別冊『システム変数 と システム関数』

10 ． システム変数とシステム関数とは

用語の説明：

- システム変数とは、事前に組み込み定義されている global変数 です。
- システム関数とは、事前に組み込み定義されている global関数 です。
  → システム変数もシステム関数も、その値や内容を自由に変更することができます。

（ a ） 記号の説明

このマニュアルで使用する記号です。

- S,I,D,P,X,A → データ型を表します。
- FILE → ファイル名を表す文字列、又は、ファイルポインタ を表します。
- /RE/ → 正規表現 を表す文字列、又は、コンパイル済み正規表現を表します。
- ＊ → 任意のデータ型を表します。
- [X] → X が省略可能であることを表します。（配列記号とは文脈で区別します。）
- X|Y → X 又は Y が選択可能であることを表します。

（ b ） 共通の性質

「システム関数」には、次の共通した性質があります。

- 常に非破壊的に動作します。（入力データを破壊しません。）
- 常に全 自 動で動作します。（メモリ管理やファイル管理は、自動で行います。）
- エラー戻り値は NULL で統一されています。

11 ． 基本パラメーター

＜システム変数＞

（ａ） 基本の定数

名前	型	意味	備考
NULL	P	ポインタ NULL	関数のエラー戻り値
TRUE	I	真の値（実体は整数１）	
FALS	I	偽の値（実体は整数０）	
VER	I	インタプリタのバージョン番号	= こすちゅーむ番号

（ｂ） データのビット幅

名前	型	意味	備考
INT_SIZE	I	整数のビット数	例：64 [bit]
DBL_SIZE	I	実数のビット数 [= DBL_MANTSIZE + DBL_EXPOSIZE + 1]	例：64 [bit]
PTR_SIZE	I	ポインタのビット数	例：64 [bit]
DBL_MANTSIZE	I	実数の仮数部のビット数	例：52 [bit]
DBL_EXPOSIZE	I	実数の指数部のビット数	例：11 [bit]

＜システム関数＞

（ｃ） 型の種別と型の判別

名前	説明
I = type (*)	引数の型を1文字で返します。 戻り値={'U'|'S'|'I'|'D'|'P'|'X'|'A'}
I = isnull(*)|istrue(*)|isfals(*)| isstr (*)|isint (*)|isdbl (*)| isptr (*)|isfunc(*)|isarry(*)|	引数が {NULL | TRUE | FALS | 文字列 | 整数 | 実数 | ポインタ | 関数 |配列} であるかどうか判定します。 戻り値={TRUE|FALS}

（ｄ） 定義の判定と消去（未定義化）

名前	説明
I = isdef(*)	引数が定義済みであるかどうか判定します。 戻り値={TRUE|FALS}
U = erase(*)	引数を消去（未定義化）します。 戻り値=無し

12 . コマンドライン引数

＜システム変数＞

(a) argc/argv/env など

名前	型	意味	備考
argc	I	コマンドライン引数の数　（コマンド名も含む）	
argv	A	コマンドライン引数の配列（コマンド名も含む）	
argr	I	現時点でshift()可能なコマンドライン引数の残数	自動更新
args	S	argv[1] からargv[argc-1]まで連結した文字列	
cmd	S	コマンド名の文字列	= argv[0]
env	A	環境変数の配列※	
※ 例えば、環境変数が PATH=/usr/bin であった場合は、env["PATH"]="/usr/bin" となります。			

＜システム関数＞

(b) コマンドライン引数の取得

名前	説明
S = shift　([*])	コマンドライン引数を、呼び出し毎に argv[1] から順に取得する
S = opshift([*])	（ 〃 ） ただし、対応する引数が option の場合にのみ取得する
I = unshift()	次の取得位置を１つ前に戻す（戻り値＝取得位置）
※ shift() 及び opshift() は、取得するコマンドライン引数が無い場合は、指定されたパラメータ * （又は、その指定も無い場合 NULL ）が戻り値となります。 ※ オプションとは '-' 文字で始まるコマンドライン引数のことです。	

13 . ファイル入出力

＜システム変数＞

(a) ファイルポインタ

名前	型	意味	備考
stdin	P	標準入力	
stdout	P	標準出力	
stderr	P	標準エラー出力	
:code	P	コード領域アクセス用	= __code__
:data	P	データ領域アクセス用	= __data__
※ :code 及び :data は、通常ファイルに記述されたスクリプト内でのみアクセス可能です。			

＜システム関数＞

(b) １文字、又は、１行の入出力

名前	説明
I = getc ([FILE]) S = gets ([FILE]) S = getn ([FILE,] Max)	指定されたファイルから、1文字、又は、1行を読み込みます。 (FILE 省略時は、標準入力を対象とします。)
I = putc ([FILE,] Chr) I = puts ([FILE,] Str) I = putn ([FILE,] Str,Max)	指定されたファイルへ、 1文字、又は、1行を書き込みます。 (FILE 省略時は、標準出力を対象とします。)
I = ungetc([FILE,] Chr) I = ungets([FILE,] Str) I = ungetn([FILE,] Str,Max)	指定されたファイルへ、 1文字、又は、1行を戻します。 (FILE 省略時は、標準入力を対象とします。)
※ 文字はChr(I)、文字列はStr(S)、最大文字数はMax(I)で指定します。	

(c) バイナリーデータの入出力

名前	説明
(Ptr,Cnt) = read ([FILE,] Cnt)	指定バイト数の読み込み。 (FILE 省略時は、標準入力を対象とします。)
Cnt = write([FILE,] Ptr,Cnt)	指定バイト数の書き込み。 (FILE 省略時は、標準出力を対象とします。)
※ Cnt(I)は、要求バイト数と実行バイト数です。又、Ptr(P)は、対象データの保存場所アドレスです。	

