製品別のシェアを求めるプログラムを作成せよ。

製品名	A	B	C	D	E	F
売上	110	200	800	700	180	400

```
90 'EX-33
100 FOR I=1 TO 6
110 READ X
120 S=S+X
130 NEXT I
140 RESTORE
150 FOR I=1 TO 6
160 READ X
170 PRINT"(;"I;")";X/S*100;"%"
180 NEXT I
190 DATA 110,200,800,700,180,400
Ok
run
( 1 ) 4.60251 %
( 2 ) 8.3682 %
( 3 ) 33.4728 %
( 4 ) 29.2887 %
( 5 ) 7.53138 %
( 6 ) 16.7364 %
```

このプログラムでは製品名がうまく表示できない。そこで第2案として次のプログラムを示す。

```
90 'EX-34
100 FOR I=1 TO 6
110 READ A$,X
120 S=S+X
130 NEXT I
140 RESTORE
150 FOR I=1 TO 6
160 READ A$,X
170 PRINT A$;X/S*100;"%"
180 NEXT I
190 DATA A,110,B,200,C,800
200 DATA D,700,E,180,F,400
Ok
run
A 4.60251 %
B 8.3682 %
C 33.4728 %
D 29.2887 %
E 7.53138 %
F 16.7364 %
```

3 BASIC の関数

この章に示される関数は組込関数と呼ばれ、BASICが前もって準備した関数である。これらの関数はプログラム中いつでも、何の定義もなしに利用できる。

関数に渡す引数は常にカッコで囲む必要がある。ただし、関数によって、引数には制限があるので、それを以下の表記法に従って示すこととする。

引 数 X と Y : 任意の数式又は数値定数あるいは数値変数

引 数 I と J : 任意の整数定数又は整数変数あるいは数式

引 数 X\$ と Y\$: 任意の文字定数又は文字変数あるいは文字の演算式

3-1 数値関数

ここでは関数の結果が数値となるものの中で、数値計算で良く利用されると思われるものを特に取り上げて説明することにする。

3-1-1 亂 数

書 式 : RND(X)

RNDは0と1の一様乱数を発生する。ただし、Xが負の場合には新しい乱数の系列を作り、Xが正のときには、1つ前に発生した乱数を与える。普通はXに正の数（例えば1）を使う。

[例] $0 \leq R < 100$ の整乱数を5個作り、それを印刷せよ。

```
10 FOR I = 1 TO 5
20 PRINT INT( RND(1) * 100 );
30 NEXT
RUN
24     30     31     51     5
```

3-1-2 三角関数

(I)

書 式 : ATN (X)

動 作 : Xに対するアーカンジェントの値をラジアンで与え、演算結果は

$-\pi/2$ から $\pi/2$ の範囲になる。式Xはどのような型の数値であってもかまわないが、ATNの演算は単精度で行なわれる。

```
[例] : 10 INPUT X
20 PRINT ATN(X)
RUN
? 3
```

sample

(Ⅱ)

書式：COS(X)

動作：Xの単位をラジアンとした場合のコサインの値を与える。COS(X)の演算は単精度で行なわれる。

[例] : 10 X = 2 * COS (.4)

20 PRINT X

RUN

1.84212

(Ⅲ)

書式：SIN(X)

動作：Xの単位をラジアンとした場合の三角関数サインの値を与える。SIN(X)の演算は单精度で行なわれる。

(Ⅳ)

書式：TAN(X)

動作：Xの単位をラジアンとした場合の三角関数タンジェントの値を与える。TAN(X)の演算は单精度で行なわれる。TAN関数がオーバーフローした場合は“Overflow”エラーが起き、プログラムの実行は停止する。

[例] : 10 Y = Q * TAN(X) / 2

3-1-3 指数関数

(Ⅰ)

書式：EXP(X)

動作：eをX乗した値を与える。Xは87.3366未満でなければならない。もしEXP関数がオーバーフローした場合は，“Overflow”エラーが起る。

[例] : 10 A = 5

20 PRINT EXP (A - 1)

RUN

54.5982

(Ⅱ)

書式：LOG(X)

動作：Xの自然対数を演算する。Xは0より大きくなくてはならない。

[例] : PRINT LOG(45/7)

