SHARP®

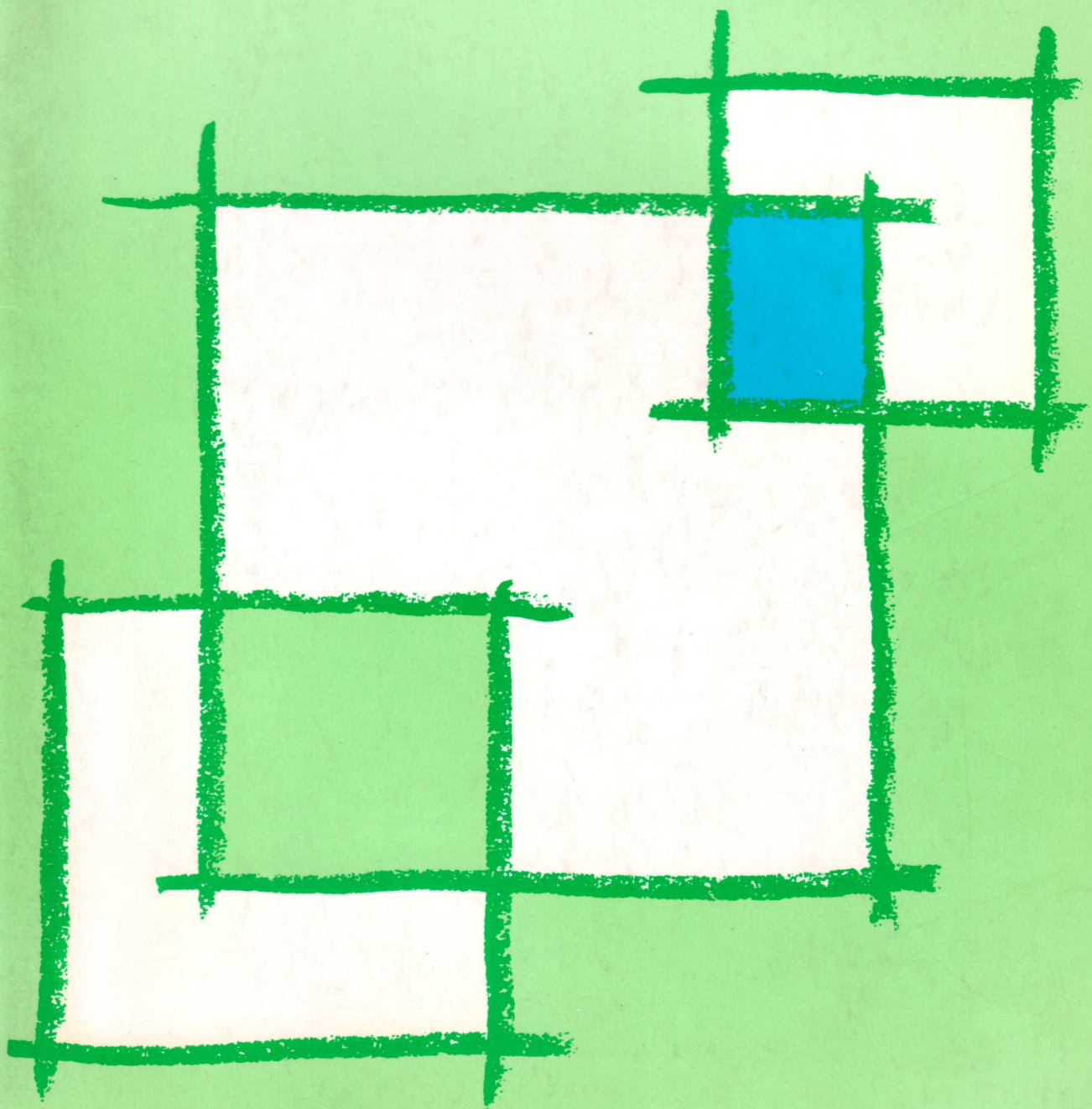
パーソナルコンピュータ

MZ-2800

形名

MZ-2861

BASIC-M28マニュアル



MZ-2800

BASIC-M28 マニュアル

© 1987 SHARP CORPORATION

ご注意

(1) 本書および本ソフトウェアの内容の全部、または一部を、当社に無断で転載、あるいは複製することは禁じられています。

(2) 本書の内容、および本ソフトウェアは改良などのため、将来、予告なく変更することがあります。

(3) 本書の内容、および本ソフトウェアについては、万全を期して作成しておりますが、万一不可解な点や、誤り、記載もれなどお気付きの点がございましたら、最寄りのお客様ご相談窓口までご連絡ください。

(4) 本ソフトウェアおよび、本書記載のソフトウェアを使用したことによるお客様の損害、および逸失利益、または第三者からのいかなる請求につきましても、当社はその責任を負いかねますので、あらかじめご了承ください。

(5) MS-DOS™は米国マイクロソフト社の登録商標です。

本書中では、キーの入力のしかたとして次の略式を使っています。

[シフト]+[停止] :左側に示されたキーを押しながら右側のキーを押すことを意味します。

↑
└─押しながら

はじめに

本書では、シャープパーソナルコンピュータMZ-2861の、2800モードで使用するBASIC言語、BASIC-M28について解説しております。

本書を十分にお読みいただき、MZ-2861のすぐれた機能を自在に使いこなしてください。

本書の内容をより効率的にご理解いただくために本書をお読みになる前に「本書の読みかた」をよくお読みください。

目　次

本書の読みかた…… 1

第1部　BASICの概要

1 BASICの起動 …… 4
MS-DOSの起動／BASICの起動／ディスクのコピー／
BASICの起動時オプション／BASICの自動実行
起動時の初期設定 …… 7
起動時の状態

2 編集と実行 …… 8
実行モード …… 8
直接実行モード／間接実行モード
命令の種類 …… 10
コマンド／ステートメント／関数／システム変数
プログラムの書式 …… 12
行番号／ラベル／文／行の長さ
プログラム中のスペース …… 14
BASICで使用する文字 …… 15
全角文字と半角文字／アルファベットの大文字と小文字
特別な記号 …… 16
予約語と省略形 …… 17
予約語／省略形
プログラムの編集 …… 20
プログラムの入力／プログラムの変更／プログラムの消去／
ピリオド(.)の特別機能
編集のためのキー操作 …… 23
カーソルの移動／タブの設定と解除／文字の挿入／文字の削除／
文字の消去／行の挿入／コンソールエリア／文字のコピー機能／
ハードコピー／そのほかの制御操作
プログラムの実行……28
メインルーチンとサブルーチン……29

3 BASICで扱うデータ …… 30
定数 …… 31
整数型／単精度実数型／倍精度実数型／文字列型

変数 …… 33
変数名／変数の型
配列 …… 34
配列名と添字／OPTION BASE／配列宣言／配列変数の消去
式と演算子 …… 36
式／算術演算
比較演算 …… 37
論理演算 …… 38
関数 …… 39
数値関数／文字列処理関数／関数定義
システム変数 …… 41
システム変数／システム定数
データの型変換 …… 42
エラー …… 44
エラーの処理
演算誤差 …… 45

4 漢字の扱いについて …… 47
全角文字
全角文字の入力(漢字入力モード)……48
コードの変換 …… 49
全角文字と半角文字の変換
文章の表示・印字機能 …… 50

5 ファイルの管理 …… 51
ファイル／ディレクトリ／ファイルの種類
ファイルの指定方法 …… 53
ファイル指定／装置名／ディレクトリパス／ファイル名／拡張子

6 ファイルの保存と再生 …… 56
プログラムファイル …… 56
プログラムの保存／プログラムの再生／書き込みチェック／
ファイルの消去／ファイル名の一覧／プログラムの合成／
プログラムの連結／ファイル名の変更／ファイルのプロテクト／
ファイルの保護／ファイルのコピー

データファイル …… 61
シーケンシャルアクセスファイル／ランダムアクセスファイル／
シーケンシャルファイルへのデータの保存、再生／
シーケンシャルファイルへのデータの追加／
ランダムファイルへのデータの保存、再生／
ランダムファイルに対するデータの追加、変更／ファイルのコピー
階層ディレクトリ …… 71
単層ディレクトリ／階層ディレクトリ／ディレクトリの作成方法／
カレントディレクトリの変更／ディレクトリの削除／
ディレクトリのコピー
7 表示モード …… 81
文字画面 …… 81
グラフィック画面 …… 83
カラーコード …… 85
16色コード／8色コード／65536色コード
文字の色指定 …… 87
グラフィックの色指定 …… 88
パレットコードとパレット機能／4096色パレット／16色モード
での色指定／65536色モードでの色指定
プライオリティ機能 …… 90
文字とグラフィック表示／文字画面どうしの表示
文字座標 …… 91
グラフィック座標 …… 92
絶対スクリーン座標／ワールド座標と絶対スクリーン座標／
ウィンドウ／ビューポートとスクリーン座標／初期設定

8 文字の表示 …… 96
画面の設定 …… 96
スクロールモード／文字フォント／スクロール範囲の設定／
カーソルの移動／ファンクションキーの内容表示／
漢字とグラフィック文字
文字のアトリビュート …… 99
表示色の指定／リバース表示／ブリンク表示／ＲＯＭ ＣＧとＰＣＧ
文字画面の読み込み …… 100
ＧＥＴ%とＰＵＴ%
画面の消去 …… 100

9 グラフィックの表示 …… 101
グラフィック命令の座標指定 …… 101
グラフィック描画時のペン機能 …… 102
グラフィック描画時の機能指定 …… 103
グラフィックの描画 …… 105
点を描く／線を描く／正多角形を描く／円を描く／色を塗る／
色を変える／ 文字パターンの描画／グラフィックパターンの描画
グラフィック画面の読み込み …… 108
ＧＥＴ@とＰＵＴ@／ＧＳＡＶＥとＧＬＯＡＤ
グラフィック画面の消去……108
画面の初期化……109

10 ＰＣＧの定義のしかた …… 111
８×８ドットの定義／８×16ドットの定義／16×16ドットの定義

11 キーボードの制御機能 …… 116
先行入力／キーボードの配列／クリック音／キーリピート／
ファンクションキー／キー割り込み機能

12 時計機能 …… 119
時計による割り込み機能

13 サウンド機能 …… 120
FM音源

14 マウスの操作 ……122
マウスの機能設定／マウスの入力／マウスによる割り込み処理

15 プリンタ制御機能 ……124
文字の印字／スプーラ機能／画面のコピー／
ファイル装置としてのプリンタ／プリンタの直接制御

16 ディスプレイテレビ制御機能 ……126

17 RS-232C インターフェイスの制御 …… 127
機能設定

18 MS-DOSへの移行 …… 128
チャイルド・プロセス

19 ハードウェアの直接制御 …… 130
機械語エリア／機械語エリアのアクセス／I/Oポートのアクセス／
機械語プログラムの実行

第2部　コマンド・ステートメントの解説

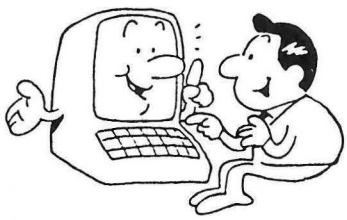
基本コマンド …… 135
一般ステートメント …… 157
基本入出力命令 …… 175
実行制御命令 …… 188
表示設定命令 …… 203
文字表示命令 …… 215
グラフィック命令 …… 236
ファイルコマンド …… 282
ファイルステートメント・関数 …… 314
数値関数 …… 345
文字列関数 …… 367
漢字処理命令 …… 384
例外処理命令 …… 398
チャイルドプロセス制御命令 …… 426
数値演算プロセッサ命令 …… 431
マウス命令 …… 432
キーボード制御命令 …… 438
ジョイスティック命令 …… 449
プリンタ制御命令 …… 452
クロック命令 …… 457
サウンド命令 …… 461
モデムホン制御命令 …… 480
ディスプレイテレビ制御命令 …… 481
機械語処理・特殊関数 …… 483

付　録

サンプルプログラム(65536色モード) …… 502
文字コード表(半角文字) …… 503
エラーメッセージ …… 504

BASICのフリーエリア …… 507
コントロールコード表 ……509
RS-232Cインターフェイス ……510
三角関数と双曲線関数 ……513
機械語サブルーチン ……514
ターミナルプログラム …… 516
BASIC-M25との互換性について…… 519
ファイルコンバートユーティリティ …… 521
エラーメッセージ(ファイルコンバータ)……525
索引(機能別) ……526
索引(アルファベット順) …… 536

本書の読みかた



本書は、以下の構成で書かれております。

第1部　BASICの概要

ここではBASIC-M28のプログラミングに必要な知識が述べられています。

第2部　コマンド・ステートメントの解説

ここではBASIC-M28で使用する各命令の機能や文法、注意事項、使用例などが説明されています。

付　録

本書の内容をより効率的にご理解いただくために、以下のようにお読みいただくことをおすすめします。

BASIC-M28を初めてお使いになる方

第1部をお読みください。ここで登場する命令は、**第2部**でその機能などについてくわしく述べていますので、併せてお読みになると、いっそう理解が深まることと思います。なお、各命令を参照するときは、**索引**（付録）を活用すれば、簡単に探し出すことができます。
また、例として登場するプログラムは、できるだけ実際にキー入力してみて、その動作を確認してみることをおすすめします。

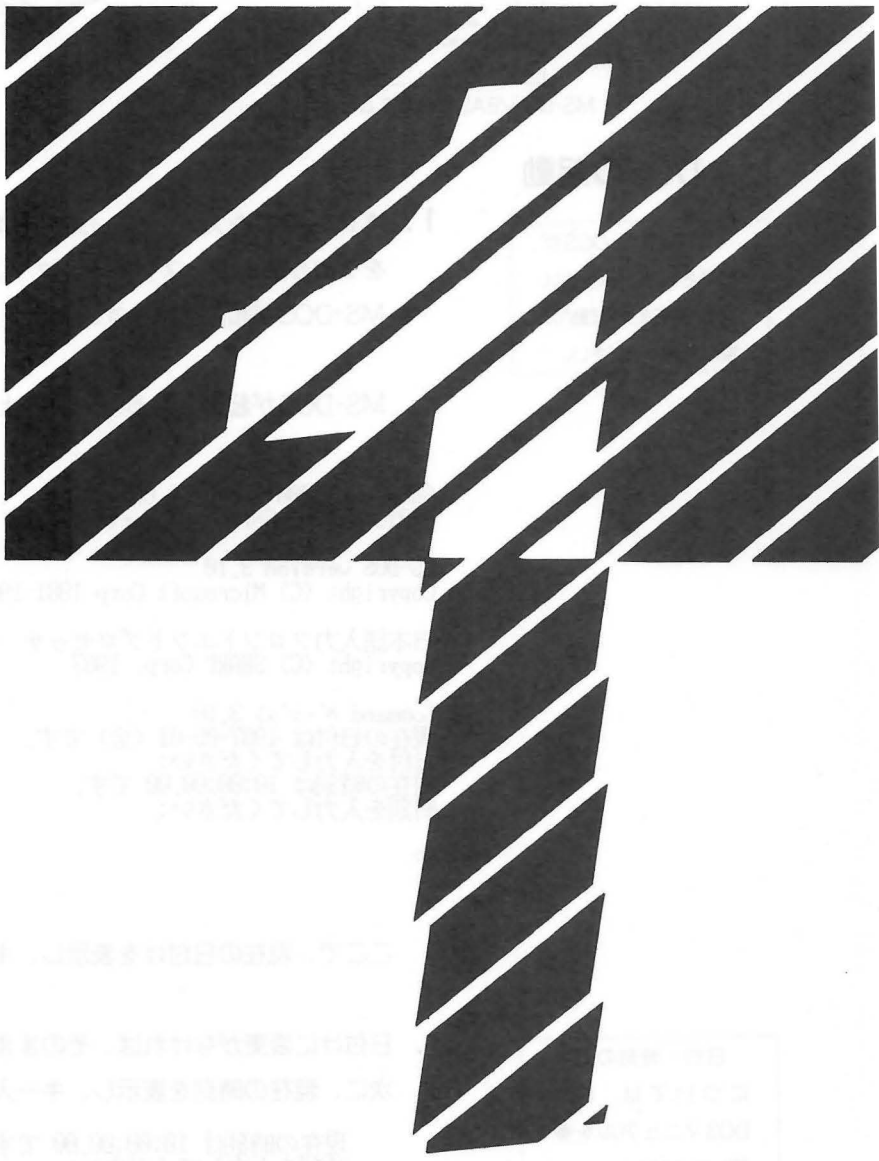
第1部をひととおりお読みいただいた方

索引(付録)を活用して、調べたい命令を探し出してください。
使用例(サンプル)のある命令は、実際にキー入力してみて、動作を確認することにより、ご自分でプログラムを組まれるときの参考としてください。

BASIC-M25で作成したプログラムをBASIC-M28で動作させたい方

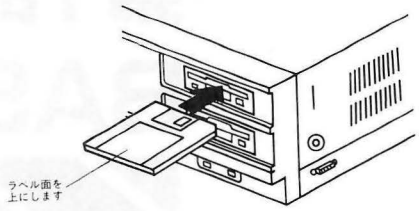
BASIC-M28、MZ-2511/21/31/20に標準装備されているBASIC-M25と、互換性を保つよう配慮されています。BASIC-M25で作成したプログラムをBASIC-M28で動作させるときの注意点については、**BASIC-M25との互換性**（付録）を参照してください。

第1部
BASICの概要



1 BASICの起動

MS-DOS/BASICシステムディスク



MS-DOSの起動

すでにMS-DOSが起動されているときは次の"BASICの起動"の項をお読みください。

1. 本体の電源を入れ、MS-DOS/BASIC-M28システムディスクをフロッピーディスクドライブ1にセットします。
MS-DOSが起動されます。☜コラム

2. MS-DOSが起動されると、次のような初期画面が表示されます。

```
SHARP MZ-2800 Personal Computer
6Z015 System Software  Version X.XX

MS-DOS version 3.10
Copyright (C) Microsoft Corp 1981-1986.  All rights reserved.

日本語入力フロントエンドプロセッサ  Version X.XX
Copyright (C) SHARP Corp. 1987

Command ﾊﾞｰｼﾞｮﾝ 3.10
現在の日付は 1987-05-01 (金) です.
日付を入力してください:
現在の時刻は 10:00:00.00 です.
時刻を入力してください:

A>
```

ここで、現在の日付けを表示し、キー入力を求めてきます。

日付・時刻の変更については、MS-DOSマニュアルを参照してください。

3. 日付けに変更がなければ、そのまま↵キーを押してください。
次に、現在の時刻を表示し、キー入力を求めてきます。

```
現在の時刻は 10:00:00.00 です.
時刻を入力してください:
```

4. 時刻に変更がなければ、そのまま↵キーを押してください。
☜コラム
次のように表示されます。

BASICの起動

プロンプト,カレントドライブについてくわしくはMS-DOSマニュアルを参照してください。

```
A>
```

ここで表示される**">"**マークを**"プロンプト"**と呼び、プロンプトの前に表示されている英記号は、BASICなど、MS-DOS上で動くソフトウェアが起動可能な装置名(カレントドライブ)を表しています。☜コラム

ASSIGNコマンドについてはMS-DOSマニュアルを参照してください。

したがって、上記のように表示されている場合、ソフトウェアが起動できる装置名はドライブAであることを示しています。

ドライブAは、MS-DOS上の**ASSIGN**コマンドで装置名の入れ替えが行われていない限り、フロッピーディスクドライブ1を示しています。☜コラム

BASIC-M28システムディスクが挿入されていることを確かめ、次のように入力してください。

5. A>BASIC [↵]

BASICが起動され、次のような初期画面が表示されます。

```
SHARP BASIC-M28 (6Z016)  Ver.x.xx      Copyright (C) 1987 by SHARP Corp.
ユーザーエリア： xxxxxxバイト

Ok
```

画面に**"Ok"**が表示され、カーソルが点滅する状態(キーの入力待ちの状態を表します。)になれば、BASICの起動は完了です。

ディスクのコピー

本機に同梱されているMS-DOS/BASIC-M28システムディスクの内容を別のフロッピーディスクにコピー(バックアップコピー)し、通常はそのコピーしたディスクの方をお使いになることをおすすめします。

これは、万一ディスクに傷や汚れが付くことによって読み書きが行えなくなったり、誤操作によってディスクの内容を消去してしまう場合に備えるためです。

DISKCOPYコマンド、**FORMAT**コマンドについては、MS-DOSマニュアルを参照してください。

バックアップコピーは、MS-DOSの**DISKCOPY**コマンドで行います。

この場合、バックアップコピーするディスクは先にMS-DOSの**FORMAT**コマンドでフォーマットしておいてください。☜コラム

なお、MS-DOSのコマンドを、BASICが起動された後に実行する場合は、system↵と入力して、再びMS-DOSのプロンプト(>)が表示されてから行うようにしてください。

```
system↵
A>
```

再びBASICを起動するときは、カレントドライブにMS-DOS／BASICシステムディスクがあることを確かめて、BASIC↵と入力してください。

BASICの起動時オプション

BASIC起動時にオプションを指定することができます。指定は次の形式で行います。

A>BASIC<オプション>↵

次のオプションが用意されています。

<オプション>　　機　能

／G … グラフィック表示メモリ領域がクリアされません。

／D … 起動画面で起動メッセージが表示されません。

／Q … BASICのプログラム終了時や、**end**文実行時、自動的に**system**コマンドを実行してMS-DOSに制御を移します。

／N … **option float on**の状態で起動します。
(指定の無い場合は**option float off**の状態で起動します。)

／T＜自動実行プログラム〉

…BASIC起動時に＜自動実行プログラム＞で指定されたプログラムが実行されます。
＜自動実行プログラム＞は最大62バイトの大きさまで可能です。

BASICの自動実行

AUTOEXEC. BATについてはMS-DOSマニュアルを参照してください。

BASICを**AUTOEXEC.BAT**ファイル内に登録すると、MS-DOSを起動した時に、自動的にBASICも起動されるようになります。☞コラム

起動時の初期設定

起動時の状態

BASICの起動時には次の初期化が行われます。

- **new**コマンドの実行により行われる初期化。
- グラフィックモードの初期化。(**init "CRT2: 640,400,16"**)
- 文字画面の初期化。(80桁25行ラインスクロール)
  (**init "CRT 1:,,,0"**)
- 音楽機能の初期化 (**play init**)
- タイマの初期化 (**time＝0**)
- 乱数の初期化 (**randomize 6**)
- ファンクションキーの初期化

(注)予約語の表示は、小文字になります。(**option list lcace**)

2 編集と実行

本BASICを十分に使いこなしていただくためには、基本的な約束ごとを覚えていただかなければなりません。ここでは、BASICの基本仕様から、プログラム作成時の編集操作方法、またその実行方法までを説明しています。

実行モード

BASICが起動されると、"Ok"が表示され、命令待ちになります。この状態で命令を実行させる方法として、直接実行モードと間接実行モードがあります。

直接実行モード

コマンド, ステートメントについては「命令の種類」を参照してください。

ダイレクトモードとも呼ばれ、コマンドやステートメントを入力し↵キーを押すと直ちに実行されます。☜コラム

このモードでは、直ちに実行が行われますが、実行手順の保存は行いません。

例:
```
print"ABC" ↵
ABC
Ok
```

間接実行モード

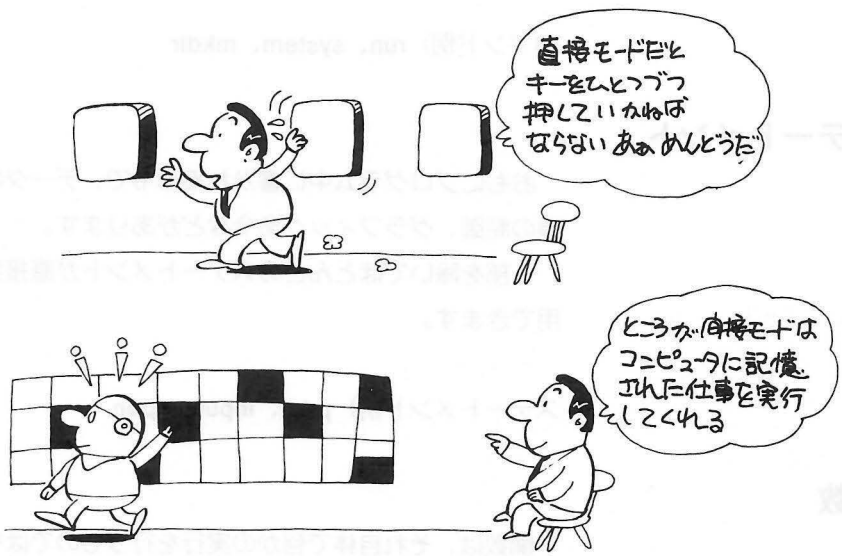
プログラムモードとも呼ばれ、BASICはまずメモリ内へ一連の命令を実行手順どおりに記憶した後に、これを解析して、実行します。この一連の命令が実行手順どおりに記憶されているものをプログラムと呼びます。

プログラムは、実行の順番を示す行番号のあとに命令を入力し、↵キーを押すことによって、命令が行番号の順番で記憶されます。

runコマンドまたは、**goto**文や**gosub**文を直接実行することにより、プログラムの実行が開始されます。

例1:
```
10 let A=123*456
20 print A
30 end
run
 56088
Ok
```

例2:
```
100 print "BASIC"
goto 100
BASIC
BASIC
Ok
```



命令の種類

BASICの命令は機能的に次のように分けることができます。

コマンド

プログラムの編集や入出力、実行にかかわる命令で、おもに直接実行モードで実行されます。

直接実行モード以外にプログラム中で実行可能な命令もあります。

コマンド例）**run、system、mkdir**

ステートメント

おもにプログラム中に書かれる命令で、データの入出力や実行手順の制御、グラフィック命令などがあります。

一部を除いてほとんどのステートメントが直接実行モードでも使用できます。

ステートメント例）**print、input、open**

関数

関数は、それ自体で何かの実行を行うものではなく、与えられた引数（関数で処理を行う対象となったり、機能を指定したりするためのデータ）により、数値演算や、文字列の加工を行い結果を返すもので、それ自体が値を持ちます。

また、関数の中には、入出力装置に対して情報を入出力するための入出力関数もあります。

関数例）**abs、left$、jis$**

システム変数

システム変数については「3. BASICで扱うデータ」の項を参照してください。

システム変数は、関数と同様にそれ自身が値を持ちます。関数のように引数はありませんが、目的に応じて内容を設定できるものもあります。

システム変数には日付・時刻やBASICのバージョン名、BASICの実行にかかわる情報などがあります。

システム変数例）**time$、dtl、err**

プログラムの書式

行の最初に番号を付けて入力するとプログラムとして扱われます。プログラムの１行の書式は次のようになります。

＜行番号＞〔＜ラベル＞：〕＜文＞〔：＜文＞〕…

〔 〕内は省略可能なことを示します。

例: 10 load "FILE1"

```
500 *HYOJI : cls 3 : print tab(10);"Please input your name."
 ↑     ↑       ↑          ↑
〈行番号〉〈ラベル〉〈文〉       〈文〉
```

行番号

行番号は、プログラム内の位置を示し**1**から**65535**までの整数値が使えます。

プログラムの各行の最初には、必ず行番号が必要です。

プログラムの各行を行番号の順に入力しなくても、コンピュータ内部では、行番号の順に記憶されています。

ラベル

ラベルは**goto, gosub**文や編集命令などで行指定をするときに、行番号の代わりに名前（ラベル名）で指定するものです。

行番号指定の場合、**renum**コマンドで行番号を変更すると、プログラム中にすでに書かれている行番号は自動的に変更されるので、新たに行番号を参照するときは、その行番号の変化を覚えておく必要があります。

一方、ラベル指定の場合は、行番号が変化しても、ラベル名は変化しないので、同じところを参照できます。

例）

```
10 rem test
15 input X
18 goto *LOOP
20 end
21 *LOOP
22 for I=1 to X
23 print I,"SQUARE=";I*I
24 next
```

renum ↵

list ↵

⇨

```
list
  10 rem test
  20 input X
  30 goto *LOOP
  40 end
  50 *LOOP
  60 for I=1 to X
  70 print I,"SQUARE=";I*I
  80 next
Ok
```

半角・全角文字については「BASICで使用する文字」の項を参照してください。

ラベル名は、最初の文字をアスタリスク（＊）にして、2文字目は半角文字のアルファベットやカタカナまたは漢字（文字コードがシフトJISコードで**&H889F**以降の全角文字）で指定します。3文字目以降は全角文字（文字コードがシフトJISコードで**&H824F**以降のもの）または半角文字のアルファベットやカタカナ、数字、アンダーバー（＿）が使え、最初のアスタリスクを除いて半角文字相当で200文字までの任意の長さで指定できます。また、予約語のスペルをラベルの中に使うこともできます。

例）**＊LOOP　＊PRINT＿OUT　＊開始**

ラベルを入れる位置は行番号の次で、あとに文が続く場合はコロン（：）で句切ります。ラベルを文の途中に入れることはできません。

文

文はコマンドまたはステートメントで構成され、一行の中には必ず文を必要とします。ただし、ラベルのみの行にすることは可能です。また、マルチステートメントとして、文と文をコロン（：）で句切ることにより、同一行に複数の文を続けて書くことができます。

行の長さ

画面上の表示では1行が40または80文字ですが、BASICでは論理的な行として、行番号を含めて最大**255**文字まで可能です。（画面上では複数の行にわたって表示します。）ただし、255文字未満の場合でも入力時に**line length error**が発生する場合（**data**文など）がありますので、その場合は2行に分割して入力してください。

プログラム中のスペース

予約語については「予約語と省略形」を、半角文字、全角文字については「BASICで使用する文字」を、変数、定数、演算・比較記号については「3. BASICで扱うデータ」の項を参照してください。

プログラム中でスペース(空白)を使用する場合は、いくつかの注意点があります。

予約語に続いて半角文字の数字、アルファベットやカタカナまたは全角文字がくる場合は、変数名と区別できなくなるので、必ず1個以上のスペースをあけて入力してください。ただし、いくつのスペースをあけてもBASIC内部では、1個のスペースとして扱われます。(リスト表示では1個分のスペースになります。)これは、予約語の前(左側)についても同様のことが起こります。

演算記号、比較記号、句切記号などの左右にスペースを入れた場合は、BASICでは詰めて扱われます。

行番号に続いて、スペースを入れずに文を書いた場合は、BASICでは1個のスペースを行番号の次に入れます。

次のような場合はスペースが入力したとおりの個数として扱われます。

- 行番号から文が始まるまでの間。
- マルチステートメントの句切記号であるコロン(:)の左右。
- rem文およびアポストロフィ(')以後のコメント部分。
- data文のデータ部分。
- ダブルクォーテーション(")で囲まれた文字定数。

行番号は右詰め表示をしますので、5桁に満たない左側の部分はスペースになります。

例) 入力とlistコマンドによる出力

```
10for   a=  1  to  200  :  next
list 10
   10 for A=1 to 200  :  next
                      └──────────そのままの個数


200   if a$ = "a A"then   99     elseprint a
list 200
  200   if A$="a A" then 99 ELSEPRINT A
                            └──────予約語になりません
```

BASICで使用する文字

全角文字と半角文字

全角文字の入力については「4. 漢字の扱いについて」の項を参照してください。

BASICで使用する文字は半角文字と全角文字があります。表示上では全角文字は半角文字の倍の幅があります。

なお、BASIC上の命令(予約語)は半角文字の英数字、記号で扱われ、全角文字の英数字、記号で書くことはできません。

BASIC内部では、使用する文字は文字コードで扱われ、半角文字は1バイト、全角文字は2バイトのコードで表されます。

半角文字
- アルファベット・数字・英記号
- カタカナ・カナ記号
- グラフィック文字

全角文字
- 漢字
- ひらがな・カタカナ・記号・英数字

- なお、半角文字のカタカナの場合、濁点(゛)や半濁点(゜)も1文字(バイト)となります。

アルファベットの大文字と小文字

配列・定義関数については「3. BASICで扱うデータ」の項を参照してください。

BASICの予約語や変数名、配列名、定義関数名、ラベル名では、半角文字のアルファベットを使用した場合、大小字と小文字は同じものとして扱われますので、入力時は、どちらを用いてもかまいません。

ただし、**list**コマンドなどによってプログラムを出力する場合は大文字と小文字のいずれかに統一されます。

予約語は、大文字で出力するか、小文字で出力するかの選択を、**option list**文で行うことができ、初期状態は小文字で出力するようになっています。

変数名、配列名、定義関数名、ラベル名では、半角文字のアルファベットはすべて大文字で出力します。

特別な記号

BASICは半角文字の記号に特別な意味を持たせているものがあります。

	空白	原則として無視されますが、予約語の前後に句切りとして必要な場合があります。
＝	イコール	代入、等号
＋	プラス	加算、正符号、文字列の結合
－	マイナス	減算、負符号、行や座標の範囲指定(ハイフンとして)
＊	アスタリスク	乗算、ラベルの接頭文字
／	スラッシュ	除算
＾	山記号	べき乗算
￥	円記号	整数除算
％	パーセント	整数型
！	感嘆符	単精度実数型
＃	シャープ記号	倍精度実数型
＄	ダラー	文字列型
＆	アンパサンド	2,8,16進法やJIS漢字コード、区点漢字コード表記の接頭文字
’	アポストロフィ	注釈文
？	クエスチョン	print(予約語)の代用
，	カンマ	データの句切り
；	セミコロン	データの句切り
”	ダブルクォーテーション	文字定数の引用符
．	ピリオド	小数点、行番号（編集時や実行中断時）予約語の省略
：	コロン	マルチステートメントの句切り
＿	アンダーバー	変数名、ラベル名の中に使用可
（）	括弧	演算時に括弧内を優先、座標指定
＜ ＞	大小記号	比較演算子

予約語と省略形

予約語

予約語とは、BASICでコマンドやステートメント、関数として使用される単語のことです。

変数名や配列名に予約語を使うことはできません。（ただし、予約語のスペルを含むものは可能です。）

予約語は半角文字の英数字、記号で表します。入力時は大文字、小文字のいずれでも可能ですが、**list**コマンドなどによる出力時は、大文字、小文字のいずれかに統一されます。リスト表示の予約語の大文字と小文字の選択は**option list**文によって行われ、BASICでの初期状態は小文字になっています。

予約語に続いて半角文字の英数字やカタカナ、アンダーバー（_）、全角文字がある場合には変数名と区別できません。このような場合、予約語の後にスペースを挿入してください。

なお、予約語を書くときは、１つの予約語の途中にスペースを入れることはできません。

例）× cls 3　　○ cls　3
　　× remmenu　○ rem　menu
　　× re num　　○ renum

省略形

BASIC上でよく使用する予約語や、長いスペル（つづり）の予約語に対して、すべてのスペルを書かずに省略できる機能があります。

省略形は予約語のスペルの一部分を書き、省略していることを表すためにピリオド(.)を付けます。

たとえば**print**の省略形は**p.**となります。

プログラム入力時に省略形で入力してもリスト表示を行うと、入力された予約語がフルスペル（省略しない形）に直されます。

省略は、最も短い省略可能な形だけでなくピリオド（.）を付けることにより省略範囲を広げることもできます。

たとえば**list**の最短の省略形は**l.**ですが、**li.**や**lis.**も可能です。

次に予約語とその最短の省略形の表を示します。

予約語	省略形	予約語	省略形	予約語	省略形	予約語	省略形
<a>		com	com	<f>		<k>	
abs	ab.	common	com.	fac	fa.	kacnv$	ka.
add	ad.	connect	conn.	filed	fie.	kanji	kan.
akcnv$	ak.	console	cons.	files	fi.	kext$	kex.
all	al.	cont	c.	fix	fix	key	ke.
and	an.	copy	cop.	float	fl.	kill	kil.
angle	ang.	cos	cos	for	f.	kinput	k.
apss	ap.	crev	cr.	fpos	fpo.	kinstr	kins.
as	as	csng	cs.	fpset	fp.	klen	kle.
asc	asc	csrlin	csr.	frac	fr.	klist	kl.
ascchr$	asc.	cvd	cvd	fre	fre	kmid$	kmid$
atn	at.	cvi	cv.			kmode	km.
attr$	att.	cvs	cvs	<g>		knj$	knj$
auto	a.			get	ge.	kpos	kp.
		<d>		gload	gl.	ktn$	kt.
<b>		data	da.	go sub	go s.	ktype	kty.
base	ba.	date$	date$	go to	go		
beep	b.	day$	day$	gosub	gos.	<l>	
bin$	bi.	dbl	db.	goto	g.	lcase	lc.
bload	bl.	def	def	gsave	gs.	left$	lef.
boot	bo.	deg	deg			len	len
bsave	bs.	degrees	deg.	<h>		let	le.
		delete	d.	hcopy	h.	lfiles	lfi.
<c>		devf	devf	help	hel.	lfo	lf.
call	ca.	devi$	dev.	hex$	he.	line	lin.
cblock	cb.	devo$	devo$	hexchr$	hex.	list	l.
cdbl	cd.	dim	di.			llist	ll.
cflash	cf.	dtl	dt.	<i>		load	lo.
cgen	cg.			if	if	loc	loc
cgpat$	cgp.	<e>		imp	im.	locate	loc.
child	chi.	edit	e.	init	ini.	lof	lof
chain	cha.	else	el.	inkey$	ink.	log	log
chdir	ch.	end	en.	inp	inp	log10	log10
chr$	chr$	environ	env.	input	i.	lpos	lpos
cint	cin.	environ$	environ$	input$	input$	lpout	lp.
circle	ci.	eof	eo.	instr	ins.	lprint	lpr. l?
clear	cle.	eqv	eq.	int	int	lsearch	lsea.
click	cli.	erase	er.	interval	inte.	lset	ls.
close	clo.	erl	erl			ltron	lt.
clr	clr	err	err	<j>			
cls	cl.	error	err.	jis$	j.	<m>	
cmt	cm.	exp	ex.			map	map
color	col.					max	ma.

予約語	省略形
merge	m.
mid$	mid$
mirror$	mi.
mkd$	mkd$
mkdir	mk.
mki$	mki$
mks$	mks$
mod	mod
mon	mo.
mouse	mou.
move	mov.
<n>	
name	na.
new	new
next	n.
not	no.
<o>	
oct$	oc.
off	of.
on	o.
open	op.
option	opt.
or	or
out	ou.
<p>	
pai	pai
paint	pa.
pattern	pat.
pause	pau.
peek	pee.
pen	pe.
play	pl.
point	poi.
poke	po.
poly	pol.
pos	pos
preset	pre.
print	p. ?
priority	prio.
pset	ps.
put	pu.
pwd$	pw.
<r>	
rad	rad
radians	rad.
randomize	ra.
read	rea.
rem	rem
renum	ren.
repeat	rep.
replace	repl.
restore	res.
resume	resu.
return	re.
right$	ri.
rmdir	rm.
rnd	rn.
roll	ro.
rset	rs.
run	r.
<s>	
save	sa.
scr	scr
screen	sc.
scrn$	scrn$
search	sea.
seg	seg
segptr	seg.
set	set
sgn	sg.
sin	si.
sng	sn.
sound	so.
space$	spa.
spc	sp.
sqr	sq.
step	ste.
stick	sti.
stop	s.
str	str
str$	str$
strig	str.
string$	strin.
sub	su.
sum	sum
swap	sw.
symbol	sym.
sysid$	sysi.
system	sy.
<t>	
tab	ta.
tan	tan
tel	tel
term	te.
then	th.
time	time
time$	ti.
to	to
tone	ton.
troff	trof.
tron	t.
<u>	
ucase	uc.
until	u.
using	us.
<v>	
val	va.
varptr	var.
verify	ve.
view	vi.
<w>	
wait	wa.
wend	we.
while	wh.
width	wi.
window	win.
write	w.
<x>	
xor	x.

プログラムの編集

プログラムは常にメモリ上に記憶されていて、１行単位で管理されます。ここではプログラムの入力、変更などの基本的な操作について説明します。

プログラムの入力

行番号の付いた１行が入力されるとプログラムの１行として記憶されます。続いて別の行番号を付けて入力すると、行番号を比較して、番号順に並べてメモリに記憶します。この行番号の順番が実行する順番になります。

続けて、プログラムを入力する場合には、自動的に行番号を発生させる**auto**コマンドや**edit**コマンドがあります。また入力されたプログラム（リスト）を確認するために**list**コマンドが用意されています。

プログラムの変更

すでに記憶されているプログラムの変更も行単位で行われます。すでに記憶されているプログラムと同じ行番号で入力すると、以前に記憶されている方の行が消去され、新たに入力された行の内容になります。

変更する内容がその行の一部分の場合は、あらかじめ目的の行を画面に表示させ、カーソル移動により表示を書き換え、[↵] キーを入力することにより変更ができます。この機能をスクリーンエディット機能と呼びます。

例）
```
list
   10 rem test
   20 init "CRT1:,,,0":klist 0
   30 end
Ok
```
↑ カーソル位置

↓ 2[↵]と入力します。

再びリスト表示させると

```
list
   10 rem test
   20 init "CRT1:,,,0":klist 2
   30 end
Ok
```

プログラム全体をスクリーンエディットする命令として、**edit**コマンドが用意されています。

プログラムの消去

特定の行を消去するときは、その行番号のみを入力します。入力が行われるとすでに記憶されているプログラム中の指定行番号の内容が消去されます。

例）
```
list
  10 print "ABC"
  20 print
  30 end
Ok
```
20 ↵ ⇨
```
list
  10 print "ABC"
  30 end
Ok
```

行番号の範囲を指定して消去する命令として、**delete**コマンドがあります。また、すべてのプログラムを消去する命令として**new**コマンドが用意されています。

ピリオド(.)の特別機能

直接実行モードで**list**や**edit**などの行番号を指定する編集コマンドにおいて、行番号の指定の代わりにピリオド（.）を使うことができます。

たとえば、次のような場合**list .**とすると目的の行を確認できます。

- プログラムの実行中に、エラーが発生して命令待ちに戻ったときのエラー発生行。
- **stop**文の実行および シフト +停止 キーの操作などによりプログラムの実行を一時中断して命令待ち状態に戻ったときに実行していた行。
- プログラムの編集が行われた場合の最後の入力行。

例）
```
list
   10 print "ABC"
   20 PINT "DEF"
   30 end
Ok
run
ABC
syntax error in 20
Ok
list.
   20 PINT "DEF"
Ok
```

本書中では、キーの入力のしかたとして次の略式を使っています。

[シフト]+[停止] :左側に示されたキーを押しながら右側のキーを押すことを意味します。

└押しながら

編集のためのキー操作

ここでは編集のための [コントロール] キーや [シフト] キーを使った操作を中心に説明します。

なお、この他に指定した行を中心にプログラムを表示する、**edit** コマンドがあります。

カーソルの移動

[↑] [↓] [→] [←]　矢印の方向にカーソルが１文字分移動します。

[コントロール] ＋ [^]　[↑] キーと同じです。

[コントロール] ＋ [_]　[↓] キーと同じです。

[コントロール] ＋ [¥]　[→] キーと同じです。

[コントロール] ＋ []]　[←] キーと同じです。

[コントロール] ＋ [F]
[シフト] ＋ [→]　右方向へ単語の先頭まで移動します。

[コントロール] ＋ [B]
[シフト] ＋ [←]　左方向へ単語の先頭まで移動します。

単語移動は英数字や全角文字・カナの並びを1つの単語と見て、半角文字のスペースや記号が句切りとみなされます。

[クリア/ホーム]
[コントロール] ＋ [K]　画面の左上（カーソルホームポジション）に移動します。

[コントロール] ＋ [→]　行末へ移動します。

[コントロール] ＋ [←]　行頭へ移動します。

[コントロール] ＋ [V]　カーソル位置で行を分けます。カーソル位置から右側の文字を次の行へ送り出します。

[コントロール] ＋ [↑]　画面の左上に移動します。[クリア/ホーム] キーと同じです。

[シフト] ＋ [↑]　画面上にプログラムリストが表示されている場合には、それ以前のプログラムリストを呼び戻して表示します。プログラムリストがない場合には、画面全体をスクロールダウンします。（**edit**コマンドによるエディットモード時には画面半ページをスクロールダ

ウンします。）

コントロール ＋ ↓　画面の左下に移動します。

シフト ＋ ↓　画面上にプログラムリストが表示されている場合には、その直後のプログラムリストを呼び出して表示します。プログラムリストがない場合には、画面全体をスクロールアップします。
（エディットモード時は画面半ページをスクロールアップします。）

タブ　現在のカーソル位置の右側のタブ設定位置に移動します。タブの設定はコントロール＋Tキーで行います。BASICの起動時は8文字ごとのタブ設定になっています。

コントロール ＋ I　タブキーと同じです。

タブの設定と解除

コントロール ＋ T　現在のカーソル位置にタブを設定します。

コントロール ＋ Y　現在のカーソル位置にタブが設定されている場合、設定を解除します。

文字の挿入

シフト＋挿入/抹消キーまたはコントロール＋Rキー
カーソル位置より行末までの文字を1文字分右にずらし、カーソル位置に空白を挿入します。

コントロール ＋ シフト ＋ 挿入/抹消 キーまたは
コントロール ＋ Aキー
インサートモードになりカーソルの点滅が速くなります。このときに文字を入力すると、カーソル位置に文字を挿入します。↵キーを押して入力を完了するか、カーソルキーを押すことにより、カーソルの点滅が元に戻り、モードの解除が行われます。

インサートモードのときに、次のキーが特別な意味を持ちます。

コントロール＋タブ　カーソル位置よりタブ位置まで空白を

挿入します。

挿入処理において文字が行末まで詰まっているときは、次の行以降をスクロールダウンさせます。

文字の削除

挿入/抹消	カーソル位置の１文字を削除します。カーソルの右側より行末までの文字は左に１文字分移動します。
後退 コントロール ＋ H	カーソルの左側の１文字を削除します。カーソル位置より行末までの文字は左に１文字分移動します。
コントロール＋後退	行頭よりカーソル位置の左側の文字を削除します。カーソル位置より行末までの文字は行頭へ移動します。

文字の消去

コントロール＋挿入/抹消 コントロール ＋ E	カーソル位置より行末までを消去します。
コントロール ＋ X	カーソルのある行をすべて消去します。カーソル位置は、行頭になります。
コントロール ＋ Z	カーソル位置より画面の終わり(右下)までを消去します。
シフト＋クリア/ホーム コントロール ＋ L	全画面を消去してカーソルを左上に移動します。
コントロール＋タブ	カーソル位置より右側のタブ位置までを消去します。操作後、カーソルはタブ位置に移動します。

行の挿入

コントロール ＋ N	現在カーソルのある行から上の行をスクロールアップします。
コントロール ＋ O	現在カーソルのある行から下の行をスクロールダウンします。

いずれもカーソルのある行は空白が挿入され、カーソル位置は変化しません。

コンソールエリア

console文によってコンソールエリアが設定されている場合、画面のスクロールやコントロールコードの適用範囲は、コンソールエリア内のみに限られます。

ただし、コンソールエリア外にカーソルを移動させて、コントロールコードを実行すると、画面全体がコントロールコードの適用範囲となります。

文字のコピー機能

画面に表示されている文字を別の位置にコピーすることができます。

まず、コピー先(コピーした文字を表示したい位置)にカーソルを移動し[コピー]キーを押します。すると、カーソルが青色になり、点滅が止まります。次に、コピーしたい文字の位置にカーソルを移動します。(カーソルは点滅しています)ここがコピー元になります。あとは[↵]キーを押すごとに1文字づつ、カーソルが点滅している位置の文字が、青色カーソルの位置にコピーされます。

例) 文字のコピー機能 ■ ←[コピー]キー操作
(カーソルが青色になります。)

文字のコピー機能 ■ ←カーソルキーによるカーソル移動

文字のコピー機能 文 ■ ←[↵]キーを押すごとに1文字ずつコピーされます。

コピー機能を解除するときは、もう一度[コピー]キーを押します。

文字のコピーを複数行にわたって行った場合、コピー先の各行は連続した行として扱われます。

ハードコピー

文字画面、グラフィック画面については「7. 表示モード」、プリンタ印字については「15. プリンタ制御機能」の項を参照してください。

[コピー]キーを使用して、プリンタへのハードコピー(画面コピー)を行うこともできます。

ハードコピーを行うには、次のキーを操作します。

[コントロール]+[コピー]

SETUPコマンドについてはMS-DOSマニュアルを参照してください。

なお、ハードコピーを文字画面、あるいはグラフィック画面のみで行うか、またはその両方で行うかといった選択はMS-DOSのSETUPコマンドを使用します。

また、ハードコピーは文字画面の40桁モード時、文字フォントの8×8ドット時、グラフィック画面の640×200ドットモード時には行えません。

そのほかの制御操作

[コントロール] + [W]　次の行と接続して論理的につながった行とします。

[シフト]+[停止]　プログラムの実行を中止します。

[停止]　プログラム実行を一時停止させます。任意の文字キーを押すと実行を再開します。

[コントロール] + [G]　ベル音を鳴らします。

[コントロール]+[シフト]+[クリア/ホーム]　先行入力のキーバッファを消去します。

[コントロール] + [M]　[↵]キーと同じ動作をします。

[コントロール] + [D]　画面表示の初期化や音楽演奏の中止を行います。

(初期化される命令)

console console@ window view view@ screen color cgen crev cflash priority color=

プログラムの実行

メモリ上に記憶されているプログラムの実行は**run**コマンドで行います。

プログラムの実行は行番号の順に行われますが、**goto**文、**gosub**文、**if〜then**文などの実行により順番を変えたり、**for〜next**文、**repeat〜until**文、**while〜wend**文などによって繰り返し実行(ループ)させることもできます。

実行が終了されるのは**end**文を実行したとき、またはプログラムの最後で次に実行する行がなくなった場合です。また、[シフト]+[停止]キーの操作や**stop**文による中断の場合は**cont**コマンドによってプログラムの実行を再開することができます。

プログラムの実行中に[停止]キーを押すことにより、実行を一時停止させることができます。一時停止の状態より実行を再開させる場合は、任意の文字キーを押します。

プログラムの実行状態を確認するための**tron**コマンドやブレークポイントを設定する**stop**コマンド、またステップ実行をするための**step**コマンドなどのデバッグコマンドも用意されています。◀コラム

プログラムが中断され、**cont**コマンドや**stop**コマンドで中断した所からのプログラム再開が可能なときは、**Ok**のあとにピリオド(.)を表示します。

プログラムの中にある誤りをバグ(BUG·虫)といいます。このバグを発見して修正する作業をデバッグと呼びます。

例)
```
list
    10 print "*";
    20 pause 10 : goto 10
    30 end
Ok.
run
*****
*Break in 20          ←[シフト]+[停止]キー操作
Ok.                   ←ピリオド(.)表示
cont
**
```

メインルーチンとサブルーチン

プログラムが大きく、複雑になってくると、プログラム中に同じ内容の文が何度も現れるようになります。

このとき、これらの文をひとまとまりにして、1つの小さなプログラムとし、必要なときにこのプログラムを呼び出せるようにしておけば、プログラム全体が見やすくなります。

この小さなプログラムをサブルーチンと呼び、サブルーチン以外のプログラム(プログラムの主な流れを形成する部分)をメインルーチンと呼びます。

メインルーチンからサブルーチンへ移るときは、**gosub**文を使用し、サブルーチンからメインルーチンへ戻るときは、**return**文を使用します。

例)

```
:
100 gosub 1000
:
500 end
```

メインルーチン

```
1000 *sub
:
1100 return
```

サブルーチン

3 BASICで扱うデータ

BASICで扱われるデータは数値型データと文字列型(ストリング)データがあり、数値型データには整数型,単精度実数型,倍精度実数型の種類があります。

バイトとは、データの大きさを表す単位です。1バイトは8ビットで構成されます。

・整数型データ

-32768から32767の範囲の整数で、BASIC内部では、2バイトのデータとして扱われています。☜コラム

・単精度実数型データ

0および$\pm 2.9387359 \times 10^{-39}$から$\pm 1.7014118 \times 10^{38}$までの範囲の、有効桁8桁の浮動小数点方式です。BASIC内部では、5バイトのデータとして扱われています。

以後、単精度型と略して呼ぶ場合があります。

・倍精度実数型データ

0および$\pm 2.938735877055719 \times 10^{-39}$から$\pm 1.701411834604692 \times 10^{38}$までの範囲の、有効桁16桁の浮動小数点方式です。BASIC内部では、8バイトのデータとして扱われています。

以後、倍精度型と略して呼ぶ場合があります。

・文字列型データ

0から255文字までの長さの文字の並んだものとして扱われます。内部では、半角文字は1バイト、全角文字は2バイトの文字コードの並びとして扱われ、最大の長さは255バイトです。

注)ここで表現しているデータのバイト数は、データのみの数です。実際に変数や配列に代入した場合は、変数名や配列名の管理用のデータ分だけバイト数が増えます。

定数

定数とは、プログラム実行中に内容が変化しないデータのことです。

整数型

数値の後に%記号を付けます。

入力時-32768から32767までの定数は自動的に整数型とみなされます。

例）１２３４５％　－１２３％　１１５６　－１５００

整数型は10進数のほかに2、8、16進数での表現ができます。

２進数　……　&Bを数の先頭に付けます。

&Ｂ０～&Ｂ１１１１１１１１１１１１１１１１

８進数　……　&Oを数の先頭に付けます。

&Ｏ０～&Ｏ１７７７７７

16進数　……　&Hを数の先頭に付けます。

&Ｈ０～&ＨＦＦＦＦ

最上位ビットは符号になりますので1の場合は負の値になります。たとえば、16進数の**&H8000～&HFFFF**は**-32768～-1**になります。

通常の表示は10進数で行われますが、**BIN$, OCT$, HEX$**の関数を使用すると、それぞれ2、8、16進数で表すことができます。

漢字対応の特別な定数として次のものがあります。

いずれも内部ではシフトJISに変換して扱われます。

JISコード　……　**&J**を先頭に付けて4桁の16進数で表します。

&J2121～&J7E7E　（範囲をこえた場合はエラーになります。）

例）print　chr$(&J3021),hex$(&J3021)

亜　889F　……　JISの**&H3021**はシフトJISでは**&H889F**になります。

区点コード　……　**&K**を先頭に付けて区と点をそれぞれ2桁の10進数で表します。

&K0101～&K9494　（範囲をこえた場合エラーになります。）

例）print　chr$(&K1601),hex$(&K1601)

亜　889F　……　16区、1点はシフトJISでは**&H889F**になります。

単精度実数型

数の後に！記号を付けます。

整数型の範囲をこえる8桁以内の整数、または小数点付きで有効桁が8桁以内の数は、自動的に単精度実数型とみなされます。

指数記号は、Eが用いられます。

例）１２３！　１２３４５６７　１０．５

１４３Ｅ＋０３　……　$143\times10^{3}=143000$

１１２８Ｅ－０５　……　$1128\times10^{-5}=0.01128$

倍精度実数型

数の後に＃記号を付けます。

有効桁が9桁以上の数は、自動的に倍精度実数型とみなされます。

指数記号は、Ｄが用いられます。

例）１２３＃　１２３４５６７８９　１．２３Ｄ＋１２

文字列型

引用符としてダブルクォーテーション(")記号で囲みます。

例）"１２３"　"ＢＡＳＩＣ"　""

"記号を続けて書くと文字長が０のＮｕｌｌ(空文字)文字列になります。(Null文字列＝ヌルストリング)

変数

変数とは、プログラム実行中に内容が変化するデータのことです。

プログラム中でデータを扱う場合、そのデータの内容がいろいろと変化するときには変数を使います。

変数名

変数を区別するための名前が変数名になります。

名前は、最初の文字を半角文字のアルファベットやカタカナまたは漢字（文字コードがシフトＪＩＳコードで**&H889F**以降の全角文字）で指定します。2文字目以降は半角文字のアルファベットやカタカナ、数字、アンダーバー（＿）および全角文字（文字コードがシフトＪＩＳコードで**&H82F4**以降のもの）を使って指定でき、最大半角文字相当で200文字までの任意の長さで指定できます。また、最後に型を示す記号（**%, !, #, $**）を付けます。

予約語（コマンド、ステートメントのスペル）と同じ変数名は使用できませんが、予約語のスペルを含むものは使用することができます。

例）ＡＢＣＤ％　ＡＢＣＤ！　ＡＢＣＤ＃　ＡＢＣＤ$

…… 同じ名前を使用していますが型が異なりますので別のものとして扱われます。

変数の型

通常は変数名で型指定を省略した場合、変数名が半角のアルファベットで始まるものは単精度実数型、変数名が漢字または半角文字のカタカナで始まるものは倍精度実数型として扱われます。

データの型を指定する場合は次の記号を付けます。

%	整数型
!	単精度実数型
#	倍精度実数型
$	文字列型

変数名が半角文字のアルファベットで始まる変数の型指定を省略したとき、自動的に決定される型を指定する命令として、**def int**、**def sng**、**def dbl**、**def str**の命令が用意されています。

配列

変数が多くなると、プログラムも複雑になってきます。

このように、データ数が多くなった場合には、配列を使用すると便利です。

配列名と添字

配列は、変数に番号を付けて並べたものです。

配列に付ける名前を配列名といい、配列名に使用できる文字や長さは一般の変数と同じです。

配列で使用する一連の番号は( )でくくり、その配列上の位置を示します。これを添字といいます。

配列データを参照するときは、この配列名と添字を指定します。なお、配列の１つ１つのデータを配列変数といいます。

配列データは次のように書きます。

配列名(添字)

たとえば、Aという配列名で5個のデータを格納する場合は次のようになります。

A(5)

A(1)	A(2)	A(3)	A(4)	A(5)

(**option base 1** を実行して、添字の最小値を１とした場合)

さらに、添字はカンマで句切ってA(3,3)のように並べることができます。

この例では、3×3＝9個のデータを格納できます。

A(3,3)

A(1,1)	A(1,2)	A(1,3)
A(2,1)	A(2,2)	A(2,3)
A(3,1)	A(3,2)	A(3,3)

(**option base 1** を実行して、添字の最小値を１とした場合)

このように、添字が2個ある配列を2次元配列といいます。

添字が3個ある場合には3次元配列、n個ある場合はn次元配列といいます。

OPTION BASE

配列を使用する場合には、まず配列の添字の最小値を設定してお

く必要があります。

配列の添字は最小値は0か1です。

この値はoption base文で設定します。

BASIC起動時には、添字の最小値は0になっており、option base 1を実行すると添字の最小値を1に設定することができます。

option base文は、配列宣言（dim文）を行った後に実行したり、2度以上option base文を実行するとエラーになるので注意してください。

配列宣言

次に使用するデータ領域を確保します。これを配列宣言といい、dim文を使います。

たとえば、**dim A(5,5,5)**とすると、A(1,1,1)からA(5,5,5)までの3次元、5×5×5＝125個の配列を使用することができます。（**option base 1**の場合）

dim文で指定する添字は最大値で、この値によって必要なデータ領域を確保します。

また、配列宣言を省略することも可能です。

配列宣言を省略して配列変数を使った場合は、

dim 配列名(10)

を宣言したのと同じになります。

宣言せずに11番以上の配列変数を指定したり、一度宣言されている配列を宣言されているよりも大きなサイズで再度、宣言することはできません。

次元の最大は255次元まで、各添字の最大は32767まで、1つの配列の要素数の最大は65535まで指定できます。ただしプログラムの1行の長さやメモリ容量の制約があるため、実際には指定できないことがあります。

例）
```
10 rem test
20 option base 1
30 dim A$(5)
      ⋮
```

配列変数の消去

dim文で宣言した配列を消去するには、clear文またはerase文を使います。

clear文を実行すると、すべての変数および配列変数が消去されます。erase文は指定した配列変数のみを消去します。

式と演算子

式

結果が数値になる式を数式、結果が文字になる式を文字列式と呼びます。

式は、演算される定数や変数などの項と、四則演算記号などの演算子により構成されます。単に定数や変数のみでも演算子の省略された式(単項式)としてみることができます。

例) A$="ABC"+"DEF" :文字列式

↑

演算子

B%=C%+10 :数式

算術演算

算術演算の優先順位は、( )内が最も優先され、以下は次の表の順番になります。

2番目の負符号は単項演算子で、符号の前が項でない場合にあとに続く項の値を負にします。

¥の整数除算は、¥の左右の項を整数型に変換(四捨五入)したあとに除算を行い、結果の少数点以下を切り捨てます。

modの剰余演算は、整数除算の余りを求めます。

¥,modのいずれも左右の項の値が整数型の範囲をこえる場合にエラーになるので注意が必要です。

算術演算子は次のとおりです。

優先順位	演算子	意 味	
1	^	べき乗算	例) 2^3 …2³
2	−	負符号	例) −A 5*−3
3	*	乗算	
3	/	除算	
4	¥	整数除算	例)4.5¥2…(結果)2
5	mod	剰余演算	例) 5 mod 2…(結果)1
6	+	加算	
6	−	減算	例) A−B 3−5

比較演算

比較演算は、数値や文字の大きさを比較して、比較演算子と合えば-１になり、合わなければ０の整数型の値になります。

整数型の-１は16ビットすべての桁が１に、０はすべての桁が０になり、論理演算と組み合わせると、複雑な判断に用いることができます。

演算子	意味
=	等しい
<> ><	等しくない
<	より小さい
>	より大きい
<= =<	より小さいか等しい
>= =>	より大きいか等しい

等号は､代入のときにも使用される記号ですから注意が必要です。

比較は数値または、文字列どうしで行われ、数値と文字列の比較はできません。

比較演算の優先順位は算術演算の次になり、比較演算どうしでは順位が同じです。

文字列の比較は、最初から１文字目どうし、２文字目どうしと順番に行い、すべての文字が同じであれば等しいとみなします。途中で文字が異なっていれば、その時点で文字コードの大きな方の文字列が大きいとみなします。文字列の長さが異なり、短いほうの文字が終了するまで同じであれば、長い方の文字列が大きいとみなされます。

例）"ＡＢＣＤ" ＝ "ＡＢＣＤ"
"ＡＢＣＤ" ＜ "ＡＢＣＥ"
"ＡＢＣＤ" ＞ "ＡＢＣ"
"ＡＢＣ " ＞ "ＡＢＣ"
"" ＜ "Ａ"

文字長が0の文字列、Null文字列(空文字)は、最も小さな文字列となります。

空白(スペース)にも文字コードがありますので文字列の中に空白があれば、１つの文字として扱われ比較を行いますので注意が必要です。

論理演算

ブール代数の演算
数学の『ブール代数』の規則に従う演算方法で、演算の結果が2つの値のうちの1つをとる演算です。2つの値は0と1などで表したりします。

論理演算は整数型で行われ、16ビットの各ビットどうしでブール代数の演算をします。☜コラム

そのとき、目的の項を整数型に変換するため、値が整数型の範囲をこえる場合エラーとなりますので注意が必要です。

演算子	順位	意味
not	1	否定
and	2	論理積
or	3	論理和
xor	4	排他的論理和(exclusive or)
imp	5	包含(implication)
eqv	6	同値(equivalence)

notは論理演算中、最も優先順位が高く、単項演算子になります。notに続く項のビットを反転させます。

X	not X
1	0
0	1

X	Y	X xor Y
1	1	0
1	0	1
0	1	1
0	0	0

X	Y	X and Y
1	1	1
1	0	0
0	1	0
0	0	0

X	Y	X imp Y
1	1	1
1	0	0
0	1	1
0	0	1

X	Y	X or Y
1	1	1
1	0	1
0	1	1
0	0	0

X	Y	X eqv Y
1	1	1
1	0	0
0	1	0
0	0	1

関数

BASICの組み込み関数として、平方根や三角関数などの数値関数、文字列処理関数、入出力関数などが用意されています。

関数の書式は関数名に続けて括弧( )で囲んで処理されるデータを書きます。この処理されるデータのことを引数と呼びます。

例) sin (pai (1/3))

↑

引数

数値関数

数値関数には次のものが用意されています。

abs	絶対値	sin	正弦
sgn	符号	cos	余弦
int	整数化	tan	正接
fix	整数部取り出し	atn	逆正接
frac	小数部取り出し	fac	階乗
sqr	平方根	sum	総和
exp	指数	pai	円周率
log	自然対数	rnd	乱数発生
log10	常用対数		

三角関数の引数や結果の角度単位は、度とラジアンの2通りの方法が可能です。角度単位はモード切り替えになっていて、最初はラジアンですが**option angle**文によりモード切り替えができます。また、角度の変換関数として**rad**と**deg**が用意されています。

なお、数値演算プロセッサ（**MZ-1M12**)(別売）を使用した場合、次の関数を高速処理することができるようになります。

sin, cos, tan, atn, log, log10, exp, sqr, fac, ^ （べき乗）

数値演算プロセッサの使用/不使用は**option float**文で指定します。

文字列処理関数

文字列に関連する関数としては次のものが用意されています。

文字列の部分取り出し……left$　mid$　right$
文字コード関連……asc　chr$
数値データ変換……val　str$　bin$　oct$　hex$
文字列の属性……len　instr
特殊変換ほか……space$　string$　hexchr$
ascchr$　mki$　mks$
mkd$　cvi　cvs　cvd

関数定義

BASICでは最初から用意されている組み込み関数のほかに、自由に関数名と演算内容を決めて使うことができます。これを定義関数と呼び、FNで始まる関数名と式を定義し、あとは定義した関数名で演算させることができます。 くわしい定義方法については**def FN**文の解説を参照してください。

システム変数

システム変数

システム変数とはBASIC内で最初から予約されている変数のことで、現在時刻やBASIC内部の状態を示すポインタなどがあります。

主なシステム変数には次のものが用意されています。

時計、タイマ関係 …… time$　day$　date$　time
ポインタ …… dtl　lpos
文字表示関係 …… csrlin
エラー処理 …… err　erl

システム定数

システム変数と同じようにBASICで予約されています。

sysid$　BASICバージョン(型式)の文字列になります。

データの型変換

BASICの数値型データは必要に応じて型変換が行われます。特に精度の異なる型変換が行われると、予定外の結果が出てしまう可能性があります。型変換の状況を良く理解して上手に活用してください。

- **精度が低くなる変換**

  精度が低くなる場合は、四捨五入が行われます。

  例）
```
print cint(12.58)
 13
Ok
print csng(3.1415926535897 93#)
 3.1415927
Ok
```

- **代入時の変換**

  変数に代入するときは、変数の型に合わせた変換が行われます。

  例）
```
list
   10 ABC%=123.4
   20 ABC#=123.4
   30 XYZ#=123.4#
   40 print ABC%,ABC#,XYZ#
Ok
run
 123        123.40000000059605    123.4
Ok
```

  …… 変数**ABC#**には単精度の定数を倍精度に変換して代入されますので、変換後の精度は単精度と同じになります。

  倍精度のデータとして代入する場合は、変数**XYZ#**への代入のように倍精度の定数(倍精度の型記号#を付けるか、9桁以上とする)を書く必要があります。

- **演算時の変換**

  整数型専用のものを除いた演算や関数は、単精度型または倍精度型で行われ、精度の異なるデータどうしの場合、精度の高い方の精度に変換して行われます。また、関数の場合は引数の型に合わせた結果を求めます。

  例）
```
print 2/3#
 .6666666666666667
Ok
print sqr(5),sqr(5#)
 2.236068            2.23606797749979
Ok
```

- **整数型演算時の変換**

  modや￥(整数除算)、論理演算などの整数型で行われる演算は、演算するデータをあらかじめ整数型に変換してから演算を行います。

  例）
```
print 100¥2.5
 33
```
  …… 2.5が整数型に変換するときに四捨五入されて3になります。

エラー

命令の入力時やプログラム実行途中に書式の間違いや範囲をこえるデータの使用、または周辺装置の異常などがあれば、コンピュータはエラーを発生して実行を中断します。このときエラーの内容に応じたエラーメッセージを表示します。文法上のエラーの場合次のように表示されます。

- **直接実行モード時**

  Syntax　error

- **関接実行モード時**

  プログラム実行時は発生した行番号も表示されます。

  Syntax error in nn……nnはエラーが発生した行番号

  プログラム実行中にエラーが発生したとき、編集コマンドの行番号指定にピリオド(.)を使用すると、ピリオドはエラー発生行番号を表すものとして扱われます。

  例）**list.** …　エラー発生行を表示します。

エラーの処理

プログラム実行中にエラーが発生したとき、実行を中断せずに発生したエラーの内容に応じた処理をして、元の処理を再開させることができます。

エラー処理関係の命令には次のものが用意されています。

on error goto	……	エラー発生時に実行する行の指定。
err	……	エラー番号を求める。
erl	……	エラー発生行を求める。
resume	……	エラー処理を終了して復帰する。

エラー番号はエラーの種類に応じて割り当てられています。エラーメッセージ表(付録)を参照してください。

演算誤差

BASICで演算処理を行った場合、有限桁の値を2進数で計算していますので、どうしても誤差が発生します。以下に誤差の発生する原因をいくつか示しますので演算処理での参考にしてください。

- **丸め誤差**

  計算結果の桁数がコンピュータで扱われる桁数をこえた場合、扱う桁数以下の部分は四捨五入(一般に「丸める」という)されてしまいます。

  単精度型では8桁までの数を浮動小数点形式(M＊10^{n}の型)で扱うので、1.23456789は内部的に1.2345679になり、2/3 の計算結果は0.66666667に丸められます。また、12345678＋0.02は、12345678になってしまいます。

- **2↔10進変換誤差**

  BASIC内部では数を2進数に変換して処理します。小数点以下の場合、10進数では桁数の少ない数値でも、2進数に直すと無限の桁数になることがあります。

  たとえば、10進数の0.1を2進数に変換すると0.00011001100…となります。

- **近似値計算**

  sin, **cos**, **exp**, べき乗などの関数は、近似値計算をしてその値を求めています。その結果、近似した分だけ、誤差が生じることになります。

- **桁落ち**

  近似値どうしの減算の場合、誤差を含んだ桁数を増やしてしまうことがあります。

  たとえば、1の位に誤差を含んでいる数どうしの100012－100001の計算をコンピュータで行うと、それぞれの誤差(絶対誤差)は±1以内、誤差率(相対誤差)は±0.01%になります。計算を行うと答えは11になります。このときも誤差は±1以内ですが誤差率は10%になってしまい誤差の影響が大きくなってしまいます。

これらが原因になって起こる誤差を最小限度にくい止めるには、絶対数の小さな数から計算していき、除算はできるだけあとに行い、そして関数計算を最後に行うようにしてください。

なお、数値演算プロセッサ(**MZ-1M12**)(別売) を使用した場合、使用しない場合と演算結果が異なる場合があります。

(数値演算プロセッサの使用/不使用は、**option float**文で指定します。)

4 漢字の扱いについて

漢字を文字列型データとして扱うことができ、プログラム中で、変数や配列名、ラベル名、定義関数名、コメントなどに使用できます。また、ファイル名にも使用できます。

全角文字

全角文字とは、漢字などのようにフォントが16×16ドットで構成された文字のことをいいます。これに対して、フォントが8×8または8×16ドットで構成された文字のことを半角文字といいます。**(97ページ参照)**

BASIC内部では全角文字は2バイトのシフトJISコードで扱われ、半角文字は1バイトのコードで扱われています。

グラフィック文字、漢字コード表はオーナーズマニュアルを参照してください。

全角文字のシフトJISコードとは、JISで定められた2バイトのJISコードを半角文字のアルファベットやカタカナと重ならないようにずらしたコードのことです。シフトJISの第1バイト目は16進数で&H81～9Fと&HE0～FCの範囲になります。

これは半角文字ではグラフィック文字のコードに相当します。そのため全角文字を扱う場合は、グラフィック文字を混在させることはできません。全角文字を扱うためのモード切り替え命令として、**kmode**文を用意しています。

kmode 0 …… 半角文字のみ扱うモード
(グラフィック文字使用可)

kmode 1 …… 全角文字も扱うことができるモード
(グラフィック文字使用不可)

JIS漢字コード
JIS(日本工業規格)により規格化されているコードです。当用漢字を中心とした約3000種の漢字、英文字、数字、特殊文字がJIS第1水準と呼ばれます。

全角文字の入力（漢字入力モード）

CONFIG. SYSファイルについてはMS-DOSマニュアル、漢字入力モードでの漢字入力および変換の方法については、オーナーズマニュアル「かな漢字変換機能」を参照してください。

漢字入力を行うためには、MS-DOSの**CONFIG. SYS**ファイルに**HISHOW. SYS**ファイル（デバイスドライバ）を登録しておく必要があります。 ☜コラム

なお、登録されていない場合はBASIC起動時に”日本語入力機能は利用できません”というメッセージが表示されます。

- メニューモード

コントロール＋変換キーを押して漢字入力モードにします。☜コラム

漢字入力モードになると、画面の最下行または最上行に入力エリアが現れます。入力および変換はこのエリア内で行われます。また、漢字入力モードを解除するときもコントロール＋変換キーを押します。

- スクリーンモード

コントロール＋無変換キーを押して漢字入力モードにします。☜コラム

漢字入力モードになると、カーソル位置が入力エリアとなります。漢字入力モードを解除するときもコントロール＋無変換キーを押します。

（注）漢字入力は文字画面が40桁モード時と、文字フォントが8×8ドット時には行えません。

コードの変換

全角文字はBASIC内部では、シフトJISコードで扱われていますが、JIS16進コードやJIS区点コードとシフトJISコードを相互変換する機能があります。

JISコード定数は&Jを付けて16進数で表します。

例）print　chr$(&J2122, &J3021)

区点コード定数は&Kを付けて10進2桁ずつで表します。

例）print　chr$(&K0102, &J1601)

シフトJISよりJISコードや区点コードに変換する関数が用意されています。

jis$ …… JIS16進コード文字列に変換します。

ktn$ …… 10進4桁の区点コード文字列に変換します。

これらの関数は引数の最初の文字についてのみ有効で、後に続く文字は、無視します。

全角文字と半角文字の変換

英数字や一部の記号、カタカナは全角文字と半角文字の両方にあります。そのため相互の変換を行う関数が用意されています。

akcnv$　……半角文字列を全角文字列に変換します。

kacnv$　……全角文字列を半角文字列に変換します。

そのほかの漢字関数

klen　……全角と半角文字が混在した文字列の文字数を求めます。

kpos　……文字列中の指定文字までのバイト数を求めます。

kext$　……文字列から英数カナ文字または漢字のみを取り出します。

kinstr　……文字列中の指定文字の位置を求めます。

kmid$　……文字列から任意の長さの文字列を取り出します。

knj$　……シフトJISコードを漢字に変換します。

ktype　……指定位置の文字のタイプを求めます。

文章の表示・印字機能

プリンタ印字については「15. プリンタ制御機能」の項を参照してください。

本BASICでは、日本語文章を画面に表示させたり、プリンタに印字させたりすることが簡単にできるようになっています。

list ＊ …… コメントを表示します。

llist ＊ …… コメントをプリンタに印字します。

auto ＊ …… アポストロフィ（'）付きの行番号の自動発生を行います。

list ＊は、プログラムの中でアポストロフィ(')で始まる行の文章のみを表示します。

また、**llist＊**を使えば、同様に文章だけをプリンタに印字することができます。

アポストロフィを使ったプログラムの作成には**auto＊**を利用すると便利です。**auto＊**は、アポストロフィ付の行番号を自動的に発生させます。

例）
```
list
    10 'お元気ですか
    20 'それではまた
Ok
list*
お元気ですか
それではまた
Ok
```

5 ファイルの管理

ファイル

ファイルとは、通常、「書類などの情報をとじたもの」というような意味で使われますが、BASICでも同様にプログラムやデータをファイルとして扱います。

ファイルを記録しておくものには、フロッピーディスクなどがあります。

ディレクトリ

ディレクトリ(directory)とは、ディスクなどに記憶されているファイルの管理台帳のことです。この部分には、ファイルごとのファイル名、書き込み禁止の有無、記録容量、記録位置、登録時間などの情報が書かれています。

ファイルの種類

拡張子については、「ファイルの指定方法」の項を参照してください。

BASICで扱うファイルの種類は次のとおりです。

1．プログラムファイル

- **BASICプログラムファイル(拡張子：BAS)** ☜コラム

  BASICで作成したプログラムのファイルです。

- **BASICアスキープログラムファイル（拡張子:DAT)**

  BASICで作成したプログラムをデータファイルとして読めるように記録したファイルです。

  ただし、この場合**BAS**形式で記録するよりも大きな容量を必要とします。

- **機械語プログラムファイル(拡張子：BIN)**

  機械語で作成したプログラムのファイルです。

2．データファイル

- **シーケンシャルアクセス・データファイル(拡張子：DAT)**

  このファイルは、ファイル中のデータが書き込まれた順に並んでいて、データをアクセス(読み書き)する際には、先頭より順番に行います(シーケンシャル・アクセス)。

  データ1個当たりの長さがまちまちでも順番に詰めて記録されますので、容量の効率も良くなります。ただし、アクセスはファイルの先頭より順番に行う必要があります。

階層ディレクトリについては「6．ファイルの保存と再生」を参照してください。

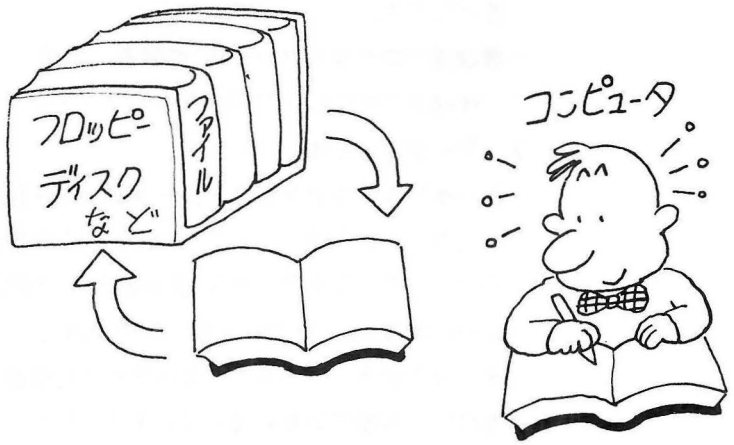
・ランダムアクセス・データファイル（拡張子：DAT）

このファイルは、ファイル中の任意の位置のデータを直接アクセス（ランダム・アクセス）できるようになっています。

ただし、データが固定長になるのでデータ長がそれぞれ違う場合、容量の効率が悪くなります。

3．階層ディレクトリ管理用ファイル（DIR）

このファイルは、ファイルの登録数が多くなるようなときグループ別にディレクトリを分けて参照する、階層ディレクトリの管理を行うファイルです。☜コラム



ファイルの指定方法

ファイル指定

ファイルの指定は、<ファイル指定>によって行います。
<ファイル指定>は、次の書式で書きます。

“〔〈装置名〉〕〔〈ディレクトリパス〉〕〈ファイル名〉〈拡張子〉”

例) A: ¥DRC¥ FILE1 .BAS
〈装置名〉 〈ファイル名〉
〈ディレクトリパス〉 〈拡張子〉

装置名

ファイルはファイル名を付けて各装置に登録します。
装置には次のようなものがあり、装置名で指定します。
装置名は論理装置名または物理装置名で指定します。
論理装置名はMS-DOSのロジカルドライブ名と同じものです。
また、物理装置名はBASIC-M25と互換性のある装置名です。
なお、RS-232Cインターフェイス、ディスプレイ画面、プリンタ、キーボードには論理装置名はありません。（物理装置名で指定してください。）

装置名		指定される装置	プログラム	シーケンシャル		ランダム
論理装置名	物理装置名		入 出 力	入 力	出 力	入 出 力
A:	FD 1:	フロッピーディスク				
B:	FD 2:	ドライブ	○	○	○	○
K:	FD 3:					
L:	FD 4:					
C:	HD 1	ハードディスクドライブ	○	○	○	○
〜	〜					
J:	HD 8:					
M:	MEM:	メモリディスク	○	○	○	○
—	COM1:	RS−232C インターフェイス	—	○	○	—
—	CRT1:	ディスプレイ画面	—	—	○	—
—	LPT:	プリンタ	—	—	○	—
—	KB:	キーボード	—	○	—	—

注）論理装置名は初期設定のものです。MS-DOSの**ASSIGN**コマンドで論理装置名を入れ替えることができます。

• フロッピーディスクドライブは最大4基まで使用できます。
また、ランダムアクセスが可能です。外部にディスクドライブを接続した場合にディスクドライブの種類に合わせてトラック数やタイミングを**init"FD:"**によりドライブ番号ごとに設定できます。

ディップスイッチの設定についてはオーナーズマニュアルを参照してください。

注）外部にディスクドライブを接続した場合は、ディスクドライブの種類に合わせて、本体のディップスイッチを設定する必要があります。☞コラム

CONFIG. SYSファイルについてはMS-DOSマニュアルを参照してください。

• ハードディスクドライブは1基使用できます。
任意分割すると、容量を最大8分割でき、みかけ上8基のドライブとして扱うことができるようになります。

シーケンシャルおよびランダムアクセスファイルについては「6. プログラムの保存と再生」を参照してください。

• メモリディスク（RAMディスク）は、グラフィック**V-RAM**や増設RAMの一部を使用し、フロッピーディスクと同様にランダムアクセスが可能です。

注）・コンピュータ本体の電源を切ると内容が消えてしまうので、必要なファイルはあらかじめフロッピーデッスクなどへ移しかえておく必要があります。

・メモリディスクを使用するためには、MS-DOSの**CONFIG. SYS**ファイルに**RAMDISK. SYS**ファイル（デバイスドライバ）を登録しておく必要があります。出荷時には登録されていますが、登録されていない場合はBASIC起動時に“RAMディスク機能は利用できません”というメッセージが表示されます。

• RS-232Cインターフェイスはシーケンシャルアクセスファイルとして扱うことができます。
• ディスプレイ画面は出力専用のファイル装置として使用します。
• プリンタは出力専用のファイル装置として使用します。
• キーボードは入力専用のファイル装置として使用します。

ディレクトリパス

ディレクトリパスについては、「6. ファイルの保存と再生」を参照してください。

ディレクトリパスは、アクセスしようとするファイルが階層ディレクトリのうちのどこにあるか示します。☞コラム

ファイル名

使用できないファイル名については MS-DOS マニュアルを参照してください。

ファイル名はファイルごとにつける名前のことです。
装置内に複数のファイルがある場合、どのファイルが指定されたのかを見分ける必要がありますので、同じファイル名を付けることはできません。ファイル名には半角文字、全角文字が使用可能で、半角文字で最大 8 文字まで可能です。（全角文字は 1 文字で 2 文字分と数えます）☜コラム

なお、ディレクトリパスやファイル名を小文字で指定した場合、自動的に大文字に変換されます。

拡張子

拡張子は、ファイル形式を指定するものです。
ファイル形式によって、次の文字を指定します。

- **.BAS:** BASICプログラムファイル
- **BIN:** 機械語プログラムファイル
- **.DAT:** BASICアスキープログラムファイル
  シーケンシャルアクセス・データファイル
  ランダムアクセス・データファイル

6 ファイルの保存と再生

プログラムファイル

プログラムを保存、再生するファイルをプログラムファイルと呼びます。

プログラムの保存

メモリ内のプログラムをフロッピーディスクなどに記録するときは、**save**、**list**、**bsave**コマンドを使用します。

(注)同一ファイル名がすでに存在するときは、以前にあったファイルの内容は更新されます。

save "〔〈装置名〉〕〈ファイル名〉"〔,{A/C}〕

アスキー形式
プログラムを文字コードに変換して記録する方式です。(シーケンシャルアクセス)データファイルと同じファイル形式なので、データファイルとして読むことができるようになります。

メインメモリ内のBASICプログラムを指定された装置へ指定されたファイル名を付けて記録します。

ファイル拡張子は**BAS**になります。また、最後に**A**または**C**を指定するとファイル形式は**アスキー形式(DAT)**で記録します。

☜コラム

Aまたは**C**を指定しないのが一般的な記録方式で、プログラムは中間コードに変換されて記録します。**アスキー形式**は、一般的な記録よりも多くの領域が必要になり、書き込み、読み出しにも時間がかかります。

なお、**C**を指定すると、行番号の前の空白を記録しないので、**A**指定より少ない領域で記録できます。

list "〔〈装置名〉〕〈ファイル名〉"〔,〔〈開始行〉ー〔〈終了行〉〕〕

メインメモリ内のBASICプログラムを指定された装置へ指定されたファイル名を付けて**アスキー形式(DAT)**で記録します。

saveコマンドと異なる点は記録する行番号を指定できることです。

指定した開始行から終了行までを記録します。

bsave "〔〈装置名〉〕〈ファイル名〉", 〈開始アドレス〉, 〈サイズ〉

メインメモリ内の機械語プログラムを指定された装置へ指定されたファイル名を付けて記録します。

ファイル拡張子は**BIN**になります。

なお**bsave**コマンドでアドレス指定できる範囲は64キロバイト以内なので、**def seg**文でベース(下限)アドレスを指定する必要があります。

プログラムの再生

フロッピーディスクなどに記録されたプログラムをメモリに読み出すときは、**load**、**run**、**bload**コマンドを使用します。

load "〔〈装置名〉〕〈ファイル名〉"〔, R〕

指定された装置に記録されているBASICプログラムをメモリ上に読み出します。

Rを指定すると、読み出し後ただちに実行を開始します。

bload "〔〈装置名〉〕〈ファイル名〉"〔, 〈アドレス〉〕〔, R〕

指定された装置に記録されている機械語プログラムをメモリ上に読み出します。

なお、**bload**コマンドでアドレス指定できる範囲は64キロバイト以内なので、**def seg**文でベースアドレスを指定する必要があります。

run "〔〈装置名〉〕〈ファイル名〉"

指定された装置に記録されているBASICプログラムをメモリ上に読み出し、実行します。

書き込みチェック

フロッピーディスクへプログラムやデータファイルの書き込みを行った後に、正しく記録されているかチェックを行うことができます。チェックを行うかどうかは**verity**コマンドで指定します。

verify on …… チェックを行う

verify off …… チェックを行わない

ファイルの消去

メモリにあるプログラムを消すときは**new**を使いますが、指定さ

れた装置に記録されているファイルを消すときは**kill**コマンドを使用します。

kill "〔〈装置名〉〕〈ファイル名〉"

このコマンドによって消去されたファイルは元に戻りませんので注意してください。

ファイル名の一覧

記録したファイル名の一覧を見るときは、**files**、**lfiles**コマンドを使用します。

files "〔〈装置名〉〕〔〈ファイル名〉〕"

lfiles "〔〈装置名〉〕〔〈ファイル名〉〕"

指定された装置に記録されているファイル名の一覧が**files**の場合は画面に表示され、**lfiles**の場合はプリンタに印字されます。

ディレクトリ名については「階層ディレクトリ」の項を参照してください。

filesの実行例

```
Directory of A:¥ ←ディレクトリ名 ☜コラム
  '14:30  "A:DATA1.DAT"  '87-04-14  1400
    ↑       ↑   ↑       ↑        ↑       ↑
作成時刻 装置名 ファイル名 拡張子 作成日 ファイルの大きさ
                                          (単位:バイト)
```

ファイル名の一覧を表示した後で、その中にあるファイルを読み出し、実行、消去したい場合は次のようにすると便利です。

①そのファイル名の最初に表示されている作成時刻のところへカーソルを移動する。

②作成時刻のかわりに**load**、**run**、**kill**を入力して⏎キーを押す。

プログラムの合成

メモリ上にあるプログラムと、ファイルに記録されたプログラムを合成(マージ)するときは**merge**コマンドを使用します。

merge "〔〈装置名〉〕〈ファイル名〉"

指定された装置に記録されているファイルが読み出され、現在メモリにあるフログラムと合成された1つのプログラムが作成されます。

プログラムの連結

プログラムが長く、メモリ内で処理できないときに、プログラムをいくつかに分割してファイルに記録しておき、**chain**で連結(チェ

イン)しながら実行することができます。

chain<ファイル指定>〔,実行開始行>〕〔,all〕

chain merge〈ファイル指定〉〔,〈実行開始行〉〕〔,all〕〔,delete〈削除範囲〉〕

chainを実行すると、現在実行中のプログラムを消去し、指定された装置に記録されているファイルを読み出し、実行します。

chain mergeの場合は、現在のプログラムを消去せずに、指定ファイルを現在のプログラムに合成し、実行します。

このとき、次のプログラムへ引き渡す変数や配列を**common**で定義することができます。

ファイル名の変更

すでに記録したファイルの登録名を**name**コマンドで変更することができます。

name"〔〈装置名〉〕〈ファイル名〉" as "〈新ファイル名〉"

ファイル名で指定したファイルのファイル名を新ファイル名に変更します。

ファイルのプロテクト

作成したファイルを誤って消去しないように**set**コマンドでファイルにプロテクト(書き換え禁止)をかけることができます。

set "〔〈装置名〉〕〈ファイル名〉","P"

指定したファイルをプロテクト状態にします。

プロテクトされたファイルに対しては、消去、ファイル名変更、ファイルへの書き込みなどが禁止されます。**files**コマンドによるファイル名表示のときに、プロテスト状態のファイルは日付表示の後に*が表示されます。

プロテクトを解除するときは**"P"**のかわりに""を指定します。

ファイルの保護

作成したファイルを読み出して内容を見たり、内容の変更、あるいは消去といったことを禁止することができます。

save "〔<装置名>〕<ファイル名>",P

指定したファイルを保護形式(秘密形式)にします。

保護形式を指定されたファイルは**list**コマンドで内容を見たり、内容を変更したりすることはできません。

また、一度ファイルを保護形式に指定すると、保護形式を解除する方法は用意されていませんのでご注意ください。

ファイルのコピー

COPYコマンドについては、MS-DOSマニュアルを参照してください。

フロッピーディスクなどに記録したファイルは、MS-DOSの**COPY**コマンドによって、別の装置(別のフロッピーディスクなど)にコピーすることができます。☜コラム

