

NECパーソナルコンピュータ
PC-9800シリーズ

NEC

Software Library

N₈₈-日本語BASIC(86)(Ver 6.1)
リファレンスマニュアル

Software Library

N₈₈-日本語BASIC(86)(Ver 6.1)
リファレンスマニュアル

はじめに

PC-9800 シリーズパーソナルコンピュータには標準プログラミング言語としてN₈₈-BASIC (86)が用意されています。

N₈₈-BASIC (86) 言語で記述されたプログラムは、本体ROMあるいは各種インタフェースボードに搭載されているN₈₈-BASIC (86) インタプリタにより解釈／実行されます。

また、N₈₈-日本語BASIC (86) システムディスクを使用することにより、ROMに搭載されているN₈₈-BASIC (86) インタプリタの持つ機能に加え、さまざまな機能を利用することができます。

本書は、N₈₈-BASIC (86) 言語が持つ機能を、命令・関数単位で詳細に解説した文法説明書です。

N₈₈-BASIC (86) インタプリタの操作法、動作環境、あるいは応用的な操作については、『N₈₈-日本語 BASIC (86) ユーザーズマニュアル』(以下『BASIC ユーザーズマニュアル』と略します)を参照してください。

具体的なプログラミングの方法については『N₈₈-日本語 BASIC (86) 入門』をご覧ください。日本語入力の具体的な方法については『日本語入力ガイド』を参照してください。

このマニュアルの構成

本書は、次の4章および付録から構成されています。

第1章 N₈₈-BASIC(86)について

N₈₈-BASIC(86)の特徴、主な機能のほか、ROMモードBASICとDISKモードBASICの違いや、プログラムの作り方などの基本的な事項を解説しています。

第2章 BASICの文法

N₈₈-BASIC(86)言語の文法・規則についての解説です。本BASICを利用する前には必ず本章を読み、その内容を理解・把握しておいてください。

第3章 命令リファレンス

N₈₈-BASIC(86)で使用可能な命令・関数をアルファベット順に並べ、個々に詳細な解説をしています。本BASICの詳しい辞書として利用してください。

第4章 サンプルプログラム

相互に関連のある命令・関数を使用して作成したサンプルプログラム集です。実際にプログラムを作成する際の参考にしてください。

付 録

エラーメッセージとその対策、キャラクタコード表、予約語一覧などの付加情報・資料です。

目次

はじめに	(3)
このマニュアルの構成	(4)
命令・関数の機能別索引	(11)

第1章 N₈₈-BASIC(86)について

1. ROMモードBASICとDISKモードBASIC	3
2. N ₈₈ -BASIC(86)の主な機能	3
3. BASICの動作モード	4
3.1 ダイレクトモード	4
3.2 プログラムモード	5
・プログラムの作り方	
・プログラムの修正	
・プログラムの保存(セーブ)と読み込み(ロード)	
4. 使用機種による相違	7

第2章 BASICの文法

1. 文	11
2. 行	11
3. 行番号	11
4. BASICで使用できる文字と特殊記号	12
・特殊記号の使い方	
5. 定数	14
5.1 文字型定数	14
5.2 数値型定数	14
5.3 整数型定数	14
・10進形式	
・8進形式	
・16進形式	
5.4 実数型定数	15
5.5 単精度実数型定数	15
5.6 倍精度実数型定数	16
6. 変数	16

6.1	変数名	16
6.2	変数の型	16
7.	配列変数	17
8.	予約語	19
9.	型変換	19
10.	式と演算	21
10.1	算術演算	21
	・整数の除算と剰余の計算	
	・0での除算	
	・桁あふれ(オーバーフロー)	
10.2	関係演算	23
10.3	論理演算	23
10.4	関数	26
10.5	文字列の演算	26
	・文字列の連結	
	・文字列の比較	
10.6	演算の優先順位	27
11.	ファイル	27
11.1	ファイルとは	27
11.2	ファイルディスクリプタ	28
	・デバイス名	
	・ファイル名	
11.3	データファイルとファイル番号	29
12.	割り込み	30
13.	ラベル名	31
14.	エラーメッセージ	32

第3章 命令リファレンス

この章の見方	37		
ABS	40	BEEP	42
AKCNV\$	40	BLOAD	42
ASC	40	BSAVE	43
ATN	41	CALL	44
ATTR\$	41	CDBL	44
AUTO	42	CHAIN	45

CHR\$	46	DEF SEG	72
CINT	46	DEF USR	73
CIRCLE	47	DELETE	73
CLEAR	48	DIM	74
CLOSE	49	DRAW	75
CLS	49	DSKF	79
CMD BREAK	50	DSKI\$	80
CMD CHANGE DUPLEX	51	DSKO\$	80
CMD DIAL	51	EDIT	81
CMD ERROR ON/OFF/STOP	53	END	82
CMD LINE CLOSE	54	EOF	82
CMD LINE ON/OFF/STOP	54	ERASE	82
CMD LINE OPEN	55	ERL/ERR	83
CMD MODE CUT	56	ERROR	83
CMD ON ERROR GOSUB	56	EXP	84
CMD ON LINE GOSUB	57	FIELD	84
CMD RECEIVE	58	FILES/LFILES	85
CMD STORE DIAL	58	FIX	86
{1} COLOR	59	FOR...TO...STEP~NEXT	86
{2} COLOR	62	FPOS	87
COLOR@	63	FRE	87
COMMON	64	GET	88
COM ON/OFF/STOP	65	GET@	89
CONSOLE	66	GOSUB	90
CONT	66	GOTO/GO TO	90
COPY	67	HELP ON/OFF/STOP	91
COS	68	HEX\$	92
CSNG	68	IF...THEN~ELSE /IF...GOTO~ELSE	92
CSRLIN	69	INKEY\$	93
CVI/CVS/CVD	69	INP	93
DATA	70	INPUT	94
DATE\$	70	INPUT#	95
DEF FN	71	INPUT\$	95
DEFINT/DEFSNG/DEFDBL /DEFSTR	72	INPUT WAIT	96
		INSTR	96

INT	97	MKI\$/MKS\$/MKD\$	116
JIS\$	97	MON	117
KACNV\$	97	NAME	117
KEXT\$	98	NEW	117
KEY	98	NEW ON	118
KEY LIST	99	OCT\$	118
KEY ON/OFF/STOP	99	ON COM GOSUB	119
KILL	100	ON ERROR GOTO	120
KINPUT	100	ON...GOSUB/ON...GOTO	121
KINSTR	101	ON HELP GOSUB	121
KLEN	101	ON KEY GOSUB	122
KMID\$.....	102	ON PEN GOSUB	123
KNJ\$	102	ON PLAY GOSUB	124
KPLOAD	103	ON STOP GOSUB	125
KTYPE	104	ON TIME\$ GOSUB	125
LEFT\$.....	104	OPEN	126
LEN	105	OPTION BASE	129
LET	105	OUT	129
LINE	106	{1} PAINT	130
LINE INPUT	107	{2} PAINT	130
LINE INPUT#	108	PEEK	132
LINE INPUT WAIT	108	PEN	133
LIST/LLIST	109	PEN ON/OFF/STOP	134
LOAD	109	PLAY	134
LOC	110	PLAY ALLOC	142
LOCATE	110	PLAY ON/OFF/STOP	143
LOF	111	POINT	144
LOG	111	{1} POINT	144
LPOS	112	{2} POINT	145
LPRINT	112	POKE	145
LPRINT USING	112	POS	146
LSET/RSET	113	PRESET	146
MAP	113	PRINT	147
MERGE	114	PRINT#	148
MID\$	115	PRINT USING	150
MID\$(関数)	115	PRINT# USING	152

PSET	152	STOP	172
PUT	153	STOP ON/OFF/STOP	172
PUT@	154	STR\$	173
RANDOMIZE	155	STRING\$	173
READ	156	SWAP	174
REM	156	TAB	174
RENUM	157	TAN	175
RESTORE	157	TERM	175
RESUME	158	TIME\$	177
RETURN	158	TIME\$ ON/OFF/STOP	178
RIGHT\$	159	TRON/TROFF	178
RND	159	USR	179
ROLL	160	VAL	179
RUN	160	VARPTR	180
SAVE	161	VIEW	181
SCREEN	162	VIEW(関数)	182
SEARCH	165	VOICE	182
SET	166	VOICE COPY	184
SGN	167	VOICE INIT	184
SIN	167	VOICE LFO	185
SPACE\$	168	VOICE REG	187
SPC	168	WAIT	187
SQR	168	WHILE~WEND	188
STATUS DIAL	169	WIDTH	188
STATUS DIAL\$	169	WIDTH LPRINT	189
STATUS ERROR	170	WINDOW	190
STATUS LINE	170	WINDOW(関数)	191
STATUS MODE	171	WRITE	191
STATUS PLAY	171	WRITE#	192

第4章 サンプルプログラム

この章の見方	195
サンプルプログラム	198

付 録

A. エラーメッセージとその対策	213
B. コントロールコード表	222
C. キャラクタコード表	223
D. 誘導関数	224
E. 予約語一覧	225
索 引	227

命令・関数の機能別索引

コマンド

プログラム編集用命令

AUTO	42
DELETE	73
EDIT	81
KEY LIST	99
LIST/LLIST	109
LOAD	109
MERGE	114
RENUM	157
SAVE	161

プログラム実行制御用命令

CONT	66
NEW	117
NEW ON	118
RUN	160
TRON/TROFF	178

ファイル操作命令

FILES/LFILES	85
KILL	100
NAME	117
SET	166

一般命令

命令

CHAIN	45
COMMON	64
DATA	70
DEF FN	71
DEFINT/DEFSNG/DEFDBL/DEFSTR	72
DIM	74
END	82
ERASE	82
FOR...TO...STEP~NEXT	86
GOSUB	90
GOTO/GO TO	90
IF...THEN~ELSE/ IF...GOTO~ELSE	92

LET	105
ON...GOSUB/ON...GOTO	121
OPTION BASE	129
RANDOMIZE	155
READ	156
REM	156
RESTORE	157
RETURN	158
STOP	172
SWAP	174
WHILE~WEND	188

関数

SEARCH	165
--------------	-----

テキスト画面制御

命令

CLS	49
[1]COLOR	59
COLOR@	63
CONSOLE	66
LOCATE	110
PRINT	147
PRINT USING	150

WRITE	191
-------------	-----

関数

CSRLIN	69
POS	146
SPC	168
TAB	174

グラフィック画面制御

命令

CIRCLE	47
CLS	49
[2]COLOR	62
DRAW	75
GET@	89
LINE	106
[1]PAINT	130
[2]PAINT	130
POINT	144
PRESET	146
PSET	152

PUT@	154
ROLL	160
SCREEN	162
VIEW	181
WINDOW	190

関数

MAP	113
[1]POINT	144
[2]POINT	145
VIEW	182
WINDOW	191

算術関数

関数

ABS	40	FIX	86
ATN	41	INT	97
CDBL	44	LOG	111
CINT	46	RND	159
COS	68	SGN	167
CSNG	68	SIN	167
CVI/CVS/CVD	69	SQR	168
EXP	84	TAN	175

エラー制御

命令

ERROR	83
ON ERROR GOTO	120
RESUME	158

関数

ERL/ERR	83
---------------	----

1バイト系文字列操作

命令

MID\$	115
-------------	-----

関数

ASC	40	LEN	105
CHR\$	46	MID\$	115
HEX\$	92	MKI\$/MKD\$/MKD\$	116
INSTR	96	OCT\$	118
LEFT\$	104	RIGHT\$	159
		SPACE\$	168
		STR\$	173
		STRING\$	173
		VAL	179

日本語文字列操作

命 令

KPLOAD103

関 数

AKCNV\$40

JIS\$97

KACNV\$97

KEXT\$98

KINSTR101

KLEN101

KMID\$102

KNJ\$102

KTYPE104

ファイル制御

命 令

CLOSE49

FIELD84

GET88

INPUT#95

LINE INPUT#108

LSET/RSET113

OPEN126

PRINT#148

PRINT# USING152

PUT153

WRITE#192

関 数

ATTR\$41

DSKF79

EOF82

FPOS87

INPUT\$95

LOC110

LOF111

キー制御

命 令		関 数	
HELP ON/OFF/STOP	91	LINE INPUT WAIT	108
INPUT	94	ON HELP GOSUB	121
INPUT WAIT	96	ON KEY GOSUB	122
KEY.....	98	ON STOP GOSUB	125
KEY ON/OFF/STOP	99	STOP ON/OFF/STOP	172
KINPUT	100		
LINE INPUT	107	INKEY\$	93

プリンタ制御

命 令		関 数	
COPY	67	LPOS	112
LPRINT	112	SPACE\$	168
LPRINT USING	112	SPC	168
WIDTH LPRINT	189		

時刻/日付け制御

命 令		関 数	
ON TIME\$ GOSUB	125	DATE\$	70
TIME\$ ON/OFF/STOP.....	178	TIME\$	177

RS-232C コミュニケーションポート制御

命 令		関 数	
COM ON/OFF/STOP	65	ON COM GOSUB	119
		OPEN	126

ライトペン制御

命 令	関 数
ON PEN GOSUB123	PEN133
PEN ON/OFF/STOP.....134	

I/O ポート入出力制御

命 令	関 数
BEEP42	INP93
OUT129	
WAIT187	

メモリ管理

命 令	関 数
CLEAR48	FRE.....87
DEF SEG72	PEEK132
POKE145	VARPTR180

機械語プログラム制御

命 令	関 数
BLOAD42	DEF USR73
BSAVE43	
CALL44	USR179

モード変更

命 令	
MON	117
TERM	175

ディスク入出力

命 令	関 数
DSKO\$	80
DSKI\$	80

サウンド制御

命 令		関 数	
ON PLAY GOSUB	124	VOICE INIT	184
PLAY	134	VOICE LFO	185
PLAY ALLOC	142	VOICE REG	187
PLAY ON/OFF/STOP	143		
VOICE	182	関 数	
VOICE COPY	184	STATUS PLAY	171

電話制御

命 令		関 数	
CMD BREAK	50	CMD ON LINE GOSUB	57
CMD CHANGE DUPLEX	51	CMD RECEIVE	58
CMD DIAL	51	CMD STORE DIAL	58
CMD ERROR ON/OFF/STOP	53		
CMD LINE CLOSE	54	関 数	
CMD LINE ON/OFF/STOP	54	STATUS DIAL	169
CMD LINE OPEN	55	STATUS DIAL\$	169
CMD MODE CUT	56	STATUS ERROR	170
CMD ON ERROR GOSUB	56	STATUS LINE	170
		STATUS MODE	171

第1章

1

N₈₈-BASIC(86)について

第1章 N₈₈-BASIC(86)について

1. ROM モード BASIC と DISK モード BASIC

PC-9800 シリーズパーソナルコンピュータには、標準プログラム言語として N₈₈-BASIC (86) が用意されています。

N₈₈-BASIC (86) には、DISK モード BASIC と ROM モード BASIC の2つのモードがあります。

ROM モード BASIC は、コンピュータ本体の ROM に組み込まれている基本機能だけを使ったシステムです。システムディスクを使用しないで、電源を ON にすると起動します。

DISK モード BASIC は、ROM モード BASIC に、ディスクファイルに対する入出力や AI かな漢字変換による日本語入力などの機能を拡張したものです。DISK モード BASIC を起動するには、本体の電源を ON にした後、N₈₈-日本語 BASIC (86) システムディスクをフロッピーディスク装置にセットしてリセットスイッチを押します。

ROM モード BASIC でもプログラムの入力および実行は可能ですが、実際のプログラミングには DISK モード BASIC が適しています。

注) BASIC の起動方法について詳しくは、BASIC ユーザーズマニュアルを参照してください。

2. N₈₈-BASIC (86) の主な機能

N₈₈-BASIC (86) には、次のような特長があります。

(1) グラフィック機能の充実

- 最大640×400の高分解能、最高4096色中16色のカラー表現によるグラフィックス。
- 強力なグラフィック命令。
- テキスト画面とグラフィック画面の合成。
- パレットの導入による多彩なカラー表示。

(2) プログラミングのしやすさ

- プログラム中の飛び先指定としてラベル名の使用が可能。
- BASIC 実行時でも、大容量のユーザーメモリを確保しているため、大規模なプログラムの作成が可能。
- 各種ハードウェアからの割り込みを BASIC 中で操作可能。

- IF...THEN~ELSE や, WHILE~WEND などにより, 構造化されたプログラムの作成が可能.
 - CHAIN によるプログラム連結機能 (DISK モードのみ).
 - 変数名として, 最長40文字まで使用可能.
- (3) フロッピディスク, 固定ディスクの使用が可能 (DISK モードのみ).
- (4) AI 逐次変換, AI 連文節変換, 逐次変換, 連文節変換あるいは単文節変換による日本語入力の機能 (DISK モードのみ).
- (5) 電話機の制御機能 (DISK モードのみ).
- (6) 倍精度数値関数のサポート (DISK モードのみ).

3. BASIC の動作モード

BASIC を起動すると, 画面には,

```
How many files (0-15) ?
```

という文字が表示されます. このメッセージは“同時にいくつのファイルを使用しますか”と尋ねているものです. ここで数を指定すると, その数だけのファイルを同時に開くことができるようになります. 詳しくは, 第3章中の OPEN の項を参照してください.


ここで, 数を指定してからリターンキーを押すか, またはたんにリターンキーを押すと, BASIC のバージョンなどを示すメッセージが表示されます. メッセージ中最後行の“Ok”という文字は BASIC の命令をキーボードから入力することができることを示しているもので, “プロンプト”と呼ばれます.

BASIC の命令は, ダイレクトモードとプログラムモードの2つの動作モードで実行されます.

3.1 ダイレクトモード

BASIC の命令を, 単独で実行させるモードをダイレクトモードといいます. ダイレクトモードでの命令の実行は, 命令をキーボードから打ち込み, リターンキーを押すという動作 (これを命令の入力といいます) によって行われます.

たとえば, 「PRINT 21+6-5」とキーを打ち込み, 終わりにリターンキーを押してみてください.

```
PRINT 21+6-5 
```

```
22
```

```
Ok
```

となります。PRINTは“画面にデータを出力せよ”という命令ですから、この場合には、“21+6-5”の計算結果として“22”が表示されます。

このように、ダイレクトモードでBASICの命令を実行するときには、命令の最後にリターンキーを押さなければなりません。リターンキーを押すことによって、初めて命令がコンピュータに送られるのです。

ダイレクトモードで入力された命令はコンピュータのメモリに格納されませんので、その場限りの計算や記憶する必要のない命令を実行するのに使います。

なお、ダイレクトモードで、マルチステートメント（「第2章 2. 行」参照）を使用してもかまいません。

3.2 プログラムモード

BASICの命令を続けて実行させるためには、プログラムを作る必要があります。プログラムを入力し、実行するモードをプログラムモードといいます。

■プログラムの作り方

プログラムを作るときには、命令の先頭に数字をつけて入力していきます。各行の終わりには必ずリターンキーを押します。すると、行は数字とともにメモリに格納されます。この数字を行番号といいます。

プログラムを実行するには、プログラムを入力した後で、RUNを入力します。プログラムは行番号の小さいものから順に実行されます。

```
10 A=2
20 B=-5
30 PRINT A+B
RUN
-3
Ok
```

メモリにはいちどに1つのプログラムしか格納することはできません。メモリに古いプログラムがあるときは、新しいプログラムを入力する前に、必ずNEWを実行して、古いプログラムを消しておかねばなりません。

なお、プログラムモードでは、マルチステートメント（「第2章 2. 行」参照）を利用することができます。

■プログラムの修正

(1) 行の追加

プログラムは行番号の小さいものから実行されます。したがって、新しく行を追加したい場合は、追加したい所の行番号(前後の行番号の間の番号)をつけた命令を入力します。た

たとえば前記のプログラムの20行と30行の間に命令を入れるときには21～29までの行番号をつけることができます。

(2) 行の削除

削除したい行の行番号だけを打ち込んでリターンキーを押せば、その行は削除されます。まとめて削除するときには、DELETE を使います (第3章中の DELETE の項参照)。

(3) 行の置き換え

訂正したい行の行番号と新しい内容を打ち込んでリターンキーを押せば、古い行の内容は自動的に新しいものに変ります。また、LIST を使ってプログラムを画面に表示し、カーソルを直したい所へ移動して直接書き換えてからリターンキーを押しても、その行は訂正されます (第3章中の LIST の項参照)。

(4) 行番号のつけかえ

行番号をつけかえたいときには、RENUM を使います (第3章中の RENUM の項参照)。

■プログラムの保存(セーブ)と読み込み(ロード)

BASIC で作成したプログラムは、本体の電源を切ると消えてしまいます。そのため、必要なプログラムは、ファイルという形でディスクに書き出し、保存(セーブ)しておかなければなりません。

プログラムをセーブするには、SAVE を用います。プログラムは一度セーブしてしまえば、あとはいつでもメモリに読み込み(ロード)、使うことができます。プログラムをロードするには、LOAD を使います。

機械語プログラムをセーブ、ロードするための、BSAVE と BLOAD も用意されています。詳しくは第3章の各命令の項を参照してください。

なお、ディスクに対するセーブ・ロードは、DISK モードでのみ可能です。

4. 使用機種による相違

PC-9800 シリーズで提供される機能は、機種や接続機器の有無により多少異なります。そのため、N₈₈-BASIC(86)で実行できる命令も機種や接続機器の有無により異なる場合がありますのでご注意ください。

機 能	使用上の注意
拡張グラフィック機能	<ul style="list-style-type: none"> • PC-9801/E/F/M では使用できません。 • PC-9801 U/VF/VM0/VM2/VM4 では 16 色グラフィックボードが別途必要です。
サウンド制御機能	<ul style="list-style-type: none"> • PC-9801 UV/UX/CV/EX 以外の機種ではサウンドボードが別途必要です。
ライトペン機能	<ul style="list-style-type: none"> • PC-9800 本体にモノクロディスプレイ用コネクタがない機種では使用できません。

第2章

2

BASICの文法

第2章 BASICの文法

1. 文

文は、BASICが行う手続き（すなわちコンピュータに実行させるべき処理）を記述する最小単位です。

文は1つの命令からなります。そして命令は、命令の名前と、命令に与える情報（パラメータ）とからなります。パラメータには定数、変数、関数（「第2章 10.4 関数」参照）のほか、これらを組み合わせた式などを指定することができます。

<u>PRINT</u>	<u>1 2 3 4</u>
命令の名前	パラメータ
命令／文	

2. 行

行とは、プログラムモードで実行する場合において、メモリ中に格納（記憶）されるプログラムとして書かれる、行単位の文の集まりのことです。

行の先頭には行番号をつけます。1行（行番号も含めて）には255バイトの範囲で文の記述が可能です。

行は、基本的には1つの文からなりますが、複数の文を記述することも可能です。1行に複数の文を記述する場合、各文をコロン（:）で区切ります。これは複文（マルチステートメント）と呼ばれ、1行の範囲内で任意の数の複文が記述できます。

3. 行番号

行番号は1から65529までの整数で指定します。

行番号は行をメモリに格納したりディスクにセーブしたりする場合の格納順序を示すものです。

行番号をつけた行は、リターンキーの入力とともにメモリに格納されます。この方法で1行以上の行をメモリに格納したものがプログラムであり、RUNなどを使って実行させることができます。

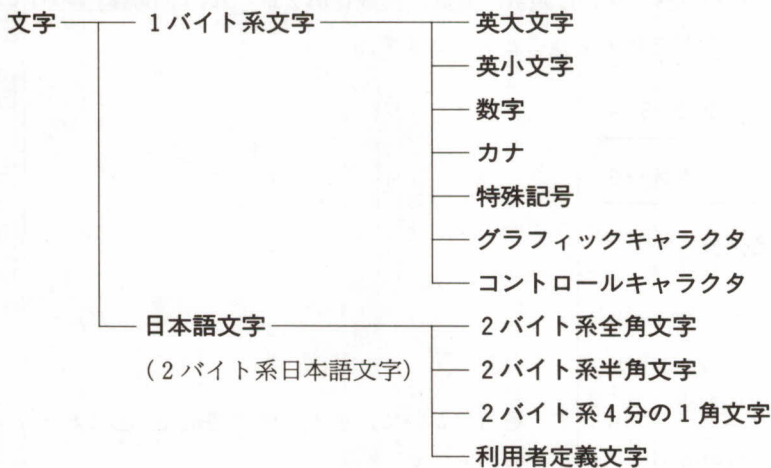
このように、行番号をともなった行を入力する方法が“プログラムモード”です。これに対し、行番号なしで命令を入力することによってただちに実行を行う方法が“ダイレクトモード”です（「第1章 3. BASICの動作モード」参照）。

プログラムの実行は行番号の小さいものから行われます。

行番号は分岐（プログラムの実行順序を変えること）の場合の分岐先として使用されるほか、LIST、DELETEなど〈行番号〉を対象とする命令のパラメータ（引数）として使われます。

4. BASICで使用できる文字と特殊記号

使用できる文字は次のとおりです。



1バイト系の英大文字と英小文字は、キーボード上から入力できる通常の英文字です。BASICでは、特定の場所（たとえば文字型定数の中…文字型定数に関しては「第2章 5.1 文字型定数」参照）を除き、英大文字と小文字とは同じ意味に解釈されます。

コントロールキャラクタは、スクリーンエディタに直接指示を与えるために使うもので、キーボード上の **CTRL** キーを押しながら英文字1文字のキーを押すことによって入力します。

日本語文字は、日本語処理システムによって入力する文字です。命令によっては使えないものもありますので注意してください。

これらの文字の詳細については、「付録B. コントロールコード表」、「付録C. キャラクタコード表」、あるいは BASIC ユーザーズマニュアル、日本語入力ガイドなどを参照してください。

■特殊記号の使い方

算術演算子（+、-、*、/）などの他にも特別な意味を持つ記号があります。ここでその意味をまとめて説明しておきます。なお、ここで説明しているのは1バイト系の特殊記号です。

2 バイト系日本語文字の中にも同様な記号がありますが、それらは文字型定数としてのみ使えるもので、特殊記号として使うことはできません。

(1) ピリオド (.)

最後に BASIC の処理の対象となった行 (現在行, 着目行などという) の行番号の値を示す記号で、命令中の 〈行番号〉 の代わりとして使用することができます。たとえば、新しい行を挿入した、エラーが発生したなどの行です。

例) LIST

(2) マイナス (-)

LIST, DELETE などで行の範囲を指定するとき、何行から何行までという場合に使います。算術演算子の負号と同じ記号です。

例) DELETE 100-200

(3) コロン (:)

マルチステートメントの区切りとして使います。

例) A=B+C:PRINT A

(4) コンマ (,)

パラメータが並ぶ場合その区切りとして使います。

例) INPUT A, B, C

COLOR 7, , , 0

(5) セミコロン (;)

PRINT などの出力文中でパラメータが並ぶ場合その区切りとして使います。

例) PRINT "A=" ; A

(6) アポストロフィ (')

REM (第3章中の REM の項参照) の代用として使います。

(7) クエッションマーク (?)

PRINT の代用として使います。

(8) アスタリスク (*)

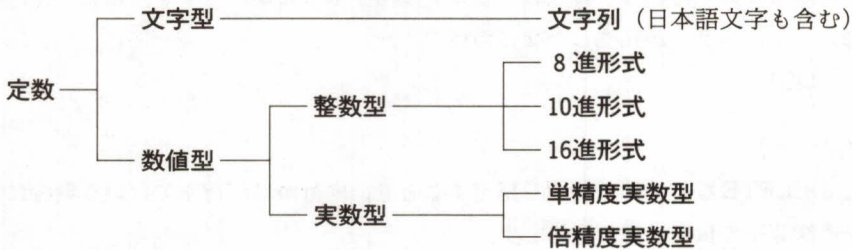
ラベル名の先頭につけます (「第2章 13. ラベル名」参照)。

(9) スペース

原則として文字型定数に含まれているスペース以外のスペースは無視されますが、命令の名前の直後には必ずスペースを入れなければなりません。

5. 定数

定数とは、それぞれ固有の値を持ったデータのことで、プログラム中などに直接表記されるものです。定数は、次のように分類することができます。定数の表記法は、型によってそれぞれ異なります。



5.1 文字型定数

文字型定数とは、255バイト以内の文字をつなぎ、その前後をダブルクォーテーション (") で囲んだ、文字列データのことです。文字列中には、1バイト系の英数字、カナ文字、特殊記号、グラフィックキャラクタならびに2バイト系日本語文字などを使用できます (混在も可能です)。

BASICの命令のなかには、2バイト系日本語文字が使用できないものもあります (各命令参照)。また、(") を文字型定数中に直接記述することはできません。(") を文字型定数としたい場合、CHR\$を用いて、例のようにCHR\$(&H22)とします。

なお、文字型定数を算術演算に使うことはできません。

例) "Good Morning"

"1 2 3 4 5 6 7 8 9"

"パーソナル コンピュータ"

"日本語 BASIC"

CHR\$(&H22) + "TEST" + CHR\$(&H22)

5.2 数値型定数

数値型定数とは、算術演算を行うことのできる数値データを直接表記したもので、大きく分けて、整数型と実数型とがあります。

5.3 整数型定数

整数型定数は、表記上の観点から、8進・10進・16進の3つの形式に分類されます。

■10進形式

-32768から+32767までのすべての整数です。負の整数の場合、数値の前には必ずマイナス

符号をつけなければなりません。正の数の前の符号は省略できます。また、同じ範囲内の実数の後ろに%をつけると、小数点以下は四捨五入され、整数型の定数とみなされます。

例) 32767
 -123
 +5
 31100.5% ←整数を表す。

■ 8進形式

頭に&Oまたは&をともなった、0から7までの数字の並び。&0~&177777の範囲内です。

例) &12345
 &07777

■ 16進形式

頭に&Hをともなった、0からFまでの並び。&H0~&HFFFFの範囲内です。

例) &H100
 &HCFFF

注意：8進形式または16進形式で入力された数値は、PRINTなどの出力文では10進形式で出力されます。10進以外の形式で出力するときは、それぞれOCT\$、HEX\$を使って文字列として出力してください。

5.4 実数型定数

実数型定数は、単精度実数型と倍精度実数型に分けられます。

5.5 単精度実数型定数

有効桁7桁の精度をもつ実数のデータです。出力のときは7桁目が四捨五入され、6桁以下で表示されます。単精度型実数の値の範囲は、 $-1.70141E+38$ ~ $1.70141E+38$ です。

単精度実数型の定数には、表記上の分類により、次の3つの形式があります。

- 7桁以下の実数
- 最後に!をともなった数
- Eを使った指数形式

例) 3525.68
 3.14!
 -7.09E-06

5.6 倍精度実数型定数

有効桁16桁の精度をもつ実数のデータです。出力のときは、16桁以下で表示されます。倍精度実数型の値の範囲は、 $-1.701411834604692D+38$ ～ $1.701411834604692D+38$ です。

倍精度実数型の定数には、表記上の分類により、次の3つの形式があります。

- 8桁以上の実数
- 最後に#をともなった数
- Dを使った指数形式

例) 1234567.890
56789.0#
-1.09432D-38

6. 変数

変数は、プログラム中で使うデータをしまっておくことができる入れ物のようなもので、1バイト系の英数字で名前(変数名)をつけます。変数は、演算、参照などに使うことができます。

なお、変数に値を格納する(入れる)前は、数値変数の値は0、文字変数はヌルストリング(空の文字列)であるものとみなされます。

6.1 変数名

変数名は長さ40文字以内の、1バイト系英数字とピリオド(.)の組み合わせで表されます。ただし、変数名の初めの1文字は英字でなくてはなりません。この規約にしたがって名前をつけられた変数は、すべて区別されます。

たとえば、次の2つの変数は異なった変数名として解釈されます。

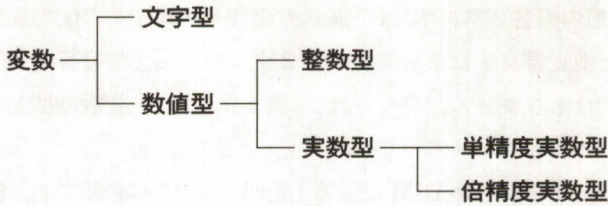
```
COUNTER.OF.TABLE.DATA.999888777666555.01  
COUNTER.OF.TABLE.DATA.999888777666555.02
```

変数名は予約語(「第2章 8. 予約語」, 「付録E. 予約語一覧」参照)であってはなりません。予約語を含んだものはかまいません。ただし、FNで始まる変数名は許されません(第3章中のDEF FNの項参照)。また英文字において大文字、小文字の区別はありません。

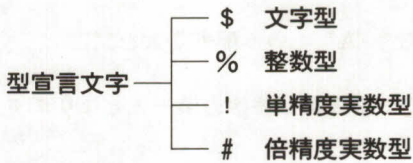
6.2 変数の型

変数にも、定数と同様、格納するデータに応じた型があります。

変数に値を格納するためには、代入文やINPUTなどを用いますが、いずれの場合にも変数の型は、格納されるデータの型と一致していなければなりません。



変数の型は、変数名の最後に型宣言文字をつけることによって区別します。型宣言文字の種類は次の4つです。



通常は、型宣言文字を省略すると、“!”がついているとみなされます(単精度実数型変数となる)。

例) $A!$
 $A\#$
 $A\%$
 $A\$$

これらは区別されるが、 $A!$ と A は同じ。

ただし、型宣言命令(DEFINT, DEFSNGなど)を用いて一括指定を行った場合、型宣言文字省略時の型は、型宣言命令によって宣言された型となります。たとえば、次の例では、 $A\sim C$ で始まる変数は、型宣言文字をつけないとみな整数型となります。

例) DEFINT A-C
 {
 $C=A+B$ ← A, B, C は、型宣言文字をつけなければ整数型
 $L=M*N$ ← $A\sim C$ 以外は、型宣言文字をつけなければ単精度実数型

7. 配列変数

配列変数は、たくさんのデータを変数に代入したり、参照する場合に使います。たとえば、データの数が100個あるとき、100個の別の変数を用意するのはたいへんです。このようなときに、 $A(0)\sim A(99)$ のように変数名の後にカッコつきの番号をつけた形で配列として扱おうと、100個のデータを1つの変数名で処理することができます。

カッコ内の番号は添字といい、配列のなかの何番目の要素であるかを表します。たとえば、 $A(0)$ は、 A という配列変数の0番目の要素という意味になります。添字には整数定数、または整数型の変数を使うことができます。

配列変数の型(すなわち配列変数内の各要素の型)は、通常の変数と同様、その配列変数名の最後(カッコの直前)につけられた型宣言文字によって決まります。ただし、型宣言命令で一括指定を行った場合は、この限りではありません。詳しくは、「第2章 6.2 変数の型」を参照してください。

プログラム内で配列を使うには、あらかじめDIMによる“配列の宣言”が必要です。配列の宣言とは、変数の名前と配列に入れる要素の個数を決めてやることです。

DIMでは次のように、その配列で使用可能な添字の最大値を指定することにより、要素の個数を決めます。

DIM A(5) 添字の最大値が5で、名前が“A”という配列変数を宣言

添字は、原則として0から始まりますので、実際の要素数は添字の値+1となります。この例では、5+1で、要素数は6個となります。

“添字が0から始まる”ということ、“添字の最小値(OPTION BASE)が0である”といい、BASICの起動時にはこの状態になっています。なお、OPTION BASEという命令を使うことにより、添字の最小値を1にすることもできます。

また、DIM中で配列変数を宣言する際、複数の添字をコンマで区切って指定すると、その配列変数は、添字の個数に応じた“次元”数をもつことになります。

たとえば、添字の最小値が0であるとして次のような配列を宣言した場合の、各配列の次元数および要素の合計数を示してみます。

DIM A(10) 1次元配列、要素数は11

DIM TA(2, 3) 2次元配列、要素数は $3 \times 4 = 12$

DIM NAME\$ (2, 5, 3) 3次元配列、要素数は $3 \times 6 \times 4 = 72$

2番目の例の場合、次のような2次元(縦方向と横方向の2次元)の配列がとられます。

TA(0, 0) TA(0, 1) TA(0, 2) TA(0, 3)

TA(1, 0) TA(1, 1) TA(1, 2) TA(1, 3)

TA(2, 0) TA(2, 1) TA(2, 2) TA(2, 3)

配列変数の次元および添字の最大値は、メモリ容量や変数の型によって制限がありますので注意してください(第3章中のDIMの項参照)。

なお、添字の最大値が10以下(添字の最小値が0でも1でもかまわない)のときは、DIMによる宣言を行わなくとも、配列変数を用いることができます。

8. 予約語

N₈₈-BASIC(86)は、命令(関数も含む)の名前、演算子の名前などを、処理上、特殊なものとして扱います。これは“予約語”と呼ばれ、文字どおり BASIC によってその使用方法を予約されているものです。したがって、予約語をそのまま変数名として使用したりすることはできません。

予約語を使用する場合には、BASIC がその語を予約語であるか否か認識できるようにするため、語の前、後ろ、またはその両方に、BASIC 文法上で許された、あるいは規定された特殊文字(スペース、ダブルクォーテーション(")、ナンバー記号(#)、コロン(:)など)を挿入しなくてはなりません。

BASIC の予約語は、「付録 E. 予約語一覧」に掲げてあります。

9. 型変換

数値データは、必要に応じて他の型に変換することができます。型の変換は次の規則にしたがって行われます。

なお、文字型と数値型の間では型変換を行うことはできませんが、数値表記の文字列に限り、数値との相互変換を関数によって行うことができます(第3章中の STR\$, VAL の項参照)。

- (1) ある型の数値データが、違った型の数値変数に代入された場合、数値は、その変数名によって宣言された型に変換されます。

```
例) 10 ABC % = 1.234
      20 PRINT ABC %
      RUN
      1
```

- (2) 精度の違う数値間の演算の場合、精度の高い方に変換されて、演算が行われます。たとえば、10#/3の場合、10#/3#として演算が行われます。

```
例) 10 A # = 10#/3
      20 B # = 10#/3#
      30 PRINT A #, B #
      RUN
      3.33333333333333 3.33333333333333
```

(3) 論理演算の場合、扱われる数値はすべて整数に変換され、結果は整数で与えられます。

```
例) 10 A=12.34
      20 B=NOT A
      30 PRINT B, A
      RUN
      -13          12.34
```

(4) 実数が整数に変換される場合は、小数点以下は四捨五入されます。このとき、整数型で扱える範囲を超えた場合はエラーが起こります。

<pre>例) 10 A%=34.4 20 B%=34.5 30 PRINT A%, B% RUN 34 35</pre>	<pre>10 A#=1.234E+07 20 B%=A# 30 PRINT B%, A# RUN Overflow in 20</pre>
--	--

(5) 倍精度実数型変数が単精度実数型変数に代入されたときは、変数の値は有効数字7桁に丸めたものとなります。単精度実数型変数の精度は7桁であり、もとの倍精度の数値との誤差の絶対値は、 $5.96E-8$ 以下となります。

```
例) 10 A#=1.23456789#
      20 B!=A#
      30 PRINT A#, B!
      RUN
      1.23456789  1.23457
```

倍精度実数型変数(あるいは倍精度実数型定数)と単精度実数型変数(あるいは単精度実数型定数)を混合して演算したり、単精度の値を倍精度実数型変数に代入したりすると、単精度値に変換する際、有効桁以降の桁に変換誤差が混入しますので注意してください。

例1) 精度の異なる数値による演算

- 好ましくない例 (演算結果に変換誤差が混入する)

```
A#=1.41421356#+0.12
```

- 改良例

```
A#=1.41421356#+0.12#
```

例2) 精度の高い変数に対する精度の低い値の代入

- 好ましくない例

A#=3.1415

- 改良例

A#=3.1415#

10. 式と演算

式とは、定数や変数を演算子（計算に使う特殊記号のこと）で結合した一般的な数式をはじめ、たんなる文字列や数値、変数だけのもの、または関数の総称です。たとえば次のものは、すべて式です。

例) 10+3/5
 A+B/C-D
 "BASIC"
 3.14
 A\$
 TAN(D)

演算は、演算子または関数を用いて行う式の操作のことで、次の5つに分類されます。

1. 算術演算
2. 関係演算
3. 論理演算
4. 関数
5. 文字列演算

以下に、それぞれの演算について説明します。

10.1 算術演算

算術演算子には次のようなものがあり、示された順序にもとづいて演算を行います。なお、算術式の中に文字定数や文字変数が入ってはいけません。

	算術演算子	演算内容	例
実行 順序 ↓	^	指数(べき乗)演算	X^Y
	-	負号	-X
	*, /	乗算, 実数の除算	X*Y, X/Y
	+, -	加算, 減算	X+Y, X-Y

演算の実行順序を変更する場合は、カッコを使用します。カッコの中の演算子は他の演算より先に実行されます。カッコ内においては通常の実行順序に従います。

	代数表記	BASICの表記
1)	$2X+Y$	$2*X+Y$
2)	$\frac{X}{Y}+2$	$X/Y+2$
3)	$\frac{(X+Y)}{2}$	$(X+Y)/2$
4)	X^2+2X+1	$X^2+2*X+1$
5)	X^{Y^2}	$X^(Y^2)$
6)	$(X^Y)^2$	X^Y^2
7)	$Y(-X)$	$Y*-X$

なお、演算の実行順序についての詳細は、「第2章 10.6 演算の優先順位」を参照してください。

■ 整数の除算と剰余の計算

整数の除算は \yen によって行われます。扱われる数値が実数の場合は、演算が実行される前に小数点以下が四捨五入されます。商は小数点以下が切り捨てられた整数となります。

例) $10 \yen 3 \rightarrow 3$ $(10 \div 3 = 3 \dots 1)$
 $23.75 \yen 5 \rightarrow 4$ $(24 \div 5 = 4 \dots 4)$

剰余の演算はMODによって行われます。扱われる数値が実数の場合は、演算が実行される前に小数点以下が四捨五入されます。結果は整数の割り算の余りです。

例) $13.3 \text{ MOD } 4 \rightarrow 1$ $(13 \div 4 = 3 \dots 1)$
 $25.68 \text{ MOD } 6.99 \rightarrow 5$ $(26 \div 7 = 3 \dots 5)$

■ 0での除算

式の実行時に0での除算が行われた場合は、“/0” (Division by zero) エラーを出力しますが、数値の型に応じてBASICが扱うことのできる最大の数を結果として処理を続行します。

また、0に対して負のべき乗演算を行った場合も同様になります。

例) PRINT 2#/0	PRINT 0^-1
/0	/0
1.701411834604692D+38	1.70141E+38

■ 桁あふれ (オーバーフロー)

代入や演算の結果がその変数の型内で表現することのできる範囲を超えた場合、桁あふれが発生します。

桁あふれが起こった場合、“OV” (Overflow)エラーを出力し、最大の数を結果として処理を続行します。

```
例) PRINT 3^300
      OV
      1.70141E+38
```

10.2 関係演算

関係演算子は2つの数値を比較するときに用います。結果は、真(-1)、偽(0)で得られ、条件判定文などプログラムの流れを変えるのに用いられます (第3章中の IF…THEN~ELSE の項参照)。

関係演算子	演算内容	例
=	等しい	X=Y
<>, ><	等しくない	X<>Y, X><Y
<	小さい	X<Y
>	大きい	X>Y
<=, =<	小さいか等しい	X<=Y, X=<Y
>=, =>	大きいか等しい	X>=Y, X=>Y

注意：=は代入文にも使うので注意すること。

IF…THEN~ELSE の中での使い方の例を次に示します。

```
例) IF X=0 THEN 1000
      IF A+B<>0 THEN X=X+1 : Y=Y+1
```

10.3 論理演算

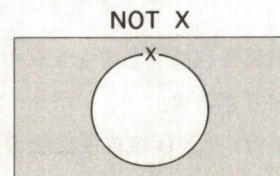
論理演算子は、ビット操作やブール演算(2進演算)を行ったり、複数の関係演算式を結合して複合条件を判定したりするのに用いられます。

論理演算子は、扱う数値を-32768から+32767までの2の補数表示の整数に変換してから、演算を行います。変換時にこの範囲外となれば“OV” (Overflow)エラーとなります。

各論理演算の内容は次のとおりです。論理演算は、扱う整数値の2進表記(16ビット分)に対してビットごとに行われます。次に示すのは、単一のビットX, Yについての演算結果を表しています。

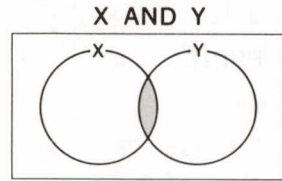
NOT : not (否定)

X	NOT X
1	0
0	1



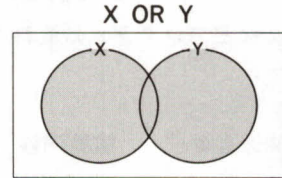
AND : and (論理積)

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0



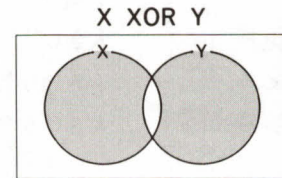
OR : or (論理和)

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0



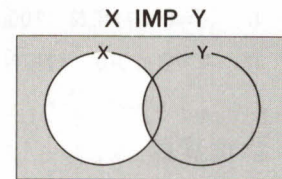
XOR : exclusive or (排他的論理和)

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0



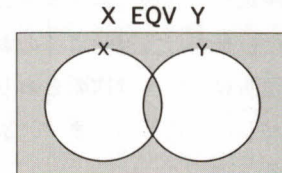
IMP : implication (包含)

X	Y	X IMP Y
1	1	1
1	0	0
0	1	1
0	0	1



EQV = equivalence (同値)

X	Y	X EQV Y
1	1	1
1	0	0
0	1	0
0	0	1



論理演算を利用することにより、バイトデータをあるビットパターンに照らし合わせて調べることができます。

たとえば、AND 演算子は機器の入出力ポートのステータスバイトの必要なビット以外のす

すべてのビットをマスクするのに使えます。また OR 演算子はある2進数を作るために2つのビットパターンを混合するのに使うことができます。

以下に、論理演算子がどのように働くかの例を示します。演算がビットごとに行われているようすに注意してください。

-1 OR 0 → -1	-1 = (1111 1111 1111 1111) ₂
	0 = (0000 0000 0000 0000) ₂
-1 OR 0 = (1111 1111 1111 1111) ₂	
48 AND 24 → 16	48 = (0000 0000 0011 0000) ₂
	24 = (0000 0000 0001 1000) ₂
48 AND 24 = (0000 0000 0001 0000) ₂	
15 XOR 60 → 51	15 = (0000 0000 0000 1111) ₂
	60 = (0000 0000 0011 1100) ₂
15 XOR 60 = (0000 0000 0011 0011) ₂	
17 EQV 12 → -30	17 = (0000 0000 0001 0001) ₂
	12 = (0000 0000 0000 1100) ₂
17 EQV 12 = (1111 1111 1110 0010) ₂	
28 IMP 9 → -21	28 = (0000 0000 0001 1100) ₂
	9 = (0000 0000 0000 1001) ₂
28 IMP 9 = (1111 1111 1110 1011) ₂	
NOT 23 → -24	23 = (0000 0000 0001 0111) ₂
NOT 23 = (1111 1111 1110 1000) ₂	

論理演算において、もし、0 (偽) と -1 (真) しか与えられなかったなら、扱う整数値の16ビット全部について同一の演算が行われることになり、演算の結果は0か-1のどちらかとなります(前例の -1 OR 0 を参照)。したがって、論理演算子で2つ以上の関係演算を結ぶようにして使用することにより、複合条件によってプログラムの流れを変えるのに用いることができます。

例) IF X < 0 OR 99 < X THEN 1000

X が負または、99 より大きければ、行番号1000へ飛ぶ。

IF 0 < X AND X < 100 THEN X = 0

X が正でかつ、100 より小さければ、X に0を代入する。

IF NOT (A = 0) THEN 20

A が0でなければ、行番号20へ飛ぶ。

10.4 関数

関数とは、指定されたある値(これを関数の引数(ひきすう)という)に対して、ある決まった演算を行い、その演算の結果を得ることができるものです(ただし、関数によっては、引数を必要としないものもあります)。

関数は、演算子を使用する他の演算とは異なり、実行後にそれ自身が値をもつ、一種の命令として扱われます。したがって、ふつうの命令のように機能別に決められた名前をもっています。

N₈₈-BASIC (86) には、SIN (正弦)、SQR (平方根) などの数値関数や CHR\$, LEFT\$ などの文字列関数といった“組み込み関数”が用意されています。これらの関数については、第3章で通常の命令とともに詳しく説明されています。

また、“ユーザー定義関数”としてユーザーが自由に定義できる関数機能もあります。これは第3章 DEF FN の項で説明します。

なお、初等関数 (SIN 関数などの数学関数) では、引数が整数や単精度のときは単精度の結果が得られますが、引数が倍精度のときは倍精度の結果を得ること (DISK モードのみ) ができます。

```
例) A=SIN(3.1416)+COS(3.1416)
     PRINT 2, 2*2, SQR(2)
```

10.5 文字列の演算

BASIC では、文字列に対して、演算を行うことができます。

■文字列の連結

文字列は、演算子“+”によって連結することができます。

```
例) 10 A$="SUPER":B$="PERSONAL":C$="COMPUTER"
     20 D$=A$+"-"+B$+"-"+C$
     30 PRINT D$
     RUN
     SUPER-PERSONAL-COMPUTER
```

■文字列の比較

文字も、数値の比較に用いられるものとまったく同じ関係演算子を用いて比較することができます。

=, <, >, <>, ><, <=, =<, >=, =>

文字列の場合それぞれの文字列の最初から1文字ずつ、文字の比較を行います。もし相互にまったく同じ文字列の場合は、その2つの文字列は等しくなりますが、1個所でも違った場合は、その文字のキャラクタコード(「付録C. キャラクタコード表」参照)の大きい方の文字列

が大きくなります。文字列の片方が短くて比較が途中で終わった場合は、短い文字列の方が小さくなります。

文字列の比較においては、空白なども意味をもちますから注意してください。

次の例は、すべて真 (-1) となります。

例) "AA" < "AB"

"BASIC" = "BASIC"

"X&" > "X #"

"PEN " > "PEN"

"cm" > "CM"

"DESK" < "DESKS"

文字列の比較は、文字列の内容を調べたり、文字をアルファベット順に並べたり(ソート)するのに使うことができます。

10.6 演算の優先順位

種々の演算には優先順位があり、実行時において次の番号順に処理されます。

- | | |
|--------------------|----------------------|
| 1 カッコで囲まれた式 | 9 関係演算子 (<, >, = など) |
| 2 関数 | 10 NOT |
| 3 ^ (指数) | 11 AND |
| 4 - (負号) | 12 OR |
| 5 *, / (乗算, 実数の除算) | 13 XOR |
| 6 ¥ (整数の除算) | 14 IMP |
| 7 MOD (整数の剰余) | 15 EQV |
| 8 +, - (加算, 減算) | |