1.86075

(Ⅲ)

書式：SQR(X)

動作：Xの平方根を与える。ただしXは正または0でなければならない。

[例] : 10 FOR X = 10 TO 25 STEP 5

20 PRINT X, SQR(X)

30 NEXT

RUN

10 3.16228

15 3.87298

20 4.47214

25 5

3-1-4 型変換に関する関数

(I)

書式: CDBL(X)

動作: Xを倍精度形式の数値に変換する。

[例] : 10 X = 454.67

20 Y# = CDBL(X)

30 PRINT Y#

RUN

454.6700134277344

(II)

書式: CINT(X)

動作: 小数部分を切り捨ててXを整数に変換する。Xが -32768 から 32767 までの範囲にないときには "Overflow" エラーが起る。

[例] : PRINT CINT(45.67)

45

(III)

書式: CSNG(X)

動作: Xを単精度形式の数値に変換する。

[例] : 10 X# = 975.3421 #

20 PRINT X#; CSNG(X#)

RUN

975.3421 975.342

(IV)

書式: FIX(X)

動作: Xの整数部分を与える。つまり FIX(X) は SGN(X) * INT(ABS(X)) と同じである。FIXとINTの主な違いは、Xが負のとき FIX は X より大きい整数を与えることである。

[例] : PRINT FIX(58.75)

58

PRINT FIX(-58.75)

-58

(V)

書式: INT(X)

動作: Xを超えない最大の整数を与える。

[例] : PRINT INT(99.89)

99

PRINT INT(-12.11)

-13

3-1-5 符号に関する関数

(I)

書式: ABS(X)

動作: 式 X の絶対値を与える。

[例] : PRINT ABS(7 * (-5))

35

(II)

書式: SGN(X)

動作: Xが正の場合は 1 を, Xが 0 の場合は 0 を, Xが負の場合は -1 を与える。

3-2 文字を扱う関数

ここでは文字を扱うあるいは文字処理に便利な関数を説明する。

3-2-1 年月日と時刻の表示

(I)

書式: DATE\$

目的: 内蔵のクロックの示す日付を表わす。

解説: 内蔵されているクロックには、時刻と同時に日付を保持する機能もあり、プログラムから日付を読み出すために使う。日付は次のような形式である。

YY/MM/DD

YYが年、MMが月、DDが日を表わすそれぞれ 2 ヶタの数字である。次のように TIME\$ と同じ方法でセットする。

DATE\$ = "80/03/10"

電源投入時には 79/01/01 にセットされている。

(II)

書式：TIME\$

目的：内蔵のクロックの時刻を示す。

解説：BASICは時刻を文字変数の形で読むことができる。TIME\$の形式は、

HH:MM:SS

で示される。HHは00から23まで、MMは00から59まで、SSは00から59までの数である。電源を投入した直後にクロックはすべて、00:00:00にセットされる。従って、あらかじめクロックをセットしない場合はTIME\$はコンピュータを使用していた時間を表わしている。

TIME\$ = "12:34:56"

のように引用句を使って時間を示す文字数を代入するようにする。これは、ダイレクトモード、プログラムモードのどちらでも可能である。

3-2-2 文字処理に関する関数

(I)

3.2 ASC

書式：ASC(X\$)

動作：文字列X\$の最初の文字のアスキーコードを与える。

[例] : 10 X\$ = "TEST"

20 PRINT ASC(X\$)

RUN

84

附録4.2に示されるキャラクター・コード表によれば、アルファベットのTは54である。ただし、この値は16進数による表現なのでアスキーコードに変換するためには、16進数で表示する必要がある。16進数を10進数に変えるには次の計算をすれば良い。

$(54)_{16} = 5 \times 16 + 4 = 84$ すなわち、Tのアスキーコードは84である。

(II)

書式：CHR\$(I)

動作：アスキーコードがIである文字を与える。通常CHR\$は特別な文字を出力するために使われる。たとえば、(CHR\$(7))によって注意を促す文を表示する前などにブザーを鳴らすことができる。

[例] : PRINT CHR\$(66)