（ｄ） 各種制御関数

名前	説明
P = open （FILE,Mode）	ファイルの明示的オープンです。（通常は不要です。）
I = close （FILE ）	ファイルの明示的クローズです。（通常は不要です。）
I = flush （FILE ）	ファイルバッファのフラッシュです。（NULL指定=全ファイル）
I = getpos(FILE ）	ファイル位置の取得です。（現在の位置、先頭=00 ）
I = setpos(FILE,I)	ファイル位置の設定です。（絶対値指定、先頭=00／末尾=-1）
I = movpos(FILE,I)	ファイル位置の移動です。（相対値指定、進行=正／後退=負）
I = rewind(FILE ）	ファイル先頭位置(00)への移動です。［ = setpos(FILE,0) ］
※ Mode(S) は、Ｃ言語 fopen() 互換のモード文字列です。	

14 ． データ表示（印刷出力）

＜システム変数＞

（ａ） 色（ ANSI Color ）

名前	型	意味	備考
{C|U|B|R}_BLA	S	黒色 {属性なし | 下線付き | ブリンク | リバース}	Bold
{C|U|B|R}_RED	S	赤色 {属性なし | 下線付き | ブリンク | リバース}	Bold
{C|U|B|R}_GRE	S	緑色 {属性なし | 下線付き | ブリンク | リバース}	Bold
{C|U|B|R}_YEL	S	黄色 {属性なし | 下線付き | ブリンク | リバース}	Bold
{C|U|B|R}_BLU	S	青色 {属性なし | 下線付き | ブリンク | リバース}	Bold
{C|U|B|R}_MAG	S	紫色 {属性なし | 下線付き | ブリンク | リバース}	Bold
{C|U|B|R}_CYA	S	水色 {属性なし | 下線付き | ブリンク | リバース}	Bold
{C|U|B|R}_WHI	S	白色 {属性なし | 下線付き | ブリンク | リバース}	Bold
{C|U|B|R}_DEF	S	デフォルト色に戻る。（４種類とも、全て同じ意味）	
※ 色の種類は、全部で８色あります。黒色(BLA)、赤色(RED)、緑色(GRE)、黄色(YEL)、青色(BLU)、紫色(MAG)、水色(CYA)、白色(WHI)。又、色の属性は、それぞれ４種類あります。属性なし(C_)、下線付き(U_)、ブリンク(B_)、リバース(R_)。 ※ 文字列に色の付ける場合は、書式付き出力関数（print()関数ファミリー）を利用するのが便利です。この場合、出力先の属性に応じて色付き又は色なしが自動で制御されます。			

（ｂ） 文字列の変換書式

名前	型	意味	備考
FMT_S2S	S	文字列　→文字列 への変換書式	初期値 = ”%s”
FMT_I2S	S	整数型　→文字列 への変換書式	初期値 = ”%d”
FMT_D2S	S	実数型　→文字列 への変換書式	初期値 = ”%+f”
FMT_P2S	S	ポインタ→文字列 への変換書式（NULL以外）	初期値 = ”%p”
FMT_N2S	S	NULL　→文字列 への変換書式（NULLのみ）	初期値 = ”NULL”
※ 全てデフォルトの変換書式です。 p() 関数や、”${変数}” 構文による変数展開などで利用されます。			

（ｃ） 簡易表示と詳細表示

システム関数	説明
void = p(*, ...)	変数、又は、式の値を簡易表示します。
void = d(*, ...)	変数、又は、式の値を詳細表示します。　← ダンプ表示
x　← 変数名のみを記述した場合。	変数 x の値を簡易表示します。　← p(x) に同じ
※ 詳細表示では、識別ＩＤ｛スコープ種別(G=Global|S=Static|L=Local)＋識別番号｝、名前、データ型、データ属性、データ値、といった内部管理情報を表示します。	

（ｄ） 書式付き出力

システム関数	説明
I = print (FMT,...)	標準　　出力用の print 文です。
I = eprint(FMT,...)	標準エラー出力用の print 文です。 = fprint(stderr,FMT,...)
I = fprint(FILE,FMT,...)	ファイル　出力用の print 文です。
S = sprint(FMT,...)	文字列　　出力用の print 文です。
void = dying(FMT,...)	この関数は { eprint(FMT,...); exit(1); } と等価です。
※ FMT(S) は、Ｃ言語 printf() 互換の書式文字列です。拡張機能として2進数出力 %b %B、及び、数値出力（整数又は実数を、その型に応じて %d %f %e 又は %E を自動選択する）%v %V が指定できます。	

15 ． 文字列操作

＜システム関数＞

（ａ） 文字列の比較

システム関数	説明
I = strcmp (S1,S2) I = strcmpi(S1,S2)	文字列 S1 と S2 の大小比較です。
I = strncmp (S1,S2,I) I = strncmpi(S1,S2,I)	文字列 S1 と S2 の大小比較です。（最大 I 文字まで）
I = startwith(S1,S2) I = endwith(S1,S2)	文字列 S1 が S2 で開始するか検査する。戻り値={TRUE|FALS} 文字列 S1 が S2 で終了するか検査する。戻り値={TRUE|FALS}
※ 戻り値は、Ｃ言語互換です。又、i 付き関数は Ignore Case 版（大文字小文字の区別なし）です。	

（ｂ） 文字列の比較（先頭と末尾）

システム関数	説明
I = startwith(S1,S2)	文字列 S1 の先頭が S2 で開始するか検査する。 戻り値={TRUE|FALS}
I = endwith(S1,S2)	文字列 S1 の末尾が S2 で終了するか検査する。 戻り値={TRUE|FALS}

（ｃ） 文字列の検索

システム関数	説明
I = strchr(S,Chr) | strrchr(S,Chr)	文字　Chr の｛順方向|逆方向｝検索です。
I = strstr(S,Str) | strrstr(S,Str)	文字列 Str の｛順方向|逆方向｝検索です。
(Is,Ie) = strreg(S,/RE/)	正規表現 /RE/ のマッチ位置検索です。｛Is=始点、Ie=終点｝
※ いずれの関数も、文字列 S 内の位置を戻り値とします。（先頭＝０）	

（ｄ） 文字列の置換

システム関数	説明
S = ssub(S,V1,V2) | gsup(S,V1,V2)	文字列 S 内の V1 を V2 に｛１回|全部｝置換します。
S = evalesc(S)	ESCシーケンスを順方向変換します。（例：”¥n”→0x0a）
S = rvalesc(S)	ESCシーケンスを逆方向変換します。（例：0x0a→”¥n”）
S|I = uc(S|I) | lc(S|I)	引数を｛大文字化|小文字化｝します。
※ パラメータ V1 には、文字、文字列、又は、正規表現が指定できます。 ※ パラメータ V2 には、文字、文字列　　が指定できます。	

