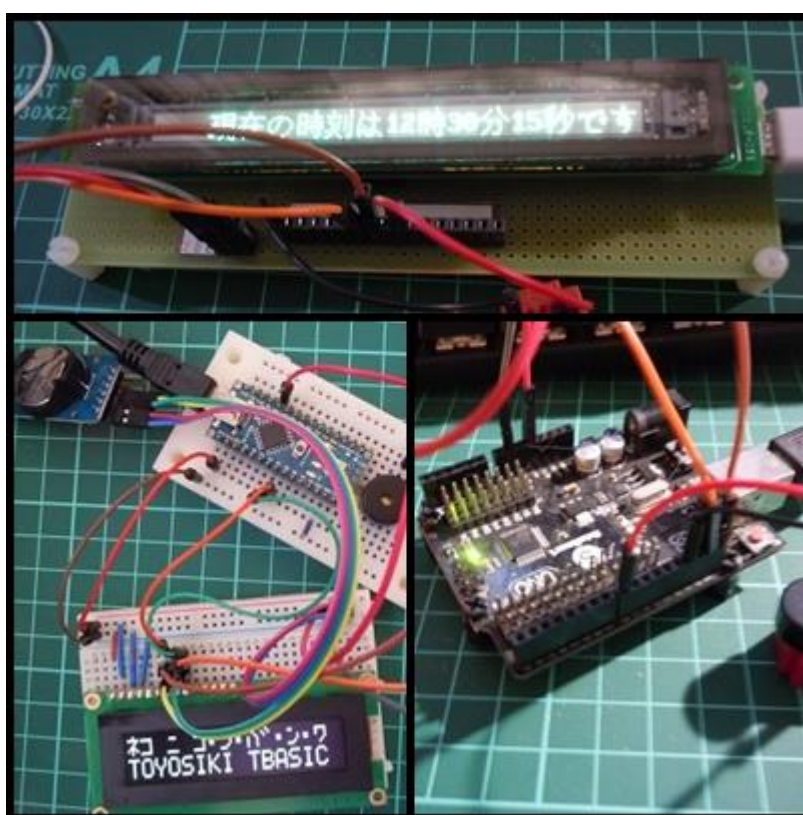


豊四季タイニーBASIC for Arduino 機能拡張版 V0.07 (+ MW25616L 実験用表示モジュール対応)

リファレンスマニュアル RV01



作成日 2019年9月25日

作成者 たま吉さん

目次

目次	i
1. 仕様や構成など	1
1.1 はじめに	1
1.2 関連情報	1
1.3 著作権・利用条件について	2
1.4 システム構成	3
1.5 仕様	4
1.6 ボードピン構成	5
1.7 メモリーマップ	7
1.8 デジタルI/O・アナログ入力	8
1.9 文字コード及び文字列	9
1.10 編集操作操作キー	10
1.11 キー入力コード	11
1.12 圧電スピーカー 単音出力対応	14
1.13 I2C 接続 EEPROM へのプログラム保存対応	15
1.14 I2C 接続 RTC DS3231 対応	17
1.15 美咲フォント対応	18
1.16 赤外線リモコン受信対応	19
1.17 NeoPixel(WS2812B) 対応	20
1.18 VFD ディスプレイ (MW25616L) 対応	21
1.19 16 ドット日本語フォント (GT20L16J1Y) 対応	21
1.20 有機 EL キャラクタディスプレイ SO1602AWWB 対応	22
2. 利用環境設定	23
2.1 ファームウェアの書き込み	23
2.2 ターミナルソフトの通信条件の設定	24
2.3 コンソール画面の表示設定	26
2.4 プログラムの保存と読み込み	27
2.5 パソコンからのプログラム読込	28
2.6 プログラムをパソコンに保存する	29
2.7 プログラムの自動起動方法	29
3. 使ってみよう！	30
3.1 入力可能状態は OK■ (カーソル)	30
3.2 Hello.world を表示してみる	30
3.3 Hello.world を Hello, Tiny BASIC に変更してみる	30
3.4 Print の前に 10 を追加してプログラムにしよう	31

3.5	プログラムを実行してみよう.....	31
3.6	プログラムに追加する.....	31
3.7	行番号を並びなおす	32
3.8	やっぱり 10 行は不要、削除したい.....	32
3.9	プログラムを保存したい.....	33
3.10	保存されたプログラムを読み込み.....	33
4.	プログラム構成要素の説明（コマンド・関数等）	34
4.1	プログラムの構造と形式.....	34
4.2	行と命令文.....	35
4.3	制御文.....	36
4.4	コマンド・関数.....	37
4.5	コマンド・関数一覧.....	38
4.6	数値.....	42
4.7	文字・文字列.....	42
4.8	文字列利用における制約.....	42
4.9	変数・配列変数.....	44
4.10	演算子.....	44
4.11	式.....	46
4.12	定数.....	47
5.	各制御文の詳細.....	50
5.1	GoTo 指定行へのジャンプ（制御命令）	50
5.2	GoSub サブルーチンの呼び出し（制御命令）	51
5.3	Return GoSub 呼び出し元への復帰（制御命令）	52
5.4	If 条件判定（制御命令）	53
5.5	For To ~ NEXT 繰り返し実行（制御命令）	54
5.6	End プログラムの終了（制御命令）	55
5.7	On Timer タイマーイベント（制御命令）	56
5.1	Timer タイマーイベント設定（制御命令）	57
5.2	On Pin 外部割込みイベント（制御命令）	58
5.3	Pin 外部割込みイベント設定（制御命令）	61
5.4	Sleep 休止（制御命令）	62
6.	各コマンド・関数の詳細.....	63
6.1	Run プログラムの実行.....	63
6.2	List プログラムリストの表示.....	64
6.3	Renum 行番号の振り直し.....	65
6.4	Delete プログラムの指定行の削除.....	66
6.5	New プログラムの消去.....	67
6.6	Load プログラムの読み込み.....	68

6.7	Save プログラムの保存	69
6.8	Erase 保存プログラムのプログラム削除.....	70
6.9	Files 保存プログラムの一覧表示	71
6.10	Format 外部接続 I2C EEPROM の領域初期化	73
6.11	Drive 外部接続 I2C EEPROM の指定	74
6.12	Rem コメント	75
6.13	Let 変数に値を代入.....	76
6.14	Print 画面への文字表示.....	77
6.15	Input 数値の入力.....	79
6.16	Cls 画面表示内容の全消去	80
6.17	Color 文字色の設定.....	81
6.18	Attr 文字表示属性の設定.....	82
6.19	Locate カーソルの移動	83
6.20	Free プログラム領域の残りバイト数の取得（数値関数）	84
6.21	Inkey キー入力の読み取り（数値関数）	85
6.22	Rnd 乱数の発生（数値関数）	86
6.23	Abs 絶対値の取得（数値関数）	87
6.24	Asc 文字から文字コードへの変換（数値関数）	88
6.25	Byte 文字列のバイト数の取得（数値関数）	90
6.26	Len 文字列の長さの取得（数値関数）	91
6.27	Map 数値のスケール変換（数値関数）	92
6.28	Grade 等級判定（数値関数）	93
6.29	Chr\$ 文字コードから文字への変換（文字列関数）	95
6.30	Bin\$ 数値から 2 進数文字列への変換（文字列関数）	96
6.31	Hex\$ 数値から 16 進数文字列への変換（文字列関数）	97
6.32	Dmp\$ 整数の小数付き数値文字列への変換（文字列関数）	98
6.33	Str\$ 変数が参照する全角を含む文字列の取得・文字列の切り出し.....	99
6.34	Wait 時間待ち	100
6.35	Tick 経過時間取得（数値関数）	101
6.36	Poke 指定アドレスへのデータ書き込み	102
6.37	Peek 指定アドレスの値参照（数値関数）	103
6.38	Vcls VFD 表示内容の全消去.....	104
6.39	Vdisplay VFD 表示のオン・オフ	105
6.40	Vbright VFD 輝度設定	106
6.41	Vmsg VFD にメッセージ文の表示.....	107
6.42	Vscroll VFD に指定長空白の表示	108
6.43	Vput VFD に直接データ送信	109
6.44	Tone 単音出力.....	110

6.45	NoTone 単音出力停止	111
6.46	Play 音楽演奏(MML 文)	112
6.47	Tempo 音楽演奏のテンポの設定	114
6.48	GetFont 美咲フォントデータ取得 (数値関数)	115
6.49	Gpio 汎用入出力ピンの設定	117
6.50	Out デジタル出力	118
6.51	PWM PWM パルス出力	119
6.52	In デジタル入力 (数値関数)	120
6.53	ShiftOut デジタルシフトアウト出力	121
6.54	ANA アナログ入力 (数値関数)	122
6.55	SHIFTIN デジタルシフト入力 (数値関数)	123
6.56	PULSEIN 入力パルス幅の計測力 (数値関数)	124
6.57	I2CR I2C スレーブデバイスからのデータ受信 (数値関数)	126
6.58	I2CW I2C スレーブデバイスへのデータ送信 (数値関数)	128
6.59	DATE 現在時刻の表示	130
6.60	GETDATE 日付の取得	131
6.61	GETTIME 時刻の取得	132
6.62	SETDATE 時刻の設定	133
6.63	DATE\$ 現在時刻文字列の取得 (文字列関数)	134
6.64	CPRINT 有機 EL キャラクタディスプレイ文字列表示	135
6.65	CCLS 有機 EL キャラクタディスプレイ 表示内容消去	136
6.66	CCURS 有機 EL キャラクタディスプレイ カーソル表示・非表示設定	137
6.67	CLOCATE 有機 EL キャラクタディスプレイ カーソルの移動	138
6.68	CCONS 有機 EL キャラクタディスプレイ コントラスト設定	139
6.69	CDISP 有機 EL キャラクタディスプレイ 表示・非表示設定	140
6.70	IR 赤外線リモコン受信 (数値関数)	141
6.71	NINIT NeoPixel 初期化	142
6.72	NBRIGHT NeoPixel 輝度設定	143
6.73	NCLS NeoPixel 表示クリア	144
6.74	NSET NeoPixel 指定ピクセルの色設定	145
6.75	NPSET NeoPixel 指定座標色設定	146
6.76	NMSG NeoPixel メッセージ文表示	147
6.77	NSCROLL NeoPixel スクロール	148
6.78	NLINE NeoPixel 直線描画	149
6.79	NUPDATE NeoPixel バッファ表示反映	150
6.80	NSHIFT NeoPixel ピクセルのシフト	151
6.81	NPOINT NeoPixel 指定座標の色コード取得 (数値関数)	152
6.82	RGB NeoPixel 256 色コードの作成 (数値関数)	153

1. 仕様や構成など

1.1 はじめに

「豊四季タイニーBASIC for Arduino 機能拡張版(+ MW25616L 実験用表示モジュール対応)」は整数型 BASIC インタプリタ実行環境です。Tetsuya Suzuki 氏が開発・公開している「TOYOSHIKI Tiny BASIC for Arduino」(以降オリジナル版と呼称)をベースに機能拡張を行い、MW25616L 実験用表示モジュール対応を行いました。本バージョンのプログラムソースはオリジナル版と同様に公開しており、自由に改造等を行うことができます。

オリジナル版からの機能拡張としては、MW25616L 表示制御機能の他に、NeoPixel 制御機能、プログラム編集機能の強化(プログラム保存機能、電子工作のための制御機能(デジタル入出力、PWM 出力、アナログ入力、I2C バス通信、シリアル通信))を追加しました。

本リファレンスマニュアルの構成は、前半に概要、システム構成、仕様、レガシーBASIC 環境を始めて利用する方向けの解説、中盤にリファレンス、後半に応用の内容となります。オリジナル版のタイニーBASIC の文法を含む、新しく追加した機能、コマンドについて解説・説明します。

1.2 関連情報

「豊四季タイニーBASIC (オリジナル版)」に関する情報

□ 開発者公開ホームページ

電腦伝説 Vintagechips 豊四季タイニーBASIC 確定版

<https://vintagechips.wordpress.com/2015/12/06/豊四季タイニーbasic 確定版/>

□ 開発者公開リソース

https://github.com/vintagechips/ttbasic_arduino/

□ 開発者執筆書籍

タイニーBASIC を C で書く

<http://www.socym.co.jp/book/1020>



原本のご紹介

タイニー BASIC を C で書く
パソコンでもマイコンでも走り、機能拡張し放題。

著者一鈴木哲哉

定価—3,200円+税

判型/ページ数—B5 変形/288 ページ

ISBN978-4-8026-1020-9

「豊四季タイニーBASIC for Arduino 機能拡張版」に関する情報

□ 公開サイト

豊四季タイニーBASIC for MW25616L 実験用表示モジュール (Tamakichi/ttbasic_mw25616l)

https://github.com/Tamakichi/ttbasic_mw25616l

□ MW25616L 実験用表示モジュールに関する情報

・「VFD ふぁん」さんの HP

<http://vfd-fun.blogspot.jp/>

1.3 著作権・利用条件について

「豊四季タイニーBASIC for Arduino 機能拡張版」の著作権は移植・機能拡張者の「たま吉」および、オリジナル版開発者の「Tetsuya Suzuki」氏にあります。本著作物の利用条件は、オリジナル版「豊四季タイニーBASIC」に従うものとします。また、プログラムの一部に「VFD ふぁん」さんが公開しているプログラムソースを利用しています。

オリジナル版は GPL (General Public License) でライセンスされた著作物であり、「豊四季タイニーBASIC for Arduino 機能拡張版」もそれに GPL ライセンスされた著作物とします。本著作物もこれに従うものとします。

また、下記の利用条件も守って下さい。

- 1) 2 次再配布においてはバイナリーファイルのみの配布を禁止、必ずプロジェクト一式で配布すること。
- 2) 上記配布において「豊四季タイニーBASIC for Arduino 機能拡張版」の明記、配布元の URL を明記する。
- 3) 営利目的の NAVER 等のまとめサイト(まとめ者に利益分配のあるもの)へのいかなる引用の禁止

補足事項

オリジナル版において開発者 Tetsuya Suzuki 氏は次の利用条件を提示しています。

- 1) TOYOSHIMI Tiny BASIC for Arduino

https://github.com/vintagechips/ttbasic_arduino

の記載内容/

(C)2012 Tetsuya Suzuki

GNU General Public License

- 2) 豊四季タイニーBASIC のリファレンスを公開

<https://vintagechips.wordpress.com/2012/06/14/豊四季タイニーbasic リファレンス公開/>

の記載内容/

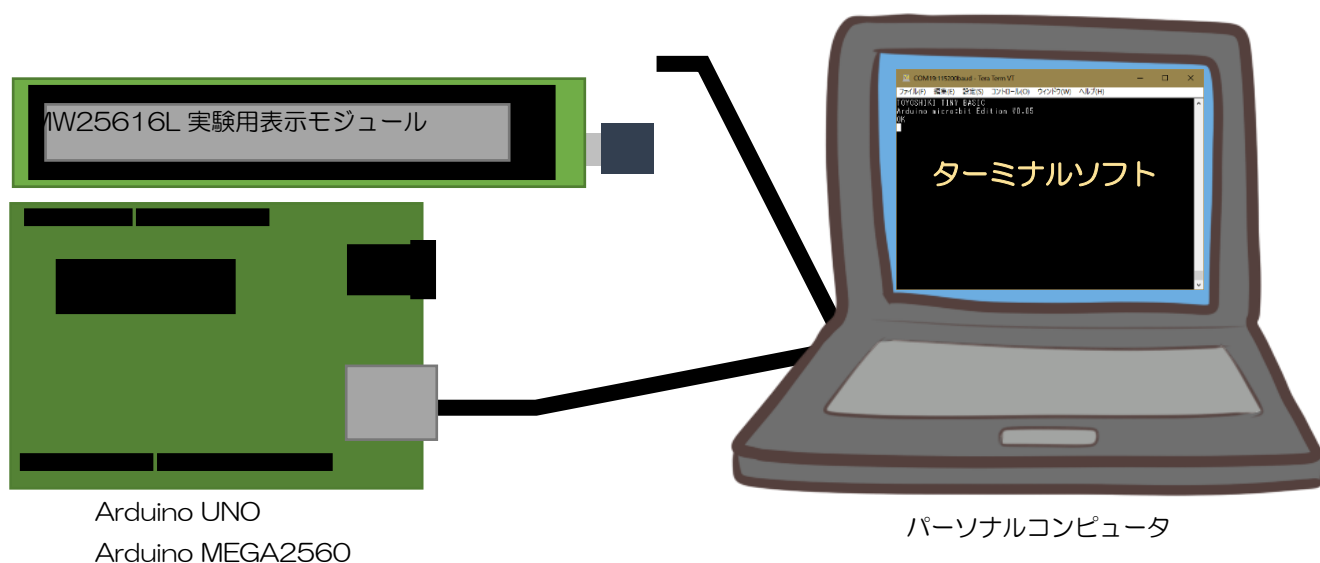
豊四季タイニーBASIC がもたらすいかなる結果にも責任を負いません。

著作権者の同意なしに経済的な利益を得てはいけません。

この条件のもとで、利用、複写、改編、再配布を認めます。

1.4 システム構成

「豊四季Tiny BASIC for Arduino 機能拡張版」は、次の構成で利用可能です。



必要なもの

- パーソナルコンピュータ：Windows 10 パソコン ※USB ポートが利用出来ること
- Arduino UNO（互換機含む）または MW25616L 実験用表示モジュール本体
Arduino MEGA2560 またはその互換機
- Micro USB ケーブル
- シリアルコンソール対応ターミナルソフト：TeraTerm 等（V4.89 以降）

豊四季 Tiny BASIC インタプリター言語を利用したプログラム作成は、パソコン上のターミナルソフト上にて、ラインエディタとコマンドを使って対話形式で行います。シフト JIS を使った全角日本語文字の利用が可能です。

```
COM5 - Tera Term VT
ファイル(E) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
>list
10 For I=1 To 5
20 For J=1 To I
30 Print " ";
40 Next
50 Print "こんにちは、世界！"
60 Next
OK
>run
こんにちは、世界！
こんにちは、世界！
こんにちは、世界！
こんにちは、世界！
こんにちは、世界！
OK
>
```

Windows 以外の OS 環境でも同様の環境が構築できれば、利用出来ると思います。

1.5 仕様

(1) ソフトウェア仕様

項目		概要
プログラミング環境		ターミナル上での CUI (キャラクユーザーインタフェース) USB ケーブル経由シリアル接続のラインエディタ+実行環境
ターミナルスクリーン画面		80×24 文字
B A S I C 言 語 仕 様	形式	整数型 BASIC (符号付 16 ビット整数 -32768~32767)
	プログラム領域	1024 バイト (MEGA2560 の場合 2048 バイト)
	利用文字コード	1 バイト ASCII コード (半角記号・英数字、半角カタカナ) 2 バイト文字 シフト JIS 表示可能文字はターミナル環境のフォント設定に依存
	利用可能変数	変数名: 26 個 A~Z の英字 1 文字 26 個, 配列変数: 32 個 @(0)~@(31) (MEGA2560 の場合 100 個 @(0)~@(99))
	ユーザー開放 作業用メモリ	SRAM 64 バイト×2 エリア ・先頭アドレス定数 MEM ~ MEM + 63 ・先頭アドレス定数 MEM2 ~ MEM2 + 63 ※コマンド実行のワークエリアと共用のため内容の保持は出来ません。
	プログラム保存方式	・内部 EEPROM メモリ 1 本 (ページ 0 指定) (MEGA2560 の場合、2 本 (ページ 0, 1 指定)) ・外部 I2C 接続 EEPROM メモリ (オプション) 4k バイト ~ 64k バイトの容量に対応、最大 63 本保存可能
	フォントデータ (オプション)	・8x8 ドット美咲フォント 500 文字 (オプション) ・ROMGT30L16J1Y 日本語 JIS 第 1、第 2 フォント (MW25616L (VFD) 表示専用)

(2) ハードウェア仕様

項目	概要
動作クロック	16MHz
VFD (オプション)	MW25616L 256x16 ドット モノラル表示、輝度調節 0~255 段階調節可能
USB I/F	USB-シリアル通信
シリアル通信	1 チャンネル 115200、データ長 8 ビット、 ストップビット 1 パリティ None
I2C インタフェース	1 チャンネル (マスター利用のみ) アドレス 7 ビット、100kbps
GPIO	・デジタル入力 1 ビット×20 ポート (MEGA2560 の場合 68 ポート) ・デジタル出力 1 ビット×20 ポート (MEGA2560 の場合 68 ポート) ・PWM 出力 6 チャンネル (MEGA2560 の場合 15 チャンネル) (周波数固定、デューティ比 1/255 刻み)
アナログ入力	8 チャンネル (分解能 10 ビット) (MEGA2560 の場合 15 チャンネル)
単音出力 (オプション)	デジタル矩形波出力による簡易単音再生

(3) サポートする周辺機器

- 圧電スピーカー : Tone、NnTone、Play コマンドによる単音出力をサポートします。
- シリアルポート : USB-シリアル変換モジュールを使ったシリアル通信をサポートします。
- I2C EEPROM : 4~64k バイト対応、プログラム保存に利用します (Save、Load 等)。
- RTC (DS3231) : Date、SetDate、GetDate、GetTime による RTC 機能の利用
- NeoPixel (WS2812B) : 最大 64 個対応、8×8 マトリックス対応
- 有機 EL キャラクタディスプレイ SO1602AWWB :
Cprint、Ccls コマンドによる文字表示をサポートします。

1.6 ボードピン構成

Arduino UNO の端子の利用割当は以下の通りです。

ピン名称 : 機能(ピン番号)

D0	: RxD 専用(0)
D1	: TxD 専用(1)
D2	: デジタル入出力(2)、外部割込み
D3	: デジタル入出力(3)、PWM(3)、外部割込み
D4	: デジタル入出力(4)
D5	: デジタル入出力(5)、PWM(5)
D6	: デジタル入出力(6)、PWM(6)
D7	: デジタル入出力(7)、保存プログラム自動起動判定用 (HIGH 自動起動、LOW 自動起動は行わない)
D8	: デジタル入出力(8)、単音出力用 (圧電スピーカー接続)
D9	: デジタル入出力(9)、PWM(9)
D10	: デジタル入出力(10)、PWM(10)
D11	: デジタル入出力(11)、PWM(11)
D12	: デジタル入出力(12)
D13	: デジタル入出力(13)
A0	: デジタル入出力(14)、アナログ入力(A0、14)
A1	: デジタル入出力(15)、アナログ入力(A1、15)
A2	: デジタル入出力(16)、アナログ入力(A2、16)
A3	: デジタル入出力(17)、アナログ入力(A3、17)
A4	: デジタル入出力(18)、アナログ入力(A4、18)、I2C(SDA)
A5	: デジタル入出力(19)、アナログ入力(A5、19)、I2C(CLK)
A6	: デジタル入出力(20)、アナログ入力(A6、20)
A7	: デジタル入出力(21)、アナログ入力(A7、21)

MW25616L 実験用表示モジュールにて解放されている端子の利用割当は以下の通りです。

ピン名称 : 機能

Pin 0	: RxD 専用(0)
Pin 1	: TxD 専用(1)
Pin 2	: デジタル入出力(2)、外部割込み
Pin 5	: デジタル入出力(5)、PWM(5)
Pin 6	: デジタル入出力(6)、PWM(6)
Pin 7	: デジタル入出力(7)、保存プログラム自動起動判定用 (HIGH 自動起動、LOW 自動起動は行わない)
Pin 8	: デジタル入出力(8)、単音出力用 (圧電スピーカー接続)
A3	: デジタル入出力(17)、アナログ入力(A3、17)
A4	: デジタル入出力(18)、アナログ入力(A4、18)、I2C(SDA)
A5	: デジタル入出力(19)、アナログ入力(A5、19)、I2C(CLK)
A6	: デジタル入出力(20)、アナログ入力(A6、20)
A7	: デジタル入出力(21)、アナログ入力(A7、21)
REST	: RESET 入力
SCK	: 使用不可
MISO	: 使用不可
MOSI	: 使用不可

Arduino MEGA2560 の端子の利用割当は以下の通りです。

ピン名称 : 機能(ピン番号)

D0	: RxD 専用(0)
D1	: TxD 専用(1)
D2	: デジタル入出力(2)、PWM(2) 、外部割込み
D3	: デジタル入出力(3)、PWM(3) 、外部割込み
D4	: デジタル入出力(4)、PWM(4)
D5	: デジタル入出力(5)、PWM(5)
D6	: デジタル入出力(6)、PWM(6)
D7	: デジタル入出力(7)、PWM(7)
D8	: デジタル入出力(8)、PWM(8)
D9	: デジタル入出力(9)、PWM(9)
D10	: デジタル入出力(10)、PWM(10)
D11	: デジタル入出力(11)、PWM(11)
D12	: デジタル入出力(12)、PWM(12)
D13	: デジタル入出力(13)、PWM(13)
D14	: デジタル入出力(14)
D15	: デジタル入出力(15)
D16	: デジタル入出力(16)
D17	: デジタル入出力(17)
D18	: デジタル入出力(18)
D19	: デジタル入出力(19)
D20	: デジタル入出力(20)
D21	: デジタル入出力(21)、I2C(SDA)
D22	: デジタル入出力(22)、I2C(SCL)
D23	: デジタル入出力(23)
D24	: デジタル入出力(24)
D25	: デジタル入出力(25)
D26	: デジタル入出力(26)
D27	: デジタル入出力(27)
D28	: デジタル入出力(28)
D29	: デジタル入出力(29)
D30	: デジタル入出力(30)
D31	: デジタル入出力(31)
D32	: デジタル入出力(32)
D33	: デジタル入出力(33)
D34	: デジタル入出力(33)
D35	: デジタル入出力(33)
D36	: デジタル入出力(33)
D37	: デジタル入出力(33)
D38	: デジタル入出力(33)
D39	: デジタル入出力(33)
D40	: デジタル入出力(43)
D41	: デジタル入出力(41)
D42	: デジタル入出力(42)
D43	: デジタル入出力(43)
D44	: デジタル入出力(44)、PWM(44)
D45	: デジタル入出力(45)、PWM(45)
D46	: デジタル入出力(46)、PWM(46)
D47	: デジタル入出力(47)
D48	: デジタル入出力(48)
D49	: デジタル入出力(49)、単音出力用 (圧電スピーカー接続)
D50	: デジタル入出力(50)
D51	: デジタル入出力(51)
D52	: デジタル入出力(52)
D53	: デジタル入出力(53)、保存プログラム自動起動判定用 (HIGH 自動起動、LOW 自動起動は行わない)
A0	: デジタル入出力(54)、アナログ入力(A0、54)
A1	: デジタル入出力(55)、アナログ入力(A1、55)
A2	: デジタル入出力(56)、アナログ入力(A2、56)
A3	: デジタル入出力(56)、アナログ入力(A3、57)
A4	: デジタル入出力(58)、アナログ入力(A4、58)
A5	: デジタル入出力(59)、アナログ入力(A5、59)
A6	: デジタル入出力(60)、アナログ入力(A6、60)
A7	: デジタル入出力(61)、アナログ入力(A7、61)
A8	: デジタル入出力(62)、アナログ入力(A8、62)
A9	: デジタル入出力(63)、アナログ入力(A9、63)
A10	: デジタル入出力(64)、アナログ入力(A10、64)
A11	: デジタル入出力(65)、アナログ入力(A11、65)
A12	: デジタル入出力(66)、アナログ入力(A12、66)
A13	: デジタル入出力(67)、アナログ入力(A13、67)
A14	: デジタル入出力(68)、アナログ入力(A14、68)
A15	: デジタル入出力(69)、アナログ入力(A15、69)

1.7 メモリーマップ

「豊四季タイニーBASIC for Arduino 機能拡張版」ではフラッシュメモリ領域 32k バイト（MEGA2560 の場合 256k バイト）、SRAM は 2k バイト（MEGA2560 の場合 8k バイト）の領域を利用しています。ここでは、これらの領域の内訳を示します。

フラッシュメモリ

32k バイト全領域を Arduino ブートローダーと豊四季タイニーBASIC にて占有しています。

（Arduino MEGA2560 利用の場合 256k バイト）

SRAM

2k バイト全領域を豊四季タイニーBASIC にて占有しています。

（Arduino MEGA2560 利用の場合 8k バイト）

仮想アドレス

タイニーBASIC では、仮想アドレス利用することで SRAM 上のデータの参照・書き込みを行うことができます。

仮想アドレスは次の構成となります。

仮想アドレス	定数名	領域サイズ(バイト)	用途
\$1900	Var	52	変数領域 (A~Z) 2×26 = 52 バイト
\$1AA0	Array	64 (MEGA2560 は 200)	配列変数領域 (@ (0) ~ @ (31)) (MEGA2560 は @ (0) ~ @ (99))
\$1BA0	Prg	1024 (MEGA2560 は 2048)	プログラム領域
\$2BA0	Mem	64	ユーザーワーク領域
\$2CA0	Mem2	64	ユーザーワーク領域 2

各領域のアドレス先頭は定数 Var、Array、Prg、Mem、Mem2 に定義されています。

プログラムから上記の定数を利用して利用することが出来ます。

（注意）

- Mem（ユーザーワーク領域）は、ラインエディタ、および下記のコマンドのワーク領域としても利用しています。

Files、Vmsg、Nmsg、Cprint、Play、List

プログラム中に、上記のコマンドを実行すると直前の MEM 領域の内容は破壊されます。

また、ラインエディタの利用でも内容が破壊されます。

- Mem2（ユーザーワーク領域 2）は、ラインエディタのワーク領域としても利用しています。

ラインエディタの利用時に内容が破壊されます。

プログラム中でユーザーワーク領域を利用している場合、データが破壊されることに注意して利用して下さい。

1.8 デジタル I/O ・ アナログ入力

「豊四季タイニー-BASIC for Arduino 機能拡張版」では Arduino のポートを使った、デジタル I/O、アナログ入力をサポートしています。

デジタル入出力の例

ボード上の LED の制御

```
10 LED On
20 Wait 500
30 LED Off
40 Wait 500
50 GoTo 10
```

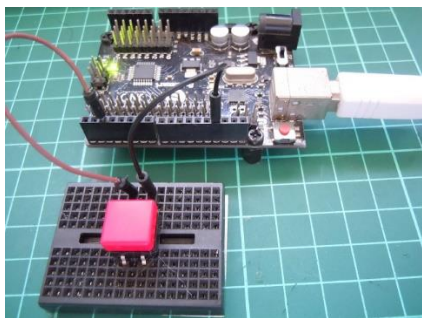
D2 ピンからのデジタル出力

```
10 Out 2,High
20 Wait 500
30 Out 2,Off
```

D2 ピンに接続したボタンの状態の取得（デジタル入力）

```
10 I=In(2,Pullup)
20 if i=Low Print "ボタンが押されました"
30 Wait 400
40 Goto 10
```

※ D2 ピンのデジタル入力は、プルアップ指定しています。



D3 ピンに接続した LED を PWM で 3 段階で輝度を変化させて点灯させる

```
10 PWM 3,255
20 Wait 500
30 PWM 3,127
50 Wait 500
60 PWM 3,63
70 Wait 500
80 PWM 3,0
```

A0 ピンからのアナログ入力、入力電圧を測定する

```
10 Cls
20 V=Map(Ana(A0),0,1023,0,5000)
30 Print Dmp$(V,3)
40 Wait 500
50 GoTo 20
>run
1.246
1.256
1.251
```