B

CHR\$関数はASC関数の逆の機能を持っている。アスキーコード66に相当する文字は10進数の66をまず16進数に変換し、次に附録4.2のキャラクター・コード表から読み取ることができる。10進数の66を $(66)_{10}$ と表わすことになると、 $(66)_{10} = 16 \times 4 + 2 = (42)_{16}$ で

ある。4-2のコード表から42を読むとアルファベットのBであることが判る。

(III)

書式：INSTR([I,] X\$, Y\$)

動作：文字列X\$の中の最初の文字列Y\$を検索し、見つかった位置を与える。Iは、もしあれば、走査を始める位置を指定する。Iは1から255の範囲の数でなければならない。もし、I>LEN(X\$)であったり、LEN(X\$)=0(ヌルストリング)であったり、またY\$が見つからなかった場合、関数の値は0となる。
X\$及びY\$は文字変数、文字式、または文字列のいずれであってもかまわない。
なお、Y\$がヌルストリングの場合、関数の値はIとなる。

[例]：10 X\$ = "ABCDEB"

20 Y\$ = "B"

30 PRINT INSTR(X\$, Y\$); INSTR(4, X\$, Y\$)

RUN

2 6

(IV)

書式：LEFT\$(X\$, I)

動作：X\$の左からI文字で構成される文字列を与える。Iは0から255までの範囲の数でなければならない。もしIが0の場合はヌルストリング(長さが0の文字列)になる。

[例]：10 A\$ = "DISK BASIC"

20 B\$ = LEFT\$(A\$, 4)

30 PRINT B\$

RUN

DISK

(V)

書式：LEN(X\$)

動作：X\$を構成する文字の長さを与える。プリントされない文字及び空白も数える。

[例]：10 X\$ = "NEC PC-8001"

20 PRINT LEN(X\$)

RUN

11

(VI)

書式：MID\$(X\$, I[, J])

動作：文字列X\$の始めからI文字目に始まってJ文字の長さの文字列を与える。IとJは0から255までの範囲の数でなければならない。Jが省略されていたり、I番目の文字より右にJ文字より少ない文字しかない場合には、I番目より右側の

文字全部が関数の値となる。もし $I > \text{LEN}(X\$)$ の場合には、 $\text{MID\$}$ の値はヌルストリングとなる。

[例] : 10 A\\$ = "GOOD"

20 B\\$ = "MORNING EVENING AFTERNOON"

30 PRINT A\\$; MID\$(B\$, 9, 7)

RUN

GOOD EVENING

OK

書式 2 : $\text{MID\$}(X\$, I \lceil J) = Y\$$

動作： $X\$$ 内の I で指定された場所から始まる J 文字を、 $Y\$$ の始めから J 文字で書き換える。

[例] : 10 A\\$ = "123456789"

20 B\\$ = "ABCDEF"

30 MID\$(A\\$, 4, 3) = B\\$

40 PRINT A\\$

RUN

123ABC789

OK

(VII)

書式 : SPACE\$(X)

動作： X 個の空白より成る文字列を与える。ただし X の小数部分は丸めて整数として使う。そしてその値は 0 から 255 までの範囲になければならない。

[例] : 10 FOR I=1 TO 5

20 X\\$ = SPACE\$(I)

30 PRINT X\\$; I

40 NEXT I

RUN

1

2

3

4

5

SPACE 関数の項も参照すること。

(VIII)

書式 : RIGHT\$(X\\$, I)

動作： $X\$$ の右から I 文字の文字列を与える。もし I と $X\$$ の長さが等しい場合には、

RIGHT\$ は $X\$$ になり、また I が 0 ならヌルストリング(長さ 0 の文字列)になる。

[例] : 10 A\$ = "N-BASIC"

20 PRINT RIGHT\$ (A\$, 5)

RUN

BASIC

(X)

書式: STR\$ (X)

動作: Xの数値を表わす文字列を与える。

[例] : PRINT STR\$ (15 + 3) + "秒"

(X)

書式: STRING\$ (I, J)

STRING\$ (I, X\$)

動作: アスキーコードがJの文字、またはX\$の最初の文字を使ったI個より成る文字列を与える。

[例] : 10 X\$ = STRING\$ (10, 45)

20 PRINT X\$ "MONTHLY REPORT" X\$

RUN

----- MONTHLY REPORT -----

(X)