なお、BASICが起動しているときは、**system**↵と入力して、MS-DOSのプロンプト(**>**)が表示されてから、**COPY**コマンドを実行してください。

system ↵
A>

child文については「18. MS-DOSへの移行」の項を参照してください。

なお、再びBASICを起動するときは、カレントドライブにMS-DOS/BASICシステムディスクがあることを確かめて、**BASIC**↵と入力してください。

また、**child**文を利用して、BASICを終了せずにMS-DOSコマンドを実行することもできます。

データファイル

プログラムの中で使用するデータを保存、再生するファイルをデータファイルと呼びます。

データファイルにはシーケンシャルアクセスファイルとランダムアクセスファイルがあります。

シーケンシャルアクセスファイル

シーケンシャルアクセス

シーケンシャルアクセスファイル(以下シーケンシャルファイルと呼びます)とは、連続的に書かれたデータファイルのことで、次のような長所と短所があります。

〈長所〉

- ファイルの中のデータは書き込まれた順に並んでいて、アクセスは最初から順番に行います。
- ファイルの最後にデータを追加することは、比較的簡単にできます。
- ファイルの中のデータは順番に記録されているので無駄がありません。
- ファイル作成のプログラムが比較的簡単にできます。

〈短所〉

- データの内容を変更するときは、すべてのデータを読み出し、変更を行った後、ファイルに書き込む必要があります。

ランダムアクセス

シーケンシャルファイルを使用してデータの保存、再生を行う場合は、次の命令を使用します。

open、close、print #、print # using、input #、write #、line input #、eof、input $

ランダムアクセスファイル

ランダムアクセスファイル(以下ランダムファイルと呼びます)とは、1レコード単位で記録位置を指定できるデータファイルのことで、次のような長所と短所があります。

〈長所〉

- ファイルの中のデータは、直接アクセスできるので、任意の位置のデータを読み書きすることができます。
- ファイルの中の一部のデータだけを変更することができます。

- ファイルの最後にデータを追加することができます。

〈短所〉

- データが少ない場合でも1レコードごとにアクセスを行うため、無駄な部分があります。
- ファイル作成のために多くの設定が必要なため、シーケンシャルファイルに比べてプログラムが複雑になります。

ランダムファイルを使用してデータの保存､再生を行う場合は、次の命令を使用します。

open、close、field、put #、get #、lset、rset、mki$、mks$、mkd$、cvi、cvs、cvd

いずれのデータファイルにおいても、**open**したファイルは必ず**close**するようにしてください。

なお、データファイルもプログラムファイルと同様に**kill**、**files**、**lfiles**、**name**、**set**命令によりファイルの消去、ファイル名の一覧表示などが行えます。

シーケンシャルファイルへのデータの保存、再生

シーケンシャルファイルへのデータの保存、再生を行う場合、プログラムファイルのように1つのコマンドだけで行うことはできません。ファイルの状態などを設定する必要があります。

その手順についてプログラム例を用いて説明をします。

```
 10 open "O",#1,"Fruits"
 20 input "種類は";K$
 30 if K$="end" then 80
 40 input "数は  ";S
 50 print #1,K$
 60 print #1,S
 70 goto 20
 80 close #1
100 open "I",#1,"Fruits"
110 if eof(#1) then 170
120 input #1,K$
130 input #1,S
140 print "種類：";K$
150 print "数は";S;"個です"
160 goto 110
170 close #1:end
```

ファイルへの書き込み ①～③

ファイルからの読み出し ④～⑥

①シーケンシャルファイルへデータを書き込み、読み出しをする

には、どのファイルにどのような操作をするのか宣言する必要があります。**open**で宣言をします。

```
10 open "O",#1,"Fruits"
```

"O"……ファイルへのデータの書き込みを設定します。
#1 ……**open**によって出力するファイルの番号を設定します。
"Fruits"…ファイル名を設定します。
この行番号10の実行でデータの書き込みが可能になります。
以後のファイル処理は、ファイル名ではなくファイル番号で行います。この設定を行うことをファイルをオープンするといいます。

②**print #**、**write #**などでファイルにデータを書き込みます。

```
20 input "種類は";K$
30 if K$="end" then 80
40 input "数は  ";S
```

行番号20と40で書き込むデータを入力します。
行番号30では行番号20で入力されたデータが"end"かどうかを判断して"end"であれば行番号80へジャンプして書き込みを終了します。

```
50 print #1,K$
60 print #1,S
```

行番号50、60では、ファイル番号1でオープンされたファイルにK$、Sのデータを書き込みます。シーケンシャルファイルにデータを書き込む命令は、**print #**のほかに**print # using**、**write #**があります。

③データの書き込みが終了したら、**close**でファイルを閉じます。閉じたファイルは、再びオープンしないと読み出しも書き込みもできません。

```
70 goto 20
80 close #1
```

行番号70で行番号20へジャンプしています。行番号20で"end"が入力されるまでくり返します。
行番号80でファイル番号1のファイルを閉じています。
以上でシーケンシャルファイルへのデータの書き込みは終わりです。

④次に書き込んだデータの読み出しをします。読み出す場合も書き込むときと同様に**open**でファイルをオープンする必要があります。このとき、書き込みを指定していた "O" の部分は "I" に変えて、読み出しの状態に設定します。

```
100 open "I",#1,"Fruits"
```

行番号100の実行でシーケンシャルファイル"Fruits"からのデータの読み出しが可能になります。

⑤**input #**、**line input #**などでファイルよりデータを読み出します。

```
110 if eof(#1) then 170
```

行番号110ではシーケンシャルファイルの終わりを**eof**関数によって調べ、終わりであれば行番号170へジャンプします。
eof関数は、データの最後ならば－１、そうでなければ０の値を返します。

```
120 input #1,K$
130 input #1,S
```

行番号120、130ではファイル番号１でオープンされたファイルからデータを変数K$、Sに読み出します。
シーケンシャルファイルでは、データを読み出す場合、書き込んだ順番に行う必要があり、一部のデータを読み出す場合でも、その前に書き込まれたデータをいったん読み出さなければなりません。

```
140 print "種類：";K$
150 print "数は";S;"個です"
160 goto 110
```

行番号140、150では読み出したデータを表示しています。
行番号160では行番号110へジャンプしています。行番号110で行番号170へジャンプするまでくり返します。

⑥データの読み出しが終了したら、**close**でファイルを閉じます。

```
170 close #1:end
```

行番号170ではファイル番号１のファイルを閉じています。

このプログラムの実行例は、次のようになります。

```
run
種類は? りんご↵
数は  ? 5↵
種類は? みかん↵
数は  ? 3↵
種類は? end↵
種類：りんご
数は 5 個です
種類：みかん
数は 3 個です
Ok
```

種類は？のところで "end" を入力するとデータの書き込みは終了します。

シーケンシャルファイルへのデータの追加

シーケンシャルファイルでは、ファイルの任意の位置への書き込み、読み出しを行えませんが、すでに作成されたファイルの最後にデータを追加することはできます。

①データを追加する場合も**open**でファイルをオープンします。

このときは、追加書き込みの **"A"** を指定します。

データの保存、再生のところで使用した例プログラムの一部を変更するだけで簡単にデータを追加するプログラムが作れます。す。

行番号10を次のように変更します。

```
10 open "O",#1,"Fruits"
          ↓
10 open "A",#1,"Fruits"
```

シーケンシャルファイルにデータを追加する場合、**open**での指定以外はデータの書き込みと同じ手順で行います。

②**print**＃、**write** ＃などでファイルに追加するデータを書き込みます。

③データの書き込みが終了したら、**close**でファイルを閉じます。

ランダムファイルへのデータの保存、再生

ランダムファイルもシーケンシャルと同様にファイルの状態を設定する必要があります。また、ランダムファイルは、データの読み書きをレコードバッファ(メインメモリ上でデータを一時的に蓄える部分)を介して行い、そのレコードバッファへの読み書きは文字列型でなければなりません。そのためシーケンシャルファイルよりも多くの手続きが必要です。

データの書き込み

データの書き込みプログラム

```
10 open "R",#1,"Fruits"
20 input "種類は";K$
30 if K$="end" then 120
40 input "数は  ";S
50 T$=mki$(S)
60 field #1,10 as A$,2 as B$
70 lset A$=K$
80 lset B$=T$
90 input "レコード番号";R
100 put #1,R
110 goto 20
120 close #1:end
```

①ランダムファイルをオープンする場合、**open**で**"R"**を指定します。ランダムファイルでは読み書きに関係なく同じ**"R"**を指定するので、1つの**open**でデータの読み書きをすることもできます。

```
10 open "R",#1,"Fruits"
```

行番号10では"Fruits"という名のランダムファイルをオープンしています。この場合、レコード長の指定を省略しているので1レコードは256バイトとなります。

②ファイルに書き込むデータを用意します。ランダムファイルに書き込むデータは文字列型のみで、数値型のものは使用できません。そのため、数値型データは文字列型に変換する必要があります。

```
20 input "種類は";K$
30 if K$="end" then 120
40 input "数は  ";S
50 T$=mki$(S)
```

数式文字列に変換する関数は、行番号50の**mki$**の他に**mks$**、**mkd$**があります。

③ランダムファイルにデータの読み書きを行う場合、1レコードのデータをバッファに蓄えておき、1レコード分ずつ読み書きを行います。また、そのレコードバッファは読み書きの前に**field**でデータの割り付けがされている必要があります。

```
60 field #1,10 as A$,2 as B$
```

行番号60ではファイル番号1用のレコードバッファの最初の10バイトをA$、次の2バイトをB$に割り付けています。
この場合、残りの244バイトは未使用となります。

④**field**で割り付けたバッファに**lset**、**rset**を使用してデータを書き込みます。

```
70 lset A$=K$
80 lset B$=T$
```

行番号70、80では**lset**でレコードバッファにデータを書き込んでいます。

⑤レコードバッファの内容を**put#**でファイルに書き込みます。
ランダムファイルにはレコード番号の位置にデータが記録されます。このレコード番号を指定することにより任意の位置のデータの読み書きが可能となります。

```
 90 input "レコード番号";R
100 put #1,R
```

行番号90ではレコード番号を入力します。
行番号100では、ファィル番号1のレコードバッファにあるデータを"Fruits"というファイルへ行番号90で入力したレコード番号の位置に書き込んでいます。

⑥データの書き込みが終了したら、**close**でファイルを閉じます。

```
110 goto 20
120 close #1:end
```

行番号120ではオープンされていたファイルを閉じています。

このプログラムの実行例は次のようになります。

```
run
種類は? りんご↵
数は  ? 5↵
レコード番号? 1↵
種類は? みかん↵
数は  ? 3↵
レコード番号? 2↵
種類は? end↵
Ok
```

種類は？のところで"end"を入力するとデータの書き込みは終了します。

行番号60でA$には10バイトの割り付けをしているので、種類の入力は10文字(全角文字は5文字)までになります。

データの読み込み

データの読み込みプログラム

```
 10 open "R",#1,"Fruits"
 20 field #1,10 as A$,2 as B$
 30 input "レコード番号";R
 40 if R=0 then 100
 50 get #1,R
 60 S=cvi(B$)
 70 print "種類:";A$
 80 print "数は";S;"個です"
 90 goto 30
100 close #1:end
```

①ファイルのオープンをします。ランダムファイルではデータの読み書きでオープンの方法は変わりません。

```
10 open "R",#1,"Fruits"
```

②**filed**でレコードバッファに文字変数を割り付けます。このとき、書き込み時と変数名を変えることはできますが、レコードバッファの割り付けは同じにする必要があります。

```
20 field #1,10 as A$,2 as B$
```

③ランダムファイル上のデータをレコードバッファに読み込みます。

ファイルから1レコード分のデータをレコードバッファに読み込むには**get #**を使います。

```
30 input "レコード番号";R
40 if R=0 then 100
50 get #1,R
```

行番号50ではファイル番号1のレコードバッファに"Fruits"というファイルから行番号30で入力したレコード番号のデータを読み込んでいます。

④レコードバッファに読み込んだデータを取り出します。

データは**field**で割り付けた文字変数を使用することにより取り出せます。また、文字列型に変換していた数値は**cvi、cvs、cvd**関数により、もとの数値型に変換し直します。

```
60 S=cvi(B$)
70 print "種類：";A$
80 print "数は";S;"個です"
```

⑤データの読み込みが終了したら、**close**でファイルを閉じます。

```
 90 goto 30
100 close #1:end
```

このプログラムの実行例は次のようになります。

（このプログラムの実行前に書き込みプログラムが実行されている必要があります。）

```
run
レコード番号? 1↵
種類：りんご
数は 5 個です
レコード番号? 2↵
種類：みかん
数は 3 個です
レコード番号? 0↵
Ok
```

レコード番号？のところで0を入力するとデータの読み込みは終了します。

ランダムファイルに対するデータの追加、変更

ランダムファイルは、データの追加、変更を自由に行うことができます。レコード番号を指定することにより、どの場所のデータも読み書きすることができます。

すでにデータが書き込まれているレコード番号と同じレコード番号で新しいデータを書き込むと、前のデータは消去されます。

ランダムファイルへのデータの追加、変更の手順は、書き込みの場合と同じです。データの書き込み(**66ページ**)の例のプログラムを実行し、確認してみてください。

ファイルのコピー

フロッピーディスクなどに記録したファイルは、MS-DOSの**COPY**コマンドによって、別の装置(別のフロッピーディスクなど)にコピーすることができます。☜コラム

COPYコマンドについては、MS-DOSマニュアルを参照してください。

なお、BASICが起動しているときは、**system**↵と入力して、MS-DOSのプロンプト(**>**)が表示されてから、**COPY**コマンドを実行してください。

```
system ↵
A>
```

child文については「18. MS-DOSへの移行」の項を参照してください。

なお、再びBASICを起動するときは、カレントドライブにMS-DOS/BASICシステムディスクがあることを確かめて、**BASIC**↵と入力してください。

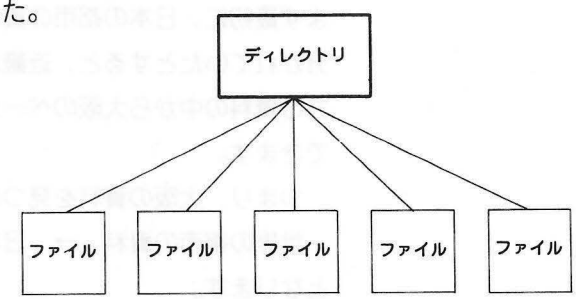
また、**child**文を利用して、BASICを終了せずにMS-DOSコマンドを実行することもできます。

階層ディレクトリ

ファイルの数が多くなると、使いたいファイルがなかなか見つからないなど、扱いにくくなります。
このようなときは、関連のあるファイルごとに、いくつかの集合に分類整理する方法が有効です。
BASICの場合、いくつかのファイルをディレクトリとして管理しており、このディレクトリをいくつか作ることで、きめの細かいファイル管理ができるようになります。

単層ディレクトリ

従来のファイルの管理は、フロッピーディスクなど１つの装置に１つのディレクトリ、すなわち、下図のような単層で行われていました。

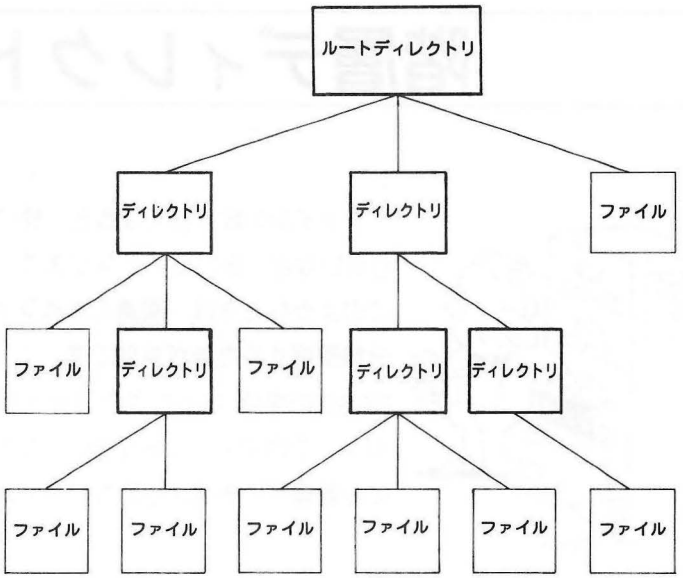


このとき、**files**を実行するとすべてのファイル名が表示されます。
しかし、この場合はファイルの数が多くなると、ファイルをさがしにくく、不便です。

階層ディレクトリ

ファイルを効率的に管理するために考えられたのが、階層構造(木構造)をもつ階層ディレクトリです。

階層ディレクトリを図で表すと次のようになります。



基本となるディレクトリを**ルートディレクトリ**と呼びます。

この中に別のディレクトリ(子ディレクトリ)を作り、ファイルを管理することができます。さらに、子ディレクトリの中で子ディレクトリを作成することもできます。

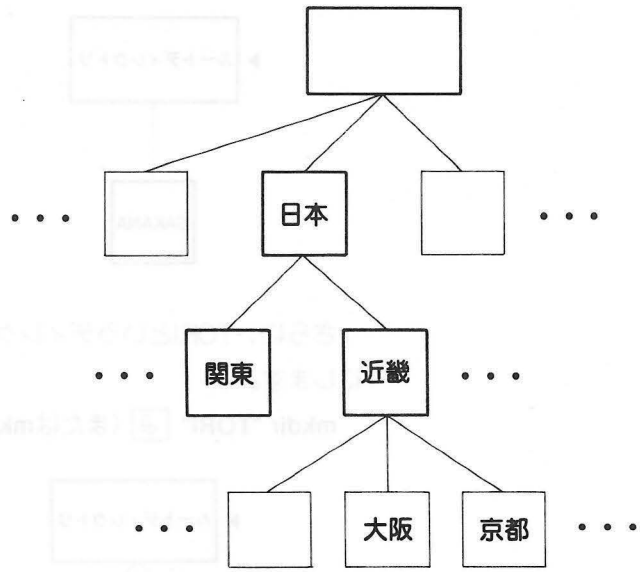
たとえば、世界の都市の中で大阪のことを調べる場合を考えます。まず最初に、日本の都市の資料が必要になります。資料が地方別に分かれていたとすると、近畿地方の資料を使います。次に、近畿地方の資料の中から大阪のページを開き、大阪のことを調べることができます。

つまり、大阪の資料を見つけるまでの道すじを示すと、

世界の都市の資料 → 日本 → 近畿地方 → 大阪

となります。

この例を、階層ディレクトリに当てはめて考えると、日本、近畿地方がディレクトリに対応し、大阪がファイルに対応します。



本BASICの階層ディレクトリでは階層の道すじを表すのに¥マークを使用します。

したがって、例は次のように表されます。

¥日本¥近畿¥

このときファイルにあたる大阪にたどりつくまでの道すじをパスといい、ディレクトリを¥で句切った

¥日本¥近畿¥

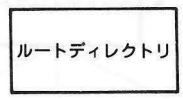
をディレクトリパスといいます。

また、階層ディレクトリの最上部のディレクトリのことをルートディレクトリと呼び、先頭の"¥"がルートディレクトリを指しています。

ディレクトリの作成方法

ディスクのフォーマットを行うと、ディレクトリが初期化され、ファイルはすべて消去されます。

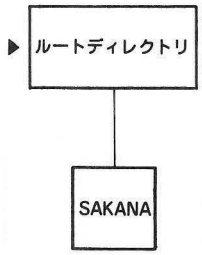
この状態では、ルートディレクトリのみが存在しています。



ここに新しくディレクトリを作成する場合、**mkdir**コマンドを使います。

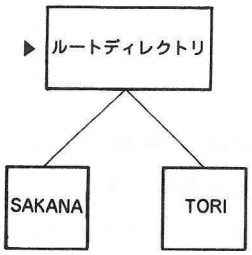
たとえば、ルートディレクトリの下にSAKANAというディレクトリを作成する場合、次のようにします。

mkdir "SAKANA" ↵ （または**mkdir"¥SAKANA"**↵）



さらに、TORIというディレクトリを追加する場合は、次のようにします。

mkdir "TORI" ↵（または**mkdir"¥TORI"**↵）



もちろん、ここでプログラムファイルやデータファイルを作成することもできます。

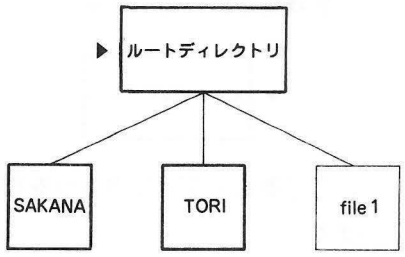
たとえば、あるプログラムをfile1というファイル名で記録する場合は、次のようにします。

save "file1" ↵（または**save"¥file 1"**↵）

filesコマンドを実行すると次のように表示されます。

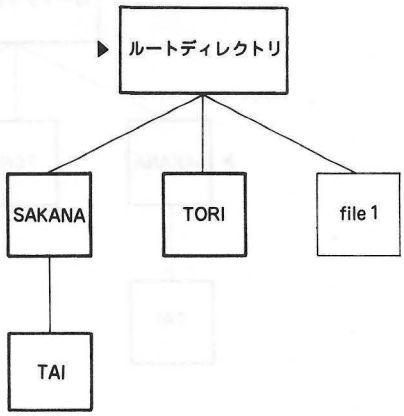
```
Directory of A:¥

'XX:XX "A:SAKANA"       'XX-XX-XX  <DIR>
'XX:XX "A:TORI"         'XX-XX-XX  <DIR>
'XX:XX "A:FILE1.BAS"    'XX-XX-XX XXXXX
```



SAKANAというディレクトリの下にTAIというディレクトリを作るには、次のようにします。

mkdir "SAKANA¥TAI" ↵



さらに下の階層を作る場合も同様に"¥"で句切って追加します。

mkdir "……¥……¥…… "

カレントディレクトリの変更

階層ディレクトリでは各ディレクトリはそれぞれ独立してファイルのアクセスを行います。したがって、同じファイル名のファイルを別々のディレクトリの中に作成することができます。

ファイルをアクセスする場合、そのファイルの存在するディレクトリに移動します。その後はそのディレクトリを起点としてファイルを指定することができます。このディレクトリのことを**カレントディレクトリ**と呼びます。

カレントディレクトリの移動には**chdir**コマンドを使います。

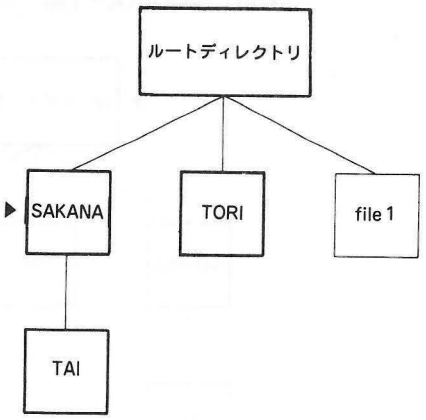
たとえば、カレントディレクトリをルートディレクトリからSAKANAというディレクトリに移す場合は、次のようにします。

chdir "SAKANA" ↵ (または**chdir"¥SAKANA"** ↵)

このとき**files**コマンドを実行すると次のように表示されます。

```
Directory of A:¥SAKANA

'XX:XX "A:."              'XX-XX-XX  <DIR>
'XX:XX "A:.."             'XX-XX-XX  <DIR>
'XX:XX "A:TAI"            'XX-XX-XX  <DIR>
```



"."と".."は、ディレクトリの階層を管理するファイルで**".."**は元の親ディレクトリを意味し、**"."**は現在のディレクトリを意味します。これらのディレクトリ管理ファイルは自動的に作成され、直接消去することはできません。

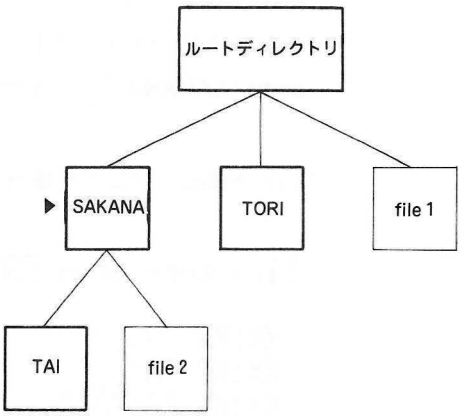
ディレクトリ管理ファイルを使って、

chdir ".." ↵

とするとカレントディレクトリから親ディレクトリに戻ることができます。

ここで、プログラムをfile2というファイル名で記録する場合、次のようにします。

save"file2" ↵



ここからさらにその下のTAIというディレクトリに移るには、次のようにします。

chdir "TAI" ↵

また、カレントディレクトリをルートディレクトリに戻すには、次のようにします。

chdir"¥" ↵

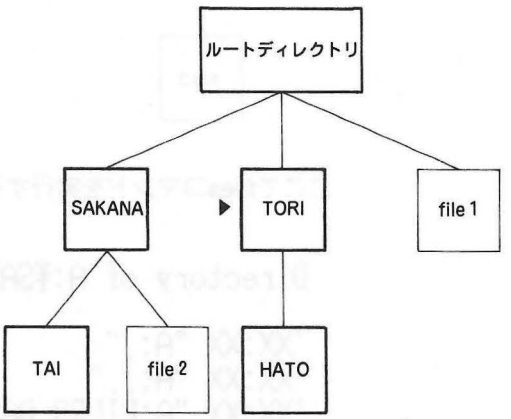
TORIというディレクトリの下にHATOというディレクトリを作るには、さきほどの方法で、

mkdir"TORI¥HATO" ↵

としてもよいのですが、いったんカレントディレクトリをTORIに移してから作る方法もあります。

chdir"TORI" ↵

mkdir"HATO" ↵



この状態でTAIというディレクトリの下にファイルを作りたいときは、

chdir"¥SAKANA¥TAI" ↵

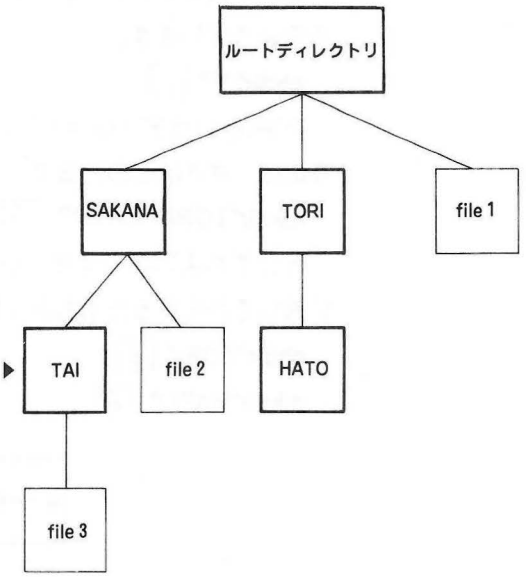
を実行して、カレントディレクトリをTAIに移してから、プログラムを作り、次のようにします。

save"file3" ↵

なお、カレントディレクトリを"TORI"に置いたまま

save"¥SAKANA¥TAI¥file3"

で行うこともできます。



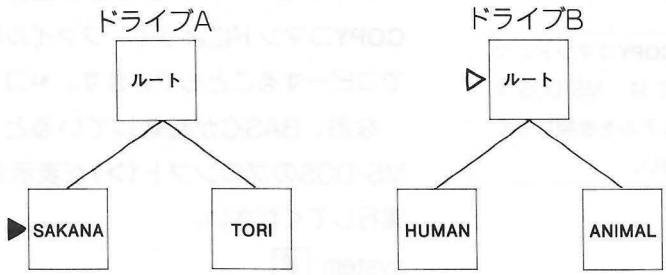
ここで**files**コマンドを実行すると次のように表示されます。

```
Directory of A:¥SAKANA¥TAI

'XX:XX "A:."                 'XX-XX-XX  <DIR>
'XX:XX "A:.."                'XX-XX-XX  <DIR>
'XX:XX "A:FILE3.BAS"         'XX-XX-XX XXXXXX
```

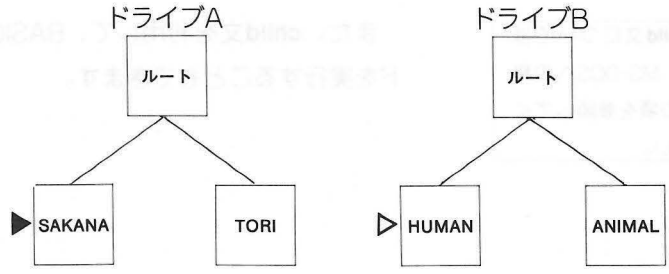
chdirコマンドで現在の装置(カレントドライブ)と違う装置名を指定した場合、指定した装置内のカレントディレクトリは変更されますが、装置の変更は行われず、従ってカレントドライブのカレントディレクトリを参照したままとなります。

例）ドライブAの"SAKANA"が参照ディレクトリのとき



▶：ドライブAのカレントディレクトリ
（参照ディレクトリ）
▷：ドライブBのカレントディレクトリ

⇩ **chdir "B:HUMAN"** ↵



▶：ドライブAのカレントディレクトリ
（参照ディレクトリ）
▷：ドライブBのカレントディレクトリ

BASICからカレントドライブを変更するには、**chdir "B:"または child"＜カレントドライブ＞", m, 1 , 0 を実行します。**

ディレクトリの削除

不要なディレクトリを削除するには**rmdir**コマンドを使います。

現在、カレントディレクトリがTORIとして、その下のHATOというディレクトリを削除する場合には、次のようにします。

rmdir"HATO" ↵

このとき、ディレクトリ"HATO"にあるファイルおよび、ディレクトリはディレクトリ管理ファイルである**". "**と**". . "**を除いてあらかじめ消去されている必要があります。

もし、ファイルを消去するときは**kill**コマンドを使用します。

ディレクトリのコピー

フロッピーディスクなどに記録したファイルは、MS-DOSの**COPY**コマンドによって、ファイル単位のほか、ディレクトリ単位でコピーすることもできます。☜コラム

なお、BASICが起動しているときは、**system**↵と入力して、MS-DOSのプロンプト(**>**)が表示されてから、**COPY**コマンドを実行してください。

system ↵
A>

COPYコマンドについては、MS-DOSマニュアルを参照してください。

なお、再びBACICを起動するときは、カレントドライブにMS-DOS/BASICシステムディスクがあることを確かめて、**BASIC**↵と入力してください。

また、**child**文を利用して、BASICを終了せずにMS-DOSコマンドを実行することもできます。

child文については「18. MS-DOSへの移行」の項を参照してください。

7 表示モード

CONFIG. SYSファイルについてはMS-DOSマニュアルを参照してください。

ディスプレイの画面には、文字(アルファベット、数字、カタカナ、漢字など)、図形(点、線、複雑な図形など)を表示することができます。

アルファベットなどの文字を表示する画面を文字画面、図形を表示する画面をグラフィック画面とよび、ディスプレイの画面には、この2種類の画面を重ねて表示しています。

なお、グラフィック表示を行うためには、MS-DOSの**CONFIG. SYS**ファイルに**GRAPHIC. SYS**ファイル(デバイスドライバ)を登録しておく必要があります。

出荷時は登録されてますが、登録されてない場合はBASIC起動時に"グラフィック機能は利用できません"というメッセージが表示されます。

文字画面

文字画面の表示文字数(横×縦)は**init"CRT1:"**または**width**文を使って、次の6通りの設定をすることができます。

ただし、横の文字数は半角文字のときの数です。(全角文字のときは半分の数になります。)

設定		表示文字数(横×縦)
init "CRT1:40, 12"	**width 40, 12**	40×12
init "CRT1:80, 12"	**width 80, 12**	80×12
init "CRT1:40, 20"	**width 40, 20**	40×20
init "CRT1:80, 20"	**width 80, 20**	80×20
init "CRT1:40, 25"	**width 40, 25**	40×25
init "CRT1:80, 25"	**width 80, 25**	80×25

- 横40桁の指定をすると2つの文字画面を使用することができ、一方の画面のみを表示させたり、２つの画面を重ね合わせることができるようになります。
  これらの指定は**screen**文で行います。

グラフィック画面

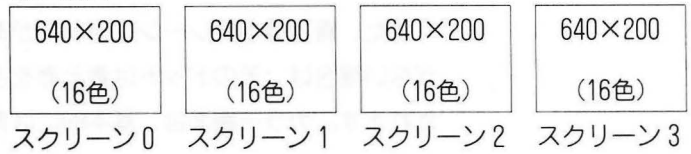
グラフィック画面の解像度(横×縦)、同時表示色は**init"CRT 2 :"**文を使って、次のような設定をすることができます。

設　　定	解像度(横×縦)	同時表示色	最大使用画面数	
init"CRT2:640, 200, 16"	640×200	16	8	①
init"CRT2:640, 200, 65536"	640×200	65536	2	②
init"CRT2:640, 400, 16"	640×400	16	4	③
init"CRT2:640, 400, 65536"	640×400	65536	1	④

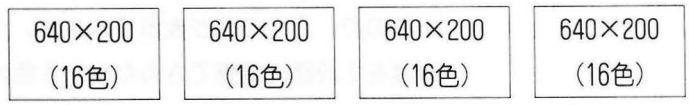
- 16色、または65536色表示を行うためには、アナログ入力方式のディスプレイが必要です。

①～④の設定での使用画面(スクリーン)は次のようになります。

①の設定

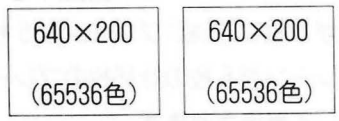


スクリーン0　スクリーン1　スクリーン2　スクリーン3



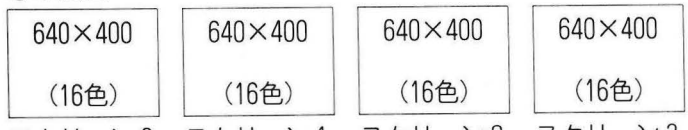
スクリーン4　スクリーン5　スクリーン6　スクリーン7

②の設定



スクリーン0　スクリーン1

③の設定



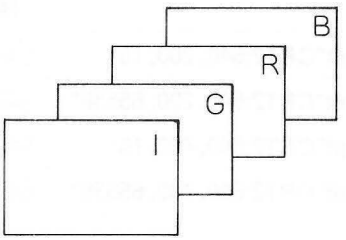
スクリーン0　スクリーン1　スクリーン2　スクリーン3

④の設定

```
640×400
(65536色)
```

スクリーン0

①や③の設定のとき(16色モード)のスクリーンは、次の図のように4つのプレーンによって構成されています。



4枚のプレーンを
合成したものが
スクリーン

Bが青、Rが赤、Gが緑というように設定されており、この3つのプレーンは合成されて1つのスクリーンとして表示されます。

たとえば、赤のプレーンにドットがあり、青と緑のプレーンにドットがない場合は、そのドットは赤で表示されます。

また、青と赤のプレーンにドットがあり、緑のプレーンにドットがない場合は、そのドットは青と赤を合成したマゼンタ(紫)で表示されます。カラー画面は、基本的には青、赤、緑の3枚のプレーンを重ねて表示したものです。3枚のプレーンの重ね合わせは8通りあるので、8色の色が表示できます。さらに、Iのプレーンにより輝度を2段階に調整できるため、8色×2＝16色が表示できることになります。

②や④の設定のとき(65536色モード)のスクリーンは、I(輝度)プレーンが1枚、G(緑)プレーンが5枚、R(赤)プレーンが5枚、B(青)プレーンが5枚の計16枚のプレーンが合成されて1つのスクリーンとして表示されます。

カラーコード

文字やグラフィックの表示色は色に番号を対応させたカラーコードにより指定します。BASICでは8色、16色、65536色の3種類のカラーコードがあります。

16色コード

グラフィック画面の16色モードでの色コードで、R.G.B.I.の4種の色信号により0～15の値で表します。

R …Red 赤
G …Green 緑
B …Blue 青
I …Intensity 明るさ

4つの信号の組み合わせで次の色が表現できます。

16色コード	表示色	色信号 I	G	R	B
0	黒	0	0	0	0
1	青	0	0	0	1
2	赤	0	0	1	0
3	マゼンタ	0	0	1	1
4	緑	0	1	0	0
5	シアン	0	1	0	1
6	黄	0	1	1	0
7	白	0	1	1	1

16色コード	表示色	色信号 I	G	R	B
8	灰	1	0	0	0
9	明青	1	0	0	1
10	明赤	1	0	1	0
11	明マゼンタ	1	0	1	1
12	明緑	1	1	0	0
13	明シアン	1	1	0	1
14	明黄	1	1	1	0
15	明白	1	1	1	1

0…信号が無い　　1…信号がある

8色コード

文字画面の色コードのことで、基本的にはR.G.B.の3種の色信号の組み合わせによって0～7の値で表します。

ただし、コードが0の黒以外はI信号を使用していますので、実際の表示色はコードの1～7が16色コードの9～15に対応します。

8色コード	表示色	色記号 IGRB
0	黒	0 0 0 0
1	明青	1 0 0 1
2	明赤	1 0 1 0
3	明マゼンタ	1 0 1 1
4	明緑	1 1 0 0
5	明シアン	1 1 0 1
6	明黄	1 1 1 0
7	明白	1 1 1 1

0…信号が無い　1…信号がある

65536色コード

グラフィック画面の65536色モードでの色コードで、R.G.Bの3種の色信号がそれぞれ32段階、I(輝度)信号が2段階に変化する組み合わせで表示され、0～65535の値で表示します。(10進数指定の場合)

コード番号は16桁の2進数で表すと、上位5桁でG信号、中位5桁でR信号、下位5桁でB信号、最下位1桁でI信号を指定します。

(例)　10000 10000 00000 1 …2進数
　　G　R　B　I…輝度、色信号

輝度	信号名			
	G	R	B	I
明るい	11111	11111	11111	1
	11110	11110	11110	1
	∫	∫	∫	∫
暗い	00000	00000	00000	0

R.G.Bの各階調を増減させる機能(**add, sub**)があります。くわしくは「9. グラフィックの表示」を参照してください。

なお、実際にコード指定する場合は、2進数で指定するときは2進数を表す**&B**を最初に付けます。

(例)　一番明るい黄色のコードは次のようになります。
　　&B1111111111000001

10進数でコード指定する場合は、上記のような2進数を10進数に変換して指定すると考えると、わかりやすいでしょう。

なお、10進数に変換したときに－32768～－1（2進数で1000000000000000～1111111111111111）となるコードは、実際には32768～65535のコードに相当します。

文字の色指定

文字の表示は、8色で表現することが可能です。

色指定は8色コードを使用します。

この8色コードの1～7の色は、16色コードの9～15に対応した色となります。

8色コードと対応する16色コードは次のようになります。

8色コード	0	1	2	3	4	5	6	7
16色コード	0	9	10	11	12	13	14	15

4096色パレット機能については、「グラフィックの色指定」を参照してください。

注） 文字の表示色は、パレットボード(**MZ-1M10**)（別売）を取り付け、4096色パレット機能を使用して表示色を設定したときは、表示色が変化します。☜コラム

グラフィックの色指定

グラフィックの表示色モードは、16色、65536色の2つのモードがあり、**init"CRT 2:"**文で設定を行います。

パレットコードとパレット機能

16色モードの色指定はパレットコードで指定します。

カラーコードは色そのものを指定するコードで、16色コードであれば0＝黒、1＝青、2＝赤というように、必ず数字に対する色が決まっています。

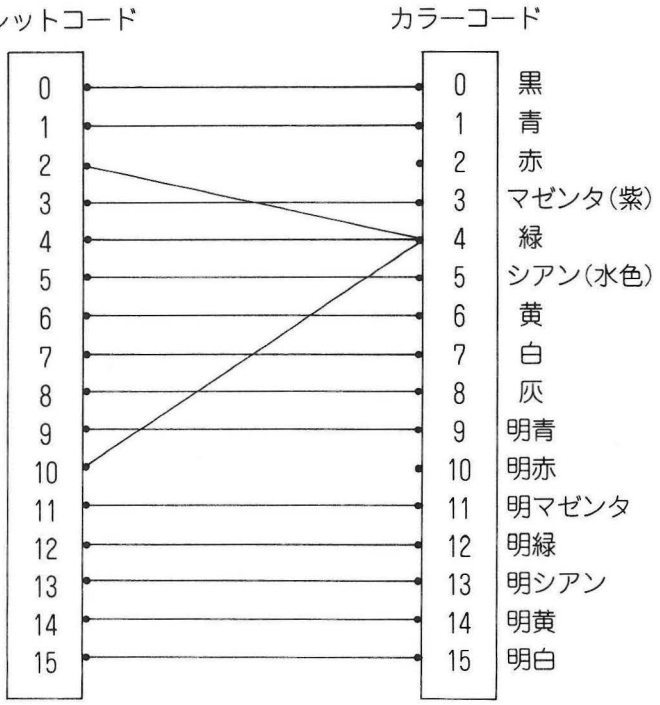
これに対してパレットコードも色を指定するコードですが、カラーコードのように数字に対する色が決まっていないので、自由に色を設定することができます。この機能をパレット機能と呼びます。

初期状態は、パレットコード＝カラーコードとなっています。

パレットコードの色の設定は**color＝**文を使います。

（例）パレットコードとカラーコードを、次のような関係に設定してみます。

```
init "CRT2:"
color=(2,4),(10,4)
```



このように設定すると緑をパレットコード4だけでなく、パレットコード2と10で表示することもできるようになります。

なお、パレットコードとカラーコードの対応を変えることにより、画面の表示色を瞬時に変更することも可能です。

4096色パレット

4096色パレット機能は、パレットボード(**MZ-1M10**)(別売)を取り付けたときに可能です。

G(緑)、R(赤)、B(青)の3原色に対してそれぞれ16段階の輝度(明るさ)レベルを設定して4096色(=16×16×16)を表現することができます。

その中から、パレットに設定されたカラーコードに対して指定できるようになります。なお、カラーコード0に対しては指定できません。

16色モードでの色指定

色指定は0～15の**パレットコードで指定**します。

また、パレットボード(**MZ-1M10**)(別売)を取り付けたときは、4096色パレットの機能を使って表示色を設定することもできます。

65536色モードでの色指定

色指定は0～65535の**カラーコードで指定**します。

パレットコードで指定することはできません。

プライオリティ機能

文字画面とグラフィック画面を同時に表示させた場合や、文字画面を2画面同時に表示させた場合に、グラフィック画面と文字画面の重なるところ、文字画面どうしの表示が重なるところではどちらかの表示が優先されて表示されます。

本BASICでは、この表示の優先順位(プライオリティ)を指定することができます。

文字とグラフィック表示

グラフィック画面が16色モードのときに、各パレットコードに対して文字との優先順位を**priority**文で設定することができます。

初期設定は文字の方がすべてのパレットコードに対して優先します。

グラフィック画面が65536色モードのときは、文字画面が常にグラフィック画面に優先します。

文字画面どうしの表示

文字画面は40桁表示の場合、2画面もつことができ、**screen**文により画面の指定や重ね合わせ表示の指定をすることができます。

重ね合わせ表示の優先順位は固定で文字スクリーン番号0の方が1より優先されます。

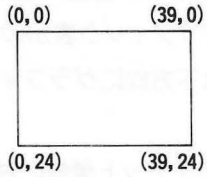
文字座標

文字座標は、文字画面に設定されている座標です。

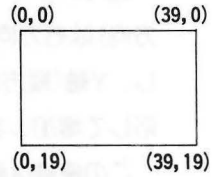
文字座標は文字の表示位置の移動や表示範囲の指定などに使用されます。

文字座標は画面の左上の位置を原点(0,0)として横方向は右へ、縦方向は下へ向かって値が増加します。

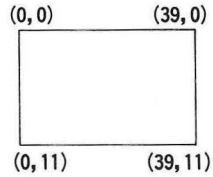
表示文字数により次の図のように設定されます。



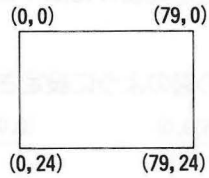
40桁×25行



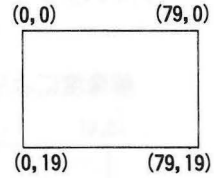
40桁×20行



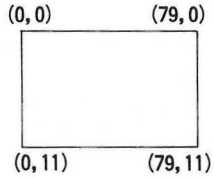
40桁×12行



80桁×25行



80桁×20行



80桁×12行

文字座標により指定を行う命令には次のものがあります。

console	……	文字のスクロール範囲の指定。
console@	……	文字の表示出力範囲の指定。
color@	……	指定範囲の文字色を変更。
crev@	……	指定範囲の文字表示の反転。
cflash@	……	指定範囲の文字表示の点滅。
locate	……	文字表示位置の指定。
get%	……	指定範囲の文字データを配列に取り込む。
put%	……	配列の文字データを指定範囲へ表示する。

グラフィック座標

グラフィック座標は、グラフィック画面に設定されている座標です。

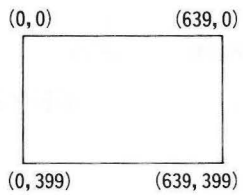
グラフィック座標には、絶対スクリーン座標、スクリーン座標、ワールド座標の3種類の座標があります。

絶対スクリーン座標

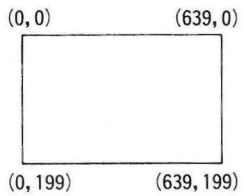
絶対スクリーン座標は、画面の左上を原点(0,0)としてX軸(横方向)は右方向にグラフィック表示の1ドット単位に対応して増加し、Y軸(縦方向)は下方向にグラフィック表示の1ドット単位に対応して増加します。

この座標は画面の1ドット単位に対応した最も基本となる座標で、グラフィックの描画命令によって図形の描かれる範囲の指定や、ディスプレイへの表示範囲の指定のときに使用されます。

解像度により次の図のように設定されます。



640×400ドット



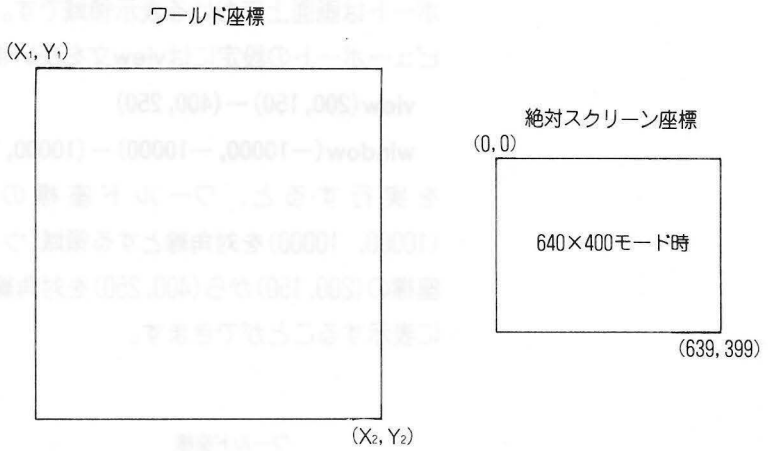
640×200ドット

ワールド座標と絶対スクリーン座標

ワールド座標は、画面の絶対スクリーン座標より、さらに大きな座標を扱うためのものです。

縦横それぞれの方向に−1.7014118E38〜1.7014118E38(単精度実数型の範囲)まで指定でき、負の座標指定をすることもできます。

次の図は、ワールド座標と絶対スクリーン座標です。



$(X_1, Y_1) = (-1.7014118E38, -1.7014118E38)$
$(X_2, Y_2) = (\ 1.7014118E38,\ \ 1.7014118E38)$

ウィンドウ

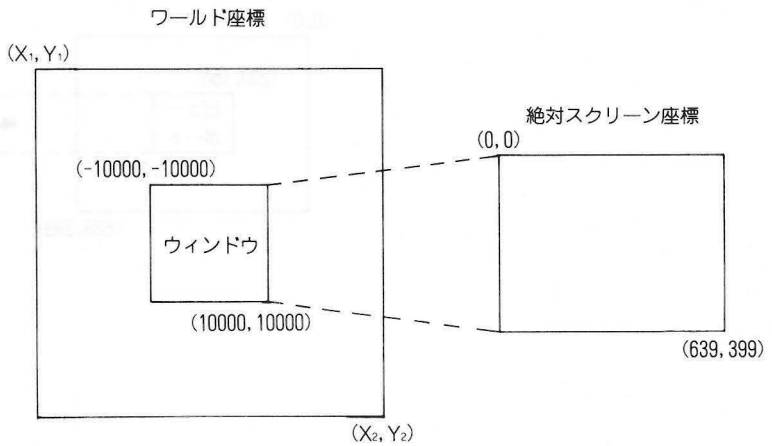
絶対スクリーン座標は見ることができますが、大きなワールド座標は見ることができません。

そこで、**window**文を使ってワールド座標の領域を指定して、絶対スクリーン座標に割り当てることによって、ワールド座標の一部を表示できるようにします。

たとえば、解像度が640×400ドットに設定されているとき、

window(−10000, −10000)−(10000, 10000)

を実行すると、ワールド座標の(− 10000, − 10000)から(10000, 10000)を対角線とする領域を絶対スクリーン座標全域(グラフィック画面全体)に表示することができます。



このワールド座標の中で指定した領域を、ウィンドウと呼びます。

ビューポートとスクリーン座標

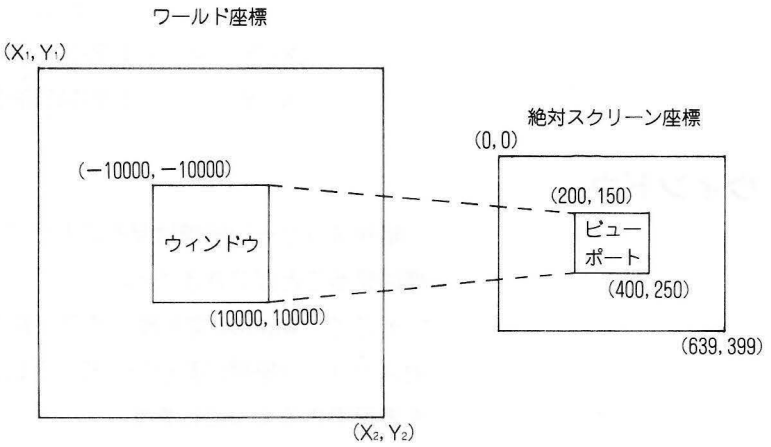
ウィンドウはワールド座標における表示領域でしたが、ビューポートは画面上における表示領域です。

ビューポートの設定には**view**文を使います。たとえば、

view(200, 150)－(400, 250)

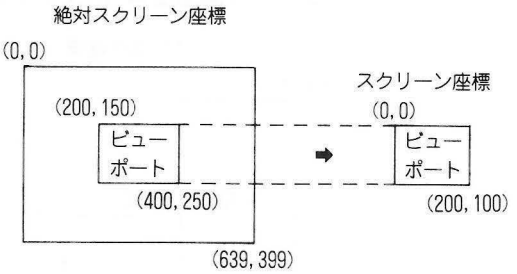
window(－10000, －10000)－(10000, 10000)

を実行すると、ワールド座標の(－10000, －10000)から(10000, 10000)を対角線とする領域(ウィンドウ)を絶対スクリーン座標の(200, 150)から(400, 250)を対角線とする領域(ビューポート)に表示することができます。



このビューポートの左上を原点(0, 0)として、画面の1ドットに対応する座標をスクリーン座標と呼びます。

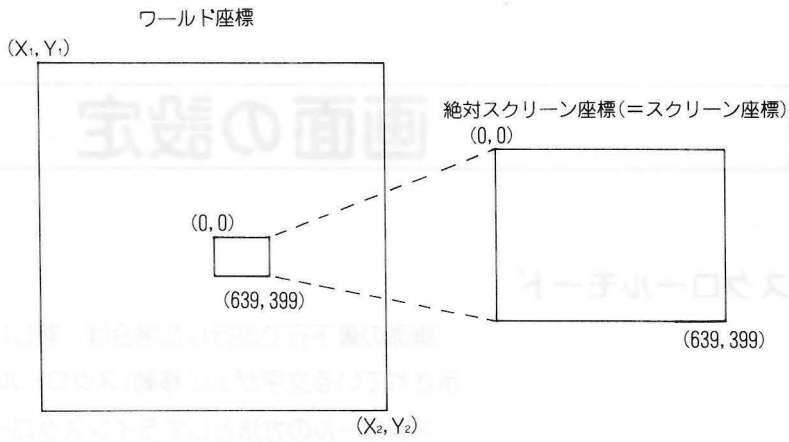
先の例ではスクリーン座標は次のようになります。



初期設定

BASICの起動後やグラフィック画面の初期化を行ったときは、ビューポートが、画面いっぱいに設定されていますので、絶対スクリーン座標とスクリーン座標は同じになります。

また、ウィンドウもスクリーン座標と同じ領域が自動的に設定されます。



8 文字の表示

キーボードから入力された文字やプログラムリストは文字画面に表示されます。

画面の設定

スクロールモード

画面の最下行で改行した場合は、新しい行を表示するために、表示されている文字が上に移動(スクロール)します。

スクロールの方法としてラインスクロールとスムーススクロールがあり、ラインスクロールは、行単位で移動しますが、スムーススクロールは文字を構成するドット単位でなめらかに移動します。スムーススクロールには高速モードと低速モードがあります。

スムーススクロールのときは**console**文による表示範囲の指定ができなくなり、一番下の行は表示に使用されなくなりますのでファンクションキーを表示させることはできません。

また、40桁表示の場合は、文字画面を2画面もちますので、スムーススクロールはできなくなります。

スクロール、ファンクションキー表示、画面数の組み合わせは次のようになります。

桁　数	スクロール	ファンクションキー表示	画面数
80	スムース	×	1
	ライン	○	
40	ライン	○	2

BASICを起動した直後は、80桁のラインスクロールモードに設定されています。

なお、スクロールモードの設定には**init"CRT1:"**文を使います。

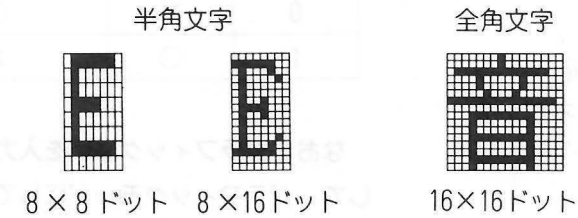
文字フォント

アルファベットなどの半角文字には8×8ドットと8×16ドットの2種類のドット構成があります。
文字画面の行数により、文字フォントは、次のようになります。

行数	フォント	
	8×8ドット	8×16ドット
25, 20	○	○
12	×	○

20または25行モードのときには8×8と8×16ドット(初期設定)のどちらかを選択することができます。設定には**init"CRT1："**文を使います。

文字フォントの例



スクロール範囲の設定

console文を使用してスクロール範囲を設定することができます。

たとえば、画面の左端から40桁と、画面の上端から10行をスクロール範囲に設定するには次のようにします。

console 0, 10, 0, 40 [↵]

このあと、**list**命令を実行した場合はこの設定された範囲内(40桁×10行)で表示が行われ、それ以外のところには表示されません。また、文字画面の消去を実行したときもこの範囲だけが消去されます。この設定を解除するには、**console**文で画面全体を指定するか[コントロール]+[D]キーを押します。

カーソルの移動

カーソルを移動して任意の位置から文字を表示する場合には、**locate**文を使用してカーソルを任意の位置に移動することができます。

ファンクションキーの内容表示

文字画面の最下行にはファンクションキーの内容を表示することができます。

ただし、最下行が**console**文によるスクロール範囲に含まれている場合やスムーススクロールモードのときは表示されません。表示状態を設定するには**klist**文を使います。

漢字とグラフィック文字

漢字とグラフィック文字は、同時に表示することはできません。どちらの文字を表示するかは、**kmode**文で設定します。

☜コラム

kmode	漢　字	グラフィック文字
0	×	○
1	○	×

グラフィック文字についてはオーナーズマニュアルを参照してください。

なお、グラフィック文字を入力するときは[GRAPH]キーを押して、グラフィックモードにしてから、入力してください。

文字のアトリビュート

画面に表示される文字の１文字ごとに色を指定したり、反転や点滅の表示をすることができます。

この色や反転、点滅のことを文字のアトリビュート(属性)といいます。

表示色の指定

文字の表示色は**color**文で指定します。

指定後に表示される文字の色はこの指定色になります。

文字カラーは０～７の値で指定します。

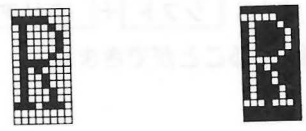
リバース表示

文字のリバース(反転)は**crev**文で設定します。

文字のフォントは8×8または8×16ドットで構成され、リバース状態は通常光っている部分を消し、消えている部分を光らせます。PCG１～PCG３を組み合わせて、ドット単位で色指定ができるPCGは、各ドット色が補色になります。☜コラム

PCGについては「10 PCGの定義のしかた」を参照してください。

通常表示　　リバース表示



ブリンク表示

文字のブリンク(点滅)は**cflash**文で設定します。

ROM CGとPCG

文字画面に表示される文字は、キャラクタジェネレータ(CG)という部分から呼び出された文字構成(パターン)が表示されますが、CGにはあらかじめ固定されたパターンが入っているROM CGと、ユーザーが自由に定義できるPCGがあります。

PCGの定義には、**def chr$**文を使用します。

なお、通常はROM CGのパターンが表示されるようになっていますので、PCGパターンを表示するには**cgen**文でROM CGとPCGの表示を切り替えます。

PCGについて、くわしくは「10 PCGの定義のしかた」を参照してください。

文字画面の読み込み

GET%とPUT%

get%は文字画面の情報を配列変数に読み込む命令です。

また、**put%**は逆に配列変数の情報を画面に表示する命令です。このget%とput%を組み合わせて用いることによって、文字画面に表示されている文字を別の位置に移動させることができます。get%を実行する場合には、画面の情報が入るだけの配列変数をあらかじめ定義しておく必要があります。

画面の消去

文字画面に表示されている内容を消去するときは、**cls**文を使用します。

また、シフト＋クリア/ホームと入力しても、文字画面を消去することができます。

9 グラフィックの表示

グラフィック表示を行うためには、MS-DOSの**CONFIG. SYS**ファイルに**GRAPHIC. SYS**ファイル(デバイスドライバ)を登録しておく必要があります。

出荷時には登録されていますが、登録されていない場合はBASIC起動時に"グラフィック機能は利用できません"というメッセージが表示されます。

グラフィック命令の座標指定

グラフィック画面の座標には、絶対スクリーン座標、スクリーン座標、ワールド座標の3種類があります。

各グラフィック命令で指定する座標は、このうちどれかを指定することになりますが、その関係は次のようになります。

絶対スクリーン座標	スクリーン座標	ワールド座標
view, view@, mouse,mouse(n)	**pattern, symbol, color replace, move, get@, put@, gsave, gload**	**pset, preset, line, connect, circle, poly, paint**

BASIC起動時には、ワールド座標はスクリーン座標と同じになっていますので**window**文で座標を変えなければ、2つの座標の区別はありません。

座標の初期化をするには、コントロール+Dと入力します。

グラフィック描画時のペン機能

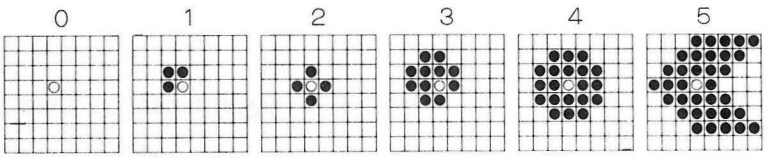
グラフィックの描画命令でドットや線で描画する場合のペン形状を選択または定義ができます。

ペン形状は指定された座標に打たれる点を8×8までの複数のドットを使って指定できます。

本BASICには0～5の6つのペンが用意されており、あらかじめペン形状が設定されていますが、1～5のペンについては形状を定義することもできます。

このペン形状を選択することにより、特定パターンの点を打ったり、太い線を引くことができます。

BASICで用意されているペン形状は次のとおりで、左上から右下へ向って4ドット目が描画の中心座標になります。



ペン形状の選択および定義はpen文により行います。

指定ペン形状で描画する命令には次のものが用意されています。

pset …… 点をセットします。

preset …… 点をリセットします。

line …… 線を引きます。

connect …… 連続した線を引きます。

circle …… 円を描きます。

poly …… 多角形を描きます。

グラフィック描画時の機能指定

グラフィックの描画命令で指定された座標の各プレーンにデータを書き込むときに論理演算などの機能指定が行えます。

機能指定は次のものがあります。なお色指定とは、16色モードのときは、パレットコードを指します。

指定可能な機能は各命令の項を参照してください。

- **pset**

この指定は指定色に該当するプレーンはセットされ、該当しないプレーンはリセットされます。

- **preset**

色の指定に関係なくバックグラウンドカラーでの**pset**になります。

ただし、**put@**文、**move**文、**pattern**文による描画の場合は補色による**pset**になります。

- **and**

16色モードのとき、この指定は、色指定に該当するプレーンを1として、すでに書かれている各プレーンごとのデータとの論理演算**AND**を行います。したがって、色指定に該当するプレーンがすでにセットされている場合は、そのままセットされ、セットされていなければリセットし、色指定に該当しないプレーンはリセットされます。

- **sub**

65536色モードのとき、この指定はR.G.B.各階調を減少させます。16色モードは使用できません。

(例)**&B0000000000000100**のカラーコードで、**sub**指定により描画を行うと、元の画面よりもB(青)の階調が2段階低い色で描画されます。

- **or**

16色モードのとき、この指定は、色指定に該当するプレーンを1として、すでに書かれている各プレーンごとの論理演算**OR**を行います。したがって、色指定に該当するプレーンはセットされ、色指定に該当しないプレーンのドットはそのままとなります。

- **add**

65536色モードのとき、この指定はR.G.B.各階調を増加させます。16色モードでは使用できません。

(例)**&B0000000001000010**のカラーコードで、**add**指定により描画を行うと、元の画面よりもR(赤)B(青)の階調が1段階高い色で描画されます。

• **xor**

この指定は、色指定に該当するプレーンを1として、すでに書かれている各プレーンごとのデータとの論理演算**XOR**を行います。したがって、指定色に該当するプレーンがセットされていればリセットし、リセットされていればセットします。また指定色に該当しないプレーンのドットはそのままとなります。

• **scr**

この指定は**single color replace**の略で、**pattern**文や**move**文などの面積をもつ描画に対して行われ、色コードに該当するプレーンのみの書き換えを行います。面積をもつ場合セットする座標とリセットする座標ができ、該当プレーンのみセット、リセットを行います。65536色モードでは利用できません。

• **fpset**

この指定は、**pattern**文において行われ、描画データのドットのある部分(フォント)のみ、**pset**になり、ドットの無い部分は何も作用しません。

命　令	16色／65536色 共通の描画機能	16色時のみ の描画機能	65536色時のみ の描画機能
pset	pset, preset, xor	and, or	add, sub
preset	pset, preset, xor	and, or	add, sub
line	pset, preset, xor	and, or	add, sub
connect	pset, preset, xor	and, or	add, sub
circle	pset, preset, xor	and, or	add, sub
poly	pset, preset, xor	and, or	add, sub
pattern	pset, preset, xor, fpset	and, or, scr	add, sub
symbol	pset, preset, xor	and, or	add, sub
move	pset, preset, and, or, xor	scr	なし
put@	pset, and, or, xor	preset	なし

描画機能一覧表

上記以外の描画命令には、描画機能は指定できません。

グラフィックの描画

点を描く

点を描くにはpsetまたはpreset文を使います。

pset文が任意の座標のドットをセットして点を表示するのに対して、preset文はドットをリセットして消します。

線を描く

線を描くにはline文を使います。

line文では、実線を引く他に、次のようなことができます。

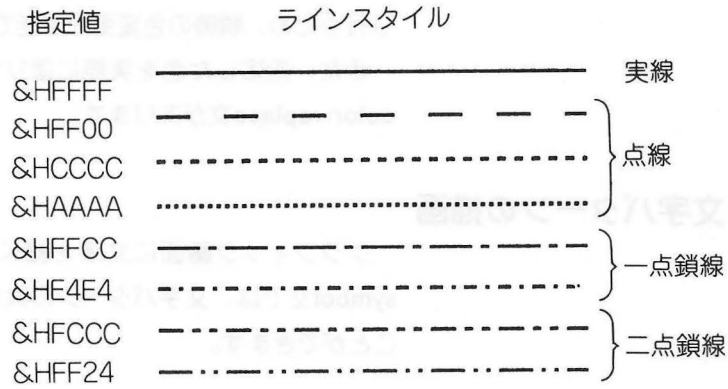
①ラインスタイルを指定することによって、点線や一点鎖線など、自由な形の線を引くことができます。

②任意のラインスタイルで、長方形を描くことができます。

③四角形を描いて、その中を任意の色またはタイルパターンで塗りつぶすことができます。

ラインスタイルは、16ドット単位のドットの並びを整数型データで指定します。この指定を省略すると、実線で引かれます。

次に、主なラインスタイルとその指定値を示します。



line文で線を引くときに指定する点は2点ですが、connect文では2点以上を指定して、線を続けて引くことができます。

connect文で座標を連続して指定すると、折れ線や多角形を描くこともできます。

正多角形を描く

多角形は、line文やconnect文を使っても描くことができますが、各頂点の座標を計算して指定をする必要があります。

poly文を使うと、circle文で円を描くのと同様に、中心座標や半径などの指定だけで正多角形を描くことができます。

円を描く

円、楕円および円弧は、circle文を使って描くことができます。

楕円を描くときは、垂直半径と水平半径の比率を指定します。

また、円弧を描くときは、開始角と終了角を指定します。

色を塗る

グラフィックの線で囲まれた任意の形の図形は、paint文を使って塗ることができます。

塗る色は、パレットコードまたはタイルパターンで指定します。

色を変える

グラフィック画面に表示されているグラフィックの色はcolor＝文を使って瞬時に他の色に変えることができます。(16色モードのみ)

color＝文での色変更は、実際に塗りつぶして変更するのではなく、パレットコードに対応させるカラーコードを変更することにより行うため、瞬時の色変更が可能です。

また、指定した色を実際に塗りつぶして変更する命令として、color replace文があります。

文字パターンの描画

グラフィック画面に文字を描くには、symbol文を使います。symbol文では、文字パターンの大きさや表示する向きを指定することができます。

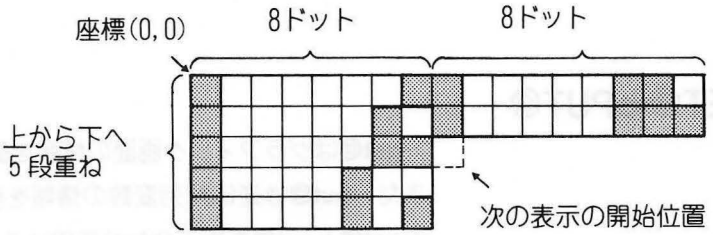
グラフィックパターンの描画

pattern文を使用して、グラフィック画面にパターンを描くことができます。

たとえば、次のプログラムを実行すると、下図のように表示されます。

```
10  A$＝hexchr$("81828384858687")
20  pattern(100,100),—5,A$
```

```
30  end
```



グラフィック画面の読み込み

GET@とPUT@

get@はグラフィック画面の情報を配列変数に読み込む命令です。また、**put@**は逆に配列変数の情報を画面に表示する命令です。この**get@**と**put@**を組み合わせて用いることによって、グラフィック画面に表示されている図形を別の位置に移動させることができます。**get@**を実行する場合には、**dim**文で画面の情報が入るだけの配列変数をあらかじめ定義しておく必要があります。

GSAVEとGLOAD

グラフィック画面の情報は、**gsave**命令を使ってファイルに記録することができます。記録したグラフィックファイルの内容は、**gload**命令を使って呼び出しグラフィック画面に表示することができます。

グラフィック画面の消去

グラフィック画面に表示されている内容を消去するときは**cls**文を使用します。

画面の初期化

画面の設定をするには、文字画面であれば**init "CRT1:"** 文を使い、グラフィック画面であれば**init "CRT2:"** 文を使います。

これらの命令を使って画面の初期化を行うこともできますが、表示されている文字が消去されたり、解像度が変わってしまうことがあります。このようなとき、コントロール+Dと入力することによって、次の画面状態を初期化することができます。

- 表示色の初期化(**color**)
- 文字属性の初期化(**cgen, cflash, crev**)
- 入出力画面の初期化(**screen**)
- カーソルの表示(**locate,,2**)
- スクロール範囲の初期化(**console**)
- ディスプレイ出力範囲の初期化(**console@**)
- ウィンドウ、ビューポートおよび表示出力の初期化
  (**view, view@, window, roll@**)
- パレットの初期化(**color=**)
- 画面の優先順位の初期化(**priority**)

たとえば、次のような状態になったときなどにコントロール+Dと入力してください。

文字が反転や点滅状態になってもとにもどらないとき
押したキーと表示される文字が違っていたり、わからないパターンが表示されるとき
文字や画面の色をもとにもどしたいとき
シフト+クリア/ホームキーを押しても文字が消えずに残るとき
キーを押しても何も表示されないとき
画面のグラフィックが**cls**文でも消えないとき
キー入力した文字がグラフィックのうしろにかくれてしまうとき
グラフィックで描いた図形が指定した大きさ、色と合わないとき
コンピュータの音を止めたいとき

また、画面に表示されている内容を消去するときは**cls**文を使用します。

cls 1　…　文字画面のみ消去します。
(シフト+クリア/ホームと同じ)

cls 2　… グラフィック画面のみ消去します。

cls 3　… 文字およびグラフィック画面を消去します。

10 PCGの定義のしかた

英数字、カナ文字、グラフィック文字、漢字などの文字は、ドットが集まって構成されています。

1つの文字を構成するドットは、半角文字で8×8または8×16ドット、全角文字で16×16ドットあり、このドットパターンがROM CGに記憶されています。

これに対してPCGがあり、ユーザーがドットパターンをデザインして、記憶させておくことができます。

(PCGはProgrammable Character Generator(プログラマブル・キャラクタ・ジェネレータ)の略です。PCGにはPCG0～PCG3がありそれぞれ2Kバイトのメモリをもっています。)

PCGは、PCG0～3の単独の使用時は単色になります。PCG1～3に青、赤、緑の色を割り当て、合成するとドット単位の8色指定が可能になります。

PCGには、次の3つのサイズがあります。

8×8ドット
8×16ドット
16×16ドット

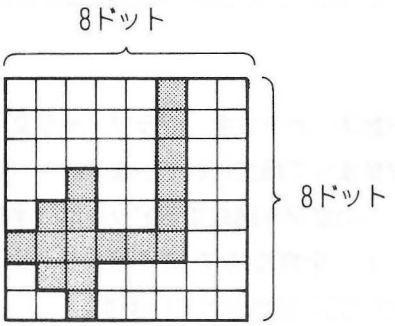
8×8ドットの定義

8×8ドットのドット単位8色のPCGを定義するには、24バイトの文字列が必要です。

これは、その文字列を表すビットパターンが1個のPCGを形成することによります。すなわち、1バイトの文字コードのビットパターンが横8×縦1ドットのパターンを表し、8バイトの文字列で横8×縦8ドットのドットパターンを構成することになります。

カラーで表示するには、青、赤、緑の3画面分必要なので、24バイト(=8バイト×3画面)の文字列が必要となります。この24バイトの文字列のうち、最初の8バイトが青、次の8バイトが赤、最後の8バイトが緑のドットパターンを表します。

（例） 次の図形を赤字に黄色で作ってみましょう。



これを２進数のパターンで表わし、さらに16進数表現に変換すると次のようになります。

（２進数表現）		（16進数表現）
00000100	→	04 ……①
00000100	→	04 ……②
00000100	→	04 ……③
00100100	→	24 ……④
01100100	→	64 ……⑤
11111100	→	FC ……⑥
01100000	→	60 ……⑦
00100000	→	20 ……⑧

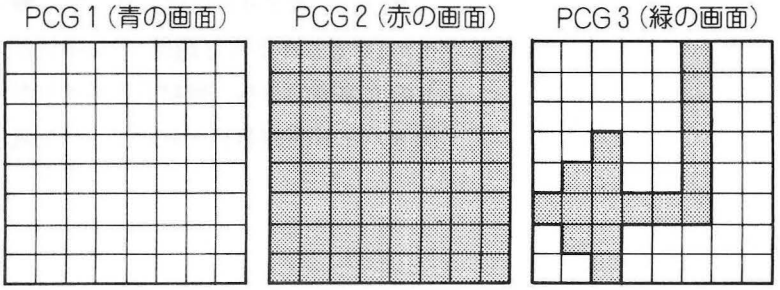
したがって、このドットパターンは、

hexchr $（"0404042464FC6020"）

① ② ③ ④ ⑤ ⑥ ⑦ ⑧

となります。

黄色は赤と緑を合成して作られます。赤地に黄色のパターンを作るためには、青の画面、赤の画面、緑の画面をそれぞれ次のように定義しなければなりません。



000000000 → 00　　11111111 → FF　　00000100 → 04
000000000 → 00　　11111111 → FF　　00000100 → 04
000000000 → 00　　11111111 → FF　　00000100 → 04
000000000 → 00　　11111111 → FF　　00100100 → 24
000000000 → 00　　11111111 → FF　　01100100 → 64
000000000 → 00　　11111111 → FF　　11111100 → FC
000000000 → 00　　11111111 → FF　　01100000 → 60
000000000 → 00　　11111111 → FF　　00100000 → 20

したがって、赤地に黄色の先のパターンを定義するのに必要な文字列は、

```
hexchr$ ("0000000000000000FFFFFFFFFFFFFFFF0404042464
FC6020")
```

となります。

このサイズのパターンを定義できるコードは0～255で、定義できるパターンの数は次のようになります。

PCG 1～3の組み合わせ（カラー）　：　256個
PCG 0～3の単独での使用(単色)　：1024個

たとえば、コードの64に定義するには**def chr$**文を使って次のようにします。

```
def chr$(64)＝hexchr$("0000000000000000FFFFFFFFFFFFFF
FF0404042464FC6020")
```

こうして定義した文字は、**cgen 1**を実行後、**print**文を使って表示することができます。(カラーモード時)

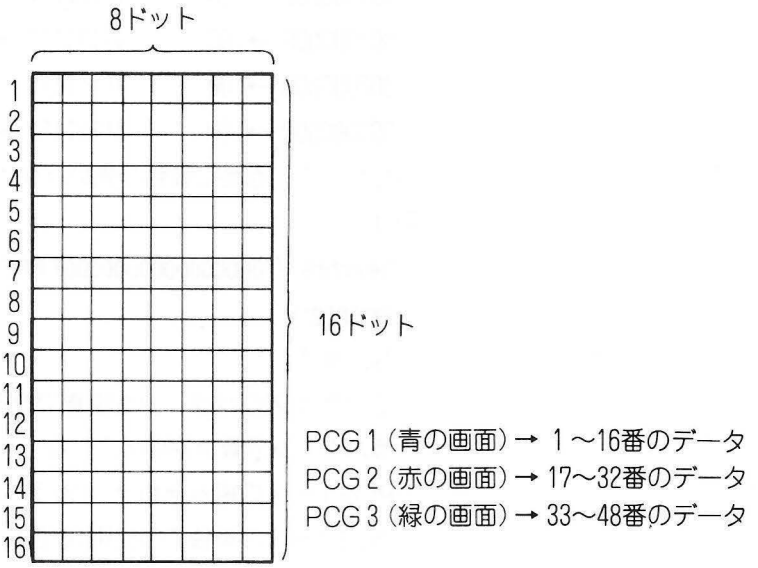
```
10  init "CRT 1：40,25,0"
20  A$=hexchr$("0000000000000000FFFFFFFFFFFFFFFF0404
    042464FC6020")
30  def chr$(64)＝A$
40  cgen 1
50  print chr$(64);
60  cgen 0：end
```

(注) 文字コード0～31の文字はコントロールコードになりますので、通常の**print**文ではコントロールが実行されます。

コントロールコードを実行せずに定義した文字を画面に表示する場合は、**open"O",**＃<ファイル番号>,**"CRT 1 :CG"**を実行し、**print**＃<ファイル番号>,<データ>で書き込みを行ってください。

8×16ドットの定義

8×16ドットのドット単位8色のPCG定義をするには、8×8ドットが縦2倍にのびたパターンのため、48バイトの文字列が必要です。



デザインしたパターンを、8×8ドットのときと同様に2進数のパターンで表し、さらに16進数表現に変換してコードの文字列にします。

このサイズのパターンを定義できるコードは&H100、&H102、&H104、……、&H1FEで、定義できるパターンの数は次のようになります。

PCG 1～3の組み合わせ　（カラー）：128個
PCG 0～3の単独での使用（単色）　：512個

（例）あるパターンをコードの&H100に定義するには、

```
def chr$(&H100)＝hexchr$("HHH……HHH")
                        96桁（48バイト）
```

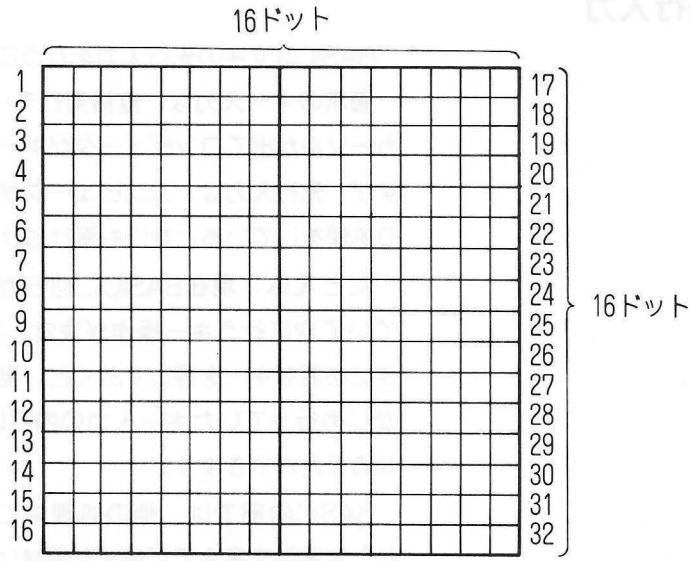
のようにします。

こうして定義したパターンは、8×8ドットのときと同様にして表示できます。

16×16ドットの定義

16×16ドットのPCG(単色)を定義するには、32バイトの文字列が必要です。

16×16ドットのPCG定義は外字ユーティリティで行うことができます。
くわしくはMS-DOSマニュアルを参照してください。



PCG 1 (青の画面) → 1～32番のデータ
PCG 2 (赤の画面) → 33～64番のデータ
PCG 3 (緑の画面) → 65～96番のデータ

デザインしたパターンを、8×8ドットのときと同様に2進数のパターンで表し、さらに16進数表現に変換してコードの文字列にします。

このサイズのパターンを定義できるコードは&HEC 9 F～&HECFC、&HED40～&HED 7 E、&HED80～&HEDE 2で、定義できるパターンは256個です。

(例) あるパターンをコードの&HEC 9 Fに定義するには、

```
def chr$(&HEC9F)＝hexchr$("HHH……HHH")
```

64桁(32バイト)

こうして定義したパターンは、**kmode 1**を実行後、**print**文を使って表示することができます。

```
kmode 1
print chr$(&HEC9F);
```

11 キーボードの制御機能

先行入力

BASICは文字の先行入力を行うことができます。

通常のキー入力は、直接実行モードの状態や**input**文の実行時にカーソルが出てコンピュータがキー入力待ちになったときに行いますが、先行入力は、コンピュータがキー入力待ちのとき以外の、他の処理をしているときにも受け付けることができます。

たとえば、現在BASICに何らかの時間のかかる処理を行わせていて次に行うキー操作が決定している場合、あらかじめ次の操作に必要なキーを押しておくと、現在の処理が終了したのち、あらかじめ行っていたキー入力の内容により自動的に次の処理に進ませることができます。

BASIC内部では、他の処理中にキーが押された場合に入力された文字64文字まで記憶する領域(キーバッファ)をもっており、そこへ次々と入力された文字が蓄えられます。その後BASICで入力状態になったときに、記憶した順番にキーバッファから1文字ずつ取り出されます。なお、キーバッファがいっぱいになった場合は、先に入力された文字が取り出されるまで入力はできなくなります。

先行入力を利用する場合、次の点に注意してください。

- 処理中のエラー発生や[シフト]+[停止]キーの操作により命令待ちに戻るときは、キーバッファを消去します。
- 処理中の一時停止のために[停止]キーを押した場合は、キーバッファを消去します。また、一時停止の解除のために押したキーも入力されません。
- 先行入力中はキーリピート動作はしません。

キーバッファの内容を設定する命令として**key 0**が用意されていて、キー入力のシミュレートやキーバッファの消去に利用できます。また[コントロール]+[シフト]+[クリア/ホーム]キーの操作によってもキーバッファを消去することができます。

キーボードの配列

キーボード配列は標準的なJIS規格に準じた英数字やカナ配列になっていますが、**init "KB:1"**の実行により、カナ配列を50音順に変

更することができます。

ただし、キーラベルはJIS配列となっています。

クリック音

キーを押したときに発生するクリック音を制御することができます。クリック音は**click**文で設定します。

クリック音はコンピュータが有効な入力として受け取ったときに発生しますので、シフトキーやコントロールキーのみを押しただけでは発生しません。

キーリピート

キーを押し続けた場合に、リピート(繰り返し)させる機能があります。キーリピートは、**repeat**文で設定します。また、**repeat**文はリピート速度を4段階で指定する機能もあります。

ファンクションキー

ファンクションキーについてはオーナーズマニュアルを参照してください。

ファンクションキーは1つのキーに複数の文字を定義することができる機能を持つキーです。☜コラム

BASICでは**F1**～**F20**の20種のキーを定義することができるようになっています。

ファンクションキーのキーラベルは**F 1**～**F 10**までですがシフトキーを押しながら、**F1**～**F10**のキーを押すと**F11**～**F20**に対応します。

key	……	ファンクションキーの定義。
klist	……	定義内容の最下行への表示制御
key list	……	定義内容のリストアップ

注) **SF 1**～**SF 4**は定義することはできません。

キー割り込み機能

プログラムの実行中に、ファンクションキーやヘルプキー、シフト+停止キーなどを押したときに割り込み(プログラムの中断)をかけ、指定のサブルーチンを割り込み処理として実行させることができます。割り込み処理として実行するサブルーチンは、それぞれ別々に設定できます。

キー割り込み用の命令として次のものが用意されています。

on key gosub　　key on/off/stop

…… **key**文による指定キーによる割り込み処理の定義と制御。

on help gosub　　help on/off/stop

…… ヘルプキーによる割り込み処理の定義と制御。

on stop gosub　　stop on/off/stop

…… シフト＋停止キーによる割り込み処理の定義と制御。

12 時計機能

コンピュータ内には、電池によってバックアップされたカレンダ時計が内蔵されて、時分秒、年月日、曜日の管理をしています。

時分秒は、システム変数の**time$**で8桁の文字列で表され、代入を行うことにより現在時刻を設定することができます。

例） **time$＝"1 2:3 4:5 6"**

…12時34分56秒に設定します。

年月日は、システム変数の**date$**で8桁の文字列で表され、代入を行うことにより日付を設定することができます。

例） **date$＝"8 7／1 2／2 5"**

…87年12月25日に設定します。

曜日は、システム変数の**day$**で3桁の文字列で表され、代入を行うことにより曜日を設定することができます。

例） **day$＝"Wed"**

…水曜日に設定します。

0.1秒単位のタイマーがシステム変数の**time**として用意されています。値は0～65535（0～6553.5秒）で、代入を行うことにより、設定ができます。

時計による割り込み機能

time$があらかじめ設定している時刻になった場合や、一定間隔で割込みを発生させるインターバルタイマによる割り込みが発生した場合に、特定のサブルーチンを割り込み処理として実行させる機能があります。

割り込み処理用の命令としては、次のものが用意されています。

on time$ gosub	……	time$による割り込み処理の定義。
time$ on/off/stop	……	time$による割り込み処理の制御。
interval	……	インターバルタイマの周期設定。
on interval gosub	……	インターバルタイマによる割り込み処理の定義。
interval on/off/stop	……	インターバルタイマによる割り込み処理の制御。

13 サウンド機能

コンピュータ内には、音楽用のLSI(OPN)が内蔵されていて、いろいろな音を出して音楽演奏ができます。

LSI内にはFM音源部とSSG音源部があり、それぞれ3チャンネル(3音)出力し、合計6チャンネルでの音楽演奏が可能です。

特にFM音源は、複雑な処理が行え、様々な楽器音を出すことができます。

音楽演奏は**play**文で行われ、演奏データを設定するとすぐに次の処理に進めますので、音楽演奏とプログラム処理が同時に行われます。また、演奏の終了時に割り込みをかける機能がありますので、プログラム中のBGMとして常時、音楽演奏をさせておくこともできます。

サウンドに関する命令は次のものが用意されています。

play	……	音楽演奏をします。
play init	……	音楽演奏の初期化をします。
play wait	……	音楽演奏の終了を待ちます。
on play gosub	……	音楽演奏終了時の割り込み処理を定義します。
play on/off/stop	……	音楽演奏終了時の割り込み処理を制御します。
sound	……	OPNのレジスタをセットします。
beep	……	スピーカーを一瞬鳴らします。

FM音源

FM音源は、サイン波(へ)の組み合わせによって波形を合成しています。

LSI内には、このサイン波メモリを持つ音色演算器(オペレータ)を4つ内蔵しており、これらオペレータを8種類の組み合わせ方法(アルゴリズム)の中から選ぶことを基本に、各オペレータに様々な変調データを与えることによって複雑な波形を作り出します。

FM音源の音色の設定は大変複雑なので、BASICではあらかじめ30種類の音色を用意していますが、**tone**文により自由に音色の設定ができるようになっています。

また、BASICで用意された音色データを**tone copy**文で読み出すこ

とができるので、音色のデータ作りの参考とすることができます。

14 マウスの操作

CONFIG. SYSファイルについてはMS-DOSマニュアルを参照してください。

マウス(**MZ-1X10**)(別売)を使用することにより、グラフィック画面上のマウスカーソルを移動させ、座標入力を行うことができます。

なお、マウスを使用するためには、MS-DOSの**CONFIG. SYS**ファイルに**MOUSE. SYS**(デバイスドライバ)を登録しておく必要があります。

出荷時は登録されていますが、登録されていない場合はBASIC起動時に"マウス機能は利用できません"というメッセージが表示されます。

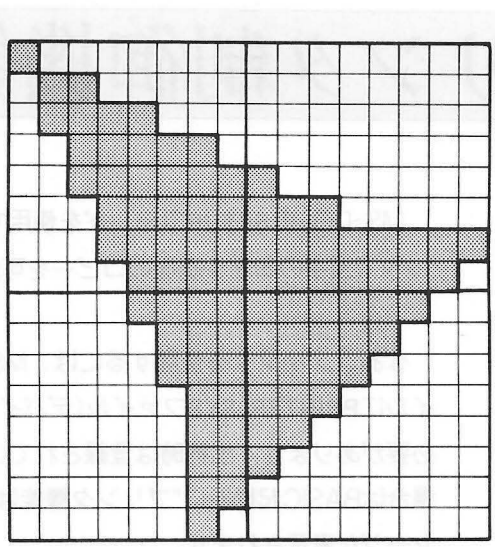
マウスの機能設定

mouse文により、次のようなマウスの機能設定を行います。

- マウスの使用開始の宣言と機能の初期化
- マウスカーソルの直接移動。
- マウスカーソルの形状設定。
- マウスカーソルの移動比率の設定。
- マウスカーソルの移動範囲の設定。
- マウスカーソルの表示色の設定。
- マウスの使用終了の宣言。

(例) マウスカーソルの形状を図のように設定するには、次のようにします。

```
10 A$ = hexchr$ ("8000600078003E003F801FE01FFF0FFE0FFC07
   F807F003E003C0038003000200")
20 mouse 2, 0, 0, A$
```



マウスの入力

mouse関数によりマウスからの入力を行います。

マウスから入力する情報は次のとおりです。

- 現在のマウスカーソル位置。
- 現在のボタン状態。
- ボタンが押されたときの座標。
- ボタンが離されたときの座標。
- 相対移動量。
- ボタンが押された回数と離された回数

マウスによる割り込み処理

マウスボタン操作や移動により割り込みをかけ、特定のサブルーチンを割り込み処理として実行させることができます。

割り込みの要因は次の5種で、それぞれに割り込み処理を定義できます。

1 …… マウスが移動したとき。
2 …… ボタン1が押されたとき。
3 …… ボタン2が押されたとき。
4 …… ボタン1が離されたとき。
5 …… ボタン2が離されたとき。

1～5は割り込みの要因の番号です。

割り込み処理に関する命令として次のものが用意されています。

on mouse gosub …… 各要因による割り込み処理行の定義。

mouse(n) on/off/stop …… 要因番号nによる割り込み処理の制御。

15 プリンタ制御機能

SETUPコマンド
CONFIG. SYSファイルについてはMS-DOSマニュアルを参照してください。

MS-DOSの**SETUP**コマンドを使用することにより、各種のプリンタで漢字の印字や画面のコピーを可能としています。

なお、プリンタを使用するには、MS-DOSの**CONFIG. SYS**ファイルに**PRINTER. SYS**ファイル(デバイスドライバ)を登録しておく必要があります。出荷時は登録されていますが、登録されていない場合はBASIC起動時に"プリンタ機能は利用できません"というメッセージが表示されます。

文字の印字

プリンタへ文字を印字させる命令として次のものが用意されています。

lprint　llist　llist＊　lsearch　lfiles　ltron

これらの命令は画面に出力されるのと同様の書式でプリンタに印字されます。

漢字などの全角文字の印字は、漢字印字の可能なプリンタに対しては、**kmode**文の設定により印字が可能です。

スプーラ機能

プリンタの動作は本体の動きに比べてたいへん遅く、連続して印字を行う場合はプリンタがそれまでのデータをすべて印字するまで待たなければなりません。

そこで本BASICでは、プリンタへの印字データがあると、印字データをいったんプリンタバッファと呼ばれる領域へ記憶しておき、次のプログラム処理に進みます。プリンタバッファ内のデータは、プリンタがデータ受け取り可能な状態になると、プリンタへ送られます。(スプーラ機能)。スプーラ機能によって、BASICの処理とスプーラの作業が並列に行われ、BASICはプリンタの動きを待たずに処理を行うことが可能になります。

画面のコピー

画面に表示されている文字や図形をプリンタにコピーする機能として**hcopy**文と[コピー]キーの操作があります。

コピー機能	hcopy文	キー操作
文字画面のみ	hcopy 1	コントロール＋コピー ※
グラフィック画面のみ	hcopy 2	
文字とグラフィック画面	hcopy 3	

※ コントロール ＋ コピー キー操作の場合、どの画面のコピーを行うかの選択はMS-DOSの**SETUP**コマンドで行います。

※グラフィック画面が65536色モード、文字画面が40桁モードで2画面重ね合わせたときはハードコピーが行えません。

ファイル装置としてのプリンタ

ファイル処理において、文字出力用の装置として**"LPT:"**の装置名で使用することができます。

プリンタの直接制御

プリンタ自身のコントロールのために、制御コードを直接送る場合は、**lpout**文を使用します。

lprint文の場合は漢字モードの切り替えや漢字コードの変換を行っていますので制御コードが送れない場合があります。

16 ディスプレイテレビ制御機能

ディスプレイテレビ(**MZ-1D24**)(別売)を使用することにより、テレビ映像/コンピュータ画面の切り替え、電源のON/OFF、音量、チャンネル切り替えの制御が可能になる他、テレビ映像とコンピュータ画面の重ね合わせ(スーパーインポーズ機能)が可能になります。

ディスプレイテレビ制御、スーパーインポーズ機能の設定は、**init"CRT4:"**文で設定します。

17 RS-232Cインターフェイスの制御

コンピュータはRS-232Cインターフェイスを1チャンネルもっており、BASIC上では**"COM1:"**の装置名で使用します。

機能設定

RS-232Cの機能設定は、**init "COMn:"**文により行います。
設定できる内容は次のとおりです。

- ボーレイト、パリティビット、データビット長、ストップビット長のデータフォーマット。
- X-ON, X-OFFまたはRR-RS信号による通信制御。
- カナ文字の伝送コード。
- DELコードの処理。
- 改行コードの設定。
- 全角文字の伝送コード。
- 通信の終了コード。

18 MS-DOSへの移行

本BASICはMS-DOS上で動作するため、BASICからMS-DOSへ、あるいはMS-DOSからBASICへ簡単に制御を移すことができるようになっています。

BASICからMS-DOSへ制御を移すときは、**system**コマンドを使用します。

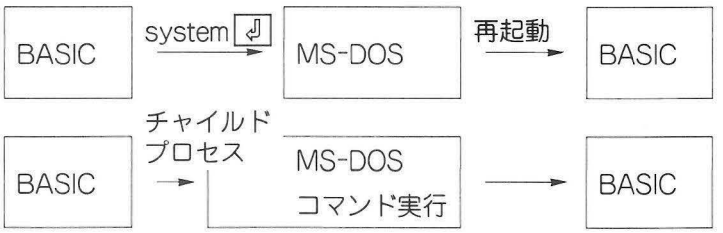
なお、MS-DOSからBASICへの制御を移すときは、BASICの再起動となるので、MS-DOSのプロンプト(**>**)が表示されているときに、カレントドライブにMS-DOS/BASICシステムディスクがあることを確かめ、BASICシステムのファイル名(BASIC)を入力することになります。

A>BASIC↵

チャイルド・プロセス

BASICからMS-DOSのコマンドを一時的に使用したい場合があります。

systemコマンドの場合は、MS-DOSに制御が移った後、BASICに制御を戻すには、BASICを再起動しなければなりませんが、チャイルド・プロセスを使用すると、BASIC上から一時的にMS-DOS上に制御を移し、MS-DOSのコマンドを実行した後、BASICを再起動することなく、BASIC上に制御を戻すことができます。



チャイルド・プロセスの実行は**child**文で行います。

なお、チャイルド・プロセス(MS-DOSのコマンド実行)はBASICの管理外の領域で行いますので、**clear**文によってBASICの管理外メモリ領域(チャイルド・プロセス領域)をあらかじめ確保しておく必要があります。

チャイルド・プロセスの環境設定には次の命令が用意されています。

environ …… チャイルド・プロセス環境の設定
environ$ …… チャイルド・プロセス環境の読み出し

19 ハードウェアの直接制御

本BASICではメモリの内容を直接読み書きしたり、I/Oポートより直接データの入出力、機械語で書かれたプログラムをサブルーチンとして実行するなどの、直接ハードウェアを制御する機能を持っています。

これらの制御は、80286CPUの制御信号やプログラミング、メモリ構成、I/Oポート入出力による内部回路の制御などを理解する必要がありますが、上手に利用するとBASICと合わせて効率的な処理が可能となります。

機械語エリア

BASIC上で機械語プログラムを利用する場合、あらかじめ機械語プログラムを作成するメモリを確保する必要があります。BASICの初期状態ではBASICがすべてのメモリを使用していますので、不用意にメモリを書き換えるとBASICを破壊するおそれがあります。

そこで、**clear**文でBASICで使用しない機械語エリアを指定します。

clear文では、機械語エリアの大きさを16バイト単位で指定し、**segptr**(2)で求められるセグメント以降から**segptr**(1)で求められるセグメントの直前までのセグメントまでのメモリが機械語エリアとなり、**segptr**(2)で求められるセグメントの直前までが、BASIC で使用するエリアとなります。したがって機械語エリアを確保するとその分のBASICのフリーエリアが少なくなります。

機械語エリアのアクセス

clear文で確保された機械語エリアは、BASIC内では使用されませんので自由に読み書きが行え、機械語プログラムを作成することもできます。

BASICでは機械語エリアのアクセスのために、次の命令が用意されています。

poke …… メモリにデータを書き込みます。
peek …… メモリよりデータを読み出します。
bsave …… 機械語エリア内の指定範囲をファイルします。
bload …… 機械語エリアへファイルを読み込みます。

以上の命令を使用して自由に機械語エリア内にプログラムを作成し、**BIN**形式のファイルを作成することもできます。

なお、これらの命令でアクセスできる範囲は、64Kバイト以内なので、これらの命令を使用する場合には、下限のアドレス(ベースアドレス)を設定する必要があります。

ベースアドレスの設定、読み出しのために、次の命令が用意されています。

def seg … ベースアドレスの設定。
segptr … プログラム実行時の各種領域(セグメント)のベースアドレス読み出し。
varptr … 変数や配列データのセグメントのベースアドレス読み出し。

I/Oポートのアクセス

80286CPUはメモリアドレスのほかにI/Oアドレスを持っていて、コンピュータ内部の制御や周辺機器の入出力に利用しています。

I/Oアドレスに対する入出力命令として次のものが用意されています。

inp …… 指定したI/Oアドレスよりデータを読み取る。
out …… 指定したI/Oアドレスにデータを出力する。

機械語プログラムの実行

機械語エリアに作成した機械語プログラムは、サブルーチンとしておくと、BASIC上から呼び出して実行することができます。

機械語サブルーチンの呼び出しは、**usr**関数または**call**文により行います。

usr関数は、引数としてデータを機械語サブルーチンに引き渡し、機械語サブルーチンよりデータを受け取りこの関数の値とします。

call文は1個または0個のデータを変数として引き渡し、機械語サブルーチンでデータの内容を書き換えることによりBASICに戻ったときに書き換えたデータが変数に代入されます。

機械語プログラムの実行に関する命令として次のものが用意されています。

def usr	……	**usr**関数のアドレスを定義する。
usr	……	機械語プログラムを関数として実行。
call	……	機械語プログラムの実行。

第2部

コマンド・ステートメントの解説

ここでは、各コマンド・ステートメント・関数・システム変数について説明します。

書　式　命令の記述方法を示します。ただし、以下の記号は実際に入力するものではなく、次のような意味があります。

〈 〉　：変数やデータのような、記述するものを表します。
ただし、一番広い意味を表すものです。

(注)

<変数>には、**<数値型変数>**と**<文字列型変数>**があります。
<変数>は、**<配列変数>**も含みます。
<データ>は、**<変数>**と**<定数>**を含みます。

〔 〕　：〔 〕で囲まれた部分は省略可能であることを表します。ただしこの後に他の指定が行われる場合はカンマやセミコロンの句切記号は必要です。

〔 〕…　：〔 〕…で囲まれた部分は、省略またはくり返しが可能であることを表します。

$\left\{\begin{matrix} A \\ B \end{matrix}\right\}$　：AまたはBのどちらかを選択します。

省略形　命令内で使用する予約語の最短の省略形を示します。ただし、省略形のないものまたは省略の意味がないものは省いてあります。
（例　init→ini.）

説　明　命令の働き、各指定の方法などを解説しています。

例　命令の使用例を示します。場合により解説を付けています。

参　考　関連する命令や解説部を示します。

サンプル　命令の使用例を示します。場合により解説を付けています。

RUN（ラン）

プログラムの実行を開始します。

書　式　run〔〈スタート行〉〕

省略形　r.