ここに紹介したコマンド・関数は一部です。

利用可能なコマンドの詳細については、「6 各コマンド・関数の詳細」を参照して下さい。

Arduino の利用可能なボードピン構成については、「1.6 ボードピン構成」を参照下さい。

1.9 文字コード及び文字列

「豊四季タイニーBASIC for Arduino 機能拡張版」は、文字（キャラクタ）コードとしてシフト JIS を採用しています。利用するターミナルソフトが全角文字をサポートしている場合は、全角日本語文字の利用も可能です。

```

COM5 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
TOYOSHIKI TINY BASIC
ARDUINO Extended version 0.07
1023byte free
OK
>load
OK
>list
10 For I=1 To 5
20 Print "こんにちは、世界！"
30 Next
OK
>run
こんにちは、世界！
こんにちは、世界！
こんにちは、世界！
こんにちは、世界！
こんにちは、世界！
OK
>

```

補足

- ターミナルソフト Tera Term で日本語を正しく表示するための設定については、「2.2 ターミナルソフトの通信条件の設定」を参照下さい。

プログラム内で全角文字は次の箇所で利用可能です。

- コメント : [Rem](#)、['](#)
- 文字列出力コマンド : [Print](#)、[Vmsg](#)、[Nmsg](#)、[GetFont](#)
- ファイル名を扱うコマンド : [Drive](#)、[Save](#)、[Load](#)

文字列を扱う関数（文字列出力コマンドで利用可能）では、全角文字もサポートしています。

関数一覧

- [Asc\(\)](#) : 文字コードを取得します
例：?Hex\$(Asc("あ"))
82A0
- [Len\(\)](#) : 文字列の文字数を取得します
例：?Len("あ Aa アイ漢字")
8
- [Byte\(\)](#) : 文字列のバイト数を取得します。
例：?Byte("あ Aa アイ漢字")
11
- [Chr\\$\(\)](#) : 文字コードに対応する文字を返します
例：?Chr\$(82A0)

補足

- プログラムでの文字列の扱い方については、「4.7 文字・文字列」および「4.8 文字列利用における制約」も参照下さい。

各コマンドの詳細については、「6 各コマンド・関数の詳細」を参照して下さい。

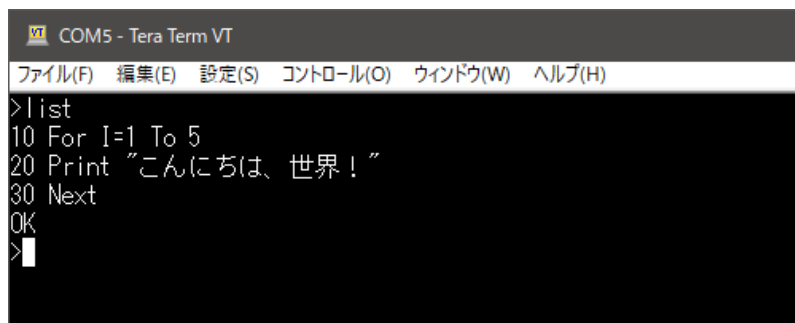
1.10 編集操作操作キー

「豊四季タイニーBASIC for Arduino 機能拡張版」では、シリアルコンソール画面上でラインエディタによるプログラム編集機能有しています。サポートする編集キーにを下記の表に示します。

編集キー一覧

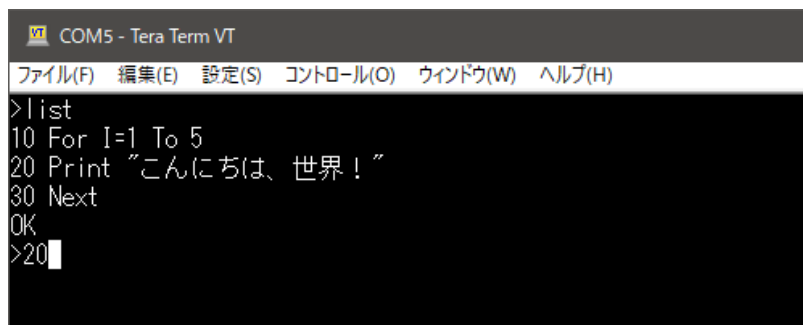
編集キー	機能
[ESC]、[Ctrl] + C、[F4]	プログラム中断、シリアルコンソールでは要2回押し
[F1]、[Ctrl] + L	画面の全消去
[F2]、[Ctrl] + D	カーソル位置の行消去
[BackSpace]、[Ctrl] + H	カーソル前の文字の削除
[Home]、[Ctrl] + A	カーソルを行の先頭に移動
[END]、[Ctrl] + O	カーソルを行の末尾に移動
[PageUP]、[↑]、[Ctrl] + Q	前の行の内容をラインエディタに表示
[PageDown]、[↓]、[Ctrl] + P	次の行の内容をラインエディタに表示
[Del]、[Ctrl] + B	カーソル位置の文字の削除
[←]	カーソルを左に移動
[→]	カーソルを右に移動
[Enter]、[Ctrl] + B	行入力の確定、改行
[Tab]	<ul style="list-style-type: none"> 行番号[Tab]で該当する行の内容をラインエディタに表示 エラー発生時に[Tab]でエラー行をラインエディタに表示

行番号 + [Tab]キーで編集対象行の指定



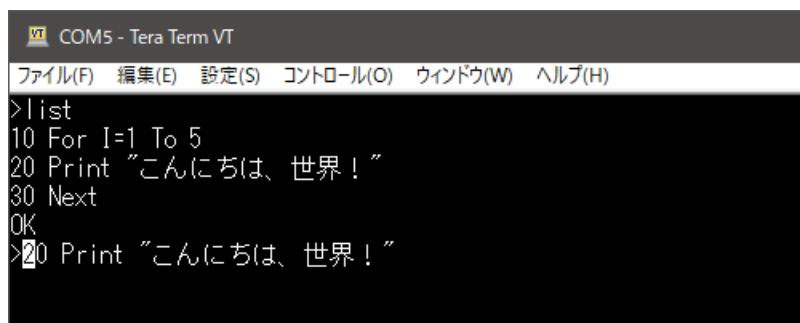
```
COM5 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
>list
10 For I=1 To 5
20 Print "こんにちは、世界！"
30 Next
OK
>
```

>(プロンプト)の後に、編集したい行番号を入力し、[Tab]キーを押すと、その行の内容を表示して編集できる状態にします。



```
COM5 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
>list
10 For I=1 To 5
20 Print "こんにちは、世界！"
30 Next
OK
>20
```

20 と入力し[Tab]キーを押すと、該当行の内容を表示します。



```
COM5 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
>list
10 For I=1 To 5
20 Print "こんにちは、世界！"
30 Next
OK
>20 Print "こんにちは、世界！"
```

この状態で、[←][→]キーでカーソル移動、[Home]、[End]キーで行頭、行末にカーソルを移動、[BS]、[Del]キーで文字削除、任意の文字入力、IME を使った日本語入力出来ます。

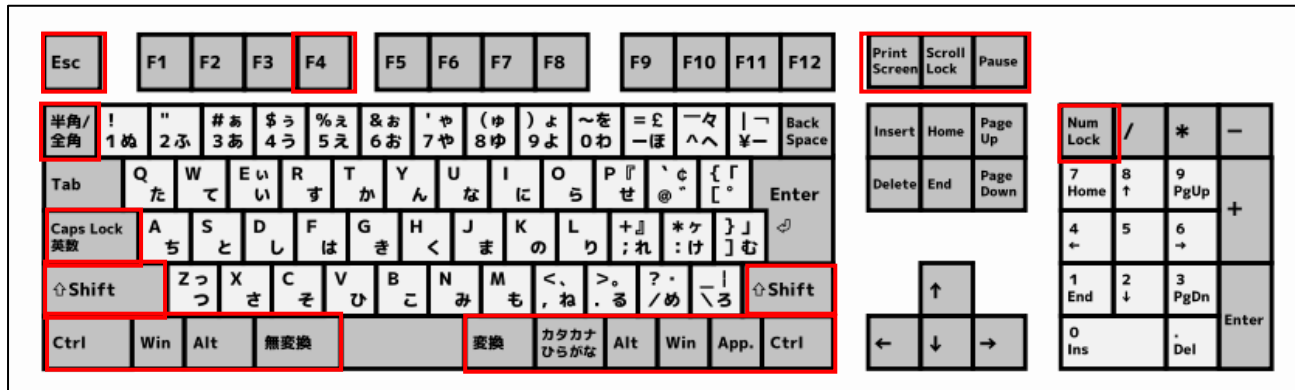
1.11 キー入力コード

[Inkey\(\)](#)関数等で入力したキーを判定する際は、キー入力コードを用います。

大部分のキー入力コードは入力する文字コード、ASCII コードと一致します。カーソルキー等の入力制御用のキーは、「豊四季タイニーBASIC for Arduino 機能拡張版」固有のコードとなります。

□ キー入力コードが取得出来ないキー

赤枠のキーは制御用または利用不可キーのためキーコードの取得が出来ません。



106/109 キーボード (※画像は ウィキペディア <https://ja.wikipedia.org/wiki/キー配列> より拝借)

□ キー入力コード一覧

キーボード入力にて取得出来るキー入力コードを以下に示します。

コードに数値の記載のあるキーに限りコードの取得が可能です。

IME を使った全角入力もサポートしています。

キー	併用なし		Shift		CTRL	
	文字	コード	文字	コード	文字	コード
[Backspace]		8				
[Enter]		13				
[Tab]		9				
[CapsLock]						
[Shift]						
[Ctrl]						
[Win]						
[Alt]						
[無変換]						
[空白 (Space)]	空白	32				
[変換]						
[かかひらがなローマ字]						
[メニュー]						
[Insert]		5				
[Home]		1				
[PageUp]		17				
[Delete]		2				
[End]		15				
[PageDown]		16				
[↑]		28				
[↓]		29				
[←]		31				
[→]		30				
[PrintScreen]						
[ScrollLock]						
[Pause/Break]						
[! ぬ]	1	49	!	33		
[" 2 ふ]	2	50	"	34		
[# 3 ああ]	3	51	#	35		
[\$ 4 うう]	4	52	\$	36		
[% 5 ええ]	5	53	%	37		
[& 6 おお]	6	54	&	38		
[' 7 やや]	7	55	'	39		
[(8 ゆゆ]	8	56	(40		
] 9 よよ]	9	57)	41		
[0 をわ]	0	48				
[- ほ]	-	45	=	61	CTRL -	29
[~ へ]	^	94	~	126		
[¥]	¥	92		124		

はサポートしていないキーです。

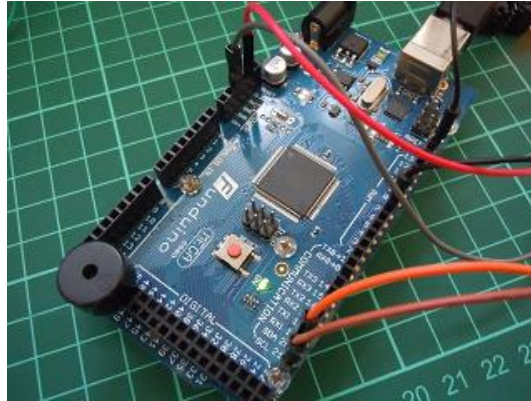
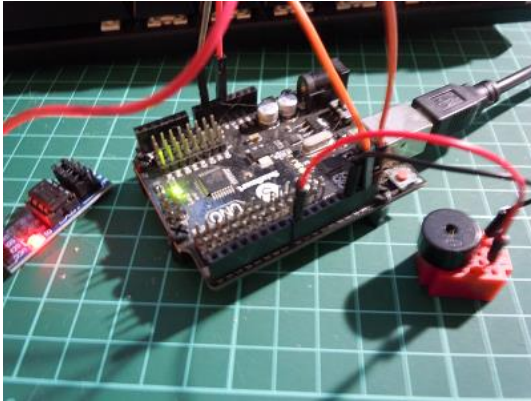
キー	併用なし		Shift		CTRL	
	文字	コード	文字	コード	文字	コード
[Q た]	q	113	Q	81	CTRL Q	17
[W て]	w	119	W	87	CTRL W	23
[E いい]	e	101	E	69	CTRL E	5
[R す]	r	114	R	82	CTRL R	18
[T か]	t	116	T	84	CTRL T	20
[Y ん]	y	121	Y	89	CTRL Y	25
[U な]	u	117	U	85	CTRL U	21
[I に]	i	105	I	73	CTRL I	9
[O ら]	o	111	O	79	CTRL O	15
[P せ]	p	112	P	80	CTRL P	16
[`@°]	@	64	`	96		
[{[「°]	[91	{	123		
[A ち]	a	97	A	65	CTRL A	1
[S と]	s	115	S	83	CTRL S	19
[D し]	d	100	D	68	CTRL D	4
[F は]	f	102	F	70	CTRL F	6
[G き]	g	103	G	71	CTRL G	7
[H く]	h	104	H	72	CHRL H	8
[J ま]	j	107	J	74	CTRK J	10
[K の]	k	107	K	75	CTRK K	
[L り]	l	108	L	76	CTRK L	12
[+:れ]	;	59	+	43		
[*:け]	:	58	*	42	CTRL *	28
[]}] む]]	93	}	125		
[Z っつ]	z	122	Z	90	CTRL Z	26
[X さ]	x	120	X	88	CTRL X	24
[C そ]	c	99	C	67	CTRL C	
[V ひ]	v	118	V	86	CTRL V	22
[B こ]	b	98	B	66	CTRL B	2
[N み]	n	110	N	78	CTRL N	14
[M も]	m	109	M	77	CTRL M	13
[<,、ね]	,	44	<	60		
[>。る]	.	46	>	62	CTRL >	30
[?/・め]	/	47	?	63	CTRL ?	31
[¥_ろ]	¥	92	_	95		

1.12 圧電スピーカー 単音出力対応

Arduino のデジタル出力 8 ピンと GND に圧電スピーカーを接続することで単音出力を行うことができます。

(Arduino MEGA2560 の場合 49 ピンを利用します)

単音出力は、PWM を利用しています。



利用部品

圧電スピーカー (圧電サウナ)

参考: 秋月電子 圧電スピーカー (圧電サウナ) (13 mm) PKM13EPYH4000-AO



単音出力機能を利用するために次のコマンドを用意しています。

コマンド一覧

[Tone](#) : 指定した周波数の単音出力します。

例: Tone 440,100

[NoTone](#) : 単音出力を停止します。

例: NoTone

[Play](#) : 指定した MML 文による演奏を行います。

例: Play "CDEFGAB"

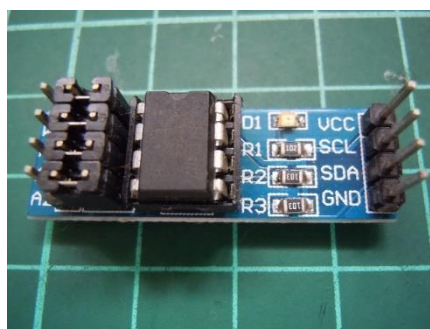
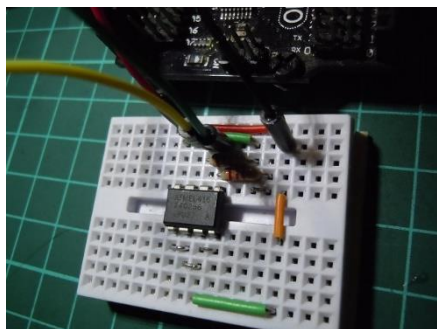
[Tempo](#) : PLAY コマンドで再生する演奏のテンポを指定します。

例: Tempo 150

各コマンドの詳細については、「6 各コマンド・関数の詳細」を参照して下さい。

1.13 I2C 接続 EEPROM へのプログラム保存対応

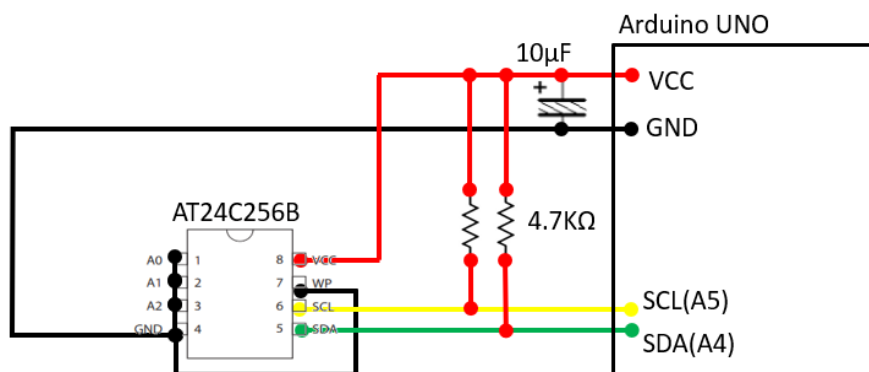
内部 EEPROM へのプログラム保存機能の他に、
I2C 接続の EEPROM にも保存に対応しています（以降、外部接続 EEPROM と呼称）。



プログラムは、ファイル名（12 文字）を付けて保存することが出来ます。
EEPROM の I2C アドレスの設定により、最大 8 個の EEPROM を同時接続して利用することが出来ます。

ハードウェア構成

AT24C256B を使った回路図及びブレッドボード上の実装



プルアップ抵抗は、2 ~ 10kΩの範囲（Arduino でプルアップしているため無く可）、
コンデンサは無くても可、I2C スレーブアドレスは\$50 に設定（A0 ~ A2 ピンで\$50 ~ \$57 の設定が可能）

対応している EEPROM は 24xxx 系（5V 対応のもの）となります。
4k バイト(32k ビット)~64k バイト(512 ビット) の容量に対応します。

動作確認 EEPROM

AT24C32、AT24C64、24LC64、AT24C256、24LC256、24FC512、AT25C32

容量が 1024 ビットタイプの EEPROM でも保存は可能ですが、利用出来る容量は 512 ビット（64k バイト）となります。

EEPROM に保存できるファイル数は、次の通りです。

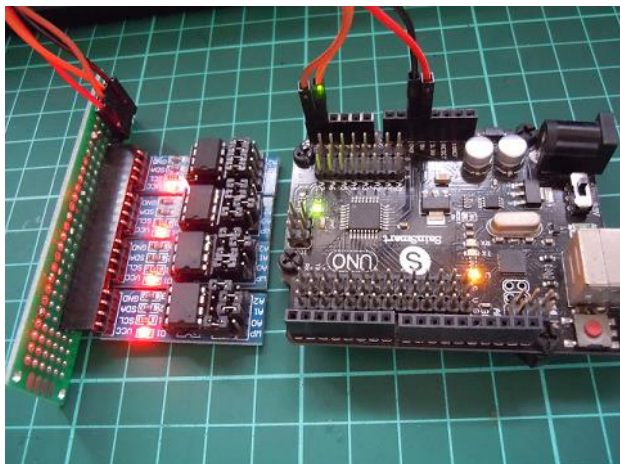
利用 EEPROM		保存可能ファイル数（個）	
EEPROM 表記	容量(k バイト)	プログラムサイズ 1024 バイト (デフォルト)	プログラムサイズ 512 バイト
24xx32	4	3	7
24xx64	8	7	15
24xx128	16	15	31
24xx256	32	31	62
24xx512	64	63	124

プログラムサイズは標準では 1024 バイトです。tconfig.h の定義で変更可能です。

```
#define PRGAREASIZE 1024 // プログラム領域サイズ(Arduino Uno 512 ~ 1024 デフォルト:1024)
```

EEPROM モジュールに異なるスレーブアドレスを割り付けることにより、複数の EEPROM の接続をサポートします。
その場合の利用する EEPROM の指定は、Drive コマンドを利用します。

市販モジュールを 4 つ同時接続した例（接続は自作アダプタ利用）



コマンド一覧

[Drive](#)

：利用する EEPROM を切り替えます。

例：Drive \$51 または Drive "B"

[Format](#)

：EEPROM の保存領域の初期化を行います。

例：Format 64,"PROG010"

[Save](#)

：プログラムを指定したファイル名で保存します。

例：Save "サンプル.BAS"

[Load](#)

：指定しプログラムをロードします。

例：Load "サンプル.BAS"

[Erase](#)

：指定したファイルを削除します。

例：Erase "サンプル.BAS"

[Files](#)

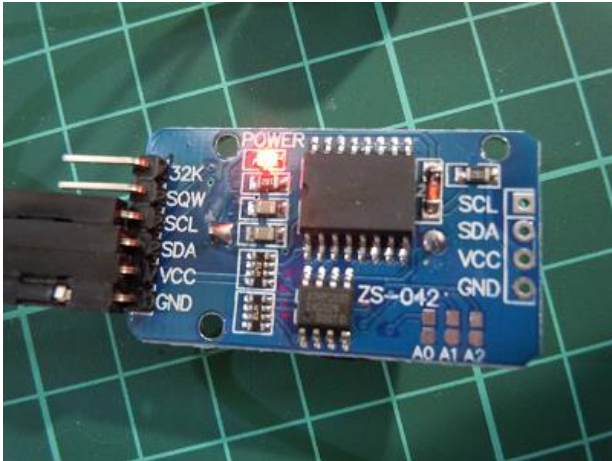
：ファイルの一覧を表示します。

例：Files ""

各コマンドの詳細については、「6 各コマンド・関数の詳細」を参照して下さい。

1.14 I2C 接続 RTC DS3231 対応

I2C バスに接続した I2C RTC DS3221 を利用した時刻の設定と参照を行うことができます。



RTC モジュールのと SCLMW25616L 実験用表示モジュールの結線

GND	GND
VCC	VCC (5V)
SDA	A4 (MEGA2560 の場合 20 ピン)
SCL	A5 (MEGA2560 の場合 21 ピン)

次の専用コマンドと関数を用意しています。

コマンド・関数一覧

Date	: 現在時刻を表示します。 例: Date
SetDate	: 時刻を設定します。 例: SetDate 2018,3,1,5,10,30,0
GetDate	: 現在日付を指定した変数に格納します。 例: GetDate A,B,C,D
GetTime	: 現在時刻を指定した変数に格納します。 例: GetTime A,B,C
Date\$: 現在時刻の文字列を返します。 例: Print Date\$()

各コマンドの詳細については、「6 各コマンド・関数の詳細」を参照して下さい。

1.15 美咲フォント対応

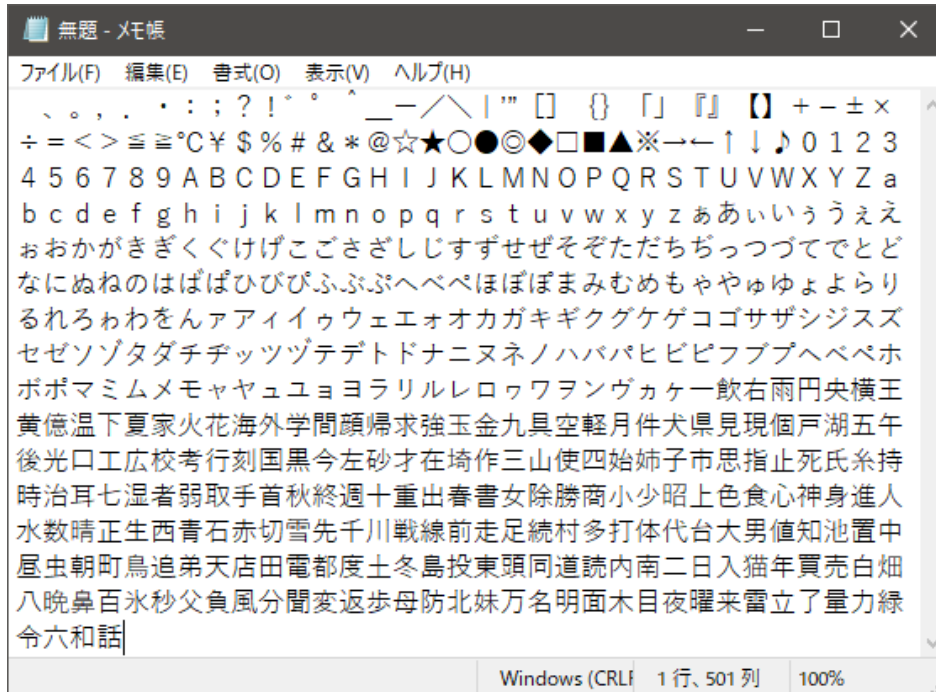
美咲フォントは、シフト JIS コード 8 x 8 ドットの日本語を含むフォントデータです。

ドットマトリックス等への日本語を含めたフォントの表示に利用可能です。

容量的な問題から収録文字数は 500 文字（非漢字が 288 文字、漢字 212）に絞っています。

ttconfig.h の設定により非漢字 288 文字の利用も可能です。

サポートする 500 文字



コマンド・関数一覧

[GetFont](#) : 指定したシフト JIS コードのフォントデータを取得します（数値関数）。

例：R = GetFont(Mem, Asc(“あ”))

各コマンドの詳細については、「6 各コマンド・関数の詳細」を参照して下さい。

1.16 赤外線リモコン受信対応

赤外線受信モジュール TL1838 等を利用して、NEC 方式の赤外線リモコンの送信データの受信を行うことができます。
 接続図



上記の例では、データ受信に端子 3 を利用していますが、任意の端子でのデータ受信が可能です。

コマンド・関数一覧

[IR\(\)](#) : 赤外線リモコンからのデータ受信を行い、その値を返します（数値関数）

例：R=IR(3)

各コマンドの詳細については、「6 各コマンド・関数の詳細」を参照して下さい。

1.17 NeoPixel(WS2812B)対応

Neopixel (WS2812B) を最大 64 個制御することが出来ます。

任意のピクセル毎に任意の色指定を行うことが出来ます。色は 0~255 の 256 色指定となります。

色指定には RGB() 関数を利用すると便利です。

また、8 x 8 ドットマトリックスの場合、美咲フォントを利用したメッセージ表示を行うことが出来ます。

結線

NeoPixel	Arduino
1 VDD	5V
2 DOUT	(未接続)
3 VSS(GND)	GND
4 DIN	MOSI (11 ピン)

次の専用コマンドと関数を用意しています。

コマンド・関数一覧

- [Ninit](#) : NeoPixel の利用を開始します。
例: Ninit Mem2,64
- [Nbright](#) : 輝度設定を行います。
例: Nbright 2
- [Ncls](#) : 表示をクリアします。
例: Ncls
- [Nset](#) : 指定 LED の色設定します。
例: Nset 0,RGB(7,0,0)
- [Npset](#) : 8x8 マトリックスの指定位置にピクセル設定します。
例: Npset 3,3,COLOR(0,7,0)
- [Nmsg](#) : 8x8 マトリックスにメッセージ文を表示します。
例: Nmsg 100,RGB(0,0,3),"こんにちは"
- [Nscroll](#) : 8x8 マトリックスを指定した方向に 1 ドット分、スクロールします。
例: Nscroll Left
- [Nline](#) : 8x8 マトリックスに直線、矩形、矩形塗りつぶしを描画します。
例: Nline 0,0,7,7
- [Nupdate](#) : バッファ内容を表示に反映します。
例: Nupdate
- [Nshift](#) : LED の表示を指定方向にシフトします。
例: Nshift 0
- [Npoint\(\)](#) : 8x8 マトリックスの指定座標の色コードを取得します。
例: C=Npoint(5,5)
- [RGB\(\)](#) : 赤・青・緑の指定を 8ビット色コードに変換します。
例: C=RGB(7,7,3)

各コマンドの詳細については、「6 各コマンド・関数の詳細」を参照して下さい。

1.18 VFD ディスプレイ (MW25616L) 対応

ボード搭載 VFD ディスプレイ MW25616L を制御するためのコマンドをサポートします。

デフォルトでは、本機能は利用出来ません。スケッチの `ttconfig.h` の下記の設定を

```
#define USE_CMD_VFD    0  // VFD モジュールコマンドの利用(0:利用しない 1:利用する デフォルト:0)
```

1 に変更し、スケッチを書き込んで下さい。

コマンド一覧

<u>Vcls</u>	: VFD の表示内容を消去します。 例: <code>Vcls</code>
<u>Vbright</u>	: 輝度を 0(0%)~255(100%) の範囲で設定します。 例: <code>Vbright 100</code>
<u>Vdisplay</u>	: 表示のオン・オフを設定します。 例: <code>Vdisplay OFF</code>
<u>Vmsg</u>	: 指定したメッセージ文を指定速度でスクロール表示します。 例: <code>Vmsg 10,"日本語のメッセージ表示テスト"</code>
<u>Vscroll</u>	: 指定長の空白を指定速度でスクロール表示します。 例: <code>Vscroll 256,10</code>
<u>Vput</u>	: 表示バイトデータを指定速度で直接指定します。 例: <code>Vput MEM,16,10</code>

各コマンドの詳細については、「6 各コマンド・関数の詳細」を参照して下さい。

1.19 16 ドット日本語フォント (GT20L16J1Y) 対応

VFD ディスプレイ MW25616L 表示専用に日本フォントを搭載しています。

VMSG コマンドにて指定したメッセージ文に対応するフォントデータを随時参照利用します。

デフォルトでは、本機能は利用出来ません。スケッチの `ttconfig.h` の下記の設定を

```
#define USE_CMD_VFD    0  // VFD モジュールコマンドの利用(0:利用しない 1:利用する デフォルト:0)
```

1 に変更し、スケッチを書き込んで下さい。

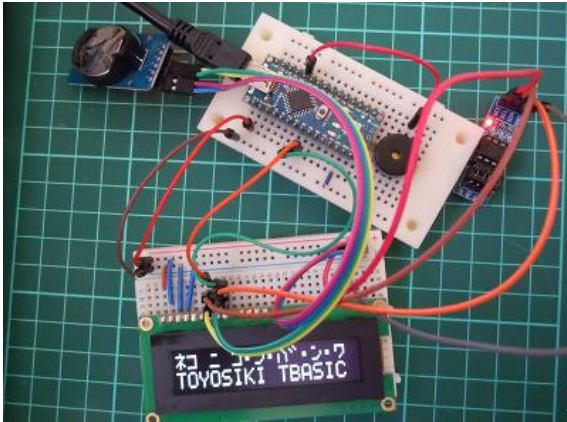
1.20 有機 EL キャラクタディスプレイ SO1602AWWB 対応

I2C 接続にて利用可能な有機 EL キャラクタディスプレイ SO1602AWWB に対応しています。

デフォルトでは、本機能は利用出来ません。スケッチの `ttconfig.h` の下記の設定を

```
#define USE_SO1602AWWB 0 // 有機 EL キャラクタディスプレイ SO1602AWWB(0:利用しない 1:利用する デフォルト:0)
```

を 1 に変更し、スケッチを書き込んで下さい。



製品情報

有機ELキャラクタディスプレイモジュール 16×2行 白色

<http://akizukidenshi.com/catalog/g/gP-08277/>

結線

有機 EL	Arduino
1 GND	GND
2 VDD	3.3V
3 /CS	GND
4 SA0	GND
7 SCL	A5 SCL(MEGA2560 の場合 21 ピン)
8 SDA_in	A4 SDA(MEGA2560 の場合 20 ピン)
9 SDA_out	A4 SDA(MEGA2560 の場合 20 ピン)

次の専用コマンドと関数を用意しています。

コマンド・関数一覧

[Cprint](#) : 指定した文字列を表示します。
例: `Cprint "Hello,World"`

[Ccls](#) : 表示内容を消去します。
例: `Ccls`

[Ccurs](#) : カーソル表示・非表示を設定します。
例: `Ccurs OFF`

[Clocate](#) : カーソルを移動します。
例: `Clocate 0,1`

[Ccons](#) : コントラストの設定を行います。
例: `Ccons 128`

[Cdisp](#) : 画面表示・非表示を設定します。
例: `Cdisp OFF`

各コマンドの詳細については、「6 各コマンド・関数の詳細」を参照して下さい。

2. 利用環境設定

本章では、「豊四季タイニー-BASIC for Arduino 機能拡張版」を使ってプログラムの開発を行うための環境及びその設定について、解説します。

2.1 ファームウェアの書き込み

「豊四季タイニー-BASIC for Arduino 機能拡張版」を利用するには、Arduino IDE にて配布スケッチをコンパイルして書き込む必要があります。