書式: VAL (X\$)

動作: 文字列X\$を表わす数値を与える。もしX\$の最初の文字が+, -, &, または数字でなければ、VALの値は0になる。

[例] : 10 READ NAME\$, CITY\$, STATE\$, ZIP\$

20 IF VAL(ZIP\$) < 90000 OR VAL(ZIP\$) > 96699 THEN

PRINT NAME\$; TAB(25); "OUT OF STATE"

30 IF VAL(ZIP\$) >= 90801 AND VAL(ZIP\$) <= 90815 THEN

PRINT NAME\$; TAB(25); "LONG BEACH"

3-3 その他の関数

(I)

書式: CONSOLE [<スクロール開始行>] [, <スクロールの長さ>] [, <ファンクションキー表示スイッチ>] [, <カラー／白黒スイッチ>]

目的: 画面の表示に関するいろいろな機能を設定する。

解説: <スクロール開始行>と<スクロールの長さ>とによって画面でスクロールが行なわれる窓を指定することができる。たとえば10行目から6行だけをスクロールさせる場合には次のように行なう。

CONSOLE 10, 6

<ファンクションキー表示スイッチ>を 1 にすれば画面の最下段にファンクションキーの内容を表示し、0 にすれば表示しなくなる。また<カラー／白黒スイッチ>を 1 にすればディスプレイ全体をカラー・モードにし、0 にすれば白黒モードにする。

[例] : CONSOLE 0, 10, 1, 1

この例は、一番上の行から 9 行目までの間をスクローリングさせ、ファンクションキーの内容を最下段に表示、画面をカラー・モードにするものである。

(II)

書式: FRE(0)

FRE(X\$)

動作: ERE 関数に渡す引数はダミーである。引数が 0 (数値) の場合は BASIC が使っていないメモリのバイト数を、引数が文字の場合は文字領域の未使用バイト数を与える。

[例] : PRINT FRE(0)

14542

(III) ERR 及び ERL 変数

エラー処理サブルーチンにはいったとき、変数 ERR はエラーの番号を変数 ERL はエラーの起きた行番号を持っている。ERR 及び ERL 変数は通常 IF…THEN 文で使われ、エラー処理ルーチンの処理の流れを制御する。

エラーの原因となった文がダイレクトステートメント（直接実行文）で起きたことを調べるには、IF ERL=65535 THEN…を使用する必要がある。
それ以外のエラーには、次のようにする。

IF ERR=エラー番号 THEN…

IF ERL=行番号 THEN…

その理由は、行番号が等号の右側にない場合は、RENUM コマンドによってリナンバーすることができないからである。また、ERL と ERR は予約変数なので、LET (代入) 文の等号の右側に書くことはできない。BASIC のエラー番号は、付録 4.3 を参照せよ。

(IV)

書式: POS(I)

動作: 画面のカーソルの現在の横位置を与える。（左端が 0），引数 I はダミーである。

(V)

書式: CSRLIN

動作: 画面上のカーソルの垂直位置を与える。（最上位行が 0），横位置を与える POS

関数と異なりダミーの引き数は必要ない。

[例] : LOCATE 4, 5:PRINT POS(0);CSRLIN

4 5 (上から 6 行目)

(V)

書式：SPC(I)

動 作：I 個の空白をプリントする。SPC 関数は PRINT 及び LPRINT 文中のみで使うことができる。I は 0 から 255 までの範囲の数でなければならない。

[例] : PRINT "OVER" SPC (15) "THERE"

OVER THERE

(VII)

書式：TAB (I)

動作：現在カーソルのある行の I 柄目の位置まで空白をプリントする。もし現在のプリント位置が I を起えていれば、TAB は無効となる。位置 0 が左端である。I は 0 から 255 までの範囲の数でなければならない。また TAB は PRINT または LPRINT 文中でしか使えない。

