SHARP®

SHARP
SOFTWARE
MANUAL

X68030用

X-BASIC ver2.0

ユーザーズリファレンス

X68030

32 bit PERSONAL WORKSTATION

X-BASIC ver2.0 ユーザーズリファレンス

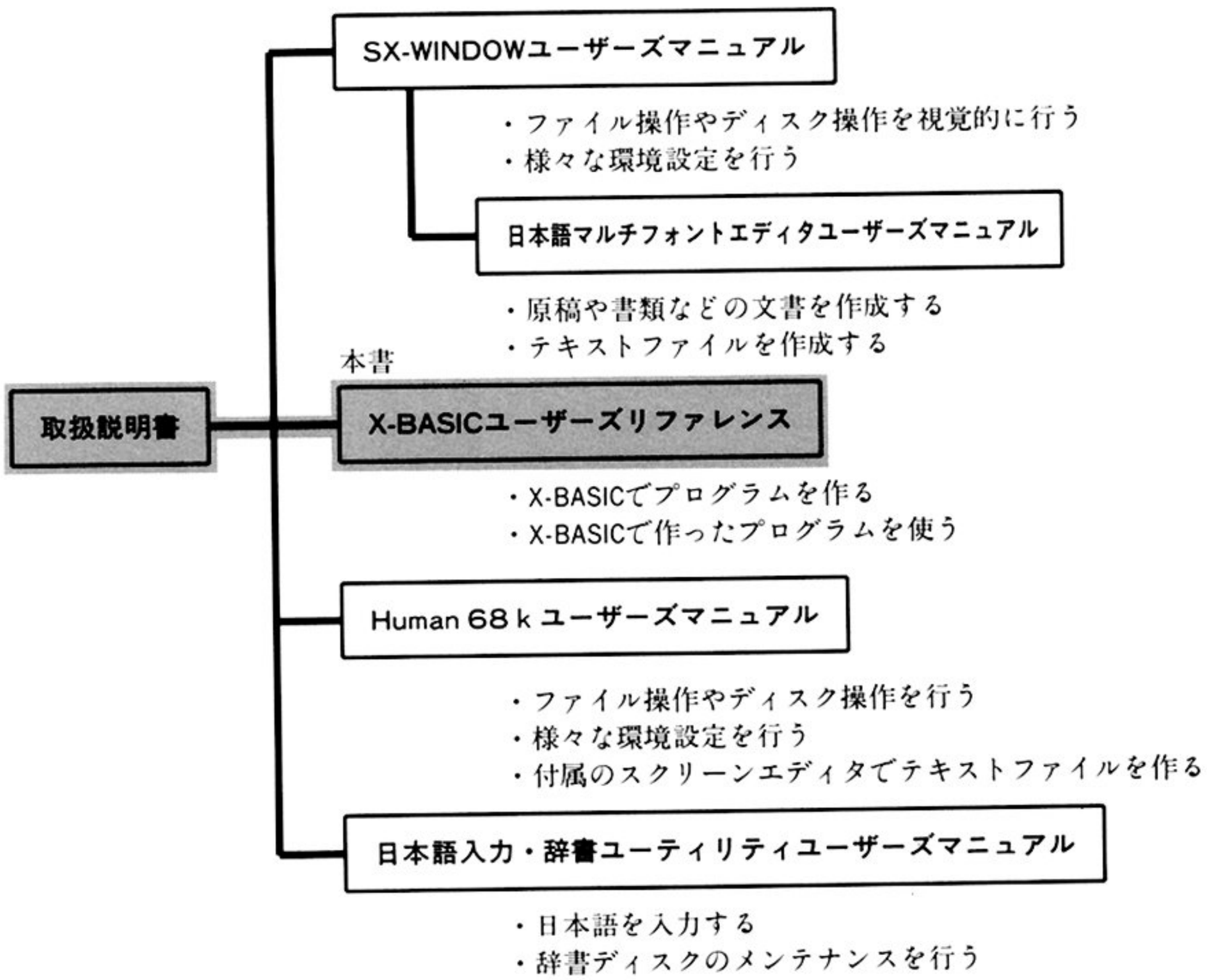
SHARP®

はじめに

本書は、シャープパーソナルワークステーション「X68030シリーズ」上で動作する「X-BASIC」の説明書です。X-BASICの機能、およびその使い方が詳細に解説されています。

なお、本機の取扱いなどについては、別冊の「取扱説明書」や「SX-WINDOWユーザーズマニュアル」にわかりやすく説明されています。初めて本機をお使いになる方は、必ず「取扱説明書」を先にご覧ください。

下図は本機に同梱の6冊のマニュアルの互いの関係を示しています。



マニュアルの使い方

このマニュアルは、本機で使用できるプログラミング言語、X-BASICの説明書で、その概念や特長、文法の解説、各コマンドや関数のリファレンス、及び資料編から構成されています。特に基本的な文法や、X-BASICに特有の機能などについては第1章から第3章をご覧ください。第4章は、実際のプログラミング時に参照する辞書としてご利用いただけるよう、X-BASICのすべてのキーワードについてアルファベット順に説明してあります。また、どのようなコマンドや関数があるかを調べる場合のために、機能別索引も用意されています。

目　　次

第１章　基礎編 3

1．1　X-BASICへようこそ ……4
　1．1．1　デスクトップ画面からの起動 …… 4
　1．1．2　終了 ……4
1．2　プログラムなんて恐くない ……4
1．3　X-BASICのプログラムスタイル ……13
1．4　準備作業　その１ ……14
1．5　制御の流れ ……15
　1．5．1　条件による処理の分岐 ……15
　1．5．2　処理の繰り返し ……16
　1．5．3　流れを変える ……18
1．6　準備作業　その２ ……20
1．7　汎用的な部品を作る ……23
1．8　プログラムの保存と再生 ……24
　1．8．1　プログラムの保存 ……24
　1．8．2　プログラムの再生 ……25
　1．8．3　保存したファイルの確認……26

コラム　X-BASIC のいろいろな起動 27

1．コマンドモードからの起動 ……27
2．プログラムの自動実行 ……27
3．起動時のスイッチ ……28
4．オリジナル X-BASIC……29
5．拡張外部関数 ……31
6．BASTOC ……31
7．Human68k の環境設定 ……33

第２章　応用編 34

2．1　画面制御 ……34
　2．1．1　グラフィック ……35
　2．1．2　スプライト ……38
　2．1．3　パレットとカラーコード……44

2.2　FM 音源制御 ……46
2.3　音声サンプリング（ADPCM） ……49
2.4　ファイル操作 ……51
2.5　その他の機能 ……54
2.5.1　マウス、ジョイスティック制御 ……54
2.5.2　文字列操作 ……55
2.5.3　子プロセス ……56

第3章　X-BASIC の文法　57

3.1　X-BASIC で使用できる文字と特殊記号 ……57
3.2　X-BASIC の変数や定数などで扱うデータ ……59
3.3　定数 ……60
3.3.1　文字定数 ……60
3.3.2　整数型定数 ……60
3.3.3　実数型（浮動小数型）定数 ……61
3.4　式と演算 ……62
3.4.1　算術演算 ……62
3.4.2　関係演算 ……64
3.4.3　論理演算 ……64
3.4.4　関数演算 ……65
3.4.5　文字列演算 ……66
3.4.6　演算の優先順位 ……67

第4章　リファレンス　68

ABS ……69
APAGE ……70
ASC ……71
ATAN ……71
ATOF ……72
ATOI ……73
AUTO ……74
A_PLAY ……75
A_REC ……76

BEEP ……77
BG_FILL ……78
BG_GET ……79
BG_PUT ……80
BG_SCROLL ……81
BG_SET ……81
BG_STAT ……82
BIN$ ……83
BOX ……84
BREAK ……85

CHAR ……85
CHDIR ……86
CHDRV ……86
CHILD ……87
CHR$ ……87
CIRCLE ……88
CLEAR ……89
CLS ……89
COLOR ……90
COLOR [] ……91
CONSOLE ……92

CONT 93
CONTINUE 94
CONTRAST 95
COS 95
CRT 96
CSRLIN 97

DATE$ 97
DAY$ 98
DELETE 98
DIM 99
DSKF 100

ECVT 101
END 102
ERRNO 102
ERROR ON/OFF 103
EXIT() 103
EXP 104

FCLOSE 104
FCLOSEALL 105
FCVT 105
FDELETE 106
FEOF 106
FGETC 107
FILES 108
FILL 109
FIX 110
FLOAT 110
FOPEN 111
FOR~NEXT 112
FPUTC 113
FREAD 114
FREADS 116
FREE 117
FRENAME 117
FSEEK 118
FUNC~ENDFUNC~RETURN( ) 119
FWRITE 120
FWRITES 121

GCVT 122
GET 123
GOSUB~RETURN 124
GOTO 125

HEX$ 125
HOME 126
HSV 127

IF~THEN~ELSE 128
IMG_COLOR 129
IMG_HOME 130
IMG_HT 131
IMG_LOAD 131
IMG_POS 132
IMG_PUT 133
IMG_SAVE 133
IMG_SCRN 134
IMG_SET 135
IMG_STILL 136
INKEY$ 136
INPUT 137
INSTR 138
INT 138
INT 139
ISALNUM 139
ISALPHA 140
ISASCII 140
ISCNTRL 141
ISDIGIT 141
ISGRAPH 142
ISLOWER 142
ISPRINT 143
ISPUNCT 143
ISSPACE 144
ISUPPER 144
ISXDIGIT 145
ITOA 146

KEY ····· 146
KEY LIST ····· 147
KILL ····· 147

LEFT$ ····· 148
LFILES ····· 108
LINE ····· 149
LINPUT ····· 150
LIST ····· 151
LLIST ····· 151
LOAD ····· 152
LOCATE ····· 152
LOG ····· 153
LPRINT ····· 203
LSEARCH ····· 213

MID$ ····· 154
MIRROR$ ····· 154
MOUSE ····· 155
MSAREA ····· 156
MSBTN ····· 157
MSPOS ····· 158
MSSTAT ····· 159
M_ALLOC ····· 160
M_ANTOFF ····· 160
M_ASSIGN ····· 161
M_BEND ····· 161
M_CHAN ····· 162
M_CONT ····· 162
M_CTRLRES ····· 163
M_DIROUT ····· 163
M_END ····· 164
M_ERRGET ····· 164
M_FREE ····· 165
M_FREEA ····· 165
M_IFCHK ····· 165
M_INIT ····· 166
M_MDREG ····· 166
M_MEAS ····· 167
M_METER ····· 167
M_MOD ····· 168
M_MODSNS ····· 168
M_MSTVOL ····· 169
M_MUTE ····· 169
M_NTOFF ····· 170
M_NTON ····· 170
M_OPMEXC ····· 171
M_OPMLFQ ····· 171
M_OPMREG ····· 172
M_OUT ····· 172
M_PAN ····· 173
M_PANFLT ····· 174
M_PCMBSY ····· 174
M_PCMCLR ····· 175
M_PCMGET ····· 175
M_PCMLEN ····· 176
M_PCMON ····· 177
M_PCMREC ····· 178
M_PCMSET ····· 179
M_PGMFLT ····· 180
M_PLAY ····· 180
M_PNMGET ····· 181
M_PNMSET ····· 182
M_PROG ····· 182
M_SNDSET ····· 183
M_SOLO ····· 183
M_START ····· 184
M_STAT ····· 185
M_STOP ····· 186
M_SYNC ····· 187
M_SYSCH ····· 187
M_TEMPO ····· 188
M_TNMGET ····· 189
M_TNMSET ····· 189
M_TRK ····· 190
M_TRNS ····· 190
M_USE ····· 191
M_VEL ····· 191
M_VGET ····· 192
M_VOL ····· 193

M_VOLFLT ····· 193
M_VSET ····· 194
M_YCOM ····· 195

NAME ····· 196
NEW ····· 196

OCT$ ····· 197

PAINT ····· 198
PALET ····· 199
PI ····· 200
POINT ····· 201
POS ····· 202
POW ····· 202
PRINT ····· 203
PSET ····· 205
PUT ····· 206

RAND ····· 207
RANDOMIZE ····· 207
REM ····· 208
RENUM ····· 208
REPEAT~UNTIL ····· 209
RGB ····· 209
RIGHT$ ····· 210
RND ····· 210
RUN ····· 211

SAVE ····· 211
SCREEN ····· 212
SEARCH ····· 213
SETMSPOS ····· 214
SGN ····· 214
SIN ····· 215
SPACE$ ····· 215
SP_CLR ····· 216
SP_COLOR ····· 216
SP_DEF ····· 217
SP_DISP ····· 219
SP_INIT ····· 220
SP_MOVE ····· 220
SP_OFF ····· 221
SP_ON ····· 221
SP_PAT ····· 222
SP_SET ····· 223
SP_STAT ····· 224
SQR ····· 225
SRAND ····· 225
STICK ····· 226
STOP ····· 227
STR ····· 227
STR$ ····· 228
STRCHR ····· 228
STRCSPN ····· 229
STRIG ····· 230
STRING$ ····· 231
STRLEN (LEN) ····· 231
STRLWR ····· 232
STRNSET ····· 232
STRRCHR ····· 233
STRREV ····· 233
STRSET ····· 234
STRSPN ····· 234
STRTOK ····· 235
STRUPR ····· 235
SWITCH~CASE~DEFAULT~ENDSWITCH ····· 236
SYMBOL ····· 237
SYSTEM ····· 238

TAN ····· 238
TIME$ ····· 239
TOASCII ····· 239
TOLOWER ····· 240
TOUPPER ····· 240

VAL ····· 241
VPAGE ····· 242
V_CUT ····· 243

WHILE~ENDWHILE……244
WIDTH……244
WINDOW……245
WIPE……246

資料編 **247**

コントロールコード一覧 …… 249
キャラクタコード表 …… 250
予約語一覧 …… 251
ノート番号：音程　対応表 …… 253
エラーコード一覧 …… 254
索引……256
機能別索引 …… 258

X-BASIC ver2.0 ユーザーズリファレンス

第１章　基礎編

・X-BASIC の特長

X-BASIC は、Human68k 上で走るインタプリタ BASIC です。func～endfunc、switch 文、ブロック if 文などを用いて、C 言語のような構造化プログラミングスタイルを実現することができます。また、インタプリタ BASIC の長所である修正（デバッグ）作業のしやすさと、スクリーンエディタによるプログラム編集のしやすさも兼ね備えています。

特に、本機の特長であるグラフィック、スプライト、サウンド等のハードウエア機能は、すべて外部関数としてサポートされており、X-BASIC の本体（核）とは独立した構造をとっています。このため別売の「C　Compiler PRO-68K」を利用することで、独自の外部関数を作成したり、X-BASIC のソースプログラムを C 言語のソースプログラムにコンバートすることができます。

具体的には次のような特長を備えています。

- ●プログラム内で種々の関数定義ができ、ローカル変数の使用により再帰呼び出しが可能
- ●ファイル構造は Human68k と同一で、C 言語のようなファイル処理が可能
- ●データ型として int（4バイト整数）、char（1バイト整数）、float（8バイト実数）、str（文字）の4つをサポート
- ● Human68k 上の日本語 FP (フロントプロセッサ) を利用して、日本語処理 (JIS 第1、第2水準漢字）が可能（日本語入力の方法については「日本語入力・辞書ユーティリティユーザーズマニュアル」を参照してください）
- ●フリーエリアおよび使用する外部関数の指定などが可能
- ●FM音源、MIDI、ADPCMの同時コントロールが可能

・起動する前に

X-BASIC を起動する前に、必ずシステムディスクと辞書ディスクのバックアップコピーを作成してください。作成方法については、「SX-WINDOWユーザーズマニュアル」または「Human68kユーザーズマニュアル」を参照してください。

今後の作業では、このバックアップされたシステムディスクや辞書ディスクを使い、マスターディスク（本機に付属の原本）は、大切に保管しておいてください。

1．1　X-BASICへようこそ

ここでは、X-BASICの起動と終了の方法を説明します。

1．1．1　デスクトップ画面からの起動

X-BASICの起動方法には、大きく分けて次の2つがあります。

・デスクトップ画面からの起動
・Human68kのコマンドモードからの起動（本章のコラム「X-BASICのいろいろな起動」を参照してください）

ここでは、デスクトップ画面からX-BASICを起動する手順を説明します。

まず、デスクトップ画面を起動します。（「SX-WINDOWユーザーズマニュアル」を参照してください）。次に、「**BASIC2**」のディレクトリアイコンをマウスでポイントして、左ボタンでダブルクリックします。

ディレクトリのウィンドウが開いたら、「**BASIC.X**」のアイコンをマウスでポイントして、左ボタンでダブルクリックします。（「**BASIC2**」はHuman68k ver.3.0システムディスクに入っています）

しばらくするとX-BASICが起動して、次の画面が表示されます。

```
X-BASIC for X68000 version *.**
Copyright **** SHARP/Hudson
 *** kbytes free
Ok
```

画面には点滅するカーソルがあります。この状態を、入力待ち状態といい、キーボードからX-BASICの命令やプログラムを入力することができます。入力した文字はカーソルの位置に表示されます。

1．1．2　終了

X-BASICを終了するには、入力待ち状態のときに、次のように入力します。

system ↵　　（↵はリターンキーを押すこと）

画面にメッセージが表示されますので、何かキーを押すか、マウスのボタンをクリックしてください。デスクトップ画面に戻ります。

また、ファンクションキーのF10キーを押しても「system」と表示されますので、表示されたら、リターンキーを押してください。同じようにして、デスクトップ画面に戻ります。

1．2　プログラムなんて恐くない

X-BASICが起動したら、次のようにキーボードから入力してください。

```
print 12-7
```

カーソルは「7」の後ろで点滅しています。では、リターンキーを押してください。画面は次のようになったはずです。

```
print 12-7
 5
Ok
```

いま入力した命令は、「12－7を計算して、その結果を画面に表示しなさい」という意味です。このように、直接X-BASICに命令を与えてすぐに結果を出させることを、ダイレクトモードで実行する、といいます。また、いま入力した「print 12－7」全体を、命令文といいます。

ダイレクトモードで入力、実行した命令は、コンピュータのメモリに記憶されません。「使い捨て」の命令は、ダイレクトモードで実行します（ただし再利用することもできます。これについては後ほど説明します)。

では、次のように入力してみましょう。

```
10 print 12-7
```

入力したらリターンキーを押してください。

```
10 print 12-7
```

今度は計算結果が画面に表示されませんでした。なぜでしょう。この「謎」の答えは、行の先頭の「10」にあります。

X-BASICでは、行の先頭に1から65535までの整数（「行番号」といいます）と1つ以上のスペースがあり、その後ろに命令文が続いているものを、「プログラム」と考えます。プログラムはコンピュータのメモリに記憶されていて、「実行しなさい」という命令を与えるまでは何もしません。

キーボードから、

```
run ↵          (↵はリターンキーを押すこと)
```

と入力します。また、ファンクションキーのF 5キーを押しても同じことができます。

```
10 print 12-7
run
 5
Ok
```

計算結果が表示されました。このように、プログラムをコンピュータのメモリに記憶させて、あとで結果を出させることを、プログラムモードで実行する、といいます。「run」は、プログラムを実行する命令です。

カーソルを「12－7」の「1」に移動して、

```
"X68000" ↵
```

と入力し、さらにカーソルを「run」の行に移動して、いきなりリターンキーを押してみましょう。

```
10 print "X68000"
run ↵
X68000
Ok
```

run 命令を再度入力しなくても、プログラムは実行されました。しかも今度は、計算結果ではなく、「X68000」という文字が表示されています。このように、X-BASIC では、画面に表示されている文字を修正して（あるいはそのまま)、何度でも再利用できます。

ここで注意。画面に表示されているプログラムを修正したり、新しく行番号をつけてプログラムを作成したときは、そのあとで必ずリターンキーを押します。リターンキーを押さずにカーソル移動キーでカーソルを他の行へ移動したときは、作成、修正の結果はコンピュータのメモリに記憶されません。

では次に、「"X68000"」を「"Human68k"」に変更してみましょう。カーソルを「X」に移動してから INS キーを押します。続けて、

```
Human
```

と入力し（リターンキーは押しません)、DEL キーを1回押します。次にカーソルを「8」の右の「0」に合わせて、

```
k
```

と入力してから DEL キーを3回押します。ここまでの変更は、まだコンピュータのメモリに記憶されていません。仕上げにリターンキーを押しましょう。

```
10 print "Human68k"
run
X68000
Ok
```

プログラムを実行します。「run」の行にカーソルを移動してリターンキーを押してください。

```
10 print "Human68k"
run ↵
Human68k
Ok
```

カーソルの位置はそのままで、

```
cls ↵
```

と入力してください。画面表示が消えて、左上に「Ok」だけが表示されたはずです。このように「cls」はテキスト画面を消去する命令です。プログラムも消えてしまったのでしょうか。

では、次のように入力してください。

```
list ↵
```

画面が消えても、メモリに記憶されているプログラムまで消えてしまうわけではありません。list 命令でいつでも表示できます。また、ファンクションキーの[F 4]キーを押しても同じことができます。「list」は、プログラムリストを表示する命令です。

```
Ok
list
    10 print "Human68k"
Ok
```

行番号の左にスペースが3文字分入っていますが、実行には何の影響もありません。list 命令の表示形式が、いつも行番号5桁分をとるようになっているだけのことです。ただし、行番号の右には、必ず1つ以上のスペースをあけなければなりません。

「10」の「1」にカーソルを移動して、

```
2 ↵
```

と入力してください。繰り返し同じ位置にカーソルを移動して、「2」を「3」に、「3」を「4」に書き換えて、そのたびにリターンキーを押します。次に、カーソルを「Ok」の下の空いているところに移動してから、

```
run ↵    (または[F 5]キーを押します)
```

と入力します。そうすると……

```
Ok
list
    40 print "Human68k"
Ok
run
Human68k
Human68k
Human68k
Human68k
Ok
```

プログラムを見てみましょう。list 命令を使います。その場で「list」と入力してリターンキーを押すか、[F 4]キーを押すかまたは「list」と表示されている行にカーソルを移動してリターンキーを押します。

```
Ok
list
    10 print "Human68k"
    20 print "Human68k"
    30 print "Human68k"
    40 print "Human68k"
Ok
```

何の役にもたたないプログラムですが、誤動作するわけではありません。それなりに「完成した」プログラムです。

いままでの操作で、プログラムは簡単に作成できるんだ、という雰囲気はわかっていただけたでしょう。

では、次にもう少しプログラムらしい例を示しましょう。

キーボードから、

new ↵
auto ↵

と入力してください。「10」という数字が表示されて、その右側でカーソルが点滅しているはずです。「new」は現在メモリ上にあるプログラムを消去する命令、「auto」は行番号を自動的に発生させる命令です。

```
new
Ok
auto
Ok
    10
```

次のプログラムを入力してください。行番号は、リターンキーを押すと自動的に表示されますので、入力する必要はありません。また、もし前の行に入力ミスがあったとしても、それはすべて入力したあとで修正できます。

```
 10 /* sample.bas
 20 int x,y,fg
 30 screen 2,0,1,1
 40 vpage(1)
 50 window(0,0,1023,1023)
 60 mouse(0)
 70 mouse(4)
 80 home(0,384,384)
 90 fill(400,400,650,650,9)
100 while fg=0
110       fg=msbtn(1,0,0)
120       mspos(x,y)
130       home(0,x,y)
140 endwhile
150 wipe()
160 end
```

「end」まで入力してリターンキーを押したら、続けて BREAK キーを押します。入力ミスがあるようでしたら、カーソル移動キーと INS キー、DEL キーなどを使って修正してください。修正したら、リターンキーを押すことをお忘れなく。

それでは実行してみましょう。プログラムを実行する命令は「run」でした。

run ↵　（または F 5 キーを押します）

と入力します。グラフィックが表示されたら、マウスを動かしてみましょう（ボタンは押しません）。四角形がマウスの動きに合わせて移動します。左ボタンをクリックすると、プログラムは終了します。うまく実行できたでしょうか。

せっかく入力したプログラムです。消してしまうのはもったいないので、フロッピーディスクに保存します。ドライブ B にフォーマットしたばかりのフロッピーディスクを入れて、次のように入力してください。

save　"b：sample.bas" ↵

プログラムの保存については、あとの節でもう少し詳しく説明します。

次に、その他のプログラムの修正方法について説明します。

(1) 行の追加

行番号は通常、10、20、30というように10ずつ増やした整数をつけていきます。これは、あとから行番号を追加しやすくするためです。プログラムを作成しているときに、行の間に命令文を追加したい、ということは頻繁におこります。

たとえば、20行と30行の間に命令文を追加するときは、画面の何も入力されていない部分にカーソルを移動して、20と30の間の整数（25など）を行番号として命令文を入力します。入力したあとは、必ずリターンキーを押してください。

```
list ↵
     10 print "これは"
     20 print "例です"
Ok
15 print "追加の" ↵
```

```
list ↵
     10 print "これは"
     15 print "追加の"
     20 print "例です"
Ok
```

(2) 行の削除

すでにある行番号だけを入力してリターンキーを押すと、その行は削除されます。

```
list ↵
     10 print "行を"
     20 print "このように"
     30 print "削除します"
Ok
20 ↵
list ↵
     10 print "行を"
     30 print "削除します"
Ok
```

まとめて削除するときには、delete 命令を使います。

```
list ↵
     10 print "複数行を"
     20 print "一度に"
     30 print "まとめて"
     40 print "削除します"
Ok
delete 20-30 ↵
list ↵
     10 print "複数行を"
     40 print "削除します"
Ok
```

(3) 行の置き換え

すでにある行番号と同じものに続けて、新しい命令文を入力してリターンキーを押すと、その行は自動的に新しいものと置き換わります。

```
list ↵
     10 print "内容が"
     20 print "置き換わります"
Ok
```

```
10 print "このように" ↵
list ↵
    10 print "このように"
    20 print "置き換わります"
Ok
```

(4)　行番号のつけ換え

プログラムを作成している途中で、行番号の間隔が不揃いになったり、行番号が連続して間に行を追加できなくなったときには、renum 命令を使います。

```
list ↵
    10 print "行番号を"
    12 print "つけ換えて"
    18 print "きれいにしましょう"
Ok
renum ↵
Ok
list ↵
    10 print "行番号を"
    20 print "つけ換えて"
    30 print "きれいにしましょう"
Ok
```

X-BASIC で扱う予約語（変数名や関数名に使用できないキーワード）は、大きく分けて次の種別があります。

コマンド
ステートメント
システム変数
関数

コマンドは、プログラム中に記述することはできません。ダイレクトモードでのみ使用します。
これまでに説明した命令（コマンド）の詳細は、第 4 章を参照してください。

用語の説明

●文：　文とは命令文のことで、コンピュータに対する命令を与えます。文には、第4章で説明する命令や、代入を行う式などがあります。

●行：　行は、コンピュータに命令を与えるときの単位で、リターンキーを押すまでに入力したものを指します。1行は255文字以内で、1行には1つ以上の文が含まれています。1行に複数の文を含む行をマルチステートメントと呼び、この場合には文と文の間をコロン（：）で区切ります。またX-BASICでは大カッコ｛｝によってdim文中のデータを複数行に分けて記述したり、if文中で実行文を複数行にわたって記述することも可能です。

●行番号：行番号は、行を複数個集めてプログラムにする場合に、行の先頭につける番号です。行番号は1～65535までの整数で、そのプログラムをコンピュータのメモリに記憶する順序を表します。また、プログラムの実行も行番号順（番号の小さい順）に行われます。

●データ：一般にはいろいろな種類の情報のことを指しますが、BASICではユーザーが指定する文字や数値のことをいいます。たとえば、「a=1」というのは、aという変数に1という値を代入するということですが、この場合の1をデータといいます。

1．3　X-BASIC のプログラムスタイル

次のプログラムを見てください。

```
10  /* Tiny EDITOR  ted.bas
20  str fname[20]
30  str buff[200]
40  int fp, fc
50  char CR=13, LF=10
60  /*
70  title()
80  fninput()
90  fp=fopen(fname,"c")
100 edit()
110 fc=fclose(fp)
120 end
130 /*
140 func title()
150 str dummy
160   cls
170   print "これは簡易エディタです。"
180   print "メモなどを作成するときにお使いください。"
190   print "なお、すでに存在するファイル名を指定したときは、無条"
200   print "件にそのファイルの内容を削除してしまいます。"
210   print
220   print "                                    何かキーを押してください  ";
230   dummy=inkey$
240   cls
250 endfunc
260 func fninput()
270   print "Input file name >";
280   linput fname
290 endfunc
300 func edit()
310   print "Input data(END = /)" : print
320   while buff <> "/"
330       linput ">";buff
340       if buff <> "/" then {
350           fwrites(buff,fp)
360           fputc(CR,fp)
370           fputc(LF,fp)
380       }
390   endwhile
400 endfunc
```

いままで BASIC を使ったことがある人でも、ちょっと違うぞ、と感じるのではないでしょうか。このプログラムには、X-BASIC を特長づけるいくつかの要素と、ぜひ実践してほしい事柄（プログラミングの「作法」）が含まれています。

BASIC ではおなじみの goto 文や gosub 文がありません。X-BASIC でも goto 文や gosub 文を使ってプログラミングできますが、使わなくても不便なことはありません。むしろ、プログラムの流れをわかりづらくする可能性のある goto 文、gosub 文は、使わないことをおすすめします。

goto 文に代わるものとして、条件判断による分岐（ブロック if 文、switch 文）や繰り返し（for～next 文、while～endwhile 文、repeat～until 文）を制御する構文があります。また、gosub 文で呼び出すサブルーチンの代わりに、より独立性の高い関数（func～endfunc）を定義することができます。

さらに、これも BASIC では多用するマルチステートメントを使っていません。マルチステートメントについても、goto 文などと同様に X-BASIC で使うことはできます。しかし、プログラムをより見やすいものにするために、１行１命令にすることをおすすめします。

このように、X-BASICはその独自の特長により、わかりやすく、また修正（デバッグ）作業のしやすいプログラムの作成、編集を可能にしています。

この他にも、X-BASICでプログラミングするために、必ず知っておかなければならないことがあります。次節以降で詳しく説明します。

1.4　準備作業　その1

プログラムで使うデータをしまっておく「入れ物」を、変数といいます。また、入れ物につける名前を、変数名といいます。

X-BASICでは、変数を使用するときには、その変数をこれから使うぞ、ということを最初に宣言しなければなりません。これを、変数の宣言といいます。また、変数を宣言するときには、その変数がどんなデータの入れ物になるのか、つまり、変数のデータ型も同時に宣言します。

データ型には次のものがあります。

整数型（int 型）	-2,147,483,648から2,147,483,647までの整数値を扱う
整数型（char 型）	0から255までの整数値を扱う
実数型（float 型）	絶対値が1.1125369292536E−308から3.5953862697246E+308までの倍精度実数値を扱う
文字型（str 型）	255文字以内の文字列を扱う

次のプログラムでは、10行から50行で、それぞれ char 型、int 型、float 型、str 型の変数を宣言し、60行から100行で値を代入しています。

```
 10 char c
 20 int i
 30 float f
 40 str s
 50 str s_max[255]
 60 c=200
 70 i=647685
 80 f=14487895.256
 90 s="X68000"
100 s_max="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
110 print c,i,f,s,s_max
120 end
```

str 型で33文字以上の文字列を扱うときには、50行の宣言のように、文字バッファのサイズを指定します。

データ型の宣言が省略されたときには、その変数は自動的に int 型になります。これは、実数値を扱うプログラムでは注意しなければならない点です。int 型の変数に実数値を代入したときは、計算結果が正しくないものになることがあります。

変数は、宣言と同時に初期値を代入することができます。

```
10 char c=200
```

初期値を代入しないときは、数値（int、char、float）型の変数には0が、文字（str）型の変数には null 文字（文字数0の文字列）が代入されます。

原則として、同じ名前の変数をメインプログラム中で再宣言することはできません。

データ型の詳細については第３章を、また、変数の宣言方法については第４章をそれぞれ参照してください。

用語の説明

●変　数：変数とは、それ自体が値を表すものではなく、必要に応じて値を代入したり、式の一部になったりするものです。

●変数名：変数につける名前のことです。X-BASICでは、変数名は64文字までの英数字の組み合わせです。変数名は、予約語を含んでいてもかまいません（予約語については、資料編「予約語一覧」を参照してください）。

１．５　制御の流れ

X-BASICには、プログラムの流れをわかりやすいものにするために、いくつかの制御構造を持った構文が用意されています。

１．５．１　条件による処理の分岐

ある一定の条件によって、そのあとに実行することを変えることができます。X-BASICでは、if文、switch文を使います。

・if文

「もし冷蔵庫にビールがあれば、飲む。なかったら、茶の間に戻る」

このような、条件によってその後の行動（作業）を分岐させることは、プログラム中でも頻繁に起こります。そのようなときには、if文(if～then～else)を使います。上の例をプログラム的な流れで表すと、

1．台所に行く
2．冷蔵庫を開ける
3．if　ビールがある　then　飲む(終わり)　else　茶の間に戻る(終わり)
4．終わり

のようになります（これはプログラムではありません）。

X-BASICのif文の特長は、分岐後の操作を複数行に渡って記述できるところにあります。たとえば、「変数ａの値が５だったら、ｘ＋ｙの値を変数ａに代入して、"OK"と変数ａの値を表示させ、５でなかったら、"NO"と変数ａの値を表示する」という分岐は、

```
 10 int a,x,y
 20 x=3
 30 y=10
 40 a=0
 50 input "Number = ";a
 60 if a=5 then {
 70       a=x+y
 80       print "OK",a
 90 } else print "NO",a
100 end
```

と記述できます。

・switch 文

「くじを引いて、1等なら VTR、2等ならラジカセ、3等ならタオルセットがもらえる。それ以外はハズレ」

ある条件で、複数の動作に分岐するときには、switch 文(switch～case～default～endswitch)を使います。上の例をプログラム的な流れで表すと、

1．くじを引く
2．switch　くじの中身
3．　case 1等 ：VTR をもらう：break　(終わり)
4．　case 2等 ：ラジカセをもらう：break　(終わり)
5．　case 3等 ：タオルセットをもらう：break　(終わり)
6．　default　：涙をのむ
7．endswitch
8．終わり

のようになります（これはプログラムではありません)。

switch 文では、すべての条件に一致しなかったとき、default 文があれば、それを実行します。

break 文は、switch 文や for 文、while 文、repeat 文から抜け出す働きをします。次のプログラムの、30行から80行と、90行から140行では、動作が違ってきます。

```
 10 int a
 20 a=1
 30 switch a
 40    case 0 : print "0" : break
 50    case 1 : print "1" : break
 60    case 2 : print "2" : break
 70    default : print "?"
 80 endswitch
 90 switch a
100    case 0 : print "0"
110    case 1 : print "1"
120    case 2 : print "2"
130    default : print "?"
140 endswitch
150 end
```

はじめの switch 文では、50行だけを実行しますが、次の switch 文は、110行から130行までを実行してしまいます。

1．5．2　処理の繰り返し

ある条件が満たされるまで、一定の処理を繰り返し実行することができます。X-BASIC では、for 文、while 文、repeat 文を使います。

・for 文

「娘が生まれてから20才になるまで、毎年1つ絵皿を作る」

一定の回数、あることを繰り返すときには、for 文（for～next)を使います。上の例をプログラム

的な流れで表すと、

1．娘が生まれる
2．for 娘の年齢＝0 to 20　（1つずつ増やす．20回で終わり）
3．　　絵皿を作る
4．next　（2．に戻る）
5．終わり

のようになります（これはプログラムではありません)。

X-BASICの for 文は、ループ回数を決める変数を1ずつ増やすことしかできません。

```
10 int i
20 for i=1 to 10
30       print "X68000    ";
40       print "is THE GREAT."
50 next
60 end
```

・while 文、repeat 文

「ハートのエースが出るまで、トランプを引く」

繰り返しを終わらせる条件はわかっていても、何回実行するのか明確でないときには、while 文 (while～endwhile)、repeat 文 (repeat～until) を使います。上の例をプログラム的な流れで表すと、

1．トランプを引く
2．while トランプ<>ハートのエース　(ハートのエースなら終わり)
3．　　トランプを引く
4．endwhile　（2．に戻る）
5．終わり

または、

1．repeat
2．　　トランプを引く
3．until トランプ＝ハートのエース（1．に戻る．ハートのエースなら終わり）
4．終わり

のようになります（これはプログラムではありません)。

この2つの構文は、条件を判断するタイミングが違います。while 文は、処理を実行する前に条件を判断するので、場合によってはループの中身を一度も実行しないこともあります。これに対し、repeat 文は、必ず処理を実行してから条件を判断します。したがって、ループの中身を1回は実行します。

また、条件判断の基準も、while 文と repeat 文ではちょっと違います。while 文は、条件が正しいとき (真のとき、または、偽になるまで、といいます) 処理を繰り返します。repeat 文は、条件が正しくないとき (偽のとき、または、真になるまで、といいます) 処理を繰り返します。

```
10 int i
20 i=0
30 while i<>10 /* 「iは10ではない」が正しいとき繰り返す
40     print "X68000"
50     i=i+1
60 endwhile
70 end

10 int i
20 i=0
30 repeat
40     print "X68000"
50     i=i+1
60 until i=10 /* 「iは10である」が正しくないとき繰り返す
70 end
```

while 文や repeat 文を使って、いつまでも繰り返しを続ける無限ループを作ることができます。無限ループを終わらせるには、ループの中で if 文と break 文を使います。

```
 10 int a
 20 a=1
 30 while a=1 /* 正しい
 40     input "Number = ";i
 50     if i=5 then {
 60         print "Number = 5"
 70         break
 80     }
 90     print "Number = ";i
100     print "Reenter"
110 endwhile
120 end
```

1．5．3　流れを変える

繰り返しを実行するときに、ある条件のときだけループを抜けたい、または、ループの中の処理をそのときだけ省略したい、ということもあるでしょう。goto 文を使えば簡単ですが、それでは行番号に依存したプログラムになってしまいます。goto の飛び先に新しい行を追加したときなどは、ちょっと困ったことになりそうです。

X-BASIC では、break 文と continue 文を使います。

・break 文

「1年間、釣銭の1円玉を毎日ビンに入れる。ただし、途中でビンがいっぱいになったら中止する」

switch 文のところでも説明しましたが、break 文は、switch 文や for 文、while 文、repeat 文から強制的に抜け出す働きをします。上の例をプログラム的な流れで表すと、

1．買物をして釣銭をもらう
2．for 日数=1 to 365
3．　　ビンに1円玉を入れる
4．　　if　ビンがいっぱい　then　break　(終わり)
5．　　買物をして釣銭をもらう
6．next　(2．に戻る)
7．終わり

のようになります（これはプログラムではありません)。

```
 10 int a,i
 20 a=0
 30 b=0
 40 for i=1 to 100
 50     print i;
 60     input " Number = ";b
 70     a=a+b
 80     if a>=45 then {
 90         print "a >= 45"
100         break
110     }
120     print "a < 45"
130     print "More"
140 next
150 end
```

制御構造が多重の入れ子（ネスト）になっているときは、break 文は、1つ上のレベルのネストへ抜け出します。

```
 10 int i,j,k
 20 str ss=">>>>>>>Break<<<<<<<"
 30 for i=0 to 2
 40     for j=0 to 2
 50         for k=0 to 2
 60             if k=1 then break
 70             print "**********************";k
 80         next
 90         print ss
100         print "**************";j
110     next
120     print "*********";i
130 next
140 end
```

・continue 文

「読破するまで、毎晩『三国志』を読む。しかし、酒を飲んだ日は読まずに眠る」

continue 文は、for 文や while 文、repeat 文のループを強制的に次のサイクルへ進めます。上の例をプログラム的な流れで表すと、

1．『三国志』を買ってくる
2．毎晩読む
3．while 読み終えない　(読み終えたら終わり)
4．　　if　酒を飲んだ　then　continue　（3．へ戻る)
5．　　毎晩読む
6．endwhile　（3．へ戻る)
7．終わり

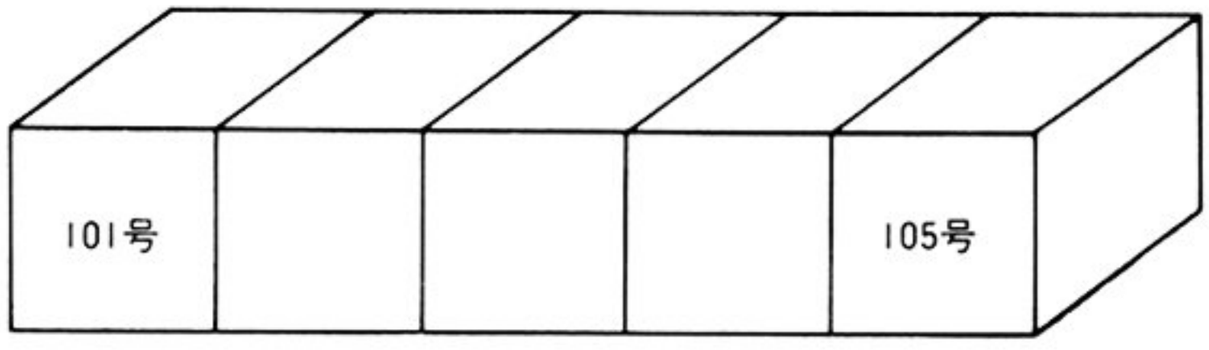
のようになります（これはプログラムではありません)。

繰り返して数値を入力するときのエラー処理などに利用できます。

```
 10 int a,b
 20 b=0
 30 repeat
 40    a=0
 50    input "Enter(end=0)> ";a
 60    if a>1000 then {
 70          print "Error!! Too large(";a;")"
 80          continue
 90    }
100    b=b+a
110 until a=0
120 print "Answer : ";b
130 end
```

１．６　準備作業　その２

同じデータ型をもつ複数個のデータの集まりを、配列といいます。マンションを例に配列の考え方を説明します。



たとえば、上のように101号室から105号室まで５部屋があるものとします。各部屋にはいろいろな家族構成があります。

101号……３人家族
102号……４人家族
　⋮
105号……２人家族

このとき、101号室などの部屋番号が変数、中に住んでいる人数がデータ（その変数の値）にあたります。

各部屋はひとつひとつが変数だと考えられますが、すべて別の変数にすると、データの入力などが繁雑になります。

```
10 int a,b,c,d,e
20 a=10 : b=20 : c=30 : d=40 : e=50
```

これを配列として宣言すれば、データの入力や利用が単純になります。

```
10 dim int a(4)={ 10 , 20 , 30 , 40 , 50 } : int i
20 for i=0 to 4
30   print "a(";i;")=";a(i)
40 next
```

配列変数は、x(0)、x(1)のように、変数名の後に括弧で番号をくくった形で用います。括弧内の番号は添字といい、配列の中の何番目の要素であるかを表します。たとえばx(2)は、xという配列変数の０番目の要素から数えて２番目の要素（x(0)、x(1)、x(2)と数える）という意味になります。

プログラム内で配列を使うには、「変数の宣言」と同じくあらかじめ dim 命令による「配列の宣言」を必ず最初に行ってください。配列の宣言とは、配列変数の名前と配列に入れるデータの型と個数を決めてやることで、

dim　型宣言　変数名 (添字の最大値)

という書式で行います。型宣言を省略すると int 型になるのは、変数の場合と同様です。dim 命令については、第4章でも説明してありますので、そちらも参照してください。

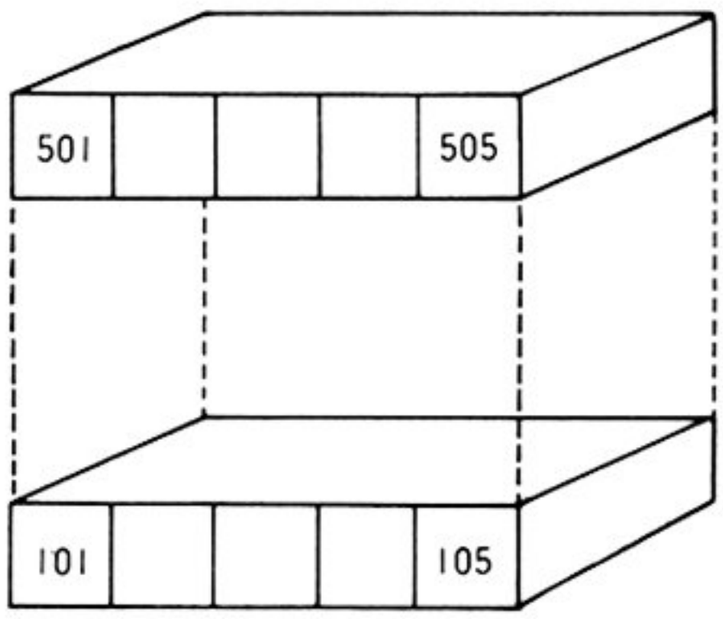
添字は0から始まるので、配列変数が持つ要素数は dim 命令で指定する添字＋1になります。たとえば、dim　a(5)と宣言すると、a(0)から a(5)までの6つの配列が使えるようになります。添字はメモリが不足しない範囲までとることができ、0から65535の値を使うことができます。

数値 (int, char, float) 型の配列については、各要素で扱える数値の範囲はデータの型に応じて決まりますが、文字 (str) 型は各要素で扱える文字列の長さを文字バッファのサイズにより指定できます。文字バッファのサイズを指定しない場合は32文字と見なされます。これより短い文字列しか代入されない場合は問題ありませんが、もっと長い文字列が代入されることを考えなければならない場合は次のようにして文字バッファのサイズの最大値（1～255）を指定します。

```
dim  str  a(10)[200]
```

これで、配列の要素 a(0)から a(10)にはそれぞれ最大200文字までの長さの文字列を代入して使うことができるようになります。

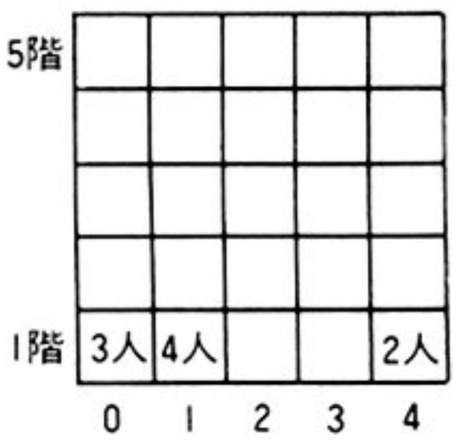
次に、先ほどのマンションが5階建てだとすると、



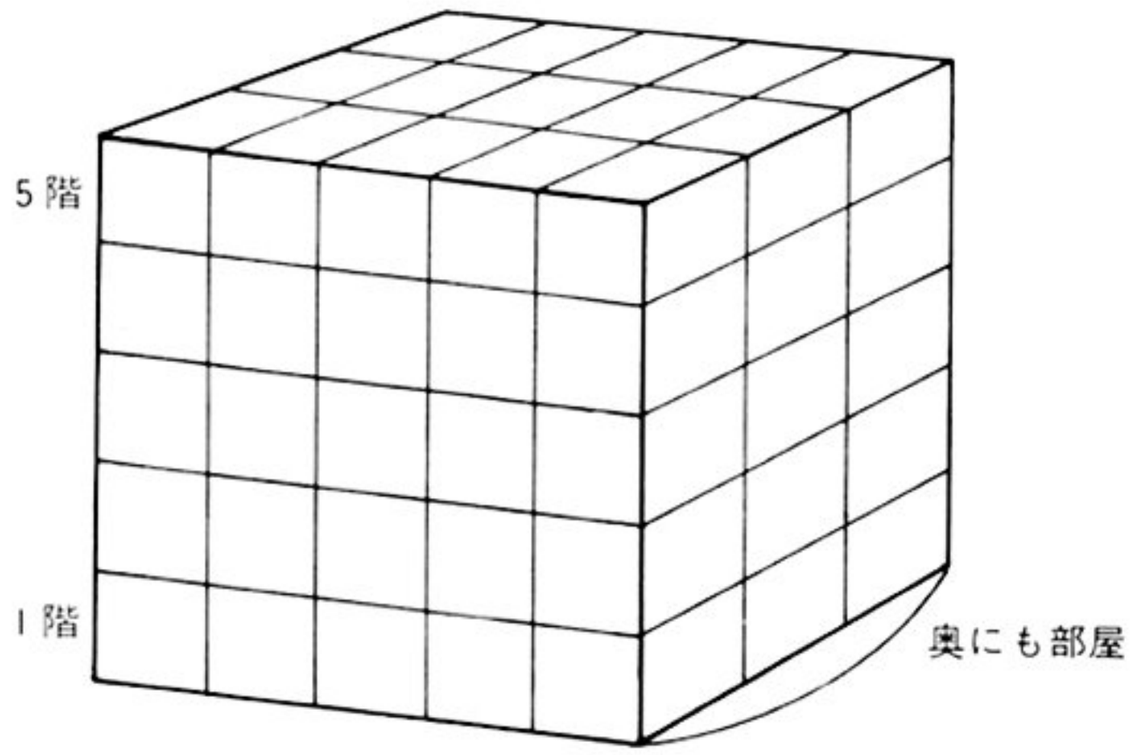
1階に5部屋で5階で25部屋になります。

1階　101～105号室
2階　201～205号室
⋮　⋮
5階　501～505号室

つまり、



のようになります。このような形式を、2次元配列といいます。さらに、このマンションが奥にも部屋があれば、3次元配列になります。



複数の添字を使った場合、配列は添字の個数に応じた次元になります。
たとえば、

```
dim  d(2,3)
```

という宣言では、以下に示すような二次元（縦方向と横方向の二次元）の配列がとられます。

d(0,0)	d(0,1)	d(0,2)	d(0,3)
d(1,0)	d(1,1)	d(1,2)	d(1,3)
d(2,0)	d(2,1)	d(2,2)	d(2,3)

```
dim  y(2,3,4)
```