11. ファイル

11.1 ファイルとは

ファイルとは、意味を持つ情報の集まりです。ディスクに保存したプログラムやデータは、ファイルとして扱われます。また他の周辺機器との入出力も、まとまったデータの入出力という意味で、ファイルというとらえかたで扱うことができます。

N₈₈-BASIC(86)では、このとらえかたにもとづき、ディスク装置をはじめとして、その他の周辺機器をも一括してファイルとしてみなすことにより、統一された入出力操作が可能となっています。

ファイルは“ファイルディスクリプタ”という一定の規則にしたがってつけられた名前によって区別されます。

11.2 ファイルディスクリプタ

ファイルディスクリプタは、周辺機器をも含むファイルを区別する名前で、次のような構成の文字列で表されます。

"[<デバイス名>:]<ファイル名>"

ファイルディスクリプタは文字列ですから、必ず、文字の並びをダブルクォーテーション(")で囲んだ文字定数、または文字列を代入した文字変数の、いずれかでなければなりません。

■デバイス名

<デバイス名>はディスク装置や他の周辺機器(まとめて入出力機器という)の名称を表すもので、アルファベット4文字、もしくはアルファベット3文字と数字1文字に、":"(コロン)をつけたものを原則としています。ただし、ディスク装置については、ドライブ番号を示す数字に":"をつけた形が用いられます。

N₈₆-BASIC(86)で定義されているデバイス名は、次のとおりです。

デバイス名	入出力機器名	入力	出力
KYBD:	キーボード	○	×
SCRN:	スクリーン	×	○
LPT:	プリンタ	×	○
1:	ディスクドライブ 1	○	○
2:	ディスクドライブ 2	○	○
}	}	}	}
30:	ディスクドライブ 30	○	○
COM1:	RS-232C 第1回線	○	○
COM2:	RS-232C 第2回線	○	○
COM3:	RS-232C 第3回線	○	○

注意:

- KYBD: および SCRN: は、DISK モード BASIC でのみ指定可能です。
- 1: ~30: は、ディスク装置の実装数分だけ指定可能です。
- COM2:, COM3: は、専用の RS-232C インタフェースボードを実装している場合のみ指定可能です。

〈デバイス名〉：は次の場合に限り、省略することができます。

- DISK モード BASIC の場合のフロッピーディスク 1 (1:)

なお、RS-232C 回線 (COMn:) をファイルとして指定する場合は、OPEN により、データビット長やパリティチェックの有無などの、種々のモードを設定しなければなりません。詳しくは第3章中の OPEN の項を参照してください。

デバイス名についての詳細は、BASIC ユーザーズマニュアルを参照してください。

■ファイル名

〈ファイル名〉とは、プログラムファイルやデータファイルにつける名前です。これはユーザーが指定します。ファイル名を必要とするのは、ディスク装置との入出力を行う場合のみです。

ファイル名は次のような書式をとります。

〈ファイル名〉 [.] [〈拡張子〉]

最初の〈ファイル名〉は6文字、ピリオドに続く〈拡張子〉は、もしあれば3文字までの文字列より構成されます。それぞれの文字数がそれ以下の場合には空いた部分に空白がうめられます。一般的には、〈ファイル名〉がファイルの名前を、〈拡張子〉がファイルの性質を表すようにしますが、とくにこれにこだわることなく自由に名前をつけてかまいません。

ふつうファイル名と言う場合は、〈拡張子〉まで含んだものを意味します。

ファイル名には1バイト系の英数カナ文字を使用できますが、コロン (:) およびキャラクタコード 0 と 255 で表される文字は使えません。

最初の〈ファイル名〉が6文字を超えて指定された場合、先頭から6文字目までの6文字が〈ファイル名〉として、続く9文字目までの3文字が〈拡張子〉としてみなされます。10文字目以降の文字は無視されます。

11.3 データファイルとファイル番号

プログラム中で、データファイルを扱うときには、まず、OPEN によってファイルの使用を宣言します。

OPEN では、扱うファイルを〈ファイルディスクリプタ〉で指定して、〈ファイル番号〉を割り当てます。ファイル番号は入出力のために必要なメモリ領域(バッファ)に対してつける固有の番号であり、以後、OPEN されたファイルへの入出力はこのファイル番号を指示することによって行います。

N₈₈-BASIC (86) が起動したときに最初に指定するファイル数(「第1章 3. BASICの動作モード」参照)は、同時にOPENすることのできるファイルの数を意味しますが、このときに指定した数の範囲内でファイル番号を指定することができます。

データファイルの種類やその使い方などに関しては、第3章のファイル関連命令の各項、およびBASICユーザーズマニュアルを参照してください。

12. 割り込み

コンピュータの頭脳であるCPU (Central Processing Unit) は、一般に同時に2つ以上の処理をすることができません。したがって、1つ仕事を処理している間は他で何が起ころうと、CPUはその処理に移ることはできませんし、それを知ることさえできないのです。このことは、1つの決まった一連の流れを頭から処理するようなプログラムの場合、さほど問題にはなりません。1つのプログラムの処理中に、即実行しなければならないような特別なことがら(「事象」と呼ぶ)が起きたときに、その処理を先に実行させるようなプログラムを必要とする場合があります。

“割り込み”とは、このような処理を実現するための手段です。たとえば、CPUがある処理を実行しているとき、ファンクションキーが押されたなどの特別な事象が発生したとします。このような場合、専用のハードウェアはCPUにその事象が起きたことを、ハードウェア的手段で伝えます。そこでCPUは、現在実行中の処理を一時停止し、特別な事象の処理を優先的に実行し、その処理の終了後、元の処理に復帰するのです。

以上のような流れを、“割り込み”と呼ぶわけですが、PC-9800シリーズでは、多くの周辺装置でこの割り込みを使える(すなわち、割り込みの事象を発生させることができる)ようになっています。N₈₈-BASIC(86)もこれらの割り込みをサポートしています。

次に、N₈₈-BASICがサポートしている割り込み機能を示します。

- | | |
|----------------|-------------------------|
| (1) STOP キー | (ON STOP GOSUB 参照) |
| (2) HELP キー | (ON HELP GOSUB 参照) |
| (3) リアルタイム・タイマ | (ON TIME\$ GOSUB 参照) |
| (4) ファンクションキー | (ON KEY GOSUB 参照) |
| (5) RS-232C 回線 | (ON COM GOSUB 参照) |
| (6) ライトペン | (ON PEN GOSUB 参照) |
| (7) サウンド | (ON PLAY GOSUB 参照) |
| (8) 電話機 | (CMD ON LINE GOSUB 参照) |
| (9) 通信エラー | (CMD ON ERROR GOSUB 参照) |

13. ラベル名

N₈₈-BASIC(86)では、プログラムの分岐先やプログラムの編集時などに利用する行番号の代わりとして、“ラベル名”を用いることができます。

あるルーチンの始まりなどに意味のあるラベル名をつけておけば、いちいち行番号を覚えておかなくともよく、また RENUM で行番号のつけ替えを行ってもそのたびに新しい行番号を確かめる必要もなくなるため、プログラムの作成を非常に楽に行うことができます。後でプログラムの修正などを行うときにもたいへん有利です。

まず、ラベル名を使わないで、ふつうに行番号を分岐先として用いたプログラムを例として示します。

```

10 INPUT A
20 IF A<0 THEN 80
30 IF A>0 THEN 60
40 PRINT "zero"
50 GOTO 90
60 PRINT "plus"
70 GOTO 90
80 PRINT "minus"
90 END

```

これは、10行で入力された値が正か負か0かを調べるプログラムです。ここで処理の流れを変える命令が20, 30, 50, 70行で使われていて、その分岐先の指定はすべて行番号になっています。

これを、ラベル名を使って書き換えると次のようになります。

```

10 INPUT A
20 IF A<0 THEN *MINUS
30 IF A>0 THEN *PLUS
40 PRINT "zero"
50 GOTO *EXIT
60 *PLUS : PRINT "plus"
70 GOTO *EXIT
80 *MINUS : PRINT "minus"
90 *EXIT : END

```

プログラムが見やすくなり、処理の内容もよくわかるようになりました。

このように、ラベル名とは、分岐先の目印として自由につけることのできるものです。ラベル名の使用については、次の注意を守らなければなりません。

- (1) ラベル名の頭には必ずアスタリスク(*)をつけなければなりません。
- (2) 先頭のアスタリスクを除いて、ラベル名は必ず1バイト系の英文字で始まらなければなりません。
- (3) ラベル名に使用できる文字は、先頭のアスタリスクを除いて1バイト系英文字と数字とピリオド(.)であり、大文字と小文字の区別はありません。
- (4) 予約語をラベルとして使用することはできません。ただし、中に含む場合はかまいません。
- (5) ラベル名の長さは、プログラムの1行の範囲(255バイト以内)によってのみ制限を受けます。
- (6) 参照されるラベル名(呼ばれる方のラベル名)は、必ず行の最初になければなりません。
- (7) 1行中でラベル名の後に続けて命令を記述し、マルチステートメントとするときは、コロン(:)またはスペースによって区切ります。

以上のような制限を無視すると“Syntax error”となります。

その他参照されるラベル名に同じものがあつた場合、二重定義されているという意味の“Duplicate label”エラーが生じます。これらのエラーの検出は、“RUN”したとき、プログラムの実行に先だつて行われますから、ラベル名のエラーがあるとプログラムを1行も実行することができません。

また、この場合のエラーメッセージはエラー箇所を示す〈行番号〉はともないません。

ラベル名は、プログラム中で分岐の目印として使用できるほか、LIST、DELETEなど〈行番号〉を対象とする命令のパラメータにも使用することが可能です。

例) LIST *START-*LAST
DELETE *LOOP1-*EXIT1

14. エラーメッセージ

N₈₈-BASIC(86)は、プログラムの実行を中断させなければならないようなエラーを実行時に検出したとき、エラーメッセージを画面に出力し、コマンドレベルにもどります。

ダイレクトモードの場合、エラーメッセージは次のような形式で出力されます。

XXXX...

プログラムモードの場合は次のよう出力されます。

```
XXXX . . . in yyyy
```

XXXX . . . はエラーメッセージで、yyyy はエラーが検出された行番号です。この行はあくまでも、BASIC がエラーを検出した行であり、真のエラーの原因は他の行にある場合がありますのでご注意ください。

エラーメッセージの内容とその対策については、「付録A. エラーメッセージとその対策」を参照してください。

命令リファレンス

第3章 命令リファレンス

この章の見方

この章では、すべての命令（関数も含む）について書式、機能などを解説しています。各命令の解説は次の構成で行われています。

1	→	KACNV\$ (DISK モード) ←2	3	10	
			↓	↓	
			関 数	□	
4	→	機 能	2 バイト系全角文字を、対応する 1 バイト系の英数カナ文字に変換します。		
5	→	書 式	KACNV\$(<文字列>)		
6	→	文 例	A\$=KACNV\$(B\$) PRINT KACNV\$("アイウABC")		
7	→	〈文字列〉中の 2 バイト系全角文字を 1 バイト系英数カナ文字に変換します。なお、このとき 2 バイト系全角文字列中の KI/KO コードはすべて取り除かれます。 なお、〈文字列〉中に、対応する 1 バイト系英数カナ文字のない 2 バイト系全角文字が含まれていると、"Illegal function call" エラーとなります。			
8	→	注意 ：〈文字列〉中の 2 バイト系半角文字は正しく変換されません。			
9	→	参照 ：AKCNV\$, サンプルプログラム 35			

1. 命令の名前。
2. DISK モード BASIC でのみ有効な命令であることを示します。
3. 関数表示。該当命令が関数である場合のみ、“関数”と明示してあります。
4. 機能。命令の機能を簡単に示します。
5. 書式。命令の記述の仕方を示します。実際の入力時には次のような決まりに従ってください。
 - アルファベットの大文字で示された項目は、そのまま 1 バイト系のアルファベットを入力します。入力する場合は、小文字でも大文字でもかまいません。BASIC は入力された命令（行）を、自動的にすべて大文字に変換します。ただし、ダブルクォーテーション (") で囲まれた文字列や、DATA 行中に記述された文字列などは、大文字と小文字の区別がなされますから、必要に応じて使い分けてください。
 - カギカッコ (< >) で囲まれた項目は、ユーザーが指定します。
 - 角カッコ ([]) で囲まれた項目は、オプションであり省略することができます。省略した場合、デフォルト値 (BASIC によって設定される値) または以前に指定した値が適用されます。

コンマ (,) で区切られる複数のパラメータがあり、しかも省略可能なパラメータがある場合、次の例のような書式で示します。

```
CLEAR 〔〈ダミーパラメータ〉〕〔, 〈メモリの上限〉〕〔, 〈スタックの大きさ〉〕〔, 〈配  
列データ領域の大きさ〉〕
```

この場合、角カッコで囲まれたパラメータを省略することが可能であることを示しています。ここで、あるパラメータ以後をすべて省略する場合は、コンマも含めて省略できます。しかし、中途のパラメータを省略したうえで後続のパラメータを指定する場合は、それ以前のコンマはすべて指定する必要があります。

たとえば、上記の例で〈スタックの大きさ〉のみ指定する場合は、次のようにします。

```
CLEAR ,, 〈スタックの大きさ〉
```

なお、次のようにパラメータの後にコンマまたはセミコロン (;) を指定するような場合は、コンマならびにセミコロンも含めて省略します。

```
INPUT 〔〈プロンプト文〉 | ; | 〕 〈変数〉〔, 〈変数〉 …〕  
      | , |
```

ただし、この約束と異なっている場合もあります。その場合には、注意書きにてそのことを説明します。

- 上記のカギカッコ、角カッコ以外の記号で丸カッコ (), コンマ (,), セミコロン (;), マイナス記号 (-), 等号 (=) などの記号は示された位置に正しく入力します。
- 省略記号 "..." の続く項目は、1 行の許す長さ (255文字) の内で任意の回数繰り返すことができます。たとえば、

```
DATA  〈定数〉〔, 〈定数〉 …〕
```

という書式ならば、次のような記述が可能です。

```
DATA  0, 10, 15, 20
```

- 座標指定の所で (Wx, Wy) と記されているのはワールド座標を、(Sx, Sy) はスクリーン座標を表しています。また STEP (x, y) という記法は相対座標による指定ができることを意味しています。その他たんに (X, Y) と記されているのはキャラクタ座標を表しています。
- 書式の中で 〈行番号〉 またはそれに類する行番号の指定をさしているものは、行番号の他にラベル名も含んでいると解釈してください。

6. 文例. 実際の入力の方の見本として簡単な例を示します。
7. 解説. 命令の使用法や詳しい機能を説明します。
8. 注意. 命令を使ううえで、特に注意すべきことをまとめています。
9. 参照. 関連の深い他の命令と、第4章中のサンプルプログラムの番号を示します。
10. 拡張機能制御命令の種別. 拡張機能を使用する際に用いる命令であることをマークをつけて表します。マークの種類と意味は、次のとおりです。

S サウンド制御機能関連の命令であることを表します。

この命令を使用するためには、サウンドボードが実装されていることが必要です。

P 電話制御機能関連の命令であることを表します。

この命令を使用するためには、モデム-NCU 内蔵電話機またはモデムが装備されていることが必要です。

なお、拡張機能を使用する際は、機器の準備とともに、ユーティリティ “switch.n88” を用いて該当する拡張機能を使用することをあらかじめ設定しておく必要があります。

DISK モードについて

N₈₈-日本語 BASIC(86)システムディスクから BASIC を起動した場合を “DISK モード” と呼び、ROM に搭載されている N₈₈-BASIC(86)を起動した場合を “ROM モード” と呼びます (「第1章 1. ROM モード BASIC と DISK モード BASIC」参照)。

大部分の命令は両 BASIC で使用可能ですが、ディスクファイルを扱う命令をはじめとする一部の命令は DISK モード BASIC でのみ使用することができます。この章では、この DISK モードでのみ使える命令には、命令の名前の後に、

(DISK モード)

と明記し、区別できるようにしてあります。

なお、算術関数の中で、初等関数 (SIN, COS, TAN などの数学関数) は引数として倍精度実数型の数値を与えると倍精度実数型の値を、その他の数値を与えると単精度実数型の値を得ることができますが、倍精度関数機能は DISK モード BASIC でしか使うことができませんので注意してください。

ABS

関 数

機 能	絶対値を得ます。
書 式	ABS(<数式>)
文 例	B=ABS(-2) PRINT ABS(-1.0000000000000001)

<数式> に指定した値の絶対値を得ます。

<数式>に倍精度実数が含まれる場合、得られる値は倍精度となりますが、他の場合には単精度となります。

参照：サンプルプログラム 8

AKCNV\$ (DISK モード)

関 数

機 能	1 バイト系の英数カナ文字を、対応する 2 バイト系全角文字に変換します。
書 式	AKCNV\$(<文字列>)
文 例	A\$=AKCNV\$(B\$) PRINT AKCNV\$("イロハ日本語 ABC 英語")

<文字列> 中の 1 バイト系文字を 2 バイト系全角文字に変換します。<文字列> 中の 2 バイト系日本語文字に対しては変換操作を行いません。

参照：KACNV\$, サンプルプログラム 35

ASC

関 数

機 能	文字のキャラクタコードを得ます。
書 式	ASC(<文字列>)
文 例	A=ASC("NEW") PRINT ASC(A\$)

<文字列> 中の最初の文字のキャラクタコードを、10 進表記で得ます。

参照：CHR\$, サンプルプログラム 23

ATN

関数

機能 逆正接(アークタンジェント)を得ます。

書式 ATN(<数式>)

文例 ANGLE=ATN(Y/X)

PRINT ATN(-3.14159/2)

<数式>の値に対する逆正接値を得ます。得られる値はラジアンで、 $-\pi/2$ から $\pi/2$ までの範囲です。

<数式>に倍精度実数が含まれる場合、得られる値は倍精度となりますが、他の場合には単精度となります。

参照: TAN

ATTR\$(DISKモード)

関数

機能 ファイル、ドライブの属性を得ます。

書式 ATTR\$(
 <ドライブ番号>
 #<ファイル番号>
 <ファイルディスクリプタ>
)

文例 PRINT ATTR\$(1)

A\$=ATTR\$("1:TOOL.BAS")

指定したドライブ、オープンされているファイルバッファ、ディスク上のファイルの現在の属性文字を得ます。属性文字は以下の意味を持ちます。

"□□□": 属性は解除されており、通常の読み出し、書き込みが可能な状態です。

"R□□": 書き込みの際に、リードアフターライト(書き込んだ内容とメモリ上の内容の比較チェック)を行います。

"□□P": 書き込みが禁止されています。

"□E□": PオプションつきでSAVEされたファイルです。

参照: SAVE, SET

AUTO

機能	行番号を自動的に発生します。
書式	AUTO [<行番号>] [, <増分>]
文例	AUTO 100, 5

AUTO を実行すると<行番号>で指定した行から、以後、リターンキーの入力ごとに<増分>ずつ増加した行番号を自動的に発生します。

<行番号>、<増分>を省略した場合には、ともに10を採用します。コンマがあつて<増分>を省略した場合には、直前に実行したAUTOコマンドの増分が用いられます。

AUTO を中止してコマンドレベルにもどるには **CTRL**+**C** または、**STOP** キーを押します。このとき、最後に発生した行番号の行は、メモリ中にプログラムとして格納されません。

プログラム中にすでに存在する行と同じ行番号を発生させた場合には、行番号の直後にアスタリスク(*)が表示され、注意を促します。このとき、文字を入力してリターンキーを押すと、その行は新しい内容に入れ代わります。また、何も入力しないでリターンキーを押した場合には、その行は削除されます。

BEEP

機能	内蔵スピーカを鳴らしたり、止めたりします。
書式	BEEP [<スイッチ>]
文例	BEEP

<スイッチ>の値が1の場合はON(鳴りっぱなし)、0の場合はOFF(鳴りやむ)となります。<スイッチ>を省略した場合には、PRINT CHR\$(7)を実行したのと同様に、一定時間スピーカを鳴らします。

参照：サンプルプログラム 31

BLOAD (DISK モード)

機能	機械語ファイルをメモリ上にロードします。
書式	BLOAD <ファイルディスクリプタ> [, <ロードアドレス>] [, R]
文例	BLOAD "SUB1", R BLOAD "SUB2", &H100

〈ファイルディスクリプタ〉には、メモリ上にロードしたい、ディスク上または RS-232C 回線の機械語ファイル(バイナリイメージのファイル。プログラムでもデータでもかまわない)を指定します。

〈ロードアドレス〉を指定した場合には、直前に実行された DEF SEG で指定されたセグメントベースに、〈ロードアドレス〉で与えられたアドレス(相対アドレス)を加えた番地からロードし始めます。

〈ロードアドレス〉を省略した場合には、直前に実行された DEF SEG で指定されたセグメントベースに、機械語ファイルをセーブする際に BSAVE で指定した開始アドレス(相対アドレス)を加えた番地からロードが行われます。

R オプションを指定すると、機械語ファイルをロード後、直ちにその先頭番地からプログラムとして実行を開始します。したがってこの場合、指定する機械語ファイルは、実行可能なプログラムでなくてはなりません。

R オプションを指定し、さらに〈ロードアドレス〉をつけてロードする場合には、セーブされた際と異なる番地にロードされても実行可能な性質をもった(リロケータブルな)プログラムを、機械語ファイルとして指定しなければなりません。

なお、R オプションの指定時には、すでに開かれているデータファイルはその状態を保持します。

参照：BSAVE, CLEAR, DEF SEG, LOAD

BSAVE (DISK モード)

機能 メモリ上の指定範囲の内容を、ディスク上あるいは RS-232C 回線上に機械語ファイルとしてセーブします。

書式 BSAVE 〈ファイルディスクリプタ〉, 〈開始アドレス〉, 〈長さ〉

文例 BSAVE "SUB1", &H100, &H2FF

〈ファイルディスクリプタ〉には、メモリ上の内容をセーブする際における、ディスク上あるいは RS-232C 回線上のファイル名を指定します。

〈開始アドレス〉には、セーブを開始する番地を相対アドレスで指定します。

〈長さ〉には、セーブするメモリ上の内容のバイト数を指定します。

BSAVE は、直前に実行された DEF SEG で指定されたセグメントベースに、〈開始アドレス〉の値を加えた番地以降の連続する〈長さ〉バイトの内容を、機械語ファイルとしてセーブします。

参照：BLOAD, CLEAR, DEF SEG, SAVE

CALL

機能	メモリ上に用意された機械語サブルーチン呼び出し、実行します。
書式	CALL <変数名> [(<引数> [, <引数> ...])]
文例	CALL MUSBR(X, Y) CALL M(ARG1\$, ARG2\$, RESULT\$)

呼び出したい機械語サブルーチンは、あらかじめメモリ上に用意しておかなくてはなりません。このためには、POKE や BLOAD を用いることができます。

<変数名>には、呼び出したい機械語サブルーチンの実行開始アドレス(相対アドレス)が代入された、変数の名前を指定します。ここで用いる変数には配列変数を用いることはできません。

<引数>には、機械語サブルーチンに渡す変数を指定します。引数としてはすべての型の変数を指定することができますが、定数や式の指定はできません。

CALL を実行すると、直前に実行された DEF SEG によって指定されたセグメントベースに、<変数名>で指定した変数に代入されている相対アドレス値を加えた番地に実行が移ります。CALL によって呼び出されたサブルーチンは、機械語の IRET 命令により BASIC に制御を戻すことができます。

<引数> の受け渡し方法については、BASIC ユーザーズマニュアルを参照してください。

参照：BLOAD, BSAVE, CLEAR, DEF SEG, POKE

CDBL

関 数

機能	整数値、単精度実数値を、倍精度実数値に変換します。
書式	CDBL(<数式>)
文例	A#=CDBL(B!/2) A#=CDBL(256%)

<数式>の値を倍精度実数値に変換した値を得ます。ただし、型変換が行われるだけで有効桁数の変化がありませんから、得られた値の精度は、変換する前の型と同じ(整数型なら整数部のみ、単精度実数型なら有効数字6桁)になります。

参照：CINT, CSNG

CHAIN (DISK モード)

機能	メモリ上のプログラムからディスク上のプログラムに実行を移します。
書式	CHAIN [MERGE] <ファイルディスクリプタ> [, <行番号>] [, ALL] [, DELETE <範囲>]
文例	CHAIN MERGE "TEST.BAS", 1000, ALL, DELETE 500-600

CHAIN を実行すると、メモリ上にあるプログラムで使用中のファイルは、そのままの状態ロードされるプログラムに引き継がれます。また、OPTION BASE もロードされるプログラムに引き継がれます。

(1) MERGE オプションを省略した場合

メモリ上のプログラムは消去され、ディスクから <ファイルディスクリプタ> で指定したプログラムがロードされ、実行されます。

<行番号>には、ロードされたプログラムの開始行番号を指定します。省略した場合は、プログラムの最初から実行されます。

ALL オプションは、ロードされるプログラムに変数、配列を引き渡すのに使います。これを指定した場合すべての変数、配列が引き渡され、省略した場合いっさい引き渡されません。必要な変数、配列だけを引き渡したいという場合は COMMON を使います (COMMON 参照)。

DELETE オプションは、この場合意味をもちません。

(2) MERGE オプションを指定した場合

メモリ上のプログラムと、ディスクからロードされたプログラムとが、合成(マージ)されてひとつのプログラムとなった後、実行されます。

<ファイルディスクリプタ>には、ディスクからロードするプログラムを指定します。この場合、ディスクからロードするプログラムは、必ずアスキーセーブされたものでなければなりません。

<行番号>には、合成後のプログラムの開始行番号を指定します。行番号を省略した場合には、合成後のプログラムの最初から実行されます。

ALL オプションの扱いは、(1)と同様です。

DELETE オプションおよび<範囲>を指定すると、メモリ上のプログラムの指定範囲が削除された後、ディスクのプログラムの合成が行われます。<範囲>は、"200-300"のように、行番号をマイナス記号でつないで指定します。DELETE オプションで指定する<範囲>の行番号は RENUM を実行すると書き換えられます。また、<範囲>の指定には、ラベル名を使うことができます。

注意・実行開始行を指定する〈行番号〉は、RENUM を実行しても書き換えられません。また、〈行番号〉としてラベル名を使うことはできません。

- DEFINT, DEFSNG, DEFDBL, DEFSTR などの宣言文は、MERGE オプションを指定した場合、合成後のプログラムに対しても有効となりますが、指定がない場合、ロードされたプログラムには何の効果も与えません。
- CHAIN を実行すると、DEF FN, あるいは ON COM GOSUB, ON ERROR GOTO などの割り込み制御は無効となります。
- MERGE オプションを指定した CHAIN を、FOR~NEXT ループ内で実行することはできません。

参照：COMMON, サンプルプログラム 1, 2

CHR\$

関 数

機 能 指定したキャラクタコードを持つ文字を得ます。

書 式 CHR\$(〈数式〉)

文 例 A\$=CHR\$(65)

PRINT CHR\$(&H4E)

〈数式〉の値のキャラクタコードを持つ文字を得ます。値が 0~255 の範囲にない場合は、“Illegal function call” エラーになります。

参照：ASC, サンプルプログラム 23

CINT

関 数

機 能 単精度実数値, 倍精度実数値を, 整数値に変換します。

書 式 CINT(〈数式〉)

文 例 A%=CINT(B!* 2)

A%=CINT(X #)

〈数式〉の値の小数点以下を四捨五入して整数に変換した値を得ます。結果の値が-32768~32767 の範囲にない場合は、“Overflow” エラーになります。

参照：CDBL, CSNG

CIRCLE

機能

円, 楕円を描きます。

書式

CIRCLE (Wx, Wy) | , <半径> [, <パレット番号 1>] [, <開始角度>] [, <終了角度>] [, <比率>] [, F [, <パレット番号 2> | <タイルistring>]]

文例

CIRCLE (80, 80), 40, 4, 0, 6.28

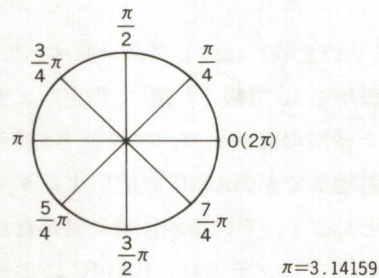
CIRCLE STEP(40, -20), 30, , , , F, TILE\$

ワールド座標の(Wx, Wy)を中心とし、<半径>で指定される大きさの円を描きます。中心座標は、STEPをつけてLP(最終参照点)からの相対座標により指定することもできます。

<パレット番号 1>は、描く円の色をパレット番号で指定します。省略された場合には、COLORで指定されているフォアグラウンドカラーのパレット番号が用いられます。

<開始角度>、<終了角度>を指定すると、指定された角度の範囲内だけに円弧を描きます。角度には $-2\pi \sim 2\pi$ の範囲(π は円周率)のラジアン値を指定します。省略するとそれぞれ、0と 2π が採用されます。<開始角度>、<終了角度>が負であった場合には、その絶対値が用いられますが、そのときには、中心から半径線が描かれますので扇形を描くことができます。

<比率>は、円の偏平率を(垂直方向の半径)/
(水平方向の半径)で指定します。これを指定することで、楕円を描くことができます。ただし、 640×400 ドットのモードでは実際に表示される画面上の長さの比率で指定し、 640×200 ドットのモードではその $1/2$ の値を指定します。省略された場合には、 640×400 ドットのモードでは1.0、 640×200 ドットのモードでは0.5が用いられます。



<比率>が1以外(640×400 ドットのとき、 640×200 ドットのときは0.5以外)の場合、垂直方向の半径と水平方向の半径のうち大きい方に<半径>が用いられます。つまり、 640×400 ドットのモードの場合、<半径>が同じでも<比率>が1以下ならば横長の楕円となり、1以上ならば縦長の楕円となります。

Fを指定すると、円を描くのと同時にその内部を<パレット番号 2>で指定されたパレットの色または<タイルistring>((2)PAINT参照)により指定された模様で塗りつぶします。<パレット番号 2>および<タイルistring>ともに省略された場合には、円を描いたパレットの色で塗りつぶします。開始角度、終了角度が指定されているときにFを指定した場合には、扇形が描かれ内部が塗りつぶされます。

CLOSE

機能	ファイルを閉じます。
書式	CLOSE [(#) <ファイル番号> [, (#) <ファイル番号>] ...]
文例	CLOSE CLOSE #1, #3

CLOSE は、OPEN によって開かれていたファイルを閉じます。閉じられたファイルに対しては、再び開かれるまで入出力を行うことはできません。

<ファイル番号>を指定すると、OPEN で指定したファイル番号に対応するファイルを閉じます。<ファイル番号>を複数個指定すれば、複数のファイルをいちどに閉じることができます。<ファイル番号>を省略した場合には、そのとき開いているファイルすべてを閉じます。

閉じられたファイルにつけられていたファイル番号は、同じあるいは異なるファイルを開くために再び使うことができます。また閉じられたファイルは、同じあるいは異なったファイル番号によって再び開くことができます。

ファイルが出力用に開かれていた場合には、CLOSE を実行すると、バッファに残っていたデータが完全に書き出されます。このため、ファイルの出力処理を正しく終了するには、CLOSE の実行が必要です。

R 指定なしの RUN および R 指定なしの LOAD は、プログラムの実行を開始する前に、開かれているすべてのファイルを自動的に閉じます。また、END あるいは NEW の実行によってもファイルは自動的に閉じられます。

注意：STOP ではファイルを閉じることはできません。また、END が実行されずにプログラムが終了する場合もファイルは閉じられません。

参照：CHAIN, END, LOAD, NEW, OPEN, STOP, サンプルプログラム 26, 27, 28, 29

CLS

機能	現在アクティブな画面をクリアします。
書式	CLS [<機能>]
文例	CLS 2

<機能>は 1, 2, 3 の値をとり、それぞれ次のように働きます。省略された場合には 1 が採用されます。

- 1 : テキスト画面のみをクリアします。テキスト表示モードが CONSOLE, [1] COLOR により、白黒リバースモードになっている場合には画面は白くなります。
- 2 : グラフィック画面のビューポート内をクリアします。グラフィック表示が SCREEN によりカラーモードに設定されている場合には、[1]COLOR により指定されたバックグラウンドカラーで、ビューポート内をクリアします。
- 3 : テキスト画面、グラフィック画面の両方をクリアします。

テキスト画面をクリアする場合、CONSOLE によって指定されたスクロール画面内をクリアします。また、グラフィック画面をクリアした場合には、LP (最終参照点) はビューポートの左上の頂点に移動します。

参照: [1] COLOR, CONSOLE, SCREEN, VIEW

CMD BREAK (DISK モード)

P

機能 ブレイク信号の送出を制御します。

- 書式**
- 1) CMD BREAK [[#] <電話機番号>] <時間>
 - 2) CMD BREAK [[#] <電話機番号>] ON
 - 3) CMD BREAK [[#] <電話機番号>] OFF

文例 CMD BREAK # 1, 2

<電話機番号> で指定された電話機に、ブレイク信号を送ったり止めたりします。

<電話機番号> には、電話機が CMD LINE OPEN の実行によって論理的に接続されたときに指定された番号を指定します。<電話機番号> を省略した場合は 1 を指定したものとみなされます。

書式 1) 指定された <時間> だけブレイク信号を電話機に発信します。<時間> の単位は 1 秒単位で 1~10 の値が指定できます。

書式 2) ブレイク信号の発信を開始します。この命令を実行すると CMD BREAK OFF を実行するまでブレイク信号を電話機に送出し続けます。

書式 3) ブレイク信号の発信を終了します。CMD BREAK ON により送出されていたブレイク信号を止めます。

注意: この命令はデータ通信モードでのみ可能です。

参照: CMD LINE OPEN

CMD CHANGE DUPLEX (DISK モード)**P**

- 機能** 通信方式 (全二重/半二重方式) を切り換えます。
- 書式** CMD CHANGE DUPLEX <ポート番号> [, <通信方式>]
- 文例** CMD CHANGE DUPLEX 1, 1

通常、BASIC は全二重通信方式で RS-232C ポートを制御します。

ところが、モデムによっては半二重方式による通信しか行えないものがあります。

これらのモデムを介して通信を行う場合、コミュニケーションポートを OPEN でオープンする前に、システムの通信方式を半二重方式に切り換えなければなりません。

<ポート番号> には通信方式を変更する RS-232C の回線番号を指定します。

- 1 : RS-232C 第 1 回線
- 2 : RS-232C 第 2 回線
- 3 : RS-232C 第 3 回線

<通信方式> に指定する値と通信方式との対応は次のとおりです。

- 0 または省略 : 全二重
- 1 : 半二重

注意: CMD CHANGE DUPLEX は、コミュニケーションファイルのオープンの前に実行しなければなりません。一度、この命令で通信方式を切り換えると、再度この命令で通信方式を切り換えるまで、指定された通信方式が有効となります。

全二重方式のモデムを使用する場合は、この命令を実行する必要はありません。使用するモデムの説明書で、通信方式を確かめてください。

CMD DIAL (DISK モード)**P**

- 機能** 電話をかけます。
- 書式** 1) CMD DIAL [[#] <電話機番号> ,] <電話番号> [, <機能>]
2) CMD DIAL [[#] <電話機番号> ,] <短縮番号>
- 文例** CMD DIAL # 1, TN\$, TNC
CMD DIAL # 1, 21

書式 1)

指定された電話番号の自動発信を行い、指定されたモードの予約を行います。

〈電話機番号〉には、CMD LINE OPEN で接続された電話機番号を指定します。〈電話機番号〉を省略した場合は 1 を指定したものとみなされます。

〈電話番号〉は 20 桁以内の文字列で、次の値を指定できます。

0~9, *, # : ダイヤルコード。通常の電話番号。
 : : ポーズ。電話機は”：”の前の位置までダイヤルした後、約 2 秒程度発信を中断し、その残りの番号をダイヤルする。これは、構内交換機 (PBX) から外線にダイヤルするとき外線への接続要求を行う必要がある場合などに使用する。

(,) , - : セパレータ。番号を見やすくするために使用。セパレータは特に意味を持たないため、次の電話番号はいずれも同じ動作をする。

×××××××××× ××-×××-××××
 ××(×××)×××× (××)(×××)(××××)

@ : 再ダイヤル。直前にダイヤルした電話番号に再度電話をかける。ただし、@と他の文字との同時指定は不可。また、本体の電源を OFF にしたりリセットスイッチを押したりした直後には、@は使用できない。

〈機能〉は自動発信後の動作を決める数値であり、次の値を指定します。

0 または省略 : ダイヤル後通話モード。ダイヤル後、すぐにこの命令の実行を終了する。相手が話し中の場合や相手が出ない (接続されない) 場合には、手動で電話を切る必要がある。相手が出た場合には通話が可能 (この場合も、電話を切るのは手動で行う)。

1 : アンサートーン検出後データ通信モード。呼び出し相手からアンサートーンが返されるまで待つ。一定時間待ってもアンサートーンが検出されない場合にはエラーとなる。呼び出し相手が出てアンサートーンが検出された場合にはデータ通信モードとなり、この命令の実行を終了する。

2 : 接続後データ通信モード。呼び出し相手が出る (接続される) まで待つ。一定時間待っても相手が出ない場合にはエラーとなる。呼び出し相手が出た場合にはデータ通信モードとなり、この命令の実行を終了する。

3 : ダイヤル後データ通信モード。ダイヤル後はすぐ、データ通信モードとなり、この命令の実行を終了する。

〈機能〉に1, 2または3を使用する場合、相手にはコンピュータなどの接続された自動着信可能な電話機が接続されていなければなりません。もし、手動着信された場合には、正常に動作しない場合があるので注意が必要です。

〈電話番号〉に再ダイヤル”@”を指定し、〈機能〉を省略した場合には、直前に指定した機能が採用されます。機能を指定した場合には、指定した機能が優先されます。

書式 2)

〈短縮番号〉で指定された電話番号の自動発信を行います。

〈短縮番号〉は1~40までの数値でなければなりません。短縮番号を指定した場合、CMD STORE DIAL によってその短縮番号に与えられている機能に関係なく、すぐ次の命令に制御が移行します。

なお、この命令はオフラインコマンドモードでのみ使用可能です。

注意: オートダイヤルの際には、次のことに注意してください。

- (1) 電話機の自動/手動スイッチを“自動”にしておきます (モデムボード+ハンドセットを使用する場合、この必要はありません)。
- (2) 受話器は電話機に置いた状態にしておきます。
- (3) 電話機のモードスイッチが AUTO あるいは CALL にセッティングされていなければなりません。
- (4) 本体のボーレート指定と電話機の通信速度の値が、一致していなければなりません。
- (5) モデムボード+ハンドセットでは、書式 2)の短縮番号による自動発信は行えません。

参照: CMD STORE DIAL

CMD ERROR ON/OFF/STOP (DISK モード)

P

機能 通信エラーによる割り込みを許可、禁止、停止します。

書式

- 1) CMD ERROR [[#] 〈電話機番号〉] ON
- 2) CMD ERROR [[#] 〈電話機番号〉] OFF
- 3) CMD ERROR [[#] 〈電話機番号〉] STOP

文例 CMD ERROR #1 ON

〈電話機番号〉で指定した電話機の通信エラーによる割り込みを許可、禁止、停止します。なお、ここで述べている通信エラーとは、データ通信モードのときモデム(あるいはモデム電話)からデータを受信した際のエラーを意味しています。

〈電話機番号〉には、CMD LINE OPEN で接続された電話機番号を指定します。〈電話機番号〉を省略した場合は 1 を指定したものとみなされます。

書式 1) 割り込み動作を許可します。この命令実行後は、通信エラー割り込みが発生すると、CMD ON ERROR GOSUB によって定義された処理ルーチンへ分岐します。

書式 2) 割り込み動作を禁止します。この命令実行後は、通信エラー割り込みが発生しても、処理ルーチンへの分岐は起こりません。

書式 3) 割り込み動作を停止します。この命令実行後は、通信エラー割り込みが発生しても割り込みが起こったことを覚えているだけで、処理ルーチンへの分岐は起こりません。しかし、その後 CMD ERROR ON によって割り込みが許可されると、その時点で処理ルーチンに分岐します。

参照：CMD ON ERROR GOSUB, サンプルプログラム 40

CMD LINE CLOSE (DISK モード)

P

機能 BASIC と論理的に接続されている電話機を切り離します。

書式 CMD LINE CLOSE [[#] 〈電話機番号〉]

文例 CMD LINE CLOSE #1

この命令は電話機と BASIC を論理的に切り離す命令で、電話制御命令の使用を終了するものです。切り離し対象となるコミュニケーションポートを OPEN でオープンしたファイルがある場合には、この命令の実行により自動的にそのファイルがクローズされます。

〈電話機番号〉には、CMD LINE OPEN で接続された電話機番号を指定します。また、〈電話機番号〉を省略すると、現在接続されている電話機がすべて切り離されます。

参照：CMD LINE OPEN

CMD LINE ON/OFF/STOP (DISK モード)

P

機能 電話機の着信による割り込みを許可、禁止、停止します。

書式

- 1) CMD LINE [[#] 〈電話機番号〉] ON
- 2) CMD LINE [[#] 〈電話機番号〉] OFF
- 3) CMD LINE [[#] 〈電話機番号〉] STOP

文例 CMD LINE #1 ON

〈電話機番号〉で指定した電話機への着信割り込みを許可、禁止、停止します。

〈電話機番号〉には、CMD LINE OPEN で接続された電話機番号を指定します。〈電話機番号〉を省略した場合は 1 を指定したものとみなされます。

書式 1) 割り込み動作を許可します。この命令実行後は、着信割り込みが発生すると、CMD ON LINE GOSUB によって定義された処理ルーチンに分岐します。

書式 2) 割り込み動作を禁止します。この命令実行後は、着信割り込みが発生しても、処理ルーチンへの分岐は起こりません。

書式 3) 割り込み動作を停止します。この命令実行後は、着信割り込みが発生しても割り込みが起こったことを覚えているだけで、処理ルーチンへの分岐は起こりません。しかし、その後 CMD LINE ON によって割り込みが許可されると、その時点で処理ルーチンに分岐します。

注意：着信割り込みとは、電話がかかってきたという事象を示しているものであり、CMD RECEIVE (自動着信) の指定とは無関係に発生します。

STATUS LINE (関数) を使用しているプログラムでは、正常な着信割り込みが発生しない場合がありますので CMD LINE ON/OFF/STOP と混在して使用しないでください。

参照：CMD LINE GOSUB, STATUS LINE, サンプルプログラム 40

CMD LINE OPEN (DISK モード)

P

機能 電話機を BASIC と論理的に接続し、オートダイヤル、自動発着信などの機能を利用可能にします。

書式 CMD LINE OPEN 〈ファイルディスクリプタ〉 [AS [#] 〈電話機番号〉]

文例 CMD LINE OPEN "COM1:" AS #1

この命令は電話機を BASIC と論理的に接続する命令であり、すべての電話制御命令の実行に先だって実行しなければなりません。

この命令を実行すると、〈ファイルディスクリプタ〉で指定されたコミュニケーションポートにつながっている電話機のモードを、初期モードからオフラインコマンドモードに移行します (電話機のモードの詳細に関しては「BASIC ユーザーズマニュアル」を参照)。

〈ファイルディスクリプタ〉には、"COM:" ("COM1:"でも同じ)、"COM2:", "COM3:" のいずれかを指定します。ただし、"COM2:", "COM3:" は、RS-232C 拡張インタフェースボードを実装している場合にのみ指定可能です。

〈電話機番号〉には1~3までの値を指定します。以後、電話制御命令はこの〈電話機番号〉を指定して行います。省略した場合は1が採用されます。

注意：〈電話機番号〉の指定のしかたは、一般のファイル入出力の際の〈ファイル番号〉と似ていますが、〈電話機番号〉と〈ファイル番号〉は別のもので、同時に同じ番号を使用しても各々が無関係に正常に動作します。

なお、この命令は電話機が初期モードにあるときのみ使用できます。

参照：CMD LINE CLOSE, STATUS MODE, サンプルプログラム 40

CMD MODE CUT (DISK モード)

P

機能	電話を切ります。
書式	CMD MODE CUT [[#] 〈電話機番号〉]
文例	CMD MODE CUT #1

〈電話機番号〉で指定された電話を切って、オフラインコマンドモードにもどります。〈電話機番号〉を省略した場合は1を指定したものとみなされます。

注意：この命令はデータ通信モードで有効です。

CMD ON ERROR GOSUB (DISK モード)

P

機能	通信エラーによる割り込みルーチンの開始行を定義します。
書式	CMD ON ERROR [[#] 〈電話機番号〉] GOSUB 〈行番号〉
文例	CMD ON ERROR #1 GOSUB *TEL.REC

〈電話機番号〉で指定した電話機に通信エラー割り込みが発生したときに分岐する、処理ルーチンの開始行を定義します。

〈電話機番号〉には、CMD LINE OPEN で接続された電話機番号を指定します。〈電話機番号〉を省略した場合は1を指定したものとみなされます。

〈行番号〉には処理ルーチンの開始行番号を指定します。

処理ルーチンからの復帰は、一般のサブルーチンの場合と同じで、RETURN によって行います。たんに RETURN としたときは中断したところから再開し、後ろに行番号を指定したときはその行から実行を再開します。

ここで述べている通信エラーとは、データ通信モードのときモデム（またはモデム電話）からデータを受信した際のエラーを意味しています。

この命令によって割り込み処理ルーチンへ分岐させるには、この事象に対する割り込みが許可状態（CMD ERROR ON の状態）になっていなくてはなりません。

注意：通信エラーの情報は、コミュニケーションバッファ内に記憶されていますので、割り込み処理ルーチン内で INPUT\$(関数) を使用してバッファ内のデータを読み取らないと、エラー情報はリセットされません。

参照：CMD ERROR ON/OFF/STOP, サンプルプログラム 40

CMD ON LINE GOSUB (DISK モード)

P

機能 電話機の着信による割り込みが発生したときの処理ルーチンの開始行を定義します。

書式 CMD ON LINE [[#] <電話機番号>] GOSUB <行番号>

文例 CMD ON LINE #1 GOSUB *TEL.REC

<電話機番号>で指定した電話機に着信割り込みがあったときに分岐する、処理ルーチンの開始行を定義します。

<電話機番号>には、CMD LINE OPEN で接続された電話機番号を指定します。<電話機番号>を省略した場合は 1 を指定したものとみなされます。

<行番号>には処理ルーチンの開始行を指定します。

処理ルーチンからの復帰は、一般のサブルーチンの場合と同じで、RETURN によって行います。たんに RETURN としたときは中断したところから再開し、後ろに行番号を指定したときは、その行から実行を再開します。

この命令によって割り込み処理ルーチンへ分岐させるには、この事象に対する割り込みが許可状態（CMD LINE ON の状態）になっていなくてはなりません。

注意：着信割り込みとは、電話がかかってきたという事象を示すもので、CMD RECEIVE(自動着信) の指定とは無関係に発生します。

STATUS LINE (関数) を使用しているプログラムでは、正常に着信割り込みが発生しない場合がありますので、CMD ON LINE GOSUB と混在して使用しないでください。

参照：CMD LINE ON/OFF/STOP, STATUS LINE, サンプルプログラム 40

CMD RECEIVE (DISK モード)

P

機能	電話機の自動着信を行うか否かの設定をします。
書式	CMD RECEIVE [[#] <電話機番号>] [<機能>]
文例	CMD RECEIVE #1 CMD RECEIVE 1

<電話機番号>には、CMD LINE OPEN で接続された電話機番号を指定します。<電話機番号>を省略した場合は1を指定したものとみなされます。

<機能>には、自動着信後の動作モードを規定する数値を指定します。指定する数値と動作モードとの関係は次のとおりです。

- 0 または省略 : 自動着信を行わない。電話がかかってきた場合、電話機のベルが鳴り受話器を取るのを待つ。
- 1 : 着信後、データ通信モードに切り換える。電話がかかってきた場合、アンサートーンを返した後にデータ通信モードとなる。
着信割り込み処理ルーチン内でこの命令を実行すると、ただちにアンサートーンを返してデータ通信モードとなる。

注意：着信を検出してからデータ通信モードに移行するまでには時間がかかりますので、STATUS MODE (関数) を使用してデータ通信モードに移行したことを確認した後、データ通信を行ってください。

この命令はオフラインコマンドモードでのみ有効です。

参照：STATUS MODE, サンプルプログラム 40

CMD STORE DIAL (DISK モード)

P

機能	短縮ダイヤルを電話機に記憶させます。
書式	CMD STORE DIAL [[#] <電話機番号>] <短縮番号> AS <電話番号> [<機能>]
文例	CMD STORE DIAL #1, 21 AS "03(117)1111"

電話番号を電話機に記憶させる命令です。

<電話機番号>には、CMD LINE OPEN で接続された<電話機番号>を指定します。<電話機番号>を省略した場合は1を指定したものとみなされます。

〈短縮番号〉には、21～40の範囲の数値を指定します。全部で合計20個までの番号を登録することができます。

〈電話番号〉には記憶させる電話番号を20桁以内の文字列で指定します。指定できる値は次のとおりです。

- 0～9, *, # : ダイヤルコード
 : : ポーズ
 (,) , - : セパレータ

また、〈電話番号〉に"" (空の文字列) を指定すると、その短縮番号に記憶されている内容がクリアされます。

〈機能〉には自動発信後の動作を規定する数値を指定します。

- 0 または省略 : ダイヤル後通話モードとなる。
 1 : アンサートーン検出後データ通信モードとなる。
 2 : 接続後データ通信モードとなる。
 3 : ダイヤル後データ通信モードとなる。

〈電話番号〉および〈機能〉の詳細は、CMD DIAL を参照してください。

注意: この命令はオフラインコマンドモードでのみ使用可能です。モデムボード+ハンドセットでは、この命令を使用することはできません。

参照: CMD DIAL

[1] COLOR

- 機能** ディスプレイ画面の各部の色およびグラフィック画面のパレットモードを指定します。
- 書式** COLOR [〈ファンクションコード〉] [, 〈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〉] [, 〈パレットモード〉]
- 文例** COLOR 7, 0, 0, 7

[1]COLOR は、テキスト画面の文字のカラーを変えたり、グラフィック画面のフォアグラウンド、バックグラウンド、ボーダーの各カラーを設定したり、グラフィック画面のパレットモードを指定したりするのに使用します。

[1] COLOR

〈ファンクションコード〉は、テキスト画面の文字にいろいろな機能を与えます。このファンクションコードは、テキスト画面がカラーモードになっているか、白黒モードになっているかによって、働きが異なります。なお、カラーモード／白黒モードの切り替えは CONSOLE で行います。〈ファンクションコード〉で指定する値は、“パレット番号”ではありませんので、[2] COLOR によってカラーパレットの変更を行っても色は変化しません。

白黒モードの場合(CONSOLE ,,, 0)

- 0 : ノーマル(通常の表示)
- 1 : シークレット(文字は表示されない)
- 2 : プリンク(点滅する)
- 3 : シークレット(1と同じ)
- 4 : リバース(反転する)
- 5 : リバースシークレット(反転して文字は表示されない)
- 6 : リバースプリング(反転して点滅する)
- 7 : リバースシークレット(5と同じ)