説　明

- ＜スタート行＞は、行番号またはラベルで指定できます。
- ＜スタート行＞を指定しない場合はプログラムの最初の行から実行され、＜スタート行＞を指定した場合は指定行から実行します。
- ＜スタート行＞を指定しない場合clear文の機能も実行され、変数や配列および定義関数の消去、ファイルのクローズ、角度単位やデータの読み出し行の初期化などが行われます。(clear文参照)
- プログラムの実行が開始されたのちは、プログラム中でのend文、stop文の実行またはプログラムの終了により命令待ちに戻ります。
- プログラム実行中にエラーが発生した場合、エラー内容とエラーが発生した行番号を表示して命令待ちに戻ります。
- プログラム実行中に[シフト]+[停止]キーが押されると、実行を中断し、中断した行番号を表示して命令待ちに戻ります。
- プログラム実行中に[停止]キーを押すと実行を一時停止します。この状態で任意の文字キーを押すと、実行を再開します。
- runのあとにファイル名を指定するとフロッピーディスクなどからプログラムを読み込み、実行します。(run ＜ファイル＞参照)

例

```
run          …… プログラムの先頭から実行します。
run 100      …… 行番号100から実行します。
run *開始    …… *開始というラベル名の付いた行から実行します。
```

参　考☞　run ＜ファイル＞ …… ファイルを呼び出しての実行。(294ページ)

CONT （コンティニュー・continue）

中断したプログラムの実行を再開します。

書　式　cont

省略形　c.

説　明

- プログラムの実行中に［シフト］＋［停止］キーを押すか、プログラムの中のstop文またはstopコマンド、stepコマンドにより実行が中断された状態のとき、中断された所から実行を再開します。
- end文による終了やエラーにより中断したとき、または中断後にプログラムの変更や消去を行った場合には、プログラムを再開することはできません。
- この命令の実行が可能なときは、Okのあとにピリオド(.)を表示します。

参　考☞

step （コマンド）　…… プログラムのステップ実行。(137ページ)
stop （コマンド）　…… ブレークポイントの設定。(138ページ)
stop （ステートメント）　…… 実行の中断。(201ページ)

サンプル

```
10 A=5
20 B=7
30 stop
40 C=A+B
50 print A;"+";B;"=";C
run
*Stop in 30
Ok.
print a,b,c
 5          7          0
Ok.
cont
 5 + 7 = 12
Ok
```

STEP （ステップ）

プログラムのステップ実行を行います。

書 式　step〔〈行数〉〕〔：{run〔〈スタート行〉〕 / goto〈スタート行〉}〕

説 明

- contコマンドのように中断されたプログラムを再開しますが、＜行数＞で指定された行数だけ進むと"**step in** nn"および、次に実行しようとする行の内容を表示して実行を中断します。nnの部分は中断時の行番号を示しますが、その行自体は実行していません。再度、**step**コマンドの実行または、**cont**コマンドの実行で中断した行から実行します。
- ＜行数＞は**1～65535**の範囲が指定でき、省略した場合は**1**になります。
- ＜スタート行＞は、行番号またはラベルで指定できます。
- 特別な処理としてあとに**run**コマンドまたは**goto**文を続けることにより、プログラムの先頭や指定行から実行を開始させることができます。この場合は入力行も1行として扱います。
- あとに**run**コマンドまたは**goto**文を続けない場合は、実行の再開ができる状態（**Ok**のあとにピリオド(.)が表示される）でないと**cont**コマンドと同様にエラーになります。
- この命令は直接実行モード以外では使用できません。

例

```
step        …… 現在位置より1行だけ実行します。
step 5      …… 現在位置より5行進みます。
step:goto 100
            …… 行番号100からの指定ですがコマンド行を1行と数えますので行番号
               100の先頭で中断されます。
```

STOP （ストップ）

プログラムのブレークポイントを設定します。

書　式　stop〈ストップ行〉

省略形　s.

説　明

- ＜ストップ行＞は、行番号またはラベルで指定できます。
- この命令で＜ストップ行＞を指定してプログラムをスタートさせると、＜ストップ行＞で指定された行まで進むと実行を中断し、"**stop in** nn" とnnの所に行番号を表示して命令待ちに戻ります。このように、プログラムの中断が行われる特定の位置をブレークポイントと呼びます。
  なお、ブレークポイントは１ヶ所のみ設定することができます。
- 実行を中断した位置は、＜ストップ行＞の先頭になりますので、＜ストップ行＞自体はこの時点ではまだ実行していません。
- 一度ブレークポイントを設定すると、ブレークポイントへ進むごとに実行を中断します。設定されたブレークポイントの解除は、ほかの位置にブレークポイントの設定を移すか、＜ストップ行＞の指定を**0**にします。
- プログラム中の**stop**文の実行によってもブレークポイントとすることができますが、解除を行う場合は、プログラムの変更になりますので、**cont**コマンドによるプログラムの再開ができなくなります。この命令でのブレークポイントは、プログラムの変更を行わずに設定や解除ができます。
- 実行を中断した状態から続けて処理を行う場合は、**cont**コマンドまたは**step**コマンドを使用します。
- この命令は直接実行モード以外では使用できません。

例　`stop 100` …… ブレークポイントを行番号100に設定します。

参　考☞　**cont** …… 中断されたプログラムの再開。(136ページ)
step …… 中断されたプログラムのステップ実行。(137ページ)

NEW （ニュー）

プログラムを消去します。

書 式　new

説 明

- BASIC上でのプログラムをすべて消去します。同時に**clear**文の機能が実行され、変数や配列および定義関数の消去、操作途中のファイルのクロ-ズ、角度単位やデ-タの読み出し行の初期化などが行われます。（**clear**文参照）
- **clear**＜サイズ＞文によってBASICエリアが制限されている場合は、BASICエリアのプログラムのみを消去し、機械語プログラムやチャイルドプロセス領域は消去しません。
- **load**コマンドによるBASICプログラムの読み込み実行時にも、プログラムを読み込む前にこの命令と同じ動作が行われます。

例

new …… プログラムを消去します。

9999 new

…… プログラム中にも書くことができますが、実行するとその時点でプログラムが消去されますので注意してください。

参 考　**delete** …… プログラムの部分的な消去。(153ページ)

サンプル

```
10 print "ｻﾝﾌﾟﾙ ﾌﾟﾛｸﾞﾗﾑ"
20 end

run
ｻﾝﾌﾟﾙ ﾌﾟﾛｸﾞﾗﾑ
Ok
new
Ok
list
Ok
```

TRON　（トレース オン・trace on）
LTRON　（ライン トレース オン・line trace on）

プログラムの実行を追跡します。

書　式　{tron / ltron}　〔〔〈表示開始行〉〕－〔〈表示終了行〉〕〕

省略形　t.　lt.

説　明

- tronとしてトレース(追跡)モードにしてプログラムを実行させると、実行している行番号を[ ]で囲んで表示します。
- <表示開始行>と<表示終了行>でトレース範囲を指定した場合、指定範囲内を実行したときにトレース状態を表示します。
- ltronとすると画面のかわりにプリンタにトレース状態を印字します。
- 解除するときは、troffコマンドを実行します。

例

```
tron
```
…… トレースモードにします。それ以後、プログラムを実行するとトレース状態を画面に表示します。

```
tron 200-300
```
…… 行番号200から300までの範囲を実行したときにトレース状態を表示します。

参　考　troff …… トレースモードの解除。(141ページ)

TROFF （トレース オフ・trace off）

TRONの機能を解除します。

書　式　troff

説　明
- tronが実行されトレース状態になっている場合、通常の状態に戻します。

参　考☞　tron …… トレースモードの実行。(140ページ)

サンプル

```
10 T=1
20 for A=2 to 5
30   T=T*A
40 next
50 print T
Ok
tron
Ok
run
[10][20][30][40][30][40][30][40][30][40][50] 120
Ok
troff
Ok
run
 120
Ok
```

BOOT（ブート）

BASICの処理を中止し、システムを再起動します。

書　式　boot

省略形　bo.

説　明

- この命令を実行するとシステムを再起動します。電源をonしたときと同じ状態になります。したがって、OS（MS-DOS）やBASIC言語およびBASICプログラムは消去されます。

参　考☞　**IPL**(Initial Program Loader)

…… 起動用補助プログラム。（オーナーズマニュアル参照）

LIST　（リスト）
LLIST　（ライン リスト・line list）

プログラムリストを出力します。

書　式

{ list / llist } 〔〔〈開始行〉〕ー〔〈終了行〉〕〕

省略形

l.　ll.

説　明

- listは画面にプログラムのリストを表示します。
- llistはプリンタにプログラムリストを印字します。
- ＜開始行＞と＜終了行＞を指定してプログラムリストを部分的に出力させることもできます。
- ＜開始行＞と＜終了行＞は、行番号またはラベルのほかにピリオド(.)での指定ができます。(**list.** とすると、エラー発生行やプログラム実行中断行が確認できます。)
- リストの表示中に[停止]キーを押すと一時停止します。その状態で任意の文字キーが押されると解除され表示を続けます。
- 表示中に[シフト]+[停止]キーを押すと、表示が中止され命令待ちに戻ります。

例

```
list　……　すべてのリストを画面に表示します。
list 50　……　行番号50のみを表示します。
llist -100
　……　プログラムの先頭から行番号100までのリストをプリンタへ印字します。
list *ABC -
　……　＊ABCというラベル名の付いた行からプログラムの最後まで表示します。
list 70-220
　……　行番号70から行番号220までのプログラムを表示します。
```

LIST〈ファイル〉（リスト〈ファイル〉）

ファイルに対してリスト出力を行います。

書　式　list〈ファイル指定〉[,[〈開始行〉]－[〈終了行〉]]

省略形　l.

説　明

- ＜ファイル指定＞で指定された装置にプログラムリストを出力します。
- ＜ファイル指定＞は次の書式で指定します。
  “〔＜装置名＞〕〔＜ディレクトリパス＞〕＜ファイル名＞＜拡張子＞”
  ＜装置名＞ ……**A:, B:, C:～J:, K:, L:, M:**
  または**FDn:, HDm:, MEM:, COM 1 :, CRT 1 :, LPT:**
  フロッピーディスク，ハードディスク，メモリディスクは＜ファイル名＞の指定が必要でアスキー形式のファイルとして記録され、ＲＳ-232Ｃインターフェイスへは、文字コードで出力されます。ディスプレイ画面は通常の**list**コマンド、プリンタは、**llist**コマンドと同じ働きをします。
- ＜ディレクトリパス＞は階層ディレクトリのパス（道すじ）を指定します。（chdirコマンド参照）
- ＜装置名＞と＜ディレクトリパス＞の指定を省略すると、chdirコマンドで設定されているカレントディレクトリ内でのファイル指定になります。
- ＜ファイル名＞は、＜拡張子＞を付けて指定します。＜拡張子＞を省略すると**.DAT**が設定されます。
- ＜開始行＞、＜終了行＞の指定方法は**list**コマンドと同じです。

例

```
list "A:ABC"
```
…… ドライブAへアスキー形式のプログラムファイルとして記録します。

```
list "Neko",70-220
```
…… 行番号70から行番号220までのプログラムをアスキー形式のプログラムファイルとして記録します。

```
list "COM1:"
```
…… RS-232Cインターフェイスへリストを出力します。

LIST＊　（リスト アスタリスク）
LLIST＊　（ライン リスト アスタリスク・line list＊）

コメント行のみをリストします。

書　式

list＊〔〈ファイル指定〉〕〔〔〈開始行〉〕ー〔〈終了行〉〕〕
llist＊〔〔〈開始行〉〕ー〔〈終了行〉〕〕

省略形

l.＊　　ll.＊

説　明

- **list＊**はプログラム中でアポストロフィ(')で始まる行(コメント行)のみの内容を行番号とアポストロフィを除いて表示します。行の途中から始まるアポストロフィまたは**rem**文によるコメントは対象となりません。
- **list＊**のあとに＜ファイル指定＞を行うと**list ＜ファイル＞**コマンドと同様に指定装置にアスキー形式のファイルを作成します。ただし、行番号は除かれていますので**load**コマンドでプログラムとして読み込むことはできません。**load＊**コマンドでコメント行として読み込むことができます。
- ＜開始行＞、＜終了行＞の指定は**list**コマンドと同じです。
- **llist＊**は、プリンタにコメント行のリストを印字します。

参　考☞

auto＊ …… コメント行の連続入力。(147ページ)
load＊ …… コメント行として読み込む。(298ページ)

サンプル

```
10 rem --- LIST* ---
20 '-----------------------------
30 '  A出版社　来月の新刊のご案内
40 '
50 '    ○○・ストーリー    文庫     ￥   420
60 '    実用●■▲大百科     新書     ￥1，980
70 '    ◎◎の旅１００％    Ｂ５版   ￥1，260
80 '-----------------------------
90 end

list*
-----------------------------
  A出版社　来月の新刊のご案内

    ○○・ストーリー    文庫     ￥   420
    実用●■▲大百科     新書     ￥1，980
    ◎◎の旅１００％    Ｂ５版   ￥1，260
-----------------------------
Ok
```

AUTO （オート）

行番号の自動発生を行います。

書　式　auto〔〈開始行〉〕〔,〈増分〉〕

省略形　a.

説　明

- この命令を実行すると、自動的に行番号を発生させ、行番号のあとにカーソルを出して入力状態とします。1行分の入力が終わり、[↵] キーが押されると、次の行番号を表示して入力状態になります。
- 発生させた行がすでに使用されていればその内容を表示し、文の先頭にカーソルを表示します。
- ＜開始行＞は、行番号またはラベルで指定できます。
- ＜開始行＞と＜増分＞は、それぞれ省略することができます。省略した場合は、それぞれに10が指定されたものとします。
- この状態は[シフト]+[停止]キーを押すことにより解除されます。

例

```
auto
```
…… 10, 20, 30, 40‥と、10から、10ステップで行番号が発生します。

```
auto 100
```
…… 100, 110, 120, 130‥と100から、10ステップで行番号が発生します。

```
auto 400,20
```
…… 400, 420, 440, 460‥と400から、20ステップで行番号が発生します。

AUTO＊　（オート アスタリスク）

アポストロフィ(’)付きの行番号の自動発生を行います。

書　式　auto＊〔〈開始行〉〕〔,〈増分〉〕

省略形　a.＊

説　明

- autoコマンドと同様に行番号を自動的に発生させますが、行番号のあとに、アポストロフィ(’)を表示して入力状態になります。アポストロフィは注釈文の始まりを意味します。(rem文参照)
- この機能は注釈文を続けて入力する場合に便利で、簡単なワードプロセッサー入力として活用することもできます。また、アポストロフィで始まる行のみを表示する命令としてlist＊コマンド、ファイルからコメント行として読み込む命令としてload＊があります。
- ＜開始行＞と＜増分＞は、それぞれ省略することができます。省略した場合は、それぞれに10が指定されたものとします。
- この状態は[シフト]＋[停止]キーを押すことにより解除されます。

参　考☞

auto　……　行番号の自動発生。(146ページ)
rem　……　注釈文。(167ページ)
list＊　……　コメント行の出力。(145ページ)
load＊　……　コメント行の読み込み。(298ページ)

EDIT （エディット）

エディットモードにして編集を行います。

書　式　edit〔〈編集行〉〕〔,〈増分〉〕

省略形　e.

説　明

- ＜編集行＞は、行番号またはラベルのほかにピリオド(.)での指定ができます。省略した場合は、プログラムの先頭から行います。
- 画面を消去して、＜編集行＞を中心とするリストを画面全体に表示し、スクリーンエディットが行える状態にします。編集キーとして次のキーが特別な働きをし、他のキーはいままでと同様の動きになります。
  ただし［コントロール］＋［D］キーは何も行いません。

  ［↑］　カーソルを上方向に移動します。カーソルが最上行にある場合は画面をスクロールダウン(下方向への移動)させ行番号の小さい方を表示します。

  ［↓］　カーソルを下方向に移動します。カーソルが最下行にある場合は画面を上に1行分スクロールアップさせ、行番号の大きい方を表示させます。

  ［シフト］＋［↑］　画面を半ページ分(画面に表示されている約半分)スクロールダウンします。

  ［シフト］＋［↓］　画面を半ページ分スクロールアップします。
- エディットモード時は入力した文字の色が変わり、［↵］キーが押されて入力が完了すると通常の表示色に戻して再表示を行います。
- ［↵］キーが入力されて再表示が行われると、次の行番号が自動的に表示されます。ただし、空行を入力した場合には行番号の表示は行われません。また、表示する行番号が65535をこえる場合には表示されません。
- ＜増分＞を省略した場合、10が指定されたものとします。
- エディットモードは、［シフト］＋［停止］キーの操作または、命令の直接実行により解除されます。またこのときに［コントロール］＋［D］キーの操作と同じ内容が実行され、表示の初期化などが行われます。

例

```
edit 1000    ……  行番号1000を中心にして、エディットモードにします。
edit ,20     ……  20ステップで行番号が発生します。
```

EDIT＊（エディット　アスタリスク）

エディットモードでコメント行を発生します。

書　式　edit＊〔〈編集行〉〕〔,〈増分〉〕

省略形　e.＊

説　明

- editコマンドと同様にスクリーンエディットが行える状態にします。editコマンドと異なる点は、行番号の自動発生が行われるときに、アポストロフィ(')を行番号の後に表示する点です。
- 〈編集行〉は、行番号またはラベルのほかにピリオド(.)での指定ができます。省略した場合は、プログラムの先頭から表示します。
- 〈増分〉を省略した場合、10が指定されたものとします。
- エディットモードは、シフト＋停止キーの操作または、命令の直接実行により解除されます。またこのときにコントロール＋Dキーの操作と同じ内容が実行され、表示の初期化などが行われます。

SEARCH （サーチ）
LSEARCH（ライン サーチ・line search）
プログラム中より指定文字列をさがし出して表示します。

書 式

{search / lsearch} "〈文字データ〉"〔,〔〈開始行〉〕−〔〈終了行〉〕〕

省略形

sea.　　lsea.

説 明

- searchはBASICのプログラム中の指定範囲内で、＜文字データ＞と同じ文字を含む行をさがし出し表示します。
- lsearchはプリンタに印字します。
- 画面表示の場合、＜文字データ＞と同じ部分の色が変化します。
- ＜文字データ＞にダブルクォーテーション(")が含まれる場合、直接ダブルクォーテーションを書くことができません。この場合、chr$(34)を使って文字式を書くか、または文字変数に代入して、指定を行います。
- listコマンドで表示される形により＜文字データ＞と比較しますので、予約語や変数の大文字と小文字などの変化には注意してください。(アルファベットの大文字と小文字は違った文字として扱われます。)
- ＜開始行＞と＜終了行＞は行番号またはラベルのほかにピリオド(.)での指定ができます。
- ＜開始行＞,＜終了行＞による範囲指定を省略すると、プログラム全体からさがします。
- kmode 1で全角文字を表示する状態のときは、表示される状態に従ったサーチを行います。たとえば、"年"の文字はkmode 0で表示させると"￣N"と表示します。"年"が含まれるプログラムでsearch "N"を行うとkmode 1の状態では、"年"の文字は見つかりませんがkmode 0の状態で行うと"年"のある行を表示します。ただし、画面の表示は"￣N"となります。
- 表示中に[停止]キーを押すと一時停止することができ、任意のキーを押すと表示を続行します。
- 表示中に[シフト]+[停止]キーを押すと、表示を中止し、命令待ちの状態に戻ります。

例

```
search "goto"
```

…… プログラム全体の中からgotoの文字を含む行をさがし出し表示します。

```
search "print "+chr$(34)+"ｵﾜﾘ"+chr$(34)
```

…… プログラム全体の中からprint "オワリ"の文字を含む行をさがし出し表示します。

```
lsearch "AAA",100-200
```

…… 行番号100から200の間でＡＡＡの文字を含む文字をさがし出し、プリンタに印字します。

サンプル

```
 10 rem --- SEARCH ---
 20 input "検索したい文字列は ";A$
 30 search A$
 40 rem A B C D E F G H I
 50 rem J K L M N O P Q R
 60 rem S T U V W X Y Z
 70 rem ｱ ｲ ｳ ｴ ｵ ｶ ｷ ｸ ｹ ｺ
 80 rem ｻ ｼ ｽ ｾ ｿ ﾀ ﾁ ﾂ ﾃ ﾄ
 90 rem ﾅ ﾆ ﾇ ﾈ ﾉ ﾊ ﾋ ﾌ ﾍ ﾎ
100 end
run
検索したい文字列は ？F
 40 rem A B C D E F G H I
```

REPLACE （リプレース）

プログラム中より指定文字列をさがし出して置き換えます。

書　式

replace〔all〕"〈文字列1〉" as "〈文字列2〉"
〔,〔〈開始行〉〕-〔〈終了行〉〕〕

省略形

repl.

説　明

- BASICのプログラム中の指定範囲内で、＜文字列１＞で指定された文字列をさがし出し、＜文字列２＞で指定された文字列に置き換えます。
- ＜開始行＞と＜終了行＞は行番号またはラベル指定できます。
- ＜開始行＞、＜終了行＞による範囲指定を省略すると、プログラム全体からさがします。
- **all**指定が省略されると、プログラム中に＜文字列１＞が現れるごとに、＜文字列１＞を含む行を表示し、カーソルを＜文字列１＞の先頭に移動して、"Y"または"N"のキー入力待ちとなります。
  このとき"Y"または↵キーが入力されると、＜文字列１＞を＜文字列２＞に置き換え、その他のキーが入力されると、置き換えは行われません。
- **all**が指定されると、確認を行わずに現れた＜文字列１＞を＜文字列２＞に置き換えます。

例

```
replace all "goto" as "gosub",100-1000
```

…行番号100から1000の間で**goto**の文字をさがし出し、**gosub**と置き換えます。

```
replace "disp" as "print"
```

…プログラム全体の中から**disp**の文字をさがし出し、**print**と置き換えます。（置き換えの確認があります。）

DELETE（デリート）

プログラムの一部を消去します。

書　式　delete〔〈開始行〉〕〔−〕〔〈終了行〉〕

省略形　d.

説　明

- ＜開始行＞から＜終了行＞までの指定範囲のプログラムを消去します。
- ＜開始行＞と＜終了行＞は、行番号またはラベルのほかにピリオド(.)での指定ができます。(ピリオドは現在行を表します。)
- ＜開始行＞のみの指定の場合は、指定された行のみを消去します。
- この命令のあとに指定がない場合はエラーになります。
- 行の指定を省略して−のみの指定の場合はすべてのプログラムを消去します。ただし変数や機械語プログラムの内容は変化しません。

例

```
delete 150-350
```
…… 行番号150から行番号350までのプログラムを消去します。

```
delete -150
```
…… プログラムの最初から行番号150までの部分を消去します。

```
delete 400-
```
…… 行番号400からプログラムの最後までを消去します。

```
delete 150
```
…… 行番号150のみを消去します。

```
delete -
```
…… すべてのプログラムを消去します。

参　考☞　new …… すべてのプログラムの消去。(139ページ)

RENUM （リナンバー・renumber）

プログラムの行番号を付け直します。

書　式　renum〔〈新行番号〉〕〔,〈増分〉〕〔,〔〈開始行〉〕ー〔〈終了行〉〕〕

省略形　ren.

説　明

- プログラムの指定された範囲の行番号を＜新行番号＞から＜増分＞のステップで付け直します。goto文, gosub文, if～goto文, およびon～goto文などの行番号の参照部分も自動的に付け直します。
  ただし、chain merge文の実行開始行は変更されません。
- 新たに付け直す行番号が65535をこえる場合や実行の順序が入れ替わるようなときは、実行できません。
- 行番号を付け直すことにより、1行の長さが255文字をこえる場合、listコマンドによる表示の際、line length error となり表示できなくなりますので注意してください。行を分割するか、小さな行番号に付け直してください。
- ＜新行番号＞と＜増分＞は、それぞれ省略することができます。省略した場合は、それぞれに10が指定されたものとします。
- ＜開始行＞と＜終了行＞は、行番号またはラベルのほかにピリオド(.)での指定ができます。

例

```
renum
```
…… プログラム全体の行番号を10から10ステップで付け直します。

```
renum 1000,10,100-
```
…… 行番号100からの行番号を行番号1000から10ステップで付け直します。

OPTION LIST （オプション リスト）

予約語表示の大文字、小文字を切り替えます。

書 式

option list {ucase / lcase}

省略形

opt. l. uc. lc.

説 明

- listコマンドなどで表示されるプログラム中の予約語部分を大文字で表示するか、小文字で表示するかを指定します。ucaseを指定すると大文字に、lcaseを指定すると小文字になります。
- BASIC起動時はlcaseの小文字での表示になっています。
- 注釈文やダブルクォーテーション(")で囲まれた部分、data文中のデータなどは予約語ではありませんので入力したとおりに出力されます。
- この命令は表示の指定になりますので、入力時は大文字、小文字のいずれでも予約語を書くことができます。
- 変数名や配列名、定義関数名、ラベル名は、常に大文字で表示されます。

例

```
option list ucase
```

…… 以後、プログラム中の予約語を大文字で表示します。

サンプル

```
list
   10 rem --- OPTION LIST ---
   20 A=int(rnd(1)*10)
   30 B=int(rnd(1)*10)
   35 C=A+B
   40 print "A =";A
   50 print "B =";B
   60 input "A + B =";D
   70 if D=C then color 5:print "OK !!" else color 2:print "Oh ! Wrong !!"
   80 color 7:goto 20
   90 end
Ok


option list ucase
Ok
list
   10 REM --- OPTION LIST ---
   20 A=INT(RND(1)*10)
   30 B=INT(RND(1)*10)
   35 C=A+B
   40 PRINT "A =";A
   50 PRINT "B =";B
   60 INPUT "A + B =";D
   70 IF D=C THEN COLOR 5:PRINT "OK !!" ELSE COLOR 2:PRINT "Oh ! Wrong !!"
   80 COLOR 7:GOTO 20
   90 END
Ok
```

SYSTEM （システム）

MS-DOSへ制御を移します。

書　式　　**system**

省略形　　**sy.**

説　明

- BASICからMS-DOSに制御を移します。
- MS-DOSに制御を移すと、次の操作が行われます。
  - ・すでに**open**されているファイルはすべて**close**されます。
  - ・**environ**文で指定された環境文字列テーブルは、MS-DOS上には引き継がれず、BASICが起動されたときの環境文字列テーブルとなります。
  - ・**chdir**文などの、ディレクトリの変更、削除、作成はそのままMS-DOSに引き継がれます。

LET (レット)

変数にデータを代入します。

書 式 〔let〕〈変数〉＝〈式〉

説 明

- イコール右辺の＜式＞の値を左辺の＜変数＞に代入します。
- ＜変数＞と＜式＞のデータ型は文字型どうし、または数値型どうしでなければいけません。数値型(整数型,単精度型,倍精度型)相互の代入は、自動的に型が変換されます。ただし、変換時にオーバーフローが起こったり、代入される変数に対しての誤差が大きくなる場合があります。
- 半角のアルファベットで始まる変数名で型指定記号(**%,！,＃,$**)によりデータ型を指定しないときは単精度型として扱われますが、**def int/def sng/def dbl/def str**の命令により省略時の型を変更できます。
- 漢字または半角文字のカタカナで始まる変数名で型指定記号が省略されたときは、倍精度型変数として扱われます。
- letは、完全省略できますので、＜変数＞＝＜式＞とすることができます。

例

A=3+4 …… 3＋4の結果の７を左辺の変数Ａに代入します。

D#=2.1

…… 単精度の値2.1を倍精度に変換して倍精度変数D＃に代入します。この場合、単精度から倍精度に変換されるため、代入値の誤差は単精度の誤差となり、倍精度に対しては大きな誤差になります。倍精度の値として代入させる場合は**D＃＝2.1＃**として値のあとに倍精度を示す記号＃を付けます。

参 考 **def int/sng/dbl/str** …… 型指定の省略時の型宣言。(158ページ)

サンプル 次のプログラムはどちらも同じです。

```
10 let A=10
20 print A
30 end

run
 10
Ok
```

```
10 A=10
20 print A
30 end

run
 10
Ok
```

DEF INT （ディファイン インテジャー・define integer）
DEF SNG （ディファイン シングル・define single）
DEF DBL （ディファイン ダブル・define double）
DEF STR （ディファイン ストリング・define string）

変数の型宣言を行います。

書 式

def int 〈変数名の範囲〉[,〈変数名の範囲〉]…　（整数型）
def sng 〈変数名の範囲〉[,〈変数名の範囲〉]…　（単精度型）
def dbl 〈変数名の範囲〉[,〈変数名の範囲〉]…　（倍精度型）
def str 〈変数名の範囲〉[,〈変数名の範囲〉]…　（文字列型）

説 明

- 変数の型は変数名の最後の型指定記号（%, !, #, $）によって指定されますが、半角文字のアルファベットで始まる変数の型指定を省略すると通常は単精度型として扱われます。この命令により省略時の変数型を別の型にすることができます。
- ＜変数名の範囲＞は、変数名の１文字目のアルファベットで指定します。A-Dとすると、変数名がA, B, C, Dで始まる変数の指定になります。この範囲の指定は半角文字のアルファベットで行い、漢字や半角文字のカタカナによる指定はできません。
- 指定された範囲内のアルファベットで始まる変数を別の型で使う場合は、型指定記号を付ければ、型指定の方が優先します。
- この宣言は、配列宣言における配列名や使用時の個々の要素の参照、定義関数の関数名にも有効となります。
- この宣言は、繰り返して使用することができ、変数名の範囲が重なった場合あとから宣言された方が有効となります。
- この宣言は、newコマンドやclear文またはrunコマンドにより解除され、初期状態に戻ります。（初期状態はdef sng A-Zが宣言されているとみなすことができます。）
- 漢字や半角文字のカタカナで始まる変数名や配列、関数名の型指定を省略した場合は、この命令での宣言には関係なく、倍精度型になります。

例

```
def dbl A-D
```
…… 変数名がA, B, C, Dで始まる変数に型指定がなければ倍精度型とします。

```
def str N,M,X-Z
```
…… 変数名がN, M, X, Y, Zで始まる変数に型指定がなければ文字列型とします。

参 考☞ 変数（第１部 BASICの概要参照）

サンプル

```
 10 rem --- DEF INT/SNG/DBL/STR ---
 20 'INT
 30 def int A-C : A=10 : B=3 : C=A/B
 40 print "整数型  A=10,B=3 ********** A/B =";C
 50 'SNG
 60 def sng A-C : A=10 : B=3 : C=A/B : print
 70 print "単精度型  A=10,B=3 ******** A/B =";C
 80 'DBL
 90 def dbl A-C : A=10# : B=3# : C=A/B : print
100 print "倍精度型  A=10,B=3 ******** A/B =";C
110 'STR
120 def str A-C : A="ABC" : B="DEF" : C=A+B : print
130 print "文字型  A='ABC',B='DEF' ... A+B =";C
140 def sng A-Z
150 end
```

```
run
整数型  A=10,B=3 ********** A/B = 3
単精度型  A=10,B=3 ******** A/B = 3.3333333
倍精度型  A=10,B=3 ******** A/B = 3.333333333333333
文字型  A='ABC',B='DEF' ... A+B =ABCDEF
Ok
```

- 変数**A, B, C**を整数型**int,** 単精度型**sng,** 倍精度型**dbl**に宣言し、それぞれについて計算結果を表示します。
- また、文字型**str**に宣言し、計算結果を表示します。

DEF FN (ディファイン ファンクション・define function)

関数を定義します。

書 式

def FN〈関数名〉(〈引数〉〔,〈引数〉〕…)＝〈演算式〉

説 明

- 演算内容を定義して新しく関数を作ることができます。
- ＜関数名＞の付け方は変数名と同じ方法で、型指定もできます。(使用時にはFNも合わせてひとつの関数名として扱われます。)
- ＜引数＞は変数名を書きますが、これは＜演算式＞の評価をするためのもので、実際の変数には影響しません。関数名の最後に型指定をすることにより、結果の型が指定できます。
- ＜演算式＞は＜引数＞を使った式を書きますが、引数以外の変数名を使った場合は、定義関数を使用したときの変数の値によって演算が行なわれます。
- ＜演算式＞の中にほかの定義関数を使うこともできます。(定義関数の入れ子型)
- 定義後はFNも含めた関数名で使い、引数として実際のデータを指定すれば、引数に対して演算処理をして、値を返します。
- 定義関数を再定義することはできません。同じ関数名で定義しようとするとエラーになります。
- 半角文字のアルファベットで始まる＜関数名＞や＜引数＞、＜演算式＞中の引数や変数名の型指定を省略すると、def int文などで宣言されている型として扱われます。したがって、定義時(この命令の実行時)と使用時の省略時の型が異なっている場合、異なる関数名として扱われたり、正しい結果が求められない場合がありますので注意してください。このような場合、省略時の型指定を途中で変更しないようにするか、＜関数名＞などに型指定を付けるようにしてください。
- clear文やnewコマンドまたはrunコマンドの実行によりすべての定義を消去します。

例

```
def FNROOT3(X)=X^(1/3)
```

…… 関数名FNROOT3(X)の関数を定義します。(立方根の演算をします。)

```
print FNROOT3(27)
```

…… 定義した関数をprint文で使用します。(結果は3と表示されます。)

```
def FNASCE(X$)=asc(right$(X$,1))
```

…… 関数名FNASCE(X$)で、文字列の最後の文字コードを求める関数を定義します。

サンプル

```
10 rem --- DEF FN ---
20 def FNC=A*B/2
30 input "底辺=",A
40 input "高さ=",B
50 print : print "面積は";FNC;"です。"
60 end
```

```
run
底辺=20
高さ=5

面積は 50 です。
Ok
```

- 三角形の面積を計算する方程式をあらかじめ**def FN**で定義します。

DIM （ディメンジョン・dimension）

配列の宣言を行います。

書　式

dim 〈配列名〉(〈添字〉〔,〈添字〉〕…)〔,〈配列名〉(〈添字〉〔,〈添字〉〕…)〕…

説　明

- 配列を使用する場合、あらかじめ配列名と添字の最大を宣言する必要があります。添字の最大は32767まで、次元の最大は255次元までですが、実際にはメモリ容量（フリーエリアの制限）や配列全体の大きさ、あるいは1行に書ける文字数などにより制約されます。
- 宣言時の各要素の内容は、数値型は0、文字型は文字長が0のNull（空文字）になります。
- 宣言したときのメモリ使用量は次のとおりです。数値型はデータが代入されても、使用量は変化しませんが、文字型は代入された文字数に応じて使用量が増えます。

整数型	個数＊2＋α バイト	αは、配列名や次元の管理に
単精度型	個数＊5＋α バイト	使用される分です。（10～数
倍精度型	個数＊8＋α バイト	百バイトまで、次元が多くな
文字型	個数＊4＋α バイト	るほど大きくなります。）

- 添字の下限は0ですが、option base文により、下限を1に変更できます。
- 宣言されている配列の再宣言や宣言されていない配列の参照、または添字の上限をこえた参照はエラーになります。ただし、宣言が行われずに添字が10までの配列を参照した場合は、自動的に10までの配列宣言が行われます。
- 変数名と同じ配列名を使うことができ、別の物として扱われますが、次元が異なっていても同じ配列名を使うことはできません。
- 宣言された配列は、newコマンドやclear文、またはrunコマンドが実行されたときに消去されます。また特定の配列を消去する命令として erase 文があります。

例

```
dim A(100),B$(100)
```
…… 数値配列Aと文字配列B$で100までの配列宣言をします。

```
dim XDATA(10,10)
```
…… XDATAの配列名で2次元配列を宣言します。

参　考

option base … 添字の下限を設定。(164ページ)

erase … 配列の消去。(165ページ)

サンプル

```
 10 rem --- DIM ---
 20 init "crt1:80,25":option base 1
 30 dim A$(5),B(5)
 40 locate 5,5 : print "+--------------+----------+"
 50 locate 5   : print "|   品    名   | 単  価   |"
 60 locate 5   : print "+--------------+----------+"
 70 locate 5   : print "|              |          |"
 80 locate 5   : print "|              |          |"
 90 locate 5   : print "|              |          |"
100 locate 5   : print "|              |          |"
110 locate 5   : print "|              |          |"
120 locate 5   : print "+--------------+----------+"
130 for I=1 to 5
140   read A$(I),B(I)
150   locate 6,7+I:print A$(I):locate 23,7+I:print B(I)
160 next
170 locate 0,15:end
180 data チョコレート,150
190 data ビスケット,250
200 data キャラメル,120
210 data ジュース,100
220 data コーヒー,100
```

```
+--------------+----------+
|   品    名   | 単  価   |
+--------------+----------+
|チョコレート  |  150     |
|ビスケット    |  250     |
|キャラメル    |  120     |
|ジュース      |  100     |
|コーヒー      |  100     |
+--------------+----------+
```

Ok

- 品名・単価は、添字が 5 の配列を使用してデータを読み込みます。

OPTION BASE （オプション ベース）

配列の添字の下限を設定します。

書 式

option base { 0 / 1 }

省略形

opt. ba.

説 明

- 配列の添字の下限を0または1に設定します。この命令はプログラムの実行開始後、配列の宣言を行うまでに実行してください。配列の宣言後に行うとエラーになります。
  また、この命令で２度以上設定を行うとエラーになります。
- この設定の解除は、newコマンドやclear文またはrunコマンドの実行により解除され、0が下限になります。

例

option base 1 …… 配列の下限を1に設定します。

参 考☞

dim …… 配列の宣言。(162ページ)

サンプル

```
10 rem --- OPTION BASE ---
20 option base 0
30 dim A$(1)
40 A$(0)="Apple":A$(1)="Orange"
50 print A$(0),A$(1)
60 end

RUN
Apple     Orange

10 rem --- OPTION BASE ---
20 option base 1
30 dim A$(1)
40 A$(0)="Apple":A$(1)="Orange"
50 print A$(0),A$(1)
60 end

run
subscript error in 40
```

- １番目のプログラムでは配列変数の下限を**0**としていますが、２番目のプログラムでは配列変数の下限を１としているので40行目でエラーが出ます。

ERASE（イレーズ）

配列を消去します。

書　式　erase 〈配列名〉[,〈配列名〉]…

省略形　er.

説　明

- 配列名を指定して配列を消去することができます。配列を消去すると消去した配列名で、再び宣言することができます。
- 消去した配列に使用していたメモリが解放されますので、フリーエリアが広くなり別の目的に使用することができます。

参　考　dim …… 配列の宣言。(162ページ)

サンプル

```
10 rem --- erase ---
20 dim X(10),Y$(10),Z(10)
30 erase X,Y$
40 dim X(100),Y$(100)
50 dim Z(100)            二重宣言

run
subscript error in 50
Ok
```

OPTION ANGLE （オプション アングル）

角度の単位を変更します。

書 式

option angle { radians / degrees }

省略形

opt. ang. rad. deg.

説 明

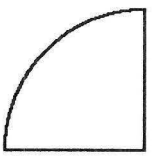
- 三角関数やグラフィック命令のcircle文やpoly文においての角度単位として、ラジアンと度のいずれを使うかを設定できます。

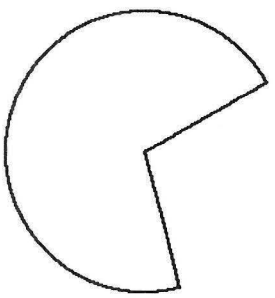
  radians ……ラジアン単位

  degrees ……度単位
- BASICの初期状態はラジアンですが、この命令でdegreesを指定することにより以後は度で扱われます。
- option angleはプログラム中で何回でも指定できます。
- この設定は、newコマンドやclear文またはrunコマンドにより解除され、ラジアン単位の状態になります。

サンプル

```
10 rem --- OPTION ANGLE ---
20 init "CRT2:640,400,16":cls 2
30 option angle radians
40 circle (160,240),100,2,-pai(1/2),-pai(1)
50 option angle degrees
60 circle (440,240),100,5,-30,-285
```





REM （リマーク・remark）

注釈を入れます。

書　式　rem〔〈注釈〉〕

説　明

- この命令は非実行文でプログラムの実行とは関係なく、プログラムリストを見やすくするための注釈（コメント）を付けます。
- BASICは実行時にrem文があるとそれ以後の文字は無視して次の行に進みます。したがって、remのあとにはどんな文字があっても実行には影響しません。
- rem文はマルチステートメント（コロン(:)で2つの文をつなぐこと）で行の途中から始めることもできますが、そのあとにマルチステートメントで文を書いても実行されません。
- rem文のかわりにアポストロフィ(')を使用して、注釈を付けることができます。
- アポストロフィはrem文と同機能ですが、行の先頭からアポストロフィで始まる注釈行は、list＊コマンドでまとめて表示することができます。また、アポストロフィで始まる注釈行を連続して入力するための行番号の発生命令としてauto＊コマンド、ファイルをアポストロフィで始まる注釈行として読み込むための命令としてload＊があります。

例

```
1000 rem メニュー表示
```

参　考☞

list＊ …… アポストロフィで始まるコメント行のみの表示。（145ページ）
auto＊ …… コメント行の連続入力。（147ページ）
load＊ …… コメント行の読み込み （298ページ）

サンプル

```
10 'プログラムの中に
20 'コメントを入れることができます
30 end
```

このように、remの代わりにアポストロフィ(')を使うことができます。

RANDOMIZE（ランダマイズ）

乱数の発生系列を変更します。

書　式　randomize 〔〈初期値〉〕

省略形　ra.

説　明

- rnd関数で発生する乱数の初期化を行います。
- ＜初期値＞は、-32768～32767の整数型のデータを指定し、同じ初期値を与えると同じパターンで発生する再現性があります。
- ＜初期値＞を省略したときは、BASICで任意の初期値を設定して乱数の初期化をします。
- ＜初期値＞によっては発生周期が短い場合がありますので注意してください。
- BASIC起動時の＜初期値＞は6になっています。

参　考☞　rnd　…　乱数発生関数。(363ページ)

サンプル

```
10 rem --- RANDOMIZE ---
20 print:input "初期値を入れてください。";A
30 randomize A
40 for I=1 to 10
50   B=int(rnd(1)*10+.5)
60   print B;
70 next
80 goto 20
90 end
```

- 初期値を変えると、異なる乱数が発生することを確認します。
- 初期値を入力すると、0 から10までの整数型の乱数を発生します。

CLEAR （クリアー）

変数、配列の消去およびBASICエリアの制限を行います。

書　式

clear〔〈機械語フログラム領域のサイズ〉〕
〔,,,〈チャイルドプロセス領域のサイズ〉〕 …(1)

clear max …(2)

省略形

cle.

説　明

- 変数や配列および定義関数の消去を行います。このときには**for～next**文、**repeat～until**文、**while～wend**文などのループ処理、**gosub～retrun**文の戻り先、**if～end if**文などの管理をしているスタック領域も消去されますので、これらの命令の途中でこの命令を実行すると、正常な処理ができなくなるので注意が必要です。
- この命令を実行すると、**open**されているファイルは**close**されますので、注意が必要です。
- 書式(1)は、機械語プログラム領域やチャイルドプロセス領域のサイズを指定して、それらの領域を確保します。

  サイズは指定した値×16バイトのメモリが確保されます。指定されたサイズのメモリが確保できないときは、**no memory error**となります。

  なお、サイズ指定を省略したときは、以前の設定値が指定されます。
- ＜機械語プログラム領域のサイズ＞はBASIC起動時 **0** が設定されています。したがって、機械語プログラムを実行する場合は＜機械語プログラム領域のサイズ＞をあらかじめ指定し、機械語プログラム領域を確保しておく必要があります。
- ＜チャイルドプロセス領域のサイズ＞はBASIC起動時 **0** が設定されています。したがって、チャイルドプロセスを実行する場合は＜チャイルドプロセス領域のサイズ＞をあらかじめ指定し、チャイルドプロセス領域を確保しておく必要があります。

  なお、＜チャイルドプロセス領域のサイズ＞は「起動しようとするチャイルドプロセスのプログラムサイズ」＋**24**キロバイト必要です。

  ただし、スタックセグメントが確保されていないプログラムをチャイルドプロセスとして起動する場合は、さらに**65**キロバイト必要です。

  （**child**文参照）
- **chlid**文で**M**パラメータを指定すると、チャイルドプロセス領域を確保する必要はありません。ただし､この場合チャイルドプロセス実行後はBASICの

直接実行モードになります。また、このとき変数の内容や機械語プログラムは初期化されます。

- <機械語プログラム領域のサイズ>および<チャイルドプロセス領域のサイズ>を指定したときは、以前にこれらの領域にあるメモリの内容は破壊されます。
- 書式(2)は、書式(1)で確保されている<機械語プログラム領域のサイズ>および<チャイルドプロセス領域のサイズ>を **0** の状態に戻します。
  書式(2)は、書式(1)で次のように指定したときと同じ意味を持ちます。
  clear 0,,, 0

サンプル

```
10 rem --- CLEAR ---
20 clear ,,;&H700
30 child "DIR"
40 clear max
```

チャイルドプロセス領域**28**キロバイト確保し、MS-DOSのDIRコマンドを実行後、領域を 0 に戻します。

SWAP（スワップ）

変数の内容を交換します。

書　式　swap 〈変数〉,〈変数〉

省略形　sw.

説　明

- 2つの変数の内容を交換します。2つの変数の型は同じ型で指定します。（整数型,単精度型,倍精度型,文字型の区分がされ、たとえ数値型どうしでも型が異なっているとエラーになります。）
- 変数として同じ型であれば配列の要素も指定できます。

例　swap A#,B#　……　倍精度型変数どうしの内容の交換をします。

サンプル

```
 10 rem --- SWAP ---
 20 dim X(10)
 30 for I=1 to 10
 40 read X(I) : print X(I);
 50 next
 60 print : print : print "並べかえ開始" : print
 70 for A=1 to 10
 80  for B=A+1 to 10
 90   if X(A)=>X(B) then 100 else swap X(A),X(B)
100  next
110 next
120 for L=1 to 10
130 print X(L);
140 next
150 end
160 data 5,3,6,1,8,2,7,9,10,4
run
 5  3  6  1  8  2  7  9  10  4

並べかえ開始

 10  9  8  7  6  5  4  3  2  1
Ok
```

- data文の数値を大きい順に並べ替えて表示します。まずA(1)とA(2)～A(10)を比較し、A(1)の方が小さい場合値を入れ替えます。次にA(2)とA(3)～A(10)を比較しA(2)の方が小さい場合入れ替えます。同様にして順次比較し、入れ替えを行い、並べ替えて表示します。

MID$= （ミドル ダラー イコール・middle$=）

文字列型変数の内容の一部を置き換えます。

書　式

mid$(〈文字列型変数〉,〈開始位置〉[,〈文字数〉])＝〈置き換え文字列〉

説　明

- ＜文字列型変数＞の内容の＜開始位置＞から＜文字数＞(バイト数)分の文字を、＜置き換え文字列＞と置き換えます。
- ＜文字数＞の指定が＜置き換え文字列＞の長さより大きい場合や＜文字数＞を省略した場合は＜置き換え文字列＞すべてが置き換えの対象になります。
- ＜文字列＞の＜開始位置＞より残りの部分が置き換える文字数より少ない場合、もとの＜文字列型変数＞の長さをこえる分は置き換えません。
- この命令の実行では＜文字列型変数＞の長さは変化しません。

サンプル

```
10 rem --- mid$= ---
20 read A$
30 for I=1 to 4
40   read B$:mid$(A$,7)=B$
50   print A$
60 next I
70 data "It is ..............."
80 data "Sunday","Monday","Tuesday","Wednesday"

run
It is Sunday........
It is Monday........
It is Tuesday.......
It is Wednesday.....
Ok
```

SEARCH （サーチ）

配列内の一致するデータをさがします。

書　式

search(〈文字配列名〉,〈文字列〉〔,〈開始位置〉〕
〔,〈ステップ値〉〕〔,〈機能〉〕) …(1)

search(〈整数配列名〉,〈整数〉〔,開始位置〕
〔,〈ステップ値〉〕) …(2)

省略形

sea.

説　明

- ＜開始位置＞はさがし始める位置を要素番号で指定します。範囲をこえる場合はエラーになります。また、指定を省略すると配列の最初からさがします。配列の最初はoption base文の指定により0または1になります。
- ＜ステップ値＞を指定すると、＜ステップ値＞で指定された数ごとに配列要素をさがします。＜ステップ値＞を省略すると、1が指定されたものとみなし、すべての配列要素からさがします。
- (書式1)は1次元の文字配列において、＜文字列式＞を配列の指定された位置よりさがし出し、最初に見つけた位置を値として返す関数です。同じ内容のものが見つからない場合は、結果が-1になります。
  ＜文字配列名＞は一致する文字をさがす配列名で、すでに宣言されている1次元の文字配列でない場合は、エラーになります。
  ＜機能＞には、0または1を指定します。
  0が指定されると、＜文字列＞と同じ値の配列要素のみをさがし、1が指定されると、＜文字列＞を含む配列要素をさがします。
- (書式2)は、1次元の整数配列において、＜整数＞で指定された位置よりさがし出し、最初に見つけた位置を値として返す関数です。数値が見つからない場合は、結果が-1になります。
  ＜整数配列名＞は一致する整数をさがす配列名で、すでに宣言されている1次元の整数配列でない場合は、エラーになります。

サンプル

```
   10 rem --- search ---
   20 dim D$(6)
   30 for A=1 to 6 : read D$(A):next
   40 data ABC,DEF,GHI,JKL,MNO,PQR
   50 input "data ",X$
   60 X=search(D$,X$)
   70 if X=-1 then print "not found":goto 90
   80 print X,D$(X)
   90 print:goto 50
run
data ABC
 1         ABC

data JKL
 4         JKL

data abc
not found

data
*Break in 50
Ok.
```

PRINT　(プリント)
LPRINT　(ライン プリント・line print)

データの表示を行います。

書　式

{print / lprint} 〔〈データ〉〕〔{, / ;} 〈データ〉〕…

省略形

p.　lpr.　または　?　l?

説　明

- printは、<データ>を画面に表示します。
- lprintはプリンタに<データ>を印字します。
- printのかわりに?を使うことができますが、リスト表示をさせるとprintの5文字に変わります。
- データの句切りとしてセミコロン(;)を使うと、前の<データ>に続けて次の<データ>を表示します。
- データの句切りとしてカンマ(,)を使うと10文字ごとのタブ位置に進めて、次のデータを表示します。
- 数値データを表示する場合、値が正のときは左側に1個のスペースを表示し、負のときはマイナス(−)記号を左側に表示します。また、それぞれの値の右側には1個のスペースを表示します。
- カンマやセミコロンを使って数値型データを続けて表示する場合に、表示する<データ>が行末から次の行にまたがって表示されるようなときは自動的に改行して次の行から表示をします。
- プリンタへの出力も上記の自動改行が行われますが、あらかじめ現在のプリンタの印字桁数をwidth lprint文により宣言しておく必要があります。
- データの直前にtab関数を使って左端からの間隔を設定することができます。
- 文字コードが31以下の文字データを画面に表示しようとすると、それぞれのコードに対応した制御を実行します。(制御内容は付録のコントロールコード表を参照してください。)
  これらのコードを画面に表示する場合は、"CRT1:CG"のファイル名でシーケンシャルファイルをopenし、書き込みを行ってください。

参　考☞　lpout　…プリンタへ制御コードを送る。(454ページ)

サンプル

```
10 rem --- PRINT  ---
20 A$="1":A=1:B=-1
30 print A$
40 print A
50 print B
60 print A,B
70 print A;B
80 end
```

```
run
1
 1
-1
 1          -1
 1 -1
Ok
```

TAB （タビュレーション・tabulation）

画面の左端よりの文字表示位置を指定します。

書　式

tab（〈数式〉）

説　明

- print文やlprint文中に用いる関数で、カーソルを現在位置より行の左端から＜数式＞分の間隔を取った位置に移動します。＜数式＞の値は0～255まで指定でき、0が左端になります。
- 現在のカーソル位置が＜数式＞による指定位置より右側にあるときは改行して次の行の指定位置に移動します。

例

```
print tab(12);"A";tab(20);"B"
```

…… 画面の左端から13文字目に"A"、21文字目に"B"を表示します。

サンプル

```
10 rem --- TAB ---
20 print tab(2);"MZ Series"
30 print tab(5);"SHARP";tab(15);"Personal Computer"
40 print tab(5);"SHARP";tab(20);"Personal Computer"
50 print tab(12);"SHARP";tab(22);"MZ Series"

run
  MZ Series
     SHARP      Personal Computer
     SHARP           Personal Computer
            SHARP      MZ Series
Ok
```

PRINT USING（プリント ユージング）
LPRINT USING （ライン プリント ユージング・line print using）

データを指定された書式で表示します。

書 式

{print / lprint} using 〈書式〉;〈データ〉[{, / ;} 〈データ〉]…

省略形

p. us. lpr. us. または ? us. l? us.

説 明

- ＜書式＞に従ったフォーマットで＜データ＞を表示します。
- 書式の制御文字は次のとおりです。

数値型

#	数値表示の桁を指定し、出力は桁の範囲内で右づめで表示します。
.	固定小数点の表示で、小数点位置をそろえる指定です。小数点以下の桁数指定もできます。
,	数値を3桁ごとに句切り、そこにカンマ(,)を入れて右づめで表示します。
＋	数値を指定する記号の前後に付けると、その位置に符号を表示します。
－	数値を指定する記号のあとに付けると、数値が負のときに数値のあとにーを表示します。
＊＊	桁の範囲内で空白となる部分ができたとき、その部分を＊で、埋めます。＊＊も表示する値の2桁分の領域を確保します。
￥￥	表示する数値の前に￥記号を付けます。￥￥で、表示する値の2桁分の領域を確保しますが、1桁分は￥の領域として使われます。
＊＊￥	￥￥と＊＊の機能を合わせたものです。＊＊￥で表示する値の3桁分の領域を確保しますが1文字は￥の領域として使われます。
^^^^	桁指定のあとに付けると指数形式で表示します。
_	制御文字を単に1文字として表示させたいときに使います。_に続く1文字を制御機能を持たない文字として扱います。制御文字以外はそのまま表示します。

数値をこの命令で表示させる場合、指数形式を指定しなければ±1D−16以上±1D16未満が扱える数値の範囲となります。この範囲をこえる数値を指数形式なしで表示させようとすると、エラーになります。

文字型

!	文字列の最初の1文字のみを表示します。
& &	&の間に複数のスペースを指定するとスペースと&を含めた桁数で文字を表示します。文字の長さが桁数に満たない場合は左づめにします。
@	@の部分に文字列を代入します。1つの書式中に複数の指定ができますが、あとに続くデータの個数が少ない場合は不足する部分の@以降を表示しません。

サンプル

```
10 rem --- print using ---
20 A=-1234.45
30 print using "#######";A
40 print using "####,###";A
50 print using "####,###.#";A
60 print using "+###,###.#";A
70 print using "####,###.#-";A
80 print using "####,###.#+";6543.21
90 print using "**##,###";6543.21
100 print using "¥¥##,###";6543.21
110 print using "**¥#,###";6543.21
120 print using "  ####.#^^^^";6543.21
200 A$="Suzuki":B$="Taro"
210 print using "!";B$,A$
220 print using "& &";B$,A$
230 print using "&     &";B$;A$
240 print using "Dear Mr. @ ";A$
```

```
run
  -1234
  -1,234
  -1,234.4
  -1,234.4
   1,234.4-
   6,543.2+
***6,543
  ¥6,543
**¥6,543
   654.3E+01
TS
TarSuz
Taro   Suzuki
Dear Mr. Suzuki
Ok
```

WRITE （ライト）

データを表示します。

書 式

write 〈データ〉[{ , / ; }〈データ〉]…

省略形

w.

説 明

- print文と同様に＜データ＞を表示する命令ですが、データごとの句切としてカンマ(,)を付け、間隔を開けずに表示します。また、文字列型データはダブルクォーテーション(")で囲んで表示します。
- 句切記号のカンマとセミコロン(;)の機能は同じで、すべてのデータを詰めて表示します。

サンプル

```
10 rem --- write ---
20 A=-123:B=5678:C$="data":D$="personal computer"
30 print A;B;C$,D$
40 write A;B;C$,D$

run
-123  5678 data      personal computer
-123,5678,"data","personal computer"
Ok
```

INPUT （インプット）

キーよりデータの入力を行います。

書　式

input〔〈コメント〉{ ; / , }〕〈入力変数〉〔,〈入力変数〉〕

省略形

i.

説　明

- この命令を実行させると、カーソルを点滅させキー入力状態になり、データが入力されると変数に代入します。
- ＜コメント＞は文字定数または文字列式で指定します。実行時には＜コメント＞を表示し、そのあとにカーソルを表示します。＜コメント＞を省略すると、"？　"を表示して入力状態となります。コメントの句切記号でカンマ(,)を使った場合は＜コメント＞を表示し、そのあとにカーソルを表示しますが、セミコロン(;)を使った場合は＜コメント＞の直後に"？　"を表示してその右側にカーソルを表示します。
- 入力はカーソルが点滅したあとでデータを入力し、[↵] キーを押すことにより完了します。＜入力変数＞を複数並べた場合は、カンマで句切って入力します。
- 文字列型変数の入力はダブルクォーテーション(")で囲むのを省略できます。ただし、カンマはデータの句切りとして扱われますのでカンマを含む文字列を入力するときはダブルクォーテーションで囲む必要があります。
- 変数の型や個数が、入力したデータと一致しない場合は、エラーを表示して、再度入力を要求します。
- 何も入力せずに [↵] キーを押すと数値変数には0、文字列変数には文字長が0のNull(空文字)を代入します。ただし、＜入力変数＞が複数のときは句切りのカンマが必要です。

サンプル

```
10 rem --- INPUT ---
20 input A$
30 print "コメントなしで入力されたものは ";A$;" です。"
40 input "数値を入力してください ";B
50 input "文字を入力してください ";B$
60 print "コメントありで入力されたものは ";B;" と ";B$;" です。"
70 input "; を使うと？が表示されます";C$
80 input ", を使うと？が表示されません",D$
90 print "今、入力したものは ";C$;" と ";D$;" です。"
100 input ", で区切って３回入力してください。";E$,F$,G$
110 print "３回入力したものは ";E$;" と ";F$;" と ";G$;" です。"
120 end
```

```
run
? Hello
コメントなしで入力されたものは Hello です。
数値を入力してください ? 012345
文字を入力してください ? ABC
コメントありで入力されたものは  12345  と ABC です。
; を使うと ? が表示されます? Goodnight
, を使うと ? が表示されませんGoodmorning
今、入力したものは Goodnight と Goodmorning です。
, で区切って3回入力してください。? abc,DEF,ghi
3回入力したものは abc と DEF と ghi です。
```

- 1行目はコメントのないとき、3，4行目はコメントのあるときの動作例です。コメントと変数との句切りのセミコロン(;)とコンマ(,)の違いは6，7行のようになります。
- 連続入力はコンマ(,)で句切って9行目のように行います。

LINE INPUT （ライン インプット）

キーより1行分のデータを入力します。

書 式

line input〔〈コメント〉{ ; / , }〕〈入力文字型変数〉

省略形

lin. i.

説 明

- この命令を実行させると、カーソルを点滅させキー入力状態になり、データが入力されると、入力データの1行全体を文字型変数に代入します。
- 入力はカーソルが点滅したあとでデータを入力し、[↵] キーを押すことにより完了します。
- **input**文と違い、入力の途中にカンマ(,)があっても句切りとみなされず、ダブルクォーテーション(")も入力できます。カーソルが最初に点滅した所から行の終りまでを入力します。
- ＜コメント＞の句切り記号のカンマ(,)とセミコロン(;)の機能は同じで、＜コメント＞を表示した直後にカーソルを点滅させます。
- この命令は、入力時には"？ "を表示しません。

参 考

input …… データのキー入力（180ページ）

line input wait …… 待ち時間を設定してのキー入力。（183ページ）

サンプル

```
10 rem --- LINE INPUT ---
20 input ", と" は使用しないで入力してください。";A$
30 print A$
40 line input ", と" を使用することができます。";B$
50 print B$
60 end

run
, と" は使用しないで入力してください。？ abcdefg
abcdefg
, と" を使用することができます。a,b,c,d,e,f,g
a,b,c,d,e,f,g
```

- **input**文ではコンマ(,)とダブルクォーテーション(")をそのまま文字として入力できませんが、**line input**文ではリターンキーを押すまで入力した文字は全て文字として変数に代入されます。

INPUT WAIT （インプット ウェイト）
LINE INPUT WAIT （ライン インプット ウェイト）

キー入力の時間制御を行います。

書 式

input wait 〈待ち時間〉,〔〈コメント〉{ ; / , }〕〈変数〉,〔〈変数〉〕…

line input wait 〈待ち時間〉,〔〈コメント〉{ ; / , }〕〈文字型変数〉

省略形

i. wa.　　lin. i. wa.

説 明

- ＜待ち時間＞で設定された時間だけキー入力を待ちます。＜待ち時間＞の単位は0.1秒単位で1～65535までの値で指定します。
- 待ち時間以外の機能はinput文またはline input文と同じですが、この命令のあとにコロン(:)でマルチステートメントにすると、制限時間内に入力された場合は実行され、入力されなかった場合は、コロン以降を実行せずに次の行から実行します。
- 制限時間内に入力されなかった場合の＜入力変数＞の値は以前のままで変化しません。

参 考

input …… キー入力の実行。(180ページ)

line input …… 1行分のキー入力の実行。(182ページ)

サンプル

```
10 rem --- INPUT WAIT ---
20 cls
30 locate 15,5
40 print time$
50 T$=""
60 locate 0,8
70 input wait 80,"変更する時刻を入力してください。(hh:mm:ss): ",T$
80 if T$<>"" then time$=T$:goto 30
90 end
```

- 現在の時刻を表示し、8秒間時刻の変更の入力待ちとなります。入力されれば新しい時刻を表示しくり返します。
- 入力待ち時間内に入力が行われなかった場合は終了します。

READ～DATA （リード～データ）

プログラム中に書かれたデータを変数に読み込みます。

書　式

read 〈入力変数〉[,〈入力変数〉]…
data 〈定数〉[,〈定数〉]…

省略形

rea.～da.

説　明

- read文は、data文の中に＜定数＞で書かれているデータを＜入力変数＞に1つずつ読み込みます。＜入力変数＞は数値型でも文字型でも指定できますが、変数とデータの型が合わない場合はエラーになります。
- data文はread文で＜入力変数＞に読み取られるデータを書きます。文字型のダブルクォーテーション(")は省略できますが、句切り記号としてカンマ(,)が使われていますのでカンマを含むデータとしたいときは、ダブルクォーテーションで囲む必要があります。また、アポストロフィ(')やコロン(:)およびデータの前後に意味のある空白を含むデータの場合も、ダブルクォーテーションで囲む必要があります。
- data文はプログラム中のどこにあってもよく、実行の順番がきても何もせずに次の行に進みます。
- データの読み込みはread文で＜入力変数＞が指定されるごとに、行番号の小さい方から順次data文をさがし、data文の中では左から右へと順番に行われます。順番にデータを読み込んでいき、読み取るデータが無くなっても読み込もうとするとエラーになります。
- restore文によって、読み取る行を変更することができ、同じデータを何回も読み直したり、いくつかのdata文を読み飛ばしたりすることができます。
- 現在のデータの読み取り位置を行番号で求めるシステム変数としてdtlがあります。

参　考

restore …… データ読み取り行の変更。(187ページ)
dtl …… 現在のデータ読み取り行を求めるシステム変数。(186ページ)

サンプル

```
10 rem --- READ-DATA ---
20 dim A$(12)
30 for I=1 to 12
40   read A$(I)
50   print A$(I);"  ";
60 next I
70 rem --- 干支のデータ ---
80 data "子","丑","寅","卯","辰","巳","午","未","申","酉","戌","亥"
90 end
```

```
run
子 丑 寅 卯 辰 巳 午 未 申 酉 戌 亥
Ok
```

- **data**文に書かれた12個の文字型定数を**read**文によって配列**A$**に代入します。

DTL （データライン・data line）

read文によるデータの読み取り行を求めます。

書　式　dtl

説　明
- read文によるデータ読み取り位置を記憶しているシステム変数で、値は行番号を示します。
- 値は直前に読み取った行を示しますので、まだデータを読み取っていない状態では0になります。

例

```
10 if dtl>500 then restore
```

…… データの読み取り位置が500をこえた場合は先頭に読み取り位置を移します。

参　考　restore …… データの読み取り位置を変更する。(187ページ)

RESTORE （リストア）

データの読み取り位置を変更します。

書　式　restore〔〈読み取り開始行〉〕

省略形　res.

説　明

- read文によるデータの読み取り位置を変更します。＜読み取り開始行＞を0または省略するとプログラムの最初に位置を移します。
- ＜読み取り開始行＞を指定するとその行のdata文または、それ以後で最も小さい行番号をもつdata文から読み取りを開始します。
- ＜読み取り開始行＞は、行番号またはラベルで指定します。

参　考　read～data　……　データを変数に読み込む。(184ページ)

dtl　……　現在のデータ読み取り行を求める。(186ページ)

サンプル

```
 10 rem --- RESTORE ---
 20 gosub 100
 30 gosub 100
 40 print
 50 restore 600:gosub 100
 60 restore 500:gosub 100
 70 end
100 read A$:print A$
110 return
500 data "D,E,F"
600 data "A,B,C"

run
D,E,F
A,B,C

A,B,C
D,E,F
Ok
```

GOTO （ゴーツー）

指定された行へ実行を移します。

書　式　goto〈ジャンプ先行〉

省略形　g.

説　明
- ＜ジャンプ先行＞で指定された行へ実行が移り、そこからプログラムの実行を続けます。
- ＜ジャンプ先行＞は、行番号またはラベルで指定できます。
- この命令は、go toと離して書くこともできます。

例　goto *LOOP …… ラベル名が＊ＬＯＯＰの行へ実行を移します。

参　考　on～goto …… 式の値によりジャンプ先を変更する。(191ページ)
gosub～return …… サブルーチンの実行。(189ページ)

サンプル

```
10 print "goto sample ";
20 goto 10
```

- "goto sample" を続けて表示します。
- プログラムの実行中止は[シフト]+[停止]キーを押してください。

GOSUB~RETURN (ゴーツーサブルーチン~リターン・go to subroutine~return)

サブルーチンを実行します。

書　式

gosub〈開始行〉
　⋮
return〔〈戻り行〉〕

省略形

gos. ～re.

説　明

- gosub文はgoto文と同様に指定された＜開始行＞へ実行を移しますが、その先にreturn文があると元のgosub文の次の文へ戻ります。
- サブルーチンの中でサブルーチンを実行することもでき、これをネスティング(入れ子型)と呼んでいます。ネスティングの深さ(回数)には制限はありませんが、メモリ(スタックエリア)の残量で制約を受けることがあります。
- return文のあとに＜戻り行＞を指定すると元のgosub文の所以外へ戻ることができます。
- ＜開始行＞と＜戻り行＞は、行番号またはラベルで指定できます。
- gosub文で呼ばれずにreturn文を単独で実行するとエラーになります。
- この命令は、go subと離して書くこともできます。

参　考

goto …… 指定行へ実行を移す。(188ページ)
on～gosub …… 式の値によりサブルーチンを変更する。(191ページ)
on～return …… 式の値により戻り先を変更する。(191ページ)

サンプル

```
10 rem ── GOSUB RETURN ──
20 print "次の計算をしなさい。":print
30 for I=1 to 5
40   A1=int(rnd(1)*10)
50   A2=int(rnd(1)*10)
60   A3=int(rnd(1)*10)
70   color 7:print A1;" + ";A2;" * ";A3;" = ";:ANS=A1+A2*A3
80   input Q
90   if Q<>ANS then gosub *MACHIGAI else gosub *GOOD
100 next I
110 color 7:end
120 *GOOD
130 color 5:print "正解です。"
140 return
150 *MACHIGAI
160 color 2:print "間違いですよ。"
170 return
```

```
run
次の計算をしなさい。

 7 + 7 * 4 = ? 30
間違いですよ。
 3 + 2 * 2 = ? 10
間違いですよ。
 3 + 8 * 0 = ? 3
正解です。
 1 + 8 * 2 = ? 17
正解です。
 9 + 0 * 6 = ? 9
正解です。
Ok
```

- 計算問題に対して正しい答えが入力された場合は、ラベル名**＊GOOD**で始まるサブルーチン実行を移します。また、間違った答が入力された場合は、ラベル名**MACHIGAI**で始まるサブルーチンに実行を移します。

ON ～ GOTO （オン～ゴーツー）
ON ～ GOSUB （オン～ゴーツーサブルーチン・on~go to subroutine）
ON ～ RETURN （オン～リターン）
ON ～ RESTORE （オン～リストア）
ON ～ RESUME （オン～リジューム）

式の値により参照行を変更します。

書　式

on 〈数値式〉 { goto / gosub / return / restore / resume } 〔〈行1〉〕〔,〈行2〉〕〔,〈行3〉〕…

省略形

o. ～g.　　～gos.　　～re.　　～res.　　～resu.

説　明

- ＜数値式＞の値が1であれば＜行1＞、2であれば＜行2＞、3であれば＜行3＞…というように、＜数値式＞の値により＜行＞の並びの何番目であるかが決定され、goto文やgosub文などの各機能を実行します。
- ＜数値式＞の小数部は四捨五入が行われ、値が0以下または並びの個数をこえる場合は何も実行せず次の文に進みます。
- ＜行＞は、行番号またはラベルで指定できます。

参　考

goto …… 指定行へ実行を移す。(188ページ)

gosub～return …… サブルーチンの実行。(189ページ)

restore …… データの読み出し位置の設定。(187ページ)

resume …… エラー処理の修了。(402ページ)

サンプル

```
 10 rem ── on goto ──
 20 input "数値を入力してください ";A
 30 on sgn(A)+2 gosub 100,200,300
 40 end
100 print "負数":return
200 print "ゼロ":return
300 print "正数":return

run
数値を入力してください ? 123
正数
Ok
run
数値を入力してください ? -123
負数
Ok
```

IF～THEN～ELSE （イフ～ゼン～エルス）

条件により処理を変えます。

書　式

if　＜条件式＞〔 { then { ⟨文⟩〔:⟨文⟩〕… / ⟨行⟩ } / goto　⟨行⟩ / gosub　⟨行⟩〔:⟨文⟩〕… } 〕〔else { ⟨文⟩〔:⟨文⟩〕… / ⟨行⟩ } 〕

省略形

if～th. el.

説　明

- ＜条件式＞の条件に合う場合はthenに続く文を実行し、条件に合わない場合はelse以降の文を実行します。
- ＜条件式＞には関係演算子（＞＜や＝）を使った関係式、論理式、算術式などの結果が数値となる式を使うことができ、式の値が0でなければthen以降、0ならばelse以降の実行と判断します。
- thenに続く文がgoto文の場合、then goto＜行＞としますが、thenまたはgotoのどちらかを省略して、goto　＜行＞またはthen　＜行＞とすることができます。then gosub　＜行＞の場合はthenを省略してgosub　＜行＞とすることができます。
- else goto ＜行＞の場合はgotoを省略してelse ＜行＞とすることができます。
- if文の中にif文を書いてネスティング（入れ子型）にすることができます。

  if　A＝1　then　if　B＝1　then　Ⓐ　else　Ⓑ　else　Ⓒ

  このような場合ⒶはA＝1とB＝1の条件のときに実行、ⒷはA＝1とB≠1のときに実行、ⒸはA≠1のときに実行されます。
- elseはその直前にあるifと対になっているとみなされます。

  if A＝0 if B＝0 then Ⓐ else Ⓑ

  このような場合 Ⓑ は A＝0，B≠0 のとき実行されます。

サンプル

```
 10 rem --- IF THEN ELSE ---
 20 input "1番目の数字は ";A
 30 input "2番目の数字は ";B
 40 if A=B then gosub *ONAJI:goto 60
 50 if A>B then gosub *DAI else gosub *SHOU
 60 print
 70 input "1番目の文字は ";A$
 80 input "2番目の文字は ";B$
 90 if A$=B$ then gosub *ONAJI:goto 110
100 if A$>B$ then gosub *DAI else gosub *SHOU
110 end
120 *DAI
130 print "1番目の方が大きいです。"
140 return
150 *SHOU
```

```
160 print "2番目の方が大きいです。"
170 return
180 *ONAJI
190 print "どちらも同じです。"
200 return

run
1番目の数字は ? 5
2番目の数字は ? 3
1番目の方が大きいです。

1番目の文字は ? W
2番目の文字は ? f
2番目の方が大きいです。
Ok
```

- 入力された数値および文字(文字コード)の大小を比較します。

ELSE IF~THEN……END IF (エルス イフ~ゼン……エンド イフ)

IF~THEN~ELSE機能の複数行型式を行います。

書 式

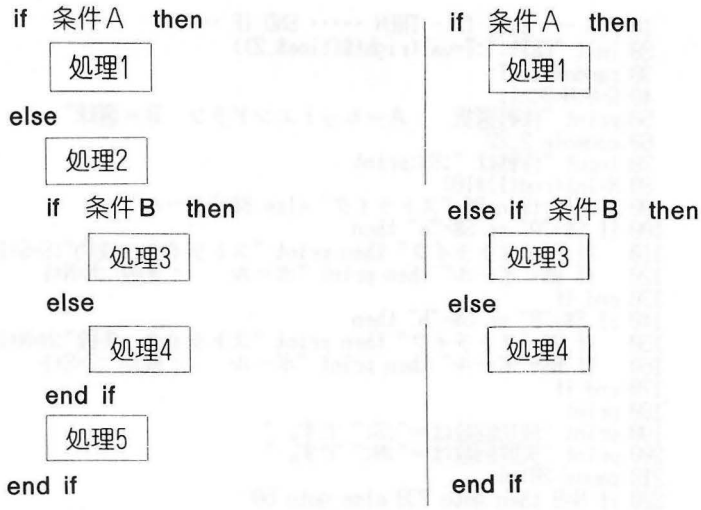
```
if 〈条件式〉 then
  〈文〉
   ⋮
[else if 〈条件式〉 then
  〈文〉
   ⋮ ]…
[else
  〈文〉
   ⋮ ]
end if
```

省略形

el. if~th. ……en. if

説 明

- 行の最初から始まるif文で、thenのあとに何も書かず行が終わる場合、条件に合う処理を複数行に渡って書くことができます。条件に合わない処理は、elseだけ書かれている行の次から複数行に渡って書くことができ、終了はend if文になります。
- if~then文とend if文は1対1で対応させる必要があります。
- else if~then文を使うことにより次々に条件ごとの処理を書いていくことができます。(次ページ参照)
- 複数行型式でのif~then文, else文, else if~then文, end if文のそれぞれの文は左右にマルチステートメント(コロン(:)で2つの文をつなぐこと)でほかの文を書くことはできません。



〔例1〕

例1の左は条件のネスティングを使用したもので、右のelse if～thenと対比させたものです。各処理と条件の関係は次のようになります。

処理1	条件A…○	条件B…無関係	
処理2	条件A…×	〃 …無関係	○…条件に合う
処理3	条件A…×	〃 …○	×…条件に合わない
処理4	条件A…×	〃 …×	
処理5	条件A…×	〃 …無関係	

右側の各処理も左側と同じ条件での実行となりますが、処理2と処理5がないのでelse ifを使用するとend ifは1つですみます。

例1は処理の流れを理解するためのもので実際には次のような使い方をします。

```
if   <条件1>  then
  処理1
else if  <条件2>  then
  処理2
else if  <条件3>  then
  処理3
else if  <条件4>  then
  処理4
   ⋮
end if
```

〔例2〕

条件1に合う場合は処理1、条件2に合う場合は処理2…というように順に条件をチェックして各処理を行う場合に便利な機能です。

サンプル

```
 10 rem --- ELSE IF - THEN ····· END IF ---
 20 init "CRT1:":T=val(right$(time$,2))
 30 randomize T
 40 S=0:N=0:cls
 50 print "作戦選択    A＝ヒットエンドラン  B＝選球"
 60 console 2,22
 70 input "作戦は ";S$:print
 80 X=int(rnd(1)*10)
 90 if X<5 then B$="ストライク" else B$="ボール"
100 if S$="A" or S$="a" then
110   if B$="ストライク" then print "ストライク：成功":S=S+1
120   if B$="ボール" then print "ボール     ：失敗":N=N+1
130 end if
140 if S$="B" or S$="b" then
150   if B$="ストライク" then print "ストライク：失敗":N=N+1
160   if B$="ボール" then print "ボール     ：成功":S=S+1
170 end if
180 print
190 print "成功回数は＝";S;" です。"
200 print "失敗回数は＝";N;" です。"
210 pause 20:cls
220 if N=9 then goto 230 else goto 60
230 console:end
```

```
作戦選択    A＝ヒットエンドラン  B＝選球

作戦は ? A

ストライク：成功

成功回数は＝ 5  です。
失敗回数は＝ 0  です。
*Break in 210
Ok.
```

- 選択した作戦が球種によって成功か失敗かを**if～then～end if**命令で判定します。

FOR～NEXT （フォア～ネクスト）

forとnextの間の文を繰り返し実行します。

書　式

for 〈カウント変数〉＝〈初期値〉 to 〈最終値〉〔step 〈増分〉〕
　　⋮
next 〔〈カウント変数〉〕〔,〈カウント変数〉〕…

省略形

f. ～ n.

説　明

- for文は＜カウント変数＞を＜増分＞ずつ増やしながら＜最終値＞をこえるまでnext文の間を繰り返し実行します。
- 最初に＜カウント変数＞に＜初期値＞を代入し、＜最終値＞をこえるかの判断を行いますので、たとえば＜増分＞が正の数で＜最終値＞が＜初期値＞より小さい場合は繰り返し部分を１度も実行せずにnext文へ進みます。
- for文の所で対応するnext文をチェックしますので、対応するnext文がない場合はエラーになり、また1つのfor文に対して複数のnext文を対応させることはできません。
- 繰り返し(ループ)の中でgoto文により飛び出した場合は、ループが終了したとみなします。
- stepの指定を省略すると＜増分＞として1が指定されたとみなします。
- このループを終了したあとの＜カウント変数＞の値は、最後の繰り返し時の変数値に＜増分＞を加えた値になります。たとえば、for A=1 to 10のループの終了後のAの値は、最終の繰り返し値10に＜増分＞1を加えた値11になり、for A=1 to 10 step 2の場合は、最終の繰り返し値が9でループ終了後のAの値は11になります。

サンプル

```
10 rem ── for next ──
20 print "数値を入れてください";:input A
30 print " NUMBER","SQUARE","SQUARE ROOT"
40 for I=1 to A
50   print I,I*I,sqr(I)
60 next

run
数値を入れてください? 4
 NUMBER    SQUARE     SQUARE ROOT
 1         1          1
 2         4          1.4142136
 3         9          1.7320508
 4         16         2
Ok
```

REPEAT～UNTIL （リピート～アンティル）

条件により繰り返し部分に戻るかを判断します。

書　式

repeat
　⋮
until 〈条件式〉

省略形

rep. ～u.

説　明

- untilのあとの＜条件式＞に合わなければrepeat文の次の文まで戻ります。
- 繰り返すかの判断は**until**文で行われますので、１回は繰り返し部分を実行します。
- **repeat**文と**until**文は１対１で対応し、ネスティング(入れ子型)させることができますが、繰り返し(ループ)の中から外へ**goto**文により飛び出した場合はループが終了したとみなします。
- repeat文実行時に対応する**until**文がない場合、またはrepeat文を実行せずにuntil文を実行した場合はエラーになります。

サンプル

```
   10 rem --- repeat until ---
   20 I=0
   30 print "数値を入れてください";:input A
   40 print " NUMBER","SQUARE","SQUARE ROOT"
   50 repeat
   60   print I,I*I,sqr(I)
   70   I=I+1
   80 until I>A
Ok
run
数値を入れてください? 7
 NUMBER    SQUARE     SQUARE ROOT
 0          0          0
 1          1          1
 2          4          1.4142136
 3          9          1.7320508
 4          16         2
 5          25         2.236068
 6          36         2.4494897
 7          49         2.6457513
Ok
```

WHILE~WEND (ホワイル~ホワイル エンド・while~while end)

whileの条件に合えばwend間を繰り返します。

書 式

while 〈条件式〉
 ⋮
wend

省略形

wh. ～we.

説 明

- ＜条件式＞の条件に合えばwhile文とwend文の間を実行し、条件に合わない場合はwend文の次の文から実行します。
- while文で繰り返すかの判断をしますので、繰り返し部分を1回も実行しない場合があります。
- while文とwend文は１対１で対応し、ネスティング(入れ子型)させることができますが、繰り返し(ループ)の中から外へgoto文により飛び出した場合はループが終了したとみなします。
- while文実行時に対応するwend文がない場合や、while文を実行せずにwend文が実行された場合は、エラーになります。

サンプル

```
10 rem --- while wend ---
20 I=0
30 print "数値を入れてください";:input A
40 print " NUMBER","SQUARE","SQUARE ROOT"
50 while I<=A
60   print I,I*I,sqr(I)
70   I=I+1
80 wend

run
数値を入れてください? 5
 NUMBER    SQUARE     SQUARE ROOT
 0          0          0
 1          1          1
 2          4          1.4142136
 3          9          1.7320508
 4          16         2
 5          25         2.236068
Ok
```

PAUSE （ポーズ）

プログラムの実行の一時停止をします。

書　式　pause ＜停止時間＞

省略形　pau.