必要環境

- ① Arduino IDE が利用可能なパソコン
- ② Arduino IDE 1.8.5 以降
- ③ Arduino Uno（互換機含む） または MW25616L 実験用表示モジュール
または Arduino MEGA2560（互換機含む）

「豊四季タイニー-BASIC for Arduino 機能拡張版」用ファームウェア書き込み手順

- ④ 「豊四季タイニー-BASIC for Arduino 機能拡張版」プロジェクトを下記のリンクから
ファイル **ttbasic_mw25616l-master.zip** をダウンロードします。
https://github.com/Tamakichi/ttbasic_mw25616l/archive/master.zip
- ⑤ ダウンロードしたファイルを解凍します。
エクスプローラーにて **ttbasic_mw25616l-master.zip** を選択し、
マウス右クリックにて開いたメニューから「すべて展開」を選択することで解凍することが出来ます。
解凍したフォルダ内の **ttbasic** フォルダがスケッチとなります。
- ⑥ Arduino IDE を起動し、スケッチ **ttbasic** を読み込みます。
ボードの設定項目は利用するボードに合わせて設定して下さい。
- ⑦ Arduino Uno などの利用するボードを接続します。
接続後は、シリアルポート等の設定を行って下さい。
- ⑧ Arduino IDE でスケッチをコンパイルし、書き込みます。

以上の操作で書き込みが完了します。次のステップとしては、ターミナルソフトを使って Arduino に接続します。

補足

スケッチの **ttconfig.h** にて利用できる機能の有効・無効の設定を行うことが出来ます。

利用する Arduino IDE、ボードによりフラッシュメモリの容量不足により書き込みが出来ない場合は、利用頻度の少ない機能を無効化することを試して見て下さい。



2.2 ターミナルソフトの通信条件の設定

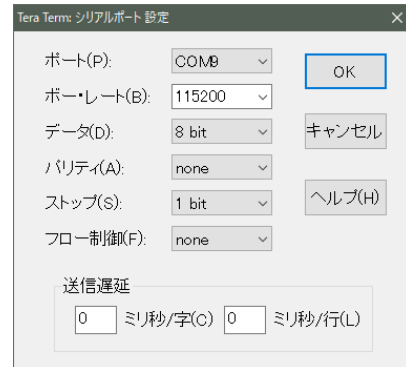
ここでは、Windows 10 パソコン上でターミナルソフト TeraTerm を利用する場合の設定について解説します。
TeraTerm をお使いの場合は、下記のサイトからダウンロード・インストールして下さい。

通信条件の設定

ファームウェアを書き込んだ後、Arduino を USB ケーブルにて続した状態にします。

①通信条件の設定

ポート	設定値
ポート	各自の環境に合わせて設定
ボー・レート	115200(任意)
データ長	8 bit
パリティビット	無し
ストップビット	1bit
フロー制御	無し



USB 接続の場合、ボー・レートの設定任意です。

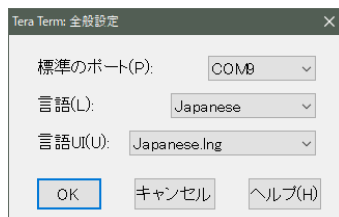
ボー・レートの設定は無視され USB 通信の最適な速度で通信を行うようです（推測）。

TeraTerm の設定は、メニュー[設定] - [シリアルポート]から[シリアルポート設定]画面を開いて行います。

②言語等に関する設定

「豊四季タイニー-BASIC for Arduino 機能拡張版」には 1 バイトコードの半角カタカナの利用をサポートします。半角カタカナ利用のためには、利用する文字列コードとしてシフト JIS を指定します。

[設定] - [全般設定] - [全般設定]画面 の設定



言語 : Japanese

言語 UI : Japanese.lng

[設定] - [端末の設定] - [端末の設定]画面 の設定



端末サイズ : 80x24 以上に設定

漢字-受信 : SJIS

漢字-送信 : SJIS

改行コード受信 : AUTO(or CR)、送信 CR

ウィンドウサイズ : ☒チェックを入れる

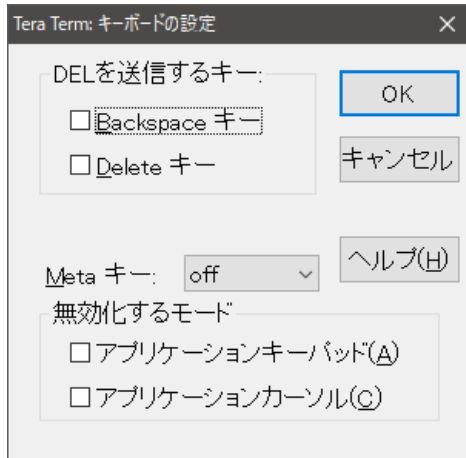
③キーボードの設定

[設定] - [キーボード] - [キーボード]画面 の設定

DEL キーを正しく動作させるには、DEL を送信するキーの設定で、次の設定を行うます。

Backspace キー ; チェックを外す

Delete キー ; チェックを外す



ターミナルソフトの設定を行い、Arduino に接続してすると、次のメッセージが表示されます。

```
TOYOSHIKI TINY BASIC
ARDUINO Extended version 0.07
1023byte free
OK
>
```

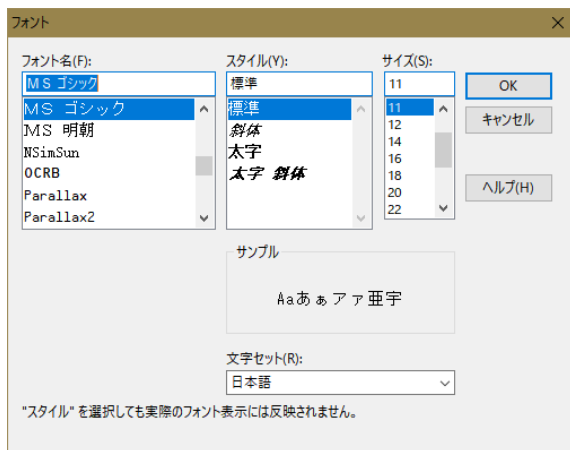
表示されない場合は、ケーブル及びターミナルソフトの通信条件の設定を見直して下さい。

ターミナルソフトのウィンドウサイズをマウス操作等で変更すると表示内容が消えてしまいます。

この場合は[ENTER]キーを押すことで、ライン編集カーソルが表示されます。

表示された文字が小さい場合はフォントを調整して下さい。

フォントは、文字セットに日本語が設定可能なフォントを利用して下さい。



最後に、設定した内容を保存します。

メニュー [設定] - [設定の保存]で TERATERM.INI に上書き保存して下さい。

2.3 コンソール画面の表示設定

「豊四季タイニーBASIC for Arduino 機能拡張版」のコマンドにてコンソール画面のフォントの色、背景色の設定を行うことができます。

コマンド

コマンド	説明
Color n [, m]	テキスト文字色、背景色を設定します。 n: 文字色 0~8 m: 背景色 0~8 COLOR コマンドの色指定については、「6.17 Color 文字色の設定」を参照して下さい。 COLOR コマンド実行後、CLS コマンドを実行するか、[F1]キーを押すことで画面全体に色設定が反映されます。
Attr n	文字表示属性を設定します。 n: 0 ノーマル、 1 下線、 2 反転、 3、プリント、 4 ボールド
Cls	画面表示のクリア キーボードの [F1]キー、CTRL+L キーでもクリアできます。

(補足) ターミナルソフト TeraTerm の設定により、利用するフォントの種類・サイズ等の指定を行うことができます。
TeraTerm の設定については、TeraTerm のマニュアルを参照して下さい。

2.4 プログラムの保存と読み込み

作成したプログラムは内部 EEPROM に保存することが出来ます。

内部 EEPROM メモリの操作

- 保存 [Save](#) プログラム番号(0～ ※1)
- 読み込み [Load](#) プログラム番号(0～ ※1)
- 保存プログラムの一覧表示 : [Files](#)

```
files
0:CLS
OK
```

FILES コマンドはプログラムの先頭行を表示します。

プログラム番号の格納領域にプログラムが保存されていない場合は、(none)が表示されます。

- 削除 [Erase](#) プログラム番号(0～ ※1)

※1 スケッチのコンパイル条件の修正による、プログラム保存領域を 512 バイト以下にした場合、保存数が自動で増えます。デフォルトではプログラム番号は 0 のみの指定となります。

保存したプログラムは、リセット時のタイミングでロードして起動することも可能です。

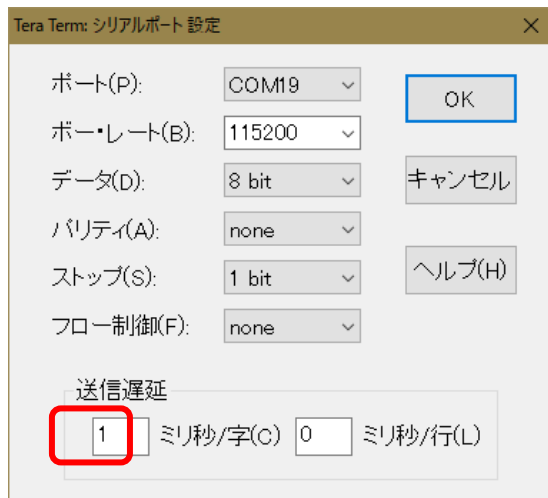
詳細は、「2.7 プログラムの自動起動」を参照して下さい。

オプション機能として、I2C 接続の EEPROM へのプログラム保存も可能です。

詳細については、「1.13 I2C 接続 EEPROM へのプログラム保存対応」を参照して下さい。

2.5 パソコンからのプログラム読み込み

パソコンからマイコンボードにプログラムを転送する方法としては、
ターミナルソフトの画面にプログラムソースコピー＆ペーストする方法が簡単でおすすめです。
この際、シリアルポートの設定にて[送信遅延]を下記のように設定して下さい。



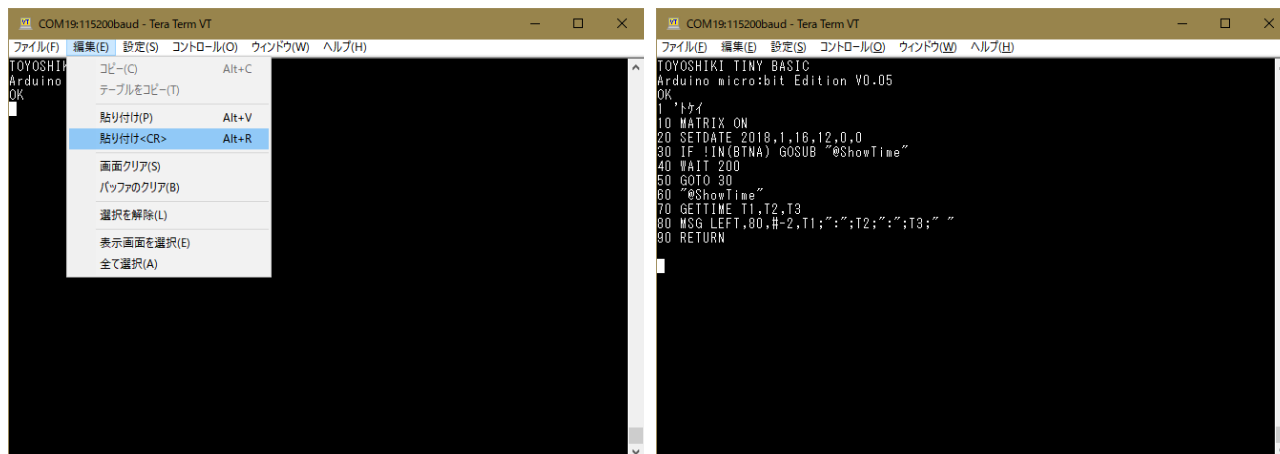
試しに、次のプログラムをコピーしてパソコンから MW25616L 実験用表示モジュールにプログラムを転送してみましょう。

```

10 'VDF表示サンプル
20 TONE 440,100:TONE 880,100
30 @(0)="熱いぞ！埼玉！"
40 @(1)="豊四季タイニーBAS I C"
50 @(2)="VFD対応バージョン"
60 @(3)="只今開発中！"
70 @(4)="ちょっとメモリーのに厳しい.."
80 VBRIGHT 255
90 FOR S=0 TO 4
100 VCLS:VMSG 10,STR$(@(S))
110 FOR I=0 TO 255 STEP 15
120 VBRIGHT 255-I:WAIT 40
130 NEXT I
140 WAIT 500
150 VBRIGHT 255:WAIT 500
160 VSCROLL 256,3
170 WAIT 500
180 NEXT S
190 GOTO 80

```

TeraTerm では、メニュー [編集] - [貼り付け<CR>] か ALT+R キーにて貼り付けることができます。



(注意) プログラムの読み込みが終わったら、送信遅延の設定は元に戻して下さい。プログラム実行時にコンソール画面からの文字入力を行うコマンド INPUT、INKEY()にて遅延が発生し、意図しない動作をする場合があります。

2.6 プログラムをパソコンに保存する

SRAM 上、内部 EEPROM メモリ上のプログラムをパソコンに保存するには、ターミナル上で LIST コマンドを実行し、ターミナルソフトをスクロールする等して、テキストを選択してコピー＆ペースト等を行ってメモ帳等に張り付けて保存して下さい。

```

COM9:115200baud - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
OK
>list
1 'VFDジグザグ表示
10 VCLS
20 @(0)=1
30 FOR I=0 TO 14
40 VPUT ARRAY,2,5
50 @(0)=@(0)<<1
60 NEXT I
70 FOR I=0 TO 14
80 @(0)=@(0)>>1
90 VPUT ARRAY,2,5
100 NEXT I
110 GOTO 30
OK
>

```

ファームウェアの更新を行う場合は事前に、内部 EEPROM メモリに保存されているプログラムをパソコンに保存して下さい。ファームウェア更新後は、保存しているプログラムが動かなくなる場合があります。

2.7 プログラムの自動起動方法

起動時またはリセット時にフラッシュメモリー上のプログラム番号 No.0 のプログラムを起動することが出来ます。その方法について説明します。

自動起動条件

- 電源 ON またはリセット時に No.7 ピン が HIGH(5V) の場合、プログラム番号 No.0 を自動起動します。
- No.7 ピンはスケッチの修正により、別のピンに変更することが可能です

利用するボードにより、起動時に No.7 ピンを GND に接続しないと、No.0 のプログラムが自動起動してしまう場合があります。この場合は、No.7 ピンを GND に接続してください。

3. 使ってみよう！

この章では、「レガシーBASIC（古き良き時代の BASIC 言語）の雰囲気知らない方々にその雰囲気をつかんでもらいましょう」ということで、軽いフットワークで解説進めます。

準備は OK？ 前章の実行環境の設定は完璧？ それでは電源を入れて使ってみよう！

3.1 入力可能状態は OK■（カーソル）

```
TOYOSHIKI TINY BASIC
ARDUINO Extended version 0.07
1023byte free
OK
>■
```

この状態は、「いつでもこいや〜」状態。
きみのコマンド入力を待っている状態。
何か打ち込んでみよう！

3.2 Hello,world を表示してみる

直接、次の文字を打ってみよう。

```
>print "Hello,world" Enter
```

入力ミスの修正は[BS]キー、[DEL]キー、カーソルキーなどいつものキーが使えるよ。
入力出来たら、最後に[Enter]キーを押してみよう。

```
>print "Hello,world" Enter
Hello,world
OK
>■
```

表示出来たね。PRINT は文字列を表示するコマンドなのです！
入力は大文字、小文字どちらでも OK！

3.3 Hello,world を Hello,Tiny BASIC に変更してみる

ターミナルソフトのコピー＆ペースト機能を使って、前回の入力をコピーします。
カーソルキー[←] [→]でカーソルを動かして、world の w に移動してみよう。
入力出来たら、最後に[Enter]キーを押してみよう。

```
>print "Hello,world"
Hello,world
OK
>■
```

ターミナルソフトのコピー＆ペースト機能を使って、前回の入力をコピーします。
[DEL]キーで world を削除して、代わりに Tiny BASIC と入力しよう。
入力後に、[Enter]キーを押すのを忘れずに！

```
>print "Hello,Tiny BASIC" Enter
Hello,Tiny BASIC
OK
>■
```

こんな感じでラインエディタを利用して効率よくプログラムを入力します。

3.4 Print の前に 10 を追加してプログラムにしよう

再びターミナルソフトのコピー＆ペースト機能を使って、前回の入力をコピーします。
カーソル操作で print の前に 10 を挿入して[Enter]キーを押してみよう。

```
>10print "Hello,Tiny BASIC" Enter
>■
```

これでプログラムとして登録された。LIST コマンドで確認してみよう。
カーソルを OK の下の空き行に移動して、list と入力して[Enter]キーを入力

```
>10 print "Hello,Tiny BASIC" Enter
>list Enter
10 Print "Hello,Tiny BASIC"
OK
>■
```

登録したプログラムの内容が表示されたね。
まだ、行番号 10 しか登録していないので、その 1 行だけの表示だ。
このように、先頭に数字(=行番号)をつけると行番号の後ろのコマンドがプログラムとして登録されるのだ。

3.5 プログラムを実行してみよう

う〜ん、気分的に、画面を綺麗にしてから表示したくなった。
画面をきれいにしてから、3回実行してみよう。

```
>cls Enter
>run Enter
Hello,Tiny BASIC
OK
>run Enter
Hello,Tiny BASIC
OK
>run Enter
Hello,Tiny BASIC
OK
>■
```

3回同じ処理が実行出来たね。
CLS は画面を消すコマンド、RUN はプログラムを実行。よく使うコマンドだよ。

3.6 プログラムに追加する

今行ったの“画面きれいにしてから、3回表示”をプログラムにやらせてみよう！
次をように入力して、プログラムとして登録するよ。

```
>5 cls Enter
>7 for i=1 to 3 Enter
>20 next Enter
```

入力したプログラムを確認！

```
>list Enter
5 Cls
7 For I=1 To 3
10 Print "Hello,Tiny BASIC"
20 Next
OK
>
```

今入力した 5,7,10 行が追加されたね。プログラムは行番号順に表示されるよ。
それでは実行！ run コマンドで実行するよ。

```
>run Enter
Hello,Tiny BASIC
Hello,Tiny BASIC
Hello,Tiny BASIC
OK
```

3 行表示出来た。
プログラムの実行順番は行番号順。CLS が最初に実行されるよ。
FOR 文は繰り返しを行う命令。3 回 10 行の表示を行ったよ。

3.7 行番号を並びなおす

もう一回、プログラムを確認しよう。

```
>list Enter
5 Cls
7 For I=1 To 3
10 Print "Hello,Tiny BASIC"
20 Next
OK
>
```

行番号の 5,7,10,20 と並びがちょっと美しくないですね。
直してみよう。

RENUM コマンドを使ってプログラムの行番号を 10 ずつに整えるよ。

```
>renum Enter
OK
>list Enter
10 Cls
20 For I=1 TO 3
30 Print "Hello,Tiny BASIC"
40 Next
OK
>
```

きれいに並んだね。

3.8 やっぱ 10 行は不要、削除したい

やっぱり、直前の表示内容は消したくない。

10 行はいらない。削除したい。行番号だけ入力して[Enter]を押してみよう。

```
>10 Enter
```

これで 10 行が削除されるよ。確認してみよう。

```
>list Enter
20 For I=1 TO 3
30 Print "Hello,Tiny BASIC"
40 Next
OK
>
```

10 行が無くなったね。

こんな感じ行番号だけの入力、行削除。よく使う操作だよ。

DELETE コマンドを使っても行の削除ができるよ。

```
>delete 10 Enter
```

20 行から始まるのは美しくない。行番号をまた振り直そう。

```
>renum Enter
OK
>list Enter
10 For I=1 TO 3
20 Print "Hello,Tiny BASIC"
30 Next
OK
>
```

3.9 プログラムを保存したい

今作成したプログラムは SRAM 上に保存されているのだ。

マイコンの電源を切ると消えてしまう・・・

消えないで欲しい・・・。そんな時、SAVE コマンドを使うよ。

SAVE コマンドはプログラムをフラッシュメモリに保存するよ。

フラッシュメモリに保存されたプログラムは電源を切っても消えないよ。

次のコマンドを打ってみましょう。

```
>save Enter
OK
```

これで保存できたはずだ。

3.10 保存されたプログラムを読み込み

本当に保存されているか確認してみましょう。

SRAM 上のプログラムを消してフラッシュメモリを読み込みと試してみよう。

NEW コマンドを打ってプログラムを初期化するよ。

```
>new Enter
OK
```

これで消えたはず。

LIST コマンドで確認。

```
>list Enter
OK
```

何も出てこない。本当に消えた。

次に LOAD コマンドでフラッシュメモリからプログラム読み込むよ。

```
>load Enter
OK
>list Enter
10 For I=1 To 3
20 Print "Hello,Tiny BASIC"
30 Next
OK
>
```

成功！ 復活出来た！・・・

保存しているプログラムは FILES コマンドで確認できるよ。

0 番に保存されているよ。

```
>files Enter
0:For I=1 To 3
OK
>
```

これで基本的な操作は終了だ。

基本はバッチリ、準備万端！ Tiny BASIC を使った更なるプログラミングをエンジョイしよう！

4. プログラム構成要素の説明（コマンド・関数等）

本章では、「豊四季タイニーBASIC for Arduino 機能拡張版」のプログラム言語について解説します。

4.1 プログラムの構造と形式

次のプログラムは、「こんにちは、世界！」を5回表示するプログラムです。

```
1 'サンプルプログラム
10 For I=1 To 5
20 For J=1 To I:Print " ";:Next
30 Print "こんにちは、世界！"
40 Next
>run
  こんにちは、世界！
  こんにちは、世界！
  こんにちは、世界！
  こんにちは、世界！
  こんにちは、世界！
OK
>
```

プログラムは複数の行（行番号+命令文）で構成されます。

- 例) 1 'サンプルプログラム : プログラム内容を説明するコメントです。
 例) 50 Print "こんにちは、世界！" : 文字を表示します。

命令文は1つまたは複数のコマンドや式で構成されます。

- 例) 20 For J=1 To I:Print " ";:Next : 行内にコロン(:)で複数のコマンドを記述出来ます。

コマンドによっては、引数を持ちます。

- 例) 50 Print "こんにちは、世界！" : Print コマンドは引数の文字列を表示します。

基本的に、プログラムは行番号順にシーケンシャルに実行されます。

- 例) 1行から20行は順番に実行されます。

制御命令により、分岐や繰り返しを行うことが出来ます。

- 例) 20行~30行はFor文により5回、繰り返し実行が行われます。

このようにプログラムは様々な要素で構成れます。

次節からは、これらの各構成要素について解説します。

4.2 行と命令文

□ 行

行とは利用者がコンピュータに対して指示を行う単位です。行は命令文で構成されます。
「豊四季タイニーBASIC for Arduino 機能拡張版」では、行の単位で利用者からの指示を処理します。

スクリーンエディタ、およびコンソールから、

```
>Print "Hello" Enter
Hello
OK
```

のように **Enter** キーの入力により、行の単位で指示を受け取りその処理を行います。
また、行の先頭に行番号を付けた場合、

```
>10 Print "Hello" Enter
```

「指定した行番号で行（命令文）をプログラムに登録せよ」

との指示であると判断し、命令文は実行せずに、行（命令文）をプログラムとしてメモリーに登録します。

```
>10 Enter
```

のように行番号だけを指定した場合は、

「指定した行番号で行（命令文）を空にせよ（削除せよ）」

との指示であると判断し、該当する行をプログラムから削除します。

□ 命令文

命令文とはコマンド、式、関数、コメント、ラベルを組み合わせて記述した命令です。
コマンドはコロン:を使って1行に複数記述することができます。

```
10 I=I+1:Locate 0,I:Color 7,0:Print "Hello":GoTo 50:'サンプル
```

コメント文

文において、REM、および省略形のシングルクォーテーション'を使った以降の文はコメントとなります。

例：

```
10 'Smple program
20 Rem Sample program
30 Print "Hello":'print hello
```

ラベル

行の先頭のダブルクォーテーションで囲った文字列はラベルとして動作します。
ラベル自体は実行時に何も行いません。GoTo 文、GoSub 文のジャンプ先の指定で利用します。
ラベルの後ろにはコマンドを続けて記述することができます。

例：

```
10 "LOOP":Print "Hello"
20 GoTo"LOOP"
```

4.3 制御文

制御文は制御命令を使ったプログラムの記述文です。複数のキーワード、式の組み合わせで構成されます。プログラムは Run コマンドを実行することにより、通常は先頭行から行番号順に逐次実行されます。この逐次実行は制御文を用いることで条件分岐、繰り返しを行うことができます。

For 文の例：繰り返し処理を行う

```
10 For A=0 To 10 Step 5
20 Print A
30 Next
```

以下に制御命令の一覧を示します。

制御命令	説明
GoTo 行番号 GoTo "ラベル"	指定行へのジャンプ 行番号：ジャンプ先の行番号 ラベル：ジャンプ先の行をラベルで指定 例：GoTo 500 GoTo "LOOP"
GoSub 行番号 GoSub "ラベル"	サブルーチンの呼び出し 行番号：呼び出しサブルーチン先頭行 ラベル：呼び出しサブルーチンをラベルで指定 例：GoSub 500 GoSub "print_sub"
Return	GOSUB 文サブルーチン呼び出しからの復帰
If 条件式 真の場合の実行文 If 条件式 真の場合の実行文 Else 偽の場合に実行文	条件判定による実行文の実行 条件式：式の計算結果が 0 以外は真、0 の場合偽 例：If X>=80 Y=Y+1;X=0 If D=1 X=X+1 Else X=X-1
For 変数=初期値 To 最終値 実行文(複数行可能) Next [変数] For 変数=初期値 To 最終値 Step 増分 繰り返し実行文 Next [変数] ※Next の変数名は省略可能	繰り返し実行 初期値：変数の初期値 最終値：変数の最終値 増分：変数の増分（マイナス値も可） 例：For I=1 To 10 Step 2 Print "I=";I Next I
End	プログラムを終了する 例：End
On Timer 周期 GoTo 行番号 On Timer 周期 GoTo "ラベル" On Timer 周期 GoSub 行番号 On Timer 周期 GoSub "ラベル"	タイマーイベント 指定した周期で GoTo、GoSub を実行可能にする タイマーイベントの制御には、Timer コマンドを利用
Timer 設定	タイマーイベントの実施、停止を制御する
On Pin ピン番号,ピンモード,ピン変化 GoTo 行番号 On Pin ピン番号,ピンモード,ピン変化 GoTo "ラベル" On Pin ピン番号,ピンモード,ピン変化 GoSub 行番号 On Pin ピン番号,ピンモード,ピン変化 GoSub "ラベル"	外部割込みイベント 指定したピン番号（2 または 3）のピン変化による割り込みにて GoTo、GoSub を実行可能にする タイマーイベントの制御には、Pin コマンドを利用
Pin 設定	外部割込みイベントの実施、停止を制御する
Sleep Sleep 休止時間	指定した時間、マイコンボードの活動を休止する 休止時間(ms)：0、16、32、64、125、250、500、1000、2000、4000、8000 0 の場合は無限に休止 休止時間省略時は 8000、 停止は外部割込みにより、キャンセル出来る

各制御命令の詳細については、「5 各制御文の詳細」を参照して下さい。

4.4 コマンド・関数

コマンド

コマンドとは何らかの処理を行うプログラム要素です。

コマンドラインからの直接利用とプログラム内での利用が可能です。

ただし、[Run](#) コマンドはプログラム中で利用した場合、エラー("Illegal command")となります。

例：コマンドラインで [Print](#) コマンドを実行

```
>print "こんにちは、世界！"
こんにちは、世界！
OK
>
```

例：プログラム内で [Print](#) コマンドを実行

```
>list
10 Print "こんにちは、世界！"
OK
>run
Print "こんにちは、世界！"
```

関数

関数とは何らかの値を返す命令文です。関数には数値関数と文字列関数の2種類があります。

数値関数は数値を返す関数です。文字列関数は文字列を返す関数です。

文字列関数は末尾に\$が付きます。

数値関数はコマンドの引数や式のなどの数値として利用します。

文字列関数は [Print](#)、[Vmsg](#) の引数にて利用できます。

例：関数の利用

```
>list
10 For I=Asc("A") To Asc("F")
20 Print "code=";Hex$(I,2)
30 Next
40 End
OK
>run
code=41
code=42
code=43
code=44
code=45
code=46
OK
>
```

上記のプログラムは文字“A”～“F”の文字コードを 16 進数へ表示するプログラムです。

10 行にて数値関数 [Asc\(\)](#) を利用しています。20 行で文字列関数 [Hex\\$\(\)](#) を利用しています。

関数は単独での利用は出来ません。コマンドや関数の引数、式中でのみ利用可能です。

例：関数の直接利用は文法エラーとなる

```
>asc("A")
Syntax error
OK
>?asc("A")
65
OK
```

4.5 コマンド・関数一覧

機能別に分類したコマンド及び関数の一覧を示します。

各コマンド及び関数の利用方法の詳細については、「7.各コマンド・関数の詳細」を参照して下さい。

□ システムコマンド

システムコマンドは、プログラムの編集や管理を行うコマンドです。

コマンド

Run	プログラムの実行 例: RUN
List List 開始行番号 List 開始行番号, 終了行番号	プログラムリストの表示 開始行番号: 表示を開始する行 終了行番号: 表示を終了する行 例: List List 200 List 200,300
Renum Renum 先頭行番号 Renum 先頭行番号, 間隔	行番号の振り直し 先頭行番号: 振り直し開始先頭行 間隔: 行間の増分(省略時は 10) 例: Renum Renum 100 Renum 100,10
Delete 行番号 Delete 先頭行番号, 末尾行番号	プログラムの指定行の削除 行番号: 削除対象行 先頭行番号, 末尾行番号: 削除対象行範囲 例: Delete 10 Delete 310,400
New	プログラムの消去と変数領域の初期化 例: NEW
Load Load プログラム番号 Load "ファイル名"	内部 EEPROM・外部 I2C EEPROM からプログラム読み込み プログラム番号: 読みだすプログラム(0~) ファイル名: 読みだすファイル(半角・全角 14 バイト迄) 例: Load Load 0 Load "sample01.bas"
Save Save プログラム番号 Save "ファイル名"	内部 EEPROM・外部 I2C EEPROM にプログラム保存 プログラム番号: 読みだすプログラム(0~) ファイル名: 読みだすファイル(半角・全角 14 バイト迄) 例: Save Save 0 Save "sample.01.bas"
Erase プログラム番号 Erase 開始プログラム番号, 終了プログラム番号 Erase "ファイル名"	内部 EEPROM・外部 I2C EEPROM の指定プログラム削除 プログラム番号: 削除するプログラム(0~) 開始プログラム番号, 終了プログラム番号: 削除するプログラム範囲 ファイル名: 削除するファイル(半角・全角 14 バイト迄) 例: Erase 0 Erase 0,1 Erase "sample01.bas"
Files Files プログラム番号 Files 開始, 終了 Files "" Files "ファイル名"	内部 EEPROM・外部 I2C EEPROM 内保存プログラム一覧表示 開始, 終了; プログラム番号: 内部 EEPROM 内プログラム番号 ファイル名: I2C EEPROM 内ファイル名(ワイルドカード?*OK) I2C EEPROM 上の指定プログラムファイル一覧表示 例: Files Files "" Files "s*0?.bas"
Format 容量 Format 容量, "ドライブ名"	I2C EEPROM の領域初期化 容量: I2C EEPROM の容量 4,8,16,32,64(k バイト単位) ドライブ名: 半角・全角 10 バイト迄 (省略時はブランク) 例: Format 32 Format 32, "ゲーム 1903"
Drive I2C スレーブアドレス Drive "ドライブ"	保存・読み込みで利用する I2C EEPROM の選択 デフォルトは\$50 (I2C EEPROM のデフォルトアドレス) I2C スレーブアドレス: 7 ビット I2C アドレス(1~127) ドライブ: A~Z の英字 (A~Z は\$50~ \$50+25 に対応) 例: Drive \$51 Drive "B"