カラーモードの場合(CONSOLE ,,, 1)

- | | | | |
|-------|--------|-------|-------|
| 0 : 黒 | 1 : 青 | 2 : 赤 | 3 : 紫 |
| 4 : 緑 | 5 : 水色 | 6 : 黄 | 7 : 白 |

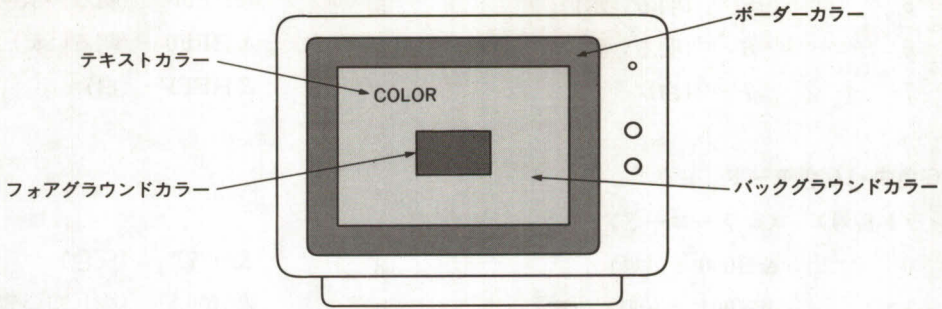
〈バックグラウンドカラー〉は、グラフィック画面の地の色を表します。このパラメータは、“パレット番号” (〈パレットモード〉の項および [2] COLOR 参照) によって指定します。このパラメータの設定後 CLS によって画面をクリアすると、設定色によって画面がぬり変えられます。また、以後 PRESET を色指定なしで実行すると、この色が採用されます。

なお、白黒モード(高分解能白黒モードを含む)では、〈バックグラウンドカラー〉として指定されたパレット番号は、0 のときは黒、0 以外のときは白とみなされます。

〈ボーダーカラー〉は、画面の中で BASIC によって使うことのできる領域外の、枠の色を表すもので、次のような値で指定します。

- | | | | |
|-------|--------|-------|-------|
| 0 : 黒 | 1 : 青 | 2 : 赤 | 3 : 紫 |
| 4 : 緑 | 5 : 水色 | 6 : 黄 | 7 : 白 |

〈ボーダーカラー〉が指定されると、カラーモードでも白黒モードでも、画面の外枠に色(白黒のディスプレイでは濃淡)がついて表示されます。なお、このパラメータは、専用高解像度ディスプレイの使用時には意味がありません。



〈フォアグラウンドカラー〉は、グラフィック画面に点や線を表示したりするときに使われる色を表します。このパラメータは、“パレット番号”によって指定します。種々のグラフィック命令(PSET, LINE, CIRCLE など)でとくに色指定をしなかった場合、この色が採用されます。

なお、白黒モード(高分解能白黒モードを含む)では、〈フォアグラウンドカラー〉として指定されたパレット番号は、0 のときは黒、0 以外のは白とみなされます。

〈パレットモード〉は、グラフィック画面に対する色指定のモードを指定します。〈パレットモード〉に指定できる値とその意味は次のとおりです。

- 0 : 8 色中・8 色モード。8 個のパレットにシステムが決めた 8 色を対応づけるモードです。
- 1 : 4096 色中・8 色モード。8 個のパレットに 4096 色中の任意の 8 色を対応づけるモードです。
- 2 : 4096 色中・16 色モード。16 個のパレットに 4096 色中の任意の 16 色を対応づけるモードです。

N₈₈-BASIC(86)の起動直後のパレットモードは 8 色中・8 色モードです。以降、〈パレットモード〉が指定された場合に限りパレットモードが切り替わります。各モードにおけるパレットとカラーコードの対応づけは、〔2〕 COLOR によって設定されますので、参照してください。

〈パレットモード〉が指定されてパレットモードが切り替わると、そのたびにパレットとカラーコードの関係は次のように初期化されます。

8 色中・8 色モード

〈パレット番号〉	〈カラーコード〉
0	0 (黒)
1	1 (明るい青)
2	2 (明るい赤)
3	3 (明るい紫)
4	4 (明るい緑)

4096 色中・8 色モード

〈パレット番号〉	〈カラーコード〉
0	&H000 (黒)
1	&H00F (明るい青)
2	&H0F0 (明るい赤)
3	&H0FF (明るい紫)
4	&HF00 (明るい緑)

[1] COLOR, [2] COLOR

5	5	(明るい水色)	5	&HF0F	(明るい水色)
6	6	(明るい黄)	6	&HFF0	(明るい黄)
7	7	(白)	7	&HFFF	(白)

4096 色中・16 色モード

〈パレット番号〉 〈カラーコード〉

0	&H000	(黒)	8	&H777	(灰色)
1	&H00F	(明るい青)	9	&H00A	(少し暗い青)
2	&H0F0	(明るい赤)	10	&H0A0	(少し暗い赤)
3	&H0FF	(明るい紫)	11	&H0AA	(少し暗い紫)
4	&HF00	(明るい緑)	12	&HA00	(少し暗い緑)
5	&HF0F	(明るい水色)	13	&HA0A	(少し暗い水色)
6	&HFF0	(明るい黄)	14	&HAA0	(少し暗い黄)
7	&HFFF	(白)	15	&HAAA	(少し暗い白)

注意：4096 色中・8 色モードおよび 4096 色中・16 色モードはアナログ RGB 対応ディスプレイが接続されている場合のみ有効です。アナログ RGB 対応ディスプレイが接続されていない場合でも、エラーにはなりません、指定どおりの色は出ません。

参照：[2] COLOR, COLOR@, CONSOLE, サンプルプログラム 15, 16

[2] COLOR

機能 カラーパレットの色を変更します。

書式 COLOR[(〈パレット番号〉, 〈カラーコード〉)]

文例 COLOR=(2, 4)

COLOR=(PAL, COL)

グラフィック画面への色の指定は、すべてカラーパレットによって行いますが、[2]COLOR は、このカラーパレットの色を決める命令です。

〈パレット番号〉には0から7(あるいは0から15)の8個(あるいは16個)のカラーパレットの固有の番号を指定します。

〈カラーコード〉には、〈パレット番号〉にどの色を対応づけるかを指定します。

パレットの個数およびカラーコードの値は、パレットモード([1]COLOR 参照)によって異なります。各モードにおける〈パレット番号〉と〈カラーコード〉の関係は次のとおりです。

8色中・8色モード

0~7の〈パレット番号〉に0~7の8個の〈カラーコード〉を任意に指定します。

4096色中・8色モード

0~7の〈パレット番号〉に&H000~&HFFFの4096個の〈カラーコード〉から任意に指定します。

4096色中・16色モード

0から15の〈パレット番号〉に&H000~&HFFFの4096個の〈カラーコード〉から任意に指定します。

どのモードにおいても、パレット番号とカラーコードは任意に対応させることができます。また、同じカラーコードを複数のパレットに対応づけてもかまいません。

等号(=), 〈パレット番号〉および〈カラーコード〉を省略して, "COLOR" とすると, パレットとカラーコードの関係を初期化することができます(DISKモードでのみ有効)。パレットとカラーコードの初期状態に関しては, [1] COLOR を参照してください。

注意: 各モードにおけるカラーコードの値と色の関係は BASIC ユーザーズマニュアルを参照してください。

参照: [1] COLOR, サンプルプログラム 15, 16

COLOR@

機能	テキスト画面に書かれた文字などに色や機能を設定します。
書式	COLOR@ (X1, Y1)-(X2, Y2) [, 〈ファンクションコード〉]
文例	COLOR@ (0, 0)-(79, 4), 2

COLOR@は、テキスト画面のキャラクタ座標の2点(X1, Y1), (X2, Y2)を対角とする四角形の領域に書かれている、文字やグラフィックキャラクタに色をつけたり、プリンクなどの機能を設定したりします。

(X1, Y1), (X2, Y2)は、必ずキャラクタ座標でなくてはなりません。

〈ファンクションコード〉は、CONSOLEによって設定されているモードによって持つ意味が異なります。この機能および指定の仕方は [1] COLOR の〈ファンクションコード〉の場合とまったく同様ですのでそちらを参照してください。省略された場合は、7を値として採用します。

この命令は、テキスト画面に書かれている文字などに対して有効です。したがって何も書かれていない場合は、何の作用もありません。またこの命令を実行した領域の上に新しく文字などを書いた場合、書かれた文字はこの命令の影響を受けません。

なお、COLOR@で指定した領域内に白黒モードで描かれたグラフィックスがある場合は、グラフィックスの色もCOLOR@で指定した色に変わります。カラーモードで描かれたグラフィックスは、COLOR@の影響を受けません。

参照：CONSOLE, [1] COLOR, サンプルプログラム 33

COMMON (DISK モード)

機能	CHAIN が実行された際、メモリ上のプログラムから、実行の移されたプログラムに変数を引き渡します。
書式	COMMON <変数名> [, <変数名> ...]
文例	COMMON A, B, XY(), NA\$

COMMON は、CHAIN によってメモリ上のプログラムからディスク上のプログラムに実行を移す際、メモリ上のプログラムの変数を実行の移されたプログラムに引き渡します。なお、CHAIN で MERGE オプションが指定されたときには、メモリ上のプログラムとディスク上のプログラムとが連結された後のプログラムに変数を引き渡します。

したがって、COMMON は必ず CHAIN とあわせて使われます。また、プログラム内において、COMMON は CHAIN の前にある必要があります。

<変数名> は 1 行の許す範囲(255 バイト以内)で何個でも並べることができますが、1 つのプログラム中の COMMON で同じ変数名を指定してはなりません。また配列変数名は()を後ろにつけ加えて表現します。

すべての変数を引き渡したいのであれば、CHAIN の ALL オプションを使う方が便利です。

参照：CHAIN, サンプルプログラム 1, 2

COM ON/OFF/STOP

機能 RS-232C 回線からの割り込みの許可、禁止、停止を制御します。

書式

- 1) COM [(回線番号)] ON
- 2) COM [(回線番号)] OFF
- 3) COM [(回線番号)] STOP

文例

COM ON

COM(2) ON

RS-232C 回線に外部からの通信が入ったことによる割り込みの許可、禁止、停止を宣言します。

RS-232 C(第2 回線/第3 回線)インタフェースボードを使用する場合、〈回線番号〉に指定できる値と意味は次のとおりです。〈回線番号〉を省略(カッコも含む)した場合は、第1 回線となります。

- 1 : RS-232C 第1 回線
- 2 : RS-232C 第2 回線
- 3 : RS-232C 第3 回線

RS-232 C(第2 回線/第3 回線)インタフェースボードを使用しない場合、〈回線番号〉は必ずカッコも含めて省略してください。

書式1) 割り込みを許可します。以後、〈回線番号〉で指定した RS-232C 回線に受信が入るごとに割り込みが発生し、ON COM GOSUB によって定義された処理ルーチンに分岐します。

書式2) 割り込みを禁止します。以後、通信があっても処理ルーチンへの分岐は起こりません。

書式3) 割り込みを停止します。以後、通信があってもそのことを覚えているだけで、処理ルーチンへの分岐は起こりません。しかし、その後、COM ON によって割り込みが許可されると、前に通信があったことによって、処理ルーチンに分岐します。

注意：プログラムの終了時には COM OFF を実行しておいてください。COM ON は、RS-232C 回線を OPEN した後実行されねば有効となりません。

なお、割り込み処理ルーチンに制御が移ると自動的に割り込み停止状態となりますので、割り込み処理ルーチンの先頭では COM OFF あるいは COM STOP を実行しないでください。

参照：ON COM GOSUB

CONSOLE

機能	テキスト画面モードの設定を行います。
書式	CONSOLE [〈スクロール開始行〉] [, 〈スクロール行数〉] [, 〈ファンクションキー表示スイッチ〉] [, 〈カラー／白黒スイッチ〉]
文例	CONSOLE 0, 24, 0, 1 CONSOLE ,, 1, 0

〈スクロール開始行〉で指定した行以降の、〈スクロール行数〉で指定した行数分を、画面上でスクロールする領域（スクロール画面）として設定します。画面のクリア（CLSあるいはPRINT CHR\$(12)）は、このスクロール画面に対して実行されます。

〈ファンクションキー表示スイッチ〉に1を指定すると、画面最下行にファンクションキーに定義されている文字列を表示します。0を指定すると、表示をとりやめます。起動直後は表示される状態になっています。

〈カラー／白黒スイッチ〉に1を指定すると、テキスト画面をカラーモードにし、0を指定すると白黒モードにします。起動直後は白黒モードになっています。

参照： [1] COLOR, COLOR@, サンプルプログラム 33

CONT

機能	STOPキーあるいはCTRL + Cの入力、またはSTOPによって停止したプログラムの実行を再開します。
書式	CONT
文例	CONT

CONTは通常デバッグのために用いられます。STOPキー（またはCTRL + C）の入力や、プログラム中のSTOPの実行によってプログラムの実行を停止すると、ダイレクトモードで停止時における変数の内容などを調べることができます。この後、CONTを実行することにより、プログラムの実行を再開することができます。

注意：実行停止中にプログラム内容の変更を行った場合には、CONTによる継続実行はできません。また、実行停止のタイミングによってはプログラムの継続実行ができない場合があります。

COPY

機能	画面情報のハードコピーを行います。
書式	COPY [<機能>]
文例	COPY 2 COPY

2種類の画面ハードコピー機能が用意されています。

(1) 通常の画面ハードコピー機能を使用する場合。

<機能>に指定できる値は1~5で、それぞれ次のように働きます。**<機能>**が省略された場合、3を値として採用します。

- 1 : テキスト画面のみを出力。
- 2 : グラフィック画面のみを出力。
- 3 : テキスト画面とグラフィック画面を重ね合わせて出力。このときテキスト画面の情報はプリンタの印字体で出力される。
- 4 : グラフィック画面のみを縦方向に縮小して出力。640×200モードの場合に有効。
- 5 : テキスト画面とグラフィック画面を重ね合わせ、さらに縦方向に縮小して出力。640×200モードの場合に有効。

注意：PC-PR201系プリンタを使用する場合は、ユーティリティ“switch.n88”を使って“PC-PR201系プリンタの使用”を設定しておいてください。

(2) 拡張画面ハードコピー機能を使用する場合(DISKモードのみ)。

あらかじめユーティリティ“switch.n88”を使って拡張画面ハードコピー機能を使うことを設定しておいてください。

<機能>に指定できる値は1~5で、それぞれ次のように働きます。**<機能>**が省略された場合、3を値として採用します。

なお、4、5の機能はカラー出力の場合のみ有効です。

- 1 : テキスト画面のみを出力。
- 2 : グラフィック画面のみを出力(カラー出力の場合、白黒が反転される)。
- 3 : テキスト画面とグラフィック画面を合成して出力(カラー出力の場合、白黒が反転される)。このときテキスト画面の情報は画面のドットイメージで出力される。
- 4 : グラフィック画面のみを出力(白黒は反転されない)(カラー出力用)。
- 5 : テキスト画面とグラフィック画面を合成して出力(白黒は反転されない)(カラー出力用)。

PC-PR601系プリンタを使用する場合、上記の1~3の機能に対して、印字方向や印字倍率を指定することができます。詳しくは BASIC ユーザーズマニュアルを参照してください。

注意：PC-PR201系またはPC-PR601系プリンタを使用する場合は、ユーティリティ“switch.n88”を使って“PC-PR201系プリンタの使用”を設定しておいてください。

また、カラー出力を行う場合、次の準備が必要です。

- PC-PR201V系カラープリンタを接続する
- ユーティリティ“switch.n88”でカラー出力を行うことを設定する

詳しくは BASIC ユーザーズマニュアルを参照してください。

高分解能白黒モードで4096色中・16色モードにおいて、ディスプレイページがページ4か8を含んでいる場合は、拡張画面ハードコピー機能を使用してください。

COS

関数

機能	余弦(コサイン)を得ます。
書式	COS(<数式>)
文例	X=RADIUS * COS(ANGLE) PRINT COS(3.14159/4)

<数式>の値に対する余弦値を得ます。<数式>の単位はラジアンで指定します。

<数式>に倍精度実数が含まれる場合、得られる値は倍精度となりますが、他の場合には単精度となります。

参照：ATN, SIN, TAN, サンプルプログラム6, 30

CSNG

関数

機能	整数値、倍精度実数値を、単精度実数値に変換します。
書式	CSNG(<数式>)
文例	A!=CSNG(B#/4) PRINT CSNG(128%)

<数式>の値を有効数字6桁の単精度実数値に変換した値を得ます。結果の値が-1.70141E+38~1.70141E+38の範囲にない場合は、“Overflow”エラーになります。

参照：CINT, CDBL

CSRLIN

関 数

機 能	現在のカーソルの行位置を得ます。
書 式	CSRLIN
文 例	Y=CSRLIN

現在のカーソルの行(垂直)位置をキャラクタ座標で得ます。値の範囲は 25 行モードでは 0～24, 20 行モードでは 0～19 となります。得られる値は、画面の最上行が 0, 最下行が 24(または 19)を意味します。

参照 : POS, サンプルプログラム 10

CVI/CVS/CVD

関 数

機 能	文字列を数値データに変換します。
書 式	1) CVI(<2 文字の文字列>) 2) CVS(<4 文字の文字列>) 3) CVD(<8 文字の文字列>)
文 例	A%=CVI(A\$) B=CVS("A3Bd") C#=CVD(NUM\$)

これらの関数は、MKI\$/MKS\$/MKD\$の各関数を用いてランダムデータファイルに書き込まれた文字型化数値データを、数値データにもどす際に使用します。

- CVI : 2 文字(2 バイト)の文字列を整数値に変換(MKI\$の逆)。
 CVS : 4 文字(4 バイト)の文字列を単精度実数値に変換(MKS\$の逆)。
 CVD : 8 文字(8 バイト)の文字列を倍精度実数値に変換(MKD\$の逆)。

実際にランダムデータファイルから数値データを読み込む際には、まず GET でフィールド変数に文字型化数値データを読み込み、次にこれらの関数を用いて数値データに変換します。

参照 : MKI\$/MKS\$/MKD\$, サンプルプログラム 27

DATA

機能	READ で読み込まれる数値定数, 文字定数を定義します。
書式	DATA <定数> [, <定数> ...]
文例	DATA 1, CBA, 1465

<定数>には文字型か数値型のデータを指定します。READ で読み込む場合、DATA で指定した文字定数は文字変数に読み込まなくてはなりませんが、数値定数は文字変数、数値変数のいずれに読み込むこともできます。但し、定数式は使えません。

DATA 行中には、1行(255バイト)に入るだけのデータをセットすることができます。データはコンマ(,)によって区切りますが、文字定数で、その中にコンマやピリオド(.), 意味のある空白を置きたい場合、および日本語文字を含む文字列である場合は、その文字定数の前後をダブルクォーテーション(")でくる必要があります。

1つのプログラムには任意の数の DATA 行を置くことができます。DATA 行はプログラム中のどこに置いてかまいません。READ は行番号の小さい方から順番に、DATA 行中のデータを読み込んでいきます。

参照: READ, RESTORE, サンプルプログラム 3, 14

DATE\$

関数

機能	日付を得ます。
書式	1) DATE\$ 2) DATE\$="yy/mm/dd"
文例	A\$=DATE\$ DATE\$="87/12/12"

DATE\$には常に現在の日付が"yy/mm/dd"(年/月/日)の形で入れられており、いつでもその内容を見ることができます。

なお、書式2)を用いることにより、日付を変更することもできます。yyには00~99、mmには01~12、ddには01~31の範囲の整数値を指定します。yy、mm、ddは、それぞれ"/"(スラッシュ)で区切るようにしてください。

注意: 日付は、バッテリーバックアップによって自動的に更新され、正しく維持されるようになっていきます。不用意に日付を変えないようにしてください。

参照: TIME\$

DEF FN

機能	利用者定義関数を指定します。
書式	DEF FN<名前>[(<パラメータリスト>)] = <関数の定義式>
文例	DEF FNA(X, Y)=X * 2+Y * 3

DEF FN は、利用者が使うことのできる関数を定義する命令です。定義する関数は、数値関数、文字関数、その混在のいずれでもかまいません。

DEF FN は、プログラム中でしか実行することはできません。

<名前> は変数名として正しい形をしたものでなければなりません。

<パラメータリスト> には、<関数の定義式> 中で使われている変数とおなじ名前の変数を用います。この変数名は <関数の定義式> を評価する際にのみ有効なものであり、プログラム中に同一名の変数があってもかまいません。

<関数の定義式> は、その関数の演算内容を記述する式で、1 行の範囲(255 バイト以内)に限られます。

定義した関数は「FN 名前(変数)」という形で呼び出して使います。定義したときの変数は仮のものであるため、呼び出すときにはその時々で必要な変数に変えて指定することができます。ただし、変数の型は同一でなければなりません。

注意：DEF FN の <関数の定義式> 中で使われている変数が、<パラメータリスト> 内にない場合は、その変数がある時点で持っている値が使われます。

DEF FN は、これによって定義される関数がプログラム中で呼ばれる前に実行されていなければなりません。

DEF FN は、DEF と FN の間に必ずスペースを入れなければなりません。スペースを省略して DEFFN とすると変数として扱われてしまいます。

DEF FN に関するエラーが検出された場合、エラー発生行は、ふつうその定義関数を呼び出した行となります。しかし、<関数の定義式> の評価中にエラーが検出された場合、エラー発生行は、その定義関数を定義した DEF FN の含まれている行となります。

参照：サンプルプログラム 39

DEFINT/DEFSNG/DEFDBL/DEFSTR

機能 変数の型宣言を行います。

書式

- 1) DEFINT <文字の範囲> [, <文字の範囲> ...]
- 2) DEFSNG <文字の範囲> [, <文字の範囲> ...]
- 3) DEFDBL <文字の範囲> [, <文字の範囲> ...]
- 4) DEFSTR <文字の範囲> [, <文字の範囲> ...]

文例

```
DEFINT A, I-K
DEFSNG B
DEFDBL X-Z
DEFSTR L-N, Q
```

<文字の範囲>で指定された文字で始まる変数を、DEFINT では整数型に、DEFSNG では単精度実数型に、DEFDBL では倍精度実数型に、DEFSTR では文字型にそれぞれ定義します。

<文字の範囲>で指定できる文字は英字1文字で、複数個指定する場合はマイナス記号でつないでその範囲を示します。

注意: この命令によって行われる型宣言より、型宣言文字による指定(%、!、#、\$)の方が優先されます。また、型宣言が行われていない文字で始まり、型宣言文字による指定もされていない変数は、すべて単精度変数とみなされます。

参照: サンプルプログラム 4

DEF SEG

機能 セグメントベースを宣言します。

書式 DEF SEG= <セグメントベース>

文例 DEF SEG=&H9E00

BSAVE, BLOAD, PEEK, POKE などの命令でメモリのアドレスを指定する場合には、まず、アクセスしようとするメモリ空間を含むセグメントを、DEF SEG であらかじめ宣言する必要があります。そして、実際のメモリのアドレスの指定の際には、該当するセグメント内での相対アドレスで指定します。

<セグメントベース>には、セグメントの物理アドレスを16で割った値を指定します。

各変数や各配列が記憶されているセグメントベースは、VARPTR 関数により得ることができます。なお、配列については配列ごとに別セグメントとなっていますので、複数の配列の要

素あるいは配列要素と変数にアクセスする場合には、そのつどセグメントベースを切り換えてアクセスすることが必要となります。

注意：DEF SEG は、DEF と SEG の間に必ずスペースを入れなければなりません。スペースを省略してDEFSEG とすると変数として扱われてしまいます。

参照：BLOAD, BSAVE, CLEAR, DEFUSR, PEEK, POKE, VARPTR

DEFUSR

機能 USR で呼び出す機械語関数の番号と実行開始アドレスを定義します。

書式 DEFUSR(<番号>)=<開始アドレス>

文例 DEFUSR3=&HF000

<番号>は、0~9 までの値で、複数の機械語関数を用いる場合の識別を行います。最大 10 個までの機械語関数を準備、利用することができます。<番号>が省略された場合には 0 と解釈されますので、USR0 と USR は同じ意味となります。

<開始アドレス>は、<番号>によって指定される機械語関数の実行開始番地を、直前に実行された DEFSEG で指定されたセグメントベースからの相対アドレスで指定します。

なお、USR で呼び出したい機械語関数は、あらかじめメモリ上に用意しておかなくてはなりません。このためには、POKE や BLOAD を用いることができます。

注意：DEFUSR は、DEF と USR の間に必ずスペースを入れなければなりません。スペースを省略してDEFUSR とすると変数として扱われてしまいます。

参照：BLOAD, BSAVE, CLEAR, DEFSEG, POKE, USR

DELETE

機能 プログラムの部分削除を行います。

書式 DELETE |<始点行番号> [-<終点行番号>]|
| [<始点行番号>] -<終点行番号>|

文例 DELETE 1050

DELETE *START-*SUB

DELETE .-500

〈始点行番号〉から〈終点行番号〉までのプログラムを削除します。〈始点行番号〉だけを指定した場合はその行だけを削除し、マイナス記号以下〈終点行番号〉を指定した場合には、プログラムの先頭から指定行までを削除します。

なお、〈始点行番号〉および〈終点行番号〉の両方とも省略することはできません。

〈行番号〉にピリオド(.)を指定すると BASIC インタプリタ内のポインタが示している現在行を指定したことになります。実際には、LIST コマンドで最後に表示された行や、編集機能で最後に修正された行が現在行となります。

DIM

機能	配列変数の要素の大きさを指定し、メモリ領域に割り当てます。
書式	DIM 〈変数名〉(〈添字の最大値〉 [, 〈添字の最大値〉 …]) [, 〈変数名〉(〈添字の最大値〉 [, 〈添字の最大値〉 …])…]
文例	DIM A(12, 2), B\$(3)

DIM は、配列変数の添字の最大値を設定し、同時にメモリ上にその配列の領域を割り当てます。

〈変数名〉には、変数名として正しい形をしたものを指定します。

〈添字の最大値〉には、配列の要素を示す添字の、最大値を指定します。添字の最小値は、OPTION BASE によって、0 か 1 に指定することができます。実際に割り当てられる配列の要素の数は、OPTION BASE が 1 のときは〈添字の最大値〉に等しく、OPTION BASE が 0 のときは〈添字の最大値〉+1 となります。

なお、OPTION BASE を指定しないで DIM を使用した場合、添字の最小値は 0 となります。

〈添字の最大値〉を複数個指定すると、個数分の次元をもつ配列変数の指定となります。1 行中(255 バイト)に表記できる範囲であれば、次元数はいくつとってもかまいません。

ただし、配列の要素数および次元数は、メモリ容量の制限を受けます。配列変数のとるメモリ領域が大きすぎると、“Out of memory” エラーとなります。

数値配列の場合、1 個の配列に対して指定できる要素の数は、整数型で 32767 個、単精度実数型で 16383 個、倍精度実数型で 8191 個までです。

DIM が実行された時点で、その配列のすべての要素の値は 0(文字型の場合はヌルストリング)に設定されます。

配列変数を消去するには、ERASE を用います。

注意：設定された最大値より大きな値の添字が用いられた場合は、“Subscript out of range” エラーが起こります。

DIM で宣言しなくとも配列変数を用いることができますが、その場合、添字の最大値は 10 まで (OPTION BASE が 0 でも 1 でもかまわない) 使用できます。

参照 : ERASE, OPTION BASE サンプルプログラム 5

DRAW (DISK モード)

機能	グラフィック描画サブコマンド列に従って、ワールド座標上で図形を描きます。
書式	DRAW <文字列式>
文例	DRAW "D50R50U50L50" DRAW "C7T150, 50Z=TILE\$;"

DRAW では、複数のグラフィック描画サブコマンドを使って図形を描きます。<文字列式>内に指定可能なグラフィック描画サブコマンドは、一般的には次のような形式です。

[<修飾子>] <識別子> [<パラメータ 1>] [, <パラメータ 2>] ...

(1) 修飾子

<修飾子>としては、B および N を指定することができます。<修飾子>は <識別子>の補助的な役割を果たすもので、次のような意味をもちます。

B <識別子>で示されるサブコマンドの実行は実際には行わず、LP(最終参照点)のみをサブコマンド実行後の値にします。

N <識別子>で示されるサブコマンド実行後、LP を (0, 0) とします。

なお、B および N を同時に指定した場合は、<識別子>で示されるサブコマンドの実行は行われず、LP が (0, 0) となります。

(2) 識別子とパラメータ

<識別子>は、いろいろな描画機能を表すもので、英字 1 文字のサブコマンドで指定します。

<パラメータ 1>、<パラメータ 2>は、各サブコマンドに対応した数値定数または変数で、以下の形式で指定します。

<数値定数> [;]

または、

= <変数> ;

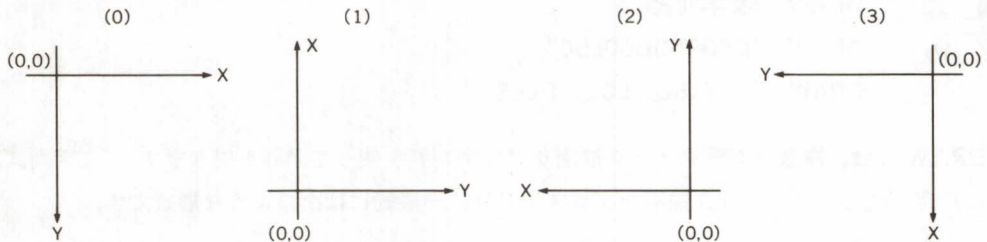
数値定数として指数形式を用いることはできません。また、16進表記で数値定数を表す場合には、その後に必ず、セミコロン(;)をつけなければなりません。

変数を指定する場合は、その直前には等号(=)を、直後にはセミコロン(;)を、それぞれ必ずつけるようにしてください。

〈識別子〉に指定するサブコマンドとその〈パラメータ〉について以下に説明します。

A 〈パラメータ 1〉

グラフィック描画サブコマンドの現座標系を、〈パラメータ 1〉に0~3までの整数値を指定することにより、以下の状態に規定します。



A サブコマンドによって規定された座標系は DRAW のグラフィック描画サブコマンドに対してのみ有効なもので、ワールド座標系が変化するわけではありません。

初期状態では、〈パラメータ 1〉の値は0となっています。

C 〈パラメータ 1〉

グラフィック描画サブコマンドによって描画される図形のパレット番号を規定します。〈パラメータ 1〉にはパレット番号を指定します。C サブコマンドによって規定されたパレット番号は、グラフィック描画サブコマンドに対して有効なものであり、フォアグラウンドカラーを変更するものではありません。

初期状態では、〈パラメータ 1〉の値はフォアグラウンドカラーとなっています。

D 〈パラメータ 1〉

LP から、現座標系の Y 軸に沿って〈パラメータ 1〉×スケール値分だけ正の方向へ移動した点まで直線を描画します。スケール値については、S サブコマンドを参照してください。

E 〈パラメータ 1〉

LP から、現座標系の X 軸に沿って〈パラメータ 1〉×スケール値分だけ正の方向に、さらに Y 軸に沿って〈パラメータ 1〉×スケール値分負の方向に移動した点まで直線を描画します。

F 〈パラメータ 1〉

LP から、現座標系の X 軸に沿って〈パラメータ 1〉×スケール値分だけ負の方向に、さらに Y 軸に沿って〈パラメータ 1〉×スケール値分負の方向に移動した点まで直線を描画します。

G <パラメータ 1>

LP から、現座標系の X 軸に沿って <パラメータ 1>×スケール値分だけ負の方向に、さらに Y 軸に沿って <パラメータ 1>×スケール値分正の方向に移動した点まで直線を描画します。

H <パラメータ 1>

LP から、現座標系の X 軸に沿って <パラメータ 1>×スケール値分だけ正の方向に、さらに Y 軸に沿って <パラメータ 1>×スケール値分正の方向に移動した点まで直線を描画します。

L <パラメータ 1>

LP から、現座標系の X 軸に沿って <パラメータ 1>×スケール値分だけ負の方向へ移動した点まで直線を描画します。

M <パラメータ 1>, <パラメータ 2>

LP から、ワールド座標系の X 座標 = <パラメータ 1>, Y 座標 = <パラメータ 2> の点まで直線を描画します。

P [<パラメータ 1>]

LP を含み、<パラメータ 1>で指定したパレット番号で描かれた境界線で囲まれた領域を、色またはタイルストリングでぬりつぶします。<パラメータ 1>が省略された場合には、C サブコマンドで指定されているパレット番号が採用されます。P サブコマンドを実行するときは、LP はウィンドウ内になければなりません。P サブコマンドによってぬりつぶしを行う際の、色やタイルストリングの指定方法については、Z サブコマンドを参照してください。

Q <パラメータ 1>, <パラメータ 2>

LP から、現座標系の X 軸に沿って <パラメータ 1>×スケール値分、Y 軸に沿って <パラメータ 2>×スケール値分それぞれ正の方向へ移動した点までを対角線とする長方形を描画します。

R <パラメータ 1>

LP から、現座標系の X 軸に沿って <パラメータ 1>×スケール値分正の方向に移動した点まで直線を描画します。

S <パラメータ 1>

グラフィック描画サブコマンドで指定される相対位置のパラメータのスケール値(倍率)を規定します。<パラメータ 1> はスケール値を表し、0 より大きな値でなければなりません。

初期状態では、<パラメータ 1> の値は 1 となっています。

T 〈パラメータ 1〉, 〈パラメータ 2〉

Q サブコマンドと同様にして長方形を描画した後、その内部をぬりつぶします。T サブコマンドによってぬりつぶしを行う際の、色やタイルストリングの指定方法については、Z サブコマンドを参照してください。

U 〈パラメータ 1〉

LP から、現座標系の Y 軸に沿って、〈パラメータ 1〉×スケール値分負の方向に移動した点まで直線を描画します。

W 〈パラメータ 1〉, 〈パラメータ 2〉

LP から、現座標系の X 軸に沿って 〈パラメータ 1〉×スケール値分、Y 軸に沿って 〈パラメータ 2〉×スケール値分それぞれ正の方向へ移動した点まで直線を描画します。

X 〈パラメータ 1〉

〈パラメータ 1〉で指定される文字変数内のグラフィック描画サブコマンド列に従い図形を描画します。文字列変数内に、再び X サブコマンドを含めることも可能です。なお、〈パラメータ 1〉には、必ず"= 〈文字変数〉;"の形を使用してください。

Y 〈パラメータ 1〉

D, E, F, G, H, L, M, Q, R, U および W サブコマンドで描画する直線または長方形のラインスタイルを規定します。〈パラメータ 1〉は、&H0~&HFFFF の範囲の整数でなければなりません。ラインスタイルの内容については、LINE を参照してください。

初期状態では、〈パラメータ 1〉の値は&HFFFF となっています。

Z 〈パラメータ 1〉

T および P サブコマンドでぬりつぶす際の、ぬりつぶしの色またはタイルストリングによる模様を規定します。〈パラメータ 1〉は、パレット番号またはタイルストリングが格納されている文字列変数でなければなりません。ただし、タイルストリングを指定する場合は、必ず"=〈文字変数〉;"の形を使用してください。タイルストリングについては、[2]PAINT を参照してください。

初期状態では、C サブコマンドにより規定されるパレット番号(C サブコマンドを実行していないときはフォアグラウンドカラー)となっています。

D, E, F, G, H, L, M, Q, R, T, U および W サブコマンドの実行では、修飾子として N が指定されていない限り、LP は指定された点に移ります。A, C, P, S, Y および Z サブコマンドの実行では LP は変化しません。

また、A, C, S, Y および Z サブコマンドの指定は、指定のある DRAW だけでなく、以降の DRAW にもその指定が引き継がれます。これらの指定は、SCREEN を実行すると初期状態にもどります。

参照：LINE, [2] PAINT, サンプルプログラム 17

DSKF (DISK モード)

関 数

機能	ディスクに関する情報を得ます。
書式	DSKF(<ドライブ番号> [, <機能>])
文例	PRINT DSKF(1) SECTORS=DSKF(2, 1)

<ドライブ番号> で指定されたディスクに関する情報を得ます。

<機能> は値として得ようとする情報の種類を決定するもので、次のように指定します。

省略時	ドライブの残り容量 (クラスタ単位)
0	最大トラック番号 (=サーフェス当たりのトラック数-1)
1	1トラック当りのセクタ数
2	最大サーフェス番号 (ディスクのサーフェス数-1)
3	1クラスタ当りのトラック数
4	クラスタ総数
5	ディレクトリトラック番号
6	1クラスタ当りのセクタ数
7	FAT の開始セクタ番号
8	FAT の終了セクタ番号
9	FAT の数
10	ディスク属性の入っているセクタ番号

ドライブの種類やディレクトリ, FAT などについては、BASIC ユーザーズマニュアルを参照してください。

- 注意・SCSI 固定ディスクの場合に得られる値は、ディスク中の各ドライブ内の相対的な値です。
- ・従来インタフェース固定ディスクの場合、<機能>省略時に得られる値は、ディスク中のドライブに関する値です。
 - ・5インチ 1D/2D フロッピーディスクの場合、<機能>3 で得られる値は、1トラック当りのクラスタ数です。

参照：DSKI\$, DSKO\$

DSKI\$ (DISK モード)

関 数

機 能	ディスクから直接、データを読み出します。
書 式	DSKI\$(<ドライブ番号>, <サーフェス番号>, <トラック番号>, <セクタ番号>)
文 例	D\$=DSKI\$(2, 1, 19, 1)

通常のファイル操作(OPEN, CLOSE など)とは別に、ディスク上の指定したセクタから直接データを読み出します。セクタ上に書かれているデータ 256 バイトをシステム用バッファに読み出すとともに、最初の 256 バイトの文字列を関数の値として返します。

文字列の長さは 255 バイトまでしか許されないために、関数の値として 256 バイトのデータの全てを得ることはできません。そこで、通常のランダムアクセスの場合と同じように FIELD によってシステム用バッファ(ファイル番号#0)へフィールド変数を割り付けることにより、これを解決することができます。あるいは、VARPTR 関数により、システム用バッファの置かれているメモリ番地を調べた上で、PEEK 関数によりバッファからデータを読み出すこともできます。

なお、システム用バッファは、ファイル番号#0としてあらかじめ用意されているもので、OPEN によって開く必要はありません。

<サーフェス番号>, <トラック番号>, <セクタ番号>として指定できる値の範囲は、指定された<ドライブ番号>のディスクドライブの種類によって異なりますが、BASIC は、これらの値の範囲を自動的に調べ、不当であった場合には“Bad track/sector”エラーとします。

詳しくは BASIC ユーザーズマニュアルを参照してください。

参照：DSKF, DSKO\$, FIELD, VARPTR

DSKO\$ (DISK モード)

機 能	ディスクに対して直接書き込みを行います。
書 式	DSKO\$ <ドライブ番号>, <サーフェス番号>, <トラック番号>, <セクタ番号>
文 例	DSKO\$ 1, 0, 19, 1

通常のファイル操作とは別に、ディスク上の指定したセクタにシステム用バッファの 256 バイトのデータを直接書き込みます。#0のシステム用バッファは、通常のディスクファイル操作の際には用いることはできませんが、DSKI\$と DSKO\$による直接操作の際にのみ、ユーザーが用いることができます。

書き込むデータの準備は、FIELD でファイル番号に#0 を指定することによって (OPEN は不要) システム用バッファへフィールド変数を割り付けた後、LSET, RSET により行います。あるいは、VARPTR 関数により、システム用バッファの置かれているメモリ番地を調べた上で、POKE によりバッファ中にデータを準備することもできます。

〈サーフェス番号〉、〈トラック番号〉、〈セクタ番号〉として指定できる値の範囲は、指定された〈ドライブ番号〉のディスクドライブの種類によって異なりますが、BASIC は、これらの値の範囲を自動的に調べ、不当であった場合には "Bad track/sector" エラーとします。

注意：DSKO\$ は、既存のディスクファイルを壊すおそれがありますので、ディスクおよびファイルの構成を正しく理解した上で用いてください。それぞれのディスクの諸元については DSKF 関数により調べることができます。詳しくは BASIC ユーザーズマニュアルを参照してください。

参照：DSKF, DSKI\$, FIELD, VARPTR

EDIT

機能 指定された行を画面に表示し、以降 キー、 キーなどによるプログラム編集を可能にします。

書式 EDIT 〈行番号〉

文例 EDIT 30

EDIT .

EDIT を実行すると〈行番号〉で指定された行が表示され、カーソルは行の先頭に移動します。以降、 キー、 キー、カーソルキーを始めとする、各種プログラム編集キーを使ったプログラムの編集が可能になります。

〈行番号〉にピリオド(.)を指定すると BASIC インタプリタ内のポインタが示している現在行を指定したことになります。実際には、LIST コマンドで最後に表示された行や、編集機能で最後に修正された行が現在行となります。

注意：P オプションつきでセーブされたファイルをロードして EDIT コマンドにより、内容を表示、変更しようとした場合には、"Illegal function call" エラーとなります。

END

機能 プログラムの終了を宣言します。

書式 END

文例 END

プログラムの実行を終了し、すべてのファイルをクローズしてコマンドレベルにもどります。END はプログラムの実行を終了させたい所に置けばよく、また、いくつ置いてもかまいません。

注意：プログラムの最後の END は省略することができますが、この場合、ファイルのクローズは行われません。

EOF

関数

機能 ファイルの終了コードを調べます。

書式 EOF(<ファイル番号>)

文例 IF EOF(1) THEN CLOSE 1:END

<ファイル番号>で指定されたファイルが終わりに達したかどうかを調べる関数です。終わりに達していれば-1(真)、そうでなければ0(偽)を返します。<ファイル番号>で指定されたファイルは、入力モードでオープンされていなければなりません。

指定されたファイルが RS-232C 回線(COM:)であった場合には、EOF 関数は、バッファが空であったときに値を真とします。

参照：サンプルプログラム 28

ERASE

機能 配列変数を消去します。

書式 ERASE <配列変数名> [, <配列変数名> ...]

文例 ERASE C, D\$

<配列変数名>には、消去したい配列の変数名を指定します。添字や()は不要です。

ERASE を実行すると指定した配列変数が消去されますから、元の配列変数に割り当てられていた分だけメモリエリアが増加します。したがって、同一変数名の配列変数を DIM で新たに宣言したり、他の目的に使用することもできるようになります。

注意：ERASE を実行せずに同一変数名で配列変数を宣言しようとする、"Duplicate Definition" エラーが起こります。

参照：DIM, OPTION BASE, サンプルプログラム 5

ERL/ERR

関 数

機能 エラーの発生した行番号および発生したエラーのエラーコードを保持していません。

書式 1) ERL
2) ERR

文例 IF ERL=100 THEN CLS:RESUME
IF ERR=7 THEN STOP

エラーが発生した時点で、ERL はエラーの発生した行番号を、ERR はエラーコードを保持しています。

一般に、ERR および ERL は ON ERROR GOTO によって指定したエラー処理ルーチンにおいて、独自のエラー処理を行うために使用します。

参照：ERROR, ON ERROR GOTO, サンプルプログラム 22

ERROR

機能 エラー発生時のシミュレート、エラーコードのユーザー定義を行います。

書式 ERROR 〈整数表記〉

文例 ERROR 200
ERROR A%

〈整数表記〉には、0 から 255 までの範囲の数を指定します。

この値が、すでに BASIC でエラーコードとして使われている場合は、エラーメッセージを表示するとともに、プログラムの実行を停止します。エラーコードが定義されていない場合は、プログラムの実行を停止します。

どちらの場合も、エラー発生時のプログラム行番号は ERL に、エラーコードは ERR にセットされます。したがって、ON ERROR GOTO によるエラー処理ルーチンを利用して、独自のエラー処理を行うことができます。

参照：ERL/ERR, ON ERROR GOTO, 付録 A. エラーメッセージとその対策, サンプルプログラム 22

EXP

関 数

機 能 e(自然対数の底)に対する指数関数の値を得ます。**書 式** EXP(<数式>)**文 例** E=EXP(1)

PRINT EXP(A/2)

<数式> の, e に対する指数演算の結果を値として得ます。

<数式>に倍精度実数が含まれる場合, 得られる値は倍精度となりますが, 他の場合には単精度となります。

FIELD(DISK モード)

機 能 ランダムファイルバッファにフィールド変数を割り当てます。**書 式** FIELD [#] <ファイル番号>, <フィールド幅> AS <文字変数> [, <フィールド幅> AS <文字変数> ...]**文 例** FIELD #1, 128 AS A\$, 64 AS B\$, 64 AS C\$

FIELD は, OPEN で設定されたランダムファイルバッファに対して, 入出力を行うためのフィールドの名前, およびその長さを割り当てます。したがって, FIELD の実行にさきだって, OPEN によりファイルをランダムモードでオープンしておかなくてはなりません。

<文字変数>には, 割り当てたいフィールドの名前を文字型の変数名として指定します。この変数は, バッファにデータを代入したり, 代入したデータを参照するのに用いられ, フィールド変数と呼ばれます。

<ファイル番号>は, OPEN によってファイルバッファをオープンしたときに指定した番号です。

<フィールド幅> は, <文字変数> で指定したフィールド変数に割り当てる文字数で, バイト数で指定します。1つのバッファには, 文字数合計が256バイトまでならば, いくつのフィールド変数を割り当ててもかまいません。

フィールド変数へ代入する(LSET あるいは RSET で行う)データの長さは, フィールド幅を超えてはなりません。

数値データをフィールドにセットする場合, 数値データの型に応じてMKI\$, MKS\$, MKD\$の各関数を用いて文字型化してから LSET/RSET を行います。このとき, 整数型の場合は2バイト長, 単精度実数型の場合は4バイト長, 倍精度実数型の場合は8バイト長にそれぞれ変換されますので, <フィールド幅> もこれに合わせて設定する必要があります。

2バイト系日本語文字列をフィールドにセットする場合, 日本語1文字あたり2バイト分の

フィールドを必要とし、また日本語文字列の前後にシフトコード(KIコードおよびKOコードそれぞれ2バイト分)が挿入されます。〈フィールド幅〉の設定を行うときにはこのことを考慮するようにしてください。たとえば、

LSET A\$="日本語"

のためには最低10バイト分のフィールドを必要としますので、フィールド幅には10以上を設定しなくてはなりません。

なお、1つのバッファに対して、異なった形式で複数のFIELDを実行してもかまいません。

注意：各フィールド変数への値の代入を通常の代入文あるいはINPUTなどで行わないでください。もし行くと、その変数は一般の文字変数領域に登録されてしまうため、FIELDでのバッファ割り当てが無効となり、正しいファイルの入出力が行われなくなります。

参照：GET, LSET/RSET, MKI\$/MKS\$/MKD\$, OPEN, PUT, サンプルプログラム
1, 2, 26, 27, 29

FILES/LFILES (DISK モード)

機能 ディスクに入っているファイルの名前、種類、大きさを出力します。

書式 1)FILES [〈ドライブ番号〉]
2)LFILES [〈ドライブ番号〉]

文例 1)FILES 2
2)LFILES

〈ドライブ番号〉で指定されたドライブ中のファイルの名前、種類、大きさを表示します。書式1の場合ディスプレイに、書式2の場合プリンタに出力します。ファイルの大きさは、クラスタという単位で表示されます(クラスタについては、BASIC ユーザーズマニュアル参照)。〈ドライブ番号〉が省略された場合には、ドライブ1が選択されます。

ファイルの種類は、出力されるファイルの名前と、拡張子との間の区切り文字によって以下のように表されます。

区切り記号が空白：アスキーセーブされたプログラムファイル

または、OPENによって作られたBASICのデータファイル

ピリオド(.)：バイナリセーブされたプログラムファイル

アスタリスク(*)：BSAVEされた機械語ファイル

参照：BSAVE, SAVE

FIX

関 数

機 能 数値の整数部を得ます。**書 式** FIX(〈数式〉)**文 例** F=FIX(B/3)

〈数式〉の値の小数点以下を取り去った値を得ます。

FIX と INT の違いは、〈数式〉の値が負のときに、INT は〈数式〉の値を超えない最大の整数を返すのに対し、FIX はたんに小数点部分だけを取り去った値を返すことです。

参照：INT

FOR...TO...STEP~NEXT

機 能 FOR から NEXT までの区間中にある一連の命令を、繰り返して実行します。**書 式** FOR 〈変数名〉= 〈初期値〉 TO 〈終値〉 [STEP 〈増分〉]

}

NEXT [〈変数名〉] [, 〈変数名〉…]

文 例 FOR J=0 TO 100 STEP 2

FOR K=10 TO 0 STEP -1

}

NEXT K, J

FOR~NEXT ループ中(FOR と NEXT の間)に置かれた命令を、FOR 中で指定した条件に従って繰り返して実行します。

〈変数名〉で指定される変数(ループ変数と呼ぶ)は、整数型または単精度型でなくてはなりません。

〈初期値〉には、変数の初期値を設定します。〈終値〉には、変数の最終値を設定します。〈増分〉には、初期値と最終値との間の増分を指定します。

文例を例にとると、はじめ J=0 が設定され、FOR 以降の命令を実行します。プログラムの実行が NEXT まで来ると、J の値が増分の 2 だけ増やされて J=2 となり、再び、FOR 以下が実行されます。J=100 となるまで FOR と NEXT の間の処理が繰り返されます。

STEP 〈増分〉が省略された場合には、〈増分〉は 1 とみなされます。

次の場合には、FOR~NEXT は実行されずに、NEXT の次へ実行が移ります。

(1) 〈増分〉が正の値で、〈初期値〉が〈終値〉より大きい場合。

(2) <増分> が負の値で、<初期値> が <終値> より小さい場合。

ただし、<変数> には <初期値> が代入されます。

1つの FOR~NEXT の中にはもう1つの FOR~NEXT を置くことができます(入れ子構造、ネスト構造などと呼ぶ)。この場合、それぞれの <変数名> には別のものを使わなければなりません。また、このとき、1つの FOR~NEXT は完全に他の FOR~NEXT の内部になければなりません。

注意：FOR と NEXT は必ず1対1に対応していなければなりません。

また、FOR~NEXT ループ内へ外部から GOTO などジャンプして入ってきたり、逆にループ内から外部へジャンプしたりするようなプログラムは、その動作が保証されなくなります。

参照：サンプルプログラム 6

FPOS

関 数

機 能 ファイル中での物理的な現在位置を得ます。

書 式 FPOS(<ファイル番号>)

文 例 HEAD=FPOS(4)

<ファイル番号> によって指定されたファイルが、最後に読み書きした位置を得ます。

指定されたファイルがディスクファイルの場合は、最後に読み書きしたセクタの番号が得られます。この番号はサーフェス0、トラック0、セクタ1を0番とした通し番号となります。

指定されたファイルがプリンタの場合には、プリンタのヘッドの位置が得られます。この値は LPOS が返す値と同一です。

参照：LOC, LPOS

FRE

関 数

機 能 メモリの未使用領域の大きさを得ます。

書 式 FRE(<機能>)