[例] : 10 PRINT "NAME"; TAB(25); "AMOUNT": PRINT

20 READ AS B\$

30 PRINT A\$: TAB(25) : B\$

40 DATA "G T JOHN", "\$25.00"

BUN

NAM

C T

G. T. JOHN \$ 25.00

4 付 錄

4. 1 キー ボード

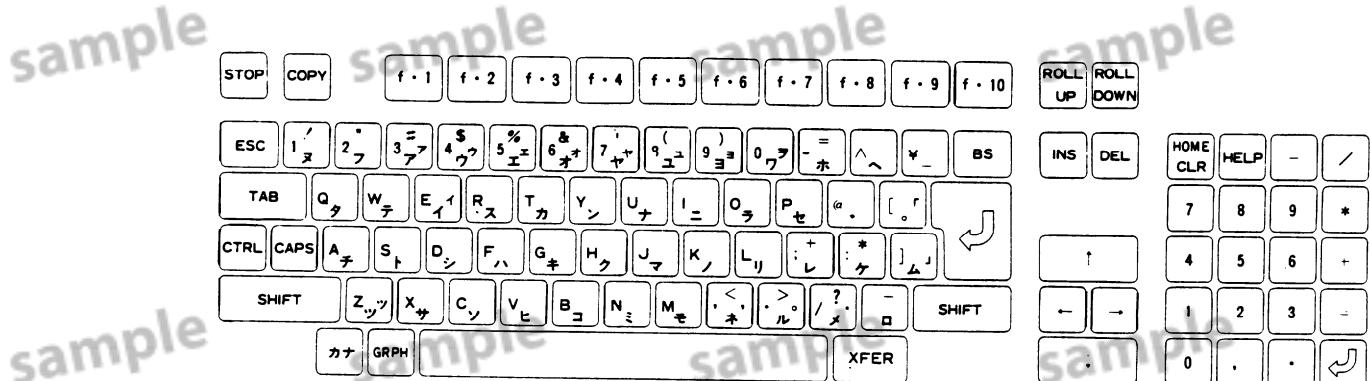


図1 キーの配列

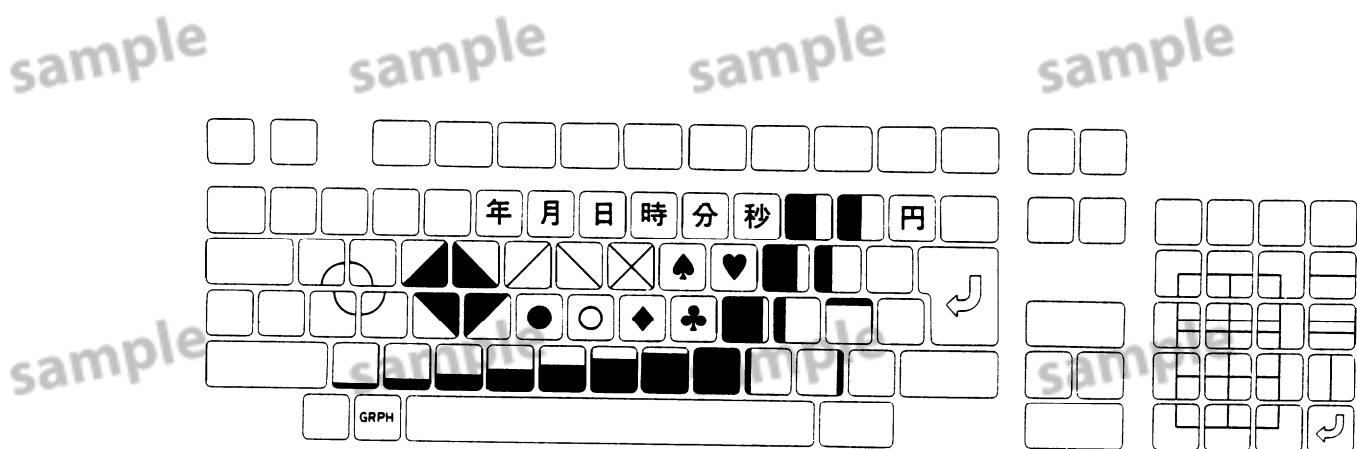


図2 グラフィックシンボルのキー配置

キャラクタ・コード表

	O	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	0	D_E	0	@	P	p					一	タ	ミ	=	X	
1	S_H	D_I	!	I	A	Q	a	q		.	ア	チ	ム	上	円	
2	S_X	D_2	"	2	B	R	b	r		「	イ	ツ	メ	二	年	
3	E_X	D_3	#	3	C	S	c	s		」	ウ	テ	モ	ニ	月	
4	E_T	D_4	\$	4	D	T	d	t		,	エ	ト	ヤ	△	日	
5	E_Q	N_K	%	5	E	U	e	u		・	オ	ナ	ユ	↖	時	
6	A_K	S_N	&	6	F	V	f	v		ヲ	カ	ニ	ヨ	↙	分	
7	B_L	E_B	'	7	G	W	g	w		ア	キ	ヌ	ラ	↗	秒	
8	B_S	C_N	(8	H	X	h	x		イ	ク	ネ	リ	♠		
9	H_T	E_M)	9	I	Y	i	y		ウ	ケ	ノ	ル	♥		
A	L_F	S_B	*	:	J	Z	j	z		エ	コ	ハ	レ	♦		
B	H_M	E_C	+	;	K	[k	{]	オ	サ	ヒ	ロ	♣		
C	C_L	→	,	<	L	¥	I	;]	ヤ	シ	フ	ワ	●		
D	C_R	←	-	=	M]	m	}]	ユ	ス	ヘ	ン	○		
E	S_O	↑	.	>	N	^	n	~]	ヨ	セ	ホ	‘	↖		
F	S_I	↓	/	?	0	o]	ツ	ソ	マ	。	↙		

4.3 エラーメッセージ

注意：エラーコードの欄は、上段がN-BASIC、下段がN_{BS}-BASICのコードを表しています。“_”で示されているのは、存在しないエラーです。

エラーメッセージ	コード	意味
Bad allocation table	[N] 64 [N _{BS}] 69	ディスクケット内部の FAT が壊れている。
Bad drive number	[N] 65 [N _{BS}] 70	ドライブ指定が誤っている。システムにつながっていないドライブを指定した。
Bad File Data	[N] 25 —	ファイル上にあるデータの形式がまちがっている。
Bad file name	[N] 62 [N _{BS}] 56	ファイル名の指定がまちがっている。
Bad file number	[N] 52 [N _{BS}] 52	オープンしていないファイルや起動時に指定していないファイルを参照した。