□ 基本コマンド・関数

コマンド

Rem [コメント文] '[コメント文]	コメント 省略形の' (シングルクォーテーション)も可能 コメント文: 任意の文字列 例: <code>Rem サンプル</code> <code>'サンプル</code>
Let 変数=式 Let @(添え字)=n1[,n2,n3,n4…]	変数・配列変数に値を代入(LETは省略可能) 配列変数では連続代入指定が可能 例: <code>Let A=5</code> <code>A=5</code> <code>@(0)=1.2.3.4.5</code>
Print Print 文字列 値; Print 文字列 値; 文字列 値;… Print #桁数, 文字列 値; 文字列 値;… ;	画面への文字表示 文末";"付加で改行の抑制可能 省略形の?の利用可能 例: <code>Print "Hello,World"</code> <code>Print "座標";"(";X;",";Y;")"</code> <code>Print "16 進数:";HEX\$(V)</code>
Input 変数 Input 変数, オーバーフロー時の既定値 Input プロンプト, 変数 Input プロンプト, 変数, オーバーフロー時の既定値	数値の入力 例: <code>Input "A=",A,-1</code>
Cls	画面表示内容の全消去
Color 文字色 Color 文字色, 背景色	文字色の設定 ※ターミナル版でのみ利用可能
Attr 属性	文字表示属性設定 ※ターミナル版でのみ利用可能
Locate 横位置, 縦位置	カーソルの移動

数値関数

Free ()	プログラム領域の残りバイト数の取得 例: <code>?Free()</code>
Inkey ()	キー入力の読み取り 例: <code>?Inkey()</code>
Rnd ()	乱数の発生 例: <code>?Rnd(6)+1</code>
Abs (整数)	絶対値の取得
Asc (変数) Asc (変数, 文字位置) Asc (文字列定数) Asc (文字列定数, 文字位置)	変数が参照する文字列の取得/文字列の切り出し (全角 SJIS 対応) 例: <code>A=Asc("あ ABCD",2)</code> <code>S="あ ABC":B=AscS,2)</code>
Byte (変数) Byte (文字列定数)	文字列のバイト数取得 例: <code>L=Byte("ABCDE")</code>
Len (変数) Len (文字列定数)	文字列長の取得 (全角 SJIS 対応) 例: <code>L=Len"あ ABC いう"</code>
Map (値, 開始範囲 1, 終了範囲 1, 開始範囲 2, 終了範囲 2)	数値のスケール変換 例: <code>A=Map(ANA(14),0,99)</code>
Grade (値, 配列 No, 個数)	指定した値の等級を求める 例: <code>G=Grade(ANA(14),0,16)</code>

文字列関数

Chr\$ (文字コード[, 文字コード, ..., 文字コード])	文字コードから文字への変換 (全角 SJIS 対応) 例: <code>?Chr\$(82A0)</code>
Bin\$ (値[, 桁指定])	数値から 2 進数文字列への変換 例: <code>?Bin\$(1234,16)</code>
Hex\$ (値[, 桁指定])	数値から 16 進数文字列への変換 例: <code>?Hex\$(1234,4)</code>
Dmp\$ (値) Dmp\$ (値, 小数桁数) Dmp\$ (値, 小数桁数, 整数部桁数)	整数の小数付き数値文字列への変換 小数桁数 0~4 整数部桁数 0~8 <code>?Dmp\$(12345,3)</code>
Str\$ (変数) Str\$ (変数, 先頭, 長さ) Str\$ (文字列定数) Str\$ (文字列定数, 先頭, 長さ)	変数が参照する文字列の取得 (全角 SJIS 対応) 例: <code>?Str\$("あいう ABC",2,1)</code>

□ 時間待ち・時間計測関連

コマンド

Wait ミリ秒	時間待ち 例: WAIT(100)
--------------------------	----------------------

数値関数

Tick() Tick (モード)	起動からの経過時間取得 モード:0 ミリ秒単位、1 秒単位、省略時はミリ秒単位 例: ?TICK()
--	--

□ 記憶領域操作関連

コマンド

Poke アドレス,データ[,データ,..データ]	指定アドレスへのデータ書き込み 例: POKE MEM.1.2.3.4.5
---	--

数値関数

Peek (アドレス)	指定アドレスの値参照 例: ?PEEK(MEM)
-----------------------------	-----------------------------

□ VFD 表示制御関連

コマンド

Vcls	VFD の表示内容の全消去 例:
Vdisplay モード	VFD の点灯のオン・オフ設定 例: Vcls
Vbriht 輝度	VFD の輝度設定 例: Vbriht 10
Vmsg 速度,メッセージ文	VFD にメッセージ文の表示 例: Vmsg 100,"Hello,World"
Vscroll スクロール幅,速度	VFD に空白挿入表示 (スクロール) 例: Vscroll 32,100
Vput データ格納アドレス,データ長,速度	VFD にデータ直接出力 例: @(0)=\$FF,\$FF,\$FF,\$FF:Vput Array,4,5

□ サウンド関連

コマンド

Tone 周波数,出力期間	指定周波数のパルス出力 例: Tone 440,100
NoTone	パルス出力の停止 例: NoTone
Play "MML 文"	MML による単音音楽演奏 例: Play "04CDEFGAB05C"
Tempo テンポ	MML による単音音楽演奏のテンポの設定 (デフォルト 120) 例: Tempo 180

□ フォント関連

数値関数

GetFont (アドレス, シフト JIS コード)	シフト JIS コードに対応するフォントデータの取得 例: R=GetFont(Mem, Asc("あ"))
---	---

□ 汎用入出力関連

Arduino の入出力ポートを利用した、デジタル入出力、アナログ入力、I2C 利用のためのコマンド・関数です。

コマンド

Gpio ピン名 ピン番号,機能名	GPIO 機能設定 ピン名利用時は GPIO 記述省略可能 例: GPIO 13,OUTPUT
Out ピン番号,出力値	デジタル出力 例: OUT 13,HIGH
PWM ピン番号, デューティー値	PWM パルス出力 例: POUT 3,128
ShiftOut データピン,クロックピン, 出力形式,出力データ	デジタルシフトアウト出力 例: SHIFTOUT 2,4,MSB,123
LED 出力値	ボード上の LED の制御 例: LED On

数値関数

In (ピン番号)	デジタル入力
In (ピン番号, 入力モード)	入力モード: Float(デフォルト)、PullUp 例: A=IN(3,PullUp)
Ana (ピン番号)	アナログ入力 例: A=ANA(A0)
ShiftIn (データピン番号, クロックピン番号, 入力形式)	デジタルシフトイン入力
ShiftIn (データピン番号, クロックピン番号, 入力形式, 条件)	例: A=SHIFTIN(2,4,LSB,LOW)
PulseIn (パルス入力ピン番号, 検出信号, タイムアウト)	入力パルス幅の測定
PulseIn (パルス入力ピン番号, 検出信号, タイムアウト, スケール)	例: D=PULSEIN(3,HIGH,200,1)
I2c (デバイスアドレス, コマンドアドレス, コマンド長, データアドレス, データ長)	I2C スレーブデバイスからのデータ受信 例: R=I2CR(\$68, MEM, 4, MEM+4, 4)
I2cw (デバイスアドレス, コマンドアドレス, コマンド長, データアドレス, データ長)	I2C スレーブデバイスへのデータ送信 例: R=I2CW(\$68, MEM, 4, MEM+4, 4)

□ RTC (時刻) 関連

コマンド

DATE	現在時刻の表示 例: DATE
GETDATE 変数 1,変数 2,変数 3,変数 4	日付の取得 変数 1 から順に年,月,日,曜日コードを格納 例: GETDATE A,B,C,D
GETTIME 変数 1,変数 2,変数 3	時刻の取得 変数 1 から順に時,分,秒を格納 例: GETTIME A,B,C
SETDATE 年,月,日, 曜日コード,時,分,秒	時刻の設定 例: SETDATE 2018,3,1,5,12,0,0

文字列関数

DATES\$()	現在時刻文字列の取得 例: ?DATES\$()
---------------------------	-----------------------------

□ 有機 EL キャラクタディスプレイ SO1602AWWB 関連

CPRINT CPRINT 文字列値; CPRINT 文字列値; 文字列値;… CPRINT #桁数,文字列値; 文字列値;… ;	数値、文字列表示 例: CPRINT "V=":V: "[V]"
CCLS	表示内容の全消去 例: CCLS
CCURS 表示指定	カーソル表示・非表示設定 例: CCURS OFF
CLOCATE 横,縦	カーソルの移動 例: CLOCATE 0,1:CPRINT "Hello"
CCONS コントラスト値	コントラストの設定 例: CCONS 128
CDISP 表示指定	ディスプレイの表示・非表示設定 例: CDISP OFF

□ 赤外線リモコン受信

数値関数

IR (ピン番号)	NEC 方式赤外線リモコン受信 例: R=IR(3)
---------------------------	-------------------------------

IR(ピン番号,リポート指定)

システムコマンド、一般コマンドの詳細については「7 各コマンド・関数」を参照して下さい。

4.6 数値

□ 数値

「豊四季タイニーBASIC for Arduino 機能拡張版」で使える数値は整数型のみとなります。

整数型は 16 ビット幅、有効範囲は-32768～32767 となります。式、数値定数、数値関数、変数、配列変数はすべてこれに従います。

数値の表記は 10 進数、16 進数の表記可能です。

10 進数表記(-32768～32767) : -1, -32767, 100

16 進数表記(1 桁～4 桁) : \$1345, \$abcd, \$Abcd

(注意)

数値を中間コードに変換(文字列数値を 2 バイト型数値に変換)する際、オーバーフローが発生した場合は overflow エラーとなります。

例①:

```
>?32768
Overflow
```

評価・演算においてオーバーフローが発生した場合はエラーとはなりません。

例②:

```
>?32767+1
-32768
```

例①、例②は一見、同じような記述ですが、結果に差異が発生することをご理解下さい。

4.7 文字・文字列

□ 文字・文字列

「豊四季タイニーBASIC for Arduino 機能拡張版」では全角シフト JIS、半角カタカナ、半角英数字の文字列を扱うことが出来ます。

文字列の表記は次の通りです。

“文字列” : ダブルクォーテーションで囲みます。

文字列は 0～255 文字まで指定可能です。

例:

```
?Print "Hello こんにちは"
Hello こんにちは
OK
```

4.8 文字列利用における制約

「豊四季タイニーBASIC for Arduino 機能拡張版」では制約付きで文字列の利用をサポートしています。

変数への代入も可能です。

例:

```
10 A="ABCDE"
20 PRINT STR$(A)
>RUN
ABCDE
OK
>
```

変数への文字列代入は、文字列格納アドレスを変数に代入することにより実現しています。

C 言語のポインタによる参照方式と同等です。変数が参照している文字列の表示を行うには STR\$()関数を利用する必要があります。

ります。

また、文字列情報を[長さ+文字列]の形式で管理しているため、文字列の参照を代入した変数の値を変更して利用することは行えません。

下記のような利用は出来ません。行った場合、意図しない動作となります。

例：

```
10 A="ABCDE"
20 A=A+1
30 PRINT STR$(A)
RUN
```

また、コマンドラインでの利用は行えません。文字列参照が正しく行うことが出来ません。

例：

```
>10 a="ABCDE"
>20 a=a+1
>30 print str$(A)
>run
BCDE
衆 K
>
>暑
```

変数に代入した文字列の操作を行う次の関数を用意しています。

- [LEN\(\)](#) 文字列の長さの取得

```
10 A="ABCDE"
20 L=LEN(A)
30 ?L
RUN
5
OK
```

- [STR\\$\(\)](#) 変数で参照している文字列の取得、部分切り出し

```
10 A="ABCDE"
20 PRINT STR$(A)
30 PRINT STR$(A,4,1)
RUN
ABCDE
D
OK
```

- [ASC\(\)](#) 指定位置の文字コードの取得

```
10 A="ABCDE"
20 C=ASC(A,4,1)
RUN
68
OK
```

4.9 変数・配列変数

□ 変数

変数とは数値を格納する保存領域です。数値と同様に式に利用できます。

変数には、通常の変数の他に**配列変数**があります

変数に格納する数値は 16 ビット幅、有効範囲は-32768~32767 となります。

変数は数値型のみ利用可能ですが、文字列の格納アドレスを代入することにより、文字列の参照を行うことが出来ます。

変数の表記

通常の変数：変数名は英字 A~Z の 1 文字の利用が可能です。

例:

```
10 A=200
20 B=300
30 C=A+B
40 S="Hello"
```

配列変数：@(添え字)の形式で添え字は数値で 0~31 (@0)~@(31)まで利用可能

Arduino MEGA2560 の場合は、添え字は 0~99(@0)~@(99)まで利用可能

添え字の数値には式、変数等の利用が可能

例:

```
10 A=5
20 @(0)=30/5,
30 @(A+1)=5
```

代入式において、指定した添え字を起点として連続代入が可能

例:

```
10 @(10)=100,200,300,400,500
```

4.10 演算子

□ 演算子

数値演算で利用できる演算子を示します。

記述例の A,B には数値、変数、配列変数、関数、カッコで囲んだ式が利用できます。

算術演算子

演算子	説明	記述例
+	足し算	A+B A と B を足す
-	引き算	A-B A から B を引く
*	掛け算	A*B A と B の積
/	割り算	A/B A を B で割る
%	剰余算	A%B A を B で割った余り

ビット演算子

演算子	説明	記述例
&	ビット毎の AND 演算	A&B A と B のビット毎の AND
	ビット毎の OR 演算	A B A と B のビット毎の OR
>>	ビット右シフト演算	A>>B A を右に B ビットシフト
<<	ビット左シフト演算	A<<B A を左に B ビットシフト
~	ビット毎の反転	~A A の各ビットを反転
^	ビット毎の XOR	A^B A と B の XOR

比較演算、論理演算の論理反転は、0 が偽、0 以外が真となります。

0 以外が真となりますので、1、-1 はとも真となります。

比較演算子

演算子	説明	記述例
=	等しいかを判定	A=B A と B が等しければ真, 異なれば偽
!= <>	異なるかを判定	A!=B A と B が異なれば真, 等しければ偽
<	小さいかを判定	A<B A が B 未満であれば真, そうでなければ偽
<=	小さいまたは等しいかを判定	A<=B A が B 以下であれば真, そうでなければ偽
>	大きいかを判定	A>B A が B より大きければ真, そうでなければ偽
>=	大きいまたは等しいかを判定	A>=B A が B 以上であれば真, そうでなければ偽

論理演算子

演算子	説明	記述例
AND	論理積	A AND B A, B が真なら真, でなければ偽
OR	論理和	A OR B A, B どちらかが真なら真, でなければ偽
!	否定	!A A が真なら偽, 偽なら真

□ 演算子の優先順序

演算子の優先度を下記に示します。優先度の数値が小さいほど優先度が高くなります。

計算結果が意図した結果にならない場合は、優先度の確認、括弧をつけて優先度を上げる等の対応を行って下さい。

演算子の優先度

優先度	演算子
1	括弧で囲った式
2	!, ~
3	*, / , % , &, , << , >>, ^
4	+, -, ,
5	=, <> , != , > , >= , < , <= , AND , OR

比較演算子と論理演算は優先度が同レベルです。比較演算子の演算を先に行う場合は、括弧を使って優先度を上げて下さい。

例：

```
?1<5 and 2>3
0
?(1<5) and (2>3)
1
```

4.11 式

□ 式

式とは 1、1+1、A、A+1、ABS(-1) などの演算・値の評価を伴う記述をいいます。

式はプログラム実行にて計算・評価が行われ、1つの整数値の値として振る舞います。

変数への値の代入、コマンド、条件判定(IF 文)、関数に引数に式が用いられた場合は、評価後の値がコマンドおよび関数に渡されます。

式の利用：

変数への値の代入(代入命令 LET は省略可能)

```
LET A=(1+10)*10/2
A=5*5+3*2
@(I+J) = I*J
```

コマンド・関数の引数

```
LOCATE X+I*2, J:PRINT "*"
OUT 13, I>J
A=EEPREAD(I+J+1)/2
```

4.12 定数

「豊四季タイニー-BASIC for Arduino 機能拡張版」では次の定数が利用出来ます。

定数はコマンドの引数や式の中で数値関数と同等に利用出来ます。

□ 1ビット入出力値 High、Low

High は 1、Low は 0 が割り当てられています。

High : 値 1

Low : 値 0

利用例：ピン番号 2 の入力を判定する

```
10 I=In(2,PullUp)
20 If I=High ?"High" Else ?"Low"
30 Wait 500
40 GoTo 10
```

□ On、Off 定数

On は 1、Off は 0 が割り当てられています。

On : 値 1

Off : 値 0

利用例：ピン番号 2 の入力を判定する

```
10 I=In(2,PullUp)
20 If I=High ?"High" Else ?"Low"
30 Wait 500
40 GoTo 10
```

□ 仮想メモリ領域先頭アドレス定数

VRAM, VAR, ARRAY, PRG, MEM, FNT, GRAM

データ領域を参照するための定数です。各定数の詳細は次の通りです。

定数名	値	領域サイズ	用途
VAR	\$1900	52	変数領域 (A~Z)
ARRAY	\$1AA0	64 (MEGA2560 は 200)	配列変数領域 (@(0)~@(32)) (MEGA2560 は @(0)~@(99))
PRG	\$1BA0	1024 (MEGA2560 は 2048)	プログラム領域
MEM	\$2BA0	64	ユーザーワーク領域
MEM2	\$2CA0	64	ユーザーワーク領域 2

定数値のアドレスは仮想的なアドレスです。実アドレスとは異なります。

PEEK、POKE、I2CW、I2CR のメモリ領域を利用するコマンドで利用できます。

定数の利用例：仮想メモリ領域先頭アドレス定数を表示する

```
10 '仮想アドレス
20 Print "仮想アドレス定数の値"
30 Print "VAR ",Hex$(Var,4)
40 Print "ARRAY:",Hex$(Array,4)
50 Print "PRG :",Hex$(Prg,4)
60 Print "MEM :",Hex$(Mem,4)
70 Print "MEM2 :",Hex$(Mem2,4)
```

実行結果

```
>run
仮想アドレス定数の値
VAR 1900
ARRAY:1AA0
PRG :1BA0
MEM :2BA0
MEM2 :2CA0
OK
>
```

□ 画面表示の定数

画面サイズを参照する定数です。

- CW : キャラクタ画面の横文字数 (デフォルト 80)
CH : キャラクタ画面の縦行数 (デフォルト 24)

利用例

```
10 FOR Y=0 TO CH-2
20 FOR X=0 TO CW-1
30 LOCATE X,Y:COLOR RND(8)+1:?"*"
40 NEXT X
50 NEXT Y
```

□ ピン番号定数

デジタル入出力ピンは番号 0~21 の整数で指定することが出来ます。

※Arduino MEGA2560 の場合、0~69

アナログ入力ピン A0~A7 (ピン番号 14~21) は、定数 A0~A7 にて指定することが出来ます。

A0:14、A1:15、A2:16、A3:17、A4:18、A5:19、A6:20、A7:21

※Arduino MEGA2560 の場合、アナログ入力ピンは A0~A15 (ピン番号 54~69) は定数 A0~A15 で指定可能

利用例

```
10 A=ANA(A0)
20 ?DMP$(MAP(A,0,1023,0,5000),3)
30 WAIT 200
40 GOTO 10
```

□ GPIO ピンモード設定定数

GPIO コマンドのモード設定を行うための定数です。

OUTPUT, PULLUP, FLOAT

利用例

```
10 A=IN(2,PULLUP)
20 IF A=OFF ?"ON"
30 WAIT 200
40 GOTO 10
```

□ ビット方向定数

LSB, MSB

ビット送信等で上位、下位を指定するための定数です。

SHIFTIN, SHFITOUT コマンドで利用します。

□ キー入力コード定数

[INKEY\(\)](#)関数によるキーボードからの入力判定等に利用するための定数です。

定数名	値
Entrer	: 13
Up	: 28
Down	: 29
Right	: 30
Left	: 31
Space	: 32

5. 各制御文の詳細

5.1 GoTo 指定行へのジャンプ（制御命令）

□ 書式

Goto 行番号
Goto "ラベル"

□ 説明

プログラムの実行を行番号、“ラベル”で指定した行にジャンプします。
行番号には数値、式（変数、関数を含む）が利用可能です。
ラベルを指定した場合、行先頭に同じラベルがある行にジャンプします。

ラベルはダブルクォーテーション(")で囲って指定します。

```
10 I=0
20 "ループ"
30 Print "@"
40 I=I+1:If I=5 Goto 60
50 GoTo "ループ"
60 End
```

上記の例では、40 行の GoTo で 60 行にジャンプ、50 行のラベル指定で 20 行にジャンプしています。
ラベルを使うとプログラムの流れがわかりやすくなります。ラベルには日本語（全角文字）も利用出来ます。

行の先頭に無いラベルは、ジャンプ先としての利用は出来ません。エラーとなります。

```
10 GoTo "hoge"
20 End
30 ? "Hello":"hoge":?"test"
>run
Undefined line number or label in 10
10 GoTo "hoge"
OK
>
```

行番号に式・変数の指定が可能ですが、[Renum](#) コマンドによる行番号付け替えに対応出来ない場合があります。

- ① GoTo N*100+500
- ② GoTo 500+N*100+500

①では [Renum](#) コマンドは何もしません。②は 500 を番号とみなして更新します。
ただし①②とも計算結果が正しい行番号となることは保証出来ません。ご注意ください。

□ エラーメッセージ

Undefined line number or label	: 指定した行、ラベルが存在しない
Overflow	: 指定した数値が -32768 ~ 32767 を超えている
Syntax error	: 文法エラー、書式と異なる利用を行った

□ 利用例

ラベルを使ったループ処理

```
100 GoSub "SUB01"
110 Gosub "SUB02"
120 N=0
130 "LOOP"
140 Print "N=";N
150 N=N+1:If N<5 GoTo "LOOP"
160 End
170 "SUB01"
180 Print "SUB01"
190 Return
200 "SUB02"
210 Print "SUB02"
220 Return
```


5.2 GoSub サブルーチンの呼び出し（制御命令）

□ 書式

GoSub 行番号

GoSub "ラベル"

□ 説明

プログラムの実行を一旦、行番号またはラベルで指定した行に移ります。

移った先で [Return](#) 命令により、呼び出した GoSub 令の次の位置に戻ります。

行番号には数値、式（変数、関数を含む）が利用できます。

ラベルを指定した場合は、行先頭に同じラベルがある行に移ります。

ラベルはダブルクォーテーション(“)で囲って指定します。

```
10 GoSub "ロゴ表示"
20 GoSub "バージョン表示"
30 GoSub 200
40 End
50 "ロゴ表示"
60 Print "Tiny BASIC for Arduino"
70 Return
80 "バージョン表示"
90 Print "Edition V0.07 2019/09/11"
100 Return
200 Print "Ready"
210 Return
```

上記の例では、10 行、20 行でラベルを指定してサブルーチンを呼び出しています。

30 行では、行番号を指定してサブルーチンを呼び出しています。

ラベルを使うとプログラムの流れがわかりやすくなります。ラベルには日本語（全角文字）も利用出来ます。

行の先頭に無いラベルは、GoSub 文の呼び出し先としての参照はされません。エラーとなります。

```
10 GoSub "hoge"
20 End
30 ? "Hello":"hoge":?"test":Return
run
Undefined line number or label in 10
10 GoSub "hoge"
OK
```

行番号には式や変数の指定が可能ですが、[Return](#) マンドによる行番号付け替えに対応出来ない場合があります。

①GoSub N*100+500

②GoSub 500+N*100+500

①では [Return](#) コマンドは何もしません。②は 500 を行番号とみなして更新します。

ただし①②とも計算結果が正しい行番号となることは保証出来ません。ご注意ください。

□ エラーメッセージ

Undefined line number or label	: 指定した行、ラベルが存在しない
Overflow	: 指定した数値が -32768 ~ 32767 を超えている
GOSUB too many nested	: GoSub のネスト数が規定(=10)を超えた
Syntax error	: 文法エラー、書式と異なる利用を行った

□ 利用例

ラベルを使ったサブルーチンの呼び出し例

```
100 GoSub "SUB01"
110 GoSub "SUB02"
120 N=0
130 "LOOP"
140 Print "N=";N
150 N=N+1:If N<5 GoTo "LOOP"
160 End
170 "SUB01":Print "SUB01":Return
200 "SUB02":Print "SUB02":Return
```

5.3 Return GoSub 呼び出し元への復帰（制御命令）

□ 書式

Return

□ 説明

直前に呼び出された GOSUB の次の処理に復帰します。

詳細は「5.2GoSub サブルーチンの呼び出し（制御命令）」を参照して下さい。

□ エラーメッセージ

Undefined line number or label	: 指定した行、ラベルが存在しない
Overflow	: 指定した数値が-32768 ～ 32767 を超えている
RETURN stack underflow	: GoSub の呼び出しがないのに Return を実行
Syntax error	: 文法エラー、書式と異なる利用を行った

□ 利用例

ラベルを使ったサブルーチンの呼び出し例

```

100 GoSub "SUB01"
110 GoSub "SUB02"
120 N=0
130 "LOOP"
140 Print "N=";N
150 N=N+1:If N<5 GoTo "LOOP"
160 End
170 "SUB01"
180 Print "SUB01"
190 Return
200 "SUB02"
210 Print "SUB02"
220 Return

```

5.4 If 条件判定（制御命令）

□ 書式

If 条件式 「真の場合の実行文」

If 条件式 「真の場合の実行文」 Else 「偽の場合の実行文」

□ 説明

条件式の真偽判定を行い、真の場合は「真の場合の実行文」を実行します。

真の場合に Else がある場合、Else 以降の「真の場合の実行文」は実行しません。

偽の場合、Else がある場合、「真の場合の実行文」を実行します。なければ次の行にスキップします。

真偽は次の条件となります。

真：評価した値が 0 以外である

偽：評価した値が 0 である

真偽判定を行う条件式には定数、数値、関数を含む計算式が記述できます。

式の例：

```
If A > 5 B=B-1
If A+1 Print A
If A & 1 Tone 440,200
If Inkey() = ASC("A") GoTo 100
```

定数の例

```
If High GoTo 100
```

数値の例

```
If 123 Print "123"
```

関数の例

```
If Rnd(1) X=X+1
```

実行文には If、GoTo、GoSub 等の制御命令を使うことも可能です。

（注意）Else 利用の制約(ぶら下がり ELSE に関する補足)

If 文をネストして利用した場合、Else 文は直前の If 文に対応します。

例：If A=1 If B=1 ? "A,B=1" Else ? "A=1,B<>1"

上記の Else は 2 番目の If 文に対応します。

この「Else 文は直前の If 文に対応」の条件で、Else 文に If 文をネスト出来ます。

例: If A=1 If B=1 ? "A,B=1" Else If B=2 ? "A=1,B=2" Else If B=3 ? "A=1,B=3"

上記では、A=1,B=1,2,3 の条件に対して対応する表示メッセージを表示する例です。

□ エラーメッセージ

IF without condition : 条件式が定義されていない

Overflow : 指定した数値が -32768 ～ 32767 を超えている

□ 利用例

y を押したら、“Y key” を表示して終了、その他のキーなら“End” を表示して終了する

何も押されていない場合は 10 行から 30 行をループする。

```
10 A=Inkey()
20 If A=Asc("y") Print "Y key":End Else If A<>0 Print "End":End
30 GoTo 10
```

5.5 For To ～ NEXT 繰り返し実行（制御命令）

□ 書式

For 変数=初期値 To 最終値 実行文(複数行可能) Next [変数]

For 変数=初期値 To 最終値 Step 増分 繰り返し実行文(複数行可能) Next [変数]

※Next の変数は省略可能

□ 説明

変数を初期値から最終値まで増やし、For から Next の間の実行文を繰り返し実行します。

変数が最終値に達した時点で繰り返しを止め、Next の次の命令の実行を行います。

Step にて増分を指定しない場合、増分は 1 となります。

Step を用いた場合は、マイナス値を含め任意の増分の指定が可能です。

例：

```
10 Print "start."
20 For I=0 To 5 Step 2
30 Print I
40 Next I
50 Print "done."
```

実行結果

```
start
0
2
4
done.
ok
```

上記の例では I を 0 から 5 まで、2 ずつ増加して 30 行の命令を繰り返し実行します。

I が 4 の時、増分 2 を足すと終了値 5 を超えた 6 となるので繰り返しを終了し、50 行を実行します。

For はネストも可能です(利用例を参照)。

For～Next の繰り返しは、利用している変数が繰り返し条件を満たさなくなった時点で繰り返しを終了します。

最初の例に“35 I=6”を追加し、ループ内で条件を満たさなくすることでループを抜けることができます。

```
10 Print "start."
20 For I=0 To 5 Step 2
30 Print I
35 I=6
40 Next I
50 Print "done."
```

実行結果

```
start
0
done.
ok
```

□ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
FOR without variable	: FOR 文で変数を指定していない
FOR without TO	: FOR 文で TO を指定していない
NEXT without counter	: NEXT に対応する FOR 文が無い
FOR too many nested	: FOR 文のネスト数が規定数(=10)を超えた
Overflow	: 指定した数値が -32768 ～ 32767 を超えている

□ 利用例

画面の指定位置に*を表示する（For 文のネストの例）

```
10 Cls
20 For Y=5 To 10
30 For X=5 To 10
40 Locate X,Y:Print "*"
50 Next
60 Next
```

5.6 End プログラムの終了（制御命令）

□ 書式

End

□ 説明

プログラムの実行を終了します。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

□ 利用例

特定の条件でプログラムを終了する

```
10 Input "I=",I
20 Print I
30 If I=5 End
40 GoTo 10
```

5.7 On Timer タイマーイベント（制御命令）

□ 書式

On Timer 周期(ミリ秒) GoTo 行番号
 On Timer 周期(ミリ秒) GoTo “ラベル”
 On Timer 周期(ミリ秒) GoSub 行番号
 On Timer 周期(ミリ秒) GoSub “ラベル”

□ 引数

ミリ秒（周期） : タイマーイベントの周期
 On | Off : タイマーイベント実行指定（デフォルトは OFF）

□ 説明

タイマーイベントに割り込み発生時の処理を定義します。
 指定した周期で [GoTo](#) 文、[GoSub](#) 文を実行します。
[GoSub](#) 文を指定した場合、現在のコマンド実行位置で [GoSub](#) で指定したサブルーチンを実行し、サブルーチンの [Return](#) コマンドにて、元の位置に戻ります。

本 On Timer 命令では、周期と挙動の定義のみを行います。
 タイマーイベントを有効にするためには、[Timer](#) コマンドにてタイマーイベントの有効と無効を指定します。
 デフォルトでは、タイマーイベントは無効（Off）となっています。

□ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
Overflow	: 指定した数値が -32768 ~ 32767 を超えている
Illegal value	: 振り直しをする新しい行番号が有効範囲を超えている
Undefined line number or label	: 指定した行、ラベルが存在しない

（注意）割り込み処理は、個々のコマンド実行が終了してから実行されます。
 例えば Wait 1000 実行中にタイマーイベントが発生した場合、Wait コマンドの実行完了後に割り込み処理が実行されます。

□ 利用例

0.5 秒間隔でボード上 LED を点灯させる

```
10 On Timer 500 GoSub "BLINK"
20 Timer On
30 "LOOP":GoTo "LOOP"
40 "BLINK":LED D
50 If D ?"Blink!"
60 D=!D
70 Return
```