文 例 PRINT FRE(0)

BASIC の利用可能なメモリ領域のうちの、各種の未使用領域の大きさを関数値として得ます。

〈機能〉に指定する値(整数値)とそれぞれの場合に得られる関数値の意味は次のとおりです。

- 0 : 未使用変数領域のバイト数
(単純変数およびストリング領域として使用可能な領域の残りバイト数)
- 1 : 未使用テキスト領域のバイト数
(プログラムテキストを入れるための領域の残りバイト数)
- 2 : 未使用変数領域と未使用テキスト領域の合計バイト数
- 3 : 配列データセグメントの未使用領域のバイト数

〈機能〉が0または2のときは、FREは、ストリング領域の整理のための“ちり集め”処理を必ず行いますので、関数が値を返すまでに時間がかかることがあります。

GET

機能	ファイル中のデータをファイルバッファに読み込みます。
書式	GET [#] <ファイル番号> [, <数値>]
文例	GET #3, 5 GET 1

〈ファイル番号〉で指定されたファイル中のデータを、対応するバッファ中に読み込みます。GETは、指定されたファイルがディスクファイルか、キーボードファイルかによってその動作が異なります。

(1) ディスクファイルの場合

ランダムファイルからのデータをランダムファイルバッファに読み込みます。指定されたディスクファイルはランダムモードでオープンされていなければなりません。

〈数値〉はファイルのレコード番号として解釈され、指定されたレコードがバッファに読み込まれます。レコード番号の最小値は1、最大値は65000です。〈数値〉が省略された場合には、直前に行われた、GET、PUTで指定されたレコードの次のレコードが読み込まれます。

なお、GETを行った後で、INPUT #、LINE INPUT #などのシーケンシャル・アクセスを行った場合には、バッファに入っているデータから入力が行われますので注意してください。

(2) キーボードファイル(KYBD:)の場合

キーボードから入力した文字をバッファ中に読み込みます。キーボードファイルは入力モードでオープンされていなければなりません。

〈数値〉は、キーボードからバッファに読み込む文字(バイト)数と解釈されます。0から255までの値で指定します。〈数値〉が省略されたとき、および0が指定されたときには256文字を

読み込みます。

GET はバッファへの読み込みを行うものです。読み込んだデータは FIELD で指定した変数 (フィールド変数) で参照することができます。

参照 : FIELD, OPEN, PUT, サンプルプログラム 2, 27, 29

GET@

機能	画面上のグラフィックパターンを配列変数に読み込みます。
書式	GET[@] (Sx1, Sy1) - (Sx2, Sy2) , <配列変数名> [(<添字 >)] STEP(x, y)
文例	GET (100, 50) - (130, 70), G%

画面上のスクリーン座標の 2 点 (Sx1, Sy1) と (Sx2, Sy2) を対角点とする四角形のグラフィックパターンを、<配列変数名> で指定された配列変数に読み込みます。

座標の指定は、ビューポート内に展開されるスクリーン座標を使用します (PUT@ も同様)。2 つ目の座標指定には、STEP を使って相対座標による指定もできます。

<配列変数名> は、グラフィックデータを読み込むために用意する数値型の配列変数の名前です。

<添字> は、グラフィックパターンを配列変数に読み込む際に、どの要素から格納し始めるかを指定するものです。省略した場合は配列の最初から格納します。<添字> を使うことにより、1 つの配列変数に対して複数のグラフィックパターンを格納することができます。

GET@ を実行する前に、DIM で配列変数の大きさを宣言しておかなければなりません。確保すべき配列変数の大きさは、1 つの配列に 1 つのパターンしか読み込まない場合、次の計算式で求めることができます。複数パターンの場合は、それぞれのパターンについて計算し、その合計分を確保します。

$$\text{必要なバイト数} = ((\text{横のドット数} + 7) \times 8) * \text{縦のドット数} * M + 4$$

ただし、白黒モードのとき ——— M=1

8 色カラーモードのとき ——— M=3

16 色カラーモードのとき ——— M=4

$$\text{添字の値} = (\text{必要なバイト数} + (N - 1)) \div N - 1$$

N は配列変数の型によって変わります。

整数型配列 ——— N=2

単精度型配列 ——— N=4

倍精度型配列 ——— N=8

この計算式によって求めた“添字の値”を、DIMで配列変数を宣言するときに〈添字の最大値〉(DIM参照)として指定します。この計算式で求められる添字の値は、1次元配列で添字の最小値が0のとき(OPTION BASE参照)のもので、一般にGET@では1次元の配列変数を用います。

注意：このGET@はPUT@と密接な関係にあり、通常ペアで使用されます。また、GET@実行後、LP(最終参照点)は(Sx2, Sy2)に移されます。

参照：DIM, OPTION BASE, PUT@, サンプルプログラム 18

GOSUB

機能 サブルーチンを呼び出します。

書式 GOSUB 〈行番号〉

文例 GOSUB *SUB1

GOSUB 550

サブルーチンとは、他から独立したプログラムで、RETURNで終わっているものをいいます。GOSUBは、指定した〈行番号〉から始まるサブルーチンを呼び出します。

なお、〈行番号〉の代わりにラベル名を使うこともできます。

1つのサブルーチンの中から他のサブルーチンを呼び出すこともできます。これを、サブルーチンの多重化と呼びます。この多重化は、メモリのスタック領域(CLEAR参照)の容量が許す限り行うことができます。

注意：サブルーチンを多重化してスタック領域が足りなくなると、“Out of memory”エラーが発生します。

参照：CLEAR, RETURN, サンプルプログラム 7, 30, 31, 33

GOTO/GO TO

機能 指定された行へジャンプし、そこからプログラムを実行します。

書式 1) GOTO 〈行番号〉

2) GO TO 〈行番号〉

文例 GOTO 500

GOTO *EXIT

〈行番号〉の行へジャンプして、プログラムの実行を継続します。

〈行番号〉の代わりにラベル名を使うこともできます。

注意：“GO”と“TO”の間にスペース1個を入れても、まったく同じ意味です。しかし2個以上の場合には、GOTOとは解釈されません。

参照：サンプルプログラム 7, 28, 29, 30

HELP ON/OFF/STOP

機能 HELPキーによる割り込みの許可、禁止、停止を制御します。

書式

- 1) HELP ON
- 2) HELP OFF
- 3) HELP STOP

文例 HELP OFF

HELPキーを押したことによる割り込みの許可、禁止、停止を宣言します。

書式1) 割り込みを許可します。以後HELPキーを押すごとに割り込みが発生し、ON HELP GOSUBによって定義された処理ルーチンに分岐します。

書式2) 割り込みを禁止します。以後HELPキーを押しても処理ルーチンへの分岐は起こりません(HELPキーは通常の動作になります)。

書式3) 割り込みを停止します。以後HELPキーを押しても押されたことを覚えているだけで、処理ルーチンへの分岐は起こりません。しかし、その後、HELP ONによって割り込みが許可されると、前にHELPキーを押したことによって、処理ルーチンに分岐します。

参照：ON HELP GOSUB

HEX\$

関数

機能 10進数を16進数に変換し、その文字列を得ます。**書式** HEX\$(**<数式>**)**文例** A\$=HEX\$(X)

PRINT HEX\$(255)

<数式>の値を16進数に変換して、その文字列を得る関数です。**<数式>**の値は、-32768から32767(または0から65535)までの範囲になければなりません。**<数式>**の値と得られる文字列との対応は次のとおりです。

<数式> の値	得られる16進表記文字列
0~32767	: 0~7FFF
-32768~-1	: 8000~FFFF
32768~65535	: 8000~FFFF

参照: OCT\$, VAL, サンプルプログラム 24

IF...THEN~ELSE/IF...GOTO~ELSE

機能 論理式の条件判断を行います。

書式

```

1) IF <論理式> THEN | <文> | [ELSE | <文> | ]
                       | <行番号> | | <行番号> |
2) IF <論理式> GOTO <行番号> [ELSE | <文> | ]
                               | <行番号> |

```

文例

```

IF A$="Y" THEN *START ELSE *EXIT
IF SS GOTO 200

```

<論理式>の条件によってプログラムの実行を制御します。

すなわち、**<論理式>**が真(0以外)ならば THEN あるいは GOTO 以降を実行します。**<論理式>**が偽(0)ならば ELSE 以降が実行します。もし、ELSE 以降が省略されている場合には、次の行が実行されます。

IF...THEN(GOTO)~ELSEにおいて、ELSEに続けて別のIFを置いて多重にすることができます。ただし、多重にできるのは1行(255バイト)に書ける範囲に限られます。

注意：〈論理式〉が変数または定数の場合には、値が0以外のときに THEN(または GOTO)以降が、0のときに ELSE 以降あるいは次の行が実行されます。

参照：サンプルプログラム 8

INKEY\$

関 数

機 能 押されているキーの文字を得ます。

書 式 INKEY\$

文 例 IF INKEY\$="" THEN *WAITING

押されているキーの文字(1文字)を取り出します。もしキーが押されていなければ INKEY\$の値はヌルストリング(空の文字列)となります。

キーボードバッファにすでに入力済みの文字が入っている場合には、バッファの先頭の1文字を取り出します。

INKEY\$では、押されたキーは画面上には表示されません。また、INKEY\$は、INPUT や INPUT\$などとは違い、キー入力待ちを行いません。

注意：CTRL + C, STOP キーは INKEY\$で取り出すことはできません。

参照：サンプルプログラム 29

INP

関 数

機 能 入力ポートから値を読み取ります。

書 式 INP(〈ポート番号〉)

文 例 A=INP(15)

〈ポート番号〉で指定された入力ポートから8ビットのデータを読み取り、それを関数値とします。〈ポート番号〉には0~32767(&H0000~&H7FFF)の値を指定します。

参照：OUT

INPUT

機能	キーボードから入力されたデータを、変数に代入します。
書式	INPUT [<プロンプト文> ;] <変数> [, <変数> ...]
文例	INPUT A INPUT "住所"; B\$

INPUT が実行されると、プログラムはキーボードからの入力待ちの状態となります。

<プロンプト文>には、データの入力を受ける前に画面に表示されるメッセージを指定します。**<プロンプト文>**にセミコロン(;)が続いた場合には、さらにクエッションマーク(?)と1桁のスペースが画面に表示されます。コンマ(,)が続いた場合には**<プロンプト文>**の後には何も表示されません。

<変数>には、キーボードから入力したデータが代入されます。**<変数>**はコンマ(,)で区切って32個まで指定することができます。**<変数>**を複数個指定した場合には、入力するデータもコンマで区切って**<変数>**の数だけ入力しなければなりません。また、対応する**<変数>**とデータの型とは一致していなければなりません。

何も入力しないでリターンキーだけを押した場合には、0 やヌルストリング(空の文字列)が入力されたとみなされます。この方法で0 やヌルストリングの入力ができますが、この場合も**<変数>**が複数の場合、必要数のコンマだけは入力しなければなりません。

文字変数に文字列を代入する場合、コンマや文字列の前後に意味のある空白を入力するには、ダブルクォーテーション(")で文字列を囲む必要があります。この場合ダブルクォーテーションは、文字として入力されません。これ以外の場合には、文字列をダブルクォーテーションで囲む必要はありません。

入力を中断したい場合、**STOP** キーあるいは**CTRL**+**C** キーを入力します。プログラムの実行中に入力を中断すると、**STOP ON** の状態にない限り、プログラムの実行も中断され、コマンドモードにもどります。このとき、それまでに入力した値はすべて無効となります(変数には値が代入されません)。

日本語の入力を行う場合には、**CTRL**+**XFER** を入力して日本語入力モードにします(日本語入力ガイド参照)。

注意：**<変数>**と代入する値の型が違っていた場合には"? Redo from start"と画面表示して再び入力待ちとなります。

また、割り込み機能を使用しているプログラム中で、INPUT の実行中に割り込み動作が行われた場合、割り込み処理ルーチンから復帰したときに INPUT の次の命令に実行が移ってしまいますので、注意が必要です。

参照：INPUT WAIT, KINPUT, LINE INPUT, STOP ON/OFF/STOP, サンプルプログラム 1, 7, 8, 9, 22, 28, 29, 39

INPUT

機能	シーケンシャルファイルからデータを読み込み、変数に代入します。
書式	INPUT #<ファイル番号>, <変数名> [, <変数名> …]
文例	INPUT # 1, A, B INPUT # 2, NAMAES

<ファイル番号>には、OPEN によって、そのファイルをオープンしたときに指定した番号を使います。

<変数名> は32個まで指定することができます。<変数名> の型は、ファイルから読み込まれるデータの型と対応していなければなりません。

ファイルに書き込まれているデータは、それが数値変数に代入される値(数字の並び)ならば、空白、コンマあるいはキャリッジリターン(CHR\$(13))で区切られていなければなりません。また、データが文字変数に代入される値(任意の文字の並び)ならば、コンマあるいはキャリッジリターンで区切られていなければなりません。

参照: OPEN, サンプルプログラム 28

INPUT\$

関数

機能	指定されたファイルより指定された長さの文字列を読み込みます。
書式	INPUT\$(<文字数> [, [#] <ファイル番号>))
文例	WORD\$=INPUT\$(6, #2)

<ファイル番号>には、OPEN によってそのファイルをオープンしたときに指定した番号を使います。<ファイル番号> が省略された場合には、キーボードからの入力を読み込みますが、INPUT と異なり入力された文字は画面には表示されません。

<文字数> には、読み込みたい文字列の長さをバイト数で指定します。

INPUT\$は指定された <文字数> の文字が入力されるのを待ち続けますが、すでにバッファに入力済みのデータがある場合には、バッファ中から文字を読み込みます。

また、INPUT\$は、**STOP** キー、**CTRL**+**C**を除くすべての文字をそのまま読み込みますので、INPUT や LINE INPUT では入力することのできないキャリッジリターン(CHR\$(13))なども入力することができます。

参照: INPUT, LINE INPUT, サンプルプログラム 22, 29

INPUT WAIT

機能 キーボードから入力されたデータを、変数に代入します。その際、入力待ち時間を制限することができます。

書式 INPUT WAIT <待ち時間>, [<プロンプト文> ; |]<変数名>[, <変数名>…]

文例 INPUT WAIT 100, "Your name"; NA\$
INPUT WAIT 50, B\$

INPUT WAIT は待ち時間に制限があることを除けば、INPUT と同様に機能します。

<待ち時間> で指定された時間だけキーボードからの入力を待ちます。<待ち時間> の単位は 0.1 秒です。入力が行われなかったときは、次の行へ実行が移ります。

注意：この文の後にマルチステートメントとして他の文を続けた場合、制限時間内に入力が行われたときは後の文が実行されますが、入力が行われなかったときは後の文は無視されて次の行へ実行が移りますから、注意してください。

参照：INPUT

INSTR

関数

機能 文字列の中から指定文字列を捜して、その文字の位置を得ます。

書式 INSTR([<位置>], <文字列 1>, <文字列 2>)

文例 J=INSTR(A\$, "J")
PRINT INSTR("ONETWOTHREE", "TWO")

<文字列 1> の中から <文字列 2> が捜され、見つければその位置が、見つからなければ 0 が、それぞれ値として得られます。

<位置> には、<文字列 1> 中で捜し始める位置を文字数(整数値)で指定します。<位置> が省略された場合は 1 が指定されたものとみなされ、<文字列 1> の最初から捜し始めます。

<文字列 2> に "" (ヌルストリング) を指定すると、INSTR の値は <位置> と同じになります。

注意：INSTR は、2 バイト系日本語文字列に対しては使用できません。KINSTR を使用してください。

INT

関 数

機 能 小数点以下を切り捨てた整数値を得ます。

書 式 INT(<数式>)

文 例 PRINT INT(1.123)

A=INT(RND * 255)

<数式> の値を超えない最大の整数を得ます。

INT と FIX の違いは、<数式> の値が負のときに、FIX はたんに小数点部分だけを取り去った値を返すのに対し、INT は <数式> の値を超えない最大の整数を返すことです。

参照：FIX, CINT

JIS\$ (DISK モード)

関 数

機 能 2 バイト系日本語文字の漢字コードを得ます。

書 式 JIS\$(<文字列>)

文 例 C\$=JIS\$(KMID\$("漢字", 2, 1))

<文字列> の最初の 2 バイトを 16 進表記 4 桁の文字コードの文字列に変換します。

<文字列> の最初に 2 バイト系日本語文字を指定した場合には、日本語文字の前に KI コードが入っているため(画面には表示されない)、KI コード(&H1B4B)が得られます。したがって 2 バイト系日本語文字のコードを得るためには、KMID\$関数などを併用しなければなりません。

<文字列> の最初の 2 バイトが 1 バイト系文字 2 つの場合は、2 つの文字の 16 進数 2 桁のキャラクタコードが連続して得られます。

<文字列> が 2 バイト以下の場合には、“Illegal function call” エラーになります。

参照：KNJ\$, サンプルプログラム 36

KACNV\$ (DISK モード)

関 数

機 能 2 バイト系全角文字を、対応する 1 バイト系の英数カナ文字に変換します。

書 式 KACNV\$(<文字列>)

文 例 A\$=KACNV\$(B\$)

PRINT KACNV\$("アイウABC")

〈文字列〉中の 2 バイト系全角文字を 1 バイト系英数カナ文字に変換します。なお、このとき 2 バイト系全角文字列中の KI/KO コードはすべて取り除かれます。

なお、〈文字列〉中に、対応する 1 バイト系英数カナ文字のない 2 バイト系全角文字が含まれていると、"Illegal function call" エラーとなります。

注意：〈文字列〉中の 2 バイト系半角文字は正しく変換されません。

参照：AKCNV\$, サンプルプログラム 35

KEXT\$ (DISK モード)

関 数

機 能 文字列の中から 1 バイト系英数カナ文字だけ、あるいは、2 バイト系日本語文字だけのどちらかを抜き出します。

書 式 KEXT\$(〈文字列〉, 〈機能〉)

文 例 A\$=KEXT\$(B\$, 0)

D\$=CHR\$(27)+"K"+KEXT\$("漢字 SAMPLE", 1)+CHR\$(27)+"H"

〈機能〉に指定する値により、特定のタイプの文字列だけを抜き出します。

- 0 : 1 バイト系英数カナ文字だけを抜き出す。
- 1 : 2 バイト系日本語文字だけを抜き出す。ただし、KI/KO コードは抜き出されな
い。

〈機能〉が 1 の場合、KI/KO コードが抜き出されませんので、得られた 2 バイト系日本語文字列を実際に使用する際には、その前後に KI/KO コードを付加しなくてはなりません。

指定したタイプの文字列がない場合、KEXT\$の値はヌルストリング(空の文字列)となります。

参照：サンプルプログラム 37

KEY

機 能 キーボードの上部にあるファンクションキーに文字列を定義します。

書 式 KEY 〈キー番号〉, 〈文字列〉

文 例 KEY 1, "Yes"+CHR\$(13)

KEY 3, CHR\$(&H22)+"BASIC"+CHR\$(&H22)

〈キー番号〉には、10個までのファンクションキーの番号を指定します。たとえば、**f•1**キーならば1を、**f•2**キーならば2を指定します。

〈文字列〉には、それぞれのキーに記憶させておく文字列を指定します。

この命令を実行すると、以後、各ファンクションキーを押すたびに、それぞれに定義した文字列を入力できます。

各ファンクションキーには、最大15バイトまでの文字列およびコントロール文字を定義できます。キーボードから直接入力できない文字は、CHR\$関数を使って定義します。いちど定義したファンクションキーの内容は、新たに定義し直すかリセットするまで変わりません。

注意：ファンクションキーに2バイト系日本語文字列を定義することはできません。

参照：CHR\$, KEY LIST

KEY LIST

機能 ファンクションキーの内容を画面に表示します。

書式 KEY LIST

文例 KEY LIST

ファンクションキーの内容の一部は画面の最下行に、表示させておくことができますが、KEY LISTを使えば、そのすべての内容を画面に表示させることができます。

参照：CONSOLE, KEY

KEY ON/OFF/STOP

機能 ファンクションキーによる割り込みの許可、禁止、停止を定義します。

書式 1)KEY[(〈キー番号〉)] ON

2)KEY[(〈キー番号〉)] OFF

3)KEY[(〈キー番号〉)] STOP

文例 KEY ON

KEY(3) ON

〈キー番号〉には、1から10までのファンクションキーの番号を指定します。(〈キー番号〉)を省略(カッコも含む)した場合には、すべてのファンクションキーを指定したことになります。

書式1) 割り込みを許可します。以後指定されたファンクションキーを押すごとに割り込みが発

生し、ON KEY GOSUB によって定義された処理ルーチンに分岐します。

書式2) 割り込みを禁止します。以後指定されたファンクションキーを押しても処理ルーチンへの分岐は起こりません(ファンクションキーの通常の動作になります)。

書式3) 割り込みを停止します。以後指定されたファンクションキーを押しても押されたことを覚えているだけで、処理ルーチンへの分岐は起こりません。しかし、その後 KEY(n) ON によって割り込みが許可されると、前にファンクションキーを押したことが有効となり、処理ルーチンに分岐します。

注意：プログラムの終了時には KEY OFF を実行しておいてください。さもないと本来のファンクションキーの機能が失われます。

参照：ON KEY GOSUB, サンプルプログラム 30

KILL (DISK モード)

機能 ディスク上のファイルを削除します。

書式 KILL <ファイルディスクリプタ>

文例 KILL "2 : DATA1"

<ファイルディスクリプタ> で指定したファイルをディスクから削除します。

削除するファイルはクローズ状態になっていなければなりません。また、1つの KILL では1つのファイルしか削除できません。KILL はすべてのディスクファイルについて使用できます。

注意：オープン状態のファイルを削除しようとするとき "File already open" エラーとなります。

KINPUT

機能 キーボードから入力された2バイト系日本語文字を、文字変数に代入します。

書式 KINPUT <変数名>

文例 KINPUT A\$

<変数名> には、文字変数を指定します。

KINPUT が実行されると、自動的に日本語入力モードに入り、キーボードからの入力待ちの状態になります。日本語の入力方法に関しては、日本語入力ガイドを参照してください。

注意：KINPUT は、他の INPUT 関連の命令と異なり、1つの文では1つの変数しか指定できません。KINPUT での入力は偶数桁からになりますので、LOCATE で奇数桁を指定した場合は、その桁数に1を加えた桁が指定されたものと解釈されます。

KINSTR (DISK モード)

関 数

機 能 2 バイト系日本語文字を含む文字列の中から指定文字列を捜して、その文字の位置を得ます。

書 式 KINSTR([<位置>], <文字列 1>, <文字列 2>)

文 例 PRINT KINSTR(3, A\$, "番号")
 NUM=KINSTR("東京名古屋京都大阪", STA\$)

<文字列 1> の中から <文字列 2> が捜され、見つければその位置が、見つからなければ 0 が、それぞれ値として得られます。

<位置> には、<文字列 1> 中で捜し始める位置を、文字数(整数値)で指定します。このとき、2 バイト系日本語も 1 文字として、また、KI/KO コードも文字数に含めて数えます。<位置> が省略された場合は 1 が指定されたものとみなされ、<文字列 1> の最初から捜し始めます。

<文字列 2> に""(マルチストリング)を指定すると、KINSTR の値は<位置>と同じになります。

KLEN (DISK モード)

関 数

機 能 2 バイト系日本語文字を含む文字列中の、特定タイプの文字の合計数を得ます。

書 式 KLEN(<文字列> [, <機能>])

文 例 PRINT KLEN(A\$, 1)
 LN=KLEN("東京 STATION")

<機能> に指定する値(整数値)により、特定のタイプの文字の合計数を得ます。

- 0 : 全体の文字数(2 バイト系日本語文字も 1 文字として、KI/KO コードも 1 文字として数える)
- 1 : 1 バイト系英数カナ文字の文字数
- 2 : 2 バイト系日本語文字の文字数(KI/KO コードは含まない)
- 3 : 2 バイト系全角文字の文字数
- 4 : 2 バイト系半角文字の文字数
- 5 : KI/KO コードの数

〈機能〉を省略した場合は0と解釈されます。

なお、2バイト系日本語文字のうち、全角の文字を2バイト系全角文字といい、半角のものを2バイト系半角文字といいます。

参照：サンプルプログラム 38

KMID\$ (DISK モード)

関 数

機 能 2バイト系日本語文字を含む文字列の中から、任意の長さの文字列を抜き出します。

書 式 KMID\$(<文字列>, <式1> [, <式2>])

文 例 K\$=KMID\$(A\$, 4, 3)
PRINT KMID\$("JAPAN 日本 USA 米国", 1, 5)

〈文字列〉中の、〈式1〉番目の文字から始まる〈式2〉文字の長さの文字列を得ます。ここで、〈式1〉、〈式2〉に指定する文字数の単位は、1バイト系英数カナ文字ならば1バイトを1文字として、2バイト系日本語文字およびKI/KOコードならば2バイトを1文字として、それぞれ数えます。

〈文字列〉の文字数が〈式1〉より小さい場合、KMID\$の値はヌルストリング(空の文字列)となります。

〈式2〉を省略した場合や、〈文字列〉中の〈式1〉番目の文字から右の文字数が〈式2〉より小さい場合は、〈文字列〉中の〈式1〉番目の文字から始まる右のすべての文字列が結果として得られます。

参照：サンプルプログラム 35, 36

KNJ\$ (DISK モード)

関 数

機 能 漢字コード文字列4桁を2バイト系日本語文字1文字に変換します。

書 式 KNJ\$(<文字列>)

文 例 PRINT KNJ\$("1B4B")+KNJ\$("3441")+KNJ\$("1B48")

〈文字列〉に指定した漢字コード4桁を、2バイト系日本語文字1文字に変換します。このとき、〈文字列〉の内容は次の条件をすべて満たしてはなりません。1つでも満たされていないと、“Illegal function call”エラーになります。

- (1) 文字列が4桁以上であること
- (2) 最初の4桁の文字が、いずれも16進数(0~F)の範囲内の文字であること
- (3) 4桁の文字の組み合わせが、漢字コードの範囲内あるいはKI/KOコードであること

〈文字列〉が4桁を超える場合、最初の4桁のみが用いられます。

なお、得られた2バイト系日本語文字を実際に使用する際には、その前後にKI/KOコードを付加しなくてはなりません。

参照：JIS\$, サンプルプログラム 34, 36

KPLOAD (DISK モード)

機能 利用者定義文字パターンをシステムに登録します。

書式 KPLOAD 〈漢字コード〉, 〈整数型配列名〉

文例 KPLOAD &H762A, CHRPTN%

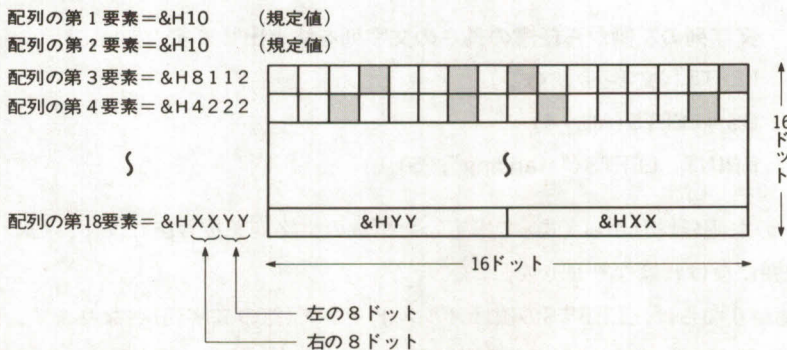
ユーザーが作字した利用者定義文字のパターンをシステムに登録する命令です。

〈漢字コード〉には登録するコードを指定します。指定可能な値の範囲は、&H7621~&H767E および&H7721~&H777E までです。指定値がこの範囲にないときは、"Illegal function call" エラーとなります。

〈整数型配列名〉には文字パターンの格納された配列変数を指定します。〈整数型配列名〉に指定する配列変数は、あらかじめDIMにより、1次元18要素の整数型配列として宣言しておいてください。

配列内に格納されている文字パターンは次のような形式でなければなりません。

配列の第3要素から第18要素までに、次のように実際の文字パターンのドットイメージを格納してください。



参照：サンプルプログラム 34

KTYPE (DISK モード)

関 数

機 能	2 バイト系日本語文字を含む文字列中の、指定位置の文字のタイプを得ます。
書 式	KTYPE(<文字列>, <式>)
文 例	B=KTYPE(A\$, 5) PRINT KTYPE("日本語", 2)

<文字列> 中の <式> 番目の文字のタイプを得ます。

<式> には、タイプを得たい文字の位置を、<文字列> の先頭から数えた文字数で指定します。この場合、1 バイト系英数カナ文字は1 バイトを1 文字と数えますが、その他の文字(KI/KO コードを含む) はすべて2 バイトを1 文字と数えます。

得られる値とタイプとの関係は次のとおりです。

得られる値	タイプ
0	: 1 バイト系英数カナ文字
1	: 2 バイト系全角文字
2	: 2 バイト系半角文字
3	: KI コード
4	: KO コード

<式> の値が0 または <文字列> の文字数より大きい場合、あるいは <文字列> がヌルストリング(空の文字列) の場合は、"Illegal function call" エラーとなります。

参照：サンプルプログラム 38

LEFT\$

関 数

機 能	文字列の左側から任意の長さの文字列を抜き出します。
書 式	LEFT\$(<文字列>, <式>)
文 例	B\$=LEFT\$(A\$, 4) PRINT LEFT\$("standing", 5)

<文字列> の左側(最初) から <式> で指定した桁数の文字列を抜き出します。<式> の値は0 から255 の範囲になければなりません。

<式> の値が0 ならば、LEFT\$ の値はヌルストリング(空の文字列) となります。<式> が <文字列> の文字数より大きければ、LEFT\$ の値は <文字列> とまったく同じになります。

参照：MID\$, RIGHT\$, サンプルプログラム 25

LEN

関 数

機 能 文字列の合計バイト数を得ます。

書 式 LEN(<文字列>)

文 例 PRINT LEN(A\$)
L=LEN("TEST")

<文字列> 全体の合計バイト数を得ます。

出力されない文字や空白も数えられます。ここで、出力されない文字とは、キャラクタコードで0から31までの、通常コントロールコードと呼ばれるものをいいます。

LENでは、<文字列>中にどんなタイプ(1バイト系, 2バイト系)の文字が含まれていても、合計のバイト数が得られます。

参照：KLEN, サンプルプログラム 25

LET

機 能 変数に値を代入します。

書 式 [LET] <変数名>= <式>

文 例 LET I=I+1
I=(I+1)*J
A\$=STRING\$(20,"*")

LETはなくてもかまいません。

<式>の内容は、数値、文字列のいかなる型であってもかまいません。ただし、<変数名>と<式>の型は同じでなければなりません。型の異なる数値変数への代入では左辺の精度に合わせた代入が行われます。

注意：代入では必ず、左辺に変数を置かなければなりません。

LINE

機能 指定した2点間に直線を描きます(パラメータ指定により四角形も描きます)。

書式 LINE [(Wx1, Wy1)]- (Wx2, Wy2) [, <パレット番号1>]
 STEP(x1, y1) STEP(x2, y2)
 [, B] [, <ラインスタイル>]
 BF <パレット番号2>
 <タイルストリング>

文例 LINE(100, 100)-(135, 100), 7,, &HF99F
 LINE-STEP(30, 30), 7, BF, 7

ワールド座標系の2点(Wx1, Wy1)と(Wx2, Wy2)を結ぶ直線を描きます。

(Wx1, Wy1)が省略された場合、LP(最終参照点)の値を採用します。STEPを使って相対座標で指定することもできます。

<パレット番号1>には、描く線の色を指定します。省略された場合は、そのときCOLORで設定されているグラフィック画面のフォアグラウンドカラーが採用されます。

次のパラメータには、BかBFのどちらかを指定することができます。BはBoxの意味で、(Wx1, Wy1)と(Wx2, Wy2)の2点を対角とする四角形の箱を描きます。BFはBox Fillの意味で、描いた四角形の内部をぬりつぶします。

<ラインスタイル>は、描く線の形態を設定するパラメータであり、&H0から&HFFFF(16進数)の値をとることができます。この値とラインの形態との関係は、図のとおりです。<ラインスタイル>が省略された場合は、直線を引きます。BFの指定時には指定できません。

<パレット番号2>は、四角形の内部をぬりつぶす際の色を、<タイルストリング>は、ぬりつぶす際の模様を指定します。<パレット番号2>および<タイルストリング>は、BF指定のときだけ有効です。BF指定があり、<パレット番号>、<タイルストリング>のいずれも省略された場合には、四角形を描いたのと同じ色で内部をぬりつぶします。

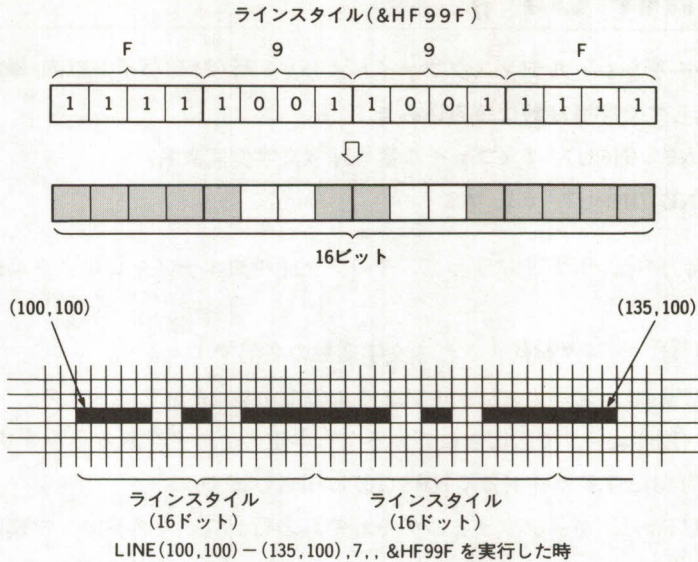
パレット番号については [2] COLOR を、<タイルストリング>については [2] PAINT を参照してください。

LINEを実行した場合、LPは(Wx2, Wy2)に移動します。

この図は文例を実行したものです。<ラインスタイル>の16進数の値を2進数16ビットで表したものと画面上の16ドットが対応しています。“1”のビットに対応するドットが表示され、“0”のビットに対応するドットは表示されません。この例では、水平座標の100から135までの36ドットの間、1点鎖線のラインスタイルが繰り返されています。線を16ドット以上引いた場合には、ラインスタイルが画面上の16ドットごとに繰り返されることになります。

このようにラインスタイルを指定すると、16ドット単位で線の形態を決めることができます

から、折れ線グラフや図面の作成の際、点線や1点鎖線などを簡単に描くことができます。



参照： [2] COLOR, [2] PAINT, サンプルプログラム 15, 16, 17, 18, 20

LINE INPUT

機能 キーボードから入力されるデータ(255バイト以内)を、区切ることなく一括して文字変数に代入します。

書式 LINE INPUT [<プロンプト文>;] <文字変数名>

文例 LINE INPUT "NAME"; NA\$

LINE INPUT B\$

LINE INPUT では、キーボードから入力されたすべてのデータがコンマなどの区切り記号も含めて <文字変数名> で指定された変数に代入されます (INPUT の場合は区切り記号で区切られたデータの1つ1つが順次抽出され、変数に代入されます)。

<プロンプト文> には、入力を受ける前に表示するメッセージを指定します。

入力を中断したい場合、**STOP** キーあるいは **CTRL** + **C** キーを入力します。プログラムの実行中に入力を中断すると、STOP ON の状態にない限り、プログラムの実行も中断され、コマンドモードにもどります。このとき、それまでに入力した値はすべて無効となります (文字変数には値は代入されません)。

参照：INPUT, LINE INPUT #, STOP ON/OFF/STOP

LINE INPUT

機能 シーケンシャルディスクファイルより、1行(255バイト以内)単位のデータを一括して文字型変数に読み込みます。

書式 LINE INPUT # <ファイル番号>, <文字型変数名>

文例 LINE INPUT #1, A\$

<ファイル番号>には、OPENによって、そのファイルをオープンしたときに指定した番号を使います。

<文字変数名>は、データが代入される文字変数の名前です。

LINE INPUT #は、シーケンシャルファイルの中から、キャリッジリターン(CR)コード+ラインフィード(LF)コードで区切られている1行単位のデータを読み込みます。ここで、CRコードはCHR\$(13)、LFコードはCHR\$(10)に相当します。

LINE INPUT #は、データファイルのそれぞれの行がスペースやコンマによっていくつかの欄に分割されているときや、アスキーセーブしたプログラムを他のプログラムのデータとして読み込むときなどに特に有効です。

参照：INPUT #, OPEN

LINE INPUT WAIT

機能 キーボードから入力されるデータを変数に代入します。その際、入力待ち時間を制限することができます。

書式 LINE INPUT WAIT <待ち時間>, [<プロンプト文> | ; |] <文字型変数名>

文例 LINE INPUT WAIT 50, "No.=", NO\$

LINE INPUT WAITは時間制限付きのLINE INPUTです。したがって、LINE INPUTとINPUT WAITの両方の機能を兼ねそなえています。詳しくはそれぞれの命令を参照してください。

参照：LINE INPUT, INPUT WAIT

LIST/LLIST

機能 メモリ上にあるプログラムの全部、または一部を表示あるいは印刷します。

書式 1)LIST [<始点行番号>] [-<終点行番号>]
 2)LLIST [<始点行番号>] [-<終点行番号>]

文例 LIST .
 LLIST 100-200

LIST を実行すると、プログラムの内容が、書式 1)は画面に、書式 2)はプリンタに、それぞれ出力されます。

パラメータの指定方法とリストの関係は次のとおりです。

- <始点行番号>-<終点行番号> : その範囲のすべてのプログラム行をリストする。
 <始点行番号>- : 始点行番号に始まりそれよりも大きい行番号のプログラム行すべてをリストする。
 -<終点行番号> : 最も若い行番号のプログラム行から終点行番号までをリストする。
 <始点行番号> : その行番号のプログラム行だけをリストする。
 行番号を省略 : プログラム行をすべてリストする。

行番号にピリオド(.)を指定すると、BASIC インタプリタ内のポインタが示している現在の行を指します(たとえば、最後に修正された行)。

リストの出力は **STOP** キーまたは **CTRL**+**C** の入力で終了します。

この命令は、実行し終るといつでもコマンドレベルにもどります。

LOAD

機能 プログラムをメモリにロードします。

書式 LOAD <ファイルディスクリプタ> [, R]

文例 LOAD "2 : DEMO", R
 LOAD "COM : N82NN"

<ファイルディスクリプタ> には、メモリにロードするプログラムファイル(ディスク、RS-232C 回線、キーボード)を指定します。LOAD を実行すると、すべての開いているファイルは閉じられ、変数に代入されている値は失われます。

R オプションをつけると、ファイルは開いたままで、プログラムをロード後、直ちに実行を開始します。

注意：RS-232C 回線からのプログラムロードの場合、プログラムのロードが完了しても、自動的に LOAD コマンドは終了しません。STOP キーで強制的に中止しなければなりません。

LOC

関 数

機 能	ファイル中での論理的な現在位置を得ます。
書 式	LOC(<ファイル番号>)
文 例	LAST=LOC(2)

<ファイル番号> で指定されたファイルの種類によって、得られる値の意味が異なります。

ランダムディスクファイルの場合

最後に読み書き(GET/PUT)を行ったレコードのレコード番号が得られます。

シーケンシャルディスクファイルの場合

オープンしてから現在までにファイルを読み書きした回数を得られます。

キーボードファイルの場合

割り込みによって受け付けられ、キーボードバッファ中にたまっている文字数が得られます。

RS-232C 回線ファイルの場合

割り込みによって受け付けられ、入力バッファ中にたまっている文字数が得られます。

読み取り動作(INPUT #, INPUT\$など)が行われずに、割り込みによる RS-232C 回線からの受信が続くと、やがてバッファが一杯になりエラーとなるため、これを監視するのに LOC を使用することができます。

LOCATE

機 能	テキスト画面のカーソルを指定位置へ移動します。
書 式	LOCATE [<X>] [, <Y>] [, <カーソルスイッチ>]
文 例	LOCATE 10, 10 LOCATE ,, 0

X は水平座標を表し、省略された場合は 0 とみなされます。Y は垂直座標を表し、省略された場合は現在カーソルがセットされている行とみなされます。これらは、ともに画面の左上を (0, 0) とするキャラクタ座標により指定します。

〈カーソルスイッチ〉はカーソルの表示状態を決めるスイッチで、0を指定すると表示されなくなり、1を指定すると表示されるようになります。

注意：水平座標が奇数桁の場合、日本語の表示あるいは日本語入力を正しく行うことはできません。

参照：サンプルプログラム 10, 33

LOF

関 数

機 能 ファイルの大きさを得ます。

書 式 LOG(〈ファイル番号〉)

文 例 MAXREC=LOF(2)

〈ファイル番号〉で指定されたファイルの種類によって、得られる値の意味が異なります。

ディスクファイルの場合

ファイルの大きさがセクタ数で得られます。この値は、ファイルがランダムファイルであれば、そのファイルの最大レコード番号に相当します。

RS-232C 回線ファイルの場合

入力バッファの残りのバイト数(入力可能な余裕バイト数)が得られます。

参照：サンプルプログラム 29

LOG

関 数

機 能 自然対数を得ます。

書 式 LOG(〈数式〉)

文 例 PRINT LOG(35/9)
LA=LOG(X * Y)

〈数式〉の、自然対数(eを底とした対数)を値として得ます。

〈数式〉に倍精度実数が含まれる場合、得られる値は倍精度となりますが、他の場合には単精度となります。

参照：EXP

LPOS

関数

機能 現在のプリンタのヘッド位置を得ます。

書式 LPOS(<式>)

文例 HEAD=LPOS(0)

使用中のプリンタの、現在のヘッド位置(桁位置, すなわち水平位置)を得ます。

<式> は、ダミーであって意味を持ちませんが、省略できません。

参照: WIDTH LPRINT

LPRINT

機能 プリンタにデータを出力します。

書式 LPRINT [<式>] [| , | <式>…] [| , |]
; ;

文例 LPRINT A\$, D2

LPRINT CHR\$(&H41) ; CHR\$(&H42) ; CHR\$(&H43)

指定した式の値や文字列などのデータを、プリンタに出力します。

LPRINT は、データをプリンタに出力する点が違うだけで、その動作は PRINT と同じです。

参照: PRINT

LPRINT USING

機能 文字列、数値などのデータを編集し、プリンタに出力します。

書式 LPRINT USING <書式制御文字列>; <式> [| , | <式>…] [| , |]
; ;

文例 LPRINT USING "@ ###"; A\$, B

LPRINT USING "###.###"; C

<書式制御文字列> によって決定される領域や書式に従って、<式> に指定された各種データをプリンタに出力します。

LPRINT USING は、データをプリンタに出力する点が違うだけで、その動作は PRINT USING と同じです。

参照：PRINT USING

LSET/RSET (DISK モード)

機能 ランダムファイルバッファのフィールドにデータを代入します。

書式 1) LSET <文字変数>=<文字列>

2) RSET <文字変数>=<文字列>

文例 LSET A\$="単価"

RSET B\$=MK\$\$(1)

<文字変数>には、FIELD でランダムファイルバッファ上に割り当てられたフィールド変数を指定します。

<文字列>を指定することにより、フィールド変数に対応するフィールドにデータ(文字型)を代入することができます。<文字列>の文字数が FIELD で割り当てられたフィールド変数の長さより短い場合、LSET では左詰め、RSET では右詰めフィールドを満たします。また、<文字列>がフィールドより長い場合は、LSET、RSET とも右側の部分が失われます。

注意：<文字変数>は、あらかじめ FIELD で定義されていなければなりません。定義されていない変数名を用いると、データを正しく代入することができません。

また、<文字列>に指定するデータは文字型でなければなりませんので、数値型データは MKI\$, MKS\$, MKD\$のいずれかにより文字型データに変換しておかなければなりません。

参照：FIELD, GET, MKI\$/MKS\$/MKD\$, PUT, サンプルプログラム 26, 29

MAP

関数

機能 スクリーン座標、ワールド座標の相互変換を行います。

書式 MAP(<数式>, <機能>)

文例 SY1=MAP(WY1, 1)

<数式>で指定されるスクリーン座標あるいはワールド座標の値を、<機能>の指定に従い、対応するワールド座標あるいはスクリーン座標に変換します。

<機能>に指定する値により、次のような変換を行います。

- 0 : <数式> をワールド座標系における x 座標とみなし、これを対応するスクリーン座標系での x 座標に変換する ($W_x \rightarrow S_x$).
- 1 : <数式> をワールド座標系における y 座標とみなし、これを対応するスクリーン座標系での y 座標に変換する ($W_y \rightarrow S_y$).
- 2 : <数式> をスクリーン座標系における x 座標とみなし、これを対応するワールド座標系での x 座標に変換する ($S_x \rightarrow W_x$).
- 3 : <数式> をスクリーン座標系における y 座標とみなし、これを対応するワールド座標系での y 座標に変換する ($S_y \rightarrow W_y$).

MAPを用いれば、ワールド座標からスクリーン座標への変換ができますから、GET や PUT のようにスクリーン座標で位置を指定する操作の場合にも、ワールド座標を変換して指定することができます。

注意：MAP は、変換の結果がスクリーン座標系あるいはワールド座標系から外れてもエラーとなりません。

MERGE (DISK モード)

機能	メモリ上のプログラムに、ディスク上のプログラムファイルを合成します。
書式	MERGE <ファイルディスクリプタ>
文例	MERGE "2:TEST"

メモリ上のプログラムと、<ファイルディスクリプタ>により指定したディスク上のプログラムファイルを合成(マージ)して1つのプログラムにし、メモリ上に置きます。

メモリ上のプログラムと、ディスク上のプログラムファイルに同一の行番号があった場合には、ディスク上のプログラムファイル中の行が採用されます。

なお、マージ後、ただちにそのプログラムを実行させたいときは、MERGE オプションつきの CHAIN を使うことができます。

注意：混合するプログラムファイルはアスキーセーブされていなければなりません。そうでない場合には、"Sequential I/O only" エラーになります。

参照：CHAIN, SAVE

MID\$

機能	文字列の一部を置き換えます。
書式	MID\$(〈文字変数〉, 〈式1〉 [, 〈式2〉]) = 〈文字列〉
文例	MID\$(A\$, 3) = "ABC" MID\$(B\$, N, 2) = C\$

〈文字変数〉に代入されている文字列の〈式1〉番目の文字から〈式2〉個の文字を、〈文字列〉の最初から〈式2〉個分の文字列で置き換えます。

〈式1〉は1から255の範囲、また、〈式2〉は0から255の範囲になければなりません。

〈文字列〉には文字変数を用いることもできます。

〈式2〉が省略された場合、または〈式2〉の値が〈文字列〉の文字数を超える場合、〈文字変数〉に代入されている文字列を、〈文字列〉のすべての文字と置き換えます。

注意: この命令は2バイト系日本語文字列にも利用できますが、その際には文字列の前後に入るKI/KOコードに注意してください。

また、1バイト系と2バイト系の混在文字列に使用する場合は、文字の境界に充分注意しないと、予期しない結果となります。

参照: MID\$(関数)

MID\$

関数

機能	文字列の中から任意の長さの文字列を抜き出します。
書式	MID\$(〈文字列〉, 〈式1〉 [, 〈式2〉])
文例	B\$ = MID\$(A\$, 2, 3) COUNTRY\$ = MID\$("JPNUSAFRAGBRCAN", N * 3 + 1, 3)

〈文字列〉の〈式1〉番目の文字から〈式2〉桁の文字列を抜き出します。

〈式1〉は1から255の範囲、また、〈式2〉は0から255の範囲になければなりません。

〈式2〉を省略した場合や、〈文字列〉の〈式1〉番目の文字から右側全部の文字数が〈式2〉より小さい場合は、〈文字列〉の〈式1〉番目の文字より右側全部の文字列を抜き出します。

〈式1〉が〈文字列〉の文字数より大きければ、MID\$の値はヌルストリング(空の文字列)となります。

注意: この命令は2バイト系日本語文字列にも利用できますが、その際には文字列の前後に入るKI/KOコードに注意してください。

また、1バイト系と2バイト系の混在文字列に使用する場合は、文字の境界に充分注意しないと、予期しない結果となります。

参照：LEFT\$, MID\$, RIGHT\$, サンプルプログラム 23, 25

MKI\$/MKS\$/MKD\$

関数

機能	数値データをその数値の内部表現に対応した文字列に変換します。
書式	MKI\$(〈整数値〉) MKS\$(〈単精度実数値〉) MKD\$(〈倍精度実数値〉)
文例	A\$=MKI\$(16961) LSET B\$=MKS\$(1.23) RSET C\$=MKD\$(3.141592654 #)

これらの関数は、数値データをランダムデータファイルに対して書き出す際、数値データはそのままの形では書き出すことができないため、これを文字型データに変換するために使用するものです。

- MKI\$: 整数値を2文字(2バイト)の文字列に変換(CVIの逆)。
MKS\$: 単精度実数値を4文字(4バイト)の文字列に変換(CVSの逆)。
MKD\$: 倍精度実数値を8文字(8バイト)の文字列に変換(CVDの逆)。

実際にランダムデータファイルに数値データを書き出す際には、まずこれらの関数を用いて数値データを文字型化し、これをLSET/RSETでフィールド変数にセットしてから、PUTを行います。

これらの関数で文字列に変換された数値データを、もとの数値型に変換するには、CVI/ CVS/CVDの各関数を使用します。

数値から文字への変換は数値の内部表現の値をそのままそれに対応する文字コードをもつ文字列にすることによって行われます。実際の数値と変換後の文字列との関係を整数値で説明すると次のようになります。

整数値が16961の場合、この数値の内部表現は&H4241(16進表記)であり、A\$=CHR(&H41)+CHR(&H42)、すなわち、A\$="AB"となります(上下バイトが逆になることに注意)。つまり、数値データをキャラクタコードとみなすわけです。

注意：STR\$関数とは変換のしかたがまったく異なります。

参照：CVI/ CVS/CVD, LSET/RSET, サンプルプログラム 26

MON (DISK モード)

機能 モニタモードに入ります。

書式 MON

文例 MON

BASIC モードからモニタモードに移ります。モニタモードは、機械語プログラムの作成、デバッグのための環境を用意します。詳しくは BASIC ユーザーズマニュアルを参照してください。 **CTRL** + **B** で、もとの BASIC モードにもどります。

NAME (DISK モード)

機能 ディスクファイルの名前を変更します。

書式 NAME <旧ファイルディスクリプタ> AS <新ファイルディスクリプタ>

文例 NAME "2 : SAMPLE . BAS" AS "2 : SAMPLE1"

<旧ファイルディスクリプタ>で表されているディスク上のファイルの名前を<新ファイルディスクリプタ>に変更します。

ディスクファイルに対するファイルディスクリプタの指定形式は、" [ドライブ番号:] ファイル名" です。

ドライブ番号が省略された場合、ドライブ1のディスクが処理の対象になります。

<旧ファイルディスクリプタ>と<新ファイルディスクリプタ>に指定されるドライブ番号は同じでなければなりません。

なお、ファイルの名前の変更を行うファイルはクローズされた状態でなければなりません。

NEW

書式 メモリ上にあるプログラムを抹消し、すべての変数を初期化します。

書式 NEW

文例 NEW

NEW はメモリ上のプログラムを抹消し、すべての変数を初期化(クリア)します。また、そのとき開かれているファイルもすべて閉じられます。

コマンドの実行が終ると、いつもコマンドレベルにもどります。

NEW ON

機能 システムを再起動します。

書式 NEW ON <式>

文例 NEW ON 1

<式> に指定された値により、いろいろな状態でシステムを再起動させます。

この命令は、ディップスイッチ SW2 の持つ機能に対応して意味づけられており、ディップスイッチの操作の一部を BASIC コマンドとして実行できるようにしたものです。

NEW ON では、ディップスイッチ SW2 のうち、2 番～4 番スイッチまでを制御することができます。

NEW ON のスイッチは図のようにそれぞれを 1 ビット (0 または 1) で表します。

ディップスイッチ SW2 の機能

2 <システムモード>	0 : BASIC モード	1 : ターミナルモード
3 <桁数>	1 : 1 行 80 桁	0 : 1 行 40 桁
4 <行数>	1 : 1 画面 25 行	0 : 1 画面 20 行

<式> に指定する値は、以下の式で求められます。

$$\langle \text{式} \rangle = 4 \text{ 番スイッチの値} * 2^3 + 3 \text{ 番スイッチの値} * 2^2 + 2 \text{ 番スイッチの値} * 2$$

たとえば、25 行×40 桁表示の状態では、N₈₈-BASIC を BASIC モードで起動させたいときには、<式> = 1 * 2³ + 0 * 2² + 0 * 2 となり、<式> には、8 を指定します。

OCT\$

関数

機能 10 進数を 8 進数に変換し、その文字列を得ます。

書式 OCT\$(<数式>)

文例 PRINT OCT\$(320)

<数式> の値を 8 進数に変換して、その文字列を得る関数です。<数式> の値は、-32768 から 32767 (または 0 から 65535) までの範囲になければなりません。<数式> の値と得られる文字列との対応は次のとおりです。

〈数式〉の値		得られる8進表記文字列
0~32767	:	0~77777
-32768~-1	:	100000~177777
32768~65535	:	100000~177777

参照: HEX\$, VAL, サンプルプログラム 24

ON COM GOSUB

機能 RS-232C 回線からの割り込みが発生したとき、分岐する処理ルーチンの開始行を指定します。

書式 ON COM [(〈回線番号〉)] GOSUB 〈行番号〉

文例 ON COM GOSUB 1000
ON COM(2) GOSUB *RECV

指定した RS-232C 回線に入力割り込みがあったときに分岐する処理ルーチンの開始行番号を定義します。

RS-232 C(第2回線/第3回線)インタフェースボードを使用する場合、〈回線番号〉に指定できる値と意味は次のとおりです。(〈回線番号〉)を省略(カッコも含む)した場合は、第1回線となります。

- 1 : RS-232C 第1回線
- 2 : RS-232C 第2回線
- 3 : RS-232C 第3回線

RS-232 C(第2回線/第3回線)インタフェースボードを使用しない場合、(〈回線番号〉)は必ずカッコも含めて省略してください。

〈行番号〉には分岐させる開始行番号を指定します。

処理ルーチンからの復帰は、一般のサブルーチンと同じで、RETURN で行います。たんに RETURN としたときは中断した所から再開し、後ろに行番号を指定したときはその行から再開します。

ただし、行番号付きの RETURN では、割り込みが発生した箇所と同じスタックレベルのプログラム行にもどるように指定してください。これを無視すると、スタック領域が異常消費され、プログラムは正しく動作しません。とくに必要がない限り、使用しないようにしてください。

注意：この命令によって割り込み処理ルーチンに分岐させるには、割り込みが発生したとき、COM ON の状態でなければなりません。
ON COM GOSUB は、RS-232C 回線を OPEN した後実行されなければ有効となりません。

参照：COM ON/OFF/STOP, OPEN, RETURN

ON ERROR GOTO

機能	エラーが起こったときに分岐する処理ルーチンの開始行を定義します。
書式	ON ERROR GOTO <行番号>
文例	ON ERROR GOTO 1000

<行番号>には分岐させる処理ルーチンの開始行番号を指定します。

この命令を実行しておくとし、エラーが起こったときに、指定した開始行から始まるエラー処理ルーチンに処理が移ります。エラー発生時のプログラム行番号は ERL に、エラーコードは ERR にセットされますので、エラー処理ルーチン内では、これらを利用して独自のエラー処理を行うことができます。

処理ルーチンから復帰するには、一般のサブルーチンとは異なり、RESUME を使います。

プログラムの実行終了後は、必ず ON ERROR GOTO 0 を実行して、エラー割り込みを無効にしておくようにしてください。さもないと、エラーが発生するたびに(プログラムがメモリ上にある限り、ダイレクトモードであっても)エラー処理ルーチンに処理が移ってしまいます。

ON ERROR GOTO 0 の実行後は、エラーが生じると通常どおりエラーメッセージが表示され、プログラムの実行は停止します。

エラー処理ルーチン内に ON ERROR GOTO 0 がある場合は、エラーによる分岐(エラートラップ)の原因となったエラーのエラーメッセージを表示し、プログラムの実行を停止します。

注意：<行番号>に指定された行が存在しない場合、“Undefined line number”エラーが起こります。

エラー処理ルーチンの中ではエラー割り込みは起こりません。エラー処理ルーチンの実行中にエラーが起きた場合には、それに対するエラーメッセージが表示され、実行は停止します。

参照：ERL/ERR, ERROR, RESUME, サンプルプログラム 22

ON...GOSUB/ON...GOTO

機能 指定されたいずれかの行に分岐します。

書式 1) ON <式> GOSUB <行番号> [, <行番号> ...]