のような三次元の配列については、縦方向と横方向に奥行きが加わった直方体をイメージしてください。

このように、配列の次元は、いくつ添字を指定するかによって決まります。次元は、理論的には255次元までとれることになっていますが、実際にはプログラムの1行の長さで指定できないので、もっと少なくなります。

以下に、いろいろな配列変数の例をあげます。

```
dim  char  a(12)……整数型(char 型)の配列変数(要素の数　13)
dim  str   b(4) ……文字型(str 型)の配列変数(要素の数　5)
dim  str   f(3,3) ……二次元文字型(str 型)の配列変数(要素の数　4×4＝16)
dim  n(2,5,3) ……三次元整数型(int 型)の配列変数(要素の数　3×6×4＝72)
```

1．7　汎用的な部品を作る

プログラムは、大きなプログラムになればなるほど、わかりにくく、また作成、修正（デバッグ）もむずかしくなります。このため、プログラムを小さな部品単位で作成すれば、だれにでもわかりやすく、また修正（デバッグ）も容易なプログラミングが可能となります。このことをモジュール化といいます。

X-BASICでは、この部品単位のプログラム作成をfunc～endfuncを使って関数として定義できるようになっています。

func～endfuncを使ったX-BASICのプログラムは次のようになります。

```
変数、配列変数の宣言   ┐
処理                   ├ メインプログラム
end                    ┘
func～                 ┐
   ＼                  ├ 関数1
endfunc                ┘
func～                 ┐
   ＼                  ├ 関数2
endfunc                ┘
   ⋮
```

実際のプログラム例を示します。

```
 10 /* 階乗を求めます！！
 20 /* メインプログラム
 30     int a
 40     float b
 50     input "階乗を求めたい数値を入力してください--->";a
 60     b=fact(a)      /* func～endfuncで定義されたfactという関数に引数aを渡して
呼び出します
 70     print "="
 80     print a;"の階乗は";b;"になります"
 90     end
100 /*
110 /* factという関数を定義します
120 /*
130 /* ここではローカル変数cによる再帰呼び出しを使用しています
140 /* ローカル変数はその使用される関数（func～endfuncで定義された関数）内で有
効な変数です
150 /* つまりfunc～endfunc内で宣言（使用）された変数はローカル変数になります
160 /* これに対し、メインプログラム中で宣言（使用）された変数a、bをグローバル
変数と呼びます
170 /* グローバル変数はfunc～endfuncで定義された関数内で使用できますが、
180 /* ローカル変数はその定義された関数内でしか使用できません
190 /*
200     func float fact(c;int)
210     if c=1 then print c;:return(1)
220     print c;"*";
230 /*
240 /* return命令により再帰呼び出し（fact関数（自分自身）に引数cを渡して呼び出
しています）を行っています
250 /* 結果として、c*(c-1)*(c-2)*(c-3)*...*3*2*1の値が戻り値としてreturn命令に
より返されます
260 /*
270     return(fact(c-1)*c)
280     endfunc
```

```
 10  /* Tiny EDITOR  ted.bas
 20  str fname[20]
 30  str buff[200]
 40  int fp, fc
 50  char CR=13, LF=10
 60  /*
 70  title()
 80  fninput()
 90  fp=fopen(fname,"c")
100  edit()
110  fc=fclose(fp)
120  end
130  /*
140  func title()
150  str dummy
160    cls
170    print "これは簡易エディタです。"
180    print "メモなどを作成するときにお使いください。"
190    print "なお、すでに存在するファイル名を指定したときは、無条"
200    print "件にそのファイルの内容を削除してしまいます。"
210    print
220    print "                                    何かキーを押してください  ";
230    dummy=inkey$
240    cls
250  endfunc
260  func fninput()
270    print "Input file name >";
280    linput fname
290  endfunc
300  func edit()
310    print "Input data(END = /)" : print
320    while buff <> "/"
330        linput ">";buff
340        if buff <> "/" then {
350            fwrites(buff,fp)
360            fputc(CR,fp)
370            fputc(LF,fp)
380        }
390    endwhile
400  endfunc
```

1．8　プログラムの保存と再生

プログラムは、コンピュータのメモリに記憶されます。しかし、コンピュータの電源が"切"(OFF)になるとそのプログラムは消えてしまいます。このため、プログラムはディスクなどの外部記憶装置に一旦保存して、必要があれば再生しなければなりません。

ここではプログラムをディスクに保存する方法と、保存されたプログラムを再生する方法を説明します。save、save@、load、load@、files 命令を使います。

1．8．1　プログラムの保存

プログラムをディスクに保存することをセーブ（save）といいます。保存するときは、あとでそのプログラムがわかるように名前（ファイル名といいます）をつけます。

プログラムを入力すると、コンピュータのメモリには、その内容が記憶されています。しかし、電源を切ったり、X-BASIC を終了させると、そのプログラムは消えてしまいます。保存する必要があるプログラムは必ずセーブしてください。

また、プログラムを実行する前にも、プログラムの暴走に備えてセーブしておいたほうがいいでし

よう。

それではセーブの方法を説明します。

入力待ち状態から、

```
save　"ファイル名"↵
```

と入力します。

ファイル名は、半角文字で18文字以内（ただし、識別できるのは8文字以内）で指定します。たとえば、ファイル名を「TEST」とすれば、

```
save　"TEST"↵
```

となります。この例では、ドライブの指定をしていませんので、自動的にX-BASICを起動したディスクに保存されます。

ドライブBに保存したいときは、ファイル名の先頭に「B：」をつけます。

```
save　"B：TEST"↵
```

ファイル名には拡張子をつけることができます。ファイル名のあとに「.」とそれに続く半角文字3文字以内で指定します。X-BASICで拡張子を省略してセーブしたプログラムには、自動的に「.bas」という拡張子がつきます。

ファイル名については別冊の「Human68kユーザーズマニュアル」を参照してください。

save命令では、プログラムをアスキー（テキストファイル）形式でセーブします。セーブされる順序は、行番号の小さい順になります。

「@」オプションを指定すると、行番号を省略した形でプログラムがセーブされます。

```
save@　"B：TEST"↵
```

1．8．2　プログラムの再生

プログラムをディスクから再生することをロード（load）といいます。

それではロードの方法を説明します。

すでにメモリにプログラムがある場合は、ロードすることによってそのプログラムは消されますので、注意してください。入力待ち状態から、

```
load　"B：TEST"↵
```

と入力します。

この例では、ドライブBにある「TEST.bas」というファイル名のプログラムをロードします。

load命令も「@」オプションが指定できます。ただし「save@」でセーブされたプログラムしか使えません。

load@命令では、save@命令でセーブされたプログラムに任意の行番号がつけれます。

また、すでにメモリ上にプログラムがあるときに、load@命令でロードされたプログラムを、マージ（混合、追加）することができます。

たとえば、行番号10～100のプログラムがメモリ上にあるとき、ドライブBから「TEST.bas」と

いう行番号が省略されたプログラムを110行目からロードすると次のようになります。

```
 10
 20
  ⋮        } メモリ上のプログラム
 90
100
load@ "B : TEST",110 ↵
list ↵
 10
 20
  ⋮        } メモリ上のプログラム
 90
100
110
  ⋮          「TEST. bas」がマージされる
200
```

1．8．3　保存したファイルの確認

ディスクに保存されているファイルの一覧を見るために files 命令があります。次のように入力してください。

```
files   "B : " ↵
```

ドライブＢのディスクのファイル一覧が表示されます。また、ファンクションキーの[Ｆ１]キーを押しても同じことができます。ただし、この場合は、カレントドライブ（現在作業中のドライブ）のディスクのファイル一覧が表示されます。

```
files"b:"
***** Kバイトが使用可能です
      "A:¥BAS¥SAMPLE1                .BAS"   /*     ···· ········ ········
      "A:¥BAS¥SAMPLE2                .BAS"   /*     ···· ········ ········
      "A:¥BAS¥LS                     .BAS"   /*     ···· ········ ········
      "A:¥BAS¥GREP                   .BAS"   /*     ···· ········ ········
      "A:¥BAS¥DEMO                   .BAS"   /*     ···· ········ ········
      "A:¥BAS¥GRAPH                  .BAS"   /*     ···· ········ ········
      "A:¥BAS¥PHOTO1                 .BAS"   /*     ···· ········ ········
      "A:¥BAS¥PHOTO2                 .BAS"   /*     ···· ········ ········
      "A:¥BAS¥PHOTO3                 .BAS"   /*     ···· ········ ········
Ok
```

コラム　X-BASIC のいろいろな起動

1．コマンドモードからの起動

デスクトップ画面でのコンピュータの操作に慣れてきたら、もうひとつの世界、Human68k のコマンドモードからX-BASIC を起動してみましょう。

デスクトップ画面をながめてみると、タンク（戦車）のアイコンが見えます。これが、Human68k のコマンドモードへの入口です。この「**COMMAND.X**」というアイコンを、マウスでポイントして、左ボタンでダブルクリックしてください。(「**COMMAND.X**」はHuman68k ver.3.0システムディスクにはいっています)

しばらくすると、デスクトップ画面が消えて、

```
A>
```

という文字（プロンプト）が表示され、そこでカーソルが点滅しているはずです。
Human68k ver.3.0システムディスクがドライブ0にはいっているときは、

```
¥BASIC2¥BASIC ↵
```

(↵はリターンキーのこと)

Human68k ver.3.0システムディスクがドライブ1にはいっているときは、

```
B：¥BASIC2¥BASIC ↵
```

とキーボードから入力して（これは英小文字でもかまいません）ください。X-BASICが起動します。

では、X-BASIC を終了しましょう。キーボードから、

```
system ↵
```

(↵はリターンキーのこと)

と入力してください。またはファンクションキーの F10 キーを押してからリターンキーを押します。
X-BASIC が終了して、Human68k のコマンドモードに戻ります。

コマンドモードに戻ったら、今度は、

```
exit ↵
```

と入力し、メッセージが表示されたら何かキーを押すか、マウスのボタンをクリックしてください。
デスクトップ画面に戻ります。

Human68k のコマンドモードでは、いろいろなコマンドが使えます。また、システム起動時の環境を設定することもできます。詳細は、「Human68k ユーザーズマニュアル」を参照してください。

2．プログラムの自動実行

X-BASIC では、起動時に自動的にファンクションキーなどの初期設定を行うプログラムをロードして実行することができます。ただし、これらの各種初期設定を行った後、このプログラムのリストは自動的にメモリ上から消えます。

コマンドモードでプロンプトが表示されているとき、

basic　test.bas ↵

と入力すれば、X-BASICの起動後、プログラム"test.bas"をロードして実行します。拡張子が".bas"のときには、拡張子は省略できます。もし、同じディレクトリに"autorun.bas"という名前のプログラムがあれば、単に

basic ↵

と入力するだけで、X-BASICの起動後、自動的にプログラム"autorun.bas"を実行します。

一般に、Human68kのコマンドモードからX-BASICを起動するための書式は、次のようになります。

basic　〔〔〈スイッチ〉〕　〔〔〈ドライブ名〉〕〔〈パス名〉〕〈プログラム名〉〕〕

〈プログラム名〉の拡張子が".bas"のとき、拡張子は省略できます。

3．起動時のスイッチ

Human68kのコマンドモードからX-BASICを起動するときには、次のスイッチを指定できます。

／f　　フリーエリアサイズの指定
／w　　水平画面サイズの指定
／c　　コンフィギュレーションファイルの指定

次節で説明するコンフィギュレーションファイルの中でも、／fスイッチ、／wスイッチと同様の設定ができます。

・／fスイッチ

例）　basic　/f160

X-BASICで使うフリーエリアをキロバイト(KB)単位で指定します。ただし、この指定で確保されるメモリは、プログラム（テキスト）と変数・配列変数などのためのエリアです。FM音源のトラックバッファなど外部関数で使うエリアや、チャイルドプロセス実行のためのエリアは含まれません。

・／wスイッチ

例）　basic　/w64

水平画面サイズ、つまり、画面の横幅に表示できる文字数(桁数)を指定します。指定できるのは64か96です。それ以外の数値では、エラーになります。

・／c スイッチ

／c スイッチについては、次節で詳しく説明します。

これらのスイッチは、同時に設定することもできます。

例）　basic　/w96　/f200

4．オリジナル X-BASIC

X-BASIC は、起動時にスイッチを指定しなければ、X-BASIC 本体（"basic.x"）と同じディレクトリ内の"basic.cnf"というファイルを探します。これは言い換えれば、X-BASIC は常に、

```
basic　/cbasic.cnf
```

というスイッチを指定して起動している、ということです。"basic.cnf"には、X-BASIC を使う上でのいろいろな動作環境を設定してあります。このようなファイルを、一般にコンフィギュレーションファイルと呼びます。コンフィギュレーションファイルがないと、X-BASIC は起動しません。

あらかじめ用意されている"basic.cnf"の内容は次のようになっています。

```
FREE　　＝ 128
WIDTH　＝ 64
BEEP　　＝ ON
CAPS　　＝ OFF
FUNC　　＝ AUDIO
FUNC　　＝ GRAPH
FUNC　　＝ MOUSE
FUNC　　＝ MUSIC
FUNC　　＝ MUSIC3
FUNC　　＝ SPRITE
FUNC　　＝ STICK
FUNC　　＝ IMAGE
```

"＝"の左側は、X-BASIC のコンフィギュレーションファイルの中で通用するコマンドと考えてください。"＝"の右側は、コマンドに与える引数です。

それぞれについて、説明します。

・FREE

例）　FREE　　＝　160

X-BASIC で使うフリーエリアをキロバイト（KB）単位で指定します。ただし、この指定で確保されるメモリは、プログラム（テキスト）と変数・配列変数などのためのエリアです。FM 音源のトラックバッファなど外部関数で使うエリアや、チャイルドプロセス実行のためのエリアは含まれません。／f スイッチと同じものです。

・WIDTH

例）　WIDTH　＝　96

水平画面サイズ、つまり、画面の横幅に表示できる文字数（桁数）を指定します。指定できるのは64か96です。それ以外の数値では、エラーになります。／wスイッチと同じものです。

・BEEP

例）　BEEP　＝　ON

エラーのときなどにビープ音を鳴らすかどうかを、ONまたはOFFで指定します。

・CAPS

例）　CAPS　＝　OFF

X-BASICの予約語の表示方法を指定します。大文字で表示するにはON、小文字で表示するにはOFFを指定します。

・FUNC

例）　FUNC　＝　SPRITE

組み込まれる外部関数を指定します。外部関数は、X-BASICの起動時にいっしょにメモリにロードされ、常駐する関数のことです。X-BASICでは、あらかじめいくつかの外部関数を用意しています。拡張子はすべて".fnc"となっています。

AUDIO …… ADPCM制御の関数
GRAPH　……グラフィック制御の関数
MOUSE　……マウス・トラックボール制御の関数
MUSIC/MUSIC3‥ FM音源、MIDI、ADPCMの制御関数
SPRITE　……スプライト制御の関数
STICK ……ジョイスティック制御の関数
IMAGE ……画像処理制御の関数
(IMAGEは、別売のカラーイメージユニット用の関数です)

これらのファイルは、コンフィギュレーションファイルと同じディレクトリになければなりません。

／cスイッチを使えば、"basic.cnf"以外のファイルをコンフィギュレーションファイルとして指定できます。これによって、必要な外部関数だけを組み込んだ、コンパクトなX-BASICを作ることができます。

コンフィギュレーションファイルは、ユーザーが簡単に作成できるテキストファイル形式です。

"basic.cnf"を書きかえたり、新たにコンフィギュレーションファイルを作成するときには、スクリ

ーンエディタ ED を使います。

Human68k のコマンドモードでプロンプトが表示されているときに、

```
ed  a:¥basic2¥basic.cnf ↵
```

と入力すると、ED が起動して、"basic.cnf"を読み込み、修正できるようになります。ED についての詳細は、「Human68k ユーザーズマニュアル」を参照してください。

Human68k システムディスクがドライブ 1 にはいっている時はa:をb:にして入力してください。

5. 拡張外部関数

X-BASIC は、いろいろな制御構造と、ローカル変数を扱える関数の定義によって、行番号に依存しないプログラムを作成しやすくなっています。ローカル変数を扱えるメリットは大きく、作成した関数は、より汎用性の高いものになります。

頻繁に利用する関数は、拡張外部関数として、X-BASIC に組み込んでおきたいところです。それには、別売の「C Compiler PRO-68K」に付属している BASIC-C コンバータ「XBAStoC (以下、BASTOC)」をお使いください。BASTOC を使うために、C 言語やアセンブリ言語の知識は必要ありません。X-BASIC で作った関数を、BASTOC が C 言語のソースプログラムにコンバート(変換)してくれます。

ただし、BASTOC でコンバートするために、注意しなければならないことや、制限される事柄もあります。

6. BASTOC

X-BASIC は、その制御構造や関数の使い方などで、C 言語に近い仕様をもった BASIC です。しかし、BASIC と C 言語はもともとの設計思想が大きく違っています。また、BASTOC は、コンバートする過程で独自の変数などを生成します。そのため、BASTOC を使う上で守らなければならない制限が発生してしまいます。詳細は、別売の「C Compiler PRO-68K」同梱のマニュアルを参照してください。

・変数名、関数名の制限

次に示すものは、変数や関数の名前として使用できません。

先頭が S で 6 桁の数字が続くもの	例)	S012345
先頭が L で 6 桁の数字が続くもの	例)	L012345
先頭が b_ ではじまるもの	例)	b_nanja
外部関数の名前と同じもの	例)	line

・配列代入の制限

宣言をしたあとで代入できる配列変数は、1 次元配列だけです。2 次元以上の配列変数は、宣言時に代入 (初期化) してください。

例)　10 dim　a(10,10)　　→　　10 dim　a(10,10)={1,2,3,4,5,6,7}

```
20 a={1,2,3,4,5,6,7}
```

・func～endfunc に関する制限

関数の引数の型が int 型の場合、float 型の数値を直接渡してはいけません。int 関数で変換してください。

例）

```
10 float  a
20 a=123.456
30 test(a) /* No good!
      ⋮
80 end
90 func  test(data;int)
      ⋮
```

→

```
10 float  a
20 a=123.456
30 test(int(a))
      ⋮
80 end
90 func  test(data;int)
      ⋮
```

また、関数内でローカル変数として宣言した str 型の変数は、戻り値として使うことはできません。str 型の変数を戻り値にする場合は、グローバル変数を使用してください。

例）

```
10 str  sa,sb
20 sb=test(123)
      ⋮
80 end
90 func  str  test(i;int)
100 str  ss
      ⋮
190 return(ss)/* No good!
200 endfunc
      ⋮
```

```
10 str  sa,sb
20 sb=test(123)
      ⋮
80 end
90 func  str  test(i;int)
100 str  ss
      ⋮
180 sa=ss
190 return(sa)
200 endfunc
      ⋮
```

7．Human68kの環境設定

X-BASICの"basic.cnf"と同じように、Human68kにも"CONFIG.SYS"というシステム構築用ファイル（コンフィギュレーションファイル）があります。この"CONFIG.SYS"によって、システムの環境を設定したり、デバイスドライバの組み込みなどを行います。X-BASICでも外部関数などを利用するときは、デバイスドライバとして次のようなものを組み込んでおく必要があります。

・FM音源、MIDI、ADPCMの制御関数　… OPMDRV3.X
・浮動小数点演算　… FLOAT2.X、FLOAT3.XまたはFLOAT4.X
・かな漢字変換を利用するとき　… ASK68K.SYS
・プリンタを利用するとき　… PRNDRV.SYSなど

なお、標準で用意されている"CONFIG.SYS"では、これらのデバイスドライバが組み込まれるようになっています。

システム構築用ファイルは、ユーザーが簡単に作成できるテキストファイル形式です。"CONFIG.SYS"を書きかえるときには、SX-WINDOWの日本語マルチフォントエディタやスクリーンエディタEDを使います。

Human68kのコマンドモードでプロンプトが表示されているときに、

```
ED A:¥CONFIG.SYS ↵
```

と入力すると、EDが起動して、"CONFIG.SYS"を読み込み、修正できるようになります。"CONFIG.SYS"やデバイスドライバ、EDの使い方の詳細については「Human68kユーザーズマニュアル」を参照してください。

なお、OPMDRV3.Xは、「C Compiler PRO-68K ver.2.0」でサポートされていたOPMDRV2.X用のX-BASIC外部関数とは互換性がなく、同時に使用することができませんのでご注意ください。

従来PCMDRV.SYSでサポートされていた機能は、OPMDRV3.Xで包含していますので、ADPCM用のX-BASIC外部関数（A_PLAY、A_REC）も従来通りお使いいただけます。

第２章　応用編

この章では、主に外部関数によって実行できる機能について説明します。

外部関数は、本機の持ついろいろな機能を X-BASIC から利用するためのものです。外部関数によって本機のほとんどの機能を引き出すことができる、といえるでしょう。ここでは、次のものについて詳しく説明します。

・画面制御（グラフィック、スプライト）
・FM 音源制御
・音声サンプリング（ADPCM）
・ファイル操作

ファイル操作はすべて標準関数で行いますが、ファイルに対する考え方が X-BASIC の特長の１つとなっているため、この章で詳しく説明します。

その他に、やはり X-BASIC の特長といえる次の機能についても解説します。

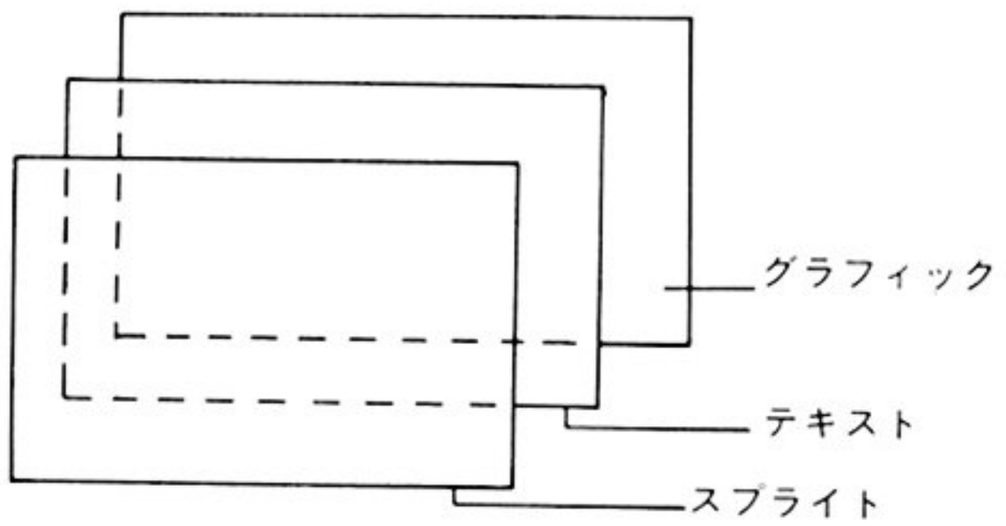
・マウス、ジョイスティック制御
・文字列操作
・子プロセス

２．１　画面制御

本機で扱う画面には、大きく分けて次の３つがあります。

・テキスト
・グラフィック
・スプライト

画面は、下の図のような優先度で重なっていると考えてください。



グラフィック画面とスプライト画面については、このあとの節で説明します。

テキスト画面は、文字を表示する画面です。プログラムの作成や編集、リストの表示を行っているのが、テキスト画面です。

テキスト画面の表示サイズは、基本的には64桁×32行と96桁×32行の２種類です。では、ダイレクトモードで、テキスト画面の表示桁数を変えてみましょう。次のように入力してください。

screen 2,0,1,0 ↵ または　width 96 ↵ （このスクリーンモードではスプライト画面は使えません）

画面が消去され、表示される文字の大きさも変わったはずです。また、同時にグラフィック画面とスプライト画面も消去されています。つづいて次のように入力します。

screen 1,0,1,0 ↵ または　width 64 ↵

これを交互に繰り返してみてください。screen 命令は、テキスト画面をはじめ次節で説明するグラフィック画面やスプライト画面のすべてのスクリーンモードを決めるときに使います。width 命令は、テキスト画面の表示桁数を変えるときに使います。

では次に、表示行数を変えてみます。表示行数というのは、テキスト画面がスクロールする行数のことです。次のように入力してください。

console　0,10,1 ↵

ここでファイル一覧を表示してみましょう。次のように入力します。

files ↵　（または F 1 キーを押します）

画面の上10行だけを使って文字がスクロールします。では、元に戻します。

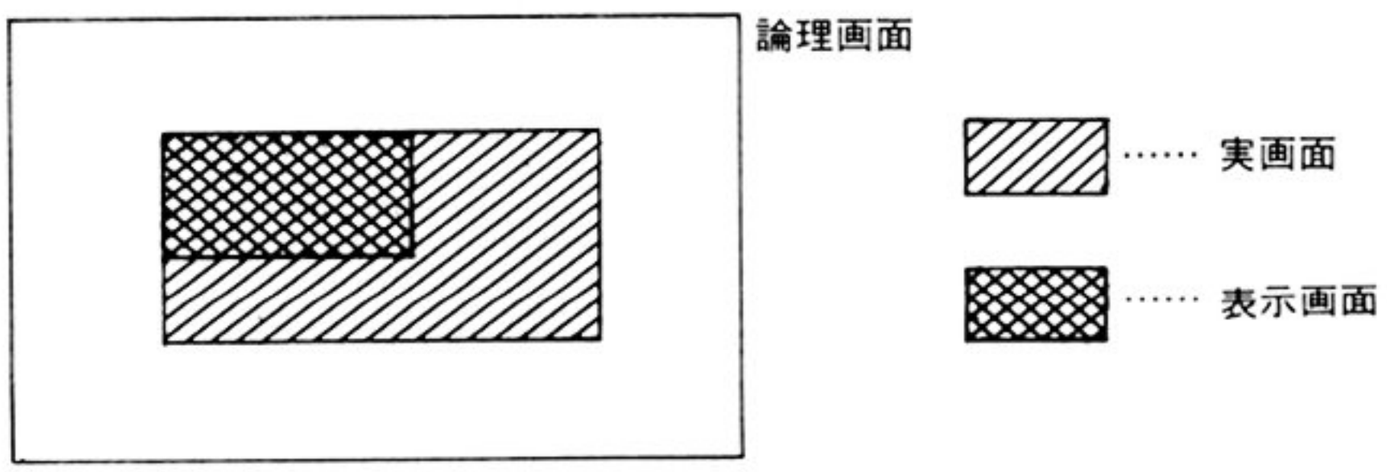
console　0,31,1 ↵

console 命令は、テキスト画面のスクロール行数(表示行数)を変えるときに使います。また、画面最下行のファンクションキー表示の有無も、console 命令で行います。詳しくは第４章を参照してください。

2.1.1　グラフィック

グラフィック画面は、点や線、絵を描く画面です。表示はすべてドット単位で行われ、ドットの位置はＸ座標とＹ座標で表します。

本機のグラフィック画面には、次の図のように、論理画面、実画面、表示画面があります。



・論理画面

論理画面は、X-BASICのグラフィック関数で、X、Y座標として引数に指定できる画面範囲（座標系）です。指定できる範囲は次のとおりです。

X座標……－32,768から32,767まで
Y座標……－32,768から32,767まで

・実画面

実画面は、実際にグラフィックを書き込むことのできる範囲です。実画面サイズには、512×512ドットと1024×1024ドットの2つがあります。また、実画面サイズと色モード（最大表示色）によって、設定できるグラフィックページ数（最大4ページ）が変わります。

・表示画面

表示画面は、実画面の中の実際に表示されている範囲です。表示画面サイズには、次のものがあります。

256×256ドット
512×512ドット
768×512ドット

では、表示画面サイズや実画面サイズなど（スクリーンモード）を変えてみましょう。次のように入力してください。ただし、グラフィックRAMをRAMディスクなどで使用している場合は、そのデータをフロッピーディスクやハードディスクなどにセーブしておいてください。以降のscreen命令を実行するとグラフィックRAMは初期化されます。

```
screen  2,0,1,1 ↵
```

スクリーンモードは次のように設定されます（このモードではスプライト画面は使えません）。

表示画面サイズ……768×512ドット
実画面サイズ……1024×1024ドット
色モード（最大表示色）……16色（65536色の中からドット単位で指定）
グラフィックページ数……1（グラフィックページ0）

また、グラフィック画面の初期化も同時に行います。では、別のスクリーンモードを設定してみましょう。次のように入力します。

```
screen  1,1,1,1 ↵
```

スクリーンモードは次のように設定されます。

表示画面サイズ……512×512ドット
実画面サイズ……512×512ドット
色モード（最大表示色）……16色（65536色の中からドット単位で指定）
グラフィックページ数……4（グラフィックページ0～3）

また、次のように入力すると、65536色同時発色のグラフィック画面が使えます。

```
screen  1,3,1,1 ↵
```

スクリーンモードは次のように設定されます。

表示画面サイズ……512×512ドット
実画面サイズ……512×512ドット
色モード（最大表示色）……65536色
グラフィックページ数……1（グラフィックページ0）

このようにscreen命令のパラメータを変えるだけで、いろいろなスクリーンモードを設定できます。グラフィック関数を使うときには、必ずスクリーンモードを設定するようにしてください。

実際に実画面の中でグラフィックを描くことのできる範囲を、クリッピングエリアといいます。screen命令を実行すると、クリッピングエリアは表示画面と同じ大きさになります（このときの実画面における表示画面の左上座標は（0,0）になります）。クリッピングエリアを実画面の大きさにまで広げておけば、表示画面サイズよりも大きな絵などを描くことができます。では、次のように入力してください。

```
screen  2,0,1,1 ↵
window  0,0,1023,1023 ↵
```

これで、実画面いっぱいにグラフィックを描くことができます。window関数は、クリッピングエリアを指定する関数です。

クリッピングエリアが表示画面より大きいときは、表示画面を移動して実画面の別の部分を見ることができます。次のプログラムを実行してみましょう。

```
 10 str  a
 20 screen 0,3,1,1
 30 window(0,0,511,511)
 40 apage(0)
 50 vpage(1)
 60 home(0,128,128)
 70 fill(140,140,350,350,30)
 80 waiting()
 90 home(0,0,0)
100 waiting()
110 home(0,0,256)
120 waiting()
130 home(0,256,0)
140 waiting()
150 home(0,256,256)
160 waiting()
170 home(0,128,128)
180 waiting()
190 width 96
200 end
210 func waiting()
220 locate 0,0
230 print "push SPACE key"
240 a=inkey$
250 endfunc
```

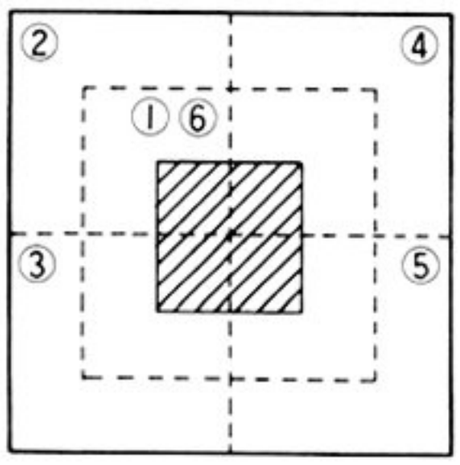
複数のグラフィックページがあるスクリーンモードのとき、実際にグラフィックを描くページ（ア

クティブページ)を指定するのが、40行のapage関数です。また、実際に表示されるページを指定するのが、50行のvpage関数です。vpage関数の指定によっては、複数のページを重ねて表示させることもできます。このとき、グラフィックページ0が一番上に表示されます。この表示の優先順位は、次のようになっています。

高　←　　→　低

グラフィックページ0　グラフィックページ1　グラフィックページ2　グラフィックページ3

表示画面の移動にはhome関数を使います。このプログラムでは、実画面のほぼ中央に四角形を描き、home関数で表示画面を次の図のように（①～⑥）移動しています。



グラフィックは、一般に次の手順で使用します。

1　スクリーンモード(表示画面サイズや実画面サイズ及び色モードなど)を設定する(screen)
2　クリッピングエリアを設定する（window)
3　アクティブページを決める（apage)
4　グラフィック画面を表示する（vpage)
5　グラフィックを描く
6　表示画面（クリッピングエリア）を移動する（home)

また、グラフィック画面の実画面内の任意の領域のグラフィックRAMデータを配列に読み込んだり、逆に配列のデータをグラフィックRAMに書き込んだりする関数（get、put）もあります。

2.1.2　スプライト

ゲームプログラムでは、いろいろな形のキャラクタが画面の中を動き回っています。これと同様のことがX-BASICで実現できます。

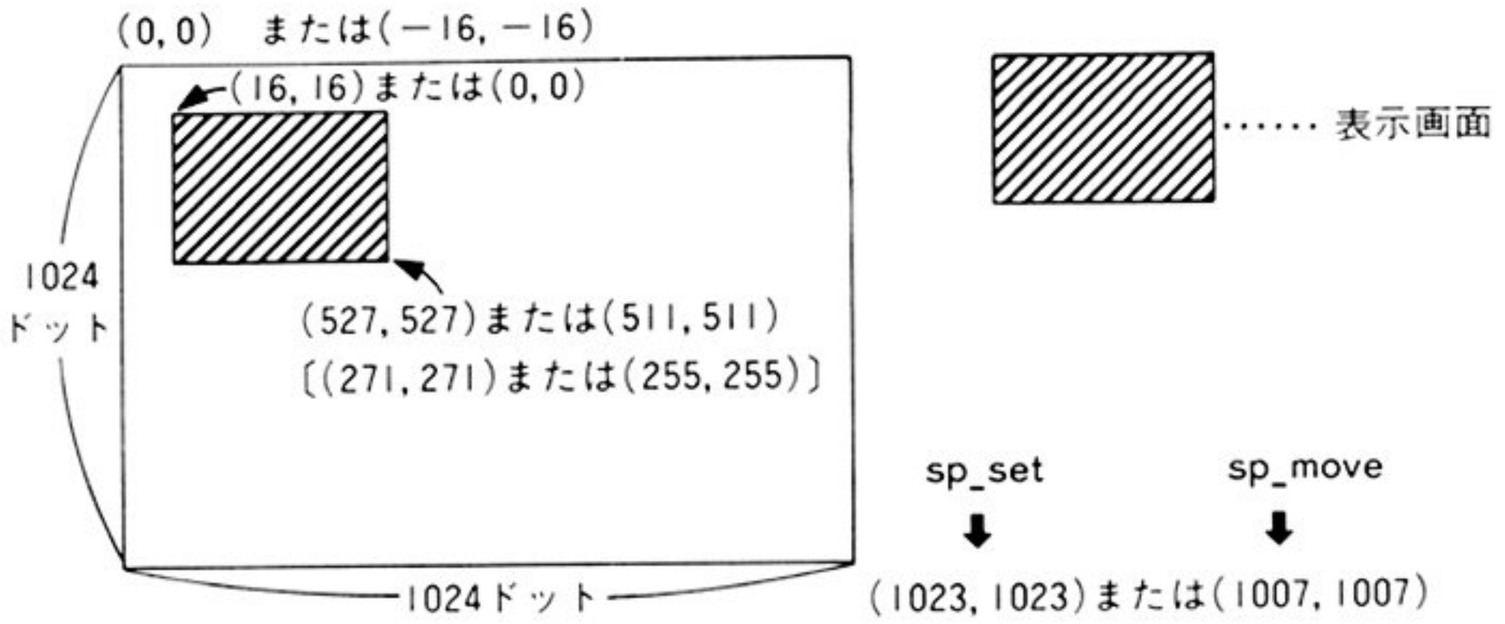
スプライト画面は、スプライトやバックグラウンドを表示するためのものです。スプライト画面では、16×16ドットで作成したスプライトのパターンを1つのキャラクタとして自由に動かすことができます。

グラフィック画面が複数のグラフィックページを持っていたように、スプライト画面にも複数のページがあります。これをプレーンといい、最大128面（プレーン0～127）まで使うことができます。各プレーンには、1つずつスプライトのパターンを表示できます。また、プレーンの番号が小さいほど、表示優先順位が高くなります。つまり、プレーンの番号が違うスプライトのパターンが重なったときに、プレーンの番号の小さい方が優先的に上に重なったように表示されます。また、スプライトとバックグラウンドの表示優先順位も変更できます。

スプライトやあとで説明するバックグラウンドのパターンは、そのパターンに対してパレットブロック（1～15）が割りあてられます。また、そのパターンのドットに対して、割りあてられたパレットブロックの中から16色のパレットを選択して設定できます。この16色のパレットには、65536色の中から任意の色を割りあてられます。

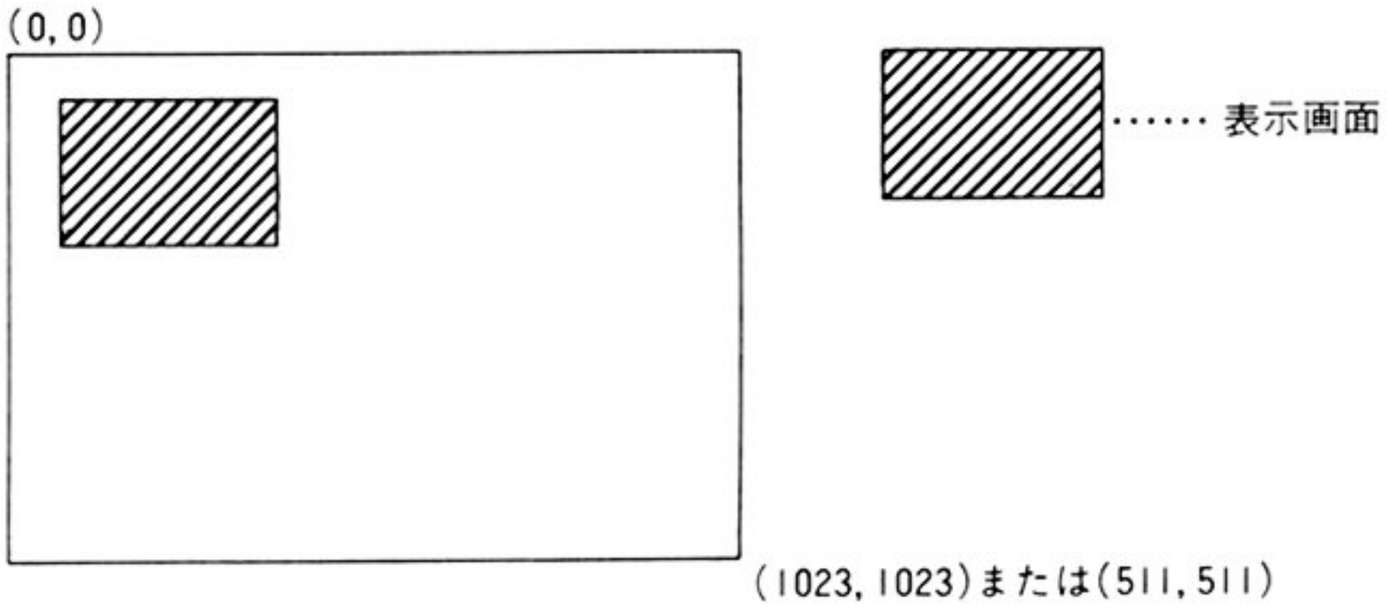
定義できるスプライトやバックグラウンドのパターンの最大数は、あとに説明するバックグラウンド（テキストページ）の使用状況によって変わります。バックグラウンド（テキストページ）を2面とも使っているときは、16×16ドットのパターンは最大128個まで定義できます。また、バックグラウンド(テキストページ)をまったく使っていなければ、16×16ドットのパターンは最大256個まで定義できます。スプライトやバックグラウンドのパターンは水平あるいは垂直反転ができます。

スプライトは、表示画面サイズが256×256ドットか512×512ドットのときに表示できます。スプライトを表示できる範囲は、次の図のとおりです。なお、図の座標は前者がsp_setで、後者がsp_moveの場合です。



バックグラウンドは、プレーンとは別に最大2面まで表示できます。バックグラウンドの特長は、テキストページ1面に最大4096個(64×64個)のパターンを定義して、このテキストページをそれぞれのバックグラウンドに割りあて、そのバックグラウンドを自由にスクロールして表示できる点です。

バックグラウンドは、表示画面サイズが256×256ドットのとき2面、512×512ドットのとき1面まで表示できます。バックグラウンドを表示できる範囲は、次の図のとおりです。



バックグラウンドのテキストページに使用するパターンサイズやバックグラウンドのサイズも、表示画面サイズによって変わります。

表示画面サイズ	パターンサイズ	バックグラウンドサイズ
256×256ドット	8×8ドット	512×512ドット
512×512ドット	16×16ドット	1024×1024ドット

スプライトは、一般に次の手順で使用します。

1　スクリーンモード（768×512以外の表示画面サイズなど）を設定する（screen)
2　スプライト画面やパターンを初期化する（sp_init　sp_clr)
3　スプライトのパターンを定義する（sp_def)
4　パレットブロックやパレットを定義する（sp_color)
5　表示するプレーンを決める（sp_on、sp_off)
6　スプライト画面を表示する（sp_disp)
7　スプライトパターンを動かす（sp_setまたはsp_move)

また、スプライトの状態を読み出す関数としてsp_stat関数、スプライトのパターンのデータを読み出す関数としてsp_pat関数があります。

3、4は、付属のスプライトエディタを使って定義することもできます。

バックグラウンドは、一般に次の手順で使用します。

1　スクリーンモード（768×512以外の表示画面サイズなど）を設定する（screen)
2　スプライト画面やパターンを初期化する（sp_init、sp_clr)
3　バックグラウンドのパターンを定義する（sp_def)
4　パレットブロックやパレットを定義する（sp_color)
5　テキストページにパターンを設定する（bg_fill、bg_put)
6　スプライト画面を表示する（sp_disp)
7　テキストページをバックグラウンドに割りあて、そのバックグラウンドを表示する（bg_set)
8　バックグラウンドを動かす（bg_scroll)

また、バックグラウンドの状態を読み出す関数として、bg_stat関数、テキストページの状態を読み出す関数としてbg_get関数があります。バックグラウンドのパターンのデータを読み出す関数としてsp_pat関数があります。

次のプログラムでは、スプライトのパターンを2個、バックグラウンドのパターンを2個定義しています。バックグラウンドを2面共スクロールさせながら、画面の中を2個のスプライトが動きます。画面の端にきたところで、スプライトは反転します。

```
10 /* sprite sample
20 int i,j,r,di,x1,y1,x2,y2,x3,y3,y4
30 char pc,pb,vr,vr_s,vr_s1,hr,hr_s,hr_s1,cd
40 float f,f1
50 /* define sprite-pattern
60 dim char SP0(255)={
70       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
80       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
90       0, 0, 0,11,11, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
100      0, 0, 0,11,11,11, 0, 0, 0, 0, 0,11, 0, 0, 0, 0,
110      0, 0, 0,11,11,11,10, 0, 0, 0, 0,11,10, 0, 0, 0,
120     13,13, 4,11,11, 2, 2, 2, 2, 2, 2, 3, 3, 3, 0, 0,
130     13, 4,12, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0,
140      4,12, 5,11,11,11,10, 3, 3, 3, 3,11,10, 3, 3, 3,
150      4,12, 5,11,11,11,10, 3, 3, 3, 3,11,10, 3, 3, 3,
160     13, 4,12, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0,
170     13,13, 4,11,11, 2, 2, 2, 2, 2, 2, 3, 3, 3, 0, 0,
180      0, 0, 0,11,11,11,10, 0, 0, 0, 0,11,10, 0, 0, 0,
190      0, 0, 0,11,11,11, 0, 0, 0, 0, 0,11, 0, 0, 0, 0,
200      0, 0, 0,11,11, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
210       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
220       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
230 }
240 dim char SP1(255)={
250       0, 0, 0, 0, 0,14, 0, 0, 0, 0,14, 0, 0, 0, 0, 0,
260       0,12,12,12, 0, 0,14, 0, 0,14, 0, 0,12,12,12, 0,
270      12,13,13,13,12, 0,14, 0, 0,14, 0,12,13,13,13,12,
280      13,13, 8,13,13,12, 0,14,14, 0,12,13,13, 8,13,13,
290      13, 8,13, 8,13,13, 0, 7, 7, 0,13,13, 8,13, 8,13,
300      13,13,13,13,13,13,13, 5, 5,13,13,13,13,13,13,13,
310      13,13, 9,13,13, 8,13, 5, 5,13, 8,13,13, 9,13,13,
320      13,13,13, 9,13,13,13, 5, 5,13,13,13, 9,13,13,13,
330      13,13,13,13, 9,13,13, 5, 5,13,13, 9,13,13,13,13,
340      12,13,13, 9,13,13,13, 5, 5,13,13,13, 9,13,13,12,
350      12,13,13,13,13,13,13, 3, 3,13,13,13,13,13,13,12,
360       0,12,13,13,13,13,13, 4, 4,13,13,13,13,13,12, 0,
370       0,12,13,13,13,13,13, 2, 2,13,13,13,13,13,12, 0,
380       0, 0,12,13,13,13, 0, 6, 6, 0,13,13,13,12, 0, 0,
390       0, 0,12,12,12, 0, 0, 0, 0, 0, 0,12,12,12, 0, 0,
400       0, 0,12, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,12, 0, 0
410 }
420 /* define background-pattern
430 dim char BG0(63)={
440       0, 8, 0, 8, 0, 8, 0, 8,
450       8, 0, 0, 0, 0, 0, 0, 0,
460       0, 0, 0, 0, 0, 0, 0, 8,
470       8, 0, 0, 0, 0, 0, 0, 0,
480       0, 0, 0, 0, 0, 0, 0, 8,
490       8, 0, 0, 0, 0, 0, 0, 0,
500       0, 0, 0, 0, 0, 0, 0, 8,
510       8, 0, 8, 0, 8, 0, 8, 0
520 }
530 dim char BG1(63)={
540       0, 0, 0, 0, 0, 0, 0, 0,
550       0,10, 0,10, 0,10, 0, 0,
560       0, 0, 0, 0, 0, 0,10, 0,
570       0,10, 0, 0, 0, 0, 0, 0,
580       0, 0, 0, 0, 0, 0,10, 0,
590       0,10, 0, 0, 0, 0, 0, 0,
600       0, 0,10, 0,10, 0,10, 0,
610       0, 0, 0, 0, 0, 0, 0, 0
620 }
630 /* main */
640 screen 0,3,1,1 /* 表示画面サイズ 256*256 (768*512以外を指定する)
650 sp_init()       /* スプライト画面の初期化
660 sp_clr(0,255)   /* スプライトパターンのクリア (0～255のパターンコード)
670 sp_disp(1)      /* スプライト画面の表示 (0なら表示せず、1なら表示する)
680 sp_on(0,3)      /* スプライトプレーンの表示 (0～127のプレーン番号)
690 /* set sprite-pattern & background-pattern
700 sp_def(0,SP0,1)      /* 16*16 dots
710 sp_def(1,SP1,1)      /* 16*16 dots
720 sp_def(8,BG0,0)      /*  8* 8 dots
730 sp_def(9,BG1,0)      /*  8* 8 dots
740 sp_def(3,SP0,1)      /* 16*16 dots
750 sp_def(4,SP1,1)      /* 16*16 dots
760 /* set palet-block
770 for i=2 to 3
780     pb=i
790     if i=2 then r=28010 else r=-8000
800     for j=0 to 15
810         pc=j
820         randomize(r+pi(i*10)*j)
830         sp_color(pc,hsv(rnd()*192,rnd()*32,31)+1,pb)
840     next
850 next
860 /* set background 0
870 bg_fill(0,pat_dat(0,0,1,8))
880 bg_set(0,0,1)
890 /* set background 1
900 bg_fill(1,pat_dat(0,0,1,9))
910 bg_set(1,1,1)
```

```
 920 /* set sprite-plane 0,1,2,3 & background 0,1
 930 i=0:di=1:hr_s1=1:y2=8
 940 while -1
 950     i=i+di
 960     /* background scroll
 970     if x3=511 then x3=0
 980     if y3=511 then y3=0
 990     x3=x3+1
1000     y3=y3+1
1010     bg_scroll(0,x3,y3)
1020     bg_scroll(1,511-x3,511-y3)
1030     /* move sprite
1040     if i<1 then hr_s=0:vr_s=0:hr_s1=1:vr_s1=1:di=1:y2=y2+8
1050     if i>286 then hr_s=1:vr_s=1:hr_s1=0:vr_s1=0:di=-1:y2=y2+8
1060     f=(pi(2)*i)/288*2
1070     f1=(pi(2)*i)/288*4
1080     y1=128-150*sin(f)
1090     y4=128-120*cos(f1)
1100     if y1<0 then vr_s=1:y1=0
1110     if y4<9 then vr_s1=0
1120     if y1>277 then vr_s=0
1130     if y4>247 then vr_s1=1
1140     if y2>271 then y2=0
1150     /* プライオリティ (sprite > background0 > background1)
1160     sp_set(0,i,y2,pat_dat(0,hr_s,1,0),3)
1170     sp_set(1,i,y1,pat_dat(vr_s,0,1,1),3)
1180     sp_set(2,288-i,272-y2,pat_dat(0,hr_s1,2,3),3)
1190     sp_set(3,288-i,248-y4,pat_dat(vr_s1,0,3,4),3)
1200 endwhile
1210 end
1220 func pat_dat(vr,hr,pb,cd)
1230 /* vr = 垂直反転 (0なら反転しない、1なら反転する)
1240 /* hr = 水平反転 (0なら反転しない、1なら反転する)
1250 /* pb = パレットブロック (1～15の値)
1260 /* cd = パターンコード (0～255の値)
1270 return(vr*32768+hr*16384+pb*256+cd)
1280 endfunc
```