（ ｅ ） 改行等の削除

システム関数	説明
S = chop (S)	{ 行末 } の改行コードを１組削除します。
S = hstrip(S)| tstrip(S)| strip(S)	{行頭 | 行末 | 両方} の空白コードを全部削除します。
※ 改行コードは、Win/Mac/Unix 形式に対応します。空白コードは、isspace() で判定します。	

（ ｆ ） 部分文字列等

システム関数	説明
I = subchr(S,Ic)	文字列 S の位置 Ic の文字を戻します。 (= S[Ic])
S = substr(S,Is,Ie)	文字列 S の位置 Is～Ie の部分文字列を戻します。
※ 位置の指定方法（先頭から指定する場合） : 先頭 = 0, 1, 2, … ※ 位置の指定方法（末尾から指定する場合） : 末尾 = -1,-2,-3, …	

（ ｇ ） 文字列の配列化

システム関数	説明
A = split(S,V [,I])	文字列 S を V で区切った各要素で配列化します。
A = psplit(S,V [,I])	(〃) ただし、空要素はスキップします。 (Packed形式)
A = scan (S,V [,I])	文字列 S を V でマッチさせた各要素で配列化します。
A = pscan (S,V [,I])	(〃) ただし、空要素はスキップします。 (Packed形式)
※ いずれの関数も、パラメータ V には、文字、文字列、又は、正規表現が指定できます。又、戻り値 A は、新規に生成された１次元配列です。（添字は０～） 第３パラメータ(I)指定時は、その添字値に対応する要素(S)のみを戻します。	

16 . データ操作

＜システム関数＞

(a) データの変換

システム関数	説明
I = int (*)	パラメータを整数値化します。【 別名 atoi() 】
D = dbl (*)	パラメータを実数値化します。【 別名 atof() 】
I|D = atov(*)	パラメータを数　値化します。（戻り値の型は自動判定）
S = str (*)	パラメータを文字列化します。
P = ptr (*)	パラメータをポインタ化します。
I = bin(S)| oct(S)| dec(S)| hex(S)	文字列 S を{2|8|10|16}進数と見なして整数化します。
/RE/ = reg(S)	文字列 S の正規表現をコンパイルします。【 別名 regexp() 】

(b) データの判定

システム関数	説明
I = isalnum(I) | isalpha(I) | isascii(I) | isblank(I) | iscntrl(I) | isdigit(I) | isgraph(I) | islower(I) | isprint(I) | ispunct(I) | isspace(I) | isupper(I) | isxdigit(I)	Ｃ言語互換の文字種別判定関数です。　　戻り値={TRUE|FALS}

(c) データの消去（未定義化）

システム関数	説明
U = erase(*)	定義済み変数（未定義化）を消去します。

(d) データの複製

システム関数	説明
* = dup(*)	パラメータを複製します。なお、配列又は（静的変数を持つ）関数以外では、代入文と同じ効果となります。

(e) サイズの計測

システム関数	説明
I = size(*)	パラメータのサイズを求めます。
I = strlen(S)	文字列 S のサイズ（長さ）を求めます。
※ 関数 size() の戻り値は、文字列→長さ（バイト数）、整数型＆実数型＆ポインタ→表現ビット数、関数型→演算ノード数、配列型→要素数となります。	

（ｆ） ソートの実行

システム関数	説明
A = sort([X,] V,...)	パラメータ V を昇順にソートします（戻り値＝ソート済み配列）
A = rsort([X,] V,...)	パラメータ V を降順にソートします（戻り値＝ソート済み配列）
※ 比較関数 X を指定しない場合は、パラメータの単純ソートとなります。この場合、パラメータ V は、｛文字列｜整数｜実数｝又はそれらを含む配列が指定できます。比較関数 X を指定した場合は、比較関数による一般ソートとなります。この場合、パラメータ V は自由に設定出来ます。なお、比較関数 X は、２つの引数を取り比較結果を数値の符号で戻す関数となります。（Ｃ言語 qsort(3) の比較関数互換。）	

（ｇ） 配列化と配列値

システム関数	説明
Ao = vals(Ai) | rvals(Ai)	配列 Ai の各要素値を要素とする、１次配列 Ao を生成します
Ao = keys(Ai) | rkeys(Ai)	配列 Ai の各添字値を要素とする、１次配列 Ao を生成します
Ao = tags(Ai) | rtags(Ai)	配列 Ai の添字＋要素を要素とする、１次配列 Ao を生成します
A = append(A,V)	配列 A の最後に値 V を追加します。
I = ndim (Ai)	（ハッシュ要素を含まない）配列 Ai の次元数を求めます
Ao = shape(Ai)	（ハッシュ要素を含まない）配列 Ai の形　状を求めます
関数 vals(),keys(),tags() は入力配列 Ai を昇順にサーチして出力配列 Ao の生成処理を行います。 一方、関数 rvals(),rkeys(),rtags() は降順にサーチして生成処理を行います。	

17．ネットワーク

（ａ） ＵＤＰ

＜典型的な処理フロー＞

1、ソケットの作成とバインド

→ 通常は省略可能です。省略時にはソケットが必要になったタイミングで（正確には、最初のＵＤＰパケット送信時に）自動で作成され、IPアドレス＆ポート番号も自動で設定されます。

→ ただし、サーバー用プログラムなど待ち受けIPアドレス＆ポート番号を指定したい場合には、事前にソケットの作成（とバインド）をして下さい。

```
sock = udp_socket( IPアドレス , ポート番号 )
```

2、回線の接続

→ UDPなので回線の接続は不要です。

3、送受信の実行

→ 送信側は、送信先のIPアドレス＆ポート番号と送信データを指定して送信します。

```
len = tx_udp( ip , port , ”DATA” )
```