2) ON <式> GOTO <行番号> [, <行番号> ...]

文例 ON A GOSUB 100, 200, 300, 400

ON N GOTO *ENTRY1, *ENTRY2, *ENTRY3, *ENTRY4

<式> の値に応じて、“<式> の値” 番目の <行番号> に処理が移ります。たとえば、<式> の値が3であったなら、行番号の並びの3番目の行が分岐先となります。

ON...GOSUB の場合、<行番号> はサブルーチン(RETURN で終わっている必要がある)の開始行でなくてはなりません。

注意：<式> の値が負の場合には、“Illegal function call” エラーが起きますが、値が0あるいは並びの行番号より大きい場合には次の行に実行が移り、エラーは起こりません。

参照：サンプルプログラム 29, 30

ON HELP GOSUB

機能 **HELP** キーによる割り込み処理ルーチンの開始行を定義します。

書式 ON HELP GOSUB <行番号>

文例 ON HELP GOSUB 1000

この命令を実行しておくと、以後プログラムの実行中に**HELP** キーが押されるたびに、指定した開始行から始まる割り込み処理ルーチンに処理が移ります。この命令は割り込みの対象として**HELP** キーを使っていることを除けば、他の割り込み定義命令(KEY, COM, PEN, STOP など)と同様な使い方で利用できます。

<行番号> には、処理ルーチンの開始行番号を指定します。

処理ルーチンからの復帰は、一般のサブルーチンの場合と同じで、RETURN によって行います。たんに RETURN としたときは、中断した所から再開し、後ろに行番号を指定したときは、その行から再開します。

ただし、行番号つきの RETURN では、割り込みが発生した箇所と同じスタックレベルのプログラム行にもどるように指定してください。これを無視すると、スタック領域が異常消費され、プログラムは正しく動作しません。とくに必要がない限り、使用しないようにしてください。

INPUT 実行中に割り込みが生じた場合は、RETURN だけで復帰させると INPUT の次の命令から実行を再開しますから、行番号を指定するか、プログラムで適切な処理をしなければなりません。

本来この命令は、アプリケーションプログラムを作成する際、プログラム中にプログラムの使い方、入力の方法などの説明を書いた表示ルーチンを用意しておき、ユーザーがそのプログラムの実行中、使い方がわからなくなったときに **HELP** キーを押して指示を受ける、といった用途のために用意されているものです。

注意：この命令によって割り込み処理ルーチンに分岐させるには、割り込みがあったとき **HELP ON** の状態でなければなりません。

参照：HELP ON/OFF/STOP, RETURN

ON KEY GOSUB

機能 ファンクションキーによる割り込みルーチンの開始行を定義します。

書式 ON KEY GOSUB <行番号> [, <行番号> …]

文例 ON KEY GOSUB 100, 200, 300, 400

この命令を実行しておくで、以後プログラムの実行中にファンクションキーが押されるたびに、各ファンクションキーに対応する指定開始行から始まる割り込みルーチンに処理が移ります。

<行番号>には、割り込みが発生したときに分岐させる処理ルーチンの開始行番号を、ファンクションキーの番号順に並べて指定します。<行番号>の並び中で、行番号の指定を省略した場合は、それに対応するファンクションキーによる割り込みは起こりません。したがって、たとえば ON KEY GOSUB 100,, 300 と指定した場合、**f•1** キーおよび **f•3** キー以外は割り込みの対象となりません。

<行番号> は最大 10 個まで並べて書くことができます。行番号の代わりにラベル名を使うこともできます。

処理ルーチンからの復帰は、一般のサブルーチンの場合と同じで、RETURN によって行います。たんに RETURN としたときは、中断した所から再開し、後ろに行番号を指定したときは、その行から再開します。

ただし、行番号付きの RETURN では、割り込みが発生した箇所と同じスタックレベルのプログラム行にもどるように指定してください。これを無視すると、スタック領域が異常消費され、プログラムは正しく動作しません。とくに必要がない限り、使用しないようにしてください。

注意：この命令によって割り込み処理ルーチンに分岐させるには、割り込みが発生したとき、KEY ON の状態でなければなりません。

この命令を実行すると、ファンクションキー本来の機能(ファンクションキーを押すとKEY で定義した文字列が入力される機能)は無視されます。ただし、定義された文字列の内容は、CONSOLE で指定することにより表示することは可能です。

参照：KEY ON/OFF/STOP, RETURN, サンプルプログラム 30

ON PEN GOSUB

機能	ライトペンが押されたときの割り込み処理ルーチンの開始行を定義します。
書式	ON PEN GOSUB <行番号>
文例	ON PEN GOSUB *PENTEST

この命令を実行しておく、以後プログラムの実行中にライトペンが押されるたびに、指定した開始行から始まる割り込み処理ルーチンに処理が移ります。この命令は割り込みの対象としてライトペンを使っていることを除けば、他の割り込み定義命令(KEY, COM, HELP, STOP など)と同様な使い方で利用できます。

<行番号>には、処理ルーチンの開始行番号を指定します。

処理ルーチンからの復帰は、一般のサブルーチンの場合と同じで、RETURN によって行います。たんに RETURN としたときは、中断した所から再開し、後ろに行番号を指定したときは、その行から再開します。

ただし、行番号つきの RETURN では、割り込みが発生した箇所と同じスタックレベルのプログラム行にもどるように指定してください。これを無視すると、スタック領域が異常消費され、プログラムは正しく動作しません。とくに必要がない限り、使用しないようにしてください。

注意：この命令によって処理ルーチンに分岐させるには、PEN ON の状態でなければなりません。

参照：PEN ON/OFF/STOP, RETURN

ON PLAY GOSUB

S

機能	PLAY 割り込み処理ルーチンの開始行を定義します。
書式	ON PLAY(<チャンネル番号>, <残りバイト数>) GOSUB <行番号>
文例	ON PLAY(1, 1024) GOSUB *NEXT.PLAY

この命令を実行しておくと、バックグラウンド(BASICプログラムの実行と独立して演奏する形態。PLAYのMBコマンド参照)で演奏中に、<チャンネル番号>のサウンドバッファ内の未演奏音楽情報が<残りバイト数>以下になったとき、指定した<行番号>から始まる処理ルーチンに処理を移します。

<チャンネル番号>に指定する値とその意味の関係は次のとおりです。

- 0 : 全チャンネルの中で最大の残りバイト数をもつもの
- 1~6 : 各チャンネル
- 負 : 全チャンネルの中で最小の残りバイト数をもつもの

<残りバイト数>は、0~32767の範囲です。

処理ルーチンからの復帰は、RETURNで行います。たんにRETURNとしたときには、中断した所から再開し、後ろに行番号を指定したときは、その行から再開します。

ただし、行番号付きのRETURNでは、割り込みが発生した箇所と同じスタックレベルのプログラム行にもどるように指定してください。これを無視すると、スタック領域が異常消費され、プログラムは正しく動作しません。とくに必要がない限り、使用しないでください。

注意: この命令によって割り込み処理ルーチンに分岐させるには、割り込みがあったときにPLAY ONの状態であればなりません。

PLAYの割り込みは、BASICの他の割り込み命令と異なり、割り込み処理ルーチンに制御が移っても自動的に割り込み停止状態となりません。したがって、処理ルーチンの先頭ではPLAY OFFまたはPLAY STOPを実行し、終わりでPLAY ONを実行する必要があります。

参照: PLAY, PLAY ON/OFF/STOP, サンプルプログラム 41

ON STOP GOSUB

機能 STOP キーによる割り込み処理ルーチンの開始行を定義します。

書式 ON STOP GOSUB <行番号>

文例 ON STOP GOSUB 100

この命令を実行しておく、以後プログラムの実行中にSTOP キーが押されるたびに、指定した割り込み処理ルーチンに処理が移ります。この命令は割り込みの対象としてSTOP キーを使っていることを除けば、他の割り込み定義命令(KEY, COM, HELP, PEN など)と同様な使い方で利用できます。

<行番号>には、処理ルーチンの開始行番号を指定します。

処理ルーチンからの復帰は、一般のサブルーチンの場合と同じで、RETURN によって行います。たんに RETURN としたときは、中断した所から再開し、後ろに行番号を指定したときは、その行から再開します。

ただし、行番号付きの RETURN では、割り込みが発生した箇所と同じスタックレベルのプログラム行にもどるように指定してください。これを無視すると、スタック領域が異常消費され、プログラムは正しく動作しません。とくに必要がない限り、使用しないようにしてください。

注意：この命令によって割り込み処理ルーチンに分岐させるには、STOP ON の状態でなければなりません。

この命令はアプリケーションプログラム実行中に、ユーザーが誤ってSTOP キーを押してしまい、プログラムの実行が中断されてしまうのを防止するためのものです。したがって、不用意にこの命令が実行されてしまうと、本来のSTOP キーおよびCTRL + C の機能は失われ、プログラムの中断ができなくなってしまいます。アプリケーションプログラムの動作が完全になってから、この命令を使うようにしてください。

なお、RS-232C 回線ファイルのデータ受信命令(INPUT #など)の実行中にSTOP キーが押された場合は、指定した割り込みルーチンに処理は移りません。

参照：RETURN, STOP/ON/OFF/STOP

ON TIME\$ GOSUB

機能 内蔵クロックによる割り込みの発生時刻と、そのとき分岐する処理ルーチンの開始行を定義します。

書式 ON TIME\$="時刻" GOSUB <行番号>

文例 ON TIME\$="07:30:00" GOSUB *MORNINGCALL

この命令を実行しておくこと、以後プログラムの実行中、指定時刻になったときに指定した開始行から始まる割り込み処理ルーチンに処理が移ります。この命令は割り込みの対象として内蔵クロックを使っていることを除けば、他の割り込み定義命令(KEY, COM, HELP, STOP など)と同様な使い方です。

〈時刻〉には、割り込み時刻を、"hh:mm:ss"の形の文字列(時:分:秒を各2桁で表したものの、TIME\$の設定と同様)で設定します。なお、この命令で制御できる分解能は2秒ですから、±1秒単位の誤差が出ることがあります。

〈行番号〉には、処理ルーチンの開始行番号を指定します。

処理ルーチンからの復帰は、一般のサブルーチンの場合と同じで、RETURNによって行います。たんにRETURNとしたときは、中断した所から再開し、後ろに行番号を指定したときは、その行から再開します。

ただし、行番号つきのRETURNでは、割り込みが発生した箇所と同じスタックレベルのプログラム行にもどるように指定してください。これを無視すると、スタック領域が異常消費され、プログラムは正しく動作しません。とくに必要がない限り、使用しないようにしてください。

注意: この命令によって割り込み処理ルーチンに分岐させるには、TIME\$ ONの状態であればなりません。

なお、この命令は、実行された時点で、設定された時刻から現在の時刻を引算して割り込み時刻を決定します。したがって、この命令の実行後にTIME\$で現在時刻の変更を行うと、割り込み時刻に狂いが生じますので注意してください。

参照: RETURN, TIME\$, TIME\$ ON / OFF / STOP, サンプルプログラム 31

OPEN

機能

ファイルを開きます。

書式

OPEN <ファイルディスクリプタ> [FOR <モード>] AS [#] <ファイル番号>

文例

```
OPEN "TEST.BAS" FOR INPUT AS #1
OPEN "LINES.DAT" FOR OUTPUT AS #2
OPEN "LINES.DAT" FOR APPEND AS #2
OPEN "2:DATA1" AS #1
OPEN "COM:N81XN" AS #1
```

この命令は、〈ファイルディスクリプタ〉で指定したファイルをオープンして入出力操作のための専用バッファを用意し、それに〈ファイル番号〉をつけます。以後、ファイルへの入出力は、〈ファイル番号〉を指示することにより行います。

プログラム中で、複数の OPEN を使うことにより、同時に複数のファイルをオープンすることができます。同時にオープンできるファイル数は、最高15ですが、BASICの起動時に"How many files (0-15)?"で指定した数を超えてオープンすることはできません。複数のファイルをオープンする場合、〈ファイル番号〉にはそれぞれ異なった数値を指定します。

〈ファイル番号〉には、1から15までの数値を指定することができますが、BASIC起動時に指定した数を超えてはなりません。

〈モード〉はファイルへのアクセス方法を指定するものです。モードには次の4種類があります。

- | | |
|---------------|--|
| INPUT | : 入力モード。既存のファイルから入力を行うよう指示。 |
| OUTPUT | : 出力モード。新しくファイルを作り、出力を行うよう指示。 |
| APPEND | : 追加モード。既存のファイルの終わりに追加出力を行うよう指示。 |
| 省略時 | : ランダムモード。ディスクファイルのランダム入出力、または RS-232C 回線ファイルへの入出力を行うよう指示。 |

FOR〈モード〉を指定した場合、シーケンシャルファイルに対して入出力を行うことを指示します。

FOR〈モード〉を省略した場合、ランダムファイルに対して入出力を行うことを指示します。

〈ファイルディスクリプタ〉に指定したファイルが RS-232C 回線ファイルの場合は、シーケンシャル入出力ですが、ランダム入出力と同様、FOR〈モード〉は省略してください。

ランダム入出力を行うためには、OPEN を実行して用意した専用バッファに対し、FIELD によりフィールド変数を割り当てなければなりません。

注意: INPUT(入力)モード、APPEND(追加)モードでは、指定されたファイルが存在しないと、OPEN 実行時に"File not found"エラーとなります。OUTPUT(出力)モードでは、常に指定された名前のファイルが新しく作られ、同一名のファイルがあった場合にはそのファイルは削除されて新しく作り直されます。ランダムモードでは、指定されたファイルが存在しないと新たに作られますが、すでに存在しているときは指定ファイルに対して入出力が行われます(OPEN 実行時にファイルの削除や破壊は起こりません)。

RS-232C 回線ファイルをオープンする場合は、〈ファイルディスクリプタ〉として、ファイル名 COM〈回線番号〉に続けて、パリティチェック、データビット長などを指定する必要があります。指定形式は次のとおりです。

COM [〈回線番号〉]: [〈パリティチェック〉 [〈データビット長〉 [〈ストップビット長〉
[〈X パラメータ〉 [〈S パラメータ〉]]]]]

〈回線番号〉は RS-232C 回線の番号を表し、1, 2, 3 で指定します。省略すると 1 となります。なお、2, 3 は、RS-232C 拡張インタフェースボードが必要です。

〈パリティチェック〉は E, O, N で表し、それぞれ偶数パリティ、奇数パリティ、パリティ無しを意味します。

〈データビット長〉は 1 文字を表すのに用いるビット数で、7 または 8 で指定します。

〈ストップビット長〉はストップビット数を決めるもので、1, 2, 3 で指定します。1 は 1 ビット、2 は 1.5 ビット、3 は 2 ビットを意味します。

〈X パラメータ〉は XON/XOFF による通信制御を行うことを指定します。X が指定された場合 XON/XOFF 制御を行います。N が指定された場合にはこの制御は行われません。

〈S パラメータ〉は〈データビット長〉に 7 を指定したときに、キャラクタコード 128 以上の文字 (カナ文字など) の入出力ができなくなるのを補助するパラメータです。S または N で指定します。S を指定したとき入出力可能で、N を指定したときは入出力不可能になります。

OPEN で扱うことのできるファイル機器は次のとおりです。

ファイル	使用できるモード
ディスクファイル(ドライブ番号:)	INPUT, OUTPUT, APPEND, 省略
RS-232C コミュニケーションファイル (COM:)	省 略
キーボードファイル (KYBD:)	INPUT
スクリーンファイル (SCRN:)	OUTPUT
プリンタファイル (LPT:)	OUTPUT

注意: スクリーンファイル(SCRN:)を指定した場合、2 バイト系日本語文字を出力することはできません。

参照: CLOSE, サンプルプログラム 1, 26, 27, 28, 29

OPTION BASE

機能 配列の添字の最小値を指定します。

書式 OPTION BASE | 0 |
| 1 |

書式 OPTION BASE 1

OPTION BASE は、配列の添字の最小値を 0 または 1 に指定します。この命令が用いられない場合、最小値は 0 となります。

この命令は、プログラム中で配列変数が宣言、または引用される前に置かなければ効果がありません。

また、添字の最小値を一度指定すると、再指定によって最小値を変更することはできません。この指定を解除、あるいは変更するには、RUN あるいは CLEAR を行わなければなりません。

注意：OPTION BASE 1 を実行して添字の最小値を 1 に指定した場合、以後、配列の添字として 0 が用いられると、“Subscript out of range”エラーが起こるようになります。また、添字の最小値を再指定しようとするとき、“Duplicate Definition”エラーになります。

参照：DIM, サンプルプログラム 5

OUT

機能 出力ポートに 1 バイトのデータを送出します。

書式 OUT <ポート番号>, <式>

文例 OUT 64, 32

<ポート番号> は出力ポートの番号で、0~32767(&H0~&H7FFF)の範囲内の整数で指定します。

<式> はポートに出力するデータで、0~255 の範囲内の整数で指定します。

参照：INP

[1] PAINT

機能 指定された境界色で囲まれた領域を、指定された色でぬります。

書式 PAINT (Wx, Wy) [, <領域色>] [, <境界色>]
STEP(x, y)

文例 PAINT (-50, 120), 3, 4
PAINT STEP(10, 10), 6

(Wx, Wy)を中心点として、<境界色>で囲まれた領域を、指定された<領域色>でぬりつぶします。(Wx, Wy)はワールド座標で指定します。STEPをつけて、LP(最終参照点)からの相対座標(x, y)で指定することもできます。

<領域色>、<境界色>はともにパレット番号で指定します。<領域色>が省略された場合には、COLORで指定されたフォアグラウンドカラーが用いられます。<境界色>が省略された場合には、<領域色>の指定と同じパレット番号が<境界色>として用いられます。

(Wx, Wy)の色が境界色と同じ色であった場合には、PAINTは、何の画面操作も行いません。

注意: PAINTはビューポート内でのみ働きますので、ビューポートの境界はPAINT動作の境界とみなされます。また、(Wx, Wy)がウィンドウの外にある場合には、“Illegal function call”エラーとなります。

参照: [1] COLOR, VIEW, サンプルプログラム 17

[2] PAINT

機能 指定された境界色で囲まれた領域を、指定されたタイルパターンで埋めます。

書式 PAINT (Wx, Wy) [, <タイルストリング>] [, <境界色>]
STEP(x, y)

文例 PAINT (255, 120), TILE\$, 4

(Wx, Wy)を中心点として、<境界色>で囲まれた領域をタイルパターンで埋めます。(Wx, Wy)はワールド座標で指定します。STEPをつけて、LP(最終参照点)からの相対座標(x, y)で指定することもできます。

<タイルストリング>は、タイリングに用いられる基本タイルの大きさと模様を決定する文字列です。タイルの大きさは、横方向は8ドット分と決められていますが、縦方向の長さは<タイルストリング>の文字列の長さで決まります。縦方向がnドットのタイルを作るためには、

〈タイルistring〉として、白黒モードでn文字、8色中・8色モードおよび4096色中・8色モードで3*n文字、4096色中・16色モードで4*n文字の長さがそれぞれ必要です。

タイルの模様は、〈タイルistring〉の文字列のキャラクタコードの2進数表現(ビットパターン)により表現されます。〈タイルistring〉の指定の方法は白黒モードとカラーモードとで異なります。

●白黒モードの場合

〈タイルistring〉中の各1文字(バイト)のコードがタイルの横8ドットの線に対応します。文字コードの2進数表現で、タイルの横8ドットのうち、1に対応するドットはセット(白)、0に対応するドットはリセット(黒)として表されます。〈タイルistring〉がn文字の長さであれば、その文字列の表す模様は、このようにして決定される横8ドットのパターンを縦にn文字分だけ並べたパターンになります。

例) CHR\$(&HAA)+CHR\$(&H55)

16進数表現	2進数表現	ドットパターン
&HAA	10101010	●○○●○○●○
&H55	01010101	○●○●○●○●

2文字からなる簡単なタイルistringの例です。この2文字は1であるビットと0であるビットが互いに逆になっています。このパターンを基本タイルとして指定された領域を埋めれば、細かい市松模様になりますが、スクリーンのドットが大変細かいため、灰色のように見えます。

●カラーモードの場合

白黒モードと同様に、タイルの模様は〈タイルistring〉に対応するビットパターンによって決定されます。白黒モードと異なる点は、8色中・8色モードおよび4096色中・8色モードの場合は3文字分ごとに、また4096色中・16色モードの場合は4文字分ごとに、タイルの横8ドット(1ライン分)の色がパレット番号として決定されることです。色の決定のされ方については、次の例を参照してください。

例) 8色中・8色モードあるいは4096色中・8色モードの場合

CHR\$(&HAA)+CHR\$(&H55)+CHR\$(&HFF)

ドット	1	2	3	4	5	6	7	8	
&HAA	1	0	1	0	1	0	1	0	← 2 ⁰ の桁
&H55	0	1	0	1	0	1	0	1	← 2 ¹ の桁
&HFF	1	1	1	1	1	1	1	1	← 2 ² の桁
対応する パレット番号	5	6	5	6	5	6	5	6	

各ドットについて、縦方向に2進数としてパレット番号に読み換えます。この場合、指定するタイルストリングのうちの最初の文字が最下位ビット(2^0)の桁を表し、以後、 2^1 の桁、 2^2 の桁、となります。

この例では、パレット番号5とパレット番号6が交互に、横8ドットに指定されたことになります。したがって、パレットが初期状態であれば水色と黄色が交互に出る模様になります。

例) 4096色中・16色モードの場合

CHR\$(&HCE)+CHR\$(&H8C)+CHR\$(&HD0)+CHR\$(&H22)

ドット	1	2	3	4	5	6	7	8	
&HCE	1	1	0	0	1	1	1	0	← 2^0 の桁
&H8C	1	0	0	0	1	1	0	0	← 2^1 の桁
&HD0	1	1	0	1	0	0	0	0	← 2^2 の桁
&H22	0	0	1	0	0	0	1	0	← 2^3 の桁
対応する パレット番号	7	5	8	4	3	3	9	0	

前の例と同じように、各ドットについて、縦方向に2進数としてパレット番号に読み換えることにより、表のような結果となります。

注意：カラーモードの場合、〈タイルストリング〉の文字数が3(4096色中・16色モードの場合は4)の倍数でないときには、文字列の後ろから、3(あるいは4)で割った余りの数だけ無視されます。また、3文字(あるいは4文字)に満たない場合には“Illegal function call”エラーとなります。

すでにタイリングされている領域を異なるパターンでタイリングしようとする、時間がかかたり、スタックを消費するために“Out of memory”エラーになったりすることがあります。

また、PAINTはビューポート内でのみ働きますので、ビューポートの境界がPAINT動作の境界とみなされます。また、(Wx, Wy)がウィンドウの外にある場合には、“Illegal function call”エラーとなります。

PEEK

関数

機能	メモリ上の指定番地の内容を読み出します。
書式	PEEK(<アドレス>)
文例	A=PEEK(&H100)

指定されたメモリ上の番地から1バイト(8ビット)のデータを読み出します。

〈アドレス〉は2バイトの値で、0~65535(&H0~&HFFFF)の範囲で指定します。この値はこの命令実行前にDEF SEGで宣言されたセグメントベースからの相対アドレスを意味します。

参照：CLEAR, DEF SEG, POKE, VARPTR

PEN

関数

機能 ライトペンから情報を得ます。

書式 PEN(〈機能〉)

文例 X=PEN(1)

ライトペンが、現在どのような状態にあるかの情報を得ます。

ライトペンからの情報には、ライトペンが押され、光を感じた瞬間のもの(トリガセンス)とライトペンが押されて光を感じている間のもの(レベルセンス)があります。

〈機能〉に指定する値により、次のような情報が得られます。

0 : トリガセンス

ペンが押された瞬間、光を感じたら-1(真)、光を感じないと0(偽)となる。この機能を指定した場合、PEN関数は一度真になるとON PEN GOSUBで定義されている割り込み処理ルーチンに入るまでその値を保持し、処理ルーチンに入ったときに偽となる。

1 : レベルセンス

ペンが押されて光を感じている限り、現在ペンが置かれているキャラクタ座標の水平座標を得る。

2 : レベルセンス

ペンが押されて光を感じている限り、現在ペンが置かれているキャラクタ座標の垂直座標を得る。

注意：〈機能〉に指定できる値は0~2の範囲で、それ以外は“Illegal function call”エラーとなります。またこの関数はPEN ONの状態でなければ値を得ることができません。

参照：ON PEN GOSUB, PEN ON/OFF/STOP

PEN ON/OFF/STOP

機能 ライトペンによる割り込みの許可, 禁止, 停止を制御します。

書式 1)PEN ON
2)PEN OFF
3)PEN STOP

文例 PEN ON

書式 1) 割り込みを許可します。以後ライトペンを押すごとに割り込みが起こり, ON PEN GOSUB で定義されている処理ルーチンに分岐します。

書式 2) 割り込みを禁止します。以後ライトペンを押しても処理ルーチンへの分岐は起こりません。

書式 3) 割り込みを停止します。以後ライトペンを押しても, そのことを覚えているだけで, 処理ルーチンへの分岐は起こりません。しかしその後 PEN ON によって割り込みが許可されると, 前にライトペンを押したことが有効となり, 処理ルーチンに分岐します。

注意: プログラム終了時には PEN OFF を実行するようにしてください。

参照: ON PEN GOSUB, PEN

PLAY

S

機能 音楽演奏を行います。

書式 PLAY [#<モード番号>,] [<文字列 1>] [, <文字列 2>] [, <文字列 3>] [, <文字列 4>] [, <文字列 5>] [, <文字列 6>]

文例 PLAY #0, "L405CDEFG", "L403GAB>CD"

<文字列 1~6> で指定する MML (ミュージックマクロランゲージ) の指示に従って, 音楽を演奏します。

<文字列 1~3> はチャンネル 1~3 に対応する FM 音源, <文字列 4~6> はチャンネル 4~6 に対応する SSG 音源で, 合わせて同時に 6 音までの同時演奏が可能です。

#<モード番号> はチャンネル 3 の FM 音源に対しどのようなモードで演奏するかを指定するものです。指定する値とモードの関係は次のとおりです。

#0 : 楽音モード (通常の演奏に適する)

- #1 : 効果音モード (効果音に適する)
 #2 : CSM モード (複合正弦波合成モード)

〈文字列1~6〉にはそれぞれ、MML のコマンドを文字列で指定します。MML には次のようなコマンドがあります。

Cx, Dx, Ex, Fx, Gx, Ax, Bx

音程をアルファベットで指定します。音長の指定は、各音程のあとの x (音長数字)で行います。数字の意味は Lx コマンド (後述) と同じで、4 は 4 分音符を表します。このコマンドで音長の指定を省略した場合は、Lx コマンドにより指定された音長をとります。

Mx SSG 音源 (チャンネル 4~6) のエンベロープ周期を設定します。x の値は 1~65535 の範囲になくてはなりません。初期値は "M255" です。



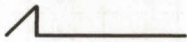

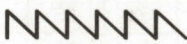
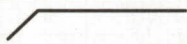


x に指定する値は、発生しようとするエンベロープ周期に対して、次の式により求められます。

$$x = 667 * T / 256$$

T : エンベロープ周期 (ms)

x : 設定値

Sx SSG 音源 (チャンネル 4~6) のエンベロープ形状を指定します。x の値は 1~15 です。x の値とエンベロープ形状の対応は次の図のとおりです。初期値は "S1" です。

x	エンベロープ形状	x	エンベロープ形状
0, 1, 2, 3, 9		11	
4, 5, 6, 7, 15		12	
8		13	
10		14	

SSG 音源には固定音量出力モードと Mx コマンド、Sx コマンドで指定される可変出力モードがありますが、各チャンネルを可変出力モードにするのは Sx コマンドです。Mx コマンドは周期を設定するだけでモードは変えませんが、固定音量出力モードにするのは、後述する Vx コマンドです。

Vx 各チャンネルに対して大まかに音量を設定します。

"V0" が最小で "V15" が最大音量です。SSG 音源に対する初期値は "V7" です。FM 音源については各音色ごとに最も適した音量が設定されるので初期値は存在しません。また、FM 音源については音量の大小により音色も多少変化します。@x コマンドで音色番号を切り換えると、@Vx コマンド、Vx コマンドによる設定は無効になります。

Lx このコマンドで音の長さを指定すると、Cx, Dx, Ex, Fx, Gx, Ax, Bx コマンドで音長を省略したときには、すべてこの長さになります。

x は 1~64 の範囲で、1 は全音符、8 は 8 分音符といったように指定します。このコマンドによる設定を省略した場合の初期値は、"L4" (4 分音符) です。

Qx 音長により定められる長さ (ステップタイム) と、その間で実際に音を出している長さ (ゲートタイム) の比を設定します。x の値は 1~8 の範囲で、ゲートタイム / ステップタイムの比は $x/8$ で求められます。初期値は "Q7" です。

Ox オクターブ値を設定します。x の値は 1~8 で、数値が大きいほどオクターブは高くなります。初期値は "O4" です。

<, > 現在のオクターブを 1 オクターブだけ上げ (>), 下げ (<) します。

Kx x で指定された高さの音を発生します。x (キー番号) と実際に発生される音程の対応は次のとおりです。ただし、FM 音源においては "O9C" は "O1C" と同じになります。

オクターブ 音程	O1	O2	O3	O4	O5	O6	O7	O8	O9
C	0	12	24	36	48	60	72	84	96
C [#]	1	13	25	37	49	61	73	85	
D	2	14	26	38	50	62	74	86	
D [#]	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 [#]	6	18	30	42	54	66	78	90	
G	7	19	31	43	55	67	79	91	
G [#]	8	20	32	44	56	68	80	92	
A	9	21	33	45	57	69	81	93	
A [#]	10	22	34	46	58	70	82	94	
B	11	23	35	47	59	71	83	95	

Tx 演奏するテンポを指定します。どのチャンネルでテンポを指定してもかまいません。ただし、全チャンネルを通してテンポはただ 1 つしか指定できないので、最後に指定したチャンネルのテンポが有効となります。x の範囲は 32~255 までで、"T120" が初期値です。T120 は "♩ = 120" に相当します。

Rx, Px

休符です。x の値は 1~64 で、音長を指定します。1 が全休符、4 が四分休符といったように指定します。

+ (#), -

いずれも音程 (Cx, Dx, Ex, Fx, Gx, Ax, Bx) のあとにつけて半音階を表します。"+" または "#" で半音上げ, "-" で半音下げます。

. (ピリオド)

音程 (Cx, Dx, Ex, Fx, Gx, Ax, Bx) に指定する音長数字のあとにつけることによって、符点音符を表します。休符 (Rx, Px) のあとにつけて符点休符とすることもできます。

^x 前後の音長をつないで 1 つにします。x の値は 1~64 で、音長を指定します。たとえば、"C8^8" のように表現される音は "C4" と等しくなります。

{ ... } x

連符を表します。x で指定された音長を { } 内の音符の個数で等分します。ただし、{ } 内で音長を指定した場合は正しい連符となりません。また、連符の中に連符や間接指定を含めることもできません。

@x x で示される音色番号の音に音色を切り換えます。

x の範囲は 0~81 です。このコマンドは、FM 音源チャンネルについてのみ有効です。音色番号と音色の対応は、初期状態では次のようになっています。音色番号の初期値は "@0" です。

内 蔵 基 本 音 色

音色番号	音色名	説 明	備 考
0	DEFAULT	(HARPSIC-11 番と同じ)	
1	BRASS 2	低域の金管楽器	
2	STRING 2	高域の弦楽器	
3	EPIANO 3	丸みを持ったエレクトリックピアノ	
4	EBASS 1	シンセサイザベース	
5	EORGAN 1	堅めのエレクトリックオルガン	
6	PORGAN 1	低域のパイプオルガン	
7	FLUTE	フルート	O5 が最適です

音色番号	音色名	説明	備考
8	OBOE	オーボエ	
9	CLARINET	クラリネット	
10	VIBRAPHON	ビブラフォン	
11	HARPSIC	ハープシコード	
12	BELL	ベル	
13	PIANO	アコースティックピアノ	
14	MUSHI	虫の鳴き声	
15	DESCENT	下降するような効果音	
16	UFO	UFOが飛んでいるような効果音	
17	GRANPRI	レーシングカーが走って行くような効果音	
18	LASER 1	SF的な効果音 1	
19	LASER 2	SF的な効果音 2	
20	SINWAVE	正弦波	
21	BRASS 1	高域の金管楽器	
22	BRASS 2	低域の金管楽器	
23	TRUMPET	トランペット	O5が最適です
24	STRING 1	低域の弦楽器	
25	STRING 2	高域の弦楽器	
26	EPIANO 1	エレクトリックピアノ	
27	EPIANO 2	堅めのエレクトリックピアノ	
28	EPIANO 3	丸みを持ったエレクトリックピアノ	
29	GUITAR	エレキギター	O5が最適です
30	EBASS 1	シンセサイザベース	
31	EBASS 2	ウッドベースに近い音	O3が最適です
32	EORGAN 1	堅めのエレクトリックオルガン	
33	EORGAN 2	やや丸いエレクトリックオルガン	
34	PORGAN 1	低域のパイプオルガン	
35	PORGAN 2	高域のパイプオルガン	
36	FLUTE	フルート	O5が最適です
37	PICCOLO	ピッコロ	
38	OBOE	オーボエ	
39	CLARINET	クラリネット	
40	GROCKEN	グロックン	
41	VIBRAPHON	ビブラフォン	
42	XYLOPHN	シロフォン	
43	KOTO	琴	
44	ZITAR	チター	

音色 番号	音色名	説 明	備 考
45	CLAVINET	クラビネット	
46	HARPSIC	ハーブシコード	
47	BELL	ベル	
48	HARP	ハーブ	O5 が最適です
49	BELL/BRASS	ベルと金管楽器のユニゾン	
50	HARMONICA	ハーモニカ	
51	STEELDRUM	スチールドラム	
52	TIMPANI	ティンパニー	O3 が最適です
53	TRAIN	汽笛の音	
54	AMBULANCE	救急車の音	長めの音が適します
55	TWEET	小鳥の声	O6, Q4 が最適です
56	RAIN DROP	雨だれの音	長めの音が適します
57	HORN	ホルン	
58	SNARE DRUM	スネアドラム	
59	COW BELL	カウベル	
60	PERC 1	バネのはじけるような音	
61	PERC 2	水の入ったグラスをはじくような音	O5 が最適です
62	PERC 3	ウインドチャイムのような音	O6 が最適です
63	MUSIC BOX	オルゴールのような音	
64	CELLO	チェロ	
65	LOW BRASS	チューバのような低域の金管楽器	
66	WW ENSMBL	木管楽器のアンサンブル	
67	AC GUITAR	アコースティックギター	
68	FLUTE/HARP	フルートとハーブのユニゾン	
69	FUNK PLUC	打弦楽器の音	
70	FUNK BASS	ファンキーなベースの音	
71	SYN LEAD	メロディーライン向けのリードに適した音	
72	METAL CLES	金属的な効果音 1	O3 が適します
73	STAIN	金属的な効果音 2	O3 が適します
74	CUBIN GLASS	氷がグラスの中を転がるような音	O5 が適します
75	HUMAN	人の声のような音	
76	WOOD BASS	ウッドベース	O3 が最適です
77	CHIMES	チャイム	O5 が最適です
78	SPACY	機械的な口笛の音	O5 が最適です
79	OBOE/B.CLAR	オーボエとバスクラリネットのユニゾン	
80	OLD STRING	古いレコードのバイオリンのような音	
81	STEEL'S CRY	金属的な泣き声のような音	

備考の欄に特に記入のないものは、どのような音域でもよいか、またはデフォルトが最も適していることを示します。

Yr, d

シンセサイザ LSI のレジスタに直接値を設定します。VOICE REG と同じ機能を行うものです。

r はレジスタ番号で 0~178, d はレジスタ内容で 0~255 の範囲です。定義されていないレジスタに対するこのコマンドの動作の保証はありません。また、サウンド制御命令がハードウェアの制御のために使用しているレジスタについて書き込みを行った場合も、動作の保証はありませんので注意してください。

@Vx FM 音源に対する音量を細かく設定します。

x は 0~127 の範囲で、0 が最小音量、127 で最大音量となります。

@Wx x で指定された長さだけ何もしないで待ちます。x は 1~64 の範囲で、1 は全音符、8 は 8 分音符といったように指定します。

@Wx コマンドは Yr, d コマンドなどで直接音を発生させる場合に有効です (Rx, Px コマンドでは、一旦発音を停止させてしまいますので音がとぎれてしまいます)。普通の音符 (Cx, Dx, Ex, Fx, Gx, Ax, Bx) のあとに @Wx がきた場合は、Rx, Px コマンドと同等です。

_x 移調を指示します。x は音程で、A, B, C... のように指定します。x によって示される音程を、C の音とみなして以降の MML を移調します。各チャンネルごとに独立して指定できます。ただし、Kx コマンドによる指定は移調しません。

移調は次の規則により行われます。

- 音程 (A, B, C, D...) をキー番号 (0~96) に直します。
- その値に、次表に示すようなベース値を加えて対応するキー番号の音を発生します。

指定調	G ^b	G	A ^b	A	B ^b	B	C	C [#]	D	D [#]	E	F
ベース値	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5

たとえば、PLAY "_DDEFG" と指定した場合、実際に発生される音は、PLAY "EF#GA" と等しくなります。

! LFO 効果 (ビブラート, トレモロ) を加えます。

音色番号指定直後や、VOICE LFO 実行直後は、LFO 効果が自動的に加えられています。ただし、音色データの値によって、LFO 効果がかからない場合があります。

* !コマンドの逆で、LOF 効果を切ります。

Zp, v 指定チャンネルの音色パラメータの内容を設定します。

p はパラメータ番号で、0～49 の範囲です。また、v はパラメータの内容で 0～16383 までの値です。p のパラメータ番号は、VOICE や VOICE COPY における音色用配列の添字に、次のように対応します。

$$p = (\text{第 1 添字}) + (\text{第 2 添字}) * 5$$

たとえば、音色用配列 (3, 1) に対応するパラメータ番号は、次のようになります。

$$3 + 1 * 5 = 8$$

v の値については、各音色パラメータの有効範囲外の値を設定しようとした場合、動作は保証されません。また、負の値は 8 ビットの符号付数値に直して指定してください。たとえば、-5 を指定したいときは、次のように 256 を加えた値を指定します。

$$256 + (-5) = 251$$

MF, MB

PLAY の演奏形態を指定します。

"MF" を指定すると、PLAY による演奏が終了してから次の命令の実行へ移ります。これを"フォアグラウンド指定"といいます。初期状態は"MF"になっています。

"MB" を指定すると、PLAY の音楽情報をサウンドバッファに格納してすぐに次の命令に移ります。これを"バックグラウンド指定"といいます。"MF"、"MB"ともに各チャンネルごとに指定できますが、"MF"で指定されたチャンネルと"MB"で指定されたチャンネルが同時にあった場合、"MF"指定のチャンネルが終了するまで次の命令へは移れません。"MB"が指定された場合であっても、次の条件が起これると演奏は中断されます。

- **STOP** キーまたは STOP の実行
- PLAY ALLOC, ERROR の実行
- エラーの発生による中断

MML の間接指定

MML の一部を文字変数や数値変数で置き換えることも可能です。このことを間接指定といいます。間接指定には、次のものがあります。

● 数値の置き換え

"L=A;" のように "=" + (変数名) + ";" で表します。ただし、変数名は配列の要素であったり型宣言文字 ("!" や "%" など) がついていてはいけません。

● 文字列の置き換え

"X=AS;" のように "X=" + (文字変数名) + ";" で表します。ただし、変数名は配列の要素であってはなりません。

注意：PLAY を使用する際には、次のことに注意してください。

- (1) PLAY を実行する前には、必ず PLAY ALLOC を実行して、サウンドバッファを各チャンネルのために用意しておかなければなりません。
- (2) エラー発生時には音色は自動的に 0 番に設定されます。
- (3) PLAY により演奏される音長は他の処理などの影響により若干長くなることがあります。
- (4) MB による演奏時、BASIC プログラムの実行は若干遅くなります。
- (5) MF による演奏時に連続して PLAY を実行させた場合、命令解析の時間のために MML の最後の音長が長めになったり、演奏が息つきをしているように聞こえることがあります。
- (6) 音量が大きすぎると音がひずむおそれがあります。また、2 チャンネル以上を使って演奏を行う場合、1 チャンネルで演奏するときと比べ音量が増し、音がひずむ場合があります。このようなときは、@Vx コマンドで音量を調節してください。
- (7) MML の中で、音を各種調節するコマンドをひんばんに使用する(たとえば、1 音ごとに音量を設定する)とリズムが少し遅くなります。

参照：VOICE, VOICE COPY, VOICE REG, PLAY ALLOC, サンプルプログラム 41

PLAY ALLOC

S

機能	サウンドバッファの確保および初期化をします。
書式	PLAY ALLOC [<CH1 バッファサイズ>] [, <CH2 バッファサイズ>] [, <CH3 バッファサイズ>] [, <CH4 バッファサイズ>] [, <CH5 バッファサイズ>] [, <CH6 バッファサイズ>]
文例	PLAY ALLOC 255, 255, 255

音楽演奏のためのサウンドバッファを、<CH1 バッファサイズ>～<CH6 バッファサイズ>で指定されたバイト数分、各チャンネル毎に確保します。省略されたチャンネルのバッファサイズは 0 となります。

PLAY ALLOC はサウンド命令を使用する前に必ず一度実行しなければなりません。確保しようとするバッファサイズ(各チャンネルの合計値)は必ず CLEAR で確保した機械語プログラム領域より小さくしなければなりません。

また、PLAY ALLOC はサウンド機能を初期化します。初期値は次のとおりです。

SSG エンベロープ : M255, S1 (ただし、固定出力にされている)

SSG 音量	: V7
音長	: L4 (R4)
音の長さの割合	: Q7
オクターブ	: O4
テンポ	: T120
FM チャンネル音色	: 0 (初期状態の音色)
音色バンク	: BASIC 起動時の状態に設定

参照 : PLAY, サンプルプログラム 41

PLAY ON/OFF/STOP

S

機能 PLAY 割り込みの許可, 禁止, 停止を制御します。

書式

- 1) PLAY ON
- 2) PLAY OFF
- 3) PLAY STOP

文例 PLAY ON

書式 1) PLAY 割り込みを, この命令の次の命令を実行した後, 許可します。その後, ON PLAY GOSUB で定義した条件が満たされるごとに割り込みが発生し, ON PLAY GOSUB で定義されている処理ルーチンへ分岐します。

書式 2) PLAY 割り込みを禁止します。この命令が実行されると, 割り込み条件が満たされても割り込みは起こらず処理ルーチンには分岐しません。

書式 3) PLAY 割り込みを停止します。この命令が実行されると, 割り込み条件が満たされても割り込みは起こらず処理ルーチンには分岐しません。しかし, 割り込みのあったことを覚えているため, 後で PLAY ON によって割り込みが許可されると, 処理ルーチンに分岐します。

注意 : PLAY の割り込みは, BASIC の他の割り込み命令と異なり, 割り込み処理ルーチンに制御が移っても自動的に割り込み停止状態となりません。したがって, 処理ルーチンの先頭では PLAY OFF または PLAY STOP を実行し, 終わりで PLAY ON を実行する必要があります。

参照 : ON PLAY GOSUB, PLAY, サンプルプログラム 41

POINT

機能 LP(最終参照点)を変更します。

書式 POINT (Wx, Wy) |
STEP(x, y) |

文例 POINT (-200, 30)
POINT STEP(20, 20)

最後にグラフィック操作の行われた座標を「最終参照点」(Last Referenced Point, 略してLP)といいます。LPは相対座標により座標指定を行う際の基点となります。グラフィック関係の命令はほとんど、実行することにより、描画と同時にこのLPも変更します。

POINTはワールド座標(Wx, Wy)でLPをたんに設定,あるいは変更する働きをします。実際に図形を描いたり,図形に変化を与えることはありません。

STEPをつけると前のLPからの相対座標による指定となります。

例1) LINE(-200, 30)-(100, 120), 3

例2) POINT(-200, 30)
LINE -STEP(300, 90), 3

例1と例2は,まったく同じ図形を描きます。

参照: LINE, [1] POINT(関数), サンプルプログラム 17

[1] POINT

関数

機能 LP(最終参照点)の値を得ます。

書式 POINT(<機能>)

文例 WX=POINT(0)

最後にグラフィック操作の行われた座標(最終参照点, 略してLP)を, <機能> に指定する値により, スクリーン座標あるいはワールド座標として得ます。

<機能> には0~3の値を指定します。

- 0 : ワールド座標系の値に変換されたLPのX座標
- 1 : ワールド座標系の値に変換されたLPのY座標
- 2 : スクリーン座標系の値に変換されたLPのX座標
- 3 : スクリーン座標系の値に変換されたLPのY座標

LP がウィンドウの外であった場合、POINT 関数によって得られる座標に対応する点は、画面上に表示されていない点であることに注意してください。またこのとき、得られた値がスクリーン座標系に変換された座標値であれば、ビューポート外になることに注意してください。