説　明

- この命令を実行すると＜停止時間＞で指定された時間が経過したあと次の処理に進みます。
- ＜停止時間＞は、0.1秒単位で1～65535の値で指定します。
- この命令で一時停止状態のときは、すべての割り込み制御はstopになり、停止状態で割り込みがあった場合は、この命令の終了後に割り込み処理を実行します。
  したがって、一時停止状態のときは[シフト]+[停止]キーで実行を中止することはできません。

例

```
100 pause 100
```
…… 10秒間実行を停止します。

サンプル

```
10 rem --- PAUSE ---
20 cls
30 for I=1 to 10
40   read MES$(I)
50   print MES$(I)
60   pause 20
70 next
80 end
90 data "ある所に、"
100 data "おじいさんと"
110 data "おばあさんが"
120 data "住んでいました。"
130 data "おじいさんは"
140 data "山へ柴刈りに、"
150 data "おばあさんは"
160 data "川へ洗濯に、"
170 data "行きました。"
180 data "おしまい！"

ある所に、
おじいさんと
おばあさんが
住んでいました。
おじいさんは
山へ柴刈りに、
おばあさんは
川へ洗濯に、
行きました。
おしまい！
Ok
```

STOP （ストップ）

プログラムの実行を中止します。

書　式　stop

省略形　s.

説　明

- プログラムの実行を中止して、命令待ち状態にします。
- このときBASICは次のメッセージを出します。

  Stop in nn　… nnは中止した行番号
- end文と異なりファイルを閉じたり各種の初期化は行われません。
- この命令で実行が中断された状態から、contコマンドやstepコマンドにより実行を再開することができます。ただし、実行の中断時にプログラムの変更を行うとend文の実行と同様の処理が行われ、contコマンドやstepコマンドによるプログラムの再開ができなくなります。
- この命令で実行を中断するときに割り込み処理の制御（key on/off/stopなど）がonになっている場合、stop状態にして中断します。contコマンドやstepコマンドによる実行再開により、割り込み処理の制御がonになります。

例　999 stop …… 行番号999で実行を中断します。

参　考☞
- end …… プログラムの終了。（202ページ）
- stop（コマンド） …… ブレークポイントの設定。（138ページ）
- cont …… 中断されたプログラムの実行再開。（136ページ）
- step …… プログラムのステップ実行。（137ページ）

サンプル　contのサンプル参照

END （エンド）

プログラムの実行を終了します。

書　式　end

省略形　en.

説　明

- end文を実行するとOkを表示して命令待ちに戻ります。
- この命令の実行で次の処理が行われます。
  - 操作途中のファイルをすべて終了します。（close）。
  - データの読み出し位置をプログラムの先頭にします。（restore）
  - エラー処理行の定義を解除します。（on error goto 0）
  - 割り込み処理行の定義と、制御を解除します。（on key gosub文、key off文など）
- プログラムの最後にend文を省略しても実行を終了しますが、この場合は以上の処理は行いません。

例　999 end　……　行番号999で実行を終了します。

参　考　stop　……　プログラムの中断。(201ページ)

サンプル

```
10 rem --- end ---
20 input A
30 if A=0 then end
40 goto 20

run
? 5
? 2
? 0
Ok
```

0を入力すると、30行でendが実行されてプログラムを終了します。

OPTION SCREEN （オプション スクリーン）

グラフィックV-RAMの使用目的を設定します。

書　式　option screen〈機能指定〉

省略形　opt. sc.

説　明

- グラフィック**V-RAM**の一部を表示用以外のメモリディスク(RAMディスク)やプリンタスプーラ用メモリとして使用することができます。この命令では**V-RAM**のどこまでを表示用として使用するかを設定します。
- ＜機能指定＞は**1～5**の値で指定します。指定による表示用とメモリディスクやプリンタスプーラ用の使用比率と、そのときに表示可能なグラフィック画面数は次のようになります。

機能指定	表示用領域	メモリディスク プリンタスプーラ領域	640×200ドット		640×400ドット	
			16色	65536色	16色	65536色
1	512キロバイト	0	8面	2面	4面	1面
2	384キロバイト	128キロバイト	6面	1面	3面	表示不可
3	256キロバイト	256キロバイト	4面	1面	2面	表示不可
4	128キロバイト	384キロバイト	2面	表示不可	1面	表示不可
5	0	512キロバイト	表示不可	表示不可	表示不可	

なお、グラフィック1画面表示するに当たって必要な領域は次のとおりです。

640×200ドット16色モード時　:64キロバイト
640×400ドット16色モード時　:128キロバイト
640×200ドット65536色モード時:256キロバイト
640×400ドット65536色モード時:512キロバイト

- メモリディスクやスプーラが設定されているエリアを表示用に使用しようとするとエラーになります。あらかじめ**init"MEM:"**によりメモリディスクやスプーラの領域を小さくしてください。
- ＜機能指定＞で**1～4**が指定されると、640×400ドット16色になり、グラフィックの表示画面（出力スクリーン）と入力画面（アクティブスクリーン）共にスクリーン**1**に設定されます。
- BASIC起動時の＜機能指定＞は**1**になります。

- **option screen**文の実行によって、**V-RAM**の内容は変化しません。また、メモリディスクやプリンタスプーラ領域の内容にも影響を与えません。
- メモリディスクはMS-DOS上で設定したものがそのまま使用されますので、MS-DOS上メモリディスク(RAMディスク)として**V-RAM**が使用されている場合は、BASIC起動時にその領域は表示用に割り当てられていません。BASIC起動時には、その時メモリディスクがグラフィック**V-RAM**全体を使用している場合は、起動メッセージとして“グラフィック表示メモリは利用できません”と表示されます。

参　考ほか　**init "MEM:"** …… メモリディスクとプリンタスプーラの設定。(284ページ)

サンプル

```
10 rem --- OPTION SCREEN ---
20 init "MEM:0,0"
30 option screen 4
40 A=fre(5)/1024
50 init "MEM:"+str$(A)+",0"
```

…**V-RAM**を表示用に128キロバイト使用し、**V-RAM**の残りメモリ領域と増設**RAM**の容量の大きい方すべてをメモリディスクに割り当てます。

INIT "CRT2:" (イニシャライズ シー アール ティー 2・initialize C,R,T,2)

グラフィック表示の初期化を行います。

書 式　init "CRT2:〔〈横ドット数〉,〈縦ドット数〉,〈表示色数〉〕"

説 明

- グラフィック表示の解像度と同時表示色を設定します。
- この命令で指定可能な組み合わせは次のとおりです。

＜横ドット数＞	＜縦ドット数＞	＜表示色数＞
640	200	16
		65536
	400	16
		65536

- この命令の実行により次の初期設定を行います。
  - ・ウィンドウ、ビューポートおよび表示出力の初期化。
    (view:view@:window:roll@)
  - ・**screen,** フォアグランドカラー・バックグラウンドカラーの初期化。
  - ・パレットやプライオリティの初期化。
- この命令の実行では、グラフィック**V-RAM**の内容の消去は行われません。
- 使用する画面数は、**option screen**文で指定します。(**option screen**文参照)
- **option screen**文の設定によって、使用できないグラフィックモードになっているものを指定した場合はエラーになります。
- グラフィック画面のうち、入力を行う画面および表示を行う画面の指定は**screen**文で指定します。
- すべての指定を省略すると、次のように設定します。

  init "CRT2:640, 400, 16"
- メモリディスクやプリンタスプーラが使用されていたり、**option screen**の設定で表示用のグラフィックVーRAMが少ないと、65536色モードに設定できないことがあります。

  この場合、以下の操作を行うことで、65536色モードに設定することができるようになります。

```
init "MEM:0,0"
option screen 1
init "CRT 2 : 640, 400(または200), 65536"
```

例

init "CRT2:640,200,16" …… 640×200ドットの16色モードに設定します。

サンプル

```
 10 rem --- INIT "CRT2:" ---
 20 cls 3
 30 init "crt2:640,200,16":gosub *DISP
 40 init "crt2:640,400,16":gosub *DISP
 50 end
 60 *DISP
 70 circle (200,50),50,10
 80 line (100,120)-(500,150),9,BF
 90 pause 30:cls 3
100 return
```

INIT "CRT1:" (イニシャライズ シー アール ティー 1・initialize C.R.T.1)

文字画面の初期化を行います。

書 式

init "CRT〔1〕:〔〈桁数〉〕〔,〈行数〉〕〔,〈フォント〉〕〔,〈スクロール〉〕"

説 明

- 文字画面に対して、桁数、行数、文字の種類、スクロールの方法を指定し、初期化を行います。このとき文字画面の消去も行われます。
- ＜桁数＞は**40**または**80**で指定します。
- ＜行数＞の指定は**12**、**20**、**25**の指定ができます。12行のときは、半角文字のフォントが8×16ドットになります。
- ＜フォント＞は、半角文字の字体を８×８のドットにするか、８×16ドットにするかを指定します。
  0の場合は８×８ドット、**1**の場合は８×16ドットになります。
  なお、12行モードのときは８×16ドットのみ指定可能です。
- ＜スクロール＞は、スクロールの方法を**0**～**2**で指定します。**0**の場合は行単位のラインスクロール、**1**の場合は低速のスムーススクロール、**2**の場合は高速スムーススクロールになります。
  高速のスムーススクロールモードにした場合、表示する文字数が多いときにはなめらかにスクロールされない場合があります。
- ＜桁数＞で40桁の指定をしたときはスムーススクロールはできません。
- スムーススクロールモード時に、**console**文によるスクロール範囲の指定や**width**文での桁数の変更を行うとスムーススクロールが行えなくなります。
  また、画面の最下行は表示には使用されず、ファンクションキーの表示はできなくなります。
- すべての指定を省略すると次のように設定します。
  init "CRT1:80, 25, 1, 0"
- 部分的な省略を行った場合は、省略部分はそれまでの設定値となります。
- この命令の実行により次の初期設定が行われます。
  文字画面の消去。(cls 1)
  文字表示色の初期化。(color)
  文字属性の初期化。(cgen:cflash:crev)
  文字入出力画面の初期化。(screen 0,(0))
  カーソルの表示。(locate ,,2)

スクロール範囲の初期化。(console)

ディスプレイ出力範囲の初期化。(console@)

例

```
init "CRT1:80,25,1,1"
```

…… 80桁、25行、8×16ドットの低速スムーススクロールに設定します。

サンプル

```
 10 rem --- INIT "CRT1:" ---
 20 init "CRT1:40,12,1,0" : gosub *PRINT
 30 init "CRT1:40,20,1,0" : gosub *PRINT
 40 init "CRT1:40,25,1,0" : gosub *PRINT
 50 init "CRT1:80,12,1,0" : gosub *PRINT
 60 init "CRT1:80,20,1,0" : gosub *PRINT
 70 init "CRT1:80,25,1,0" : gosub *PRINT
 80 end
 90 *PRINT
100 for I=1 to 10
110   print "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
120 next
130 pause 10
140 return
```

- 文字画面の桁数・行数を変えて表示します。

SCREEN （スクリーン）

文字およびグラフィック画面の入出力設定を行います。

書　式

screen〔T_A〕〔,(To_1〔,To_2〕)〕〔,G_A〕〔,(Go)〕〔,P_A〕〔,(Po)〕

T_A…〈文字入力〉　　To_1, To_2 …〈文字出力〉

G_A…〈グラフィック入力〉　　Go　…〈グラフィック出力〉

P_A…〈入力プレーン〉　　Po　…〈出力プレーン〉

省略形

sc.

説　明

- 文字画面・グラフィック画面は、モードにより、複数の画面をもつことができます。
  この命令では、それぞれの画面に対して、入力を行う画面（アクティブスクリーン）と表示（出力）を行う画面（出力スクリーン）を指定します。
  また、グラフィック画面はプレーンごとの入出力も設定できます。
- 表示モードによる画面数は次のようになります。

	横サイズ	縦サイズ	表　示　色	画　面　数	プレーン数	重ね合わせ
文字	40	12・20・25	8	2	—	0と1
	80			1	—	×（0のみ）
グラフィック	640	200	16	8＊	4	×
			65536	2＊	—	×
		400	16	4＊	4	×
			65536	1＊	—	×

プレーン数は1画面当たりの数です。
＊）**option screen**の設定が1の場合

- ＜文字入力＞は**print**文などで文字の入力される画面を指定し、＜文字出力＞はディスプレイ上に文字が出力（表示）される画面を指定します。
  それぞれ**0**（80桁モード），**0**または**1**（40桁モード）を指定します。なお、40桁モードのときは、To_1に**0**，To_2に**1**を指定することによって2画面の重ね合わせを行うことができます。
  このとき、ドットの重なる部分は画面番号**0**の方が優先表示されます。
  ＜文字出力＞の指定
  screen ,(0)　　…　文字画面0のみ表示。
  screen ,(1)　　…　文字画面1のみ表示。
  screen ,(0, 1)　　…　文字画面0と1の重ね合わせ表示。
- ＜グラフィック入力＞はグラフィック描画命令により図形の描かれる画面を

指定し、<グラフィック出力>はディスプレイ上にグラフィックが出力(表示)される画面を指定します。
それぞれ指定できる画面は次のとおりです。

横ドット	縦ドット	表 示 色	最大画面数	画面指定
640	200	16	8	0～7
		65536	2	0，1
	400	16	4	0～3
		65536	1	0

option screenの設定が1のとき

<グラフィック出力>の指定
screen ,,,(<スクリーン番号>)

- <入力プレーン>はグラフィック入力画面のプレーンを、<出力プレーン>はグラフィック出力画面のプレーンをそれぞれ0～15の値で指定します。(16色モード時)
この値はグラフィックのパレットコードに相当し、2進数に変換したときに、1と指定されたプレーンが入出力の対象となります。
なお、65536色モード時は<入力プレーン>および<出力プレーン>の指定はできません。(全プレーンが入力プレーンおよび出力プレーンとなります。)

プレーン指定(16色モード)

指定	I G R B	指定	I G R B	指定	I G R B	指定	I G R B
0	0 0 0 0	4	0 1 0 0	8	1 0 0 0	12	1 1 0 0
1	0 0 0 1	5	0 1 0 1	9	1 0 0 1	13	1 1 0 1
2	0 0 1 0	6	0 1 1 0	10	1 0 1 0	14	1 1 1 0
3	0 0 1 1	7	0 1 1 1	11	1 0 1 1	15	1 1 1 1

I,G,R,Bの各プレーンで1となる部分が入出力するプレーンになります。

- 各指定の省略時は現在の設定値となりますが、すべての指定を省略した場合または初期設定は次のようになります。
<グラフィック表示用のメモリ領域があるとき>
screen 0，(0)，0，(0)，15，(15)　…16色モード
screen 0，(0)，0，(0)　…65536色モード
<グラフィック表示用のメモリ領域がないとき>
screen 0，(0)

サンプル

```
 10 rem --- SCREEN ---
 20 init "CRT1:":klist 0
 30 cls 3:width 40,12:pen 2
 40 screen 0,(0)
 50 color 1:print "この画面番号は0です":pause 30
 60 screen 1,(1)
 70 color 2:locate ,2:print "この画面番号は1です":pause 30
 80 screen 0,(0,1)
 90 color 7:locate ,4:print "画面が重ね合わせられました":pause 30
100 screen 1:cls
110 screen 0:cls:end
```

COLOR （カラー）

画面の表示色を設定します。

書 式

color 〔〈文字カラー〉〕〔,〈バックグラウンドカラー〉〕
〔,〈ボーダーカラー〉〕〔,〈フォアグラウンドカラー〉〕
〔,〈バックモード〉〕

省略形

col.

説 明

- <文字カラー>は文字画面に表示する色を指定します。指定以後は指定色が文字色になります。
- <バックグラウンドカラー>はグラフィック描画時のバックの色を指定します。グラフィックの消去(cls 2)やpreset文と描画機能のpresetで色指定を省略した場合、ここで指定された色で消去、または描画を行います。
- <ボーダーカラー>は文字やグラフィックのドットのない部分の色を指定します。グラフィックが16色モードのときのみ指定することができ、color=文のパレットコード0の色と同じ扱いになります。
- <フォアグラウンドカラー>はグラフィック描画時の描画色になります。
- <バックモード>を1とすると文字のバック部分が不透明となり、グラフィック表示が見えなくなります。0の指定により解除され、グラフィック表示が見えるようになります。これはグラフィックを消去せずに文字画面を見るときに利用できます。
- 各設定値を省略した場合、省略部分は現在の設定値のままになります。
- 文字およびグラフィックのinitまたはoption screen文の設定や[コントロール]+[D]キーの操作によっても初期設定が行われます。
- 設定値の範囲と初期値は次のとおりです。

表 示 色		文 字	バックグラウンド	ボーダー	フォアグラウンド
グラフィック16色	範 囲	0～7	0～15	0～15	0～15
	初期値	7	0	0	15
グラフィック65536色	範 囲	0～7	0～65535	指定不可	0～65535
	初期値	7	0	指定不可	65535

サンプル

```
 10 rem --- COLOR ---
 20 init "CRT2:640,400,16":cls 3
 30 for I=0 to 7
 40   color I:print "色、いろいろ。色、いろいろ。"
 50 next
 60 for I=1 to 7
 70   color 0,0,I
 80   pause 10
 90 next
100 color 7,0,0,0
110 end
```

- **color**命令を使用して、文字列"色、いろいろ。色、いろいろ"の表示色を8段階変化させた後、ボーダーカラーを8段階変化させます。

CLS （クリア スクリーン・clear screen）

画面表示の消去を行います。

書 式　cls〔〈画面指定〉〕

省略形　cl.

説 明

- ＜画面指定＞で指定された画面の消去を行います。
  ＜画面指定＞は１～３の値で行い機能は次のとおりです。
  1または省略　…　文字画面のみ消去します。
  2　…　グラフィック画面のみ消去します。
  3　…　文字およびグラフィック画面の両方を消去します。
- 文字画面の消去はscreen文で設定された＜文字入力＞面の、console文で設定されているスクロール範囲になります。
- グラフィック画面の消去は**screen**文で設定されている入力画面の入力プレーンで、**view**文で設定されたビューポート内の範囲になり、**color**文で設定されている＜バックグラウンドカラー＞で塗りつぶします。

例　cls 3 …… 文字とグラフィックの画面を消去します。

サンプル

```
10 rem --- CLS ---
20 init "CRT2:640,400,16":cls 3
30 print:print "0 1 2 3 4 5 6 7 8 9"
40 print:print "A B C D E F G H I J K L M N O P Q R S T U V W X Y Z"
50 line (20,20)-(200,200),9,BF
60 pen 4:circle (400,200),100,10
70 pause 20:cls 2
80 pause 20:cls 1
90 end
```

- 表示した文字とグラフィックをグラフィック、文字の順で消去します。

WIDTH （ウィドス）

文字の出力桁数や行数を設定します。

書 式

width 〔〈桁数〉〕〔,〈行数〉〕 …(1)
width "〈装置名〉",〈サイズ〉 …(2)
width ＃〈ファイル番号〉,〈サイズ〉 …(3)

省略形

wi.

説 明

- （書式１）
  - ・画面に表示する文字の＜桁数＞と＜行数＞の設定を行います。
    ＜桁数＞は横方向の文字数で**40**または**80**、＜行数＞は縦方向の文字数で**12**、**20**、**25**の指定ができます。
  - ・40桁の指定を行うと２つの文字画面を使用することができます。２つの文字画面の切り替えは**screen**文で行います。
    （実行直後は**screen 0,(0)**になります。）
  - ・＜桁数＞または＜行数＞のいずれかを省略することができ、省略値は現在の桁数または行数になります。
  - ・この命令を実行すると**console**文の設定は解除され、画面を消去（**cls 1**）します。
  - ・スムーススクロールの設定がされている場合に、桁数を40桁にするとスムーススクロールが解除されます。
- （書式２）
  - ・＜装置名＞で指定された装置に対して、１行の長さを指定します。
  - ・＜装置名＞は、**"LPT:"**と**COM1:"**が指定できます。
  - ・＜サイズ＞は１～255の範囲でバイト数（文字数）で指定します。
    なお、255を指定した場合１行の長さが無限大となります。
  - ・**"LPT:"**（プリンタ）が指定されると、プリンタの１行に印字される文字数（半角文字）が、＜サイズ＞で指定される文字数となります。
    "COM１:"（RS-232Cインターフェイス）が指定されると、RS-232Cインターフェイスにデータを送出するときに、１行の長さが＜サイズ＞で指定されたバイト数になると、復帰改行のコードを送出します。
    復帰改行のコードは、**init "COM:"**文で**"&H0D**または**&H0D＋&H0A**に設定されます。
    （**init "COM"**文参照）
- （書式３）
  - ・＜ファイル番号＞で**open**される装置に対して、１行の長さを指定しま

す。

・＜ファイル番号＞はプリンタとRS-232Cインターフェイスに対してオープンされたファイルにのみ使用可能です。

・＜サイズ＞は1～255の範囲でバイト数(文字数)で指定します。
なお、255を指定した場合1行の長さが無限大となります。

・プリンタが指定されると、プリンタの1行に印字される文字数の最大(半角文字)が＜サイズ＞で指定される文字数となります。また、RS-232Cインターフェイスが指定されると、RS-232Cインターフェイスにデータを送出するときに、1行の長さが＜サイズ＞で指定されたバイト数になると、復帰改行のコードを送出します。
復帰改行のコードは、**init "COM:"**文で**"&H0D**または**&H0D+&H0A**に設定されます。
(init "COM"文参照)

- (書式2)や(書式3)で同じ装置やファイルに対して2回以上設定が行われた場合は、最後に指定した値が設定されます。

例

```
width ,20
```
…… 桁数を変更せずに、行数を20行に設定します。

```
width "LPT:",80
```
…プリンタの1行の長さを80桁に設定します。

参 考☞ init "CRT1:" …… 文字画面の初期化。(207ページ)

サンプル

```
10 rem --- WIDTH ---
20 init "CRT1:40,12,1,0":cls
30 for C=1 to 6
40   if C=1 then
50     X=40 : Y=12
60     else if C=2 then
70     X=40 : Y=20
80     else if C=3 then
90     X=40 : Y=25
100    else if C=4 then
110    X=80 : Y=12
120    else if C=5 then
130    X=80 : Y=20
140    else if C=6 then
150    X=80 : Y=25
160  end if
170 width X,Y
180 print "(";X;",";Y;")":print
190 for I=1 to (X)
200   print "O";
210 next I:pause 20:next C
220 end
```

- 文字"O"を行数と桁数を変化させながら表示します。

CONSOLE （コンソール）

文字のスクロール範囲を指定します。

書 式 console 〔[〈開始行〉],〈行数〉〔,〔〈開始桁〉],〈桁数〉〕〕

省略形 cons.

説 明

- 文字のスクロール範囲を指定します。指定後は文字の表示や消去(cls 1)、スクロールが、その範囲内で行われます。
- カーソルの上下左右の移動は範囲内に限られていますが、locate文により設定範囲外へカーソルを移動させることができます。
  カーソルをスクロール範囲外に移動した場合、コントロールコード(**&H0～&HIF,** および **&H7F**)は画面全体に対し実行されます。
- 〈開始行〉または〈開始桁〉を省略したときはそれぞれ現在の設定値になります。
- 最下行を範囲に含めると**klist 1**または**klist 2**を実行しても、ファンクションキー表示を行いません。
- すべての指定を省略すると現在の表示モードでの最大範囲になります。ただし、漢字モード(**kmode 1**)やファンクションキー表示モード(**klist 1**または**klist 2**)のときは最下行は範囲に含まれません。
- **init "CRT1:"**によりスムーススクロールモードに設定されている場合に、この命令を実行して画面の一部をスクロール範囲にするとスムーススクロールは行われません。
- この命令による設定は **init "CRT 1 :"** や **width** 文の実行、または コントロール +D キーの操作により解除され、スクロール範囲は最大になります。

例

```
console 0,25,0,80
```

…… 文字画面が80桁、25行モードのときに最大の範囲をスクロール範囲とします。

参 考 init "CRT1:" …… 文字画面の初期化。(207ページ)
width …… 文字画面の最大桁数と最大行数を設定する。(215ページ)

サンプル

```
 10 rem --- CONSOLE ---
 20 init "crt1:80,25"
 30 C=0
 40 for I=1 to 2000
 50   print "*";
 60 next
 70 console 10,6
 80 for L=1 to 10
 90   C=C+1:if C=8 then C=1
100 color C
110   print "スクロール、スクロール！"
120   pause 5
130 next
140 pause 30
150 console:color 7:cls
160 end
```

- 画面全体に＊を表示した後、**console** 命令で文字のスクロール範囲を10行目から6行指定します。
- その後、**"スクロール、スクロール！"**という文字を表示していきます。

CONSOLE@ (コンソール アットマーク)

文字画面のディスプレイの出力範囲を設定します。

書 式 console@ 〔[〈開始行〉],〈行数〉[,[〈開始桁〉],〈桁数〉]]

省略形 cons. @

説 明

- ディスプレイに表示される文字表示の範囲を指定します。
  この命令は、文字画面の表示範囲を設定するだけなので、範囲外の部分は表示されないだけで**print**文による文字の書き込みやスクロールは**console**文で設定されたスクロール範囲で行われます。
- ＜行数＞または＜桁数＞を**0**にすると文字画面のディスプレイへの出力は行われません。
- スムーススクロールのときに、最下行を出力範囲に指定できません。
- この命令による設定は、**init "CRT1:"** や **width** 文の実行、または コントロール +D キーの操作により初期化され、文字画面全体を出力します。
- すべての指定を省略すると初期化され、全体を出力します。

例 `console@ ,0` …… 行数が0なので文字画面が表示されなくなります。

参 考 ☞ view@ …… グラフィック画面のディスプレイ出力範囲の設定。(241ページ)

サンプル

```
 10 rem --- console@ ---
 20 init "CRT1:80,25"
 30 for A=1 to 1920:print "C";:next
 40 for A=1 to 5
 50   for X=80 to 0 step -1
 60     Y=int(X*25/80)
 70     console@ 0,Y,0,X
 80   next : pause 5
 90   for X=79 to 0 step -1
100     Y=int(X*25/80)
110     console@ Y,25-Y,X,80-X
120   next : pause 5
130 next
```

COLOR@ （カラーアットマーク）

指定範囲の文字色を変更します。

書　式　color@ （X_1, Y_1）－（X_2, Y_2）, 〈色コード〉

省略形　col. @

説　明

- 文字座標（X_1, Y_1）と（X_2, Y_2）を対角とする範囲に表示している文字の色を〈色コード〉で指定した色に変更します。
- 座標は、（X_1, Y_1）で左上、（X_2, Y_2）で右下を指定しますので、$X_1 \leqq X_2$, $Y_1 \leqq Y_2$ でないとエラーになります。
- ＜色コード＞は**0～7**の範囲で指定できます。

例

```
color@ (0,0)-(10,10),6
```

参　考☞　cflash@ …… 指定範囲の文字を点滅させる。(223ページ)
crev@ …… 指定範囲の文字を反転させる。(225ページ)

サンプル

```
 10 rem --- COLOR@ ---
 20 init "crt1:80,25"
 30 for I=1 to 2000
 40   print "@";
 50 next
 60 for L=1 to 6
 70   color@ (L*5,L*2)-(79-L*5,24-L*2),L
 80 next
 90 pause 20
100 end
```

- 文字"@"を画面全体に表示した後、色コードを変えながら指定範囲に着色します。

LOCATE（ロケート）

カーソルの移動を行います。

書　式　locate〔〈X座標〉〕〔,〈Y座標〉〕〔,〈カーソルスイッチ〉〕

省略形　loc.

説　明

- 文字座標の<X座標>,<Y座標>の位置にカーソルを移動します。
  移動したのちに、print文などで文字を表示させるとその位置から表示します。
- <カーソルスイッチ>はカーソルの表示を制御し、次のようになります。
  0　…指定以後カーソルを表示しません。
  1　…カーソルを常時表示します。
  2　…直接実行命令およびinputなどのキー入力待ちのときにカーソルを表示します。
- <X座標>の省略値はコンソールエリアの左端のX座標で、<Y座標>や<カーソルスイッチ>を省略したときは現在の値のまま変化しません。
- インサートモード(文字コード&H01)のときにこの命令を実行すると、インサートモードを解除します。

例

```
locate 10,10  ……  文字座標(10,10)へカーソルを移動します。
locate ,10    ……  11行目の先頭(左端)へカーソルを移動します。
```

参　考☞　pos ……　カーソルの水平位置を求める。(228ページ)
csrlin ……　カーソルの垂直位置を求める。(227ページ)

サンプル

```
10 rem --- LOCATE ---
20 init "crt1:80,25":kmode 0
30 for I=1 to 223
40 X=int(rnd(1)*80)
50 Y=int(rnd(1)*24)
60 locate X,Y : print chr$(I+32);
70 next
80 kmode 1
90 end
```

- LOCATE命令を使用して文字座標の位置を乱数を与えて変化させ、文字を表示します。

CFLASH (キャラクタ フラッシュ・character flash)

文字を点滅させます。

書 式

cflash { [0] / 1 }

省略形

cf.

説 明

- 指定を 1 とすると、以後はprint文やlistコマンドなどで画面に表示される文字が点滅します。指定を 0 とするか省略することにより通常の表示に戻ります。
- この状態はinit "CRT1:", width文の実行やコントロール+Dキーを押すことによっても解除されます。

例

```
cflash
```
…… 文字の点滅表示状態であれば解除します。

参 考☞

cflash@ …… 指定範囲の文字を点滅させる。(223ページ)

crev …… 文字を反転させる。(224ページ)

サンプル

crevのサンプル参照

CFLASH@ (キャラクタ フラッシュ アットマーク・character flash@)

範囲を指定して文字を点滅させます。

書　式　cflash@ $(X_1, Y_1)-(X_2, Y_2)$,〈モード〉

省略形　cf. @

説　明
- 文字座標の(X_1, Y_1)と(X_2, Y_2)を対角とする範囲に表示している文字を点滅します。
- 座標は、(X_1, Y_1)で左上、(X_2, Y_2)で右下を指定しますので、$X_1 \leqq X_2, Y_1 \leqq Y_2$の範囲でないとエラーになります。
- 〈モード〉を1とすると範囲内の文字が点滅し、0とすると点滅を解除します。

例
```
cflash@ (0,0)-(10,10),1
```

参　考
cflash …… 以後の表示文字を点滅させる。(222ページ)
color@ …… 指定範囲の文字色を変更させる。(220ページ)
crev@ …… 指定範囲の文字を反転させる。(225ページ)

サンプル　crev@のサンプル参照

CREV (キャラクタ リバース・character reverse)

文字を反転させます。

書 式

crev { [0] / 1 }

省略形

cr.

説 明

- 指定を 1 とすると、以後は**print**文や**list**コマンドなどで画面に表示される文字が反転します。指定を **0** とするか省略することにより通常の表示に戻ります。
- この状態は**init "CRT1:", width**文の実行や コントロール +D キーを押すことによっても解除されます。
- 表示の反転は文字のフォント(光っている部分)が透明(ボーダーカラー)になり、バックがいままでのフォントの色になります。また、1ドット単位で色指定ができるPCGは各ドットの色が補色になります。

例

crev 1 …… 以後表示される文字を反転します。

参 考

crev@ …… 指定範囲の文字を反転する。(225ページ)

cflash …… 文字を点滅させる。(222ページ)

サンプル

```
 10 rem --- CREV CFLASH ---
 20 init "CRT1:40,12"
 30 crev 0:cflash 0:gosub 90
 40 crev 1:cflash 0:gosub 90
 50 crev 0:cflash 1:gosub 90
 60 crev 1:cflash 1:gosub 90
 70 pause 80:init "CRT1:"
 80 end
 90 for A=1 to 4
100   for C=0 to 7
110     color C:print "A";
120   next
130 next:print
140 return
```

CREV@ (キャラクタ リバース アットマーク・character reverse @)

範囲を指定して文字を反転させます。

書　式　crev@ (X_1, Y_1)－(X_2, Y_2), 〈モード〉

省略形　cr. @

説　明

- 文字座標の(X_1, Y_1)と(X_2, Y_2)を対角とする範囲に表示している文字を反転します。
- 座標は(X_1, Y_1)で左上、(X_2, Y_2)で右下を指定しますので、$X_1 \leqq X_2$, $Y_1 \leqq Y_2$でないとエラーになります。
- 〈モード〉を1とすると範囲内の文字が反転し、0とすると反転を解除します。

例

```
crev@ (0,0)-(40,10),1
```

参　考

crev …… 以後の文字を反転する。(224ページ)

color@ …… 指定範囲の文字色を変更する。(220ページ)

cflash …… 指定範囲の文字を点滅させる。(222ページ)

サンプル

```
 10 rem --- CREV@ CFLASH@ ---
 20 init "CRT1:"
 30 for A=1 to 640:print "H";:next
 40 for A=1 to 7
 50   color@ (A*10,0)-(A*10+9,8),A
 60 next
 70 cflash@ (0,2)-(79,3),1
 80 crev@ (0,4)-(79,5),1
 90 crev@ (0,6)-(79,7),1
100 cflash@ (0,6)-(79,7),1
110 pause 80:init "CRT1:"
120 end
```

CGEN (キャラクタ ジェネレータ・character generator)

キャラクタジェネレータの切り替えを行います。

書 式

cgen { 〔0〕 / 1〔,〈PCG指定〉〕 }

省略形

cg.

説 明

- 画面に表示する文字の種類を選択します。以後は、print文などで表示する文字を指定のキャラクタジェネレータにより表示します。

指　　定	選択される CG(キャラクタ ジェネレータ)
0 または省略	通常のROM CGによる表示になります。
1	PCG 1～3を組み合わせてドット単位で色を指定できるPCGになります。
1,0～3	4種のPCGのうちいずれかを選択します。

- 文字表示のときの文字コードの対応は次のようになります。

  8×8ドットのときは、各PCGに256種の定義がされており、文字コードの0～255に対応します。

  8×16ドットのときは、各PCGに128種の定義がされており、0～255のうち偶数値で指定します。これは8×8ドットのときの偶数、奇数と続く部分を使用するためです。

例

cgen 1 …… ドット単位で色指定のできるPCG表示にします。

参 考

def chr$ …… PCGフォントの定義。(230ページ)

cgpat$ …… 文字のパターンを読む。(232ページ)

CSRLIN （カーソル ライン・cursor line）

カーソルの垂直位置を求めます。

書 式 csrlin

省略形 csr.

説 明

- 現在のカーソルの垂直位置を示すシステム変数です。画面の上端を0として下に向かって数が大きくなります。
- 結果は25行モードでは 0 ～24、20行モードでは 0 ～19、12行モードでは 0 ～11になりますが、スムーススクロールモードでは最下行が使用されないため、最大値が 1 つ少なくなります。

例

```
if csrlin>22 then locate 0,0
```

…… カーソル位置が22行をこえるならば左上に移動させます。

参 考 pos …… カーソルの水平位置を求める関数。(228ページ)

locate …… カーソルを指定位置に移動する。(221ページ)

サンプル

```
10 rem --- CSRLIN ---
20 cls:for I=0 to 8
30 print "ABCDE ····· ";
40 print csrlin;"行目"
50 next
60 end
```

POS （ポジション・position）

カーソルの水平位置を求めます。

書　式　pos （〈数式〉）

説　明

- 現在のカーソルの水平位置を求める関数で、結果は画面の左端を0として右に向って数が大きくなります。
- ＜数式＞はダミーでどんな値でも良く、通常は0を指定します。
- 40桁モードのときは0～39、80桁モードのときは0～79の値になります。

参　考　csrlin …… カーソルの垂直位置を示すシステム変数。(227ページ)

locate …… カーソルを指定位置に移動する。(221ページ)

サンプル

```
10 rem --- POS  ---
20 cls
30 for L=0 to 9
40 print "*";
50 locate pos(0)-1,1 : print right$(str$(L),1);
60 locate pos(0),0
70 next
80 print : print
90 end

**********
0123456789
Ok
```

SCRN$ （スクリーン ダラー・screen$）

文字画面より連続した文字を取り出します。

書 式　scrn$(〈X座標〉,〈Y座標〉[,〈文字数〉])

説 明

- 文字画面の指定位置より、指定長で表示しているデータを求める関数です。
- 文字座標の＜X座標＞,＜Y座標＞が開始位置になり、＜文字数＞分の文字列として表します。
- ＜文字数＞は、半角文字(全角文字は半角文字の2文字分)での数で、最大**255**まで指定できます。省略したときの文字数は**1**になります。
- 取り込み範囲の最初の文字が全角文字の右半分にかかる場合や、最後の文字が全角文字の左半分にかかる場合は、2バイトのコードになりますから、返される文字長が増えます。ただし、文字長が**255**をこえる場合は**255**をこえた範囲の文字は入力できません。

例

```
A$=scrn$(0,0,40)
```

…… 画面の左上から40文字分の表示データをA$に代入します。

サンプル

```
 10 rem --- SCRN$  ---
 20 cls
 30 kmode 0
 40 for I=1 to 250
 50   print chr$(int(rnd(1)*223)+32);
 60   A$=scrn$(5,2,5)
 70 next
 80 print:print:print
 90 color@ (5,2)-(9,2),2
100 print A$
110 kmode 1
120 end
```

- **scrn$**命令を使用して、2行目5桁より5文字を取り出して表示します。

DEF CHR$ (デファイン キャラクタ ダラー・define character$)

PCGを定義します。

書　式　def chr$(〈コード〉[,〈PCG指定〉])＝〈文字列〉

説　明

- PCGにはドット数が8×8ドット、8×16ドット、16×16ドット（外字）の3種があります。表示の方法としてPCG0～3のいずれかを単独で使用するものと、PCG1～3を組み合わせてドット単位で色を指定できるものとがあります。
  この命令では＜コード＞によりドット数、＜PCG指定＞によりPCG 0 ～ 3の組み合わせが指定され、それぞれの大きさによるフォントデータを＜文字列＞により指定します。

＜コード＞	＜PCG指定＞	定義されるPCG	文字列の大きさ
00～FF(H) 8×8ドット	省　略	PCG1～3	24バイト
	0	PCG0	8バイト
	1	PCG1	8バイト
	2	PCG2	8バイト
	3	PCG3	8バイト
100～1FE(H) 8×16ドット 偶数値	省　略	PCG1～3	48バイト
	0	PCG0	16バイト
	1	PCG1	16バイト
	2	PCG2	16バイト
	3	PCG3	16バイト
EC9F～ECFC ED40～ED7E ED80～EDE2	16×16ドット外字 / 省略（指定しません）	PCG0～3	32バイト

- ＜文字列＞に与えるデータは8×8ドットと8×16ドットの**PCG**の場合、パターンを上から下に向けて並べたものです。またドット単位に色を指定する場合には**B. R. G.** の順にデータを与えます。16×16ドットの**PCG**の場合は左側のパターンを上から下に、次に右側のパターンを上から下に与えます。
- 定義した**PCG**の表示は、8×8ドットと8×16ドットの場合には**cgen**文で**PCG**表示モードに切り替えた後、行います。**print chr$(〈コード〉);**の実行により、画面の表示が8×8ドットのときは0～255のコードに対応したPCGを表示し、8×16ドットモードのときは0～255の偶数のコードに対応したPCGを表示します。（奇数で指定した場合は指定値-1の値とします。）

16×16ドットは外字となり、**kmode 1**(漢字表示モード)の状態で**print chr$(〈外字コード〉);**で指定すれば表示できます。外字はJISコード表現では**&J7821～&J787E、&J7921～797E、&J7A21～7A64**になります。

- PCG1～3の組み合わせで表示する場合、PCG1，2，3にそれぞれ**B.R.G**の色が割り当てられます。
- 8×8ドットや8×16ドットのフォントを文字コードの0～31(&H00～&H1F)に定義した場合、表示しようとするとコントロールコードが実行されます。そのため、文字コード0～31に定義したフォントを表示するときは、次のようにしてください。(**cgen**文は実行されているものとします)。

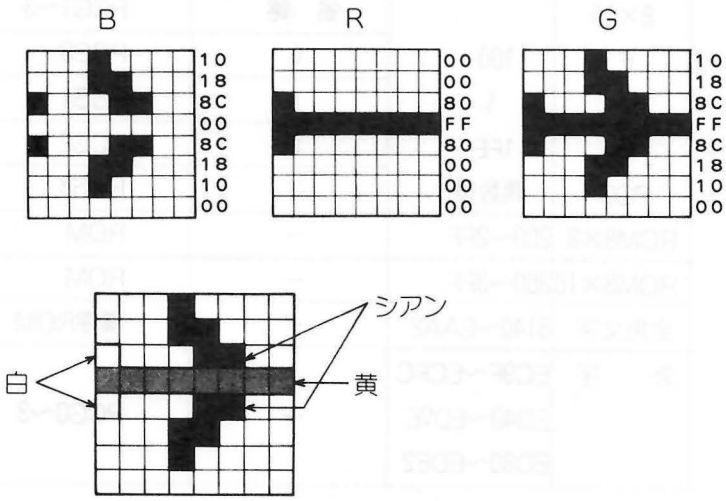
```
open "O",#1,"CRT1：CG"
print #1,chr$(<コード>);
```

参　考　**cgen** …… PCG表示の切り替え。(226ページ)

cgpat$ …… PCGフォントの読み出し関数。(232ページ)

サンプル

```
10 rem --- def chr$ ---
20 init "CRT1:40,25,0"
30 B$=hexchr$("10188C008C181000")
40 R$=hexchr$("000080FF80000000")
50 G$=hexchr$("10188CFF8C181000")
60 def chr$(64)=B$+R$+G$
70 cgen 1:print chr$(64);:cgen
```



CGPAT$ (キャラクタ ジェネレータ パターン ダラー・character generator pattern$)

文字のパターンを読み出します。

書 式 cgpat$(〈コード〉〔,〈PCG指定〉〕)

省略形 cgp.

説 明

- PCGやROM CGのフォントパターンを文字列データとして読み出す関数です。**def chr$**文と逆の働きをします。
- ＜コード＞は読み出す文字のデータで、＜PCG指定＞はPCGの読み出しを行うときに1面ずつの指定を行い、省略時はPCG1～3の連続したデータとなります。

＜コード＞指定と読み出すデータ

文字種	＜コード＞	＜PCG指定＞	読み出すCG	バイト数
8×8		省 略	PCG1～3	24
ド	00	0	PCG0	8
ッ	～	1	PCG1	8
ト	FF	2	PCG2	8
PCG		3	PCG3	8
8×16		省 略	PCG1～3	48
ド	100	0	PCG0	16
ッ	～	1	PCG1	16
ト	1FE	2	PCG2	16
PCG	偶数値	3	PCG3	16
ROM8×8	200～2FF	－	ROM	8
ROM8×16	300～3FF	－	ROM	16
全角文字	8140～EAA2	－	漢字ROM	32
外 字	EC9F～ECFC ED40～ED7E ED80～EDE2	－	PCG0～3	32

- 読み出されたデータの構成(内容)はdef chr$文の定義データと同じです。

サンプル

```
 10 rem --- CGPAT$ ---
 20 D1$=cgpat$(&H88A4)
 30 print "'愛'の文字データは、次の通りです。"
 40 kmode 0
 50 print:print ascchr$(D1$):print
 60 kmode 1
 70 end

run
'愛'の文字データは、次の通りです。

007F2211FF80A9284F440F1462010EF07C800810FE028A28E400F81020C0300E

Ok
```

- **cgpat$**により"愛"の文字パターンを読み出した後、16進データに変換し表示します。

GET% （ゲット パーセント）

文字画面の表示データを取り込みます。

書　式　get% (X1, Y1)－(X2, Y2), 〈配列変数名〉[(〈要素番号〉)]

説　明

- 文字画面に表示している(X1, Y1)－(X2, Y2)の範囲のデータを数値型配列変数に取り込みます。取り込まれるデータは文字コードだけではなく、色の情報についても行われます。
- 配列変数にはoption base文で設定されている下限から順に記憶されますが、記憶を開始する要素番号を指定することもできます。
- 読み取りは文字画面の指定範囲の左上の文字から順に右側へ行き、範囲の右端にくると次の行の左端から順に行われます。
- データを取り込む配列変数はあらかじめ必要な大きさを宣言しておきます。1文字当り4バイトのデータになりますので、必要なバイト数は横幅×縦の幅×4になります。配列変数は型により1つの配列要素当りのバイト数が異なり、整数型は2、単精度型は5、倍精度型は8で先の必要バイト数を1つ当りの要素バイト数で割れば必要な配列の要素数が求められます。

  必要な要素数＝（横幅×縦の幅×4＋要素のバイト数－1）￥要素のバイト数
- 1文字当りのデータは次のように記憶されます。

  1バイト目　文字コードまたは漢字ROMアドレス.（2バイト目のKが0の場合は文字コード、1の場合は、ROMアドレスになります。）

  MSB [文字コード] LSB　　MSB [ROMアドレス(low) | R | X] LSB

  2バイト目　　漢字属性

  MSB [K | L | ROMアドレス(high)] LSB

  3バイト目　　色属性

  MSB [- | - | G | R | B] LSB

  4バイト目　　文字属性

  MSB [Bl | Rv | S1 | S0 | P | C | - | -] LSB

  C：1 …… PCGによる表示
  0 …… ROM CGでの表示
- この命令で読み取ったデータはput%文により画面の任意の位置に表示ができます。

参　考　put% …… get%で読み取ったデータの表示。(235ページ)

PUT% （プット パーセント）

GET%文で読み取ったデータを表示します。

書　式　put% （X_1, Y_1）－（X_2, Y_2）, 〈配列変数名〉〔(〈要素番号〉)〕

説　明

- GET%文で配列に読み取った文字表示のデータを（X_1, Y_1）－（X_2, Y_2）の範囲で画面に表示します。
  ＜配列変数名＞だけの指定ではその配列の下限（option base文で0または1に設定されている）から順に取り出し、指定範囲の左上から右へ順に進み右端にきた場合は次の行の左端に進んで右側へと表示していきます。
- ＜配列変数名＞のあとに＜要素番号＞を指定した場合は、指定された要素番号より読み取りを始めます。

参　考　get%　……　文字画面のデータを取り込む。（234ページ）

サンプル

```
 10 rem --- get% put% ---
 20 init "CRT1:40,25":klist 0
 30 dim X%(55):C=1
 40 for A=65 to 90
 50   crev A mod 2
 60   color C:print chr$(A);
 70   C=C+1:if C=8 then C=1
 80 next:color 7:cflash:crev
 90 get% (0,0)-(25,0),X%
100 put% (0,3)-(25,3),X%
110 put% (0,5)-(12,6),X%
120 put% (0,8)-(6,11),X%
130 put% (30,0)-(30,19),X%
140 put% (34,0)-(34,19),X%(12)
150 locate 0,20
```

COLOR＝（カラー イコール）

カラーパレットの設定を行います。

書 式

color=(⟨パレットコード⟩,⟨カラーコード⟩)[,(⟨パレットコード⟩,⟨カラーコード⟩)]… (1)

color=(⟨カラーコード⟩,⟨G輝度⟩,⟨R輝度⟩,⟨B輝度⟩)[,(⟨カラーコード⟩,⟨G輝度⟩,⟨R輝度⟩,⟨B輝度⟩)]… (2)

(1)…16×16色パレット用　(2)…15×4096色パレット用

省略形

col.=

説 明

- グラフィック画面が16色モードのとき、0～15の<パレットコード>に対して画面上に出力される色を設定します。
- 65536色モードのときはカラーパレットの設定は行えません。
- 書式(1)は**0～15**の<パレットコード>に対して**0～15**の<カラーコード>で指定します。
- 書式(2)はカラーパレットボード(**MZ-1M10**)(別売)を取り付けた場合に有効で、**G.R.B.**の3原色に対してそれぞれ0～15の輝度レベルを設定して4096種の色を表現することができます。
  その色を16色×16色パレット回路を通って出てきた信号(<カラーコード>)に対して指定します。
  <カラーコード>は0～15までありますが、<カラーコード>0に対する指定は無効になります。
- 書式(1)ではグラフィック画面の表示色指定で、文字表示色は変化しませんが、(2)の書式では画面の表示色に対して指定が行われますので文字の表示色も同様の変化をします。この場合、文字の色コード**1～7**はカラーコードの**9～15**と同じになります。
- 書式(1)を実行すると、16色×16色パレット回路からディスプレイへ直接信号を出力(図1)します。書式(2)を実行すると、16色×16色パレット回路から、15色×4096色パレット回路を通してディスプレイへ信号を出力(図2)します

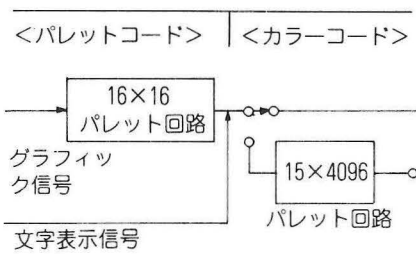


図1

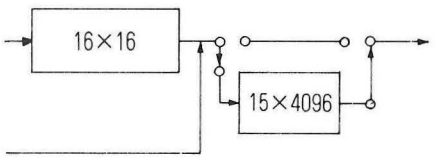


図2

したがって書式(1)を実行すると15×4096色の指定は無効となり、書式(2)を

実行すると15×4096色の指定が有効となります。このような回路構成となっていますので15×4096色パレットを使用されるときは16×16色パレットは初期状態のままで指定を行わない方が、<パレットコード>と<カラーコード>が1対1で対応するため表示色がわかりやすくなります。

サンプル

```
 10 rem --- COLOR= ---
 20 init "CRT2:640,200,16":cls 3
 30 Y=199:C=1
 40 repeat
 50   YY=Y-10*(Y+50)/200
 60   for A=Y to YY step -1
 70     line (0,A)-(639,A),C
 80   next
 90   C=C+1:if C=8 then C=4
100 Y=YY
110 until Y<40
120 for I=-300 to 959 step 40
130   line (320,-100)-(I,199),0
140 next
150 for A=2 to 7:P(A)=A:next
160 for A=1 to 1001
170   for B=1 to 7
180     P(B)=P(B) mod 7+1
190   next
200   color=(1,P(1)),(2,P(2)),(3,P(3)),(4,P(4)),(5,P(5)),(6,P(6)),(7,P(7))
210 pause 1
220 next
```

PRIORITY （プライオリティ）

表示の優先順位を設定します。

書 式 priority 〔〈優先順位指定〉〕

省略形 prio.

説 明

- グラフィック画面が16色モードのときに、グラフィック画面と文字画面の重ね合わせ表示の優先順位を指定します。
  65536色モードのときは、優先順位を指定することはできません。（常に文字画面が優先されます。）
  グラフィック画面の各パレットコードに対して設定することができ、**0～65535**の整数で指定します。
  指定した整数を2進数に変換したとき、ビットが1のときは、そのビットに対応するパレットコードのグラフィック表示が文字表示より優先して表示され、ビットが0のときは文字表示がそのビットに対応するパレットコードのグラフィック表示より優先して表示されます。

2進数	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
パレットコード	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
表示色（パレット初期状態）	明白	明黄	明シアン	明緑	明マゼンタ	明赤	明青	灰	白	黄	シアン	緑	マゼンタ	赤	青	黒

各ビットが1の場合、指定色は文字より優先し、0の場合は文字の方が優先します。

- 優先順位の指定を省略すると初期状態になります。（**priority 0**）
- コントロール＋D キーの操作またはinit "CRT2:"の実行により初期化されます。
- 文字画面にchr$(4)の文字コードを表示することによっても初期化されます。

サンプル

```
 10 rem --- PRIORITY ---
 20 init "crt2:640,200,16":cls 3
 30 for I=0 to 15
 40   line (I*40,0)-(I*40+39,100),I,BF
 50 next
 60 for I=0 to 23
 70   print string$(8,"SHARP");
 80 next
 90 PR=1
100 for I=0 to 15
110   priority PR:PR=PR*2:pause 10
120 next
130 cls 3:end
```

VIEW （ビュー）

グラフィックのビューポートを設定します。

書　式　view〔(X₁, Y₁)－(X₂, Y₂)〔,〈領域色〉〕〔,〈境界色〉〕〕

省略形　vi.

説　明

- グラフィック画面上の実際にグラフィック描画が行なわれる範囲（ビューポート）を指定します。画面の左上を(0, 0)とする絶対スクリーン座標の(X₁, Y₁)－(X₂, Y₂)を対角とする範囲がその指定となり、そのときに＜領域色＞を指定すると、その範囲を指定された色で塗りつぶし、＜境界色＞を指定すると、指定色でビューポートの枠取りをします。
- ＜領域色＞や＜境界色＞を省略した場合はそれぞれの機能は行われません。
- すべての指定を省略すると初期化され、画面全体がビューポートになります。
- 座標は範囲の左上と右下を指定しますので、X₁≦X₂、Y₁≦Y₂でない場合はエラーになります。
- この指定により以後のグラフィック描画はビューポート内で行われ、範囲外へのグラフィック描画は行われません。また、ビューポートの左上がスクリーン座標(0, 0)になります。
- コントロール＋D キーの操作またはinit "CRT 2:"の実行により初期化されます。

例

```
view (0,0)-(199,159)
```

サンプル

```
  10 rem --- VIEW ---
  20 init "CRT2:640,400,16":cls 3
  30 option angle degrees
  40 color ,,,7:gosub 1000
  50 view (0,0)-(320,200)
  60 color ,,,9:gosub 1000
  70 view (320,0)-(639,399)
  80 color ,,,10:gosub 1000
  90 view (160,160)-(480,240)
 100 color ,,,12:gosub 1000
 990 view:end
1000 poly (320,200),180,,185,0,13320
1010 return
```

VIEW@ （ビュー アットマーク）

ディスプレイ上に表示するグラフィック画面の範囲を指定します。

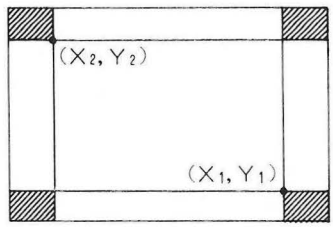
書　式　view@〔$(X_1, Y_1)-(X_2, Y_2)$〕

省略形　vi. @

説　明

- 画面の左上を(0, 0)とする絶対スクリーン座標の$(X_1, Y_1)-(X_2, Y_2)$を対角とする範囲をディスプレイ上に表示します。範囲以外の部分はディスプレイ上に表示されませんが、図形の描画などは行うことができます。
- 座標指定を省略すると初期化され、グラフィック画面全体を表示します。
- 横方向の指定のX_1とX_2は、8ドット単位で表示を行います。指定値をXとした場合、(X¥8)＊8から8ドット分が指定されたことになります。(¥は整数除算の記号です。)
- 横方向または縦方向で開始位置より終了位置が1つ少ない場合は、画面を出力しません。(例 **view@ (0, 10)－(0, 9)**)
- (X_1, Y_1)が表示開始位置、(X_2, Y_2)が表示終了位置の指定になりますので、$X_1 > X_2$であれば範囲の上下、$Y_1 > Y_2$であれば範囲の左右の部分を表示することになり、範囲内が表示されなくなります。

  例)　$X_1 > X_2, Y_1 > Y_2$であれば次のようになります。



▨部分が表示され、それ以外の部分は表示しません。

- コントロール＋D キーの操作またはinit "CRT 2:"の実行により初期化され、画面全体を表示します。

例

```
view@ (0,0)-(100,100)
```

サンプル

```
 10 rem --- VIEW @ ---
 20 init "CRT2:640,200,16"
 30 cls 3
 40 for X=0 to 639 step 5
 50   line (X,0)-(639-X,199),(X mod 15)+1
 60 next
 70 for Y=0 to 199 step 3
 80   line (0,Y)-(639,199-Y),(Y mod 15)+1
 90 next
100 for I=1 to 5
110   view@ (100,50)-(539,149):pause 5
120   view@ (0,0)-(639,199):pause 5
130 next
140 cls 3
150 end
```

- グラフィック表示後、**view@**命令で表示範囲を変更します。

VIEW（ビュー）

現在のビューポートの設定位置を求めます。

書　式　view(〈機能〉)

省略形　vi.

説　明

- ビューポートの現在の設定位置を求めます。
  結果は絶対スクリーン座標値になります。
- ＜機能＞は0〜3の値で設定します。
  0…ビューポートの左上端のＸ座標
  1…ビューポートの左上端のＹ座標
  2…ビューポートの右下端のＸ座標
  3…ビューポートの右下端のＹ座標

例

```
print view(0)
```

…… ビューポートの左上端のＸ座標を表示します。

WINDOW （ウインドゥ）

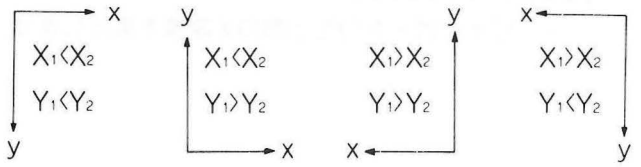
ビューポートに論理座標を設定します。

書　式　window〔$(X_1, Y_1)-(X_2, Y_2)$〕

省略形　win.

説　明

- view文で設定されたビューポートに$(X_1, Y_1)-(X_2, Y_2)$の論理座標（ワールド座標）を割り当てます。ビューポートの左上が論理座標(X_1, Y_1)、右下が論理座標(X_2, Y_2)になります。
- 座標指定を省略すると現在のビューポートに対して絶対スクリーン座標と同じワールド座標を設定します。
- スクリーン座標は左上を起点としてX軸は右方向、Y軸は下方向に座標値が増加しますが、ワールド座標は、座標の増加方向を自由に指定することができます。これはX_1とX_2、Y_1とY_2の大小により決まり、次のようになります。



- コントロール＋Dキーの操作またはinit "CRT 2:"の実行により初期化され、画面全体に設定されたビューポートに対してスクリーン座標と同じワールド座標が割り当てられます。

例

```
window (0,199)-(319,0)
```

…… スクリーン座標に対してY軸を反転させます。

サンプル

```
  10 rem --- WINDOW ---
  20 init "CRT2:640,400,16":cls 3
  30 option angle degrees
  40 color ,,,12:gosub 1000
  50 window (0,399)-(319,0)
  60 color ,,,10:gosub 1000
  70 window (0,0)-(319,199)
  80 color ,,,9:gosub 1000
  90 window (0,0)-(639,199)
 100 color ,,,14:gosub 1000
 990 window:end
1000 poly (160,100),90,,144,90,810
1010 return
```

WINDOW （ウインドゥ）

現在のウインドゥの設定位置を求めます。

書　式　window(〈機能〉)

省略形　win.

説　明

- ウインドゥの現在の設定位置を求めます。
  結果はワールド座標値になります。
- ＜機能＞は0～3の値で指定します。
  0…ウインドゥの左上端のＸ座標
  1…ウインドゥの左上端のＹ座標
  2…ウインドゥの右下端のＸ座標
  3…ウインドゥの右下端のＹ座標

例

```
print window(1)
```

…… ウインドゥの左上端のＹ座標を表示します。

PEN （ペン）

ドットおよび線描画のペン形状を指定します。

書 式

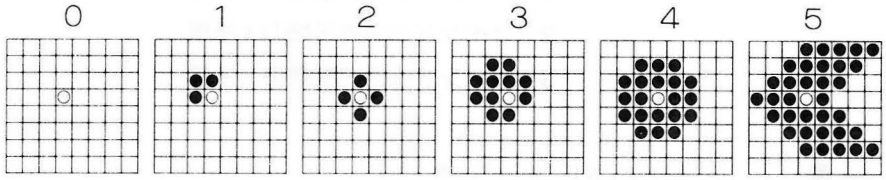
pen〔〈ペン番号〉〕　　…ペンの選択
pen 〈ペン番号〉,〈ペン形状〉　　…ペン形状の指定

説 明

- グラフィック描画時のペン形状を指定します。ペン形状が有効なのは次の命令です。

  pset　preset　line　circle　connect　poly

- ペンの形状は6種類あり、＜ペン番号＞は0～5の数で指定します。BASICの起動時は次の形状が設定されています。



  ○のドットが中心位置になります。

- ペン番号0以外は形状の指定ができます。＜ペン形状＞は8バイトの文字列データで指定し、1文字目がいちばん上の横8ドットで左端がMSB、右端がLSBになります。ペン番号5の形状指定をする場合は、hexchr$("1F3E7CF87C3E1F00")になります。また、中心となる位置は左側から4ドット目、上から4ドット目の場所になります。
- ペンの選択で＜ペン番号＞を省略した場合は0が指定されたものとします。

例

```
pen 5:circle (160,100),90,1
```

…… ペン番号5番で円を描きます。

サンプル

```
 10 rem --- PEN ---
 20 init "crt2:640,200,16"
 30 for I=0 to 5
 40   pen I
 50   gosub *DRAW
 60 next
 70 end
 80 *DRAW
 90 cls 3
100 circle (320,100),100,2
110 line (0,0)-(250,10),3,B
120 pset (50,150),5
130 pause 10
140 return
```

- **pen**番号を変化させながら円と四角形を表示します。同時にペンの形状も表示します。

PSET （ポイント セット・point set）

ドットのセットを行います。

書 式　pset（X,Y）〔,〈色〉〕〔,〈機能〉〕

〈機能〉…**pset, preset, and, or, xor（16色モード）**

pset, preset, xor, add, sub（65536色モード）

省略形　ps.

説 明

- ワールド座標上の指定された位置に対して指定された＜色＞で点をセットします。
- (X,Y)がセットする位置でstep（X,Y)による相対座標での指定が可能です。
- 実行後グラフィックポインタはドットをセットした座標になります。
- ＜色＞の指定は16色モードではパレットコード、65536色モードではカラーコードになり、省略すると**color**文による＜フォアグラウンドカラー＞になります。
- ＜機能＞を省略するとpsetとして描画します。
- pen文によりセットするドット形状の選択、設定が可能です。

例

```
pset (100,100)
```

…… ワールド座標(100,100)を＜フォアグラウンドカラー＞でセットします。

参 考　preset …… ドットをリセットする。(249ページ)

サンプル

```
 10 rem --- PSET ---
 20 init "crt2:640,200,16":cls 3
 30 for I=1 to 300
 40   P=(I mod 6)
 50   C=(I mod 15)+1
 60   X=int(rnd(1)*640)
 70   Y=int(rnd(1)*200)
 80   pen P
 90   pset (X,Y),C
100 next
110 end
```

PRESET （ポイント リセット・point reset）

ドットのリセットを行います。

書 式　preset (X,Y)〔,〈色〉〕〔,〈機能〉〕

〈機能〉…pset, preset, and, or, xor (16色モード)

pset, preset, xor, add, sub (65536色モード)

省略形　pre.

説 明

- <色>を指定すると指定色で点がセットされます。この命令は<色>の省略値が<バックグラウンドカラー>であること以外はpset文と同じです。
- <色>の指定は16色モードではパレットコード、65536色モードではカラーコードになります。
  <色>の指定を省略するとcolor文で設定されている<バックグラウンドカラー>でワールド座標上の指定位置をセットします。
- 座標指定はstep (X,Y)による相対座標での指定が可能です。
- この命令の実行後のグラフィックポインタは、リセットした座標になります。
- <機能>を省略した場合はpsetになります。
- pen文によりリセットするドット形状の選択、設定ができます。

例

```
preset step (10,10)
```

…… 現在のグラフィックポインタよりの相対座標(10,10)の位置の点を<バックグラウンドカラー>でセットします。

参 考　pset …… ドットをセットする。(248ページ)

LINE （ライン）

指定座標間を直線で結びます。

書　式

line〔(X1, Y1)〕－(X2, Y2)〔,〈機能〉〕〔,〈色1〉〕

〔,{ [B]〔,〈ラインスタイル〉〕 / BF〔,{ 〈色2〉 / 〈タイルパターン〉 }〕 }〕

〈機能〉…pset, preset, and, or, xor（16色モード）
pset, preset, xor, add, sub（65536色モード）

説　明

- ワールド座標上の(X1, Y1)と(X2, Y2)の間を直線で結びます。
  開始座標の(X1, Y1)を省略するとグラフィックポインタが開始座標になります。実行後のグラフィックポインタは(X2, Y2)になります。
- 開始座標の(X1, Y1)と終了座標の(X2, Y2)には、step（X, Y)による相対座標での指定が可能です。開始座標に使用した場合は、この命令の実行時点のグラフィックポインタから、終了座標に使用した場合は、開始座標からの相対座標指定になります。
- ＜機能＞を省略した場合は**pset**になります。また、＜機能＞の省略はあとに指定が続く場合でも、＜機能＞の直前のカンマ(,)を含めて省略します。
  Bを指定すると＜色1＞で(X1, Y1)－(X2, Y2)を対角とする四角形の箱**(box)**を書きます。
  BFを指定すると四角形を描き、さらに箱内を塗りつぶします。**(box fill)**
- ＜色１＞の指定は16色モードではパレットコード、65536色モードではカラーコードになります。
  ＜色１＞を省略した場合は**color**文で指定されている＜フォアグラウンドカラー＞になります。
- ＜ラインスタイル＞はドットの並びを０～65536の値で指定します。指定された値を２進数に変換し、１となるビットを描画します。
  省略すると、65535が指定されます。
- ＜色２＞を指定すると、＜色２＞で指定された色で塗りつぶします。指定方法は＜色１＞と同じです。
  16色モードの場合、＜タイルパターン＞を指定すると、パターンで塗りつぶします。
  パターンの指定方法は**paint**文と同じです。（**paint**文参照）
  65536色モードの場合には＜タイルパターン＞は指定できません。
  なお、＜色２＞および＜タイルパターン＞を省略した場合は、箱の外枠を描

いた色で塗りつぶしを行います。

- pen文により線を引くためのドット形状の選択、設定ができます。

例

```
line (100,50)-(200,100),2,BF
```

参　考☞　connect　……　連続した直線を引く。(252ページ)

サンプル

```
 10 rem --- LINE ---
 20 init "crt2:640,200,16":cls 3
 30 for I=1 to 50
 40   P=(I mod 6)
 50   C=(I mod 15)+1
 60   X=int(rnd(1)*640)
 70   XX=int(rnd(1)*640)
 80   Y=int(rnd(1)*200)
 90   YY=int(rnd(1)*200)
100   pen P
110   line (X,Y)-(XX,YY),C
120 next
130 end
```

CONNECT （コネクト）

直線を続けて引きます。

書　式　connect〔(X1, Y1)〕－(X2, Y2)…〔－(Xn, Yn)〕〔,〈色〉〕〔,〈機能〉〕
〔,〈ラインスタイル〉〕
〈機能〉…pset, preset, and, or, xor（16色モード）
pset, preset, xor, add, sub（65536色モード）

省略形　conn.

説　明

- ワールド座標の指定座標間を順番に直線で結んでいきます。
- 開始座標の(X1, Y1)を省略するとグラフィックポインタからになります。実行後のグラフィックポインタは直線の最終座標(Xn, Yn)になります。
- 各座標はstep (X, Y)による相対座標での指定が可能です。開始座標は、この命令の実行時点のグラフィックポインタから、ほかの座標は、その直前の座標からの相対指定になります。
- ＜色＞の指定は16色モードではパレットコード、65536色モードではカラーコードになります。
- ＜色＞を省略するとcolor文で指定されている＜フォアグラウンドカラー＞になります。
- ＜ラインスタイル＞は2バイトの整数型データで16ドットを指定します。
- pen文により線を引くためのドット形状の選択、設定ができます。

例

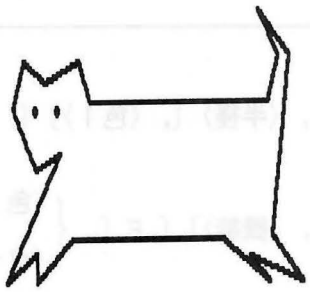
```
connect (100,50)-(150,100)-(50,100)-(100,50),3
```

参　考☞　line …… 指定座標間に直線を引く。(250ページ)

サンプル

```
  10 rem --- CONNECT ---
  20 dim X(21),Y(21)
  30 cls 3:init "CRT2:640,200,16"
  40 for L=1 to 21
  50   read X(L),Y(L)
  60   Y(L)=Y(L)/2
  70 next
  80 pen 1
  90 connect (X(1),Y(1))-(X(2),Y(2))-(X(3),Y(3))-(X(4),Y(4))-(X(5),Y(5))-(X(6),
Y(6)),5
 100 connect (X(6),Y(6))-(X(7),Y(7))-(X(8),Y(8))-(X(9),Y(9))-(X(10),Y(10))-(X(1
1),Y(11)),5
 110 connect (X(11),Y(11))-(X(12),Y(12))-(X(13),Y(13))-(X(14),Y(14))-(X(15),Y(1
5))-(X(16),Y(16)),5
 120 connect (X(16),Y(16))-(X(17),Y(17))-(X(18),Y(18))-(X(19),Y(19))-(X(20),Y(2
0))-(X(21),Y(21)),5
 130 pen 3
 140 pset (86,58),2:pset (102,58),2
 150 end
```

```
160 data 78,82,70,114,86,158,104,146,64,238,86,218,86,238,112,208,220,208,254,
238,240,220,278,238,256,206,270,78,248,48,262,78,254,110,126,110,118,82,94,96,78
,82
```



CIRCLE （サークル）

円を描きます。

書　式

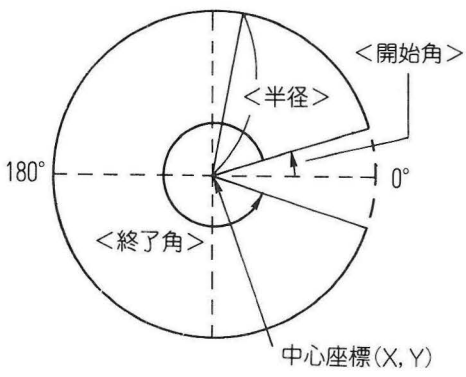
circle 〔{(Wx, Wy) / STEP(X, Y)}〕, 〈半径〉〔, 〈色１〉〕〔, 〈開始角〉〕
〔, 〈終了角〉〕〔, 〈比率〉〕〔, 〈機能〉〕〔, F〔, {〈色２〉 / 〈タイルパターン〉}〕
〔, 〈傾き〉〕

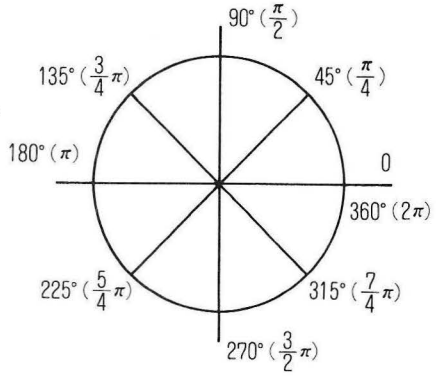
省略形

ci.

説　明

- ワールド座標(X, Y)を中心として＜半径＞で指定された大きさの円を描きます。
- 中心座標の指定はstep（X, Y)による相対座標での指定が可能です。
- 実行後のグラフィックポインタは中心座標になります。
- ＜色１＞は円弧を描く色を指定します。
  16色モードのときは０～15のパレットコードで、65536色モードのときは０～65535のカラーコードで指定します。
  省略すると、color文で指定されている＜フォアグラウンドカラー＞となります。
- ＜開始角＞と＜終了角＞は中心座標の右側から始まり左回りで角度が増えます。角度の単位はラジアンと度が使用でき、その選択はoption angle文によって行います。＜開始角＞の省略時は0、＜終了角＞の省略時は360度またはラジアンで2πになります。





角度指定（　）内はラジアン

- ＜開始角＞や＜終了角＞の値を負にすると中心まで直線を引き扇形にすることができます。
- ＜開始角＞は＜終了角＞より小さい値としてください。＜開始角＞が＜終了

角＞より大きい値になる場合は、＜終了角＞の指定を360度(ラジアンの場合は2π)＋＜終了角＞としてください。

- ＜比率＞を指定することにより、だ円を描くことができます。
  640×400ドットモードのときは**1**,640×200ドットモードのときは**0.5**を指定すると真円になり､それ以外は次のようになります。
  (比率＝垂直半径/水平半径)
  ＜比率＞**<**＜真円比率＞…横長のだ円になります。
  ＜比率＞**>**＜真円比率＞…縦長のだ円になります。

- ＜機能＞は、円弧を描く機能を、**F**指定を行うと塗りつぶしの機能を指定するもので、**pset,preset,and,or,xor**(16色モード)、**pset,preset,add,sub,xor**(65536色モード)のいずれかを指定します。
  省略時には、**pset**が指定されます。
  なお、＜機能＞に限り、省略時にはカンマ(,)を含めて省略してください。
- **F**指定を行うと、円の内部を塗りつぶします。＜色2＞を指定すると、指定された色で塗りつぶします。色の指定方法は＜色1＞と同じです。
  16色モードのとき、＜タイルパターン＞を指定すると、パターンで塗りつぶします。パターンの指定方法は**paint**文と同じです。(**paint**文参照)
  65536色モードのときはタイルパターンを指定することはできません。
  なお、＜色2＞および＜タイルパターン＞を省略した場合は、円弧を描いた色で塗りつぶしを行います。
- ＜傾き＞は＜開始角＞から＜終了角＞までの範囲に描画すべき円弧に対して、描画する直前に回転処理を行う機能です。
  たとえば＜開始角＞を30度、＜終了角＞を40度、＜傾き＞を20度とすると、実際には50度から60度の範囲に円弧が描かれます。
- pen文により線を引くためのドット形状の選択、設定ができます。

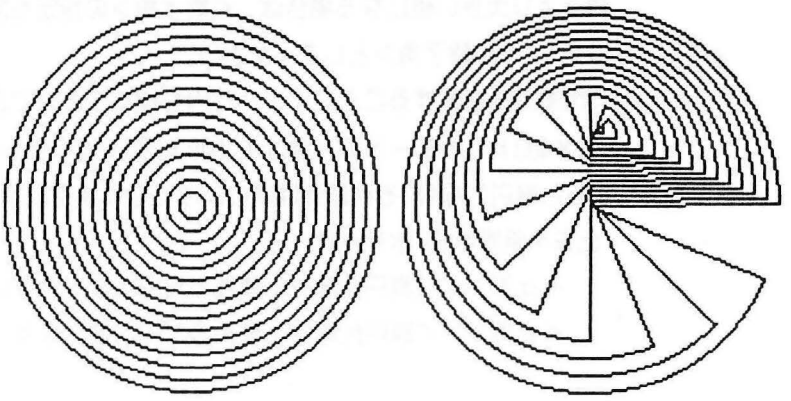
例

```
circle (150,100),50,4
```

…… 中心座標(150,100)で半径50の円を描きます。

サンプル

```
10 rem --- CIRCLE ---
20 init "CRT2:640,400,16":cls 2
30 option angle degrees
40 for R=1 to 15
50   circle (160,200),R*10,R
60   circle (480,140+R*4),R*10,R,-1,-22.5*R
70 next R
```



POLY （ポリゴン・polygon）

多角形を描きます。

書　式

poly （X, Y）, 〈半径〉〔, 〈色〉〕〔, 〈ステップ角〉〕〔, 〈初期角〉〕
〔, 〈終了角〉〕〔, 〈機能〉〕〔, 〈ラインスタイル〉〕
〈機能〉…pset, preset, and, or, xor（16色モード）
pset, preset, add, sub, xor（65536色モード）

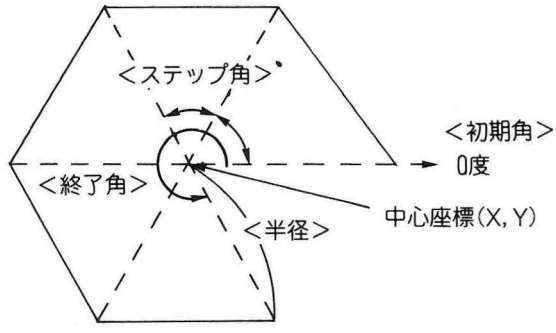
省略形

pol.

説　明

- ワールド座標の（X, Y）を中心として、＜初期角＞から＜ステップ角＞ごとの頂点を＜終了角＞になるまで線を結んで行きます。＜半径＞が中心座標より頂点までの長さになります。
- 中心座標の指定はstep （X, Y）による相対座標での指定が可能です。
- 実行後のグラフィックポインタは中心座標になります。
- ＜色＞の指定は16色モードではパレットコード、65536色モードではカラーコードになります。
- ＜色＞を省略するとcolor文で指定されている＜フォアグラウンドカラー＞になります。
- ＜ステップ角＞、＜初期角＞、＜終了角＞の指定はラジアンと度が使用でき、その選択はoption angle文により行ないます。それぞれの省略時の値は＜ステップ角＞が1度＜初期角＞は0度、＜終了角＞は360度になります。
  一般的に正n角形を描くには、次のように指定します。

＜ステップ角＞＝360/n（度）
＜終了角＞＝＜初期角＞＋360



- ＜機能＞を省略した場合はpsetになります。
- ＜ラインスタイル＞はline文と同様に2バイトの整数形で指定し、16ドットの指定を行います。
- pen文により線を引くためのドット形状の選択、設定ができます。

例

```
poly (160,100),70,,144,90,810
```

…… 星形を描きます。

参 考☞ **circle** …… 円を描く。(254ページ)

サンプル

```
 10 rem --- POLY  ---
 20 init "CRT2:640,200,16"
 30 option angle degrees
 40 cls 3:dim X(4),Y(4),C(4),S(4)
 50 for I=0 to 4
 60   read X(I),Y(I),C(I),S(I)
 70   poly (X(I),Y(I)),40,C(I),S(I),90,810
 80 next
 90 paint (0,0),4,not 0
100 end
110 data 100,100,8,144
120 data 200,100,9,72
130 data 300,100,10,120
140 data 400,100,11,45
150 data 500,100,12,60
```

PAINT （ペイント）

境界内を塗りつぶします。

書　式

paint（X, Y）〔, { 〈色〉 / 〈タイルパターン〉 }〕{ 〔,〈境界色〉〕… / 〔, not 〈境界色〉〕 }

省略形

pa.

説　明

- ワールド座標の(X, Y)の位置から＜境界色＞で囲まれた範囲内を指定された＜色＞または＜タイルパターン＞で塗りつぶします。ただし、ビューポートの外側は塗りつぶしません。
  なお、＜タイルパターン＞による塗りつぶしは16色モードの場合にのみ行えます。65536色モードでは＜タイルパターン＞の指定はできません。
- 塗り始める位置の指定はstep（X, Y)による相対座標での指定が可能です。
- この命令の実行後のグラフィックポインタは、塗り始めの位置になります。
- ＜色＞は塗りつぶしに使用する色で、省略した場合はcolor文で指定されている＜フォアグラウンドカラー＞になります。
- ＜色＞の＜境界色＞の指定は16色モードではパレットコード、65536色モードではカラーコードになります。
- ＜境界色＞は複数個の指定ができ、省略された場合は塗りつぶす＜色＞が境界になります。not ＜境界色＞は、＜境界色＞以外のすべての色を境界色とする指定になります。
- 塗り始める位置がすでに境界色となっている場合は、塗りつぶしを行いません。16色モードの場合は、＜タイルパターン＞での塗りつぶしが行えます。
- ＜タイルパターン＞は単色の塗りつぶしではなく、横8ドットで縦は任意のドット数の模様で指定できます。指定は文字列データで、縦をnドットとするとn×4バイトの長さになります。画面にはこのデータの1バイトずつを2進数に対応させて各プレーンに次のように展開します。

B	R	G	I

例）**PT$＝hexchr$（"AACCF000AACCF0FF"）**

16色モードでPT$をパターンデータとすると次のような8×2ドットのパターンの繰り返しになります。

サンプル

```
 10 rem --- PAINT ---
 20 init "CRT2:640,400,16":cls 2
 30 window (-160,-100)-(159,99)
 40 option angle degrees
 50 X1=0:Y1=0
 60 for R=0 to 99 step .5
 70   X2=sin(12*R)*R:Y2=cos(12*R)*R
 80   line (X1,Y1)-(X2,Y2)
 90   line (-X1,-Y1)-(-X2,-Y2)
100   line (Y1,-X1)-(Y2,-X2)
110   line (-Y1,X1)-(-Y2,X2)
120 X1=X2:Y1=Y2
130 next R
140 circle (0,0),99
150 paint (0,-95),9,15
160 paint (95,0),10,15
170 paint (0,95),12,15
180 paint (-95,0),14,15
```

PATTERN （パターン）

グラフィック画面にパターンを描きます。

書 式

pattern (X,Y),〈段数〉,〈パターン〉[,〈フォント色〉][,〈バック色〉][,〈機能〉]

〈機能〉…pset, preset, and, or, xor, scr, fpset(16色モード)

pset, preset, add, sub, xor, fpset(65536色モード)

省略形

pat.

説 明

- ビューポート内のスクリーン座標(X,Y)のところから、指定された＜段数＞にしたがってパターンを描きます。
- 開始位置の指定はstep (X,Y)による相対座標での指定が可能です。
- ＜段数＞は±255までの指定ができ、正の場合は上方向、負の場合は下方向に重ねます。
- ＜パターン＞は文字列型データで、1バイトずつ文字コードを2進数に展開して画面に表示します。文字列の長さが段数をこえる場合は、最初の位置より右に8ドット分ずれた位置より続けます。
- この命令の実行後のグラフィックポインタは、最後のパターンデータの次の位置になります。
- ＜フォント色＞はドットの部分、＜バック色＞はそれ以外の部分の色を指定します。指定を省略した場合はcolor文の指定による＜フォアグラウンドカラー＞と＜バックグラウンドカラー＞になります。
- ＜色＞の指定は16色モードではパレットコード、65536色モードではカラーコードになります。
- ＜機能＞は次のようになり、省略時はpsetになります。

pset	フォント色、バック色にしたがって描きます。
preset	フォント色、バック色をそれぞれ補色で描きます。
and, or, xor	すでに表示されているデータと論理演算をして描きます。
scr	Single Color Replaceの略で、＜フォント色＞に該当するプレーンのみ描かれ、該当しないプレーンには作用しません。バック部分は該当プレーンではリセットされます。＜バック色＞を指定しても何も行いません。
fpset	フォント部分についてのみ＜フォント色＞によるpsetが行われ、バックの部分は変化しません。したがって、＜バック色＞を指定しても何も行いません。
add, sub	すでに表示されているデータの色の階調を増減させます。

サンプル

```
   10 rem --- PATTERN ---
   20 option angle degrees
   30 init "crt2:640,200,16":klist 0:cls 3
   40 RED$=hexchr$("00000000FFFFFFFF00000000FFFFFFFF00000000FFFFFFFF000
00000FFFFFFFF")
   50 BLUE$=hexchr$("FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF")
   60 for Y=0 to 199 step 16
   70   for X=0 to 639 step 16
   80     pattern (X,Y),16,RED$,10
   90 next X,Y
  100 for Y=0 to 96 step 16
  110   for X=0 to 256 step 16
  120     pattern (X,Y),16,BLUE$,1
  130 next X,Y
  140 for Y=16 to 80 step 16
  150   for X=16 to 224 step 16
  160     poly (X,Y),4,15,144,90,810
  170 next X,Y
  180 end
```

SYMBOL （シンボル）

任意の大きさで文字パターンを描きます。

書　式

symbol (X, Y), 〈文字列〉〔, 〈横倍率〉〕〔, 〈縦倍率〉〕
〔, 〈色〉〕〔, 〈角度〉〕〔, 〈機能〉〕〔, 〈フォント〉〕〔, 〈スムージング〉〕
〈機能〉… pset, preset, and, or, xor (16色モード)
pset, preset, add, sub, xor (65536色モード)

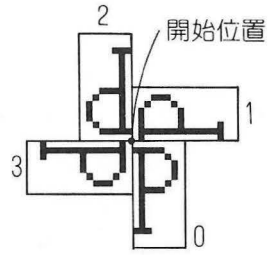
省略形

sym.

説　明

- ビューポート内のスクリーン座標(X, Y)の位置から、＜文字列＞による文字フォントを縦横の倍率を設定して描きます。また、4種類の角度を指定して文字の向きを変えることができます。
- 開始座標は文字の左上部分になります。また、開始座標の指定はstep（X, Y）による相対座標での指定が可能です。
- ＜横倍率＞と＜縦倍率＞は1～255までの整数で指定でき、省略した場合はそれぞれ1になります。
- ＜色＞は描画色で、省略した場合はcolor文で指定されている＜フォアグラウンドカラー＞になります。
- ＜色＞の指定は16色モードではパレットコード、65536色モードではカラーコードになります。
- ＜角度＞は90度ごとの4方向で0～3の整数で指定します。

  0 … 通常状態で省略時はこの値になります。
  1 … 左回転で 90度。
  2 … 左回転で180度。
  3 … 左回転で270度。



- ＜機能＞を省略した場合はpsetになります。なお、この命令で描かれるのはフォント部分のみで、フォント以外の部分は変化しません。
  ＜フォント＞は半角文字のドット数の選択です。0を指定すると8×8ドット、1を指定すると8×16ドットになります。また、省略時は、現在半角文字表示に用いられているフォントになります。
  この命令の実行後のグラフィックポインタは、続いて同倍率、同角度で文字パターンを描く場合の次の位置になります。
- ＜スムージング＞は、文字を拡大したときにおきる斜めの線の凸凹を補正する機能です。1と指定すると補正を行います。0指定および省略時は補正を

行いません。

なお、補正が行われるのは、＜横倍率＞、＜縦倍率＞共に4倍までです。

例

```
symbol (16,16),"ABC",2,2  ……  縦、横2倍で "ABC"を描きます。
```

サンプル

```
 10 rem --- SYMBOL ---
 20 init "crt2:640,200,16":cls 3
 30 for I=0 to 3
 40   symbol (100,50),"SHARP",1,1,10,I
 50   symbol (200,100),"MZ",2,2,12,I
 60   symbol (400,150),"Series",1,3,14,I
 70 next
 80 end
```

COLOR REPLACE （カラー リプレース）

グラフィック画面の表示色を変更します。

書 式 color replace (X1, Y1)－(X2, Y2), (〈旧色〉, 〈新色〉)〔, (〈旧色〉, 〈新色〉)〕…

省略形 col. repl.

説 明

- グラフィック画面上の表示に対して、指定範囲内の特定色をほかの色に変更します。この場合の色変更は、color=文のようにディスプレイ出力の信号を切り替えるのではなく、グラフィック**V-RAM**に書かれているデータを変更します。
- (X1, Y1)－(X2, Y2)のエリア指定はビューポートの左上を(0, 0)とするスクリーン座標に対して行い、エリア内の＜旧色＞、＜新色＞の組み合わせで変更していきます。
- 開始座標の(X1, Y1)と終了座標の(X2, Y2)には、step (X, Y)による相対座標での指定が可能です。開始座標に使用した場合は、この命令の実行時点のグラフィックポインタから、終了座標に使用した場合は、開始座標からの相対座標になります。
- この命令の実行後のグラフィックポインタは、(X2, Y2)の位置になります。
- ＜旧色＞と＜新色＞の指定は16色モードではパレットコード、65536色モードではカラーコードになります。
- この命令では**screen**文で指定されている入力プレーンには関係なくすべてのプレーンに対して描画が行われます。

例

```
color replace (0,0)-(100,100),(1,2),(2,1)
```

…… 指定範囲の1と2の色を入れ替えます。

サンプル

```
10 rem --- COLOR REPLACE ---
20 init "CRT2:640,400,16":cls 3:pen 4
30 for R=1 to 180 step 6
40   C=C+1:if C=16 then C=1
50   circle (320,200),R,C
60 next
70 for A=2 to 15:B=16-A
80   color replace (0,0)-(319,399),(1,A),(A,1)
90   color replace (319,0)-(639,399),(15,B),(B,15)
100 next
```

MOVE （ムーブ）

グラフィック表示の移動を行います。

書　式

move （X_1, Y_1), Lx, Ly, (X_2, Y_2)〔,〈プレーン〉〕〔,〈移動処理〉〕〔,〈機能〉〕
〔,〈透明色〉〕

〈機能〉… pset, preset, and, or, xor, scr(16色モード)
pset, preset, add, sub, xor(65536色モード)

省略形

mov.

説　明

- グラフィック画面に描かれている図形を移動します。スクリーン座標の(X_1, Y_1)を左上の起点として、横幅がLx、縦の長さがLyの範囲の図形をスクリーン座標(X_2, Y_2)の位置へ移動します。
- 開始座標の(X_1, Y_1)と終了座標の(X_2, Y_2)には、step （X, Y)による相対座標での指定が可能です。開始座標に使用した場合は、この命令の実行時点のグラフィックポインタから、終了座標に使用した場合は、開始座標からの相対座標になります。
- この命令の実行後のグラフィックポインタは、(X_2, Y_2)になります。
- ＜プレーン＞は＜機能＞でscrを指定した場合に有効で移動する色の指定となりますが、実際には指定色を表示するために必要なプレーンのデータが移動します。たとえば、16色モードでコードが6の黄色を指定した場合、黄色を構成するコード2の赤とコード4の緑の色も移動します。＜プレーン＞の指定を省略した場合はcolor文で指定されている＜フォアグラウンドカラー＞になります。
- ＜移動処理＞を0とすると移動後に以前のデータを消去し、1とすると以前のデータは消去されません。省略時は0が指定されたものとみなします。
- ＜機能＞の指定により移動先に描かれているデータとの論理演算などが行われます。scr(Single Color Replace)は＜プレーン＞で指定されるプレーンについてのデータを書き換え、そのほかのプレーンには影響しない機能です。
- ＜透明色＞は移動するデータの中で透明にしたい色があれば、そのパレット番号を指定します。

サンプル

```
10 rem --- MOVE ---
20 init "CRT2:640,400,16":cls 3:pen 4
30 line (0,0)-(79,79),8,BF
40 option angle degrees
50 poly (40,40),36,,144,90,810
60 for A=0 to 159
```

```
 70    move (A,A),80,80,(A+1,A+1)
 80  next
 90  for C=0 to 3
100    for A=160 to 559
110      move (A,160),80,80,(A+1,160),2^C,,scr
120    next
130  next
```

ROLL （ロール）

グラフィック画面のスクロールを行います。

書　式　roll〔〈縦移動量〉〕〔,〈横移動量〉〕〔,{N/Y}〕

省略形　ro.