Bad file mode	[N] 54 —	シーケンシャルでオープンしたファイルに対してランダムアクセスをしようとした。またはその逆。
Bad track/ sector	[N] 66 [N _{BS}] 71	トラック、セクタ番号の指定が誤っている。 (DSKO\$, DSKI\$)
Can't continue	[N] 17 [N _{BS}] 17	CONT 命令によって実行を再開することができない（ポインタの値が壊されている）。
Communications Buffer Overflow	[N] 27 —	周辺機器との入出力のためのバッファがオーバーフローした。（N _{BS} -BASIC では Line buffer overflow）
Deleted record	[N] 67 [N _{BS}] 72	消去済みのレコードをアクセスしようとした。
Direct statement in file	[N] 63 [N _{BS}] 57	アスキー形式のファイルを読み込む際にダイレクトステートメントが存在した。
Disk BASIC Feature	[N] 26 —	ディスクが接続されていないとき、ディスク BASIC の命令を実行した。
エラーメッセージ	コード	意味
Disk full	[N] 60 [N _{BS}] 68	ディスク上に書き込むスペースがないのに書き込みもうとした。
Disk I/O error	[N] 57 [N _{BS}] 64	ディスクとの入出力中にエラーが発生した。致命的なエラーであり回復させることはできない。
Disk offline	[N] 73 [N _{BS}] 62	入出力の可能な状態でないディスクをアクセスしようとした。
Division by Zero (/0)	[N] 11 [N _{BS}] 11	0 による割り算が実行されて、その結果の値がオーバーフローした。
Duplicate Definition	[N _{BS}] 10	配列またはユーザー関数を二重定義しようとした。（N-BASIC では、Redimensioned array）
Duplicate label	[N _{BS}] 31	同じラベル名が 2 つ以上存在している。
Feature not available	[N _{BS}] 33	利用不可能な機能を指定した。
FIELD overflow	[N] 50 [N _{BS}] 50	FIELD 文において 256 バイト以上の大きさの領域を指定した。
File already exists	[N] 58 [N _{BS}] 65	NAME 文によって変更しようとしたファイル名が既に存在している。
File already open	[N] 55 [N _{BS}] 54	既にオープンされているファイルに対して OPEN、KILLなどを実行した。
File not found	[N] 53 [N _{BS}] 53	LOAD、SAVE、KILL などで現在のディスクに存在しないファイルを指定した。
File not open	[N] 71 [N _{BS}] 60	オープンされていないファイルを参照しようとした。