参照：POINT

〔2〕 POINT

関 数

機 能	スクリーン座標系の指定された座標に表示されているドットの色を得ます。
書 式	POINT(Sx, Sy)
文 例	COL=POINT(473, 120)

スクリーン座標系の(Sx, Sy)で指定された座標に表示されているドットの色を、パレット番号で得ます。(Sx, Sy)がビューポートの外にある場合、〔2〕POINTの値は-1となります。

白黒モードの場合、ドットの色はリセット(0:黒を表す)またはセット(1:白を表す)で表され、〔2〕POINTで得られる値はこのいずれかとなります。

POKE

機 能	メモリ上の指定番地へデータを書き込みます。
書 式	POKE <アドレス>, <式>
文 例	POKE &HF000, &HFF

指定されたメモリ上の番地に1バイト(8ビット)のデータを書き込みます。

<アドレス>は2バイトの値で、0~65535(&H0~&HFFFF)の範囲で指定します。この値はこの命令実行前にDEF SEGで宣言されたセグメントベースからの相対アドレスを意味します。

<式>は書き込まれるデータで、0~255(&H0~&HFF)の値でなければなりません。もし小数点以下があると整数化(小数点以下を四捨五入)してから実行されます。

注意：この命令は、現在のメモリの内容を書き換えてしまうため、不用意に使うとBASICが使っている作業領域を壊してしまい、誤動作の原因となることもあります。使うときには、メモリマップなどで使用可能な領域かどうかを確認してください。

参照：DEF SEG, PEEK, VARPTR

POS

関 数

機 能 テキスト画面上の現在のカーソルの桁位置を得ます。

書 式 POS(<数式>)

文 例 CX=POS(0)

現在のカーソルの桁(水平)位置をキャラクタ座標で得ます。値の範囲は 0~79(または 39)となります。得られる値は、0 が画面の左端の桁を、79(または 39)が右端の桁を、それぞれ意味します。

<数式>はダミーであり、どんな数式を指定しても結果に変わりはありませんが、省略することはできません。

参照：CSRLIN, サンプルプログラム 10

PRESET

機 能 画面上の任意の座標のドットを消去します。

書 式 PRESET | (Wx, Wy) | [, <パレット番号>]
| STEP(x, y) |

文 例 PRESET (100, 100)

ワールド座標(Wx, Wy)で示されるドットを消去します。

(Wx, Wy)の代わりに、STEPをつけて相対座標(x, y)で指定することもできます。

この命令は、オプションの<パレット番号>をつけない方が一般的な使い方です。その場合、(Wx, Wy)で表される座標のドットを、現在 [1] COLOR によって設定されているバックグラウンドカラーでぬり変えます(すなわちドットが消える)。

<パレット番号>を指定した場合は、PSET とまったく同じに機能し、そのパレット番号のカラーでドットを表示します。

PRESET を実行すると、LP(最終参照点)は、指定した座標点に変更されます。

参照：[1] COLOR, サンプルプログラム 17

PRINT

機能 画面にデータを出力します。

書式 PRINT [<式>] [, | <式> …] [, |]
? ; ;

文例 PRINT "COLUMN=" ; X, "LINE=" ; Y

<式>に、数値や文字列を指定すると、その数値、文字列がそのまま出力され、数値変数や文字変数が指定されると、それらの変数に代入されている数値や文字列が出力されます。<式>が省略された場合には改行のみを行います。

数値を出力する場合、その後ろに必ず空白が入ります。また、数値の前には符号用の桁が用意されていて、プラスのときには空白、マイナスのときにはマイナス符号が表示されます。

複数のデータを表示する場合、区切り記号として、コンマ(,)、セミコロン(;), 空白を使います。データを表示する領域は、あらかじめ各行を14文字ごとに分割して定められており、区切り記号によって表示のしかたが変わります。

コンマ(,)を使った場合は、次の領域から表示し、セミコロン(;)あるいは空白を用いた場合には、直前に出力したもののすぐ後ろに出力します。

ダブルクォーテーション(")で囲まれた文字列の前後に、他の文字列や変数を並べる場合、その間の区切り記号を省略することができます。このときは、セミコロンと同様の働きをします。

<式>の最後にセミコロンやコンマを指定すると改行が起こらず、その行で次のPRINTによる出力を続行します。

PRINTの短縮形として疑問符(?)を使うことができます。プログラム内で"? "を用いた場合、後でLISTを行うと、自動的に"PRINT"に変換されます。

注意: 単精度の数値で、指数形式(浮動小数点形式)でなくても6桁以下の桁数で精度に影響をおよぼさず表示できるものは、通常的小数点形式(固定小数点形式)で表示されます。同様に倍精度の数値で16桁以下の桁数で精度に影響をおよぼさず表示できるものは、固定小数点形式の表示になります。また、表示する数値の長さが現在のカーソルの位置より後方にとれない場合(それを表示すると次の行にわたってしまう場合)は改行してからそれを表示します。

参照: LPRINT, PRINT USING, サンプルプログラム12


```
A$="CAMERA"
B$="93604-1"
PRINT #1, A$; ", " ; B$
```

とすると、ディスクには次のように書き出されます。

```
CAMERA, 93604-1
```

","を使わないで、PRINT #1, A\$; B\$とした場合には、ディスクにはCAMERA93604-1と書き出されてしまいます。これでは区切りの記号がありませんから2つの別の文字列として読み込むことはできません。

文字式それ自身がデータとしてコンマ、セミコロン、意味のある始めの空白、キャリッジリターン(CHR\$(13)), またはラインフィード(CHR\$(10))などを含む場合は、ダブルクォーテーションコード(CHR\$(34))によって囲んでディスクに書き出さなければなりません。たとえば、

```
A$="CAMERA, □ AUTOMATIC"
B$="□□□ 93604-1"
PRINT #1, A$; ", " ; B$
```

とするとディスクには、

```
CAMERA, □ AUTOMATIC, □□□ 93604-1
```

と書き出されます。これではA\$の中のコンマがデータの区切りとなってしまう、INPUT #1, A\$, B\$としてデータを読み込むと、A\$にはCAMERAを、B\$には□ AUTOMATICを読み込んでしまい、□□□ 93604-1は読み込まれな残ってしまいます。

これを避けるには、

```
PRINT #1, CHR$(34) ; A$ ; CHR$(34) ; CHR$(34) ; B$ ; CHR$(34)
```

と指定します。この場合ディスクには、

```
"CAMERA, □ AUTOMATIC" "□□□ 93604-1"
```

と書き出されますので、INPUT #1, A\$, B\$とすれば、"CAMERA, □ AUTOMATIC"をA\$に、"□□□ 93604-1"をB\$に読み込むことができます。

注意：出力対象ファイルが"SCRN:"の場合、日本語データの出力はできません。

参照：INPUT #, PRINT # USING, WRITE #, サンプルプログラム 28

PRINT USING

機能

文字列、数値などのデータを編集し、画面に出力します。

書式

PRINT USING <書式制御文字列>; <式> [| , | <式> ...] [| , |]
; ;

文例

PRINT USING "#####. "; A, B, C

<書式制御文字列>によって決定される領域や書式に従って、<式>に指定された各種データを画面に出力します。複数の<式>を並べる場合、区切り記号としてはコンマ(,)またはセミコロン(;)のいずれを使っても結果は変わりません。

<書式制御文字列>中に指定できる書式制御文字は、次のとおりです。

●文字変数や文字列の書式制御文字

!与えられた文字変数や文字定数の最初の1文字だけを出力します。

& <n個の□(空白)> &...与えられた文字変数や文字定数の先頭から(n+2)文字の文字列を出力します。与えられた文字列が(n+2)文字より長い場合は余分な文字は無視され、短い場合には文字列は左づめで出力され、残った部分には空白が出力されます。

@文字列全体の出力に使います。複数個指定した場合、1つの"@ "に対して<式>の中の1つの文字列が、代入され出力されます。"@ "の数が<式>の個数より多い場合は、余った"@ "は無視されます。

注意：日本語を含む文字列を編集することはできません。

●数値の書式制御文字

.....数値を出力する桁数を指定します。指定した桁数より数値の桁数が小さいときには、右づめで出力されます。

. (ピリオド) ...小数点の位置を指定します。小数点以下の部分で冗長となる桁には0が出力されます。

+ <書式制御文字列>の最初または最後につけた場合、数値の符号がそれぞれ前または後ろに出力されます。2個以上の"+ "を並べた場合には、余分は後述の制御文字以外の文字と同じ扱いとなり、そのまま出力されます。

- (マイナス記号) ... <書式制御文字列>の最後につけた場合、数値が負の数のときに数値の後ろに"- "が出力されます。前につけたり、2個以上並べた場合には後述の制御文字以外の文字と同じ扱いとなり、そのまま出力されます。

* * …… 〈書式制御文字列〉の先頭につけた場合、数値領域の左側に空白部分ができたとき、そこを“*”で埋めて出力します。この“**”は2桁分の領域を確保します。

¥ ¥ …… 〈書式制御文字列〉の先頭につけた場合、出力される数値の直前に“¥”を出力します。“¥¥”は2桁分の領域を確保しますが、このうち1桁分は“¥”の出力領域として使われます。後述の指数形式の書式指定を行った場合には、正しく出力されないことがあります。

* * ¥ …… 〈書式制御文字列〉の先頭につけた場合、上記の2つ(**と¥¥)両方の機能となります。“**¥”は3桁分の領域を確保しますが、このうち1桁分は“¥”の出力領域として使われます。

, (コンマ)…桁数指定の“#”の並びの中においた場合、数値の整数部が3桁毎に“, ”で区切られて出力されます。ただし、上記の“, ”より右側においた場合は、数値の最後に“, ”が出力され、3桁毎の区切りは行われません。

^^^ ……桁数指定の“#”の後につけた場合、指数形式(浮動小数点形式)で出力されます。

●その他の書式制御文字

_ (下線)…前述の制御文字1文字をたんに文字として表示するために使用します。“_”に続く1文字は常に書式制御機能を持たない文字として出力されます。

●制御文字以外の文字

前述の制御文字以外の文字を指定した場合、数値の前や後ろにそのキャラクタが出力されます。ただし、日本語文字は指定できません。

●数値の領域を超えた場合

指定した数値領域より数値の桁数が大きい場合、数値の直前に“%”が出力されます。数値を丸めた(四捨五入した)ことによって桁数が領域より大きくなった場合も、丸めた数値の前に“%”が出力されます。

●複数の書式制御文字を指定した場合

〈書式制御文字列〉中に複数の書式制御文字を指定した場合、その制御文字の並びが1つのパターンとなって、指定された複数の〈式〉に対し、順に繰り返してあてはめられます。

たとえば、

```
PRINT USING "@=¥¥### "; "BOOKS", 2500, "TICKETS", 1440, "DRINKS", 4300
```

とすると、“@=¥¥### ”のパターンが以降に指定される式の並びにあてはめられる結果、

PRINT USING, PRINT # USING, PSET

BOOKS=¥2500 TICKETS=¥1440 DRINKS=¥4300

というように出力されます。

参照：LPRINT USING, PRINT, サンプルプログラム 13

PRINT # USING

機能 文字列、数値などのデータを編集し、ファイルに出力します。

書式 PRINT # <ファイル番号>, USING <書式制御文字列>; <式> [| , | <式> ...]
; |

[| , |]
; |

文例 PRINT #2, USING "@####"; CASE\$, NUMBER

<ファイル番号>には、OPEN によって、そのファイルをオープンしたときに使った番号を指定します。

PRINT # USING は、<式> に指定した文字列や数値を <書式制御文字列> の形式にもとづいて編集し、ファイルに出力します。

この命令は、その対象がファイルであることを除けば PRINT USING と機能は同じです。

参照：LPRINT # USING, OPEN, PRINT #, PRINT USING

PSET

機能 画面上の任意の座標にドットを表示します。

書式 PSET (Wx, Wy) [, <パレット番号>]
STEP(x, y)

文例 PSET (100, 100), 4

ワールド座標(Wx, Wy)の位置にドットを表示します。

(Wx, Wy)の代わりに、STEP をつけて相対座標(x, y)で指定することもできます。

<パレット番号>でドットの色を指定します。省略した場合は、現在 [1] COLOR によって設定されているフォアグラウンドカラーが採用されます。

PSET を実行すると、LP(最終参照点)は、指定した座標点に変更されます。

参照：[1] COLOR, [2] COLOR, POINT, PRESET, サンプルプログラム 6, 17

PUT

機能	ファイルバッファ中のデータをファイルに書き出します。
書式	PUT [#] <ファイル番号> [, <数式>]
文例	PUT #3, COUNT PUT 2

<ファイル番号> で指定されたファイルに、対応するバッファの内容を書き出します。

PUT は、指定されたファイルがディスクファイルか、またはプリンタあるいはスクリーンファイルかによってその動作が異なります。

(1) ディスクファイルの場合

PUT はランダムバッファ中のデータをランダムファイルに書き出します。指定されたファイルはランダムモードでオープンされていなければなりません。

<数式> はファイルのレコード番号として解釈され、指定されたレコードにバッファ中のデータが書き出されます。レコード番号の最小値は 1、最大値は 65000 です。<数式> が省略された場合には、直前に行われた、GET、PUT で指定されたレコードの次のレコードに書き出します。

(2) プリンタ(LPT :, LPT1 :)あるいはスクリーンファイル(SCRN :)の場合

バッファ中のデータを、プリンタあるいは画面に出力します。指定されたファイルは出力モードでオープンされていなければなりません。

<数式> は、バッファから、ファイルに対して書き出す文字(バイト)数と解釈されます。0 から 255 までの値で指定します。<数式> が省略されたとき、および 0 が指定されたときには 256 文字を書き出します。

注意：書き出すデータはあらかじめ FIELD、LSET/RSET により準備しておかねばなりません。

参照：FIELD、GET、LSET/RSET、OPEN、サンプルプログラム 1、26、29

PUT@

機能	グラフィックパターンや漢字を画面に表示します。
書式	1) PUT[@] (Sx, Sy), <配列変数名> [(<添字>)] [, <条件>] [, <フォアグラウンドカラー>, <バックグラウンドカラー>] 2) PUT[@] (Sx, Sy), KANJI(<漢字コード>) [, <条件>] [, <フォアグラウンドカラー>, <バックグラウンドカラー>]
文例	PUT@ (100, 100), G%, PSET PUT (0, 0), KANJI(&H3021)

書式 1)

GET@によって配列変数に読み込まれたグラフィックパターンを、画面上の指定座標に表示します。

座標(Sx, Sy)はGET@の場合と同様で、ワールド座標でなくスクリーン座標で指定します。

<配列変数名>には、表示したいグラフィックパターンが格納されている配列変数の名前を指定します。

<添字>は、配列変数内に格納されている複数のグラフィックパターンのうち、どの要素から表示し始めるのかを指定するものです。省略した場合は、配列の最初から表示し始めます。<添字>の指定は、GET@でパターンを格納したときの値と対応させるようにしてください。

<条件>とは、グラフィックパターンを画面に表示する際のいろいろな条件を指定するものです。

- PSET** : 配列内のグラフィックパターンをそのまま表示します。
- PRESET** : 白黒モードの場合は配列内のパターンを反転して表示します。カラーモードの場合は各ドットのパレット番号を、7-(そのドットのパレット番号)(4096色中・16色モードの場合は、15-(そのドットのパレット番号))として表示します。
- OR** : 配列内のグラフィックパターンと、すでにある画面上のグラフィックパターンとをドットごとにOR(論理和)し、その結果を画面に表示します。
- AND** : 配列内のパターンと画面上のパターンをドットごとにAND(論理積)し、その結果を画面上に表示します。
- XOR** : 配列内のパターンと画面上のパターンをドットごとにXOR(排他的論理和)し、その結果を画面に表示します。

<条件>を省略した場合は、XORを指定したものとみなされます。

これらの<条件>は画面モードによって演算の対象が違います。白黒モードの場合、ドットがあるかないかを対象とし、カラーモードの場合、ドットごとのパレット番号を対象とします。たとえばすでに画面にあるドットの色がパレット番号3で、配列内のグラフィックパターンの色がパレット番号6のときに“AND”を指定すると、パレット番号2が表示されます(0011

AND 0110 → 0010).

〈フォアグラウンドカラー〉、〈バックグラウンドカラー〉は、白黒モードのときにGET@で読み込んだパターンをカラーモードにおいて表示するときのみ、有効なオプションパラメータです。この2つのパラメータは両方とも指定するか、両方とも指定しないかのどちらかしか許されません。

〈フォアグラウンドカラー〉は、白黒モードで読み込んだ際に白であったドットに対しての色指定で、〈バックグラウンドカラー〉は、同様に黒であったドットに対しての色指定です。カラーモードにおいて、それぞれをパレット番号によって指定すると任意の色に変えることができます。

書式 2)

〈漢字コード〉で指定された漢字または非漢字をグラフィック画面に表示します。

使用できる文字は、JIS 第一水準、第二水準および利用者定義文字です。これらの中から任意の文字を〈漢字コード〉(JIS コード)によって指定することにより、画面上に日本語文字を表示することができます。使い方および機能は、〈配列変数名〉(〈添字〉)の代わりにKANJI(〈漢字コード〉)を使用することを除き、書式 1)の場合と同様です。

なお、利用者定義文字は、Kpload によってシステムに登録することができます。

書式 1), 2)とも、この命令を実行するとLP(最終参照点)は(Sx, Sy)に移動します。

注意: GET@とPUT@は、〈フォアグラウンドカラー〉、〈バックグラウンドカラー〉を指定して、白黒モードで読み込んだパターンをカラーモードで表示する用途の他は、原則として同一画面モードで使用するようにしてください。

なお、1つのPUT@では1つの漢字しか表示することはできません。

参照: GET@, Kpload, サンプルプログラム 18, 19

RANDOMIZE

機能 新しい乱数系列を設定します。

書式 RANDOMIZE [〈式〉]

文例 RANDOMIZE 230

〈式〉に、新しい乱数の種(seed)を指定することによって、RND関数で得られる乱数系列を変更します。〈式〉には数値や変数などを-32768から32767の範囲で指定します。〈式〉を省略すると、メッセージが表示されてseedを要求してきますので、〈式〉に同様な範囲の値を入力

してください。もし、この範囲外の値を入力すると“Overflow”エラーとなります。

参照：RND, サンプルプログラム 32

READ

機能 DATA で用意した数値や文字のデータを読み込み、変数に代入します。

書式 READ <変数> [, <変数> …]

文例 READ ZIP, ADDR\$

READ は DATA と組み合わせて使わなければなりません。READ は DATA 行中に指定されたデータを 1 対 1 の対応で変数に割り当てていきます。

<変数>は、DATA 行中で指定したデータが文字定数である場合は文字変数でなくてはなりません。しかし、数値定数の場合は文字変数、数値変数のいずれにも読み込ませることができます。

1 つの READ で、複数の DATA 行を順番に参照したり、またいくつかの READ で 1 つの DATA 行を参照したりすることができます。

READ 中の<変数>の数が DATA のデータ数を超過してしまった場合は、“Out of DATA”エラーとなります。指定された<変数>が DATA のデータ数よりも少ない場合には、読まれなかったデータから、その次の READ が読み始めます。もしそれ以上 READ がない場合には、余分のデータは無視されます。

始めから、あるいは途中から DATA を読み直すには、RESTORE を使います。

注意：READ では、型が一致しない場合には、“Type mismatch”エラーではなく、“Syntax error”エラーが起こりますので注意してください。

参照：DATA, RESTORE, サンプルプログラム 3, 14

REM

機能 プログラムに注釈文を入れます。

書式 REM [<注釈文>]

文例 REM *** Main-Program ***

REM に続く文字列は、プログラムの注釈になるだけで、プログラムの実行にまったく影響を与えません。LIST を実行すると入力した内容がそのまま出力されます。REM の代わりにアポストロフィ(')を使うこともできます。

注意：REM に続く注釈文の後に、コロン(:)で区切って他の命令を続けても無視されてしまいます。

RENUM

機能	プログラムの行番号を新しくつけ直します。
書式	RENUM [<新行番号>] [, <旧行番号>] [, <増分>]
文例	RENUM 1000, 10

<新行番号> は、新しくつける行番号の最初の行番号で、省略すると 10 が採用されます。

<旧行番号> は行番号のつけ替えを始める現在のプログラムの行番号です。省略するとそのプログラムの最初の行番号が採用されます。

<増分> は新しくつける各行番号のあいだの増分で、省略すると 10 となります。

RENUM は、GOTO, GOSUB, THEN, ON...GOTO, ON...GOSUB および ERL で参照している行番号も新しい行番号に対応して変更します。もし、これらの命令の参照する行番号が存在しない場合には、次のようなエラーメッセージが表示されます。

```
"Undefined line xxxx in yyyy"
```

これは、行番号 yyyy 中に、参照されていない行番号 xxxx がある、という意味です。この場合、参照されていない行番号 xxxx は RENUM によって変更されませんが、行番号 yyyy は変更されてしまいますのでエラーの発生した行番号の所在が分からなくなってしまいます。したがって、RENUM を行う前にプログラムをセーブしておいたほうが安全です。

注意：65529 を超える行番号は発生することができません。このような場合には "Illegal function call" エラーとなります。

RESTORE

機能	READ で読む DATA 行の先頭行を指定します。
書式	RESTORE [<行番号>]
文例	RESTORE RESTORE 800

〈行番号〉を指定すると、READはその行番号のDATA行からデータを読み始めます。省略した場合はプログラム中の最初のDATA行から読み始めます。

参照：DATA, READ, サンプルプログラム 3

RESUME

機能 エラー処理ルーチンを終了し、元のプログラムの実行を再開します。

書式

- 1) RESUME [0]
- 2) RESUME NEXT
- 3) RESUME 〈行番号〉

文例 RESUME *START

書式 1) エラーの原因となった文からプログラムの実行を再開します。この場合 0 は省略できます。

書式 2) エラーの原因となった文の次の文からプログラムの実行を再開します。

書式 3) 〈行番号〉で指定した行から実行を再開します。

参照：ON ERROR GOTO, サンプルプログラム 22

RETURN

機能 サブルーチンを終了し、元のプログラムの実行を再開します。

書式 RETURN [〈行番号〉]

文例

```
RETURN
RETURN 200
```

サブルーチンの実行を終了し、そのサブルーチンを呼び出した GOSUB の次の命令から実行を再開します。1つのサブルーチン内に複数の RETURNがあってもかまいません。

RETURN では、〈行番号〉を指定して、特定の行にリターンさせることもできます。ただし、サブルーチンの多重化を行っている場合や、FOR~NEXTの内部から GOSUB でサブルーチンを呼び出している場合、リターン先の〈行番号〉には、そのサブルーチンの呼び出し側と同じスタックレベルのプログラム行を指定しなければなりません。これを無視すると、スタック領域が異常消費され、プログラムは正しく動作しません。とくに必要がない限り、使用しないようにしてください。

注意：RETURN を単独で使うことはできません。プログラムの実行中、GOSUB なしで RETURN に出会うと、“RETURN without GOSUB” エラーが起こります。

参照：GOSUB, サンプルプログラム 7

RIGHT\$

関数

機能 文字列の右側から任意の長さの文字列を抜き出します。

書式 RIGHT\$(〈文字列〉, 〈式〉)

文例 PRINT RIGHT\$("ABCDabcd", N*4)

〈文字列〉の右側(最後)から〈式〉で指定した桁数の文字列を抜き出します。〈式〉の値は 0 から 255 の範囲になければなりません。

〈式〉の値が 0 ならば、RIGHT\$ の値はヌルストリング(空の文字列)となります。〈式〉が〈文字列〉の文字数より大きければ、RIGHT\$ の値は〈文字列〉とまったく同じになります。

参照：LEFT\$, MID\$, サンプルプログラム 25

RND

関数

機能 乱数を得ます。

書式 RND [(〈数式〉)]

文例 R=RND

R=RND(-3)

0 以上 1 未満の乱数を得ます。得られる乱数は、RUN および CLEAR が実行されるごとにいつも同系列となります。ただし、RANDOMIZE を使用すれば、この系列を変えることが可能です。

RND 関数の機能は、指定する〈数式〉の値によって次のように異なります。

負の数 : 乱数系列を初期化する。

0 : 1 つ前に発生した乱数の値をとる(繰り返す)。

正の数 : 次の乱数を発生する。

なお、(〈数式〉)を省略して(カッコも含む)、たんに“RND”とした場合は、正の数を指定したのと同じ機能になります。

参照：RANDOMIZE, サンプルプログラム 32

ROLL

機能	グラフィック画面を上下あるいは左右にスクロールさせます。		
書式	ROLL [〈上下方向ドット数〉] [, 〈左右方向ドット数〉] [, <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td style="padding: 2px 5px;">N</td></tr><tr><td style="padding: 2px 5px;">Y</td></tr></table>]]	N	Y
N			
Y			
文例	ROLL -16, 32, Y ROLL , 4		

〈上下方向ドット数〉には、画面の縦方向のドットの数を指定します。許される値の範囲は640×200ドットの画面モードでは-199から199、640×400ドットの画面モードでは-399から399までの値で、正の場合には上方向に、負の場合には下方向に、指定のドット数(絶対値)だけスクロールします。省略した場合は、縦方向のスクロールは行いません。

〈左右方向ドット数〉には、画面の横方向のドットの数を指定します。許される値の範囲は-639から639までの値で、正の場合には左方向に、負の場合には右方向に、指定のドット数(絶対値)以下で最も大きい8の倍数と等しいドット数だけ、スクロールします。省略した場合は、左右のスクロールは行いません。

N または Y を指定すると、スクロールにより新たに現れた領域を決められた色でクリアすることができます。Y を指定するとバックグラウンドカラーでクリアし、N を指定するか省略したときには、パレット番号0でクリアします。

注意: PC-9801 では〈左右方向ドット数〉および N, Y は指定することができません。また、〈上下方向ドット数〉に負の値を指定することはできません。スクロールにより新たに現れた領域は、パレット番号0でクリアされます。

参照: サンプルプログラム 19

RUN

機能	メモリにあるプログラムの実行を開始します。また、ディスクからプログラムをメモリにロードし、そのプログラムを実行します。
書式	1) RUN [〈行番号〉] 2) RUN 〈ファイルディスクリプタ〉 [, R]
文例	RUN *MAIN RUN "2:TEST"

RUN は、プログラムの実行に先だち、変数をすべて初期化し、オープンされているすべてのデータファイルを閉じます。

書式 1) 〈行番号〉を指定すると、メモリ上のプログラムの、その行から実行が始まります。省

SCREEN

機能 グラフィック画面に対して種々のモードを設定します。

書式 SCREEN [**画面モード**] [, **画面スイッチ**] [, **アクティブページ**] [, **ディスプレイページ**]

文例 SCREEN 1, 0, 0, 7

〈画面モード〉は、カラー、白黒、分解能など、グラフィック画面の最も基本的なモードを設定するパラメータです。

指定値		分解能(横×縦)	使用可能ページ数
0	カラーモード	640×200	4
1	白黒モード	640×200	12(16*)
2	高分解能白黒モード	640×400	6(8*)
3	高分解能カラーモード	640×400	2

ここで、*のついた数値は、16色モードのときの値を表します。

〈画面スイッチ〉は、指定する値により、現在グラフィック画面に描かれているパターンなどを一時的に消去するものです。

指定値

0(または1) グラフィック画面の表示を行う。

2(または3) 現在グラフィック画面に表示されている図形などを一時的に消去する。

〈アクティブページ〉と〈ディスプレイページ〉はグラフィック命令で書き込むページと表示するページの選択を行うためのものです。

〈アクティブページ〉と〈ディスプレイページ〉に指定できる値とその意味はパレットモードによって異なります(パレットモードに関しては [1] COLOR を参照してください)。

(1) 8色中・8色モードあるいは4096色中・8色モードの場合

〈アクティブページ〉にはグラフィック命令によって書き込むページを指定します。画面モードによって扱える画面数は異なります。

〈アクティブページ〉に指定できる値とページ番号との対応関係

指定値	画面モード	書き込まれるページ番号
0~3	カラーモード(0)	1~4
0~11	白黒モード(1)	1~12
0~5	高分解能白黒モード(2)	1~6
0, 1	高分解能カラーモード(3)	1, 2

〈ディスプレイページ〉には、画面モードに対応したページのうちのどのページを表示するかを指定します。

〈ディスプレイページ〉に指定できる値と表示されるページ番号との対応関係

指定値	カラーモード (0)	白黒モード (1)	高分解能白黒モード (2)	高分解能カラーモード (3)
0	全ページ表示しない	全ページ表示しない	全ページ表示しない	全ページ表示しない
1	ページ1のみ表示	ページ1のみ表示	ページ1のみ表示	ページ1のみ表示
2	ページ2のみ表示	ページ2のみ表示	ページ2のみ表示	×
3	×	ページ1, 2を合成表示	ページ1, 2を合成表示	×
4	×	ページ3のみ表示	ページ3のみ表示	×
5	×	ページ1, 3を合成表示	ページ1, 3を合成表示	×
6	×	ページ2, 3を合成表示	ページ2, 3を合成表示	×
7	×	ページ1, 2, 3を合成表示	ページ1, 2, 3を合成表示	×
8	全ページ表示しない	全ページ表示しない	全ページ表示しない	全ページ表示しない
9	×	ページ4のみ表示	×	×
10	×	ページ4のみ表示	×	×
11	×	ページ4, 5を合成表示	×	×
12	×	ページ6のみ表示	×	×
13	×	ページ4, 6を合成表示	×	×
14	×	ページ5, 6を合成表示	×	×
15	×	ページ4, 5, 6を合成表示	×	×
16	全ページ表示しない	全ページ表示しない	全ページ表示しない	全ページ表示しない
17	ページ3のみ表示	ページ7のみ表示	ページ4のみ表示	ページ2のみ表示
18	ページ4のみ表示	ページ8のみ表示	ページ5のみ表示	×
19	×	ページ7, 8を合成表示	ページ4, 5を合成表示	×
20	×	ページ9のみ表示	ページ6のみ表示	×
21	×	ページ7, 9を合成表示	ページ4, 6を合成表示	×
22	×	ページ8, 9を合成表示	ページ5, 6を合成表示	×
23	×	ページ7, 8, 9を合成表示	ページ4, 5, 6を合成表示	×
24	全ページ表示しない	全ページ表示しない	全ページ表示しない	全ページ表示しない
25	×	ページ10のみ表示	×	×
26	×	ページ11のみ表示	×	×
27	×	ページ10, 11を合成表示	×	×
28	×	ページ12のみ表示	×	×
29	×	ページ10, 11を合成表示	×	×
30	×	ページ11, 12を合成表示	×	×
31	×	ページ10, 11, 12を合成表示	×	×

×印：指定不可

(2) 4096色中・16色モードの場合

〈アクティブページ〉にはグラフィック命令によって書き込むページを指定します。画面モードによって扱える画面数は異なります。

〈アクティブページ〉に指定できる値とページ番号との対応関係

指定値	画面モード	書き込まれるページ番号
0~3	カラーモード(0)	1~4
0~15	白黒モード(1)	1~16
0~7	高分解能白黒モード(2)	1~8
0, 1	高分解能カラーモード(3)	1, 2

〈ディスプレイページ〉には、画面モードに対応したページのうちのどのページを表示するかを指定します。

〈ディスプレイページ〉に指定できる値と表示されるページ番号との対応関係

指定値	カラーモード (0)	白黒モード (1)	高分解能白黒モード (2)	高分解能カラーモード (3)
0	全ページ表示しない	全ページ表示しない	全ページ表示しない	全ページ表示しない
1	ページ1のみ表示	ページ1のみ表示	ページ1のみ表示	ページ1のみ表示
2	ページ2のみ表示	ページ2のみ表示	ページ2のみ表示	×
3	×	ページ1, 2を合成表示	ページ1, 2を合成表示	×
4	×	ページ3のみ表示	ページ3のみ表示	×
5	×	ページ1, 3を合成表示	ページ1, 3を合成表示	×
6	×	ページ2, 3を合成表示	ページ2, 3を合成表示	×
7	×	ページ1, 2, 3を合成表示	ページ1, 2, 3を合成表示	×
8	×	ページ4のみ表示	ページ4のみ表示	×
9	×	ページ1, 4を合成表示	ページ1, 4を合成表示	×
10	×	ページ2, 4を合成表示	ページ2, 4を合成表示	×
11	×	ページ1, 2, 4を合成表示	ページ1, 2, 4を合成表示	×
12	×	ページ3, 4を合成表示	ページ3, 4を合成表示	×
13	×	ページ1, 3, 4を合成表示	ページ1, 3, 4を合成表示	×
14	×	ページ2, 3, 4を合成表示	ページ2, 3, 4を合成表示	×
15	×	ページ1, 2, 3, 4を合成表示	ページ1, 2, 3, 4を合成表示	×
16	全ページ表示しない	全ページ表示しない	全ページ表示しない	全ページ表示しない
17	×	ページ5のみ表示	×	×
18	×	ページ6のみ表示	×	×
19	×	ページ5, 6を合成表示	×	×
20	×	ページ7のみ表示	×	×
21	×	ページ5, 7を合成表示	×	×
22	×	ページ6, 7を合成表示	×	×
23	×	ページ5, 6, 7を合成表示	×	×
24	×	ページ8のみ表示	×	×
25	×	ページ5, 8を合成表示	×	×
26	×	ページ6, 8を合成表示	×	×
27	×	ページ5, 6, 8を合成表示	×	×
28	×	ページ7, 8を合成表示	×	×
29	×	ページ5, 7, 8を合成表示	×	×
30	×	ページ6, 7, 8を合成表示	×	×
31	×	ページ5, 6, 7, 8を合成表示	×	×
32	全ページ表示しない	全ページ表示しない	全ページ表示しない	全ページ表示しない
33	ページ3のみ表示	ページ9のみ表示	ページ5のみ表示	ページ2のみ表示
34	ページ4のみ表示	ページ10のみ表示	ページ6のみ表示	×
35	×	ページ9, 10を合成表示	ページ5, 6を合成表示	×
36	×	ページ11のみ表示	ページ7のみ表示	×
37	×	ページ9, 11を合成表示	ページ5, 7を合成表示	×
38	×	ページ10, 11を合成表示	ページ6, 7を合成表示	×
39	×	ページ9, 10, 11を合成表示	ページ5, 6, 7を合成表示	×
40	×	ページ12のみ表示	ページ8のみ表示	×

指定値	カラーモード (0)	白黒モード (1)	高分解能白黒モード (2)	高分解能カラーモード (3)
41	×	ページ 9, 12 を合成表示	ページ 5, 8 を合成表示	×
42	×	ページ 10, 12 を合成表示	ページ 6, 8 を合成表示	×
43	×	ページ 9, 10, 12 を合成表示	ページ 5, 6, 8 を合成表示	×
44	×	ページ 11, 12 を合成表示	ページ 7, 8 を合成表示	×
45	×	ページ 9, 11, 12 を合成表示	ページ 5, 7, 8 を合成表示	×
46	×	ページ 10, 11, 12 を合成表示	ページ 6, 7, 8 を合成表示	×
47	×	ページ 9, 10, 11, 12 を合成表示	ページ 5, 6, 7, 8 を合成表示	×
48	全ページ表示しない	全ページ表示しない	全ページ表示しない	全ページ表示しない
49	×	ページ 13 のみ表示	×	×
50	×	ページ 14 のみ表示	×	×
51	×	ページ 13, 14 を合成表示	×	×
52	×	ページ 15 のみ表示	×	×
53	×	ページ 13, 15 を合成表示	×	×
54	×	ページ 14, 15 を合成表示	×	×
55	×	ページ 13, 14, 15 を合成表示	×	×
56	×	ページ 16 のみ表示	×	×
57	×	ページ 13, 16 を合成表示	×	×
58	×	ページ 14, 16 を合成表示	×	×
59	×	ページ 13, 14, 16 を合成表示	×	×
60	×	ページ 15, 16 を合成表示	×	×
61	×	ページ 13, 15, 16 を合成表示	×	×
62	×	ページ 14, 15, 16 を合成表示	×	×
63	×	ページ 13, 14, 15, 16 を合成表示	×	×

この〈アクティブページ〉と〈ディスプレイページ〉については、8色モードと16色モードでは、指定できる値とその意味が異なることに注意してください。

SCREEN を実行すると、そのとき設定されていたウィンドウ、ビューポート、LP(最終参照点)は初期状態にもどります。なお、LPの初期状態はワールド座標、スクリーン座標とも(0, 0)です。

参照： [1] COLOR, サンプルプログラム 20, 21

SEARCH

関 数

- 機能** 配列変数の中から指定された値を捜し出し、その要素の順位を得ます。
- 書式** SEARCH(〈配列変数名〉, 〈整数表記〉 [, 〈開始添字〉] [, 〈ステップ値〉])
- 文例** NUM=SEARCH(A%, 100, 0, 3)

〈配列変数名〉で指定された配列変数の要素の中から〈整数表記〉で指定された値を捜し、最初に見つかった要素の添字を得ます。指定された値がその配列変数内で見つからなかった場合、SEARCHの値は-1となります。

〈配列変数名〉は整数型の一次元配列でなければなりません。またこの配列変数はSEARCHの実行に先だってDIMによって宣言されていなければなりません。

〈整数表記〉には捜したい値を指定します。この値は整数でなければならず、実数を指定した場合は整数化してから実行します。

〈開始添字〉には、配列中のどの要素から捜し始めるかを指定します。省略した場合には、配列の最初から捜し始めます。

〈ステップ値〉を指定した場合は、その間隔で捜します。省略した場合は1が採用されます。

SEARCHはランダムアクセスファイルのインデックスを捜し出したり、GET@、PUT@で用いる配列データを捜し出ししたりするのに使うことができます。

参照：DIM, OPTION BASE

SET (DISK モード)

機能 ファイル属性のセット・リセットを行います。

書式

- 1) SET <ドライブ番号>, | "P" |
 |R" |
- 2) SET <ファイルディスクリプタ>, | "P" |
 |R" |
- 3) SET # <ファイル番号>, | "P" |
 |R" |

文例

```
SET 1, "P"
SET "2 : NOTWRITE", "P"
SET #1, "R"
```

指定したドライブ、ファイル、ファイルバッファに、"P"または"R"(属性文字)によって指定された属性をつけます。

"P"はライトプロテクト(書き込み禁止)の属性をつけます。この属性が指定されると、PRINT#, PUT, WRITE#などの書き込み動作が禁止されるとともに、ファイルのKILLもできなくなります。

"R"はリードアフターライト(書き込み確認)の属性をつけます。この属性が指定されると、書き込みを行った直後に読み出しを行い、正しく書き込みが行われたかの確認が行われるようになります。

これら以外の文字が属性文字として指定された場合には、そのとき、設定されている属性が解除されます。

書式 1) <ドライブ番号> が指定された場合には、指定されたドライブ(ディスク媒体)に対して属性がつけられます。

書式 2) <ファイルディスクリプタ>が指定された場合には、指定されたファイルのみに属性がつけられます。同一ディスク中の他のファイルには影響をおよぼしません。

書式 3) <ファイル番号>が指定された場合には、そのファイルが開かれている間だけ、指定された属性が作用します。

参照：ATTR\$

SGN

関 数

機 能	符号を調べます。
書 式	SGN(<数式>)
文 例	PRINT SGN(RESULT)

<数式> に指定された式の値の正負により、SGNの値は次のようになります。

正の場合	: 1
0	: 0
負の場合	: -1

参照：サンプルプログラム 9

SIN

関 数

機 能	正弦(サイン)を得ます。
書 式	SIN(<数式>)
文 例	X=RADIUS*SIN(ANGLE) PRINT SIN(-3.14159/2)

<数式> の値に対する正弦値を得ます。<数式> の単位はラジアンで指定します。

<数式>に倍精度実数が含まれる場合、得られる値は倍精度となりますが、他の場合には単精度となります。

参照：ATN, COS, TAN, サンプルプログラム 6, 30

SPACE\$

関 数

機 能 任意の長さの空白文字列を得ます。

書 式 SPACE\$(〈数式〉)

文 例 QTTY\$=NUM\$+SPACE\$(10)+RES\$

〈数式〉に指定した数だけの空白(スペース、キャラクタコード&H20)が連なった文字列を得ます。〈数式〉の値は0から255までの範囲内でなければなりません。

参照：SPC, サンプルプログラム 25

SPC

関 数

機 能 任意の数だけの空白を出力します。

書 式 SPC(〈数式〉)

文 例 PRINT "SUPER" ; SPC(10) ; "BASIC"

出力の対象となる行の中で、〈数式〉に指定した数だけの空白(スペース)を出力します。

ただし、SPCは、PRINTやLPRINTなどの出力文中でのみ使用することができるもので、文字式としては使えません。

〈数式〉には、-32768から32767までの範囲を指定できますが、負の数はすべて0とみなされます。また、正の数の場合でも、WIDTHで決められている現在の水平表示桁数以上の場合は、〈数式〉をその表示桁数で割った余りを値とします。

参照：SPACE\$, TAB

SQR

関 数

機 能 平方根を得ます。

書 式 SQR(〈数式〉)

文 例 DISTANCE=SQR(X*X+Y*Y)

〈数式〉に指定した値の平方根(スクエアルート)を得ます。

〈数式〉に倍精度実数が含まれる場合、得られる値は倍精度となりますが、他の場合には単精度となります。

参照：サンプルプログラム 8

STATUS DIAL (DISK モード)関数 **P****機能** 電話機に記憶されている電話番号の機能情報を調べます。**書式** STATUS DIAL([[#] <電話機番号>,) <短縮番号>**文例** STATUS DIAL(# 1, 21)

<電話機番号>で指定された電話機が記憶している<短縮番号>に対応する、自動発信後のモードを得ます。

<電話機番号>には、CMD LINE OPEN で接続された電話機の番号を指定します。<電話機番号>が省略された場合は、1が指定されたものとみなされます。

<短縮番号>は1~40までの数値でなければなりません。得られる値と自動発信後のモードの関係は次のとおりです。

- 0 : ダイヤル後通話モードとなる
- 1 : アンサートーン検出後データ通信モードとなる
- 2 : 接続後データ通信モードとなる
- 3 : ダイヤル後データ通信モードとなる

各値の意味の詳細については、CMD DIAL を参照してください。<短縮番号>に対応する電話番号が記憶されていない場合には、-1が得られます。

注意 : この関数はオフラインコマンドモードでのみ使用可能です。PC-9863N, PC-9865 モデムボードでは、この関数を使用することはできません。

STATUS DIAL\$ (DISK モード)関数 **P****機能** 電話機に記憶されている電話番号を調べます。**書式** STATUS DIAL\$([[#] <電話機番号>,) <短縮番号>**文例** STATUS DIAL\$(# 1, 21)

<電話機番号>で指定されている電話機に記憶されている<短縮番号>に対応する、電話番号を得ます。

<電話機番号>には、CMD LINE OPEN で接続された電話機の番号を指定します。<電話機番号>が省略された場合は、1が指定されたものとみなされます。

<短縮番号>は1~40の数値でなければなりません。<短縮番号>に対応する電話番号が記憶されていない場合にはヌルストリング (空の文字列) が得られます。

注意：この関数はオフラインコマンドモードでのみ使用可能です。PC-9863N, PC-9865 モデムボードでは、この関数を使用することはできません。

STATUS ERROR (DISK モード)

関数 **P**

機能	モデムからのデータ受信時における通信エラーの有無を調べます。
書式	STATUS ERROR(((#) <電話機番号>))
文例	STATUS ERROR(#1)

<電話機番号>で指定した電話機がモデムからデータを受信したとき、エラーが発生したか否かを調べます。

<電話機番号>には、CMD LINE OPEN で接続された電話機の番号を指定します。<電話機番号>が省略された場合は、1 が指定されたものとみなされます。

エラーの状態は 0~7 の数字で得られます。得られる数値とエラーの状態の対応は次のとおりです。

- 0 : エラーなし
- 1 : パリティエラー
- 2 : オーバーランエラー
- 3 : オーバーランエラー+パリティエラー
- 4 : フレーミングエラー
- 5 : フレーミングエラー+パリティエラー
- 6 : フレーミングエラー+オーバーランエラー
- 7 : フレーミングエラー+オーバーランエラー+パリティエラー

注意：この関数はデータ通信モードでのみ使用可能です。

参照：STATUS LINE, STATUS MODE

STATUS LINE (DISK モード)

関数 **P**

機能	着信があったかどうかを調べます。
書式	STATUS LINE(((#) <電話機番号>))
文例	STATUS LINE(#1)

〈電話機番号〉で指定した電話に着信があったかどうかを調べます。

〈電話機番号〉には、CMD LINE OPEN で接続された電話機の番号を指定します。〈電話機番号〉が省略された場合は、1 が指定されたものとみなされます。

着信があった場合には-1 (真) が、着信がない場合には0 (偽) が得られます。

注意：この関数を実行すると、着信があったという事象をクリアしてしまいますので、着信割り込みを使用するプログラムでこの関数を使用すると正しく動作しません。CMD ON LINE GOSUB などと混在して使用しないようにしてください。

参照：CMD ON LINE GOSUB, STATUS ERROR, STATUS MODE

STATUS MODE (DISK モード)

関数 **P**

機能 電話機の現在のモードを調べます。

書式 STATUS MODE[[[#] 〈電話機番号〉]]

文例 STATUS MODE(#1)

〈電話機番号〉で指定した電話機の現在のモードを調べます。

〈電話機番号〉には、CMD LINE OPEN で接続された電話機の番号を指定します。〈電話機番号〉が省略された場合は、1 が指定されたものとみなされます。

得られる数値と電話機のモードの対応は次のとおりです。

- 0 : 初期モード (電話機が論理的に接続されていない状態)
- 1 : オフラインコマンドモード
- 2 : データ通信モード
- 3 : 通話モード

電話機のモードの詳細に関しては「BASIC ユーザーズマニュアル」を参照してください。

参照：CMD LINE OPEN, サンプルプログラム 40

STATUS PLAY

関数 **S**

機能 サウンドバッファの未演奏データのバイト数を得ます。

書式 STATUS PLAY(〈チャンネル番号〉)

文例 PRINT STATUS PLAY(0)

〈チャンネル番号〉で指定されたチャンネルのサウンドバッファの未演奏データのバイト数を
得ます。〈チャンネル番号〉と得られる値の関係は次のとおりです。

- 1~6 : 対応するチャンネルの未演奏データバイト数
- 0 : すべてのチャンネルの未演奏データバイト数の中で最大のバイト数
- 負の数 : すべてのチャンネルの未演奏データバイト数の中で最小のバイト数

参照 : PLAY

STOP

機能 プログラムの実行を一時中断し、コマンドモードにもどります。

書式 STOP

文例 STOP

STOP はプログラムの実行を一時中断するためのもので、プログラムのどこにおいてもかま
いません。STOP を実行すると、次のメッセージが表示されます。

Break in 〈行番号〉

ここで、〈行番号〉は実行の中断されたプログラム行の行番号を表します。

STOP が実行されると、BASIC は常にコマンドレベルにもどります。また、CONT コマ
ンドによりプログラムの実行を再開することができます。ただし、INPUT, LINE INPUT,
INPUT\$関数などの入力文の途中でストップした場合、継続することはできません。

STOP は END とは異なり、そのときオープンされていたデータファイルをクローズしませ
んので注意してください。

参照 : CONT, END

STOP ON/OFF/STOP

機能 STOP キーおよび CTRL + C による割り込みの許可、禁止、停止を制御します。

- 書式**
- 1) STOP ON
 - 2) STOP OFF
 - 3) STOP STOP

文例 STOP ON

書式 1) 割り込みを許可します。以後 **STOP** キーあるいは **CTRL** + **C** を押すごとに割り込みが発生し、ON STOP GOSUB によって設定された処理ルーチンに分岐します。

書式 2) 割り込みを禁止します。以後 **STOP** キーあるいは **CTRL** + **C** を押しても処理ルーチンへの分岐は起こりません (**STOP** キーあるいは **CTRL** + **C** は通常の動作になります)。

書式 3) 割り込みを停止します。以後 **STOP** キーあるいは **CTRL** + **C** を押しても押されたことを覚えているだけで、処理ルーチンへの分岐は起こりません。しかし、その後、STOP ON によって割り込みが許可されると、前に **STOP** キーあるいは **CTRL** + **C** を押したことによって、処理ルーチンに分岐します。

参照：ON STOP GOSUB

STR\$

関数

機能 数値を文字列に変換します。

書式 STR\$(**<数式>**)

文例 LISTING\$=STR\$(NUM)+NAME\$

<数式> に指定した値を、文字列に変換します。

変換後の文字列の最初の文字は、**<数式>** の値が正ならば空白(スペース)となり、負ならば“-”(マイナス記号)となります。

<数式> には、整数型および実数型(倍精度、単精度)の数値を指定することができます。

注意：STRING\$ と区別すること。

参照：STRING\$, VAL, サンプルプログラム 24

STRING\$

関数

機能 任意の文字を任意の数だけ連結した文字列を得ます。

書式 STRING\$(**<式>**, **| <文字式> |**)
| <数式> |

文例 A\$=STRING\$(50,"+")

〈文字式〉に指定した文字、または〈数式〉に指定したキャラクタコードに該当する文字を、〈式〉に指定した数だけ連結した文字列を得ます。

〈文字式〉の場合には最初の1文字のみが用いられます。〈数式〉の場合は、値が0から255の範囲内でなくてはなりません。

参照：STR\$

SWAP

機能 2つの変数の値を入れ換えます。

書式 SWAP 〈変数〉, 〈変数〉

文例 SWAP A\$, B\$

どの型(整数型, 単精度型, 倍精度型, または文字型)の変数でも, SWAP によってその値を交換することができます。

ただし, 2つの変数の型が一致していないと, "Type mismatch" エラーが起こります。

参照：サンプルプログラム 10

TAB

関数

機能 出力対象行の任意の位置まで空白を空けます。

書式 TAB(〈数式〉)

文例 PRINT "SUPER" ; TAB(10) ; "BASIC"

出力の対象となる行の行頭から, 〈数式〉に指定した桁数分だけ, 空白(スペース)を空けます。ただし, TAB は, PRINT や LPRINT などの出力文中でのみ使用することができるもので, 文字式としては使えません。

〈数式〉には, -32768 から 32767 までの範囲を指定できますが, 負の数はすべて0とみなされます。また, 正の数の場合でも, WIDTH で決められている現在の水平表示桁数以上の場合は, 〈数式〉をその表示桁数で割った余りを値とします。

TAB では, SPC と違い〈数式〉で指定する数は常に行頭からの桁数, つまり位置を表しますから, 作表などに使用すると便利です。

参照：SPC, サンプルプログラム 22, 30

TAN

関 数

機 能 正接(タンジェント)を得ます。

書 式 TAN(<数式>)

文 例 Y=X*TAN(ANGLE)