→ 受信側は、送信元のIPアドレス＆ポート番号と受信データが得られます。

```
( ip , port , buf ) = rx_udp()
```

4、その他（デフォルトソケット値について）

→ UDPの送受信関数は、ソケット値が指定された場合はその値を利用しますが、指定が省略された場合は UDP_SOCKET の値（デフォルトソケット値）を利用します。

＜サンプルプログラム＞

【例cl】UDPのechoクライアントを構築します。具体的には、ローカルホスト上のUDPエコーサーバーに挨拶メッセージを送信して、サーバーからの送信（返答）メッセージを受信して表示します。

```
ip   = 127.0.0.1                          # 送信先のIPアドレス
port = 7                                  # 送信先のポート番号（echo）

tx_udp( ip, port, "Hello, UDP!!" )        # UDPで挨拶メッセージを送信します。
( ip, port, buf ) = rx_udp()              # UDPで返答メッセージを受信します。

print("[%s]¥n",buf)                       # 受信したメッセージを表示します。
```

★解説：関数 tx_udp() は、ソケット値が省略されているため UDP_SOCKET の値を利用します。
しかし、UDP_SOCKET は初期値０（無効）であるため、利用直前にUDPソケットが自動で作成され UDP_SOCKET に設定されます。（なお、関数 tx_udp() は設定後の値を利用します。）

関数 rx_udp() は、ソケット値が省略されているため UDP_SOCKET の値を利用します。

【例sv】UDPのechoサーバーを構築します。具体的には、UDPのポート７番（echo）で待ち受け受信をして、そこで受信したデータをそのまま送信元に送信（返答）します。

```
sock = udp_socket( 0, "echo" )            # ポート番号を指定してソケットを作成します。

while( TRUE ){
       ( ip, port, buf ) = rx_udp()       # UDPで受信します。
       tx_udp( ip, port, buf )            # UDPで送信（返答）します。
}
```

★解説：待ち受けポート番号を指定するために、関数 udp_socket() を実行します。
なお、作成されたUDPソケットの値 sock は、UDP_SOCKET にも自動で設定されます。

関数 rx_udp() は、ソケット値が省略されているため UDP_SOCKET の値を利用します。
関数 tx_udp() も、同様です。

【実行】上記の例示コードをテキストファイル "sv" と "cl" に保存して実行します。
なお、エコーサーバーの動作を確認するためクライアントを3回連続で実行します。

```
% tt sv &
[1] 1234

% tt cl ; tt cl ; tt cl
[Hello, UDP!!]
[Hello, UDP!!]
[Hello, UDP!!]
```

＜システム変数＞

ＵＤＰデフォルトソケット値：

名前	型	意味	備考
UDP_SOCKET	I	UDP関連関数が、UDPソケットの指定省略時に使用するUDPソケット番号です。初期値は０（無効）です。	自動設定＆自動更新 ※
※ UDP_SOCKET が０（無効）の時に、UDP関連関数が（ソケット指定を省略して）実行されると、新しいUDPソケットが作成されて自動で値が設定（更新）されます。あるいは、UDPソケット作成関数が実行されると、その都度、新しいUDPソケットが作成されて自動で値が設定（更新）されます。			

ＵＤＰタイムアウト値：

名前	型	意味	備考
UDP_TIMEOUT	D	UDP関連関数のタイムアウト値です。初期値は INF（無限ブロッキング）です。	単位［秒］ ※
※ UDP_TIMEOUT の値に、例えば３を設定すると３秒後にタイムアウトします。また、０を設定するとノンブロッキング動作となります。			

＜システム関数＞

ＵＤＰソケットの作成：

名前	説明
sock = udp_socket([ip,port])	指定された ip(I|S) & port(I|S) にて、UDP送受信用のソケットを作成してバインドします。なお、数値０を指定すると自動設定となります。 戻り値は、作成されたソケット番号 sock(I) です。
※ 作成されたソケット番号は、デフォルトソケット UDP_SOCKET にも設定されます。	

ＵＤＰパケットの送信と受信：

名前	説明
len = tx_udp([sock,]ip,port,buf[,len])	ソケット sock(I) を利用して、送信先 ip(I|S) & port(I|S) に、buf(S) のデータを len(I) バイト、UDPパケットで送信します。なお、len(I) 省略時は buf(S) の全データが送信されます。 戻り値は、送信したUDPデータ部のバイト数 len(I) です。 ※
(ip,port,buf,len) = rx_udp([sock])	ソケット sock(I) を利用して、自分宛のUDPパケットを受信します。 戻り値は、送信元 ip(I) & port(I)、受信したUPDデータ buf(S)、及び、そのバイト数 len(I) です。 ※
※ ソケット sock 省略時には、デフォルトソケット UDP_SOCKET が使用されます。又、バイト数 len は、UDPペイロードデータのバイト数です。	

（ｂ） ＴＣＰ

＜典型的な処理フロー＞

1、ソケットの作成とバインド

→ 通常は省略可能です。省略時にはソケットが必要になったタイミングで自動で作成され、IPアドレス＆ポート番号も自動で設定されます。

→ ただし、サーバー用プログラムなど待ち受けIPアドレス＆ポート番号を指定したい場合には、事前にソケットの作成（とバインド）をして下さい。

```
sock = tcp_socket( IPアドレス , ポート番号 )
```

2、回線の接続

→ サーバー側は listen() で待ち受けします。

```
sock = listen ()
```

→ クライアント側は、接続先のIPアドレス＆ポート番号を指定して connect() で接続します。

```
sock = connect( IPアドレス , ポート番号 )
```

3、送受信の実行

→ 送信側は、送信データを指定して送信します。

```
tx_tcp( ”DATA” )
```

→ 受信側は、受信データが得られます。

```
buf = rx_tcp()
```

4、その他（デフォルトソケット値について）

→ TCPの送受信関数は、ソケット値が指定された場合はその値を利用しますが、指定が省略された場合は TCP_SOCKET の値（デフォルトソケット値）を利用します。