上記のプログラムは、
 10 行で 500 ミリ秒間隔でサブルーチン 40 行から 70 行の処理の呼び出しを定義しています。
 20 行でタイマーイベントを有効化しています。
 30 行は、この行で無限ループをします。
 40 行から 70 行は LED の点滅を行います。10 行の定義で 500 ミリ秒間隔で呼び出されます。

5.1 Timer タイマーイベント設定（制御命令）

□ 書式

Timer On | Off

□ 引数

On | Off : タイマーイベント実行指定（デフォルトは Off）

□ 説明

タイマーイベントの有効と無効を指定します。

デフォルトでは、タイマーイベントは無効（Off）となっています。

タイマーイベントを利用するには、事前に On Timer 命令で周期とアクションの指定を行う必要があります。

□ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
Overflow	: 指定した数値が-32768 ~ 32767 を超えている
Illegal value	: 振り直しをする新しい行番号が有効範囲を超えている

□ 利用例

0.5 秒間隔でボード上 LED を点灯させる

```
10 On Timer 500 GoSub "BLINK"
20 Timer On
30 "LOOP":GoTo "LOOP"
40 "BLINK":LED D
50 If D ?"Blink!"
60 D=!D
70 Return
```

上記のプログラムは、

10 行で 500 ミリ秒間隔でサブルーチン 40 行から 70 行の処理の呼び出しを定義しています。

20 行でタイマーイベントを有効化しています。

30 行は、この行で無限ループをします。

40 行から 70 行は LED の点滅を行います。10 行の定義で 500 ミリ秒間隔で呼び出されます。

5.2 On Pin 外部割込みイベント（制御命令）

□ 書式

On Pin ピン番号,ピンモード,ピン変化 GoTo 行番号
 On Pin ピン番号,ピンモード,ピン変化 GoTo "ラベル"
 On Pin ピン番号,ピンモード,ピン変化 GoSub 行番号
 On Pin ピン番号,ピンモード,ピン変化 GoSub "ラベル"

□ 引数

ピン番号 : 2、3
 ピンモード : PullUp、Float
 ピン変化 : Low、Change、Falling、Rising

□ 説明

外部割込みイベントに割り込み発生時の処理を定義します。

外部割込みイベントを検出するピンは D2、D3 ピンのみ指定可能です。

外部割込みは、Sleep コマンドにてボードが休止状態となっている場合の復旧（ウェイク）として利用することが出来ます。

ピンモードは、指定したピンのデジタル入力の設定として、

PullUp : プルアップ
 Float : フロート（プルアップ無し、Arduino pinMode()の INPUT と同等）

のいずれかを指定します。

ピン変化は、割り込みとして検知するデジタル入力の条件を指定します。

Low : 入力が Low(0V)の場合に割り込みを発生
 Change : 入力に変化した場合に割り込みを発生
 Falling : 入力が High から Low に変化した場合に割り込みを発生
 Rising : 入力が Low から High に変化した場合に割り込みを発生

割り込み発生した場合は、[GoTo](#) 文、[GoSub](#) 文を実行します。

[GoSub](#) 文を指定した場合、現在のコマンド実行位置で [GoSub](#) で指定したサブルーチンを実行し、サブルーチンの [Return](#) コマンドにて、元の位置に戻ります。

本 On Pin 命令では、外部割込みに対する定義のみを行います。

外部割り込みイベントを有効にするためには、Pin コマンドにて外部割込みを有効設定する必要があります。

デフォルトでは、外部割込みイベントは無効（Off）となっています。

（注意）外部割込みイベントが発生すると、外部割り込みイベントの有効設定は初期化（無効化）されます。
 連続してイベントを受け付けるにはその都度、Pin コマンドで外部割込みを有効設定する必要があります。

（注意）割り込み処理は、個々のコマンド実行が終了してから実行されます。
 例えば Wait 1000 実行中にタイマーイベントが発生した場合、Wait コマンドの実行完了後に割り込み処理が実行されます。

□ エラーメッセージ

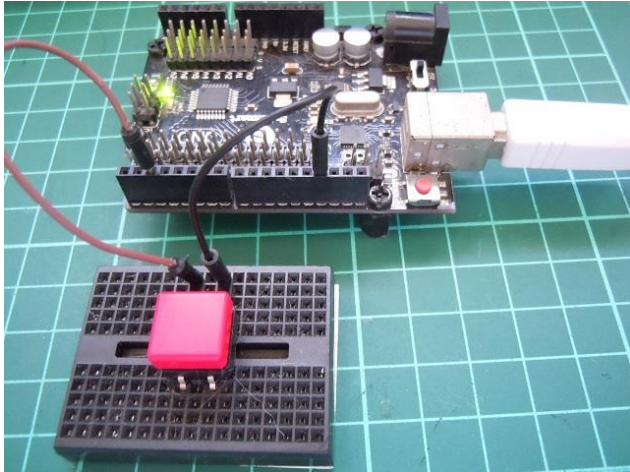
Syntax error : 文法エラー、書式と異なる利用を行った
 Overflow : 指定した数値が -32768 ～ 32767 を超えている
 Illegal value : 振り直しをする新しい行番号が有効範囲を超えている
 Undefined line number or label : 指定した行、ラベルが存在しない

□ 利用例

① 外部割り込みイベントの発生でボード上のLEDを点滅させる

```
>list
10 On Pin 2,PullUp,Rising GoSub "割り込み"
20 Pin 2,On
40 GoTo 40
50 "割り込み"
70 LED On:Wait 50:LED Off
80 Pin 2,On
90 Return
```

実装の様子



D2 ピンにボタンを接続（D2、GND に接続）し、ボタンを押すと D2 に Low（0V）が入力されます。ボタンを押さない状態では、D2 はプルアップにより High の入力となります。

20 行で D2 ピンによる外部割り込みを定義しています。

プルアップ指定し、入力が LOW から HIGH に変化した場合に、GosSub にてラベル”割り込み”の 50 行～90 行のサブルーチン呼び出しを行います。

40 行は外部割り込み待ちを行っています。50 行～90 行は、外部割り込みイベントにて実行されるサブルーチンです。

LED を 50 ミリ秒点灯後、消灯します。80 行では再度、外部割り込みイベントを受け付けるために Pin コマンドで割り込みを有効化しています。

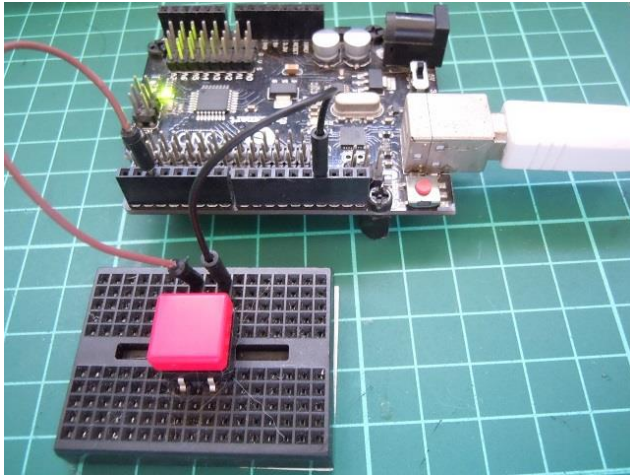
② 外部割込みにより、Sleep コマンドによる休止状態から抜ける

```

1 '外部割込みによるスリープ抜け
10 D=0
20 On Pin 2,PullUp,Rising GoSub "ext"
30 Pin 2,On
40 ?"スリープ"
50 Sleep 0
60 ?"スリープ抜け"
70 Input "継続? ",C
80 D=!D
90 LED D
100 Wait 500
110 GoTo 30
120 "ext"
130 Return

```

実装の様子



D2 ピンにボタンを接続（D2、GND に接続）し、ボタンを押すと D2 に Low（0V）が入力されます。ボタンを押さない状態では、D2 はプルアップにより High の入力となります。

20 行で D2 ピンによる外部割込みを定義しています。

プルアップ指定し、入力が LOW から HIGH に変化した場合に、Gosub にてラベル”ext”の 120 行～130 行のサブルーチン呼び出しを行います。このサブルーチンはスリープ抜け用のため、処理としては何も行いません。

30 行は、D2 ピンからの外部割込みを有効にしています。

50 行は、Sleep コマンドにてボードを休止状態にします。引数として 0 を指定しているため、スリープ時間は無期限となります。

60 行～110 行はスリープを抜けた後に実行する処理です。

外部イベントによるスリープ抜け後は、外部イベントを無効となっています。

Input コマンドで入力による継続待ちを行っています。入力後は、30 行にジャンプし、Sin コマンドで再度、外部割込みを有効化しています。50 行で再び、スリープ状態となります。

5.3 Pin 外部割込みイベント設定（制御命令）

□ 書式

Pin ピン番号,On | Off

□ 引数

ピン番号 : 対象ピン番号 2,3

On | Off : 外部割込みイベント有効・設定指定（デフォルトは Off）

□ 説明

指定したピン番号の外部割込みイベントの有効、無効を設定します。

デフォルトは、無効（Off）となっています。

外部割込みイベントを利用するには、事前に On Pin 命令で外部割込みイベントの定義が必要です。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Overflow : 指定した数値が-32768 ~ 32767 を超えている

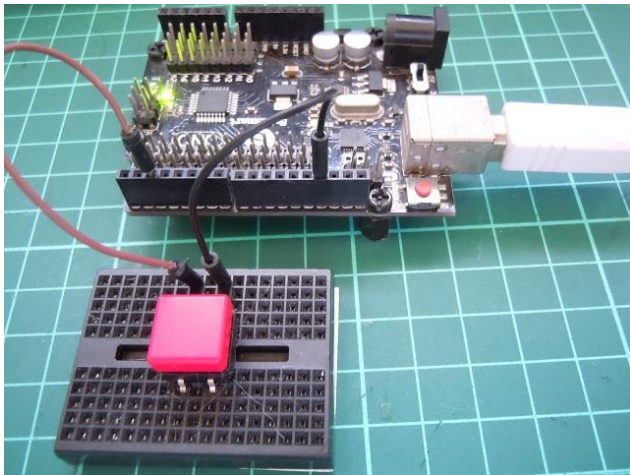
Illegal value : 振り直しをする新しい行番号が有効範囲を超えている

□ 利用例

外部割込みイベントの発生でボード上の LED を点滅させる

```
>list
10 On Pin 2,PullUp,Rising GoSub "割込み"
20 Pin 2,On
40 GoTo 40
50 "割込み"
70 LED On:Wait 50:LED Off
80 Pin 2,On
90 Return
```

実装の様子



D2 ピンにボタンを接続（D2、GND に接続）し、ボタンを押すと D2 に Low（0V）が入力されます。ボタンを押さない状態では、D2 はプルアップにより High の入力となります。

20 行で D2 ピンによる外部割込みを定義しています。

プルアップ指定し、入力が LOW から HIGH に変化した場合に、GosSub にてラベル”割込み”の 50 行～90 行のサブルーチン呼び出しを行います。

40 行は外部割込み待ちを行っています。50 行～90 行は、外部割込みイベントにて実行されるサブルーチンです。

LED を 50 ミリ秒点灯後、消灯します。80 行では再度、外部割込みイベントを受け付けるために Pin コマンドで割り込みを有効化しています。

5.4 Sleep 休止（制御命令）

□ 書式

Sleep

Sleep 休止時間

□ 引数

休止時間 (ms) : 0、16、32、64、125、250、500、1000、2000、4000、8000

0 の場合は無限休止、省略時は 8000

上記以外の値を指定した場合、近い下位の値を休止時間として採用する

□ 説明

Arduino を指定時間、休止状態にします。

休止状態では、キーボードの[ESC]キー、[CTRL]+[C]キーによるプログラム終了も受け付けませんが、外部割込みイベントにて復帰することが出来ます。

（注意）引数 0 を指定した場合、無限休止となります。この場合、外部割込みイベントでのみ休止状態から復帰することが出来ます。外部割込みイベントの設定を行っていない場合はリセットにて復帰して下さい。

□ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
Overflow	: 指定した数値が-32768 ~ 32767 を超えている
Illegal value	: 振り直しをする新しい行番号が有効範囲を超えている

□ 利用例

0.5 秒間隔でボード上 LED を点灯させる

```
10 LED On
20 Sleep 500
30 LED Off
40 Sleep 500
50 GoTo 10
```

上記のプログラムは、20 行、40 行で 500 ミリ秒（0.5 秒）ボードを休止することで LED 点滅のタイミングをとっています。

6. 各コマンド・関数の詳細

6.1 Run プログラムの実行

□ 書式

Run

□ 引数

なし

□ 説明

プログラムを実行します。プログラム実行中は、カーソルが非表示となります。

実行中のプログラムは次の条件で終了します。

- 1) 全てのプログラムを実行し終わった。
- 2) [Eed](#) コマンドで実行を終了した
- 3) [ESC]キー（2回押し）、[CTRL+C]キー、[F4]キーで強制終了した
- 4) プログラムエラーで終了した

(注意) Wait コマンドで長い時間待ちを行っている場合は、時間待ち終了まで[ESC]キー、[CTRL+C]キーによる強制終了を行うことが出来ません。
Sleep コマンドでスリープモード中は、キーボード等による強制終了を行うことが出来ません。

Run コマンドは、話形式でのみ利用可能です。

プログラム内で実行した場合は、エラー：Illegal command となります。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った、引数に数値以外を指定した
Illegal command : プログラム内で Run コマンドを実行しようとした

□ 利用例

プログラムを実行する

```
>list
10 For I=1 To 10
20 Print "*";
30 Next
40 Print
50 End
OK
>run
*****
OK
>
```

実行中プログラムを[ESC]キーで強制終了する

```
>list
10 Print "無限ループ中"
20 GoTo 20
OK
>run
無限ループ中
Break in 20
20 GoTo 20
OK
>
```

6.2 List プログラムリストの表示

□ 書式

List

List 表示開始行番号

List 表示開始行番号, 表示終了行番号

□ 引数

表示開始行番号 : 表示を開始する行番号 (0 ~ 32767)

表示終了行番号 : 表示を終了する行番号 (0 ~ 32767)

□ 説明

プログラムリストを表示します。

引数を指定しない場合、全てのプログラムを表示します。

表示開始番号のみ指定した場合、その番号以降のプログラムリストを表示します。

表示開始番号、表示終了番号を指定した場合、その範囲のプログラムリストを表示します。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った、引数に数値以外を指定した

Illegal value : 指定した引数の値が有効範囲でない

Overflow : 指定した数値が-32768 ~ 32767 を超えている

□ 利用例

```
>list
10 I=0
20 Print "*";
30 I=I+1:If I<10 GoTo 20
40 Print
50 End
OK
>list 20
20 Print "*";
30 I=I+1:If I<10 GoTo 20
40 Print
50 End
OK
>list 20,30
20 Print "*";
30 I=I+1:If I<10 GoTo 20
OK
>
```

6.3 Renum 行番号の振り直し

□ 書式

Renum

Renum 開始行番号

Renum 開始行番号,増分

□ 引数

開始番号 : 振り直しをする新しい行番号の開始番号 (1 ~ 32767)

増分 : 行番号の増分 (1 ~ 32767)

□ 説明

プログラムの行番号を指定した条件で振り直します。

引数を省略した場合は、行番号を 10 行から 10 間隔で振り直します。

開始番号のみ指定した場合、指定した開始番号から 10 間隔で振り直します。

開始番号と増分を指定した場合、指定した開始番号から指定した増分で振り直します。

振り直しにおいて、[GoTo](#) 文、[GoSub](#) 文のとび先の行番号も更新されます。

(注意) GOTO,GOSUB に存在しない行番号を指定している場合、更新は行われません。
また、行番号に計算式を利用している場合、正しい更新が行われない場合があります。
GOTO 100+N*10
の記載の場合、先頭の数値 100 を行番号とみなして更新します。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った、引数に数値以外を指定した

Illegal value : 振り直しをする新しい行番号が有効範囲を超えている

Overflow : 指定した引数または振り直し行番号が -32768 ~ 32767 を超えている

□ 利用例

行番号を 100 から 10 間隔で振り直す

```
>list
10 I=0
20 Print "*";
30 I=I+1:If I<10 GoTo 20
40 Print
50 End
OK
>renum 100,10
OK
>list
100 I=0
110 Print "*";
120 I=I+1:If I<10 GoTo 110
130 Print
140 End
OK
>
```

6.4 Delete プログラムの指定行の削除

□ 書式

Delete 行番号

Delete 先頭行番号, 末尾行番号

□ 引数

行番号 : 削除対象の行番号 1～32767

先頭番号 : 削除対象範囲の先頭番号 1～32767

末尾番号 : 削除対象範囲の末尾番号 1～32767

□ 説明

プログラムの指定した行、指定した範囲（先頭行番号、末尾行番号）の行を削除します。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った、引数に数値以外を指定した

Overflow : 指定した数値が-32768 ～ 32767 を超えている

Illegal value : 先頭番号と末尾番号の指定に矛盾がある

□ 利用例

プログラム内の 20 行から 50 行の範囲を削除する

```

>list
10 ?"AAAAAA"
20 ?"BBBBBB"
30 ?"CCCCCC"
40 ?"DDDDDD"
50 ?"EEEEEE"
60 ?"FFFFFF"
70 ?"GGGGGG"
OK
>delete 20,50
OK
>list
10 ?"AAAAAA"
60 ?"FFFFFF"
70 ?"GGGGGG"
OK
>

```


6.5 New プログラムの消去

□ 書式

New

□ 引数

なし

□ 説明

プログラム領域のプログラムの消去、変数、配列変数を初期化します。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った、引数に数値以外を指定した

Overflow : 指定した数値が-32768 ～ 32767 を超えている

□ 利用例

プログラムを消去する

```
>list
10 I=0
20 Print "*";
30 I=I+1:If I<10 GoTo 20
40 Print
50 End
OK
>run
*****
OK
>?i
10
OK
>new
OK
>list
OK
>?i
0
OK
>
```

6.6 Load プログラムの読み込み

□ 書式

Load

Load プログラム番号

Load "ファイル名"

□ 引数

プログラム番号 : 0 ~ 内部 EEPROM のプログラム番号 (省略時は 0)

ファイル名 : 外部 I2C EEPROM 内のプログラムを保存したファイル名 (半角・全角 14 バイト迄)

□ 説明

内部 EEPROM または外部 I2C EEPROM からプログラムを読み込みます。

引数を省略した場合は、内部 EEPROM のプログラム番号 0 を読み込みます。

プログラムを読みの際、変数領域は初期化されます。

外部 I2C EEPROM については、「1.13I2C 接続 EEPROM へのプログラム保存対応」も参照して下さい。

プログラム中で実行された場合は、該当プログラムをロードしてそのプログラムを実行します。

その再、変数領域は初期化されるため、変数の値は引き継ぐことは出来ません。

引数にプログラム番号を指定した場合は、指定したプログラム番号を内部 EEPROM から読み込みます。

プログラム番号 0~で指定します。デフォルトでは 0 のみ指定可能ですが、コンパイルコンフィグレーションの設定で、プログラムサイズを 512 バイト以下にした場合、0~1 の指定が可能となります。

(Arduino MEGA2560 利用の場合、プログラムサイズ 2048 バイト、0~1 の指定が可能)

文字列("で囲む)指定した場合、ファイル名と見なし、外部 I2C EEPROM からファイルを読み込みます。

ファイル名の半角英数字は大文字・小文字の区別はしません。

Load マンドは、システムコマンド専用のコマンドです。対話形式でのみ利用可能です。

プログラムリスト内に記述して利用した場合はエラーとなります。

□ エラーメッセージ

Syntax error : 書式と異なる利用を行った、プログラム番号に変数、式を指定した

Illegal value : プログラム番号の指定が範囲外である

Bad filename : 指定したファイルが存在しない または 指定したファイル名が正しくない

I2C Device Error : I2C EEPROM が認識できない

Overflow : 指定した数値が -32767 ~ 32767 を超えている

□ 利用例

内部 EEPROM からプログラム番号 0 を読み込む

```
>load 0
OK
>
```

外部 I2C EEPROM ファイル"TEST01.BAS"を読み込む

```
>files ""
Drive:EPPROM32_1
猫踏んだ.BAS      性能.BAS      TEST01.BAS
3/3(31) files
OK
>load "test01.bas"
OK
```

6.7 Save プログラムの保存

□ 書式

Save

Save プログラム番号

Save "ファイル名"

□ 引数

プログラム番号 : 0 ~ 内部 EEPROM メモリのプログラム番号 (省略時は 0)

ファイル名 : 外部 I2C EEPROM に保存するファイル名 (半角・全角 14 バイト迄)

□ 説明

プログラムを内部 EEPROM または外部 I2C EEPROM に保存します。

引数の省略、数値を指定した場合は内部 EEPROM に保存します。

プログラムは最大で 2 つ保存可能です。保存先はプログラム番号 0~7 で指定します。

引数の省略した場合、プログラム番号 0 に保存します。

外部 I2C EEPROM については、「1.13I2C 接続 EEPROM へのプログラム保存対応」も参照して下さい。

引数を省略した場合は、内部 EEPROM のプログラム番号 0 に保存します。

引数にプログラム番号を指定した場合、指定したプログラム番号に保存します。

プログラム番号 0~7 で指定します。デフォルトでは 0 のみ指定可能ですが、コンパイルコンフィグレーションの設定で、プログラムサイズを 512 バイト以下にした場合、0~7 の指定が可能となります。

(Arduino MEGA2560 利用の場合、プログラムサイズ 2048 バイト、0~7 の指定が可能)

文字列 (" で囲む) 指定した場合、ファイル名と見なし、外部 I2C EEPROM に指定したファイル名で保存します
ファイル名の半角英数字は大文字・小文字の区別はしません。

□ エラーメッセージ

Syntax error : 書式と異なる利用を行った、プログラム番号に変数、式を指定した

Illegal value : プログラム番号の指定が 0~7 の範囲外である

Bad filename : 指定したファイル名が正しくない

Device full : 外部 I2C EEPROM の保存領域が一杯のため保存出来ない。

I2C Device Error : I2C EEPROM が認識できない

Overflow : 指定した数値が -32768 ~ 32767 を超えている

□ 利用例

プログラムを内部 EEPROM のプログラム番号 0 に保存する

```
>save 0
OK
>
```

プログラムを外部 I2C EEPROM にファイル名 "TEST01.BAS" として保存する

```
>save "test01.bas"
OK
>
```

6.8 Erase 保存プログラムのプログラム削除

□ 書式

Erase プログラム番号

Erase 開始プログラム番号, 終了番号

Erase "ファイル名"

□ 引数

プログラム番号 : 削除対象のプログラム番号 (0 ~)

開始プログラム番号, 終了番号 : 削除対象のプログラム番号の範囲 (0 ~)

ファイル名 : 半角・全角 14 バイト迄の文字列

□ 説明

内部 EEPROM 保存されている指定プログラム(プログラム番号 0~)、
または、外部 I2C EEPROM に保存されている指定ファイルを削除します。

(注意) 豊四季 Tiny BASIC のスケッチを書き込み後、FILES コマンドにて内部 EEPROM 内のプログラム一覧を表示した場合、意味不明の文字列が大量に表示される場合があります。
この場合は Erase コマンドを実行して、プログラム保存領域を削除して下さい。

外部 I2C EEPROM については、「1.13I2C 接続 EEPROM へのプログラム保存対応」も参照して下さい。

□ エラーメッセージ

Syntax error : 書式と異なる利用を行った、プログラム番号に変数、式を指定した

Illegal value : プログラム番号の指定が範囲外である

Bad filename : 指定したファイルが存在しない または 指定したファイル名が正しくない

I2C Device Error : I2C EEPROM が認識できない

Overflow : 指定した数値が -32768 ~ 32767 を超えている

□ 利用例

内部 EEPROM に保存されているプログラムの削除を行う。

```
>erase 0
OK
>files
0:(none)
OK
```

外部 I2C EEPROM に保存されているファイルの削除を行う。

```
>erase "SAMPLE01.BAS"
OK
```

6.9 Files 保存プログラムの一覧表示

□ 書式

Files

Files 開始プログラム番号

Files 開始プログラム番号, 終了プログラム番号

Files "ファイル名"

□ 引数

開始プログラム番号 : 表示開始プログラム番号 0~

終了プログラム番号 : 表示終了プログラム番号 0~

ファイル名 : 半角・全角 14 バイト迄で文字列、ワイルドカード*?の指定可能

□ 説明

内部 EEPROM または、外部接続 I2C EEPROM に保存されているプログラムの一覧を表示します。

引数を指定しない場合は、内部 EEPROM のプログラム番号(0~)の先頭行をリスト表示します。

外部 I2C EEPROM については、「1.13I2C 接続 EEPROM へのプログラム保存対応」も参照して下さい。

開始プログラム番号, 終了プログラム番号を指定した場合、その範囲のリストを表示します。

プログラム番号にプログラムが保存されていない場合は(none)と表示されます。

プログラム先頭行にコメントをつけると、一覧表示でのプログラムの内容が分かり易くなります。

例:

```
>files
0: 'VFD表示サンプル
OK
>
```

(注意) 豊四季タイニーBASIC (ファームウェア) の新規利用または更新を行った直後は、内部 EEPROM 上の既存データの内容の不整合により正しく表示できない場合があります。その場合は、ERASE コマンドにてプログラムの消去を行って下さい。

ファイル名を指定した場合、外部 I2C EEPROM に保存されているファイル一覧を表示します。

ファイル名の指定が"または""の場合、全てのファイルを一覧表示します。

ファイル名にはワイルドカード"*"(任意の文字列に一致)、“?”(任意の 1 文字に一致)の指定が可能です。

半角英字は大文字・小文字の区別はしません。

例:

```
>files ""
Drive:EEPROM32_1
猫踏んだ.BAS      性能.BAS      TEST01.BAS      さっちゃん.BAS
TEST10.BAS
5/5(31) files
OK
>
>files "t*0?.bas"
Drive:EEPROM32_1
TEST01.BAS
1/5(31) files
OK
>
```

上記一覧表示の"Drive:EEPROM32_1"はフォーマット時に指定した I2C EEPROM のドライブ名です。

"5/5(31) files"の表示の数字は、

ファイル名指定の条件に該当するファイル数 / 保存全ファイル数 (保存可能ファイル数)

を意味します。

□ エラーメッセージ

Syntax error	: 書式と異なる利用を行った、プログラム番号に変数、式を指定した
Illegal value	: 指定した開始プログラム番号, 終了プログラム番号の値が正しくない。
Bad filename	: 指定したファイルが存在しない または 指定したファイル名が正しくない
I2C Device Error	: I2C EEPROM が認識できない
Overflow	: 指定した数値が -32768 ~ 32767 を超えている

□ 利用例

プログラムを作成し、先頭行にコメントを付けて内部 EEPROM に保存します。

保存しているプログラムを Files コマンドで確認します。

コメントには日本語(シフト JIS) の記述も可能です。

```
>10 'こんにちは、世界
>20 ?"こんにちは、世界"
>save
OK
>files
0: 'こんにちは、世界
OK
>run
こんにちは、世界
OK
>
```

6.10 Format 外部接続 I2C EEPROM の領域初期化

□ 書式

Format 容量

Format 容量, "ドライブ名"

□ 引数

容量 : 利用する EEPROM の容量 4、8、16、32、64 のいずれかを指定

ドライブ名 : 利用する EEPROM のドライブ名称 (省略時はblank)

□ 説明

外部 I2C EEPROM の領域初期化を行います。利用出来る EEPROM は I2C 接続の 24xxx 系列となります。
容量は 4k バイト (32k ビット) ~ 64k バイト (512k ビット) に対応します。

領域初期化の際、利用する EEPROM の容量(4~64)を指定します。

ドライブ名は利用者が自由に付けることが出来ます。省略時はblankとなります。

ドライブ名は、FILES コマンドでファイル一覧を表示する際に表示されます。

EEPROM に保存できるファイル数は、プログラムサイズが 1024 バイトの場合は次のようになります。

EEPROM 表記	容量(k バイト)	保存可能ファイル数 (個)
24xx32	4	3
24xx64	8	7
24xx128	16	15
24xx256	32	31
24xx512	64	63

デフォルトでは、対象となる I2C EEPROM のスレーブアドレスは\$50(7 ビットアドレス) です。

デフォルトと異なるスレーブアドレスの EEPROM を利用する場合は、DRIVE コマンドにて指定することが出来ます。

外部 I2C EEPROM については、「1.13I2C 接続 EEPROM へのプログラム保存対応」も参照して下さい。

□ エラーメッセージ

Syntax error : 書式と異なる利用を行った、プログラム番号に変数、式を指定した

Illegal value : 容量の指定が範囲外である

Bad filename : 指定したドライブ名が正しくない

I2C Device Error : I2C EEPROM が認識できない

Overflow : 指定した数値が-32768 ~ 32767 を超えている

□ 利用例

RTC モジュールについている容量 4k バイトの I2C EEPROM (スレーブアドレス\$57)の初期化を行う。

```
>DRIVE $57
OK
>FORMAT 4,"EEPROM4_1"
OK
>files ""
Drive:EEPROM4_1

0/0(3) files
OK
>
```

6.11 Drive 外部接続 I2C EEPROM の指定

□ 書式

Drive I2C スレーブアドレス

Drive "ドライブ"

□ 引数

I2C スレーブアドレス : 利用対象の I2C EEPROM のスレーブアドレス \$50 ~ \$57 (デフォルト \$50)

ドライブ : 利用対象の I2C EEPROM のドライブ A ~ H (デフォルト A)

□ 説明

利用対象の外部接続 I2C EEPROM のスレーブアドレスを指定します。

デフォルトのスレーブアドレスは\$50 です。異なるアドレスを利用したい場合や、スレーブアドレスの異なる複数の EEPROM を同時接続して利用する場合に利用します。

I2C スレーブアドレスは、“A”~“H”または“a”~“h”の文字で指定で切り替えることも出来ます。

アドレス\$50+n は A から n 番目に対応します。

対応例：

\$50: "A" (デフォルト)

\$51: "B"

\$57: "H" (RTC モジュールに搭載している EEPROM 等)

外部 I2C EEPROM については、「1.13I2C 接続 EEPROM へのプログラム保存対応」も参照して下さい。

□ エラーメッセージ

Syntax error : 書式と異なる利用を行った、プログラム番号に変数、式を指定した

Illegal value : 容量の指定が範囲外である

I2C Device Error : I2C EEPROM が認識できない

Overflow : 指定した数値が-32768 ~ 32767 を超えている

□ 利用例

RTC モジュールに搭載している容量 4k バイトの I2C EEPROM (スレーブアドレス\$57)の初期化を行う。

```
>DRIVE "h"
OK
>FORMAT 4,"EEPROM4_1"
OK
>files ""
Drive:EEPROM4_1

0/0(3) files
OK
>
```


6.12 Rem コメント

□ 書式

Rem コメント

' コメント

□ 引数

コメント : 任意の文字列

□ 説明

プログラムに説明等の記載を行います。

'(シングルクォート)はREM の省略形です。

Rem および以降の文字以降はすべてコメントとみなし、プログラムとして実行されません。

プログラムの先頭行にコメントを付けた場合は、[Files](#) コマンドで保存プログラム一覧を表示時に、各プログラムの見出しとなります。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

□ 利用例

コメントの記述例

```
>list
1 '1 から 10 の合計を求める
10 Let I=1:Let S=0
20 Let S=S+I
30 Let I=I+1
40 If I<=10 Goto 20
50 Print "1 から 10 までの合計=";S
OK
>
```

6.13 Let 変数に値を代入

□ 書式

Let 変数=値

Let 配列変数=値, 値, 値, 値...

変数=値

配列変数=値, 値, 値, 値...