<数式>の値に対する正接値を得ます。<数式>の単位はラジアンで指定します。

<数式>に倍精度実数が含まれる場合、得られる値は倍精度となりますが、他の場合には単精度となります。

参照 : ATN, COS, SIN

TERM

機 能 システムをターミナルモードにします。

書 式 TERM "[COM:] [<パリティチェック> [<データビット長> [<ストップビット長> [<Xパラメータ> [<Sパラメータ> [<DELコード受信処理> [<リターンキー送信処理> [<CRコード受信処理> [<日本語シフトコード>]]]]]]]]]" [, [<通信方式>] [, <変数領域の大きさ>]]

文 例 TERM "COM : E71XS", F, 1000

BASICモードからターミナルモードに制御を移行します。ターミナルモードでは、RS-232C回線を介して他のコンピュータや機器との通信を行うことが可能となります。また、ターミナルモード時においても、リモート BASIC プロトコルにより、N₈₈-BASICの機能を利用することができます。詳しくは BASIC ユーザーズマニュアルを参照してください。

COM : で、RS-232C 回線を使って外部機器と入出力を行うことを宣言します。なお、COM : は省略することができますが、省略した場合は、後に続くすべてのパラメータを指定しなければなりません。

各パラメータは、次のように設定します。

<パリティチェック> は、通信時に使用されるパリティを決定します。

- E : 偶数パリティ
- O : 奇数パリティ
- N : パリティ無し

〈データビット長〉は、1文字を表すために必要なビット数です。

7 : 7ビット

8 : 8ビット

〈ストップビット長〉は、ストップビット数を決定します。

1 : 1ビット

2 : 1.5ビット

3 : 2ビット

〈Xパラメータ〉は、XON/OFFによる通信制御を行うことを指定します。

X : XON/OFF制御を行う。

N : XON/OFF制御を行わない。

〈Sパラメータ〉は〈データビット長〉に7を指定したときに、キャラクタコード128以上の文字(カナ文字など)の入出力を可能にするためのパラメータです。

S : 入出力可能

N : 入出力不可能

〈DELコード受信処理〉は、受信したDELコードの処理方法を決定するものです。

B : BSコードに変換されてBSコードとして処理されます。

N : NULコードに変換されてNULコードとして処理されます。

〈リターンキー送信処理〉は、キーを押したときに送信するコードを決定します。

C : CRコードを送信します。

L : CRコード、LFコードをこの順に送信します。

〈CRコード受信処理〉は、CRコードを送信したときの動作を決定します。

C : CRコード受信時に復帰動作と改行動作を行う。

L : CRコード受信時に復帰動作だけを行う。

〈日本語シフトコード〉は、英数かな文字列のなかで日本語文字列の始まりを表示するコード(KIコード)と、日本語文字の終わりを表示するコード(KOコード)とを決定します。

P : KIコード=&H1B4B(ESC・Kに相当)、KOコード=&H1B48(ESC・Hに相当)が使用される。

I : KIコード=&H1A70(SB・pに相当)、KOコード=&H1A71(SB・qに相当)が使用される。

〈通信方式〉は、通信を半二重で行うか、全二重で行うかを決定します。

H : 半二重通信モード

F : 全二重通信モード

〈変数領域の大きさ〉は、リモート BASIC プロトコルを利用する際に用いられる変数領域の大きさを決定します。省略された場合 1024 に設定されます。

ターミナルモードから BASIC にもどるときには、**SHIFT** キーを押しながら **STOP** キーを押します。

注意：〈変数領域の大きさ〉を除くすべてのパラメータは、省略された場合、ユーティリティ "switch.n88" であらかじめ設定されていた RS-232C の初期設定の値が採用されます。また、途中のパラメータを省略して後のパラメータを指定する場合は、省略記号としてブランク(空白)を指定しなければなりません。

また、パラメータには、必ず大文字を使ってください。

ターミナルモードを RS-232C 第 2 回線および第 3 回線で使うことはできません。

TIME\$

関 数

機 能 時刻を得ます。

書 式 1) TIME\$
2) TIME\$="hh : mm : ss"

文 例 PRINT TIME\$
TIME\$="23 : 15 : 00"

TIME\$には常に現在の時刻が"hh : mm : ss"(時 : 分 : 秒)の形で入れられており、いつでもその内容を見ることができます。時刻表示は、24 時制です。

なお、書式 2) を用いることにより、時刻を変更することもできます。hh には 00~23, mm には 00~59, ss には 00~59 の範囲の整数文字列を指定します。

注意：時刻は、バッテリーバックアップによって自動的に更新され、正しく維持されるようになっています。不用意に時刻を変えないようにしてください。

参照：DATE\$, サンプルプログラム 31

TIMES ON/OFF/STOP

機能	リアルタイムタイマによる割り込みの許可、禁止、停止を制御します。
書式	1) TIMES ON 2) TIMES OFF 3) TIMES STOP
文例	TIMES ON

書式 1) 割り込みを許可します。以後、設定された時刻になると割り込みが発生し、ON
TIMES GOSUB によって設定された処理ルーチンに分岐します。

書式 2) 割り込みを禁止します。以後、設定された時刻になっても処理ルーチンへの分岐は起
りません。

書式 3) 割り込みを停止します。以後、設定された時刻になってもそのことを覚えているだけ
で、処理ルーチンへの分岐は起こりません。しかし TIMES ON によって割り込みが
許可されると、先ほどの割り込みで処理ルーチンに分岐します。

注意：プログラムの終了時には TIMES OFF を実行しておいてください。

参照：ON TIMES GOSUB, TIMES, サンプルプログラム 31

TRON/TROFF

機能	プログラムの実行状態を追跡します。
書式	1) TRON 2) TROFF
文例	TRON TROFF

TRON を実行してからプログラムを RUN させると、以後実行しているプログラムの行番号
がカギカッコ () つきでテキスト画面に表示され続けます。

TRON 状態を中止するには、TROFF を実行します。なお、NEW を実行した場合も TRON
状態は解除されます。

USR

関数

機能 メモリ上に用意された機械語関数を呼び出します。

書式 USR[<番号>](<引数>)

文例 I=USR3(J)

呼び出したい機械語関数はあらかじめメモリ上に用意しておき、また DEF USR により、その実行開始アドレスを設定しておかなくてはなりません。機械語関数を用意するには、POKE や BLOAD を用いることができます。

<番号> には、DEF USR により定義された番号を 0 から 9 までの値で指定します。省略された場合には 0 と解釈されます。

<引数> には、BASIC から機械語関数に渡す変数、定数、式を指定します。

USR が実行されると、直前に実行された DEF SEG で指定されたセグメントベースに、DEF USR により定義された機械語関数の実行開始アドレス(相対アドレス値)が加えられた番地に実行が移されます。USR によって呼び出された機械語関数は、機械語の IRET 命令により BASIC に制御をもどすことができます。

なお、機械語プログラムを呼び出す方法としては、USR の他に CALL が用意されています。CALL では複数の引数を機械語プログラムに受け渡すことができます。

参照：BLOAD, CALL, CLEAR, DEF USR, POKE

VAL

関数

機能 文字列表記の数値を実際の数値に変換します。

書式 VAL(<文字列>)

文例 A=VAL("&H20")

<文字列> に指定した文字列表記の数値を、実際の数値に変換します。

<文字列> には、整数表記(8進形式, 10進形式, 16進形式)および実数表記(通常の小数点形式, 指数形式)のいずれも指定できます。

<文字列>の最初の文字が+, -, &, または数字でなければ、VALの値は0になります。また、<文字列>中に数字を表す文字以外の文字が含まれていると、その文字以降の文字は無視されます。ただし、16進表記ならばA~Fも数字とみなされ、8進表記ならば8および9は数字とみなされないことに注意してください。

なお、文字列中のスペースは無視されます。

参照：STR\$

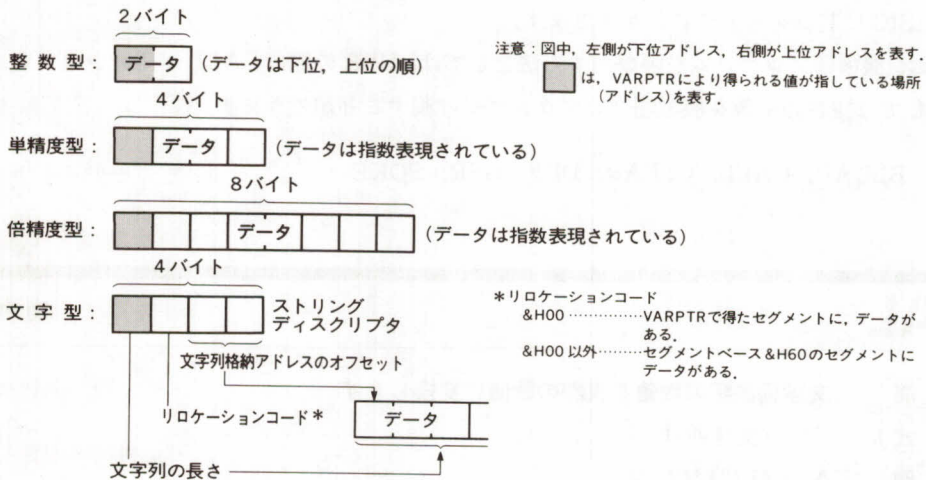
VARPTR

関 数

- 機 能** 変数の値が格納されているメモリ番地、ファイルに割り当てられているファイルコントロールブロックの開始番地を得ます。
- 書 式** 1) VARPTR(<変数名> [, <機能>])
2) VARPTR(# <ファイル番号> [, <機能>])
- 文 例** PRINT HEX\$(VARPTR(A, 1))
PRINT HEX\$(VARPTR(A))
SEGM%=VARPTR(# 1, 1)
ADDR%=VARPTR(# 1)

書式 1) <変数名>で指定した変数あるいは配列変数のデータが格納されている領域のメモリ番地を得ます。

<機能>に0を指定すると相対アドレスが得られ、0以外の整数を指定するとセグメントベースが得られます。省略した場合は、0が指定されたものとみなされます。



書式 2) 指定した<ファイル番号>に割り当てられているファイルコントロールブロックの開始番地を得ます。

<機能>に0を指定すると相対アドレスが得られ、1を指定するとセグメントベースが得られます。省略した場合は、0が指定されたものとみなします。

なお、ファイルに割り当てられている入出力バッファの開始番地は、このファイルコントロールブロックの33バイト目からの2バイトに格納されています。

参照: DEF SEG

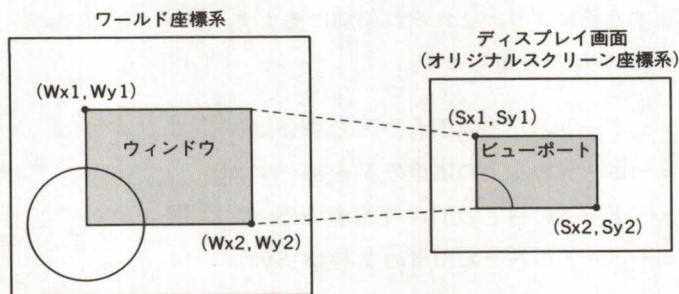
VIEW

機能	ディスプレイ画面上での表示領域(ビューポート)を指定します。
書式	VIEW (Sx1, Sy1)–(Sx2, Sy2) [, <領域色>] [, <境界色>]
文例	VIEW (100, 30)–(200, 75),, 7

オリジナルスクリーン座標系上の(Sx1, Sy1)を左上の頂点、(Sx2, Sy2)を右下の頂点とする長方形を図形表示領域として指定します。ここでいうオリジナルスクリーン座標とは、ディスプレイ画面のドットと1対1に対応して、WINDOW, VIEWにより変化することのない物理的な座標のことです。

VIEWが実行されると、WINDOWにより指定されているワールド座標系上の領域内の図形は、VIEWで指定した領域中に表示されるようになります。

この領域をビューポート(View Port)といいます。



<領域色>にパレット番号を指定すると、パレット番号の色でビューポート内をぬりつぶします。

<境界色>にパレット番号を指定すると、パレット番号の色でビューポートの枠を描きます。CLSによって消去される画面の範囲は、この枠によって囲まれた中だけで、枠は消されることはありません。

VIEWは、図形の表示領域の指定を行うだけで、実際の画面に対する操作は行いませんので、VIEWによりビューポートを変更することによって、以前のビューポート中の図形が移動するようなことはありません。また、VIEWはすべてのグラフィック画面表示の対象範囲を指定された領域に限定してしまいますから、ビューポートの外に点や線などを表示することはできなくなります。

VIEWの指定を変えることにより、1つの図形を描くプログラムでも、画面上の異なる位置に異なる大きさで図形を描くことができます。

Sx1 < Sx2, Sy1 < Sy2が成り立たない場合、あるいはこれらの座標がディスプレイ画面から外れている場合には、“Illegal function call”エラーとなります。

一度設定されたビューポートは、次にVIEWあるいはSCREENが実行されるまで変化しません。また、VIEWはLP(最終参照点)をビューポートの左上の頂点に移動します。

注意：ワールド座標系がビューポート内に展開されるのは WINDOW の実行後であり、VIEW のみ実行してウィンドウは初期状態のままであれば、ビューポート内の座標は、(ワールド座標)=(スクリーン座標)となります。

参照：CLS, SCREEN, WINDOW, サンプルプログラム 20, 21

VIEW

関数

機能 現在のビューポートの設定位置を得ます。

書式 VIEW(<機能>)

文例 SX1=VIEW(0)

VIEW により設定されている現在のビューポートの設定位置(Sx1, Sy1 および Sx2, Sy2)を、<機能> に指定する値により、それぞれ個別に得ます。

<機能> には 0~3 の数値を指定します。

- 0 : ビューポートの左上の頂点の X 座標(Sx1)
- 1 : ビューポートの左上の頂点の Y 座標(Sy1)
- 2 : ビューポートの右下の頂点の X 座標(Sx2)
- 3 : ビューポートの右下の頂点の Y 座標(Sy2)

注意：VIEW 関数は、ディスプレイ画面上のビューポートの位置を得る関数ですから、得られる値は、スクリーン座標系の値となります。

参照：MAP, VIEW, WINDOW, WINDOW(関数)

VOICE

S

機能 FM 音源の音色バンクを再定義します。

書式 VOICE <音色番号>, <整数型配列名>

文例 VOICE 5, A%

FM 音源の音色を自由に設定するための命令です。

サウンド制御命令では、FM 音源で最大 82 種類の音色を使用することができます。これらの音色は 0~81 の番号で管理されており、初期状態では内蔵音色のデータが各音色番号に割り当てられています。82 種類のうち、0 番のデフォルト音色以外はすべて、再定義することによって自分の好みの音色を割り当てることができます。

音色の再定義は、音色番号の音色パラメータ（音色のデータ）を別の音色パラメータに変えることによって行います。音色番号と音色パラメータを合わせて音色バンクといいます。

〈音色番号〉には、音色を再定義したい音色番号（1～81）を指定します。

〈整数型配列名〉には、新しく与える音色パラメータを格納した整数型配列変数の配列変数数を指定します。配列変数にはあらかじめ音色パラメータを指定しておかなければなりません。

配列変数の各要素と音色パラメータとは、次のように対応しています（配列名は、仮にVOICE1%とします）。

ここで、OP は各オペレータの番号を表し、1～4 を指定します。

VOICE1% (0, 0)	:	フィードバック/アルゴリズム
VOICE1% (0, 1)	:	オペレータマスク
VOICE1% (0, 2)	:	LFO 波形
VOICE1% (0, 3)	:	LFO SYNC ディレイ
VOICE1% (0, 4)	:	LFO 速さ
VOICE1% (0, 5)	:	LFO ピッチ変調深さ（微調整）
VOICE1% (0, 6)	:	LFO 振幅変調深さ（微調整）
VOICE1% (0, 7)	:	LFO ピッチ変調深さ（粗調整）
VOICE1% (0, 8)	:	未使用
VOICE1% (0, 9)	:	未使用
VOICE1% (OP, 0)	:	アタック系数
VOICE1% (OP, 1)	:	ディケイ系数
VOICE1% (OP, 2)	:	サステイン系数
VOICE1% (OP, 3)	:	リリース系数
VOICE1% (OP, 4)	:	サステインレベル
VOICE1% (OP, 5)	:	出力レベル
VOICE1% (OP, 6)	:	キーボードレイトスケーリング深さ
VOICE1% (OP, 7)	:	マルチプル
VOICE1% (OP, 8)	:	デチューンレイト
VOICE1% (OP, 9)	:	LFO 振幅変調深さ

再定義された音色はPLAY ALLOC, CLEAR, VOICE INIT が実行されるまで有効です。

注意：この命令やVOICE COPYで使用する配列は、5×10の大きさを持ち、OPTION BASEが0のものでなくてはなりません。したがって、プログラムの初めに、

```
OPTION BASE 0
DIM VOICE1%(4,9)
```

VOICE, VOICE COPY, VOICE INIT

のように宣言しておきます。

VOICE で配列を使用した後は ERASE を実行しないでください。ERASE を行うとサウンド制御命令の動作は保証されなくなります。ERASE を行う場合は、PLAY ALLOC、VOICE INIT などを実行してからにしてください。

参照：PLAY, PLAY ALLOC, VOICE COPY, VOICE INIT

VOICE COPY

S

機能 FM 音源の音色パラメータを配列にコピーします。

書式 VOICE COPY <音色番号>, <整数型配列名>

文例 VOICE COPY 4, PARA%

<音色番号> で指定された FM 音源の音色番号の音色パラメータを、<整数型配列名> にコピーします。

<音色番号> は 0~81 の値で指定します。<整数型配列> の内容は VOICE と同じ構成になっています。

参照：VOICE, VOICE INIT

VOICE INIT

S

機能 FM 音源の音色番号と音色パラメータを初期化します。

書式 VOICE INIT

文例 VOICE INIT

FM 音源の音色バンク(音色番号と音色パラメータ)をすべて初期化します。

VOICE で使用した音色パラメータ格納用の整数配列変数は、この命令の実行により、通常の配列変数となります (ERASE を行っても、別の用途に使用してもかまいません)。

注意：この命令を実行しても、SSG 音源は初期化されません。

参照：VOICE, VOICE COPY

VOICE LFO

S

機能 各チャンネルの出力にLFO効果を与えます。

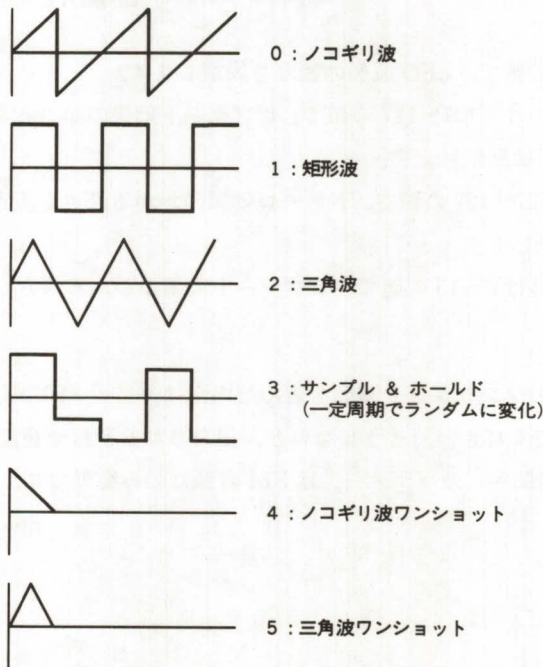
書式 VOICE LFO <チャンネル番号> [, <波形>] [, <SYNCディレイタイム>] [, <速さ>] [, <ピッチ変調深さ(微)>] [, <振幅変調深さ>] [, <ピッチ変調深さ(粗)>]

文例 VOICE LFO 2, 3, 10, 1500, -15, 0, 7

<チャンネル番号>で指定されたチャンネルの音にビブラート、トレモロなどのLFO効果をつけます。

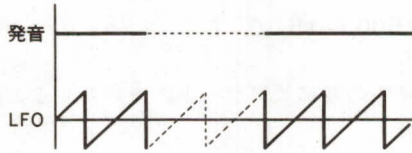
<波形>は0~5の範囲で、LFOの変調波形を次の中から選びます。

- 0 : ノコギリ波
- 1 : 矩形波
- 2 : 三角波
- 3 : サンプル&ホールド
(一定周期でランダムに変化)
- 4 : ノコギリ波ワンショット
- 5 : 三角波ワンショット

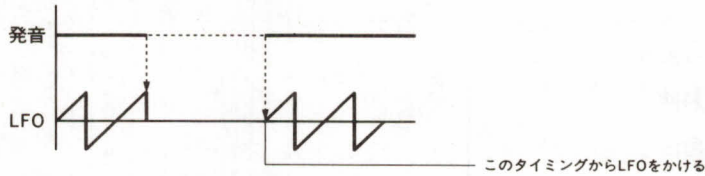


〈SYNCディレイタイム〉は0~255の値で、音が出てからLFO効果が現れるまでの時間を設定します。0の場合は、音の出るタイミングと関係なく、非同期にLFO効果を付加します。1~255の場合は、音が出てから〈SYNCディレイタイム〉で指定した時間だけ経過した後に同期してLFO効果を付加します。

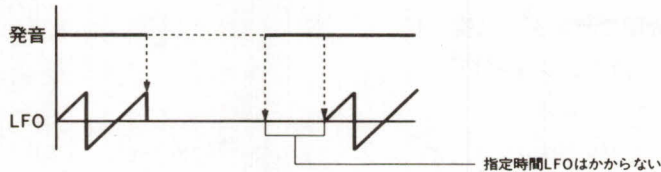
(a) SYNCディレイタイム=0のとき



(b) SYNCディレイタイム=1のとき



(c) SYNCディレイタイム>1のとき



〈速さ〉は0~16383の値で、LFO波形の速度を設定します。

〈ピッチ変調深さ(微)〉は-127~127の値で、ビブラート効果のかかる深さを設定します。値が負のときはLFO波形は反転します。

〈振幅変調深さ〉は-127~127の値で、トレモロ効果のかかる深さを設定します。値が負のときはLFOの波形は反転します。

〈ピッチ変調深さ(粗)〉は0~15の値で、ビブラート効果のかかる深さをおおまかに設定します。

注意：〈波形〉に4、5のワンショット波形を選んだ場合は、必ず〈SYNCディレイタイム〉を1以上の値にしてください。そうしないと、出力される音程や音量が不定となります。なお、〈振幅変調深さ〉のパラメータはFM音源にのみ有効です。

参照：PLAY

VOICE REG

S

機能	シンセサイザ LSI のレジスタに値を設定します。
書式	VOICE REG <レジスタ番号>, <式>
文例	VOICE REG &H28, 240

<レジスタ番号>には&H00～&HB2 (0～178) の値で、書き込むべきシンセサイザ LSI の内部レジスタ番号を指定します。

<式>には0～255の範囲の値を指定します。

この命令はMML (ミュージックマクロランゲージ) のYr, d コマンドと同じ機能をもっています。

注意: シンセサイザ LSI の内部には、サウンド制御命令の制御に直接関係するものもありますので、この命令でレジスタを直接変更するときには注意してください。特にレジスタ番号&H21～&H27, &H2D～&H2F の各レジスタへの書き込みは行わないでください。

参照: PLAY

WAIT

機能	コンピュータの入力ポートをモニタする間、プログラムの実行を停止します。
書式	WAIT <ポート番号>, <式1> [, <式2>]
文例	WAIT 1, &H22, &HFF

WAIT は、指定した入力ポートのビットパターンが指定した状態になるまでプログラムの実行を停止します。

<ポート番号>は入力ポートの番号で、0～32767(&H0～&H7FFF)の範囲内で指定します。

WAIT は、まずポートから読み込んだデータと<式2>とのXORをとり、次にその結果と<式1>のANDをとります。もしその結果が0(偽)なら、BASICはもう一度ポートの状態を読み込み、同じ操作を繰り返します。もし結果が0でない(真)ならば、プログラムの実行は次の文に移ります。<式2>を省略した場合は0とみなされます。

注意: WAIT の実行により、プログラムの実行が無限ループに入ってしまう場合があります。その場合にはコンピュータをリセットしなければなりません。

参照: INP, OUT

WHILE~WEND

機能 WHILE から WEND までの区間中にある一連の文を、指定条件が満足されている間、繰り返して実行します。

書式 WHILE <論理式>
 {
 WEND

文例 WHILE J=<5
 {
 WEND

WHILE~WEND ループ中(WHILE と WEND の間)におかれた文を、<論理式>が真である(0 でない)間、繰り返して実行します。<論理式> が偽(0)になると繰り返しは終わり、WEND に続く文に制御が移ります。

最初から<論理式>が偽の場合は、WHILE と WEND との間におかれた文は一度も実行されません。

WHILE~WEND は FOR~NEXT と同じ入れ子構造にすることができます。この場合には、それぞれの WEND はそれ以前の最も近くにある WHILE と対応します。WEND は WHILE と対になり、ループの終わりを示す働きをしますので省略することはできません。

注意: WHILE~WEND は必ず 1 対 1 に対応していなければなりません。

また、WHILE~WEND ループ内へ外部から GOTO などジャンプして入ってきたり、逆にループ内から外部へジャンプしたりするようなプログラムは、その動作が保証されなくなります。

参照: FOR...TO...STEP~NEXT, サンプルプログラム 11

WIDTH

機能 各種入出力機器やファイルに対して 1 行の長さなどを指定します。

書式 1) WIDTH <桁数> [, <行数>]
 2) WIDTH <ファイルディスクリプタ>, <サイズ>
 3) WIDTH # <ファイル番号>, <サイズ>

文例 WIDTH 80, 25
 WIDTH "LPT1 : ", 80
 WIDTH # 1, 100

書式 1) テキスト画面に表示する文字の量を指定します。1行あたり 40 文字、または 80 文字の〈桁数〉を設定します。また〈行数〉は 20 行か 25 行かのどちらかです。〈行数〉を省略した場合には、そのときの行数のまま変化しません。

書式 2) 〈ファイルディスクリプタ〉で指定したデバイスファイルに対して、1 行の長さを指定します。ここで指定可能なデバイスファイルは、RS-232C 回線ファイルとプリンタです。〈サイズ〉の値は 0 から 255 まで許されています。0 が 256 と解釈される以外は、指定した値が 1 行の文字数となります。たとえば、この命令がプリンタに対して実行されたとすれば、プリンタの 1 行に印字される文字数は〈サイズ〉で指定したものとなります。このようにプリンタに対して実行した場合、WIDTH LPRINT と同等の機能を得ることができます。初期状態では 255 に設定されています。

書式 3) 〈ファイル番号〉に割り当てられているバッファ (OPEN 参照) に対して、その大きさを〈サイズ〉で指定します。ここで指定できるデバイスファイルは、RS-232C 回線ファイルとプリンタです。〈サイズ〉は 0 から 255 まで許されています。0 が 256 と解釈される以外は、指定した値が 1 行の文字数となります。以後このファイル番号とデバイスとの入出力はこの〈サイズ〉単位で行われます。初期状態では 255 に設定されています。

なお、RS-232C 回線ファイルに対して WIDTH を実行すると、ファイルに対するデータの送出時、〈サイズ〉で指定された桁数の位置ごとに改行 (CR) コードを送出することになります。

注意: 2 バイト系日本語 1 文字は文字数 2 文字 (バイト) として数えます。また、日本語文字列の前と後にはそれぞれ文字数で 2 文字分の制御コード (KI/KO) が挿入されますのでこれも計算に入れなければなりません。

参照: WIDTH LPRINT, サンプルプログラム 33

WIDTH LPRINT

機能 プリンタに出力する 1 行あたりの文字数を設定します。

書式 WIDTH LPRINT 〈文字数〉

文例 WIDTH LPRINT 80

プリンタに印字する場合の、1 行あたりの文字数を設定します。〈文字数〉には 0 から 255 までの値を指定できます。0 が 256 と解釈される以外は、指定した値が 1 行の文字数となります。

注意: 2 バイト系日本語 1 文字は文字数 2 文字 (バイト) として数えます。また、日本語文字列の前と後にはそれぞれ文字数で 2 文字分の制御コード (KI/KO) が挿入されますのでこれも計算に入れなければなりません。

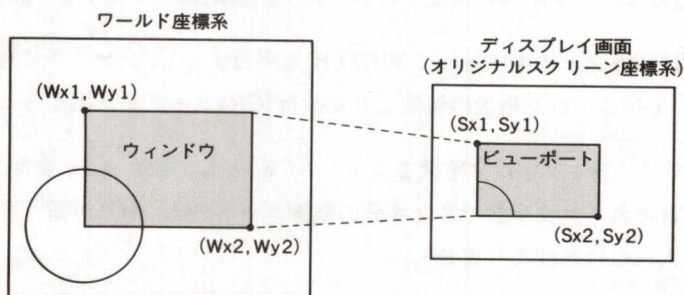
参照: WIDTH

WINDOW

機能	ビューポートに表示するワールド座標系内の領域を指定します。
書式	WINDOW (Wx1, Wy1)–(Wx2, Wy2)
文例	WINDOW (–300, –50)–(100, 70)

ワールド座標系上の(Wx1, Wy1)を左上の頂点、(Wx2, Wy2)を右下の頂点とする長方形で囲まれた領域を、ディスプレイ画面上のビューポート(VIEWで指定した領域)内に表示するよう設定します。

このようにして指定された領域はウィンドウと呼ばれ、ウィンドウから外れたところに描かれる図形は表示されなくなります。ウィンドウの大きさを変えると、対象領域の大きさが変わりますから、同じグラフィックス命令で描かれる図形でも、ビューポート内では異なった大きさで表示されることになります。



WINDOWも、VIEWと同じように領域の指定を行うだけで実際の画面に対する操作は行いませんので、WINDOWによってウィンドウの設定を変更するだけで、ビューポート中に表示される図形が変化することはありません。また、図形が何も描かれない領域をウィンドウとして設定すると、ディスプレイ画面上のビューポートには何も描かれないことになりますので、注意してください。

$Wx1 < Wx2$, $Wy1 < Wy2$ が成り立たない場合、あるいはこれらの座標がワールド座標系から外れている場合には、“Illegal function call” エラーとなります。WINDOWで指定する座標は、ワールド座標ですから、負の値や実数値もとることができます。

一度設定されたウィンドウは、次にWINDOWあるいはSCREENが実行されるまで変化しません。またWINDOWはLP(最終参照点)をウィンドウの左上の頂点に移動します。

注意: ワールド座標系がビューポート内に展開されるのはWINDOWの実行後であり、VIEWのみ実行してウィンドウは初期状態のままであれば、ビューポート内の座標は、(ワール

ド座標)=(スクリーン座標)となります。

参照：VIEW

WINDOW

関数

機能 現在のウィンドウの設定位置を得ます。

書式 WINDOW(<機能>)

文例 WY1=WINDOW(1)

WINDOWにより設定されている現在のウィンドウの位置(Wx1, Wy1 および Wx2, Wy2)を、<機能>に指定する値により、それぞれ個別に得ます。

<機能>には0~3の数値を指定します。

- 0 : ウィンドウの左上の頂点の X 座標(Wx1)
- 1 : ウィンドウの左上の頂点の Y 座標(Wy1)
- 2 : ウィンドウの右下の頂点の X 座標(Wx2)
- 3 : ウィンドウの右下の頂点の Y 座標(Wy2)

注意：WINDOW 関数で得られる値は、ワールド座標系の値となります。

参照：MAP, VIEW, VIEW(関数), WINDOW

WRITE

機能 画面にデータを出力します。

書式 WRITE <式> [| , | <式> ...]
 | ; |

文例 WRITE LISTNUM, GOODS\$, COST

<式>に指定された数値式あるいは文字式の値を、画面に出力します。

<式>はコンマ(,)あるいはセミコロン(;)により区切ります。PRINTと違い、コンマとセミコロンとの機能上の区別はありません。

WRITEは、ほぼ、PRINTと同じように式の値を出力しますが、不要な空白桁は詰め、それぞれの式の値の間は必ずコンマで区切ります。また、文字列はダブルクォーテーション(")で囲んで出力します。

WRITE は、各式の値を出力したのち、改行を行います。

参照：PRINT, WRITE #, サンプルプログラム 12

WRITE

機能	ファイルにデータを書き出します。
書式	WRITE # <ファイル番号>, <式> [, <式> ...] ;
文例	WRITE # 1, LISTNUM, GOODS\$, COST

出力モードでオープンしたファイル(シーケンシャルファイル, スクリーンファイル, プリンタなど)や, RS-232C 回線ファイルに, <式> により指定した文字列, 数値などのデータを書き出します。

<ファイル番号>には, OPEN によって, そのファイルをオープンしたときに使った番号を指定します。

<式>を複数指定する場合は, コンマ(,)あるいはセミコロン(;)により区切ります。PRINT #と違い, コンマとセミコロンとの機能上の区別はありません。

WRITE #は, ほぼ, PRINT #と同じように式の値を出力しますが, 不要な空白桁は詰め, それぞれの式の値の間は必ずコンマで区切ります(コンマそのものをデータとして出力します)。また, 文字列はダブルクォーテーション(")で囲んで出力します。

WRITE #は, 各式の値を出力したのち, 改行コード(CHR\$(13))を書き出します。

WRITE #は, 区切り記号としてコンマを必ず出力し, 不要な空白の出力は行わないため, PRINT #に比べファイルの使用領域を節約することができます。

注意：出力対象ファイルが "SCRN:" の場合, 日本語データを出力することはできません。

参照：PRINT #, WRITE

サンプルプログラム

第4章 サンプルプログラム

この章の見方

この章には、N₈₈-BASIC(86)の各種命令・関数を使用したサンプルプログラムが掲載されています。

それぞれのプログラムは、相互に関連のある命令・関数を使って構成してあります。また、ポイントとなる各命令・関数には、右側に解説がつけてあります。

なお、各プログラムの番号は、「第3章 命令リファレンス」中の各命令の“参照”部分に示されたサンプルプログラム番号と対応しています。

以下に、本章中に掲載されているサンプルプログラムの番号・タイトルと、各プログラム中で使用されている主な命令の、一覧を示します。

- | | |
|--|--|
| 1. プログラムの連結 (その1)
CHAIN, COMMON, OPEN, FIELD,
PUT | 8. 条件分岐
IF...THEN~ELSE, ABS, SQR |
| 2. プログラムの連結 (その2)
CHAIN, COMMON, FIELD, GET | 9. 式の値による分岐
ON...GOTO, SGN |
| 3. DATA 行の読み込み
READ, DATA, RESTORE | 10. 変数の値の入れ換え
SWAP, LOCATE, POS, CSRLIN |
| 4. 変数の型宣言
DEFINT, DEFSNG, DEFDBL | 11. WHILE~WEND ループ
WHILE~WEND |
| 5. 配列変数の宣言と OPTION BASE
DIM, OPTION BASE, ERASE | 12. PRINT と WRITE
PRINT, WRITE |
| 6. FOR~NEXT ループ
FOR...TO...STEP~NEXT, SIN, COS,
PSET | 13. データの編集出力
PRINT USING |
| 7. サブルーチンの呼び出しとリターン
GOSUB, RETURN | 14. 円の描画
CIRCLE, READ, DATA |
| | 15. パレット (8色中・8色モード)
{1}COLOR, LINE, {2}COLOR |

16. パレット (4096 色中・16 色モード)
〔1〕COLOR, LINE, 〔2〕COLOR
17. DRAW
POINT, DRAW, PSET, PRESET,
LINE, PAINT
18. グラフィックパターンの読み込みと表示
GET@, PUT@, LINE, CIRCLE
19. グラフィック画面のスクロール
PUT@, ROLL
20. ビューポートの設定
SCREEN, WINDOW, VIEW, LINE
21. ウィンドウの設定
SCREEN, WINDOW, VIEW, CIRCLE
22. エラー処理ルーチン
ON ERROR GOTO, ERROR, ERR,
RESUME
23. 文字とキャラクタコードの変換
ASC, MID\$, CHR\$
24. 8進変換と16進変換
OCT\$, HEX\$, STR\$
25. 文字列の抜き出し
LEFT\$, RIGHT\$, MID\$, LEN, SPACE\$
26. 数値データの文字型化
MKI\$, MKS\$, MKD\$, OPEN, CLOSE,
FIELD, LSET, PUT
27. 文字型化数値データの数値化
CVI, CVS, CVD, OPEN, CLOSE,
FIELD, GET
28. シーケンシャルファイル
OPEN, CLOSE, PRINT #, INPUT #,
EOF
29. ランダムファイル
OPEN, FIELD, CLOSE, LOF, GET,
LSET, PUT, INPUT\$, INKEY\$
30. キー割り込みルーチン
ON KEY GOSUB, KEY(n) ON,
RETURN, SIN, COS
31. タイマー割り込みルーチン
TIME\$, TIME\$ ON,
ON TIME\$ GOSUB, RETURN, BEEP
32. 乱数の発生
RANDOMIZE, RND
33. テキスト画面のモード設定と文字に対する色・機能の設定
CONSOLE, WIDTH, LOCATE,
COLOR@, GOSUB, RETURN
34. 利用者定義文字の登録
KPLOAD, KNJ\$
35. 2バイト系日本語文字列の変換と抜き出し
AKCNV\$, KACNV\$, KMID\$
36. 2バイト系日本語文字列と漢字コードの相互変換
JIS\$, KNJ\$, KMID\$
37. 特定タイプの文字列の抜き出し
KEXT\$

38. 2バイト系日本語文字を含む文字列の長さ
とタイプ

KLEN, KTYPE

39. 利用者定義関数

DEF FN

40. 電話制御機能(拡張機能)

CMD ERROR ON, CMD LINE ON,
CMD LINE OPEN, CMD MODE CUT,
CMD ON ERROR GOSUB,
CMD ON LINE GOSUB,
CMD RECEIVE, STATUS MODE

41. サウンド機能(拡張機能)

ON PLAY GOSUB, PLAY,
PLAY ALLOC, PLAY ON

1. プログラムの連結(その1)

```

100 DIM IDENT(200)
110 COMMON IDENT(),COUNT —— 引き渡す変数を宣言。
120 OPEN "DATA" AS #1 —— "DATA" ファイルをランダムモードでオープン。
130 FIELD #1,4 AS I$,20 AS N$,50 AS A$ —— 会員番号, 氏名, 住所の各フィールド変数を定義。
140 INPUT "会員番号(1-999)":I
150 IF I=0 THEN GOTO *PRINTOUT
160 IF I>1000 THEN STOP
170 INPUT "氏名":NAME1$
180 INPUT "住所":ADR$
190 LSET I$=MK$(I)
200 LSET N$=NAME1$
210 LSET A$=ADR$ —— 各データを各フィールド変数にセット。
220 PUT #1,I —— 会員番号と同番のレコードにデータを書き出す。
230 COUNT=COUNT+1
240 IDENT(COUNT)=I —— 書き出された会員番号(レコード番号)を憶えておく。
250 PRINT:GOTO 140
260 *PRINTOUT
270 CHAIN "CHAIN.S02" —— ^CHAIN.S02" に実行を移す。
280 END —— "DATA" ファイルはオープンしたまま。

```

2. プログラムの連結(その2)

```

100 COMMON IDENT(),COUNT —— 引き渡される変数を宣言。
110 PRINT:PRINT
120 FIELD #1,4 AS I$,20 AS N$,50 AS A$ —— "DATA" ファイルはOPEN状態のままCHAIN
130 IF COUNT=0 THEN PRINT "END":END —— されるので, 新たにOPENしなくてよい。
140 GET #1,IDENT(COUNT) —— 実際に書き出されているレコードのみを読み込む。
150 PRINT "会員番号:":CVS(I$)
160 PRINT "氏名:":N$
170 PRINT "住所:":A$
180 COUNT=COUNT-1
190 PRINT:GOTO 130

```

3. DATA 行の読み込み

```

100 READ A,B,C
110 RESTORE —— 最初のDATA行(190行)から読み出すように指定。
120 READ D,E,F
130 RESTORE *STRDATA —— *STRDATA行(210行)以降のDATA行(220行)から読み出すように指定。
140 READ A$,B$,C$
150 PRINT A,B,C
160 PRINT D,E,F
170 PRINT A$,B$,C$
180 END
190 DATA 1,2,3
200 DATA 4,5,6 —— このプログラムでは, 結局このDATA行は読み出されない。
210 *STRDATA
220 DATA AA,BB,CC

```

4. 変数の型宣言

```

100 DEFINT J-M —— J~Mで始まる変数を整数型として定義。
110 DEFSNG A,B —— A, Bで始まる変数を単精度実数型として定義。
120 DEFDBL D —— Dで始まる変数を倍精度実数型として定義。
130 J1=1.23:K=65.643 —— J1, Kには整数に変換された値が代入される。
140 ABC=1.23:BBB=65.634
150 D=3.141592654000003#
160 D%=3.141592654000003# —— D%は, 整数型宣言文字(%)が付加されているため,
170 PRINT "J1=":J1,,"K=":K —— 120行の宣言にかかわらず, 整数型として扱われる。
180 PRINT "ABC=":ABC,,"BBB=":BBB
190 PRINT "D=":D,,"D%=":D%
200 END

```

5. 配列変数の宣言と OPTION BASE

```

100 OPTION BASE 1
110 DIM KEISAN(9,9)
120 FOR I=1 TO 9:FOR J=1 TO 9
130 KEISAN(I,J)=I*J
140 NEXT J,I
150 FOR I=1 TO 9:FOR J=1 TO 9
160 PRINT USING "####":KEISAN(J,I);
170 NEXT J:PRINT
180 NEXT I
190 PRINT:PRINT
200 ERASE KEISAN
210 DIM KEISAN(9,9)
220 FOR I=1 TO 9:FOR J=1 TO 9
230 KEISAN(I,J)=I+J
240 NEXT J,I
250 FOR I=1 TO 9:FOR J=1 TO 9
260 PRINT USING "####":KEISAN(J,I);
270 NEXT J:PRINT
280 NEXT I
290 END

```

添字の最小値を1とする。
 要素数9×9の2次元配列変数KEISAN()を宣言。
 KEISAN()を消去。
 KEISAN()を新たに宣言。

6. FOR~NEXT ループ

```

100 SCREEN 0:CLS 3
110 FOR P=1 TO 7:C=0
120 FOR R=10 TO 200 STEP 20
130 FOR I=0 TO 3.14 STEP .05
140 Y=SIN(I)*COS(I)
150 X=SIN(I)*SIN(I)
160 Y=Y*R+100
170 X=X*R*2.5+100-C
180 PSET (X,Y),P
190 NEXT I
200 C=C+10
210 NEXT R,P
220 END

```

— 楕円をプロットする。
 — 楕円の大きさを変える。
 — 描き始める位置を変える。
 — パレット番号を変える。

7. サブルーチンの呼び出しとリターン

```

100 *START
110 INPUT "底辺:";TEIHEN
120 INPUT "高さ:";TAKASA
130 GOSUB *MENSEKI
140 PRINT "面積は ";MENSEKI
150 PRINT
160 GOTO *START
170 *MENSEKI
180 MENSEKI=TEIHEN*TAKASA/2
190 RETURN

```

三角形の面積を計算するサブルーチン。
 サブルーチンの最後はRETURNで終わる。

8. 条件分岐

```

100 INPUT A
110 FLG=0
120 IF A<0 THEN A=ABS(A):FLG=1
130 PRINT SQR(A);
140 IF FLG THEN PRINT "i" ELSE PRINT
150 END

```

入力された値が負ならばFLGに1を代入。
 平方根の計算。
 入力値が負のときには、平方根の結果に
 "i" (複素表示)を付加する。

9. 式の値による分岐

```

100 *ENTRY
110 INPUT "N=":N
120 SG=SGN(N)+1 _____ 入力値に対するSGN関数の値-1, 0, +1を, 0, 1, 2に変換.
130 ON SG GOTO *ZERO,*PLUS _____ 入力値の負, 0, 正に従って, *MINUS,
140 *MINUS:PRINT "MINUS!":GOTO *ENTRY *ZERO, *PLUSに飛ぶ.
150 *ZERO:PRINT "ZERO!":GOTO *ENTRY
160 *PLUS:PRINT "PLUS!":GOTO *ENTRY

```

10. 変数の値の入れ換え

```

100 DIM A(10),B(10)
110 CLS
120 FOR I=1 TO 10
130 A(I)=I*2:B(I)=I*3:GOSUB *PRINTAB
140 NEXT I:LOCATE 0,Y+3
150 FOR I=1 TO 10
160 SWAP A(I),B(I):GOSUB *PRINTAB _____ A()とB()の各要素をすべて交換.
170 NEXT I:LOCATE 0,Y+2
180 END
190 *PRINTAB _____ A()とB()をそれぞれ表示するサブルーチン.
200 X=POS(0):Y=CSRLIN
210 LOCATE X,Y:PRINT USING "###";A(I);
220 LOCATE X,Y+1:PRINT USING "###";B(I);
230 LOCATE X+5,Y
240 RETURN

```

11. WHILE~WEND ループ

```

100 I=1
110 WHILE I<=20 _____ Iが20以下のうちは120行~190行を繰り返す.
120 PRINT USING "## -";I;
130 J=1
140 WHILE J<=I _____ JがI以下のうちは150行~170行を繰り返す.
150 PRINT USING "###";J;
160 J=J+1
170 WEND
180 PRINT:I=I+1
190 WEND

```

12. PRINT と WRITE

```

100 A%=123
110 B!=9.87654E+31
120 C#=3.14159265359#
130 D$="N88-BASIC"
140 PRINT A%;B!,C#;D$ _____ PRINTとWRITEでは出力結果が異なる.
150 WRITE A%;B!,C#;D$
160 END

```

13. データの編集出力

```

100 PRINT USING "!";"NEC COMPUTER"
110 PRINT USING "&      &";"NEC COMPUTER"
120 PRINT USING "#####";123.456
130 PRINT USING "#####.#";123.456
140 PRINT USING "+#####.#";123.456
150 PRINT USING "#####.#-";-123.456
160 PRINT USING "*****.#";123.456
170 PRINT USING "¥¥#####.#";123.456
180 PRINT USING "**¥#####.#";123.456
190 PRINT USING "#####.#";1234.56
200 PRINT USING "#####.#^^^^";1234.56
210 PRINT USING "¥¥#####.#-";123.456
220 PRINT USING "#####";123456!
230 PRINT USING "NEC @ COMPUTER";"PERSONAL"
240 END
    
```

書式制御文字列の典型的な使い方。

14. 円の描画

```

100 SCREEN 0,0:CLS 3
110 FOR R=1 TO 100 STEP 5
120 CIRCLE(150,100),R,(R MOD 7)+1,,,R/100
130 NEXT R
140 ST=.00001
150 DATA 25,5,40,13,17
160 FOR I=1 TO 5
170 READ DAT:EN=ST+DAT/100*3.14*2
180 CIRCLE(450,100),100,I,-ST,-EN
190 ST=EN
200 NEXT I
    
```

半径, パレット番号, 比率を変えながら楕円を描く。

DATA行(150行)の百分率データを円の角度に変換。

パレット番号, 開始角度, 終了角度を変えながら扇形を描き, 円グラフを作る。

15. パレット(8色中・8色モード)

```

100 COLOR ,,,,0:CLS 3
110 FOR I=1 TO 7
120 COLOR=(I,7)
130 NEXT I
140 FOR I=1 TO 7
150 LINE(0,I*20)-STEP(639,10),I,BF
160 NEXT I
170 FOR I=1 TO 7
180 FOR J=1 TO 7
190 COLOR=(J,I)
200 GOSUB *WAITSUB
210 NEXT J
220 NEXT I
230 COLOR
240 END
250 *WAITSUB
260 FOR K=0 TO 100
270 NEXT K
280 RETURN
    
```

8色中・8色モード。

すべてのパレットを白に設定。

パレット番号ごとに1本ずつ, 計7本の帯を描く。

各パレット番号の帯に, 1~7のカラーコードを順に設定していく。

全パレットを初期化。

遅延用サブルーチン。

16. パレット(4096色中・16色モード)

```

100 SCREEN 3,0:COLOR ,,,,2:CLS 3 —— 高分解能カラーモード, 4096色中・16色モード。
110 FOR I=1 TO 15
120 COLOR=(I,&HFFF) —— すべてのパレットを白に設定。
130 NEXT I
140 FOR I=1 TO 15
150 LINE(0,I*20)-STEP(639,10),I,BF —— パレット番号ごとに1本ずつ, 計15本の帯を描く。
160 NEXT I
170 FOR I=1 TO 15
180 FOR J=1 TO 15
190 COLOR=(J,16*I^2) —— 各パレット番号の帯に, カラーコードを順に設定していく。
200 GOSUB *WAITSUB
210 NEXT J
220 NEXT I
230 COLOR —— 全パレットを初期化。
240 END
250 *WAITSUB —— 遅延用サブルーチン。
260 FOR K=0 TO 100
270 NEXT K
280 RETURN

```

17. DRAW

```

100 SCREEN 0,0,0,1:CLS 3
110 POINT(320,100)
120 A$="C4U60R60D60L60"
130 DRAW A$
140 DRAW "BE45A2S0.5X=A$:"
150 DRAW "A0BE10P"
160 PSET(440,40)
170 LINE -STEP(60,60),7,BF
180 PRESET(455,55)
190 LINE -STEP(30,30),0,B
200 PAINT(456,56),0
210 END

```

DRAWで図形を描く。

LINE, PAINTで同じ図形を描く。

18. グラフィックパターンの読み込みと表示

```

100 SCREEN 0,0:CLS 3:COLOR ,,,,1
110 XD=40:YD=20
120 BYTE=((XD+7)¥8)*YD*3+4
130 FACT=BYTE¥2+1
140 DIM G%(FACT)
150 LINE(0,0)-STEP(XD-1,YD-1),1,B
160 CIRCLE(XD/2-1,YD/2-1),YD/2,2
170 GET(0,0)-STEP(XD-1,YD-1),G% —— 図形をG%()に読み込む。
180 FOR X=0 TO 500 STEP 100
190 PUT(X,100),G% —— 読み込んだ図形を6か所にPUT。
200 NEXT

```

8色モードで40×20ドットのパターンを読み込むための配列変数G%()を確保する。

19. グラフィック画面のスクロール

```

100 SCREEN 1,0:CLS 3
110 FOR I=&H3000 TO &H4F00 STEP &H100
120 FOR J=&H21 TO &H7E
130 KCODE=I+J
140 PUT(X,168),KANJI(KCODE),PSET —— 漢字をコード順に表示。
150 X=X+20
160 IF X>623 THEN X=0:ROLL 18 —— 18ドット分ずつスクロールアップ。
170 NEXT J
180 NEXT I

```

20. ビューポートの設定

```

100 SCREEN 0,0:CLS 3
110 WINDOW(0,0)-(639,199)
120 C=6:GOSUB *RECT
130 VIEW(1,1)-(638,99),0,7
140 C=5:GOSUB *RECT
150 VIEW(214,1)-(428,198),0,7
160 C=4:GOSUB *RECT
170 VIEW(0,0)-(639,199):CLS 3
180 END
190 *RECT
200 LINE(50,50)-(600,150),C,BF
210 LOCATE 0,0:PRINT "どれかキーを押してください";
220 A$=INPUT$(1)
230 RETURN
    
```