●スプライトエディタの起動方法

スプライトエディタの起動方法とコマンドについて説明します。

スプライトエディタは、グラフィックRAMを使用します。グラフィックRAMをRAMディスクとして使用しているときは、RAMディスクのデータをフロッピーディスクなどに退避してから起動してください。

・起動

現在メモリ上にあるプログラムを消去してよいかどうか確認します。よければ、X-BASICの入力待ち状態のときに、次のように入力してください。

```
load　"a:¥ etc ¥ defsptool" ↵
```

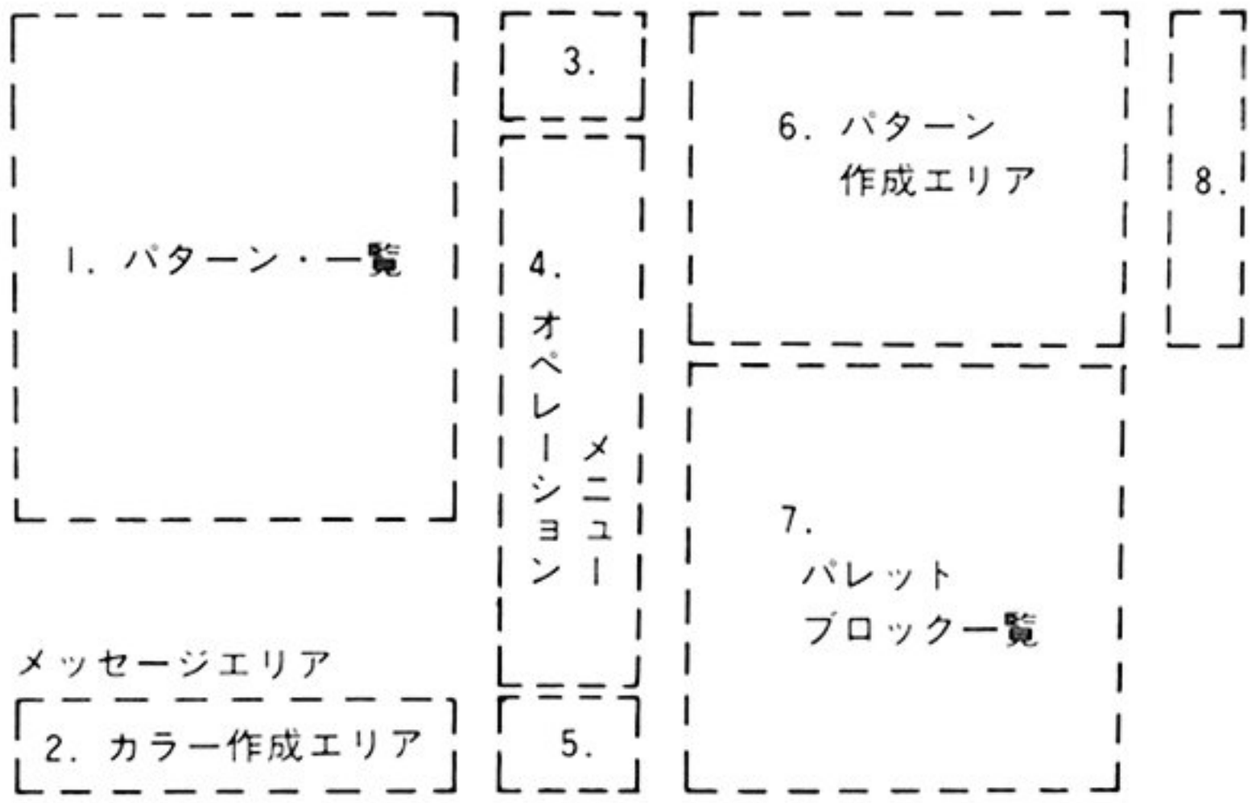
スプライトエディタのプログラムがロードされます。この場合は、ドライブAの「ETC」というディレクトリにある「DEFSPTOOL.BAS」をロードしました。続いて、ファンクションキーのF5キーを押すか、「run」と入力してリターンキーを押してください。

しばらくすると画面に、

説明は必要ですか？ [Y／N]

と表示されます。はじめて使うときは、キーボードからＹと入力してください。ここでＮと入力すると、スプライトエディタが起動します。また、説明の表示が終わると、スプライトエディタが起動します。

・起動画面の各部の名称と機能



1．パターン一覧（左上）：

パターン番号０～127に定義されているパターンを表示します。

2．カラー作成エリア（左下）：

ペンカラーのRGBI各成分が表示されています。RGBI各成分を変更してカラーを作成します。

3．作成中のパターン（中央上）：

表示画面が256×256のときの大きさです。

4．オペレーションメニュー（中央）：

マウスの左ボタンでダブルクリックすることにより各機能を実行します。

5．カラー作成エリア（カラー表示部分）（中央下）：

ペンカラーあるいは作成中の色が表示されます。マウスの左ボタンでダブルクリックするとペンカラーに色を設定します。

6．パターン作成エリア（右上）：

マウスの右ボタンをクリックするとそのドットをセットします。マウスの左ボタンをクリックするとそのドットをリセットします。

7．パレットブロック一覧（右下）：

マウスの左ボタンでクリックするとパレットブロック及びパレットが選択されます。左側の白い印は選択されているパレットブロックを示します。

8．現在選択されているパレットブロック（右端上）：

マウスの左ボタンでクリックするとパレットが選択されます。右側の白い印はペンカラーを示します。

ほとんどの操作は、マウスの左ボタンをクリックすることで実行します。また、スプライトはスプライトエディタ起動時には初期化しません。そのため、スプライトのパターンやパレットを定義していない状態でスプライトエディタを起動するとパターン及びパレットブロック一覧には意味のないパターンやパレットが表示されます。init及びclr機能で初期化してください。

・メニューの機能

各オペレーションは、すべてマウスの左ボタンによるダブルクリックで実行されます。

→	：作成中のパターンを１ドット右へ移動します。
←	：作成中のパターンを１ドット左へ移動します。
↑	：作成中のパターンを１ドット上へ移動します。
↓	：作成中のパターンを１ドット下へ移動します。
90	：作成中のパターンを反時計回りに90度回転します。
180	：作成中のパターンを180度回転します。
↑↓	：作成中のパターンを上下反転します。
←→	：作成中のパターンを左右反転します。
get	：パターンを取り込みます。
put	：パターンを定義します。
fill	：パターンを指定された色で塗りつぶします。
save	：パターンまたはパレットを保存（セーブ）します。
palet	：パターン一覧上のパターンのパレットブロックを変更します。
init	：全パレットを初期化します。
clr	：全パターンをクリアします。
help	：説明を表示します。
end	：スプライトエディタを終了します。

２．１．３　パレットとカラーコード

X-BASICでは、表示される文字の色とグラフィック関数やスプライト関数でドット単位に使用される色をパレットコードで設定します。

また、X-BASICではカラーコードとして０～65535までの値を指定できます。

ここでは、このパレットコードとカラーコードの関係を説明します。

たとえば、グラフィック画面のスクリーンモードが、ドット単位に65536色中最大16色まで表示できるような設定になっていたとします。この場合のパレットコードは０～15までの値をとることができます。もちろん、カラーコードは０～65535までの値を指定できます。つまり、そのスクリーンモードで表示できる最大の色数に応じて指定できるパレットコードの範囲が決まり、その各パレットコードに対して０～65535の範囲でカラーコードが割りふられるわけです。

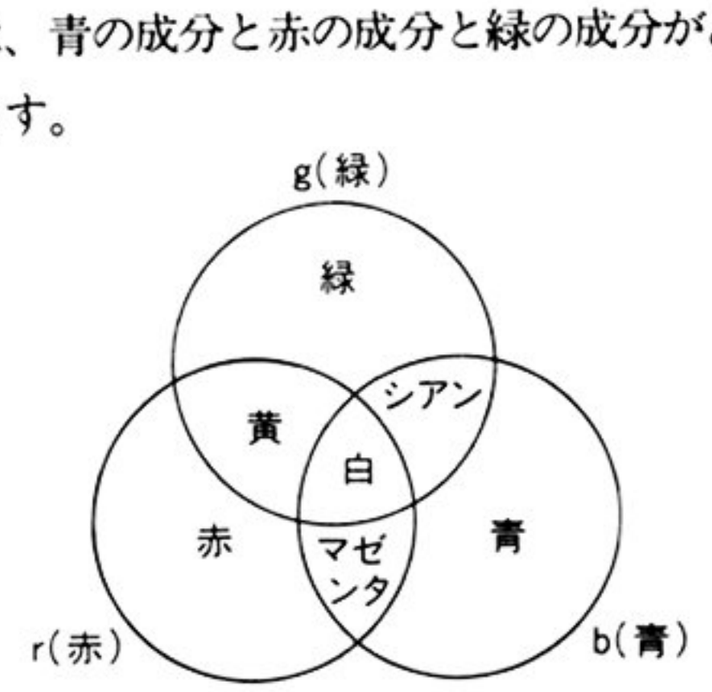
それぞれの画面に応じて指定できるパレットコードは次のようになります。

テキスト画面……パレットは4つ（第4章のcolor、color［ ］命令を参照してください。）
グラフィック画面……色モード（ドット単位の最大表示色）
16色　……パレットコードは0～15
256色　……パレットコードは0～255
(色モード(ドット単位の最大表示色)が65536色のときはパレットコードとカラーコードは1対1に対応します。第4章のpalet関数を参照してください。)
スプライト画面……パターン単位のパレットブロックは1～15
ドット単位のパレットコードは0～15
(スプライトやバックグラウンドで使われるパターンは、パターン単位に1～15までのパレットブロックがまず割りあてられます。さらにそのパターンのドット単位に0～15までのパレットコードが設定されます。第4章のsp_color関数を参照してください。)

つづいて任意の色のカラーコードを求めてみます。

X-BASICには、カラーコードを求める関数としてrgb関数、hsv関数があります。

ご承知のように光の三原色には、青の成分と赤の成分と緑の成分があり、これらの成分の組み合わせで、いろいろな色が表現できます。



ここでは少し暗いマゼンタのカラーコードを求めてみます。
上図をみて、rgb関数を使用すると、

```
print　rgb(25,0,25) ↵
```

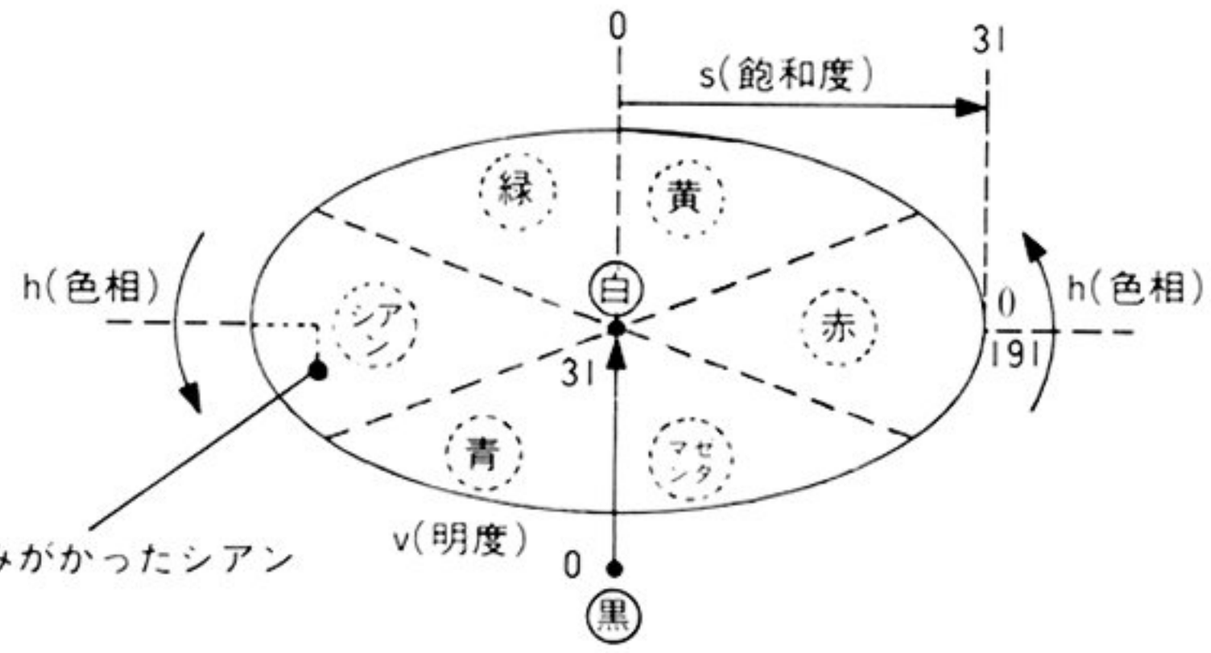
それでは、次のプログラムを入力して実行してみてください。

```
10 /* rgb
20 int ai
30 char ac,bc,cc
40 screen 1,3,1,1
50 apage(0)
60 vpage(1)
70 input "赤の成分を入力してください(0～31)--->";ac
80 input "緑の成分を入力してください(0～31)--->";bc
90 input "青の成分を入力してください(0～31)--->";cc
100 ai=rgb(ac,bc,cc)+1
110 fill(0,0,100,100,ai)
120 end
```

次に少し暗い白みがかったシアンのカラーコードを求めてみます。上図をみて、rgb関数で求めるのはむずかしいようです。

次の図をみて、hsv関数を使用すると、

print　hsv(96,27,26) ↵



それでは、次のプログラムを入力して実行してみてください。

```
 10 /* hsv
 20 int ai
 30 char ac,bc,cc
 40 screen 1,3,1,1
 50 apage(0)
 60 vpage(1)
 70 input "色相を入力してください(0～191)--->";ac
 80 input "飽和度を入力してください(0～31)--->";bc
 90 input "明るさを入力してください(0～31)--->";cc
100 ai=hsv(ac,bc,cc)+1
110 fill(0,0,100,100,ai)
120 end
```

このように、光の三原色であらわされる7色から求めることができるような比較的単純な色のカラーコードは、rgb関数を使って求めることができます。

もっと複雑な色のカラーコードを求める場合はhsv関数を使うことができます。

なお、いずれの場合も算出できるカラーコードは0～65535の偶数値となります。0～65535の奇数値のカラーコードを表現する場合は、rgb関数やhsv関数で返ってきた結果に1を加えてください。rgb関数、hsv関数の詳細については第4章を参照してください。

2．2　FM音源制御

本機ではYM2151というFM音源LSIを内蔵しています。X-BASICでは、このFM音源で音楽を演奏する場合、MMLという言語を使い、音色や音符などの楽譜データを、トラックバッファに書き込みます。トラックバッファは、FM音源のチャンネルとは別に管理されるため、必要に応じて自由に各チャンネルに割り当てることができます。

また、演奏時にチャンネルを指定することもできるので、より効率的な楽譜データの記述や多彩な音楽演奏が可能です。

X-BASICでは、一般に次の手順でFM音源を使用します

1、FM音源を初期化する …… (m_init)
2、MMLデータを格納するトラックバッファを確保する …… (m_alloc)
3、トラックバッファにFM音源の各チャンネルを割り当てる…… (m_assign)
4、演奏のテンポを設定する …… (m_tempo)
5、トラックバッファにMMLデータを格納する …… (m_trk)
6、演奏する…… (m_play)

※FM音源やMMLデータについての詳細は、「Human68kユーザーズマニュアル」を参照してください。

※それぞれのトラックに割り当てるトラックバッファの総合計が、64kBを越えて使用される場合は、CONFIG.SYSファイル内のOPMDRV3.Xの行を書き換える必要があります。
詳しくは「Human68kユーザーズマニュアル」を参照してください。
次に記載されているのは、MMLのデータをX-BASICで演奏するサンプルプログラムです。
各関数についての詳細は、第4章を参照してください。

```
100 /* music sample  "おお、スザンナ！！"     S.C.Foster
110 int ai
120 str as,bs,cs,sl[50]                        /*文字変数の宣言
130 sl="音色変更=[H] 演奏=[P] 停止=[S] 再開=[C] 終了=[E]:"
140 m_init()                                   /*FM音源の初期化
150 m_sysch("OPM")                             /*音源のﾁｬﾝﾈﾙ番号の割当
160 cls
170 M_CHANGE():MML_IN():m_play()
180 repeat
190    locate 0,10:print chr$(5);sl;:input "",cs
200    if cs="H" or cs="h" then {  /*もしHなら音色変更
210    M_CHANGE():MML_IN():m_play()}
220    if cs="P" or cs="p" then {  /*もしPなら演奏
230    m_play()}
240    if cs="S" or cs="s" then {  /*もしSなら停止
250    m_stop()}
260    if cs="C" or cs="c" then {  /*もしCなら再開
270    m_cont()}
280    if cs="E" or cs="e" then {  /*もしEなら終了
290    m_stop()}
300 until cs="e" or cs="E"                 /*cs=Eならrepeat終了
310 end
320 /*---------音色変更----------
330 func int M_CHANGE()
340 for ai=1 to 5
350    m_alloc(ai,2000)                    /*トラックバッファの確保
360    m_assign(ai,ai)                     /*ﾄﾗｯｸﾊﾞｯﾌｧへﾁｬﾝﾈﾙを割当
370 next
380 locate 0,0:print chr$(5);
390 linput "音色番号の指定（１～２００）－－＞？";as
400 print chr$(5);
410 linput "テンポの指定（２０～３００）－－＞？";bs
420 endfunc()
430 /*-----ＭＭＬデータの格納-----
440 func int MML_IN()
450 m_trk(1,"q7 l16 o4 v14 t"+bs+"@"+as)
460 m_trk(2,"q7 l16 o4 v10 t"+bs+"@"+as)
470 m_trk(3,"q7 l16 o4 v10 t"+bs+"@"+as)
480 m_trk(4,"q7 l16 o3 v15 t"+bs+"@45")
490 m_trk(5,"q7 l16 o0 v15 t"+bs+"@47")
500 m_trk(1,"<c&d e8g8g8a8  g8e8c8.d  e8e8d8c8 d4.")
```

```
510 m_trk(2,"e&f  g8<c8c8d8 c8>g8g8.f g8g8f8e8 f4.")
520 m_trk(3,"r8   >c8g8e8g8 c8g8e8g8  c8g8e8g8 >b8<g8d8")
530 m_trk(1,"c&d e8g8g8.a  g8e8c8.d  e8e8d8d8 c4.")
540 m_trk(2,"e&f g8<c8c8.d c8>g8g8.f g8g8f8f8 e4.")
550 m_trk(3,"g8  c8g8e8g8  c8g8e8g8  c8g8d8g8 c8g8c8")
560 m_trk(1,"c&d e8g8g8.a  g8e8c8.d  e8e8d8c8 d4.")
570 m_trk(2,"e&f g8<c8c8.d c8>g8g8.f g8g8f8e8 f4.")
580 m_trk(3,"r8  c8g8e8g8  c8g8e8g8  c8g8e8g8 >b8<g8d8")
590 m_trk(1,"c&d e8g8g8a8  g8.ec8.d  e8e8d8.d c4r4")
600 m_trk(2,"e&f g8<c8c8d8 c8.>gg8.f g8g8f8.f e4r4")
610 m_trk(3,"g8  c8g8e8g8  c8g8e8g8  c8g8d8g8 c8c8d8e8")
620 m_trk(1,"f4f4      a8a4a8   g8.ge8c8 d4r8")
630 m_trk(2,"c4c4      f8f4f8   e8.eg8g8 f4r8")
640 m_trk(3,"f8c8f8c8 c8c8d8f8 e8g8f8e8 d8>b8<c8")
650 m_trk(1,"c&d e8g8g8.a  g8e8c8d8  e8e8d8.d   c4r8")
660 m_trk(2,"e&f g8<c8c8.d c8>g8g8a8 <c8c8>b8.b <c4r8")
670 m_trk(3,"d8  c8g8e8g8  c8g8e8f8  g8g8f8.f   e8g8c8r8")
680 for ai=0 to 23
690    m_trk(4,"c8r8ccr8"):m_trk(5,"r8c8r8c8")
700 next
710 endfunc
```

2．3　音声サンプリング（ADPCM）

本機では、ADPCMという方式で音声をサンプリングして、録音・再生することができます。X-BASICには、そのための関数が用意されています。

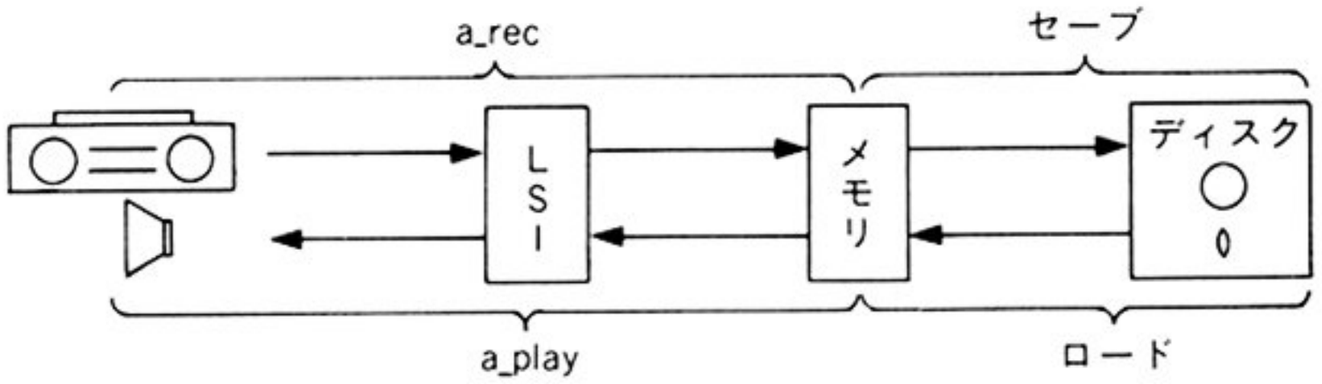
サンプリングして録音した音声はメモリに格納されます。これをファイルに書き出すことで、音声データをフロッピーディスクやハードディスクなどに記録して、あとで再生することができます。

録音は、本機のオーディオ入力端子（AUDIO IN）にラジカセなどのライン出力端子（LINE OUT）を接続して行います。また、再生は、内蔵スピーカから行うか、オーディオ出力端子（AUDIO OUT）に接続したオーディオ機器から行います。

録音・再生できる音声の長さは、サンプリング周波数によって変わります。サンプリング周波数が高ければ、録音するときに一度に消費するメモリが大きくなります。詳細は、第4章を参照してください。

ADPCMによる録音と再生は、一般に次の手順で使用します。

1　PCMデータ（音声）を格納する配列変数を宣言する（dim）
2　録音する（m_pcmrec，a_rec）
(2-1　PCMデータをファイルに保存する)
(2-2　PCMデータをファイルから読み込む)
3　再生する（m_pcmon，a_play）



本機とオーディオ機器を接続して、次のプログラムを実行してください。

```
100 /* M_PCMREC M_PCMON etc sample
110 int  ai,bi,ei
120 str  as,ms,fs,ts[50]
130 ts="3.9k=[0] 5.2k=[1] 7.8k=[2] 10.4k=[3] 15.6k=[4] :"
140 dim char dc(30000)
150 dim str  hs(4)={"3.9kHz","5.2kHz","7.8kHz","10.4kHz","15.6kHz"}
160 screen 2,0,1,0
170 error off
180 /* main
190 repeat
200   cls:input"録音=[R]  再生=[P]  終了=[E]:",ms
210   if ms="r" or ms="R" then { MPCMREC() } else {
220   if ms="p" or ms="P" then { MPCMPLY() }      }
230 until ms="e" or ms="E"
240 end
250 /*-----record-----
260 func MPCMREC()
270 repeat
280   cls
290   print"録音ｻﾝﾌﾟﾘﾝｸﾞ周波数を選択してください(単位Ｈｚ)"
300   print ts;:input"",bi
310 until bi<5 and bi>-1
```

```
320 print:print hs(bi);"で録音します！":print
330 repeat
340   as="":input "[←┘]キーを押すと録音を開始します:",as
350 until as=""
360 print "録音中です！":m_pcmrec(bi,30000,dc)
370 repeat
380   cls
390   print "録音データをセーブします。"
400   locate 4,5:print chr$(5);
410   input"ファイル名:",fs
420   ei=fopen(fs+".PCM","r"):fclose(ei)
430   if ei=-1 then { MPWRITE() } else {
440   print"そのファイル名のデータがあります。"
450   repeat
460     locate 4,13:print chr$(5);
470     input"上書きしますか? ok=[Y] no=[N] end=[E]",as
480   until strchr("YNEyne",asc(as))<>-1
490   if as="Y" or as="y" then { MPWRITE() }}
500 until as="e" or as="E"
510 endfunc
520 func MPWRITE()
530   ai=fopen(fs+".PCM","c"):fwrite(dc,30000,ai)
540   fclose(ai):as="e"
550 endfunc
560 /*----- play -----
570 func MPCMPLY()
580 repeat
590   locate 0,0
600   print "録音データをロードします。"
610   locate 4,5:print chr$(5);
620   input"ファイル名(戻る=[E]):",fs
630   print:if fs="e" or fs="E" then return()
640   ai=fopen(fs+".PCM","r"):fread(dc,30000,ai):fclose(ai)
650   if ai=-1 then print"その名前のデータはありません。"
660 until ai<>-1
670 repeat
680   locate 0,7
690   print"録音時のサンプリング周波数を入力してください"
700   print ts;:input "",bi
710 until bi<5 and bi>-1
720 m_pcmset(0,bi,30000,dc):m_pcmon(0,bi)
730 endfunc
```

2．4　ファイル操作

X-BASICでは、どのようなファイルもすべて、単なるデータの連続として扱います。ファイルをランダムに操作するときは、ポインタという「目印」を使ってデータを読み書きします。

データの読み書きは、1バイト単位で行う場合と、データの型（char、int、float）単位で行う場合があります。また、文字列単位でも読み書きを行えます。この場合はそのデータの型に応じてポインタが移動していきます。

この他にファイルをランダムアクセスして、データの型にかかわりなくファイル内の任意の位置にポインタを移動することもできます。

なお、ファイル操作で使用するドライブ名やファイル名の詳細については別冊の「Human68kユーザーズマニュアル」を参照してください。

ファイル操作の基本的な流れは、次のようになります。

1　ファイルをオープンする（または新規に作成する）(fopen)
(2　ファイルをランダムアクセスしてポインタを移動する（fseek))
3　ファイルからデータを読み出す (fread、fgetc、freads)、またはファイルへデータを書き込む（fwrite、fputc、fwrites)
4　ファイルをクローズする（fclose、fcloseall)

あるファイルを操作するためには、まずそのファイルをオープンしなければなりません。ファイルにはオープンモードと呼ばれるモードがあり、それぞれ次のような意味があります。

クリエイトモード("c")……ファイルを新規に作成します
　　すでに同じ名前のファイルが存在するときはデータがすべて失われます
リードモード("r")……すでに存在するファイルを読み出し専用としてオープンします
ライトモード("w")　……すでに存在するファイルを書き込み専用としてオープンします
リードライトモード("rw")　…すでに存在するファイルを読み書き両用としてオープンします

ファイルをオープンすると、ファイルごとに番号がつけられます。以後の読み書きは、このファイル番号を使って行います。

オープンしたファイルは、処理が終ったら必ずクローズしなければなりません。ファイルをクローズしないと、データが壊れてしまうことがあります。

プログラムでファイルを操作するときには、まず、ファイル番号をしまっておく変数(int型)を宣言します。もし複数のファイルをオープンするのであれば、この変数も複数用意します。また、ファイルを読み書きするときなどに、関数がいろいろな情報を戻り値として返します（ただし、戻り値を無視してもかまわないこともあります)。そのための変数（int型）も宣言します。

```
10 int fp /* ファイル番号
20 int fc /* 戻り値
```

ファイル名があらかじめ決っているときは、それも変数（str型）にしておいたほうがよいでしょう。

```
30 str fname="sample.dat"
```

準備ができたら、ファイルを作成して、１文字書き出し、ファイルをクローズしてみます。

```
40 fp=fopen(fname,"c") /* クリエイトモードでオープン
50 fc=fputc('a',fp)    /* １文字書き出す
60 fc=fclose(fp)       /* ファイルをクローズ
70 end
```

文字はキャラクタコードでファイルに書き込まれます。
では、いま作成したファイルの内容を画面に表示してみましょう。

```
10 int fp /* ファイル番号
20 int fc /* 戻り値
30 int dt /* 読み出したデータ
40 str fname="sample.dat"
50 fp=fopen(fname,"r") /* リードモードでオープン
60 dt=fgetc(fp)        /* １文字読み込む
70 print chr$(dt)      /* 文字の表示
80 fc=fclose(fp)       /* ファイルをクローズ
90 end
```

このファイルに文字を追加して書き出したあと、３番目の文字を読み出して表示します。

```
10 int fp /* ファイル番号
20 int fc /* 戻り値
30 char dt /* データ
40 dim char dts(5)={'b','c','d','e','f'}
50 int i,dp
60 str fname="sample.dat"
70 fp=fopen(fname,"w") /* ライトモードでオープン
80 dp=fseek(fp,0,2)    /* ファイルの最後へポインタを移動
90 for i=0 to 4
100     dt=dts(i)
110     fc=fputc(dt,fp) /* データを書き出す
120 next
130 fc=fclose(fp)       /* ファイルをクローズ
140 fp=fopen(fname,"r") /* リードモードでオープン
150 dp=fseek(fp,2,0)    /* ファイルの先頭からポインタを２つ進める
160 dt=fgetc(fp)        /* １文字読み込む
170 print cha$(dt)      /* 文字の表示
180 fc=fclose(fp)       /* ファイルをクローズ
190 end
```

次のプログラムは、テキストファイルを作成するエディタです。ただし、編集の機能などはありません。また、常にクリエイトモードでファイルをオープンするので、注意してください。

```
10 /* Tiny EDITOR  ted.bas
20 str fname[20]
30 str buff[200]
40 int fp, fc
50 char CR=13, LF=10
60 /* main
70 fninput()
80 fp=fopen(fname,"c")
90 edit()
100 fc=fclose(fp)
110 end
120 /* fninput
130 func fninput()
140   print "Input file name >";
150   linput fname
160 endfunc
170 /* edit
180 func edit()
190   print "Input data(END = /)" : print
200   while buff <> "/"
210       linput ">";buff
220       if buff <> "/" then {
230           fwrites(buff,fp)
240           fputc(CR,fp)
250           fputc(LF,fp)
260       }
270   endwhile
280 endfunc
```

RS-232C（AUX）やプリンタ（PRN）などのデバイスも、ファイル操作と同様に扱うことができます。次のプログラムは、RS-232Cポートからデータを読み込み、それをプリンタに出力するプログラムです。

```
 10 /* RS-232Cからデータを読み、プリンタに出力    rstolp.bas
 20 str fn1="AUX"
 30 str fn2="PRN"
 40 int fp1,fp2,c1
 50 /* main
 60 init()
 70 type()
 80 fclose(fp1)
 90 fclose(fp2)
100 end
110 /* init
120 func init()
130 str dummy
140     print"準備ができたら何かキーを押してください"
150     dummy=inkey$
160     fp1=fopen(fn1,"r")
170     fp2=fopen(fn2,"w")
180 endfunc
190 /* type
200 func type()
210     while c1 <> &H1A
220         c1=fgetc(fp1)
230         fputc(c1,fp2)
240     endwhile
250 endfunc
```

２．５　その他の機能

いままで説明してきたものの他にも、X-BASICには多くの関数が用意されています。
また、子（チャイルド）プロセスによるHuman68kのコマンドの実行もサポートしています。

２．５．１　マウス、ジョイスティック制御

マウスやジョイスティックも、X-BASICで制御することができます。詳細は、第４章を参照してください。

・マウス制御

X-BASICでは、マウスの初期化、マウスカーソルの移動範囲の設定、ボタンの状態の確認などを行えます。次のプログラムでは、マウスを初期化したあと、マウスカーソルの移動範囲を設定しています。

```
 10 /* mouse sample
 20 screen 2,0,1,1
 30 vpage(1)
 40 mouse(0)
 50 msarea(0,0,600,400)
 60 box(0,0,600,400,15,&HFFFF)
 70 mouse(1)
 80 locate 1,5
 90 print "(0,0)<--->(600,400)の範囲でマウスを動かしてください!"
100 end
```

・ジョイスティック制御

X-BASICでは、ジョイスティックのスティックとトリガーの状態を調べることができます。次のプログラムは、ジョイスティックのスティックがどの方向に動かされたかを表示します。

```
 10 /* joy_stick sample
 20 int ai,bi
 30 cls
 40 while bi<>1
 50     ai=stick(1)
 60     locate 10,11
 70     switch ai
 80     case 1:print "left back ( 左後 )     =";ai:break
 90     case 2:print "back ( 後 )            =";ai:break
100     case 3:print "right back ( 右後 )    =";ai:break
110     case 4:print "left ( 左 )           =";ai:break
120     case 6:print "right ( 右 )          =";ai:break
130     case 7:print "left forward ( 左前 )  =";ai:break
140     case 8:print "forward ( 前 )        =";ai:break
150     case 9:print "right forward ( 右前 )=";ai:break
160     default:locate 10,10:print "ジョイスティックを動かしてください!"
170     locate 10,11:print "                                    "
180     endswitch
190 endwhile
200 end
```

2．5．2　文字列操作

X-BASICでは、ファイル操作などで扱う文字列に対して、いろいろな文字列操作を行う関数を数多く用意しています。

- ●指定された文字の種類を調べる関数……（isalnum、isalphaなど）
- ●文字列から指定された文字や文字列を検索する関数……（instr、strchr、strcspnなど）
- ●指定された文字や文字列を変換する関数……（mirror$、strlwr、strnset、toasciiなど）
- ●文字列から指定された文字を区切り文字とした文字列に分解する関数……（strtok）
- ●指定された文字でできた文字列を作成する関数……（space$、string$、strsetなど）
- ●文字列と数値を変換する関数……（asc、atof、bin$、chr$、ecvtなど）

次のプログラムは、ファイルのデータの１文字修正を行うものです。

```
 10 /* ファイルの１文字修正サンプルです
 20 int ai,bi,po
 30 str as,bs,cs,ds,che1,che2,syu1,syu2
 40 screen 2,0,1,1
 50 cls
 60 linput "ファイル名を入力してください--->?";as
 70 ai=fopen(as,"c")
 80 while -1
 90     print "'X68'を入力すると一旦ファイルをクローズします"
100     linput "文字や文字列を入力してください--->?";bs
110     /* chr$(13) = cr,chr$(10) = lf,chr$(26) = ctrl+z
120     if bs="X68" then bs=bs+chr$(26):break
130     bs=bs+chr$(13)+chr$(10)
140     fwrites(bs,ai)
150 endwhile
160 fwrites(bs,ai)
170 fclose(ai)
180 /* update
190 cls
200 linput "修正はありますか？<y/n>--->";che1
210 if che1="n" then end
220 linput "修正後のデータを格納するファイル名を入力してください--->?";bs
230 ai=fopen(as,"rw")
240 bi=fopen(bs,"c")
250 repeat
260     cls
270     freads(cs,ai)
280     print cs
290     length=strlen(cs)
300     linput "この中に修正する文字はありますか？<y/n>--->";che1
310     if che1<>"y" then {
320                       cs=cs+chr$(13)+chr$(10)
330                       fwrites(cs,bi)
340                       continue}
350   while -1
360       linput "修正したい１文字を入力してください--->?";syu1
370       po=strchr(cs,asc(syu1))
380       if po=-1 then print "該当文字がありません!!":continue
390       linput "修正後の１文字を入力してください--->?";syu2  /* 削除の場合
はこの行を取る
400       ds=cs
410       cs=left$(cs,po)+syu2+right$(cs,length-po-1)  /* 削除の場合はこの行
の"+syu2"を取る
420       print "次のように修正します"
430       print cs
440       linput "よろしいですか？<y/n>--->";che2
450       if che2<>"n" then {
460                         cs=cs+chr$(13)+chr$(10)
```

```
470                                 fwrites(cs,bi)
480                                 break}
490          cs=ds
500          print cs
510      endwhile
520 until feof(ai)=-1
530 cls
540 print "修正前のファイルの内容と修正後のファイルの内容を表示します"
550 fseek(ai,0,0)
560 fseek(bi,0,0)
570 print "修正前のファイルの内容を表示します!"
580 repeat
590      freads(cs,ai)
600      print cs
610 until feof(ai)=-1
620 print "修正後のファイルの内容を表示します!"
630 repeat
640          freads(ds,bi)
650          print ds
660 until feof(bi)=-1
670 fcloseall()
680 end
```

２．５．３　子プロセス

X-BASICには、Human68kのコマンドを実行するコマンドがあります。そのため、フロッピーディスクのフォーマットやコピー、スクリーンエディタEDを使ったテキストファイルの編集などを、X-BASICを終了することなく実行できます。

入力待ちの状態で”！”に続けてHuman68kのコマンドを入力してリターンキーを押します。ドライブＢのフロッピーディスクをフォーマットするのであれば、次のように入力します。

```
!format  b: ↵
```

Human68kでのコマンドの実行が終了すると、

BASICへ戻ります、何かキーを押してください

というメッセージが表示されます。キーを押すと、X-BASICに戻ります。

第 3 章　X-BASIC の文法

この章は、まだ説明していない X-BASIC の文法についてまとめてあります。

3．1　X-BASIC で使用できる文字と特殊記号

X-BASIC で使用できる文字は、英文字(大文字、小文字)、数字、カタカナ、ひらがな、特殊記号、漢字、そして制御文字から構成されています。文字についての詳細は、資料編の「キャラクタコード表」を参照してください。

半角文字と全角文字

通常 X-BASIC では、カーソル 1 つ分のサイズをもつ半角文字を扱います。キーボードからコマンドを与えたり、プログラムを記述するには半角の英数字を使い、カタカナやひらがな(全角)、漢字(全角)は文字列データとして使うものと考えてください。

特殊記号

使用できる文字の中でも、特殊記号と呼ばれる文字はそれぞれ特別な意味を持っています。特殊記号の使い方については、第 4 章「リファレンス」で必要なたびごとに説明しますが、ここではそれぞれの意味をまとめて説明します。なお、特殊記号のうち、演算子については、3．4「式と演算」、書式指定文字については第 4 章の print 命令で説明します。

(空白)　　プログラム中の命令語と命令語、変数名などの区切りとして使い、またプログラムを見やすくするために入れることができます。ただし、予約語(資料編「予約語一覧」参照)の中には空白を入れることはできません。また、文字定数(3．3「定数」参照)の中の空白は文字としての意味を持っています。文中の空白はプログラムの実行には影響を与えませんが、コンピュータのメモリには空白も記憶されます。

！(感嘆符)　　チャイルドプロセスコマンド。第 4 章 child 命令参照。
例)　!dir

”(引用符)　　文字列を囲みます。
例)　print　”ABC”

#(番号記号)　　float 型属性文字。
例)　print　3#／4#

’(アポストロフィ)　　文字データを囲んで char 型の数値データとします。

例）　char　a='A'

( )括弧

演算の優先順位を変えたり、関数の引数リストをくくったりするのにつかいます。

例）　print (1 + 1) * 3

a=pow(2, 3)

, (カンマ)

データを複数個指定するときに、その区切りとして使います。print 命令で使われた場合には、8桁ごとに区切って表示されます。

例）　print　"a","b","c"

－(ハイフン)

数値の範囲を指定するときに使います。

例）　list 命令で表示する行の範囲を指定するとき、

list　100－200

とすれば、100行から200行までが表示されます。

／＊(斜線とアスタリスク)

注釈文の先頭に置きます。第4章 rem 命令参照。

例）　／＊コノギョウ　ハ　コメントデス

：(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= 2 + 3 :print　a

；(セミコロン)

データを複数個指定するときに、その区切りとして使います。print 命令で使われた場合には、データは前のデータに続いて表示されます。

例）　print　"a" ; "b" ; "c"

また func～endfunc で定義される関数名に続く引数リスト内の変数名に特別に型を指定するのに使います。

例）　func　foo(i;float　, j;int)

? (疑問符)

print 命令の代わりに使います。

例）　?　25＊12

[ ]角括弧

str 型変数の文字バッファサイズを指定します。第4章 str 命令参照。

例）　dim　str　a(10)[20]

{ }大括弧

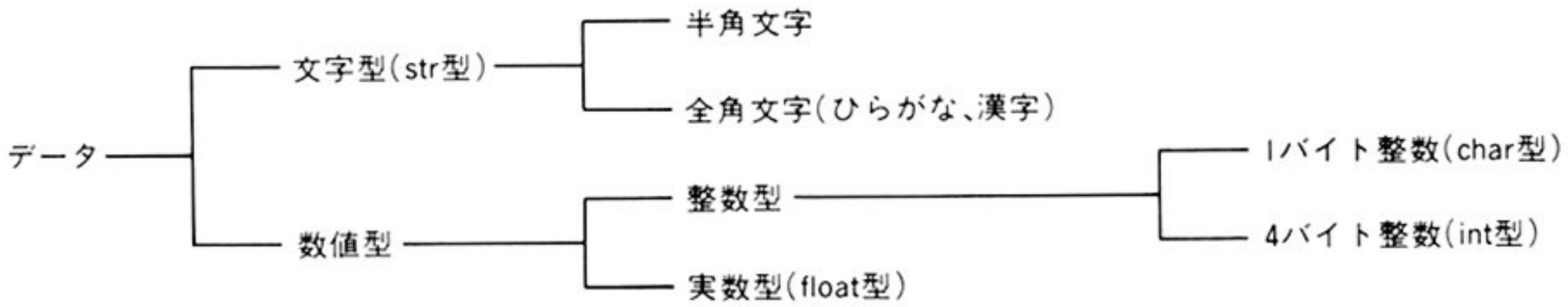
複数行にわたってデータや if 文を記述するのに使います。

例）　10　dim　str　c(1)={"a",

20　　"b"}

3．2　X-BASICの変数や定数などで扱うデータ

X-BASICの変数や定数などで扱うデータは、次のように分類することができます。



文字列(str型)

255文字以内の文字をつないだデータです。文字列には、X-BASICで使用できる文字ならばどれでも使うことができます。ただし、文字列を算術演算(3．4「式と演算」参照)に使うことはできません。

数値

数値は算術演算を行うことができるデータです。数値には整数型と実数型とがあり、数値の取る値はそれぞれで異なります。

整数(int型)

4バイト(32ビット)で表現される符号付き整数で、-2^{31}から$2^{31}-1$(−2,147,483,648から2,147,483,647)までの値を使用できます。整数型の数値を使う演算は、実行時間が短くてすみます。X-BASICは整数型を基本にしているので、特にデータ型を指定しないときは整数(int)型になります。

整数(char型)

1バイト(8ビット)で表現される符号無しの整数で、0から255の値を使用できます。キャラクタコードなど、1バイト単位のデータ制御に最も有効です。

実数(float型)

実数はすべて倍精度実数で、8バイト(64ビット)で表現されます。

1ビットの符号部(正、負)と11ビットの指数部及び52ビットの仮数部から構成されています。この結果表現できる絶対値は、1.1125369292536E−308から3.5953862697246E+308となります。

また、数値型データは必要に応じて、その型を他の型に変換することができます。ただし、文字型と数値型の間でこの変換を行うにはval、str$、ascおよびchr$の各関数を使います(第4章参照)。

型の変換は、次の規則にしたがって行われます。

- 数値を異なるデータ型の数値変数に代入する場合、数値は代入先の数値変数のデータ型に変換されます。

●データ型の異なる数値を使った演算では、データ型の低い方が高い方のデータ型に変換されて演算が行われます。
●論理演算の場合、扱われる数値はすべて整数型に変換され、結果は整数(int 型)で得られます。
●実数が整数に変換される場合は、小数点以下は切り捨てられます。このとき、切り捨てた結果が整数型で扱える範囲を越えていればエラーとなります。

３．３　定数

定数とは、それぞれに固有の値を持ったデータのことです。定数の書き方は、３．２で述べたデータ型によって異なります。

３．３．１　文字定数

文字定数は、文字列を引用符(”)で囲んで表します。文字列の長さは255文字以内でなければなりません。また、数字を引用符で囲むと文字列とみなされ、算術演算に用いることはできません。文字列は常にキャラクタコードが０で終わります。はじめのキャラクタコードが０のものはヌルストリング(null 文字、無文字列)として扱います。

例) "THANK YOU"
"3.14159265"

３．３．２　整数型定数

整数型の定数には、以下の４つの形式があります。

●10進形式

int 型なら－2,147,483,648から2,147,483,647まで、char 型なら０から255までの整数です。負の整数の場合、数値の前に必ずマイナス符号(－)をつけなければなりませんが、プラス符号(＋)は省略できます。

例) 0
－123

● 8 進形式

先頭に&O(または&o)をつけて８進数(０～７)を並べたものです。範囲は int 型では&O ０～&O37777777777までです。&O37777777777～&O20000000000までは負の数で、－１から－2,147,483,648に対応します。

例)&O12345……10進数では5349
&O37777777777……10進数では－１

char 型では、&O ０～&O377が０～255に対応します。

●16進形式

先頭に&H(または&h)をつけて16進数(0～7、A～Fまたはa～f)を並べたものです。範囲はint型では&H 0～&HFFFFFFFFまでです。&HFFFFFFFF～&H80000000までは負の数で、－1から－2,147,483,648に対応します。

例)&H100……10進数では256

&HFFFFFFFF……10進数では－1

char型では、&H0～&HFFが0～255に対応します。

● 2進形式

先頭に&B（または&b）をつけて2進数（0か1）を並べたものです。範囲はint型では&B 0～&B11111111111111111111111111111111(32桁)までです。&B11111111111111111111111111111111～&B10000000000000000000000000000000までは負の数で、－1から－2,147,483,648に対応します。

例)&B100000000……10進数では256

&B1111...(32桁)……10進数では－1

char型では、&B 0～&B11111111が0～255に対応します。

＊注意：2進形式、8進形式、16進形式で入力された数値は、printやlprint命令で出力すると、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bin$、oct$、hex$を使って文字列として出力してください。

●その他

’(アポストロフィ)で囲まれた1文字の文字データは、そのキャラクタコードを値としたchar型の数値となります。

例)'A'……char型の65

3．3．3　実数型(浮動小数型)定数

実数型の定数には、以下の3つの形式があります。

- int型の範囲を超える数値
- 14桁以内の数値とE(またはe)を使った指数表示の組み合わせ(Eまたはeの範囲は±308)
- 最後に.（ピリオド)か#(番号記号)が付けられた数値

例)3000000000

1235E-20

5.