□ 引数

変数 : 変数、または配列変数

配列変数 : 配列変数 @(値)

配列の値(添え字)には式、数値、変数、配列変数、数値定数の利用が可能

値 : 式、数値、変数、配列変数、数値定数

□ 説明

変数に値を代入します。Letは省略可能です。

値は式、数値、定数、変数、配列変数等を評価した整数値です。

次の2行の実行結果は等価です。

```
Let X=X+1
X=X+1
```

配列変数への代入は、複数の値を指定した連続代入が可能です。

次の4行の実行結果は等価です。

```
10 Let @(3)=1,2,3,4,5
20 Let @(3)=1:Let @(4)=2:Let @(5)=3:Let @(5)=4:Let @(5)=5
30 @(3)=1,2,3,4,5
40 @(3)=1: @(4)=2: @(5)=3: @(5)=4: @(5)=5
```

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Overflow : 指定した数値が-32768 ~ 32767 を超えている

□ 利用例

1 から 10 までの整数の合計を求める

```
>list
1 '1 から 10 の合計を求める
10 Let I=1:Let S=0
20 Let S=S+I
30 Let I=I+1
40 If I<=10 Goto 20
50 Print "1 から 10 までの合計=";S
OK
>run
1 から 10 までの合計=55
OK
>
```

6.14 Print 画面への文字表示

□ 書式

Print

Print 文字列|値

Print 文字列|値; 文字列|値;...

Print #桁数,文字列|値; 文字列|値;... ;

※ ...は可変指定を示す

※ ; は連結指定、カンマ', 'も可能

□ 引数

文字列 : 文字列定数または文字列関数

値 : 数値定数、変数、配列変数、または数値関数、式

連結・改行抑制指定 : セミicolon';' または カンマ', '
文末に付けると改行抑制される

桁数 : #値の形式で指定する、値が正の場合、不足桁を空白埋め、負の場合 0 埋め

□ 説明

指定した文字列、値をスクリーン画面のカーソル位置に表示します。

文字列は文字列定数（""で囲った文字列）または、文字列関数の指定が可能です。

値には、定数（2 進数、10 進数、16 進数）、数値関数、式の指定が可能です。

```
>print "Hello,World"
Hello,World
OK
>print 123*3
369
OK
>print hex$($7b*3)
171
OK
```

連結・改行抑制 ';', ','

文字列、値は連結指定のセミicolon';', またはカンマ', 'にて連結して表示することが出来ます。

また最後に';'または', 'が付加されている場合は改行しません。

```
>list
10 Print "こんにちは、";
20 Print "世界！"
30 M=9:D=15
40 PRINT "今日は",M,"月",D,"日です。"
OK
>run
こんにちは、世界！
今日は 9 月 15 日です。
OK
```

数値関数、文字列関数も同じように連結出来ます。

```
>list
10 H=1234
20 Print "10 進数",H,"は";
30 Print "16 進数で表すと";
40 Print Hex$(H);"となります。"
50 Print "2 進数では";Bin$(H);"です。"
OK
>run
10 進数 1234 は 16 進数で表すと 4D2 となります。
2 進数では 10011010010 です。OK
```

数値の整形表示 '#値'

'#値' にて任意の桁数（指定桁に満たない場合は空白を入れる）にて表示します。

値の前に '-'（マイナス）を付加した場合、空白をではなく、0(零)で不足桁を補います。

'#値' は任意の位置、任意の回数指定できます。

```
>Print 1;":";2;":";3
1:2:3
OK
>Print #2,1;":";2;":";3
 1: 2: 3
OK
>Print #-2,1;":";2;":";3
01:02:03
OK
>
```

表示位置の指定

[Locate](#) コマンドを併用することで、スクリーン画面の任意の位置に文字を表示することが出来ます。

```
10 Locate 10,2
20 Print "Hello!"
```

色、属性の指定

[Color](#) コマンド、[Attr](#) マンドを併用することで、文字の前景色、背景色の指定、点滅、アンダーライン等の属性を付加することが出来ます。

```
10 Color 4,3
20 Print "Hello,";
40 Attr 2
50 Print "World".
60 Color 7,0:Atr 0
```

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った、引数に数値以外を指定した
 Overflow : 指定した数値が -32768 ～ 32767 を超えている

□ 利用例

時刻形式で表示する

```
10 A=10:B=52:C=00
20 Locate 0,0:Print #-2,A;":";B;":";C
```

実行結果

```
10:52:00
```

6.15 Input 数値の入力

□ 書式

Input 変数

Input 変数, オーバーフロー時の既定値

Input プロンプト, 変数

Input プロンプト, 変数, オーバーフロー時の既定値

□ 引数

変数 : 入力した値を格納する変数または配列変数

プロンプト : 文字列定数 "プロンプト"

オーバーフロー時の既定値 : -32768 ~ 32767

□ 説明

現在のカーソル位置にて数値入力を行い、指定した変数にその値を格納します。

入力できる文字は符号 '-', '+', 数字 '0' ~ '9'、入力訂正の[BS]、[DEL]、入力確定の[Enter]キーです。

それ以外の入力はありません。

引数に変数のみを指定した場合、"変数名:"を表示しその後ろの位置から数値を入力します。

```
>input a
A:
```

プロンプトを指定した場合は、そのプロンプトを表示しその後ろ位置から数値を入力します。

```
>input "Value=",a
Value=
```

オーバーフロー時の既定値を指定した場合、入力値した数値がオーバーフローを発生した場合は、オーバーフロー時の既定値を変数に設定します。オーバーフロー時の既定値を設定していない場合は、Overflow エラーとなります。

オーバーフロー時の既定値なし

```
>input a
A:111111
Overflow
OK
>
```

オーバーフロー時の既定値あり

```
>input a,-1
A:111111
OK
?a
-1
OK
>
```

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った、引数に数値以外を指定した

Overflow : 指定した数値が-32768 ~ 32767 を超えている

□ 利用例

```
>input a
A=1234
OK
>input "Value=",a
Value=1234
OK
```

6.16 Cls 画面表示内容の全消去

□ 書式

Cls

Cls 消去対象画面

□ 引数

なし

□ 説明

画面上に表示している内容を全て消します。

[Color](#) コマンドで色指定した場合、文字色、背景色の設定を全画面に反映されます。

□ エラーメッセージ

Illegal value : 指定した引数が有効範囲でない

Overflow : 指定した数値が-32768 ~ 32767 を超えている

Syntax error : 文法エラー、書式と異なる利用を行った

□ 利用例

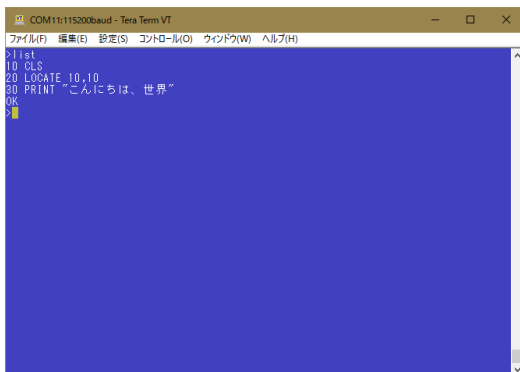
画面の表示内容を消去後に、指定位置に文字列を表示する。

```
10 Cls
20 Locate 10,10
30 Print "こんにちは、世界"
```

画面を持文字色を白、背景色を青に設定し、全画面に反映する。

```
>color 7,4
Ok
>cls
OK
>list
10 Cls
20 Locate 10,10
30 Print "こんにちは、世界"
OK
>
```

実行結果



6.17 Color 文字色の設定

□ 書式

Color 文字色

Color 文字色, 背景色

□ 引数

文字色: 色コード 0 ~ 8

背景色: 色コード 0 ~ 8

□ 説明

シリアルコンソールの文字色の設定を行います。指定した色は以降の文字表示に反映されます。

文字色、背景色で指定する色コードに対する色は次の表の通りです。

表 1 色コード

色コード	色
0	黒
1	赤
2	緑
3	茶
4	青
5	マゼンタ
6	シアン
7	白(デフォルト)
8	黄

(注意) 利用するターミナルソフトにより色が正しく表示されない場合があります。
属性指定との併用では正しく表示されない場合があります。

□ エラーメッセージ

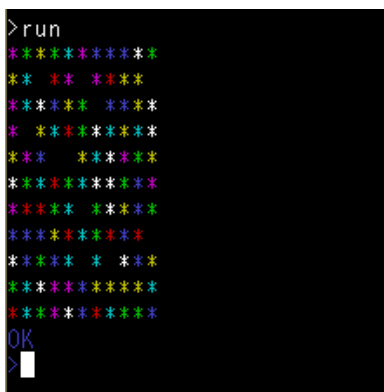
Syntax error : 文法エラー、書式と異なる利用を行った
 Illegal value : 色コードに範囲外の値を指定した
 Overflow : 指定した数値が-32768 ~ 32767 を超えている

□ 利用例

ランダムな色で*を表示する

```
10 For I=0 To 10
20 For J=0 To 10
30 Color Rnd(8): ? "*";
35 Wait 100
40 Next J
50 ?
60 Next I
```

実行結果



```
>run
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
OK
>
```

6.18 Attr 文字表示属性の設定

□ 書式

Attr 属性

□ 引数

属性：属性コード 0 ～ 4

□ 説明

シリアルコンソールの文字の表示属性を設定します。指定した表示属性は以降の文字表示に反映されます。

属性に指定する属性コードは次の表の通りです。

表 2 属性コード

属性コード	機能
0	標準(デフォルト)
1	下線
2	反転
3	ブリンク
4	ボールド

(注意) 利用するターミナルソフトにより色が正しく表示されない場合があります。
属性指定との併用では正しく表示されない場合があります。

□ エラーメッセージ

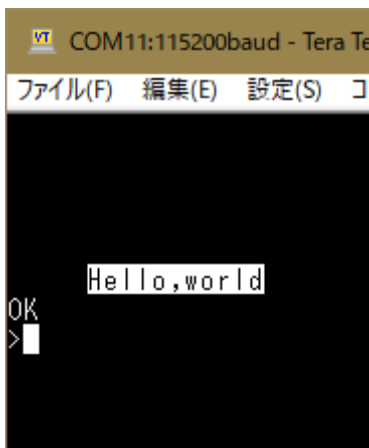
Syntax error : 文法エラー、書式と異なる利用を行った
 Illegal value : 属性コードに範囲外の値を指定した
 Overflow : 指定した数値が-32768 ～ 32767 を超えている

□ 利用例

Hello,world を反転表示する

```
10 CLS
20 LOCATE 5,5
30 ATTR 2:?"Hello,world"
40 ATTR 0
```

実行結果



6.19 Locate カーソルの移動

□ 書式

LOCATE 横位置, 縦位置

□ 引数

横位置: 画面上の横位置 0 ~ 79

縦位置: 画面上の縦位置 0 ~ 23

□ 説明

カーソルを指定した位置に移動します。

指定した位置が有効でない場合は、有効値の最大値（または最小値）の位置にカーソルを移動します。

Locate 100,50 は Locate 79,23 となります。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Overflow : 指定した数値が-32768 ~ 32767 を超えている

□ 利用例

画面の指定位置にメッセージを表示する

```
10 CLS
20 LOCATE 5,0:PRINT "Hello,world."
30 LOCATE 6,1:PRINT "TinyBASIC"
40 LOCATE 7,2:PRINT "Thank you."
```

実行結果

```
Hello,world.
TinyBASIC
Thank you.
```

6.20 Free プログラム領域の残りバイト数の取得（数値関数）

□ 書式

Free()

□ 引数

なし

□ 戻り値

プログラム領域の残りバイト数 0 ～ 1023

□ 説明

プログラム領域の残りバイト数を返します。

作成したプログラムサイズを確認する場合は、下記の記述にて行うことが可能です。

```
PRINT 1024-FREE()
```

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

'(' or ')' expected : 引数部の記述が正しくない

□ 利用例

プログラムサイズを調べる

```
>PRINT 1024-FREE()
287
OK
>
```

6.21 Inkey キー入力の読み取り（数値関数）

□ 書式

Inkey()

□ 引数

なし

□ 戻り値

押したキーの文字コード 0 ～ 255

キーが押されていない場合は 0 を返します

□ 説明

キーボードの入力をチェックしキーコードを返します。キーが押されていない場合は 0 を返します。

キーコードについては「1.11 キー入力コード」を参照下さい。

（注意） [ESC]、[CTRL-C]はプログラム中断用のため、キー入力の読み取りは出来ません。
 キーボードにおいて利用出来ないキーはコードの取得が出来ません。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

'(' or ')' expected : '（ または ）'がない

□ 利用例

入力したキーのコードを調べる

```
10 Cls
20 I= Inkey()
30 If I>0 Locate 0,0:?"#3,I
40 GoTo 20
```

6.22 Rnd 乱数の発生（数値関数）

□ 書式

Rnd(値)

□ 引数

値 : 0 ~ 32767

式、変数、配列変数、数値、数値定数の指定が可能

□ 戻り値

0 から指定した値未満の乱数

□ 説明

0 から指定した値-1 の範囲の乱数を発生させ、その値を返します。

R=RND(10)

の場合、変数 R には 0~9 範囲の値が代入されます。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Overflow : 指定した数値が-32768 ~ 32767 を超えている

'(' or ')' expected : '（ または ）'が無い

□ 利用例

5 回のジャンケンの表示

```

10 @(0)="グー","チョキ","パー"
20 For I=1 To 5
30 R=Rnd(3)
40 Print I,":";Str$(@(R))
50 Next
>run
1:チョキ
2:パー
3:グー
4:パー
5:チョキ
OK
>

```

6.23 Abs 絶対値の取得（数値関数）

□ 書式

Abs(値)

□ 引数

値 : -32767 ~ 32767

式、変数、配列変数、数値、数値定数の指定が可能

□ 戻り値

指定した値の絶対値

□ 説明

指定した値の絶対値を返します。

（注意） -32768 の絶対値 32768 はオーバーフローとなるため、-32768 を指定した場合はオーバーフローエラーとなります。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った
 Overflow : 指定した数値が-32767 ~ 32767 を超えている
 '(' or ')' expected : '(' または ')' が無い

□ 利用例

変数の値の絶対値を表示する

```
10 For I=-3 To 3
20 Print Abs(I)
30 Next
>Run
3
2
1
0
2
3
OK
>
```

6.24 Asc 文字から文字コードへの変換（数値関数）

□ 書式

Asc(文字列)

Asc(文字列, 文字位置)

Asc(変数)

Asc(変数, 文字位置)

□ 引数

文字列: "文字列"

"あいう ABC" の形式とし、ダブルクォーテーション文字を囲みます

文字位置: 1~32767

変換対象となる左からの文字位置を指定します。

変数: 文字列参照している変数または配列変数

□ 戻り値

指定文字に対応するシフト JIS コード (0 ~ \$FFFF)

□ 説明

指定した全角文字または半角文字に対応する文字コードを返します。

2 文字以上の文字列を指定した場合、先頭の文字コードを返します。

```
10 ?Hex$(Asc("あ ABCD"))
>run
82A0
OK
```

文字位置を指定した場合、指定した位置の文字コードを返します。

全角 1 文字は 1 文字とカウントします。次の例では 4 番目の文字“う”の文字コードを取得しています。

```
10 ?Hex$(Asc("A あ B う CD",4))
>run
82A4
OK
```

変数を指定した場合、変数が参照している文字列のコードを返します。

```
10 A="AB あ CDEF"
20 ?Hex$(Asc(A))
30 ?Hex$(Asc(A,3))
>run
41
82A0
OK
```

(注意) 変数の文字列参照はプログラム中にのみ有効です。コマンドラインでは文字列参照を正しく行うことが出来ません。

□ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
Illegal value	: 文字指定位置が不当
'(' or ')' expected	: '(', ' ' または ')', ' ' がない
Overflow	: 指定した数値が-32768 ~ 32767 を超えている

□ 利用例

文字列の各文字のシフト JIS コードを表示します。

```
10 S="こんにちは、世界！"
20 L=Len(S)
30 For I=1 To L
40 Print Hex$(Asc(S,I),2)
50 Next
```

実行結果

```
run
>run
82B1
82F1
82C9
82BF
82CD
8141
90A2
8A45
8149
OK
>
```

6.25 Byte 文字列のバイト数の取得（数値関数）

□ 書式

Byte(変数)

Byte(文字列)

□ 引数

変数 : 変数、または配列変数

文字列 : "文字列"の形式（ダブルクォーテーションで囲み）

□ 戻り値

文字列の長さ 0 ～ 32767

□ 説明

文字列定数、変数で参照してる文字列のバイト数をカウントし、その値を返します。

```

10 ?Byte("12345678")
20 A="ABCDEF"
30 @(0)="abcdef"
40 ? Byte(A)
50? Byte(@(0))
>run
8
6
6
OK

```

(注意) 文字列のサポートは限定的です。
 他の文字列関数と組み合わせた利用はサポートしていません。
 例) 下記のような記述は出来ません。
 ?BYTE(BIN\$(100))

本関数は全角 1 文字は 2 バイトとしてカウントします。全角を 1 文字として扱いたい場合は、[Len\(\)](#)を利用して下さい。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った
 Overflow : 指定した数値が-32768 ～ 32767 を超えている
 '(' or ')' expected : '(', または ')' が無い

□ 利用例

変数 A が参照している文字列のバイト数を Byte() 関数で取得し、各文字の文字コードを表示します。

```

10 'Byte()サンプル
20 A="Hello!"
30 For I=1 To Byte(A)
40 ?Str$(A,I,1);";";Asc(A,I)
50 Next

```

実行結果

```

run
H:72
e:101
l:108
l:108
o:111
!:33
OK

```


6.26 Len 文字列の長さの取得（数値関数）

□ 書式

Len(変数)

Len(文字列)

□ 引数

変数 : 変数、または配列変数

文字列 : "文字列"の形式（ダブルクォーテーションで囲み）

□ 戻り値

文字列の長さ 0 ～ 32767

□ 説明

文字列定数、変数で参照してる文字列の文字数をカウントし、その値を返します。

全角文字、半角文字とも 1 文字としてカウントします。

```

10 ?Len("1 あ 2 い 34 う 5678")
20 A="AB さいたま CDEF"
30 @(0)="abcdef 埼玉"
40 ?Len(A)
50 ?Len(@(0))
>run
11
10
8
OK
>

```

(注意) 文字列のサポートは限定的です。
 他の文字列関数と組み合わせた利用はサポートしていません。
 例) 下記のような記述は出来ません。
 ?LEN(BIN\$(100))

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Overflow : 指定した数値が-32768 ～ 32767 を超えている

'(' or ')' expected : '（ または ）'が無い

□ 利用例

変数 A が参照している文字列の長さ Len 関数で取得し、各文字の文字コードを表示します。

```

10 'Len()サンプル
20 A="こんにちは世界!"
30 For I=1 To Len(A)
40 ?Str$(A,I,1);":";Hex$(Asc(A,I))
50 Next

```

実行結果

```

>run
こ:82B1
ん:82F1
に:82C9
ち:82BF
は:82CD
世:90A2
界:8A45
!:21
OK
>

```

6.27 Map 数値のスケール変換（数値関数）

□ 書式

Map(値, 下限, 上限, 新下限, 新上限)

□ 引数

値 : 変換対象の数値 -32768 ～ 32767
 下限 : 対象数値の数値の下限 -32768 ～ 32767
 上限 : 対象数値の数値の上限 -32768 ～ 32767
 新下限 : 新しい対象数値の数値の下限 -32768 ～ 32767
 新上限 : 新しい新しい対象数値の数値の下限 -32768 ～ 32767

□ 戻り値

スケール変換後の値 -32767 ～ 32767

□ 説明

指定した値のスケール変換を行い、その値を返します。

例：

```
A=Map(Ana(A1),0,1023,0,99)
```

上記の例ではアナログ入力値 0～1023 を 0 ～ 99 のレンジに変換し、その値を返します。

アナログ入力や画面座標指定、PWM パルス出力等にて MAP 関数を使うことで、簡単にスケール変換を行うことができます。

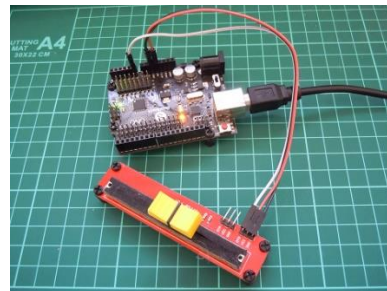
□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った
 Overflow : 指定した数値が-32768 ～ 32767 を超えている
 '(' or ')' expected : '(' または ')' が無い

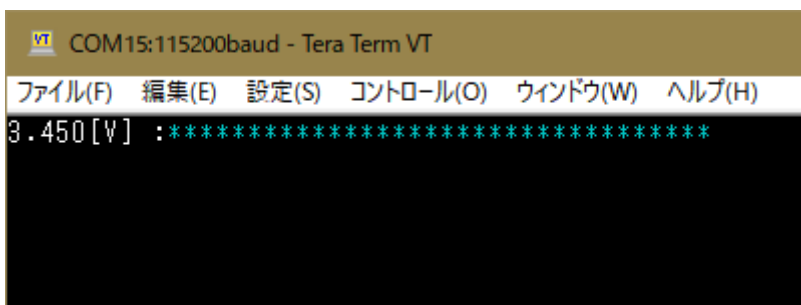
□ 利用例

スライド抵抗によるアナログ入力値 0～1023 を 0～5V で表示する

```
10 Cls
20 V=Map(Ana(A1),0,1023,0,5000)
30 Color 7:Locate 0,0:?Dmp$(V,3);"[V] :";
40 N=V/100
50 Color 6
60 For I=1 To 50
70 If I<=N ?""*; Else ?" ";
80 Next
90 Wait 50
100 Goto 20
```



実行結果



スライダバーの操作に応じて、リアルタイムに電圧値とバーグラフを更新表示します。

6.28 Grade 等級判定 (数値関数)

□ 書式

Grade(判定対象値, 配列番号, 個数)

□ 引数

判定対象値 : 変換対象の数値 -32768 ~ 32767

配列番号 : 0 ~ 15

個数 : 1 ~ 16

□ 戻り値

等級 0 ~ 99 (配列番号=0)、範囲外 -1

□ 説明

判定対象値と配列に格納した値を等級の閾値として判定し、配列番号を等級値として返します。

配列に格納されている閾値は大きい順にソートされている必要があります。

範囲外の場合は -1 を返します。

例 :

```

10 @(0)=5120,2560,1280,640,320,160,80,40,20,10
20 Input "V=",V
30 G=Grade(V,0,10)
40 ?G
50 GoTo 20

```

実行結果

```

run
V=12
9
V=10000
0
V=1000
3
V=200
5
V=500
4
V=3
-1

```

上記の例では入力値に該当する等級を返します。

10000 を入力した場合、5120 以上であるため 0 を返します。1000 を入力した場合、640 以上であるため 3 を返します。4 を入力した場合は配列に格納されている最小値が 10 であるため、範囲外と判定し -1 を返します。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Illegal value : 指定した値が有効範囲外である

Overflow : 指定した数値が -32768 ~ 32767 を超えている

'(' or ')' expected : '(' または ')' が無い

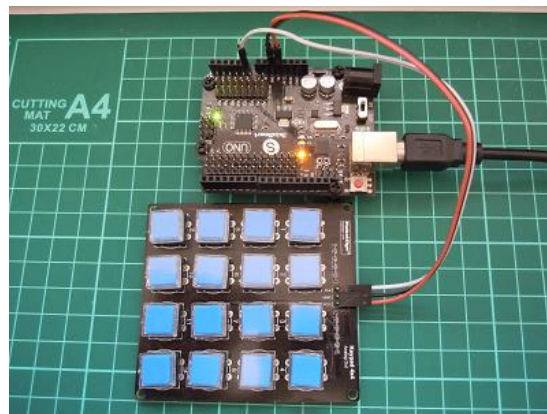
□ 利用例

アナログ入力 4x4 キーパッドのキー入力判定を行う。

```

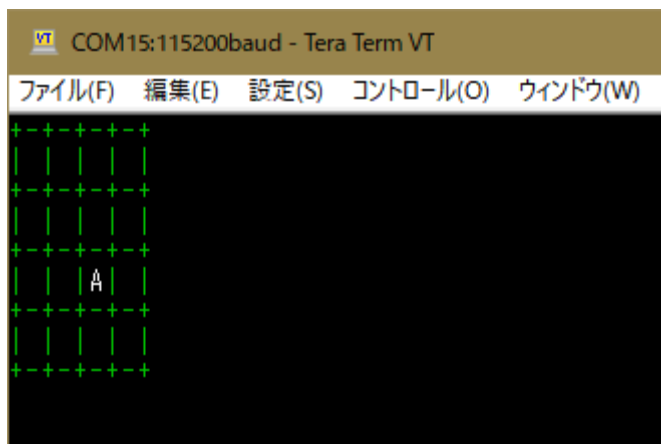
10 'Keypad 4x4
20 Cls:GoSub 130
30 K=-1:X=1:Y=1
40 @(0)=1013,920,840,780,670,630,590,560
50 @(8)=502,477,455,435,400,320,267,228
60 K=G
70 G=Grade(Ana(A1),0,16)
80 If G<>K Wait 1 GoTo 60
90 IF G<0 Locate 0,0:GoSub 130:GoTo 60
100 X=(G%4)*2+1:Y=(G/4)*2+1
110 Color 7:Locate X,Y:?Hex$(G,1)
120 GoTo 60
130 Color 2
140 For Y=1 To 4
150 ?"+---+---+"
160 ?"|   |   |   |"
170 Next
180 ?"+---+---+"
190 Return

```



実行結果

押したキーに 4x4 のマス目に対応する 16 進数(0~A)を表示します。



6.29 Chr\$ 文字コードから文字への変換（文字列関数）

□ 書式

Chr\$(文字コード)

Chr\$(文字コード, 文字コード, ... , 文字コード)

□ 引数

文字コード アスキーコード または シフト SJIS 文字コード: 0~\$FFFF

□ 戻り値

指定した文字コードに対する文字

□ 説明

指定したシフト文字コードに対応する文字を返します。 全角 SJIS、半角アスキーコードに対応します。

文字コードは複数指定することで、複数文字の変換可能です。

本関数は [Print](#)、[Vmsg](#)、[Cprint](#)、[Nmsg](#) 引数にて利用可能です。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

'(' or ')' expected : '(', 'または ')' が無い

Overflow : 指定した数値が -32768 ~ 32767 を超えている

□ 利用例

ひらがな`あいうえお`を表示する

10 ?Chr\$(\$82A0,\$82A2,\$82A4,\$82A6,\$82A8)

run

あいうえお

OK

6.30 Bin\$ 数値から 2 進数文字列への変換（文字列関数）

□ 書式

Bin\$(数値)

Bin\$(数値 , 桁数)

□ 引数

数値：変換対象の整数値(-32768 ～ 32767)

桁数：出力桁数(0 ～ 16)

□ 戻り値

2 進数文字列(1 桁～16 桁)

□ 説明

指定した数値を 2 進数文字列に変換します。

数値には式、変数、定数等の指定が可能です。

本関数は [Print](#)、[Vmsg](#)、[Cprint](#)、[Nmsg](#) 引数にて利用可能です。

桁数を指定した場合は数値が指定した桁数に満たない場合は、0 で桁を補います。

指定した桁数を超える場合は数値の桁数を優先します。

桁数を指定しない場合は、先頭の 0 は付加されません。

□ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
'(' or ')' expected	: '(' または ')' がない
Illegal value	: 桁数の値が正しくない
Overflow	: 指定した数値が -32768 ～ 32767 を超えている

□ 利用例

2 進数で変数の内容を表示する

```
10 A=1234:B=1
20 Print Bin$(A)
30 Print Bin$(B)
40 Print Bin$(A,4)
50 Print Bin$(B,4)
```

```
run
10011010010
1
10011010010
0001
OK
```

6.31 Hex\$ 数値から 16 進数文字列への変換（文字列関数）

□ 書式

Hex\$(数値)

Hex\$(数値 , 桁数)

□ 引数

数値：変換対象の整数値(-32768 ～ 32767)

桁数：出力桁数(0 ～ 4)

□ 戻り値

16 進数文字列(1 桁～4 桁)

□ 説明

指定した数値を 16 進数文字列に変換します。

数値には式、変数、定数等の指定が可能です。

本関数は [Print](#)、[Vmsg](#)、[Cprint](#)、[Nmsg](#) 引数にて利用可能です。

桁数を指定した場合は数値が指定した桁数に満たない場合は、0 で桁を補います。

指定した桁数を超える場合は数値の桁数を優先します。

桁数をしない場合は、先頭の 0 は付加されません。

□ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
'(' or ')' expected	: '(', または ')' がない
Illegal value	: 桁数の値が正しくない
Overflow	: 指定した数値が -32768 ～ 32767 を超えている

□ 利用例

入力した整数値を 16 進数表示する

```

10 Input "value=",V
20 Print Hex$(V,4)
30 GoTo 10

```

6.32 Dmp\$ 整数の小数付き数値文字列への変換（文字列関数）

□ 書式

Dmp\$(数値)

Dmp\$(数値, 小数点桁数)

Dmp\$(数値, 小数点桁数, 整数部桁数)

□ 引数

数値 : 変換対象整数 -32768 ～ 32767

小数点桁数 : 小数点以下の桁数を指定 0 ～ 4（省略時は2）

整数部桁数 : 小数点以上の桁数を指定 0 ～ 8（省略時は0）

□ 戻り値

小数点を付加した数値文字列

□ 説明

指定した数値を小数点付きの文字列数値に変換します。

本関数は [Print](#)、[Vmsg](#)、[Cprint](#)、[Nmsg](#) 引数にて利用可能です。引数の小数点以下の桁数を n とした場合、数値÷10ⁿの計算を行い、小数点以下の数値を含めて表示します。

```
Print Dmp$(3141,3)
```

```
3.141
```

小数点以下の桁数を指定しない場合、小数点桁数は2となります。

```
Print Dmp$(3141)
```

```
3.14
```

整数部桁数を指定した場合、桁数に満たない場合は空白で補完します。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Illegal value : 指定した引数の値が有効範囲を超えている

Overflow : 指定した数値が-32768 ～ 32767 を超えている

Illegal value : 桁数の値が正しくない

'(' or ')' expected : '(' または ')' が無い

□ 利用例

アナログ入力値 0～1023 を 0～5.0V で表示する

```
10 Cls
20 V=Map(Ana(A0),0,1023,0,3300)
30 Locate 0,0:?Dmp$(V,3)
40 Wait 200
50 GoTo 20
```

実行結果

```
run
2.011
2.215
2.294
2.330
```

アナログ入力値 0～1023 を [Map](#) 関数にて 0～5000 に変換し、

0～5000 の数値を Dmp\$関数にて “0.000” ～ “5.000” の文字列数値に変換して表示しています。

6.33 Str\$ 変数が参照する全角を含む文字列の取得・文字列の切り出し

□ 書式

Str\$(変数)

Str\$(変数, 先頭位置, 長さ)

Str\$(文字列)

Str\$(文字列, 先頭位置, 長さ)

□ 引数

変数 : 文字列を参照している変数または配列変数

先頭位置 : 切り出す文字位置先頭 1~32767

長さ : 切り出す文字の長さ 1~32767

文字列 : ダブルクォーテーションで囲った文字列定数("文字列")

□ 戻り値

指定した文字列および変数が参照している文字列を全部または一部を切り出して返します。

全角SJS1文字は1文字としてカウントします。

□ 説明

指定した文字列および変数が参照している文字列を全部または一部を切り出して返します。

引数に先頭位置、長さを指定した場合は、その条件にて文字列を切り出して返します。

先頭位置は1からの指定となります。

本関数は [Print](#)、[Vmsg](#)、[Cprint](#)、[Nmsg](#) 引数にて利用可能です。

例:

```
10 S="Hello,world! こんにちは世界!"
20 ?Str$(S)
30 ?Str$(S,14,8)
40 ?Str$("あいうえお",5,1)
```