説　明

- グラフィック画面に描画されているデータ全体を上下左右にスクロール(移動)させます。
- <縦移動量>および<横移動量>はドット数で表し、値が正の場合は上(左)に、値が負の場合は下(右)にスクロールします。このとき、スクロールして表示できなくなった部分のグラフィックデータは失われます。
  なお、指定できる値の範囲は次のとおりです。
  640×200ドットモード時:(縦)－199～199
  (横)－639～639
  640×400ドットモード時:(縦)－399～399
  (横)－639～639
- <縦移動量>および<横移動量>を省略すると、それぞれのスクロールは行いません。
  ただし、<縦移動量>および<横移動量>を共に省略することはできません。
- **N, Y**指定で、スクロールによって新たに表示された領域の色を指定することができます。
  Nを指定するとパレットコード0(初期設定は黒)に設定されている色で、**Y**を指定すると**color**文で設定したバックグラウンドカラーで表示されます。
  省略すると、**N**が設定されます。

例　`roll -16` …… 下方向へ16ドット分のスクロールを行います。

参　考☞　**roll@** …… グラフィック画面のディスプレイ出力のスクロール。(270ページ)

サンプル

```
10 rem ---ROLL ---
20 init "CRT2:640,200,16":cls 3
30 for I=1 to 20
40   color ,I mod 16
50   roll 5,-16,Y
60 next
70 end
```

ROLL@ （ロール アットマーク）

グラフィックのディスプレイ表示のスクロールを行います。

書　式　roll@〔〈移動量〉〕

省略形　ro. @

説　明

- グラフィック画面をディスプレイに出力する位置をスクロール(移動)します。
- <移動量>は、現在の表示より移動を行う量をドット数で指定します。値が正の場合は上方向、負の場合は下方向に位置を移します。
- 表示のスクロールはディスプレイへの出力信号を制御しますので、グラフィックの座標は移動にともない上下にずれます。
- 指定を省略すると、初期化します。
- 移動により、はみ出す部分は、移動方向と反対側に表示します。

例　roll@ 100 …… グラフィック表示を上方向に100ドット分ずらします。

参　考☞　roll …… 描画されているグラフィックデータのスクロール。(268ページ)

サンプル

```
10 rem --- ROLL@ ---
20 init "CRT2:640,400,16":cls 3
30 option angle degrees
40 poly (320,100),80,12,144,90,810
50 circle (320,300),80,10
60 for A=1 to 400:roll@ 1:next
70 for A=1 to 400:roll@ -1:next
80 for A=1 to 400:roll@ 200:next
90 for A=1 to 400:roll@ 201:next
```

MAP （マップ）

スクリーン座標とワールド座標の相互変換を行います。

書 式　map(〈数式〉,〈機能〉)

説 明

- ＜数式＞をスクリーン座標または、ワールド座標として相互の変換を行います。変換内容は＜機能＞に0～3の値を指定します。

＜機能＞の値		変換内容
0	……	＜数式＞をワールド座標のX座標として、スクリーン座標のX座標に変換した値を返します。
1	……	＜数式＞をワールド座標のY座標として、スクリーン座標のY座標に変換した値を返します。
2	……	＜数式＞をスクリーン座標のX座標として、ワールド座標のX座標に変換した値を返します。
3	……	＜数式＞をスクリーン座標のY座標として、ワールド座標のY座標に変換した値を返します。

POINT （ポイント）

グラフィックポインタの位置を設定します。

書 式

point { (X, Y) | step (X, Y) | screen (X, Y) | screen step (X, Y) }

省略形

poi.

説 明

- グラフィック命令の描画位置を記憶しているグラフィックポインタの値を指定します。グラフィックポインタはグラフィック描画命令において、step (X, Y)による相対座標指定の起点となるもので、描画命令実行後はその命令での指定座標に変わります。
- グラフィックポインタの位置指定はワールド座標、またはビューポート内の左上を(0, 0)とするスクリーン座標で行います。screenを指定するとスクリーン座標、指定しない場合はワールド座標による設定になります。また、step (X, Y)は現在のグラフィックポインタに対する相対座標指定になります。

サンプル

```
 10 rem --- POINT (ｽﾃｰﾄﾒﾝﾄ)  ---
 20 init "CRT2:640,200,16":cls 2
 30 on stop gosub *STOP:stop on
 40 input "始点を入力してください (x,y) ":P1,P2
 50 cls 3:pen 2:point (P1,P2)
 60 input "X座標の移動距離=";X
 70 input "Y座標の移動距離=";Y
 80 line -step (X,Y),(abs(X) mod 15)+1
 90 cls 1:goto 60
100 end
110 *STOP
120 stop off:return 100
```

- point命令により任意の位置にグラフィックポインタを設定した後、入力されたX座標、Y座標の移動距離分だけ軌跡を描きます。

POINT（ポイント）

グラフィックの指定座標の表示色やグラフィックポインタの位置を求めます。

書　式

point(X,Y)　…(1)
point(〈機能〉)　…(2)

省略形

poi.

説　明

- 書式(1)はビューポート内の左上を(0,0)とするスクリーン座標の(X,Y)の位置に表示されている色を求める関数です。
- 16色モードの場合、指定された値がビューポートの範囲外の場合は-1の値を返します。
- 求められる値は16色モードのときはパレットコード、65536色モードのときはカラーコードになります。
- 書式(2)は現在のグラフィックポインタの位置を求めます。＜機能＞は0～3まであり、それぞれ次の値を返します。

  0 … 現在のX座標をワールド座標で求める。
  1 … 現在のY座標をワールド座標で求める。
  2 … 現在のX座標をスクリーン座標で求める。
  3 … 現在のY座標をスクリーン座標で求める。

サンプル

```
10 rem --- POINT (関数)  ---
20 init "CRT2:640,200,16"
30 cls 3:randomize time/100
40 for L=1 to 150
50   X=int(rnd(1)*639+.5)
60   Y=int(rnd(1)*149+.5)+50
70   C=int(rnd(1)*15)+1
80   line (X,Y)-step (50,25),C,BF
90 next
100 print "最後に描かれた座標は  (":point(0);",";point(1);")" : print
110 input "X座標を入れて下さい=",XS
120 input "Y座標を入れて下さい=",YS
130 print "入力した座標の点の色は、カラーコード":point(XS,YS)
140 end
```

- 任意の点を指定することにより、**point**関数でその点の色を調べて表示します。

GET@ （ゲット アットマーク）

グラフィック画面データを配列に取り込みます。

書 式　get@ (X₁, Y₁)－(X₂, Y₂), 〈配列変数名〉[(〈要素番号〉)][, 〈データ構成〉]

説 明

- グラフィック画面の座標(X₁, Y₁)－(X₂, Y₂)を対角線とする領域のデータを数値型配列変数に取り込みます。
  座標はスクリーン座標で指定します。**step(X, Y)**による相対座標の指定も可能です。
  (スクリーン座標はビューポートの左上端の座標を(0, 0)とする座標です。)
- データは16色モードの場合、範囲の左上のドットから右方向に8ドットずつ各プレーンを**B,R,G,I**の順に読み取っていき、右端まで読むと次のラインの左端に行き8ドットずつ右へと読んでいきます。水平ドット数が8で割り切れない場合は範囲をこえたデータもそのまま読み取られます。
- データを読み取る配列を＜配列変数名＞で指定し、＜要素番号＞は読み取ったデータの記憶を開始する要素番号を指定します。＜要素番号＞の指定を省略した場合はoption base文で指定されている下限より記憶されます。
- ＜データ構成＞は、画面データの構成方法を**0**または**1**で指定します。(16色モード時) **0**が指定されると、グラフィック画面を8ドット単位に句切ったとき、左側のビットをデータの左側ビットに対応(左側MSB右側LSB)させて取り込みます。**1**が指定されると、グラフィック画面を8ドット単位に句切ったとき、左側のビットのデータの右側のビットに対応(左側LSB右側MSB)させて取り込みます。(**1**指定はBASIC-M25と互換性のあるモードです。)
- ＜データ構成＞が省略されると、**0**が設定されます。(65536色モード時は**0**指定のみ可能)
- 取り込みの対象となるデータは、グラフィック入力が行われるグラフィック画面のすべてのプレーンにあるデータです。
- 読み取りデータの記憶される配列は、データのバイト数に応じた宣言をしておく必要があります。
  必要なバイト数は水平方向のドット数を8で割り、余りがある場合は1を加え、ライン数とプレーン数(**n**)を掛け合わせ、**w**を加えた数になります。この必要バイト数を配列要素1つ当りのバイト数で割り、余りが出た場合に1を加えた数が必要要素数になります。要素1つ当たりのバイト数(**m**)は整数型は2、単精度型は5、倍精度型は8になります。
  また、次の式で求めることができます。

必要要素数＝(((abs(X_2−X_1)＋8)￥8)＊(abs(Y_2−Y_1)＋1)＊n＋w＋(m−1))￥m

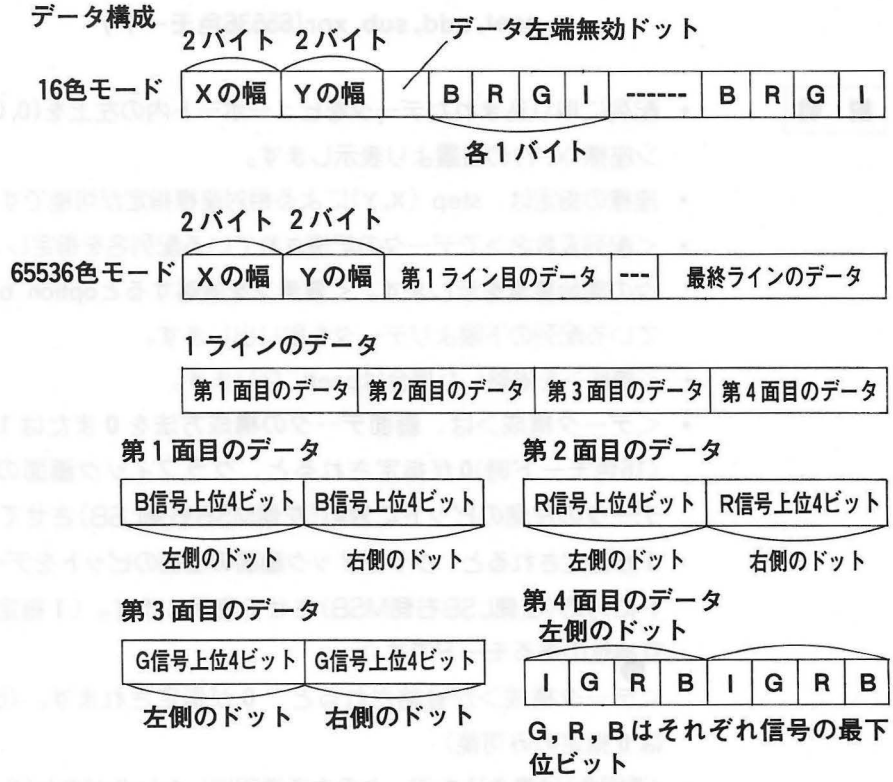
m…要素ひとつ当たりのバイト数

n…プレーン数(16色時：4，65536色時：16)

w…16色時：5，65536色時：4を指定します。

- この命令で読み取ったデータはput@文によりビューポート内の任意の位置へ表示することができます。
- 1つの配列には256キロバイト以上のデータを取り込むことはできません。

データ構成



参　考 put@　…get@文で読み取ったデータの表示。(276ページ)

サンプル　put@のサンプル参照

PUT@ （プット アットマーク）

グラフィックパターンの表示を行います。

書 式

put@ (X, Y), 〈配列変数名〉〔(〈要素〉)〕〔, 〈機能〉〕〔, 〈データ構成〉〕
〔, 〈透明色〉〕

put@ (X, Y), kanji(〈漢字コード〉)〔, 〈機能〉〕〔, 〈フォント色〉〕
〔, 〈バック色〉〕

〈機能〉…pset, preset, and, or, xor(16色モード)
pset, add, sub, xor(65536色モード)

説 明

- 配列に取り込まれたデータをビューポート内の左上を(0, 0)とするスクリーン座標(X, Y)の位置より表示します。
- 座標の指定は、step (X, Y)による相対座標指定が可能です。
- ＜配列変数名＞でデータの記憶されている配列名を指定し、＜要素＞がデータの開始要素を示します。＜要素＞を省略するとoption base文で指定されている配列の下限よりデータを取り出します。
- ＜機能＞を省略した場合はpsetになります。
- ＜データ構成＞は、画面データの構成方法を **0** または **1** で指定します。(16色モード時)**0**が指定されると、グラフィック画面の左側のビットをデータの左側のビットに対応(左側MSB右側LSB)させて表示します。
  1 が指定されると、グラフィック画面の左側のビットをデータの右側のビットに対応(左側LSB右側MSB)させて表示します。(**1** 指定はBASIC-M25と互換性のあるモードです。)
  ＜データ構成＞が省略されると、**0** が指定されます。(65536色モード時は **0** 指定のみ可能)
- 〈透明色〉は書き込むデータの中で透明にしたい色があれば、そのコードを指定します。
- ＜漢字コード＞はJIS漢字コードで表した整数型データで指定します。＜フォント色＞はドット部分の色、＜バック色＞はそれ以外の部分の色を指定します。＜フォント色＞や＜バック色＞を省略した場合は、それぞれ**color**文で指定されている＜フォアグラウンドカラー＞と＜バックグラウンドカラー＞になります。座標(X,Y)は文字の左上の位置になります。(この機能は漢字のコードにJIS漢字コードを使用しています。内部と同じシフトJISコードでお使いになる場合は**symbol文**を使用してください。)
- 16色モードのとき、データの表示は**screen**文で指定されている＜グラフィック入力＞の＜入力プレーン＞に行われます。65536色モードのときは

必ず全プレーンに表示が行われます。

- **get@**によって配列に取り込まれたときのグラフィック画面モードと**put@**によって表示するときの画面モードが異なっている場合には正常に描画できない場合があります。

参　考☞　get@　……　グラフィックデータを配列に読み取る。(274ページ)

サンプル

```
10 rem --- GET@ PUT@ ---
20 init "CRT2:640,200,16"
30 line (0,0)-(39,39),pset ,7,BF,1
40 dim A((5*40*4+6)/2)
50 get@ (0,0)-(39,39),A
60 X=rnd(1)*320
70 Y=rnd(2)*100
80 put@ (X,Y),A
90 goto 60
```

GSAVE （グラフィック セーブ・graphic save）

グラフィック表示をファイルに書き込みます。

書 式　gsave 〈ファイル指定〉[,(X_1, Y_1)－(X_2, Y_2)][,〈データ構成〉]

省略形　gs.

説 明

- ビューポート内の左上を(0, 0)とするスクリーン座標(X_1, Y_1)－(X_2, Y_2)の範囲のグラフィックデータをファイルとして記録します。
- 記録範囲を省略するとビューポートの全体を記録します。
- この命令により記録される画面はscreenに文により設定されている＜グラフィック入力＞の全プレーンデータです。
- ＜ファイル指定＞は次のように行います。

  "〔＜装置名＞〕〔＜ディレクトリパス＞〕＜ファイル名＞＜拡張子＞"

  ＜装置名＞…**A:, B, C:～J:, K:, L:, M:**

  または**FDn:, HDm:, MEM:**
- ＜装置名＞と＜ディレクトリパス＞の指定を省略すると、**chdir**コマンドで設定されているカレントディレクトリ内でのファイル指定になります。
- ＜拡張子＞を省略すると**DAT**が設定されます。
- ＜データ構成＞は、画面データの構成方法を **0** または **1** で指定します。

  (16色モード時) **0** が指定されると、グラフィック画面を 8 ドット単位に句切ったとき、左側のビットをデータの左側のビットに対応(左側MSB右側LSB)させて書き込みます。(**1** 指定はBASIC-M25と互換性のあるモードです。)

  1 が指定されると、グラフィック画面を 8 ドット単位に句切ったとき、左側のビットをデータの右側のビットに対応(左側LSB右側MSB)させて書き込みます。(**1** 指定はBASIC-M25と互換性のあるモードです。)

  ＜データ構成＞が省略されると、**0** が設定されます。(65536色モード時は **0** 指定のみ可能)

例

```
gsave "A:ARGO",(0,0)-(100,100)
```

…… ドライブAに(0, 0)-(100, 100)の範囲の図形を**"ARGO"**というファイル名で記録します。

参 考　gload …… gsave文で書き込んだデータの呼び出し。(280ページ)

サンプル

```
 10 rem --- GSAVE GLOAD ---
 20 init "CRT2:640,200,16"
 30 cls 3:pen 1
 40 for I=1 to 639 step 60
 50   for L=0 to 199 step 20
 60     line (0,0)-(I,L),(L mod 16)+1
 70   next
 80 next
 90 gsave "GSVDUM"
100 cls 3
110 gload "GSVDUM"
120 end
```

- グラフィック表示を**gsave**命令で”**GSVDUM**”というファイル名で登録した後、**gload**命令で読み出して表示します。

GLOAD （グラフィック ロード・graphic load）

ファイルからグラフィックデータを呼び出します。

書 式 gload 〈ファイル指定〉[,(X,Y)][, 〈データ構成〉] [, 〈透明色〉]

省略形 gl.

説 明

- gsave文で記録されたグラフィックファイルからデータを呼び出しグラフィック画面に表示します。
- 表示する左上の位置を、ビューポートの左上を(0,0)とするスクリーン座標(X,Y)で指定します。座標を省略すると書き込まれたのと同じ位置に表示します。
- <ファイル指定>は次のようになります。
  "〔<装置名>〕〔<ディレクトリパス>〕<ファイル名><拡張子>"
- <装置名>と<ディレクトリパス>の指定を省略すると、**chdir**コマンドで設定されているカレントディレクトリ内でのファイル指定になります。
- <拡張子>を省略すると**DAT**が設定されます。
- <データ構成>は、画面データの構成方法を **0** または **1** で指定します。(16色モード時) **0** が指定されると、グラフィック画面を 8 ドット単位に句切ったとき、左側のビットをデータの左側のビットに対応(左側MSB右側LSB)させて取り込みます。
  1 が指定されると、グラフィック画面を 8 ドット単位に句切ったとき、左側のビットをデータの右側のビットに対応(左側LSB右側MSB)させて取り込みます。(**1** 指定は、BASIC-M25と互換性のあるモードです。)
  <データ構成>が省略されると、**0** が設定されます。(65536色モード時は **0** 指定のみ可能)
- <透明色>は、読み込んだデータの中で透明にしたい色があれば、そのコードを指定します。
- 呼び出したデータが描かれるのは、**screen**文で設定されている<グラフィック入力>の<入力プレーン>になります。なお、65536色時は必ず全プレーンに表示が行われます。
- gsave文でファイルに書き込んだときの画面モードと、呼び出したときの画面モードが異なっている場合は、正常に描くことはできませんので注意してください。

例

```
gload "A:ARGO"
```

…… ドライブAの**"ARGO"**というファイルから図形データを呼び出して、登録時と同じ位置に描きます。

参　考 gsave …… グラフィック画面をファイルに書き込む。(278ページ)

サンプル　gsaveのサンプル参照

INIT "FDn:" (イニシャライズ フロッピーディスク・initialize Floppy disk drive)

フロッピーディスクドライブの種類を選択します。

書 式

init "〔FD〕n: { 〈ドライブタイプ〉
〈シリンダ数〉,〈ステップレート〉,〈タイミング〉,
〈モータオフ時間〉,〈メディアタイプ〉,〈セクタ数〉,
〈セクタサイズ〉 }"

説 明

- 使用するフロッピーディスクドライブに合わせて、容量や各種のタイミングの設定で行います。
  設定は、ドライブ番号ごとに<ドライブタイプ>による代表的な機種での選択、または個別の設定になります。(外部に別のフロッピーディスクドライブを接続した場合、使用するディスクドライブの種類により、記憶容量や各種コントロールタイミングが異なりますので、フロッピーディスクドライブに合わせた設定が必要になります。)
- フロッピーディスクドライブの装置名は"FD1:"～"FD4:"の4種でドライブ番号1～4に対応しています。"FD1:"は、ドライブ番号を省略して"FD:"として表すことができます。また、"FD"を省略して"1:"～"4:"のドライブ番号のみで表すこともできます。
- <ドライブタイプ>は**F1, F2, F4**でフロッピーディスクドライブの選択をします。指定を行うことにより次のように各設定が行われます。
  <ドライブタイプ>と指定時の設定値

〈ドライブタイプ〉	機　種　名	シリンダ数	ステップレート	タイミング
F 1	内蔵ドライブ	77	3 msec.	F
F 2	MZ-1F07/02	40	6 msec.	F
F 4	2 HD ミニフロッピーディスクドライブ	77	3 msec.	F

〈ドライブタイプ〉	モータオフ時間	メディアタイプ	セクタ数	セクタサイズ
F 1	2 sec.	0	8	1024
F 2	2 sec.	1	16	256
F 4	2 sec.	0	8	1024

- <シリンダ数>はフロッピーディスクの片面のトラック数で35, 40, 77, 80のいずれかを指定します。
- <ステップレート>は、ヘッドがトラック間を移動するタイミングのことです。3 msec., 6 msec., 12msec., 20msec., 30msec. の5種の設定を3

, 6 , 12, 20, 30のいずれかで指定します。

- <タイミング>は、ドライブモータが回転を始めて(MOTOR-ON)回転が安定するまでの待ち時間やヘッドが移動して読み書きを行うまでの時間(セットリングタイム)などの各種タイミングを**F**(fast)または**S**(slow)で指定します。

  F …… MOTOR-ON後の時間は500msec. セットリングタイムは15msec. に設定されます。

  S …… MOTOR-ON 後の時間は1sec. セットリングタイムは60msec. に設定されます。

  タイミングの指定を誤ると、読み書き時にエラーが発生する可能性があります。

- <モータオフ時間>はドライブの読み書きが終了してからドライブモータを停止するまでの時間で、0.1秒単位で0～25.3秒を0～253の値で指定します。(例えば10秒の場合は100と指定します。)省略すると、以前に設定されていた値になります。
- <メディアタイプ>は2HDのディスクの場合0を、2DD/2Dのディスクの場合1を指定します。
- <セクタ数>は1トラック当りのセクタ数を指定します。
- <セクタサイズ>は1セクタ当りのバイト数を指定します。
- <メディアタイプ>、<セクタ数>、<セクタサイズ>は**devi$, devo$**を実行する場合に必要となります。
- 内蔵ドライブや2HDミニフロッピーディスクドライブで、2DDのディスクに対して**devi$, devo$**を実行する場合は、個別設定のみ可能です。<ドライブタイプ>で指定することはできません。
- BASICの起動時は"FD1:"～"FD4:"のすべてのドライブは<ドライブタイプ>F1に設定してあります。

例

```
init "FD3:80,3,F,20,1,9,512"
```

……ドライブ番号3のフロッピーディスクドライブを80シリンダでステップレート3msec. タイミングをfast、モータオフ時間を2秒、メディアタイプを2DD、セクタ数を9、セクタサイズを512バイトとします。

INIT "MEM:" (イニシャライズ メモリファイル・initialize memory file)

メモリディスクやスプーラの容量を設定します。

書 式

init"MEM:〔〈メモリディスクサイズ〉〕,〈スプーラサイズ〉
〔,〈メモリディスクエリア〉〕**"**

説 明

- ＜メモリディスクサイズ＞はグラフィック**V-RAM**や増設RAMをフロッピーディスクドライブと同じように、ファイル装置として割り当てるメモリサイズをキロバイト単位で0～6144までの値で指定します。
  なお、増設RAMを使用しない場合は0～512までの値指定となります。
- ＜スプーラサイズ＞はグラフィック**V-RAM**をスプーラ(プリンタバッファ)として割り当てるメモリサイズをキロバイト単位で0～512までの値で指定します。
- それぞれの指定は0から最大値までの任意の整数で行いますが、実際には、**0**以外は**16K**バイトごとに切り上げて設定されます。また、**0**を指定するとこの機能は解除され(以前に設定されていたものは消去し)初期化します。
- それぞれで指定されたサイズのメモリは、増設RAMや**option screen**文で設定されたグラフィック**V-RAM**のメモリ用の領域を使用します。
- ＜メモリディスクサイズ＞を設定した場合は、それ以後**"M:"**または**"MEM:"**の装置名でファイル装置として使用できます。
- ＜スプーラサイズ＞を設定した場合は、init "LPT:"文により、スプーラ機能の設定をします。
- この命令で設定可能なメモリ容量は、fre関数で調べることができます。
- ＜メモリディスクエリア＞はメモリディスクをグラフィック**V-RAM**と増設RAMのいずれのメモリ領域に確保するかを指定するもので、0または1で指定します。

  0…グラフィック**V-RAM**を使用

  1…増設RAMを使用

  省略すると、0が指定されます。
- この命令で確保されるメモリディスクは、MS-DOS上で確保されるものと同じです。したがって、MS-DOS上で確保したメモリディスクは、そのままBASICで使用可能です。
  (**system**コマンドや**child**文でMS-DOS上に戻ったときも、そのまま引き継がれます。)
  ただし、この命令を実行すると内容は初期化されます。
- この命令で確保されるスプーラは、BASIC上でのみ使用可能です。
  MS-DOS上で使用することはできません。

BASIC起動時、<スプーラサイズ>は0に設定されています。

- この命令が実行されると、それまでオープンされていたファイルはクローズされます。

参　考　init "LPT:" …… プリンタスプーラの設定。(452ページ)

option screen …… グラフィックV-RAMを、メモリディスクやスプーラ用メモリに設定する。(203ページ)

サンプル

```
10 rem --- INIT "MEM:" ---
20 init "MEM:0,0"
30 option screen 4
40 if fre(4)>fre(5) then print "拡張メモリに";fre(4)/1024;"キロバイト確保します":init "MEM:"+str$(fre(4)/1024)+",0,1" else print "グラフィックRAMに":fre(5)/1024;"キロバイト確保します":init "MEM:"+str$(fre(5)/1024)+",0,0"
```

- メモリディスクを初期化し、現在メモリディスクとして利用可能なメモリのすべてをメモリディスクとして割り当てます。

INIT "COMn:" (イニシャライズ コミュニケーションポート・initialize communication port)

RS-232Cインターフェイスの各種設定をします。

書 式

init "COMn:〈ボーレイト〉,〈パリティ〉〈データビット長〉
〈ストップビット長〉〈通信制御〉〈カナ〉〈DEL受信〉〈CR送信〉
〈CR受信〉〈全角文字〉〈エンドコード〉"
nは1のみ

説 明

- RS-232Cインターフェイスの通信機能を設定します。
- RS-232Cインターフェイスの通信条件や制御信号の設定を次のように行います。〔 〕内は省略値です。設定の詳細は付録の「RS-232Cインターフェイス」を参照してください。

<ボーレイト>	75, 150, 300, 600, 1200, 2400, 4800, 9600, 19200
<パリティ>	E O 〔N〕
<データビット長>	7 〔8〕
<ストップビット長>	1 2 〔3〕 (1, 1½, 2ビットに対応します。)
<通信制御>	X R 〔N〕
<カナ>	S 〔N〕
<DEL受信>	N B 〔D〕
<CR送信>	C 〔L〕
<CR受信>	C 〔L〕
<全角文字>	J M P 〔N〕
<エンドコード>	文字コードで&H40～5Fまたは&H60～7Fに相当する1文字で指定し、受信時の&H0～&H1Fに対応します。

ボーレイトのあとにはカンマ(,)が必要で、ほかの指定はすべて1文字になります。また、それぞれの指定は省略でき、省略時はそれぞれ省略値が設定されます。指定を省略するときは、空白とします。ボーレートおよびカンマ(,)を省略することはできません。

- この命令での設定は、装置名として"COMn："を使用するコマンドやファイルオープンに対して有効になります。

例

```
init "COM1:9600,N83XNDLLND"
```

FILES （ファイルズ・files）
LFILES （エル ファイルズ・LPT files）

登録されているファイルを表示します。

書 式

{files / lfiles} "〔〈装置名〉〕〔〈ディレクトリパス〉〕〔〈ファイル名〉〕"

＜装置名＞…A:, B:, C:～J:, K:, L:, M:
または、FDn:, HDm:, MEM:
nは1～4，mは1～8

省略形

fi.　　lfi.

説 明

- filesコマンドは指定された装置に登録されているファイルを画面に表示します。lfilesコマンドはプリンタに印字します。
- 表示は次の書式で行われます。

  '＜時＞：＜分＞"＜ファイル名＞"' ＜年＞－＜日＞ ＊××××××

  ＜時＞：＜分＞…ファイルを作成した時刻を表示します。

  ＜年＞－＜月＞－＜日＞…ファイルを作成した日付を表示します。

  ＊　…ファイルが書き込み禁止（ライトプロテクト）状態のときに表示します。

  ×××××× …ファイルサイズをバイト数で表示します。ディレクトリ管理ファイルの場合は、この部分に＜DIR＞と表示されます。
- ＜装置名＞には、フロッピーディスクドライブ、ハードディスクドライブ、メモリディスクがあります。
  それぞれ論理装置名（A:, B:, C:～J:, K:, L:, M:）または物理装置名（FDn:, HDm:, MEM:）で指定します。
  ＜装置名＞を省略すると現在の装置名（カレントドライブ）が指定されます。
- ＜ディレクトリパス＞は階層ディレクトリのパス（道すじ）を指定します。省略すると、カレントディレクトリが指定されます。
- ＜ファイル名＞を省略すると、ディレクトリ内のすべてのファイルを表示します。
- ＜ファイル名＞は拡張子を付けて指定します。
  ＜ファイル名＞を指定すると、特定のファイルのみを表示することができます。
  このとき、ファイル名の指定に省略記号が使え、次の方法で特定の文字を含むファイルをすべて表示することができます。

? …… この記号の部分は1文字分の代用を意味します。

例) "A？？Z"と指定すると次のようなものが該当します。

"AAZZ", "AXYZ", "ATOZ"

＊ …… この記号は0～複数文字の代用になります。

例) "A＊"と指定すると次のようなものが該当します。

"AZ", "ABCXYZ", "A＋Z"

?, ＊記号をワイルドカードと呼びます。

- 停止キーによる表示の一時停止、シフト＋停止キーによる実行の中断が可能です。
- この命令では、システムファイルを表示することはできません。

例

```
files "A:"
```

……ドライブA:のディレクトリを表示します。

参 考 chdir …… カレントディレクトリの変更。(289ページ)

CHDIR （チェンジディレクトリ・change directory）

カレントディレクトリを変更します。

書　式

chdir "〔〈装置名〉〕〈ディレクトリパス〉"

＜装置名＞…**A:, B:, C:～J:, K:, L:, M:**

または、**FDn:, HDm:, MEM:**

nは1～4，mは1～8

省略形

ch.

説　明

- この命令により、指定された＜装置名＞のカレントディレクトリを変更します。

  カレントディレクトリとは現在参照しているディレクトリのことで、**files**コマンドや**open**文などのファイル指定を行う命令で＜ディレクトリパス＞を省略した場合に対象となるディレクトリのことです。
- ＜装置名＞には、フロッピーディスクドライブ、ハードディスクドライブ、メモリディスクがあります。

  それぞれ論理装置名（**A:, B:, C:～J:, K:, L:, M:**）または物理装置名（**FDn:, HDm:, MEM:**）で指定します。
- ＜ディレクトリパス＞ は階層ディレクトリのパス（道すじ）を指定します。

  ＜ディレクトリパス＞を指定した場合は、 その指定ディレクトリを新たなカレントディレクトリとします。＜ディレクトリパス＞の最初に¥マークを指定した場合は、ルートディレクトリからの指定になり、¥マークを省略するとカレントディレクトリより＜ディレクトリパス＞をたどり指定されたディレクトリにカレントディレクトリを変更します。
- カレントディレクトリのすぐ上のディレクトリへカレントディレクトリを変更する場合は、＜ディレクトリパス＞に**".."**（親ディレクトリ）を指定することができます。
- ＜ディレクトリパス＞を省略した場合は、現在の装置（カレントドライブ）を指定された＜装置名＞に変更します。
- ＜装置名＞や＜ディレクトリパス＞をすべて省略した場合は、現在の装置（カレントドライブ）のルートディレクトリにカレントディレクトリを変更します。

例

```
chdir "¥UTY¥COPY"
```

… カレントドライブのルートディレクトリにあるUTYというディレクトリ下のCOPYというディレクトリにカレントディレクトリを変更します。

```
chdir "UTY¥COPY"
```

… カレントドライブのカレントディレクトリにあるUTYというディレクトリ下のCOPYというディレクトリにカレントディレクトリを変更します。

```
chdir "COPY"
```

… カレントドライブのカレントディレクトリ下のCOPYというディレクトリへカレントディレクトリを変更します。

```
chdir ".."
```

… カレントディレクトリを親ディレクトリへ変更します。

```
chdir "B:UTY¥COPY"
```

… ドライブBのカレントディレクトリにあるUTYというディレクトリ下のCOPYというディレクトリにドライブBのカレントディレクトリを変更します。

参 考 pwd$ …… カレントディレクトリの階層を求めるシステム変数。(344ページ)

MKDIR （メイク ディレクトリ・make directory）

ディレクトリファイルを作成します。

書 式

mkdir "〔〈装置名〉〕［〈ディレクトリパス〉］〈ディレクトリ名〉"

＜装置名＞…A:, B:, C:～J:, K:, L:, M:

または、FDn:, HDm:, MEM:

nは1～4，mは1～8

省略形

mk.

説 明

- 新たなディレクトリを作成し、親ディレクトリにディレクトリファイルとして登録します。
- ディレクトリファイルの消去はrmdirコマンドで行います。killコマンドでは消去できません。
- ＜装置名＞には、フロッピーディスクドライブ、ハードディスクドライブ、メモリディスクがあります。
  それぞれ論理装置名（A:, B:, C:～J:, K:, L:, M:）または物理装置名（FDn:, HDm:, MEM:）で指定します。
  ＜装置名＞を省略すると現在の装置（カレントドライブ）の指定になります。
- ＜ディレクトリパス＞はディレクトリファイルを登録する親ディレクトリを指定します。
  ＜ディレクトリパス＞の最初に￥マークがある場合はルートディレクトリからパス（道すじ）をたどり、それ以外の場合は、カレントディレクトリよりパスをたどります。
- ＜ディレクトリパス＞の指定がない場合は、chdirコマンドで設定されている、カレントディレクトリの指定になります。
- ＜ディレクトリ名＞は＜装置名＞や＜ディレクトリパス＞によって指定されたディレクトリ内に作成されるディレクトリファイルのファイル名を指定します。
- 新たなディレクトリ内には"."と".."の2つのディレクトリ管理ファイルが作成されます。".."は親ディレクトリを管理するファイルで、カレントディレクトリ内に、このファイルがある場合、chdir ".."とすると親ディレクトリへカレントディレクトリを移すことができます。"."は自身のディレクトリを管理するファイルです。

例

```
mkdir "ABC"
```

… カレントディレクトリ下にABCというディレクトリを作成します。

```
mkdir "¥ABC"
```

… ルートディレクトリ下にABCというディレクトリを作成します。

```
mkdir "¥ABC¥ABC1"
```

… ルートディレクトリにあるABCというディレクトリ下にABC１というディレクトリを作成します。

参　考☞

rmdir …… ディレクトリファイルの消去。(293ページ)

chdir …… カレントディレクトリの変更。(289ページ)

RMDIR （リムーブ ディレクトリ・remove directory）

ディレクトリファイルを削除します。

書　式　rmdir "〔〈装置名〉〕〔〈ディレクトリパス〉〕〈ディレクトリ名〉"

＜装置名＞…**A:, B:, C:～J:, K:, L:, M:**

または、**FDn:, HDm:, MEM:**

nは 1 ～ 4 , mは 1 ～ 8

省略形　rm.

説　明

- ディレクトリファイルの削除を行います。
- ＜装置名＞を省略すると、現在の装置（カレントドライブ）の指定になります。

  それぞれ論理装置名（**A:, B:, C:～J:, K:, L:, M:**）または物理装置名（**FDn:, HDm:, MEM:**）で指定します。
- ＜装置名＞を省略すると、現在の装置（カレンドライブ）の指定になります。
- ＜ディレクトリパス＞はディレクトリファイルが登録されている親ディレクトリを指定します。

  ＜ディレクトリパス＞の最初に¥マークがある場合はルートディレクトリからパス（道すじ）をたどり、それ以外の場合は、カレントディレクトリよりパスをたどります。

  ＜ディレクトリパス＞の指定がない場合は、**chdir**コマンドで設定されている、カレントディレクトリの指定になります。
- 削除するディレクトリ内に"**.** "と"**..**"以外のファイルが残っていると削除できません。あらかじめ削除するディレクトリ内のファイルを消去しておく必要があります。

参　考　mkdir …… ディレクトリファイルの作成。(291ページ)

RUN〈ファイル〉（ラン）

指定ファイルを読み出し実行します。

書　式

run "〔〈装置名〉〕〈ファイル名〉"〔,R〕

＜装置名＞…**A:, B:, C:～J:, K:, L:, M:**

または、**FDn:, HDm:, MEM:, COM 1 :**

nは 1 ～ 4 、mは 1 ～ 8

省略形

r.

説　明

- 指定されたプログラムをメモリ上に読み出し、実行します。（**load**および**run**）
- **BAS**形式または**アスキー**形式（**DAT**）のBASICプログラムファイルを指定した場合は、指定ファイルを見つけたあとに現在のプログラムを消去（**new**）して、指定ファイルを読み出し実行（**run**）します。したがって、指定プログラムが見つからない場合はエラーになり、現在のプログラムは消去されません。ただし、＜装置名＞に**COM1:**が指定されている場合は、現在のプログラムを消去します。
- アスキー形式のファイルを指定した場合、内容がBASICプログラムでないときはエラーとなります。
- ＜装置名＞には、フロッピーディスクドライブ、ハードディスクドライブ、メモリディスク、RS-232Cインターフェイスがあります。
  それぞれ論理装置名（**A:, B:, C:～J:, K:, L:, M:**）または物理装置名（**FDn:, HDm:, MEM:, COM1:**）で指定します。
- ファイルの指定で＜装置名＞を省略した場合は、現在の装置（カレントドライブ）のカレントディレクトリから指定ファイル名をさがします。階層ディレクトリでファイルを指定する場合は＜装置名＞と＜ファイル名＞の間にディレクトリパスを指定します。
- ＜装置名＞が**COM1:**の場合は、アスキー形式のファイルとみなし、＜ファの指定を無視します。
  なお、読み出しは **init "COMn:"**文で指定されたエンドコードにより終了し、実行を開始します。

- ＜ファイル名＞で拡張子を省略した場合、**BAS**形式のファイルとみなします。
- **R**を指定すると、現在**open**されているファイルをそのままにしてプログラムをロード後、実行を開始します。
  Rを指定しない場合は**open**されているファイルを**close**します。

例

```
run "日本語wp"
```

…… カレントドライブのカレントディレクトリ下の"日本語wp"というプログラムを読み出して実行します。

```
run "COM1:"
```

…RS-232Cインターフェイスからプログラムを読み出して実行します。

参 考☞

load …… プログラムの呼び出し。(296ページ)

run(コマンド) …… プログラムの実行。(135ページ)

bload …… 機械語プログラムの読み込み。(301ページ)

LOAD （ロード）

プログラムファイルをメモリに読み出します。

書 式

load "〔〈装置名〉〕〈ファイル名〉"〔,R〕

＜装置名＞…A:, B:, C:～J:, K:, L:, M:

またはFDn:, HDm:, MEM:, COM1:, KB:

nは1～4、mは1～8

省略形

lo.

説 明

- 指定装置より、BASICプログラムを読み出します。
- ＜装置名＞には、フロッピーディスクドライブ、ハードディスクドライブ、メモリディスク、RS-232Cインターフェイス、キーボードがあります。
  それぞれ論理装置名(A:, B:, C:～J:, K:, L:, M:)または物理装置名(FDn:, HDm:, MEM:, COM1:, KB:)で指定します。
  ＜装置名＞を省略した場合は、現在の装置(カレントドライブ)のカレントディレクトリから指定ファイル名をさがします。
  階層ディレクトリでファイルを指定する場合は＜装置名＞と＜ファイル名＞の間にディレクトリパスを指定します。
- ＜装置名＞としてKB:を指定すると、↵キーを2回続けて押すことによりこの命令は終了します。(init"KB:"参照)
  また、＜装置名＞としてCOM1:を指定すると、アスキー形式のファイル入力とみなし、＜ファイル名＞の指定を無視します。この場合、init"COM:"文で指定されたエンドコードによりこの命令は終了します。(init"COM:"文参照)
- ＜ファイル名＞は拡張子を付けて指定します。拡張子を省略すると.BASが設定されます。
  ＜ファイル名＞は"COM1:"と"KB:"以外の装置では省略できません。
- 指定されたファイルを見つけたあとにメモリ上のプログラムを消去(new)して指定ファイルを読み込みます。したがって、指定ファイルが見つからなかった場合は、メモリ上のプログラムは消去されません。
- アスキー形式のファイルも読み込むことができますが、内容がBASICプログラムになっていない場合は途中でエラーを出し、実行を中止します。
- Rを指定すると、プログラム読み出し後ただちに実行を開始します。
  指定しない場合は、プログラム読み出し後ダイレクトモードになります。

例

```
load "A:AUTO-RUN.BAS"
```

…… ドライブAの"AUTO-RUN.BAS"というファイル名の付いたプログラムを読み込みます。

参考 save …… プログラムの記録。(299ページ)

bsave …… 機械語プログラムの記録。(303ページ)

LOAD＊（ロード アスタリスク）

アスキーファイルをメモリに読み出します。

書　式　load＊"〔〈装置名〉〕〈ファイル名〉"

省略名　lo. ＊

説　明

- 指定装置より、**list＊**で作成したアスキー形式ファイルを読み出します。
- ＜装置名＞には、フロッピーディスクドライブ、ハードディスクドライブ、メモリディスクがあります。それぞれ論理装置名(**A:, B:, C:～J:, K:, L:, M:**)または物理装置名(**FDn:, HDm:, MEM:**)で指定します。
  ＜装置名＞を省略した場合は、現在の装置(カレントドライブ)のカレントディレクトリから指定ファイル名をさがします。階層ディレクトリでファイルを指定する場合は＜装置名＞と＜ファイル名＞の間にディレクトリパスを指定します。
- ＜ファイル名＞は拡張子を付けて指定します。拡張子を省略すると、**.DAT**が設定されます。
- 指定されたファイルを見つけたあとにメモリ上のプログラムを消去(**new**)して指定ファイルを読み込みます。したがって、指定ファイルが見つからなかった場合は、メモリ上のプログラムは消去されません。

参考☞　**list＊** …　コメント行の装置記録。(145ページ)

SAVE （セーブ・save）

プログラムを指定の装置に記録します。

書 式

save "〔〈装置名〉〕〈ファイル名〉"〔,{A/C/P}〕

＜装置名＞…A:, B:, C:～J:, K:, L:, M:

または、FDn:, HDm:, MEM:, COM1:,CRT1:

nは1～4、mは1～8

省略形

sa.

説 明

- プログラムを＜装置名＞と＜ファイル名＞を指定して装置に記録します。
- ＜装置名＞には、フロッピーディスクドライブ、ハードディスクドライブ、メモリディスク、RS-232Cインターフェイス、キーボードがあります。
  それぞれ論理装置名（A:, B:, C:～J:, K:, L:, M:）または物理装置名（FDn:, HDm:, MEM:, COM1:）で指定します。
  ＜装置名＞を省略すると現在の装置（カレントドライブ）のカレントディレクトリに登録します。階層ディレクトリのディレクトリ階層を指定する場合は＜装置名＞と＜ファイル名＞の間にディレクトリパスを指定します。
- 1つのディレクトリ内で同じファイル名を使うことはできません。
- ＜ファイル名＞は拡張子を付けて指定します。
  拡張子を省略すると.BASが設定されます。
- A, Cを指定すると、プログラムはアスキー形式（DAT）で記録されます。
  Cを指定すると、行番号の前の空白を記録しないので、A指定より少ない領域で記録できます。
  ＜装置名＞がCOM1:,CRT1:の場合はA,Cを指定してアスキー形式のファイルにしてください。
  アスキー形式のファイルの場合、拡張子を省略すると.DATが設定されます。
- Pを指定すると、プログラムは保護形式で記録されます。
  以後、このファイルを読み出しても、listコマンドで内容を見たり、editコマンドで内容を変更することはできなくなります。
  なお、いったんPを指定して記録すると、これを解除することはできませんのでご注意ください。

例

```
save "A:TEST.BAS"
```

… TEST. BASというファイル名でドライブAのカレントディレクトリへ記録します。

```
save "B:WORK¥TEST",A
```

… TEST. DATというファイル名でドライブBのカレントディレクトリにあるWORKというディレクトリ下にアスキー形式で記録します。

```
save "COM1:",A
```

… メモリ上のプログラムをアスキー形式でRS-232Cインターフェイスに送出します。

参 考☞

bsave …… 機械語プログラムの記録。(303ページ)

load …… プログラムを読み込む。(296ページ)

BLOAD （バイナリーロード・binary load）

機械語プログラムを読み込みます。

書　式　bload "〔〈装置名〉〕〈ファイル名〉"〔,〈アドレス〉〕〔,R〕

＜装置名＞… **A:, B:, C:～J:, K:, L:, M:**

または、**FDn:, HDm:, MEM:, COM1:**

nは1～4、mは1～8

省略形　bl.

説　明

- 指定された機械語プログラムをメモリ上に読み込みます。
  読み込むエリアはあらかじめ**clear**文により設定しておきます。エリアが確保されていない場合は**memory protection error**になります。
- ＜装置名＞には、フロッピーディスクドライブ、ハードディスクドライブ、メモリディスク、RS-232Cインターフェイスがあります。
  それぞれ論理装置名（**A:, B:, C:～J:, K:, L:, M:**）または物理装置名（**FDn:, HDm:, MEM:, COM1:**）で指定します。
- ＜装置名＞の指定を省略すると現在の装置（カレントドライブ）のカレントディレクトリのファイル名で指定されたファイルを読み込みます。また、階層ディレクトリによりファイル指定を行う場合は＜装置名＞と＜ファイル名＞の間にディレクトリパスを指定します。
- ＜アドレス＞は読み込みの開始アドレスを指定します。
  ＜アドレス＞が指定されると、**def seg**文で指定されたセグメントベースに＜アドレス＞を加えた番地へプログラムを読み込みます。
- ＜アドレス＞が省略されると、**def seg**文で指定されたセグメントベースの先頭番地へプログラムを読み込みます。
  def seg文によるセグメントベースの指定が無い場合は**memory protection error**になります。（**def seg**文参照）
- ＜アドレス＞は0～65535の値で指定します。
- **R**を指定するとプログラム読み込み後実行を開始します。

参　考　**bsave** …… 機械語ファイルの記録。（303ページ）

run＜ファイル＞ …… BASICや機械語のプログラムファイルの**load**および**run**。（294ページ）

サンプル

```
10 rem --- BLOAD ---
20 clear &H400
30 def seg=segptr(2)
40 bload "test2",&HA00
```

- 機械語プログラム領域を128KB確保し、領域の先頭から&HA 00 番地目へプログラムを読み込みます。

BSAVE （バイナリー セーブ・binary save）

機械語プログラムを記録します。

書 式　bsave "〔〈装置名〉〕〈ファイル名〉",〈開始アドレス〉,〈サイズ〉

＜装置名＞…**A:, B:, C:～J:, K:, L:, M:**

または、**FDn:, HDm:, MEM:, COM1:**

nは1～4、mは1～8

省略形　bs.

説 明

- clear文で設定している機械語エリアのプログラムをファイル名と開始アドレス、サイズを指定しファイルとして登録します。ファイルの形式は**BIN**形式となります。
- ＜装置名＞には、フロッピーディスクドライブ、ハードディスクドライブ、メモリディスク、RS-232Cインターフェイスがあります。
  それぞれ論理装置名（**A:, B:, C:～J:, K:, L:, M:**）または物理装置名（**FDn:, HDm:, MEM:, COM1:**）で指定します。
- ＜装置名＞の指定を省略すると現在の装置（カレントドライブ）のカレントディレクトリに記録されます。階層ディレクトリのディレクトリ階層を指定する場合は＜装置名＞と＜ファイル名＞の間にディレクトリパスを指定します。
- ＜ファイル名＞は拡張子を付けて指定します。拡張子を省略すると**.BIN**が設定されます。
- ＜開始アドレス＞が指定されると、**def seg**文で指定されたセグメントベースに＜開始アドレス＞を加えた番地から記録します。
  ＜開始アドレス＞が省略されると、**def seg**文で指定されたセグメントベースの先頭番地から記録します。
  def seg文によるセグメントベースの指定が無い場合は **memory protection error**になります。（**def seg**文参照）
- ＜サイズ＞は記録する機械語プログラムのバイト数を指定します。
- ＜開始アドレス＞、＜サイズ＞は0～65535の値で指定します。

参 考　bload …… 機械語ファイルの読み出し。（301ページ）

run ＜ファイル＞ …… BASICや機械語のプログラムファイルのloadおよびrun。（294ページ）

サンプル

```
10 rem --- BSAVE ---
20 clear &H400
30 def seg=segptr(2)
40 bsave "test2",&HA00,&H100
```

MERGE （マージ）

プログラムを併合します。

書　式　merge "〔〈装置名〉〕〈ファイル名〉"

＜装置名＞… **A:, B:, C:～J:, K:, L:, M:**
または、**FDn:, HDm:, MEM:, COM1:**
nは 1 ～ 4 、mは 1 ～ 8

省略形　**m.**

説　明

- メモリ上にあるプログラムに指定したプログラムファイルの内容を併合(合成)します。
- 併合できるプログラムファイルは**BAS**と**アスキー**形式のものです。ただし、**アスキー**形式のものでも、プログラムの書式が合っていないものは読み込み途中でエラーが起こる可能性があります。
- 内容の併合はメモリ上のプログラムにプログラムファイルの内容が追加されるようになり、追加する側の行番号が追加される側の行番号と同じであれば結果は追加する側の内容になります。
- ＜装置名＞には、フロッピーディスクドライブ、ハードディスクドライブ、メモリディスク、RS-232Cインターフェイスがあります。
  それぞれ論理装置名(**A:, B:, C:～J:, K:, L:, M:**)または物理装置名(**FDn:, HDm:, MEM:, COM1:**)で指定します。
- ＜装置名＞を省略した場合は現在の装置(カレントドライブ)のカレントディレクトリより指定ファイルを読み出します。
- 階層ディレクトリでファイルを指定する場合は＜装置名＞と＜ファイル名＞の間に＜ディレクトリパス＞を指定します。
- ＜ファイル名＞は拡張子を付けて指定します。拡張子を省略すると**.BAS**が設定されます。
- ＜装置名＞に**"COM 1 :"**を指定すると**アスキー**形式のファイルとみなし＜ファイル名＞は無視されます。

例

```
merge "ABC"
```

…… 現在メモリ上にあるプログラムにカレントディレクトリ下の"ABC"というプログラムを併合します。

参　考　chain merge …… 指定プログラムを併合して実行する。(314ページ)

サンプル

```
10 rem --- MERGE ---
20 for N=1 to 5
30   read KIND$(N),PRICE(N)
40   print KIND$(N),PRICE(N)
50 next
60 end

60 rem --- DATA ---
70 data "トマト",180
80 data "きゅうり",100
90 data "キャベツ",200
100 data "レタス",220
110 data "大根",160
120 end
merge "DATA"
Ok
list
   10 rem --- MERGE ---
   20 for N=1 to 5
   30   read KIND$(N),PRICE(N)
   40   print KIND$(N),PRICE(N)
   50 next
   60 rem --- DATA ---
   70 data "トマト",180
   80 data "きゅうり",100
   90 data "キャベツ",200
  100 data "レタス",220
  110 data "大根",160
  120 end
Ok
```

- **"DATA"**のプログラムをディスクにセーブした後、**"MERGE"**のプログラムに**merge**命令によってディスク上の**"DATA"**のプログラムを併合し、１本のプログラムにします。

KILL （キル）

ファイルを消去します。

書 式

kill "〔〈装置名〉〕〈ファイル名〉" (1)

kill all "〔〈装置名〉〕〔〈ディレクトリ名〉〕" (2)

＜装置名＞…**A:, B:, C:～J:, K:, L:, M:**

または、**FDn:, HDm:, MEM:**

nは1～4，mは1～8

省略形

kil.

説 明

- 指定されたファイルを消去し登録を抹消します。
- (書式1)を使用すると指定されたファイルのみを消去し、(書式2)を使用すると指定されたディレクトリ内のファイルすべてを消去します。
- ＜装置名＞には、フロッピーディスクドライブ、ハードディスクドライブ、メモリディスクがあります。
  それぞれ論理装置名(**A:, B:, C:～J:, K:, L:, M:**)または物理装置名(**FDn:, HDm:, MEM:**)で指定します。
- ＜装置名＞の指定を省略すると現在の装置(カレントドライブ)のカレントディレクトリの＜ファイル名＞で指定されたファイルを消去します。また、階層ディレクトリによりファイル指定を行う場合は＜装置名＞と＜ファイル名＞の間にディレクトリパスを指定します。
- ＜ファイル名＞は拡張子を付けて指定します。拡張子を省略することはできません。
- ＜ファイル名＞にワイルドカードを使用できます。
  (ワイルドカードについては**files/lfiles**コマンド参照)
- 指定されたファイルが存在しない場合はエラーとなります。
- 指定されたファイルがオープンされている場合はエラーとなります。
- 指定されたファイルがプロテクトされている場合は消去されずにエラーとなります。setコマンドによりあらかじめプロテクトをはずしておく必要があります。
- ディレクトリ管理ファイルはこのコマンドでは消去できません。(**rmdir**)

例

```
kill "TEST.BAS"
```

… カレントドライブのカレントディレクトリ下のTEST．BASというファイルを消去します。

```
kill "B:¥DRC¥TEST.BAS"
```

… ドライブBのルートディレクトリ内のDRCというディレクトリ下のTEST．BASというファイルを消去します。

```
kill all "B:"
```

… ドライブBのカレントディレクトリ下のファイルすべてを消去します。

```
kill all "¥DRC"
```

… カレントドライブのルートディレクトリ内のDRCというディレクトリ下のファイルすべてを消去します。

```
kill all ""
```

… カレントドライブのカレントディレクトリ下のファイルすべてを消去します。

参 考 set …… ファイルのプロテクトを設定する。(311ページ)

rmdir …… ディレクトリファイルの消去。(293ページ)

NAME （ネーム）

ファイルの登録名を変更します。

書　式

name "〔〈装置名〉〕〈ファイル名〉"as"〈新ファイル名〉"

＜装置名＞…**A:, B:, C:～J:, K:, L:, M:**

または、**FDn:, HDm:, MEM:**

nは1～4，mは1～8

省略形

na.

説　明

- ＜装置名＞の指定ファイルを＜新ファイル名＞で登録します。
- ＜装置名＞には、フロッピーディスクドライブ、ハードディスクドライブ、メモリディスクがあります。
  それぞれ論理装置名（**A:, B:, C:～J:, K:, L:, M:**）または物理装置名（**FDn:, HDm:, MEM:**）で指定します。
- ＜装置名＞を省略した場合は現在の装置（カレントドライブ）のカレントディレクトリ内で指定ファイルをさがします。
- 階層ディレクトリによりファイル指定を行う場合は、＜装置名＞と＜ファイル名＞の間にディレクトリパスを指定します。
- ＜ファイル名＞は拡張子を付けて指定します。拡張子を省略することはできません。
- ＜ファイル名＞と＜新ファイル名＞を同じファイル名にすることはできません。
- プロテクトとしてあるファイルに対して行った場合は、エラーになります。
  あらかじめsetコマンドによりプロテクトをはずしておく必要があります。
  （**set**文参照）

例

```
name "TEST1.BAS" as "TEST2.BAS"
```

… カレントドライブのカレントディレクトリにあるTEST 1というファイルをTEST2というファイル名に変更します。

```
name "A:TEST1.BAS" as "TEST2.BAS"
```

… ドライブAのカレントディレクトリ下のTEST 1というファイルをTEST 2というファイル名に変更します。

```
name "A:DRC¥TEST1.BAS" as "TEST2.BAS"
```

… ドライブAのDRCというディレクトリ下のTEST 1というファイルをTEST 2というファイル名に変更します。

参　考　　set　……　ファイルのプロテクトを設定する。(311ページ)

SET （セット）

ファイルにプロテクトをかけます。

書 式

set "〔〈装置名〉〕〈ファイル名〉", { "P" / "" } (1)

set ＃〈ファイル番号〉, { "P" / "" } (2)

＜装置名＞…**A:, B:, C:～J:, K:, L:, M:**
または、**FDn:, HDm:, MEM:**
nは1～4，mは1～8

説 明

- ファイルを誤操作によって消去したり、書き換えてしまうのを防ぐためにファイルごとにプロテクト(書き換え禁止)をかけることができます。ファイル指定のあとに文字データとして**"P"**を指定するとそのファイルはプロテクト状態になり、**files**コマンドによるファイル表示のときに、日付表示のあとに**＊**を表示します。
- プロテクトされたファイルに対しては次の操作を禁止します。
  killコマンドによる消去。
  nameコマンドによる名前の変更。
  open **"A"**または**"O"**によるファイルへの書き込み。
  ランダムアクセスファイルへの書き込み。(open "R"時のput＃)
  以上の操作を行うと**write protect error**になります。
- プロテクトの解除はファイル指定のあとに""を指定します。
- ＜装置名＞には、フロッピーディスクドライブ、ハードディスクドライブ、メモリディスクがあります。
  それぞれ論理装置名(**A:, B:, C:～J:, K:, L:, M:**)または物理装置名(**FDn:, HDm:, MEM:**)で指定します。
- ＜装置名＞の指定を省略すると現在の装置(カレントドライブ)のカレントディレクトリで＜ファイル名＞で指定されたファイルをさがします。また、階層ディレクトリによりファイル指定を行う場合は、＜装置名＞と＜ファイル名＞の間にディレクトリパスを指定します。
- ＜ファイル名＞は拡張子を付けて指定します。拡張子を省略することはできません。
- 書式(1)で指定すると、ファイルそのものを書き換え禁止にしますが、書式(2)で指定すると、そのファイルが**open**されている間だけ書き換え禁止にします。

例

```
set "ABC.BAS","P"
```

…… カレントディレクトリ内のファイル名が"ABC.BAS"のファイルにプロテクトをかけます。

参 考 **attr$** …… ファイルのプロテクト状態を調べる関数。(336ページ)

VERIFY ON/OFF （ベリファイ オン/オフ）

フロッピーディスクへの書き込みチェックを指定します。

書　式

verify { on / off }

省略形

ve.

説　明

- フロッピーディスクへの書き込みを行った後に、書き込んだデータを読み出してチェックを行うかどうかを指定します。指定を**on**とするとチェックを行い、**off**とするとチェックを行いません。
- BASIC起動時は起動直前の設定と同じになっています。
- 書き込みを行ったあとにチェックを行うことをリードアフターライト（read after write）といいます。

例

```
verify on
```

…… 以後リードアフターライトを行います。

CHAIN （チェイン）

現在実行中のプログラムから指定プログラムファイルを読み出し実行します。

書　式

chain 〈ファイル指定〉[,〈実行開始行〉][,all]
chain merge 〈ファイル指定〉[,〈実行開始行〉][,all][,delete 〈削除範囲〉]

省略形

cha.

説　明

- chainは、現在実行中のプログラムを消去して、指定プログラムファイルを読み出し実行します。
- ＜実行開始行＞は、指定プログラムを読み込んだあとに実行を開始する行を、行番号で指定します。省略した場合はプログラムの最初から実行します。なお、省略する場合はカンマ(,)も含めて省略してください。
- allは呼び出したプログラムにすべての変数や配列を引き渡すときに指定します。指定がない場合はすべての変数や配列は消去されますが、common文で指定されている変数や配列は消去しません。
- chain mergeは現在のプログラムを消去せずに指定ファイルを現在のプログラムに併合します。この場合はdeleteの指定によって、あらかじめ現在のプログラムの一部を消去しておくことができます。
  ＜削除範囲＞の指定は＜開始行＞－＜終了行＞でdeleteコマンドと同じです。
- ＜ファイル指定＞は次の書式の文字列型データで指定します。
  "[＜装置名＞][＜ディレクトリパス＞]＜ファイル名＞"
  ＜装置名＞には、フロッピーディスクドライブ、ハードディスクドライブ、メモリディスク、RS-232Cインターフェイスがあります。
  それぞれ論理装置名(A:, B:, C:～J:, K:, L:, M:)または物理装置名(FDn:, HDm:, MEM:, COM1:)で指定します。
- ＜ディレクトリパス＞は階層ディレクトリのパス(道すじ)を指定します。(chdir コマンド参照)
- ＜ディレクトリパス＞の指定を省略すると、chdirコマンドで設定されているカレントディレクトリ内でのファイル指定になります。
- ＜ファイル名＞は拡張子を付けて指定します。拡張子を省略すると.BASが設定されます。

例

```
100 chain "ABC",all
```

…… 変数や配列を消去せずにカレントディレクトリ内の"ABC"のプログラムを読み出し、プログラムの最初より実行を開始します。

参　考☞　merge　……　プログラムの併合。(305ページ)

common　……　共通変数の定義。(316ページ)

COMMON （コモン）

共通変数の定義を行います。

書　式　common〈変数名〉[,〈変数名〉] …

省略形　com.

説　明

- chain文の実行を行ったときに、次のプログラムへ引き渡す変数や配列を定義します。次のプログラムへ移ったとき、この命令で定義した変数や配列は、消去されずそのままの変数名や配列名で値が残ります。
- ＜変数名＞に配列変数を指定する場合は、配列名の後に( )を付けます。
- chain文ではcommon文で指定されている変数や配列を残すか、またはすべての変数や配列を残すかの選択ができます。

例

```
common A,B$,X()
```

…… 変数AとB$、配列X( )を共通変数として定義します。

参　考☞　chain …… 新しいプログラムを読み出し、実行する。(314ページ)

OPEN （オープン）

データファイルの操作を開始します。

書 式

open { "I",＃〈ファイル番号〉,〈ファイル指定〉
"O",＃〈ファイル番号〉,〈ファイル指定〉
"A",＃〈ファイル番号〉,〈ファイル指定〉
"R",＃〈ファイル番号〉,〈ファイル指定〉〔,〈レコード長〉〕
"C",＃〈ファイル番号〉,〈ファイル指定〉 }

省略形

op.

説 明

- データファイルのアクセスを開始するために、ファイルに<ファイル番号>を割り当てます。**I, O, A, R, C**はアクセスの形態を指定します。

"I"(Input) … シーケンシャルアクセスファイルの読み出しを開始します。**フロッピーディスク, ハードディスク, メモリディスク, RS-232Cインターフェイス, キーボード**の装置に対して可能になります。

"O"(Output) … シーケンシャルアクセスファイルの作成(書き込み)を開始します。**フロッピーディスク, ハードディスク, メモリディスク, RS-232Cインターフェイス, プリンタ, CRT**の各装置に対して可能になります。

"A"(Append) … すでに作成されているシーケンシャルアクセスファイルに対して追加書き込みを開始します。**フロッピーディスク, ハードディスク, メモリディスク**の各装置に対して可能になります。

"R"(Random) … ランダムアクセスファイルの操作を開始します。**フロッピーディスク, ハードディスク, メモリディスク**の各装置に対して可能になります。

"C" … RS-232Cインターフェイスに対してシーケンシャルアクセスファイルの読み出しと書き込みを開始します。

- <ファイル番号>は1～127の整数値を指定します。これはファイルの操作に関する領域を割り当てることになり、以後のデータのアクセスはこの番号を指定します。

ファイル操作の終了はclose文によって行い、割り当てた領域を解放します。したがって使用途中のファイル番号(openで指定してcloseされていない)をほかのファイルに割り当てることはできません。

- ＜ファイル指定＞は次の書式の文字列型データで指定します。
  "〔＜装置名＞〕〔＜ディレクトリパス＞〕＜ファイル名＞"
- ＜装置名＞は**A:, B:, C:～J:, K:, L:, M:**の論理装置名または**FDn:, HDm:, MEM:, COM1:, CRT1:, LPT:, KB:**の物理装置名で指定します。
- ＜ディレクトリパス＞は階層ディレクトリのパス(道すじ)を指定します。(**chdir**コマンド参照)
- ＜ディレクトリパス＞の指定を省略すると、**chdir**コマンドで設定されているカレントディレクトリ内でのファイル指定になります。
- ＜ファイル名＞は拡張子を付けて指定します。拡張子を省略すると**.DAT**が設定されます。
- **LPT:, COM1:, KB:**の各装置の＜ファイル名＞は無視されます。また、**COM1:**の通信機能の設定は**init "COMn:"**文で行って下さい。
- ＜レコード長＞は、ランダムアクセスファイルの1レコードのバイト数を**1～2048**までの値で指定できます。
- ＜レコード長＞の省略時は256バイトが1レコードになります。
- ＜ファイル指定＞を**"CRT1：CG"**として**open"O"**を実行した場合、画面にコントロールコードを実行せずに表示します。

参　考☞

print# (321ページ) **print# using** (323ページ) **write #** (325ページ)
…… シーケンシャルアクセスファイルの書き込み。

input# (324ページ), **line input#** (326ページ)
…… シーケンシャルアクセスファイルの読み出し。

put# (329ページ), **get#** (330ページ)
…… ランダムアクセスファイルの読み書き。

close …… ファイル操作の終了。(320ページ)

サンプル

```
10 rem --- OPEN and CLOSE ---
20 open "O",#1,"SEQDUM.DAT"
30 print #1,"Ｋｅｎｊｉ"
40 print #1,"１８歳"
50 print #1,"男性"
60 print #1,"オートバイ, パソコン"
70 print #1,"Ｙａｓｕｏ"
80 print #1,"２４歳"
90 print #1,"男性"
100 print #1,"ゴルフ, スキー"
110 close #1
120 end
```

```
10 rem --- OPEN and CLOSE ---
20 open "I",#1,"SEQDUM.DAT"
30 while not eof(1)
40   input #1,A1$
50   input #1,A2$
60   input #1,A3$
70   input #1,A4$
80   print "氏名 :";A1$
90   print "年令 :";A2$
100  print "性別 :";A3$
110  print "趣味 :";A4$
120 print "-------------------------"
130 wend
140 close #1
150 end

run
氏名 :Ｋｅｎｊｉ
年令 :１８歳
性別 :男性
趣味 :オートバイ, パソコン
-------------------------
氏名 :Ｙａｓｕｏ
年令 :２４歳
性別 :男性
趣味 :ゴルフ, スキー
-------------------------
```

- 左側のプログラムを実行して作った**"SEQDUM．DAT"**というファイルに記録されているデ-タを読み込み、表示します。
- ファイルの終わりが検出されるとファイルを**close**し、終了します。

CLOSE （クローズ）

ファイル操作を終了します。

書　式　close〔＃〈ファイル番号〉〔,＃〈ファイル番号〉〕…〕

省略形　clo.

説　明

- open文でオープン(開始)されたファイルの操作を終了します。
- ファイル番号は複数指定することができ、指定しない場合はオープンされているすべてのファイルの操作をクローズ(終了)します。
- **open文の"O", "A", "R", "C"**でファイルオープンをして、書き込みを行った場合は、この命令で終了しないと正常に登録されません。
- end文を実行すると自動的にこの命令の機能が行われます。

例　1000 close …… 全てのオープン中のファイル操作を終了します。

参　考☞　open …… ファイル操作の開始。(317ページ)

PRINT ＃　（プリント シャープ）

オープンしたファイルへデータを書き込みます。

書　式　　print ＃〈ファイル番号〉,〈データ〉[{ ; / , } 〈データ〉]…

省略形　　p．＃　？＃

説　明

- **open "O","A","R",**または**"C"**の指定でファイルオープンを行ったのちファイル操作開始時の<ファイル番号>を指定してデータを書き込みます。
- 書き込むデータに対する注意は**print**文と同じです。
- 数値型のデータは値が正の場合1個のスペース、負の場合はマイナス(ー)記号を付け、最後に1個のスペースを付けます。
- 句切り記号に**",**"を使用すると間隔があきます。詰めて書き込む場合は句切り記号として";"を使用してください。
- この命令の最後が＜データ＞で終了している場合は改行コード**(&H0D&H0A)**が付加され、句切り記号(,または;)となっている場合は改行コードは付加されません。
- 文字列型データもそのまま書き込まれますので、文字コードが**&H20**より小さなコードを書き込んだ場合は、データ入力時に正常に入力できないことがあります。特に**&H 0 D**は改行コードとして入力時のデータ句切りになります。
- ＜データ＞をすべて省略すると、改行コードのみを書き込みます。

例

```
1020 print #1,A
```

…… ファイル番号１番に割り当てられているファイルに変数Ａの内容を書き込みます。

参　考[注6]　**input＃**(324ページ), **line input＃**(326ページ)

…… 書き込んだデータの読み出し。

サンプル

```
 10 rem --- PRINT# ---
 20 open "O",#1,"PRTDUM"
 30 for I=1 to 3
 40    read HINMEI$,TANKA
 50    print #1,HINMEI$,TANKA
 60 next I
 70 close #1
 80 print "品名        単価":print
 90 open "I",#1,"PRTDUM"
100 for I=1 to 3
110   input #1,HYOUJI$
120   print HYOUJI$
130 next I
140 close #1
150 end
160 data "ﾃﾞｨｽｸ",500
170 data "ﾃﾞｨｽﾌﾟﾚｲ",128000
180 data "ﾌﾟﾘﾝﾀ",80000
run
品名        単価

ﾃﾞｨｽｸ       500
ﾃﾞｨｽﾌﾟﾚｲ    128000
ﾌﾟﾘﾝﾀ       80000
Ok
```

- **print #**命令で**"PRTDUM"**というシーケンシャルファイルにデータを書き込んだ後、読み出し表示します。

PRINT # USING (プリント シャープ ユージング)

書式設定をしてファイルに書き込みます。

書 式

print #〈ファイル番号〉,using 〈書式〉;〈データ〉[{ , / ; } 〈データ〉]…

省略形

p. #us. ? #us.

説 明

- シーケンシャルアクセスファイルに対する書き込みにおいて書式設定をした書き込みを行います。
- 指定する書式はprint using文と同じで、画面に表示されるものと同じように書き込みが行われます。
- 書き込むデータに対する注意はprint #文と同じです。

参 考☞ print using …… 書式指定をして画面にデータを表示する。(177ページ)

サンプル

```
 10 rem --- print # using ---
 20 open "O",#1,"TESTDT2.DAT"
 30 for A=1 to 10
 40   print using "####";A,A*A
 50   print #1,using "####";A,A*A
 60 next
 70 close #1
 80 ' このデータは line input # のサンプルで
 90 ' ファイル名を変更することにより
100 ' 読み出すことができます
```

INPUT ＃ （インプット シャープ）

オープンされたファイルよりデータを読み出します。

書　式　input　＃〈ファイル番号〉,〈変数〉〔,〈変数〉〕…

省略形　i. ＃

説　明

- open文で"I","R"または"C"の指定でオープンされたファイルからデータを読み出し、変数に代入します。（open文参照）
- ＜変数＞に数値変数名を指定すると、ファイルから読み出したデータにカンマ(,)、空白、復帰改行コードが現れると、データの句切りとみなします。
  ただし、データの先頭に空白がある場合は句切りとみなしません。なお、読み出したデータが数値でない場合はエラーとなります。
- ＜変数＞に文字変数名を指定すると、ファイルから読み出したデータにカンマ(,)、復帰改行コードが現れると、データの句切りとみなします。
  ただし、ダブルクォーテーション(")で囲まれた文字列を１つの文字列として扱いますので、カンマがダブルクォーテーションで囲まれた文字列中にある場合は句切りとみなしません。
  なお、ダブルクォーテーション自体はデータとして入力されません。
- 句切り記号が見当たらない場合は、255文字目のあとに句切りがあるものとします。
- この命令は画面を使って入力を行うinput文が、ファイルからの入力となったものとみることができます。

サンプル

```
10 rem --- input # ---
20 open "I",#1,"TESTDT1.DAT"
30 for A=1 to 10
40   input #1,X,Y
50   print X,Y
60 next
70 close #1
80 ' write # のサンプルのデータを読み出せます
```

WRITE ＃ (ライト シャープ)

オープンしたファイルへデータを書き込みます。

書 式　write ＃〈ファイル番号〉[,〈データ〉[{ , ; } 〈データ〉]…]

省略形　w. ＃

説 明

- open "O"またはopen "A","R"によりオープンされているファイルに対して、画面表示のwrite文と同形式でデータを書き込みます。
- 句切り記号のカンマ(,)とセミコロン(;)は同じ機能で間隔をあけずに書き込みます。
- 文字列型のデータはダブルクォーテーション(")で囲まれます。
- データとデータの間はカンマを付けて書き込みが行われます。
- ファイル番号に続くすべての<データ>を書き込んだあとに改行コードを書き込みます。また、<データ>をすべて省略した場合は改行コードのみを書き込みます。
- この命令の最後を句切り記号で終了させることはできません。この命令の最後は必ずデータにしてください。

サンプル

```
10 rem --- write # ---
20 open "O",#1,"TESTDT1.DAT"
30 for A=1 to 10
40   write A,A*A
50   write #1,A,A*A
60 next
70 close #1
80 ' このデータは input #   line input #
90 ' のサンプルで読み出せます
```

LINE INPUT ＃ (ライン インプット シャープ)

ファイルより行単位でデータを入力します。

書　式　line input ＃〈ファイル番号〉,〈文字列型変数〉

省略形　lin. i. ＃

説　明

- **open"I", R", "C"**によりオープンされたファイルからデータを行単位で読み込みます。
- ファイルに書き込まれているデータの中で改行コード(**&H0D**または**&H0D, &H0A**)を句切りとして＜文字列型変数＞に入力します。句切り記号が見当たらない場合は**255**文字目のあとに句切りがあるものとします。

サンプル

```
10 rem --- line input # ---
20 open "I",#1,"TESTDT1.DAT"
30 for A=1 to 10
40   line input #1,X$
50   print X$
60 next
70 close #1
80 ' write #   print # using のサンプルの
90 ' データを読み出せます
```

FIELD （フィールド）

レコードバッファに変数を割り付けます。

書　式　field ＃〈ファイル番号〉,〈フィールド長〉as〈文字変数〉[,〈フィールド長〉as〈文字変数〉]…

省略形　fie.

説　明

- ランダムアクセスファイルをアクセスするときに使用する変数を定義します。ランダムアクセスファイルは1レコードのレコードバッファを介してデータを読み書きします。このレコードバッファをどのように変数に割り付けるかを設定します。＜フィールド長＞で1つの＜文字変数＞に割り当てるバイト数を指定します。＜ファイル番号＞は open "R"文でファイルオープンされたときの番号になります。
- 1つのレコード内ではフィールド長の合計がopen "R"文で設定されたレコード長をこえない範囲でいくつでも設定できます。
- この命令によりランダムアクセスファイル用に割り付けられた変数を入力文(input, read, devi$文など)の入力変数として使用したり代入文の左辺に使用した場合は、フィールドの割り付けが無効となり、通常の文字変数として扱われます。field文で指定された変数にデータを設定するにはlset文やrset文で行います。
- 同じ＜ファイル番号＞を使用して1つのレコードに重複してフィールド変数を割り付けることができますが、同じ変数名であとから指定された場合は、それまでの割り付けが無効になります。また、＜ファイル番号＞が異なっていても同じ変数名を使用すると、同じ変数名を使用している以前の割り付けが無効になります。

例

```
field #1,2 as A$,5 as B$,8 as C$
```

…… ファイル番号1番のレコードバッファのうち、2バイトをA$,5バイトをB$,8バイトをC$に割り当てます。

参　考　lset／rset …… レコードバッファにデータを書き込む。(331ページ)

サンプル

```
 10 rem --- FIELD ---
 20 open "R",#1,"RNDFIL.DAT"
 30 field #1,20 as A$,5 as B$,5 as C$
 40 MAXREC=lof(1)
 50 MAXREC=MAXREC+1
 60 input "商品名 ";AA$
 70 input "仕入値 ";BB
 80 input "売り値 ";CC
 90 lset A$=AA$
100 rset B$=mks$(BB)
110 rset C$=mks$(CC)
120 put #1,MAXREC
130 input "入力を続けますか ";ANS$
140 if ANS$="y" or ANS$="Y" then 50 else 160
150 goto 130
160 close #1
170 end
run
商品名 ? ﾄﾏﾄ
仕入値 ? 25
売り値 ? 40
入力を続けますか ? y
商品名 ? ｷｬﾍﾞﾂ
仕入値 ? 90
売り値 ? 120
入力を続けますか ? n
```

```
 10 rem --- FIELD ---
 20 open "R",#1,"RNDFIL.DAT"
 30 field #1,20 as A$,5 as B$,5 as C$
 40 NOWREC=0
 50 while not eof(1)
 60   NOWREC=NOWREC+1
 70   get #1,NOWREC
 80   B=cvs(B$)
 90   C=cvs(C$)
100   print "商品名 ";A$
110   print "仕入値 ";B
120   print "売り値 ";C
130   print "--------------------"
140 wend
150 close #1
160 end
run
商品名 ﾄﾏﾄ
仕入値  25
売り値  40
--------------------
商品名 ｷｬﾍﾞﾂ
仕入値  90
売り値  120
--------------------
```

- ランダムファイルを**open**し、**field**命令でそれぞれのデータの大きさを定義した後、商品名は左づめで、仕入値と売り値はいったん文字列データに変換した後右づめで記録します。
- 一番目のプログラムで記録したファイルを**open**し、データを読み込みます。このとき、文字列データで記録されている仕入値と売り値を再び実数型のデータに変換します。ファイルの終わりが検出されると、終了します。

PUT ＃ （プット シャープ）

ファイルへデータを書き込みます。

書　式　put ＃〈ファイル番号〉[,〈レコード番号〉]

説　明

- ＜ファイル番号＞で指定されたファイルにデータを書き込みます。
- 指定されたファイル装置がフロッピーディスクドライブ、ハードディスクドライブ、メモリディスクの場合は、＜レコード番号＞で指定したレコードへ1レコード分のデータを書き込みます。
  ＜レコード番号＞は1～65535の値で指定し、省略するとそれ以前に**get＃**文または**put＃**文で指定されたレコードの次のレコードへ書き込まれます。

サンプル

```
 10 rem --- put # ---
 20 open "R",#1,"名簿",32
 30 field #1,20 as 名前$,5 as 年令$
 40 for A=1 to 10
 50   read N,A$,B$
 60   lset 名前$=A$
 70   rset 年令$=B$
 80   put #1,N
 90 next
100 close #1
1000 data  9,山田 太郎,38,  4,鈴木 政男,54,  5,田中 和彦,28
1010 data 10,吉田 次郎,18,  3,坂野 春吉,31,  6,南条 義雄,20
1020 data  2,小山 三郎,25,  1,板倉 賢一,40,  8,丸山 明  ,22
1030 data  7,前川 一  ,31
1040 ' このデータは get # のサンプルで読み出すことができます
```

GET ＃　（ゲット シャープ）

ファイルからデータを読み出します。

書　式　　get ＃〈ファイル番号〉[,〈レコード番号〉]

説　明

- ＜ファイル番号＞で指定されたファイル中のデータを読み出します。
- 指定されたファイル装置がフロッピーディスクドライブ、ハードディスクドライブ、メモリディスクの場合は、＜レコード番号＞で指定したレコードから1レコード分のデータを読み出します。
  ＜レコード番号＞は1～65535の値で指定し、省略するとそれ以前に**put＃**文または**get＃**文で指定されたレコードの次のレコードが読み出されます。

サンプル

```
 10 rem --- get # ---
 20 open "R",#1,"名簿",32
 30 field #1,20 as 名前$,5 as 年令$
 40 input "番号を入力 (1-10  end->999)  ",N
 50 print
 60 if N=999 then 110
 70 get #1,N
 80 print N;名前$;年令$
 90 print
100 goto 40
110 close #1
120 ' put # のサンプルのデータを読み出せます
130 ' 999を入力すると終了します
```

LSET （レフト セット・left set）
RSET （ライト セット・right set）
レコードバッファにデータを書き込みます。

書　式

lset〈フィールド変数〉＝〈文字列型データ〉
rset〈フィールド変数〉＝〈文字列型データ〉

省略形

ls．rs.

説　明

- **field**文でレコードバッファに割り付けられている変数にデータを書き込む場合は、**rset**文または**lset**文により行います。**lset**文は左詰め、**rset**文は右詰めでデータを書き込みます。
- 書き込む＜文字データ＞が**field**文で指定されている変数の長さより短い場合は、残りの部分にスペースが書き込まれます。また、データが長い場合は長い部分を切り捨てます。

参　考

open "R" …… ランダムアクセスファイルのオープン。(317ページ)
put# …… ランダムアクセスファイルのデータ書き込み。(329ページ)
get# …… ランダムアクセスファイルのデータ読み出し。(330ページ)
field …… レコードバッファに変数を割り付ける。(327ページ)

サンプル

```
 10 rem --- LSET RSET ---
 20 open "R",#1,"test"
 30 field #1,10 as A$,10 as B$
 40 input "文字列を入力して下さい。";A1$
 50 input "文字列を入力して下さい。";B1$
 60 lset A$=A1$
 70 rset B$=B1$
 80 put #1,1
 90 print "ファイル作成終了"
100 print
110 get #1,1
120 print A$
130 print B$
140 print "ファイル出力終了"
150 close #1
160 end
run
文字列を入力して下さい。? 012
文字列を入力して下さい。? XYZ
ファイル作成終了

012
       XYZ
ファイル出力終了
```

- 1番目に入力した文字列を**lset**命令で左づめにデータを設定し、2番目に入力した文字列を**rset**命令で右づめにデータを設定してファイルに記録します。記録したデータを再び読み出して表示します。

MKI$ （メイク インテジャー ダラー・make integer$）
MKS$ （メイク シングル ダラー・make single$）
MKD$ （メイク ダブル ダラー・make double$）

数値データを内部表現に対応した文字列に変換します。

書 式

mki$(〈整数型データ〉)
mks$(〈単精度型データ〉)
mkd$(〈倍精度型データ〉)

説 明

- 数値データを内部表現に対応した文字列データに変換して返します。結果の文字長は整数型が2、単精度型が5、倍精度型が8バイトになります。
- ランダムファイルへのデータの書き込みで、lset文やrset文でレコードバッファにデータを書き込むとき、そのデータは文字列型である必要があります。このため、数値型データを書き込むときは、mki$、mks$、mkd$を使用して文字列型データに変換します。

サンプル

```
 10 rem --- mki$ mks$ mkd$ ---
 20 open "R",#1,"RNDDT.DAT",32
 30 field #1,2 as A$,2 as B$,5 as C$
 40 for A=1 to 10
 50   lset A$=mki$(A)
 60   lset B$=mki$(A*A)
 70   lset C$=mks$(sqr(A))
 80   put #1,A
 90 next
100 close #i
110 ' このデータは cvi cvs cvd のサンプルで
120 ' 読み出すことができます
```

CVI （コンバート ツー インテジャー・convert to integer）
CVS （コンバート ツー シングル・convert to single）
CVD （コンバート ツー ダブル・convert to double）

内部表現に対応した文字列データを数値データに変換します。

書 式

cvi（〈2文字の文字列式〉）
cvs（〈5文字の文字列式〉）
cvd（〈8文字の文字列式〉）

説 明

- 内部表現に対応した文字列データを数値データに変換して返します。
- それぞれの＜文字列式＞の長さは内部表現に合っている必要があります。
- get＃文でランダムファイルからレコードバッファにデータを読み込むとき、そのデータは文字列型になっています。このため、文字列型に変換して書き込んでいた数値データを数値型データとして読み込むときは、cvi、cvs、cvdを使用して数値型データに変換します。

サンプル

```
 10 rem --- cvi cvs cvd ---
 20 open "R",#1,"RNDDT.DAT",32
 30 field #1,2 as A$,2 as B$,5 as C$
 40 for A=1 to 10
 50   get #1,A
 60   X=cvi(A$)
 70   Y=cvi(B$)
 80   Z=cvs(C$)
 90   print X,Y,Z
100 next
110 close #1
120 ' mki$ mks$ mkd$ のサンプルのデータを
130 ' 読み出せます
```

DEVO$ (デバイス アウトダラー・device out$)

指定の装置へ1レコードのデータを書き込みます。

書　式

devo$"〈装置名〉",〈レコード番号〉,〈文字列型データ〉,〈文字列型データ〉

＜装置名＞…**FDn:, HDm:, MEM:**

nは1～4，mは1（物理番号）

説　明

- ランダムアクセスが可能な装置へレコード単位でデータを書き込みます。書き込み位置の指定は、＜レコード番号＞により指定します。
- ＜装置名＞には、**FDn:, HDm:, MEM:**があります。なお、この場合ハードディスクドライブは物理番号指定となります。
  ＜レコード番号＞は0～65535の値で指定します。
- 書き込むデータは128バイトの文字列型変数2つで指定します。
- この命令はファイル名で指定するようなBASICで管理された書き込みではなく、直接装置内の記録をアクセスします。不用意に使用するとファイルの内容が変わってしまったり、ファイル管理をしている部分を破壊するおそれがありますので注意が必要です。（特にファイル管理部分が破壊された場合、そのときは見かけ上正常でもあとで記録された容量が増えてきたときなどに異常が発生することがあります。）

参　考

devi$ …… 1レコードの読み出し。(335ページ)

DEVI$ (デバイス インプットダラー・device input$)

指定の装置より1レコードのデータを読み出します。

書　式

devi$"〈装置名〉",〈レコード番号〉,〈文字列型変数〉,〈文字列型変数〉

＜装置名＞…FDn:, HDm:, MEM:

nは1～4, mは1（物理番号）

省略形

dev.

説　明

- ランダムアクセスが可能な装置からレコード単位でデータを読み込みます。読み出し位置の指定は、＜レコード番号＞により指定します。
- ＜装置名＞には、FDn:, HDm:, MEM:があります。なお、この場合ハードディスクドライブは物理番号指定となります。
- ＜レコード番号＞は0～65535の値で指定します。
- 読み出したデータは2つの文字列型変数に128バイトずつ代入されます。

参　考

devo$ …… 1レコードの書き込み。(334ページ)

サンプル

```
10 rem --- devi$ ---
20 rem - disk dump -
30 open "o",#1,"CRT1:CG"
40 input "レコード番号  ";R
50 devi$ "FD1:",R,A$,B$
60 for A=1 to 1
70   for B=0 to 127 step 16
80     X$=mid$(A$,B+1,16):Y$=ascchr$(X$)
90     print right$("0000"+hex$(A*128+B),4)+" :";
100    for C=1 to 32 step 2
110      print mid$(Y$,C,2)+" ";
120    next
130    print #1,X$;
140    print
150  next
160  A$=B$
170 next
180 close #1
```

ATTR$ (アトリビュート ダラー・attribute$)

ファイルの属性を調べます。

書 式

attr$(〈ファイル指定〉) (1)

attr$(〔#〕〈ファイル番号〉) (2)

省略形

att.