426#

3．4　式と演算

式とは、演算(計算)の文法を示すものであり、定数や変数を演算子(計算に使う特殊記号のこと)で結んだものです。式の演算結果は1個の数値または文字列になりますから、単に数値や文字だけのものも式と呼びます。

例）"BASIC"

```
10+3/5
2
a+b/c-d
```

X-BASICの演算は、次の5つに分けられます。

- **算術演算**
- **関係演算**
- **論理演算**
- **関数演算**
- **文字列演算**

以下にこれらの演算について説明します。

3．4．1　算術演算

算術演算子には次のようなものがあります。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。

演算子	演算	例
−	マイナス符号	−x
*、/	乗算、除算	x * y、x/y
¥、mod	整数の除算、剰余	x ¥ y、x mod y
+、−	加算、減算	x+y、x−y
shr	右シフト	x shr y
shl	左シフト	x shl y

演算の優先順序も、この順番になります。順序を変更したいときには括弧( )を使えば、括弧内の演算子は他の演算子より先に実行されます。

参考

●整数での演算

変数や定数のときと同様に、演算でもデータ型を意識しないと予想外の結果になることがあります。X-BASICで扱う数は、データ型が指定されていないとすべて整数(int型)と見なされますから、

```
? 1/3
```

の答えは0となります。これを実数の演算として扱うには、

```
? 1.0/3(または？ 1#/3)
```

あるいは

```
? 1/3.0(または？ 1/3#)
```

のようにしなければなりません。

●￥と mod

整数の除算では、／のかわりに￥を使って割り切れなかった場合の答の小数点以下を切り捨てることができます。扱う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切り捨てられます。また、余りを求めたいときには mod を使います。この場合も、扱う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切り捨てられます。

```
例)print  10 ￥ 3
      3  (10/3＝3……1)
   print  10 ￥ 2.5
      5  (10/2＝5……0)
   print  13.3 mod 4
      1  (13/4＝3……1)
```

● 0での除算

数値を0で割ったときには、"0による除算が行われました"というエラーメッセージが表示され、プログラムを中断します。0に対して負のべき乗を行った場合も同様です。

●オーバーフロー(桁あふれ)

代入や演算の結果が、その変数の型の中で表現できる範囲を越えた場合、桁あふれがおこります。この場合、"オーバーフローしました"というエラーメッセージが表示され、プログラムを中断します。

● shr と shl(シフト演算)

普通の算術では使われませんが、2進数を扱うコンピュータの世界では多用されます。整数値を2進形式のビット列と考え、そのビット列をある回数右や左に移動(シフト)させることをシフト演算といいます。

```
例)? 6 shr 2
     1  (&b00000110→&b00000001)
   ? 3 shl 3
    24  (&b00000011→&b00011000)
```

シフト後の桁には、０が入ります。

３．４．２　関係演算

関係演算子は、２つの数値または２つの文字列を比較するときに用います。関係演算の結果は、真(－１)、偽(０)で返され、if 命令などでプログラムの流れを変えるのに用いられます。以下に、関係演算子の意味と例をあげます。

関係演算子	内容	例
＝	左辺と右辺が等しい	x＝y
<>	左辺と右辺が等しくない	x<>y
<	右辺より左辺が小さい	x<y
>	右辺より左辺が大きい	x>y
<=	右辺より左辺が小さいかまたは等しい	x<=y
>=	右辺より左辺が大きいかまたは等しい	x>=y

例)if　x=0　then　print "A"

(もしｘが0だったら、"A"を表示)

if　a+b<>0　then　x=a+b

(もしａ+ｂが０でなかったら、ｘにａ+ｂの値を代入せよ)

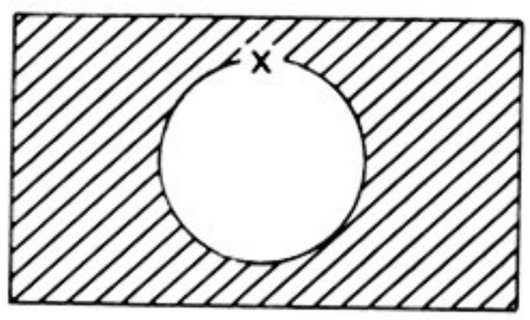
詳しくは、第４章を参照してください。

３．４．３　論理演算

論理演算子は複数の条件を調べたり、ビット操作やブール演算を行ったりするのに用います。論理演算の結果は、ビットごとに０または１で返されます。各論理演算の内容を以下に示します。

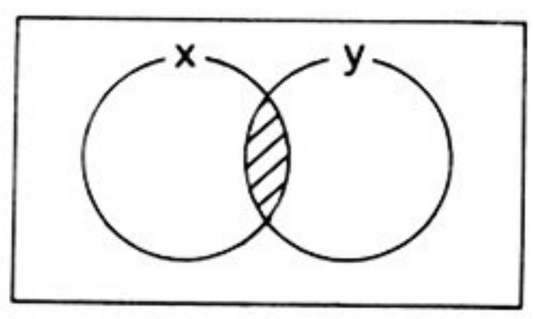
not(否定)

x	not x
1	0
0	1



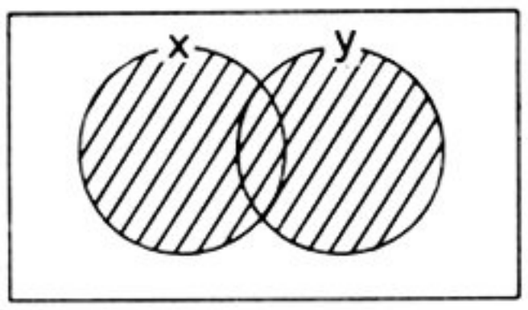
and(論理積)

x	y	x and y
1	1	1
1	0	0
0	1	0
0	0	0



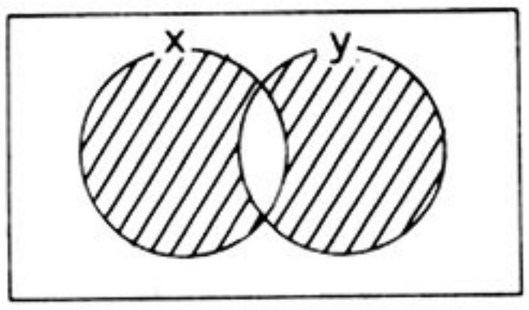
or(論理和)

x	y	x or y
1	1	1
1	0	1
0	1	1
0	0	0



xor(排他的論理和)

x	y	x xor y
1	1	0
1	0	1
0	1	1
0	0	0



if 命令では、これらの論理演算子を使って、複数の条件を判断することができます。

例)if　x<0　or　99<x　then　x=0

(x が負の数または99より大きければ、x に 0 を代入する)

if　0<x　and　x<100　then　x=0

(x が正の数でしかも100より小さければ、x に 0 を代入する)

if　not(a=0)　then　x=0

(a が 0 でなければ、x に 0 を代入する)

詳しくは、第 4 章を参照してください。

用語の説明

●ビット

コンピュータの扱うデータの最小単位。1 ビットで 0 か 1 かの 2 つの状態を表します。

●ビット操作

ビットの並びの中の任意のビットを ON/OFF する(1 にしたり 0 にしたりする)ような操作をいいます。

●ブール演算

イギリスの数学者ブールが唱えた論理演算。普通の演算が数値を扱うのに対して、真か偽かを扱う演算です。BASIC では、真は 0 以外、偽は 0 という値を取ります。

3．4．4　関数演算

関数とは、基本的には指定されたデータに対して、ある決まった処理を行うもので、その処理の結果を返すものです。

X-BASIC の関数には、sin や sqr などの数値を扱うものや、chr$ や left$ などの文字列を扱うものなどがあります。また、最初から定義されたものではなく、プログラマが自由に関数を定義するための命令(func～endfunc)も持っています。

これらの関数については、第4章で詳しく説明します。

参考

X-BASICの命令は、大きく3つに分けることができます。

●コマンド　list、newなどがその代表で、主にプログラムを書いたり編集したりするための命令です。ダイレクトモードでのみ使用する命令です。

●ステートメント　print、ifなどがその代表で、主にプログラム中で核となる命令です。

●関数　sin、cosなどがその典型で、与えられたデータに対して、何らかの処理を行いその結果を返すものです。

なお、関数名と引数をくくる左括弧との間はあけないでください。

3．4．5　文字列演算

●文字列の連結

文字列は、演算子＋を使ってつなげることができます。たとえば、

```
10  str  a="ハルガ":str  b="キタ":str  c
20  c=a+b
30  print  c
40  end
```

このプログラムを実行すると、画面には「ハルガキタ」と表示されます。

●文字列の比較

文字列も、数値の比較に使われるものと全く同じ関係演算子(＝、＜、＞、＜＞、＜＝、＞＝)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先頭から1文字ずつ文字を比較します。両者が全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文字でも違った場合は、その文字のキャラクタコード(資料編「キャラクタコード表」参照)の大きい方の文字列が大きいと判断されます。また、文字列の一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いと判断されます。

なお、文字列の比較においては、空白なども意味を持ちますから注意してください。

```
例)"AB"      <   "BC"
   "BC"      >   "AAA"
   "CAT"     <   "CATS"
   "PEN"     >   "  PEN"
   "cm"      >   "CM"
   "BASIC"   =   "BASIC"
```

文字列を比較することによって、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かえたりするのに利用できます。

3．4．6　演算の優先順位

演算は次の順位によって行われます。

1)　カッコで囲まれたもの
2)　関数
3)　マイナス符号(－)
4)　＊、/
5)　¥
6)　mod
7)　＋、－
8)　shr、shl
9)　関係演算子(＝、＜、＞など)
10)　not
11)　and
12)　or
13)　xor

第4章　リファレンス

書　式　命令や関数の書き方です。

〈　〉で囲まれた項目は、必要とされる引数やデータです。

〔　〕で囲まれた項目は、省略可能です。

...　は、同様に繰り返し、引数やデータを与えることを表わします。

これ以外の文字(予約語、{　}、[　]などの記号類)はそのまま入力しなければなりません。

コマンドやステートメントなどは、原則として命令名とデータの間には、空白を入れてください。また、関数名と引数をくくる左括弧の間には、空白を入れないでください。

省略形　省略できる命令や関数の書き方です。なお、引数、括弧などは省略できません。

引　数　関数が原則としてどういう型の引数を必要とするかを表わしています。

数値型としてint、char、float、文字型としてstrの4つのデータ型がありますが、複数の種類の引数を取る関数については、それぞれのデータ型に応じて括弧でくくってまとめてあります。数値型であれば何でもよい場合は数値型と表記しています。また、引数が特に変数名や配列名(配列の場合は次元も明記しています)でなければならない場合(定数、式などが引数とならない)は、そのように明記しています。

戻り値　すべての関数について、その戻り値の型が書いてあります。int、char、float、str、および意味を持たない戻り値(戻り値が存在しない場合も含む)としてvoid、の5つの型があります。

機　能　各命令や関数の機能および、引数の意味などを説明しています。

➜　関連した命令や、文法など参考になるものを挙げてあります。

用　例　簡単なサンプルプログラムや用例です。

ABS

標準関数

書　式　abs(n)

引　数　float

戻り値　float

機　能　数値nの絶対値を返します。

用　例

```
10 /* abs sample
20 int ai,bi,ci,di
30 float af
40 screen 1,3,1,1
50 vpage(1)
60 line(0,255,511,255,65535,&HFFFF)
70 for ai=1 to 5
80     for bi=0 to 511
90         af=(2*pi()*bi)/512*ai
100        ci=255-abs(sin(af)*200)
110        di=rnd()*65535
120        pset(bi,ci,di)
130    next
140 next
150 end
```

APAGE

GRAPH

書　式　apage(pa)

引　数　char

戻り値　void

機　能　グラフィック画面のアクティブページpa(グラフィック画面で実際に読み書きするページ)を指定します。ページ番号は0～3で、最大値はグラフィック画面の実画面サイズ及び色モードによります。

グラフィック画面の実画面サイズ及び色モードによるアクティブページpaの値の範囲

1024×1024(16色モード) ……0
512×512(16色モード) ……0～3
512×512(256色モード)……0、1
512×512(65536色モード)……0

➜　home、screen、vpage、window

用　例

```
10 /* apage sample
 20 int ai,bi,ci
 30 screen 1,1,1,1
 40 vpage(&B1111)
 50 for ai=0 to 8
 60 /* active page(0)
 70      apage(0)
 80      bi=rnd()*15
 90      fill(50,50,200,200,bi)
100      symbol(210,50,"ｱｸﾃｨﾌﾞﾍﾟｰｼﾞ0",1,1,1,bi,0)
110      pause()
120 /* active page(1)
130      apage(1)
140      bi=rnd()*15
150      fill(100,100,250,250,bi)
160      symbol(260,100,"ｱｸﾃｨﾌﾞﾍﾟｰｼﾞ1",1,1,1,bi,0)
170      pause()
180 /* active page(2)
190      apage(2)
200      bi=rnd()*15
210      fill(150,150,300,300,bi)
220      symbol(310,150,"ｱｸﾃｨﾌﾞﾍﾟｰｼﾞ2",1,1,1,bi,0)
230      pause()
240 /* active page(3)
250      apage(3)
260      bi=rnd()*15
270      fill(200,200,350,350,bi)
280      symbol(360,200,"ｱｸﾃｨﾌﾞﾍﾟｰｼﾞ3",1,1,1,bi,0)
290      pause()
300 next
310 end
320 func pause()
330 for ci=0 to 1000
340 next
350 endfunc
```

ASC

標準関数

書　式　asc(st)

引　数　str

戻り値　int

機　能　文字列stの先頭文字のキャラクタコードを返します。
stがnull文字の場合は0を返します。文字とキャラクタコードの対応については、「キャラクタコード表」を参照してください。

➡　chr$、「キャラクタコード表」

用　例

```
10 /* asc sample
20 str as,bs
30 as="a"
40 bs="abcd"
50 print asc("axyz")
60 print asc(as)
70 print asc(bs)
80 print asc(as)*asc(bs)
90 end
```

ATAN

標準関数

書　式　atan(n)

引　数　float

戻り値　float

機　能　数値nの逆正接(アークタンジェント)を返します。$-\pi/2$から$\pi/2$までの値を返します。

用　例

```
10 /* atan sample
20 int ai,bi,ci,di
30 float af
40 screen 1,3,1,1
50 vpage(1)
60 line(0,255,511,255,65535,&HFFFF)
70 for ai=1 to 3
80     for bi=0 to 511
90         af=bi-255
100        ci=255-atan(af)*ai*40
110        di=rnd()*65535
120        pset(bi,ci,di)
130    next
140 next
150 end
```

ATOF

標準関数

書　式　atof(st)

引　数　str

戻り値　float

機　能　文字列stをfloat型の数値に変換します。stは、float型の数値に変換できる文字(+、−、0～9、．、e、E)の連続です。もしfloat型の数値として判定できない文字があると、そこで変換するのをやめます。

stの先頭の空白文字やタブ文字は無視されます。また、小数点の前後には少なくとも1つ以上の数字が必要です。さらに、指数表現では、eあるいはEの後ろに少なくとも1つ以上の数字が必要です。

stがnull文字の場合は、エラーとなります。

変換できる文字列stの書式は次の通りです。

〔whitespace〕〔sign〕〔digits〕〔.digits〕〔e〔またはE〕〔sign〕digits〕

whitespace ……空白文字またはタブ文字で無視されます。

sign ……”＋”または”－”

digits ……1つ以上の10進数で、小数点の前に何もなければ、小数点の後ろに少なくとも1つの数字が必要です。

指数は"e"または"E"と符号をともなった10進数で構成されます。

➜　ecvt、fcvt、gcvt

用　例

```
 10 /* atof sample
 20 float af,bf,cf
 30 str as,bs,cs[50]
 40 as="3000000000000"
 50 bs="-2.51235e-2"
 60 cs="0.5102123542131543512312313453543543513213"
 70 af=atof(as)
 80 bf=atof(bs)
 90 cf=atof(cs)
100 print af*2.5E+030#
110 print bs+"+"+cs+"=";bf+cf
120 print bs+"*"+cs+"=";bf*cf
130 end
```

ATOI

標準関数

書　式　atoi(st)

引　数　str

戻り値　int

機　能　文字列stをint型の数値に変換します。stは、int型の数値に変換できる文字(0～9、+、－)の連続です。もしint型の数値として判定できない文字があると、そこで変換するのをやめます。

stの先頭の空白文字やタブ文字は無視されます。また、stには少なくとも1つ以上の数字が必要です。

stがnull文字の場合は、エラーとなります。

変換できる文字列stの書式は次のとおりです。

〔whitespace〕〔sign〕digits

whitespace ……空白文字またはタブ文字で無視されます。

sign ……"+" または "－"

digits ……1つ以上の10進数

➡　itoa

用　例

```
10 /* atoi sample
20 int ai,bi
30 str as,bs
40 as="300000"
50 bs="-512"
60 ai=atoi(as)
70 bi=atoi(bs)
80 print as+"+("+bs+")=";ai+bi
90 print as+"*("+bs+")=";ai*bi
100 end
```

AUTO

コマンド

書　式　auto　〔〈行番号〉〕〔,〈増分〉〕

省略形　a .

機　能　プログラム行の先頭に行番号を自動的に発生させます。
〈行番号〉も〈増分〉も省略できますが、この場合は自動的に10が採用されます。たとえば、

auto	行番号は10を先頭に20、30…と付けられます(auto　10,10と同じ)。
auto 100	行番号は100を先頭に110、120…と付けられます。
auto　,5	行番号は10を先頭に15、20…と付けられます。

プログラム中にすでに存在する行と同じ行番号が発生した場合には、その行の内容がすべて表示されます。このとき、何も入力しないでリターンキーを押せば、その行の内容は変わりません。
auto機能を解除するときは、[BREAK]キー(または[CTRL]+[C]キー)を押します。解除したときのプログラム行はメモリに保存されません。

A_PLAY

書　式　a_play(na,sf,md[, lng])

引　数　数値型一次元配列名(na),char(sf,md,lng)

戻り値　void

機　能　指定された配列naのPCMデータを再生します。

na ……PCMデータを格納している数値型一次元配列名

sf ……サンプリング周波数

0……3.9KHz

1……5.2KHz

2……7.8KHz

3……10.4KHz

4……15.6KHz

音声を再生する場合、原則としてa_rec関数で録音したときのサンプリング周波数sfを使用してください。それ以外を使用すると、録音した音声を正常に再生できません。

md ……音声出力モード(本機後面のオーディオ出力端子からの音声信号出力モード)

0……出力カット

1……左出力

2……右出力

3……ステレオ出力

lng ……再生する配列naの添字0からの長さ

lngのとりうる値は、0から、配列名naで宣言された配列の添字＋1までです。

lngを省略すると、すべてのPCMデータを再生します。

用　例

```
10 /* a_play sample
20 int ai
30 char ac
40 dim char aca(1023)
50 /* read BEEP.SYS file
60 ai=fopen("a:¥SYS¥BEEP.SYS","rw")
70     fread(aca,1024,ai)
80 fclose(ai)
90 /* beep
100 repeat
110     print "ｂｅｅｐ音をステレオ出力で発生します!"
120     print "サンプリング周波数 3.9KHz---0"
130     print "サンプリング周波数 5.2KHz---1"
140     print "サンプリング周波数 7.8KHz---2"
150     print "サンプリング周波数10.4KHz---3"
160     print "サンプリング周波数15.6KHz---4"
170     input "サンプリング周波数を入力してください(0---4) ";ac
180 until ac<5
190 a_play(aca,ac,3)
200 end
```

A_REC

AUDIO

書　式　a_rec(na,sf〔,lng〕)

引　数　数値型一次元配列名(na),char(sf,lng)

戻り値　void

機　能　指定された配列naにPCMデータを録音します。

na　……PCMデータを格納する数値型一次元配列名

sf　……サンプリング周波数

0……3.9KHz(1秒間に消費するメモリサイズ1950バイト)

1……5.2KHz(1秒間に消費するメモリサイズ2600バイト)

2……7.8KHz(1秒間に消費するメモリサイズ3900バイト)

3……10.4KHz(1秒間に消費するメモリサイズ5200バイト)

4……15.6KHz(1秒間に消費するメモリサイズ7800バイト)

lng ……録音する配列naの添字0からの長さ

lngのとりうる値は、0から、配列名naで宣言された配列の添字＋1までです。

lngを省略すると、すべてのPCMデータを録音します。

用　例

```
 10 /* a_rec sample
 20 int ai
 30 char ac
 40 str as
 50 dim char aca(65535)
 60 screen 2,0,1,0
 70 /* record
 80 repeat
 90     print "録音するためのサンプリング周波数"
100     print "  (0=3.9KHz,1=5.2KHz,2=7.8KHz,3=10.4KHz,4=15.6KHz)"
110     input "録音するためのサンプリング周波数を入力してください(0---4)--->";
ac
120 until ac<5
130 /*
140 print "この例ではサンプリング周波数によって次のような録音時間になります"
150 switch ac
160     case 0:print "                サンプリング周波数( 3.9KHz)--->約33秒間":break
170     case 1:print "                サンプリング周波数( 5.2KHz)--->約25秒間":break
180     case 2:print "                サンプリング周波数( 7.8KHz)--->約16秒間":break
190     case 3:print "                サンプリング周波数(10.4KHz)--->約12秒間":break
200     case 4:print "                サンプリング周波数(15.6KHz)--->約 8秒間"
210 endswitch
220 while as<>"y"
230 print "録音しますので'y'ｷｰを押してください"
240 as=inkey$
250 endwhile
260 print "録音中です"
270 a_rec(aca,ac,65536)
280 as=""
290 print "<test.dat>というファイルに録音データをセーブします"
300 print "(すでにある<test.dat>というファイルは消去されますので注意してく
ださい)"
310 print "セーブしてもよい場合は'y'ｷｰを押してください"
```

```
  320 as=inkey$
  330 if as="y" then {
  340                    ai=fopen("a:¥test.dat","c")
  350                        fwrite(aca,65536,ai)
  360                    fclose(ai)
  370                    }
  380 /* play
  390 repeat
  400     print "再生するためのサンプリング周波数"
  410     print "  (0=3.9KHz,1=5.2KHz,2=7.8KHz,3=10.4KHz,4=15.6KHz)"
  420     input "再生するためのサンプリング周波数を入力してください(0---4)--->";
ac
  430 until ac<5
  440 /*
  450 print "ここでは、65536をさきほどの録音時間とみなして、再生させたい時間に相
当する値(バイト数)を入力します"
  460 repeat
  470     input "再生するためのバイト数を入力してください(0---65536)--->";ai
  480 until ai>-1 or ai<65537
  490 switch ac
  500     case 0:print "再生するためのサンプリング周波数とバイト数--->";" 3.9KHz
";ai:break
  510     case 1:print "再生するためのサンプリング周波数とバイト数--->";" 5.2KHz
";ai:break
  520     case 2:print "再生するためのサンプリング周波数とバイト数--->";" 7.8KHz
";ai:break
  530     case 3:print "再生するためのサンプリング周波数とバイト数--->";"10.4KHz
";ai:break
  540     case 4:print "再生するためのサンプリング周波数とバイト数--->";"15.6KHz
";ai
  550 endswitch
  560 as=""
  570 while as<>"y"
  580     print "再生しますので'y'ｷｰを押してください"
  590     as=inkey$
  600 endwhile
  610 print "ステレオ出力で再生します"
  620 a_play(aca,ac,3,ai)
  630 end
```

BEEP

ステートメント

書　式　beep

機　能　ビープ音を鳴らします。print　chr$(7)と同じ機能です。

用　例

```
 10 /* beep sample
 20 str as
 30 cls
 40 while as<>"b"
 50     locate 10,10
 60     print "'b'ｷｰを押すとビープ音が発生します!"
 70     as=inkey$
 80 endwhile
 90 beep
100 end
```

BG_FILL

SPRITE

書　式　bg_fill(pg,pd)

引　数　char(pg),int(pd)

戻り値　int

機　能　指定されたテキストページpgを、指定されたパターンデータpdで埋めつくします。

pg　……テキストページ(0、1)

pd　……パターンデータ(0～&HCFFF)

パターンデータpdは、次のようなビット構成になっています。(ビット12、13は通常0になります。)

ビット15……垂直反転(0：通常、1：垂直反転)

ビット14……水平反転(0：通常、1：水平反転)

ビット8～11…パレットブロックpb(1～15)

ビット0～7…パターンコードcd(0～255)

垂直反転を指定すると、実際に定義されたパターンが上下反転して表示されます。また水平反転を指定すると、実際に定義されたパターンが左右反転して表示されます。

パレットブロックpbは各パターンに対して指定するパレットで、詳細についてはsp_color関数を参照してください。

テキストページ1しか使用しない場合のパターンコードcdは16×16ドットパターンの場合、0から191までの値をとることができます。テキストページ0、1とも使用する場合のパターンコードcdは16×16ドットパターンの場合0から127までの値をとることができます。テキストページ0、1とも使用しない(バックグラウンドを使用しない)場合のパターンコードcdは16×16ドットパターンの場合、0から255までの値をとることができます。

BG_GET

書　式　bg_get(pg,x,y)

引　数　char

戻り値　int

機　能　指定されたテキストページpgの指定された座標からパターンデータを読み出します。

pg　……テキストページ(0、1)
x……X座標(0～63)
y……Y座標(0～63)

読み出されたパターンデータは、次のようなビット構成になっています。(ビット12、13は通常0になります。)

ビット15……垂直反転(0：通常、1：垂直反転)
ビット14……水平反転(0：通常、1：水平反転)
ビット8～11…パレットブロックpb(1～15)
ビット0～7…パターンコードcd(0～255)

垂直反転になっている場合は、実際に定義されたパターンに対して上下が反転して表示されていることを意味します。水平反転になっている場合は、実際に定義されたパターンに対して左右が反転して表示されていることを意味します。
パレットブロックpbは各パターンに対して割りあてられているパレットで、1から15の値を返します。詳細については、sp_color関数を参照してください。
テキストページ1しか使用しない場合のパターンコードcdは16×16ドットパターンの場合、0から191までの値を返します。テキストページ0、1とも使用する場合のパターンコードcdは16×16ドットパターンの場合、0から127までの値を返します。テキストページ0、1とも使用しない（バックグラウンドを使用しない）場合のパターンコードcdは16×16ドットパターンの場合0から255までの値を返します。

➡　bg_put

BG_PUT

SPRITE

書　式　bg_put(pg,x,y,pd)

引　数　char(pg,x,y),int(pd)

戻り値　int

機　能　指定されたテキストページpgの指定された座標に指定されたパターンデータpdを設定します。

pg　……テキストページ(0、1)
x……X座標(0～63)
y……Y座標(0～63)
pd　……パターンデータ(0～&HCFFF)

パターンデータpdは、次のようなビット構成になっています。(ビット12、13は通常0になります。)

ビット15……垂直反転(0：通常、1：垂直反転)
ビット14……水平反転(0：通常、1：水平反転)
ビット8～11…パレットブロックpb(1～15)
ビット0～7…パターンコードcd(0～255)

垂直反転を指定すると、実際に定義されたパターンが上下反転して表示されます。また、水平反転を指定すると、実際に定義されたパターンが左右反転して表示されます。
パレットブロックpbは各パターンに対して指定するパレットで、詳細については、sp_color関数を参照してください。
テキストページ1しか使用しない場合のパターンコードcdは16×16ドットパターンの場合、0から191までの値をとることができます。テキストページ0、1とも使用する場合のパターンコードcdは16×16ドットパターンの場合、0から127までの値をとることができます。テキストページ0、1とも使用しない(バックグラウンドを使用しない)場合のパターンコードcdは16×16ドットパターンの場合、0から255までの値をとることができます。

➜　bg_get

BG_SCROLL

SPRITE

書　式　bg_scroll(b[,x][,y])

引　数　char(b),int(x,y)

戻り値　int

機　能　バックグラウンドbのスクロール座標を設定します。

b……バックグラウンド(0、1)

x……スクロールX座標(0～511または1023)

y……スクロールY座標(0～511または1023)

なお、表示画面サイズが256×256の場合は、X、Y座標は0から511までの値をとることができます。表示画面サイズが、512×512の場合はX、Y座標は、0から1023までの値をとることができます。ただし、表示画面サイズが512×512の場合はバックグラウンド1は表示されませんので注意してください。

➜　bg_stat

BG_SET

SPRITE

書　式　bg_set(b[,pg][,md])

引　数　char

戻り値　int

機　能　バックグラウンドbへのテキストページpgの割りあてと表示の設定を行います。

b……バックグラウンド(0、1)

pg ……テキストページ(0、1)

md ……バックグラウンドの表示(0＝表示しない、1＝表示する)

なお、表示画面サイズが512×512の場合のみ、バックグラウンド1は表示されませんので注意してください。表示画面サイズが256×256の場合は、バックグラウンド0、1とも表示できます。

➜　bg_stat

BG_STAT

SPRITE

書　式　bg_stat(b,md)

引　数　char

戻り値　int

機　能　指定されたバックグラウンドbの状態を読み出します。

b ……バックグラウンド(0、1)

md ……モード(0～3)

モードmdには読み出す内容を指定します。

0＝X座標（0～511または1023)

1＝Y座標（0～511または1023)

2＝テキストページ（0、1）

3＝バックグラウンドの表示(0＝表示されていない、1＝表示されている)

なお、表示画面サイズが256×256の場合は、X、Y座標には0から511までの範囲の値を返します。表示画面サイズが512×512の場合は、X、Y座標には0から1023までの範囲の値を返します。ただし、表示画面サイズが512×512の場合は、バックグラウンド1は表示されませんので注意してください。

➜　bg_scroll、bg_set

BIN$

書　式　bin$ (i)

引　数　int

戻り値　str

機　能　整数値iを２進表現の文字列に変換し返します。なお、上位の "0" は取り除かれます。

➜　hex$、oct$

用　例

```
10 /* bin$ sample
20 int ai,bi
30 ai=-255
40 bi=2100542300
50 print bin$(ai)
60 print bin$(bi)
70 print bin$(ai)+bin$(bi)
80 end
```

BOX

GRAPH

書　式　box(x1,y1,x2,y2,p[,ls])

引　数　int

戻り値　void

機　能　グラフィック画面上にボックス(四角形)を描きます。

x1 ……始点X座標
y1 ……始点Y座標
x2 ……終点X座標
y2 ……終点Y座標
p……パレットコード(色)
ls……ラインスタイル

x1、y1、x2、y2は、－32768から32767までの値をとることができます。クリッピングエリアについてはwindow関数で指定された範囲になります。また、パレットコードpは0から65535までの値をとることができ、その最大値は、グラフィック画面の実画面サイズ及び色モードによります。

ラインスタイルlsは0～65535の値をとり、これを16ビットのビットパターンと見なし、実線(&HFFFF)や点線(&HAAAA)などを指定することができます。

用　例

```
 10 /* box sample
 20 int ai,bi,ci,di,ei,fi,gi
 30 screen 1,3,1,1
 40 vpage(1)
 50 for ai=0 to 100
 60     bi=rand() mod 512
 70     ci=rand() mod 512
 80     di=rand() mod 512
 90     ei=rand() mod 512
100     fi=rnd()*65535
110     gi=65535
120     box(bi,ci,di,ei,fi,gi)
130 next
140 wipe()
150 for ai=0 to 100
160     bi=rand() mod 512
170     ci=rand() mod 512
180     di=rand() mod 512
190     ei=rand() mod 512
200     fi=rnd()*65535
210     gi=rnd()*65535
220     box(bi,ci,di,ei,fi,gi)
230 next
240 end
```

BREAK

ステートメント

書　式　break

機　能　最も内側のfor、repeat、switch、while文から抜け出します。

➡　continue

用　例

```
 10 /* break sample
 20 int ai
 30 ai=1
 40 switch ai
 50     case 0:print "0":break
 60     case 1:print "1":break
 70     case 2:print "2":break
 80     default:print "?"
 90 endswitch
100 for ai=0 to 10
110     if ai=5 then print "5":break
120     print ai
130 next
140 while ai<>0
150     if ai=2 then print "2":break
160     ai=ai-1
170     print ai
180 endwhile
190 print "complete!"
200 end
```

CHAR

ステートメント

書　式　char　〈変数名〉〔＝〈定数式〉〕〔,...〕

機　能　変数を整数(char)型として宣言します。同時に値を代入することもできます。
原則として、変数宣言はメインプログラムの先頭で行ってください。また、同じ名前の変数をメインプログラム中で再宣言することはできません。

➡　float、int、str

用　例

```
10 /* char sample
20 char ac=41
30 char bc='A'
40 char cc=68,dc='b'
50 print ac
60 print bc
70 print cc,dc
80 end
```

CHDIR

コマンド

書　式　chdir　"<ディレクトリ名>"

機　能　ディレクトリを変更します。<ディレクトリ名>の詳細については、「Human68kユーザーズマニュアル」を参照してください。

用　例

```
chdir "a:¥bin"
```

CHDRV

コマンド

書　式　chdrv　"<ドライブ名>"

機　能　カレントドライブを変更します。<ドライブ名>の詳細については、「Human68kユーザーズマニュアル」を参照してください。

用　例

```
chdrv "b:"
```

CHILD

コマンド

書　式　!<Human68kのコマンド>

機　能　チャイルドプロセスを実行します。
たとえば、"command. x"をチャイルドプロセスで実行するためには、同じディレクトリに"command. x"というファイルがあり、さらに"command. x"をロードするだけのエリアが、フリーエリアとは別にメモリ上になければなりません。
なお、ここで使用できるHuman68kのコマンドについての詳細は、「Human68kユーザーズマニュアル」を参照してください。

用　例

```
!dir *.*
!ed test.dat
!format b:
```

CHR$

標準関数

書　式　chr$ (ch)

引　数　char

戻り値　str

機　能　キャラクタコードchの文字を返します。ch＝0のときはnull文字を返します。
chは0～255の範囲でなければなりません。また、その文字がコントロールキャラクタとして定義されていると、基本的には文字は表示されず、そのコントロールコードの持つ機能が実行されます。ただし中には実行されないものもあります。文字とキャラクタコードの対応については、「キャラクタコード表」を参照してください。

➜　asc、「キャラクタコード表」

用　例

```
10 /* chr$ sample
20 char ac,bc
30 ac=&H41
40 bc=&H31
50 print chr$(13)
60 print chr$(ac)
70 print chr$(bc)
80 print chr$(ac)+chr$(bc)
90 end
```

CIRCLE

GRAPH

書　式　circle(x,y,r,p〔,s〕〔,e〕〔,hv〕)

引　数　int

戻り値　void

機　能　グラフィック画面上に円や楕円や扇型、円弧を描きます。

x……円の中心のX座標

y……円の中心のY座標

r……円の半径

p……パレットコード(色)

s……開始角度(度)

e……終了角度(度)

hv……偏平率

x、yは、－32768から32767までの値をとり、半径rは0から32767までの値をとることができます。クリッピングエリアについてはwindow関数で指定された範囲になります。また、パレットコードpは0から65535までの値をとることができ、その最大値はグラフィック画面の実画面サイズおよび色モードによります。

開始角度s、終了角度eは通常0°から360°の値をとります。－1°から－360°の値を使用すると扇型を描くことができます。

偏平率hvは0～65535までの値をとることができます。

hv＜256なら横長楕円

hv＝256なら縦横比1の真円(ただし768×512以外の表示画面サイズでは真円となりません)

hv＞256なら縦長楕円

用　例

```
10 /* circle sample
20 int ai,bi,ci,di,ei,fi,gi
30 screen 2,0,1,1
40 vpage(1)
50 for ai=0 to 100
60     bi=rand() mod 768
70     ci=rand() mod 512
80     di=rand() mod 200
90     ei=ai mod 16
100    circle(bi,ci,di,ei,0,360,256)
110 next
120 wipe()
130 screen 1,3,1,1
140 for ai=0 to 500
150    bi=500-ai
160    ci=hsv(ai mod 192,31,31)
170    circle(256,256,100,ci,0,360,bi)
180 next
190 wipe()
```

```
200 locate 32,15:print "弧"
210 locate 32,30:print "扇"
220 for ai=0 to 17
230     bi=ai*5
240     ci=90-bi:di=-90+bi
250     ei=90+bi:fi=-90-bi
260     gi=hsv(ai mod 192,31,31)
270     circle(256,200,150,gi,ci,ei,400)
280     circle(256,439,150,gi,di,fi,400)
290 next
300 end
```

CLEAR

コマンド

書　式　clear

省略形　cl.

機　能　変数とスタックを初期化します。プログラム中では使えません。

➜　free

CLS

ステートメント

書　式　cls

機　能　テキスト画面のみを消去します。

用　例

```
10 /* cls sample
20 int ai,bi
30 screen 2,0,1,1
40 vpage(1)
50 for ai=0 to 50
60     fill(rnd()*767,rnd()*511,rnd()*767,rnd()*511,rnd()*16)
70 next
80 for ai=0 to 3
90     for bi=0 to 200
100        print "X68000      ";
110    next
120    for bi=0 to 10000
130    next
140    cls
150 next
160 for ai=0 to 10000
170 next
180 wipe()
190 end
```

COLOR

ステートメント

書　式　color　〈属性〉

省略形　col.

機　能　テキスト画面の文字の属性を設定します。

文字の属性には、4つのパレットと、強調、リバースがあり、0～15の値を使用することで設定できます。デフォルトとしては次のような属性がそれぞれ0から15に対応します。

0	ノーマル	黒	8	リバース	黒
1	ノーマル	シアン	9	リバース	シアン
2	ノーマル	黄色	10	リバース	黄色
3	ノーマル	白	11	リバース	白
4	ノーマル強調	黒	12	リバース強調	黒
5	ノーマル強調	シアン	13	リバース強調	シアン
6	ノーマル強調	黄色	14	リバース強調	黄色
7	ノーマル強調	白	15	リバース強調	白

文字の属性の0と4と8と12は、パレット(p0)に対応します。また、文字の属性の1と5と9と13は、パレット(p1)に対応し、2と6と10と14は、パレット(p2)に対応し、3と7と11と15は、パレット(p3)に対応します。

各パレットを変更するには、color [ ]命令を使用してください。

用　例

```
10 /* color sample
20 int ai
30 for ai=0 to 15
40     color ai:print "personal workstation X68000   "
50 next
60 color 3
70 end
```

COLOR [ ]

ステートメント

書　式　color [c0,c1,c2,c3]

省略形　col.

機　能　テキスト画面の各パレットにカラーコードを割りあてます。

c0 ……属性0、4、8、12のパレット(p0)に割りあてるカラーコード

c1 ……属性1、5、9、13のパレット(p1)に割りあてるカラーコード

c2 ……属性2、6、10、14のパレット(p2)に割りあてるカラーコード

c3 ……属性3、7、11、15のパレット(p3)に割りあてるカラーコード

カラーコードは0～65535の値を使用できます。属性についてはcolor命令を参照してください。

用　例

```
10 /* color [] sample
20 int ai,bi,ci,di,ei,fi
30 cls
40 for ai=0 to 15
50     color ai
60     print "personal workstation X68000    ";
70     print "personal workstation X68000    "
80 next
90 color 3
100 print "途中でBREAKしたときは、'run 210-'を入力したあと、リターンキーを押して実行してください"
110 for ai=0 to 31
120     for bi=0 to 31
130         for ci=0 to 191
140             di=hsv(ci,bi,ai)+1
150             ei=hsv(ci,31-bi,31-ai)+1
160             fi=hsv(ci,bi,31-ai)+1
170             color [di,ei,fi,65535]
180         next
190     next
200 next
210 /* color default value
220 color [0,rgb(0,31,31)+1,rgb(31,31,0)+1,65535]
230 color 3
240 end
```

CONSOLE

ステートメント

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〉

省略形　cons.

機　能　〈スクロール開始行〉で指定した行から〈スクロール行数〉分をテキスト画面のスクロールエリアとします。

表示画面サイズが256×256以外のテキスト画面の最下行は、ファンクションキーの表示エリアになります。〈スクロール開始行〉のとりうる範囲は0から31まで、〈スクロール行数〉のとりうる範囲は1から32までです。〈スクロール開始行〉と〈スクロール行数〉と〈ファンクションキー表示〉の値をすべて加えた値は最大32までとなります。ただし、〈スクロール開始行〉で31を指定したり、あるいは〈スクロール行数〉で32を指定した場合は、〈ファンクションキー表示〉は0に設定してください。

表示画面サイズが256×256のテキスト画面の場合は、常時〈ファンクションキー表示〉は0に設定し、〈スクロール開始行〉は0から15まで、〈スクロール行数〉は1から16までの範囲の値をとることができます。〈スクロール開始行〉と〈スクロール行数〉と〈ファンクションキー表示〉の値をすべて加えた値は最大16までとなります。

なお、〈ファンクションキー表示〉は0ならファンクションキー表示OFF、1ならファンクションキー表示ONを意味します。

console命令を実行すると、実行後、カーソルは表示画面の左上に移動し、またwidth命令を実行すると、console　0,31,1に再設定されます。なお、console　,,0はファンクションキーのみ表示OFFとし、console　,,1はファンクションキーのみ表示ONにします。この場合、スクロールエリアは変化しません。

用　例

```
 10 /* console sample
 20 int ai,bi,ci,di,ei,fi
 30 screen 2,0,1,1
 40 vpage(1)
 50 locate 0,3
 60 print "表示画面サイズは768*512（横96桁＊縦32行）です"
 70 while ai<>-1
 80      input "スクロール開始行を0から31の範囲で入力してください---> bi";bi
 90      if bi<0 or bi>31 then print "biの値を設定し直してください":continue
100      input "スクロール行数を1から32の範囲で入力してください---> ci";ci
110      if ci<1 or ci>32 or bi+ci>32 then print "bi,ciの値を設定し直してくださ
い":continue
120      if bi+ci=32 then print "ファンクションキーは表示できません":di=0:break
130      while ei<>-1
140           input "ファンクションキーを表示する場合は1を表示しない場合は0を入
力してください---> di";di
150           if di<0 or di>1 then print "diの値を設定し直してください":continue
160           ei=-1
170      endwhile
180      ai=-1
190 endwhile
200 pause()
210 cls
```

```
220 print "console ";bi;",";ci;",";di;"を設定します"
230 console bi,ci,di
240 pause()
250 cls
260 for ai=0 to 500
270     print "X68000       ";"X68000HD   ";
280 next
290 pause()
300 console 0,31,1
310 cls
320 end
330 func pause()
340 for fi=0 to 10000
350 next
360 endfunc
```

CONT

コマンド

書　式　cont

省略形　c.

機　能　BREAK キー(または CTRL + C キー)入力、またはstop命令によって停止したプログラムの実行を再開します。実行停止中にlist命令やprint命令などを実行することはできますが、プログラムそのものの変更を行ったときは、cont命令による実行の再開はできません。

➡　stop

CONTINUE

ステートメント

書　式　continue

機　能　for、repeat、while文の次のループを強制的に開始します。

➜　break

用　例

```
 10 /* continue sample
 20 int ai
 30 print "繰り返して、continueを使用しない場合"
 40 for ai=0 to 5
 50     if ai=2 then print "X68000   "
 60     print "ai=";ai;
 70     print "personal workstation X68000   "
 80 next
 90 print "同じ繰り返して、continueを使用した場合"
100 for ai=0 to 5
110     if ai=2 then print "X68000   ":continue
120     print "ai=";ai;
130     print "personal workstation X68000   "
140 next
150 ai=-1
160 print "0から20までの偶数値、奇数値を表示します!"
170 while ai<>20
180     ai=ai+1
190     if (ai mod 2)=0 then print ai;:continue
200     print ai
210 endwhile
220 end
```

CONTRAST

GRAPH

書　式　contrast(ch)

引　数　char

戻り値　void

機　能　画面全体のコントラストを指定します。

ch ……0(文字やグラフィック、スプライトの表示が見えない)～15(文字やグラフィック、スプライトの表示がよく見える)

用　例

```
 10 /* contrast sample
 20 int ai,bi,ci
 30 screen 1,3,1,1
 40 vpage(1)
 50 symbol(200,200,"SHARP",2,2,2,rgb(31,0,0)+1,0)
 60 for ai=0 to 15
 70     bi=15-ai
 80     contrast(bi)
 90     for ci=0 to 1000
100     next
110 next
120 symbol(100,100,"X68000    ",3,3,2,rgb(0,0,31)+1,0)
130 for ai=0 to 15
140     contrast(ai)
150     for bi=0 to 1000
160     next
170 next
180 end
```

COS

標準関数

書　式　cos(n)

引　数　float

戻り値　float

機　能　数値n(ラジアン)の余弦(コサイン)を返します。

用　例

```
 10 /* cos sample
 20 int ai,bi,ci,di
 30 float af
 40 screen 1,3,1,1
 50 vpage(1)
 60 line(0,255,511,255,65535,&HFFFF)
 70 for ai=1 to 3
 80     for bi=0 to 511
 90         af=(2*pi()*bi)/512*ai
100         ci=255-cos(af)*200
110         di=rnd()*65535
120         pset(bi,ci,di)
130     next
140 next
150 end
```

CRT

IMAGE

書　式　crt(cd)

引　数　char

戻り値　void

機　能　テレビ・ビデオ画面とコンピュータ画面を切り換えます。

cd ……切り換えコード(0～3)

コードcdの値と意味は次のとおりです。

0＝テレビ・ビデオ画面の映像を表示

1＝コンピュータ画面を表示

2＝テレビ・ビデオ映像のコントラストを下げてスーパーインポーズ表示

3＝テレビ・ビデオ映像のコントラストを通常にしてスーパーインポーズ表示

実行前にscreen命令であらかじめ表示画面サイズを256×256または、512×512に、ディスプレイ解像度をlow(標準解像度)に設定します。カラーイメージユニットのテロッパ機能を使って、コンピュータ画像やスーパーインポーズ画像をビデオに録画する場合、crt(3)にあらかじめ設定してください。

なお、この関数を実行するにはカラーイメージユニットが必要です。

CSRLIN

システム変数

書　式　csrlin

戻り値　int

機　能　テキスト画面の現在のカーソルのy座標を返します。代入することはできません。

用　例

```
10 /* csrlin sample
20 print "X68000   "
30 print "X68000   "
40 print csrlin
50 end
```

DATE$

システム変数

書　式　date$

引　数　str

戻り値　str

機　能　現在の日付を文字列として返します。代入することによって日付を設定できます。

用　例

```
10 /* date$ sample
20 str as,bs
30 as=date$
40 print as
50 bs=as
60 date$=bs
70 print date$
80 end
```

DAY$

システム変数

書　式　day$

戻り値　str

機　能　現在の曜日を文字列として返します。代入することはできません。

用　例

```
10 /* day$ sample
20 str as
30 as=day$
40 print as
50 end
```

DELETE

コマンド

書　式　delete　〈行番号の範囲〉

省略形　del.