ウィンドウの設定。
 ビューポートの設定を変えながら、同一座標の長方形を描く。
 ビューポートをもとに戻す。
 べた塗りの長方形を描くサブルーチン。

21. ウィンドウの設定

```

100 SCREEN 0,0:CLS 3
110 C=1
120 VIEW(200,50)-(400,150),,7
130 WINDOW(-100,-100)-(100,100)
140 GOSUB *CIRC
150 WINDOW(-100,-100)-(0,0)
160 GOSUB *CIRC
170 WINDOW(0,0)-(100,100)
180 GOSUB *CIRC
190 WINDOW(-500,-500)-(500,500)
200 GOSUB *CIRC
210 VIEW(0,0)-(639,199):CLS 3
220 END
230 *CIRC
240 CIRCLE(0,0),100,C
250 LOCATE 0,0:PRINT "どれかキーを押してください";
260 A$=INPUT$(1)
270 C=C+1:CLS 2
280 RETURN
    
```

ビューポートの設定。
 ウィンドウの設定を変えながら、同一座標の円を描く。
 ビューポートをもとに戻す。
 円を描くサブルーチン。

22. エラー処理ルーチン

```

100 PRINT "XのY乗を求めます"
110 ON ERROR GOTO *ERRORMES
120 *START
130 INPUT "X:";X:INPUT"Y:";Y
140 IF X=0 THEN ERROR 250
150 Z=X^Y:PRINT X;"の";Y;"乗は";Z;"です"
160 *RETRY
170 INPUT "もう一度やりますか(Y/N)";A$:PRINT
180 IF A$="Y" OR A$="y" THEN *START
190 ON ERROR GOTO 0
200 END
210 *ERRORMES
220 IF ERR=250 THEN PRINT "定義されません"
230 IF ERR=6 THEN PRINT "オーバーフローです"
240 RESUME *RETRY
    
```

エラー発生時の分岐先を定義。
 エラー番号250を独自に定義。
 プログラム終了時には必ず実行しておく。
 エラー処理ルーチン。
 エラー番号250のときの処理。
 オーバーフローのときの処理。
 エラー処理ルーチンからの戻りにはRESUMEを使う。

23. 文字とキャラクタコードの変換

```

100 LINE INPUT "英数字を打ってください ";A$
110 IF A$="" THEN 190
120 B$=""
130 FOR I=1 TO LEN(A$)
140   D$=MID$(A$,I,1):D=ASC(D$)
150   IF D>=97 AND D<=122 THEN D$=CHR$(D-32) —— 英小文字のみを、英大文字に変換。
160   B$=B$+D$
170 NEXT I
180 PRINT:PRINT B$
190 END

```

24. 8進変換と16進変換

```

100 FOR I=0 TO 16
110   DE$=RIGHT$(" "+STR$(I),3)
120   OC$=RIGHT$(" "+OCT$(I),3)
130   HE$=RIGHT$(" "+HEX$(I),3)
140   PRINT "10進: ";DE$,
150   PRINT " 8進: ";OC$,
160   PRINT "16進: ";HE$
170 NEXT I

```

0~16の整数を、10進表記、8進表記、16進表記の文字列に変換。

25. 文字列の抜き出し

```

100 INPUT"英数字をなにか打ってください ",A$
110 PRINT
120 PRINT "全体を2つに分割します"
130 AL$=LEFT$(A$,LEN(A$)/2)
140 AR$=RIGHT$(A$,LEN(A$)-LEN(AL$)*2)
150 PRINT AL$;SPACE$(3);AR$
160 PRINT
170 PRINT "2文字ずつに分けます"
180 FOR I=1 TO LEN(A$) STEP 2
190   PRINT MID$(A$,I,2);SPACE$(3);
200 NEXT I
210 END

```

A\$を左右半分ずつに分割する。

A\$を2文字ずつに分ける。

26. 数値データの文字型化

```

100 OPEN "RDATA" AS #1
110 FIELD #1,2 AS A$,4 AS B$,8 AS C$
120 A%=987:A1=123.456:A#=1234567891234#
130 LSET A$=MKI$(A%)
140 LSET B$=MKS$(A1)
150 LSET C$=MKD$(A#)
160 PUT #1
170 PRINT "このプログラムで書き込むデータは"
180 PRINT "次のプログラムで読み出せます"
190 CLOSE:END

```

ランダムモードで"RDATA"をオープン。
整数用2バイト、単精度実数用4バイト、倍精度実数用8バイトのフィールド変数を定義。

各型の数値を文字型データに変換し、フィールド変数にセット。

ファイルに書き込む。

27. 文字型化数値データの数値化

```

100 OPEN "RDATA" AS #1
110 FIELD #1,2 AS A$,4 AS B$,8 AS C$
120 PRINT "前のプログラムで書き込んだデータを"
130 PRINT "読み出します"
140 GET #1
150 A%=CVI(A$)
160 A1=CVS(B$)
170 A#=CVD(C$)
180 PRINT A%:PRINT A1:PRINT A#
190 CLOSE:END

```

ランダムモードで"RDATA"をオープン。
整数用2バイト、単精度実数用4バイト、倍精度実数用8バイトのフィールド変数を定義。

ファイルから読み込む。

文字型データを、各型の数値に戻す。

28. シーケンシャルファイル

```

100 F$="DATA.D"+RIGHT$(DATE$,2)
110 OPEN F$ FOR OUTPUT AS #1_____ "DATA.Dxx"(xxは日付)を出力モードでオープン。
120 PRINT #1,DATE$;",";TIME$_____ 日付と時刻を書き出す。
130 PRINT "今日のデータを入力してください"
140 PRINT:PRINT "リターンキーだけ押すと終了します"
150 PRINT:INPUT "品名":NA$
160 IF NA$="" THEN CLOSE:GOTO *INDATA_____ 入力の終了。ファイルのクローズ。
170 INPUT "価格":PRC
180 INPUT "個数":N%
190 PRINT #1,NA$;",";PRC;",";N%_____ 品名、価格、個数の各データを書き出す。
200 GOTO 140
210 *INDATA
220 TOTAL=0
230 OPEN F$ FOR INPUT AS #1_____ 同じファイルを入力モードでオープン。
240 INPUT #1,DA$,TI$_____ 日付と時刻を読み込む。
250 PRINT:PRINT "日付 : ";DA$,"時刻 : ";TI$
260 PRINT
270 IF EOF(1) THEN *TOTALPRN
280 INPUT #1,NA$,PRC,N%_____ 品名、価格、個数の各データを読み込む。
290 SUM=PRC*N%_____ 小計。
300 PRINT NA$;TAB(10);PRC;"*";N%;"=";SUM_____ 品名、価格、個数および小計の表示。
310 TOTAL=TOTAL+SUM_____ 合計。
320 GOTO 270
330 *TOTALPRN
340 PRINT:PRINT "TOTAL=";TOTAL_____ 合計の表示。
350 PRINT:PRINT "今日のデータを ";F$;" として作成しました"
360 CLOSE:END_____ ファイルをクローズ、終了。

```

29. ランダムファイル

```

100 CLS:PRINT"住所録のファイル名は?"
110 PRINT:LINE INPUT F$
120 IF F$="" THEN 100
130 OPEN F$ AS #1_____ 指定ファイル名のファイルをランダムモードでオープン。
140 FIELD #1,30 AS NAM$,20 AS TEL$,50 AS ADR$_____ 名前(30バイト)、TEL(20バイト)
150 *MENU
160 PRINT:PRINT
170 PRINT "<<メニュー>>":PRINT
180 PRINT " 検索/修正 : 1"
190 PRINT " 新しく登録 : 2"
200 PRINT " 終わり : 0"
210 PRINT:PRINT "操作を番号で選択してください ";
220 Q$=INPUT$(1):PRINT Q$
230 ON VAL(Q$)+1 GOTO *EXIT,*INNAME,*NEWENTRY
240 GOTO *MENU
250 *INNAME_____ 検索のルーチン。
260 IF LOF(1)<>0 THEN 330
270 PRINT:PRINT F$+" にはデータがありません"
280 PRINT:PRINT"新しく登録しますか?(Y/N) ";
290 Q$=INKEY$:IF Q$="" THEN GOTO 290
300 IF Q$="N" OR Q$="n" THEN GOTO *MENU
310 IF Q$="Y" OR Q$="y" THEN GOTO *NEWENTRY
320 GOTO 290
330 NH=1
340 PRINT:PRINT "捜す名前は?":PRINT:INPUT "名前:":HNAM$ _____ 検索したい名前の入力。
350 IF HNAM$="" THEN *MENU
360 HL=LEN(HNAM$)
370 IF NH>LOF(1) THEN PRINT:PRINT"その名前は登録されていません":GOTO *MENU
380 GET #1,NH
390 IF HNAM$<>LEFT$(NAM$,HL) THEN NH=NH+1:GOTO 370
400 PRINT:PRINT"名前: ";NAM$_____ ファイル中から1レコードずつ読み込み、名前を検索する。
410 PRINT"TEL: ";TEL$ _____ 検索結果の表示。
420 PRINT"住所: ";ADR$
430 PRINT:PRINT"修正しますか?(Y/N) ";
440 Q$=INKEY$:IF Q$="" THEN GOTO 440
450 IF Q$="N" OR Q$="n" THEN PRINT "N":GOTO *MENU
460 IF Q$="Y" OR Q$="y" THEN PRINT "Y":GOTO *CHANGE
470 GOTO 440

```

```

480 *CHANGE _____ 修正のルーチン。
490 PRINT:PRINT"リターンキーだけ押すとその項目は修正されません"
500 DNAMS="":DTEL$="":DADR$=""
510 PRINT:INPUT"名前":DNAMS
520 IF DNAMS<>" " THEN LSET NAM$=DNAMS
530 INPUT"TEL":DTEL$
540 IF DTEL$<>" " THEN LSET TEL$=DTEL$
550 INPUT"住所":DADR$
560 IF DADR$<>" " THEN LSET ADR$=DADR$
570 PUT #1,NH:GOTO *MENU _____ 修正後のデータを書き出す。
580 *NEWENTRY _____ 新規登録のルーチン。
590 PRINT:PRINT"新しく登録します"
600 NH=LOF(1)+1 _____ ファイルの最終レコードの次のレコードを指定。
610 PRINT:INPUT"名前":NNAMS
620 IF NNAMS="" THEN *MENU
630 INPUT"TEL":NTEL$
640 INPUT"住所":NADR$
650 LSET NAM$=NNAMS
660 LSET TEL$=NTEL$
670 LSET ADR$=NADR$
680 PUT #1,NH:GOTO *MENU _____ データを書き出す。
690 *EXIT _____ 終了のルーチン。
700 PRINT:PRINT"終了"
710 CLOSE:END _____ ファイルのクローズ、終了。

```

各データを各フィールド変数にセット。

各データを各フィールド変数にセット。

30. キー割り込みルーチン

```

100 CLS
110 ON KEY GOSUB *SINSUB,*COSSUB,*ENDSUB _____ f・1, f・2, f・3が押されたときの分岐先を定義。
120 FOR I=1 TO 3
130 KEY(I) ON _____ f・1, f・2, f・3の各キーの割り込みを許可。
140 NEXT I
150 DEG=0:PI=3.1416:FLAG=1
160 PRINT"f・1 または f・2 を押してください (f・3 で終了). "
170 *START
180 TH=DEG/180*PI
190 LOCATE 0,10
200 ON FLAG GOTO *SINC,*COSC
210 *SINC:PRINT"TH=":DEG:TAB(12):"SIN(TH)=":SIN(TH)::GOTO *EXIT _____ 10度ごとの
220 *COSC:PRINT"TH=":DEG:TAB(35):"COS(TH)=":COS(TH)::GOTO *EXIT _____ 10度ごとの
230 *EXIT:DEG=DEG+10 _____ 余弦値を表示。
240 FOR I=0 TO 1000:NEXT _____ 表示遅延用タイマー。
250 LOCATE 0,10:PRINT STRING$(70," ");
260 GOTO *START
270 *SINSUB:FLAG=1:RETURN _____ f・1が押されたらFLAG=1にする。
280 *COSSUB:FLAG=2:RETURN _____ f・2が押されたらFLAG=2にする。
290 *ENDSUB:KEY OFF:END _____ f・3が押されたら終了。

```

31. タイマー割り込みルーチン

```

100 CLS:CONSOLE ,,,1:FLG=0
110 PRINT"目覚まし時計"
120 LOCATE 0,10:PRINT"現在の時刻は ":TIMES$
130 LOCATE 0,2:INPUT"セット時刻は (HH:MM:SS) ":ST$
140 ON TIMES$=ST$ GOSUB 200 _____ セット時刻到来時の分岐先を定義。
150 TIMES$ ON _____ クロック割り込みを許可。
160 LOCATE 13,10:PRINT TIMES$: _____ 現在時刻の表示、セット時刻までは
170 IF FLG=0 THEN 160 _____ 160行~170行が繰り返し実行される。
180 CLS:LOCATE 30,10:PRINT"おはよう!!!"
190 END
200 PRINT
210 COLOR 7:BEEP 1
220 PRINT"もう起きる時間ですよ!!!"
230 IF INKEY$="" THEN 220 _____ なにかキーを押さない限り、メッ
240 BEEP 0:FLG=1 _____ セージが表示され、スピーカは鳴
250 RETURN _____ りっぱなしとなる。
_____ なにかキーが押されると、スピーカは
_____ 鳴りやみ、メインルーチンに戻る。

```

32. 乱数の発生

```

100 RANDOMIZE
110 DIM SUM(6)
120 FOR I=1 TO 100
130 DA=INT(RND*6+1)
140 SUM(DA)=SUM(DA)+1
150 NEXT I
160 FOR I=1 TO 6
170 PRINT I;"-":SUM(I);"% "
180 NEXT I

```

33. テキスト画面のモード設定と文字に対する色・機能の設定

```

100 WIDTH 80,25:CLS
110 CONSOLE 0,25,0,0 _____ 白黒モード.
120 LOCATE 0,0
130 PRINT "白黒モード":PRINT
140 GOSUB *PRINTCHR
150 GOSUB *KEYSUB
160 CONSOLE 0,25,0,1 _____ カラーモード.
170 LOCATE 0,0
180 PRINT "カラーモード":PRINT
190 GOSUB *PRINTCHR
200 END
210 *PRINTCHR
220 FOR I=0 TO 31
230 PRINT "-COMPUTER-";
240 NEXT I
250 GOSUB *KEYSUB
260 FOR I=0 TO 7
270 COLOR@(I*10,2)-(I*10+9,5),I _____ 文字にファンクションコードを設定.
280 NEXT I
290 RETURN
300 *KEYSUB _____ キー入力待ちサブルーチン.
310 LOCATE 0,10
320 PRINT "どれかキーを押してください";
330 A$=INPUT$(1)
340 RETURN

```

34. 利用者定義文字の登録

```

100 DIM CHRPTN%(17) _____ 利用者定義文字パターンを格納するための配列変数を確保.
110 CHRPTN%(0)=16:CHRPTN%(1)=16
120 FOR I=2 TO 17:CHRPTN%(I)=2^(I-2)-1:NEXT I _____ パターンを格納する.
130 KPLOAD &H762A,CHRPTN% _____ パターンをシステムに登録.
140 FOR I=0 TO 4
150 PRINT KNJ$("1B4B")+KNJ$("762A")+KNJ$("1B4B"); _____ 表示.
160 NEXT

```

35. 2バイト系日本語文字列の変換と抜き出し

```

100 A0$="イロハ漢字ABC"
110 PRINT A0$
120 K0$=AKCNV$(A0$) _____ 1バイト系英数カナ文字を2バイト系全角文字に変換.
130 PRINT K0$
140 K1$=KMID$(K0$,1,4)+KMID$(K0$,7,4) _____ 2バイト系全角文字に変換された文字列から
"漢字"を取り除く.
150 PRINT K1$
160 A1$=KACNV$(K1$) _____ 2バイト系全角文字を1バイト系英数カナ文字に変換.
170 PRINT A1$

```

36. 2バイト系日本語文字列と漢字コードの相互変換

```

100 A$=JIS$(KMID$("漢字",2,1))
110 PRINT A$
120 B$=KNJ$("1B4B")+KNJ$(A$)+KNJ$("1B48")
130 PRINT B$

```

最初のKIコードをスキップし、
"漢"のJISコードを求める。
"漢"を表示するため、前後にKI
/KOコードを付加する。

37. 特定タイプの文字列の抜き出し

```

100 A$="イロハ漢字ABC"
110 PRINT A$
120 KI$=CHR$(&H1B)+CHR$(&H4B):KO$=CHR$(&H1B)+CHR$(&H48)
130 PRINT KEXT$(A$,0)
140 PRINT KI$+KEXT$(A$,1)+KO$

```

1バイト系英数カナ文字のみを抜き出して表示。
2バイト系日本語文字を抜き出し、前後
にKI/KOコードを付加して表示。

38. 2バイト系日本語文字を含む文字列の長さタイプ

```

100 A$="ABC漢字イロハ"
110 FOR I=0 TO 5
120 PRINT KLEN(A$,I):
130 NEXT I
140 PRINT:PRINT
150 FOR J=1 TO KLEN(A$,0)
160 PRINT KTYPE(A$,J):
170 NEXT J

```

各タイプの文字の合計数を、タイプ別に表示。
文字列中の各文字のタイプを順に表示。

39. 利用者定義関数

```

100 DEF FNONSEKI(R)=3.14159*R*R
110 DEF FNENSHUU(R)=3.14159*2*R
120 INPUT "半径は":HANKEI
130 PRINT "面積:":FNONSEKI(HANKEI)
140 PRINT "円周:":FNENSHUU(HANKEI)
150 END

```

円の面積を求める利用者定義関数の定義。
円周を求める利用者定義関数の定義。

40. 電話制御機能(拡張機能)

```

100 CMD LINE OPEN "COM1:" AS #1
110 CMD ON LINE #1 GOSUB *1L
120 CMD LINE #1 ON
130 GOTO 130
140 *1L
150 CMD RECEIVE #1,1
160 WHILE FLN$(#1)<>2:WEND
170 OPEN "COM1:E71N" AS #1
180 CMD ON ERROR #1 GOSUB *1E
190 CMD ERROR #1 ON
200 FLN$="":A$=""
210 FOR I=1 TO 3000:NEXT
220 PRINT #1,CHR$(&H15):
230 IF LOC(1)=0 THEN 230
240 A$=INPUT$(1,1)
250 IF A$=CHR$(&HD) THEN 230
260 IF A$=CHR$(&HA) THEN 290
270 FLN$=FLN$+A$
280 GOTO 230
290 IF FLN$="" THEN 360
300 OPEN FLN$ FOR OUTPUT AS #2
310 IF LOC(1)=0 THEN 310
320 A$=INPUT$(1,1)
330 IF A$=CHR$(&H4) THEN 360
340 PRINT #2,A$:
350 GOTO 310

```

電話機をコミュニケーションポート1に論理的に接続
着信割り込みルーチンの設定、
着信割り込みの許可、
着信を待機する、
着信時、データ通信モードに入る、
完全にデータ通信モードになるまで待つ、
プロトコル設定、
通信エラー割り込みルーチンの設定、
受信した最初の1行をFLN\$に代入、
受信ファイルをFLN\$に代入された名前でオープン、
キャラクタコード&H4を受信、通信を終了、
受信データをファイルに書き込む、

```

360 CLOSE
370 CMD MODE CUT #1 _____ 電話を切る。
380 CMD RECEIVE #1,0 _____ 自動着信を禁止。
390 RETURN
400 *IE _____ 通信エラー割り込みルーチン。
410 PRINT "通信エラー発生":BEEP
420 DMY$=INPUT$(LOC(1),1) _____ 通信エラー情報のダミー読み取り。
430 RETURN 360 _____ エラー状態はリセットされる。

```

41. サウンド機能(拡張機能)

```

100 CLEAR ,&H5E00
110 GOSUB *BEGIN :GOSUB *MPLAY
120 ON PLAY (3,0) GOSUB *MPLAY — チャンネル番号3のサウンドバッファ内の未演奏音楽情報が0以下
130 PLAY ON _____ になったときに*MPLAYに飛ぶように宣言。
140 X=RND*639:Y=RND*399:C=(RND*100 MOD 7)+1 _____
150 CIRCLE (X,Y),RND*20,C _____ 140~170行の間でPLAY割り込み
160 PAINT (X,Y),(RND*100 MOD 7)+1,C _____ を待っている。
170 GOTO 140 _____
180 REM
190 *MPLAY _____ 演奏ルーチン。
200 PLAY STOP:CLS 3
210 PLAY A$,B$,B$ _____
220 PLAY B$,A$,B$ _____ 演奏。
230 PLAY B$,B$,A$ _____
240 PLAY ON
250 RETURN
260 *BEGIN _____ 初期化ルーチン。
270 CLS 3
280 CONSOLE 0,25,0,1
290 SCREEN 3,0 _____ サウンドバッファの確保及び初期化。チャンネル1~3のサウン
300 PLAY ALLOC 255,255,255 _____ ドバッファには255バイトが割り当てられ、他のサウンドバッフ
310 INI$="MB@55@V100" _____ アは0バイトとなる。
320 PLAY INI$,INI$,INI$ _____ チャンネル1~3の初期化設定を行う。
330 A1$="T1604L8C4CC4DE4EE4ED4CD4EC4C03G4RO4E4." _____
340 A2$="EEFG4GG4GF4EF4GE4.R4GE2RGE2RGER8GER8GE4.R4" _____ 演奏データ。
350 A$=A1$+A2$
360 B$="R2R2R2R2R2R2"
370 RETURN

```


付 録

A. エラーメッセージとその対策

BASICの実行中にエラーが発生すると、メッセージが表示されます。ここでは、こうしたエラーメッセージと、その原因および対策について説明します。

メッセージの右側の数字(69など)は、“エラーコード”と呼ばれるもので、エラー発生時、ERR(関数)に格納されます。

“原因:”には、そのエラーが発生した際の原因を簡略に説明してあります。複数の原因が考えられる場合は、列挙してあります。

“対策:”には、そのエラーの原因をとり除く方法か、またはエラーの発生を未然に防ぐための対策が説明してあります。

エラーについての詳細は、第3章中の ERL/ERR, ERROR, ON ERROR GOTO, RESUME などの項をご覧ください。

Bad allocation table 69

原因: ディスク内部の FAT が壊れている。

対策: データを読み取ることができないので、ディスクを再フォーマットする。ただし、以前のファイルはすべて失われる。

または他のOSでフォーマットされたディスクであるので、BASICのディスクに換える。

Bad drive number 70

原因: LOAD, SAVE, KILL, OPEN などのファイル操作命令でドライブの指定がまちがっている。たとえばシステムにつながっていないドライブを指定したなど。

対策: 正しいドライブを指定する。

Bad file name 56

原因: LOAD, SAVE, KILL, OPEN などのファイル操作命令で、ファイル名の指定がまちがっている。

対策: 正しいファイル名を指定する。

Bad file number 52

原因: 同時オープンファイル数を超えるファイル番号を指定しようとした。

対策: ファイル番号を小さくするか、同時オープンファイル数を増やす。

Bad telephone number 173

原因: 電話機番号として指定できない文字を指定している。

対策: 正しい電話機番号を指定する。

Bad telephone set number	172
原因 : 1, 2, 3 以外の電話機番号を指定している。	
対策 : 正しい電話機番号を指定する。	
Bad track / sector	71
原因 : DSKO\$, DSKI\$ で、サーフェス番号、トラック番号、セクタ番号の指定が誤っている。	
対策 : 正しいサーフェス番号、トラック番号、セクタ番号を指定する。	
Can't continue	17
原因 : STOP 命令や CTRL+C など で実行を停止した場合、プログラムを書き換えたりすると、CONT によって実行を再開することができない。	
対策 : CONT を使用したい場合は、実行の中断中にプログラムを書き換えてはならない。実行中断のタイミングによっては、プログラムを書き換えたりしなくとも実行の再開ができなくなることもあるので注意。	
Can't execute across mode	176
原因 : 現在のモードでは実行できない命令 / 関数を実行しようとした。	
対策 : 各モードで実行可能な命令 / 関数を実行するようにプログラムの流れを正す。	
Communication I/O error	174
原因 : 電話機の故障、電話機の初期設定不良、接続不良などにより電話機に対するコマンドレベルの入出力ができない。	
対策 : 原因となる箇所を確認してみる。	
Direct statement in file	57
原因 : LOAD でアスキー形式のプログラムファイルをロードするときに、ダイレクトステートメント (行番号のない行) が存在した。	
対策 : プログラムファイルの行番号が何かの理由で破壊されているか、あるいは、指定ファイルがプログラムファイルではないデータファイルであると考えられるので、ファイルの内容を確認する。	
Disk full	68
原因 : SAVE, PRINT #, PUT などのディスクに書き出す命令を実行したとき、ディスク上に空き領域がなかった。	
対策 : 不要なファイルを削除して、空き領域を確保する。 別のドライブ上のディスクを使用する。 同じドライブ上でディスクを差し替えて使用する。ただしこの方法は、ファイルがオープン状態のときには使えないので注意。	

Disk I/O error	64
原因: ディスクとの入出力中にエラーが発生した。	
対策: 入出力を行おうとしたディスクがフォーマットされていないか、または物理的に破壊されていると考えられるので、ディスクの再フォーマットを行ってみる。 なお、1MBタイプのコピーディスクのサーフェス0、トラック0を直接読み書きすることはできないので注意。	
Disk offline	62
原因: LOAD, SAVE, KILL, OPENなどのファイル操作命令で、ディスクのセットされていないドライブに対して入出力を行おうとした。	
対策: ディスクを正しくドライブにセットする。	
Division by Zero (/0)	11
原因: 0による除算、すなわち $n/0$ (n は任意の数) や 0^{-1} が実行されて、その結果得られた値がオーバーフローを起こした。	
対策: 除数が0にならないようにする。	
Duplicate Difinition	10
原因: 一度宣言した配列を二重定義しようとした。 DIMで宣言しないで配列を使った場合に、その配列を再定義しようとした。	
対策: 別の配列名を使うか、あるいはERASEで古い配列を消去した上で再び同じ配列名で定義しなおす。	
Duplicate label	50
原因: プログラム中に同じラベル名が2つ以上存在している。	
対策: ラベル名を変更して同一名のラベル名が存在しないようにする。	
Feature not available	31
原因: 現在の状態では使用できない命令を実行しようとした。	
対策: その命令を実行するために必要なハードウェアなどを装備あるいは準備する。	
FIELD overflow	33
原因: FIELDで、ランダムファイルのレコード長として合計256バイト以上の領域を指定した。	
対策: フィールドの合計長は255バイト以内にする。	
File already exist	65
原因: NAMEによって新たに付けようとしたファイル名がすでに存在している。	
対策: 新しく付けるファイル名を変更するか、すでに存在しているファイルのファイル名を変更する。	

File already open	54
原因：すでにオープンされているファイルに対して、OPEN, KILL, NAME などを行った。	
対策：CLOSE でいったんファイルを閉じる。	
File not found	53
原因：LOAD, SAVE, KILL, NAME などのファイル操作命令で、指定したファイルが見つからない。	
対策：正しいドライブ名およびファイル名を指定する。	
File not open	60
原因：PRINT #, INPUT #など、ファイル番号を指定して入出力を行う命令で、オープンされていないファイルを参照しようとした。	
対策：入出力命令を実行する前に、OPEN でファイルをオープンしておく。	
File write protected	61
原因：次のうちいずれかの書き込み禁止属性が付けられているファイルに、書き込みを行おうとした。	
・SET によって、ファイルディスクリプタ、ドライブのディスク、ファイル番号に付けられたもの。	
・フロッピディスクに張られたシールによるもの。	
対策：書き込み禁止を解除する。	
FOR without NEXT	26
原因：FOR~NEXT が正しく対応していない。	
対策：プログラムの流れをたどり、FOR~NEXT を正しく対応させる。	
Illegal direct	12
原因：ダイレクトモードで使用できない命令をダイレクトモードで使った。	
対策：プログラムモード上で実行する。	
Illegal function call	5
原因：命令や関数の使い方がまちがっている。すなわち、引数とその関数の許容する範囲を超えたり、結果がその関数のとり得る範囲を超えたりしている。たとえば、次のような場合である。	
・LOG 関数で負や0の引数を指定した。	
・SQR 関数で負の引数を指定した。	
・MID\$, LEFT\$, RIGHT\$, STRING\$, SPACE\$, INSTR, ON...GOSUB などにおいて、不適当な引数が用いられた。	

・ほか。

対策：原因がさまざま考えられるので、各命令、関数の使い方をリファレンスで確認する。

Illegal operation 74

原因：スクロールウィンドウ内に表示データがいっぱい詰まっている場合、またはスクロールウィンドウ内でキー入力したデータがいっぱい詰まっている場合に、スクロール開始行に対してインサートモードでキー入力した。

対策：インサートモードを抜けてから、キー入力を行う。

Input past end 55

原因：INPUT #, GET などのファイルからデータを読み込む命令で、ファイル中のすべてのデータを読み尽くしたあとに、さらに入力命令が実行された。

対策：ファイル中のデータの数と、入力命令で読み込む数とを合わせる。
EOF, LOF などの関数を使って、ファイルの終了を検出するようにする。

Line buffer overflow 23

原因：1行で入力できる文字の範囲（255バイト）を超えて入力が行われた。

対策：1行の文字数を255バイト以内に作る。

Line busy 175

原因：電話の呼び出し相手が話し中である。

対策：電話をかけ直す。プログラムではエラー処理を行って、このエラーが出たらしばらく時間待ちし、改めてかけ直すというアルゴリズムに使うことができる。

Missing operand 22

原因：命令中で、必要なパラメータが指定されていない。

対策：書式を確かめ、必要なパラメータを指定する。

NEXT without FOR 1

原因：FOR～NEXT が正しく対応していない。

対策：プログラムの流れをたどり、FOR～NEXT を正しく対応させる。

No RESUME 19

原因：エラー処理ルーチンの終わりに RESUME がなく、プログラムの実行が継続できない。

対策：RESUME, END, ON ERROR GOTO 0 のどれかで終わらせるようにする。

Out of DATA 4

原因：DATA 行中のデータが足りず、READ で読むべきデータがない。

対策：DATA 行中のデータの数と、READ で読み込む数とを合わせる。READ と DATA の対応と、RESTORE が正しく使われているかどうかを確認する。

Out of memory

7

原因：次のような理由により、メモリ容量が足りなくなった。

- ・プログラムが長すぎる。
- ・配列が大きすぎる。
- ・FOR～NEXT, GOSUBなどでネスティング(入れ子構造)が深くなりすぎて、スタックがいっぱいになった。
- ・ほか。

対策：プログラム、配列を小さくする。ネスティングを浅くする。

FRE を使ってフリーなメモリを調べてみて、メモリの空き容量が異常に少ない場合には、そのプログラムを実行する前に別のプログラム中で、BASIC のプログラム領域を小さく設定したままにしている可能性がある。このようなときには、CLEAR を実行して、BASIC の使うメモリを大きくする。また、起動時にファイルバッファの容量を多く取っていることもあるので、再起動して同時オープンファイル数を必要最小限にする。

Out of octave

134

原因：発生できる音階を超えて音を出そうとした。たとえば”O9C+”や”O1C-”を指定した場合など。

対策：オクターブを設定する命令を正しく使う。

Out of string space

14

原因：文字列の全体量が、使用できるメモリ容量を超えてしまった。

対策：文字列、配列文字列を小さくする。

Overflow (OV)

6

原因：演算結果や入力された数値が、許される範囲を超えた。

対策：データの型と、とる値の範囲が正しいかどうかを確認する。

Rename across disks

73

原因：異なるドライブ上にあるファイルに対してNAMEを実行しようとした。

対策：NAMEでは同じドライブ上のファイルを指定する。

Request denied

177

原因：電話機に対してコマンドを送出しても電話機からそのコマンドを否定された。

対策：使用可能な電話機を使用する。電話機を正しく設定する。電話機の電源をいったん切る。

RESUME without error	20
原因：エラー処理ルーチンでないところに、RESUMEがある。	
対策：ON ERROR GOTOで指定されたところ以外、たとえばメインルーチン内にRESUMEがあるとこのエラーが発生するので、メインルーチンの終わりには必ずENDを置くようにする。プログラムの流れを確認すること。	
RETURN without GOSUB	3
原因：GOSUB～RETURNが正しく対応していない。	
対策：GOSUBで呼び出されたサブルーチン以外、たとえばメインルーチン内にRETURNがあるとこのエラーが発生するので、プログラムの流れに沿って、GOSUBとRETURNの対応を確認する。	
RS-232C (2nd/3rd) board not ready	82
原因：RS-232C 拡張ボードが実装されていないのに、電話機と標準以外のRS-232C回線をCMD LINE OPENで接続しようとした。	
対策：RS-232C 拡張ボードを増設するか、標準のRS-232C回線を使用する。	
Sequential I/O only	59
原因：シーケンシャル入出力以外の入出力命令を使用した。	
MERGEでバイナリファイルを指定した。	
対策：シーケンシャル入出力命令を使用するようにする。	
MERGEでディスクから読み込むファイルは、アスキー形式でセーブしておく。	
Sound buffer not allcated	133
原因：サウンドバッファの確保をしないで、サウンド制御命令を使用した。	
対策：PLAY ALLOCで十分なサウンドバッファを確保する。	
Sound buffer overflow	132
原因：サウンドバッファ領域が足りない。	
対策：PLAY ALLOCで十分なサウンドバッファを確保する。	
String formula too complex	16
原因：文字式が複雑すぎる。	
対策：複雑な文字式は、いくつかの式に分ける。	
String too long	15
原因：1つの文字変数内の文字数が255バイトを超えている。	
対策：2つの変数に分けて処理する。	

Subscript out of range	9
原因：配列変数の添字の値が DIM および OPTION BASE で指定されたときの範囲を外れている。	
DIM による宣言を省略した場合に、添字の値が 10 を超えている。	
対策：配列変数の定義による添字の範囲と引用する配列変数の範囲を確認する。	
Syntax error	2
原因：命令が書式どおりになっていない。	
予約語以外の命令がある。	
READ と DATA との対応で、読み込む変数とデータとの間で型の不一致が生じた。	
対策：各命令を書式どおり正しく記述する。	
READ と DATA では、読み込む変数とデータの型を合わせる。	
Telephone set already open	171
原因：すでに BASIC と接続している電話機に対して CMD LINE OPEN を実行した。	
対策：CMD LINE CLOSE でいったん電話機を切り離す。	
Telephone set not open	170
原因：電話機と BASIC が論理的に接続されていない。	
対策：初期モード以外で実行される電話制御命令を実行する前には、CMD LINE OPEN を実行しておく。	
Type mismatch	13
原因：式の左辺、右辺、関数の引数などにおいて、変数の型が一致していない。	
対策：関数に与えるデータの型を確認する。	
Undefined label	32
原因：定義されていないラベル名を参照した。	
対策：ラベル名を正しく定義する。	
Undefined line number	8
原因：GOTO, GOSUB, IF...THEN~ELSEなどで、存在しない行番号を飛び先に指定した。	
対策：行番号を確認する。	

Undefined user function 18

原因：定義されていない利用者定義関数，または機械語関数を引用した。

対策：利用者定義関数は，引用する前に DEF FN で定義する。機械語関数は，DEFUSR で実行開始アドレスを定義しておく。

Unprintable error 21, 24, 25, 28, 他

原因：メッセージの定義されていないエラー。

対策：ERROR <整数表記>を実行してこのエラーが生ずる場合，その<整数表記>番のエラーを利用者独自のエラー処理に利用することができる。

WEND without WHILE 30

原因：WHILE～WENDが正しく対応していない。

対策：プログラムの流れをたどり，WHILE～WENDを正しく対応させる。

WHILE without WEND 29

原因：WHILE～WENDが正しく対応していない。

対策：プログラムの流れをたどり，WHILE～WENDを正しく対応させる。

システム予約 51

B. コントロールコード表

キャラクタ コード	対応する キー	エディタ中の機能	プログラム中の機能
1	CTRL+A	HELP キーと同じ	—
2	CTRL+B	カーソルを1つ前のワードへ移動	—
3	CTRL+C	STOP キーと同じ	—
4	CTRL+D	1ワード削除	—
5	CTRL+E	カーソル位置から行末までを消去	—
6	CTRL+F	カーソルを1つ先のワードへ移動	—
7	CTRL+G	スピーカを鳴らす	スピーカを鳴らす
8	CTRL+H	BS キーと同じ	バックスペース
9	CTRL+I	TAB キーと同じ	水平タブ
10	CTRL+J	行の連結(インサートモードでは行の分割)	ラインフィード
11	CTRL+K	カーソルをホームポジションへ移動	ホームポジション
12	CTRL+L	テキスト画面のクリア	テキスト画面のクリア
13	CTRL+M	リターンキーと同じ	キャリッジリターン
15	CTRL+O	画面への表示の無効/有効の切り換え	—
18	CTRL+R	インサートモードに入る/出る	—
19	CTRL+S	表示(スクロール)の一時停止	—
21	CTRL+U	1行消去	—
24	CTRL+X	カーソルを1行の最後へ移動	—
28	→	カーソルを右へ移動	カーソルを右へ移動
29	←	カーソルを左へ移動	カーソルを左へ移動
30	↑	カーソルを上へ移動	カーソルを上へ移動
31	↓	カーソルを下へ移動	カーソルを下へ移動

備考 プログラム中の機能は、対応するコードが画面に出力される際(プログラムの実行中)に働きます。

C. キャラクタコード表

上位4ビット→

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
下位4ビット↓	0	^{D_E}	0	@	P	'	p				ー	タ	ミ			×	
	1	^{S_H} _{D₁}	!	1	A	Q	a	q			。	ア	チ	ム			円
	2	^{S_X} _{D₂}	"	2	B	R	b	r			「	イ	ツ	メ			年
	3	^{E_X} _{D₃}	#	3	C	S	c	s			」	ウ	テ	モ			月
	4	^{E_T} _{D₄}	\$	4	D	T	d	t			、	エ	ト	ヤ			日
	5	^{E_Q} _{N_K}	%	5	E	U	e	u			・	オ	ナ	ユ			時
	6	^{A_K} _{S_N}	&	6	F	V	f	v				ヲ	カ	ニ	ヨ		分
	7	^{B_L} _{E_B}	'	7	G	W	g	w				ア	キ	ヌ	ラ		秒
	8	^{B_S} _{C_N}	(8	H	X	h	x				イ	ク	ネ	リ		♠
	9	^{H_T} _{E_M})	9	I	Y	i	y				ウ	ケ	ノ	ル		♥
	A	^{L_F} _{S_B}	*	:	J	Z	j	z				エ	コ	ハ	レ		♦
	B	^{H_M} _{E_C}	+	;	K	[k	{				オ	サ	ヒ	ロ		♣
	C	^{C_L}	→	,	<	L	¥					ヤ	シ	フ	ワ		●
	D	^{C_R}	←	-	=	M]	m	}			ユ	ス	ヘ	ン		○
	E	^{S_O}	↑	.	>	N	^	n	~			ヨ	セ	ホ	"		◁
	F	^{S_I}	↓	/	?	O	_	o				ツ	ソ	マ	°		▷

D. 誘導関数

関数として用意していない三角関数のうち、いくつかは、用意された関数を使って作り出すことができます。以下の数式を参考にしてください(誤差の範囲に注意が必要です)。

目的とする関数	組み込み関数からの誘導式
セカント	$\text{SEC}(X) = 1/\text{COS}(X)$
コセカント	$\text{CSC}(X) = 1/\text{SIN}(X)$
コタンジェント	$\text{COT}(X) = 1/\text{TAN}(X)$
アークサイン	$\text{ARCSIN}(X) = \text{ATN}(X/\text{SQR}(-X * X + 1))$
アークコサイン	$\text{ARCCOS}(X) = -\text{ATN}(X/\text{SQR}(-X * X + 1)) + 1.5708$
アークセカント	$\text{ARCSEC}(X) = \text{ATN}(\text{SQR}(X * X - 1)) + (\text{SGN}(X) - 1) * 1.5708$
アークコセカント	$\text{ARCCSC}(X) = \text{ATN}(1/\text{SQR}(X * X - 1)) + (\text{SGN}(X) - 1) * 1.5708$
アークコタンジェント	$\text{ARCCOT}(X) = -\text{ATN}(X) + 1.5708$
ハイパーボリック・サイン	$\text{SINH}(X) = (\text{EXP}(X) - \text{EXP}(-X))/2$
ハイパーボリック・コサイン	$\text{COSH}(X) = (\text{EXP}(X) + \text{EXP}(-X))/2$
ハイパーボリック・タンジェント	$\text{TANH}(X) = -\text{EXP}(-X)/(\text{EXP}(X) + \text{EXP}(-X)) * 2 + 1$
ハイパーボリック・セカント	$\text{SECH}(X) = 2/(\text{EXP}(X) + \text{EXP}(-X))$
ハイパーボリック・コセカント	$\text{CSCH}(X) = 2/(\text{EXP}(X) - \text{EXP}(-X))$
ハイパーボリック・コタンジェント	$\text{COTH}(X) = \text{EXP}(-X)/(\text{EXP}(X) - \text{EXP}(-X)) * 2 + 1$
ハイパーボリック・アークサイン	$\text{ARCSINH}(X) = \text{LOG}(X + \text{SQR}(X * X + 1))$
ハイパーボリック・アークコサイン	$\text{ARCCOSH}(X) = \text{LOG}(X + \text{SQR}(X * X - 1))$
ハイパーボリック・アークタンジェント	$\text{ARCTANH}(X) = \text{LOG}((1 + X)/(1 - X))/2$
ハイパーボリック・アークセカント	$\text{ARCSECH}(X) = \text{LOG}((\text{SQR}(-X * X + 1) + 1)/X)$
ハイパーボリック・アークコセカント	$\text{ARCCSCH}(X) = \text{LOG}((\text{SGN}(X) * \text{SQR}(X * X + 1) + 1)/X)$
ハイパーボリック・アークコタンジェント	$\text{ARCCOTH}(X) = \text{LOG}((X + 1)/(X - 1))/2$

E. 予約語一覧

これらの予約語を、変数名やラベル名として使用することはできません。ただし、予約語を含む文字列（例：ABSN）は、使用してもかまいません。

ABS	CONT	EOF	INPUT
AKCNV\$	COPY	EQV	INPUT\$
ALLOC	COS	ERASE	INSTR
AND	CSNG	ERL	INT
ASC	CSRLIN	ERR	IRESET
ATN	CVD	ERROR	ISSET
ATTR\$	CVI	EXP	JIS\$
AUTO	CVS	FIELD	KACNV\$
BEEP	DATA	FILES	KANJI
BLOAD	DATE\$	FIX	KEXT\$
BSAVE	DEF	FN	KEY
CALL	DEFDBL	FOR	KILL
CDBL	DEFINT	FRE	KINPUT
CHILD	DEFSNG	GET	KINSTR
CHR\$	DEFSTR	GOSUB	KLEN
CINT	DELETE	GOTO	KMID\$
CIRCLE	DELIM	GO TO	KNJ\$
CLEAR	DIM	HELP	KPLOAD
CLOSE	DRAW	HEX\$	KTYPE
CLS	DSKF	IEEE	LEFT\$
CMD	DSKI\$	IF	LEN
COLOR	DSKO\$	IFC	LET
COM	EDIT	IMP	LFILES
COMMON	ELSE	INKEY\$	LINE
CONSOLE	END	INP	LIST

LLIST	ON	RESTORE	TAB
LOAD	OPEN	RESUME	TAN
LOC	OPTION	RETURN	THEN
LOCATE	OR	RIGHT\$	TIMES
LOF	OUT	RND	TIMEOUT
LOG	PAINT	ROLL	TO
LPOS	PEEK	RSET	TROFF
LPRINT	PEN	RUN	TRON
LSET	PLAY	SAVE	USING
MAIL	POINT	SCREEN	USR
MAP	POKE	SEARCH	VAL
MERGE	POLL	SEG	VARPTR
MID\$	POS	SET	VIEW
MKD\$	PPOLL	SGN	VOICE
MKI\$	PPR	SIN	WAIT
MKS\$	PRESET	SPACE\$	WBYTE
MOD	PRINT	SPC	WEND
MON	PSET	SQR	WHILE
MOTOR	PUT	SRQ	WIDTH
NAME	RANDOMIZE	STATUS	WINDOW
NEXT	RBYTE	STEP	WRITE
NEW	READ	STOP	XOR
NOT	REM	STR\$	
OCT\$	REN	STRING\$	
OFF	RENUM	SWAP	

索引

A

ABS	40
AKCNV\$	40
AND	24
ASC	40
ATN	41
ATTR\$	41
AUTO	42

B

BEEP	42
BLOAD	42
BSAVE	43

C

CALL	44
CDBL	44
CHAIN	45
CHR\$	46
CINT	46
CIRCLE	47
CLEAR	48
CLOSE	49
CLS	49
CMD BREAK	50
CMD CHANGE DUPLEX	51
CMD DIAL	51
CMD ERROR ON/OFF/STOP	53
CMD LINE CLOSE	54
CMD LINE ON/OFF/STOP	54
CMD LINE OPEN	55
CMD MODE CUT	56
CMD ON ERROR GOSUB	56
CMD ON LINE GOSUB	57

CMD RECEIVE	58
CMD STORE DIAL	58
COLOR	59,62
COLOR@	63
COMMON	64
COM ON/OFF/STOP	65
CONSOLE	66
CONT	66
COPY	67
COS	68
CSNG	68
CSRLIN	69
CVD	69
CVI	69
CVS	69

D

DATA	70
DATE\$	70
DEF FN	71
DEF SEG	72
DEF USR	73
DEFDBL	72
DEFINT	72
DEFSNG	72
DEFSTR	72
DELETE	73
DIM	74
DISK モード	3
DRAW	75
DSKF	79
DSKI\$	80
DSKO\$	80

E

EDIT	81
END	82
EOF	82
EQV	24
ERASE	82
ERL	83
ERR	83
ERROR	83
EXP	84

F

FIELD	84
FILES	85
FIX	86
FOR...TO...STEP~NEXT	86
FPOS	87
FRE	87

G

GET	88
GET@	89
GOSUB	90
GOTO	90
GO TO	90

H

HELP ON/OFF/STOP	91
HEX\$	92

I

IF...GOTO~ELSE	92
IF...THEN~ELSE	92
IMP	24
INKEY\$	93
INP	93
INPUT	94
INPUT #	95
INPUT\$	95
INPUT WAIT	96
INSTR	96
INT	97

J

JIS\$	97
-------------	----

K

KACNV\$	97
KEXT\$	98
KEY	98
KEY LIST	99
KEY ON/OFF/STOP	99
KILL	100
KINPUT	100
KINSTR	101
KLEN	101
KMID\$	102
KNJ\$	102
KPLOAD	103
KTYPE	104

L

LEFT\$	104
LEN	105
LET	105
LFILES	85
LINE	106
LINE INPUT	107
LINE INPUT WAIT	108
LINE INPUT #	108
LIST	109
LLIST	109
LOAD	109
LOC	110
LOCATE	110
LOF	111
LOG	111
LPOS	112
LPRINT	112
LPRINT USING	112
LSET	113

M

MAP	113
MERGE	114

MID\$	115
MKD\$	116
MKI\$	116
MKS\$	116
MOD	22
MON	117

N

NAME	117
NEW	117
NEW ON	118
NOT	23

O

OCT\$	118
ON COM GOSUB	119
ON ERROR GOTO	120
ON...GOSUB	121
ON...GOTO	121
ON HELP GOSUB	121
ON KEY GOSUB	122
ON PEN GOSUB	123
ON PLAY GOSUB	124
ON STOP GOSUB	125
ON TIME\$ GOSUB	125
OPEN	126
OPTION BASE	129
OR	24
OUT	129
OV	23,218

P

PAINT	130
PEEK	132
PEN	133
PEN ON/OFF/STOP	134
PLAY	134
PLAY ALLOC	142
PLAY ON/OFF/STOP	143
POINT	144,145
POKE	145
POS	146

PRESET	146
PRINT	147
PRINT #	148
PRINT USING	150
PRINT # USING	152
PSET	152
PUT	153
PUT@	154

R

RANDOMIZE	155
READ	156
REM	156
RENUM	157
RESTORE	157
RESUME	158
RETURN	158
RIGHT\$	159
RND	159
ROLL	160
ROM モード	3
RSET	113
RUN	160

S

SAVE	161
SCREEN	162
SEARCH	165
SET	166
SGN	167
SIN	167
SPACE\$	168
SPC	168
SQR	168
STATUS DIAL	169
STATUS DIAL\$	169
STATUS ERROR	170
STATUS LINE	170
STATUS MODE	171
STATUS PLAY	171
STOP	172
STOP ON/OFF/STOP	172

STR\$	173
STRING\$	173
SWAP	174

T

TAB	174
TAN	175
TERM	175
TIME\$	177
TIME\$ ON/OFF/STOP	178
TROFF	178
TRON	178

U

USR	179
-----------	-----

V

VAL	179
VARPTR	180
VIEW	181,182
VOICE	182
VOICE COPY	184
VOICE INIT	184
VOICE LFO	185
VOICE REG	187

W

WAIT	187
WHILE~WEND	188
WIDTH	188
WIDTH LPRINT	189
WINDOW	190,191
WRITE	191
WRITE #	192

X

XOR	24
-----------	----

あ

エラーメッセージ	32,213
演算子	21
演算の優先順位	27

オーバーフロー	22
---------------	----

か

拡張子	29
型宣言文字	17
型変換	19
関係演算	23
関数	26
キャラクタコード	223
行	11
行番号	11
コントロールコード	222

さ

算術演算	21
式	21
次元	18
指数	21
実数型定数	15
数値型定数	14
整数型定数	14
セーブ	6
添字	17

た

単精度実数型定数	15
ダイレクトモード	4
定数	14
デバイス名	28
デフォルト値	37

は

倍精度実数型定数	16
配列変数	17
パラメータ	38
ファイル	27
ファイルディスクリプタ	28
ファイル番号	29
ファイル名	29
プログラムモード	5
プロンプト	4
文	11

変数	16
変数の型	16
変数名	16

% (パーセント)	17
# (シャープ)	17
* (アスタリスク)	13,21

ま

マルチステートメント	11
文字型定数	14
文字列の演算	26

や

予約語	19,225
誘導関数	224

ら

ラベル名	31
ロード	6
論理演算	23

わ

割り込み	30
------	----

記号

8 進形式	15
10 進形式	14
16 進形式	15
(スペース)	13
, (コンマ)	13
. (ピリオド)	13
: (コロン)	13
; (セミコロン)	13
? (クエッション)	13
! (エクスクラメーション)	17
^ (アップアロー)	21
/ (スラッシュ)	21
' (アポストロフィ)	13
- (マイナス)	13,21
+ (プラス)	21
= (イコール)	23
< (小なり)	23
> (大なり)	23
¥ (エン)	22
\$ (ドル)	17

NEC



PCN25
6992030