＜サンプルプログラム＞

【例cl】TCPのechoクライアントを構築します。具体的には、ローカルホスト上のTCPエコーサーバーに挨拶メッセージを送信して、サーバーからの送信（返答）メッセージを受信して表示します。

```
ip   = 127.0.0.1                    # 送信先のIPアドレス
port = 7                            # 送信先のポート番号（echo）

connect(ip,port)                    # TCPサーバーに接続します。

tx_tcp( "Hello, TCP!!" )            # TCPで挨拶メッセージを送信します。
buf = rx_tcp()                      # TCPで返答メッセージを受信します。

print("[%s]¥n",buf)                 # 受信したメッセージを表示します。
```

★解説：関数 connect() は、ソケット値が省略されているため TCP_SOCKET の値を利用します。
しかし、TCP_SOCKET は初期値０（無効）であるため、利用直前にTCPソケットが自動で作成され TCP_SOCKET に設定されます。（なお、関数 connect() は設定後の値を利用します。）

関数 rx_tcp() 、ソケット値が省略されているため TCP_SOCKET の値を利用します。
関数 tx_tcp() も、同様です。

【例sv】TCPのechoサーバーを構築します。具体的には、TCPのポート７番（echo）で待ち受け受信をして、そこで受信したデータをそのまま送り元に送信（返答）します。

```
sock = tcp_socket( 0, "echo" )      # ポート番号を指定してソケットを作成します。
while( tmp = listen(sock) ){        # クライアントからの接続要求をリスンします。
       buf = rx_tcp()               # TCPで受信します。
       tx_tcp(buf)                  # TCPで送信（返答）します。
}
```

★解説：待ち受けポート番号を指定するために、関数 tcp_socket() を実行します。
なお、作成されたTCPソケットの値 sock は、TCP_SOCKET にも自動で設定されます。

関数 listen() は、ソケット sock にて接続要求を待ち受けます。クライアントからの接続要求が到着してTCP回線の接続が確立すると、接続済みの新しいソケットが生成され、その値を戻り値とします。また、この新しいソケット値 tmp は、TCP_SOCKET にも自動で設定されます。

関数 rx_tcp() は、ソケット値が省略されているため TCP_SOCKET の値を利用します。
関数 tx_tcp() も、同様です。

【実行】上記の例示コードをテキストファイル "sv" と "cl" に保存して実行します。
なお、サーバーの動作を確認するためクライアントを3回連続で実行します。

```
% tt sv &
[1] 1234

% tt cl ; tt cl ; tt cl
[Hello, TCP!!]
[Hello, TCP!!]
[Hello, TCP!!]
```

＜システム変数＞

ＴＣＰデフォルトソケット値：

名前	型	意味	備考
TCP_SOCKET	I	TCP関連関数が、TCPソケットの指定省略時に使用するTCPソケット番号です。初期値は０（無効）です。	自動設定＆自動更新 ※
※ TCP_SOCKET が０（無効）の時に、TCP関連関数が（ソケット指定を省略して）実行されると、新しいTCPソケットが作成されて自動で値が設定（更新）されます。あるいは、TCPソケット作成関数が実行されると、その都度、新しいTCPソケットが作成されて自動で値が設定（更新）されます。			

ＴＣＰタイムアウト値：

名前	型	意味	備考
TCP_TIMEOUT	D	TCP関連関数のタイムアウト値です。初期値は INF（無限ブロッキング）です。	単位［秒］ ※
※ TCP_TIMEOUT の値に、例えば３を設定すると３秒後にタイムアウトします。また、０を設定するとノンブロッキング動作となります。			

＜システム関数＞

ＴＣＰソケットの作成：

名前	説明
sock = tcp_socket([ip,port])	指定された ip(I|S) & port(I|S) にて、TCP送受信用のソケットを作成してバインドします。なお、数値０を指定すると自動設定となります。 戻り値は作成されたソケット番号 sock(I) です。
※ 作成されたソケット番号は、デフォルトソケット TCP_SOCKET にも設定されます。	

ＴＣＰ回線の接続：

名前	説明
sock = listen([sock])	ソケット sock(I) を利用して、TCP回線の接続を待ち受けします。 戻り値は、新たに生成された接続済みのソケット番号 sock(I) です。 ※1
sock = connect([sock,]ip,port)	ソケット sock(I) を利用して、指定された ip(I|S) & port(I|S) に、TCP回線の接続を行います。 戻り値は、接続済みのソケット番号 sock(I) です。 ※2
※1 ソケット sock 省略時には、前回リンスしたソケットが再び使用されます。もし、それが無い場合は、デフォルトソケット TCP_SOCKERT が使用されます。 ※2 ソケット sock 省略時には、デフォルトソケット TCP_SOCKET が使用されます。	

ＴＣＰパケットの送信と受信：

名前	説明
len = tx_tcp([sock,]buf[,len])	ソケット sock(I) を利用して、接続先に buf(S) のデータを len(I) バイト、TCPパケットで送信します。なお、len(I) 省略時は buf(S) の全データが送信されます。 戻り値は送信したTCPデータ部のバイト数 len(I) です。 ※
(buf,len) = rx_tcp([sock])	ソケット sock(I) を利用して、自分宛のTCPパケットを受信します。 戻り値は、受信したTCPデータ buf(S)、及び、そのバイト数 len(I) です。 ※
※ ソケット sock 省略時には、デフォルトソケット TCP_SOCKET が使用されます。又、バイト数 len は、TCPペイロードデータのバイト数です。	

（ｃ） ＲＡＷ（ＰＩＮＧ）

＜典型的な処理フロー＞

1、ソケットの作成とバインド

→ 省略可能です。省略時はソケットが必要になったタイミングで自動で作成され、アドレスも自動で設定されます。

```
sock = raw_socket(アドレス)
```

3、ping() の実行

→ 送信先のアドレスを指定して ping() を実行します。

```
ping( ip )
```

4、その他（ソケットの指定について）

→ 省略可能です。ping() 関数は、ソケット値が指定された場合はその値を利用しますが、指定が省略された場合は RAW_SOCKET の値（デフォルトソケット値）を利用します。なお RAW_SOCKET は、最後に作成されたRAWソケットの値が自動で設定＆更新されます。

＜サンプルプログラム＞

【例】www.google.com に ping します。

```
ip  = "www.google.com"                  # 送信先を設定します。
ret = ping( ip )                        # ping を実行します。

print("%f [ms]¥n",ret*1000)             # 往復時間を表示します。
```