機　能　指定されたプログラム行を削除します。
行番号の指定方法によって、さまざまな形の削除が行えます。

delete　10	行番号10の行を削除します。
delete　−100	プログラムの先頭から行番号100までの行を削除します。
delete　200−	行番号200からプログラムの最後までの行を削除します。

➜　list、new、renum

DIM

ステートメント

書　式

dim 〔〈型宣言〉〕 〈変数名〉(〈添字の最大値〉〔,〈添字の最大値〉...〕)〔[〈文字バッファのサイズ〉]〕〔={〈代入定数式データリスト〉}〕〔,〈変数名〉...〕

機　能

配列変数を使うことを宣言します。

dim文中で代入するデータは、={　}で囲めば複数行に分けて記述できます。ただし、複数行にわけて記述した場合は、そのプログラムをsave@命令でセーブできても、load@命令でロードできませんので注意してください。

同じデータ型の配列はカンマで区切れば後に続けて１行に入力可能な文字数の範囲内で宣言できます。なお、〈型宣言〉を省略すると、int型の配列として扱われます。

設定できる添字の最小値は０、最大値はフリーエリアが許す範囲です。

〈文字バッファのサイズ〉は１～255の範囲で設定でき、str型の配列の１要素のとりえる文字列の最大の長さを指定します。省略した場合は32文字となります。

原則として、配列変数の宣言はメインプログラムの先頭で行ってください。同じ名前の配列を再宣言することはできません。

用　例

```
10 /* dim sample
20 int ai,bi
30 dim int aia(2)={15,-845,55555555}
40 dim int bia(1,1)={465,5953,
50                   -536,222}
60 dim char aca(2,2,2)
70 dim float afa(5)
80 dim str asa(1)={"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa",
90                 "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"}
100 dim str bsa(1)[50]={"bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb",
110                     "asdsadafasd"}
120 for ai=0 to 2
130     print "aia(";ai;")=";aia(ai)
140 next
150 for ai=0 to 1
160     for bi=0 to 1
170     print "bia(";ai;",";bi;")=";bia(ai,bi)
180     next
190 next
200 print "asa(0)=";asa(0)
210 print "bsa(0)=";bsa(0)
220 end
```

DSKF

標準関数

書　式　dskf(d)

引　数　char

戻り値　int

機　能　指定されたドライブdの空きディスク容量を返します。エラーの場合は－1を返します。

d……0　カレントドライブ
　　1　ドライブA
　　2　ドライブB
　　・　・
　　・　・

用　例

```
10 /* dskf sample
20 int ai
30 ai=dskf(2)
40 print ai
50 end
```

ECVT

標準関数

書　式　ecvt(f,i1,i2,i3)

引　数　float(f), int(i1), int型変数名(i2,i3)

戻り値　str

機　能　浮動小数値fを文字列に変換し、その文字列を返します。文字列として返されるのは数値のみです。

f ……文字列に変換したい浮動小数値

i1 ……文字列の桁数

i2 ……小数点の位置を返します。正の場合は、小数点はi2＋1の位置にあります。0または負の場合は、小数点はi2－1の位置にあります。

i3 ……符号を表わす値を返します。正の場合は0、負の場合は1になります。

➡ atof、fcvt、gcvt

用　例

```
10 /* ecvt sample
20 int ai,bi
30 float af,bf,cf
40 af=25000000000000#
50 bf=-2.6546681E+205#
60 cf=-5.2365426E-018#
70 print str$(af)
80 print ecvt(af,20,ai,bi),ai,bi
90 print str$(bf)
100 print ecvt(bf,20,ai,bi),ai,bi
110 print str$(cf)
120 print ecvt(cf,20,ai,bi),ai,bi
130 end
```

END

ステートメント

書　式　end

機　能　プログラムを終了します。プログラム内にいくつ置いてもかまいません。ただし、最初に実行したend命令よりあとの命令文は実行されません。

用　例

```
10 /* end sample
20 test()
30 end
40 func test()
50 print "X68000    "
60 endfunc
```

ERRNO

システム変数

書　式　errno

戻り値　int

機　能　error offのモードの場合のみ有効で、外部関数でエラーが発生しているかどうかを知りたいときに使います。代入することはできません。戻り値としては次のような値を返します。

0　……エラーが発生していないことを示します
正の整数　……外部関数上でエラーが発生していることを示します
負の整数　……システム上でエラーが発生していることを示します

ERROR　ON/OFF

ステートメント

書　式　error　onまたはerror　off

機　能　error　onにすると、外部関数でエラーが発生したときにエラーメッセージを表示して実行を停止するモードになります。error　offにすると外部関数でエラーが発生してもエラーメッセージを表示せず、次のステートメントに実行が進められます。run命令実行直後はerror　onのモードになっています。なお、error　offのモードが有効なときは、プログラムでerror　offを実行している間のみです。プログラムが終了して、"Ok"が表示されるとerror　onのモードになります。

用　例

```
 10 /* error on/off sample
 20 int ai
 30 cls
 40 print "error offモードにします"
 50 error off
 60 line(0,0,1111111100,100,8888888)
 70 print "X68000   "
 80 for ai=0 to 10000
 90 next
100 cls
110 print "error on モードにします"
120 error on
130 line(0,0,1111111100,100,8888888)
140 print "X68000   "
150 end
```

EXIT()

ステートメント

書　式　exit()

機　能　現在オープンされているファイルをクローズするなどの処理をした後、親プロセスに戻ります。

用　例

```
10 /* exit() sample
20 print "X68000   "
30 exit()
40 end
```

EXP

標準関数

書　式　exp(n)

引　数　float

戻り値　float

機　能　自然対数の底 "e" のn乗を返します。

用　例

```
10 /* exp sample
20 float af,bf
30 af=5.3#
40 bf=-2.8#
50 print exp(af)
60 print exp(bf)
70 end
```

FCLOSE

標準関数

書　式　fclose(fp)

引　数　int

戻り値　int

機　能　fopen関数によってオープンされたファイルをクローズします。クローズした場合、戻り値として0を返します。エラーの場合は－1を返します。

fp　……ファイル番号

用　例

```
 10 /* fclose sample
 20 int ai,bi
 30 str as
 40 linput "ファイル名を入力してください--->?";as
 50 ai=fopen(as,"c")
 60 bi=fopen(as,"r")
 70 print "指定ファイルがクローズされます"
 80 fclose(ai)
 90 fclose(bi)
100 end
```

FCLOSEALL

標準関数

書　式　fcloseall( )

戻り値　int

機　能　オープン中のすべてのファイルをクローズします。戻り値として、クローズしたファイルの個数を返します。エラーの場合は－１を返します。

用　例

```
 10 /* fcloseall() sample
 20 int ai,bi,ci
 30 str as
 40 linput "ファイル名を入力してください--->?";as
 50 ai=fopen(as,"c")
 60 bi=fopen(as,"r")
 70 print "指定ファイルがクローズされます"
 80 ci=fcloseall()
 90 print "クローズされたファイルは";ci;"個です"
100 end
```

FCVT

標準関数

書　式　fcvt(f, i1,i2,i3)

引　数　float(f),int(i1) , int型変数名(i2,i3)

戻り値　str

機　能　浮動小数値fを文字列に変換し、その文字列を返します。文字列として返されるのは数値のみです。

f ……文字列に変換したい浮動小数値

i1　……文字列の小数点以下の桁数

i2　……小数点の位置を返します。正の場合は、小数点はi２＋１の位置にあります。０または負の場合は、小数点はi２－１の位置にあります。

i3　……符号を表わす値を返します。正の場合は０、負の場合は１になります。

➡　atof、ecvt、gcvt

用　例

```
 10 /* fcvt sample
 20 int ai,bi
 30 float af,bf,cf
 40 af=25000000000000#
 50 bf=-2.6546681E+205#
 60 cf=-5.2365426E-018#
 70 print str$(af)
 80 print fcvt(af,20,ai,bi),ai,bi
 90 print str$(bf)
100 print fcvt(bf,20,ai,bi),ai,bi
110 print str$(cf)
120 print fcvt(cf,20,ai,bi),ai,bi
130 end
```

FDELETE

標準関数

書　式　fdelete(fn)

引　数　str

戻り値　int

機　能　ファイルfnを削除します。削除できた場合、戻り値として0を返します。エラーの場合は－1を返します。

fn ……ファイル名

用　例

```
 10 /* fdelete sample
 20 int ai
 30 str as
 40 cls
 50 linput "削除したいファイル名を入力してください--->?";as
 60 print "ファイル("+as+")を削除します"
 70 ai=fdelete(as)
 80 if ai<>0 then print "ファイルの削除ができません"
 90 print "ファイルの削除ができました"
100 end
```

FEOF

標準関数

書　式　feof(fp)

引　数　int

戻り値　int

機　能　指定されたファイルの終了チェックを行います。ファイルが終わりなら－1、終わっていなければ0を返します。エラーの場合も－1を返します。

fp ……ファイル番号

用　例

```
 10 /* feof sample
 20 int ai
 30 str as
 40 dim int aia(5),bia(0)
 50 linput "ファイル名を入力してください--->?";as
 60 as="a:¥"+as
 70 ai=fopen(as,"c")
 80     fwrite(aia,6,ai)
 90 fclose(ai)
100 ai=fopen(as,"rw")
110     while feof(ai)<>-1
120         fwrite(bia,1,ai)
130         print feof(ai);
140     endwhile
150 fclose(ai)
160 end
```

FGETC

標準関数

書　式　fgetc(fp)

引　数　int

戻り値　int

機　能　指定されたファイルから１文字ずつ読み込みます。戻り値として、読み込んだ文字のキャラクタコードを返します。ファイルの最後まで読み込んだ場合や、エラーの場合は－１を返します。

fp ……ファイル番号

ファイル内のデータポインタの位置は、１バイトずつ移動していきます。ファイル内をランダムアクセスしてデータポインタを移動する場合は、fseek関数を使用します。

➡　fputc

用　例

```
10 /* fgetc sample
20 int ai,bi
30 char ac
40 str as="personal workstation X68000    ",bs
50 dim char aca(31),bca(31)
60 linput "ファイル名を入力してください--->?";bs
70 for ai=0 to 31
80     aca(ai)=asc(mid$(as,ai+1,1))
90 next
100 ai=fopen(bs,"c")
110     for bi=0 to 31
120         ac=aca(bi)
130         fputc(ac,ai)
140     next
150 fclose(ai)
160 ai=fopen(bs,"rw")
170 bi=0
180     while feof(ai)<>-1
190         bca(bi)=fgetc(ai)
200         bi=bi+1
210     endwhile
220 fclose(ai)
230 print "aca=";
240 for ai=0 to 31
250     print chr$(aca(ai));
260 next
270 print:print "bca=";
280 for ai=0 to 31
290     print chr$(bca(ai));
300 next
310 end
```

FILES

コマンド

書　式　〔l〕files　〔"<ドライブ名><ファイル名>"〕

省略形　〔l〕fi.

機　能　指定されたドライブのファイルの一覧を表示します。
ドライブ名、ファイル名が省略された場合、カレントドライブのカレントディレクトリのすべてのファイル名を画面に表示します。ドライブ名のみ指定された場合は指定されたドライブのカレントディレクトリのすべてのファイル名を表示します。ドライブ名とファイル名が指定された場合には、指定されたファイルの情報が表示されます。
ワイルドカードも使用できます。
<ドライブ名>、<ファイル名>の詳細については、「Human68kユーザーズマニュアル」を参照してください。
lfiles命令では出力先がプリンタとなります。

用　例

```
files "a:¥*.*"
```

FILL

書　式　fill(x1,y1,x2,y2,p)

引　数　int

戻り値　void

機　能　グラフィック画面上に中を塗りつぶしたボックス(四角形)を描きます。

x1 ……始点X座標

y1 ……始点Y座標

x2 ……終点X座標

y2 ……終点Y座標

p……パレットコード(色)

x1、y1、x2、y2は、−32768から32767までの値をとることができます。クリッピングエリアについてはwindow関数で指定された範囲になります。また、パレットコードpは0から65535までの値をとることができ、その最大値はグラフィック画面の実画面サイズおよび色モードによります。

用　例

```
 10 /* fill sample
 20 int ai,bi,ci,di,ei,fi
 30 screen 1,3,1,1
 40 vpage(1)
 50 for ai=0 to 200
 60     bi=rand() mod 512
 70     ci=rand() mod 512
 80     di=rand() mod 512
 90     ei=rand() mod 512
100     fi=rnd()*65535
110     fill(bi,ci,di,ei,fi)
120 next
130 end
```

FIX

標準関数

書　式　fix(f)

引　数　float

戻り値　float

機　能　浮動小数値fの小数点以下を切りすてた値を返します。

用　例

```
10 /* fix sample
20 float af=2.63#
30 float bf=-5.6#
40 float cf=51
50 print fix(af)
60 print fix(bf)
70 print fix(cf)
80 end
```

FLOAT

ステートメント

書　式　float　〈変数名〉[=〈定数式〉][,...]

機　能　変数を実数(float)型として宣言します。同時に値を代入することもできます。
原則として、変数宣言はメインプログラムの先頭で行ってください。また、同じ名前の変数をメインプログラム中で再宣言することはできません。

➜　char、int、str

用　例

```
10 /* float sample
20 float af=25000000000000#
30 float bf=-5.68759412E+200#
40 float cf=1#/3,df=-3.254#
50 print af
60 print bf
70 print cf,df
80 end
```

FOPEN

標準関数

書　式　fopen(fn,md)

引　数　str

戻り値　int

機　能　指定されたファイルを、モードを決めてオープンします。戻り値として、ファイル番号を返します。エラーの場合は－１を返します。

fn ……ファイル名

md……モード

"r" ＝すでに存在しているファイルを読み込み用(リード)としてオープンする。ファイルがない場合はエラー。

"w" ＝すでに存在しているファイルを書き込み用(ライト)としてオープンする。ファイルがない場合はエラー。

"rw"＝すでに存在しているファイルを読み書き両用(リード／ライト)としてオープンする。ファイルがない場合はエラー。

"c" ＝新規ファイルを読み書き両用(リード／ライト)としてオープンする。同一ファイルが、すでに存在している場合は、そのファイルを消去してから、読み書き両用(リード／ライト)として新しくオープンする。

特に、既存ファイルのデータに対して、追加、修正する場合は、"w"、"rw" モードを使用します。

オープンされたばかりのファイルでは、ファイル内のデータポインタの位置は、ファイルの先頭にあります。データポインタを移動する場合は、fseek関数を使用します。

用　例

```
 10 /* fopen sample
 20 int ai,bi
 30 str as
 40 dim int aia(99),bia(100),cia(0)
 50 linput "ファイル名を入力してください--->?";as
 60 for ai=0 to 99
 70     aia(ai)=&H12345678
 80 next
 90 /* create mode
100 ai=fopen(as,"c")
110     fwrite(aia,100,ai)
120 fclose(ai)
130 /* read/write mode
140 ai=fopen(as,"rw")
150     /* update */
160      fseek(ai,40,0)
170      cia(0)=&H87654321
180      bi=fwrite(cia,1,ai)
190      if bi<=0 then print "update error!"
200     /* append */
210      fseek(ai,0,2)
220      cia(0)=&H87654321
230      bi=fwrite(cia,1,ai)
```

```
240       if bi<=0 then print "append error!"
250      /* read */
260       fseek(ai,0,0)
270       bi=fread(bia,101,ai)
280       if bi<=0 then print "no data(r/w)!"
290       print bi
300 fclose(ai)
310 /* create mode
320 ai=fopen(as,"c")
330      bi=fread(bia,101,ai)
340      if bi<=0 then print "no data(c)!"
350      print bi
360 fclose(ai)
370 end
```

FOR～NEXT

ステートメント

書　式

for　〈int型変数名〉＝〈初期値〉　to　〈最終値〉
〈ループの内容〉
next

機　能

forからnextまでの間で指定された〈ループの内容〉を、指定された回数だけ繰り返し(ループ)ます。
増分は1です。
〈初期値〉が〈最終値〉と等しい場合は、1回だけ〈ループの内容〉が実行されます。また、〈初期値〉が〈最終値〉より大きい場合は、〈ループの内容〉は一度も実行されません。

用　例

```
 10 /* for next sample
 20 int ai,bi
 30 cls
 40 input "ステップを入力してください--->";ai
 50 print
 60 for bi=0 to 10
 70      ci=bi*ai                        /* step
 80      print bi,ci
 90 next
100 end
```

FPUTC

標準関数

書　式　fputc(ch,fp)

引　数　char(ch),int(fp)

戻り値　int

機　能　指定されたファイルに１文字ずつ書き込みます。戻り値として、書き込んだ文字のキャラクタコードchを返します。ファイルへの書き込みができなかった場合や、エラーの場合は、－１を返します。

ch ……書きこむキャラクタコード

fp ……ファイル番号

キャラクタコードchは、0から255の値をとります。ファイル内のデータポインタの位置は、１バイトずつ移動していきます。ファイル内をランダムアクセスしてデータポインタを移動する場合は、fseek関数を使用します。

➡　fgetc

用　例

```
 10 /* fputc sample
 20 int ai,bi
 30 char ac
 40 str as="personal workstation X68000    ",bs
 50 dim char aca(31)
 60 linput "ファイル名を入力してください--->?";bs
 70 for ai=0 to 31
 80     aca(ai)=asc(mid$(as,ai+1,1))
 90 next
100 ai=fopen(bs,"c")
110     for bi=0 to 31
120         ac=aca(bi)
130         fputc(ac,ai)
140     next
150 fclose(ai)
160 end
```

FREAD

標準関数

書　式　fread(na,n,fp)

引　数　数値型一次元配列名(na),int(n,fp)

戻り値　int

機　能　指定されたファイルから、指定された要素数分nを指定された配列naに読み込みます。戻り値として、実際に読み込んだ要素数を返します。エラーの場合は－1を返します。

na ……数値型一次元配列名(int型または、char型または、float型の一次元配列名)

n……要素数（1バイト(char型)または4バイト(int型)または8バイト(float型)単位)

fp ……ファイル番号

ファイル内のデータポインタの位置は、指定された配列naのデータ型に対応して、char型であれば1バイトずつ、int型であれば4バイトずつ、float型であれば8バイトずつ移動していきます。ファイル内をランダムアクセスしてデータポインタを移動する場合は、fseek関数を使用します。

➡　fwrite

用　例

```
 10 /* fread sample
 20 int ai,bi,ci
 30 char ac,bc
 40 float af
 50 str as
 60 dim int aia(3),bia(3)
 70 dim char aca(15),bca(15)
 80 dim float afa(1),bfa(1)
 90 cls
100 linput "ファイル名を入力してください--->?";as
110 /* write file
120 ai=fopen(as,"c")
130 repeat
140 input "データの型タイプを入力してください( int = 0 , char = 1 , float = 2
)--->";ac
150 if ac=0 then {
160     for bi=0 to 3
170         input "int型データを入力してください--->";ci
180         aia(bi)=ci
190     next
200     fwrite(aia,4,ai)}
210 if ac=1 then {
220     for bi=0 to 15
230         input "char型データを入力してください--->";bc
240         aca(bi)=bc
250     next
260     fwrite(aca,16,ai)}
270 if ac=2 then {
280     for bi=0 to 1
290         input "float型データを入力してください--->";af
300         afa(bi)=af
```

```
310      next
320      fwrite(afa,2,ai)}
330 until ac<=2
340 fclose(ai)
350 cls
360 /* read file
370 ai=fopen(as,"rw")
380 print "書き込んだファイル("+as+")の内容を表示します":print
390 if ac=0 then {
400      fread(bia,4,ai)
410      for bi=0 to 3
420           print "データのポインタの位置(";fseek(ai,bi*4,0);") = ";
430           print bia(bi)
440      next}
450 if ac=1 then {
460      fread(bca,16,ai)
470      for bi=0 to 15
480           print "データのポインタの位置(";fseek(ai,bi,0);") = ";
490           print bca(bi)
500      next}
510 if ac=2 then {
520      fread(bfa,2,ai)
530      for bi=0 to 1
540           print "データのポインタの位置(";fseek(ai,bi*8,0);") = ";
550           print bfa(bi)
560      next}
570 fclose(ai)
580 end
```

FREADS

標準関数

書　式　freads(st,fp)

引　数　str型変数名(st),int(fp)

戻り値　int

機　能　指定されたファイルから1行をstr型変数stに読み込みます。戻り値として、ファイルから読み込んだ文字数を返します。ファイルの最後まで読み込んだ場合や、すでに読み込んでいる場合あるいはエラーの場合は、－1を返します。

st　……読み込んだ文字列を格納するstr型変数名

fp　……ファイル番号

1行の読み込みを終了するのは、改行(CR(&H0D)、LF(&H0A))が現われたときか、stの文字バッファがいっぱいになったときです。ファイルの読み込みを終了するのは、ファイルが終わったときか、ファイル終了コード（&H1A）が現われたときです。なお、stの文字バッファには、改行およびファイル終了コードは書き込まれません。ファイル内をランダムアクセスしてデータポインタを移動する場合は、fseek関数を使用します。

➡　fwrites

用　例

```
 10 /* freads sample
 20 int ai
 30 str as,bs[255]
 40 cls
 50 linput "ファイル名を入力してください--->?";as
 60 /* write file
 70 ai=fopen(as,"c")
 80     print "書き込むファイル名は、"+as+"です"
 90 while bs<>chr$(26)
100     bs=""
110     print "終了する場合は、'end'を入力してください!!"
120     linput "文字や文字列を入力してください--->?";bs
130     /* cr = chr$(13) , lf = chr$(10)
140     bs=bs+chr$(13)+chr$(10)
150     /* ctrl + Z = chr$(26)
160     if bs="end"+chr$(13)+chr$(10) then print "ファイルをクローズします":bs
=chr$(26)
170     fwrites(bs,ai)
180 endwhile
190 fclose(ai)
200 cls
210 for ai=0 to 10000:next
220 /* read file
230 print "書き込まれたファイル("+as+")の内容を表示します"
240 bs=""
250 ai=fopen(as,"rw")
260 repeat
270     print bs
280     freads(bs,ai)
290 until feof(ai)=-1
300 fclose(ai)
310 end
```

FREE

システム変数

書　式　free

戻り値　int

機　能　現在のメモリ上のフリーエリアを返します。代入することはできません。

用　例

```
10 /* free sample
20 print free
30 end
```

FRENAME

標準関数

書　式　frename(fn1, fn2)

引　数　str

戻り値　int

機　能　ファイル名fn1のファイルをファイル名fn2に変更します。変更できた場合、戻り値として0を返します。エラーの場合は－1を返します。

fn1 ……変更したいファイル名

fn2 ……変更後のファイル名

用　例

```
10 /* frename sample
20 int ai
30 str as,bs
40 cls
50 linput "変更したいファイル名を入力してください--->?";as
60 linput "新しくつけるファイル名を入力してください--->?";bs
70 print "ファイル名("+as+")をファイル名("+bs+")に変更します"
80 ai=frename(as,bs)
90 if ai<>0 then print "ファイル名の変更ができません"
100 print "ファイル名の変更ができました"
110 end
```

FSEEK

標準関数

書　式　fseek (fp,os,md)

引　数　int

戻り値　int

機　能　指定されたファイル内のデータポインタを移動します。ファイルをシークモードmdで指定された位置から1バイト単位でランダムアクセスするときに使用します。シークモードmdで指定された位置から、オフセットosで指定されたところまで、ランダムアクセス後、新しいデータポインタの位置を返します。エラーの場合は－1を返します。

fp　……ファイル番号
os　……オフセット（バイト単位）
md ……シークモード
　　0＝ファイルの先頭
　　1＝現在の位置
　　2＝ファイルの終わり

用　例

```
 10 /* fseek sample
 20 int ai,bi
 30 char ac
 40 str as="personal workstation X68000    ",bs
 50 dim char aca(31),bca(9)
 60 linput "ファイル名を入力してください--->?";bs
 70 for ai=0 to 31
 80     aca(ai)=asc(mid$(as,ai+1,1))
 90 next
100 ai=fopen(bs,"c")
110     for bi=0 to 31
120         ac=aca(bi)
130         fputc(ac,ai)
140     next
150 fclose(ai)
160 ai=fopen(bs,"rw")
170     fseek(ai,20,0)
180     for bi=0 to 9
190         fseek(ai,1,1)
200         bca(bi)=fgetc(ai)
210         fseek(ai,-1,1)
220         print chr$(bca(bi));
230     next
240 fclose(ai)
250 end
```

FUNC～ENDFUNC～RETURN( )ステートメント

書　式

func 〔〈戻り値の型〉〕 〈関数名〉(〔〈引数リスト〉〕)
〈関数の内容〉
endfunc

機　能

関数を定義します。func～endfuncの関数は、必ずend命令以後に定義します。また、func～endfuncの関数内では、ローカル変数を指定でき、再帰呼び出しもできるような構造になっています。ただし、原則として配列変数などはメインプログラムの先頭で宣言してください。

通常、引数はカンマ(,)で区切って〈引数リスト〉に記述します。引数がない場合は省略可能です。

それぞれの引数につづけてセミコロン(;)を書き、その後にデータ型を指定することができます。このデータ型の指定を省略すると引数のデータ型はint型になります。

メインプログラム中で、定義された関数に渡す引数のデータ型や個数は、〈引数リスト〉のデータ型や個数と一致させてください。

func～endfuncで定義された関数から演算結果などの値を返したい場合は次のような命令をfunc～endfuncでかこまれた中で使用します。

return (〔〈戻り値〉〕)

func～endfuncで定義された関数からこのreturn命令によって返される戻り値のデータ型は、〈戻り値の型〉で指定することができます。省略した場合はint型になります。

func～endfuncで定義された関数は、プログラムとしてメモリ上に存在していればダイレクトモードでも実行できます。

用　例

```
100 /* func sample. coast creation */
110 float s
120 while s<1 or s>=2
130   input "ratio 1 to 2";s
140 endwhile
150 s = (s-1)/10+1
160 screen 1,2,1,1
170 s=sqr(s*s-1)
180 float x0=100, x1=412, y0=0, y1=0
190 fractal(x0,x1,y0,y1,1)
200 line(100, 50, 412, 50, 255, 65535)
210 end
220 func fractal(x0;float,x1;float,y0;float,y1;float,sp;int)
230   float l, r, x2, y2
240   l=sqr((x1-x0)*(x1-x0)+(y1-y0)*(y1-y0))
250   if l<2 or sp>=9 then {
260     line(x0,y0/3+50,x1,y1/3+50,255,65535) : return()
270   }
280   r=rnd()+rnd()+rnd()-2
290   x2=(x0+x1)/2+s*(y1-y0)*r
300   y2=(y0+y1)/2+s*(x0-x1)*r
310   sp = sp + 1
320   fractal(x0,x2,y0,y2,sp)
330   fractal(x2,x1,y2,y1,sp)
340 endfunc
```

FWRITE

標準関数

書　式　fwrite(na, n, fp)

引　数　数値型一次元配列名(na), int(n, fp)

戻り値　int

機　能　指定された配列naのデータを、指定された要素数分nだけ指定されたファイルに書き込みます。戻り値として、実際に書き込んだ要素数を返します。エラーの場合は－1を返します。

na ……数値型一次元配列名(int型または、char型または、float型の一次元配列名)

n……要素数（1バイト(char型)または4バイト(int型)または8バイト(float型)単位)

fp ……ファイル番号

ファイル内のデータポインタの位置は、指定された配列naのデータ型に対応して、char型であれば1バイトずつ、int型であれば4バイトずつ、float型であれば8バイトずつ移動していきます。ファイル内をランダムアクセスしてデータポインタを移動する場合は、fseek関数を使用します。

➡　fread

用　例

```
 10 /* fwrite sample
 20 int ai,bi
 30 str as
 40 dim int aia(5),bia(5),cia(0)
 50 dim char aca(23),bca(23),cca(0)
 60 linput "ファイル名を入力してください--->?";as
 70 for ai=0 to 5
 80     aia(ai)=&H12345678
 90 next
100 /* write
110 ai=fopen(as,"c")
120     fwrite(aia,6,ai)
130 fclose(ai)
140 /* read
150 ai=fopen(as,"rw")
160     fseek(ai,8,0)
170     fread(cia,1,ai)
180     print "int type data (No.2)=";hex$(cia(0))
190     fseek(ai,8,0)
200     fread(cca,1,ai)
210     print "char type data (No.8)=";hex$(cca(0))
220 /* int read
230     fseek(ai,0,0)
240     fread(bia,6,ai)
250 /* char read
260     fseek(ai,0,0)
270     fread(bca,24,ai)
280 fclose(ai)
290 print "int array";"                    ";"char array"
300 for ai=0 to 5
310     bi=ai*4
320     print "bia(";ai;")=";hex$(bia(ai));
330     print "        ";
340     print "bca(";bi;")=";hex$(bca(bi));"   ";
350     print "bca(";bi+1;")=";hex$(bca(bi+1));"   ";
360     print "bca(";bi+2;")=";hex$(bca(bi+2));"   ";
370     print "bca(";bi+3;")=";hex$(bca(bi+3))
380 next
390 end
```

FWRITES

標準関数

書　式　fwrites(st, fp)

引　数　str型変数名(st), int(fp)

戻り値　int

機　能　指定されたファイルにstr型変数stの１行を書き込みます。戻り値として、ファイルに書き込んだ文字数を返します。エラーの場合は、－１を返します。

st　……書き込むべき文字列を格納しているstr型変数名

fp　……ファイル番号

null文字(キャラクタコードの０)は書き込みません。ファイル内をランダムアクセスしてデータポインタを移動する場合は、fseek関数を使用します。

➜　freads

用　例

```
10 /* file sample
20 int ai
30 char ac,bc
40 str as,bs[255],cs,ds,es,fs,bun
50 cls
60 linput "ファイル名を入力してください--->?";as
70 /* write file
80 ai=fopen(as,"c")
90     print "書き込むファイル名は、"+as+"です"
100 while bs<>chr$(26)
110     bs=""
120     print "終了する場合は、'end'を入力してください!!"
130     linput "文字や文字列を入力してください--->?";bs
140     /* cr = chr$(13) , lf = chr$(10)
150     bs=bs+chr$(13)+chr$(10)
160     /* ctrl + Z = chr$(26)
170     if bs="end"+chr$(13)+chr$(10) then print "ファイルをクローズします":bs
=chr$(26)
180     fwrites(bs,ai)
190 endwhile
200 fclose(ai)
210 for ai=0 to 10000:next
220 print
230 re_file("書き込まれた")
240 print
250 /* update file
260 ai=fopen(as,"rw")
270 while -1
280     linput "修正する文字がありますか？<y/n>---> ";cs
290     if cs="n" then break
300     linput "修正したい文字を入力してください--->?";ds
310     repeat
320         ac=fgetc(ai)
330         if ac=13 then bc=bc+1:continue
340         if ac=10 then bc=bc+1:continue
350         if ac=26 then break
360         if bc=1 or bc=2 then print
370         if asc(ds)=ac then {
380             color 2
390             print chr$(ac)
```

```
400                color 3
410                input "この文字でよろしいですか？<y/n>---> ";es
420                if es<>"y" then continue
430                linput "新しく修正する文字を入力してください--->?";fs
440                fseek(ai,-1,1)
450                fputc(asc(fs),ai)
460                es="":continue} else {
470            color 3}
480            print chr$(ac);
490            bc=0
500        until feof(ai)=-1
510        if ac<>26 then print "修正する文字がありませんでした":break
520        break
530 endwhile
540 print:print
550 re_file("修正された")
560 end
570 /* read file
580 func re_file(bun;str)
590 print bun+"ファイル("+as+")の内容を表示します"
600 bs=""
610 ai=fopen(as,"rw")
620 fseek(ai,0,0)
630 repeat
640     print bs
650     freads(bs,ai)
660 until feof(ai)=-1
670 fcloseall()
680 endfunc
```

GCVT

標準関数

書　式　gcvt(f, i)

引　数　float(f), int(i)

戻り値　str

機　能　浮動小数値fを文字列に変換し、その文字列を返します。

f ……文字列に変換したい浮動小数値

i ……文字列の桁数

通常は指定された桁数iの実数表現に変換しますが、それができない場合は指数表現に変換します。先頭の”0”は変換の際に取り除かれます。

➡　atof、ecvt、fcvt

用　例

```
 10 /* gcvt sample
 20 float af,bf,cf
 30 af=25000000000000#
 40 bf=-2.6546681E+205#
 50 cf=-5.2365426E-018#
 60 print str$(af)
 70 print gcvt(af,20)
 80 print str$(bf)
 90 print gcvt(bf,20)
100 print str$(cf)
110 print gcvt(cf,20)
120 end
```

GET

書　式　get(x1,y1,x2,y2,na)

引　数　int(x1,y1,x2,y2)，数値型一次元配列名(na)

戻り値　void

機　能　グラフィック画面の指定領域のドットパターンを指定された配列naに読み込みます。

x1 ……始点X座標

y1 ……始点Y座標

x2 ……終点X座標

y2 ……終点Y座標

na ……ドットパターンを読み込む数値型一次元配列名

x1、y1、x2、y2は、window関数で指定された範囲内の値を取ることができます。また、配列naへのドットパターンの読み込みかたとしてはグラフィック画面の実画面サイズおよび色モードにより次のようになります。

実画面サイズ1024×1024、16色モード(4ビット／ドット構成)

読み込み順＝(x1，y1)，(x1+1，y1),……,(x2，y1),(x1，y1+1)，(x1+1，y1+1),……,(x2，y2)

char型配列の場合＝2ドット単位

int型配列の場合＝8ドット単位

float型配列の場合＝16ドット単位

実画面サイズ512×512、16色モード

実画面サイズ1024×1024、16色モードと同様

但し、apage関数によって、指定されたアクティブページ(0～3)に対してのみ有効

実画面サイズ512×512、256色モード(1バイト／ドット構成)

読み込み順＝実画面サイズ1024×1024、16色モードと同様

char型配列の場合＝1ドット単位

int型配列の場合＝4ドット単位

float型配列の場合＝8ドット単位

但し、apage関数によって、指定されたアクティブページ(0、1)に対してのみ有効

実画面サイズ512×512、65536色モード(2バイト／ドット構成)

読み込み順＝実画面サイズ1024×1024、16色モードと同様

char型配列の場合＝2要素1ドット単位

int型配列の場合＝2ドット単位

float型配列の場合＝4ドット単位

グラフィック画面への書き込みについてはput関数を参照してください。

➜　put

GOSUB～RETURN

ステートメント

書　式　gosub 〈行番号〉

機　能　指定したサブルーチンを呼び出し、return命令でもどります。
ただし、gosub命令を使用したプログラムは、行番号に依存したプログラムになるためload@命令、save@命令は使用できません。また、renum命令による〈行番号〉の変更もできなくなります。

➜　goto

用　例

```
 10 /* gosub return sample
 20 print "super ";
 30 gosub 100
 40 print "SHARP"
 50 end
100 print "personal workstation X68000    "
110 return
```

GOTO

ステートメント

書　式　goto 〈行番号〉

機　能　指定した行にジャンプします。
ただし、goto命令を使用したプログラムは、行番号に依存したプログラムになるためload@命令、save@命令は使用できません。また、renum命令による〈行番号〉の変更もできなくなります。
goto命令で、for～next、while～endwhileなどのブロック構造から外に出てしまうとプログラムの動作が正常でなくなる可能性があります。

用　例

```
 10 /* goto sample
 20 print "super ";
 30 goto 100
 40 end
100 print "personal workstation X68000   "
```

HEX$

標準関数

書　式　hex$(i)

引　数　int

戻り値　str

機　能　整数値iを16進表現の文字列に変換し返します。なお、上位の”0”は取り除かれます。

➡　bin$、oct$

用　例

```
10 /* hex$ sample
20 int ai,bi
30 ai=-255
40 bi=100542300
50 print hex$(ai)
60 print hex$(bi)
70 print hex$(ai)+hex$(bi)
80 end
```

HOME

GRAPH

書　式　home(pa, x, y)

引　数　char(pa), int(x, y)

戻り値　void

機　能　グラフィック画面の実画面における表示画面の左上座標を指定します。

pa ……アクティブページ

x……実画面における表示画面の左上X座標

y……実画面における表示画面の左上Y座標

アクティブページpaはグラフィック画面の実画面サイズ及び色モードにより最大値(0～3)が決まり、x、yも同様にグラフィック画面の実画面サイズにより、実画面サイズが512×512の場合は0から511まで、あるいは実画面サイズが1024×1024の場合は0から1023までの値をとることができます。

screen命令を実行すると(x,y)には(0,0)が再設定されます。

➜　apage、screen、vpage、window

用　例

```
10 /* home sample
20 int ai,bi
30 float af
40 screen 1,1,1,1
50 vpage(0)
60 window(0,0,511,511)
70 apage(0)
80 kyu(0)
90 apage(1)
100 kyu(1)
110 apage(2)
120 kyu(2)
130 apage(3)
140 kyu(3)
150 vpage(15)
160 while -1
170 for ai=0 to 511
180     af=pi()*ai/512
190     bi=sin(af)*511
200     home(0,ai,bi):home(1,bi,ai)
210     home(2,511-ai,bi):home(3,bi,511-ai)
220 next
230 endwhile
240 end
250 func kyu(ai)
260 circle(128,128,50,ai*4+1)
270 paint(128,128,ai*4+1)
280 circle(256,256,50,ai*4+2)
290 paint(256,256,ai*4+2)
300 circle(384,384,50,ai*4+3)
310 paint(384,384,ai*4+3)
320 endfunc
```

HSV

書　式　hsv(hu, sa, va)

引　数　char

戻り値　int

機　能　色相(hu)、飽和度(sa)、明るさ(va)からカラーコードを求めます。

hu ……0～191

sa ……0～31

va ……0～31

なお、このhsv関数によって得られるカラーコードは、0～65534までの偶数値になります。したがって、それぞれの偶数値に1を加えることで、0～65535までのカラーコードを指定できます。

用　例

```
10 /* hsv sample
20 int ai,bi,ci
30 screen 1,3,1,1
40 vpage(1)
50 for ai=0 to 15
60     for bi=0 to 3
70         for ci=0 to 2
80             kyu(bi*128,ci*170,ai*12+bi*3+ci)
90         next
100    next
110 next
120 end
130 func kyu(di;int,ei;int,fi;int)
140 int gi,hi,ii
150 fill(di,ei,di+127,ei+169,0)
160 for gi=0 to 31
170     hi=31-gi
180     for ii=0 to 1
190         circle(di+64,ei+64,hi*2+ii,hsv(fi,31,gi)+ii,0,360,256)
200         paint(di+64,ei+64,hsv(fi,31,gi)+ii)
210     next
220 next
230 endfunc
```

IF～THEN～ELSE

ステートメント

書　式　if 〈条件〉 then 〈文〉〔else 〈文〉〕

機　能　〈条件〉で条件判断を行い、その結果に応じて〈文〉を実行します。
ifに続けて〈条件〉を指定します。その〈条件〉が満たされたときはthen以下の〈文〉が実行され、そうでないときはelse以下の〈文〉が実行されます。
else以降は省略することができます。その場合、条件が満たされなかったときは次の行が実行されます。
条件には、以下のような比較演算子と論理演算子とが使えます。論理演算子は、複数の条件を指定するときに使います。

比較演算子	意　味	例
式1 ＝ 式2	式1が式2と等しい	if　a＝1
式1<>式2	式1が式2と等しくない	if　a<>0
式1 < 式2	式1が式2より小さい	if　a<10
式1 > 式2	式1が式2より大きい	if　a>b
式1<＝式2	式1が式2と等しいか、式2より小さい	if　a<＝b+1
式1>＝式2	式1が式2と等しいか、式2より大きい	if　a>＝"n" (aはstr型)

論理演算子	例	意　味
not	if　not　(a＝1)	aが1でなければthen以下を実行
and	if　a＝0　and　b＝3	aが0で、bが3のときのみthen以下を実行
or	if　a<0　or　b＝0	aが0より小さいか、bが0のときにthen以下を実行

then節またはelse節は、{ }と組み合わせて、複数行の条件判断に使用できます。

用　例

```
10 /* if then else sample1 */
20 int ai,bi
30 input "(1)1+2=";ai
40 input "(2)2+3=";bi
50 if ai=3 then {
60      if bi=5 then {
70          print "(1),(2)ok!" } else {
80              print "(1)ok!,(2)error!" }} else {
90      if bi=5 then {
100         print "(1)error,(2)ok!" } else {
110             print "(1)error!,(2)error!" }}
120 end
```

```
10 /* if then else sample */
20 int ai
30 input "1+2=";ai
40 if ai<=0 then print "error!" else {
50     if ai=1 then print "error!" else {
60         if ai=2 then print "error!" else {
70             if ai=3 then print "ok!" else {
80                 print "error!" }}}}
90 end
```

IMG_COLOR

書　式　img_color(md)

引　数　char

戻り値　void

機　能　取り込み画像のグラフィックパレットを補正します。

md ……グラフィック画面の色モード(0～2)

モードmdの値と意味は次のとおりです。

0＝グラフィック画面16色モード

1＝グラフィック画面256色モード

2＝グラフィック画面65536色モード

IMG_HOME

IMAGE

書　式　img_home(x, y, dir, cnt, wt)

引　数　int(x, y, cnt, wt), char(dir)

戻り値　void

機　能　グラフィック画面の表示画面の位置を移動します。

x……グラフィック画面の実画面における表示画面の左上X座標
y……グラフィック画面の実画面における表示画面の左上Y座標
dir ……移動方向(1～9)
cnt ……移動ドット数
wt ……ウェイト時間(×約1.2μ秒)

移動方向dirの値と意味は次のとおりです。

1＝左下
2＝下
3＝右下
4＝左
6＝右
7＝左上
8＝上
9＝右上

用　例

```
 10 /* img_home sample
 20 int i
 30 screen 1,3,1,1
 40 console ,,0
 50 img_color(2)
 60 vpage(1)
 70 symbol(0,0,"X68000",2,4,2,32556,0)
 80 /*
 90   img_home(0,0,2,100,3000)
100   img_home(0,411,8,100,3000)
110   img_home(0,0,3,256,3000)
```

IMG_HT

IMAGE

書　式　img_ht(x1,y1,x2,y2,p,a)

引　数　int(x1,y1,x2,y2), char(p,a)

戻り値　void

機　能　グラフィック画面とテレビ・ビデオ画面との半透明表示を制御します。

x1 ……グラフィック画面の半透明領域の始点X座標
y1 ……グラフィック画面の半透明領域の始点Y座標
x2 ……グラフィック画面の半透明領域の終点X座標
y2 ……グラフィック画面の半透明領域の終点Y座標
p……指定された領域内のグラフィック画面の半透明表示の設定(0＝設定しない、1＝設定する)
a……半透明表示の制御(0＝半透明表示を実行しない、1＝半透明表示を実行する)

なお、この関数を実行するにはカラーイメージユニットが必要です。また、CZ-674C/510C/500C/310C/300Cでは、この関数を使用できません。

IMG_LOAD

IMAGE

書　式　img_load(st〔,x〕〔,y〕〔,b〕)

引　数　str(st), int(x, y, b)

戻り値　int

機　能　グラフィック画面をロードします。戻り値として、正常のときは0、エラーのときは−1を返します。

st ……主ファイル名
x……ロードするグラフィック画面の左上X座標
y……ロードするグラフィック画面の左上Y座標

表示画面サイズが512×512の場合

$x = 0, y = 0$

表示画面サイズが256×256の場合

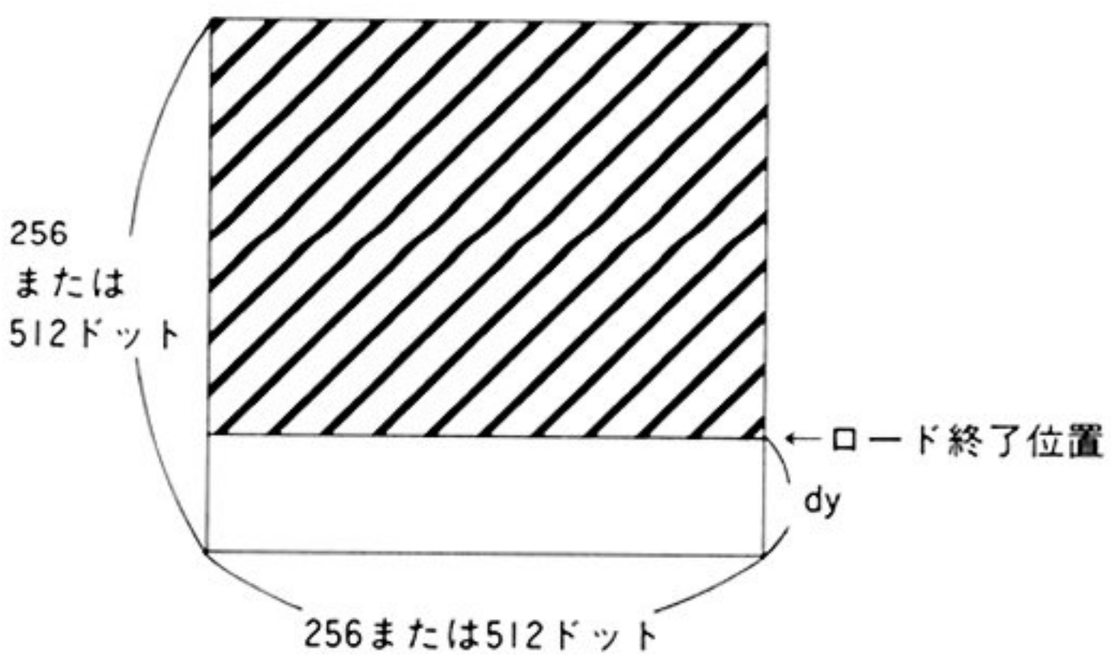
$0 \leqq x \leqq 256, 0 \leqq y \leqq 256$

テレビやビデオから画像を取り込むとき、画面下にブランキング部分が同時に取り込まれます。その部分を表示しないようにロードするために、終了位置を指定することができます。

b……ロード終了位置の指定

$$b=\frac{dy}{n}$$

dy…下図のとおりY座標の下からのドット数



拡張子	n	b
GS0	…32	0～6
GS3	…16	0～7
GM0	…16	〃
GM3	… 8	〃
GL0	… 8	〃
GL3	… 4	設定不可

拡張子の意味については、img.save関数を参照してください。

IMG_POS

IMAGE

書　式　img_pos(hc)

引　数　char

戻り値　void

機　能　画像取り込み時の画面水平位置を補正します。

hc ……水平位置補正データ(0～255)

●表示画面サイズが512×512の場合
画像取り込みする際、はじめは&H9Aにセットし、静止画にした後&H2Cに設定します。

●表示画面サイズが256×256の場合
はじめは&HEBにセットし、静止画にした後、&H24に設定します。
なお、この関数を実行するにはカラーイメージユニットが必要です。また、CZ-674C/510C/500C/310C/300Cでは、この関数を使用できません。

IMG_PUT

IMAGE

書　式　img_put(x1,y1,x2,y2,ia)

引　数　int(x1,y1,x2,y2)，int型一次元配列名(ia)

戻り値　void

機　能　グラフィック画面の(x1,y1)から(x2,y2)まで、8ドットごとに移動しながら、配列iaのデータを256×256ドットのボックスで表示します。

x1 ……グラフィック画面の始点X座標 (0≦x1≦256)
y1 ……グラフィック画面の始点Y座標 (0≦y1≦256)
x2 ……グラフィック画面の始点X座標 (0≦x2≦256)
y2 ……グラフィック画面の始点Y座標 (0≦y2≦256)
ia ……get関数で読み込んだint型一次元配列名

IMG_SAVE

IMAGE

書　式　img_save(st〔,x〕〔,y〕)

引　数　str(st), int(x, y)

戻り値　int

機　能　グラフィック画面をセーブします。戻り値として、正常の時は0、エラーの時は－1を返します。

st ……主ファイル名

拡張子はグラフィック画面のスクリーンモードに応じて以下のように付加されます。

表示画面サイズ	色モード	拡張子
256×256	16色	GS0
256×256	256色	GM0
256×256	65536色	GL0
512×512	16色	GS3
512×512	256色	GM3
512×512	65536色	GL3

x……セーブするグラフィック画面の左上X座標
y……セーブするグラフィック画面の左上Y座標

表示画面サイズが512×512の場合

x = 0, y = 0

表示画面サイズが256×256の場合

0 ≦ x ≦256, 0 ≦ y ≦256

IMG_SCRN

IMAGE

書　式　img_scrn(sz, co, dis)

引　数　int

戻り値　void

機　能　グラフィック画面のスクリーンモードを設定します。
ただし、グラフィック画面クリア、パレット・コントラスト・表示モードの初期化は行いません。

sz ……表示画面サイズ(0～2)
　0＝256×256
　1＝512×512
　2＝768×512
co ……グラフィック画面の実画面サイズ及び色モード(0～3)
　0＝1024×1024　16色
　1＝512×512　16色
　2＝512×512　256色
　3＝512×512　65536色
dis ……ディスプレイ解像度(0、1)
　0＝標準解像度
　1＝高解像度

IMG_SET

書　式　img_set(c[,c][,c][,c][,c])

引　数　char

戻り値　void

機　能　カラーイメージユニットの各種コントロールを行います。

c ……カラーイメージユニットのコントロールコード(&H00～&HFF)

コントロールコード一覧表（使用ポートアドレス E8E005H）

入力する値		コード	命　令	備　考
0	00(H)	00000	65536色(2^{16})モード	
1	01	00001	空き	
2	02	00010		
3	03	00011	コンピュータコントロールモード	赤色ランプ点灯
4	04	00100	マニュアルモード	緑色ランプ点灯
5	05	00101	カラーイメージ　モノクロ画像	リセット、色の濃さUP DOWNいずれかで解除される。
6	06	00110	テロッパ　出力ON	
7	07	00111	テロッパ　出力OFF	リセットで解除される。
8	08	01000	カラーイメージ　リセット	各映像調整を標準の位置に戻す。
9	09	01001	カラーイメージ　色の濃さUP	
10	0A	01010	カラーイメージ　色の濃さDOWN	
11	0B	01011	カラーイメージ　色あいUP	
12	0C	01100	カラーイメージ　色あいDOWN	
13	0D	01101	カラーイメージ　映像UP	
14	0E	01110	カラーイメージ　映像DOWN	
15	0F	01111	256色(2^8)モード	
16	10	10000	テロッパ　リセット	各映像調整を標準の位置に戻す。
17	11	10001	テロッパ　色の濃さUP	
18	12	10010	テロッパ　色の濃さDOWN	
19	13	10011	テロッパ　色あいUP	
20	14	10100	テロッパ　色あいDOWN	
21	15	10101	テロッパ　映像UP	
22	16	10110	テロッパ　映像DOWN	
23	17	10111	16色(2^4)モード	
24 \ 31	18 \ 1F	11000 \ 11111	空き	