説 明

- 指定されたファイルの属性を調べる関数で、結果は文字列型データとなります。
- ファイルの属性に応じて、次のような結果となります。
  "P"：指定されたファイルがプロテクト(書き換え禁止)状態である。
  " "(1個のスペース)：指定されたファイルは書き換え可能である。
  "E"：指定されたファイルが保護形式である。
  Null(空文字)：指定されたファイルが存在しない。
- <ファイル指定>は次の書式の文字列型データで指定します。
  "〔<装置名>〕〔<ディレクトリパス>〕<ファイル名>"
- <装置名>には、フロッピーディスクドライブ、ハードディスクドライブ、メモリディスクがあります。
  それぞれ論理装置名(**A:, B:, C:～J:, K:, L:, M:**)または物理装置名(**FDn:, HDm:, MEM:**)で指定します。
- <ディレクトリパス>は階層ディレクトリのパス(道すじ)を指定します。
- <ディレクトリパス>の指定を省略すると、**chdir**コマンドで設定されているカレントディレクトリ内でのファイル指定になります。
- <ファイル名>は、拡張子を付けて指定します。<ファイル名>および拡張子を省略することはできません。
- ファイル番号は**open**されているファイル番号を指定します。
  指定された<ファイル番号>のファイルが**open**されていない場合はエラーが発生します。
- プロテクト状態の変更は**set**文を使用します。(**set**文参照)
  保護形式を解除することはできません。

参 考

set …… ファイルのプロテクトを設定する。(311ページ)

save …… ファイルを保護形式で記録する。(299ページ)

サンプル

```
 10 rem --- ATTR$ ---
 20 print "'ATTRDUM.BAS'の属性を表示します。"
 30 gosub *CHECK
 40 print
 50 print "'ATTRDUM.BAS'にプロテクトをかけます。"
 60 set "ATTRDUM.BAS","P"
 70 gosub *CHECK
 80 print
 90 print "'ATTRDUM.BAS'のプロテクトをはずします。"
100 set "ATTRDUM.BAS",""
110 gosub *CHECK
120 end
130 *CHECK
140 AT$=attr$("ATTRDUM.BAS")
150 if AT$="" then print "そのファイルはありません。"
160 if AT$=" " then print "このファイルにはプロテクトはかかってません。"
170 if AT$="P" then print "このファイルにはプロテクトがかかってます。"
180 return
run
'ATTR$ dummy'の属性を表示します。
このファイルにはプロテクトはかかってません。

'ATTR$ dummy'にプロテクトをかけます。
このファイルにはプロテクトがかかってます。


'ATTR$ dummy'のプロテクトをはずします。
このファイルにはプロテクトはかかってません。
Ok
```

- このプログラムを実行する前に簡単なプログラムを**"ATTRDUM. BAS"**というファイル名で記録してください。
- ファイルにプロテクトをかけたりはずしたりしながら、**attr$**でファイルの属性を調べます。

EOF （エンド オブ ファイル・end of file）

ファイルの終りであるかどうかを調べます。

書 式　eof([#]〈ファイル番号〉)

説 明

- ＜ファイル番号＞でオープンされているファイルのデータを最後まで読み取ったかどうかを調べる関数です。
- 結果は数値でファイルエンドであれば-1、ファイルエンドでなければ0を返します。
- オープンしたファイルの装置名が**フロッピーディスク,ハードディスク,メモリディスク**でシーケンシャルファイルアクセスファイルの場合は、記録された最後のデータを読み取ったときにこの関数の値が-1になります。
  ランダムアクセスファイルの場合は、記録されている最大レコード番号以上のレコード番号で読み取ったときにこの関数の値が-1になります。
- 装置名が**RS-232Cインターフェイス**の場合は、**init "COMn:"**文で指定されたエンドコードを読み出したときに、-1になります。

例

```
if eof(#1) then close #1
```

…… もしファイル番号1がファイルの終りであればそのファイル処理を終了します。

LOF (エル オー エフ・length of a file)

ファイルの大きさを求めます。

書 式 lof(〔#〕〈ファイル番号〉)

説 明

- ＜ファイル番号＞でオープンされているファイルの大きさを求める関数です。
- 結果はシーケンシャルアクセスファイルの場合は使用バイト数、ランダムアクセスファイルの場合は使用している最大のレコード番号になります。
- 装置名が**RS-232Cインターフェイス**の場合は、バッファの残量をバイト数で返します。
- 装置名が**キーボード**の場合は、キーバッファの空き容量をバイト数で返します。

サンプル

```
 10 rem --- LOF (RND) ---
 20 open "R",#1,"RNDFIL.DAT"
 30 MAXREC=lof(1)
 40 print
 50 print "最大レコードＮｏ．＝";MAXREC
 60 close #1
 70 end
run

最大レコードＮｏ．＝ 3
Ok
```

- **field**命令のサンプルで作成したランダムファイルを**open**した後、**lof**でそのファイルを使っている最大レコード番号を表示します。

```
 10 rem --- LOF (SEQ) ---
 20 open "I",#1,"SEQDUM.DAT"
 30 MAXBYT=lof(1)
 40 print
 50 print "使用バイト数＝";MAXBYT
 60 close #1
 70 end
Ok
run

使用バイト数＝ 90
Ok
```

- **open**命令で作成したシーケンシャルファイルを**open**した後、**lof**でそのファイルで使っているバイト数を表示します。

LOC (エル オー シー・location counter of a file)

ファイル中での論理的な現在位置を求めます。

書 式

loc([#]〈ファイル番号〉)

説 明

- <ファイル番号>でオープンされているファイルに対して、現在どの位置をアクセスしているかを求める関数です。ファイルの先頭からの位置を返します。
- シーケンシャルアクセスファイルの場合は、そのファイルがオープンされてから読み書きされたレコード数、ランダムアクセスファイルの場合は最後に読み書きされたレコード番号を返します。
- 装置名が**RS-232Cインターフェイス**の場合は、入力バッファ中に残っている文字数をバイト数で返します。
- 装置名が**キーボード**の場合は、キーバッファ内に残っている文字数をバイト数で返します。

サンプル

```
 10 rem --- LOC (RND) ---
 20 open "R",#1,"RNDFIL.DAT"
 30 field #1,20 as A$,5 as B$,5 as C$
 40 print "  商品名                              仕入値      売値 "
 50 print "=============================================="
 60 while not eof(1)
 70   REC=loc(1)+1
 80   get #1,REC
 90   B=cvs(B$):C=cvs(C$)
100   print "ﾚｺｰﾄﾞ No.=";REC
110   print A$,B,C
120   print "----------------------------------------------"
130 wend
140 close #1
150 end
run
 商品名                              仕入値      売値
==============================================
ﾚｺｰﾄﾞ No.= 1
ﾄﾏﾄ                                 20          35
----------------------------------------------
ﾚｺｰﾄﾞ No.= 2
ｷｬﾍﾞﾂ                               150         250
----------------------------------------------
Ok
```

- **field**命令のサンプルで作成したランダムファイルを**open**した後、**loc**で現在のレコード番号とその内容を表示します。

```
  10 rem --- LOC (SEQ) ---
  20 open "I",#1,SEQDUM.DAT"
  30 print "氏 名              年令   性別   趣 味"
  40 print "=================================================="
  50 while not eof(1)
  60   SEC=loc(1)
  70   input #1,A$,B$,C$,D$
  80   print "セクタ数=";SEC
  90   print using "&                  &&      &&    &&                &",A
$,B$,C$,D$
 100   print "--------------------------------------------------"
 110 wend
 120 close #1
 130 end
run
氏 名              年令   性別   趣 味
==================================================
セクタ数= 1
Ｋｅｎｊｉ          １８歳  男性  オートバイ，パソコン
--------------------------------------------------
セクタ数= 1
Ｙａｓｕｏ          ２４歳  男性  ゴルフ，スキー
--------------------------------------------------
Ok
```

- **open**のサンプルで作成したシーケンシャルファイルを**open**した後、**loc**で現在のセクタ数とその内容を表示します。

DEVF （デバイス フリー・device free）

ファイル装置の空き容量を求めます。

書 式

devf("〈装置名〉")

＜装置名＞…A:, B:, C:～J:, K:, L:, M:

または、FDn:, HDm:, MEM:

nは1～4, mは1～8

説 明

- 外部記憶装置の未使用の記憶容量をバイト単位で求める関数です。
- ＜装置名＞には、フロッピーディスクドライブ、ハードディスクドライブ、メモリディスクがあります。
  それぞれ論理装置名（A:, B:, C:～J:, K:, L:, M:）または物理装置名（FDn:, HDm:, MEM:）で指定します。
- ＜装置名＞を""（空文字）としたときは、現在の装置（カレントドライブ）が設定されます。

例

```
print devf("A:")
```

…… ドライブAにセットしているフロッピーディスクの空き容量を表示します。

INPUT$（インプットダラー）

文字長を指定してファイルまたはキーボードから入力を行います。

書　式　input$(〈文字長〉[,[#]〈ファイル番号〉])

省略形　i.$

説　明

- open"I"によりオープンされたファイルまたはキーボードより、指定した文字長の文字データを読み取る関数です。
- 〈ファイル番号〉を省略して文字長のみを指定した場合は、キーボードからの入力になります。
  このとき、画面にカーソルが出ますが入力文字は表示しません。また、<文字長>分のデータが入力されるまで待ちます。
  なお、コントロールコードの入力も可能なので、コントロール+Cと入力しても実行は中断されません。
- 指定した<ファイル番号>がキーボードの場合、<ファイル番号>を省略したときと同様になります。

例

```
print input$(5)
```

…… キーボードより5文字のデータを読み取り表示します。

サンプル

```
10 rem --- INPUT$ ---
20 cls
30 input "Password (8ﾓｼﾞ) = ",PASS$
40 cls:print "Retype !":print
50 X$=input$(8)
60 if PASS$=X$ then print "Good ! " else print "Wrong !"
70 end
```

- 8文字のパスワードを入力するといったん消去し、再び入力するよう求めてきます。**input $**により8文字のキー入力があった時点で正誤を判定します。

PWD$ （パスワード ダラー・pathword$）

カレントディレクトリのパスを求めます。

書 式 pwd$ 〔("〈ドライブ名〉")〕

省略形 pw.

説 明

- システム変数としてカレントディレクトリのディレクトリパスを文字列型データで返します。
- ディレクトリパスの長さが64バイト以上になると、データを返すことができずエラーとなります。

  例: ¥DRC1 ¥DRC2 ¥DRC3

  5バイト＋5バイト＋5バイト （計15バイト）

  （1階層）
- <ドライブ名>を指定すると、現在の装置(カレントドライブ)以外のカレントディレクトリのディレクトリパスを求めることができます。

  なお、<ドライブ名>を""としたとき、または省略したときはカレントドライブが指定されます。

例

```
print pwd$
```

…… カレントドライブのカレントディレクトリのディレクトリパスを表示します。

```
print pwd$("B:")
```

…… ドライブBのカレントディレクトリのディレクトリパスを表示します。

参 考 chdir …… カレントディレクトリの変更。(289ページ)

ABS （アブソリュート・absolute）

絶対値を求めます。

書　式　abs(〈数式〉)

省略形　ab.

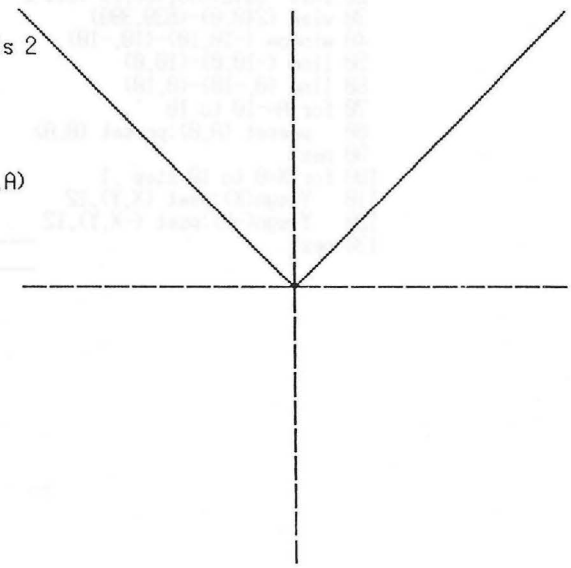
説　明

- ＜数式＞の絶対値を返します。値の符号が正の場合はそのまま、負の場合は符号が除かれた値になります。
- ＜数式＞をXとすると　|X|　の計算になります。
- 結果の型は＜数式＞の型と同じになります。ただし、整数型の−32768の絶対値を求めた場合には、単精度型になります。

例　A=abs(−123)　……　変数Aには-123の絶対値123が代入されます。

サンプル

```
10 rem --- ABS(X) ---
20 init "CRT2:640,400,16":cls 2
30 view (240,0)-(639,399)
40 window (-10,10)-(10,-10)
50 line (-10,0)-(10,0)
60 line (0,-10)-(0,10)
70 for A=-10 to 10
80   preset (A,0):preset (0,A)
90 next
100 for X=-10 to 10 step .1
110   Y=abs(X)
120   pset (X,Y),12
130 next
```



SGN （サイン・sign）

符号を求めます。

書　式　sgn(〈数式〉)

省略形　sg.

説　明

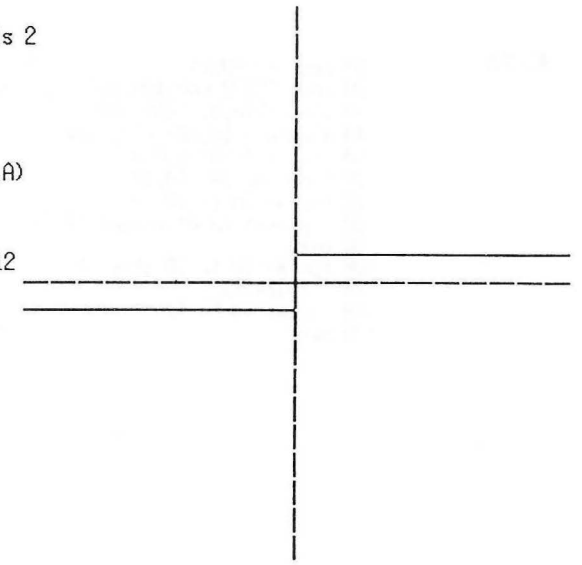
- ＜数式＞の符号を調べ次の値を返します。また、＜数式＞がいずれの型であっても結果の型は整数型になります。

  数式＞0　のとき　　1
  数式＝0　のとき　　0
  数式＜0　のとき　−1

例　A=sgn(−123)　……　変数Ａには-1が代入されます。

サンプル

```
 10 rem --- SGN(X) ---
 20 init "CRT2:640,400,16":cls 2
 30 view (240,0)-(639,399)
 40 window (-10,10)-(10,-10)
 50 line (-10,0)-(10,0)
 60 line (0,-10)-(0,10)
 70 for A=-10 to 10
 80   preset (A,0):preset (0,A)
 90 next
100 for X=0 to 10 step .1
110   Y=sgn(X):pset (X,Y),12
120   Y=sgn(-X):pset (-X,Y),12
130 next
```



INT （インテジャー・integer）

値をこえない最大の整数を求めます。

書　式　int(〈数式〉)

説　明

- ＜数式＞の値をこえない最大の整数を返します。値の符号が正の場合は小数以下を切り捨てるとみなすことができますが、負の場合は絶対値が大きくなります。
- 結果の型は数式の型と同じになります。

例

```
print int(2.5),int(-2.5)
```

…… 2.5は2、-2.5は-3になります。

参　考

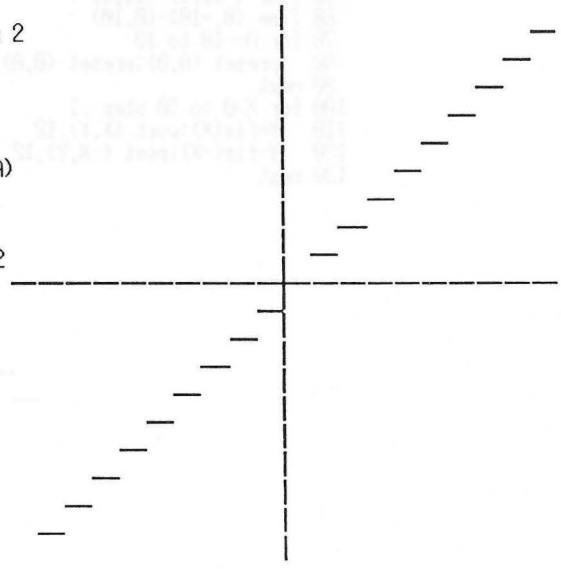
fix …… 整数部を求める。(348ページ)

frac …… 小数部を求める。(349ページ)

cint …… 整数型に変換する。(364ページ)

サンプル

```
 10 rem --- INT(X) ---
 20 init "CRT2:640,400,16":cls 2
 30 view (240,0)-(639,399)
 40 window (-10,10)-(10,-10)
 50 line (-10,0)-(10,0)
 60 line (0,-10)-(0,10)
 70 for A=-10 to 10
 80   preset (A,0):preset (0,A)
 90 next
100 for X=0 to 10 step .1
110   Y=int(X):pset (X,Y),12
120   Y=int(-X):pset (-X,Y),12
130 next
```



FIX （フィックス）

整数部を求めます。

書　式　fix(〈数式〉)

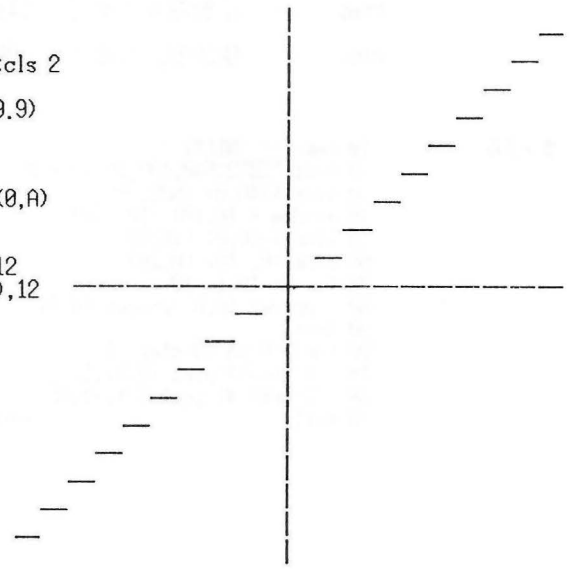
説　明
- ＜数式＞の小数点以下を切り捨て整数部分を取り出した値を返します。
- 結果の型は＜数式＞の型と同じになります。

例　A=fix(123.45) …… 変数Aには、123が代入されます。

参　考
- int …… 値をこえない最大の整数を求める。(347ページ)
- frac …… 小数部を求める。(349ページ)
- cint …… 値を整数型に変換する。(364ページ)

サンプル

```
10 rem --- FIX(X) ---
20 init "CRT2:640,400,16":cls 2
30 view (240,0)-(639,399)
40 window (-10,10)-(9.9,-9.9)
50 line (-10,0)-(10,0)
60 line (0,-10)-(0,10)
70 for A=-10 to 10
80   preset (A,0):preset (0,A)
90 next
100 for X=0 to 10 step .1
110   Y=fix(X):pset (X,Y),12
120   Y=fix(-X):pset (-X,Y),12
130 next
```



FRAC （フラクション・fraction）

小数部を求めます。

書　式　frac(〈数式〉)

省略形　fr.

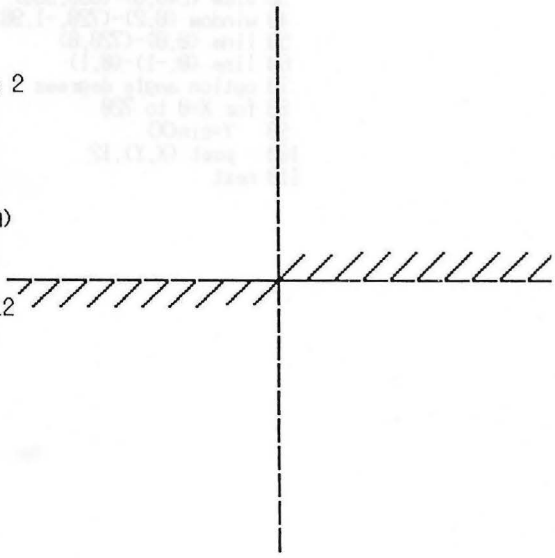
説　明

- ＜数式＞の整数部を取り去り小数点以下の値を取り出した値を返します。
- ＜数式＞の値が倍精度型の場合は結果も倍精度型になり、それ以外の場合、結果は単精度型になります。
- ＜数式＞の値が負の場合は、結果も負になります。
- ＜数式＞の整数部の桁数が多いときは、結果の精度が低くなる場合があります。

例　A=frac(123.125)　……　変数Ａには0.125が代入されます。

サンプル

```
10 rem --- FRAC(X) ---
20 init "CRT2:640,400,16":cls 2
30 view (240,0)-(639,399)
40 window (-10,10)-(10,-10)
50 line (-10,0)-(10,0)
60 line (0,-10)-(0,10)
70 for A=-10 to 10
80   preset (A,0):preset (0,A)
90 next
100 for X=0 to 10 step .1
110   Y=frac(X):pset (X,Y),12
120   Y=frac(-X):pset (-X,Y),12
130 next
```



SIN （サイン・sine）

正弦を求めます。

書　式　sin(〈数式〉)

説　明

- ＜数式＞を角度として正弦(sin X)を返します。角度単位はラジアンまたは度で指定することができ、option angle文で設定された方の角度単位になります。
- ＜数式＞の値が倍精度型の場合は結果も倍精度型になり、それ以外の場合、結果は単精度型になります。

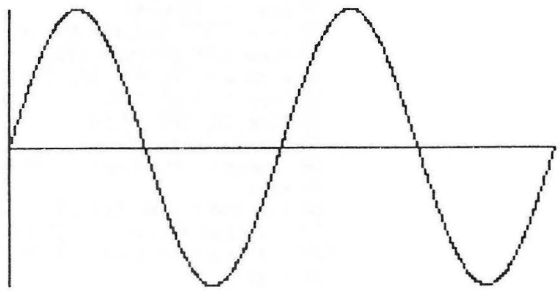
例

```
option angle degrees:print sin(30)
```

…… 30度の正弦、0.5を表示します。

サンプル

```
10 rem --- SIN(X) ---
20 init "CRT2:640,400,16":cls 2
30 view (240,0)-(639,399)
40 window (0,2)-(720,-1.98)
50 line (0,0)-(720,0)
60 line (0,-1)-(0,1)
70 option angle degrees
80 for X=0 to 720
90   Y=sin(X)
100  pset (X,Y),12
110 next
```



COS （コサイン・cosine）

余弦を求めます。

書　式　cos(〈数式〉)

説　明

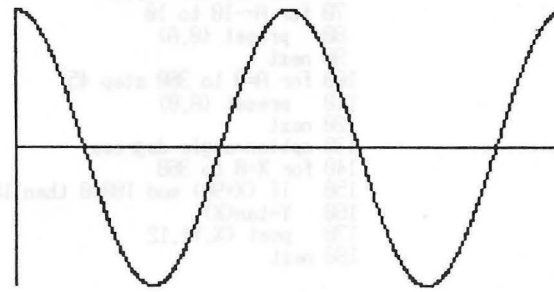
- ＜数式＞を角度として余弦(**cos X**)を返します。角度単位はラジアンまたは度で指定することができ、**option angle**文で設定された方の角度単位になります。
- ＜数式＞の値が倍精度型の場合は結果も倍精度型になり、それ以外の場合、結果は単精度型になります。

例　option angle radians:A=cos(pai(1/3))

…… ラジアンでπ/3(60°)の余弦、0.5を変数Aに代入します。

サンプル

```
10 rem --- COS(X) ---
20 init "CRT2:640,400,16":cls 2
30 view (240,0)-(639,399)
40 window (0,2)-(720,-1.98)
50 line (0,0)-(720,0)
60 line (0,-1)-(0,1)
70 option angle degrees
80 for X=0 to 720
90   Y=cos(X)
100  pset (X,Y),12
110 next
```



TAN （タンジェント・tangent）

正接を求めます。

書 式　tan(〈数式〉)

説 明

- ＜数式＞を角度として正接(tan X)を返します。角度単位はラジアンまたは度で指定することができ、**option angle**文で設定された方の角度単位になります。
- ＜数式＞の値が倍精度型の場合は結果も倍精度型になり、それ以外の場合、結果は単精度型になります。

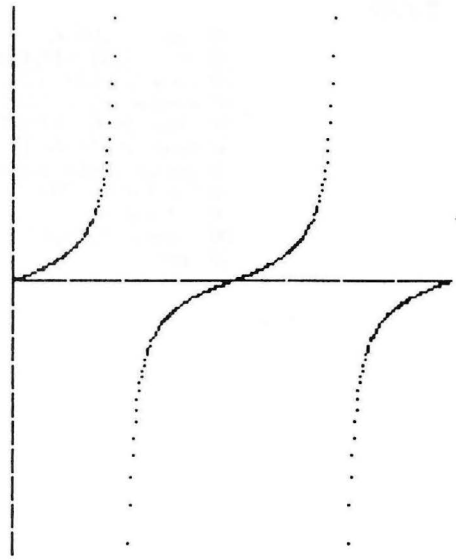
例

```
option angle degrees:print tan(45)
```

…… 45度の正接、1を表示します。

サンプル

```
10 rem ── TAN(X) ──
20 init "CRT2:640,400,16":cls 2
30 view (320,0)-(639,399)
40 window (0,10)-(360,-9.9)
50 line (0,0)-(720,0)
60 line (0,-10)-(0,10)
70 for A=-10 to 10
80   preset (0,A)
90 next
100 for A=0 to 360 step 45
110   preset (A,0)
120 next
130 option angle degrees
140 for X=0 to 360
150   if (X+90) mod 180=0 then 180
160   Y=tan(X)
170   pset (X,Y),12
180 next
```



ATN（アーク タンジェント・arc tangent）

逆正接を求めます。

書 式　atn(〈数式〉)

説 明

- ＜数式＞を正接として逆正接（arctan X）を返します。角度単位はラジアンまたは度で指定することができ、option angle文で指定された方の角度単位になります。

  結果の範囲　ラジアン　$-\pi/2$ ～ $\pi/2$

  度　$-90°$ ～ $90°$

- ＜数式＞の値が倍精度型の場合は結果も倍精度型になり、それ以外の場合、結果は単精度型になります。

例

```
option angle degrees:print atn(1)
```

…… 正接が1の角度、45を表示します。

サンプル

```
 10 rem --- ATN ---
 20 option angle degrees
 30 init "CRT1:80,25":init "CRT2:640,200,16"
 40 view (0,0)-(220,100)
 50 window (360,50)-(449,-50)
 60 cls 3:pen 0
 70 line (360,50)-(360,-50),7
 80 line (360,0)-(449,0),7
 90 pen 2
100 for T=360 to 449 step .8
110   if T=270 then 170
120   TN=tan(T):AT=atn(T)
130   locate 0,15
140   color 2:print using "tan :##.########    ",TN;
150   color 5:print using "atn :##.########",AT
160   gosub *GRAPH
170 next
180 pen 0:color 7
190 end
200 *GRAPH
210 pset (T,TN),10:pset (T,AT*1000-89840),13
220 return
```

```
tan :47.73929340    atn :89.87233580
```

SQR（スクェア ルート・square root）

平方根を求めます。

書 式　sqr(〈数式〉)

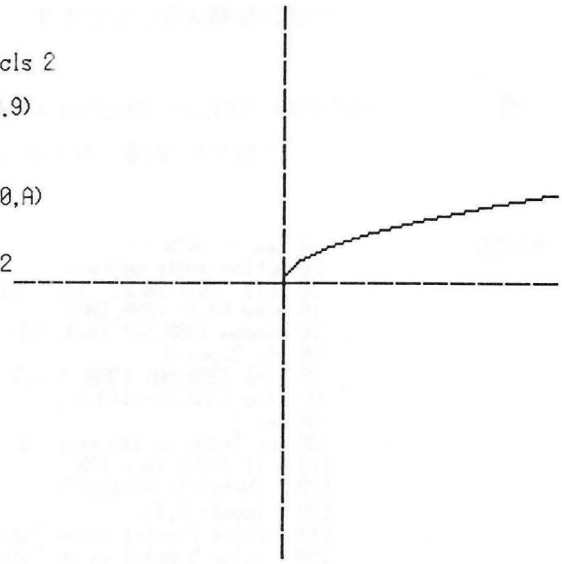
説 明

- ＜数式＞の平方根を返します。値が負の場合はエラーになります。
- ＜数式＞の値が倍精度型の場合は結果も倍精度型になり、それ以外の場合、結果は単精度型になります。

例　print sqr(5#) …… 5の平方根を倍精度で表示します。

サンプル

```
10 rem ── SQR(X) ──
20 init "CRT2:640,400,16":cls 2
30 view (240,0)-(639,399)
40 window (-10,10)-(9.9,-9.9)
50 line (-10,0)-(10,0)
60 line (0,-10)-(0,10)
70 for A=-10 to 10
80   preset (A,0):preset (0,A)
90 next
100 for X=0 to 10 step .1
110   Y=sqr(X):pset (X,Y),12
120 next
```



EXP（イクスポネンシャル・exponential）

指数関数を求めます。

書　式

exp(〈数式〉)

説　明

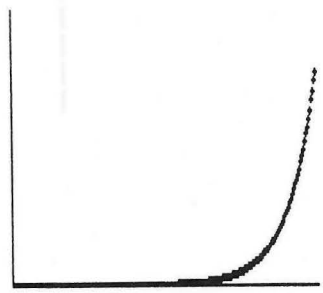
- eに対する指数関数(eの＜数式＞乗)の値を返します。
- ＜数式＞の最大値は約88.02969です。
- ＜数式＞の値が倍精度型の場合は結果も倍精度型になり、それ以外の場合、結果は単精度型になります。

例

```
A=exp(2)
```

サンプル

```
 10 rem --- EXP ---
 20 option angle degrees
 30 init "crt1:80,25":init "CRT2:640,200,16"
 40 view (0,0)-(220,100)
 50 window (1,200000)-(12,0)
 60 cls 3:pen 0
 70 line (1,0)-(12,0),7
 80 line (1,0)-(1,200000),7
 90 pen 2:color 2
100 for X=1 to 12 step .05
110   N=exp(X)
120   locate 0,15
130   print using " EXP : ######.###",N
140   gosub *PSET
150 next
160 pen 0:color 7
170 end
180 *PSET
190 pset (X,N),2
200 return
```



```
EXP :  154817.162
```

LOG （ロガリズム・logarithm）

自然対数を求めます。

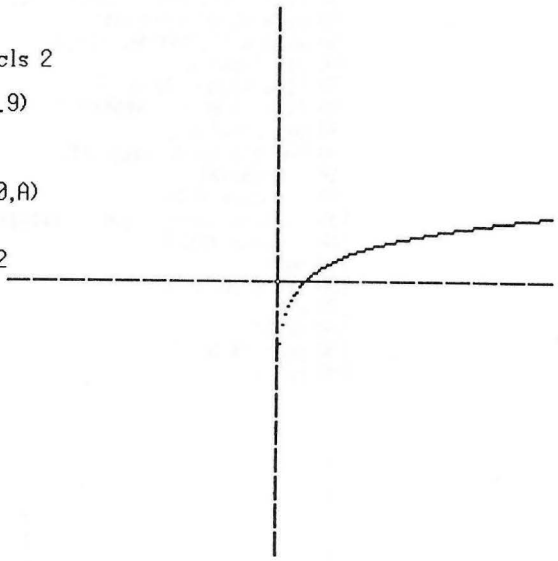
書　式　log(〈数式〉)

説　明
- ＜数式＞の自然対数値(log eX)を返します。
- ＜数式＞の値が倍精度型の場合は結果も倍精度型になり、それ以外の場合、結果は単精度型になります。

例
```
print log(2)
```

参　考☞　log10 …… 常用対数関数 (357ページ)

サンプル
```
 10 rem --- LOG(X) ---
 20 init "CRT2:640,400,16":cls 2
 30 view (240,0)-(639,399)
 40 window (-10,10)-(9.9,-9.9)
 50 line (-10,0)-(10,0)
 60 line (0,-10)-(0,10)
 70 for A=-10 to 10
 80   preset (A,0):preset (0,A)
 90 next
100 for X=.1 to 10 step .1
110   Y=log(X):pset (X,Y),12
120 next
```



LOG10 （ログ 10・logarithm 10）

常用対数を求めます。

書　式　log10(〈数式〉)

説　明
- ＜数式＞の常用対数値(log $_{10}$X)を返します。
- ＜数式＞の値が倍精度型の場合は結果も倍精度型になり、それ以外の場合、結果は単精度型になります。

例

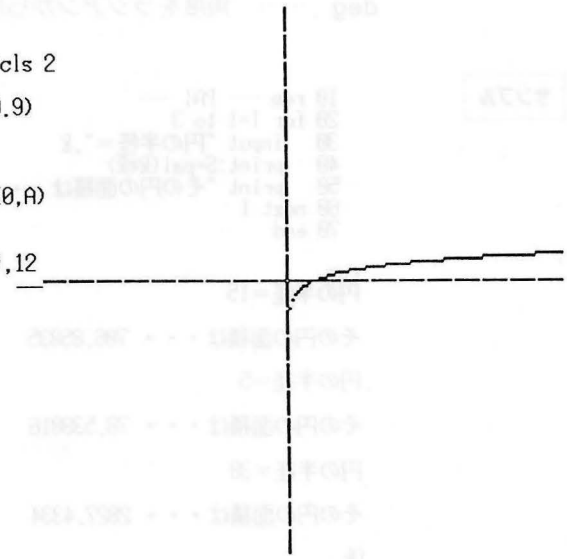
```
print log10(1000)
```

…… 1000は10^3となりますので、3と表示します。

参　考☞　log …… 自然対数関数。(356ページ)

サンプル

```
10 rem --- LOG10(X) ---
20 init "CRT2:640,400,16":cls 2
30 view (240,0)-(639,399)
40 window (-10,10)-(9.9,-9.9)
50 line (-10,0)-(10,0)
60 line (0,-10)-(0,10)
70 for A=-10 to 10
80   preset (A,0):preset (0,A)
90 next
100 for X=.1 to 10 step .1
110   Y=log10(X):pset (X,Y),12
120 next
```



PAI（パイ・π）

円周率を求めます。

書　式　pai〔(〈数式〉)〕

説　明

- 円周率の＜数式＞倍の値を返します。
- ＜数式＞の値が倍精度型の場合は結果も倍精度型になり、それ以外の場合、結果は単精度型になります。
- 引数を省略してpaiとすると、pai(1)と同じになり、単精度型の定数として利用できます。

例　print pai(2) …… 2πの値を表示します。

参　考　rad …… 角度を度からラジアンに変換する関数。(359ページ)

deg …… 角度をラジアンから度に変換する関数。(360ページ)

サンプル

```
10 rem --- PAI ---
20 for I=1 to 3
30   input "円の半径＝",R
40   print:S=pai(R*R)
50   print "その円の面積は・・・";S:print
60 next I
70 end

run
円の半径＝15

その円の面積は・・・ 706.85835

円の半径＝5

その円の面積は・・・ 78.539816

円の半径＝30

その円の面積は・・・ 2827.4334

Ok
```

RAD （ラジアン・radian）

度からラジアンに変換します。

書　式　rad(〈数式〉)

説　明
- ＜数式＞を度とみなしてラジアンに変換した値を返します。
- ＜数式＞の値が倍精度型の場合は結果も倍精度型になり、それ以外の場合、結果は単精度型になります。

例　print rad(180) …… 180°をラジアンに変換して表示します。

参　考　option angle …… 関数や角度指定を行う命令の角度単位を設定する。(166ページ)

deg …… ラジアンから度に変換する。(360ページ)

サンプル

```
10 rem --- RAD ---
20 cls 3
30 option angle degrees
40 for L=1 to 2
50   for X=90 to 360 step 90
60     gosub *PRINT
70     circle (160,100),99,5,0,X:pause 20
80   next:cls 3
90 next
100 end
110 *PRINT
120 if L=1 then T=X:M$=" 度" else T=rad(X):M$=" ﾗｼﾞｱﾝ"
130 locate 0,0
140 print using "###.##### ",T;
150 print M$
160 return
```

- 度単位で開始角を指定して円を描いた後、ラジアン単位で開始角を指定して円を描きます。

DEG （デグリー・degree）

ラジアンから度に変換します。

書　式　deg(〈数式〉)

説　明
- ＜数式＞をラジアンとみなして度に変換した値を返します。
- ＜数式＞の値が倍精度型の場合は結果も倍精度型になり、それ以外の場合、結果は単精度型になります。

例

```
print deg(pai(2))
```

…… ラジアンの2πを度に変換して表示します。

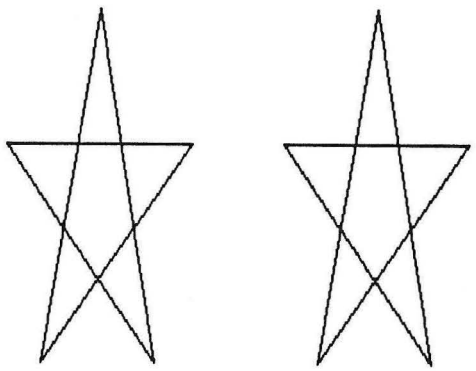
参　考　option angle …… 関数や角度指定を行う命令の角度単位を設定する。(166ページ)

rad …… 度からラジアンに変換する。(356ページ)

サンプル

```
 10 rem --- DEG ---
 20 init "CRT2:640,200,16":cls 3
 30 option angle degrees
 40 for I=1 to 2
 50   gosub *ANG
 60   locate 21*I,1
 70   color I:print M$;X
 80   poly (200*I,100),70,I,144,90,810!
 90 next
100 color 7:end
110 *ANG
120 if I=1 then M$="ﾗｼﾞｱﾝ =":X=rad(144) else M$="度 =":X=144
130 return
```

```
ﾗｼﾞｱﾝ = 2.5132741     度 = 144
```



FAC（ファクトリアル・factorial）

階乗を求めます。

書　式　fac(〈数式〉)

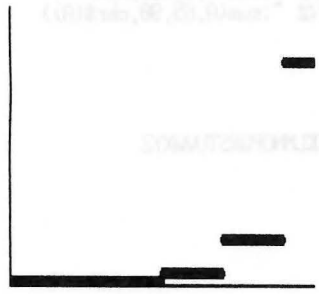
説　明

- ＜数式＞の階乗(X！)を返します。
- ＜数式＞の値は0から33までの整数で与えます。小数部は四捨五入します。
- ＜数式＞の値が倍精度型の場合は結果も倍精度型になり、それ以外の場合、結果は単精度型になります。

例　A=fac(3)　……　3の階乗、1×2×3の値を変数Aに代入します。

サンプル

```
 10 rem --- FAC ---
 20 init "crt1:80,25":init "CRT2:640,200,16":cls 3
 30 view (0,0)-(220,100)
 40 window (0,150)-(5,0)
 50 line (0,0)-(5,0),7
 60 line (0,150)-(0,0),7
 70 pen 3:color 2
 80 for X=0 to 5 step .02
 90   N=fac(X)
100   locate 0,15
110   print using " Fac (#) :",X;
120   print N
130   gosub *PSET
140 next
150 pen 0:color 7
160 end
170 *PSET
180 pset (X,N),13
190 return
```



```
Fac (5) : 120
```

SUM （サメーション・summation）

式の総和を求めます。

書　式　sum(〈変数〉,〈初期値〉,〈最終値〉,〈式〉)

説　明

- ＜変数＞を＜初期値＞から＜最終値＞まで変化させた式の総和を返します。

$$\sum_{\text{変数}=\text{初期値}}^{\text{最終値}} \text{式}$$

- ＜初期値＞と＜最終値＞は整数型で、小数点以下の部分があれば四捨五入します。また実行後の変数値は保証されませんので注意してください。
- ＜式＞は数値および文字式が可能です。
- ＜初期値＞より＜最終値＞が大きい場合には1度だけ式を実行しますが、結果は0またはNull(空文字)になります。

例

```
print sum(A,1,10,A*A)
```

…… 1から10のそれぞれの値を2乗した合計、385を表示します。

```
print sum(A,65,90,chr$(A))
```

…… 文字コードが65～90までのアルファベット、A～Zを並べた文字を表示します。

サンプル

```
10 rem --- SUM ---
20 A=0
30 print "１から１００までの総和は ";sum(A,1,100,A)
40 print "１から１００までの二乗は ";sum(A,1,100,A*A)
50 print "文字コード65から90までは ";sum(A,65,90,chr$(A))
60 end

run
１から１００までの総和は  5050
１から１００までの二乗は  338350
文字コード65から90までは ABCDEFGHIJKLMNOPQRSTUVWXYZ
Ok
```

RND（ランダム・random）

乱数を求めます。

書　式　rnd〔(〈数式〉)〕

説　明

- 0より大き<1より小さな疑似乱数を返します。＜数式＞の値が正の場合は次々に新しい数を発生しますが、0の場合は直前と同じ数、負の場合は初期化します。
- 乱数の発生系列を変更する命令としてrandomize文があります。
- ＜数式＞の値が倍精度型の場合は結果も倍精度型になり、それ以外の場合、結果は単精度型になります。
- ＜数式＞を省略すると、次の乱数を単精度型で発生します。

参　考　randomize　……　乱数の発生系列の変更。(168ページ)

サンプル

```
10 rem --- RND ---
20 init "CRT2:640,200,16":cls 3
30 for I=1 to 10
40   RN=rnd(1):CO=int(RN*14+.5)
50   locate ,5:print "乱数=";
60   print using "#.######",RN
70   locate 16:print "色 =";CO
80   circle (320,100),80,15
90   paint (320,100),CO,15
100  pause 10
110 next
120 end
```

- 乱数によって円の色を変化させます。

CINT (コンバート ツー インテジャー・convert to integer)

数値を整数型に変換します。

書 式

cint(〈数式〉)

説 明

- ＜数式＞の値を整数型データに変換して返します。
- ＜数式＞の値に小数点以下の数が含まれる場合は、四捨五入します。
- -32768～32767の範囲をこえる場合はエラーになります。

例

```
print cint(200/3)
```

…… 小数点以下を四捨五入して67を表示します。

サンプル

```
 10 rem --- CINT ---
 20 for I=1 to 3
 30   input "数値=",X
 40   XI=int(X) : XC=cint(X)
 50   print "INT(";X;")=";XI,"CINT(";X;")=";XC
 60   print
 70 next
 80 end

run
数値=10.4
INT( 10.4 )= 10     CINT( 10.4 )= 10

数値=20.7
INT( 20.7 )= 20     CINT( 20.7 )= 21

数値=15.5
INT( 15.5 )= 15     CINT( 15.5 )= 16

Ok
```

CSNG (コンバート ツー シングル・convert to single)

数値を単精度型に変換します。

書 式 csng(〈数式〉)

省略形 cs.

説 明

- ＜数式＞の値を単精度実数型データに変換して返します。

例

```
print 2/3#,csng(2/3#)
```

…… 2/3#の結果は、倍精度で0.6666666666666667ですが、単精度型にすると0.66666667になります。

サンプル

```
 10 rem --- CSNG ---
 20 cls:locate 0,6
 30 print "****階乗計算****"
 40 for I=1 to 20
 50   A#=fac(I):B=csng(A#)
 60   locate 0,8
 70   print "倍精度=";A#:print "単精度=";B
 80   pause 10
 90 next
100 end
```

CDBL (コンバート ツー ダブル・convert to double)

数値を倍精度型に変換します。

書 式　cdbl(〈数式〉)

省略形　cd.

説 明

- ＜数式＞の値を倍精度実数型データに変換して返します。
- 単精度実数型のデータを変換した場合の結果は、単精度実数型の精度になります。

サンプル

```
 10 rem --- CDBL ---
 20 cls:locate 0,6
 30 print "＊＊＊＊階乗計算＊＊＊＊"
 40 for I=1 to 20
 50   A#=fac(I):B=csng(A#):C#=cdbl(B)
 60   locate 0,8
 70   print "単精度＝";B:print "倍精度＝";C#
 80   pause 10
 90 next
100 end
```

LEFT$ （レフト ダラー）

文字列の左側から任意長の文字列を取り出します。

書　式　left$(〈文字列式〉,〈文字数〉)

省略形　lef.

説　明
- ＜文字列式＞の左側から＜文字数＞で指定された長さの文字を返します。
- ＜文字数＞は、バイト数での指定になり、0から255までの整数で小数点以下の部分があれば四捨五入します。
- ＜文字数＞の指定が＜文字列式＞の文字長(バイト数)より大きい場合は＜文字列式＞がそのまま結果となり、＜文字数＞が0の場合はNull(空文字)になります。

例　A$=left$("ABCD",2)　……　A$には"AB"が代入されます。

MID$ （ミドル ダラー・middle$）

文字列の途中から任意長の文字列を取り出します。

書 式　mid$(〈文字列式〉,〈開始位置〉[,〈文字数〉])

説 明

- ＜文字列式＞の左から＜開始位置＞番目の文字より＜文字数＞で指定された長さの文字列を返します。
- ＜開始位置＞は1から255、＜文字数＞は0から255までの整数で指定し、小数点以下の部分があれば四捨五入します。いずれの指定も、バイト数での指定になります。
- ＜文字数＞を省略した場合や＜開始位置＞より右側の文字長が＜文字数＞に満たない場合は＜開始位置＞より右側すべてが結果になります。
- ＜文字数＞が0の場合や＜文字列式＞の文字長（バイト数）より＜開始位置＞が大きい場合の結果はNull（空文字）となります。

例　A$=mid$("ABCD",2,2)　……　A$には"BC"が代入されます。

サンプル

```
10 rem --- mid$ ---
20 A$="SHARPpersonalcomputerMZSERIES"
30 print A$
40 print:print mid$(A$,6,8);" ";mid$(A$,14,8)
50 print:print mid$(A$,1,5);" ";mid$(A$,22,2);"-";mid$(A$,24,6)

run
SHARPpersonalcomputerMZSERIES

personal computer

SHARP MZ-SERIES
Ok
```

RIGHT$ （ライト ダラー）

文字列の右側から任意長の文字列を取り出します。

書　式　right$(〈文字列式〉,〈文字数〉)

省略形　ri.

説　明

- ＜文字列式＞の右側から＜文字数＞で指定された長さの文字を返します。
- ＜文字数＞は、バイト数での指定になり、0から255までの整数で行い小数以下の部分があれば四捨五入します。
- ＜文字数＞の指定が＜文字列式＞の文字長(バイト数)より大きい場合は＜文字列式＞がそのまま結果となり、＜文字数＞が0の場合はNull(空文字)になります。

例　A$=right$("ABCD",2)　……　A$には"CD"が代入されます。

サンプル

```
10 rem --- RIGHT$ ---
20 MOJI$="ABCDEFGHIJKLMNOPQRSTU"
30 for I=1 to 21
40 PRIN$=right$(MOJI$,I)
50 print PRIN$
60 next
70 end
run
U
TU
STU
RSTU
QRSTU
PQRSTU
OPQRSTU
NOPQRSTU
MNOPQRSTU
LMNOPQRSTU
KLMNOPQRSTU
JKLMNOPQRSTU
IJKLMNOPQRSTU
HIJKLMNOPQRSTU
GHIJKLMNOPQRSTU
FGHIJKLMNOPQRSTU
EFGHIJKLMNOPQRSTU
DEFGHIJKLMNOPQRSTU
CDEFGHIJKLMNOPQRSTU
BCDEFGHIJKLMNOPQRSTU
ABCDEFGHIJKLMNOPQRSTU
```

ASC （アスキー・ascii）

文字列の最初の文字の文字コードを求めます。

書 式　asc(〈文字列式〉)

説 明

- ＜文字列式＞の最初の文字コードを整数型データとして値を返します。
- kmode 0の場合は0から255の1バイトの範囲になりますが、kmode 1の場合では1バイト目が129～159(&H81～9F)または224～252(&HE0～FC)の場合は全角文字として2バイトコードで結果を返します。

例　A=asc("ABCD"):B=asc("漢字")

…… Aには65、Bには-30017(&H8ABF)が代入されます。

サンプル

```
 10 rem --- ASC ---
 20 A=asc("H")
 30 B=asc("e")
 40 C=asc("l")
 50 D=asc("l")
 60 E=asc("o")
 70 F=asc("!")
 80 print A,chr$(A)
 90 print B,chr$(B)
100 print C,chr$(C)
110 print D,chr$(D)
120 print E,chr$(E)
130 print F,chr$(F)
140 end
run
 72       H
 101      e
 108      l
 108      l
 111      o
 33       !
```

- A～Fまでの各変数に指定した文字列の文字コードを求め、その後元の文字列に戻した結果を表示します。

CHR$ （キャラクタ ダラー・character$）

文字コードにより文字データを求めます。

書 式

chr$(〈文字コード〉[,〈文字コード〉]…)

説 明

- ＜文字コード＞の値をもつ文字列データを返します。
- 通常は0から255までの1バイトの整数で指定しますが、全角文字のコードにも対応できるように2バイトの整数(10進数で-32678～32767、16進数で$0～$FFFF)での指定も可能です。

例

```
print chr$(65)
```

…… "A"を表示します。

```
kmode 1:print chr$(&J3021,&H4142)
```

…… "亜AB"と表示します。

サンプル

```
 10 rem --- CHR$ ---
 20 A$=chr$(72)
 30 B$=chr$(101)
 40 C$=chr$(108)
 50 D$=chr$(108)
 60 E$=chr$(111)
 70 F$=chr$(33)
 80 print A$,asc(A$)
 90 print B$,asc(B$)
100 print C$,asc(C$)
110 print D$,asc(D$)
120 print E$,asc(E$)
130 print F$,asc(F$)
140 end
run
H           72
e           101
l           108
l           108
o           111
!           33
```

- chr$で文字コードを文字列に変換して変数に与え、その文字と対応する文字コードを表示します。

VAL （バリュー・value）

数字の文字列を数値データに変換します。

書　式　val(〈文字列〉)

説　明

- 数字で構成されている文字列を数値データに変換します。
- 変換はプログラム中での定数とほとんど同じですが、数値に変換が不可能な場合は0になります。

例

```
print val("123")  ……  123を表示します。
print val("&H001F")  ……  31を表示します。
```

サンプル

```
10 rem --- VAL ---
20 A$="100" : print "A$=";A$
30 B$="5"   : print "B$=";B$
40 C$="12"  : print "C$=";C$
50 print "文字列として加算すると. . . ";A$+B$+C$
60 print
70 print "数値に変換して加算すれば. . . ";val(A$)+val(B$)+val(C$)
80 end
```

```
RUN
A$=100
B$=5
C$=12
文字列として加算すると. . . 100512

数値に変換して加算すれば. . .  117
```

- 数値をそれぞれ文字列として与え、文字列として加算した場合と、**val**命令により数値に変換した後加算する場合を表示します。

STR$ （ストリング ダラー・string$）

数値データを文字列データに変換します。

書 式　str$(〈数式〉)

説 明

- 数値データを文字列データに変換します。
- 変換はprint文での表示と同じ様式になりますので、有効桁数をこえる場合は指数表示の形になります。
- ＜数式＞の値が正の場合は最初に1個のスペースを付け、負の場合は、最初に負信号(−)を付けます。

例　A$=str$(2*3) ……　A$に" 6"を代入します。

サンプル

```
 10 rem --- STR$  ---
 20 for I=1 to 3
 30   read A
 40   print A;"*";A;"=";A*A
 50   A$=str$(A*A)
 60   print len(A$)-1;" 桁"
 70   print
 80 next
 90 end
100 data 10,100,1000

run
 10 * 10 = 100
 3  桁

 100 * 100 = 10000
 5  桁

 1000 * 1000 = 1000000
 7  桁
```

- 桁数を得るためstr$で演算結果をいったん文字列に変え、その長さを調べます。

LEN（レングス・length）

文字列の文字長を求めます。

書　式　len(〈文字列式〉)

説　明
- 文字列データの文字長(バイト数)を整数型データとして返します。
- 2バイトコードの全角文字は2文字とします。全角文字が混在している場合の文字数を調べる場合はklen関数で行います。

例　A=len("漢字")　……　Aには4が代入されます。

参　考　klen　……　全角文字を含む文字列の文字数を求める関数。(389ページ)

サンプル

```
 10 rem --- LEN ---
 20 print "名前          郵便番号      住所"
 30 print
 40 for I=1 to 3
 50   read NA$,BA$,AD$
 60   print NA$;space$(13-len(NA$));BA$;space$(13-len(BA$));AD$
 70 next I
 80 end
 90 data "松本　康之","１０４","東京都港区"
100 data "坂口　慎一朗","５９０－０２","和泉市青葉台"
110 data "倉橋　徹","５５９","大阪市住之江区"

run
名前          郵便番号      住所

松本　康之    １０４        東京都港区
坂口　慎一朗  ５９０－０２  和泉市青葉台
倉橋　徹      ５５９        大阪市住之江区
Ok
```

- 画面表示で、名前、郵便番号がそれぞれ13桁に満たない場合は、13桁となるようにlenで不足する文字数を求め空白を埋めます。

MIRROR$ （ミラー ダラー）

文字コードの対称コードを求めます。

書 式　mirror$(〈文字列式〉)

省略形　mi.

説 明

- ＜文字列式＞の1バイトごとの文字コードを8ビットの2進数で左右対称にして返します。

サンプル

```
10 rem --- MIRROR$ ---
20 init "CRT2:640,200,16":cls 2
30 A$=cgpat$(&H352) ' "R" (8*16)
40 B$=mirror$(A$)
50 pattern (400,20),-16,A$
60 pattern (440,20),-16,B$
```

SPACE$　（スペース ダラー）
SPC　　（スペース・space）

任意の長さのスペース文字を求めます。

書　式

space$(〈数式〉)
spc(〈数式〉)

省略形

spa.　sp.

説　明

・ ＜数式＞の個数分のスペースを並べた文字列型データを返します。個数は0から255までの範囲で指定します。

例

A$=space$(10)　……　A$に10個分のスペースを代入します。

サンプル

```
10 rem --- SPACE$ ---
20 A$="ABC":B$="DEF"
30 for I=1 to 20 step 2
40   print A$;space$(I);B$
50 next
60 end

run
ABC DEF
ABC   DEF
ABC     DEF
ABC       DEF
ABC         DEF
ABC           DEF
ABC             DEF
ABC               DEF
ABC                 DEF
ABC                   DEF
Ok
```

STRING$ (ストリング ダラー・string$)

文字列の任意の繰り返しを求めます。

書 式

string$(〈数式〉, {〈文字列式〉 / 〈文字コード〉})

省略形

stri.

説 明

- ＜文字列式＞または＜文字コード＞で示される文字列を＜数式＞の個数分並べた文字列として返します。結果の文字数(バイト数)が255までの範囲で文字列と個数を指定できます。

例

```
print string$(3,"AB")  ……  "ABABAB"を表示します。
```

サンプル

```
10 rem --- string$ ---
20 for M=4 to 9
30   print M;"月度",
40   read N
50   print string$(N,"*")
60 next
70 data 15,16,19,17,28,32

run
 4 月度   ***************
 5 月度   ****************
 6 月度   ******************
 7 月度   *****************
 8 月度   ****************************
 9 月度   ********************************
Ok
```

INSTR （インストリング・instring）

文字列中の特定文字をさがし位置を求めます。

書　式　instr([〈開始位置〉,]〈文字列式1〉,〈文字列式2〉)

省略形　ins.

説　明

- ＜文字列式1＞の中から＜文字列式2＞をさがし、見つかった位置をバイト数で返します。見つからないときは0を返します。
- ＜開始位置＞はさがし始める位置をバイト単位で指定します。省略した場合は＜文字列式1＞の最初からさがします。
- ＜文字列式2＞にNull(空文字)を指定すると、＜開始位置＞と同じ値を返します。
- 特定文字のさがし出しは、表示形式で判断をしますのでkmode 1が指定されている場合、左から順に見て2バイトコードとなるものは2バイトをまとめて判断します。したがって1文字ずつの判断を行う場合はkmode 0の状態で行ってください。

例

```
print instr("ABCDEFGHIJ","FG")
```

…… "FG"の先頭位置の6を表示します。

```
 10 rem --- INSTR ---
 20 open "I",#1,"INSTRDUM.DAT"
 30 input #1,A$
 40 close #1
 50 kmode 0
 60 B=instr(A$,"ﾊﾔｶﾜ")
 70 kmode 1
 80 C$=mid$(A$,B,10)
 90 print C$
100 end
run
ﾊﾔｶﾜｼｮｳM35
Ok
```

```
10 rem --- Make INSTR dummy ---
20 open "O",#1,"INSTRDUM.DAT"
30 print #1,"ﾔﾏﾀﾞ ﾒｸﾞﾐF28ﾊﾔｶﾜｼｮｳM35ﾏﾂﾓﾄﾀﾞ ｲｽｽﾞM28"
40 close #1
50 end
```

- "MAKE INSTR dummy"というプログラムを実行し、"INSTRDUM. DAT"というファイルを作成してください。
- ファイルをopenした後、instrで"ハヤカワ"のデータを探し出し表示します。

HEX$ (ヘクサデシマル ダラー・hexadecimal$)

数値データを16進数の文字列に変換します。

書　式　hex$(〈数式〉)

省略形　he.

説　明

- ＜数式＞の値を16進数に変換して文字列型データとして返します。
- ＜数式＞の値は整数型の範囲で-32768から32767まで、結果は"0"から"FFFF"までになります。また、32768から65535までの値で指定することもでき、"8000"から"FFFF"に変換します。

例　print hex$(255) …… "FF"と表示します。

サンプル

```
 10 rem --- HEX$ ---
 20 for I=0 to 20
 30   print using "１０進数＝#####",I;
 40   print "      １６進数＝";hex$(I)
 50 next
 60 end
run
１０進数＝    0      １６進数＝0
１０進数＝    1      １６進数＝1
１０進数＝    2      １６進数＝2
１０進数＝    3      １６進数＝3
１０進数＝    4      １６進数＝4
１０進数＝    5      １６進数＝5
１０進数＝    6      １６進数＝6
１０進数＝    7      １６進数＝7
１０進数＝    8      １６進数＝8
１０進数＝    9      １６進数＝9
１０進数＝   10      １６進数＝A
１０進数＝   11      １６進数＝B
１０進数＝   12      １６進数＝C
１０進数＝   13      １６進数＝D
１０進数＝   14      １６進数＝E
１０進数＝   15      １６進数＝F
１０進数＝   16      １６進数＝10
１０進数＝   17      １６進数＝11
１０進数＝   18      １６進数＝12
１０進数＝   19      １６進数＝13
１０進数＝   20      １６進数＝14
Ok
```

OCT$ （オクタル ダラー・octal$）

数値データを8進数の文字列に変換します。

書　式　oct$(〈数式〉)

省略形　oc.

説　明

- ＜数式＞の値を8進数に変換して文字列データとして返します。
- ＜数式＞の値は整数型の範囲で-32768から32767まで、結果は"0"から"177777"までになります。また、32768から65535までの値で指定でき、"100000"から"177777"に変換します。

例　print oct$(255) …… "377"と表示します。

サンプル

```
10 rem --- OCT$ ---
20 for I=0 to 10
30   print using "10進数=####",I;
40   print "        8進数=";oct$(I)
50 next
60 end

run
10進数=   0        8進数=0
10進数=   1        8進数=1
10進数=   2        8進数=2
10進数=   3        8進数=3
10進数=   4        8進数=4
10進数=   5        8進数=5
10進数=   6        8進数=6
10進数=   7        8進数=7
10進数=   8        8進数=10
10進数=   9        8進数=11
10進数=  10        8進数=12
Ok
```

BIN$ (バイナリ ダラー・binary$)

数値データを2進数の文字列に変換します。

書 式

bin$(〈数式〉)

省略形

bi.

説 明

- ＜数式＞を2進数に変換して文字列型データとして返します。
- ＜数式＞の値は整数型の範囲で-32768から32767まで、結果は"0"から"1111111111111111"までになります。また、32768から65535までの値で指定でき、"1000000000000000"から"1111111111111111"に変換します。

例

`print bin$(255)` …… "11111111"と表示します。

サンプル

```
10 rem --- BIN$ ---
20 for I=1 to 10
30   print using "10進数=###",I;
40   print "        2進数=";bin$(I)
50 next
60 end

run
10進数=  1        2進数=1
10進数=  2        2進数=10
10進数=  3        2進数=11
10進数=  4        2進数=100
10進数=  5        2進数=101
10進数=  6        2進数=110
10進数=  7        2進数=111
10進数=  8        2進数=1000
10進数=  9        2進数=1001
10進数= 10        2進数=1010
Ok
```

HEXCHR$ (ヘクサデシマル キャラクタ ダラー・hexadecimal character$)

16進数表記の文字列を文字コードとして文字列に変換します。

書　式　hexchr$(〈文字列式〉)

省略形　hex.

説　明

- 16進数で表された＜文字列式＞の2文字ずつを文字コードとして1文字のデータに変換し、文字列型データとして返します。
- 16進数表記の文字列の中の空白は句切りとして扱われます。

例　print hexchr$("414243") …… "ABC"と表示します。

サンプル

```
10 rem --- HEXCHR$ ---
20 input "文字列を入れてください？",MOJI$
30 ASCMOJI$=ascchr$(MOJI$)
40 HEXMOJI$=hexchr$(ASCMOJI$)
50 print
60 print "ＡＳＣＣＨＲ＄変換後：";ASCMOJI$
70 print "ＨＥＸＣＨＲ＄変換後：";HEXMOJI$
80 end
```

```
run
文字列を入れてください？SHARP-MZ

ＡＳＣＣＨＲ＄変換後：53484152502D4D5A
ＨＥＸＣＨＲ＄変換後：SHARP-MZ
Ok
```

- 入力された文字をいったん文字コードに変換した後、**hexchr$**で再び文字に戻します。

ASCCHR$ (アスキー キャラクタ ダラー・ascii character$)

文字列の文字コードを16進表記に変換します。

書　式　ascchr$(〈文字列式〉)

省略形　asc.

説　明

- ＜文字列式＞の文字コードを1文字ずつ2桁の16進数に変換し、文字列型データとして返します。
- ＜文字列式＞の長さが127をこえる場合は、結果が256文字以上になりますのでエラーになります。

例　print ascchr$("ABC") …… "414243"と表示します。

サンプル

```
 10 rem --- ASCCHR$ ---
 20 input "文字列を入れて下さい ?",MOJI$
 30 ASCMOJI$=ascchr$(MOJI$)
 40 LN=len(MOJI$)
 50 dim DATA$(LN),PR$(LN)
 60 for I=1 to LN
 70   DATA$(I)=hex$(asc(mid$(MOJI$,I,1)))
 80 next
 90 print:print "ASCCHR$変換後：";ASCMOJI$:print
100 for L=1 to LN
110   PR$(L)=mid$(MOJI$,L,1)
120   print "HEX$(ASC(";mid$(MOJI$,L,1);"))=";DATA$(L)
130 next
140 end
```

```
run
文字列を入れて下さい ?Sharp-MZ

ASCCHR$変換後：53686172702D4D5A

HEX$(ASC(S))=53
HEX$(ASC(h))=68
HEX$(ASC(a))=61
HEX$(ASC(r))=72
HEX$(ASC(p))=70
HEX$(ASC(-))=2D
HEX$(ASC(M))=4D
HEX$(ASC(Z))=5A
Ok
```

- 入力された文字列を**ascchr$**により16進の文字コードに変換します。

KMODE （漢字モード・kanji mode）

漢字モードにします。

書　式

kmode {[0] / 1}

省略形

km.

説　明

- 漢字などの全角文字を扱うためのモード切り替えを行います。指定を**1**とすると漢字モードとなり、指定を省略または**0**とすると漢字モードを解除します。BASICの起動時は**kmode 1**に設定されています。
- 漢字モードのときは文字コードが16進数で**81～9F**、または**E0～FC**の範囲であればそれに続く1バイトと合わせて、2バイトの全角文字として扱われ表示などを行います。このモード指定により実行内容が変化する文字表示以外の命令は次のとおりです。

  コマンド …… **search, replace**
  グラフィック …… **symbol**
  関数 …… **instr, asc**

  kmode 1のとき**width**文、**init "CRT1:"**文や**console**文によるスクロール範囲の初期化時に、最下行がスクロール範囲に含まれなくなります。
- 文字画面が40桁モードのときおよび8×8ドットの文字フォントのときは漢字入力を行えません。

例

```
kmode 1　……　漢字モードにします。
```

AKCNV$ (アスキー 漢字 コンバート ダラー・ascii kanji convert$)

半角文字を全角文字に変換します。

書　式　akcnv$(〈変換文字列〉)

省略形　ak.

説　明

- ＜変換文字列＞中にある半角文字を全角文字に変換した文字列を求める関数です。ただし、変換が行われるのは数字、アルファベット、カナ文字や一部の記号などの全角と半角の相方にある文字のみです。全角文字や変換できない半角文字はそのままになります。

例

```
print akcnv$("ABC") ……　全角文字で"ABC"を表示します。
print akcnv$("ﾌﾟﾘﾝﾀ")
  ……　全角文字で"プリンタ"の4文字を表示します。
```

サンプル

```
10 rem --- AKCNV$ ---
20 input "文字列を入れてください(半角)：",MOJI$
30 HEXKAN$=akcnv$(MOJI$)
40 print
50 print "全角変換後：";HEXKAN$
60 end

run
文字列を入れてください(半角)：SHARP-MZ

全角変換後：ＳＨＡＲＰ－ＭＺ
Ok
```

- 半角で入力された文字を**akcnv$**により全角文字に変換します。

KACNV$ (漢字 アスキー コンバート ダラー・kanji ascii convert$)

全角文字を半角文字に変換します。

書 式 kacnv$(〈文字列〉)

省略形 ka.

説 明

- ＜文字列＞中の全角文字を半角文字に変換した文字列を求める関数です。ただし、変換が行われるのは数字、アルファベット、カナ文字や一部の記号などの全角と半角の相方にある文字のみです。半角文字や変換できない全角文字はそのままになります。

例

```
print kacnv$("ＡＢＣあいうえお")
```

…… 半角文字で"ABCｱｲｳｴｵ"と表示します。

```
print kacnv$("プリンタ")
```

…… 半角文字で"ﾌﾟﾘﾝﾀ"の5文字を表示します。

サンプル

```
10 rem --- KACNV$ ---
20 init "CRT1:"
30 input "文字列を入れて下さい（全角）：",MOJI$
40 HEXKAN$=kacnv$(MOJI$)
50 print
60 print "半角変換後　　：";HEXKAN$
70 end
```

```
文字列を入れて下さい（全角）：ぱそこん

半角変換後　　：ﾊﾟｿｺﾝ
Ok
```

- KACNV$で入力された全角文字を、半角文字に変換して表示します。

JIS$ （ジス ダラー）

シフトJISコードよりJISコードに変換します。

書 式　jis$(〈変換文字列〉)

省略形　j.

説 明

- ＜変換文字列＞の最初の2バイトをシフトJISコードとみなして、4バイトのJIS16進文字列に変換する関数です。
- 16進JISコードに変換できないものはエラーになります。
- JIS16進の文字列は**hexchr$**関数を使用すると2バイトコードに変換できます。
- JIS16進よりシフトJISに変換するには**&J**を付けた定数が利用できます。

例　print jis$(″漢″)　……　"3441"と表示します。

サンプル

```
10 rem ── JIS$ ──
20 cls 3:MOJI$="春夏秋冬"
30 for I=1 to 8 step 2
40   MIDMOJI$=mid$(MOJI$,I,2)
50   JISMOJI$=jis$(MIDMOJI$)
60   print MIDMOJI$;"をＪＩＳコード変換 ->";JISMOJI$
70 next
80 end
```

```
春をＪＩＳコード変換 ->3D55
夏をＪＩＳコード変換 ->3246
秋をＪＩＳコード変換 ->3D29
冬をＪＩＳコード変換 ->455F
Ok
```

- **"春夏秋冬"**という文字列を一字ずつ取り出し、**jis$**により**JIS**コードに変換した後、文字列に変換します。

KTN$ （区点 ダラー・kuten$）

シフトJISコードよりJIS区点コードに変換します。

書　式　ktn$(〈変換文字列〉)

省略形　kt.

説　明

- ＜変換文字列＞の最初の2バイトをシフトJISコードとみなして、4バイトのJIS区点コード文字列に変換する関数です。
- JIS区点コードに変換できないものはエラーになります。
- JIS区点コードよりシフトJISに変換するには**&K**を付けた定数が利用できます。

例　print ktn$("漢字")　……　"2033"と表示します。

サンプル

```
10 rem ── KTN$ ──
20 init "CRT1:"
30 input "文字（全角）を入力してください　->",MOJI$
40 JISMOJI$=hex$(val("&J"+jis$(MOJI$)))
50 KTNCODE$=ktn$(MOJI$)
60 KTNMOJI$=hex$(val("&K"+KTNCODE$))
70 print
80 print "その文字のシフトＪＩＳコード　　:";JISMOJI$
90 print "その文字の　区点コード　　:";KTNCODE$
100 print "その文字の区点->シフトＪＩＳ変換後:";KTNMOJI$
110 end
```

```
文字（全角）を入力してください　->和

その文字のシフトＪＩＳコード　　:9861
その文字の　区点コード　　:4734
その文字の区点->シフトＪＩＳ変換後:9861
Ok
```

- 入力された文字をいったんシフト**JIS**コードに変換した後、**ktn$**により区点コードに変換し、さらにもう一度シフト**JIS**コードに変換します。

KLEN （漢字レングス・kanji length）

全角文字を含む文字列の文字数を求めます。

書　式　klen(〈文字列〉〔,〈機能〉〕)

省略形　kle.

説　明

- 全角文字および半角文字が混在した＜文字列＞の文字数を求める関数です。したがって、全角文字は2バイトで1文字とみなされます。
- ＜機能＞を指定することによって、次のタイプの文字数を求めることができます。
  - **0**　…全文字数を求めます。
  - **1**　…英数カナ文字の文字数を求めます。
  - **2**　…漢字の文字数を求めます。
- ＜機能＞を省略した場合は**0**が設定されます。
- この関数は**kmode**文による指定に関係なく動作します。

例

```
print klen("漢字ｻｰﾏﾙﾌﾟﾘﾝﾀ")
```

…… 11と文字数を表示します。

```
print klen("漢字ｻｰﾏﾙﾌﾟﾘﾝﾀ" 1)
```

… 英数カナ文字数"9"が表示されます。

サンプル

```
 10 rem --- KLEN ---
 20 dim NAME$(5),LENGTH(5)
 30 for I=1 to 5
 40   read NAME$(I)
 50   LENGTH(I)=klen(NAME$(I))
 60   print NAME$(I);"さんの名前の長さ：";LENGTH(I)
 70 next
 80 print
 90 for I=1 to 5
100   for L=I+1 to 5
110     if LENGTH(I)<LENGTH(L) then swap NAME$(I),NAME$(L):swap LENGTH(I),LENG
TH(L)
120   next
130 next
140 for I=1 to 5
150   print "名前の長さが";I;"番目に長い、";NAME$(I);"さん。"
160 next
170 end
180 data "松本康之","倉橋徹","遠山美枝子","水乃小路博幸","谷仁"

run
松本康之さんの名前の長さ： 4
倉橋徹さんの名前の長さ： 3
遠山美枝子さんの名前の長さ： 5
水乃小路博幸さんの名前の長さ： 6
谷仁さんの名前の長さ： 2
```

```
名前の長さが 1 番目に長い、水乃小路博幸さん。
名前の長さが 2 番目に長い、遠山美枝子さん。
名前の長さが 3 番目に長い、松本康之さん。
名前の長さが 4 番目に長い、倉橋敏さん。
名前の長さが 5 番目に長い、谷仁さん。
Ok
```

- **klen**で名前の長さを求めた後、長いもの順に並べ替えます。

KPOS（漢字ポジション・kanji position）

文字列の先頭より指定文字までのバイト数を求めます。

書　式　kpos(〈文字列〉,〈位置〉)

省略形　kp.

説　明

- ＜文字列＞の中の＜位置＞番目の文字が、先頭から何バイト目になるかを求める関数です。半角文字は1バイトで1文字、全角文字は2バイトで1文字とみなされます。
- ＜位置＞が＜文字列＞の文字数より大きい場合は結果が**0**になります。
- この関数は**kmode**文による指定に関係なく動作します。

例

```
print kpos("11月3日は文化の日",7)
```

…… 7文字目の"文"は10バイト目になります。

サンプル

```
10 rem --- KPOS ---
20 cls
30 MOJI$="ｋｐｏｓ関数は指定文字が何バイト目にあるかを求めます"
40 locate 0,0 :print MOJI$
50 locate 0,3:input "何文字目ですか ",N
60 P=kpos(MOJI$,N)
70 color@ (0,0)-(P-1,0),2
80 locate 0,5 : print N;"文字目の文字は";P;"バイト目です。"
90 end
```

- 表示された文字列の任意の文字を指定することにより、**kpos**でその文字が先頭から何バイト目にあたるかを表示します。

KEXT$（漢字テキストダラー・kanji text$）

文字列中より英数カナ文字または漢字を取り出します。

書　式　kext$(〈文字列〉，〈機能〉)

省略形　kex.

説　明

- 英数カナ文字または漢字のいずれかを＜文字列＞で指定した文字列より取り出します。
- ＜機能＞は**0**または**1**で指定します。
  - **0**…　文字列中より英数カナ文字だけを取り出します。
    取り出される文字は文字コードの0～7FHとA0H～DFHおよびFFHです。
  - **1**…　文字列中より漢字だけを取り出します。
    取り出される漢字はシフトJISコードの文字です。
- ＜機能＞で指定したタイプの文字が無い場合は、”　”(空文字)が返されます。
- この関数は**kmode**文の指定に関係なく動作します。

例

```
print kext$("ABC漢字DE日本",1)
```

…　”漢字日本”と表示されます。

KINPUT（漢字インプット・kanji input）

キーボードから漢字を入力し変数に代入します。

書　式　kinput　〈変数名〉[,〈変数名〉]

省略形　k.

説　明

- この命令が実行されると自動的に漢字入力モードに入り、キーボードからの入力待ちとなります。
- **input**文との違いは、自動的に漢字入力モードに入る点と実行終了時漢字入力モードから抜ける点です。

例

```
kinput A$
```

KINSTR（漢字インストリング・kanji instring）

文字列中より特定の文字列をさがし位置を求めます。

書　式　kinstr　([〈開始位置〉,] 〈文字列１〉, 〈文字列２〉)

省略形　kins.

説　明

- ＜文字列１＞で指定された文字列中より＜文字列２＞で指定された文字列をさがし、その位置を文字数で返します。見つからないときは**0**を返します。
- ＜開始位置＞はさがし始める位置を文字単位で指定します。開始位置は１～255の値で指定します。
- ＜開始位置＞を省略すると、＜文字列１＞の先頭からさがします。なお、＜開始位置＞が＜文字列１＞より大きい場合は**0**を返します。
- ＜文字列２＞が"　"(空文字)の場合は、＜開始位置＞と同じ値を返します。
- この関数は**kmode**文による指定に関係なく動作します。

例

```
print kinstr("123漢字456","漢字")
```
… "漢字"は４文字目になります。

```
print kinstr(2,"123漢字日本語","日本語")
```
… ２文字目から"日本語"は６文字目になります。

KMID$（漢字ミドルダラー・kanji middle$）

漢字を含む文字列中より任意の長さの文字列を取り出します。

書　式　kmid$ （〈文字列〉，〈開始位置〉〔，〈文字数〉〕）

省略形　kmi.

説　明

- ＜文字列＞の左から＜開始位置＞番目の文字より＜文字数＞で指定された長さの文字列を返します。
- ＜開始位置＞は 1 ～255、＜文字数＞は 0 ～255の値で指定します。いずれも文字数での指定になります。
- ＜文字数＞を省略した場合や＜開始位置＞より右側の文字数が＜文字数＞に満たない場合は＜開始位置＞より右側すべての文字が返されます。
- ＜文字数＞が **0** の場合や＜文字列＞の文字数より＜開始位置＞が大きい場合結果は**""**(空文字)となります。
- この関数は**kmode**文の指定に関係なく動作します。

例

```
print kmid$("kmid$関数",5,3)
```

…　5 文字目から"$関数"の 3 文字が表示されます。

KNJ$（漢字ダラー・kanji$）

漢字コード文字列を漢字に変換します。

書　式　knj$ （〈文字列〉）

説　明

- ＜文字列＞の先頭から4桁をシフトJISコードとみなし、漢字1字に変換して返します。
- ＜文字列＞の内容が次のような場合、エラーとなります。
  - ・文字列が3桁以下の場合。
  - ・取り出した4桁の文字列の中に0～9，A～F以外の文字を含む場合。
  - ・取り出した4桁の文字の組み合わせがシフトJISコードの範囲外の場合。

例

```
print knj$("889F")
```

… "亜"と表示します。

KTYPE（漢字タイプ・kanji type）

漢字を含む文字列中の特定位置の文字タイプを求めます。

書　式　ktype　(〈文字列〉, 〈位置〉)

省略形　kty.

説　明

- ＜文字列＞の中の＜位置＞番目の文字のタイプを求めます。
  結果は次の値で返されます。
  0　…英語カナ文字
  　　(グラフィック文字以外の半角文字)
  1　…漢字
- ＜文字列＞が空文字の場合、エラーとなります。
- ＜位置＞は１以上の値を指定します。
- この関数は**kmode**文の指定に関係なく動作します。

例

```
print ktype(″漢字ﾀｲﾌﾟ″,2)
```

…　２文字目"字"に対して１と表示されます。

ON ERROR GOTO （オン エラー ゴーツー）

エラー処理の宣言を行います。

書　式　on error goto 〈エラー処理開始行〉

省略形　o. err. g.

説　明

- プログラム実行中にエラーが発生した場合、エラーに対応する処理を行う行を＜エラー処理開始行＞として宣言します。
- ＜エラー処理開始行＞は、行番号またはラベルで指定できます。
- この命令で宣言を行っておくと、エラーが発生した場合コマンド待ちに戻らずに、システム変数のerrとerlにエラー番号とエラー発生行の情報を設定して、＜エラー処理開始行＞に実行を移します。
- エラー処理内では、errとerlの値により処理を行い、resume文によりエラー処理を終了し、もとの処理に戻ります。
- エラーが発生して定義された処理行へ実行を移したあと、resume文を実行するまでにエラーが発生すると、エラー処理は行われずにエラーメッセージを出して命令待ちに戻ります。
- プログラム中に複数の宣言がある場合は、エラーの発生する直前に実行されている宣言の＜エラー処理開始行＞が有効となります。
- ＜エラー処理開始行＞を0とするとエラー処理の宣言を解除します。

例

```
on error goto *ERROR
```

…… エラーが発生した場合ラベル名＊ERRORの行からエラー処理を開始するように宣言します。

参　考　resume …… エラー処理の終了。(402ページ)

err …… エラー番号を求めるシステム変数。(400ページ)

erl …… エラー発生行を求めるシステム変数。(401ページ)

サンプル

```
 10 rem --- ON ERROR GOTO ---
 20 on error goto *ER
 30 input "A=",A:input "B=",B
 40 print "A/B =";A/B
 50 print:goto 30
 60 end
 70 *ER
 80 if err=2 then color 2:print:print "0では割れません！"
 90 print:color 7
100 resume 30
```

```
run
A=10
B=20
A/B = .5

A=20
B=0
A/B =
0では割れません!

A=
*Break in 30
Ok.
```

- エラーが発生すると、**on error goto**命令によりエラー処理行(**＊ER**)に移り、メッセージを表示します。

ERR （エラー ナンバー・error number）

エラー番号を求めます。

書 式

err

説 明

- プログラムの実行中にエラーが発生した場合に、エラーコードが設定されるシステム変数です。
- 直接実行モードでエラーが発生した場合は、errに値を設定しません。
- エラーコードについては付録の「エラーコード表」を参照してください。

例

```
if err=1 then print "文法誤りです。"
```

…… もしエラーコードが1であれば "**文法誤りです**"と表示します。

参 考

on error go to …… エラー処理の宣言。(398ページ)

erl …… エラー発生行を求める。(401ページ)

サンプル

```
 10 rem --- ERR/ERL ---
 20 on error goto *ER
 30 input "A=",A:input "B=",B
 40 print "A/B =";A/B
 50 print:goto 30
 60 end
 70 *ER
 80 color 2:print:print "エラーが発生しました！"
 90 color 3:print "エラーＮｏ．";err
100 color 6:print "エラー発生行番号 ";erl
110 color 7:print
120 resume 30

run
A=10
B=0
A/B =
エラーが発生しました！
エラーＮｏ．2
エラー発生行番号 40

A=
*Break in 30
Ok.
```

- エラーが発生すると、errによりそのエラー番号とerlによりそのエラーが発生した行番号を表示します。

ERL （エラー ライン・error line）

エラー発生行を求めます。

書 式　erl

説 明
- プログラムの実行中にエラーが発生した場合に、エラーが発生した行番号が設定されるシステム変数です。
- 直接実行モードでエラーが発生した場合は、erlに値を設定しません。

例
```
if erl=100 then resume next
```
…… もしエラーの発生した行が行番号100であれば、エラーの発生した次の文に戻ります。

参 考☞ on error goto …… エラー処理の宣言。(398ページ)
err …… エラー番号を求める。(400ページ)

サンプル　errのサンプル参照。

RESUME（リジューム）

エラー処理を終了してプログラムを再開します。

書　式

resume [{ next / 〈戻り行〉 }]

省略形

resu.

説　明

- on error goto文により、エラー処理に移り、エラー処理を終了して元のプログラムに戻る場合に使用します。この命令により、エラー時の処理行の再設定を行いますので、この命令以外で元のプログラムへ移った場合は、次にエラーが発生してもエラー処理を行いません。
- 戻る場所は次のように指定できます。

resume	……	エラーの発生した文に戻ります。
resume next	……	エラーの発生した次の文に戻ります。
resume 0	……	プログラムの先頭に戻ります。
resume ＜戻り行＞	……	指定された行に戻ります。

- エラー処理中のエラーは、エラー処理は行われませんが、resume文でエラーが発生(戻り行などの指定の誤りがある場合)した場合は、エラー処理が行われますので注意が必要です。

例

```
if erl=1010 then resume *START
```

…… もしエラーの発生した行が行番号1010であれば＊STARTのラベル名の付いた行から実行を再開します。

参　考☞

on error go to …… エラー処理行の宣言。(398ページ)

err …… エラー番号を求める。(400ページ)

erl …… エラー発生行を求める。(401ページ)

ERROR（エラー）

エラーを発生させます。

書　式　error〈エラー番号〉

省略形　err.

説　明

- エラーを疑似的に発生させ、エラー処理のシミュレーションなどに利用します。
- ＜エラー番号＞は0～255までの整数値で指定し、実行時のこの値をerrに設定します。エラー処理が宣言されていない場合はエラー番号に対応したエラーメッセージを出して実行を中止します。エラーが定義されていない番号の場合、エラーメッセージは"unprintable error"になります。
- 定義されていないエラー処理を利用してユーザー独自のエラーを定義することもできます。
- ＜エラー番号＞に範囲外のデータを指定すると"data error"が発生します。

例

```
100 error 3
```

…… 行番号100でエラーコード3のエラーを発生します。この場合、errに3、erlに100が設定されます。

```
250 if X<0 then error 103
```

…… Xの値が負であれば103番のエラーが発生します。エラー処理の宣言が行われているとerrに103、erlに250をそれぞれ設定してエラー処理の開始行に移ります。エラー処理の中で、このエラーコードに対応する処理を行えばユーザー独自のエラー定義になります。

参　考☞
on error goto …… エラー処理の宣言。(398ページ)
err …… エラー番号を求める。(400ページ)
erl …… エラー発生行を求める。(401ページ)

ON KEY GOSUB (オン キー ゴーツーサブルーチン・on key go to subroutine)

割り込みキーによる割り込み処理行を定義します。

書　式　on key gosub〔〈処理行1〉〕〔,〈処理行2〉〕…

省略形　o. ke. gos.

説　明

- **key**文で指定された割り込みキー(BASIC起動時は**F1～F20**のファンクションキー)を押したり離したりしたときに割り込み処理が行われる開始行を＜処理行＞としてそれぞれのキーに対して定義します。(**key**文参照)
- ＜処理行＞は、行番号またはラベルで指定できます。
- **key**文で指定したキー番号1の割り込みが起ったときに処理する行を＜処理行1＞、キー番号2の割り込みが起ったときに処理する行を＜処理行2＞というように、順次**key**文で指定したキーの番号に合わせて割り込み処理の開始行を指定していきます。
- この命令では割り込み処理行を定義をするだけで、割り込み処理の実行制御はkey on/off/stop文により行います。
- 割り込み処理は、割り込み発生時に＜処理行＞からサブルーチンとして実行しますので、割り込み処理の終了は**return**文になります。
- この命令での定義は、**run**コマンドおよび**end**文の実行により解除されます。
- この命令の実行時に＜処理行＞で指定された行がない場合はエラーになります。

参　考　key on/off/stop　……　キー割り込みの制御。(406ページ)

サンプル

```
 10 rem --- ON KEY GOSUB ---
 20 cls
 30 on key gosub 110,120,130,140,150:key on
 40 print "あなたの行ってみたい都市はどこですか?"
 50 print "F1・・・ニューヨーク      F2・・・パリ      F3・・・ロンドン"
 60 print "F4・・・その他            F5・・・調査終了"
 70 locate 0,5
 80 print "ニューヨーク";NYK;"人   パリ";PAR;"人   ロンドン";LON;"人   その他"
;HOK;"人"
 90 goto 70
100 end
110 NYK=NYK+1 : return
120 PAR=PAR+1 : return
130 LON=LON+1 : return
140 HOK=HOK+1 : return
150 key off
160 return 100
```

```
あなたの行ってみたい都市はどこですか？
Ｆ１・・・ニューヨーク　　Ｆ２・・・パリ　　Ｆ３・・・ロンドン
Ｆ４・・・その他　　Ｆ５・・・調査終了

ニューヨーク ９人　パリ ６人　ロンドン ２人　その他 ８人
Ok
```

- ファンクションキー(**F1～F4**)を押すごとにそれぞれ対応する都市の人数を加算します。
- プログラムの終了は、**F5** を押すことにより行います。

KEY ON/OFF/STOP （キー オン／オフ／ストップ）

キーによる割り込みを制御します。

書　式

key 〔(〈キー番号〉)〕 { on / off / stop }

省略形

ke. o. /of. /s.

説　明

- **key**文で指定されたキーによる割り込み処理を制御します。＜キー番号＞の指定は**1**～**20**の整数値で表し、**key**文で指定されたキー番号に対応します。(BASIC起動時は**F1**～**F20**のファンクションキーに対応します。)＜キー番号＞の指定を省略すると**key**文での指定キーすべてに対する指定となります。
- **on**を指定するとファンクションキーによる割り込みが可能となり、**on key gosub**文で定義されている割り込み処理の開始行から、サブルーチンとして実行します。
- **off**を指定すると割り込み機能は解除され、キーを押しても割り込み処理は行われません。
- **stop**を指定するとキーを押したり離したりしても割り込み処理は行われませんが、割り込みがあったことは記憶されていますので、あとで**on**が指定されたときに、割り込み処理を実行します。ただし、**stop**中にキーを何回押しても、実行されるのはそれぞれのキーに対して１回のみです。
- **on**または**stop**の状態で**input**文や**line input**文による入力時に、指定されたキーを押すと**input**文をぬけ(**on**の場合は割り込み処理を行い)、次の処理に進みます。このときの**input**文の入力変数の値は変化しません。
- **run**コマンドおよび**end**文の実行やエラーの発生、または実行中断時のプログラム変更などにより、このほかの割り込み機能も含めて、すべての割り込み処理の実行制御は**off**になります。
- **on key gosub**文で割り込み処理行が定義されていない場合は、割り込みが発生しても何も処理は行われません。
- **on**または**stop**に指定されているキーは、文字入力には使用できません。

例

```
key (1) on  ……  key文で指定されたキー番号1によるキー割り込みを可能とします。
key off     ……  すべてのファンクションキー割り込みを解除します。
```

参　考☞　**on key gosub** …… キー割り込みの処理行を定義する。(404ページ)

ON STOP GOSUB (オン ストップ ゴーツーサブルーチン・on stop go to subroutine)

シフト ＋ 停止 キーによる割り込み処理行を定義します。

書　式　on stop gosub 〈処理行〉

省略形　o. s. gos.

説　明

- シフト＋停止キーを押したときに、割り込み処理が行われる開始行を＜処理行＞として定義します。
- この命令では割り込み処理行を定義するだけで、割り込み処理の実行を制御するのは、stop on/off/**stop**文により行います。
- ＜処理行＞は、行番号またはラベルで指定できます。
- 割り込み処理は、割り込み発生時に＜処理行＞からサブルーチンとして実行されますので、割り込み処理の終了は**return**文になります。
- この命令での定義は、**run**コマンドおよび**end**文の実行により解除されます。
- この命令の実行時に＜処理行＞で指定された行がない場合はエラーになります。
- この割り込み機能が**on**されると本来のシフト＋停止キーの操作による実行の中断機能はなくなります。この機能はプログラム実行中に不用意にプログラムの実行を中断されないようにするものです。したがって、実行中に何らかの方法で実行を終了するようにプログラミングしておいてください。

例

```
on stop gosub 1000
```
…… 割り込み処理行を行番号1000と定義します。

```
on stop gosub *BREAK
```
…… 割り込み処理行を＊BREAKというラベルで定義します。

参　考　**stop on/off/stop** …… シフト＋停止キーによる割り込みの制御。(409ページ)

サンプル

```
 10 rem --- ON STOP GOSUB ---
 20 cls:on stop gosub *BREAK
 30 stop on
 40 print "シフト＋停止　キーで表示が変わります":pause 1
 50 goto 40
 60 *BREAK
 70 stop off
 80 for I=1 to 20
 90   print "表示が変わりました":pause 1
100 next I
110 return 30
120 end
```

- シフト＋停止キーを押すことによって割り込みが処理され、"シフト＋停止キーで表示が変わります"という表示から"表示が変わりました"という表示に変わります。

STOP ON/OFF/STOP (ストップ オン/オフ/ストップ)

シフト + 停止 キーによる割り込みを制御します。

書 式

stop { on / off / stop }

省略形

s. o. /of. /s.

説 明

- シフト + 停止 キーによる割り込み処理を制御します。
- onを指定するとシフト + 停止 キーによる割り込みが可能になり、シフト + 停止 キーが押されたとき **on stop gosub** 文で定義されている割り込み処理の開始行からサブルーチンとして実行します。
- **off**を指定すると割り込み機能は解除され、キー操作を行っても割り込み処理は行われずプログラムを中断します。
- **stop**を指定するとキー操作を行っても割り込み処理は行われませんが、割り込みがあったことは記憶されていますので、あとでonが指定されたときに割り込み処理を実行します。ただし、stop中に何回割り込みがあっても、実行は1回のみです。
- onまたはstopの状態でinput文やline input文による入力時にこのキー操作をするとinput文をぬけ(onの場合は割り込み処理を行い)、次の処理に進みます。このときのinput文の入力変数の値は変化しません。
- runコマンドおよびend文の実行やエラーの発生、または実行中断時のプログラム変更などにより、このほかの割り込み機能も含めて、すべての割り込み処理の実行制御はoffになります。
- on stop gosub文で割り込み処理行が定義されていない場合は、割り込みが発生しても何も処理は行われません。
- onまたはstop状態となっている場合は、キー操作でのプログラムの中断はできません。

例

```
stop on
```

…… シフト + 停止 キーの操作による割り込みを可能とします。

ON HELP GOSUB (オン ヘルプ ゴーツーサブルーチン・on help go to subroutine)

ヘルプ キーによる割り込み処理行を定義します。

書　式　on help gosub 〈処理行〉

省略形　o. hel. gos.