File write protected	(N) [N _{BB}]	72 61	書き込み禁止属性が付けられているファイルに書き込もうとした。
FOR without NEXT	(N) [N _{BB}]	— 26	FOR～NEXT が正しく対応していない (FOR が多い)。
Illegal direct	(N) [N _{BB}]	12 12	ダイレクトモードで使用できない文を実行しようとした。
Illegal function call	(N) [N _{BB}]	5 5	ステートメントおよび関数において、機能の呼び方が誤っている。
Illegal operation	(N) [N _{BB}]	— 74	不正なキーボード操作をした。
Input past end	(N) [N _{BB}]	61 55	ファイル中の全てのデータを読み尽した後に、さらに入力文が実行された。
Internal error	(N) [N _{BB}]	51 51	BASIC 内部にエラーが生じた。
Line buffer overflow	(N) [N _{BB}]	23 23	1 行で入力できる文字の範囲を超えて入力が行われた。
Missing operand	(N) [N _{BB}]	22 22	ステートメント中必要なパラメータが指定されていない。
NEXT without FOR	(N) [N _{BB}]	1 1	FOR～NEXT が正しく対応していない。 (NEXT が多い)
No RESUME	(N) [N _{BB}]	19 19	エラー処理ルーチンの中に RESUME 文がなく、プログラムの実行が継続できない。
Out of DATA	(N) [N _{BB}]	4 4	読むべきデータがないのに READ 文が実行された。
Out of memory	(N) [N _{BB}]	7 7	メモリ容量が足りなくなつた (プログラムが大きすぎる、またはスタックを消費し尽した)。
Out of string space	(N) [N _{BB}]	14 14	文字列を格納するメモリ領域がなくなつた。
Overflow (OV)	(N) [N _{BB}]	6 6	演算結果や入力された数値が、許される範囲を越えた。
Port not initialized	(N) —	28 —	インターフェイス用の LSI の機能設定がなされていない。
Position not on screen	(N) —	24 —	指定したカーソル位置などが、画面の範囲外になっている。
Redimensioned array	(N) —	10 —	配列またはユーザー関数を二重定義しようとした (N _{BB} -BASIC では Duplicate Definition)。
Rename across disks	(N) [N _{BB}]	68 73	NAME 文において、異なるドライブ間でリネームしようとした。
RESUME without error	(N) [N _{BB}]	20 20	エラー処理ルーチンに制御を飛ばなかったのに RESUME 文がある。
RETURN without GOSUB	(N) [N _{BB}]	3 3	GOSUB～RETURN が正しく対応していない。 (RETURN が多い)
Sequential after PUT	(N) [N _{BB}]	69 58	PUT 文実行後、シーケンシャルファイルをアクセスしようとした。
Sequential I/O only	(N) [N _{BB}]	70 59	シーケンシャル入出力以外は行ってはならない。
String formula too complex	(N) [N _{BB}]	16 16	文字式が複雑すぎる。 (カッコのネスティングが深すぎるなど)
String too long	(N) [N _{BB}]	15 15	1 つの文字変数内の文字数が 255 文字を超えている。
Subscript out of range	(N) [N _{BB}]	9 9	配列変数の添字の値が規定の範囲から外れている。

Syntax error	[N] [N] 2 2	文の記述が文法通りになっていない。
Tape read error	[N] [N] 29 27	テープからの読み込み時にエラーが発生した。
Too many files	[N] 67 —	ファイルの数が許容される範囲(255個)を超えた。
Type mismatch	[N] [N] 13 13	式の左辺・右辺、関数の引数などにおいて変数の型が一致していない。
Undefined label	— [N] 32	定義されていないラベル名で参照した。
Undefined line number	[N] [N] 8 8	飛び先として必要とされる行番号(GOTO, GOSUBなどの飛び先)が存在しない。
Undefined user function	[N] [N] 18 18	DEFFN 文によって定義されていないユーザー関数を引用しようとした。
Unprintable error	[N] [N] 21 21	メッセージの定義されていないエラー。
WEND without WHILE	— [N] 30	WHILE～WEND が正しく対応していない。 (WEND が多い)
WHILE without WEND	— [N] 29	WHILE～WEND が正しく対応していない。 (WHILE が多い)