なお、この関数を実行するにはカラーイメージユニットが必要です。また、CZ-674C/510C/500C/310C/300Cでは、この関数を使用できません。

IMG_STILL

IMAGE

書　式　img_still(sw)

引　数　char

戻り値　void

機　能　画像取り込みモードswを設定します。

sw ……画像取り込みモードの設定

0＝実行(静止画)

1＝待機(動画)

実行前に、screen命令で、あらかじめ表示画面サイズを256×256または512×512に、ディスプレイ解像度をLow(標準解像度)に設定します。

また、画像取り込みを実行するプログラムの終了時には、img_still(0)を実行します。

なお、この関数を実行するにはカラーイメージユニットが必要です。また、CZ-674C/510C/500C/310C/300Cでは、この関数を使用できません。

INKEY$

システム変数

書　式　inkey$

戻り値　str

機　能　キーボードから入力された1文字を返します。代入することはできません。キー入力されるまで待ちます。

➜　input、「キャラクタコード表」

用　例

```
10 /* inkey$ sample
20 str as
30 while as<>"z"
40      print "end of code = 'z'"
50      as=inkey$
60      print as
70 endwhile
80 end
```

INPUT

ステートメント

書　式　input　〔"〈プロンプト文〉"〈；または，〉〕〈変数名〉〔，〈変数名〉...〕

機　能　キーボードから入力されるデータを、指定したデータ型の変数へ代入します。データを入力したら、リターンキーを押してください。

input文では、〈プロンプト文〉と〈変数名〉の間にセミコロン（；）あるいは、カンマ（，）が指定でき、セミコロンのときは〈プロンプト文〉の後に疑問符（？）が表示され、カンマのときはなにも表示されません。また、変数に入力できる最大文字数は、セミコロン（；）を使用した場合は、その表示画面の水平最大表示文字数－(プロンプト文字数＋３)になり、カンマ（，）を使用した場合は、その表示画面の水平最大表示文字数－(プロンプト文字数＋１)になります。ただし、str型変数の場合は、設定された文字バッファのサイズ以上のデータは代入されません。なおカンマ（，）や引用符（"）は入力できません。

変数は、カンマ（，）で区切って複数個指定することもできます。この場合、入力するデータもカンマ（，）で区切って、変数の数だけ入力してからリターンキーを押さなければなりません。入力するデータの個数が足りないと、"データの個数がたりません"と表示されて入力待ちとなり、個数が多いと、"データの個数が多過ぎます"と表示され、余分に入力されたデータは無視されます。また、対応する変数の型と入力されるデータの型は、一致しなければなりません。型が違っていた場合には、"データの型が違います"と表示されて再び入力待ちとなります。

➡　inkey$

用　例

```
10 /* input sample
20 int ai,bi
30 char ac,bc
40 float af,bf
50 str as[255],bs[255]
60 input "int data ai,bi = ",ai,bi
70 input "char data ac,bc = ";ac,bc
80 input "float data af,bf = ",af,bf
90 input "str data as = ";as
100 print "int data ai = ";ai,"bi=";bi
110 print "char data ac = ";ac,"bc=";bc
120 print "float data af = ";af,"bf=";bf
130 print "str data as = ";as
140 end
```

INSTR

標準関数

書　式　instr(i, st1, st2)

引　数　int(i), str(st1, st2)

戻り値　int

機　能　文字列st2を文字列st1の指定された位置iから探してその位置を返します。位置iは1～255の値をとります。ただし次の場合は0を返します。

- 文字列st2が見つからなかった場合
- 指定された位置iが0または負の場合
- 文字列st1またはst2がnull文字の場合

用　例

```
10 /* instr sample
20 str as,bs
30 as="personal workstation X68000   "
40 bs="X68"
50 print instr(2,as,bs)
60 end
```

INT

ステートメント

書　式　int　〈変数名〉〔=〈定数式〉〕〔,...〕

機　能　変数を整数(int)型として宣言します。同時に値を代入することもできます。
原則として、変数宣言はメインプログラムの先頭で行ってください。また、同じ名前の変数をメインプログラム中で再宣言することはできません。

➜　char, float, str

用　例

```
10 /* int sample
20 int ai=542684521
30 int bi=-4265312
40 int ci=23563,di=28
50 print ai
60 print bi
70 print ci,di
80 end
```

INT

標準関数

書　式　int(f)

引　数　float

戻り値　int

機　能　浮動小数値fを越えない最大の整数値を返します。

用　例

```
10 /* int() sample
20 float af=2.63#
30 float bf=-5.6#
40 float cf=51
50 print int(af)
60 print int(bf)
70 print int(cf)
80 end
```

ISALNUM

標準関数

書　式　isalnum(ch)

引　数　char

戻り値　int

機　能　キャラクタコードchの文字が英数字（'a'～'z'、'A'～'Z'、'0'～'9'）であるかどうか調べます。英数字であれば－1、そうでなければ0を返します。

ch ……調べたい文字のキャラクタコード(0～255)

用　例

```
10 /* isalnum sample
20 char ac,bc,cc
30 ac=asc("a")
40 bc=asc("5")
50 cc=asc("+")
60 print isalnum(ac)
70 print isalnum(bc)
80 print isalnum(cc)
90 end
```

ISALPHA

標準関数

書　式　isalpha(ch)

引　数　char

戻り値　int

機　能　キャラクタコードchの文字が英字（'a'～'z'、'A'～'Z'）であるかどうか調べます。英字であれば－1、そうでなければ0を返します。

ch ……調べたい文字のキャラクタコード(0～255)

用　例

```
10 /* isalpha sample
20 char ac,bc,cc
30 ac=asc("a")
40 bc=asc("5")
50 cc=asc("+")
60 print isalpha(ac)
70 print isalpha(bc)
80 print isalpha(cc)
90 end
```

ISASCII

標準関数

書　式　isascii(ch)

引　数　char

戻り値　int

機　能　キャラクタコードchの文字がアスキー文字(&H00～&H7F)であるかどうか調べます。アスキー文字であれば－1、そうでなければ0を返します。

ch ……調べたい文字のキャラクタコード(0～255)

用　例

```
 10 /* isascii sample
 20 char ac,bc,cc,dc
 30 ac=asc("a")
 40 bc=asc("5")
 50 cc=asc("+")
 60 dc=asc("ミ")
 70 print isascii(ac)
 80 print isascii(bc)
 90 print isascii(cc)
100 print isascii(dc)
110 end
```

ISCNTRL

標準関数

書　式　iscntrl(ch)

引　数　char

戻り値　int

機　能　キャラクタコードchの文字がコントロール文字(&H00～&H1F、&H7F)であるかどうか調べます。コントロール文字であれば－1、そうでなければ0を返します。

ch ……調べたい文字のキャラクタコード(0～255)

用　例

```
 10 /* iscntrl sample
 20 char ac,bc,cc,dc
 30 ac=asc("a")
 40 bc=asc("5")
 50 cc=asc("+")
 60 dc=13                /* cntrl+M */
 70 print iscntrl(ac)
 80 print iscntrl(bc)
 90 print iscntrl(cc)
100 print iscntrl(dc)
110 end
```

ISDIGIT

標準関数

書　式　isdigit(ch)

引　数　char

戻り値　int

機　能　キャラクタコードchの文字が数字('0'～'9')であるかどうか調べます。数字であれば－1、そうでなければ0を返します。

ch ……調べたい文字のキャラクタコード(0～255)

用　例

```
10 /* isdigit sample
20 char ac,bc,cc
30 ac=asc("a")
40 bc=asc("5")
50 cc=asc("+")
60 print isdigit(ac)
70 print isdigit(bc)
80 print isdigit(cc)
90 end
```

ISGRAPH

標準関数

書　式　isgraph(ch)

引　数　char

戻り値　int

機　能　キャラクタコードchの文字が空白文字以外の表示可能な文字(&H21～&H7E)であるかどうか調べます。表示可能であれば－1、そうでなければ0を返します。

ch ……調べたい文字のキャラクタコード(0～255)

用　例

```
 10 /* isgraph sample
 20 char ac,bc,cc,dc
 30 ac=asc("a")
 40 bc=13                  /* cntrl+M */
 50 cc=asc(" ")
 60 dc=asc("ミ")
 70 print isgraph(ac)
 80 print isgraph(bc)
 90 print isgraph(cc)
100 print isgraph(dc)
110 end
```

ISLOWER

標準関数

書　式　islower(ch)

引　数　char

戻り値　int

機　能　キャラクタコードchの文字が英小文字('a'～'z')であるかどうか調べます。英小文字であれば－1、そうでなければ0を返します。

ch ……調べたい文字のキャラクタコード(0～255)

用　例

```
10 /* islower sample
20 char ac,bc,cc
30 ac=asc("a")
40 bc=asc("A")
50 cc=asc("5")
60 print islower(ac)
70 print islower(bc)
80 print islower(cc)
90 end
```

ISPRINT

標準関数

書　式　isprint(ch)

引　数　char

戻り値　int

機　能　キャラクタコードchの文字が表示可能な文字(&H20～&H7E)であるかどうか調べます。表示可能であれば－1、そうでなければ0を返します。

ch ……調べたい文字のキャラクタコード(0～255)

用　例

```
 10 /* isprint sample
 20 char ac,bc,cc,dc
 30 ac=asc("a")
 40 bc=13                /* cntrl+M */
 50 cc=asc(" ")
 60 dc=asc("ミ")
 70 print isprint(ac)
 80 print isprint(bc)
 90 print isprint(cc)
100 print isprint(dc)
110 end
```

ISPUNCT

標準関数

書　式　ispunct(ch)

引　数　char

戻り値　int

機　能　キャラクタコードchの文字が表示可能な記号文字(&H20～&H2F、&H3A～&H40、&H5B～&H60、&H7B～&H7E)であるかどうか調べます。表示可能な記号文字であれば－1、そうでなければ0を返します。

ch ……調べたい文字のキャラクタコード(0～255)

用　例

```
10 /* ispunct sample
20 char ac,bc,cc
30 ac=asc("|")
40 bc=asc("+")
50 cc=asc(".")
60 print ispunct(ac)
70 print ispunct(bc)
80 print ispunct(cc)
90 end
```

ISSPACE

標準関数

書　式　isspace(ch)

引　数　char

戻り値　int

機　能　キャラクタコードchの文字が空白文字(&H09～&H0D、&H20)であるかどうか調べます。空白文字であれば－1、そうでなければ0を返します。

ch ……調べたい文字のキャラクタコード(0～255)

用　例

```
10 /* isspace sample
20 char ac,bc,cc
30 ac=asc("a")
40 bc=asc("+")
50 cc=asc(" ")
60 print isspace(ac)
70 print isspace(bc)
80 print isspace(cc)
90 end
```

ISUPPER

標準関数

書　式　isupper(ch)

引　数　char

戻り値　int

機　能　キャラクタコードchの文字が英大文字('A'～'Z')であるかどうか調べます。英大文字であれば－1、そうでなければ0を返します。

ch ……調べたい文字のキャラクタコード(0～255)

用　例

```
10 /* isupper sample
20 char ac,bc,cc
30 ac=asc("a")
40 bc=asc("A")
50 cc=asc("5")
60 print isupper(ac)
70 print isupper(bc)
80 print isupper(cc)
90 end
```

ISXDIGIT

標準関数

書　式　isxdigit(ch)

引　数　char

戻り値　int

機　能　キャラクタコードchの文字が16進数の文字　('0'～'9'、'a'～'f'、'A'～'F')　かどうか調べます。16進数の文字であれば－1、そうでなければ0を返します。

ch ……調べたい文字のキャラクタコード(0～255)

用　例

```
 10 /* is?????? sample
 20 int ai
 30 char ac
 40 cls
 50 color 3
 60 print "'isxdigit'の文字のチェックを行います（該当すれば黄色で表示し、該当
しなければ白色で表示します）"
 70 print "★---コントロールコード"
 80 print "□---スペース（空白）"
 90 print "●---シフトＪＩＳ（２バイト）コードの先頭コード"
100 print "◆---シフトＪＩＳ（２バイト）コードの先頭コード"
110 print
120 for ai=1 to 256
130     ac=ai-1
140     if ac<>0 and (ac mod 16)=0 then print
150     if ac>=0 and ac<=31 then {
160         if isxdigit(ac)=-1 then color 2:print "★";:continue else print "
★";:continue}
170     if ac=32 or ac=127 or ac=160 then {
180         if isxdigit(ac)=-1 then color 2:print "□";:continue else print "
□";:continue}
190     if ac>=128 and ac<=159 then {
200         if isxdigit(ac)=-1 then color 2:print "●";:continue else print "
●";:continue}
210     if ac>=224 then {
220         if isxdigit(ac)=-1 then color 2:print "◆";:continue else print "
◆";:continue}
230     if isxdigit(ac)=-1 then color 2 else color 3
240     print " "+chr$(ac);
250 next
260 end
```

ITOA

標準関数

書　式　itoa(i)

引　数　int

戻り値　str

機　能　整数値iを文字列に変換し、その文字列を返します。

➜　atoi

用　例

```
10 /* itoa sample
20 int ai,bi
30 ai=300000
40 bi=-512
50 print itoa(ai)
60 print itoa(bi)
70 print itoa(ai)+itoa(bi)
80 end
```

KEY

ステートメント

書　式　key　〈ファンクションキーナンバー〉,〈文字列〉

機　能　ファンクションキーの内容を定義します。〈ファンクションキーナンバー〉は1～20で、1～10はF1～F10キーを、11～20はSHIFTキーを押しながらF1～F10キーを押した場合に対応します。

〈文字列〉の長さは32文字まで有効です。CTRL＋A（CTRLキーを押しながらAキーを押す)、CTRL＋M（CTRLキーを押しながらMキーを押す)のような文字を定義するには、"@A"、"@M"のように"@"と対応する各大文字を並べて使います。

なお引用符(")は"@@"となります。

➜　「コントロールコード一覧」

用　例

```
10 /* key sample
20 key 3,"renum@M"
30 key 11,"@@X68000@@"
40 key 19,"endfunc"
50 end
```

KEY LIST

コマンド

書　式　key　list

機　能　ファンクションキーの内容一覧を表示します。

用　例

```
key list
```

KILL

コマンド

書　式　kill　"〔<ドライブ名>〕<ファイル名>"

省略形　ki.

機　能　指定したファイルを削除します。<ドライブ名>、<ファイル名>の詳細については、「Human68kユーザーズマニュアル」を参照してください。

用　例

```
kill "a:¥test.dat"
```

LEFT$

標準関数

書　式　left$ (st, i)

引　数　str(st), int(i)

戻り値　str

機　能　文字列stの先頭から指定された文字数iの文字列を取り出して返します。

st　……元の文字列(文字式)

i ……取り出したい文字数

文字数iは、1から255の値をとります。

指定された文字数iが、文字列stの総文字数より大きいときは、文字列stをそのまま取り出して返します。

指定された文字数iが、0または負の場合はnull文字を返します。

➜　mid$、right$

用　例

```
10 /* left$ sample
20 str as
30 as="personal workstation X68000    "
40 print left$(as,8)
50 end
```

LINE

書　式　line(x1,y1,x2,y2,p[,ls])

引　数　int

戻り値　void

機　能　グラフィック画面上に直線を引きます。

x1 ……始点X座標
y1 ……始点Y座標
x2 ……終点X座標
y2 ……終点Y座標
p……パレットコード(色)
ls ……ラインスタイル

x1、y1、x2、y2は、－32768から32767までの値をとることができます。クリッピングエリアはwindow関数で指定された範囲になります。また、パレットコードpは0から65535までの値をとることができ、その最大値はグラフィック画面の実画面サイズ及び色モードによります。

ラインスタイルlsは0～65535の値をとり、これを16ビットのビットパターンと見なし、実線(&HFFFF)や点線(&HAAAA)などを指定することができます。

用　例

```
 10 /* line sample
 20 int ai,bi,ci,di,ei,fi,gi
 30 screen 1,3,1,1
 40 vpage(1)
 50 for ai=0 to 100
 60     bi=rand() mod 512
 70     ci=rand() mod 512
 80     di=rand() mod 512
 90     ei=rand() mod 512
100     fi=rnd()*65535
110     gi=65535
120     line(bi,ci,di,ei,fi,gi)
130 next
140 wipe()
150 for ai=0 to 100
160     bi=rand() mod 512
170     ci=rand() mod 512
180     di=rand() mod 512
190     ei=rand() mod 512
200     fi=rnd()*65535
210     gi=rnd()*65535
220     line(bi,ci,di,ei,fi,gi)
230 next
240 end
```

LINPUT

ステートメント

書　式　linput　〔"〈プロンプト文〉";〕〈str型変数名〉

機　能　指定されたstr型変数に、キーボードから文字列を入力します。文字列を入力したら、リターンキーを押してください。input命令と違い、一度に一つのstr型変数にしか代入できず、また数値型の変数への代入はできません。カンマ(,)、引用符(")なども代入することができます。

linput文では、疑問符(?)は表示されません。また、str型変数に入力できる最大文字数は、その表示画面の水平最大表示文字数－(プロンプト文字数＋1)となります。ただし、設定されている文字バッファのサイズ以上のデータは代入されません。

用　例

```
10 /* linput sample
20 str as,bs[255]
30 as="X68000"
40 while bs<>as
50      print "end of string='X68000'"
60      linput "input string (1line)--->";bs
70      print bs
80 endwhile
90 end
```

LIST

コマンド

書　式　〔l〕list　〔〈行番号１〉〕〔－〔〈行番号２〉〕〕

省略形　〔l〕l.

機　能　メモリにあるプログラムリストを画面に表示したり、プリンタに印刷します。
表示を開始する〈行番号１〉を省略すると、プログラムリストの先頭から表示が始まり、表示を終了する〈行番号２〉を省略すると、プログラムリストの最後までが表示されます。たとえば、

list	プログラムリストの先頭から最後までが表示されます。
list　50	行番号50の行のみ表示されます。
list　－100	プログラムリストの先頭から100行目までが表示されます。
list　200－	プログラムリストの200行目から最後までが表示されます。

listの機能を一時停止したいときは、CTRL + S キーを押します。何かキーを押せば、表示を再開します。また、listの機能を終了したいときは、BREAK キー（またはCTRL + C キー）を押します。
list命令の代わりにllist命令を用いれば、プログラムリストをプリンタに印刷することができます。

用　例

```
list 10-200
```

LOAD

コマンド

書　式　load〔@〕　"〔<ドライブ名>〕<ファイル名>"〔,<行番号>〕〔,<増分>〕

省略形　lo.

機　能　指定したBASICプログラムをロードします。
プログラムがロードされると、それまでメモリ上にあったプログラムは消されます。
load@は行番号を除いて(save@によって)セーブされたプログラムに<行番号>とその<増分>を指定(しなければともに10)してロードします。この場合、プログラムは、指定された行番号から、指定された増分を加えた行番号を付加しながらロードされます。すでにプログラムがメモリにある場合、ロードしたプログラムがマージ(混合、追加)されます。<ドライブ名>、<ファイル名>についての詳細は、「Human68kユーザーズマニュアル」を参照してください。

➜　save

用　例

```
load "a:¥test.dat"
```

LOCATE

ステートメント

書　式　locate　<X座標>,<Y座標>〔,<カーソルスイッチ>〕

省略形　loc.

機　能　カーソルをテキスト画面の<X座標>、<Y座標>で指定されたキャラクタ座標位置へ移動します。<X座標>、<Y座標>のとりうる範囲は、スクリーンモードとスクロールエリアによります。<カーソルスイッチ>が0のときは、カーソル表示OFFで、1ならカーソル表示ONです。
なお、<カーソルスイッチ>が有効なときは、プログラムで<カーソルスイッチ>を0にして実行している間のみです。プログラムが終了して"Ok"が表示されるとカーソル表示ONになります。

用　例

```
 10 /* locate sample
 20 int ai,bi
 30 cls
 40 for ai=0 to 20
 50     locate ai*4,ai,1:print "personal workstation X68000   "
 60     locate 95-ai*4,ai+10,1:print "personal workstation X68000   "
 70     for bi=0 to 5000
 80     next
 90 next
100 end
```

LOG

標準関数

書　式　log(n)

引　数　float

戻り値　float

機　能　数値nの自然対数を返します。n＞0でなければなりません。

用　例

```
 10 /* log sample
 20 int ai,bi,ci,di
 30 float af
 40 screen 1,3,1,1
 50 vpage(1)
 60 line(0,255,511,255,65535,&HFFFF)
 70 for ai=1 to 3
 80     for bi=1 to 511
 90         af=bi/150.05222#
100         ci=255-log(af)*ai*60
110         di=rnd()*65535
120         pset(bi,ci,di)
130     next
140 next
150 end
```

MID$

標準関数

書　式　mid$ (st, i1, i2)

引　数　str (st), int (i1, i2)

戻り値　str

機　能　文字列stの中の指定された位置i1から指定された文字数i2分の文字列を取り出して返します。

st　……元の文字列(文字式)
i1　……取り出し始める位置
i2　……取り出したい文字数

位置i1および文字数i2は、1から255の値をとります。
取り出したい文字数i2が文字列stの取り出し始める位置i1からの文字数よりも大きいときは、指定された位置i1以降のすべての文字列を取り出して返します。取り出し始める位置i1が、0あるいは負の場合や、文字列stの長さを越えている場合はnull文字を返します。取り出したい文字数i2が、0または負の場合もnull文字を返します。

➜　left$、right$

用　例

```
10 /* mid$ sample
20 str as
30 as="personal workstation X68000    "
40 print mid$(as,10,11)
50 end
```

MIRROR$

標準関数

書　式　mirror$ (st)

引　数　str

戻り値　str

機　能　文字列stの並びを逆転して返します。

用　例

```
10 /* mirror$ sample
20 str as
30 as="personal workstation X68000    "
40 print mirror$(as)
50 end
```

MOUSE

書　式　mouse(ch)

引　数　char

戻り値　int

機・能　マウスカーソルの表示、マウスの初期化などを行います。戻り値は、ch＝3以外では、通常0、エラーのときは－1を返します。

ch ……0＝マウスの初期化(右ボタン専有……ソフトキーボード使用可)
1＝マウスカーソルを表示する
2＝マウスカーソルを消す
3＝マウスカーソルの表示状態を返す(表示している…－1、表示していない…0)
4＝マウスの初期化(右ボタン開放……ソフトキーボード使用不可)

用　例

```
 10 /* mouse sample
 20 char ac
 30 mouse(0)
 40 while -1
 50     print "マウスの初期化--->0"
 60     print "マウスカーソルの表示--->1"
 70     print "マウスカーソルの消去--->2"
 80     print "マウスカーソルの表示状態--->3"
 90     print "ソフトキーボードの表示禁止--->4"
100     input "いずれかの値を入力して下さい(0---4)";ac
110     switch ac
120     case 0:print "マウスの初期化をしました":mouse(0):break
130     case 1:print "マウスカーソルを表示しました":mouse(1):break
140     case 2:print "マウスカーソルを消去しました":mouse(2):break
150     case 3:print "マウスカーソルの表示状態=====";
160     if mouse(3)<>0 then print "マウスカーソル表示中":break
170     if mouse(3)=0 then print "マウスカーソル消去中":break
180     case 4:print "ソフトキーボードを表示禁止にしました":mouse(4)
190     endswitch
200 endwhile
210 end
```

MSAREA

MOUSE

書　式　msarea(x1,y1,x2,y2)

引　数　int

戻り値　int

機　能　マウスカーソルの移動範囲を指定します。範囲は表示画面サイズによって決まります。
戻り値は通常0、エラーのときは－1を返します。

x1 ……始点X座標

y1 ……始点Y座標

x2 ……終点X座標

y2 ……終点Y座標

用　例

```
10 /* msarea sample
20 screen 2,0,1,1
30 vpage(1)
40 mouse(0)
50 msarea(0,0,600,400)
60 box(0,0,600,400,15,&HFFFF)
70 mouse(1)
80 locate 1,5
90 print "(0,0)<--->(600,400)の範囲でマウスを動かしてください!"
100 end
```

MSBTN

書　式　msbtn(n, b, t)

引　数　char(n, b), int(t)

戻り値　int

機　能　マウスボタンが指定された状態になるまでの時間を返します。戻り値は、0ならばドラッグされたことを示し、－1ならば指定された待ち時間の最大値tを越えたことを示します。

n……待つボタンの状態
　　0＝ボタンが離されるまで
　　1＝ボタンが押されるまで
b……ボタンの左右
　　0＝左
　　1＝右
t ……待ち時間の最大値(0、1でずっと待つ)

状態が変わればこれはリセットされます。

用　例

```
 10 /* msbtn sample
 20 int ai=1,bi,ci,di,ei,fi,gi,hi,ii,ji,ki
 30 screen 2,0,1,1
 40 vpage(1)
 50 mouse(0)
 60 msarea(0,0,767,511)
 70 box(0,0,767,511,15,&HFFFF)
 80 mouse(1)
 90 locate 5,10
100 print "マウスの左ボタンをダブルクリックしてください。十字の線が動きます。!
!"
110 while ai<>0
120     msstat(bi,ci,di,ei)
130         if di<>-1 then continue
140     fi=msbtn(0,0,80)
150         if fi<10 or fi>70 then continue
160     fi=msbtn(1,0,80)
170         if fi<10 or fi>70 then continue
180     fi=msbtn(0,0,80)
190         if fi<10 or fi>70 then continue
200     line(ii,0,ii,511,0,&HFFFF)
210     line(0,ji,767,ji,0,&HFFFF)
220     box(0,0,767,511,15,&HFFFF)
230     mspos(gi,hi)
240     line(gi,0,gi,511,12,&HFFFF)
250     line(0,hi,767,hi,12,&HFFFF)
260     ii=gi
270     ji=hi
280 endwhile
290 end
```

MSPOS

MOUSE

書　式　mspos(x, y)

引　数　int型変数名

戻り値　int

機　能　マウスカーソルの座標をint型変数x、yに返します。

用　例

```
 10 /* mspos sample
 20 int ai=1,bi,ci
 30 screen 2,0,1,1
 40 vpage(1)
 50 mouse(0)
 60 msarea(0,0,767,511)
 70 box(0,0,767,511,15,&HFFFF)
 80 mouse(1)
 90 locate 10,10,0
100 print "move mouse cursor!(0,0)<--->(767,511)"
110 while ai<>0
120     mspos(bi,ci)
130     locate 10,11,0
140     print "mouse cursor (X) = ";bi
150     locate 10,12,0
160     print "mouse cursor (Y) = ";ci
170 endwhile
180 end
```

MSSTAT

書　式　msstat (x, y, bl, br)

引　数　int型変数名

戻り値　int

機　能　マウスの状態をint型変数x、y、bl、brに返します。

x……X方向の相対移動量(－128～127)

y……Y方向の相対移動量(－128～127)

bl　……左ボタンの状態(押されていれば－1、押されていなければ0を返す)

br　……右ボタンの状態(押されていれば－1、押されていなければ0を返す)

用　例

```
10 /* msstat sample
20 int ai=1,bi,ci,di,ei
30 screen 2,0,1,1
40 vpage(1)
50 mouse(0)
60 msarea(0,0,767,511)
70 box(0,0,767,511,15,&HFFFF)
80 mouse(1)
90 locate 7,10,0
100 print "(0,0)<--->(767,511)の範囲でマウスを動かしてください!"
110 while ai<>0
120     msstat(bi,ci,di,ei)
130     locate 7,11,0
140     print "マウスのX方向の相対移動量(-128---127)---> ";bi
150     locate 7,12,0
160     print "マウスのY方向の相対移動量(-128---127)---> ";ci
170     locate 7,13,0
180     print "マウスの左ボタンの状態---> ";
190     if di<>0 then print "押されています  " else print "押されていません"
200     locate 7,14,0
210     print "マウスの右ボタンの状態---> ";
220     if ei<>0 then print "押されています  " else print "押されていません"
230 endwhile
240 end
```

M_ALLOC

MUSIC

書　式　m_alloc(t,s)

引　数　char(t) ,int(s)

戻り値　int

機　能　指定されたトラックtのトラックバッファを確保します。

t……トラック番号(1～80)

s……バッファサイズ(1～65536)

全トラックバッファ内のデータはクリアされます。
また、バッファサイズに1～3を指定しても、4バイト確保します。
tを省略すると、すべてのトラックをバッファサイズSで確保します。
なお、総トラックのバッファサイズ（それぞれのトラックに割り当てられているトラックバッファの総合計）は、CONFIG.SYSファイル内で指定してください。

DEVICE = ¥SYS¥OPMDRV3.X #nnn

(nnn：総トラックバッファサイズ[単位：KB])

戻り値として、正常のときは0、エラーのときは－1を返します。

M_ANTOFF

MUSIC3

書　式　m_antoff(c)

引　数　char

戻り値　int

機　能　指定のチャンネルcへオールノートオフを出力します。

c……チャンネル番号(1～25)

cが省略された場合、すべてのチャンネルへオールノートオフを出力します。
戻り値として、正常のときは0、エラーのときは－1を返します。

M_ASSIGN

MUSIC

書　式　m_assign(c,t)

引　数　char

戻り値　int

機　能　指定のチャンネルcで使用するトラックtを設定します。

c……チャンネル番号(1～25)

t……トラック番号(1～80)

また、tが省略された場合は、チャンネルcに設定されているトラック番号を返します。複数のチャンネルに同一のトラックを割りあてると、それぞれ同じ楽譜データを演奏します。また、複数のトラックを１つのチャンネルに割りあてると、最も新しく割りふられたトラックの楽譜データを演奏します。

デフォルト値は、チャンネル１～８が、それぞれトラック１～８に対応します。

戻り値として、正常のときは０、エラーのときは－１を返します。

M_BEND

MUSIC3

書　式　m_bend(c,b)

引　数　char

戻り値　int

機　能　指定のチャンネルのピッチベンドを設定します。

c……チャンネル番号(1～25)

b……ピッチベンドデータ(0～127)

cが省略された場合は、すべてのチャンネルにピッチベンドbを設定します。bは省略できません。

戻り値として、正常のときは０、エラーのときは－１を返します。

M_CHAN

MUSIC3

書　式　m_chan(c1,c2)

引　数　char

戻り値　int

機　能　指定のチャンネルのデータ出力チャンネルを一時的に変更します。

c1　……チャンネル番号(1～25)

c2　……出力チャンネル番号(1～25)

c2が省略された場合は、現在の設定値を返します。

m_assign関数やMMLデータ@Nとは異なり、チャンネル変更前のパラメータ（トラック番号、音色番号、ボリュームなど）を引き継ぎます。

m_init関数実行まで有効です。

戻り値として、正常のときは0、エラーのときは－1を返します。

➜ m_init

M_CONT

MUSIC

書　式　m_cont([c1][,c2][,c3][,c4][,c5][,c6][,c7][,c8])

引　数　char

戻り値　int

機　能　c1からc8で指定されたチャンネルにおいて、一時停止した演奏を再開します。

c1～c8……チャンネル番号(1～25)

c1からc8をすべて省略すると、現在一時停止しているすべてのチャンネルの演奏を再開します。

戻り値として、正常のときは0、エラーのときは－1を返します。

➜ m_stop

M_CTRLRES

MUSIC3

書　式　m_ctrlres(c)

引　数　char

戻り値　int

機　能　指定のチャンネルcへピッチベンドオフ／モジュレーションオフ／ダンパーオフ／オールノートオフを出力します。

c……チャンネル番号(1～25)

cが省略された場合、すべてのチャンネルへピッチベンドオフ／モジュレーションオフ／ダンパーオフ／オールノートオフを出力します。
戻り値として、正常のときは0、エラーのときは－1を返します。

M_DIROUT

MUSIC3

書　式　m_dirout(lng,ca)

引　数　int(lng) ,char型一次元配列名(ca)

戻り値　int

機　能　lngバイトのMIDIデータをMIDIへ出力します。

lng……lngのとりうる値は1から配列名caで宣言された配列の添字＋1までです。

ca ……MIDIデータを格納しているchar型一次元配列名

➡ m_out

M_END

MUSIC3

書　式　m_end(m)

引　数　int

戻り値　int

機　能　演奏を停止する小節を設定します。

m……演奏を中止する小節番号

すべてのチャンネルの”|:”、”D.S”、”D.C”、”:|”などの位置が同じでないと正常な演奏ができません。
戻り値として、正常のときは0、エラーのときは－1を返します。

➜ m_start

M_ERRGET

MUSIC3

書　式　m_errget()

引　数　int

機　能　直前のMUSIC関数のエラーコードを返します。
正常終了している場合は0を返し、エラーの場合は1～65のエラーコードを返します。
エラーコードについては、「資料編　エラーコード一覧」を参照してください。

M_FREE

MUSIC

書　式　m_free(t)

引　数　char

戻り値　int

機　能　指定のトラックtのトラックバッファの残りバイト数を返します。

t ……トラック番号(1～80)

➜ m_freea、m_use

M_FREEA

MUSIC3

書　式　m-freea()

引　数　int

機　能　ドライバ起動時に確保したトラックバッファの全バイト数を返します。

➜ m_free、m-use

M_IFCHK

MUSIC3

書　式　m_ifchk()

引　数　int

機　能　MIDIインターフェイスの有無をチェックします。
インターフェイスがつながっている場合は0、つながっていない場合は－1を返します。

M_INIT

MUSIC

書　式　m_init(md)

引　数　char(md)

戻り値　int

機　能　トラックバッファを初期化します。各トラックバッファは4バイトに設定されます。また、各チャンネルのパラメータ（トラック番号、音色番号、ボリュームなど）も初期化されます。

mdに1を指定した場合、FM音源の音色データも初期化します。

mdに0を設定または省略した場合は、音色データは初期化しません。

md……初期化モード

0＝トラックバッファの初期化

1＝トラックバッファ、音色データの初期化

M_MDREG

MUSIC3

書　式　m_mdreg(gr,adr,dt)

引　数　char

戻り値　int

機　能　MIDI制御IC（YM3802）のレジスタに値を書き込みます。grにはグループナンバー、adrにはアドレスナンバーを指定します。dtにはレジスタに書き込む値を0～255の範囲で指定します。

戻り値として、正常のときは0、エラーのときは－1を返します。

注：間違ったパラメータを指定すると、システムが正常に動作しなくなる場合があります。

YM3802の機能を十分に理解の上でご使用ください。

M_MEAS

MUSIC3

書　式　m_meas()

引　数　int

機　能　現在演奏中の小節番号を返します。
MMLデータ”|:”、”D.S.”などのくり返し記号の場合は、くり返し回数がそのまま加算されますので、実際の小節番号とは異なる場合があります。

➡ m_end、m_start

M_METER

MUSIC3

書　式　m_meter(n,d)

引　数　char

戻り値　int

機　能　曲の拍子を設定します。

n……分子（2～16)
d……分母（2，4，8，16)

デフォルト値は4/4拍子に設定されています。
戻り値として、正常のときは0、エラーのときは－1を返します。

M_MOD

MUSIC3

書　式　m_mod(c,m)

引　数　char

戻り値　int

機　能　指定のチャンネルcのモジュレーションを設定します。

c……チャンネル番号(1～25)

m　……モジュレーションデータ(0～127)

cが省略された場合は、すべてのチャンネルにモジュレーションmを設定します。
mは省略できません。
戻り値として、正常のときは0、エラーのときは－1を返します。

➜ m_modsns

M_MODSNS

MUSIC3

書　式　m_modsns(m)

引　数　char

戻り値　int

機　能　FM音源全体のモジュレーションの深さを設定します。

m　……モジュレーションの深さ(0～127)

mが省略された場合は、現在の設定値を返します。
戻り値として、正常のときは0、エラーのときは－1を返します。

➜ m_mod

M_MSTVOL

MUSIC3

書　式　m_mstvol(vol)

引　数　char

戻り値　int

機　能　マスターボリュームを設定します。

vol…… ボリュームデータ(0～127)

デフォルト値は127

volが省略された場合、現在の設定値を返します。
m_init関数を実行するまで有効です。
PCMのチャンネルに対しては無効です。
戻り値として、正常のときは0、エラーのときは－1を返します。

➜ m_init、m_vol

M_MUTE

MUSIC3

書　式　m_mute([c1][,c2][,c3][,c4][,c5][,c6][,c7][,c8])

引　数　char

戻り値　int

機　能　c1からc8で指定したチャンネルの音をミュート（消音）します。
ノート以外のデータは常に出力されます。

c1～c8……チャンネル番号(1～25)

c1からc8を省略すると、ミュートが解除されます。
戻り値として、正常のときは0、エラーのときは－1を返します。

➜ m_solo

M_NTOFF

MUSIC3

書　式　m_ntoff(c,nt,vel)

引　数　char

戻り値　int

機　能　指定のチャンネルへノートオフを出力します。

c……チャンネル番号(1～25)
nt　……ノート番号(0～127)
vel……ベロシティデータ　(0～127)

velが省略された場合、ベロシティ値を64で出力します。
戻り値として、正常のときは0、エラーのときは－1を返します。

➜ m_nton

M_NTON

MUSIC3

書　式　m_nton(c,nt,vel)

引　数　char

戻り値　int

機　能　指定のチャンネルへノートオンを出力します。

c……チャンネル番号(1～25)
nt　……ノート番号(0～127)
vel……ベロシティデータ(0～127)

velが省略された場合、ベロシティ値を64で出力します。
戻り値として、正常のときは0、エラーのときは－1を返します。

➜ m_ntoff

M_OPMEXC

MUSIC3

書　式　m_opmexc(md)

引　数　char

戻り値　int

機　能　FM音源にセットする音色パラメータのうち、LFQ/PMD/AMDなどの全体に影響するパラメータを音色切り替え時にセットするかどうかを選択します。

md……パラメータの設定スイッチ
0＝セットしない
1＝セットする　（デフォルト値は1）

mdが省略された場合、現在の状態を返します。
戻り値として、正常のときは0、エラーのときは－1を返します。

➜ m_vset

M_OPMLFQ

MUSIC3

書　式　m_opmlfq(spd)

引　数　char

戻り値　int

機　能　FM音源のLFOの発振周波数（スピード）を設定します。

spd……スピード(0～255、デフォルト値は198)

spdが省略された場合、現在のデータを返します。
戻り値として、正常のときは0、エラーのときは－1を返します。

M_OPMREG　　MUSIC3

書　式　m_opmreg(reg,data)

引　数　char

戻り値　int

機　能　指定のOPMレジスタにデータをセットします。

reg……OPMレジスタ番号(0～255)
data　……OPMへ書き込むデータ

dataが省略された場合、現在のregで示すOPMレジスタの内容を返します。
一度も指定のレジスタが書き換えられていない場合は、－1を返します。
戻り値として、正常のときは0、エラーのときは－1を返します。

M_OUT　　MUSIC3

書　式　m_out(data)

引　数　char

戻り値　int

機　能　1バイトデータをMIDIへ出力します。

data　……MIDIデータ(0～255)

戻り値として、正常のときは0、エラーのときは－1を返します。

➡ m_dirout

M_PAN

書　式　m_pan(c,pan)

引　数　char

戻り値　int

機　能　指定のチャンネルcにパンポットpanを設定します。

c……チャンネル番号(1～25)

pan……パンポットデータ(0～127)

デフォルト値は64

0＝ライト(右端)

.

.

64＝センター(中央)

.

.

127＝レフト(左端)

cが省略された場合は、すべてのチャンネルにpanを設定します。

panが省略された場合は、現在の設定値を返します。

戻り値として、正常のときは0、エラーのときは－1を返します。

M_PANFLT

MUSIC3

書　式　m_panflt(c,md)

引　数　char

戻り値　int

機　能　MMLデータのPANの切り替えコマンドPnを、演奏時に出力するかどうかを選択します。

c……チャンネル番号(1～25)
md……0＝Pnを出力する
　　1＝Pnを出力しない

cが省略された場合は、すべてのチャンネルにmdを設定します。
mdが省略された場合は、現在の設定値を返します。
戻り値として、正常のときは0、エラーのときは－1を返します。

➜ m_pgmflt、m_volflt

M_PCMBSY

MUSIC3

書　式　m_pcmbsy()

引　数　int

機　能　PCMの実行状態を調べます。

&H00：何も実行していない
&H02：出力中(ADPCMOUT,m_pcmon,a_playを実行中)
&H04：入力中(ADPCMINP,m_pcmrec,a_recを実行中)
&H12：出力中(ADPCMAOTを実行中)
&H14：入力中(ADPCMAINを実行中)
&H22：出力中(ADPCMLOTを実行中)
&H24：入力中(ADPCMLINを実行中)

M_PCMCLR

MUSIC3

書　式　m_pcmclr()

引　数　int

機　能　PCMバッファをクリアします。
m_pcmset関数などで登録したPCMデータが失われます。
戻り値として、正常のときは0、エラーのときは－1を返します。

M_PCMGET

MUSIC3

書　式　m_pcmget(nt,ca)

引　数　char(nt),char型一次元配列名(ca)

戻り値　int

機　能　登録されているPCMデータをcaに読み込みます。

nt ……ノート番号
ca ……PCMデータバッファを格納するためのchar型一次元配列名

戻り値として、正常のときは0、エラーのときは－1を返します。

➜ m_pcmlen、m_pcmset

M_PCMLEN

MUSIC3

書　式　m_pcmlen(nt)

引　数　char

戻り値　int

機　能　ノート番号ntに登録されているPCMデータのサイズを返します。

nt　……ノート番号(0～127)

ntを省略すると、PCMバッファの残り容量を返します。
戻り値として、正常のときは0、エラーのときは－1を返します。

➜ m_pcmset、m_pcmget

M_PCMON

書　式　m_pcmon(nt,sf,md)

引　数　char

戻り値　int

機　能　指定されたPCMデータを再生します。

nt ……ノート番号(0～127)

sf ……サンプリング周波数

0 ……3.9kHz

1 ……5.2kHz

2 ……7.8kHz

3 ……10.4kHz

4 ……15.6kHz

音声を再生する場合、原則としてm_pcmrec関数で録音したときのサンプリング周波数sfを使用してください。

それ以外を使用すると、録音した音声を正常に再生できません。

md……音声出力モード(オーディオ出力端子)

1 ……左

2 ……右

3 ……ステレオ(左右両方)

sf、mdが省略された場合は、sf＝4　md＝3で再生します。

戻り値として、正常のときは0、エラーのときは－1を返します。

➡ m_pcmrec、m_pcmset

用　例

```
100 /*pcm sample
110 int ai
130 dim char aca(24575)
140 ai=fopen("B:¥SAMPLE¥sample.PCM","r")
150 fread(aca,24576,ai)
160 fclose(ai)
170 m_pcmset(3,4,24576,aca)
180 m_pcmon(3)
```

M_PCMREC

MUSIC3

書　式　m_pcmrec(sf,lng,ca)

引　数　char(sf),int(lng),char型一次元配列名(ca)

戻り値　int

機　能　外部入力音（オーディオ入力端子）をPCM録音します。
実行直後に入力待ちになり、約4秒以内に入力がなければ実行を中止します。

sf　……サンプリング周波数
　0　……3.9kHz（1秒間に消費するメモリサイズ1950バイト）
　1　……5.2kHz（1秒間に消費するメモリサイズ2600バイト）
　2　……7.8kHz（1秒間に消費するメモリサイズ3900バイト）
　3　……10.4kHz（1秒間に消費するメモリサイズ5200バイト）
　4　……15.6kHz（1秒間に消費するメモリサイズ7800バイト）
lng……録音する配列caの添字0からの長さ（最大＝32767）
　lngのとりうる値は、1から、配列名caで宣言された配列の添字＋1までです。
ca……　PCMを格納するchar型一次元配列名
lngを省略すると、すべてのPCMデータを録音します。
戻り値として、正常のときは0、エラーのときは－1を返します。

➜ m_pcmon、m_pcmset

M_PCMSET

書　式　m_pcmset(nt,sf,lng,ca)

引　数　char(nt,sf)、int(lng)、char型一次元配列名(ca)

戻り値　int

機　能　指定の音程にPCM音を登録します。

指定できる音程は'C-2'(0)から'G＃8'(127)までです。

nt ……ノート番号(0～127)

sf ……サンプリング周波数

0 ……3.9kHz

1 ……5.2kHz

2 ……7.8kHz

3 ……10.4kHz

4 ……15.6kHz

lng……登録する配列caの添字0からの長さ(最大＝32767)

lngのとりうる値は、1から、配列名caで宣言された配列の添字＋1までです。

ca ……PCMデータが格納されている一次元配列名

戻り値として、正常のときは0、エラーのときは－1を返します。

➜ m_pcmget、m_pcmlen、m_pcmon、m_pcmrec

用　例

```
100 /*pcm sample
110 int ai
130 dim char aca(20000)
140 ai=fopen("B:¥SAMPLE¥sample.PCM","r")
150 fread(aca,20000,ai)
160 fclose(ai)
170 m_pcmset(3,4,20000,aca)
180 m_pcmon(3)
```

M_PGMFLT　　MUSIC3

書　式　m_pgmflt(c,md)

引　数　char

戻り値　int

機　能　MMLデータの音色の切り替えコマンド@nを、指定チャンネルの演奏時に出力するかどうかを選択します。

c……チャンネル番号(1～25)
md……0＝@nを出力する
　　1＝@nを出力しない

cが省略された場合は、すべてのチャンネルにmdを設定します。
mdが省略された場合は、現在の設定値を返します。
戻り値として、正常のときは0、エラーのときは－1を返します。

➜ m_panflt、m_volflt

M_PLAY　　MUSIC

書　式　m_play([c1][,c2][,c3][,c4][,c5][,c6][,c7][,c8])

引　数　char

戻り値　int

機　能　c1～c8で指定されたチャンネルに割り当てられたトラックの楽譜データを演奏します。

c1～c8……チャンネル番号(1～25)

c1～c8をすべて省略すると、全チャンネルを演奏します。
戻り値として、正常のときは0、エラーのときは－1を返します。

用　例

```
 10 /* MUSIC SAMPLE
 20 /*
 30 /* SYMPHONIA  NO.14 in Bb major J.S BACH
 40 /*
 50 str A$[255]
 60 /* PART 1
 70 m_init()
 80 for I=1 to 6 : m_alloc(I,2000):next
 90 for I=1 to 6 : m_assign(I,I):next
100 /*
110 m_trk(1,"Q7 @34 V9   O4 T66") /* TRUMPET
120 m_trk(2,"Q7 @35 V10  O4 ")    /* HORN
```

```
130 m_trk(3,"Q7 @37 V10  O3 ")     /* TUBA
140 m_trk(4,"Q8 @6  V8   O5 ")     /* CEMBALO
150 m_trk(5,"Q8 @6  V8   O5 ")     /* CEMBALO
160 m_trk(6,"Q8 @6  V8   O4 ")     /* CEMBALO
170 /* PART 1
180 A$="L16 B-4&B-B-AGF8G8DE-F8 : <R8F8&FE-DC>B-8<D8>AB-<C8"
190 A$=A$+">F<FD>B-<E-8G8&GE-C>A&A8<F8 : D4E4F8>B-4A8"
200 TRK_WRT(1):TRK_WRT(4)
210 /* PART 2
220 A$="L16R1 : >B-FB-<DC8E-8&E-8DE {FEFEFEFE}8 &ED32E32"
230 A$=A$+"F4G8B-8E-8G8<C8C8& : C>FB-<D>G8B-8A8DE-FGE-F"
240 TRK_WRT(2):TRK_WRT(5)
250 /* PART 3
260 A$=   "L8 B-<DC>B-AGFE- : D4F4G4<C4"
270 A$=A$+"D4&L16DD>B-G<C8E-8&E-C>AF : B-4&B-B-AGF8G8DE-F8"
280 TRK_WRT(3):TRK_WRT(6)
290 m_play()
300 end
310 func TRK_WRT(T)
320   m_trk(T,A$)
330 endfunc
```

M_PNMGET

MUSIC3

書　式　m_pnmget(nt)

引　数　char

戻り値　str

機　能　ノート番号ntに登録されているPCMの音色名を取得します。

nt　……ノート番号(0～127)

➜ m_pnmset

M_PNMSET

MUSIC3

書　式　m_pnmset(nt,pn)

引　数　char(nt) ,str (pn)

戻り値　int

機　能　ノート番号ntにPCMの音色名を登録します（半角10文字以内)。

nt ……ノート番号(0～127)

pn ……PCMの音色名

戻り値として、正常のときは0、エラーのときは－1を返します。

➡ m_pnmget

M_PROG

MUSIC3

書　式　m_prog(c,p)

引　数　char

戻り値　int

機　能　指定のチャンネルcに音色番号pで示される音色を設定します。

c……チャンネル番号(1～25)

p……音色番号(FM音源は1～200、MIDIは1～128)

cが省略された場合は、すべてのチャンネルにvoを設定します。

pが省略された場合は、現在の設定値を返します。

戻り値として、正常のときは0、エラーのときは－1を返します。

M_SNDSET

MUSIC3

書　式　m_sndset(ca)