説　明

- ヘルプ キーを押したときに、割り込み処理が行われる開始行を＜処理行＞として定義します。
- この命令では割り込み処理行を定義するだけで、割り込み処理の実行を制御するのは、**help on/off/stop**文により行います。
- ＜処理行＞は、行番号またはラベルで指定できます。
- 割り込み処理は、割り込み発生時に＜処理行＞からサブルーチンとして実行されますので、割り込み処理の終了は**return**文になります。
- この命令での定義は**run**コマンドおよび**end**文の実行により解除されます。
- この命令の実行時に＜処理行＞で指定された行がない場合はエラーになります。

例

```
on help gosub *HELP
```

…… 割り込み処理行を＊HELPというラベルで定義します。

参　考　**help on/off/stop** …… ヘルプ キーによる割り込み制御。
(412ページ)

サンプル

```
 10 rem --- ON HELP GOSUB ---
 20 on help gosub *HELP
 30 init "CRT1:"
 40 print "全角カタカナで入力してください":print
 50 for I=1 to 4
 60   read ENG$,JAP$
 70   print ENG$;" の和訳は（わからないときは［ヘルプ］キーを押して！）:";
 80   help on:input YAKU$
 90   if JAP$=YAKU$ then print "正解です！" else print "まちがいです！":goto 7
0
100 next
110 end
120 *HELP
130 help off
140 print "答えは  ";JAP$;"  です。"
150 return 100
160 data "ＣＡＴ","ネコ","ＤＯＧ","イヌ","ＣＨＡＩＲ","イス","ＢＯＯＫ","ホン"
```

```
全角カタカナで入力してください

ＣＡＴ の和訳は（わからないときは［ヘルプ］キーを押して！）：？ ネコ
正解です！
ＤＯＧ の和訳は（わからないときは［ヘルプ］キーを押して！）：？ イヌ
正解です！
ＣＨＡＩＲ の和訳は（わからないときは［ヘルプ］キーを押して！）：？
答えは　イス　です。
ＢＯＯＫ の和訳は（わからないときは［ヘルプ］キーを押して！）：？
答えは　ホン　です。
Ok
```

- 英単語の和訳問題で、答えがわからないときは ヘルプ キーを押すと割り込み処理がされ、正解が表示されます。

HELP ON /OFF/STOP (ヘルプ/オン/オフ/ストップ)

[ヘルプ] キーによる割り込みを制御します。

書 式

help { on / off / stop }

省略形

hel. o. /of. /s.

説 明

- [ヘルプ]キーによる割り込み処理を制御します。
- **on**を指定すると[ヘルプ]キーによる割り込みが可能となり、[ヘルプ]キーが押されたとき **on help gusub** 文で定義されている割り込み処理の開始行から、サブルーチンとして実行します。
- **off**を指定すると割り込み機能は解除され、キー操作を行っても割り込み処理は行われません。
- **stop**を指定するとキー操作を行っても割り込み処理は行われませんが、割り込みがあったことは記憶されていますので、あとで**on**が指定されたときに割り込み処理を実行します。ただし、**stop**中に何回割り込みがあっても、実行は1回のみです。
- **on**または**stop**の状態で**input**文や**line input**文による入力時にこのキー操作をすると**input**文をぬけ(**on**の場合は割り込み処理を行い)、次の処理に進みます。このときの**input**文の入力変数の値は変化しません。
- **run**コマンドおよび**end**文の実行やエラーの発生、または実行中断時のプログラム変更などにより、このほかの割り込み機能も含めて、すべての割り込み処理の実行制御は**off**になります。
- **on help gosub**文で割り込み処理行が定義されていない場合は、割り込みが発生しても何も処理は行われません。

例

```
help on
```
…… [ヘルプ]キーによる割り込みを可能とします。

参 考

on help gosub …… 割り込み処理行の定義。(410ページ)

ON COM GOSUB (オン コミュニケーション ゴーサブルーチン・on communication go to subroutine)

RS−232Cインターフェイスによる割り込み処理行を定義します。

書　式　on com gosub〈処理行〉

省略形　o. com gos.

説　明

- RS−232Cインターフェイスにデータが入力され割り込みが発生したときに、割り込み処理が行われる開始行を＜処理行＞として定義します。
- この命令では割り込み処理行を定義するだけで、割り込み処理の実行を制御するのはcom on/off/stop文で行います。
- ＜処理行＞は、行番号またはラベルで指定できます。
- 割り込み処理は、割り込み発生時に＜処理行＞からサブルーチンとして実行されますので、割り込み処理の終了はreturn文になります。
- この命令での定義はrunコマンドおよびend文の実行により解除されます。
- この命令の実行時に＜処理行＞で指定された行がない場合はエラーになります。

例

```
on com gosub *COMIN
```

…… RS-232Cインターフェイスによる割り込み処理行を＊COMINというラベルで定義します。

COM ON /OFF/STOP (コミュニケーション オン/オフ/ストップ・communication on/off/stop)

RS−232Cインターフェイスによる割り込み処理を制御します。

書　式

com〔(〈チャンネル番号〉)〕{on / off / stop}

省略形

com o./ of. /s.

説　明

- RS-232Cインターフェイスによる割り込み処理を制御します。
  ＜チャンネル番号＞の指定は1となります。
- onを指定すると割り込みが可能となり、on com gosub 文で定義されている割り込み処理の開始行から、サブルーチンとして実行します。
- offを指定すると割り込み機能は解除され、割り込みが発生しても割り込み処理は行われません。
- stopを指定すると割り込み発生時には割り込み処理は行われませんが、割り込みがあったことは記憶されていますので、あとでon が指定されたときに割り込み処理を実行します。
- 割り込み処理はステートメントとステートメントの間に行われ、ステートメント実行中はstopの状態になります。したがってpaint文やpoly文などの処理時間がかかる処理中に割り込みがかかった場合は、その処理が終了してから割り込み処理が行われます。また、input文やline input文は入力終了後に割り込み処理を実行します。
- シフト＋停止キーの操作または、stop文の実行によりプログラムが中断され、contコマンドの実行が可能な場合は、中断時にonとなっていたものはstopと同じ扱いになり、プログラムの再開で再びonになります。
- runコマンドおよびend文の実行やエラーの発生、または実行中断時のプログラム変更などにより、このほかの割り込み機能も含めて、すべての割り込み処理の実行制御はoffになります。
- on com gosub文で割り込み処理行が定義されていない場合は、割り込みが発生しても何も処理は行われません。

例

com on …… RS−232Cインターフェイスによる割り込みを可能とします。

参　考

on com gosub …… RS−232Cインターフェイスによる割り込み処理行の定義。(413ページ)

ON MOUSE GOSUB (オン マウス ゴーツーサブルーチン・on mouse go to subroutine)

マウスによる割り込み処理を定義します。

書　式　on mouse gosub 〔〈処理行1〉〕〔,〈処理行2〉〕……〔,〈処理行5〉〕

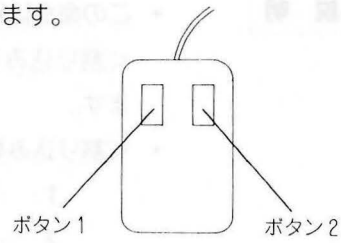
省略形　o. mou. gos.

説　明

- この命令は別売のマウス(MZ-1X10)の操作による各種割り込み要因に対する割り込み処理を定義します。

割り込み要因は次のように1～5の5種類があります。

1 …… マウスが移動したとき。
2 …… ボタン1が押されたとき。
3 …… ボタン2が押されたとき。
4 …… ボタン1が離されたとき。
5 …… ボタン2が離されたとき。



- ＜処理行＞の1～5は、割り込み要因の1～5のそれぞれに対応した割り込み処理の先頭行を行番号またはラベルで指定します。
- この命令では割り込み処理の行われる開始行を定義するだけで、割り込み処理の制御はmouse on/off/stop文で行われます。
- 割り込み処理は、割り込み発生時に＜処理行＞からサブルーチンとして実行されますので、割り込み処理の終了はreturn文になります。
- この命令での定義はrunコマンドおよびend文の実行により解除されます。
- この命令実行時に＜処理行＞で指定された行がない場合はエラーになります。

例

```
on mouse gosub ,*TRG_1
```

…… マウスのボタン1が押されたときに実行する割り込み処理行を＊TRG_1というラベルで定義します。

参　考　mouse on/off/stop …… マウスによる割り込みの制御。(416ページ)

サンプル　mouse on/off/stopのサンプル参照

MOUSE ON/OFF/STOP (マウス オン/オフ/ストップ)

マウスによる割り込み処理を制御します。

書 式

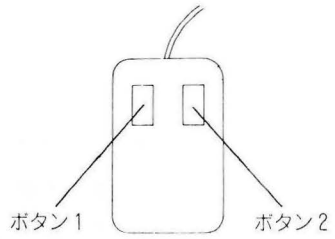
mouse〔(〈割り込み要因〉)〕{ on / off / stop }

省略形

mou. o. /of. /s.

説 明

- この命令は別売のマウス(MZ-1X10)による割り込み処理を制御します。
  <割り込み要因>は1～5で指定し、指定を省略するとすべての指定となります。
- <割り込み要因>は次のように対応します。
  - 1 …マウスが移動したとき。
  - 2 …ボタン1が押されたとき。
  - 3 …ボタン2が押されたとき。
  - 4 …ボタン1が離れたとき。
  - 5 …ボタン2が離れたとき。



- onを指定すると割り込みが可能となり、on mouse gosub文によって定義されている割り込み処理の開始行から、サブルーチンとして実行します。
- offを指定すると割り込み機能は解除され、割り込みが発生しても割り込み処理は行われません。
- stopを指定すると割り込み発生時には割り込み処理は行われませんが、割り込みがあったことは記憶されていますので、あとでonが指定されたときに割り込み処理を実行します。
- 割り込み処理はステートメントとステートメントの間に行われ、ステートメント実行中はstop状態になります。したがってpaint文やpoly文などの処理時間がかかる処理中に割り込みがかかった場合は、その処理が終了してから割り込み処理が行われます。また、input文やline input文は入力終了後に割り込み処理を実行します。
- runコマンドおよびend文の実行やエラーの発生、または実行中断時のプログラム変更などにより、このほかの割り込み機能も含めて、すべての割り込み処理の実行制御はoffになります。
- on mouse gosub文で割り込み処理行が定義されていない場合は、割り込みが発生しても何も処理は行われません。

例

```
mouse on
```

…… マウスによるすべての割り込みを可能とします。

参 考☞ on mouse gosub …… マウスによる割り込み処理行を定義する。(415ページ)

サンプル

```
  10 rem --- on mouse gosub ---
  20 mouse 6
  30 init "CRT2:640,400,16":cls 3:color ,,,4
  40 mouse 0:mouse 4,0,16,599,339:mouse 1,320,200,1
  50 gosub *MOVE
  60 on mouse gosub *MOVE,*T1_ON,*T2_ON,,*T2_OFF
  70 if inkey$="" else mouse 6:end
  80 mouse on:goto 70
1000 *MOVE :mouse stop
1010 X=mouse(0):Y=mouse(1)
1020 locate 0,0:print "(x,y)=(";X;",";Y;")";chr$(5)
1030 if T2 gosub *XLINE:X1=X:Y1=Y:gosub *XLINE
1040 return
2000 *T1_ON :mouse stop
2010 if T2 else 2040
2020 mouse 1,,,0:line (XS,YS)-(X1,Y1):mouse 1,,,1
2030 XS=X1:YS=Y1:pset (XS,YS),,xor
2040 return
3000 *T2_ON :mouse stop
3010 if F and not T2 then 5010
3020 XS=mouse(0):YS=mouse(1)
3030 X1=XS:Y1=YS:T2=1:gosub *XLINE
3040 return
5000 *T2_OFF:mouse stop
5010 gosub *XLINE:T2=0:return
6000 *XLINE
6010 mouse 1,,,0:line (XS,YS)-(X1,Y1),xor:mouse 1,,,1
6020 F=not F
6030 return
9000 ' ボタン２を押しながら移動すると線をためし引きします
9010 ' そのときにボタン１を押すと線を引きます
9020 ' いずれかのキーを押すとプログラムを終了します
```

ON TIME$ GOSUB (オン タイム ダラー ゴーツーサブルーチン・on time$ go to subroutine)

リアルタイムクロックでの割り込み処理を定義します。

書　式　on time$="〈時〉:〈分〉:〈秒〉" gosub 〈処理行〉

省略形　o. ti. = gos.

説　明

- リアルタイムクロックが設定時刻になったときに割り込みを発生させ、割り込み時間と割り込み処理の開始行を定義します。
- この命令では割り込み処理の行われる開始行を定義するだけで、割り込み処理の制御はtime$ on/off/stop文で行われます。
- ＜処理行＞は、行番号またはラベルで指定できます。
- 割り込み処理は、割り込み発生時に＜処理行＞からサブルーチンとして実行されますので、割り込み処理の終了はreturn文になります。
- この命令での定義はrunコマンドおよびend文の実行により解除されます。
- この命令の実行時に＜処理行＞で指定された行がない場合はエラーになります。
- この命令の実行時に、システム変数のtime$と割り込み時刻との差を計算しますので、time$の設定はこの命令の実行前に行ってください。
- 割り込みは＜分＞単位で処理され、＜秒＞の指定は"00"とみなします。ただし＜秒＞の指定を省略することはできません。

例

```
on time$="12:45:00" gosub 1000
```

…… 12時45分になると行番号1000から処理をするように定義します。

参　考　time$ on/off/stop …… リアルタイムクロックの割り込み制御。(419ページ)

TIME$ ON/OFF/STOP (タイム ダラー オン/オフ/ストップ)

リアルタイムクロックの割り込みを制御します。

書 式

time$ { on / off / stop }

省略形

ti. o. /of. /s.

説 明

- リアルタイムクロックからの割り込みを制御します。
- onを指定すると割り込みが可能となり、on time$ gosub文で定義されている時刻に割り込みが発生し、処理行からサブルーチンとして実行します。
- offを指定すると割り込み機能が解除され、割り込みが発生しても割り込み処理は行われません。
- stopを指定すると割り込み発生時には割り込み処理は行われませんが、割り込みがあったことは記憶されていますので、あとでonが指定されると割り込み処理を実行します。
- 割り込み処理はステートメントとステートメントの間に行われ、ステートメント実行中はstop状態になります。したがって、paint文やpoly文などの処理時間がかかる処理中に割り込みがかかった場合は、その処理が終了してから割り込み処理が行われます。また、input文やline input文は入力終了後に割り込み処理を実行します。
- runコマンドおよびend文の実行やエラーの発生、または実行中断時のプログラム変更などにより、このほかの割り込み機能も含めて、すべての割り込み処理の実行制御はoffになります。
- on time$ gosub文で割り込み処理行が定義されていない場合は、割り込みが発生しても何も処理は行われません。

例

```
time$ on
```

…… リアルタイムクロックによる割り込みを可能とします。

参 考

on time$ gosub …… リアルタイムクロックの割り込み定義。(418ページ)

INTERVAL （インターバル）

インターバルタイマの設定をします。

書　式　interval〈周期〉

省略形　inte.

説　明

- 一定間隔で割り込みを発生させるインターバルタイマの周期を設定します。
- ＜周期＞は0.1秒単位で1～65535までの整数で指定します。
- この命令の実行によりインターバルタイマが初期化されスタートします。したがって、＜周期＞を50とした場合、この命令の実行を起点として5秒ごとの割り込みが発生します。
- 割り込み処理をするには、on interval gosub文で割り込み処理の定義、interval on/off/stop文により割り込み処理の制御を行います。

例　`interval 100` …… インターバルタイマの周期を10秒とします。

参　考☞

on interval gosub …… インターバルタイマの割り込み処理の定義。
（421ページ）

interval on/off/stop …… インターバルタイマの割り込み処理の制御。
（422ページ）

ON INTERVAL GOSUB (オン インターバル ゴーツーサブルーチン・on interval go to subroutine)

インターバルタイマによる割り込み処理を定義します。

書　式　on interval gosub〈処理行〉

省略形　o. inte. gos.

説　明

- interval文により設定された一定間隔で発生する割り込みに対する割り込み処理の開始行を＜処理行＞として定義します。
- この文では割り込み処理の行われる開始行を定義するだけで、割り込み処理の制御は**interval on/off/stop**文で行われます。
- ＜処理行＞は、行番号またはラベルで指定できます。
- 割り込み処理は、割り込み発生時に＜処理行＞からサブルーチンとして実行されますので、割り込み処理の終了は**return**文になります。
- この命令での定義は**run**コマンドおよび**end**文の実行により解除されます。
- この命令の実行時に＜処理行＞として指定された行がない場合はエラーになります。

例

```
on interval gosub 1000
```

…… インターバルタイマによる割り込み処理行を行番号1000として定義する。

参　考　interval on/off/stop …… インターバルタイマの割り込み処理の制御をする。(422ページ)

interval …… インターバルタイマの設定。(420ページ)

サンプル

```
 10 rem --- ON INTERVAL GOSUB ---
 20 cls 3:interval 100:locate ,,0
 30 on stop gosub *STOP:stop on
 40 on interval gosub *BEEP:interval on
 50 locate 0,6:color 7:print "現在の時刻:";time$
 60 locate 0,8:color 2:print "１０秒おきにブザーを鳴らします。
 70 goto 50
 80 color 7:end
 90 *BEEP
100 interval off
110 beep:beep
120 interval on:return
130 *STOP
140 stop off
150 locate ,,1
160 return 80
```

INTERVAL ON/OFF/STOP (インターバル オン/オフ/ストップ)

インターバルタイマによる割り込み処理を制御します。

書 式

interval { on / off / stop }

省略形

inte. o./of./s.

説 明

- 一定間隔で発生するインターバルタイマによる割り込み処理を制御します。
- onを指定すると割り込み処理が可能となり、on interval gosub文で定義されている処理行からサブルーチンとして実行します。
- offを指定すると割り込み機能が解除され、割り込みが発生しても割り込み処理は行われません。
- stopを指定すると割り込み発生時には割り込み処理は行われませんが、割り込みがあったことは記憶されていますので、あとでonが指定されたときに割り込み処理を実行します。
- 割り込み処理はステートメントとステートメントの間に行われ、ステートメント実行中はstop状態になります。したがって、paint文やpoly文などの処理時間がかかる処理中に割り込みがかかった場合は、その処理が終了してから割り込み処理が行われます。また、input文やline input文は入力終了後に割り込み処理を実行します。
- run コマンドおよびend文の実行やエラーの発生、または実行中断時のプログラム変更などにより、このほかの割り込み機能も含めて、すべての割り込み処理の実行制御はoffになります。
- on interval gosub文で割り込み処理行が定義されていない場合は、割り込みが発生しても何も処理は行われません。

例

```
interval on
```

…… インターバルタイマによる割り込み処理を可能とします。

参 考

interval gosub …… インターバルタイマによる割り込み処理を定義する。(421ページ)

interval …… インターバルタイマの設定。(420ページ)

ON PLAY GOSUB (オン プレイ ゴーツーサブルーチン・on play go to subroutine)

音楽演奏終了時に発生する割り込み処理を定義します。

書　式　on play gosub〈処理行〉

省略形　o. pl. gos.

説　明

- play 文による音楽演奏で、演奏が終了したときに発生する割り込みに対する処理の開始行を＜処理行＞として定義します。
- この命令では割り込み処理の行われる開始行を定義するだけで、割り込み処理の制御はplay on/off/stop文で行われます。
- ＜処理行＞は、行番号またはラベルで指定できます。
- 割り込み処理は割り込み発生時に＜処理行＞からサブルーチンとして実行されますので、割り込み処理の終了はreturn文になります。
- この命令での定義はrunコマンドおよびend文の実行により解除されます。
- この命令の実行時に＜処理行＞として指定された行がない場合は、エラーになります。
- この命令はplay on文を実行するまでに実行してください。

例

```
on play gosub 1000
```

…… 音楽演奏終了時の割り込み処理行を行番号1000として定義します。

参　考　play on/off/stop …… 音楽演奏終了時の割り込みの制御。(425ページ)

サンプル

```
 10 rem --- ON PLAY GOSUB ---
 20 on play gosub *PRINT:play on
 30 init "CRT2:640,200,16":cls 3
 40 play init
 50 for N=1 to 2
 60   read C1$,C2$,C3$ : play C1$,C2$,C3$
 70 next
 80 play wait:beep:play off
 90 end
100 *PRINT
110 for I=0 to 10
120   PA=int(rnd(1)*100)
130   X1=int(rnd(1)*640)*.5:Y1=int(rnd(1)*200)*.5
140   X2=int(rnd(1)*640)   :Y2=int(rnd(1)*200)
150   P=(Y2 mod 6):C=(X2 mod 15)+1
160   pen P
170   circle (X2,Y2),PA,C
180   line (X1,Y1)-(X2,Y2),C,BF
190 next
200 return
210 data "Q8T60@7@V127Q8@V120@19O5RA8>C16D8.C8<B8A8"
```

```
220 data "Q8T60@7@V127Q7@V115@10FFFF"
230 data "Q8T60@7@V127Q8@V100@405A2B2"
240 data "ER8E8F8E16F8G16E16D16C<A8>C8F8E8C8D8E2RR8R8"
250 data "EE8E16D16CC16C16D16E16FFDDEEEE"
260 data "B2B-2A2A2F+2G+2"
```

- グラフィック表示後音楽を鳴らし、**on play gosub**命令により音楽が終了すると再びグラフィックを表示します。

PLAY ON/OFF/STOP (プレイ オン/オフ/ストップ)

音楽演奏終了時の割り込み機能を制御します。

書 式

play { on / off / stop }

省略形

pl. o. /of. /s.

説 明

- 音楽演奏終了時に発生する割り込みを制御します。
- onを指定すると割り込み処理が可能となり、on play gosub文で定義されている処理行からサブルーチンとして実行します。
- offを指定すると割り込み機能が解除され、割り込みが発生しても割り込み処理は行われません。
- stopを指定すると割り込み発生時には割り込み処理は行われませんが、割り込みがあったことは記憶されていますので、あとでonが指定されたときに割り込み処理を実行します。
- 割り込み処理はステートメントとステートメントの間に行われ、ステートメント実行中はstop状態になります。したがって、paint文やpoly文などの処理時間がかかる処理中に割り込みがかかった場合は、その処理が終了してから割り込み処理が行われます。また、input文やline input文は入力終了後に割り込み処理を実行します。
- runコマンドおよびend文の実行やエラー発生、または実行中断時のプログラム変更どにより、このほかの割り込み機能も含めて、すべての割り込み処理の実行制御はoffになります。
- on play gosub文で割り込み処理行が定義されていない場合は、割り込みが発生しても何も処理は行われません。

例

```
play on
```
…… 音楽演奏終了時の割り込みを可能とします。

参 考

on play gosub …… 音楽演奏終了時の割り込み処理行の定義。(423ページ)

CHILD(チャイルド)

チャイルドプロセスを呼び出し、実行します。

書　式　**child〔〈チャイルドプロセス名およびチャイルドプロセスに引き渡す情報〉〕〔,M〕〔,[〈画面クリアスイッチ1〉]〔,[〈画面クリアスイッチ2〉]]**

省略形　**chi.**

説　明

- この命令を実行するとBASICの環境から一時的にMS-DOSの環境に制御を移行して、チャイルドプロセスを実行することができます。
  チャイルドプロセスを実行後、制御は再びBASICに戻ります。
  チャイルドプロセスとはMS-DOS下で実行されるプログラムのことです。
- <チャイルドプロセス名およびチャイルドプロセスに引き渡す情報>には、起動するチャイルドプロセス名(プログラム名)とチャイルドプロセスに引き渡す情報を124字以内の文字列で指定します。
  指定形式は次のとおりです。
  　"チャイルドプロセス名␣引き渡し情報"（␣:スペース）
  　例:　"COPY　A:TEST B:"
  　　チャイルドプロセス　引き渡し情報
  なお、チャイルドプロセス名の指定形式は次のとおりです。
  　〔<ドライブ名>〕<プログラム名>
- 起動されるチャイルドプロセス(プログラム)は、指定されたドライブのカレントディレクトリ下のプログラムです。
  ドライブ名が省略された場合、カレントドライブが指定されます。
- チャイルドプロセスを特定のディレクトリ下のプログラムとする場合、**environ**文でディレクトリ情報を環境文字列"PATH"にあらかじめ設定しておく必要があります。(**environ**文参照)
- <チャイルドプロセス名およびチャイルドプロセスに引き渡す情報>を省略した場合は、MS-DOSに制御が移った後、プロンプトが表示され命令待ちの状態になります。
  以後、MS-DOSを直接操作できるようになります。
  BASICへ制御を戻すときは、MS-DOSの**EXIT**コマンドを実行する必要があります。(MS-DOSマニュアル参照)
- **M**を指定しない場合は、チャイルドプロセスは**clear**文で確保されたチャイルドプロセス領域に読み出され、実行されます。
  チャイルドプロセスに必要な領域が確保されていないと、エラーとなります。
  なお**M**を指定しない場合は、カンマ(,)も含めて省略してください。

- **M**を指定した場合、チャイルドプロセス領域を確保する必要はありませんが、BASICのデータ領域をチャイルドプロセス領域にします。したがって、チャイルドプロセスを実行すると、実行終了後処理は中断され、BASICダイレクトモード(命令待ち)になります。このとき、配列および変数は初期化され、機械語領域の内容も破壊されます。
- <画面クリアスイッチ1>は、チャイルドプロセスを実行するときに、画面を消去するかどうかを指定します。
  - 省略あるいは**0**の場合
    …画面を消去した後、チャイルドプロセスを起動します。
  - **1**の場合
    …画面はそのままで、チャイルドプロセスを起動します。
- <画面クリアスイッチ2>は、チャイルドプロセスからBASICに制御を戻すときに、チャイルドプロセスの画面を消去するかどうかを指定します。
  - **0**の場合
    …チャイルドプロセスの画面はそのままで、BASICに制御を戻します。
  - 省略あるいは**1**の場合
    …チャイルドプロセスの画面を消去後、BASICに制御を戻します。
- 現在の装置(カレントドライブ)のルートディレクトリ下にコマンドファイル(**command. com**)が用意されていない場合、エラーとなります。
- チャイルドプロセス起動直後のカーソル位置は<画面クリアスイッチ1>により変わります。
  - 省略あるいは**0**の場合:ホームポジション
  - **1**の場合　　:移動しない
- チャイルドプロセスからBASICへ制御が戻った場合、カーソル位置は<画面クリアスイッチ2>により変わります。
  - **0**の場合　　:移動しない
  - 省略あるいは**1**の場合:ホームポジション
- 実行できないチャイルドプロセスの場合は、"コマンドまたはファイル名が違います"と表示されます。
  また、ドライブの指定が誤まっている場合は、"ドライブの指定が違います"と表示されます。
  これらはMS-DOSのエラーメッセージです。メッセージ表示後BASICのダイレクトモード(命令待ち)に戻ります。
- 環境文字列のCOMSPECの内容が変更されているとチャイルドプロセスを実行できないことがあります。

例

```
child "COPY TEST.BAS B:"
```

…カレントドライブのTEST.BASをドライブBにコピーします。

```
child "DIR",1,0
```

…BASICの画面はそのままでカレントドライブのディレクトリをカーソル位置から表示します。

```
child
```

…MS-DOSの命令待ちとなります。

参　考☞

clear　…チャイルドプロセス領域の確保。(169ページ)

environ　…環境文字列の定義。(429ページ)

ENVIRON（インバイロン）

環境文字列テーブルの内容を変更します。

書　式　environ〈文字列〉

省略形　env.

説　明

- 環境文字列テーブルは、**child**文によってMS-DOSの環境に移行したときに、有効となる文字列の集まりです。
- ＜文字列＞は次のように指定します。
  "＜名前＞＝＜パラメータ＞"
- ＜名前＞に"PATH"を指定すると、"PATH"に続＜＜パラメータ＞はチャイルドプロセスが起動するときのディレクトリパスとなります。
- ＜名前＞および＜パラメータ＞は英数字、記号で指定します。ただし、大文字小文字を区別しますので、"path"としてもディレクトリパスの指定はできません。
- 同じ＜名前＞の環境文字列を登録することはできません。同じ＜名前＞で登録すると、新しく登録する方に＜パラメータ＞が更新されます。
- "＜名前＞=;"＜名前＞="とすると、指定された＜名前＞と＜パラメータ＞は環境文字列テーブルから削除されます。
  ただし、＜名前＞が"PATH"の場合は"PATH"が環境文字列テーブルに残されます。
- 新しく登録した環境文字列は環境文字列テーブルの最後に追加されます。
- この命令はBASIC終了時のMS-DOSの環境文字列テーブルには影響を与えません。
- 環境文字列を変更する場合、全体で256バイトを越えてパラメータを設定することはできません。

例

```
environ "PATH=B:¥UTY"
child "B:TEST"
```

…ドライブBのルートディレクトリ内のUTYというディレクトリ下のTESTというファイルが起動されます。

ENVIRON$ （インバイロンダラー）

指定された環境文字列のパラメータを求めます。

書　式

environ$(" 〈名前〉 ")　(1)

environ$(〈番号〉)　(2)

説　明

- (書式1)は<名前>で指定された環境文字列のパラメータを求めます。
- (書式2)は環境文字列テーブル内で先頭から何番目に位置するかを<番号>で指定することによって、特定の環境文字列のパラメータを求めます。
- <番号>は0～32767の値で指定します。
- 指定されたパラメータが無い場合は空文字が返されます。

例

```
print environ$("PATH").
```

…環境文字列"PATH"に設定されているディレクトリパスを表示します。

OPTION FLOAT（オプションフロート）

数値演算プロセッサの使用を可能にします。

書　式

option float { on / off }

省略形

opt. fl.

説　明

- 数値演算プロセッサ(**MZ-IM12**)(別売)を使用するかどうかを指定します。
  onと指定すると数値演算プロセッサが装着されているとき、数値演算プロセッサを使用します。
  offと指定すると使用しません。
- 数値演算プロセッサを使用した場合、次の関数を高速処理することができるようになります。
  sin, cos, tan, atn, log log 10, exp, sgr, fac, ^(べき乗)

例

```
option float on
```

MOUSE （マウス）

マウスの機能を設定します。

書　式　mouse 〈機能〉〔,〈指定〉〕…

省略形　mou.

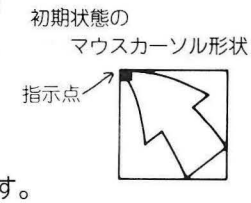
説　明

- 別売のマウス(**MZ-1 ×10**)に対して各種の機能を設定します。＜機能＞は**0～6**まであり、次のようになります。なお、マウス処理の座標はすべて絶対スクリーン座標になります。

mouse 0

各種機能を初期化し、マウスを使用可能にします。初期化される機能は次のとおりです。

- マウスカーソルを画面中央に移動させ、オフ(表示しない)状態にします。
- マウスカーソル形状を初期状態に設定し、指示点は左上端になります。なお、マウスカーソルは表示されませんので、表示するには**mouse 1 , , , 1**を実行してください。



- 縦横の移動比率(434ページ参照)を**8**にします。
- 移動範囲を画面全体として、カーソル色は緑になります。

mouse 1 , 〔X〕 , 〔Y〕 , ＜カーソルスイッチ＞

マウスカーソルの移動と、オン、オフの制御をします。

X, Yはマウスカーソルを移動させる座標になります。マウスカーソルの指示点が指定座標になるような移動です。

Xまたは**Y**を省略すると、現在のマウスカーソルのX座標またはY座標が設定されます。

＜カーソルスイッチ＞を1(オン)にするとマウスカーソルを表示し、0(オフ)にするとマウスカーソルは表示しません。

mouse 2 , ＜横指示点＞ , ＜縦指示点＞ , ＜XOR文字列＞
〔, ＜AND文字列＞〕

マウスカーソルの形状および指示点を指定します。

マウスカーソルは16×16ドットの範囲内で任意の形状を定義することができます。

＜横指示点＞、＜縦指示点＞は0～15の値で指定し、この指示点がマウス入力時の座標になります。なお、グラフィックが640×200ドットのときは、カーソルは640×400ドット時の倍の大きさで表示されます。

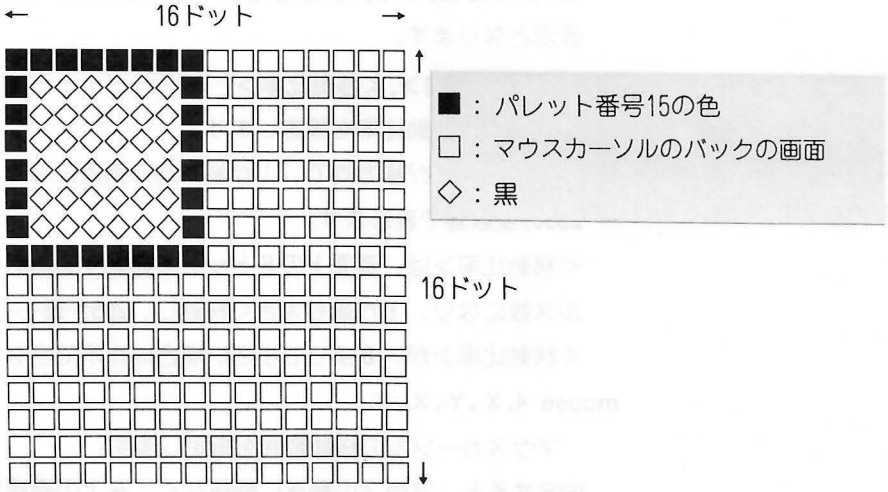
マウスカーソルの形状は、<XOR文字列>および<AND文字列>で指定します。

形状の指定はドットのONをビット１、ドットのOFFをビット０に対応させて指定します。

マウスカーソルと文字列の対応は次のようになります。

<XOR文字列>と<AND文字列>でカーソル形状を指定する場合

ドットの状態		画面上のドットの状態
AND文字列	XOR文字列	
1	0	マウスカーソルのバックの画面
1	1	マウスカーソルのバックの画面のパレット番号を反転させたパレット番号
0	0	黒
0	1	パレット番号15の色



たとえば、上の図のようなマウスカーソルを表示するときは、XOR文字列とAND文字列は次のようになります。

```
XOR文字列=chr$(&HFF, &H00)+chr$(&H81, &H00)
          +chr$(&H81, &H00)+chr$(&H81, &H00)
          +chr$(&H81, &H00)+chr$(&H81, &H00)
          +chr$(&H81, &H00)+chr$(&HFF, &H00)
          +chr$(&H00, &H00)+chr$(&H00, &H00)
                           :
          +chr$(&H00, &H00)+chr$(&H00, &H00)
AND文字列=chr$(&H00, &HFF)+chr$(&H00, &HFF)
          +chr$(&H00, &HFF)+chr$(&H00, &HFF)
          +chr$(&H00, &HFF)+chr$(&H00, &HFF)
          +chr$(&H00, &HFF)+chr$(&H00, &HFF)
          +chr$(&HFF, &HFF)+chr$(&HFF, &HFF)
                           :
          +chr$(&HFF, &HFF)+chr$(&HFF, &HFF)
```

＜XOR文字列＞だけでカーソル形状を指定する場合

文字列の指定のしかたは＜XOR文字列＞と＜AND文字列＞での指定のしかたと同じです。

ただし、XOR文字列とグラフィック情報との間にXORの関係があるため、グラフィック情報とマウスカーソルが重なる場合、重なった部分の色が変化します。

また、**mouse 0**により初期化されるマウスカーソルはXOR文字列だけの表示となります。

mouse 3, ＜方向＞, ＜移動比率＞

マウスの移動比率を設定します。

＜方向＞は**0**が横方向で、1が縦方向の指定になり、＜移動比率＞は**1～255**の整数値で表します。

＜移動比率＞は、画面上の8ドットを移動するのに必要なマウスからのパルス数になり、**1**が最も大きく移動し、**255**が最も小さくなります。また、＜移動比率＞が、**8**よりも小さい場合は指示できない座標が出てきます。

mouse 4, X_1, Y_1, X_2, Y_2

マウスカーソルの移動範囲を指定します。

指定すると、マウスの動きに関係なく、左上の座標**X_1, Y_1**と右下の座標**X_2, Y_2**を対角とする範囲の外へ指示点を移動させることはできなくなります。

mouse 1で移動範囲外の座標を指定した場合、マウスカーソルは移動範囲内で**mouse 1**で指定された座標に最も近い点に表示されます。

なお、**mouse 0**実行時の移動範囲はグラフィック画面全体になっています。

mouse 5,＜色＞

マウスの表示色を指定します。

＜色＞の指定は次のようになります。

＜色＞の指定	表示色
0	青(パレット番号1)
1	赤(パレット番号2)
2	緑(パレット番号4)
3※	灰(パレット番号8)

※グラフィック画面が65536色モード時は＜色＞に3を指定することはできません。

mouse 0 実行時のマウスカーソルの色はパレット番号0に対応した色になります。

なお、この命令はXOR文字列だけで表示しているマウスカーソルに対してのみ有効です。

mouse 6

マウスの処理を終了します。

この命令を実行後再びマウスを使用する場合は、再び**mouse 0** を実行し、マウスの初期化を行います。

- グラフィックの描画や消去を行うときは必ずマウスカーソルをオフ(表示しない)状態にしてください。

サンプル　**mouse on/off/stop**のサンプル参照

MOUSE(n)　(マウス)

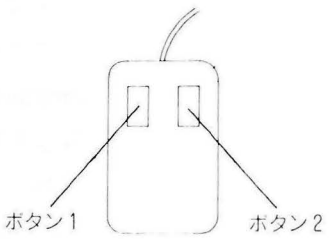
マウスからの情報を読み取ります。

書　式　mouse(〈機能〉[,〈ボタン番号〉])

省略形　mou.

説　明

- 別売のマウス(**MZ-1X10**)から座標やボタンの情報を読み取る関数です。
<機能>は0～10まであり、<機能>に応じて<ボタン番号>として1か2のボタン番号を指定します。この関数で求められる座標はすべて絶対スクリーン座標になります。



mouse(0)
　マウスカーソルのX座標上の現在値を求めます。

mouse(1)
　マウスカーソルのY座標上の現在値を求めます。

mouse(2,<ボタン番号>)
　<ボタン番号>で指定されたボタンの状態を求めます。ボタンが押されている場合は−1、押されていない場合は0の値が返ります。

mouse(3,<ボタン番号>)
　最後にこの関数を実行してから指定されたボタンが押された回数を求めます。

mouse(4,<ボタン番号>)
　<ボタン番号>で指定されたボタンが最後に押されたときのX座標を求めます。

mouse(5,<ボタン番号>)
　<ボタン番号>で指定されたボタンが最後に押されたときのY座標を求めます。

mouse(6,<ボタン番号>)
　最後にこの関数を実行してから指定されたボタンが離された回数を求めます。

mouse(7,<ボタン番号>)
　<ボタン番号>で指定されたボタンが最後に離されたときのX座標を求めます。

mouse(8,＜ボタン番号＞)

＜ボタン番号＞で指定されたボタンが最後に離されたときのY座標を求めます。

mouse(9)

最後にこの関数を実行してからのX座標の移動量を求めます。

mouse(10)

最後にこの関数を実行してからのY座標の移動量を求めます。

- mouse 0の実行でマウスを使用可としていないと正しい値は返りません。

サンプル　mouse on/off/stopのサンプル参照

INIT "KB:" (イニシャライズ キーボード・initialize keyboard)

キーボードの機能を設定します。

書　式

init "KB:[〈カナ配列〉][,〈文字選択〉][,〈LOCK〉][,〈漢字入力〉]"

説　明

- ＜カナ配列＞は、キーボードのカナ配列をJISで定められた配列にするか、50音順の配列にするかを選択します。
  - 0 …… JIS配列。
  - 1 …… 50音順配列。
- ＜文字選択＞は、英数、カナ、グラフィック文字の入力状態を指定します。
  - 0 …… 英数入力。
  - 1 …… カナ入力。
  - 2 …… グラフィック文字入力。
- ＜LOCK＞は、アルファベットの大文字、小文字の入力を指定します。
  - 0 …… 小文字入力。
  - 1 …… 大文字入力。
- ＜漢字入力＞は、漢字入力状態を指定します。
  - 0 …… 漢字入力状態を解除します。
  - 1 …… 漢字入力状態にします。(メニューモード)
  - 2 …… 漢字入力状態にします。(スクリーンモード)
- 漢字入力機能はMS-DOSのものを使用しますので、MS-DOS起動時に仮名漢字変換のデバイスドライバが読み込まれていないと、漢字入力機能は使用できません。
- それぞれの指定を省略した場合は、現在の状態を保ちます。
- BASICの起動時は、すべての指定が0になっています。
- キーボード自体のカナ配列はJIS配列になっています。50音順配列については、オーナーズマニュアルを参照してください。

例

```
init "KB:1"
```
…… キーボードのカナ配列を50音順に設定します。

CLICK ON/OFF (クリック オン/オフ)

キーのクリック音を制御します。

書 式

click { on / off }

省略形

cli. o. /of.

説 明

- キーボードのキーを押すごとに出るクリック音を制御します。onを指定すると音が出るようになり、offを指定すると音は出なくなります。
- BASICの起動時はclick onの状態になります。
- クリック音はコンピュータが有効な文字入力を行ったときに発生します。したがって シフト キーや コントロール キーを単独に押しただけではクリック音は発生しません。

例

```
click off  …… クリック音を止めます。
click on   …… クリック音が出るようにします。
```

REPEAT ON/OFF （リピート オン/オフ）

キーリピートを制御します。

書 式

repeat {on [,〈スピード〉]
off}

省略形

rep. o./of.

説 明

- キーを押し続けたときのキーリピートとリピート速度を指定します。onを指定するとリピート可能となり、offを指定するとキーを押し続けても、リピートを行いません。
- repeat onの指定のときに<スピード>でリピート速度を指定します。

<スピード>

0 …リピートは行いません。（**repeat off**と同様）
1 …低速リピート。
2 …中速リピート。（省略時はこの速度になります。）
3 …高速リピート。
4 …最高速リピート。

<スピード>が**4**に設定されている場合は[コントロール]+[D]キーの操作により<スピード>を**2**に設定します。また、<スピード>を省略した場合は**2**が指定されたものとします。

- BASICの起動時は**repeat on**の状態になります。
- リピートは、ダイレクトモードまたは**input**文の実行時などキー入力待ちのときにリピート状態になります。**先行入力時はリピートを行いません。**

例

```
repeat on  …… 中速リピートに設定します。
repeat off …… キーを押し続けてもリピートしなくなります。
```

INKEY$ (インプット キー ダラー・input key$)

キーボードより1文字入力を行います。

書 式 inkey$

省略形 ink.

説 明

- キーボードより1文字のデータを入力するシステム変数です。
  入力がなかった場合は入力を待つのではなく、値をNull(空文字)として次の処理に進みます。
- 入力は先行入力のキーバッファの最初の文字となりますので、先行入力によりすでに入力されている場合は、順次1文字ずつ入力します。あらかじめキーバッファを消去する場合はkey 0,""を実行してください。
- リアルタイム入力を行うときは repeat on, 4 としてリピート速度を最高速にしておきます。ただしリピートが最高速の場合、通常のキー入力はできませんので、プログラムを終了する場合やinput文の実行前にはもとの速度に戻しておく必要があります。
- シフト+停止キーの操作はプログラム実行の中止になりますので入力することはできません。

例

```
A$=inkey$ …… 入力した1文字を変数A$に代入します。
100 if inkey$="A" then print "a"
    …… もし入力された文字が大文字の"A"であれば小文字の"a"を表示します。
```

サンプル

```
 10 rem --- INKEY$ ---
 20 input "A, B";A,B
 30 print "どの計算をするか、1～3で選んで下さい。"
 40 print " 1・・・C＝A＋B  2・・・C＝A－B  3・・・C＝A＊B"
 50 D$=inkey$:if D$="" then 30
 60 D=val(D$):if D<1 or D>3 then beep:goto 30
 70 on D gosub 100,110,120
 80 print:print "答えは";C;"です。"
 90 end
100 C=A+B:return
110 C=A-B:return
120 C=A*B:return

run
A, B? 50,85
どの計算をするか、1～3で選んで下さい。
 1・・・C＝A＋B  2・・・C＝A－B  3・・・C＝A＊B
どの計算をするか、1～3で選んで下さい。
 1・・・C＝A＋B  2・・・C＝A－B  3・・・C＝A＊B
どの計算をするか、1～3で選んで下さい。
 1・・・C＝A＋B  2・・・C＝A－B  3・・・C＝A＊B
どの計算をするか、1～3で選んで下さい。
 1・・・C＝A＋B  2・・・C＝A－B  3・・・C＝A＊B
どの計算をするか、1～3で選んで下さい。
 1・・・C＝A＋B  2・・・C＝A－B  3・・・C＝A＊B
どの計算をするか、1～3で選んで下さい。
 1・・・C＝A＋B  2・・・C＝A－B  3・・・C＝A＊B
どの計算をするか、1～3で選んで下さい。
 1・・・C＝A＋B  2・・・C＝A－B  3・・・C＝A＊B

答えは-35 です。
Ok
```

- 数値を入力すると、計算方法を選択するようメッセージが表示されます。
  inkey$により **1 ～ 3** の番号が入力された時点で、計算結果が表示されます。
  （この場合は 2 を選択しています。）

KEY （キー）

割り込みキーを定義します。

書　式

key〈ファンクションキー番号〉，〈文字列式〉　(1)

key〈割り込み番号〉，〈キーコード〉〔，〈キー属性〉〕
〔，〈キー属性マスク〉〕〔，〈タイミング〉〕　(2)

省略形

ke.

説　明

- （書式１）の場合
  - 20種のファンクションキー（F11～F20は［シフト］キーを押しながら［Ｆ１］～［Ｆ10］のキーを押します。）に対して、それぞれのキーが押されたときの入力文字列を定義します。
  - ＜キー番号＞は１～20で指定し、＜文字列式＞で押されたときの内容を指定します。＜文字列式＞は20キー合わせて300バイトまで定義できます。
  - klist文により、最下行ファンクション表示の制御や定義書式での定義内容の表示ができます。
- （書式２）の場合
  - キーボード上の指定したキーが押されたとき、または離されたときに割り込みを発生させるよう定義します。
    割り込みによって呼び出されるサブルーチンはon key文で指定し、実際にサブルーチンが呼び出されるかどうかはkey on文で設定を行います。
  - ＜割り込み番号＞は検出を行いたいキーの登録番号を１～20の値で指定します。on key文やkey on文で指定されるキー番号と同じものです。
  - ＜キーコード＞は検出を行いたいキー番号を以下の表に従って、１～127の値で指定します。

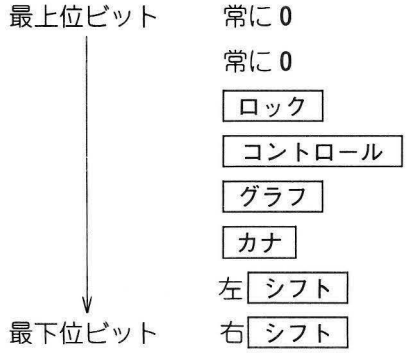
下位＼上位	0	1	2	3	4	5	6	7
0		(0)		0	@	P	↑	
1	F1	(1)	(=)	1	A	Q	↓	
2	F2	(2)		2	B	R	←	
3	F3	(3)		3	C	S	→	
4	F4	(4)		4	D	T	挿／抹	停止
5	F5	(5)		5	E	U	クリア／ホーム	ALGO
6	F6	(6)		6	F	V	※	コピー
7	F7	(7)		7	G	W	※	ヘルプ
8	F8	(8)		8	H	X	※	右シフト
9	F9	(9)		9	I	Y	※	左シフト
A	F10	(*)		:	J	Z		カナ
B		(+)	後退	;	K	[		GRAPH
C		(,)	タブ	,	L	¥	取消	コントロール
D		(−)	↵	−	M	]	前候補	ロック
E		(.)	ESC	.	N	^	変換	
F		(／)	SPACE	／	O	_	無変換	

( )で囲まれたキーは、テンキー上のキーです。
※のキーは指定できません。

<キーコード>は、上記の表に基づいて、以下の計算式にしたがって求めます。

<キーコード>＝(上位コード)＊16＋(下位コード)

<キー属性>と<キー属性マスク>は **0 ～255**の値で指定します。
指定形式は 8 ビットの 2 進数を10進数に変換して指定します。
各ビットは次のキーに対応しています。



<キー属性>と<キー属性マスク>の各ビットは次のような意味を持ちます。

＜キー属性＞

モードかどうかを判定する場合(ロック, GRAPH, カナキー)

0:そのモードでないときキーを検出

1:そのモードのときキーを検出

押されているかどうかを判定する場合(コントロール, 右シフト, 左シフトキー)

0:押されていないときキーを検出

1:押されているときキーを検出

＜キー属性マスク＞

0:＜キー属性＞の対応ビットを0にする

1:＜キー属性＞の設定に従って検出する

キーの検出チェックは、キーボードからの属性データと＜キー属性マスク＞のAND演算を行った結果と、＜キー属性＞とが等しいかどうかにより行います。

たとえば、コントロールキーが押されているときキー検出を行うようにする場合は、＜キー属性＞＜キー属性マスク＞共に(00010000=)16を指定します。

・＜キー属性＞の右シフトと左シフトの指定が共に1の場合は、どちらのシフトキーが押されても割り込みが発生します。

・＜キー属性＞、＜キー属性マスク＞を省略すると0が設定されます。

・＜タイミング＞には、キーを押されたときに検出するか、離れたときに検出するかを0または1の値で指定します。

0:キーが離されたときに検出します。

1:キーが押されたときに検出します。

・＜タイミング＞を省略すると1が設定されます。

・BASIC起動時には割り込みキー1～20は、ファンクションキーF1～F20の押されたことを検出するように設定されています。

例

```
key 2,"list"+chr$(5,13)
```

…… F2のキーに[l],[i],[s],[t],コントロール+[E],[↵]を定義します。定義後F2のキーを押すと"list"を表示して行末まで消去し、listコマンドを実行します。

参考

key list … ファンクションキーの定義状態を表示。(447ページ)

on key … 呼び出しサブルーチンの定義。(404ページ)

key on … キーによる割り込み制御。(406ページ)

サンプル

```
 10 rem --- KEY ---
 20 key 1,1,&H10,&H10,1
 30 key 2,1,&H3,&H3,1
 40 key 3,1,,,0
 50 on key gosub *ON1,*ON2,*OFF
 60 key (1) on:key (2) on:key (3) on
 70 A$=inkey$
 80 if A$="" then 70
 90 print A$
100 if A$=chr$(13) then end else 70
110 *ON1
120 print "コントロール＋Ｆ１":return
130 *ON2
140 print "シフト＋Ｆ１":return
150 *OFF
160 print "Ｆ１オフ":return
```

- コントロール キーが押されているときに Ｆ１ キーを押すと"コントロール+F1"と表示し、シフト キーが押されているときに Ｆ１ キーを押すと"シフト+F1"と表示します。
  また、Ｆ１ キーが離されると"Ｆ１オフ"と表示します。

KEY LIST／KLIST（キーリスト）

ファンクションキーの定義状態を表示します。

書　式

key　list
klist〔〈機能〉〕

省略形

ke. l.　kl.

説　明

- 文字画面の最下行ファンクションキー表示の制御やすべての定義状態の表示を行います。
- ＜機能＞は、0～2と指定なし（省略）の4種の表示機能があります。
  - klist 0とすると最下行の定義状態の表示を消します。
  - klist 1とすると最下行にファンクションキーの定義状態の表示が行われます。
  - klist 2とすると最下行に定義状態と時計を表示します。
- key listまたはklistのみの場合はすべてのキーの定義状態を表示します。表示はkey文による定義の書式になっていますので、スクリーンエディットにより定義の変更が簡単に行えます。
- 機能1、2を指定したとき最下行がconsole文による表示範囲に含まれる場合やスムーススクロールモードのときは表示されませんが、のちにラインスクロールで最下行がconsole文の指定範囲外になった場合は、klist 1またはklist 2の指定が行われていれば定義状態を表示します。
- 最下行の表示は、リバース（反転）文字で行われ、80桁表示のときは10個分、40桁表示のときは5個分の定義内容の表示になります。ただし機能2のときは時計表示に右端の1個分を使用しますので、定義の表示は1個分少なくなります。
- 定義文字中の文字コードが31（&H1F）以下のコントロールコードは、最下行表示の場合はCGの対応するフォントで表示し、定義書式による表示の場合は、chr$( )による表示になります。

例

```
klist 1
```

…… 最下行にファンクションキーの表示を行います。

参　考

key …… ファンクションキーの定義。（443ページ）

KEY 0 (キーゼロ)

キーボード入力のシミュレーションを行います。

書 式　key 0,〈文字列〉

省略形　ke.0

説 明

- キーの先行入力に用いられるキーバッファへあらかじめ文字を定義しておき、キー入力を自動的に行わせます。
- <文字列>があらかじめ定義しておく文字で64文字まで指定できます。<文字列>には文字コードが&H20以下のコントロールコードも定義することができます。
- シミュレーションは直接実行モード時、input文、line input文およびinkey$の実行時に行われます。
- 停止キー操作やシフト+停止キー操作、コントロール+シフト+挿入/抹消キー操作を行うとキーバッファは消去されますので注意が必要です。
- この命令を実行すると、それまでにキーバッファに残っているデータは消去され、新たに<文字列>の内容がキーバッファに入ります。

例

```
key 0,"list"+chr$(5,13)
```

…… 直接モードで実行すると、listコマンドを実行します。

```
key 0,""
```

…… キーバッファを消去します。

関数

STICK（スティック）

ジョイスティックの状態を調べます。

書　式　stick(〈スティック番号〉)

省略形　sti.

説　明

- ジョイスティックの状態を調べて値を返す関数です。
- <スティック番号>は、ジョイスティック1,2を次のように指定します。

  1 …… ジョイスティック1

  2 …… ジョイスティック2

スティックが倒されている方向により次の値を返し、スティックが倒されていない場合は、0を返します。

```
7     8     9
   ↖  ↑  ↗
4  ←  0  →  6
   ↙  ↓  ↘
1     2     3
```

例

```
if stick(1)=4 then print "Left"
```

…… ジョイスティック1が左に倒された場合、"Left"と表示します。

サンプル

```
 10 rem --- stick ---
 20 init "crt1:80,25":klist 0
 30 cls:on stop gosub *STOP
 40 stop on:X=40:Y=12
 50 gosub *STICK
 60 locate X,Y,0
 70 color 4:print "*";
 80 goto 50
 90 color 7:end
100 *STICK
110 S=stick(1)
120 if S=2 then Y=Y+1:if Y>23 then Y=23
130 if S=8 then Y=Y-1:if Y<1 then Y=1
140 if S=4 then X=X-1:if X<1 then X=1
150 if S=6 then X=X+1:if X>78 then X=78
160 return
170 *STOP
180 stop off:locate ,,1:cls 3:return 90
```

- ジョイスティックの倒されている方向(上下左右)をstickにより調べて、"*"を表示します。

STRIG (スティック トリガー・stick trigger)

ジョイスティックのトリガーボタンの状態を調べます。

書 式 strig(〈スティック番号〉)

省略形 str.

説 明

- ジョイスティックのトリガーボタンが押されているかを調べる関数です。
- <スティック番号>は、ジョイスティック1,2を次のように指定します。
  1 …… ジョイスティック1
  2 …… ジョイスティック2

- この関数の結果はトリガーボタンの状態で0～3の値になり、次のように対応します。

値	トリガーボタン1	トリガーボタン2	
0	OFF	OFF	ON…押されている
1	ON	OFF	OFF…押されていない
2	OFF	ON	
3	ON	ON	

STICK ON/OFF （スティック オン/オフ）

ジョイスティックでカーソルキーの代用をさせます。

書　式

stick { on / off }

省略形

sti.　o.　of.

説　明

- stick onとするとジョイスティック1をカーソルキーの代用として使用できます。
- stick関数を使用する場合は、stick offとして通常の状態に戻してください。

INIT "LPT:" (イニシャライズ エル ピー ティー・ initialize L.P.T.)

プリンタスプーラの設定をします。

書 式

init "LPT:[0 / 1 / 2]"

説 明

- init "MEM:"文で設定されたプリンタスプーラの機能設定を次のように指定します。(プリンタスプーラについては「第1部のプリンタ制御機能」を参照してください。)

  0 …… ダイレクトモード

  バッファの初期化を行い、ダイレクトモードにします。

  1 …… スプーラモード

  バッファの初期化を行い、スプーラモードにします。

  2 …… ダイレクトモード

  この指定を行う前にスプーラモードであった場合は、バッファ内のデータをすべてプリンタに出力したのちにダイレクトモードにします。

- 指定を省略してinit "LPT:"とするとバッファの初期化を行います。
- バッファの初期化を行うと、それまでにバッファ内に記憶されているデータは消去します。
- プリンタが画面のハードコピーなどのビットイメージ印字を行っている途中にバッファの初期化を行うと、以後のプリンタ印字が正常に行われない場合があります。このような場合、一度プリンタの電源を切ってプリンタ自身のリセットを行ってください。
- init "MEM:"文でスプーラ用メモリサイズを設定した直後は、0を指定したのと同様のダイレクトモードになります。

例

```
init "LPT:1"
```
…… スプーラモードにします。

参 考

init "MEM:" …… メモリディスクとスプーラのサイズを設定する。(284ページ)

option screen …… グラフィックV-RAMの一部をメモリディスクやプリンタスプーラに使用できるように設定する。(203ページ)

HCOPY（ハード コピー・hard copy）

画面の表示をプリンタにコピーします。

書　式　hcopy [〈機能〉][,〈左マージン〉][,〈倍率〉]

省略形　h.

説　明

- 画面に表示されている文字や図形をプリンタにコピー(印字)します。
- 〈機能〉は、コピーする画面を選択し、1～3の値で指定します。
  1…文字画面のみをコピーします。
  2…グラフィック画面のみをコピーします。
  3…文字画面とグラフィック画面の両方をコピーします。
  省略時は3の機能になります。
- プリンタへのコピーはディスプレイ表示の次の設定の影響を受けます。
  console@ 文による文字表示出力範囲。
  view@文によるグラフィック表示出力範囲。
  screen文による出力プレーン。
  グラフィック画面どうしの優先順位。
  color=文で設定されたカラーコード0。
- 〈左マージン〉はプリンタのヘッドのホームポジションからのドット数を指定します。0～32768の値で指定します。省略時は0となります。
- 〈倍率〉は1と2が指定できます。省略時1となります。
- 文字画面の40桁モード2画面重ね合わせ時、文字フォントの8×8ドット時、グラフィック画面の65536色モード時はハードコピーは行えません。

例　hcopy
…… ディスプレイに表示されている画面全体のグラフィックと文字をプリンタへコピーします。

サンプル

```
10 rem --- HCOPY ---
20 init "CRT2:640,400,16"
30 cls 3:pen 1
40 for I=0 to 639 step 60
50   for L=0 to 399 step 20
60     line (0,0)-(I,L),(L mod 16)+1
70   next
80 next
90 hcopy 2
100 end
```

LPOUT (ライン プリンタ アウトプット・line printer output)

プリンタにデータを出力します。

書 式 lpout 〈出力文字列〉

省略形 lp.

説 明

- lprint文の場合コントロールコードを送ろうとしても、漢字モードの切り替えやシフトJISコードからJISコードへの変換などを行っていますので、そのままデータをプリンタへ出力することはできません。この命令は<出力データ>の内容をそのまま、プリンタに出力するために用意されています。
- <出力データ>は文字列型の定数や変数、式で指定し、データの1バイトずつを文字コードとしてプリンタへ送り出します。また、lprint文と異なり、データの終了時に改行コードは送られませんので改行が必要な場合は、<出力データ>の中に改行コードを含める必要があります。
- 1つのlpout文でいくつものデータやコードを送る場合は、プラス(+)記号でつなぎます。セミコロン(;)を使うことはできません。

例

```
lpout chr$(&H1B)+"8"
```

…… プリンタへ&H1B、&H38の2バイトのデータを送ります。

参 考 lprint …… プリンタへの文字印字。(175ページ)

WIDTH LPRINT (ウィドス ライン プリント・width line print)

プリンタの印字幅を設定します。

書 式　width lprint 〈印字幅〉

省略形　wi. l ?

説 明

- プリンタに出力する場合の、1行に印字される文字数を指定します。
- ＜印字幅＞は1～255までの値で指定でき、1行に表示される半角文字の数で表します。ただし、＜印字幅＞を255にした場合は、この機能はなくなります。
- BASIC起動時の＜印字幅＞は255になっています。
- この命令はプリンタのもつ印字幅より幅のせまい用紙を使用した場合などに使用してください。

例　width lprint 72 …… 印字幅を72桁に設定します。

サンプル

```
   10 rem --- WIDTH LPRINT ---
   20 ST$="abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890 !#$% &'()=~|`{+*"
   30 for LOOP=20 to 80 step 10
   40   width lprint LOOP : lprint ST$
   50 next
   60 end
```

- 印字幅を20桁から80桁まで10桁きざみで変えて、文字をプリンタに出力します。

LPOS （ライン ポジション・line position）

現在のプリンタのヘッド位置を求めます。

書 式　lpos(〈数式〉)

説 明

- lprint文によるプリンタ印字で、次に文字を印字する位置がどこにあるのかを求める関数です。
- ＜数式＞はダミーで数値データであればどのような値でもかまいません。通常は0を指定します。
- この関数の値は、次の印字位置が左端から半角文字で何文字分進んだ位置にあるかを示します。したがって、左端にある場合は、値が0になります。
- この関数でのヘッド位置は論理的な位置になりますので、実際のプリンタの物理的なヘッド位置とは異なります。画面上のカーソル水平位置を求めるpos関数をプリンタに対応させたものとみることができます。

例

```
print lpos(0)
```

…… 現在のプリンタのヘッド位置を画面に表示します。

システム変数

TIME$ （タイム ダラー）

時刻の読み出しおよび設定を行います。

書 式

time$ …読み出し

time$="〈時〉:〈分〉:〈秒〉" …設定

省略形

ti.

説 明

- 内蔵のリアルタイムクロックに対するシステム変数で、現在時刻を文字列型データとして持ち、代入を行うことにより設定ができます。
- 時刻は24時間で8文字の文字列型データとなり、<時>、<分>、<秒>それぞれ2桁ずつで表し、句切りにコロン(:)を使用します。

例

```
print time$ …… 現在時刻を表示します。
time$="18:05:00"
    …… 現在時刻を午後6時5分0秒に設定します。
```

参 考

date$ …… 日付の読み出しと設定。(458ページ)

day$ …… 曜日の読み出しと設定。(459ページ)

on time$ gosub …… 時刻による割り込み処理。(418ページ)

サンプル

```
 10 rem --- TIME$ ---
 20 init "CRT2:640,200,16"
 30 interval 50:klist 0:cls 3
 40 on stop gosub *STOP
 50 on interval gosub *SYMBOL
 60 interval on:stop on
 70 symbol (0,0),"現在の時刻は・・・",4,5,11
 80 view (80,100)-(639,199)
 90 goto 90
100 end
110 *SYMBOL
120 interval off:cls 3
130 symbol (80,0),time$,7,5,15
140 interval on:return
150 *STOP
160 stop off
170 init "CRT2:":cls 3
180 return 100
```

DATE$ （デート ダラー）

日付の読み出しおよび設定を行います。

書 式

date$ …読み出し

date$="〈年〉/〈月〉/〈日〉" …設定

説 明

- 内蔵のリアルタイムクロックに対するシステム変数で、現在の日付を文字列型データとしてもち、代入することにより設定ができます。曜日は自動的に計算され設定されます。
- 日付は8文字の文字型データとなり、＜年＞、＜月＞、＜日＞それぞれ2桁ずつで表し、句切りにスラッシュ(/)を使用します。
- time$が"23：59：59"から"00：00：00"になるときに自動的に次の日付に更新します。また、うるう年の処理も行いますので年は西暦の下2桁で指定してください。

例

```
print date$ …… 現在の日付を表示します。
date$="87/10/01" …… 日付を1987年10月1日に設定します。
```

参 考

day$ …… 曜日の読み出しと設定。(459ページ)

time$ …… 時刻の読み出しと設定。(457ページ)

DAY$ （デイ ダラー）

曜日の読み出しおよび設定を行います。

書 式

day$ …読み出し

day$＝"〈曜日〉" …設定

説 明

- 内蔵のリアルタイムクロックに対するシステム変数で、現在の曜日を文字列型データとして持ち、代入を行うことにより設定ができます。
- ＜曜日＞は3文字の文字列型データで表します。読み出し時は1文字目が大文字であとは小文字になりますが、指定は大文字、小文字のどちらでも使用できます。

Ｓｕｎ	日曜日
Ｍｏｎ	月曜日
Ｔｕｅ	火曜日
Ｗｅｄ	水曜日
Ｔｈｕ	木曜日
Ｆｒｉ	金曜日
Ｓａｔ	土曜日

- time$が"23：59：59"から"00：00：00"になるときに自動的に更新します。

例

```
print day$ …… 曜日を表示します。
day$="SAT" …… 土曜日に設定します。
```

参 考 date$ …… 日付の読み出しと設定。(458ページ)

time$ …… 時刻の読み出しと設定。(457ページ)

TIME （タイム）

タイマーの読み出しおよび設定を行います。

書 式

time　　…読み出し
time＝〈数式〉　…設定

説 明

- 内蔵タイマーに対するシステム変数で0.1秒単位の値をもち、＜数式＞の値を代入することにより設定ができます。
- タイマーは0.1秒単位で**0～65535**までの値をもちます。
- タイマーの値が**65535**になるとそのまま停止します。

例

```
print time　……　現在のタイマーを表示します。
time=0　……　タイマーを0に設定します。
```

サンプル

```
  10 rem --- TIME ---
  20 cls
  30 print "＊＊キー タイプ練習プログラム＊＊":print
  40 for LOOP=1 to 20
  50   EX$=chr$(int(rnd(1)*62)+161)
  60   if EX$="" then 50
  70   locate LOOP*2,3:print EX$;:time=0
  80   KY$=inkey$:if KY$="" then 80
  90   if KY$=EX$ then locate 20,8:print using "所要時間＝###.#秒",
time/10:goto 110
 100   beep:goto 80
 110   pause 10
 120 next
 130 end
```

- 表示されたカナ文字と同じものをキー入力するまでの所要時間を**time**変数により算出して表示します。

BEEP （ビープ）

スピーカを一瞬鳴らします。

書　式　beep

省略形　b.

説　明
- この命令を実行するとスピーカから一瞬、「ピッ」という音が出ます。
- この音は文字コード7で発生する音（BELL）と同じです。

例　beep …… 「ピッ」という音を出します。

PLAY （プレイ）

音楽演奏をします。

書　式

play 〈パート１〉[,〈パート２〉]…[,〈パート６〉]
play init
play wait

省略形

pl.

説　明

- ＜パート１＞から＜パート６＞による音楽演奏や演奏機能に関する制御を行います。
- ＜パート＞は音階や音長、オクターブ、音量、音色などの演奏データを文字列型データで表します。＜パート１＞～＜パート３＞はFM音源のチャンネル1～3に、＜パート４＞～＜パート６＞はSSG音源のチャンネルA, B, Cに対応し、同時に演奏が行われます。(演奏データの詳細は次ページ以降を参照してください。)
- ＜パート1～6＞の演奏はplay文の実行とともに、同時に行われます。
  各＜パート＞の演奏時間が異なる場合の演奏時間は、一番長い＜パート＞の演奏時間となり、演奏時間の短い＜パート＞は休符が続くものとします。
- 演奏の実行は、各＜パート＞の指定とともに開始され、プログラムは次の処理に進みますが、演奏中(以前のplay文の実行による演奏が終了していない)であれば演奏の終了を待ち、演奏の終了後に各＜パート＞の設定を行い演奏を開始し、次のプログラム処理に進みます。また、音楽演奏の終了時に割り込みをかける機能があり、on play gosub文で処理の定義、play on/off/stop文で処理の制御を行います。
- play initはオクターブや音長、音色などの初期化を行います。

＜パート＞	初期化に相当する演奏データ
1～3	"O４L４T１２０Q８@０@V１００@M０"
4～6	"O４L４T１２０Q８V１２M２５５"

- play waitはこの命令の実行時に音楽が演奏中であれば、演奏の終了を待ちます。通常は各パートを指定して演奏を開始するとすぐに次の処理に進み、プログラム処理と同時に演奏が行われますが、音楽演奏時にこの命令を実行すると音楽演奏が終了するまで待ち、次の処理に進みます。

＜パート＞の演奏データは次のようになります。

機能一覧

文 字	機能
C～G, A, B	音階
＋, ＃	音階を半音上げる
－	音階を半音下げる
Rℓ	休符(ℓ:1～32)
.	音長を1.5倍にする
Nx	音階コードxによる音の発生(x:0～95)
Qn	音の出る時間比率の設定(n:1～8)
&	前後の音を継ぐ(タイ)
{ }ℓ	連符。音長ℓを{ }内の音符数で割った音を発生させる(ℓ:1～32)
Lℓ	省略時の音長指定(ℓ:1～32)
Ts	テンポ指定(s:32～255)
On	オクターブ指定(n:1～8)
＞	オクターブを1つ上げる
＜	オクターブを1つ下げる
@Vn	FM音源の音量設定(n:0～127)
@n	FM音源の音色設定(n:0～29)
Vn	SSG音源の音量設定(n:0～15)
Sn	SSG音源のエンベロープ形状設定(n:0～15)
Mn	SSG音源のエンベロープ周期設定(1～16383)
Yr, d	OPNのレジスタrにデータdをセット
@Wℓ	ℓの長さだけ状態を維持する(ℓ:1～32)
@Mn	FM音源チャンネル3のモード設定(n:0～2)
＝＜変数名＞;	値を変数で指定する

FM音源はチャンネル1～3、SSG音源はチャンネル4～6になります。

- 音符の指定

＜音階＞〔{ ＃ ＋ － }〕〔**＜音長＞**〕〔**.**〕

音階はアルファベットのC～G, A, Bを使用して次のように対応します。

C	D	E	F	G	A	B
ド	レ	ミ	ファ	ソ	ラ	シ

音階に続いて＃または＋を付けると半音上がります。Bの半音上がった音は1オクターブ上のCになります。

音階に続いて－を付けると半音下がります。Cの半音下がった音は1オクターブ下のBになります。

音長は**1～32**で指定し、省略した場合は**L**で設定された音長になります。音長は全音符を１として次のように対応します。

全音符	2分音符	4分音符	8分音符	16分音符	32分音符
1	2	4	8	16	32

ピリオド(**.**)を付けると音長を1.5倍にします。

但し１の場合にはピリオド(**.**)を指定しても音長は1.5倍になりません。この時には**&**(タイ)を使用してください。(例 **play "C1&C2"**)

- 休符の指定

R〔**＜音長＞**〕〔**.**〕

＜音長＞とピリオドの機能は音符の指定と同じで、＜音長＞の省略時もLによる指定になります。この間は音が発生しませんが音色によっては前の音の余韻が残る場合があります。(エンベロープの**RR**(Release Rate)の設定によります。)

- 連符の指定

{ }〔**＜音長＞**〕

{ }内の音符を、＜音長＞を音符数で等分に割った長さで演奏します。＜音長＞の省略時は**L**で設定されている長さになります。音長を{ }内の音符数で割った値が**32**の指定より短かい場合はエラーになります。

- 音を続ける(タイ)

&

この記号の前後の音を続けて１つの音とします。前後の音程が異なる場合は前の音だけがQ8の指定をされたのと同じになります。

- 音の発生比率の設定

Q＜比率＞

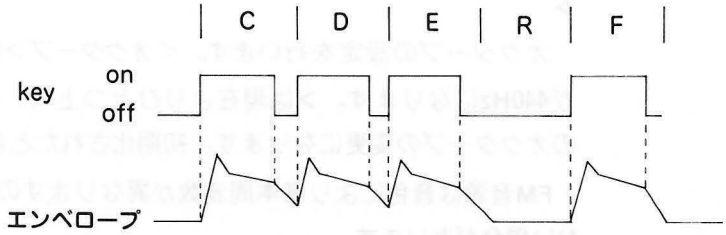
１つの音符で、その音長の間で実際に音を出している時間の指定です。＜比率＞は1～8で指定し、次のように対応しています。

指定	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
音の長さ	1/8	2/8	3/8	4/8	5/8	6/8	7/8	8/8

＜比率＞を小さくするとスタッカートになります。

音の発生をkey-on、停止をkey-offと呼びます。エンベロープとの関係は次のようになります。

例）play "Q6CDERF" とした場合



- 音階コードによる音の発生

N＜音階コード＞

＜音階コード＞による音階の音を発生します。音長はLにより設定された長さになります。

＜音階コード＞は0～95で指定し次のようになります。

	O1	O2	O3	O4	O5	O6	O7	O8
C	0	12	24	36	48	60	72	84
C＋（D－）	1	13	25	37	49	61	73	85
D	2	14	26	38	50	62	74	86
D＋（E－）	3	15	27	39	51	63	75	87
E	4	16	28	40	52	64	76	88
F	5	17	29	41	53	65	77	89
F＋（G－）	6	18	30	42	54	66	78	90
G	7	19	31	43	55	67	79	91
G＋（A－）	8	20	32	44	56	68	80	92
A	9	21	33	45	57	69	81	93
A＋（B－）	10	22	34	46	58	70	82	94
B	11	23	35	47	59	71	83	95

- 省略時の音長設定

L＜音長＞

音符や休符、N、{ }で音長を省略したときの音長を設定します。＜音長＞は1～32で音符の指定と同じです。初期化されたときは"L 4 "になります。

- テンポの設定

  T＜スピード＞

  ＜スピード＞に1分間の4分音符の数を指定します。＜スピード＞は32～255までの値になります。初期化されたときは"T120"になります。

- オクターブの設定

  O＜オクターブ＞

  ＜

  ＞

  オクターブの設定を行います。＜オクターブ＞は1～8で指定し、"O4A"が440Hzになります。＞は現在よりひとつ上へ、＜は現在よりひとつ下へのオクターブの変更になります。初期化されたときは"O4"になります。

  FM音源は音色により基本周波数が異なりますので"O4A"が440Hzにならない場合があります。

- FM音源の音量の設定

  @V＜音量＞

  FM音源のパート内で使用し、そのパートの音量を設定します。＜音量＞は0～127までで指定し、127が最大音量になります。初期化されたときは"@V100"になります。この指定はSSG音源のパートでは無視します。

- FM音源の音色の設定

  @＜音色＞

  FM音源のパート内で使用し、そのパートの音色をあらかじめ用意された30種の音色の中から選びます。＜音色＞は0～29までで指定し、初期化されたときは"@0"になります。この指定はSSG音源のパートでは無視します。＜音色＞は音色表を参照してください。

- SSG音源の音量設定

  V＜音量＞

  SSG音源のパート内で使用し,そのパートの音量を設定します。＜音量＞は、0～15で指定し、"V15"が最大音量になり"V0"は音が出ません。初期化されたときは"V12"になります。この指定を行うとSによるエンベロープ形状の設定は無効になります。この指定はFM音源のパートでは無視します。

- SSG音源のエンベロープの設定

  SSG音源のエンベロープの設定は各パート別に行うことはできません。

  S＜形状番号＞

  SSG音源のパート内で使用し、エンベロープ形状を設定します。＜形状番号＞は、0～15で指定します。形状番号によるエンベロープ形状はsound文を参照してください。この指定を行うとVによる音量は無効になります。この指定はFM音源のパートでは無視します。

M＜周期＞

SSG音源のパート内で使用し、エンベロープ周期を設定します。＜周期＞は、1～16383までで指定し、16383がいちばん周期が長くなります。エンベロープの周期は＜周期＞を7812.5で割った値が秒単位の周期になります。

この指定はFM音源のパートでは無視します。初期化された場合は"M255"になります。

- OPNのレジスタに値を設定する

Y＜レジスタ番号＞,＜データ＞

OPNの内部レジスタへ直接データを設定します。＜レジスタ番号＞は0～180、＜データ＞は0～255で指定します。値を16進数で指定する場合は$または&Hをつけてください。

- 状態の維持

@W〔＜長さ＞〕

＜長さ＞で指定された時間だけ現在の状態を保ちます。＜長さ＞は、音符の音長と同じで省略時は、Lの設定値となります。この指定はYによりレジスタを設定したあとに使用し、設定値による音を出すために用意されています。(key-onはレジスタ設定により行ってください。)

- FM音源のチャンネル3のモード切り替え

@M＜モード番号＞

FM音源のチャンネル3はモード切り替えが可能でパート3でこの指定を行います。＜モード番号＞は次のように対応します。

0 …… 通常の音楽モード
1 …… CSM(サイン波合成)モード
2 …… 効果音モード

この指定はSSG音源のパートでは無視します。

- 値の変数による指定

＝＜変数名＞;

演奏データの中での数値指定を行う所で使用し、そのときの指定された変数の値がその位置で使用されます。

例)
```
10 for A=10 to 120 step 10
20 play "@V=A;CDE"
30 next
```

…… 音量を10ステップで上げながらドレミの音を出します。

• 音色表

play文の"@n"の指定やtone copy文で指定される音色コードです。

音色番号	音　　色	
0	HARPSIC	ハープシコード
1	BRASS 1	ブラス系の音
2	BRASS 2	
3	TRUMPET	トランペット
4	STRING 1	ストリング系の音
5	STRING 2	
6	EPIANO 1	エレクトリックピアノ
7	EPIANO 2	
8	EPIANO 3	
9	GUITAR	ギター
10	EBASS 1	エレクトリックベース
11	EBASS 2	
12	EORGAN 1	エレクトリックオルガン
13	EORGAN 2	
14	PORGAN 1	パイプオルガン
15	PORGAN 2	
16	FLUTE	フルート
17	PICCOLO	ピッコロ
18	OBOE	オーボエ
19	CLARINET	クラリネット
20	GROCKEN	グロッケン
21	VIBRPHN	ビブラホン
22	XYLOPHN	シロホン
23	KOTO	琴
24	ZITAR	チター
25	CLAV	クラビネット
26	BELL	ベル
27	HARP	ハープ
28	HARMONICA	ハーモニカ
29	TIMPANI	ティンパニー

注）音色により、基本周波数が異なりますので注意してください。("O4A"の音が440Hzにならない場合があります。)

音色により、音量が異なりますので複数の音を演奏して使用される場合は；音色ごとに@Vnで音量を調整してください。

TONE （トーン）

FM音源の音色を設定します。

書 式　tone〈整数配列名1〉〔,〈整数配列名2〉〕〔,〈整数配列名3〉〕

説 明

- 整数型の配列に設定されている音色データをFM音源の各チャンネルに設定します。＜整数配列名＞の1～3はそれぞれFM音源の1～3チャンネルに対応します。また配列は2次元配列で添字の下限を**0(option base 0)**として**(4,9)**の大きさが必要です。
- 配列内のデータは引数の右順が0のときは全体的な設定で、1～4はそれぞれのオペレータの指定になり、次の内容を設定します。

n	(0,n)	(1,n)～(4,n)
0	Feedback/Algorithm	Attack Rate
1	Operator mask	Decay Rate
2	Wave Form	Sustain Rate
3	Sync	Release Rate
4	Speed	Sustain Level
5	Pitch Modulation Depth	Output Level
6	Amplitude Modulation Depth	Keyboard Rate Scaling Depth
7	未使用	Multiple
8		Detune
9		Amplitude Modulation Sensitvity

（設定の内容は次ページを参照してください。）

例

```
tone PIANO%,BELL%
```

…… FM音源のチャンネル1に整数型配列**PIANO%(4,9)**よりチャンネル2は整数型配列**BELL%(4,9)**より指定値を設定します。

参 考　tone copy …… BASICで用意された音色データを配列に取り込む。(475ページ)

(0, 0);Feedback/Algorithm

以下の意味をもつ6ビットのデータになっています。

D7	D6	D5	D4	D3	D2	D1	D0
未　使　用		Feedback			Algorithm		

Feedback(0～7)

第1オペレータの変調度を設定します。

第1オペレータは自分自身の出力を変調信号としているため、この変調度をこれにより制御します。

	0	1	2	3	4	5	6	7
変調度	OFF	π/16	π/8	π/4	π/2	π	2π	4π

Algorithm(0～7)

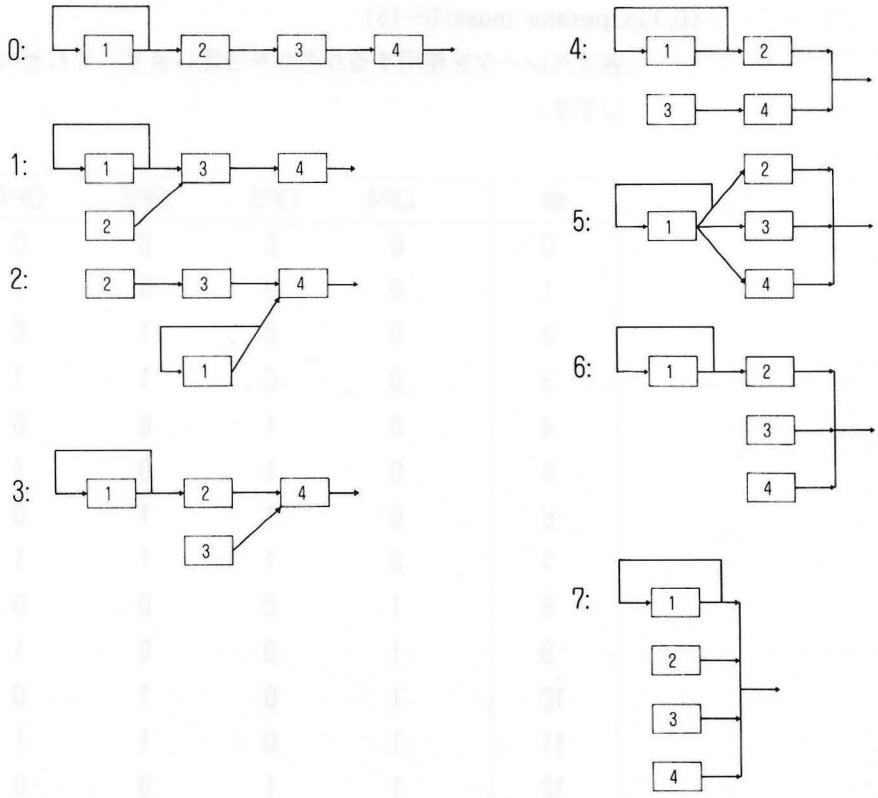
4つのオペレータの接続のしかたを設定します。値はそれぞれ次のようなアルゴリズムに対応します。

なお、変調をかけるオペレータをモジュレータ、変調をかけられるオペレータをキャリアと呼びます。ただし、第1オペレータだけは自分自身の出力を変調信号としています。

たとえば、アルゴリズム4の場合は1、3がモジュレータ、2、4がキャリアとなります。

一般に、モジュレータの数が多いと変調が多くかかるので、複雑な波形を持つ明るく鋭い音色になります。

また、キャリアの数が多いと音源の数が増えることになるので、厚みのある音色になります。



・0、1、2、3のアルゴリズムは、変調が多くかかるので、明るく鋭い音色を作るのに向いています。
また、複雑な波形が作り出せるので、ソロ楽器に向いたアルゴリズムです。

・4のアルゴリズムは、2つのキャリアを使って幅広い音色を作り出すことができます。2つのキャリアのピッチをずらして、コーラス効果を生むこともできます。

・5、6、7のアルゴリズムは、キャリアが3つまたは4つあり、落ち着いた厚みのある音色を作り出すことができます。
特に7のアルゴリズムはオルガン系の音色を作り出すのに向いています。
各キャリアのピッチをずらすと、コーラス効果を生むことができます。

(0,1):Operator mask(0～15)

各オペレータを使用するか否かを設定します。それぞれの値は次のとおりです。

値	OP4	OP3	OP2	OP1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

0…使用しない　　1…使用する

(0,2):Wave Form (0～3)

0　ノコギリ波

1　矩形波

2　三角波

3　ランダム

(0,3):Sync

key onとLFO(Low Frequency Oscillator)の同期を指定します。

0　Key-onと無関係にLFOが動作します。

1　Key-onと同期してLFOが動作します。

(0,4):Speed(0～255)

LFOの速度を決めます。大きいほど速くなります。

(0,5):Pitch Modulation Depth(－127～127)

音階に対してLFOをかける深さを設定します。絶対値が大きいほど深くかかります。負の値にすると、波形の極性が逆になります。

(0, 6):Amplitude Modulation Depth(−127〜127)

音量に対してLFOをかける深さを設定します。絶対値が大きいほど深くかかります。負の値をとると、波形の極性が逆になります。

(0, 7):未使用

(0, 8):未使用

(0, 9):未使用

OP=1, 2, 3, 4(オペレータ番号)

(OP, 0):Attack Rate(0〜31)

(OP, 1):Decay Rate(0〜31)

(OP, 2):Sustain Rate(0〜31)

(OP, 3):Release Rate(0〜15)

(OP, 4):Sustain Level(0〜15)

(OP, 5):Output Level(127〜0)

これは減数量の指定で、0が最大出力になります。

(OP, 6):Keyboard Rate Scaling Depth(0〜3)

高音になるほどEGの変化速度を速くする度合を設定します。

(OP, 7):Multiple(0〜15)

周波数比を設定します。0で$\frac{1}{2}$になります。

(OP, 8):Detune(−3〜3)

音階のずれの度合を設定します。

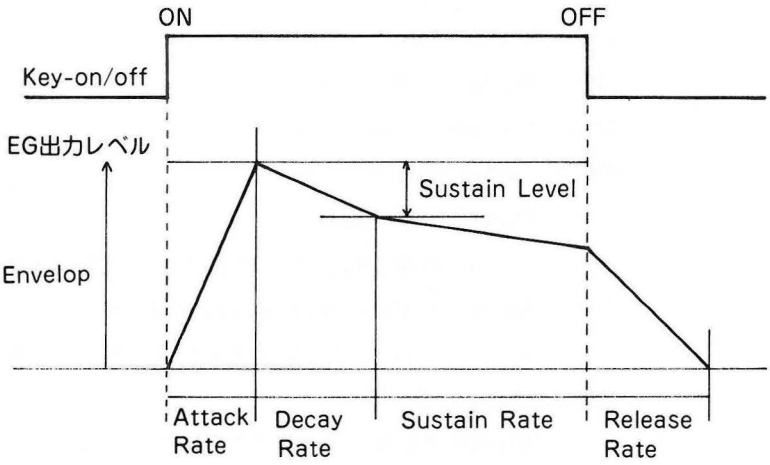
(OP, 9):Amplitude Modulation Sensitivity(0〜2)

音量に対するLFOをかける度合を設定します。数値が大きいほど変化の幅が大きくなります。

EG(Envelop Generator)

楽器の音は時間とともに音量、音色が変化します。この時間的な変化をエンベロープといいます。そして、このエンベロープを人工的に作る機能をEG(Envelop Generator)といいます。

EGは以下のパラメータをもっており、それを設定することによって、より自然の楽器に近づけることができます。



AR(Attack Rate)0～31

音が発生(Key-on)してからEGの出力レベルが最大になるまでの時間を設定します。数値が大きいほど短くなります。

DR(Decay Rate)0～31

EGの出力レベルが最大になってからSustain Levelに達するまでの時間を指定します。数値が大きいほど短くなります。

SR(Sustain Rate)0～31

Sustain LevelからEGの出力レベルが0になるまでの時間を設定します。数値が大きいほど短くなります。

RR(Release Rate)0～15

Keyを離してからEGの出力レベルが0になるまでの時間を設定します。数値が大きいほど短くなります。

SL(Sustain Level)0～15

DecayからSustainに移るときのレベル(減衰量)を設定します。数値が大きいほどEGの出力レベルが低くなります。

TONE COPY（トーン コピー）

BASICで用意された音色データを配列に取り込みます。

書　式　tone copy 〈音色番号〉,〈整数型配列名〉

省略形　ton. cop.

説　明

- ＜音色番号＞で指定された音色のデータを＜整数型配列名＞で設定された配列に取り込みます。
- ＜音色番号＞はplay文で使用される音色に対応し、0～29の値で指定します。（play文の音色表を参照してください。）
- 音色データを取り込む配列はあらかじめdim文で(4,9)の大きさで定義しておきます。また、添字の下限は0(option base 0)に設定してください。
- 配列内のデータの内容は、tone文でFM音源に設定する値と同じです。

例

```
tone copy 27,HARP%
```

…… 音色番号27の音色データを(4,9)の大きさの整数型配列変数HARP%に取り込みます。

参　考　tone …… 音色データをFM音源に設定する。(469ページ)

サンプル

```
 10 rem --- TONE COPY ---
 20 option base 0
 30 dim NEIRO%(4,9)
 40 tone copy 1,NEIRO%
 50 for I=0 to 4
 60   for L=0 to 9
 70     print "(";I;",";L;")";NEIRO%(I,L),
 80   next
 90 next
100 end

run
( 0 , 0 ) 58        ( 0 , 1 ) 15        ( 0 , 2 ) 2         ( 0 , 3 ) 1
( 0 , 4 ) 200       ( 0 , 5 ) 1         ( 0 , 6 ) 0         ( 0 , 7 ) 0
( 0 , 8 ) 0         ( 0 , 9 ) 0         ( 1 , 0 ) 13        ( 1 , 1 ) 6
( 1 , 2 ) 2         ( 1 , 3 ) 8         ( 1 , 4 ) 1         ( 1 , 5 ) 27
( 1 , 6 ) 0         ( 1 , 7 ) 1         ( 1 , 8 ) 0         ( 1 , 9 ) 1
( 2 , 0 ) 15        ( 2 , 1 ) 8         ( 2 , 2 ) 0         ( 2 , 3 ) 8
( 2 , 4 ) 1         ( 2 , 5 ) 43        ( 2 , 6 ) 0         ( 2 , 7 ) 3
( 2 , 8 ) 2         ( 2 , 9 ) 1         ( 3 , 0 ) 21        ( 3 , 1 ) 14
( 3 , 2 ) 0         ( 3 , 3 ) 8         ( 3 , 4 ) 1         ( 3 , 5 ) 34
( 3 , 6 ) 0         ( 3 , 7 ) 1         ( 3 , 8 ) 0         ( 3 , 9 ) 1
( 4 , 0 ) 18        ( 4 , 1 ) 3         ( 4 , 2 ) 0         ( 4 , 3 ) 8
( 4 , 4 ) 2         ( 4 , 5 ) 0         ( 4 , 6 ) 1         ( 4 , 7 ) 1
( 4 , 8 ) 1         ( 4 , 9 ) 1
Ok
```

- 音色番号1(BRASS 1)のデータをtone copyにより配列に取り込み表示します。

TONE LFO (トーン エル エフ オー・tone Low Frequency Oscillator)

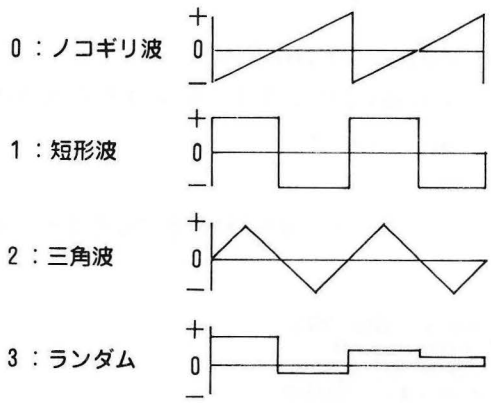
ビブラートやトレモロ機能を設定します。

書　式　tone lfo 〈チャンネル〉[,〈波形〉][,〈同期〉][,〈スピード〉][,〈PMD〉][,〈AMD〉]

省略形　ton. lf.