【実行】例示コードをテキストファイル "cl" に保存して実行します。

```
% tt cl
9.119000 [ms]
```

＜システム変数＞

RAW（PING）関連のパラメーター：

名前	型	意味	備考
RAW_SOCKET	I	RAWソケット省略時に使用するRAWソケット番号	自動設定＆自動更新
RAW_TIMEOUT	D	RAW送受信関数のタイムアウト値	単位［秒］　※
※ RAW_TIMEOUT の初期値は INF（無限ブロッキング）です。例えば、３を設定すると３秒後にタイムアウトします。また、０を設定するとノンブロッキング動作となります。			

＜システム関数＞

RAWソケットの作成：

名前	説明
sock = raw_socket([ip])	指定された ip(I|S) にて、RAW送受信用のソケットを作成します。 なお、数値0を指定すると、自動設定となります。 戻り値は作成されたソケット番号 sock(I) です。
※ 作成されたソケット番号は、デフォルトソケット RAW_SOCKET(I) に設定されます。	

PING送信とPING受信：

名前	説明
ret = ping([sock,]ip)	指定された ip(I|S) に ICMP ECHO を送信し、戻って来た ICMP ECHOREPLY を受信します。 戻り値は、往復時間 ret(D) です。[単位=秒]
※ ソケット sock(I) 省略時には、デフォルトソケット UDP_SOCKET(I) が使用されます。	

（ｄ） その他

＜システム関数＞

ソケットの情報：

名前	説明
(l_ip, l_port, l_type, r_ip, r_port, r_type) = sockinfo (sock)	ソケット sock(I) の以下の情報を戻り値とします。※ ローカル側 の IPアドレス(I)、port(I)、type(I)。 リモート側 の IPアドレス(I)、port(I)、type(I)。
(l_ip, l_port, l_type) = lsockinfo(sock) = lsock (sock)	ソケット sock(I) の以下の情報を戻り値とします。※ ローカル側 の IPアドレス(I)、port(I)、type(I)。
(r_ip, r_port, r_type) = rsockinfo(sock) = rsock (sock)	ソケット sock(I) の以下の情報を戻り値とします。※ リモート側 の IPアドレス(I)、port(I)、type(I)。
※ type は、ソケットタイプを表す次の１文字です。{ 'R'=RAW | 'U'=UDP | 'T'=TCP }	

IPアドレスとポート番号：

名前	説明
S = ip2str(I|S)	IPアドレス又はホスト名(I|S)を文字列に変換します。
I = ip2int(I|S)	IPアドレス又はホスト名(I|S)を整数値に変換します。
S = port2str(Proto,I|S)	ポート番号又はサービス名(I|S)を文字列に変換します。 ※
I = port2int(Proto,I|S)	ポート番号又はサービス名(I|S)を整数値に変換します。 ※
※ Proto は、プロトコル種別を表す次の１文字です。{ 'U'=UDP | 'T'=TCP }	

18 ． 日付と時刻

【 参 考 】
・UNIX時刻 とは、1970-01-01 00:00:00 からの経過時刻［単位＝実数秒］のことです。
・UTC とは、協定世界時のことです。すなわち、世界時間のことです。
・LTZ とは、ローカルタイムゾーン設定に基づく地方時間のことです。（例：日本標準時）

＜システム変数＞

（ａ） タイムゾーンと時差

名前	型	意味	備考
UTC	S	協定世界時を表す ”UTC” の文字列	
LTZ	S	ローカルタイムゾーンを表す文字列	日本の例：”JST”
LTZ_OFFSET	I	UTCを基準としたLTZ時刻の時差（秒）	日本の例：+32400
※ LTZ と LTZ_OFFSET の初期値は、インタプリター起動時に OS より自動設定します。LTZ_OFFSET の値を変更することにより、スクリプト内の時差を自由に設定することが出来ます。なお、これらの値を変更したとしても OS には影響を与えません。			

（ｂ） デフォルトの変換書式

名前	型	意味	備考
FMT_SDATE	S	{年月日への｜年月日からの} 変換書式	初期値 = ”%Y-%m-%d”
FMT_STIME	S	{時分秒への｜時分秒からの} 変換書式	初期値 = ”%H:%M:%S”
※全て、Ｃ言語 {strftime(3)｜strptime(3)} 互換の変換書式となっています。			

＜システム関数＞

（ｃ） 現在時刻（=UNIX時刻）の取得

システム関数	説明
D = time()	現在の UNIX時刻 を取得します。【単位＝実数秒】
※ UNIX時刻は、国や地域などに依存しない世界共通の時刻となります。（時差調整や夏時間調整が存在しません。）又、単一の数値で表現できる利便性もあるため、ツインテールdeエンジェルモード!!に含まれる「日付と時刻」の処理関数の多くは、このUNIX時刻を処理対象としています。	

（ｄ） プロセス時刻の取得

システム関数	説明
A = times()	Process時間,UsrCPU時間,SysCPU時間を取得します。
※ 戻り値 A は構造体です。メンバー名と設定値の意味は以下の通りです。【単位＝実数秒】 A.ptime = スクリプト開始 ～ 関数 times() 実行時点までの経過時間 (Process時間) A.utime = Process時間のうちユーザーモードの実行時間 (UsrCPU 時間) A.stime = Process時間のうちカーネルモードの実行時間 (SysCPU 時間)	

（ｅ） UNIX時刻の操作（分解と合成）

システム関数	説明
(year,mon,day[,hour,min,sec]) = t2ymd (D)	UNIX時刻(D)から、UTC 年月日時分秒を求めます。
(year,mon,day[,hour,min,sec]) = t2lymd(D)	UNIX時刻(D)から、LTZ 年月日時分秒を求めます。
D = ymd2t (year,mon,day[,hour,min,sec])	UTC 年月日時分秒から、UNIX時刻(D)を求めます。
D = lymd2t (year,mon,day[,hour,min,sec])	LTZ 年月日時分秒から、UNIX時刻(D)を求めます。
※ 上記４関数において、{year|mon|day|hour|min} は整数型、{sec}は実数型です。 関数 ymd2t() 及び lymd2t() では、年月日[時分秒]を表す１つの文字列でも日時の指定が可能です。	