引　数　char型一次元配列(ca)

戻り値　int

機　能　SOUND　PRO-68KのSNDファイル形式のデータをFM音源用のデータとして登録します。
同時に音色名も登録します。

ca ……SNDファイル形式のデータ
1音色80バイトで最大200音色(＝16000バイト)まで指定できます。

戻り値として、正常のときは0、エラーのときは－1を返します。

M_SOLO

MUSIC3

書　式　m_solo(c)

引　数　char

戻り値　int

機　能　指定されたチャンネルc以外のすべての音をミュートします。

c……チャンネル番号(1～25)

cを省略すると解除されます。
戻り値として、正常のときは0、エラーのときは－1を返します。

➜ m_mute

M_START

MUSIC3

書　式　m_start(m)

引　数　int

戻り値　int

機　能　演奏を開始する小節を設定します。

m　……演奏を開始する小節番号

m_play関数を実行することにより、小節番号mの小節より演奏を開始します。すべてのチャンネルのMMLデータ“｜：”、“D.S.”、“D.C.”、“：｜”などの数や位置が同じでないと正常な演奏ができません（一度設定された小節番号は、演奏が終わるとクリアされます)。
戻り値として、正常のときは0、エラーのときは－1を返します。

➔ m_end、m_meas、m_play

M_STAT

書　式　m_stat(c)

引　数　char

戻り値　int

機　能　指定のチャンネルが演奏中なら1を、停止中なら0を返します。

c……チャンネル番号(1～25)

cが省略された場合は、すべてのチャンネルの状態をbit 0 ～24に返します。
bit0～24がチャンネル1～25に対応しています。それぞれのbitが0ならば停止中、1ならば演奏中です。

用　例

```
 10 /* m_stat sample
 20 m_init()
 30 m_alloc(1,1000)
 40 m_alloc(2,1000)
 50 m_assign(2,1)
 60 m_assign(3,1)
 70 m_assign(4,2)
 80 m_trk(1,"l12 crccrdereeredrcdrecrc>g<rr")
 90 m_trk(1,"    e3  rfgrggrgfrefrge3  rg")
100 m_trk(1,"    e3  rge3  rgergerge3  rr")
110 m_trk(2,"l4 rrrrrrrr")
120 m_play(1,4)
130 while m_stat(4)
140 endwhile
150 m_play(2,4)
160 while m_stat(4)
170 endwhile
180 m_play(3,4)
190 end
```

M_STOP

MUSIC

書　式　m_stop([c1][,c2][,c3][,c4][,c5][,c6][,c7][,c8])

引　数　char

戻り値　int

機　能　c1～c8で指定されたチャンネルの演奏を一時停止します。

c1～c8……チャンネル番号(1～25)

チャンネルをすべて省略すると、全チャンネルの演奏を一時停止します。
戻り値として、正常のときは0、エラーのときは－1を返します。

➔ m_cont

用　例

```
 10 /*
 20 /* music sample
 30 /* "おお、スザンナ!!"        S.C.Foster
 40 /*
 50 str as,bs,cs,ds,part[255]
 60 m_init()
 70 m_alloc(1,2000)
 80 m_assign(1,1)
 90 linput "音色データの番号を入力して下さい(1～68)--->?";as
100 linput "テンポを入力してください(32～200)--->?";bs
110 part="@"+as+" o4 v9 q7 l16"+" t"+bs
120 wr_trk(part)
130 part="g&a b8<d8d8e8> <d8>b8g8.a b8b8a8g8 a4.ga&"
140 wr_trk(part)
150 part="b8<d8d8.e> <d8>b8g8.a& b8b8a8a8 g4.ga&"
160 wr_trk(part)
170 part="b8<d8d8.e> <d8>b8g8.a b8b8a8g8 a4r8ga&"
180 wr_trk(part)
190 part="b8<d8d8e8> <d8.>bg8.a bb8.a8.a g4r4"
200 wr_trk(part)
210 part="<c4c4> <e8e4e8> <d8.d>b8g8 a4r8ga&"
220 wr_trk(part)
230 part="b8<d8d8.e> <d8>b8g8a8 b8b8a8.a g4r8"
240 wr_trk(part)
250 m_play()
260 linput "演奏を停止しますか<y/n>--->?";cs
270 if cs="n" then end
280 m_stop()
290 linput "演奏を再開しますか<y/n>--->?";ds
300 if ds="y" then m_cont()
310 end
320 func wr_trk(part;str)
330 m_trk(1,part)
340 endfunc
```

M_SYNC

MUSIC3

書　式　m_sync(c)

引　数　char(vo)

戻り値　int

機　能　同期モードを設定します。

同期モード

c……0　内部のMIDIクロックに合わせて演奏
1　外部のMIDIクロックに合わせて演奏
2　外部のFSKクロックに合わせて演奏

外部同期にする場合は、m_sync(1)実行後　m_play関数で待機状態になりMIDIのスタートコマンド($FA)を待ちます。
cを省略すると、現在の状態を返します。
戻り値として、正常のときは0、エラーのときは－1を返します。

M_SYSCH

MUSIC3

書　式　m_sysch(cm)

引　数　str

戻り値　int

機　能　各音源に対するチャンネル番号の割り当てを変更します。
(ドライバ組み込み時のオプションスイッチと同じ)

cm ……各音源のチャンネルの割り当てを示す文字列

"OPM"……	OPM=ch1～8	PCM=ch9	MIDI=ch10～25
"OMP"……	OPM=ch1～8	MIDI=ch9～24	PCM=ch25
"MOP"……	MIDI=ch1～16	OPM=ch17～24	PCM=ch25
"MPO"……	MIDI=ch1～16	PCM=ch17	OPM=ch18～25
"POM"……	PCM=ch1	OPM=ch2～9	MIDI=ch10～25
"PMO"……	PCM=ch1	MIDI=ch2～17	OPM=ch18～25

cmを省略した場合は、現在のモードを数値で返します。
(0="OPM"、1="OMP"、2="MOP"、3="MPO"、4="POM"、5="PMO")
戻り値として、正常のときは0、エラーのときは－1を返します。

M_TEMPO

MUSIC

書　式　m_tempo(te)

引　数　int

戻り値　int

機　能　テンポteを1分間に4分音符を何回打つかで指定します。
テンポは、20～300の範囲です（デフォルト値は120)。
teが省略された場合は、現在のテンポを返します。
戻り値として、正常のときは0、エラーのときは－1を返します。

用　例

```
 10 /*
 20 /* m_tempo sample
 30 /* "おお、スザンナ!!"        S.C.Foster
 40 /*
 50 int ai
 60 char ac
 70 str as,part[255]
 80 m_init()
 90 m_alloc(1,2000)
100 m_assign(1,1)
110 linput "音色データの番号を入力して下さい(1～68)--->?";as
120 input "テンポを入力してください(32～200)--->";ac
130 part="@"+as+" o4 v9 q7 l16"
140 wr_trk(part)
150 part="g&a b8<d8d8e8> <d8>b8g8.a b8b8a8g8 a4.ga&"
160 wr_trk(part)
170 part="b8<d8d8.e> <d8>b8g8.a& b8b8a8a8 g4.ga&"
180 wr_trk(part)
190 part="b8<d8d8.e> <d8>b8g8.a b8b8a8g8 a4r8ga&"
200 wr_trk(part)
210 part="b8<d8d8e8> <d8.>bg8.a bb8.a8.a g4r4"
220 wr_trk(part)
230 part="<c4c4> <e8e4e8> <d8.d>b8g8 a4r8ga&"
240 wr_trk(part)
250 part="b8<d8d8.e> <d8>b8g8a8 b8b8a8.a g4r8"
260 wr_trk(part)
270 m_play()
280 for ai=0 to 30000:next
290 m_tempo(ac)
300 for ai=0 to 40000:next
310 m_tempo(120)
320 end
330 func wr_trk(part;str)
340 m_trk(1,part)
350 endfunc
```

M_TNMGET

MUSIC3

書　式　m_tnmget(vo)

引　数　char

戻り値　str

機　能　音色番号voで示される音色データに登録されている音色名を取得します。

vo ……音色番号(1～200)

戻り値として音色名を示す文字列を返します（半角で10文字以内)。

➡ m_tnmset

M_TNMSET

MUSIC3

書　式　m_tnmset(vo,tn)

引　数　char(vo) ,str(tn)

戻り値　int

機　能　音色番号voで示される音色データに音色名を登録します。

vo ……音色番号(1～200)

tn ……音色名を示す文字列(半角10文字以内)

戻り値として、正常のときは0、エラーのときは－1を返します。

➡ m_tnmget

M_TRK

MUSIC

書　式　m_trk(t,st)

引　数　char(t) ,str(st)

戻り値　int

機　能　MMLによる楽譜データをトラックtに書き込みます。厳密にいうとトラックtに楽譜データを追加していきます。

t……トラック番号(1～80)

st　……MML(ミュージックマクロランゲージ)データ

MMLデータの終端は、ヌル文字（0）です。

戻り値として、正常のときは0、エラーのときは－1を返します。

M_TRNS

MUSIC3

書　式　m_trns(c,stp)

引　数　char

戻り値　int

機　能　指定のチャンネルcの音程をシフトします(半音ごとに±2オクターブの範囲)。
stpが省略された場合は、現在の設定値を返します。

c……チャンネル番号

stp…… 音程を移調するデータ(0～48)

デフォルト値は24

0で－2オクターブ、48で＋2オクターブ

cが省略された場合は、すべてのチャンネルにstpを設定します。

戻り値として、正常のときは0、エラーのときは－1を返します。

M_USE

MUSIC3

書　式　m_use(t)

引　数　char

戻り値　int

機　能　指定のトラックバッファｔの使用容量を返します。

t……トラック番号(1～80)

tを省略した場合は、すべてのトラックバッファの合計使用容量を返します。
戻り値として、正常のときは0、エラーのときは－1を返します。

M_VEL

MUSIC3

書　式　m_vel(c,vel)

引　数　char

戻り値　int

機　能　指定のチャンネルcのベロシティを設定します。

c……チャンネル番号(1～25)
vel……ベロシティデータ(0～127)

cが省略された場合は、すべてのチャンネルにvelを設定します。
velが省略された場合は、現在の設定値を返します。
戻り値として、正常のときは0、エラーのときは－1を返します。

M_VGET

MUSIC

書　式　m_vget(vo,ca)

引　数　char(vo),char型二次元(4,10)配列名(ca)

戻り値　int

機　能　音色データを配列caに読み込みます。

vo ……音色番号(1～200)

ca ……音色データを格納するためのchar型二次元配列名

配列の次元は、(4,10)

戻り値として、正常のときは0、エラーのときは－1を返します。

➜ m_vset

用　例

```
10 /* m_vget sample
20 int ai,bi
30 char ac
40 dim char aca(4,10)
50 input "音色データの番号を入力してください(1～68)--->";ac
60 cls
70 print "音色データの番号(";ac;")のデータです"
80 print
90 m_vget(ac,aca)
100 for ai=0 to 10
110     print
120     for bi=0 to 4
130     print using "#####";aca(bi,ai);:print "    ";
140     next
150 next
160 end
```

M_VOL

MUSIC3

書　式　m_vol(c,vol)

引　数　char

戻り値　int

機　能　指定のチャンネルcにボリュームvolを設定します。

c……チャンネル番号(1～25)

vol……ボリュームデータ(0～127)

cが省略された場合は、すべてのチャンネルにvolを設定します。
volが省略された場合は、現在の設定値を返します。
戻り値として、正常のときは0、エラーのときは－1を返します。

➜ m_mstvol

M_VOLFLT

MUSIC3

書　式　m_volflt(c,md)

引　数　char

戻り値　int

機　能　MMLデータのボリューム設定コマンドVn, @Vnを、指定チャンネルの演奏時に出力するかどうかを選択します。

c……チャンネル番号(1～25)

md……0＝Vn, @Vnを出力する

1＝Vn, @Vnを出力しない

cが省略された場合は、すべてのチャンネルにmdを設定します。
mdが省略された場合は、現在の設定値を返します。
戻り値として、正常のときは0、エラーのときは－1を返します。

➜ m_panflt、m_pgnflt

M_VSET

MUSIC

書　式　m_vset(vo,ca)

引　数　char(vo),char型二次元(4,10)配列名(ca)

戻り値　int

機　能　配列caに格納されている音色データを指定された音色番号voのメモリバッファに登録します。

vo ……音色番号(1～200)　1～68は登録済
ca ……音色データが格納されているchar型二次元配列名
　　配列の次元は(4,10)

音色データの配列形式

	0		1～4(オペレータ)	
0	フィードバック／アルゴリズム	(0～63)	AR	(0～31)
1	スロットマスク	(0～15)	D1R	(0～31)
2	ウェーブフォーム	(0～3)	D2R	(0～31)
3	シンクロ	(0,1)	RR	(0～15)
4	スピード	(0～255)	D1L	(0～15)
5	PMD	(0～127)	TL	(0～127)
6	AMD	(0～7)	KS	(0～3)
7	PMS	(0～3)	MUL	(0～15)
8	AMS	(0～3)	DT1	(0～7)
9	L,RPAN	(0～3)	DT2	(0～3)
10			AMSイネーブル	(0,1)

なお、オペレータ1はM1(モジュレータ1)、オペレータ2はC1(キャリア1)、オペレータ3はM2(モジュレータ2)、およびオペレータ4はC2(キャリア2)に対応しています。

戻り値として、正常のときは0、エラーのときは－1を返します。

➜ m_vget

＊音色データの詳細は、『Human68kユーザーズマニュアル』の「7.3.2　FM音源の音色データ（レジスタ)」を参照してください。

M_YCOM

書　式　m_ycom(md)

引　数　char

戻り値　int

機　能　md＝0でYコマンドをMIDIのコントロールデータと解釈します。
md＝1でYコマンドをOPMレジスタの書き換えと解釈します。
mdを省略した場合は、現在のモードを返します（デフォルト＝1）。
戻り値として、正常のときは0、エラーのときは－1を返します。

NAME

コマンド

書　式　name　"〔<ドライブ名>〕<ファイル名1>", "<ファイル名2>"

省略形　na.

機　能　指定したドライブのファイル名<ファイル名1>を<ファイル名2>に変更します。<ドライブ名>、<ファイル名>についての詳細は、「Human68kユーザーズマニュアル」を参照してください。

用　例

```
name "a:¥test.bas","a:¥test.dat"
```

NEW

コマンド

書　式　new

機　能　メモリにあるプログラムを消去し、すべての変数を初期化します。
new命令を実行すると、メモリにあるプログラムはすべて消去され、定義されている変数もすべて初期化されます。

➜　delete

OCT$

書　式　oct$ (i)

引　数　int

戻り値　str

機　能　整数値iを８進表現の文字列に変換し返します。なお、上位の"0"は取り除かれます。

➡　bin$、hex$

用　例

```
10 /* oct$ sample
20 int ai,bi
30 ai=-255
40 bi=100542300
50 print oct$(ai)
60 print oct$(bi)
70 print oct$(ai)+oct$(bi)
80 end
```

PAINT

GRAPH

書　式　paint(x, y, p)

引　数　int

戻り値　void

機　能　グラフィック画面上の任意の単色領域内を塗りつぶします。
window関数で指定されたクリッピングエリア外の領域は無視されます。

x……塗りつぶしたい領域内の1点のX座標
y……塗りつぶしたい領域内の1点のY座標
p……パレットコード(色)

x、yは、−32768から32767までの値をとることができます。また、パレットコードpは0から65535までの値をとることができ、その最大値はグラフィック画面の実画面サイズおよび色モードによります。

用　例

```
10 /* paint sample
20 int ai,bi,ci,di,ei,fi
30 screen 1,3,1,1
40 vpage(1)
50 for ai=0 to 50
60     bi=rand() mod 512
70     ci=rand() mod 512
80     di=rand() mod 200
90     ei=rnd()*65535
100    circle(bi,ci,di,ei,0,360,400)
110 next
120 for ai=0 to 300
130    p()
140 next
150 wipe()
160 for ai=0 to 50
170    bi=rand() mod 512
180    ci=rand() mod 512
190    di=rand() mod 512
200    ei=rand() mod 512
210    fi=rnd()*65535
220    box(bi,ci,di,ei,fi,&HFFFF)
230 next
240 for ai=0 to 300
250    p()
260 next
270 end
280 func p()
290 paint(rnd()*511,rnd()*511,rnd()*65535)
300 endfunc
```

PALET

書　式　palet(p,c)

引　数　int

戻り値　void

機　能　グラフィック画面の各パレットにカラーコードcを割りあてます。

p……パレットコード(0～255)

c……カラーコード(0～65535)

このパレットコードpの最大値はグラフィック画面の実画面サイズおよび色モードによります。

なお、グラフィック画面の実画面サイズ及び色モードが512×512、65536色の場合は、パレットコードpとカラーコードcは1対1に対応し、同値として扱われるので、このpalet関数は無効となります。この場合パレットコードpは0から65535の値をとることができます。

用　例

```
10 /* palet sample
20 int ai=15,bi,ci,di
30 dim int aia(7)
40 screen 2,0,1,1
50 vpage(1)
60 fill(100,100,667,411,15):aia(0)=65535  /* color code = 65535
70 fill(120,120,647,391,13):aia(1)=65473  /* color code = 65473
80 fill(140,140,627,371,11):aia(2)=63551  /* color code = 63551
90 fill(160,160,607,351,9):aia(3)=63489   /* color code = 63489
100 fill(180,180,587,331,7):aia(4)=2047   /* color code = 2047
110 fill(200,200,567,311,5):aia(5)=1985   /* color code = 1985
120 fill(220,220,547,291,3):aia(6)=63     /* color code = 63
130 fill(240,240,527,271,1):aia(7)=1      /* color code = 1
140 while -1
150 ci=bi mod 8
160 palet(ai,aia(ci))
170 if ai=1 then ai=15:continue
180 ai=ai-2
190 bi=bi+1
200 for di=0 to 300
210 next
220 endwhile
230 end
```

PI

標準関数

書　式	pi([n])
引　数	float
戻り値	float
機　能	円周率のn倍を返します。数値nを省略するとn＝1と見なされます。

n……倍数

用　例

```
10 /* pi sample
20 print pi()
30 print pi(2)
40 print pi(4.675#)
50 end
```

POINT

書　式　point(x,y)

引　数　int

戻り値　int

機　能　グラフィック画面上のドットのパレットコードを返します。

x……ドットのX座標

y……ドットのY座標

x、yはwindow関数で指定されたクリッピングエリア内の値をとることができます。戻り値として、0から65535のパレットコードを返します。ただし、そのパレットコードの最大値はグラフィック画面の実画面サイズおよび色モードによります。

用　例

```
10 /* point sample
20 int ai=255,bi=255,ci,di=1,ei=1,fi
30 screen 2,0,1,1
40 vpage(1)
50 line(100,100,667,100,3,&HFFFF)
60 line(100,100,100,411,9,&HFFFF)
70 line(100,411,667,411,11,&HFFFF)
80 line(667,100,667,411,13,&HFFFF)
90 while -1
100 ai=ai+di
110 bi=bi+ei
120 ci=point(ai,bi)
130 switch ci
140     case 3:ei=1:fi=1:break
150     case 9:di=1:fi=1:break
160     case 11:ei=-1:fi=1:break
170     case 13:di=-1:fi=1:break
180 endswitch
190 if fi=1 then fi=0:continue
200 pset(ai,bi,5)
210 endwhile
220 end
```

POS

システム変数

書　式　pos

戻り値　int

機　能　テキスト画面の現在のカーソルのx座標を返します。代入することはできません。

用　例

```
10 /* pos sample
20 print "X68000    "
30 print "X68000    HD      ";
40 print pos
50 end
```

POW

標準関数

書　式　pow(n1,n2)

引　数　float

戻り値　float

機　能　数値n1のn2乗の計算をします。n1＝0かつn2＜0の場合とn1＜0かつn2が整数でない場合は、エラーとなります。

用　例

```
10 /* pow sample
20 float af,bf,cf,df
30 af=5.3#
40 bf=2.8#
50 cf=2.23#
60 df=-2.87#
70 print pow(af,bf)
80 print pow(cf,df)
90 end
```

PRINT

ステートメント

書　式　〔l〕print　〔using　〈書式制御文字列〉〕〔〈式〉...〕

省略形　〔l〕p.

機　能　テキスト画面に文字列や数値を表示します。

print命令に続けて数値や文字列を指定すると、その数値、文字列がテキスト画面に表示されます。また、数値型変数やstr型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テキスト画面に表示されます。

print命令だけを指定すると改行が行われます。

数値を表示する場合、その後ろに必ず空白が１つ入ります。また、数値の前には符号用の桁が用意され、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、それらの区切りとして、カンマ(,)、セミコロン(;)が使えます。カンマを使うとprint命令の後ろのデータはテキスト画面の一番左の列に表示され、８桁ごとに区切って表示されます。区切りにセミコロンを使うとprint命令の後ろのデータがつながってテキスト画面に表示されます。一番最後のデータの後ろにセミコロンを置くと、つぎのprint文のデータが前のデータにつながって表示されます。

lprint命令を使うと、テキスト画面の代わりにプリンタに出力することができます。

print命令の省略形として？(疑問符)を使うことができます。

print命令の後に、usingと書式指定子を続けることによって、様々なフォーマットでデータを表示、出力することができます。

print　using命令を使用した場合は、複数データを扱うことはできません。必ず、１つのデータになります。

１．数値型

#	表示する数値の桁数を指定します。数値の桁数が、指定された桁数より少なければ右詰めで表示、出力されます。
.	小数点の位置を指定します。はみ出す部分は四捨五入されます。
＋－	正負の符号をつけます。負の符号は、#の列の後にのみ設定できます。
＊＊	空白の部分をアスタリスク(＊)で埋めます。
￥￥	数値の先頭に￥マークを付けます。
＊＊￥	空白をアスタリスク(＊)で埋め、先頭に￥マークをつけます。
^^^^^	数値を指数表現で表示、出力します。
,	３桁ごとにカンマ(,)をふります。

２．文字型

!	文字列の先頭の１文字のみを表示、出力します。
&(空白)&	文字列を&から&までの文字数分だけ表示、出力します。
@	文字列をすべて表示、出力します。

３．その他

_(アンダースコア)	この後の書式指定子の１文字は、単なる文字として表示、出力します。書式指定子以外の文字があった場合は、そのまま表示、出力されます。

用　例

```
 10 /* print using sample
 20 print using "!";"SHARP X68000   "
 30 print using "&       &";"SHARP X68000   "
 40 print using "######";680.4086#
 50 print using "######.#";680.0686#
 60 print using "+#####.#";680.0286#
 70 print using "#####.#-";-680.0586#
 80 print using "******.#";6800.186#
 90 print using "¥¥####.#";680.8786#
100 print using "**¥##.#";680.9886#
110 print using "#####,.#";6800.086#
120 print using "#####.#^^^^";1234.56#
130 print using "¥¥###._-";680.0086#
140 print using "#####";12345678.5488#
150 print using "This is a @. ";"X68000   "
160 end
```

PSET

書　式　pset(x,y,p)

引　数　int

戻り値　void

機　能　グラフィック画面上にドットを表示します。

x……ドットのX座標

y……ドットのY座標

p……パレットコード(色)

x、yは、−32768から32767までの値をとることができます。クリッピングエリアについてはwindow関数で指定された範囲になります。またパレットコードpは0から65535までの値をとることができ、その最大値はグラフィック画面の実画面サイズおよび色モードによります。

用　例

```
10 /* pset sample
20 int ai,bi,ci,di
30 float af
40 screen 1,3,1,1
50 vpage(1)
60 for ai=1 to 5
70     for bi=0 to 511
80         af=(2*3.141**bi)/512*ai
90         ci=255-sin(af)*200
100        di=rnd()*65535
110        pset(bi,ci,di)
120    next
130 next
140 end
```

PUT

GRAPH

書　式　put(x1,y1,x2,y2,na)

引　数　int(x1,y1,x2,y2)，数値型一次元配列名(na)

戻り値　void

機　能　get関数で読み込んだドットパターンを指定された配列naから読み出してグラフィック画面上の指定領域に表示します。

x1 ……始点X座標
y1 ……始点Y座標
x2 ……終点X座標
y2 ……終点Y座標
na ……ドットパターンが格納されている数値型一次元配列名

x1、y1、x2、y2は、window関数で指定された範囲内の値をとることができます。また、配列na内のデータ構造については、get関数を参照してください。
とくに、put関数を実行する前にあらかじめget関数で読みこんだときと同じグラフィック画面の実画面サイズおよび色モードにして、領域の大きさや配列naのデータ型、添字などは同じになるように指定してください。

➜　get

用　例

```
 10 /* get & put sample
 20 int ai,bi,ci
 30 dim int aia(4899),bia(4899)    /* 140*70/2bytes
 40 screen 1,3,1,1
 50 vpage(1)
 60 rainbow()
 70 box(0,0,139,69,65535,&HFFFF)
 80 get(0,0,139,69,aia)
 90 /* save gram data to file
100 ai=fopen("a:test.dat","c")
110     fwrite(aia,4900,ai)
120 fclose(ai)
130 wipe()
140 /* load gram data from file
150 ai=fopen("a:test.dat","rw")
160     fread(bia,4900,ai)
170 fclose(ai)
180 wipe()
190 for ai=0 to 10
200     bi=ai*30
210     put(bi,bi,bi+139,bi+69,bia)
220     for ci=0 to 1000:next
230 next
240 end
250 func rainbow()
260 for ai=0 to 63
270     bi=ai*3
280     circle(69,69,ai,hsv(bi,31,31)+1,10,170,380)
290 next
300 endfunc
```

RAND

標準関数

書　式　rand()

戻り値　int

機　能　0以上32767以内の整数型乱数を返します。

➡　randomize、rnd、srand

用　例

```
10 /* rand sample
20 int ai
30 for ai=0 to 10
40     print rand()
50 next
60 end
```

RANDOMIZE

標準関数

書　式　randomize(i)

引　数　int

戻り値　void

機　能　rnd()関数の乱数系列を初期化します。

i ……乱数のシード(−32768≦i≦32767)

➡　rand、rnd、srand

用　例

```
10 /* randomize sample
20 int ai,bi
30 for ai=0 to 2
40     randomize(int(32767*rnd()-32768*rnd()))
50     for bi=0 to 10
60         print rnd()
70     next
80 next
90 end
```

REM

ステートメント

書　式　／＊〔〈注釈文〉〕

機　能　プログラム中に注釈を入れます。

／＊に続く文字列は、プログラム中の注釈になるだけで、プログラムの実行にはまったく関係しません。プログラムを見やすくする覚え書きとして使用してください。

用　例

```
10 /* rem (/*) sample
20 /*
30 /* super personal workstation
40 /*
50 print "X68000   "
60 end
```

RENUM

コマンド

書　式　renum　〔〈新行番号〉〕〔,〈旧行番号〉〕〔,〈増分〉〕

省略形　ren.

機　能　プログラムの行番号を付け直します。

〈新行番号〉、〈増分〉とも、省略すると10が採用されます。〈旧行番号〉を指定しないと、そのプログラムの先頭から新しい行番号に変わります。たとえば、

renum

プログラム全体の行番号を付けかえます。新しい行番号は10から始まり、20、30……と付けられます。

renum　100,50

行番号50に新しい行番号として100が付けられ、以降110、120……と付け直されます。

renum　100,,100

プログラム全体の行番号を付けかえます。新しい行番号は100から始まり、200、300……と付けられます。

renum命令は、goto命令、gosub命令、then節、else節などで使われている行番号は変更しません。

REPEAT～UNTIL

ステートメント

書　式

repeat
〈ループの内容〉
until 〈式〉

機　能

〈式〉の値が真になるまで、指定された〈ループの内容〉を繰り返し(ループ)ます。
繰り返しの終了判定は〈ループの内容〉の実行後に行われます。少なくとも、1回は〈ループの内容〉を実行します。

用　例

```
10 /* repeat until sample
20 int ai
30 repeat
40      input "1+2=";ai
50 until ai=3
60 print "ok!"
70 end
```

RGB

GRAPH

書　式

rgb(r, g, b)

引　数

char

戻り値

int

機　能

r(赤)、g(緑)、b(青)の成分を0～31で選び、カラーコードを求めます。なお、このrgb関数によって得られるカラーコードは、0～65534までの偶数値になります。したがって、それぞれの偶数値に1を加えることで0～65535までのカラーコードを指定できます。

用　例

```
10 /* rgb sample
20 int ai,bi,ci
30 screen 1,3,1,1
40 vpage(1)
50 locate 7,4:print "0"
60 locate 4,17:print "Blue"
70 locate 33,4:print "Green"
80 locate 62,4:print "31"
90 locate 6,30:print "31"
100 for ai=0 to 31
110     locate 1,1:print "Red = ";ai
120     for bi=0 to 31
130         for ci=0 to 31
140             fill(bi*14+64,ci*14+64,bi*14+78,ci*14+78,rgb(ai,bi,ci)
150         next
160     next
170 next
180 end
```

RIGHT$

標準関数

書　式　right$ (st, i)

引　数　str (st), int (i)

戻り値　str

機　能　文字列stの終りから指定された文字数i分の文字列を取り出して返します。

st　……元の文字列(文字式)

i ……取り出したい文字数

文字数iは、1から255の値をとります。

指定された文字数iが文字列stの総文字数より大きいときは、文字列stをそのまま取り出して返します。指定された文字数iが、0または負の場合はnull文字を返します。

➜　left$、mid$

用　例

```
10 /* right$ sample
20 str as
30 as="personal workstation X68000   "
40 print right$(as,9)
50 end
```

RND

標準関数

書　式　rnd()

戻り値　float

機　能　0以上1未満の実数型乱数を返します。

➜　rand、randomize、srand

用　例

```
10 /* rnd sample
20 int ai
30 for ai=0 to 10
40     print rnd()
50 next
60 end
```

RUN

コマンド

書　式　run 〔〈行番号〉〕

run "〔〈ドライブ名〉〕〈ファイル名〉"

省略形　r.

機　能　メモリにあるプログラムを実行します。

〈行番号〉を省略すると、先頭行からプログラムを実行します。

ドライブ名、ファイル名が指定された場合は、指定されたファイルをロードして実行します。

〈ドライブ名〉、〈ファイル名〉については「Human68kユーザーズマニュアル」を参照してください。

途中でプログラムの実行を中止したいときは、CTRL + C キーまたは BREAK キーを押します。

SAVE

コマンド

書　式　save〔@〕 "〔〈ドライブ名〉〕〈ファイル名〉"〔,〈行番号1〉〕〔－〔〈行番号2〉〕〕

省略形　sa.

機　能　メモリ上のBASICプログラムを、セーブします。

saveではプログラムをアスキー形式でセーブします。セーブされる行番号の範囲は、list命令と同様に行われます。save@は、行番号を省略した形でプログラムがセーブされます。

〈ドライブ名〉、〈ファイル名〉については「Human68kユーザーズマニュアル」を参照してください。

➡　load

用　例

```
save "a:¥test.dat"
```

SCREEN

ステートメント

書　式　screen 〔〈表示画面サイズ〉〕,〔〈グラフィック画面の実画面サイズ及び色モード〉〕,〈ディスプレイ解像度〉〔,〈グラフィック画面の表示ON/OFF〉〕

省略形　sc .

機　能　画面全体（テキスト画面、グラフィック画面、スプライト画面）のスクリーンモードを設定します。

〈表示画面サイズ〉

0……256×256
1……512×512
2……768×512(〈ディスプレイ解像度〉が1(高解像度モード)のときのみ使用可能)

ただし、〈表示画面サイズ〉が768×512以外のときにのみ、スプライト画面は使用できます。また、〈表示画面サイズ〉が768×512のときは、〈グラフィック画面の実画面サイズ及び色モード〉は、0しか設定できません。

〈グラフィック画面の実画面サイズ及び色モード〉(使用可能なグラフィックページ)

0……1024×1024、16色(グラフィックページ0)
1……512×512、16色(グラフィックページ0～3)
2……512×512、256色(グラフィックページ0、1)
3……512×512、65536色(グラフィックページ0)

ただし、〈グラフィック画面の実画面サイズ及び色モード〉が0以外のときは、〈表示画面サイズ〉は0または1しか設定できません。

〈ディスプレイ解像度〉

0……Low(標準解像度)
1……High(高解像度)

ただし、screen　,,0は〈ディスプレイ解像度〉のみをLowに設定し、screen　,,1は〈ディスプレイ解像度〉のみをHighに設定します。この場合、グラフィックRAMのデータは保存されたままです。〈ディスプレイ解像度〉がLowのときは〈表示画面サイズ〉で768×512は使用できません。

〈グラフィック画面の表示ON/OFF〉

0……表示OFF(グラフィック関数使用不可)
　　グラフィックRAMをグラフィック以外の目的で使用する場合に設定してください。
1……表示ON(グラフィック関数使用可)
　　グラフィックRAMをグラフィック関数などで使用する場合に設定し

てください。(グラフィックRAM初期化)

なお、一旦<グラフィック画面の表示ON/OFF>を1に設定するとすべてのグラフィックRAMのデータは消去されます。特にグラフィックRAMをRAMディスクとして使用している場合は、RAMディスクが破壊されますので、あらかじめRAMディスクのデータをディスクなどにセーブしてください。

screen命令実行後、window関数(グラフィック画面のクリッピングエリア)は表示画面サイズに、home関数(グラフィック画面の実画面における表示画面の左上座標)は(0,0)に再設定されます。また、テキスト画面は、通常表示ON、スプライト画面及びグラフィック画面は表示OFFです。

グラフィック関数を使用する場合は、必ず最初にscreen命令で初期化を行ってください。

SEARCH

コマンド

書　式　〔l〕search　"<検索文字列>"

省略形　〔l〕se.

機　能　指定した<検索文字列>を含むプログラム行をすべて表示します。
lsearch命令を使用すれば、プリンタに出力できます。

用　例

```
search "X68"
```

SETMSPOS

MOUSE

書　式　setmspos(x,y)

引　数　int

戻り値　int

機　能　マウスカーソルの位置を指定します。範囲は、msarea関数によって指定された範囲です。戻り値として、エラーのとき－1を返します。

用　例

```
 10 /* setmspos sample
 20 int ai=1,bi,ci
 30 screen 2,0,1,1
 40 vpage(1)
 50 mouse(0)
 60 msarea(0,0,767,511)
 70 box(0,0,767,511,15,&HFFFF)
 80 mouse(1)
 90 while ai<>0
100     print "mouse cursor area (X) = 0---767"
110     print "mouse cursor area (Y) = 0---511"
120     input "mouse cursor position (X,Y) = ";bi,ci
130     setmspos(bi,ci)
140 endwhile
150 end
```

SGN

標準関数

書　式　sgn(n)

引　数　float

戻り値　float

機　能　数値nの符号を調べ、正の数だったら1を、nが0の場合は0を、負の数だったら－1を返します。

用　例

```
10 /* sgn sample
20 print sgn(4523453.453#)
30 print sgn(0)
40 print sgn(-442134.24568#)
50 end
```

SIN

標準関数

書　式　sin(n)

引　数　float

戻り値　float

機　能　数値n(ラジアン)の正弦(サイン)を返します。

用　例

```
 10 /* sin sample
 20 int ai,bi,ci,di
 30 float af
 40 screen 1,3,1,1
 50 vpage(1)
 60 line(0,255,511,255,65535,&HFFFF)
 70 for ai=1 to 3
 80     for bi=0 to 511
 90         af=(2*pi()*bi)/512*ai
100         ci=255-sin(af)*200
110         di=rnd()*65535
120         pset(bi,ci,di)
130     next
140 next
150 end
```

SPACE$

標準関数

書　式　space$ (i)

引　数　int

戻り値　str

機　能　長さi文字の空白文字(&H20)からなる文字列を返します。

長さiは1から255の値をとり、0または負の場合にはnull文字を返します。

用　例

```
10 /* space$ sample
20 str as[255]
30 as=space$(254)+"¥"
40 print as
50 end
```

SP_CLR

SPRITE

書　式　sp_clr([cd1][,cd2])

引　数　char

戻り値　int

機　能　パターンコードcd1からパターンコードcd2で指定した範囲のスプライトやバックグラウンドのパターンを初期化します。

パターンコードcd1を省略した場合は、パターンコード0からパターンコードcd2までのパターンを初期化します。パターンコードcd2を省略した場合は、パターンコードcd1で指定したパターンだけ、またパターンコードcd1、cd2とも省略した場合は全パターン(パターンコード0～255)が初期化されます。

cd1、cd2…パターンコード(0～255)

SP_COLOR

SPRITE

書　式　sp_color(p,c[,pb])

引　数　char(p, pb), int(c)

戻り値　int

機　能　スプライトやバックグラウンドのパターン単位にパレットブロックpbを指定し、そのパターンのドット単位に（各パターンにおけるパレットブロックpbの中から）パレットコードpを設定し、そのパレットにカラーコードcを割りあてます。なお、パレットブロックpbを省略したときは、パレットブロック1と見なされます。パレットコード0には透明を設定してください。

戻り値として、指定されたパレットコードpに対する設定前のカラーコードを返します。

p……パレットコード(0～15)

c……カラーコード(0～65535)

pb ……パレットブロック(1～15)

SP_DEF

書　式　sp_def(cd,ca〔, sz〕)

引　数　char(cd, sz), char型一次元(255または63)配列名(ca)

戻り値　int

機　能　スプライトやバックグラウンドのパターンを定義します。

cd ……パターンコード(0～255)

ca ……スプライトやバックグラウンドのパターンのデータが格納されているchar型一次元配列名、要素数256(sz＝1のとき)または64(sz＝0のとき)

sz ……パターンサイズ(0、1)

パターンサイズszが1の場合、定義できるパターンは16×16ドットで、1ドットにつき配列caの1要素を使用します。0番目の要素がパターンの左上端のドット、15番目の要素がパターンの右上端のドットに対応します。この場合の配列caの要素数は256になります。16×16ドットのパターンが使用されるのは、スプライトのパターンや表示画面サイズが512×512の場合のバックグラウンドのパターンを定義するときです。また、パターンサイズszが1のとき、バックグラウンドのテキストページ1しか使用しない場合のパターンコードcdは0から191までの値をとることができ、テキストページ0、1とも使用する場合のパターンコードcdは0から127までの値をとることができます。バックグラウンドのテキストページ0、1とも使用しない（バックグラウンドを使用しない）場合のパターンコードcdは0から255までの値をとることができます。

パターンサイズszが0の場合、定義できるパターンは8×8ドットで、1ドットにつき配列caの1要素を使用します。0番目の要素がパターンの左上端のドット、7番目の要素がパターンの右上端のドットに対応します。この場合の配列caの要素数は64になります。8×8ドットのパターンが使用されるのは表示画面サイズが256×256の場合のバックグラウンドのパターンを定義するときです。

パターンサイズszが1のとき、パターンコードnというパターンが定義されていたとします。この16×16ドットのパターンと同じものを定義するためには、パターンサイズszが0のとき、4つのパターンコードが必要になります。この4つのパターンコードには、16×16ドットのパターンの内、左上の8×8ドットのパターンが、パターンコード4nに、左下の8×8ドットのパターンがパターンコード4n＋1、以下同様に右上が4n＋2、右下が4n＋3に対応します。ただし、パターンサイズszが0のときでも定義できるパターンの最大数は256です。

なお、パターンサイズszを省略すると、パターンサイズszは1と見なされます。また、1ドットのデータは4ビットでこの値がそのドットのパレットコードpとなります。

➡　sp_pat

用　例

```
 10 /* sprite sample
 20 int i,j,di,x1,y1,x2
 30 char pc,pb,vr,hr,cd
 40 float f
 50 /* define sprite-pattern
 60 dim char SP0(255)={
 70       1, 3, 5, 7, 9,11,13,15,15,13,11, 9, 7, 5, 3, 1,
 80       1, 3, 5, 7, 9,11,13,15,15,13,11, 9, 7, 5, 3, 1,
 90       1, 3, 5, 7, 9,11,13,15,15,13,11, 9, 7, 5, 3, 1,
100       1, 3, 5, 7, 9,11,13,15,15,13,11, 9, 7, 5, 3, 1,
110       1, 3, 5, 7, 9,11,13,15,15,13,11, 9, 7, 5, 3, 1,
120       1, 3, 5, 7, 9,11,13,15,15,13,11, 9, 7, 5, 3, 1,
130       1, 3, 5, 7, 9,11,13,15,15,13,11, 9, 7, 5, 3, 1,
140       1, 3, 5, 7, 9,11,13,15,15,13,11, 9, 7, 5, 3, 1,
150       1, 3, 5, 7, 9,11,13,15,15,13,11, 9, 7, 5, 3, 1,
160       1, 3, 5, 7, 9,11,13,15,15,13,11, 9, 7, 5, 3, 1,
170       1, 3, 5, 7, 9,11,13,15,15,13,11, 9, 7, 5, 3, 1,
180       1, 3, 5, 7, 9,11,13,15,15,13,11, 9, 7, 5, 3, 1,
190       1, 3, 5, 7, 9,11,13,15,15,13,11, 9, 7, 5, 3, 1,
200       1, 3, 5, 7, 9,11,13,15,15,13,11, 9, 7, 5, 3, 1,
210       1, 3, 5, 7, 9,11,13,15,15,13,11, 9, 7, 5, 3, 1,
220       1, 3, 5, 7, 9,11,13,15,15,13,11, 9, 7, 5, 3, 1
230 }
240 /* define background-pattern
250 dim char BG0(63)={
260       0, 8, 0, 8, 0, 8, 0, 8,
270       8, 0, 0, 0, 0, 0, 0, 0,
280       0, 0, 0, 0, 0, 0, 0, 8,
290       8, 0, 0, 0, 0, 0, 0, 0,
300       0, 0, 0, 0, 0, 0, 0, 8,
310       8, 0, 0, 0, 0, 0, 0, 0,
320       0, 0, 0, 0, 0, 0, 0, 8,
330       8, 0, 8, 0, 8, 0, 8, 0
340 }
350 dim char BG1(63)={
360       0, 0, 0, 0, 0, 0, 0, 0,
370       0,10, 0,10, 0,10, 0, 0,
380       0, 0, 0, 0, 0, 0,10, 0,
390       0,10, 0, 0, 0, 0, 0, 0,
400       0, 0, 0, 0, 0, 0,10, 0,
410       0,10, 0, 0, 0, 0, 0, 0,
420       0, 0,10, 0,10, 0,10, 0,
430       0, 0, 0, 0, 0, 0, 0, 0
440 }
450 /* main */
460 screen 0,3,1,1          /* 表示画面サイズ 256*256 (768*512以外ならスプライト
画面使用可能)
470 sp_init()             /* スプライト画面の初期化
480 sp_clr(0,255)         /* スプライトパターンのクリア (0以上255以下のパターン
コード)
490 sp_disp(1)            /* スプライト画面の表示 (0なら表示せず、1なら表示する
)
500 sp_on(0,1)            /* スプライトプレーンの表示 (0以上127以下のプレーン番
号)
510 /* set sprite-pattern & background-pattern
520 sp_def(0,SP0,1)       /* 16*16 dots
530 sp_def(4,BG0,0)       /*  8* 8 dots
540 sp_def(5,BG1,0)       /*  8* 8 dots
550 /* set palet-block
560 pb=14
570 for i=0 to 15
580     pc=i
590     sp_color(pc,hsv(40,31,i*2+1)+1,pb)
600 next
610 pb=15
620 for i=0 to 15
630     pc=i
640     sp_color(pc,hsv(90,31,i*2+1)+1,pb)
650 next
660 /* set background 0
```

```
670 bg_fill(0,pat_dat(0,0,1,4))
680 bg_set(0,0,1)
690 /* set background 1
700 bg_fill(1,pat_dat(0,0,1,5))
710 bg_set(1,1,1)
720 /* set sprite-plane 0,1 & background 0,1
730 i=0:di=1
740 while -1
750     i=i+di
760     /* background scroll
770     if x1=511 then x1=0
780     if y1=511 then y1=0
790     x1=x1+1
800     y1=y1+1
810     bg_scroll(0,x1,y1)
820     bg_scroll(1,511-x1,511-y1)
830     /* move sprite
840     if i<1 then di=1
850     if i>286 then di=-1
860     f=(pi(2)*i)/288*8
870     x2=128-120*sin(f)*cos(f)*sin(f+pi()/2)
880     /* プライオリティ (sprite > background0 > background1)
890     sp_set(0,x2,i,pat_dat(0,0,15,0),3)
900     /* プライオリティ (background0 > sprite > background1)
910     sp_set(1,i,x2,pat_dat(0,0,14,0),2)
920     for j=0 to 50:next
930 endwhile
940 end
950 func pat_dat(vr,hr,pb,cd)
960 /* vr = 垂直反転 (0なら反転しない、1なら反転する)
970 /* hr = 水平反転 (0なら反転しない、1なら反転する)
980 /* pb = パレットブロック (1以上15以下の値)
990 /* cd = パターンコード (0以上255以下の値)
1000 return(vr*32768+hr*16384+pb*256+cd)
1010 endfunc
```

SP_DISP

SPRITE

書　式　sp_disp(ch)

引　数　char

戻り値　int

機　能　スプライト画面（スプライトやバックグラウンド）の表示の設定をします。なお、スプライト画面は表示画面サイズが768×512以外のとき使用できます。

ch ……0 (スプライト画面を表示しない)
　　1 (スプライト画面を表示する)

SP_INIT

SPRITE

書　式　sp_init()

戻り値　int

機　能　スプライト画面（スプライトやバックグラウンド）を初期化します。初期化するものは、各パターンやパレットなどです。なお、スプライト画面は表示画面サイズが768×512以外のとき使用できます。

SP_MOVE

SPRITE

書　式　sp_move(s,〔x〕,〔y〕〔,cd〕)

引　数　char(s,cd),int(x,y)

戻り値　int

機　能　パターンコードcdのスプライトパターンをプレーンsに割りあて、指定された座標位置にそのスプライトを表示します。

s……プレーン(0～127)
x……X座標(－16～1007)
y……Y座標(－16～1007)
cd ……パターンコード(0～255)

表示画面の左上座標は、sp_setと異なり（0,0）になります。
sp_moveはsp_setのパラメータの一部を標準的な値に設定した関数です。つまり、垂直、水平反転は通常、パレットブロックは1、優先度は3に、自動的に設定されます。より高度な使い方をする場合は、sp_setを使用してください。
バックグラウンドのテキストページ1しか使用しない場合のパターンコードcdは16×16ドットパターンの場合0から191までの値をとることができます。テキストページ0、1とも使用する場合のパターンコードcdは16×16ドットパターンの場合0から127までの値をとることができます。テキストページ0、1とも使用しない（バックグラウンドを使用しない）場合のパターンコードcdは16×16ドットパターンの場合0から255までの値をとることができます。

➜　sp_set

SP_OFF

SPRITE

書　式　sp_off([s1][,s2])

引　数　char

戻り値　int

機　能　プレーンs1からプレーンs2で指定した範囲のスプライトが表示されません。プレーンs1を省略した場合は、プレーン0からプレーンs2までのスプライトが表示されません。プレーンs2を省略した場合は、プレーンs1で指定したスプライトだけ、またプレーンs1、s2とも省略した場合は全スプライト(プレーン0～127)が表示されません。

s1、s2……プレーン(0～127)

SP_ON

SPRITE

書　式　sp_on([s1][,s2])

引　数　char

戻り値　int

機　能　プレーンs1からプレーンs2で指定した範囲のスプライトが表示されます。プレーンs1を省略した場合は、プレーン0からプレーンs2までのスプライトが表示されます。プレーンs2を省略した場合は、プレーンs1で指定したスプライトだけ、またプレーンs1、s2とも省略した場合は全スプライト(プレーン0～127)が表示されます。

s1、s2……プレーン(0～127)

SP_PAT

SPRITE

書　式　sp_pat(cd, ca[, sz])

引　数　char(cd, sz), char型一次元(255または63)配列名(ca)

戻り値　int

機　能　スプライトやバックグラウンドのパターンのデータを、指定された配列caに読み出します。

cd ……パターンコード(0～255)

ca ……スプライトやバックグラウンドのパターンのデータを格納するchar型一次元配列名、要素数256(sz＝1のとき)または64(sz＝0のとき)

sz ……パターンサイズ(0、1)

読み出されたパターンのデータの詳細や、パターンコードcdや配列ca、パターンサイズszの詳細については、sp_def関数を参照してください。

➜　sp_def

用　例

```
 10 /* sp_pat sample
 20 int ai,bi
 30 char ac
 40 dim char aca(255)
 50 screen 1,3,1,1
 60 input "スプライト(16*16ﾄﾞｯﾄ)のパターンコードを入力してください(0～255)--->
";ac
 70 sp_pat(ac,aca,1)
 80 for ai=0 to 255
 90     if ai<>0 and (ai mod 16)=0 then print
100     print using "###";aca(ai);
110 next
120 end
```

SP_SET

書　式　sp_set(s, 〔x〕, 〔y〕, 〔pd〕〔, pr〕)

引　数　char(s, pr), int(x, y, pd)

戻り値　int

機　能　パターンコードcdのスプライトパターンをプレーンsに割りあて、指定された座標位置にそのスプライトを表示します。

s……プレーン(0～127)
x……X座標(0～1023)
y……Y座標(0～1023)
pd ……パターンデータ(0～&HCFFF)
pr ……優先度(0～3)

表示画面の左上座標は(16,16)になります。

パターンデータpdは、次のようなビット構成になっています。(ビット12、13は通常0になります。)

ビット15……垂直反転(0：通常、1：垂直反転)
ビット14……水平反転(0：通常、1：水平反転)
ビット8～11…パレットブロックpb(1～15)
ビット0～7…パターンコードcd(0～255)

垂直反転を指定すると、実際に定義されたパターンが上下反転して表示されます。また、水平反転を指定すると、実際に定義されたパターンが左右反転して表示されます。

パレットブロックpbは各パターンに対して指定するパレットで、詳細については、sp_color関数を参照してください。

テキストページ1しか使用しない場合のパターンコードcdは16×16ドットパターンの場合、0から191までの値をとることができます。テキストページ0、1とも使用する場合のパターンコードcdは16×16ドットパターンの場合0から127までの値をとることができます。テキストページ0、1とも使用しない(バックグラウンドを使用しない)場合のパターンコードcdは16×16ドットパターンの場合0から255までの値をとることができます。

優先度prは、指定する値によってスプライトとバックグラウンド0、1の表示優先順位を次のように設定します。

0……スプライトは表示されない
1……バックグラウンド0＞バックグラウンド1＞スプライト
2……バックグラウンド0＞スプライト＞バックグラウンド1
3……スプライト＞バックグラウンド0＞バックグラウンド1

なお、ここでは左側にあるほど重なったときの表示は優先されることを意味しています。

➜　sp_stat

SP_STAT

SPRITE

書　式　sp_stat(s, md)

引　数　char

戻り値　int

機　能　指定されたスプライトの状態を読み出します。

s ……プレーン(0～127)

md ……モード(0～3)

モードmdには読み出す内容を指定します。

0＝X座標(0～1023)

1＝Y座標(0～1023)

2＝パターンデータpd(0～&HCFFF)

3＝優先度pr(0～3)

モードmdで2や3を指定したときに読み出された内容（パターンデータpdや優先度prなど）の詳細については、sp_set関数を参照してください。

➜　sp_set

SQR

標準関数

書　式　sqr(n)

引　数　float

戻り値　float

機　能　数値nの平方根(ルート)を返します。n≧0でなければなりません。

用　例

```
10 /* sqr sample
20 float af,bf
30 af=5.3#
40 bf=3000.455#
50 print sqr(af)
60 print sqr(bf)
70 end
```

SRAND

標準関数

書　式　srand(i)

引　数　int

戻り値　void

機　能　rand( )関数の乱数系列を初期化します。

i……乱数のシード(0≦i≦65535)

➡　rand、randomize、rnd

用　例

```
10 /* srand sample
20 int ai,bi
30 for ai=0 to 2
40     srand(ai)
50     for bi=0 to 10
60         print rand()
70     next
80 next
90 end
```

STICK

書　式　stick(j)

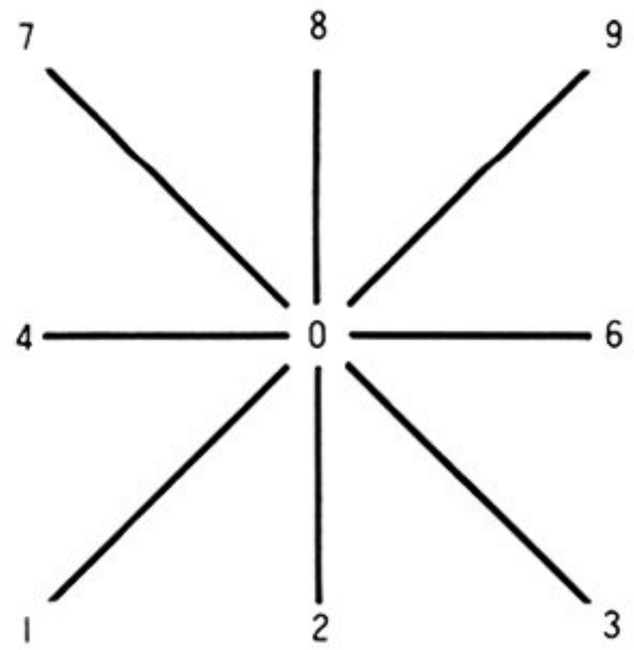
引　数　int

戻り値　int

機　能　ジョイスティックjの状態を調べます。

j ……1＝ジョイスティック1
　　2＝ジョイスティック2

戻り値は次のようになります。



用　例

```
 10 /* stick sample
 20 int ai,bi
 30 cls
 40 while bi<>1
 50      ai=stick(1)
 60      locate 10,11
 70      switch ai
 80      case 1:print "left back ( 左後 )      =";ai:break
 90      case 2:print "back ( 後 )             =";ai:break
100      case 3:print "right back ( 右後 )     =";ai:break
110      case 4:print "left ( 左 )             =";ai:break
120      case 6:print "right ( 右 )            =";ai:break
130      case 7:print "left forward ( 左前 )   =";ai:break
140      case 8:print "forward ( 前 )          =";ai:break
150      case 9:print "right forward ( 右前 ) =";ai:break
160      default:locate 10,10:print "ジョイスティックを動かしてください!"
170      locate 10,11:print "                                  "
180      endswitch
190 endwhile
200 end
```

STOP

ステートメント

書　式　stop

機　能　プログラムの実行を停止します。
プログラムの実行を停止して、直接命令が入力できる状態にします。stop命令はプログラム内のどこにおいてもかまいません。
プログラムを再開するにはcont命令を実行します。

➡　cont

用　例

```
10 /* stop sample
20 print "personal "
30 print "input 'cont'!"
40 stop
50 print "workstation "
60 print "input 'cont'!"
70 stop
80 print "X68000   "
90 end
```

STR

ステートメント

書　式　str　〈変数名〉[[〈文字バッファのサイズ〉]][=〈文字定数式〉][,...]