4.4 予 約 語

ABS	CVI	FPOS	KLOAD	OCT\$	RANDOMIZE	SGN	TROFF
AKCNV\$	CVS	FRE	KTYPE	OFF	RBYTE	SIN	TRON
AND	DATA	GET	LEFT\$	ON	READ	SPACE\$	USING
ASC	DATE\$	GOTO	LEN	OPEN	REM	SPC	USR
ATN	DEF	GO TO	LET	OPTION	RENUM	SQR	VAL
ATTR\$	DEFDBL	GOSUB	LFILES	OR	RESTORE	SRQ	VARPTR
AUTO	DEFINT	HELP	LINE	OUT	RESUM	STATUS	VIEW
BEEP	DEFSNG	HEX\$	LIST	PAINT	RETURN	STEP	WAIT
BLOAD	DEFSTR	IEEE	LLIST	PEEK	RIGHT\$	STOP	WBYTE
BSAVE	DELETE	IF	LOAD	PEN	RND	STR\$	WEND
CALL	DIM	IMP	LOC	POINT	ROLL	STRING\$	WHILE
CDBL	DRAW	INKEY\$	LOCATE	POKE	RSET	SWAP	WIDTH
CHAIN	DSKF	INP	LOF	POLL	RUN	TAB	WINDOW
CHR\$	DSKIS\$	INPUT	LOG	POS	SAVE	TAN	WRITE
CINT	DSKO\$	INPUT\$	LPOS	PRESET	SCREEN	TERM	XOR
CIRCLE	EDIT	INSTR	LPPINT	PRINT	SEARCH	THEN	
CLEAR	ELSE	INT	LSET	PSET	SEG	TIME\$	
CLOSE	END	IRESET	MAP	PUT	SET	TO	
CLS	EOF	ISET	MERGE				
CMD	EQV	JIS\$	MID\$				
COLOR	ERASE	KACNV\$	MKD\$				
COM	ERL	KANJI	MKI\$				
COMMON	ERR	KEXT\$	MKS\$				
CONSOLE	ERROR	KEY	MOD				
CONT	EXP	KILL	MON				
COPY	FIELD	KINPUT	MOTOR				
COS	FILES	KINSTR	NAME				
CSNG	FIX	KLEN	NEXT				
CSRLIN	FN	KMID\$	NEW				
CVD	FOR	KNJ\$	NOT				

4.5 漢字の入力方法

1. WIDTH 80, 25 または WIDTH 80, 20 とする。
2. CTRL + XFER キーを押す。
3. 入力文字は英大文字, 英小文字, カタカナ, ひらがな, の 4 種を選べる。
 - イ. 英大文字: CAPS キーを押す。
 - ロ. 英小文字: CAPS キーを戻す。
 - ハ. カタカナ: カナキーを押す。
- ニ. ひらがな: カナモードにして CTRL+Q を押す。更に, CTRL+Qによりカナモードに戻る。
4. ローマ字入力でカナあるいはひらがなに変換したいときは CTRL+V キーを押す。
5. 漢字変換する場合は, 上の 3 - ハ, 3 - ニあるいは 4 の方法でカナまたはひらがなで表示した後, XFER キーを押す。
6. 表示された漢字を番号で選択する。該当する漢字がない場合は更に, XFER を押してみる。
7. ローマ字入力の場合, “ん”は N + , キーまたは N + 子音で入力できる。
8. 入力した文字を漢字変換しないで使いたい場合は SHIFT+XFER キーを押す。
9. 通常のモードに戻るにはリターン・キーを押す。

sample

sample

sample

sample

sam

sample

sample

sample

不 許 複 製

慶應義塾大学ビジネス・スクール

Contents Works Inc.