（ｆ） 時刻文字列の操作

システム関数	説明
S = sdate(D) | lsdate(D)	UNIX時刻(D) から "YYYY-MM-DD" {UTC|LTZ} 文字列を生成します。（※1）
S = stime(D) | lstime(D)	UNIX時刻(D) から "hh:mm:ss" {UTC|LTZ} 文字列を生成します。（※1）
S = sftime(FMT,D) | lsftime(FMT,D)	UNIX時刻(D) から {UTC|LTZ}で表現された時刻文字列(S) を生成します。（※2）
D = sptime(FMT,S) | lsptime(FMT,S)	{UTC|LTZ}で表現された時刻文字列(S) から UNIX時刻(D) を生成します。（※3）
※1 生成される文字列は、システム変数 {FMT_SDATE | FMT_STIME} で変更可能です。 ※2 FMT は、変換書式の文字列です。Ｃ言語 strftime(3) 関数の変換書式と同一です。 ※3 FMT は、変換書式の文字列です。Ｃ言語 strptime(3) 関数の変換書式と同一です。	

（ｇ） うるう年の判定

システム関数	説明
I = isleap(I)	西暦年(I)のうるう年判定をします。（戻り値＝TRUE | FALS)

（ｈ） 時刻構造体の操作

システム関数	説明
A = t2utc(D) | t2ltz(D)	UNIX時刻(D) から、{UTC|LTZ} 時刻構造体 を求めます。
D = utc2t(A) | ltz2t(A)	{UTC|LTZ} 時刻構造体(A) から、UNIX時刻(D) を求めます。
A = utc2ltz(A)	UTC 時刻構造体(A) から、LTZ 時刻構造体(A) を求めます。
A = ltz2utc(A)	LTZ 時刻構造体(A) から、UTC 時刻構造体(A) を求めます。
A = tm_norm(A)	時刻構造体を正規化します。（例：3時65分→4時05分）

時刻構造体の書式（フォーマット）

時刻構造体のメンバー	有効性	型	備　考
t.year = 西暦年４桁	○	I	
t.mon = 月　（０１～１２）	○	I	
t.day = 日　（０１～３１）	○	I	
t.hour = 時　（００～２３）	○	I	
t.min = 分　（００～５９）	○	I	
t.sec = 秒　（００～６０）	○	D	実数型
t.wday = 曜日（００～０６）	△	I	日曜＝0、月曜＝1 ～ 土曜＝6
t.yday = 年日（０～３６５）	△	I	正月＝0、大晦日＝364 又は 365
t.isdst = 夏時間（TRUE/FALS）	○	I	

注意１）○の項目は、常に有効かつ常に必須です。
注意２）△の項目は、入力時には設定不要（未利用）ですが、出力時には利用可能となります。

19 ． 数学関連（整数と実数）

＜システム変数＞

（ａ） 基本の数学定数

名前	型	意味	備考
INF	D	無限大	isinf() で検査可能
NAN	D	非数 (Not a Number)	isnan() で検査可能
M_E	D	自然対数の底	
M_PI	D	円周率	

（ｂ） 最大値と最小値

名前	型	意味	備考
INT_MAX INT_MIN	I	整数の最大値 と 整数の最小値	正の値 と 負の値
DBL_MAX DBL_MIN	D	実数の最大値 と 実数の最小値	正の値 と 負の値
INT_QUANT	I	整数の正の最小値	
INT_EPSILON	I	整数のうち X!=X+1　となる正の最小値	イプシロン値
DBL_QUANT	D	実数の正の最小値	
DBL_EPSILON	D	実数のうち X!=X+1.0 となる正の最小値	イプシロン値

（ｃ） 角度の単位

名前	型	意味	備考
SYS_ANGLE	D	システムの角度単位 (RAD 又は DEG)	初期値 = RAD

＜システム関数＞

（ｄ） 実数の処理

システム関数	説明
D = ceil(D) | floor(D) | trunc(D)	実数の丸め込み {切り上げ | 切り下げ | ０の方向}
D = rint(D) | round(D)	実数の丸め込み {偶数丸め | 四捨五入}
D = abs (D)	絶対値
(Dint,Dfra) = modf (D)	実数 D から {整数=Dint＆少数=Dfra} への分解
D = Dint + Dfra	{整数=Dint＆少数=Dfra} から実数 D への合成
(Dman,Iexp) = frexp(D)	実数 D から {仮数=Dman＆指数=Iexp} への分解
D = ldexp(Dman,Iexp)	{仮数=Dman＆指数=Iexp} から実数 D への合成

（ｅ） 指数と対数

システム関数	説明
D = log(D) | log2(D) | log10(D)	{底ｅ|底 2|底10} の対数
D = exp(D) | exp2(D) | exp10(D)	{　ｅ|　 2|　10} の累乗
D = pow(Dx,Dy) | mod(Dx,Dy)	DxのDy乗 | Dx／Dyの余剰
D = sqrt(D) | cbrt(D)	平方根 | 立方根
D = lin (D) | dbi (D)	リニア値 | デシベル値

（ｆ） 角度の単位

システム関数	説明
D = rad2deg(D) | deg(D)	Rad → Deg 変換
D = deg2rad(D) | rad(D)	Deg → Rad 変換
(Deg,Min,Sec) = rad2dms(D)	Rad → 度分秒 変換
(Deg,Min,Sec) = deg2dms(D)	Deg → 度分秒 変換
D = dms2rad(Deg,Min,Sec)	度分秒 → Rad 変換
D = dms2deg(Deg,Min,Sec)	度分秒 → Deg 変換

（ｇ） 三角関数

システム関数	説明
D = sin (D) | cos (D) | tan (D)	{正弦|余弦|正接}　関数
D = asin (D) | acos (D) | atan (D)	{正弦|余弦|正接} 逆関数
D = atan2(Y,X)	{正接} 逆関数（２変数指定）
角度単位の初期値は rad です。（システム変数 SYS_ANGLE で変更できます。）	