機　能　変数を文字(str)型として宣言します。同時に文字列を代入することもできます。また、〈文字バッファのサイズ〉を1～255の範囲で設定できます。〈文字バッファのサイズ〉を指定しない場合は32文字と見なされます。str型変数および定数は、常にキャラクタコードが0で終わっています。
原則として、変数宣言はメインプログラムの先頭で行ってください。また、同じ名前の変数をメインプログラム中で再宣言することはできません。

➡　char、float、int

用　例

```
 10 /* str sample
 20 str as="personal workstation   X68000   "
 30 str bs="super personal workstation   X68000    HD"
 40 str cs[255]="super personal workstation   X68000   "+"super personal works
tation   X68000      "+"super personal workstation   X68000    HD   "
 50 str ds="SHARP",es="株式会社"
 60 print as
 70 print bs
 80 cs=cs+cs
 90 print cs
100 print ds,es
110 end
```

STR$

標準関数

書　式　str$ (n)

引　数　float

戻り値　str

機　能　浮動小数値nを文字列に変換して返します。

➡　val

用　例

```
 10 /* str$ sample
 20 float af,bf,cf
 30 str as,bs,cs[40]
 40 af=3000000000000#
 50 bf=2.51235E+200#
 60 cf=-2.3541786563E-300#
 70 as=str$(af)
 80 bs=str$(bf)
 90 cs=str$(cf)
100 print as+bs+cs
110 print bs+cs+as
120 print cs+as+bs
130 end
```

STRCHR

標準関数

書　式　strchr(st, ch)

引　数　str(st), char(ch)

戻り値　int

機　能　文字列stの中で最初にキャラクタコードchの文字を見つけた位置を返します。位置は文字列stの先頭が0になります。見つからなかった場合は、－1を返します。
キャラクタコードchは、0から255の値をとります。

用　例

```
10 /* strchr sample
20 char ac
30 str as
40 as="personal workstation X68000    "
50 ac=asc("w")
60 print strchr(as,ac)
70 end
```

STRCSPN

標準関数

書　式　strcspn(st1, st2)

引　数　str

戻り値　int

機　能　文字列st2の中のどれか１文字を文字列st1の中で最初に見つけた位置を返します。位置は文字列st1の先頭が０になります。見つからなかった場合は、文字列st1の長さを返します。

文字列st2がnull文字の場合も、文字列st1の長さを返します。また文字列st1がnull文字の場合は０を返します。

用　例

```
10 /* strcspn sample
20 str as,bs
30 as="personal workstation X68000   "
40 bs="zk4q"
50 print strcspn(as,bs)
60 end
```

STRIG

STICK

書　式　strig(j)

引　数　int

戻り値　int

機　能　ジョイスティックjのトリガーボタンの状態を調べます。

j ……1＝ジョイスティック1
　　2＝ジョイスティック2

戻り値は次のようになります。

j＝1のとき	0……トリガーA、Bとも押されていない
（ジョイスティック1）	1……トリガーAのみ押されている
	2……トリガーBのみ押されている
	3……トリガーA、Bとも押されている
j＝2のとき	0……トリガーA、Bとも押されていない
（ジョイスティック2）	1……トリガーAのみ押されている
	2……トリガーBのみ押されている
	3……トリガーA、Bとも押されている

用　例

```
   10 /* strig sample
   20 int ai
   30 cls
   40 while -1
   50      locate 10,10
   60      ai=strig(1)
   70      if ai=0 or ai>4 then print "ジョイスティック１のトリガーを押してくださ
い！  ":continue
   80      if ai=1 then print "ジョイスティック１のトリガーＡが押されました    ":
continue
   90      if ai=2 then print "ジョイスティック１のトリガーＢが押されました    ":
continue
  100      if ai=3 then print "ジョイスティック１のトリガーＡ、Ｂが押されました"
  110 endwhile
  120 end
```

STRING$

標準関数

書　式　string$ (i, st)

引　数　int(i), str(st)

戻り値　str

機　能　文字列stの先頭1文字だけでできた、長さi文字の文字列を返します。長さiは1から255の値をとり、 0または負の場合にはnull文字を返します。

用　例

```
10 /* string$ sample
20 str as[255]
30 as=string$(255,"¥")
40 print as
50 end
```

STRLEN(LEN)

標準関数

書　式　strlen(st)

len(st)

引　数　str

戻り値　int

機　能　文字列stの長さ(0～255)を返します。文字列stがnull文字の場合は0を返します。

用　例

```
10 /* strlen (len) sample
20 str as
30 as="personal workstation X68000   "
40 print strlen(as)
50 print len(as)
60 end
```

STRLWR

標準関数

書　式　strlwr(st)

引　数　str型変数名

戻り値　str

機　能　str型変数stの中の英大文字をすべて英小文字に変換します。

用　例

```
10 /* strlwr sample
20 str as
30 as="PERsonal workSTATION X68000    "
40 strlwr(as)
50 print as
60 end
```

STRNSET

標準関数

書　式　strnset(st, ch, i)

引　数　str型変数名(st), char(ch), int(i)

戻り値　str

機　能　str型変数stの先頭から文字数i分をキャラクタコードchの文字に変換します。
キャラクタコードchは、0から255の値をとります。文字数iが0または負のときはstr型変数stの内容は変わりません。

用　例

```
10 /* strnset sample
20 char ac
30 str as
40 as="personal workstation X68000    "
50 ac=asc("¥")
60 strnset(as,ac,8)
70 print as
80 end
```

STRRCHR

標準関数

書　式　strrchr(st, ch)

引　数　str(st), char(ch)

戻り値　int

機　能　文字列stの中でキャラクタコードchの文字が最後に見つかった位置を返します。位置は文字列stの先頭が0になります。先頭でしか見つからなかった場合は0、まったく見つからなかった場合は－1を返します。キャラクタコードchは、0から255の値をとります。

用　例

```
10 /* strrchr sample
20 char ac
30 str as
40 as="personal workstation X68000   "
50 ac=asc("o")
60 print strrchr(as,ac)
70 end
```

STRREV

標準関数

書　式　strrev(st)

引　数　str型変数名

戻り値　str

機　能　str型変数stの並びを逆転します。

用　例

```
10 /* strrev sample
20 str as
30 as="personal workstation X68000   "
40 strrev(as)
50 print as
60 end
```

STRSET

標準関数

書　式　strset(st, ch)

引　数　str型変数名(st), char(ch)

戻り値　str

機　能　str型変数stのすべての文字を、キャラクタコードchの文字に変換します。
キャラクタコードchは、0から255の値をとります。

用　例

```
10 /* strset sample
20 char ac
30 str as
40 as="personal workstation X68000   "
50 ac=asc("¥")
60 strset(as,ac)
70 print as
80 end
```

STRSPN

標準関数

書　式　strspn(st1, st2)

引　数　str

戻り値　int

機　能　文字列st2に属さない文字を文字列st1の中で最初に見つけた位置を返します。位置は文字列st1の先頭が0になります。見つからなかった場合は、文字列st1の長さを返します。また、文字列st1あるいはst2がnull文字の場合は0を返します。

用　例

```
10 /* strspn sample
20 str as,bs
30 as="personal workstation X68000   "
40 bs="personal wktin68000"
50 print strspn(as,bs)
60 end
```

STRTOK

標準関数

書　式　strtok(st1, st2)

引　数　str

戻り値　str

機　能　文字列st2に含まれる文字(どれか1文字)を文字列st1の中から探し、見つかれば先頭からその位置までの文字列を返します。
最初のstrtok関数の実行の場合は、文字列st1の先頭に文字列st2に含まれる文字があっても無視します。続いて、文字列st1をnull文字にしてstrtok関数を実行すると次に文字列st2に含まれる文字が見つかったところまでの文字列を返します。文字列st2に含まれる文字が見つからなくなると、null文字を返します。文字列st2が、null文字の場合は、文字列st1をそのまま返します。

用　例

```
10 /* strtok sample
20 str as[50],bs,cs
30 as="super personal workstation X68000   "
40 bs="zqx 123"
50 print strtok(as,bs)
60 print strtok(cs,bs)
70 print strtok(cs,bs)
80 print strtok(cs,bs)
90 print strtok(cs,bs)
100 end
```

STRUPR

標準関数

書　式　strupr(st)

引　数　str型変数名

戻り値　str

機　能　str型変数stの英小文字をすべて英大文字に変換します。

用　例

```
10 /* strupr sample
20 str as
30 as="PERsonal workSTATION X68000   "
40 strupr(as)
50 print as
60 end
```

SWITCH～CASE～DEFAULT～ENDSWITCH

ステートメント

書　式

switch　〈式0〉
　case　〈式1〉:〈文1〉
　case　〈式2〉:〈文2〉
　・　・
　・　・
　case　〈式n〉:〈文n〉
　〔default:〈文d〉〕
endswitch

機　能

〈式0〉と同じ値をもつcaseの〈式n〉の後の〈文n〉を実行します。〈式0〉と同じ値をもつcaseの〈式n〉がなくてdefaultがある場合は、〈文d〉を実行してから、endswitch以後の命令文に進みます。defaultがなければendswitch以後の命令文に進みます。
caseの〈式n〉の中には、整数及び文字定数のみ入ります。

用　例

```
   10 /* switch case default endswitch sample
   20 int ai
   30 str as
   40 cls
   50 input "あなたの血液型を入力してください( AB型='ab' , A型='a' , B型='b' , O
型='o' )--->";as
   60 print
   70 switch as
   80      case "ab":print "あなたの血液型はＡＢ型です":break
   90      case "a":print "あなたの血液型はＡ型です":break
  100      case "b":print "あなたの血液型はＢ型です":break
  110      case "o":print "あなたの血液型はＯ型です":break
  120      default:print "？？？？？"
  130 endswitch
  140 print
  150 input "あなたの生まれた年代を入力してください( 明治=0 , 大正=1 , 昭和=2 )-
->";ai
  160 print
  170 switch ai
  180      case 0:print "あなたは明治生まれです":break
  190      case 1:print "あなたは大正生まれです":break
  200      case 2:print "あなたは昭和生まれです":break
  210      default:print "？？？？？"
  220 endswitch
  230 end
```

SYMBOL

書　式　symbol(x, y, st, h, v, mo, p, an)

引　数　int(x, y, p), str(st), char(h, v, mo, an)

戻り値　void

機　能　グラフィック画面上に文字列を表示します。

x……始点X座標
y……始点Y座標
st……文字列(文字式)
h……横方向の倍率(1～255)
v……縦方向の倍率(1～255)
mo……文字フォントの種類

	(半角)	(全角)
0	6×12	12×12
1	8×16	16×16
2	12×24	24×24

p……パレットコード(色)
an……回転角度(0～3、ただし90度単位)

x、yは、−32768から32767までの値をとることができます。クリッピングエリアについてはwindow関数で指定された範囲になります。またパレットコードpは0から65535までの値をとることができ、その最大値はグラフィック画面の実画面サイズおよび色モードによります。

用　例

```
10 /* symbol sample
20 int ai,bi,ci,di
30 char ac
40 float af
50 screen 1,3,1,1
60 vpage(1)
70 randomize(15000)
80 for ai=1 to 10
90     ac=ai
100    symbol(256,256,"X68",ac,ac,2,rnd()*65535,ac mod 4)
110 next
120 for ai=0 to 5000:next
130 wipe()
140 for ai=0 to 3
150     ac=ai
160     for bi=90 to 420
170         af=2*pi()*bi/331
180         ci=sin(af)*160
190         di=cos(af)*160
200         symbol(bi,256-ci,"X68",1,1,2,rnd()*65535,ac)
210         symbol(bi,256-di,"X68",2,2,2,rnd()*65535,ac)
220     next
230 wipe()
```

```
240 next
250 for ai=1 to 15
260     ac=ai
270     symbol(150-ai*10,150-ai*10,"X68",ac,ac,2,rnd()*65535,0)
280     for bi=0 to 400:next
290 next
300 end
```

SYSTEM

コマンド

書　式　system

省略形　sys.

機　能　親プロセスに制御を戻します。プログラム中では使用できません。プログラム中で使用する場合は、exit( )命令を使用してください。

TAN

標準関数

書　式　tan(n)

引　数　float

戻り値　float

機　能　数値n(ラジアン)の正接(タンジェント)を返します。

用　例

```
10 /* tan sample
20 int ai,bi,ci,di,ei=1
30 float af
40 screen 1,3,1,1
50 vpage(1)
60 line(0,255,511,255,65535,&HFFFF)
70 for ai=1 to 3
80     for bi=0 to 511
90         af=(2*pi()*bi)/512*ai
100        if af>=ei*pi()/2 then ei=ei+2:continue
110        ci=255-tan(af)*100
120        di=rnd()*65535
130        pset(bi,ci,di)
140    next
150    ei=1
160 next
170 end
```

TIME$

システム変数

書　式　time$

引　数　str

戻り値　str

機　能　現在の時刻を文字列として返します。代入することによって時刻を設定できます。

用　例

```
10 /* time$ sample
20 str as,bs
30 as=time$
40 print as
50 bs=as
60 time$=bs
70 print time$
80 end
```

TOASCII

標準関数

書　式　toascii(ch)

引　数　char

戻り値　int

機　能　キャラクタコードchの文字が非アスキー文字（&H80～&HFF）であればアスキー文字（&H00～&H7F）に変換して返します。
キャラクタコードchには0から255の値が入り、戻り値もその範囲のキャラクタコードを返します。非アスキー文字以外の場合はそのままのキャラクタコードを返します。

用　例

```
10 /* toascii sample
20 char ac,bc,cc
30 ac='a'                  /* &b01100001 */
40 bc='ｱ'                  /* &b11001111 */
50 cc=&B11100111
60 print chr$(toascii(ac));"=0";bin$(toascii(ac))
70 print chr$(toascii(bc));"=0";bin$(toascii(bc))
80 print chr$(toascii(cc));"=0";bin$(toascii(cc))
90 end
```

TOLOWER

標準関数

書　式　tolower(ch)

引　数　char

戻り値　int

機　能　キャラクタコードchの文字が英大文字('A'～'Z')であれば英小文字('a'～'z')に変換して返します。

キャラクタコードchには0から255の値が入り、戻り値もその範囲のキャラクタコードを返します。英大文字以外の場合はそのままのキャラクタコードを返します。

用　例

```
10 /* tolower sample
20 char ac,bc,cc
30 ac='a'
40 bc='A'
50 cc='5'
60 print chr$(tolower(ac))
70 print chr$(tolower(bc))
80 print chr$(tolower(cc))
90 end
```

TOUPPER

標準関数

書　式　toupper(ch)

引　数　char

戻り値　int

機　能　キャラクタコードchの文字が英小文字('a'～'z')であれば英大文字('A'～'Z')に変換して返します。

キャラクタコードchには0から255の値が入り、戻り値もその範囲のキャラクタコードを返します。英小文字以外の場合はそのままのキャラクタコードを返します。

用　例

```
10 /* toupper sample
20 char ac,bc,cc
30 ac='a'
40 bc='A'
50 cc='5'
60 print chr$(toupper(ac))
70 print chr$(toupper(bc))
80 print chr$(toupper(cc))
90 end
```

VAL

標準関数

書　式　val(st)

引　数　str

戻り値　float

機　能　文字列stを浮動小数値に変換して返します。
&B、&H、&O形式の整数値を表現した文字列も変換できます。

➡　str$

用　例

```
 10 /* val sample
 20 float af,bf,cf
 30 str as,bs,cs[40]
 40 as="3000000000000"
 50 bs="-2.51235e-2"
 60 cs="0.5102123542131543512312313453543543513213"
 70 af=val(as)
 80 bf=val(bs)
 90 cf=val(cs)
100 print "("+as+")*("+bs+")*("+cs+")=";af*bf*cf
110 print "("+as+")/("+bs+")/("+cs+")=";af/bf/cf
120 print "("+as+")+("+bs+")+("+cs+")=";af+bf+cf
130 print "("+as+")-("+bs+")-("+cs+")=";af-bf-cf
140 end
```

VPAGE

GRAPH

書　式　vpage(i)

引　数　char

戻り値　void

機　能　各グラフィックページの表示ON/OFFを指定します。グラフィックページ番号は0～3で最大値はグラフィック画面の実画面サイズ及び色モードによります。

グラフィック画面の実画面サイズ及び色モードによるiの値の範囲

1……表示ON

0……表示OFF

1024×1024(16色モード)……0、1(グラフィックページ0のみ)

512×512(16色モード)……0から15

i	グラフィック ページ3	グラフィック ページ2	グラフィック ページ1	グラフィック ページ0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

512×512(256色モード)……0から3

i	グラフィック ページ1	グラフィック ページ0
0	0	0
1	0	1
2	1	0
3	1	1

512×512(65536色モード)……0、1(グラフィックページ0のみ)

➡ apage、home、screen、window

用 例

```
 10 /* vpage sample
 20 int ai,bi,ci
 30 float af
 40 screen 1,1,1,1
 50 vpage(0)
 60 window(0,0,511,511)
 70 apage(0)
 80 kyu(400,0)
 90 apage(1)
100 kyu(320,10)
110 apage(2)
120 kyu(240,20)
130 apage(3)
140 kyu(160,30)
150 while -1
160 for ai=0 to 511
170     af=pi()*ai/512*2
180     bi=abs(sin(pi()/2-af)*400)
190     if bi=0 then {
200                   vpage(&B10):pause()
210                   vpage(&B100):pause()
220                   vpage(&B1000):pause()
230                   vpage(&B100):pause()
240                   vpage(&B10):pause()
250                   }
260     home(0,0,bi):home(1,0,bi)
270     home(2,0,bi):home(3,0,bi)
280     vpage(&B1)
290 next
300 endwhile
310 end
320 func kyu(ai,bi)
330 circle(256,461+bi,50,5,0,360,ai)
340 paint(256,461+bi,5)
350 endfunc
360 func pause()
370 for ci=0 to 250
380 next
390 endfunc
```

V_CUT

IMAGE

書 式 v_cut(sw)

引 数 char

戻り値 void

機 能 スーパーインポーズ時のテレビ・ビデオ画面表示を制御します。

sw ……制御スイッチ(0＝テレビ・ビデオ画面表示ON、1＝テレビ・ビデオ画面表示OFF)

なお、この関数を実行するにはカラーイメージユニットが必要です。

WHILE～ENDWHILE

ステートメント

書　式

while　〈式〉
〈ループの内容〉
endwhile

機　能

〈式〉の値が偽になるまで、指定された〈ループの内容〉を繰り返し(ループ)ます。
繰り返しの終了判定は〈ループの内容〉の実行前に行われます。〈式〉の内容によっては、一度も〈ループの内容〉を実行しない場合があります。

用　例

```
10 /* while endwhile sample
20 int ai
30 while ai<>3
40      input "1+2=";ai
50 endwhile
60 print "ok!"
70 end
```

WIDTH

コマンド

書　式

width　〈文字数〉

省略形

w.

機　能

表示文字数を変更します。〈文字数〉は96か64です。
この命令を実行するとテキスト画面、グラフィック画面やスプライト画面は消去されます。また、console　0,31,1が再設定されます。テキスト画面のパレットもデフォルトの値に再設定されます。

WINDOW

書　式　window(x1,y1,x2,y2)

引　数　int

戻り値　void

機　能　グラフィック画面の実画面上でクリッピングするエリア(クリッピングエリア)を指定します。

x1 ……実画面上でのクリッピングエリアの左上X座標
y1 ……実画面上でのクリッピングエリアの左上Y座標
x2 ……実画面上でのクリッピングエリアの右下X座標
y2 ……実画面上でのクリッピングエリアの右下Y座標

x1、y1、x2、y2はグラフィック画面の実画面サイズが512×512の場合は0から511まで、実画面サイズが1024×1024の場合は0から1023までの値をとることができます。screen命令を実行するとx1、y1、x2、y2にはその表示画面サイズの大きさで再設定されます。ただし、この場合(x1,y1)には(0,0)が再設定されます。

➡　apage、home、screen、vpage

用　例

```
 10 /* window sample
 20 int ai,bi,ci,di,ei,fi
 30 screen 1,3,1,1
 40 vpage(1)
 50 while -1
 60 input "クリッピングエリアの左上X座標(0～511)を入力してください";ai
 70 print ai
 80 input "クリッピングエリアの左上Y座標(0～511)を入力してください";bi
 90 print bi
100 input "クリッピングエリアの右下X座標(0～511)を入力してください";ci
110 print ci
120 input "クリッピングエリアの右下Y座標(0～511)を入力してください";di
130 print di
140 print "クリッピングエリアは(";ai;",";bi;",";ci;",";di;")です"
150 window(ai,bi,ci,di)
160 box(ai,bi,ci,di,65535,&HFFFF)
170 for ai=0 to 200
180     bi=rand() mod 512
190     ci=rand() mod 512
200     di=rand() mod 512
210     ei=rand() mod 512
220     fi=rnd()*65535
230     fill(bi,ci,di,ei,fi)
240 next
250 wipe()
260 cls
270 endwhile
280 end
```

WIPE

GRAPH

書　式　wipe( )

戻り値　void

機　能　グラフィック画面のアクティブページのクリッピングエリア内を消去します。

➡　apage、window

用　例

```
 10 /* wipe sample
 20 int ai,bi,ci
 30 screen 1,1,1,1
 40 vpage(0)
 50 apage(0)
 60 fill(50,50,460,260,9)
 70 kyu(200,0,9)
 80 apage(1)
 90 kyu(200,15,0)
100 apage(2)
110 kyu(150,11,0)
120 symbol(170,166,"SHARP",3,2,2,5,0)
130 apage(3)
140 kyu(100,13,0)
150 symbol(202,196,"X68000    ",1,1,2,3,0)
160 vpage(&B1111)
170 for ai=1 to 3
180     locate 13,20:print "グラフィックページ ";ai;"をワイプします!!"
190     for bi=0 to 10000
200     next
210     apage(ai)
220     wipe()
230 next
240 apage(0)
250 locate 13,20:print "グラフィックページ  0 をワイプします!!"
260 for bi=0 to 10000
270 next
280 wipe()
290 end
300 func kyu(ai,bi,ci)
310 circle(256,256,ai,bi,-10,-170,200)
320 paint(256,250,bi)
330 circle(256,256,30,ci,0,180,200)
340 paint(256,250,ci)
350 endfunc
```

資料編

■コントロールコード一覧

CTRL +	コード	機　　能	対応キー
@	00	ダミー	
A	01	インサートモードにする(トグル動作でON/OFFする)	INS
B	02	現在のワードの先頭にカーソルを戻す	
C	03	実行を停止する	BREAK
D	04	イニシャライズする	
E	05	現在のカーソル以降1行を消す	
F	06	次のワードの先頭にカーソルを進める	
G	07	カーソル位置の文字を消す	DEL
H	08	バックスペース(カーソルの前の位置の文字を消す)	BS
I	09	水平TABを行う	HTAB
J	0A	現在のカーソル以降を次の行に分ける	
K	0B	カーソルを画面のホーム位置に移す	HOME
L	0C	テキスト画面を消去する	CLR
M	0D	キャリッジリターンをする	↵、ENTER
N	0E	現在のカーソルから上を上方向にスクロールする	ROLL UP
O	0F	現在のカーソルから下を下方向にスクロールする	ROLL DOWN
P	10	ダミー	
Q	11	実行の一時停止を解除する	
R	12	ダミー	
S	13	実行を一時停止する	
T	14	ダミー	
U	15	ダミー	
V	16	ダミー	
W	17	現在のカーソルがある行と次の行をつなぐ	
X	18	ダミー	
Y	19	ダミー	
Z	1A	現在のカーソルより下のテキスト画面をすべて消去する	
[	1B	エスケープコード	ESC
¥	1C	カーソルを右へ移動する	→
]	1D	カーソルを左へ移動する	←
^	1E	カーソルを上へ移動する	↑
_	1F	カーソルを下へ移動する	↓

■キャラクターコード表

上位4ビット→

下位4ビット↓	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	コントロールキャラクタ			0	@	P	'	p				—	タ	ミ		
1			!	1	A	Q	a	q			。	ア	チ	ム		
2			"	2	B	R	b	r			「	イ	ツ	メ		
3			#	3	C	S	c	s			」	ウ	テ	モ		
4			$	4	D	T	d	t			、	エ	ト	ヤ		
5			%	5	E	U	e	u			・	オ	ナ	ユ		
6			&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
7			'	7	G	W	g	w			ァ	キ	ヌ	ラ		
8			(	8	H	X	h	x			ィ	ク	ネ	リ		
9			)	9	I	Y	i	y			ゥ	ケ	ノ	ル		
A			*	:	J	Z	j	z			ェ	コ	ハ	レ		
B			+	;	K	[	k	{			ォ	サ	ヒ	ロ		
C			,	<	L	¥	l	|			ャ	シ	フ	ワ		
D			−	=	M	]	m	}			ュ	ス	ヘ	ン		
E			.	>	N	^	n	~			ョ	セ	ホ	゛		
F			/	?	O	_	o				ッ	ソ	マ	゜		

2バイトコード文字の1バイト目

2バイトコード文字の1バイト目

■予約語一覧

予約語はX-BASICが管理するキーワードで、変数名やfunc～endfuncで定義する関数名としては使えません。

ABS	DAY$	GCVT	KILL	M_METER	M_TRNS
APAGE	DEFAULT	GET	LEFT$	M_MOD	M_USE
ASC	DELETE	GOSUB	LEN	M_MODSNS	M_VEL
ATAN	DIM	GOTO	LFILES	M_MSTVOL	M_VGET
ATOF	DSKF	HEX$	LINE	M_MUTE	M_VOL
ATOI	ECVT	HOME	LINPUT	M_NTOFF	M_VOLFLT
AUTO	ELSE	HSV	LIST	M_NTON	M_VSET
A_PLAY	END	IF	LLIST	M_OPMEXC	M_YCOM
A_REC	ENDFUNC	IMG_COLOR	LOAD	M_OPMLFQ	NAME
BEEP	ENDSWITCH	IMG_HOME	LOCATE	M_OPMREG	NEW
BG_FILL	ENDWHILE	IMG_HT	LOG	M_OUT	NEXT
BG_GET	ERRNO	IMG_LOAD	LPRINT	M_PAN	OCT$
BG_PUT	ERROR	IMG_POS	LSEARCH	M_PANFLT	OFF
BG_SCROLL	EXIT	IMG_PUT	MID$	M_PCMBSY	ON
BG_SET	EXP	IMG_SAVE	MIRROR$	M_PCMCLR	PAINT
BG_STAT	FCLOSE	IMG_SCRN	MOUSE	M_PCMGET	PALET
BIN$	FCLOSEALL	IMG_SET	MSAREA	M_PCMLEN	PI
BOX	FCVT	IMG_STILL	MSBTN	M_PCMON	POINT
BREAK	FDELETE	INKEY$	MSPOS	M_PCMREC	POS
CASE	FEOF	INPUT	MSSTAT	M_PCMSET	POW
CHAR	FGET	INSTR	M_ALLOC	M_PGMFLT	PRINT
CHDIR	FILES	INT	M_ANTOFF	M_PLAY	PSET
CHDRV	FILL	ISALNUM	M_ASSIGN	M_PNMGET	PUT
CHR$	FIX	ISALPHA	M_BEND	M_PNMSET	PURANDT
CIRCLE	FLOAT	ISASCII	M_CHAN	M_PROG	RANDOMIZE
CLEAR	FOPEN	ISCNTRL	M_CONT	M_SNDSET	RENUM
CLS	FOR	ISDIGIT	M_CTRLRES	M_SOLO	REPEAT
COLOR	FPUTC	ISGRAPH	M_DIROUT	M_STAT	RETURN
CONSOLE	FREAD	ISLOWER	M_END	M_START	RGB
CONT	FREADS	ISPRINT	M_ERRGET	M_STOP	RIGHT$
COUTINUE	FREE	ISPUNCT	M_FREE	M_SYNC	RND
CONTRAST	FRENAME	ISSPACE	M_FREEA	M_SYSCH	RUN
COS	FSEEK	ISUPPER	M_IFCHK	M_TEMPO	SAVE
CRT	FUNC	ISXDIGIT	M_INIT	M_TNMGET	SCREEN
CSRLIN	FWRITE	ITOA	M_MDREG	M_TNMSET	SEARCH
DATE$	FWRITES	KEY	M_MEAS	M_TRK	SETMSPOS

SGN
SIN
SPACE$
SP_CLR
SP_COLOR
SP_DEF
SP_DISP
SP_INIT
SP_MOVE
SP_OFF
SP_ON
SP_PAT
SP_SET
SP_STAT
SQR
SRAND
STICK
STOP
STR
STR$
STRCHR
STRCSPN
STRIG
STRING$
STRLEN
STRLWR
STRNSET
STRRCHR
STRREV
STRSET
STRSPN
STRTOK
STRUPR
SWITCH
SYMBOL
STSTEM
TAN
THEN
TIME$
TOASCII
TOLOWER
TOUPPER
UNTIL
USING
VAL
VPAGE
V_CUT
WHILE
WIDTH
WINDOW
WIPE

■ノート番号：音程　対応表

ADPCMで取り込んだ音のデータファイルを音程に割り付け、曲の途中でその音程を演奏することで取り込んだ音を演奏中に再生することができます。

取り込んだ音データの音程への割り付けはm_pcmsetコマンドで行います。

m_pcmsetコマンドで指定するノート番号と音程の対応は次の通りです。

	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
O0>	00	01	02	03	04	05	06	07	08	09	10	11
O0	12	13	14	15	16	17	18	19	20	21	22	23
O1	24	25	26	27	28	29	30	31	32	33	34	35
O2	36	37	38	39	40	41	42	43	44	45	46	47
O3	48	49	50	51	52	53	54	55	56	57	58	59
O4	60	61	62	63	64	65	66	67	68	69	70	71
O5	72	73	74	75	76	77	78	79	80	81	82	83
O6	84	85	86	87	88	89	90	91	92	93	94	95
O7	96	97	98	99	100	101	102	103	104	105	106	107
O8	108	109	110	111	112	113	114	115	116	117	118	119
O8<	120	121	122	123	124	125	126	127				

用　例

```
100 /*pcm sample
110 int ai
120 dim char ac(20000)
130 m_pcmclr()
140 ai=fopen("B:¥sample¥sample.PCM","r")
150 fread(ac,20000,ai)                    /*PCMファイルの読み込み
160 fcloseall()
170 m_pcmset(12,4,20000,ac)
180 m_alloc(1,2000)                       /*トラックバッファの確保
190 m_assign(9,1)                         /*ﾄﾗｯｸﾊﾞｯﾌｧへﾁｬﾝﾈﾙを割当
200 m_trk(1,"q7 l16 v15 t100 @10")
210 m_trk(1,"o4c8d8e8f8g8a8b8<c8")
220 m_trk(1,"o0c8")                       /*12番(o0c)登録音を再生
230 m_trk(1,"o5c8>b8a8g8f8e8d8c8")
240 m_play()
250 end
```

■エラーコード一覧

エラー番号	内　　容
1	テンポの指定が無効です
2	トラック番号が無効です
3	OPMDRV3.X が登録されていません
4	サイズの指定が無効です
5	メモリーの確保ができません
6	チャンネル番号が無効です
7	配列の指定に誤りがあります
8	OPMDRV3.Xは現在停止中です
9	ベロシティの値がありません
10	ベロシティの値が無効です
11	録音モードの指定が無効です
12	録音の準備ができていません
13	パンポットの値がありません
14	パンポットの値が無効です
15	トランスポーズの値がありません
16	トランスポーズの値が無効です
17	パラメータを両方省略する事はできません
18	パラメータの指定に誤りがあります
19	MMLの文法に誤りがあります
20	[ ]の中の指定に誤りがあります
21	]がありません
22	繰り返しの値が無効です
23	繰り返し番号の指定がありません
24	繰り返し番号の指定が無効です
25	オクターブの番号がありません
26	オクターブの指定が無効です
27	長さの値が無効です
28	トラック容量が足りません
29	音色番号の指定がありません
30	@Wの値がありません
31	@Wの値が無効です
32	テンポの値がありません
33	テンポの値が無効です
34	長さの値がありません
35	音量の値がありません

エラー番号	内　　容
36	音量の値が無効です
37	キーコードの値がありません
38	キーコードの値が無効です
39	音色番号が無効です
40	} がありません
41	{ } の中に音符が多すぎます
42	{ } の中に音符がありません
43	Qの指定に誤りがあります
44	{ } の中の文法に誤りがあります
45	Yの指定に誤りがあります
46	@Lの指定に誤りがあります
47	Pの指定に誤りがあります
48	タイの指定に誤りがあります
49	和音の指定に誤りがあります
50	和音の数が多すぎるか、または後の ' がありません
51	@Vの指定に誤りがあります
52	@Pの指定に誤りがあります
53	@Uの指定に誤りがあります
54	@Nの指定に誤りがあります
55	@Mの指定に誤りがあります
56	@Dの指定に誤りがあります
57	@Bの指定に誤りがあります
58	@Tの指定に誤りがあります
59	@Yの指定に誤りがあります
60	PCMのバッファ不足で、これ以上登録できません
61	PCMは使用できません
62	OPMDRV3.Xのバージョンが違います
63	無効なサンプリング周波数を指定しました
64	無効な出力モードを指定しました
65	無効なデータサイズを指定しました

索　引

A～Z

ADPCM …… 49
autorun. bas ……28
BASIC2 ……4
BREAK キー……9
DEL キー ……6
FM音源制御……46
IEEE フォーマット ……33
INS キー ……6
MIDI …… 3
MML ……46
null 文字(ヌルストリング) ……14

あ

アスキー形式 ……25
オーバーフロー(桁あふれ) ……63
オープンモード ……51

か

カーソル ……4,5
カーソル移動キー ……6
外部関数 …… 28, 34
拡張子 ……25
画面範囲(座標系) ……36
カラーコード ……44
関係演算 …… 64
関数 …… 11, 16
キーワード ……11
行 ……12
行番号 ……5,7,12
グラフィック画面……35
グラフィックページ数……36
クリエイトモード……51
クリッピングエリア……37
クローズ ……51
コマンド …… 11, 66
コマンドモード ……27
コンフィギュレーションファイル……29,30

さ

サブルーチン ……13
サンプリング周波数……49
システム変数 ……11
実画面 ……36
実数型 …… 14
シフト演算 ……63
条件判断 ……17
初期値(デフォルト値) ……14
書式 …… 68
処理の分岐 ……15
スイッチ ……28
水平画面サイズ ……28
スクリーンモード ……35,36
スクロール ……35
ステートメント ……11,66
スプライト ……38
スプライト画面 ……38
スペース……7
セーブ(SAVE) ……24
制御構造 ……15
整数型 ……14

た

ダイレクトモード ……5
チャイルド(子)プロセス……28
データ ……12
ディレクトリ ……28
テキスト画面 ……35
テキストファイル形式……30
デバイスドライバ……33
デバッグ ……14
特殊記号 ……57

トラックバッファ……28，46

な

入力待ち状態 ……4

は

倍精度実数値 …… 14
配列の宣言 ……21
配列変数 ……20
バックグラウンド……38
パレットコード ……44
パレットブロック……39
ビープ音 …… 30
引数……29，68
ビット操作 ……65
標準関数 ……34
表示画面 ……36
ファンクションキー表示 ……35
浮動小数点数 ……33
フリーエリアサイズ……28
プレーン ……38
プログラムモード …… 5
プロンプト ……27
文 ……12
変数 …… 14，15
変数名 …… 14，15
変数の宣言 ……14
変数のデータ型 ……14

ま

マージ(混合、追加)……25
マルチステートメント …… 12
命令文……5，9
メインプログラム ……14，23
文字型 ……14
文字バッファのサイズ ……14，58
モジュール化 ……23
戻り値 ……68

や

予約語 ……11

ら

ライトモード ……51
リードモード ……51
リードライトモード……51
リターンキー ……4，5
ループ ……17
ローカル変数 ……31
ロード(LOAD) ……25
論理演算 ……64
論理画面 ……36

■機能別索引

ADPCMの制御（AUDIO．FNC）

A _PLAY ……75
A _REC ……76

コマンド

AUTO ……74
CHDIR……86
CHDRV ……86
CHILD……87
CLEAR ……89
CONT ……93
DELETE …… 98
FILES ……108
KEY LIST……147
KILL ……147
LIST ……151
LOAD ……152
NAME ……196
NEW ……196
RENUM ……208
RUN ……211
SAVE……211
SEARCH ……213
SYSTEM ……238
WIDTH……244

データの変換

ASC ……71
ATOF ……72
ATOI ……73
BIN$ ……83
CHR$ ……87
ECVT ……101
FCVT ……105
FIX……110
GCVT ……122
HEX$ ……125
INT ……139
ITOA……146
OCT$ ……197
STR$ ……228
TOASCII ……239
TOLOWER ……240
TOUPPER ……240
VAL ……241

ファイル入出力

DSKF……100
FCLOSE ……104
FCLOSEALL ……105
FDELETE ……106
FEOF……106
FGETC ……107
FOPEN ……111
FPUTC ……113
FREAD ……114
FREADS ……116
FRENAME ……117
FSEEK ……118
FWRITE ……120
FWRITES ……121

グラフィックの制御（GRAPH．FNC）

APAGE ……70
BOX ……84
CIRCLE ……88
CONTRAST ……95
FILL ……109
GET ……123
HOME ……126
HSV ……127
LINE ……149
PAINT ……198
PALET ……199
POINT ……201
PSET ……205
PUT ……206
RGB ……209
SYMBOL ……237
VPAGE ……242
WINDOW ……245
WIPE ……246

数値演算

ABS ……69
ATAN ……71
COS ……95
EXP ……104
LOG ……153
PI ……200
POW ……202
RAND ……207
RANDOMIZE ……207
RND ……210
SGN ……214
SIN ……215
SQR ……225
SRAND ……225
TAN ……238

マウスの制御（MOUSE．FNC）

MOUSE ……155
MSAREA ……156
MSBTN ……157
MSPOS ……158
MSSTAT ……159
SETMSPOS ……214

FM 音源の制御（MUSIC．FNC）

M_ALLOC …… 160
M_ANTOFF …… 160
M_ASSIGN …… 161
M_BEND …… 161
M_CHAN …… 162
M_CONT …… 162
M_CTRLRES …… 163
M_DIROUT …… 163
M_END …… 164
M_ERRGET …… 164
M_FREE …… 165
M_FREEA …… 165

M_IFCHK……165
M_INIT……166
M_MDREG……166
M_MEAS……167
M_METER……167
M_MOD……168
M_MODSNS……168
M_MSTVOL……169
M_MUTE……169
M_NTOFF……170
M_NTON……170
M_OPMEXC……171
M_OPMLFQ……171
M_OPMREG……172
M_OUT……172
M_PAN……173
M_PANFLT……174
M_PCMBSY……174
M_PCMCLR……175
M_PCMGET……175
M_PCMLEN……176
M_PCMON……177
M_PCMREC……178
M_PCMSET……179
M_PGMFLT……180
M_PLAY……180
M_PNMGET……181
M_PNMSET……182
M_PROG……182
M_SNDSET……183
M_SOLO……183
M_START……184
M_STAT……185
M_STOP……186
M_SYNC……187
M_SYSCH……187
M_TEMPO……188
M_TNMGET……189
M_TNMSET……189
M_TRK……190
M_TRNS……190
M_USE……191
M_VEL……191
M_VGET……192
M_VOL……193
M_VOLFLT……193
M_VSET……194
M_YCOM……195

スプライトの制御（SPRITE. FNC）

BG _FILL……78
BG _GET……79
BG _PUT……80
BG _SCROLL……81
BG _SET……81
BG _STAT……82
SP _CLR……216
SP _COLOR……216
SP _DEF……217
SP _DISP……219
SP _INIT……220
SP _MOVE……220
SP _OFF……221
SP _ON……221
SP _PAT……222
SP _SET……223
SP _STAT……224

文字列処理

INSTR ……138
ISALNUM ……139
ISALPHA……140
ISASCII……140
ISCNTRL……141
ISDIGIT ……141
ISGRAPH……142
ISLOWER……142
ISPRINT ……143
ISPUNCT……143
ISSPACE ……144
ISUPPER ……144
ISXDIGIT……145
LEFT$ ……148
MID$ ……154
MIRROR$ ……154
RIGHT$ ……210
SPACE$ ……215
STRCHR ……228
STRCSPN ……229
STRING$ ……231
STRLEN (LEN) ……231
STRLWR ……232
STRNSET ……232
STRRCHR ……233
STRREV ……233
STRSET ……234
STRSPN ……234
STRTOK ……235
STRUPR ……235

ジョイスティック入力（STICK．FNC）

STICK ……226
STRIG ……230

ステートメント

BEEP ……77
BREAK ……85
CHAR ……85
CLS ……89
COLOR ……90
COLOR [ ] ……91
CONSOLE ……92
CONTINUE ……94
DIM ……99
END ……102
ERROR ON／OFF……103
EXIT( )……103
FLOAT……110
FOR～NEXT ……112
FUNC～ENDFUNC～RETURN( ) …119
GOSUB～RETURN ……124
GOTO ……125
IF～THEN～ELSE……128
INPUT ……137
INT ……138
KEY ……146
LFILES……108
LINPUT ……150
LLIST……151
LOCATE ……152
LPRINT ……203
LSEARCH……213
PRINT ……203

REM ……208
REPEAT～UNTIL……209
SCREEN ……212
STOP……227
STR ……227
SWITCH ~ CASE ~ DFFAULT -ENDSWITCH …236
WHILE～ENDWHILE ……244

システム変数

CSRLIN …… 97
DATE$ …… 97
DAY$ …… 98
ERRNO……102
FREE……117
INKEY$ ……136
POS ……202
TIME$ ……239

イメージ処理（IMAGE．FNC）

CRT …… 96
IMG _COLOR……129
IMG _HOME ……130
IMG _HT……131
IMG _LOAD ……131
IMG _POS ……132
IMG _PUT ……133
IMG _SAVE ……133
IMG _SCRN ……134
IMG _SET ……135
IMG _STILL ……136
V _CUT ……243

シャープ株式会社

本　　社　〒545 大阪市阿倍野区長池町22番22号　電話(06) 621-1221(大代表)
電子機器事業本部　〒329-21 栃木県矢板市早川町174番地　電話(0287)43-1131(大代表)

TMAN-3741CEZZ
T0593-A ②