実行結果

```
>run
Hello,world! こんにちは世界!
こんにちは世界!
お
OK
```

上記の例では、変数Sが参照している文字列“Hello,world! こんにちは世界!”に対して、

20行は全て出力、30行は14番目からの文字から8文字“こんにちは世界!”を出力、

40行は直接指定した文字列“あいうえお”の5番目の文字から1文字“お”を出力しています。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Illegal value : 指定した文字位置、長さの値が不当

'(' or ')' expected : '(' または ')' が無い

Overflow : 指定した数値が-32768 ~ 32767を超えている

□ 利用例

変数が参照している文字列を切り出して表示します。

```
10 'モジ'シツカ サンプル
20 S="Hello,Tiny BASIC"
30 L=LEN(S)
40 PRINT STR$(S); " LEN=";L
50 PRINT STR$(S,1,5)
60 C=ASC(S,12)
>RUN
Hello,Tiny BASIC LEN=16
Hello
```

6.34 Wait 時間待ち

□ 書式

Wait 待ち時間(ミリ秒)

□ 引数

待ち時間： 0 ～ 32767 （単位 ミリ秒）

□ 説明

引数で指定した時間(ミリ秒単位)、時間待ち(ウェイト)を行います。

最大で 32767 ミリ秒（32.8 秒）の時間待ちが可能です。

長い時間待ちを行う必要がある場合は、Tick 関数や GetTime を使った方法を検討してください。

類似機能として、ボードを指定時間休止する [Sleep](#) もあります。

（注意） 時間待ち中はキー操作によるプログラム中断を行うことは出来ません。
短い時間の指定にてループ処理を行う等の対策を行ってください。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った
Illegal value : 待ち時間に範囲外の値を指定した
Overflow : 指定した数値が-32768 ～ 32767 を超えている

□ 利用例

画面上の指定位置に経過した秒数を 1 秒間隔で更新表示する

```
10 Cls
20 C=0
30 Locate 0,0
40 Print #-5,C;"秒"
50 Wait 1000
60 C=C+1
70 GoTo 30
```

6.35 Tick 経過時間取得（数値関数）

□ 書式

TICK()

TICK(モード)

□ 引数

モード 0: 経過時間をミリ秒単位で取得する（デフォルト）

1: 経過時間を秒単位で取得する

□ 戻り値

モード指定に従った経過時間を返す。

モード 0: 0 ～ 32767 ミリ秒の経過時間を返す(約 32.767 秒でオーバーフロー)

モード 1: 0 ～ 32767 秒の経過時間を返す(約 9.1 時間でオーバーフロー)

□ 説明

起動からの経過時間を返します。

モード指定無し、または 0 を指定の場合、ミリ秒単位の経過時間を返します。

1 を指定した場合は、秒単位の経過時間を返します。

（注意） オーバーフローが発生しますので、長時間の測定には利用することが出来ません。

□ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
'(' or ')' expected	: '(', または ')' が無い、引数の指定が正しくない
Illegal value	: モードの指定値が正しくない
Overflow	: モードの指定値が -32767 ～ 32767 を超えている

□ 利用例

1 万回ループの実行時間を測定する（プログラム: TICK.BAS）

```

10 A=0
20 T=Tick()
30 For I=1 To 10000
40 A=A+1
50 Next
60 Print A
70 Print Tick()-T;"msec"

>run
10000
788msec

```

6.36 Poke 指定アドレスへのデータ書き込み

□ 書式

Poke 仮想アドレス, データ

Poke 仮想アドレス, データ, データ, ... データ (可変個数指定)

□ 引数

仮想アドレス : 仮想アドレス(16 ビット) \$0000 ~

データ : 書き込むデータ (下位 8 ビットのみ有効)

□ 説明

指定した仮想アドレスに指定したデータを書き込みます。

仮想アドレスの指定には次の定数を利用することで有用な領域への書き込みが簡単に行えます。

Var	: 変数領域
Array	: 配列変数領域(@0)~
Prg	: プログラム領域
Mem	: ユーザーワーク領域
Mem2	: ユーザーワーク領域 2

仮想アドレスの詳細については、「1.7 メモリーマップ」の「仮想アドレス」を参照下さい。

ユーザーワーク領域、ユーザーワーク領域 2 は利用者が自由に利用出来る領域です。

それ以外の領域は BASIC の実行にて利用する領域です。

指定した仮想アドレスに上記以外の領域を指定した場合はエラーとなります。

例として変数 X 値格納アドレスに 2 バイトデータを書き込んでみます。

```
10 X=0
20 Poke Var+(ASC("X")-Asc("A"))*2,$34,$12
30 Print Hex$(X,2)
>run
1234
OK
```

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Out of range value : 領域外のアドレスを指定した

Overflow : 指定した数値が -32768 ~ 32767 を超えている

□ 利用例

変数 A から Z の内容をユーザーワーク領域に保存する

```
10 For I=0 To 51
20 Poke Mem+I, Peek(Var+I)
30 Next
```

6.37 Peek 指定アドレスの値参照（数値関数）

□ 書式

Peek(仮想アドレス)

□ 引数

仮想アドレス： 参照を行う仮想アドレス(16 ビット) \$0000 ～

□ 戻り値

指定した仮想アドレスに格納されている 1 バイトデータ (0～255)

□ 説明

指定した仮想アドレスに格納されている値（1バイト）を返します。

仮想アドレスの指定には次の定数を利用することで有用な領域への参照が簡単に行えます。

Var	: 変数領域
Array	: 配列変数領域(@0)～
Prg	: プログラム領域
Mem	: ユーザーワーク領域
Mem2	: ユーザーワーク領域 2

仮想アドレスの詳細については、「1.7 メモリーマップ」の「仮想アドレス」を参照下さい。

ユーザーワーク領域は利用者が自由に利用出来る領域です。

それ以外の領域は BASIC の実行にて利用する領域です。

エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
Out of range value	: 領域外のアドレスを指定した
'(' or ')' expected	: 括弧の指定が正しくない
Overflow	: 指定した数値が-32768 ～ 32767 を超えている

□ 利用例

変数 A から Z の内容をユーザーワーク領域に保存する

```
10 For I=0 TO 51
20 Poke Mem+I, Peek(Var+I)
30 Next
```

6.38 Vcls VFD 表示内容の全消去

□ 書式

Vcls

□ 引数

なし

□ 説明

VFD の表示内容を消去します。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Overflow : 指定した数値が-32768 ～ 32767 を超えている

□ 利用例

VFD に「こんにちは、世界！」の表示と消去を繰り返します。

```

10 Vmsg 0,"こんにちは、世界！"
20 Wait 500
30 Vcls
40 Wait 500
50 GoTo 10

```

□ 補足

VFD の表示内容を一時的に見えなくする方法として次の別の方法があります。

- ① 輝度を 0 にする

Vbright 0

- ② 表示を OFF にする

Vdisplay Off

上記の場合、VFD の表示内容自体は保持されています。

- ① の場合は、VBbright 255、②の場合は Vdisplay On で再び表示出来ます。

6.39 Vdisplay VFD 表示のオン・オフ

□ 書式

Vdisplay モード

□ 引数

モード : On または 0 以外は表示
Off または 0 は非表示

□ 説明

VFD の表示のオン・オフ設定を行います。

Vbright コマンドで輝度設定を行っている場合は、その設定輝度にて表示されます。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った
Illegal value : 範囲外の値を指定した
Overflow : 指定した数値が-32768 ～ 32767 を超えている

□ 利用例

VFD に「こんにちは、世界！」の表示と非表示を繰り返します。

```
10 Vcls
20 Vmsg 0,"こんにちは、世界！"
30 Wait 500
40 Vdisplay Off
50 Wait 500
60 Vdisplay On
70 GoTo 30
```

6.40 Vbriht VFD 輝度設定

□ 書式

Vbriht 輝度

□ 引数

輝度 : 0(0%) ~ 255(100%)

□ 説明

VFD の表示の輝度を設定します。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った
 Illegal value : 範囲外の値を指定した
 Overflow : 指定した数値が-32768 ~ 32767 を超えている

□ 利用例

VFD に「こんにちは、世界！」の表示をフェーフォアウトします。

```
10 Vcls
20 Vmsg 0,"こんにちは、世界！"
30 Vbriht 10
40 For I=100 To 0 Step -10
50 Vbriht I
60 Wait 50
70 Next I
80 Wait 500
90 Vbriht 255
100 GoTo 40
```


6.41 Vmsg VFD にメッセージ文の表示

□ 書式

Vmsg 速度, メッセージ文

□ 引数

速度 : メッセージ文のスクロール速度 0 ~ 1024(ミリ秒)、または-1
0の場合は、即表示、-1の場合はデータ送信のみ(待ち時間なし、ラッチなし)

メッセージ文 : 文字列、変数、数値、関数を結合した文 ([Print](#) コマンドのフォーマット)

□ 説明

VFD に指定したメッセージ文を表示します。メッセージ文には全角文字(シフト JIS)の利用が可能です。

メッセージ文は [Print](#) コマンドと同じフォーマットによる指定が可能です。

速度には左スクロールする 1 ドット単位のウェイト時間(ミリ秒)を指定します。

0を指定した場合、即時表示します。

-1 を指定した場合、データの送信のみで表示には反映されません。次の表示(ラッチ)のタイミングで表示に反映されます。

速度に-1 を指定してデータ送信のみを行った後、速度に 0 以上を指定することで一緒に表示に反映されます。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った
Illegal value : 範囲外の値を指定した
Overflow : 指定した数値が-32768 ~ 32767 を超えている

□ 利用例

配列に格納した文字列を切り替えて表示する

```
10 'VDF表示サンプル
20 Tone 440,100:Tone 880,100
30 @(0)="熱いぞ!埼玉!"
40 @(1)="豊四季タイニーBASIC"
50 @(2)="VFD対応バージョン"
60 @(3)="只今開発中!"
70 @(4)="ちょっとメモリー的に厳しい.."
80 Vbriht 255
90 For S=0 To 4
100 Vcls:Vmsg 10,Str$(@(S))
110 For I=0 To 255 Step 15
120 Vbriht 255-I:Wait 40
130 Next
140 Wait 500
150 Vbriht 255:Wait 500
160 Vscroll 256,3
170 Wait 500
180 Next S
190 GoTo 80
```

□ 補足

データ送信のみを利用することで、左から右にスクロールする表示を行うことができます。

```
10 Vcls
20 S="こんにちは、世界!"
30 L=Len(S)
40 For I=256 To 1 Step -2
50 Vmsg -1,Str$(S)
60 Vscroll I,0
70 Vscroll 256,-1
80 Next I
90 Vbriht 255
100 GoTo 40
```

6.42 Vscroll VFD に指定長空白の表示

□ 書式

Vscroll スクロール幅, 速度

□ 引数

スクロール幅 : 1~256 (ドット)

速度 : スクロール速度 0 ~ 1024(ミリ秒)、または-1

0 の場合は、即表示、-1 の場合はデータ送信のみ (待ち時間なし、ラッチなし)

□ 説明

VFD に指定長の空白をスクロール表示することで、表示内容を左にスクロールさせます。

0 を指定した場合、即時表示します。

-1 を指定した場合、データの送信のみで表示には反映されません。次の表示 (ラッチ) のタイミングで表示に反映されます。

速度に-1 を指定してデータ送信のみを行った後、速度に 0 以上を指定することで一緒に表示に反映されます。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Illegal value : 範囲外の値を指定した

Overflow : 指定した数値が-32768 ~ 32767 を超えている

□ 利用例

配列に格納した文字列を切り替えて表示する

```

10 'VDF表示サンプル
20 Tone 440,100:TONE 880,100
30 @(0)="熱いぞ! 埼玉!"
40 @(1)="豊四季タイニーBASIC"
50 @(2)="VFD対応バージョン"
60 @(3)="只今開発中!"
70 @(4)="ちょっとメモリー的に厳しい.."
80 Vbright 255
90 For S=0 To 4
100 Vcls:Vmsg 10,Str$(@(S))
110 For I=0 To 255 Step 15
120 Vbright 255-I:Wait 40
130 Next
140 Wait 500
150 Vbright 255:Wait 500
160 Vscroll 256,3
170 Wait 500
180 Next
190 GoTo 80

```

□ 補足

データ送信のみを利用することで、左から右にスクロールする表示を行うことができます。

```

10 Vcls
20 S="こんにちは、世界!"
30 L=Len(S)
40 For I=256 To 1 Step -2
50 Vmsg -1,Str$(S)
60 Vscroll I,0
70 Vscroll 256,-1
80 Next
90 Vbright 255
100 GoTo 40

```

6.43 Vput VFD に直接データ送信

□ 書式

VPUT データ格納アドレス, データ長, 速度

□ 引数

データ格納アドレス : ビットマップデータ格納アドレス

データ長 : 1~512 (バイト)

速度 : スクロール速度 0 ~ 1024(ミリ秒)、または-1

: 0 の場合は、即表示、-1 の場合はデータ送信のみ (待ち時間なし、ラッチなし)

□ 説明

VFD に直接、データ (無加工データ) を送信します。

送信するデータはデータ格納アドレスに指定します。

速度に0を指定した場合、即時表示します。

-1 を指定した場合、データの送信のみで表示には反映されません。次の表示 (ラッチ) のタイミングで表示に反映されます。

速度に-1 を指定してデータ送信のみを行った後、速度に0以上を指定することで一緒に表示に反映されます。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Illegal value : 範囲外の値を指定した

Overflow : 指定した数値が-32768 ~ 32767 を超えている

□ 利用例

配列@()の値のビット位置を変化させて、その値 (2 バイト) のアドレスから2バイトを表示用データとして送信します。

```
1 'VFDジグザグ表示
10 Vcls
20 @()=1
30 For I=0 To 14
40 Vput Array,2,5
50 @()=@()<<1
60 Next
70 For I=0 To 14
80 @()=@()>>1
90 Vput Array,2,5
100 Next I
110 GoTo 30
```

実行結果



6.44 Tone 単音出力

□ 書式

Tone 周波数

Tone 周波数,出力期間

□ 引数

周波数 : 0 ~ 32767 (Hz) 0 の場合は消音

出力期間 : 0 ~ 32767 (ミリ秒) 0 の場合は、継続再生

□ 説明

指定した周波数の単音出力を行います。**PN8 ピン**を単音出力として使用します。

単音出力ピンに圧電スピーカー（圧電サウナダ）を接続すること音を出すことができます。

出力期間の指定がある場合は、その期間パルスを出します(ミリ秒単位)。

出力期間の指定がある場合、出力完了待ちを行います。

出力期間の指定がない場合は、[NoTone](#) コマンドで停止指示をするまでパルスを出し続けます。

音階・周波数対応表

	ド	ド#	レ	レ#	ミ	ファ	ファ#	ソ	ソ#	ラ	ラ#	シ
1	33	35	37	39	41	44	46	49	52	55	58	62
2	65	69	73	78	82	87	93	98	104	110	117	123
3	131	139	147	156	165	175	185	196	208	220	233	247
4	262	277	294	311	330	349	370	392	415	440	466	494
5	523	554	587	622	659	698	740	784	831	880	932	988
6	1047	1109	1175	1245	1319	1397	1480	1568	1661	1760	1865	1976
7	2093	2217	2349	2489	2637	2794	2960	3136	3322	3520	3729	3951
8	4186	4435	4699	4978	5274	5588	5920	6272	6643	7040	7459	7902

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Overflow : 指定した数値が-32768 ~ 32767 を超えている

□ 利用例

スペースキーを押したら音を鳴らす

```
10 If Inkey() = 32 Tone 880,50
20 GoTo 10
```

6.45 NoTone 単音出力停止

□ 書式

NoTone

□ 引数

なし

□ 説明

[Tone](#) コマンドによるパルス出力を停止します。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

□ 利用例

音を停止する

```
10 Tone 400  
20 Wait 200  
30 NoTone
```

6.46 Play 音楽演奏(MML 文)

□ 書式

Play "MML 文"

□ 引数

MML 文 : MML を定義した文字列

□ 説明

※本コマンドは、コンパイルオプションの設定にて利用可能となります（デフォルトでは利用不可）

MML (Music Macro Language)にて定義した文法を元に、接続している圧電スピーカーを使って単音による、音楽演奏を行います。音の出力は**ピン番号 8**を利用します。

MML 文には次のコマンドを利用することが出来ます。

- 音階記号 [#|+|-][長さ][.]
 音階指定 : C、D、E、F、G、A、B ~ または c、d、e、f、g、a、b
 順番に ド、レ、ミ、ファ、ソ、ラ、シ、ド の音階に対応します。
 # : 半音上げる (省略可能)
 + : 半音上げる (省略可能)
 - : 半音下げる (省略可能)
 長さ : 1、2、4、8、16、32、64 (省略時は L による長さ、デフォルト値 4)、省略可能
 1 は全音符、2 は 2 分音符、4 は四分音符、64 は 64 分の一音符
 . : 長さを半分伸ばす
- R[長さ] : 休符
 長さ : 1、2、4、8、16、32、64 (省略時は L による長さ、デフォルト値 4)、省略可能
 1 は全音符、2 は 2 分音符、4 は四分音符、64 は 64 分の一音符
 . : 長さを半分伸ばす
- L<長さ>
 音の長さを指定します。省略時の長さの指定。初期値は 4 (四分音符)
 長さ : 1、2、4、8、16、32、64 (省略時は L による長さ、デフォルト値 4)、省略可能
 1 は全音符、2 は 2 分音符、4 は四分音符、64 は 64 分の一音符
 . : 長さを半分伸ばす
- O<オクターブ>
 音の高さを指定します。
 オクターブ : 1~8 初期値は 4
- < : 1 オクターブ上げる
- > : 1 オクターブ下げる
- T<テンポ> : テンポを指定する。初期値は 120、32~255
- 空白文字 : スキップします。

演奏の中断は[ESC]キー 2 回押し、または[CTRL]+C キーにて行うことが出来ます。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った
 Illegal MML : MML 文の文法エラー
 Overflow : 指定した数値が-32768 ~ 32767 を超えている

□ 利用例

「ドラミファソラシド」と演奏する。

```
PLAY "CDEFGAB<C"
OK
```

「ねこふんじゃった」を演奏する。

※MML は、下記のサイトに公開されているものを利用しています。

- 主体性の無いページ (<http://astr.me.land.to/>)

MML (<http://astr.me.land.to/tool/mabi/>)

ねこふんじゃった! (<http://astr.me.land.to/tool/mabi/mml/nekof.htm>)

```
10'ねこふんじゃった
20 PLAY "L16D+C+R8F+RF+RD+C+R8F+RF+RD+C+L8RF+RF+R"
30 PLAY "L16FRFRD+C+R8FRFRD+C+R8FRFRD+C+"
40 PLAY "L8RFRFRL16F+RF+RD+C+R8F+RF+RD+C+R8F+RF+RD+C+"
50 PLAY "L8RF+RF+RL16FRFRD+C+R8FRFRD+C+R8FR"
60 PLAY "L16FRD+C+L8RFRFRL16F+RF+RD+C+L8RF+RF+RF+RF+RF+RF+R"
70 PLAY "L16FRFRD+C+L8RFRFRFRFRFRFR"
80 PLAY "L16F+RF+RD+C+R8F+RF+RD+C+R8F+RF+RD+C+"
90 PLAY "L8RF+RF+RL16FRFRD+C+R8FRFRD+C+R8FRFRD+C+"
100 PLAY "L8RFRFRL16F+RF+R8.F+RC+C+D8C+8.FRF+"

```

6.47 Tempo 音楽演奏のテンポの設定

□ 書式

Tempo テンポ

□ 引数

テンポ : 32 ~ 500 (デフォルト 120)

□ 説明

[Play](#) コマンドによる音楽演奏のテンポの設定を行います。

デフォルト値は 120 です。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Illegal value : 引数に範囲外の値を指定した

Overflow : 指定した数値が -32768 ~ 32767 を超えている

□ 利用例

テンポを 160 に設定します。

```
10 Tempo 160
20 Play "CDEFGAB"
```


6.48 GetFont 美咲フォントデータ取得（数値関数）**□ 書式**

GetFont(フォントデータ格納アドレス、シフト JIS コード)

□ 引数

フォントデータ格納アドレス : 0 ~ 32767(\$0000 ~ \$FFFF の有効な仮想アドレス領域)
シフト JIS コード : \$0000 ~ \$FFFF

□ 戻り値

0 : 該当フォントあり
1 : 該当フォント無し

□ 説明

指定したシフト JIS コードに対応するフォントデータ 8 バイトを取得します。
戻り値として、対応するフォントが存在する場合は 0、存在しない場合は、1 を返します。
フォントデータ 8 バイトは、フォントデータ格納アドレスで指定した領域に格納します。

フォントデータ 8 バイトは次の構成となります。

アドレス	MSB	LSB
fontdata + 0		
+ 1		
+ 2		
+ 3		
+ 4		
+ 5		
+ 6		
+ 7		

フォントはフラッシュメモリ容量の制約から、漢字・非漢字合わせて500文字の対応となります。
利用可能な文字については「1.15 美咲フォント対応」を参照下さい。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った
Illegal value : 指定した引数の値が不当である
Out of range value : 指定した値が有効範囲を超えている
Overflow : 指定した数値が-32768 ~ 32767 を超えている
'(' or ')' expected : '(' または ')' がない

□ 利用例

「あ」のフォントを取得し、フォントパターンをキャラクタ文字で表示する

```
>list
1 'フォント取得
10 R=GetFont(Mem2,Asc("あ"))
20 For I=0 To 7
30 D=Peek(Mem2+I)
40 For J=0 To 7
50 If D&($80>>J) ?"■"; Else ?" ";
60 Next
70 ?
80 Next
OK
>run
  ■
 ■■■■■
  ■
  ■■■■■
 ■■■ ■■
■ ■■ ■■
 ■■■ ■
  ■■ ■

OK
>
```

6.49 Gpio 汎用入出力ピンの設定

□ 書式

Gpio ピン番号, モード

□ 引数

ピン番号 : 2 ~ 21 (利用出来ないピンあり)、A0~A7

Arduino MEGA2560 利用の場合、ピン番号 : 2~69、A0~A15

モード :

Output : デジタル出力

Float : デジタル入力 (フロート状態 : Arduino の INPUT 指定と同じ)

PullUp : デジタル入力 (内部プルアップ抵抗有効)

□ 説明

ボード上の指定したピン番号の入出力機能の設定を行います。

Arduino の pinMode 関数に相当します。

ピン番号 2~21 の一部は利用出来ないピンがあります。詳細については「1.6 ボードピン構成」を参照して下さい。

重要

V0.07 以降では、デジタル入出力を行うコマンドおよび関数 ([Out](#)、[In](#)、[ShiftOut](#)、[ShiftIn](#)、[PulseIn](#)) の利用において、Gpio コマンドでのモード設定は不要になりました。旧版との互換性と、ピンモードの設定のみを行う場合のみ利用して下さい。

(注意) デフォルトでは、INPUT_FL が設定されています。

□ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
Cannot use GPIO fuinction	: ピン番号に利用出来ないモード設定を行った
Overflow	: 指定した数値が -32768 ~ 32767 を超えている
Illegal value	: 指定した数値が不当である

□ 利用例

ピン番号 2 をデジタル入力 (内部プルアップ) の設定します。

```
10 Cls
20 Gpio 2, PullUp
```

6.50 Out デジタル出力

□ 書式

Out ピン番号, 出力値

□ 引数

ピン番号: 2 ~ 21 (利用出来ないピンあり)、A0~A7

Arduino MEGA2560 利用の場合、ピン番号: 2~69、A0~A15

□ 説明

指定ピンから、指定した出力を行います。

出力値:

Low または 0 : 0V を出力する (Off の指定も可能)

High or 0 以外の値 : 5V を出力する (On の指定も可能)

ピン番号 2~21 の一部は利用出来ないピンがあります。詳細については「1.6 ボードピン構成」を参照して下さい。

□ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
Cannot use GPIO fuinction	: ピン番号に利用出来ないモード設定を行った
Illegal value	: ピン番号、モードに範囲外の値を指定した
Overflow	: 指定した数値が-32768 ~ 32767 を超えている

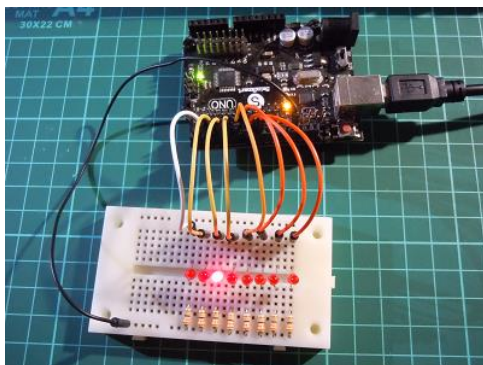
□ 利用例

ボード上のLED(ピン番号 13)を点滅させます。

```
10 'L 効
20 Out LED ,On
30 Wait 300
40 Out LED ,Off
50 Wait 300
60 GoTo 20
```

ピン番号 2~9 に接続しているLED を順次点灯させます。

```
10 'L チカ 2
20 For I=2 To 9
30 For J=2 To 9:Out J,I=J:Next:Wait 100
70 Next
80 GoTo 20
```



LED に接続する抵抗は 330Ωを利用しています。

6.51 PWM PWM パルス出力

□ 書式

PWM ピン番号, デューティー比

□ 引数

ピン番号 : 3, 5, 6, 9, 10, 11

Arduino MEGA2560 利用の場合、ピン番号 : 2~13、44~46

デューティー比 : 0 ~ 255

0 が 0% となります。

255 が 100% となります。

□ 説明

指定ピンから、PWM パルス出力を行います。周波数は 490Hz となります (Arduino Uno 互換)

パルス出力を停止する場合は、デューティー比に 0 を指定して下さい。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Illegal value : ピン番号、モードに範囲外の値を指定した

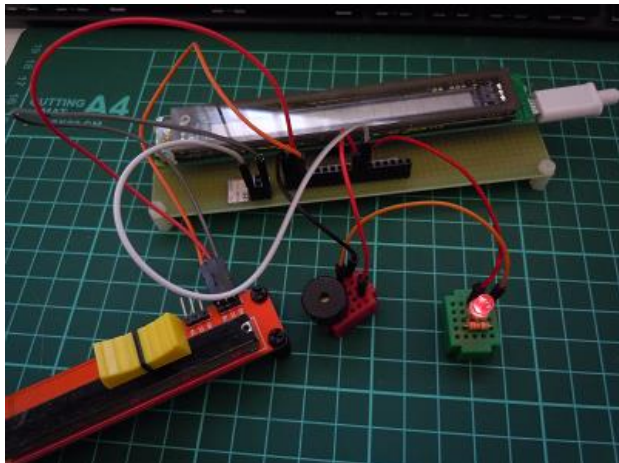
Overflow : 指定した数値が -32768 ~ 32767 を超えている

□ 利用例

アナログ入力ピン A6 (ピン 20) に接続したスライド抵抗でピン 6 に接続している LED の明るさを制御します。

```
10 A=Ana(A6)>>2
20 If A<10 A=0
30 PWM 6,A
40 GoTo 10
```

実行結果



6.52 In デジタル入力（数値関数）

□ 書式

In(ピン番号)

In(ピン番号, 入力モード)

□ 引数

ピン番号 : 2 ~ 21（利用出来ないピンあり）、A0～A7

Arduino MEGA2560 利用の場合、ピン番号：2～69、A0～A15

入力モード : Float 内部プルアップ無効(デフォルト)

PullUp 内部プルアップ有効

□ 戻り値

取得した値 0(Low) または 1(High)

□ 説明

指定ピンのデジタル入力値を読み取り、その値を返します。

入力モードをし指定することで、内部でのプルアップを有効にすることが出来ます。省略時はプルアップ無効です。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Illegal value : ピン番号、モードに範囲外の値を指定した

Cannot use GPIO fuinction : ピン番号に利用出来ないモード設定を行った

Overflow : 指定した数値が-32768 ~ 32767 を超えている

'(' or ')' expected : '(' または ')'がない

□ 利用例

ピン番号 2 からデジタル入力値を読み取り、その値を画面に随時表示します。

```

10 Cls
30 A=In(2,PullUp)
40 Locate0,0: ? A;
50 GoTo 30

```

6.53 ShiftOut デジタルシフトアウト出力

□ 書式

SHIFTOUT データピン番号, クロックピン番号, 出力形式, 出力データ

□ 引数

データピン番号 : データを出力するピン 2 ~ 21 または A0 ~ A7

クロックピン番号 : クロックを出力するピン 2 ~ 21 または A0 ~ A7

Arduino MEGA2560 利用の場合、ピン番号 : 2~69、A0~A15

出力形式 : 出力するデータの順番を下記にて指定

LSB または 0 : 下位ビットから出力する

MSB または 1 : 上位ビットから出力する

出力データ : 出力するデータ (下位 8 ビットのみ有効)

□ 説明

クロックにて同期を行い、データピンから 1 バイト分のデータを 1 ビットずつ出力します。

Arduino の shiftOut() と同等の動作をします。

データピン、クロックピンは事前に GPIO コマンドによる機能設定 (デジタル出力) が必要です。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Illegal value : ピン番号、モードに範囲外の値を指定した

Overflow : 指定した数値が -32768 ~ 32767 を超えている

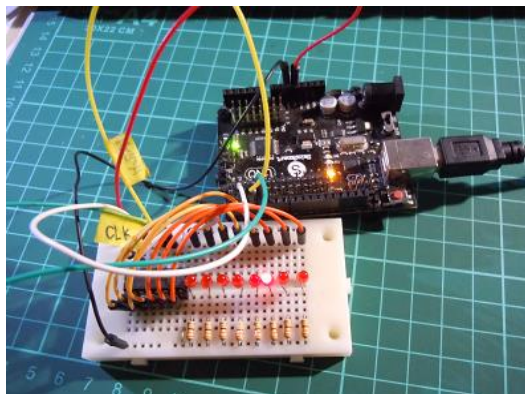
□ 利用例

シフトレジスタ 74HC595 を使って外付け 8 個の LED を制御する

```

1 'シフトレジスタ 74HC595 テスト
2 'ピン 2 DATA、3:LATCH、4:CLK
10 Cls:T=100
20 "[ループ]"
30 D=$80
40 For I=0 To 6
50 GoSub "[LED8 個制御]"
60 D=D>>1
70 Wait T
80 Next
90 For I=0 To 6
100 GoSub "[LED8 個制御]"
110 D=D<<1
120 Wait T
130 Next
140 GoTo "[ループ]"
150 "[LED8 個制御]"
160 Out 3,LOW
170 ShiftOut 2,4,MSB,D
180 Out 3,High
190 Return

```



6.54 ANA アナログ入力（数値関数）

□ 書式

ANA(ピン番号)

□ 引数

ピン番号： 14 ～ 21 または定数 A0 ～ A7（Arduino の A0～A7 に対応）
 Arduino MEGA2560 利用の場合、ピン番号： 54～69、A0～A15

□ 戻り値

取得した値 0～1023(10 ビット)

□ 説明

指定ピンのアナログ入力値を読み取り、その値を返します。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った
 Illegal value : ピン番号、モードに範囲外の値を指定した
 Overflow : 指定した数値が-32768 ～ 32767 を超えている
 '(' or ')' expected : '(' または ')'がない

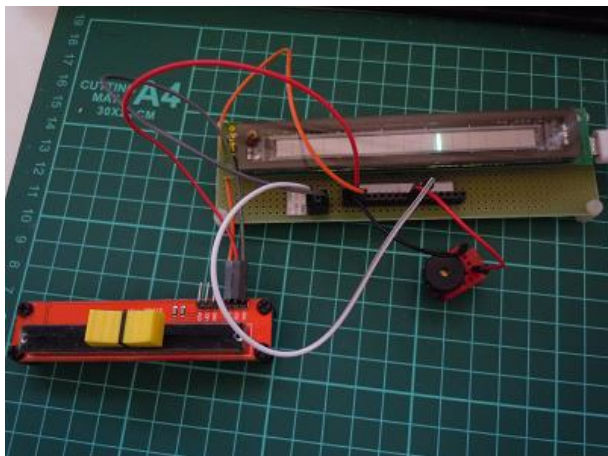
□ 利用例

□ 利用例

ピン A6（No.20）に接続したスライド抵抗器で VFD に表示している縦線の表示位置を操作します。

```
10 @(0)=$FFFF
20 A=ANA(A6)>>2:IF A=0 A=1
30 VPUT ARRAY,2,-1
40 VSCROLL A,0
50 VSCROLL 256,-1
60 WAIT 20
70 GOTO 20
```