説　明

- LFOとは(Low Frequency Oscillator)の略で、低い周波数を発生させ、ビブラート(音程を細かく変化させる)やトレモロ(音量を細かく変化させる)の周期に使用します。この命令では、LFOの周期や波形、変調の深さを設定します。
- <チャンネル>は、play文のパート番号1～6で指定します。
- <波形>は、LFOの変化を次の波形0～3選択します。



- <同期>は1とすると音の発生(key-on)と同時にLFO波形を発生させ、0とすると音の発生に関係なくLFO波形を発生します。
- <スピード>は0～255でLFOの周波数を指定し、値が大きくなるほど周波数が高くなります。
- <PMD>(Pitch Modulation Depth)の値は－127～127で、音程に対してLFOをかける深さを指定します。この絶対値が大きいほど変化の幅が大きくなります。負の値を指定した場合、LFOの波形の極性が逆転します。(ビブラートをかけるのに使います。)

- <AMD>(Amplitude Modulation Depth)の値は−127〜127で、音量に対してLFOをかける深さを指定します。この絶対値が大きいほど変化の幅が大きくなります。負の値の場合、LFO波形の極性が逆転します。(トレモロをかけるのに使います。)

例

```
tone lfo 1,0,1,128,0,64
```

…… チャンネル1をノコギリ波で同期し、スピードは128、ビブラートをかけずトレモロを64の深さに設定します。

SOUND （サウンド）

SSGレジスタをセットします。

書　式　sound 〈レジスタ番号〉,〈データ〉

省略形　so.

説　明

- 本体内のOPN(音楽用IC)のレジスタへデータを書き込み、OPNの直接制御を行います。
- ＜レジスタ番号＞は0～179(&H0～&HB3)まであり、＜データ＞は8ビットで、0～255(&H0～&HFF)の値で指定します。
- SSG音源のレジスタは次のとおりです。

レジスタ番号	レジスタ機能	ビット構成							
		7	6	5	4	3	2	1	0
0	チャンネルA周波数	下位8ビットT_L							
1						上位4ビットT_H			
2	チャンネルB周波数	下位8ビットT_L							
3						上位4ビットT_H			
4	チャンネルC周波数	下位8ビットT_L							
5						上位4ビットT_H			
6	ノイズ周波数				5ビット				
7	チャンネル設定	$\overline{in}$/out		ノイズ			トーン		
		IOB	IOA	C	B	A	C	B	A
8	チャンネルA音量				M	4ビット			
9	チャンネルB音量				M	4ビット			
10	チャンネルC音量				M	4ビット			
11	エンベロープ周期	下位8ビットE_L							
12		上位8ビットE_H							
13	エンベロープ形状					4ビット			

各チャンネルの周波数は2MHzを16で割りさらにT_HとT_Lの12ビット値で割った値になります。

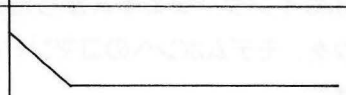
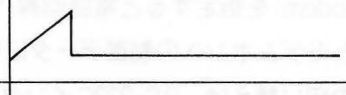
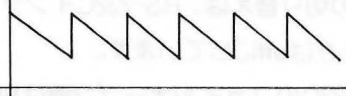
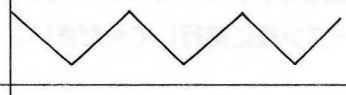
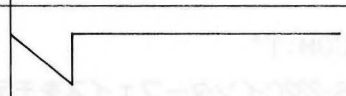
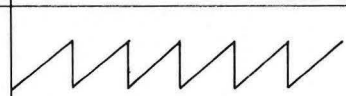
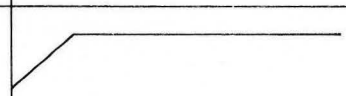
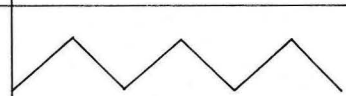
ノイズ周波数は2MHzを16で割りさらに5ビットの値で割った値になります。

チャンネル設定のIOBは0のインプット、IOAは1のアウトプットに固定され、ノイズとトーンのチャンネルA, B, Cはそのビットが0であれば選択になります。

エンベロープ周期は2MHzを256で割りさらにE_H とE_L の16ビットデータで割った値の周波数による周期になります。

各チャンネルの音量は4ビットで0～15の値になりますがMのビットが1の場合は音量がきかず、エンベロープモードになります。

エンベロープ形状

R13の値	エンベロープパターン
0, 1, 2, 3, 9	
4, 5, 6, 7, 15	
8	
10	
11	
12	
13	
14	

TEL （テレフォン・telephone）

モデムホンへの制御を切り替えます。

書　式　　tel "COM〔1〕:{M / T}"

説　明

- RS-232Cインターフェイスから別売のモデムホン(MZ-1X19)に出力するデータを、モデムホンへのコマンドとするか、伝送データとするかを切り替えます。
- M(Modem)を指定すると電話回線への伝送、T(Telephone control)を指定するとモデムホンへの制御データになります。
- 制御の切り替えは、RS-232CインターフェイスのER信号を、Mのときはon、Tのときはoffにしています。
- RS-232Cのファイルオープン時は**ER**を**on**にしますのでこの命令は、ファイルオープン後に実行してください。

例

```
tel "COM:T"
```

……RS-232Cインターフェイスをモデムホンへのコマンド状態にします。

INIT "CRT4:" (イニシャライズ シー アール ティー 4・initialize C,R,T,4)

ディスプレイテレビの制御を行います。

書　式　init　"CRT4:　〈機能〉"

説　明

- 専用ディスプレイテレビ(**MZ-1D24**)の制御を行います。
- ＜機能＞は電源のON/OFF、音量、表示画面、チャンネルなどの制御データを文字列型データで表します。
  (制御データの詳細は次の表を参照してください。)
- この命令の実行によって専用ディスプレイテレビの電源をONにしたときは、テレビ放送が表示されます。コンピュータ画面を表示するときは、ディスプレイテレビのテレビ／外部・コンピュータ入力切り替えスイッチを押すか、**init"CRT4:S1"**を実行してコンピュータ画面にする必要があります。
- テレビ映像とコンピュータ画面の重ね表示(スーパーインポーズ)をする場合は、本体後面システムスイッチのスイッチ**SW 1 , 2** をonにします。
- 指定するチャンネル番号は放送局のチャンネル番号とは関係ありません。くわしくは、専用ディスプレイテレビの取扱説明書を参照してください。

＜機能＞の制御データは次のようになります。

機能一覧

文字	機能
PO	電源をONにします。
PF	電源をOFFにします。
V＋	音量を大きくします。
V－	音量を小さくします。
VN	標準の音量にします。
VM	消音します。 （実行すると音声が消えます。もう一度実行すると元の音量に戻ります。）
S0	テレビの映像を表示します。
S1	コンピュータの画面を表示します。
S2	テレビ映像のコントラストを下げて、コンピュータの画面を重ねて表示します。
S3	テレビ映像とコンピュータの画面を同じコントラストで重ねて表示します。
C1 ～ C12	テレビのチャンネルを指定のチャンネルに切り換えます。

例

```
init "CRT4:POVNS3C5"
```

…… 専用ディスプレイテレビの電源をONにして、テレビ映像とコンピュータ画面を同じコントラストで重ねて表示します。また、テレビのチャンネルは5番に切り替え、音量は標準に設定します。

関数

PEEK （ピーク）

メモリよりデータを読み出します。

書　式

peek(〈アドレス〉)　　…(1)

peek@(〈ブロック番号〉,〈オフセット〉)　　…(2)

説　明

- メモリより1バイトのデータを読み出す関数です。
- 書式(1)は機械語エリア使用時のメモリマップによる＜アドレス＞を指定しての読み出しです。
  読み出し時のセグメントは、**def seg**文で指定します。
- 書式(2)は＜ブロック番号＞に64キロバイトのメモリブロック番号と、ブロック内のアドレスに相当する＜オフセット＞を指定して読み出します。
  ＜ブロック番号＞とシステムメモリの関係は次のようになります。

システムメモリ	＜ブロック番号＞	
メインメモリ	0〜11	
グラフィックV-RAM	12・13	B(青)の画面(16色モード)
	14・15	R(赤)の画面(16色モード)
	16・17	G(緑)の画面(16色モード)
	18・19	I(輝度)の画面(16色モード)
辞書ROM	20〜23	
PCG	52	
漢字ROM	53〜180	
テキストV-RAM	181	
増設RAM	198〜389	
IPL用ROM	421	

- ＜アドレス＞は-32768〜65535(&H0〜&HFFFF)、＜ブロック番号＞は0〜421(&H0〜&H1A5)、＜オフセット＞は-32768〜65535(&H0〜&HFFFF)の値で指定します。
- ＜アドレス＞は、整数型データで指定しますので、10進数では-32768〜32767の値になりますが、この命令では32768〜65535の値で指定することもできます。32768〜65535は、整数型データの-32768〜-1(&H8000〜&HFFFF)に対応します。
- PCG、漢字ROMのメモリエリアをアクセスする場合は、＜オフセット＞に偶数値を指定します。＜オフセット＞に奇数値を指定すると1を減じた偶数値が設定されます。なお、＜オフセット＞は0〜16382(&H0〜&H3FFE)の値で指定します。
- テキストV-RAM、IPL用ROMのメモリエリアをアクセスする場合は、＜オ

フセット>を0～16383(&H0～&H3FFF)の値で指定します。

- テキストV-RAMのメモリエリアをアクセスする場合、<オフセット>の値を4で割って3余る<オフセット>位置にメモリはありません。
- 読み出されるデータは8ビットのデータとなり、0～255(&H0～&HFF)の値になります。

例

A=peek(&HFF00)

…… メモリの&HFF00番地のデータを変数Aに代入します。

参考

poke …… メモリへのデータの書き込み。(485ページ)

サンプル

```
 10 rem --- PEEK ---
 20 while 1
 30   input "X座標――",X
 40   if X<0 or X>=80 then 20
 50   input "Y座標――",Y
 60   if Y<0 or Y>=25 then 40
 70   OFS=Y*80*3+X
 80   T1=peek@(160,OFS)
 90   T2=peek@(160,OFS+1)
100   AT=peek@(160,OFS+2)
110   print "V－RAMの内容は";T1;T2;AT;"です"
120  wend
```

・文字画面が80桁モードのとき、X座標、Y座標を入力し、その座標のテキストV-RAMの内容を表示します。

POKE （ポーク）

メモリにデータを書き込みます。

書　式

poke　〈アドレス〉,〈データ〉[,〈データ〉]…　(1)

poke@　〈ブロック番号〉,〈オフセット〉,〈データ〉[,〈データ〉]…　(2)

省略形

po.

説　明

- メモリに1バイト単位のデータを書き込みます。
- 書式(1)は機械語エリア使用時のメモリマップによる＜アドレス＞を指定しての書き込みです。
  書き込み時のセグメントは、**def seg**文で指定します。
- 書式(2)は＜ブロック番号＞に**64**キロバイトのメモリブロック番号と、ブロック内のアドレスに相当する＜オフセット＞を指定しての書き込みです。
  ＜ブロック番号＞とシステムメモリの関係は次のようになります。

システムメモリ	＜ブロック番号＞
メインメモリ	**0〜11**
グラフィックV-RAM	**12・13**　B(青)の画面(16色モード)
	14・15　R(赤)の画面(16色モード)
	16・17　G(緑)の画面(16色モード)
	18・19　I(輝度)の画面(16色モード)
PCG	**52**
テキストV-RAM	**181**
増設RAM	**198〜389**

- ＜アドレス＞は**-32767〜65535**(**&H0〜&HFFFF**)、＜ブロック番号＞は**0〜389**(**&H0〜&H185**)、＜オフセット＞は**-32768〜65535**(**&H0〜&HFFFF**)の値で指定します。
- ＜アドレス＞は、整数型データを指定しますので、10進数では**-32768〜32767**の値になりますが、この命令では**32768〜65535**の値で指定することもできます。**32768〜65535**は、整数型データの**-32768〜-1**(**&H8000〜&HFFFF**)に対応します。
- PCGのメモリエリアをアクセスする場合は、＜オフセット＞に偶数値を指定します。＜オフセット＞に奇数値を指定すると1を減じた偶数値が設定されます。なお、＜オフセット＞は**0〜16382**(**&H0〜&H3FFE**)の値で指定します。
- テキストV-RAMのメモリエリアをアクセスする場合は、＜オフセット＞を

0～16383(&H0～&H3FFF)の値で指定します。なお、＜オフセット＞の値を4で割って3余る＜オフセット＞位置にメモリはありません。

- ＜データ＞は8ビットの範囲内の0～255(&H0～&HFF)までの値で指定します。また、＜データ＞を並べて書くと順次＜アドレス＞または＜オフセット＞を増やしながら＜データ＞を書き込みます。ただし、範囲をこえる＜アドレス＞または＜オフセット＞になるときは、エラーとなります。
- この命令は内部のメモリを直接書き換えますのでclear文で指定された機械語エリア、またはV-RAMなどのメモリブロックに対して行ってください。不用意に実行すると、システムを破壊することがあります。

例

```
poke@ 181,&H1000,&HFF
```

…… メモリブロック181(テキストV-RAM)のオフセット&H1000のメモリに&HFFのデータを書き込みます。

参考

peek …… メモリの読み出し。(483ページ)

clear …… 機械語エリアの設定。(169ページ)

DEF SEG(デファインセグメント・define segment)

セグメントベースアドレスを宣言します。

書　式　　**def seg=〈セグメントアドレス〉**

説　明

・64キロバイトのメモリ空間(セグメント)の開始アドレス(ベースアドレス)を宣言します。

peek関数、**poke**文などによってメモリをアクセスできる範囲は1セグメント内となりますので、システムメモリすべてをアクセスするために、この命令によってアクセスするセグメントのベースアドレスを指定し、**peek**関数、**poke**文などではベースアドレスからの相対アドレスを指定するようになっています。この命令で指定したセグメントのベースアドレスは以下の命令に対して有効です。

peek、poke、call、def usr、bload、bsave

・＜セグメントアドレス＞は16バイト単位で指定します。

・プログラム実行時の各種領域のセグメントベースアドレスは**segptr**関数で求めることができます。

・変数や配列のデータのセグメントのベースアドレスは**varptr**関数で求めることができます。

参　考☞　　**segptr**　…各種メモリ領域のセグメントベースアドレス読み出し。(488ページ)

varptr　…変数や配列データのセグメントベースアドレス読み出し。(490ページ)

サンプル

```
10 rem --- DEF SEG ---
20 clear &H3FF
30 def seg=segptr(2)
40 bload "TEST"
```

• 機械語プログラム用のメモリエリアを16キロバイト確保し、その先頭アドレスから"TEST"というファイルを読み込みます。

SEGPTR(セグメントポインタ・segment pointer)

各種メモリ領域のベースアドレスを求めます。

書　式　segptr（〈機能〉）

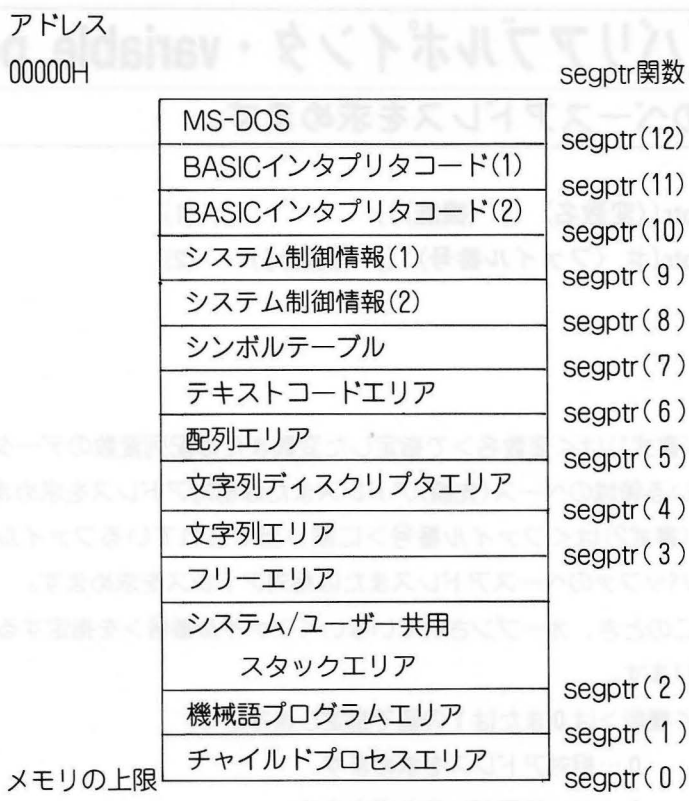
省略形　seg.

説　明

- BASICが管理する各種メモリ領域(エリア)のアドレスを16バイト単位で読み出します。
- 〈機能〉はどのメモリ領域のベースアドレスを求めるかを指定するもので、**0**～**7**の値で指定します。

  ＜機能＞　　求められるベースアドレス

  0　…メモリの上限(メインメモリの最大アドレス＋1)
  1　…チャイルドプロセスエリア
  2　…機械語プログラムエリア
  3　…フリーエリア
  4　…文字列エリア
  5　…文字列管理情報(ディスクリプタ)エリア
  6　…配列エリア
  7　…テキストコードエリア
  8　…シンボルテーブル
  9　…システム制御情報(2)
  10　…システム制御情報(1)
  11　…BASICインタプリタコード(2)
  12　…BASICインタプリタコード(1)
- BASICのメモリ構成と、この命令で求められるベースアドレスの位置は次のようになります。



システム制御情報／シンボルテーブル領域とテキストコード領域は合計最大64キロバイト、文字列ディスクリプター文字列領域も最大64キロバイトとなります。

サンプル

```
 10 rem ---SEGPTR ---
 20 dim AR(7)
 30 for I=1 to 7
 40   AR(I)=(segptr(I-1)-segptr(I))*16
 50 next
 60 print "フリーエリアは";AR(3)-2048;"バイトです"
 70 print "テキストコードエリアは";AR(7);"バイトです"
 80 print "文字列エリアは";AR(4);"バイトです"
 90 print "配列エリアは";AR(6);"バイトです"
100 print "機械語プログラムエリアは";AR(2);"バイトです"
110 print "チャイルドプロセスエリアは";AR(1);"バイトです"
```

VARPTR(バリアブルポインタ・variable pointer)

変数・配列のベースアドレスを求めます。

書　式

varptr(〈変数名〉〔,〈機能〉〕)　(1)
varptr(＃〈ファイル番号〉〔,〈機能〉〕)　(2)

省略形

var.

説　明

- (書式1)は＜変数名＞で指定した変数または配列変数のデータが格納されている領域のベース(先頭)アドレスまたは相対アドレスを求めます。
- (書式2)は＜ファイル番号＞に割り当てられているファイルオープン用のバッファのベースアドレスまたは相対アドレスを求めます。
  このとき、オープンされていない＜ファイル番号＞を指定するとエラーになります。
- ＜機能＞は0または1の値で指定します。
  0…相対アドレスを求めます。
  1…ベースアドレスを求めます。
  ＜機能＞を省略すると、0が設定されます。

例

```
print hex$(varptr(A,1))
```

…変数Aのセグメントベースアドレスを16進で表示します。

DEF USR （ディファイン ユーザ・define user）

機械語プログラムの開始アドレスを定義します。

書　式　def usr〔〈番号〉〕＝〈オフセット〉

説　明

- 機械語プログラムを関数として呼び出すusr0～usr9の関数に機械語開始アドレスを定義します。
  セグメントはdef seg文で指定します。機械語開始アドレスは次のようになります。
  機械語開始アドレス＝セグメント＊&H10＋＜オフセット＞
- ＜番号＞は0～9までの整数で指定し、usr0～usr9の10種の定義ができます。＜番号＞を省略すると、usr0になります。
- ＜オフセット＞はusr関数を使用したときに実行される機械語プログラムのセグメント内の実行開始オフセットを**－32768～65535(&H0～&HFFFF)**の値で指定します。
- ＜オフセット＞は、整数型データで指定しますので、10進数では**－32768～～32767**の値になりますが、この命令では**32768～65535**の値で指定することもできます。**32768～65535**は、整数型データの**－32768～－1 (&H8000～&HFFFF)**に対応します。
- 定義された関数は、その関数名で使用すると定義された＜開始アドレス＞より機械語プログラムをサブルーチンとして実行します。機械語プログラムの終了は、機械語プログラム内の**FAR RET**(リターン)命令になります。
- 機械語プログラムはclear文で指定した機械語エリア内で作成してください。

参　考☞　peek（483ページ）　poke（485ページ）……　メモリの読み出し、書き込み。
call　……　機械語プログラムの実行。（493ページ）
clear　……　機械語プログラムの設定。（169ページ）
bload（301ページ）　bsave（303ページ）……　機械語プログラムのファイル操作。
def seg　……　セグメントの宣言。（487ページ）

サンプル

```
10 rem --- DEF USR ---
20 def seg = &H8000
30 def usr = &HF000
```

……　usr 0 の開始番地を**&H8F000(&H8000＊&H10＋&HF000)**に指定します。

USR （ユーザ・user）

機械語プログラムを関数として呼び出します。

書　式　usr〔〈番号〉〕(〈引数〉)

説　明

- def usr文で定義された機械語プログラムに＜引数＞を与えて実行し、機械語プログラムより渡されたデータがこの関数の値になります。
- ＜番号＞は0～9の値で指定し、usr0～usr9になり、＜番号＞を省略するとusr0になります。
- ＜引数＞は機械語プログラムへ渡すデータになり、データの型がCPUレジスタCHレジスタへ、データの記憶されているアドレスがCPUレジスタのDS：BXレジスタにセットされます。
- この関数の結果は、機械語プログラムでCPUレジスタのCHレジスタとBXレジスタに渡された情報によるデータになります。機械語プログラムで、これらのCPUレジスタを変更していない場合は＜引数＞の値がそのまま結果になります。
- 機械語プログラムに渡されるデータは次のようになります。

データ型	CHレジスタ	データの並び（DS：BX←先頭アドレス）
整　数	2	下位 | 上位
単精度	5	指数 | 仮　数　部
倍精度	8	指数 | 仮　数　部
文字列	3	長さ | L | H

（文字列型のL, Hは、文字列が格納されているアドレスを示します。）

- 機械語プログラムでエラーにするときは、CPUのキャリーフラグを1にし、ALにエラーコードを設定してリターン(FAR RET)してください。
  BASIC内のエラーコードとして扱われます。

例

```
A$=usr("ABC")
```

…… "ABC"を引数としてUSR0で定義されている機械語サブルーチンを実行し、結果を変数A$に代入します。

参　考☞　def usr …… 機械語関数の定義。(491ページ)

CALL （コール）

機械語サブルーチンを呼び出します。

書　式　call 〈オフセット〉[(〈変数〉)]

省略形　ca.

説　明

- 指定された<オフセット>から機械語サブルーチンとして実行します。このときに機械語サブルーチンへデータを渡すことができます。実行される機械語サブルーチンのセグメントは、直前の**def seg**文で指定されたセグメントです。

  コールされるアドレス＝セグメント＊**&H10**＋<オフセット>

- <オフセット>は**-32768～65535**(**&H0～&HFFFF**)の値を変数または定数で指定します。
- <変数>は機械語プログラムに渡すデータで、定数を使用することはできません。
- 機械語プログラムへは**CPU**の**AL**レジスタへデータの有無(**1**…あり,**0**…なし)**DS:BX**レジスタにデータが記憶されている先頭アドレス、**CH**レジスタにデータの型の情報が入り、指定されたアドレスからの実行になります。

  実行時のレジスタ

  DS, ES＝インタプリタ実行時の**DS, ES**
  SS:SP＝インタプリタのスタック
  AL＝データの有無(1：あり,0：なし)
  CH＝データの型
  DS:BX＝データの先頭アドレス
  その他…不定

  機械語プログラムからBASICに戻るには、**RET FAR**を行ってください。
- 機械語プログラムより、データを返す場合は機械語プログラム内で、BASICより渡されたデータを書き換えることにより行われ、キャリーフラグを**0**にしてBASICに戻ったとき<変数>に値が返ります。キャリーフラグを**1**にしてBASICに戻ると、その時の**AL**の値をエラー番号としてエラー処理を行います。
- 機械語プログラムは**clear**文で機械語エリアを設定してそのエリア内に作成してください。
- **field**文でフィールド変数として指定されている文字型変数をこの命令で使用するとフィールド変数の機能が解除されますので注意してください。

例

```
call &HF000
```

…… **def seg**文で指定されたセグメントの**&HF000**番地より機械語サブルーチンを実行します。

参　考

peek（483ページ）、poke（485ページ） …… メモリの読み出し、書き込み。

def usr（491ページ）、usr（492ページ）… 機械語プログラムを関数としての定義と実行。

clear …… 機械語プログラムエリアの設定。（169ページ）

bload（301ページ）、bsave（303ページ） …… 機械語プログラムのファイル操作。

def seg …… セグメントの宣言。（487ページ）

OUT （アウト）

I/Oポートにデータを出力します。

書　式　　out 〈ポートアドレス〉,〈数式〉

説　明

- CPUのI/Oアドレスを＜ポートアドレス＞として＜数式＞の値を出力します。
- ＜ポートアドレス＞は16ビットで、−32768〜32767の整数型データで指定します。また、32768〜65535(&H8000〜&HFFFFに対応)で指定することもできます。
- ＜数式＞は8ビットで、0〜255(&H0〜&HFF)の値で指定します。
- この命令はコンピュータ内部のI/Oポートを直接操作しますので、コンピュータ内部で使用されている場所に不用意に出力するとコンピュータが正常に動作しなくなるので注意が必要です。

例

```
out &H20,asc("A")
```

…… ポートアドレス&H20に"A"の文字コード(65)を出力します。

参　考　　inp …… I/Oポートよりデータを入力する関数。(496ページ)

INP （インプット・input）

I/Oポートよりデータを入力します。

書　式　inp(〈ポートアドレス〉)

説　明

- CPUのI/Oアドレスを＜ポートアドレス＞として入力した値を返す関数です。
- ＜ポートアドレス＞は16ビットで、−32768〜32767の整数型データで指定します。また、32768〜65535（&H8000〜&HFFFFに対応）で指定することもできます。
- 結果は8ビットで0〜255（&H0〜&HFF）の値が返ります。
- この命令はコンピュータ内部のI/Oポートより直接入力する命令です。

例

```
A=inp(&H10)
```

…… ポートアドレス&H10により入力したデータを変数Aに代入します。

参　考　out …… I/Oポートへのデータの出力。（495ページ）

WAIT （ウェイト）

I/Oポートからの入力が指定状態になるまで実行を停止します。

書　式　wait 〈ポートアドレス〉,〈式1〉[,〈式2〉]

省略形　wa.

説　明

- ＜ポートアドレス＞で指定したI/Oポートからの入力データと＜式2＞の論理演算xorを行い，その結果と＜式1＞の論理演算andを行った結果が、0の場合は、そのまま入力データの監視を行い、0以外のときは、次の処理に進みます。
- 実行の停止状態のときは、キーボードからの先行入力や、各種の割り込み処理は行われません。したがって、シフト＋停止キーによる実行の中断もできなくなります。
- ＜ポートアドレス＞、＜式1＞、＜式2＞は0～255（&H0～&HFF）の値で指定します。

例　wait &HFE,1,1

…… ポートアドレス&HFEからの入力データのビット0（D0）が0になるまで実行を停止します。

サンプル

```
 10 rem --- WAIT ---
 20 P=inp(&HE0)
 30 for I=&H0 to &HFF
 40   for L=&H0 to &H20
 50     X=(P xor L) and I
 60     if X<>0 then gosub *WAIT else gosub *NOT_WAIT
 70   next
 80 next
 90 color 7:end
100 *WAIT
110 color 7:print "wait &HE0,";I;",";L;"を実行します。"
120 wait &HE0,I,L
130 return
140 *NOT_WAIT
150 color 2:print "wait &HE0,";I;",";L;"を実行するとＩ／Ｏ入力待ちになります。"
160 return
```

- ポートアドレス&HE0からの入力データに対し＜式１＞（変数I）、＜式２＞（変数L）の値をいろいろと変えて、どのような値をとるときにI/O入力待ちになるかを表示します。

FRE （フリー・free）

メモリの未使用量を求めます。

書　式　　fre(〈機能〉)

説　明

• プログラム、変数、配列、プリンタスプーラやメモリディスクなどで使用されていないメモリ量をバイト数で求める関数で ＜機能＞ により次の値が返ります。

0　…システム全体の空きメモリの容量を求めます。
次のそれぞれのメモリ容量の合計となります。
• メインメモリのフリーエリア
• 増設RAMの空き容量
• グラフィック**V-RAM**の空き容量

1　…テキストコード(BASICプログラム)エリアとして使用できる空きメモリの容量を求めます。
次の２つのメモリ容量の内小さい方を求めます。
・64キロバイトからシステム制御情報／シンボルテーブルの容量とテキストコードエリアの容量を差し引いたもの
・メインメモリのフリーエリア

2　…配列エリアとして使用できる空きメモリの容量を求めます。
メインメモリのフリーエリアの容量と一致します。

3　…文字列エリアとして使用できる空きメモリの容量を求めます。
次の２つのメモリ容量の内小さい方を求めます。
・64キロバイトから文字列エリアの容量を差し引いたもの
・メインメモリのフリーエリア

4　…メモリディスク用のメモリとして使用できる空きメモリの容量を求めます。
次の２つのメモリ容量の内大きい方を求めます。
・増設RAMの空きメモリの容量
・グラフィック**V-RAM**の空きメモリの容量

5 …グラフィック表示やプリンタスプーラ用に使用できる空きメモリの容量を求めます。
グラフィック**V-RAM**の空きメモリの容量と一致します。

6 …文字列ディスクリプタエリアとして使用できる空きメモリの容量を求めます。
次の2つのメモリ容量の内小さい方を求めます。
・64キロバイトから文字列ディスクリプタエリアの容量を差し引いたもの
・メインメモリのフリーエリア

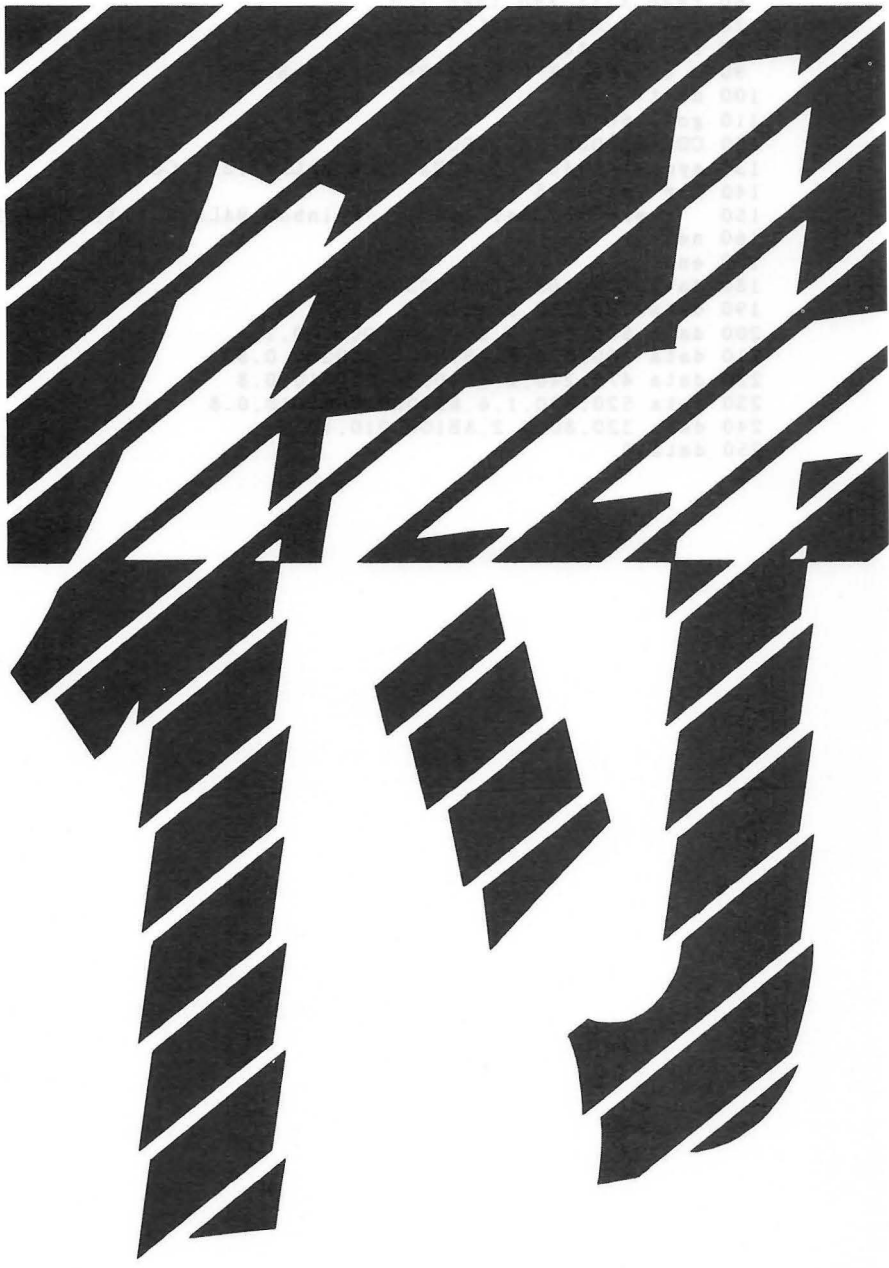
例

```
print fre(2)
```

…… 配列に使用できるサイズを表示します。

参 考☞ BASICのフリーエリア(付録)

付 録



サンプルプログラム（65536色モード）

```
 10 rem --- ６５５３６色モードサンプル ---
 20 init "mem:0,0":option screen 1:init "crt2:640,400,65536"
 30 cls 3:klist 0
 40 pen 5
 50 option angle degrees
 60 read X:if X=0 then 120
 70 read Y,MUL,COL,L
 80 for R=1 to 29
 90   circle (X-L*R,Y),R*MUL,(31-R)*COL
100 next
110 goto 60
120 COL=&B100010000000100
130 symbol (140,180),"Rainbow BALLs",3,3,COL,,,,1
140 for I=1 to 5
150   symbol (140+I,180-I),"Rainbow BALLs",3,3,COL+I*3*&B100001000010,,,,1
160 next
170 end
180 data 130,100,3,&B1000000,1
190 data 300,300,3,&B10,1
200 data 500,120,2,&B100000000000,1
210 data 100,320,2,&B100001000000,0.8
220 data 470,240,2,&B100001000010,0.8
230 data 520,330,1.6,&B100000000010,0.8
240 data 320,80,2.2,&B1000010,1
250 data 0
```

文字コード表 （半角文字）

- 文字コードが**&H81～&H9F, &HE0～&HFC**はシフトJISコードの1バイト目になります。半角文字で使用する場合は**kmode 0**として漢字モードを解除してください。
- 文字コードが**&H1F**以下の文字はコントロールコードになりますので、通常の**print**文ではコントロールが実行されます。コントロールコードを画面に表示する場合は、**"CRT1:CG"**のファイル名でシーケンシャルアクセスのファイルオープンを行い、**print#**文で書き込みを行ってください。

エラーメッセージ

BASICの動作中にエラーが発生した場合、次の形式でコンピュータよりディスプレイ上にエラーメッセージが表示されます。

エラーの表示

××××　error　　…直接実行モード(ダイレクトモード)

××××　error　in　nn　　…プログラム実行中

××××はエラーの内容、nnはエラーの発生行になります。

ファイル処理や周辺装置を使用している場合は、その装置名やファイル指定を左側に表示する場合があります。

エラー番号	表示メッセージ	エラーの内容
1	syntax error	文法上の誤りがある
2	over flow error	数値が扱える範囲をこえた(0での除算など)
3	data error	指定データが規定外である
4	type mismatch error	指定のデータ型が異なる
5	string length error	文字データの長さが255をこえた 指定の文字データが規定長になっていない
6	no memory error	メモリ容量不足となった 許されている容量をこえて領域を確保しようとした
7	w-def. FN error	定義関数を2重に定義しようとした
8	line length error	プログラムの1行の長さが制限をこえた
9	unprintable error	未定義
10	IF-ENDIF mismatch error	ifとend ifが対応していない
11	FOR-NEXT mismatch error	forとnextが対応していない
12	WHILE-WEND mismatch error	whileとwendが対応していない
13	REPEAT-UNTIL mismatch error	repeatとuntilが対応していない
14	GOSUB-RETURN mismatch error	gosub文で呼ばれていないのにreturnしようとした
15	undef. array/FN error	定義されていない関数や配列を指定した
16	undef. line error	存在しない行番号またはラベルを指定した
17	can't CONT error	プログラムの実行再開が不可能なとき再開しようとした
18	memory protection error	許されている領域外に処理を行おうとした
19	instruction error	ダイレクトコマンドとステーメントを混同した 現在の状態では使用できない機能を指定した
20	sequence error	読み込もうとしたプログラムファイルの書式がおかしい

エラー番号	表示メッセージ	エラーの内容
21	RESUME error	エラー処理中でないのにresumeしようとした
22	count error	関数の引数や入力変数の個数が合わない
23	subscript error	配列の指定が不適当 配列の大きさが不足する
24	READ-DATA mismatch error	対応するdata文がないのにreadしようとした
25	SWAP nesting error	※
26	too complex error	式が複雑で一度に処理できない
27	too many nesting error	ループまたはif文、サブルーチンなどのネスティングが多い
28	system id error	BASICで扱えないファイルである
29	framing error	RS-232C フレーミングエラー
30	over run error	RS-232C オーバーランエラー
31	parity error	RS-232C パリティエラー
32	unprintable error	未定義
33	unprintable error	
34	unprintable error	
35	unprintable error	
36	unprintable error	
37	unprintable error	
38	unprintable error	
39	unprintable error	
40	not found error	指定ファイルがみつからない
41	hardware error	ディスクのハードウェアトラブルが起こった
42	already exist error	※
43	already open error	すでにオープンされているファイルを指定した
44	not open error	オープンされていないファイルを使用した
45	not empty error	※
46	write protect error	プロテクトされたファイルまたはディスクを書き換えようとした
47	field error	フィールド定義のできないファイルを指定した フィールドの合計がレコード長をこえた
48	open mode error	オープンされたファイルに対するアクセス型式が合わない
49	unprintable error	未定義
50	not ready error	指定装置が可動状態になっていない
51	too many files error	ファイルの登録数が制限をこえた

エラー番号	表示メッセージ	エラーの内容
52	disk mismatch error	※
53	no file space error	ファイルを作成するスペースがなくなった
54	unformat error	フォーマットされていないディスクを使用した
55	file length error	※
56	disk change error	ファイルオープン中にディスクを交換した
57	unprintable error	未定義
58	device name error	装置名の指定が不適当
59	can't execute error	指定装置に使用できないファイル型式を指定した 実行できない命令の使用または実行できないモードを指定した
60	file name error	ファイル名の指定が不適当
61	file mode error	※
62	unprintable error	未定義
63	file end error	ファイルが終了しているのに読み込もうとした
64	logical number error	ファイル番号の指定がおかしい
65	LPT：not ready error	プリンタが可動状態でない(PW-off,off line)
66	unprintable error	未定義
67	unprintable error	
68	device mode error	※
69	unprintable error	未定義
70	check sum error	※
71	グラフィック機能は利用できません	
72	日本語入力機能は利用できません	
73	プリンタ機能は利用できません	
74	RAMディスク機能は利用できません	
75	マウス機能は利用できません	
76		
|	unprintable error	未定義
255		

※ BASIC-M25との互換性のため存在するエラーで、BASIC-M28では通常発生しないものです。
(**error**文でエラー発生を行うことはできます。)

BASICのフリーエリア

BASICは起動時にメインメモリに読み込まれ、残りの領域にプログラムや変数、配列を記憶します。この部分をフリーエリアといいます。フリーエリアは、プログラムの実行内容などにより異なります。

エリア(領域)は、記憶する内容により、次のように分かれています。

- テキストコードエリア

  BASICのプログラムを記憶したりダイレクトモードで一時的な命令の記憶に使用され、システム制御情報やシンボルテーブルと合わせて最大64キロバイトのエリアを使用します。

- 文字列管理情報(ディスクリプタ)・文字列データエリア

  文字列型の変数や配列の記憶や文字列式の展開に使用され、それぞれ最大**64**キロバイトのエリアを使用します。

- 配列エリア

  配列の記憶に使用します。使用量の制限はありません。

- 機械語プログラムエリア

  BASICが使用しないメモリエリアで、ユーザーが自由に読み書きすることができます。このエリアの設定は、**clear**文で行います。

- チャイルドプロセスエリア

  BASICが使用しないメモリエリアで、MS-DOSのコマンドを実行するために使用されます。このエリアの設定は**clear**文で行います。

- メモリディスク、プリンタスプーラ用エリア

  メモリディスクやスプーラ用メモリとして**init "MEM:"**文で設定します。このエリアは、グラフィック**V-RAM**および増設RAMを使用することができます。

  グラフィック**V-RAM**の設定は**option screen**文で設定します。

メインメモリ(768キロバイト)

00000H

MS-DOS
BASICインタプリタコード(1)
BASICインタプリタコード(2)
システム制御情報(1)
システム制御情報(2)
シンボルテーブル
テキストコードエリア
配列エリア
文字列ディスクリプタエリア
文字列エリア
フリーエリア
システム/ユーザー共用スタックエリア
機械語プログラムエリア
チャイルドプロセスエリア

メモリの上限

グラフィック**V-RAM**(512キロバイト)

メモリディスクまたはプリンタスプーラ

増設RAM

メモリディスク

コントロールコード表

文字コードの31(&H1F)以下は、制御機能をもったコントロールコードになります。

コード		機能	キー操作	
10進	16進		コントロール＋	その他
0	00	なし(Null)	@	
1	01	インサートモード	A	コントロール＋シフト＋挿入/抹消
2	02	左側の単語の先頭へカーソルを移動	B	シフト＋←
3	03	なし	C	
4	04	画面の初期化や音楽演奏の中止	D	
5	05	カーソル位置より行末まで消去	E	コントロール＋挿入/抹消
6	06	次の単語の先頭へカーソルを移動	F	シフト＋→
7	07	ベル音(beep)	G	
8	08	カーソルの左側の1文字を削除	H	後退
9	09	水平タブ	I	タブ
10	0A	なし	J	
11	0B	カーソルをホームポジションへ移動	K	クリア/ホーム
12	0C	画面の消去	L	シフト＋クリア/ホーム
13	0D	キャリッジリターン	M	↵
14	0E	カーソル行より上をスクロールアップ	N	
15	0F	カーソル行より下をスクロールダウン	O	
16	10	なし	P	
17	11	なし	Q	
18	12	カーソル位置に空白を挿入	R	シフト＋挿入/抹消
19	13	なし	S	
20	14	カーソル位置にタブをセット	T	
21	15	なし	U	
22	16	カーソル以降の文字を次行に送る	V	
23	17	カーソル行と次の行を結合	W	
24	18	カーソル行を消去	X	
25	19	カーソル位置のタブを解除	Y	
26	1A	カーソル位置以降のすべての文字を消去	Z	
27	1B	なし(ESC)	[	ESC
28	1C	カーソルを右へ移動	¥	→
29	1D	カーソルを左へ移動	]	←
30	1E	カーソルを上へ移動	^	↑
31	1F	カーソルを下へ移動	_	↓
127	7F	カーソル位置の１文字を消去	なし	挿入/抹消

たとえば、print chr$(12)とすると画面を消去し、print chr$(7)とすると、「ピッ」と音を鳴らすことができます。

RS-232Cインターフェイス

本機にはRS-232Cのシリアルインターフェイスを内蔵しており、BASIC上では、**"COM 1 :"**の装置名でデータの転送を行うことができます。

init "COMn:"文では、ボーレートなどを次のように設定します。(〔 〕は省略値)

書式　**"COMn:<ボーレイト>,<パリティ><データビット長>**
<ストップビット長><通信制御><カナ><DEL受信>
<CR送信><CR受信><全角文字><エンドコード>"

<ボーレイト>　ボーレイトとして次の値が指定できます。

75, 150, 300, 600, 1200, 2400, 4800, 9600, 19200

<パリティ>

E　…偶数パリティビットを付加します。

O　…奇数パリティビットを付加します。

〔N〕　…パリティビットは付加しません。

<データビット長>

7　…データビット長を7ビットにします。

〔8〕　…データビット長を8ビットにします。

<ストップビット長>

1　…ストップビット長を1ビットにします。

2　…ストップビット長を1½ビットにします。

〔3〕　…ストップビット長を2ビットにします。

<通信制御>

X　…XON(&H11)・XOFF(&H13)コードによる制御を行います。

R　…RS信号による制御を行います。

〔N〕　…上記のどちらの制御も行いません。

通信制御とは、データの受信時に受信側のデータ受信処理が間に合わないときに、送信側に送信の一時停止を要求し、受信できるようになると送信再開を要求することです。Xを指定すると、XOFFコード(&H13)を送信側へ送ることにより送信の一時停止を要求し、XONコード(&H11)により送信再開を要求します。Rを指定すると、RS信号をOFFにすることにより送信の一時停止を要求し、ONにすることにより送信再開を要求します。

＜カナ＞

S　…データ長7ビットでSO(&H0E)・SI(&H0F)コードによるカナの送受信を行います。(データ長８ビット時は利用できません。)

〔N〕　…データ長7ビットでカナの送受信を行いません。

データ長７ビット時のカナの送受信は、SOコード(&H0E)があるとそれ以後のデータはカナとみなし、SIコード(&H0F)があるとそれ以後のデータを英数字とみなします。

＜DEL受信＞　DEL(&H7F)コード受信時の処理を指定します。

N　…DEL(&H7F)コードをNull(&H00)コードに変換します。

B　…DELコードをBS(&H08)に変換します。

〔D〕　…DELコードは上記のいずれの変換も行わず、そのまま受信します。

＜CR送信＞

C　…CR(&H0D)コードを復帰＋改行コードとして送信します。

〔L〕　…CR＋LF(&H0A)コードを復帰＋改行コードとして送信します。

＜CR受信＞

C　…CR(&H0D)コードの受信のみで復帰改行動作をします。

〔L〕　…CR＋LF(&H0A)コードの受信で復帰改行動作をします。

＜全角文字＞

J　…KI(漢字イン&H1B・4B)・KO(漢字アウト&H1B・48)により全角文字の送受信を行います。

M　…KIコードを&H1B・09、KOコードを&H1B・01として全角文字の送受信を行います。

P　…KIコードを&H1B・24・40、KOコードを&H1B・28・48として全角文字の送受信を行います。

〔N〕　…全角文字はシフトJISコードで送受信を行います。
(データ長が７ビットのときにNを指定すると正しく全角文字が送受信できないことがあります。)

＜エンドコード＞

エンドコードはデータ転送の終了を表すコードで&H00～&H１Fまでのコードを指定します。指定方法はキー操作で[コントロール]キーと合わせて入力される文字を指定します。指定文字はコードで&H40～５Fか&H60～７Fになります。たとえば、エンドコードが&H04ならばキー操作上では[コントロール]＋[D]となりますので指定はD(&H44)またはd(&H64)になります。

この指定により、ファイルの入力オープン時にエンドコードを入力する

とeof関数が真になります。スペースを指定、または指定を省略した場合はこの機能を持ちません。

＜ボーレイト＞の後はカンマ(,)で句切り、ほかの指定は1文字ずつを順に並べます。また、それぞれの指定の省略時はそれぞれ省略値が設定されますが、途中の指定を省略するときは、空白とします。ボーレートおよびカンマ(,)を省略することはできません。

例） **"COM:9600, N83XNDLLND"**

…"COM1:"を9600ボー、パリティ無し、8データビット、2ストップビット、Xon-Xoffの通信制御を行い、カナは8ビット、DELコードはそのまま、CR送受信は&H0D＋&H0A、全角文字はシフトJISコード、エンドコードは&H04に設定します。

三角関数と双曲線関数

BASICであらかじめ用意されている関数(組み込み関数)以外の三角関数や双曲線関数とそれらの逆関数を求める例を示します。

secant	A=1/cos(X)
cosecant	B=1/sin(X)
cotangent	C=1/tan(X)
arcsine	D=atn(X/sqr(1-X*X))
arccosine	E=-atn(X/sqr(1-X*X))+pai/2
arcsecant	F=atn(sqr(X*X-1))+(sgn(X)-1)*pai/2
arccosecant	G=atn(1/sqr(X*X-1))+(sgn(X)-1)*pai/2
arccotangent	H=-atn(X)+pai/2
hyperbolic sine	I=(exp(X)-exp(-X))/2
hyperbolic cosine	J=(exp(X)+exp(-X))/2
hyperbolic tangent	K=-exp(-X)/(exp(X)+exp(-X))*2+1
hyperbolic secant	L=2/(exp(X)+exp(-X))
hyperbolic cosecant	M=2/(exp(X)-exp(-X))
hyperbolic cotangent	N=exp(-X)/(exp(X)-exp(-X))*2+1
arc-hyperbolic sine	O=log(X+sqr(X*X+1))
arc-hyperbolic cosine	P=log(X+sqr(X*X-1))
arc-hyperbolic tangent	Q=log((1+X)/(1-X))/2
arc-hyperbolic secant	R=log((sqr(1-X*X)+1)/X)
arc-hyperbolic cosecant	S=log((sgn(X)*sqr(X*X+1)+1)/X)
arc-hyperbolic cotangent	T=log((X+1)/(X-1))/2

変数Xに値を代入しておき、式を実行すると左辺の変数に結果が代入されます。

def FN文を使用して関数として定義して使うこともできます。

例)

```
def FNSEC(X)=1/cos(X)      … 定義
print FNSEC(pai/3)         … 使用例
```

この例の三角関数の角度単位はラジアンになりますので度単位になっている場合はoption angle radiansとしてラジアン単位に戻してください。

この例は組み込み関数以外の関数を求めるための参考資料です。Xの値によっては精度が低くなる場合がありますので注意してください。(特に減算による桁落ちに注意してください。)

機械語サブルーチン

本BASICでは次の命令で機械語サブルーチン・プログラムを呼び出すことができます。

usr

call

bload（**R**指定があるとき）

1．呼び出されるアドレス

それぞれの命令で呼び出されるアドレスは次のようになります。

- セグメント

  usr　：**def usr**文が実行されたときの**def seg**文の設定値（BASIC起動時0）

  call　：この命令が実行されたときの**def seg**文の設定値（BASIC起動時0）

  bload：この命令で読み出されたプログラムの先頭のセグメント

- オフセット

  usr　：**def usr**文で指定されたオフセット

  call　：この命令で指定されたオフセット

  bload：この命令で読み出されたプログラムの先頭のオフセット

2．呼び出されるときのパラメータ

それぞれの命令で呼び出されるときのパラメータの値は次のような意味を持ちます。

・**AL**　…　**パラメータがあるかどうか**

0　：　パラメータなし

1　：　パラメータあり

（**usr**では必ず1, **bload**では必ず0になります。）

・**CH**　…　**パラメータの型**（**AL**<>0のときのみ）

2　：　整数型

5　：　単精度実数型

8　：　倍精度実数型

3　：　文字型

・**DS:BX**　…　**パラメータのアドレス**（**AL**<>0のときのみ）

数値型なら**DS:BX**にデータが置かれます。

文字型なら**DS:BX**の構成は次のようになります。

DS:BX　：　文字列のバイト数

DS:BX＋1　：　文字列のオフセット(LOW)

DS:BX＋2　：　文字列のオフセット(HIGH)

実際の文字列は**DS:**[HIGH＊100H＋LOW]に置かれます。

・**SS:SP**　…　**BASICのスタック**

ユーザーが使用することもできますが(容量2KB)、システム割り込みに100バイト程度使用するので、使用する場合は常に100バイト以上の余裕を持たせてください。また、BASICに戻る場合はもとの**SS:SP**の値に戻してください。

・**ES**　…　**DSと同じ**

3．リターンするときのパラメータ

リターンするときのパラメータの値は次のような意味を持ちます。

・**CF**(キャリーフラグ)　…　**エラーが発生したかどうか**

0　：　エラーは発生しなかった

1　：　エラーが発生したのでエラー処理を行う

・**AL**　…　**エラー番号(0～127)**　(**CF**＝1のときのみ)

・**CH**　…　**変数に代入するデータの型**

(呼び出し時**AL**＝1の場合のみ)

usr 関数の場合は、どの型でもかまいません。**call** 文の場合は、数値型の場合、入力時の型と一致している必要はありません。なお、数値型と文字型の間の型変更はできません。

(**type mismach error** となります。)

・**BX**　…　**変数に代入するデータのオフセット**

(呼び出し時に**AL**＝1の場合のみ)

セグメントは自動的に呼び出し時の**DS**が採用されます。

4．制限条件

次のレジスタは保存しておいてください。

SS:SP

BASICに戻る場合には、リターンファー(**FAR RET**)で戻ってください。

ターミナルプログラム

システムディスクに収められている本ターミナルプログラム(TERM.BAS)は、RS-232Cインターフェイスを使用して、本機をターミナル(端末装置)として使用するためのBASICプログラムです。

ターミナルプログラムの基本的な動作は、キー入力した文字を文字コードとして相手側に送り、相手側からのデータは、文字コードとして画面に表示します。また、本ターミナルプログラムは全二重(キーを押すたびに相手側に文字コードを送り、相手側からデータを受信するごとにデータを1文字ずつ表示)モードにて動作します。

RS-232Cのパラメータセットについて

ボーレート等のパラメータの設定は、キーより [L][I][S][T][ ][*][R][S][D][T][S][E][T][-][*][R][S][D][T][S][E][T][E][↵] と入力しますとパラメータの設定部分のリストが画面に表示されますので、変更の必要なパラメータのみ変更します。

ファンクションキーについて

ターミナルプログラムではファンクションキーに次の機能をもたせています。

[F1] … **コントロールコードの実行切り替え**

このキーを押すごとに&H1F以下のコントロールコードを実行するか、表示するかを切り替えます。表示する場合は、そのコードに応じたフォントが表示されますが、CR(&H0D)は表示した後に実行します。また、X-ONやX-OFF、SIやSO、KIやKOなどの制御コードを使用している場合は、そのコードの表示は行われません。

- LTRLと表示されているときはコントロールコードを実行します。(初期状態)
- N-LTRLと表示されているときはコントロールコードを実行しないで表示します。

[F2] … **エコーバックの有無**

このキーはキー入力された文字を表示するか、しないかの切り替えを行います。

- ECHOonと表示されているときはキー入力した文字を表示しません。(初期状態)

- ECHOoffと表示されているときはキー入力した文字を表示します。

F3 … **プリンタへの印字切り替え**

このキーを押すごとに画面の表示と同時にプリンタへ印字するか、画面のみの表示にするかを切り替えます。

- LPTonと表示されているときは画面表示のみ行います。(初期状態)
- LPToffと表示されているときは画面表示と印字を行います。

F4 … **ファイルのダウンロード**

このキーを押すとそれ以後の受信データをファイルへ記録します。キーを押すとファイル名を聞いてきますのでBASICのファイル指定と同じ方法で指定し入力します。この機能の終了は、再び F4 キーを押すか、エラーの発生により行います。(END—DNと表示されているときは受信中を表します。) この機能で作成されたファイルはシーケンシャルデータ (DAT) 型式ですので、BASICのプログラムにより読み出すことができます。

F5 … **ファイルのアップロード**

このキーを押すとファイルの内容を送り出します。キーを押すとファイル名を聞いてきますのでDAT形式のファイルを指定するとその内容を相手側に送り出します。送り出しの途中で中止する場合は、再び F5 キーを押します。(END—UPと表示されているときは送信中を表します。)

F6 … このキーは、漢字出力モードと英数カナ文字出力モードの切り替えを行います。初期設定は、漢字出力モードです。

F7 … このキーを押すたびに、キーボードよりキー入力を有効にするか、キー入力を無効にするかを切り替えます。(ただし、ファンクションキーは常時キー入力が可能です。) 初期設定はキー入力が有効な状態です。

F8 … このキーは、送信データの入力ラインのマーク (#) を表示するかしないかを切り替えます。初期

設定は表示する状態です。

F9 … このキーを押すことによって、ターミナルプログラムを終了します。

最下行にファンクションキーの定義状態が表示されている場合は、そのファンクションキーを押したときに切り替えられる状態を表示します。

なお、受信エラーが発生した場合は、エラーとなりプログラムの実行を中止します。

エスケープシーケンスについて

ターミナルモードでは、コントロールコードによる制御以外に次のエスケープシーケンスによる制御も行われます。

(ESC)(@) ……画面とプリンタの双方に出力します。
(ESC)(A) ……画面のみ表示とします。
(ESC)(") ……キー入力を有効とします。
(ESC)(#) ……キー入力を無効とします。
(ESC)(*) ……画面を消去し、カーソルを左上に移動します。
(ESC)(Y) ……カーソル位置より下の行を消去します。

(ESC)(=)(*l*)(*c*)

…… カーソルを*l*行*c*桁に移動します。*l*と*c*は画面の左上を(0,0)とした位置による値に&H20を加えた値に対応する文字コードになります。

例) (6,4)に移動する場合
(ESC)(=)(&)($)

(ESC)(T)

…… カーソルのある行を消去してカーソルをその行の左端に移動します。

(ESC)(E)

…… カーソルのある行から下の行をスクロールダウンさせ、カーソルのある行に空白行を挿入します。

(ESC)(R)

…… カーソルのある行の下側をスクロールアップさせ、カーソルのある行を削除します。

BASICーM25との互換性について

BASICーM28とBASICーM25

BASIC-M28は、MZ-2500シリーズMZ-2511/21/31/20に標準装備されている、BASIC-M25と互換性を保つように配慮されています。ただし、BASIC-M28は、MZ-2861の2800モードで動作するため、ハードウェアの違いや、オペレーティングシステムとしてMS-DOSを使用する関係上、一部BASIC-M25と仕様が異なります。ここでは、BASIC-M25で作成されたプログラムをBASIC-M28上で使用する場合に注意する点について述べています。

一般的な注意

BASIC-M28は、MZ-2861の2800モードを活かすように設計されています。したがって、次のような点がBASIC-M25と異なります。

- MZ-2861では、ボイスレコーダ、デジタル方式のモノクロディスプレイおよびボイスボードを使用できませんので、これらの周辺機器や拡張ボードに対する命令は削除されています。
- MZ-2861のRS-232Cインターフェイスは1チャンネル(チャンネルAのみ)です。
- MZ-2861の2800モードでグラフィック画面の320×200ドットモードおよび640×400ドット4色モード、文字画面の64色モードは使用できません。
  また、メモリ構成が異なりますので、メモリアドレスの指定方法などが異なります。
- MS-DOS上で動作するため、階層ディレクトリやディスクのファイル管理方法が異なります。
  したがって、BASIC-M25で作成されたファイルはそのまま(変換なしに)BASIC-M28で使用することはできません。
- ターミナルモードは削除されています。(ターミナルプログラムを利用してください。)
- モニタ機能は削除されています。

BASICーM25から削除された命令

次の命令は、BASIC-M28上では削除されていますので、使用することはできません。

・基本コマンド	**new on, term, mon**
・表示設定命令	**init"CRT3:"**
・グラフィック命令	**cblock**
・ファイル関数	**fpos**
・サウンド命令	**voice, voice@**
・ボイスレコーダ制御命令	**init"CMT:", cmt==, cmt(関数), apss**

BASICーM25とは互換性のない命令

次の命令は、BASIC-M28では命令仕様が変更されていますので、BASIC-M25とは一部の機能に互換性がありません。

・表示設定命令	**option screen, init"CRT2:", screen**
・グラフィック命令	**priority**
・ファイルコマンド	**init"MEM:, files/lfiles, chdir**
	mkdir, rmdir, bload, bsave
・ファイルステートメント	**lof, loc, devf, pwd$**
	run＜ファイル＞
・マウス命令	**mouse(関数)**
・機械語処理	**peek/peek@, poke/poke@, def usr**
	call

追加予約語

次の予約語は、BASIC-M28で追加されたものです。

add, child, environ, environ$, float, kext$, kinput, kinstr, kmid$, knj$, ktype, seg, segptr, sub, system, varptr

ファイルコンバート

BASIC-M25で作成されたファイルをBASIC-M28上で動作させるためには、MS-DOSで管理できるファイルに変換する必要があります。

ファイル変換は、システムディスク内に納められている「ファイルコンバートユーティリティ」を使用してください。

ファイルコンバートユーティリティ

(注)M28のみ存在する命令(予約語)を含むプログラムはM25にファイル変換しないでください。

BASIC-M28上で動作させるためには、MS-DOSで管理できるファイル形式にする必要があります。
このファイル変換(コンバート)を行うには、システムディスク内に収められている「ファイルコンバートユーティリティ」を使用します。

ファイルコンバートユーティリティを使用すると、次のファイルコンバートが行えます。

①MZ-2500のBASIC管理ファイルからMS-DOS管理ファイルへのコンバート

②MS-DOS管理ファイルからMZ-2500のBASIC管理ファイルへのコンバート

なお、いずれの場合もコンバートできるファイルは

- BASICプログラムファイル(MZ-2500では**BTX**形式)
- BASICアスキープログラムファイル、シーケンシャルアクセスデータファイル(MZ-2500では**BSD**形式)
- ランダムアクセスデータファイル(MZ-2500では**BRD**形式)
- 機械語ファイル(MZ-2500では**OBJ**形式)

です。

(注) 機械語ファイルの内容の変換は行われません。

起動画面

MS-DOSでプロンプトが表示されている状態で

A>FCNV.COM [↵] (システムディスクがドライブAにあるとき)

と入力すると、ファイルコンバートユーティリティが起動され、次のような画面が表示されます。

```
<<ファイルコンバート ユーティリティ VX. XX>>

        1: M25BASIC  =>  MS-DOS
        2: MS-DOS    =>  M25BASIC
        3: 終了

      カーソルキーを移動させて選択して下さい
```

カーソルキーでコンバートを行いたいタイプへ帯を移動させ、[↵]キーを押してください。

MZ-2500 BASICからMS-DOSへのコンバート

起動画面で1を選択した後、次の画面が表示されます。

```
<<ファイルコンバート ユーティリティ VX. XX>>
DRIVE:

M25BASIC  =>  MS-DOS

ドライブ名は
```

ここで、コンバートを行うディスクの入ったドライブ名を入力して、[↵]キーを押してください。
このとき、ディクレトリパスも同時に指定することができます。

例: A: [↵]

次のような画面が表示されます。

```
<<ファイルコンバート ユーティリティ VX.XX>>
DRIVE:A:
[file 1]  file 2  file 3

M25BASIC  =>  MS-DOS

旧ファイル名は
```

ここでは、指定されたドライブに挿入されたディスク内のファイル名が一覧表示されます。また、ディクレトリパスが指定された場合はディレクトリ内のファイル名が表示されます。
なお、ファイル数が多く、すべてのファイルを表示できない場合は次のようなメッセージが表示されます。

Shift+up:Roll down　　Shift+down:Roll up

このときは、[シフト]+[↓]と押すと画面が上方向にスクロールし、[シフト]+[↑]と押すと画面が下方向にスクロールして、ファイルを表示します。

ここで、コンバートしたいファイル名にカーソルキーで帯を移動させ、[↵]キーを押してください。
次のような画面が表示されます。

```
<<ファイルコンバート ユーティリティ VX. XX>>
DRIVE:A:
file 1   file 2   file 3


M25BASIC ＝>  MS-DOS
旧ファイル名:file 1


新ファイル名は
```

ここで、コンバート後のファイル名を入力し、[↵]キーを押します。
ディレクトリパスも同時に指定することができます。

例: B:FILE 1 . BAS[↵]

次のような画面が表示されます。

```
<<ファイルコンバート ユーティリティ VX. XX>>
DRIVE:A:
file 1   file 2   file 3


M25BASIC  ＝>  MS-DOS
旧ファイル名:file 1
新ファイル名:B:FILE 1 . BAS
ファイル形式:BTX
よろしいですか？ [実行]  再設定  メインへ戻る
```

ここで、そのまま[↵]キーを押すとコンバートを実行します。
再設定をする場合は「再設定」、起動画面に戻る場合は「メインへ戻る」にカーソルキーで帯を移動させ、[↵]キーを押してください。

MS-DOSからMZ-2500 BASICへのコンバート

手順はMZ-2500BASICからMS-DOSへのコンバートの場合とほぼ同じです。

- 起動画面で **2** を選択します。
- ソースドライブを指定します。
- コンバートするファイル名を指定します。
- コンバート後のファイル名を指定します。

• ファイル形式を指定します。

画面下段に次のようなメッセージが表示されますので、コンバートするファイルの形式に合わせて、カーソルキーで帯を移動させ、[↵]キーを押してください。

```
ファイル形式は BTX  BSD  BRD  OBJ
```

• コンバートを実行します。

エラーメッセージ(ファイルコンバータ)

ファイルコンバート中にエラーが発生した場合、次のようなメッセージが表示されます。

表示メッセージ	エラーの内容
ドライブの指定が間違っています。	ドライブの指定方法が誤っています。
指定されたファイルが存在しません。	指定されたファイルまたはディレクトリが存在しません。
ドライブの準備ができていません。	存在しないドライブを指定したか、ドライブが可動状態でない、またはドライブにディスクが入っていないことを表します。
ライトプロテクトされたファイルです。	指定されたファイルがロックされています。
ライトプロテクトされたディスクです。	ディスクが書き込み禁止になっています。
読み込むことができません。	
書き込むことができません。	
ディスクに異常があります。	ディスクの内容が異常です。
ファイルの大きさが制限を越えました。	ファイルのサイズが大きすぎます。
同一ドライブのコンバートはできません。	コンバート元ドライブとコンバート先ドライブが同じになっています。
メモリー不足です。コンバートできません。	コンバートファイルを読み出すメモリ容量が不足しています。
ファイルを作成するスペースがありません。	ディスクの空き容量が不足しています。
ファイル名入力エラー。	ファイル名(ディレクトリパス名)の指定方法が誤っています。

上記のエラーメッセージが表示された場合、任意のキーを押すことによってエラーが解除されます。

同一ファイル名が存在するときは

同一のファイルが存在する場合は、次のメッセージが表示されます。

同一ファイルが存在します。更新しますか? NO YES

索引（機能別）

基本コマンド

run …… 135
プログラムの実行を開始します。

cont …… 136
中断したプログラムの実行を再開します。

step …… 137
プログラムのステップ実行を行います。

stop …… 138
プログラムのブレークポイントを設定します。

new …… 139
プログラムを消去します。

tron / ltron …… 140
プログラムの実行を追跡します。

troff …… 141
tron の機能を解除します。

boot …… 142
BASIC の処理を中止し、システムを再起動します。

list / llist …… 143
プログラムリストを出力します。

list〈ファイル〉 …… 144
ファイルに対してリスト出力を行います。

list＊ / llist＊ …… 145
コメント行のみをリストします。

auto …… 146
行番号の自動発生を行います。

auto＊ …… 147
アポストロフィ (') 付きの行番号の自動発生を行います。

edit …… 148
エディットモードにして編集を行います。

edit＊ …… 149
エディットモードでコメント行を発生します。

search / lsearch …… 150
プログラム中より指定文字列をさがし出して表示します。

replace …… 152
プログラム中より指定文字列をさがし出して置き換えます。

delete …… 153
プログラムの一部を消去します。

renum …… 154
プログラムの行番号を付け直します。

option list …… 155
予約語表示の大文字、小文字を切り替えます。

system …… 156
MS–DOS へ制御を移します。

一般ステートメント

let …… 157
変数にデータを代入します。

def int / def sng / def dbl / def str …… 158
変数の型宣言を行います。

def FN …… 160
関数を定義します。

dim …… 162
配列の宣言を行います。

option base …… 164
配列の添字の下限を設定します。

erase …… 165
配列を消去します。

option angle …… 166
角度の単位を変更します。

rem …… 167
注釈を入れます。

randomize …… 168
乱数の発生系列を変更します。
clear …… 169
変数、配列の消去およびBASICエリアの制限を行います。
swap …… 171
変数の内容を交換します。
mid$= …… 172
文字列型変数の内容の一部を置き換えます。
search（関数） …… 173
配列内の一致するデータをさがします。

基本入出力命令

print / lprint …… 175
データの表示を行います。
tab …… 176
画面の左端よりの文字表示位置を指定します。
print using / lprint using …… 177
データを指定された書式で表示します。
write …… 179
データを表示します。
input …… 180
キーよりデータの入力を行います。
line input …… 182
キーより1行分のデータを入力します。
input wait / line input wait …… 183
キー入力の時間制御を行います。
read～data …… 184
プログラム中に書かれたデータを変数に読み込みます。
dtl …… 186
read文によるデータの読み取り行を求めます。
restore …… 187
データの読み取り位置を変更します。

実行制御命令

goto …… 188
指定された行へ実行を移します。
gosub～return …… 189
サブルーチンを実行します。
on～goto / gosub / return / restore / resume …… 191
式の値により参照行を変更します。
if～then～else …… 192
条件により処理を変えます。
else if～then……end if …… 194
if～then～else機能の複数行型式を行います。
for～next …… 197
forとnextの間の文を繰り返し実行します。
repeat～until …… 198
条件により繰り返し部分に戻るかを判断します。
while～wend …… 199
whileの条件に合えばwend間を繰り返します。
pause …… 200
プログラムの実行の一時停止をします。
stop …… 201
プログラムの実行を中止します。
end …… 202
プログラムの実行を終了します。

表示設定命令

option screen …… 203
グラフィックV-RAMの使用目的を設定します。
init "CRT2:" …… 205
グラフィック表示の初期化を行います。
init "CRT1:" …… 207
文字画面の初期化を行います。
screen …… 209
文字およびグラフィック画面の入出力設定を行います。

color……212
画面の表示色を設定します。
cls……214
画面表示の消去を行います。

文字表示命令

width……215
文字の出力桁数や行数を設定します。
console……217
文字のスクロール範囲を指定します。
console@……219
文字画面のディスプレイの出力範囲を設定します。
color@……220
指定範囲の文字色を変更します。
locate……221
カーソルの移動を行います。
cflash……222
文字を点滅させます。
cflash@……223
範囲を指定して文字を点滅させます。
crev……224
文字を反転させます。
crev@……225
範囲を指定して文字を反転させます。
cgen……226
キャラクタジェネレータの切り替えを行います。
csrlin……227
カーソルの垂直位置を求めます。
pos……228
カーソルの水平位置を求めます。
scrn$……229
文字画面より連続した文字を取り出します。
def chr$……230
PCG を定義します。
cgpat$……232
文字のパターンを読み出します。
get%……234
文字画面の表示データを取り込みます。
put%……235
get% 文で読み取ったデータを表示します。

グラフィック命令

color=……236
カラーパレットの設定を行います。
priority……238
表示の優先順位を設定します。
view（ステートメント）……240
グラフィックのビューポートを設定します。
view@……241
ディスプレイ上に表示するグラフィック画面の範囲を指定します。
view（関数）……243
現在のビューポートの設定位置を求めます。
window（ステートメント）……244
ビューポート論理座標を設定します。
window（関数）……245
現在のウインドゥの設定位置を求めます。
pen……246
ドットおよび線描画のペン形状を指定します。
pset……248
ドットのセットを行います。
preset……249
ドットのリセットを行います。
line……250
指定座標間を直線で結びます。
connect……252
直線を続けて引きます。
circle……254
円を描きます。
poly……257
多角形を描きます。

paint……259
境界内を塗りつぶします。
pattern……261
グラフィック画面にパターンを描きます。
symbol……263
任意の大きさで文字パターンを描きます。
color replace……265
グラフィック画面の表示色を変更します。
move……266
グラフィック表示の移動を行います。
roll……268
グラフィック画面のスクロールを行います。
roll@……270
グラフィックのディスプレイ表示のスクロールを行います。
map……271
スクリーン座標とワールド座標の相互変換を行います。
point（ステートメント）……272
グラフィックポインタの位置を設定します。
point（関数）……273
グラフィックの指定座標の表示色やグラフィックポインタの位置を求めます。
get@……274
グラフィック画面データを配列に取り込みます。
put@……276
グラフィックパターンの表示を行います。
gsave……278
グラフィック表示をファイルに書き込みます。
gload……280
ファイルからグラフィックデータを呼び出します。

ファイルコマンド

init "FDn:"……282
フロッピーディスクドライブの種類を選択します。
init "MEM:"……284
メモリディスクやスプーラの容量を設定します。
init "COMn:"……286
RS-232C インターフェイスの各種設定します。
files / lfiles……287
登録されているファイルを表示します。
chdir……289
カレントディレクトリを変更します。
mkdir……291
ディレクトリファイルを作成します。
rmdir……293
ディレクトリファイルを削除します。
run〈ファイル〉……294
指定ファイルを読み出し実行します。
load……296
プログラムファイルをメモリに読み出します。
load*……298
アスキーファイルをメモリに読み出します。
save……299
プログラムを指定の装置に記録します。
bload……301
機械語プログラムを読み込みます。
bsave……303
機械語プログラムを記録します。
merge……305
プログラムを併合します。
kill……307
ファイルを消去します。
name……309
ファイルの登録名を変更します。
set……311
ファイルにプロテクトをかけます。

verify on / off ……313
フロッピーディスクへの書き込みチェックを指定します。

ファイルステートメント・関数

chain ……314
現在実行中のプログラムから指定プログラムファイルを読み出し実行します。
common ……316
共通変数の定義を行います。
open ……317
データファイルの操作を開始します。
close ……320
ファイル操作を終了します。
print＃ ……321
オープンしたファイルへデータを書き込みます。
print＃using ……323
書式設定をしてファイルに書き込みます。
input＃ ……324
オープンされたファイルよりデータを読み出します。
write＃ ……325
オープンしたファイルへデータを書き込みます。
line input＃ ……326
ファイルより行単位でデータを入力します。
field ……327
レコードバッファに変数を割り付けます。
put＃ ……329
ファイルへデータを書き込みます。
get＃ ……330
ファイルからデータを読み出します。
lset / rset ……331
レコードバッファにデータを書き込みます。
mki$ / mks$ / mkd$ ……332
数値データを内部表現に対応した文字列に変換します。
cvi / cvs / cvd ……333
内部表現に対応した文字列データを数値データに変換します。
devo$ ……334
指定の装置へ1レコードのデータを書き込みます。
devi$ ……335
指定の装置より1レコードのデータを読み出します。
attr$ ……336
ファイルの属性を調べます。
eof ……338
ファイルの終りであるかどうかを調べます。
lof ……339
ファイルの大きさを求めます。
loc ……340
ファイル中での論理的な現在位置を求めます。
devf ……342
ファイル装置の空き容量を求めます。
input$ ……343
文字長を指定してファイルまたはキーボードから入力を行います。
pwd$ ……344
カレントディレクトリのパスを求めます。

数値関数

abs ……345
絶対値を求めます。
sgn ……346
符号を求めます。
int ……347
値をこえない最大の整数を求めます。
fix ……348
整数部を求めます。

frac ····· 349
小数部を求めます。
sin ····· 350
正弦を求めます。
cos ····· 351
余弦を求めます。
tan ····· 352
正接を求めます。
atn ····· 353
逆正接を求めます。
sqr ····· 354
平方根を求めます。
exp ····· 355
指数関数を求めます。
log ····· 356
自然対数を求めます。
log10 ····· 357
常用対数を求めます。
pai ····· 358
円周率を求めます。
rad ····· 359
度からラジアンに変換します。
deg ····· 360
ラジアンから度に変換します。
fac ····· 361
階乗を求めます。
sum ····· 362
式の総和を求めます。
rnd ····· 363
乱数を求めます。
cint ····· 364
数値を整数型に変換します。
csng ····· 365
数値を単精度型に変換します。
cdbl ····· 366
数値を倍精度型に変換します。

文字列関数

left$ ····· 367
文字列の左側から任意長の文字列を取り出します。
mid$ ····· 368
文字列の途中から任意長の文字列を取り出します。
right$ ····· 369
文字列の右側から任意長の文字列を取り出します。
asc ····· 370
文字列の最初の文字の文字コードを求めます。
chr$ ····· 371
文字コードにより文字データを求めます。
val ····· 372
数字の文字列を数値データに変換します。
str$ ····· 373
数値データを文字列データに変換します。
len ····· 374
文字列の文字長を求めます。
mirror$ ····· 375
文字コードの対称コードを求めます。
space$ / spc ····· 376
任意の長さのスペース文字を求めます。
string$ ····· 377
文字列の任意の繰り返しを求めます。
instr ····· 378
文字列中の特定文字をさがし位置を求めます。
hex$ ····· 379
数値データを16進数の文字列に変換します。
oct$ ····· 380
数値データを 8 進数の文字列に変換します。

bin$ …… 381
数値データを2進数の文字列に変換します。
hexchr$ …… 382
16進数表記の文字列を文字コードとして文字列に変換します。
ascchr$ …… 383
文字列の文字コードを16進数表記に変換します。

漢字処理命令

kmode …… 384
漢字モードにします。
akcnv$ …… 385
半角文字を全角文字にします。
kacnv$ …… 386
全角文字を半角文字に変換します。
jis$ …… 387
シフトJISコードよりJISコードに変換します。
ktn$ …… 388
シフトJISコードよりJIS区点コードに変換します。
klen …… 389
全角文字を含む文字列の文字数を求めます。
kpos …… 391
文字列の先頭より指定文字までのバイト数を求めます。
kext$ …… 392
文字列中より英数カナ文字または漢字を取り出します。
kinput …… 393
キーボードから漢字を入力し変数に代入します。
kinstr …… 394
文字列中より特定の文字列をさがし位置を求めます。
kmid$ …… 395
漢字を含む文字列中より任意の長さの文字列を取り出します。
knj$ …… 396
漢字コード文字列を漢字に変換します。
ktype …… 397
漢字を含む文字列中の特定位置の文字タイプを求めます。

例外処理命令

on error goto …… 398
エラー処理の宣言を行います。
err …… 400
エラー番号を求めます。
erl …… 401
エラー発生行を求めます。
resume …… 402
エラー処理を終了してプログラムを再開します。
error …… 403
エラーを発生させます。
on key gosub …… 404
割り込みキーによる割り込み処理行を定義します。
key on / off / stop …… 406
キーによる割り込みを制御します。
on stop gosub …… 407
シフト+停止キーによる割り込み処理行を定義します。
stop on / off / stop …… 409
シフト+停止キーによる割り込みを制御します。
on help gosub …… 410
ヘルプキーによる割り込み処理行を定義します。
help on / off / stop …… 412
ヘルプキーによる割り込みを制御します。

on com gosub ……413
RS–232C インターフェイスによる割り込み処理行を定義します。
com on / off / stop……414
RS–232C インターフェイスによる割り込み処理を制御します。
on mouse gosub……415
マウスによる割り込み処理を定義します。
mouse on / off / stop ……416
マウスによる割り込み処理を制御します。
on time$ gosub ……418
リアルタイムクロックでの割り込み処理を定義します。
time$ on / off / stop……419
リアルタイムクロックの割り込みを制御します。
interval ……420
インターバルタイマの設定をします。
on interval gosub……421
インターバルタイマによる割り込み処理を定義します。
interval on / off / stop ……422
インターバルタイマによる割り込み処理を制御します。
on play gosub ……423
音楽演奏終了時に発生する割り込み処理を定義します。
play on / off / stop……425
音楽演奏終了時の割り込み機能を制御します。

チャイルドプロセス制御命令

child……426
チャイルドプロセスを呼び出し、実行します。
environ……429
環境文字列テーブルの内容を変更します。
environ$……430
指定された環境文字列のパラメータを求めます。

数値演算プロセッサ命令

option float ……431
数値演算プロセッサの使用を可能にします。

マウス命令

mouse（ステートメント）……432
マウスの機能を設定します。
mouse（関数）……436
マウスからの情報を読み取ります。

キーボード制御命令

init "KB:" ……438
キーボードの機能を設定します。
click on / off……439
キーのクリック音を制御します。
repeat on / off……440
キーリピートを制御します。
inkey$ ……441
キーボードより 1 文字入力を行います。
key……443
割り込みキーを定義します。
key list / klist ……447
ファンクションキーの定義状態を表示します。
key 0……448
キーボード入力のシミュレーションを行います。

ジョイスティック命令

stick ……449
ジョイスティックの状態を調べます。
strig ……450
ジョイスティックのトリガーボタンの状態を調べます。

stick on / off ……451
ジョイスティックでカーソルキーの代用をさせます。

プリンタ制御命令

init "LPT:"……452
プリンタスプーラの設定をします。
hcopy……453
画面の表示をプリンタにコピーします。
lpout ……454
プリンタにデータを出力します。
width lprint ……455
プリンタの印字幅を設定します。
lpos……456
現在のプリンタのヘッド位置を求めます。

クロック命令

time$……457
時刻の読み出しおよび設定を行います。
date$ ……458
日付の読み出しおよび設定を行います。
day$ ……459
曜日の読み出しおよび設定を行います。
time ……460
タイマーの読み出しおよび設定を行います。

サウンド命令

beep……461
スピーカを一瞬鳴らします。
play……462
音楽演奏をします。
tone ……469
FM 音源の音色を設定します。
tone copy ……475
BASIC で用意された音色データを配列に取り込みます。
tone lfo……476
ビブラートやトレモロ機能を設定します。
sound……478
SSG レジスタをセットします。

モデムホン制御命令

tel ……480
モデムホンへの制御を切り替えます。

ディスプレイテレビ制御命令

init "CRT4:" ……481
ディスプレイテレビの制御を行います。

機械語処理・特殊関数

peek……483
メモリよりデータを読み出します。
poke……485
メモリにデータを書き込みます。
def seg……487
セグメントベースアドレスを宣言します。
segptr ……488
各種メモリ領域のベースアドレスを求めます。
varptr ……490
変数・配列のベースアドレスを求めます。
def usr ……491
機械語プログラムの開始アドレスを定義します。
usr ……492
機械語プログラムを関数として呼び出します。
call ……493
機械語サブルーチンを呼び出します。
out……495
I/O ポートにデータを出力します。
inp ……496
I/O ポートよりデータを入力します。

wait ····· 497
I/O ポートからの入力が指定状態になるまで実行を停止します。
fre ····· 498
メモリの未使用量を求めます。

索引（アルファベット順）

A

abs 345
add(描画機能) 103
akcnv$ 385
and(論理演算) 38
and(描画機能) 103
asc 370
ascchr$ 383
atn 353
attr$ 336
auto 146
auto* 147

B

beep 461
bin$ 381
bload 301
boot 142
bsave 303

C

call 493
cdbl 366
cflash 222
cflash@ 223
cgen 226
cgpat$ 232
chain 314
chdir 289
child 426
chr$ 371
cint 364
circle 254
clear 169
click on/off 439
close 320
cls 214
color 212
color replace 265
color@ 220
color= 236
common 316
com on/off/stop 414
connect 252
console 217
console@ 219
cont 136
cos 351
crev 224
crev@ 225
csng 365
csrlin 227
cvd 333
cvi 333
cvs 333

D

data 184
date$ 458
day$ 459
def chr$ 230
def dbl 158
def FN 160
def int 158
def seg 487
def sng 158
def str 158
def usr 491

deg 360
delete 153
devf 342
devi$ 335
devo$ 334
dim 162
dtl 186

E

edit 148
edit* 149
else if~then...end if 194
end 202
end if 194
environ 429
environ$ 430
eof 338
eqv 38
erase 165
erl 401
err 400
error 403
exp 355

F

fac 361
field 327
files 287
fix 348
for~next 197
fpset 104
frac 349
fre 498

G

get@ 274
get% 234
get# 330
gload 280
gosub~return 189
goto 188
gsave 278

H

hcopy 453
help on/off/stop 412
hex$ 379
hexchr$ 382

I

if~then~else 192
imp 38
init "COMn:" 286
init "CRT1:" 207
init "CRT2:" 205
init "CRT4:" 481
init "FDn:" 282
init "KB:" 438
init "LPT:" 452
init "MEM:" 284
inkey$ 441
inp 496
input 180
input wait 183
input$ 343
input # 324
instr 378
int 347
interval 420
interval on/off/stop 422

J

jis$ 387

K

kacnv$ ····· 386
kext$ ····· 392
key ····· 443
key list ····· 447
key on/off/stop ····· 406
key 0 ····· 448
kill ····· 307
kinput ····· 393
kinstr ····· 394
klen ····· 389
klist ····· 447
kmid$ ····· 395
kmode ····· 384
knj$ ····· 396
kpos ····· 391
ktn$ ····· 388
ktype ····· 397

L

left$ ····· 367
len ····· 374
let ····· 157
lfiles ····· 287
line ····· 250
line input ····· 182
line input wait ····· 183
line input # ····· 326
list ····· 143
list＜ファイル＞ ····· 144
list＊ ····· 145
llist ····· 143
llist＊ ····· 145
load ····· 296
load＊ ····· 298
loc ····· 340
locate ····· 221
lof ····· 339
log ····· 356
log10 ····· 357
lpos ····· 456
lpout ····· 454
lprint ····· 175
lprint using ····· 177
lsearch ····· 150
lset ····· 331
ltron ····· 140

M

map ····· 271
merge ····· 305
mid$ ····· 368
mid$= ····· 172
mirror$ ····· 375
mkd$ ····· 332
mkdir ····· 291
mki$ ····· 332
mks$ ····· 332
mod ····· 36
mouse(ステートメント) ····· 432
mouse(関数) ····· 436
mouse on/off/stop ····· 416
move ····· 266

N

name ····· 309
new ····· 139
next ····· 197
not ····· 38

O

oct$ ····· 380
on com gosub ····· 413
on error goto ····· 398

on～gosub …… 191
on～goto …… 191
on help gosub …… 410
on interval gosub …… 421
on key gosub …… 404
on mouse gosub …… 415
on play gosub …… 423
on～restore …… 191
on～resume …… 191
on～return …… 191
on stop gosub …… 407
on time$ gosub …… 418
open …… 317
option angle …… 166
option base …… 164
option float …… 431
option list …… 155
option screen …… 203
or〈論理演算〉…… 38
or〈描画機能〉…… 103
out …… 495

P

pai …… 358
paint …… 259
pattern …… 261
pause …… 200
peek …… 483
pen …… 246
play …… 462
play on/off/stop …… 425
point〈ステートメント〉…… 272
point〈関数〉…… 273
poke …… 485
poly …… 257
pos …… 228
preset …… 249
print …… 175
print using …… 177
print # …… 321
print # using …… 323
priority …… 238
pset …… 248
put@ …… 276
put% …… 235
put# …… 329
pwd$ …… 344

Q

R

rad …… 359
randomize …… 168
read～data …… 184
rem …… 167
renum …… 154
repeat on/off …… 440
repeat～until …… 198
replace …… 152
restore …… 187
resume …… 402
return …… 189
right$ …… 369
rmdir …… 293
rnd …… 363
roll …… 268
roll@ …… 270
rset …… 331
run …… 135
run〈ファイル〉…… 294

S

save …… 299
scr …… 104

screen ····· 209
scrn$ ····· 229
search(コマンド) ····· 150
search(関数) ····· 173
segptr ····· 488
set ····· 311
sgn ····· 346
sin ····· 350
sound ····· 478
space$ ····· 376
spc ····· 376
sqr ····· 354
step ····· 137
stick ····· 449
stick on off ····· 451
stop(コマンド) ····· 138
stop(ステートメント) ····· 201
stop on off stop ····· 409
str$ ····· 373
strig ····· 450
string$ ····· 377
sub(描画機能) ····· 103
sum ····· 362
swap ····· 171
symbol ····· 263
sysid$ ····· 41
system ····· 156

T

tab ····· 176
tan ····· 352
tel ····· 480
time ····· 460
time$ ····· 457
time$ on off stop ····· 419
tone ····· 469
tone copy ····· 475
tone lfo ····· 476
troff ····· 141
tron ····· 140

U

until ····· 198
usr ····· 492

V

val ····· 372
varptr ····· 490
verify on off ····· 313
view(ステートメント) ····· 240
view(関数) ····· 243
view@ ····· 241

W

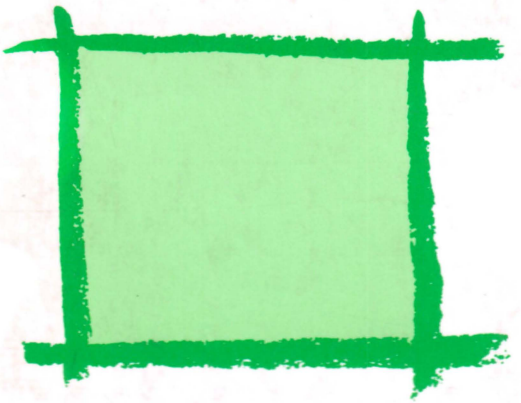
wait ····· 497
wend ····· 199
while～wend ····· 199
width ····· 215
width lprint ····· 455
window(ステートメント) ····· 244
window(関数) ····· 245
write ····· 179
write# ····· 325

X

xor(論理演算) ····· 38
xor(描画機能) ····· 104

Y

Z



シャープ株式会社

本　　社　〒５４５　大阪市阿倍野区長池町22番22号
　　電話　06(621)1221（大代表）
情報システム　〒639-11　奈良県大和郡山市美濃庄町492
事業本部　　電話　07435(3)5521（大代表）

©1987 SHARP CORPORATION
7D　2.6-I(TINSJ1409ACZZ)①