（ｈ） 双曲関数

システム関数	説明
D = sinh(D) | cosh(D) | tanh(D)	双曲線 {正弦|余弦|正接}　関数
D = asinh(D) | acosh(D) | atanh(D)	双曲線 {正弦|余弦|正接} 逆関数

（ｉ） ベッセル関数

システム関数	説明
D = j0(D) | j1(D) | jn(D,D)	第一種ベッセル関数
D = y0(D) | y1(D) | yn(D,D)	第二種ベッセル関数

（ｊ） 座標変換

システム関数	説明
(X,Y,Z) = xrot(x,y,z,θ)	Ｘ座標軸を +θ 回転した後の座標を求める
(X,Y,Z) = yrot(x,y,z,θ)	Ｙ座標軸を +θ 回転した後の座標を求める
(X,Y,Z) = zrot(x,y,z,θ)	Ｚ座標軸を +θ 回転した後の座標を求める
※ 角度単位の初期値は rad です。（システム変数 SYS_ANGLE で変更できます。）	

（ｋ） 統計関数

システム関数	説明
D = max(V) | med(V) | min(V)	{最大値|中央値|最小値} を求める
I = argmax(V) | argmin(V)	{最大値|最小値} のインデックスを求める
(Imin,Imax) = argmed(V)	中央値のインデックスを求める（戻り値は２つ。）
D = sum (V) | ave (V)	{合計値|平均値} を求める
D = svar(V) | uvar(V)	標本分散 (Sample)|不偏分散 (Unbias) を求める
D = sdev(V) | udev(V)	標本標準偏差(Sample)|不偏標準偏差(Unbias) を求める
D = lgamma(D) | tgamma(D)	{loge |True } ガンマ関数
D = erf(D) | erfc(D)	{誤差 |余誤差} 関数
※ パラメーター V は {整数、実数、又は、それらの配列} の任意並びです。 ※ 関数 argmed() の戻り値は、引数が奇数個の時 Imin=Imax 、偶数の時 Imin!=Imax となります。 ※ 標本分散と標本標準偏差は 1/N の式、不偏分散と不偏標準偏差は 1/(N-1) の式です。	

（ｌ） 乱数発生

システム関数	説明
I = rand(MAX)	0 <= x < MAX(I) の整数乱数を発生します。（0以上MAX未満）
D = urand()	0.0 <= x < 1.0 の一様乱数を発生します。
D = nrand([AVE,DEV])	正規乱数を発生します。{平均値=AVE(D)／標準偏差=DEV(D)}
void = seed(I)	乱数のシードを手動設定します。（デフォルトでは、自動設定）
※ 関数 nrand() の引数を省略した場合は、標準正規分布（AVE=0.0／DEV=1.0）となります。	

（ｍ） 無限大と非数

システム関数	説明
I = isinf(D)	無限大かどうか判定します。(戻り値＝TRUE/FALS)
I = isnan(D)	非 数かどうか判定します。(戻り値＝TRUE/FALS)

20．プロセスと割り込み

＜システム定数＞

（ａ） プロセス関連の定数

名前	型	意味	備考
PID	I	自プロセスIDの値	= PID
PPID	I	親プロセスIDの値	
$$	I	自プロセスIDの値	= PID
$?	I	（直近）外部コマンドの終了値	

（ｂ） シグナル関連の定数

名前	型	意味	備考
SIG_DFL	P	割り込み動作設定用（既定動作）	rxsig()で使用
SIG_IGN	P	割り込み動作設定用（受信無視）	rxsig()で使用
SIG_ERR	P	割り込み設定時のエラー戻り値［ = NULL ］	rxsig()で使用
NSIG	I	シグナル番号の定義数	0 ～（NSIG-1）が有効
シグナル番号（I）= SIGHUP / SIGINT / SIGQUIT/ SIGILL / SIGABRT/ SIGFPE / SIGKILL/ SIGSEGV/ SIGPIPE/ SIGALRM/ SIGTERM/ SIGUSR1/ SIGUSR2/ SIGCHLD/ SIGCONT/ SIGSTOP/ SIGTSTP/ SIGTTIN/ SIGTTOU/			

（ｃ） プロセス関数

システム関数	説明
I = pid() | ppid()	{自|親} プロセスＩＤを取得します。
D = sleep (D)	スリープ状態に Ｉ[秒] 入ります。（戻り値＝残時間[秒]）
void = pause()	ポーズ　状態に入ります。
void = exit(I)	スクリプトを終了値 Ｉ で終了します。

（ｄ） 外部コマンド

システム関数	説明
I = system(S)	外部コマンド S を実行します。（戻り値＝コマンドの終了値）
So = `S` [< Si]	外部コマンド S を実行します。（戻り値＝標準出力）
※ 記号 ` ` にて実行する場合は、オプションで標準入力 Si(S) の指定が出来ます。又、標準出力は戻り値 So(S) としてキャプチャされます。（標準エラー出力はキャプチャされません。）なお、終了値は $? にセットされます。	

（ｅ） 割り込み関数

システム関数	説明
I = tx_sig(Ipid,Isig)	プロセス（Ipid番）にシグナル（Isig番）を送信します。
I = rx_sig(Func,Isig)	シグナル（Isig番）受信時の処理関数（Func）を登録します。
※ Func(X) は、１つの引数を持つ任意のユーザー関数です。シグナル受信時に自動で割り込み実行されます。（仮引数名は任意です。又、実引数には受信したシグナル番号がセットされます。）なお、Func(X) の代わりに {SIG_DFL|SIG_IGN} を指定すると、シグナル受信時の動作が、それぞれ {既定動作|受信無視} となります。	

21 ． ガーベージコレクション

＜システム関数＞

（ａ） 明示的実行

システム関数	説明
void = gc_collect(void)	ガーベージコレクションを明示的に実行します。　※
P　= alloc(I)	Iバイトのメモリー領域を確保て、その先頭アドレスを戻します。
void = free(P)	関数 alloc() で確保したメモリー領域を明示的に解放します。※
※ 通常は、完全自動でメモリ管理がされていますので、当関数を明示的に実行する必要はありません。 しかし、大量のデータを連続的に使用＆破棄する場合などにおいて、適切なタイミングで当関数を実行するこによって、メモリー使用量を削減することができます。	