実行結果



VFD に表示されている縦線がスライド抵抗の位置に応じて移動します。

6.55 SHIFTIN デジタルシフト入力（数値関数）

□ 書式

SHIFTIN(データピン番号, クロックピン番号, 入力形式)

SHIFTIN(データピン番号, クロックピン番号, 入力形式, 条件)

□ 引数

データピン番号 : 2 ~ 21 または A0 ~ A7

クロックピン番号 : 2 ~ 21 または A0 ~ A7

Arduino MEGA2560 利用の場合、ピン番号 : 2~69、A0~A15

入力形式 : 入力するデータの順番を下記にて指定

LSB または 0 下位ビットから入力する

MSB または 1 上位ビットから入力する

条件 : LOW または HIGH

データピンからのデータを読み取るクロックのタイミングを指定します。

LOW : クロックが LOW の場合にデータを読み取る。

HIGH : クロックが HIGH の場合にデータを読み取る。

□ 戻り値

入力値（1 バイト）

□ 説明

クロックにて同期を行い、データピンから 1 バイト分のデータを 1 ビットずつ入力します。

Arduino の shiftIn() と同等の動作をします。

引数に条件を指定した場合、クロックが指定した状態の時にデータピンからデータを読み取ります。

条件を指定していない場合は、条件は HIGH（Arduino の shiftIn() と同様の仕様）となります。

データピン、クロックピンは事前に GPIO コマンドによる機能設定（デジタル入出力）が必要です。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Illegal value : ピン番号、モードに範囲外の値を指定した

Overflow : 指定した数値が -32768 ~ 32767 を超えている

'(' or ')' expected : '(' または ')' がない

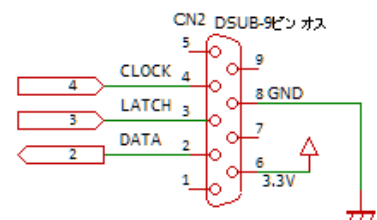
□ 利用例

ファミコン用ゲームパッドからボタン操作情報を取得する

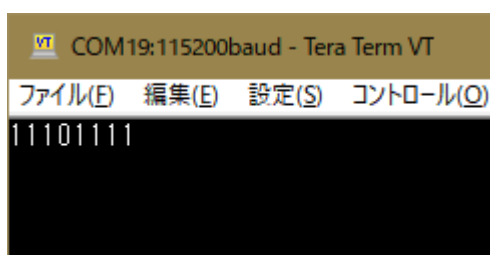
```

10 CLS
20 GPIO 2, INPUT_FL: 'データ
30 GPIO 3, OUTPUT: 'ラッチ
40 GPIO 4, OUTPUT: 'クロック
50 OUT 3, HIGH
60 OUT 3, LOW
70 R=SHIFTIN(2,4, LSB, LOW)
80 LOCATE 0,0: ?BIN$(R,8)
90 WAIT 100
100 GOTO 50

```



実行結果



6.56 PULSEIN 入力パルス幅の計測力（数値関数）

□ 書式

PULSEIN(パルス入力ピン番号, 検出信号, タイムアウト)

PULSEIN(パルス入力ピン番号, 検出信号, タイムアウト, スケール値)

□ 引数

パルス入力ピン番号 : 2 ~ 21 または A0 ~ A7

Arduino MEGA2560 利用の場合、ピン番号: 2~69、A0~A15

検出信号 : LOW、HIGH または 0、1 測定対象のパルス

タイムアウト : パルス検出待ちタイムアウト時間 0 ~ 32767 (ミリ秒)

スケール値 : 計測時間のスケール変換 1 ~ 1327671 (デフォルト値 1)

□ 戻り値

正常時: 測定したパルス幅 0 ~ 32767 (単位はスケール値 × マイクロ秒)

タイムアウト時: 0

オーバーフロー時: -1

□ 説明

パルス入力ピン番号で指定したピンから入力される信号のパルス幅を計測し、その値を返します。

測定完了は、入力ピンの状態が**検出信号**の状態になった時点で計測を開始し、**検出信号**の状態でなくなった時点で計測を終了します。たとえば、測定する**検出信号**が **HIGH** の場合、PULSEIN() はピンが **HIGH** になるのを待ち、**HIGH** になった時点で計測を開始し、ピンが **LOW** になるタイミングで計測を終了し、そのパルスの長さを返します。**タイムアウト**内に完全なパルスが受信されなかった場合は 0 を返します。

測定したパルスが整数値 32767 を超えた場合は、オーバーフローとし **-1** を返します。

オーバーフローを回避したい場合は、**スケール値**を適宜調整して下さい。

スケール値が 1 の場合、1~31767 マイクロ秒までのパルスの測定を行うことができます。この場合、500kHz~15Hz までのパルスを測定出来ます。15Hz より遅い、信号の測定はオーバーフローが発生します。

スケール値が 1000 の場合、1~31767 ミリ秒までのパルスの測定を行うことができます。この場合、500Hz~0.015Hz までのパルスの測定が可能です。

本関数は Arduino の pulseIn() 関数を利用して計測しています。

測定値はあくまでも目安であり、精度はあまり高くありません。

データピン、クロックピンは事前に GPIO コマンドによる機能設定（デジタル入出力）が必要です。

□ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
Illegal value	: ピン番号、モードに範囲外の値を指定した
Overflow	: 指定した数値が -32768 ~ 32767 を超えている
'(' or ')' expected	: '(' または ')' がない

□ 利用例

超音波距離センサ HC-SR04 を使った距離計測

```

10 '超音波距離センサ HC-SR04 を使った距離計測
20 CLS
30 A=2:B=3:T=180
40 K=(3315+T*6)/20
50 GPIO A,OUTPUT
60 GPIO B,INPUT_FL
70 "@LOOP"
80 OUT A,LOW:OUT A,HIGH:OUT A,LOW
90 D=PULSEIN(B,HIGH ,200,1)
100 ?DMP$(D/100*K,2)
110 WAIT 200
120 GOTO "@LOOP"

```

変数 T は室内温度の 10 倍の値を指定します。距離計算において、温度補正を行っています。

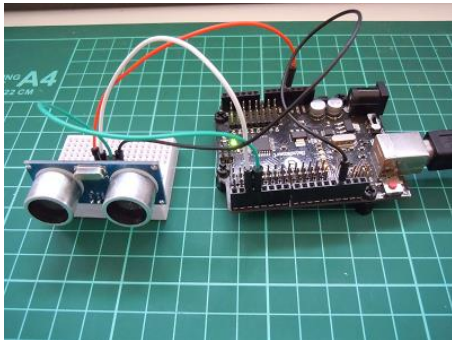
接続

HC-SR04 Trig 端子 : 2

HC-SR04 Echo 端子 : 3

HC-SR04 GND 端子 : GND

HC-SR04 VCC 端子 : 5V



実行結果

```

VT COM15:115200baud - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
4.38
4.38
4.38
4.38
4.38
6.57
10.95
6.57
0.00
4.38
0.00
6.57
0.00
6.57
41.61
41.61
39.42
43.80
48.18
0.00
6.57
0.00
6.57

```

6.57 I2CR I2C スレーブデバイスからのデータ受信（数値関数）

□ 書式

I2CR(デバイスアドレス,コマンドアドレス,コマンド長,受信データアドレス,データ長)

□ 引数

デバイスアドレス : 0 ~ 127 (\$00 ~ \$7F) (7ビット指定)
I2C スレーブアドレスを7ビット形式で指定
コマンドアドレス : 0 ~ 32767 (\$0000 ~ \$1FFFF)
送信するコマンドが格納されている仮想アドレス
コマンド長 : 0 ~ 32767
送信するコマンドのバイト数
受信データアドレス : 0 ~ 32767 (\$0000 ~ \$1FFFF)
受信データを格納する SRAM 内仮想アドレス
データ長 : 0 ~ 32767
受信するデータのバイト数

□ 説明

I2C スレーブデバイスからデータを受信します。

送信先はデバイスアドレスにて I2C スレーブアドレスを指定します。

受信において、コマンド等の制御データの送信が必要な場合は、コマンドアドレス、コマンド長にて送信するデータを指定します。受信のみを行う場合は、コマンドアドレス、コマンド長に 0 を設定します。送信受信するデータは仮想アドレスにて指定します。

仮想アドレスの指定には次の定数を利用することで有用な領域への参照が簡単に行えます。

VAR : 変数領域
ARRAY : 配列変数領域
PRG : プログラム領域
MEM : ユーザーワーク領域

I2C 通信には次の接続ピンを利用します。

18 or A4 : (Arduino A4 ピン SDA)	Arduino MEGA2560 の場合 21
19 or A5 : (Arduino A5 ピン SCL)	Arduino MEGA2560 の場合 22

□ 戻り値

0 : 正常終了
1 : 通信バッファに対してデータが長すぎる
2 : アドレス送信に NACK が返された
3 : データ送信に NACK が返された
4 : その他のエラー

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った
Illegal value : 指定した引数の値が不当である
Out of range value : 指定した値が有効範囲を超えている
Overflow : 指定した数値が -32768 ~ 32767 を超えている

□ 利用例

I2C EEPROM(AT24C256 スレーブアドレス \$50)にデータの読み書きを行う

```
100 POKE MEM+0,$00,$00
110 POKE MEM+2,64,65,66,67
120 R=I2CW($50,MEM,2,MEM+2,4)
130 ? "snd r=";R
140 POKE MEM+6,0,0,0,0
150 WAIT 5
160 R=I2CR($50,MEM,2,MEM+6,4)
170 ? "rcv r=";R
180 ? "rcv data:";
190 FOR I=0 TO 3
200 ? PEEK(MEM+6+I);" ";
210 NEXT I
220 ?
```

実行結果

```
run
snd r=0
rcv r=0
rcv data:64 65 66 67
OK
```

6.58 I2CW I2C スレーブデバイスへのデータ送信（数値関数）

□ 書式

I2CW(デバイスアドレス,コマンドアドレス,コマンド長,データアドレス,データ長)

□ 引数

デバイスアドレス : 0 ~ 127 (\$00 ~ \$7F) (7ビット指定)
I2C スレーブアドレスを7ビット形式で指定
コマンドアドレス : 0 ~ 32767 (\$0000 ~ \$1FFFF)
送信するコマンドが格納されている SRAM 内仮想アドレス
コマンド長 : 0 ~ 32767
送信するコマンドのバイト数
データアドレス : 0 ~ 32767 (\$0000 ~ \$1FFFF)
送信するデータが格納されている SRAM 内仮想アドレス
データ長 : 0 ~ 32767
送信するデータのバイト数

□ 説明

I2C スレーブデバイスにデータを送信します。

送信先はデバイスアドレスにて I2C スレーブアドレスを指定します。

送信するデータは SRAM 先頭からの仮想アドレスにて指定します。

送信するデータはあらかじめ、設定しておく必要があります。

コマンドとデータの区別はありません。デバイスに対しては、単純にコマンド、データの順に送信しています。

仮想アドレスの指定には次の定数を利用することで有用な領域への参照が簡単に行えます。

VAR : 変数領域
ARRAY : 配列変数領域
PRG : プログラム領域
MEM : ユーザーワーク領域

I2C 通信には次の接続ピンを利用します。

18 or A4 : (Arduino A4 ピン SDA)	Arduino MEGA2560 の場合 21
19 or A5 : (Arduino A5 ピン SCL)	Arduino MEGA2560 の場合 22

□ 戻り値

0 : 正常終了
1 : 通信バッファに対してデータが長すぎる
2 : アドレス送信に NACK が返された
3 : データ送信に NACK が返された
4 : その他のエラー

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った
Illegal value : 指定した引数の値が不当である
Out of range value : 指定した値が有効範囲を超えている
Overflow : 指定した数値が -32768 ~ 32767 を超えている

□ 利用例

I2C EEPROM(AT24C256 スレーブアドレス \$50)にデータの読み書きを行う

```
100 POKE MEM+0,$00,$00
110 POKE MEM+2,64,65,66,67
120 R=I2CW($50,MEM,2,MEM+2,4)
130 ? "snd r=";R
140 POKE MEM+6,0,0,0,0
150 WAIT 5
160 R=I2CR($50,MEM,2,MEM+6,4)
170 ? "rcv r=";R
180 ? "rcv data:";
190 FOR I=0 TO 3
200 ? PEEK(MEM+6+I);" ";
210 NEXT I
220 ?
```

実行結果

```
run
snd r=0
rcv r=0
rcv data:64 65 66 67
OK
```

接続しているI2C スレーブを調べる

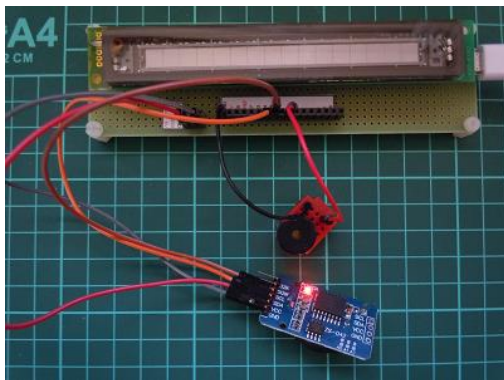
(補足) I2C スレーブアドレスのみ送信し、正常終了(ACK を返した)のアドレスを調べています。

```
10 FOR I=0 TO $7F
20 C=I2CW(I,MEM,0,MEM,0)
30 IF C=0 PRINT HEX$(I,2);" ";
40 NEXT I
50 PRINT :PRINT "done."
```

実行結果

```
run
57 68
done.
OK
```

実行結果は、接続しRTC モジュール (I2C EEPROM 搭載) のI2C アドレスを検出しました。



6.59 DATE 現在時刻の表示**□ 書式**

DATE

□ 引数

なし

□ 説明

I2C 接続 RTC DS3231 から現在の時刻を読み、その情報を画面に表示します。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った
I2C Device Error : RTC DS3231 が認識できない
Overflow : 指定した数値が-32768 ~ 32767 を超えている

□ 利用例

現在の時刻を表示する

```
DATE
2018/01/24 [Wed] 20:55:03
OK
```


6.60 GETDATE 日付の取得

□ 書式

GETDATE 年格納変数,月格納変数,日格納変数,曜日格納変数

□ 引数

年 格 納 変 数： 取得した西暦年を格納する変数を指定

月 格 納 変 数： 取得した月を格納する変数を指定

日 格 納 変 数： 取得した日を格納する変数を指定

曜 日 格 納 変 数： 取得した曜日コードを格納する変数を指定

□ 説明

I2C RTC DS3231 から日付情報を取得し、その値を指定した変数に格納します。

格納される値は次の通りです。

年 格 納 変 数： 西暦年 4 桁整数 2000 ~ 2099

月 格 納 変 数： 1 ~ 12

日 格 納 変 数： 1 ~ 31

曜 日 格 納 変 数： 曜日コード 1 ~ 7 (1:日 3:月 4:火 5:水 6:木 7:金 8:土)

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

I2C Device Error : RTC DS3231 が認識できない

Overflow : 指定した数値が-32768 ~ 32767 を超えている

□ 利用例

現在の日付を取得する

```
>GETDATE A,B,C,D
OK
>PRINT #-2,a,"/",b,"/",c
2018/01/20
OK
>
```

6.61 GETTIME 時刻の取得

□ 書式

GETTIME 時格納変数,分格納変数 秒格納変数

□ 引数

時格納変数： 取得した時を格納する変数を指定

分格納変数： 取得した分を格納する変数を指定

秒格納変数： 取得した秒を格納する変数を指定

□ 説明

I2C RTC DS3231 から時刻情報を取得し、その値を指定した変数に格納します。

格納される値を次の通りです。

時格納変数： 0 ～ 23 整数

分格納変数： 0 ～ 59 整数

秒格納変数： 0 ～ 59 整数

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

I2C Device Error : RTC DS3231 が認識できない

Overflow : 指定した数値が-32768 ～ 32767 を超えている

□ 利用例

現在の時間を取得する

```
>GETTIME A,B,C
OK
>PRINT #-2,a,":",b,":",c
12:03:32
OK
>
```

(補足) PRINT 文の #-2 は数値を 2 桁(0 付き)で表示する指定です。

6.62 SETDATE 時刻の設定

□ 書式

SETDATE 年,月,日, 曜日,時,分,秒

□ 引数

年 : 2000 ~ 2099 西暦年 4 桁の整数
 月 : 1 ~ 12 整数
 日 : 1 ~ 31 整数
 曜日 : 1:日、2:月、3:火、4:水、5:木、6:金、7:土
 時 : 0 ~ 23 整数
 分 : 0 ~ 59 整数
 秒 : 0 ~ 59 整数

□ 説明

指定した時刻を外部接続 I2C 接続 RTC DS3231 に設定します。

□ エラーメッセージ

Illegal value : 指定した引数の数値が有効範囲以外
 I2C Device Error : RTC DS3231 が認識できない
 Syntax error : 文法エラー、書式と異なる利用を行った
 Overflow : 指定した数値が-32768 ~ 32767 を超えている

□ 利用例

時刻の設定を行い、時刻表示を行う

```
1 'トイ
10 CLS
20 SETDATE 2018,3,17,6,16,40,0
30 LOCATE 0,0:DATE
40 WAIT 1000
50 GOTO 30
```

6.63 DATE\$ 現在時刻文字列の取得（文字列関数）**□ 書式**

DATE\$()

□ 引数

なし

□ 説明

I2C RTC DS3231 から現在の時刻を読み、その文字列を返します。

本関数は [PRINT](#)、[VMSG](#)、[CPRINT](#) の引数にて利用可能です。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った
I2C Device Error : RTC DS3231 が認識できない
Overflow : 指定した数値が-32768 ~ 32767 を超えている

□ 利用例

現在の時刻を表示する

```
?DATE$()  
2018/03/17 [Sat] 16:49:16  
OK  
>
```

6.64 CPRINT 有機 EL キャラクタディスプレイ文字列表示**□ 書式**

CPRINT

CPRINT 文字列値

CPRINT 文字列値; 文字列値;...

CPRINT #桁数,文字列値; 文字列値;... ;

※ ...は可変指定を示す

※ ; は連結指定、カンマ','も可能

□ 引数

文字列 : 文字列定数または文字列関数

値 : 数値定数、変数、配列変数、または数値関数、式

連結指定 : セミコロン'; または カンマ','

桁数 : #値の形式で指定する、値が正の場合、不足桁を空白埋め、負の場合 0 埋め

□ 説明

指定した文字列、値を I2C 接続有機 LE キャラクタディスプレイ SO1602AWWB に表示します。

文字列は文字列定数（""で囲った文字列）または、文字列関数の指定が可能です。

値には、定数（2 進数、10 進数、16 進数）、数値関数、式の指定が可能です。

引数の指定は、[PRINT](#) コマンドと同等ですが、末尾に";"の有無にかかわらず、改行は行われません。

引数の指定方法については、「6.14Print 画面への文字表示」を参照して下さい。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

I2C Device Error : SO1602AWWB が認識できない

Illegal value : 範囲外の値を指定した

Overflow : 指定した数値が-32768 ~ 32767 を超えている

□ 利用例

画面にメッセージを表示する

```

10 '有機 LE キャラクタディスプレイ サンプル
20 CCLS
30 CLOCATE 0,0:CPRINT "Hello,World"
40 CLOCATE 0,1:CPRINT "ねこ・ツ・パ・ツ・ワ"

```

6.65 CCLS 有機 EL キャラクタディスプレイ 表示内容消去**□ 書式**

CCLS

□ 引数

なし

□ 説明

I2C 接続有機 LE キャラクタディスプレイ SO1602AWWB の表示内容を消去します。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

I2C Device Error : SO1602AWWB が認識できない

Overflow : 指定した数値が-32768 ~ 32767 を超えている

□ 利用例

画面にメッセージを表示する

```
10 '有機 LE キャラクタディスプレイ サンプル
20 CCLS
30 CLOCATE 0,0:CPRINT "Hello,World"
40 CLOCATE 0,1:CPRINT "ねこ コ・ソ・バ・ソ・ワ"
50 WAIT 2000
60 CCLS
70 CLOCATE 0,0:CPRINT "トヨキ タイニー BASIC"
80 CLOCATE 0,1:CPRINT "for Arduino"
90 WAIT 2000
100 GOTO 20
```

6.66 CCURS 有機 EL キャラクタディスプレイ カーソル表示・非表示設定**□ 書式**

CCURS 表示設定

□ 引数

表示設定 : 0 非表示、1 表示 または OFF 非表示、ON 表示

□ 説明

I2C 接続有機 LE キャラクタディスプレイ SO1602AWWB のカーソルの表示・非表示の設定を行います。
デフォルトでは、カーソルは非表示の設定になっています。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った
 I2C Device Error : SO1602AWWB が認識できない
 Overflow : 指定した数値が-32768 ~ 32767 を超えている

□ 利用例

カーソルを表示する

```
>CCURS ON
OK
>
```

6.67 CLOCATE 有機 EL キャラクタディスプレイ カーソルの移動**□ 書式**

CLOCATE 横位置, 縦位置

□ 引数

横位置: 画面上の横位置 0 ~ 15

縦位置: 画面上の縦位置 0 ~ 1

□ 説明

I2C 接続有機 LE キャラクタディスプレイ SO1602AWWB のカーソルを指定した位置に移動します。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った
 I2C Device Error : SO1602AWWB が認識できない
 Illegal value : 範囲外の値を指定した
 Overflow : 指定した数値が-32768 ~ 32767 を超えている

□ 利用例

画面にメッセージを表示する

```

10 '有機 LE キャラクタディスプレイ サンプル
20 CCLS
30 CLOCATE 0,0:CPRINT "Hello,World"
40 CLOCATE 0,1:CPRINT "ネコ コ・ツバ・ツ・ワ"
50 WAIT 2000
60 CCLS
70 CLOCATE 0,0:CPRINT "トヨチ タイニー BASIC"
80 CLOCATE 0,1:CPRINT "for Arduino"
90 WAIT 2000
100 GOTO 20

```


6.68 CCONS 有機 EL キャラクタディスプレイ コントラスト設定**□ 書式**

CCONS コントラスト値

□ 引数

コントラスト値 : 0 ~ 255

□ 説明

I2C 接続有機 LE キャラクタディスプレイ SO1602AWWB のコントラスト（明るさ）の設定を行います。
0 が最低のコントラスト、255 が最大のコントラストになります。

□ エラーメッセージ

Syntax error	: 文法エラー、書式と異なる利用を行った
I2C Device Error	: SO1602AWWB が認識できない
Illegal value	: 範囲外の値を指定した
Overflow	: 指定した数値が-32768 ~ 32767 を超えている

□ 利用例

コントラストを最低の 0 に設定する

```
>CCONS 0
OK
>
```

6.69 CDISP 有機 EL キャラクタディスプレイ 表示・非表示設定**□ 書式**

CDISP 表示設定

□ 引数

表示設定 : 0 非表示、1 表示 または OFF 非表示、ON 表示

□ 説明

I2C 接続有機 LE キャラクタディスプレイ SO1602AWWB の表示・非表示の設定を行います。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った
 I2C Device Error : SO1602AWWB が認識できない
 Illegal value : 範囲外の値を指定した
 Overflow : 指定した数値が-32768 ~ 32767 を超えている

□ 利用例

表示メッセージを点滅表示する

```

10 '有機 LE キャラクタディスプレイ サンプル
20 CCLS
30 CLOCATE 0,0:CPRINT "Hello,World"
40 CLOCATE 0,1:CPRINT "ねこ じょうばん ちゃん"
50 FOR I=0 TO 5
60 CDISP I & 1
70 WAIT 400
80 NEXT I

```

6.70 IR 赤外線リモコン受信（数値関数）**□ 書式**

IR(ピン番号)

IR(ピン番号,リピート指定)

□ 引数

ピン番号 : 2 ~ 21（利用出来ないピンあり）、A0～A7

Arduino MEGA2560 利用の場合、ピン番号：2～69、A0～A15

リピート指定 : 0 リピートなし、1 リピートあり（デフォルト）

□ 戻り値

正常時 : 受信したコマンドコード：0～255

タイムオーバー：-1

データエラー : -2 または -3

□ 説明

赤外線受信モジュールにて、赤外線リモコンの送信データ受信し、その値を返します。

赤外線リモコンの信号形式は NEC 形式にのみ対応します。

NEC 形式では、受信データは4バイトの4バイトで構成されています。

カスタムコード : 2 バイト、赤外線リモコンの識別コード

コマンドコード : 1 バイト、ボタン毎にユニークなコード

コマンドコード反転 : 1 バイト、コマンドコードのビット反転データ

本関数では、上記のうちコマンドコードのみを返します。

リピート指定は、赤外線リモコンのボタンを押し続けた場合の挙動を指定します。

0：リピートなしの場合、ボタンを押し続けた場合、1 回だけコマンドコード返し、それ以降は-1 を返します。

1：リピートありの場合、ボタンを押し続けている間は、そのコマンドコードを返します。

リピート指定を省略した場合は、リピートありとなります。

赤外線 IR の読み取りは 100msec 間隔で行っています。この間に赤外線信号の受信で出来ない場合は、-1（タイムオーバー）を返します。データにエラーがある場合は-2 または-3 を返します。

□ エラーメッセージ

Syntax error : 文法エラー、書式と異なる利用を行った

Illegal value : 範囲外の値を指定した

Overflow : 指定した数値が-32768 ~ 32767 を超えている

□ 利用例

ポート 3 を使って赤外線受信モジュール TL1838 による赤外線リモコン受信を行う

```

10 '赤外線 IR
20 R=IR(3)
30 IF R>=0 ?R
40 GOTO 20

```

6.71 NINIT NeoPixel 初期化

□ 書式

NINIT バッファアドレス, ピクセル数

□ 引数

バッファアドレス, : 0 ~ 32767 (\$0000 ~ \$FFFF の有効な仮想アドレス領域)

ピクセル数 : 1 ~ 64

□ 説明

NeoPixel 利用のための初期化を行います。

バッファアドレスは、表示用の色情報を格納する領域の先頭アドレスを指定します。1 ピクセルにつき 1 バイト利用します。

64 個の場合は連続した 64 バイトの領域が必要となります。

ピクセル数は、NeoPixel の個数を指定します。

□ エラーメッセージ

□ 利用例

6.72 NBRIGHT NeoPixel 輝度設定

□ 書式

NBRIGHT 輝度

NBRIGHT 輝度,更新フラグ

□ 引数

輝度, : 0 ~ 5 (デフォルト 0)

更新フラグ : 0 更新なし 0 以外 : 更新あり (デフォルト 1)

□ 説明

NeoPixel の輝度を設定します。

輝度は数値が大きいほど明るくなります。デフォルトは0です。

更新フラグに 0 以外を指定した場合は、現在表示している内容に対して直ちに輝度を反映させます。

更新フラグに 0 を指定した場合は、次の更新のタイミングで指定した輝度を反映させます。

□ エラーメッセージ

□ 利用例

6.73 NCLS NeoPixel 表示クリア**□ 書式**

NCLS

NCLS 更新フラグ

□ 引数

更新フラグ : 0 更新なし 0 以外 : 更新あり (デフォルト 1)

□ 説明

NeoPixel の表示用バッファをクリアします。

デフォルトでは、NeoPixel の表示にも反映させて、表示内容をクリアします。

更新フラグに 0 を指定した場合、バッファのみをクリアし、表示には反映させません。

□ エラーメッセージ**□ 利用例**

6.74 NSET NeoPixel 指定ピクセルの色設定**□ 書式**

NSET ピクセル番号, 色コード

NSET ピクセル番号, 色コード, 更新フラグ

□ 引数

ピクセル番号 : 0 ~ 63

色コード : 0 ~ 255

更新フラグ : 0 更新なし 0 以外 : 更新あり (デフォルト 1)

□ 説明**□ エラーメッセージ****□ 利用例**

6.75 NPSET NeoPixel 指定座標色設定**□ 書式**

NPSET 横座標, 縦座標, 色コード

NPSET 横座標, 縦座標, 色コード, 更新フラグ

□ 引数

横座標 : 0 ~ 7

縦座標 : 0 ~ 7

色コード : 0 ~ 255

更新フラグ : 0 更新なし 0 以外 : 更新あり (デフォルト 1)

□ 説明**□ エラーメッセージ****□ 利用例**

6.76 NMSG NeoPixel メッセージ文表示**□ 書式**

NMSG 色コード, 速度, メッセージ文

□ 引数

色コード	: 0 ~ 255
速度	: 0 ~ 5000
メッセージ文	: 文字列定数および文字列関数を含む文字列

□ 説明**□ エラーメッセージ****□ 利用例**

6.77 NSCROLL NeoPixel スクロール

□ 書式

NSCROLL 方向

NSCROLL 方向, 更新フラグ

□ 引数

方向 : 0 = 左、1 = 右、2 = 上、3 = 下

更新フラグ : 0 更新なし 0 以外 : 更新あり (デフォルト 1)

□ 説明

□ エラーメッセージ

□ 利用例

6.78 NLINE NeoPixel 直線描画**□ 書式**

NLINE X1, Y1, X2, Y2, 色コード

NLINE X1, Y1, X2, Y2, 色コード ,モード

NLINE X1, Y1, X2, Y2, 色コード, モード ,更新フラグ

□ 引数

X1, Y1 : 0 ～ 7、始点座標

X2, Y2 : 0 ～ 7、終点座標

色コード : 0 ～ 255

モード : 0 直線、1 矩形 2 塗りつぶし四角

更新フラグ : 0 更新なし 0 以外：更新あり（デフォルト 1）

□ 説明**□ エラーメッセージ****□ 利用例**

6.79 NUPDATE NeoPixel バッファ表示反映

□ 書式

NUPDATE

□ 引数

なし

□ 説明

□ エラーメッセージ

□ 利用例

6.80 NSHIFT NeoPixel ピクセルのシフト

□ 書式

NSHIFT 方向

NSHIFT 方向, 更新フラグ

□ 引数

方向

:

更新フラグ

: 0 更新なし 0 以外: 更新あり (デフォルト 1)

□ 説明

□ エラーメッセージ

□ 利用例

6.81 NPOINT NeoPixel 指定座標の色コード取得（数値関数）**□ 書式**

NPOINT(横座標, 縦座標)

□ 引数

横座標	: 0 ~ 7
縦座標	: 0 ~ 7

□ 戻り値

指定座標のが有効範囲内の場合 色コード 0 ~ 255

指定座標のが有効範囲外の場合 -1

□ 説明

8x8 ドット構成の NeoPixel モジュールにおいて、指定した座標の色コードを取得します。

指定した座標が有効範囲外の場合は、-1 を返します。

□ エラーメッセージ**□ 利用例**

6.82 RGB NeoPixel 256 色コードの作成（数値関数）**□ 書式**

RGB(赤, 緑, 青)

□ 引数

赤	: 0 ~ 7
緑	: 0 ~ 7
青	: 0 ~ 3

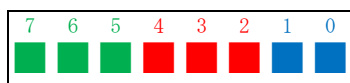
□ 戻り値

指定座標のが有効範囲内の場合 色コード 0 ~ 255

□ 説明

赤、緑、青の色成分から256色コード（8ビット）を作成し、その値を返します。

8ビット 256色コードは次の構成となります（下記の数値はビット番号）。



緑と赤が3ビット、青が2ビットの情報量であることに注意して下さい。

□ エラーメッセージ**□ 利用例**

