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,...) A = argsort(V) | argrsort(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） {昇順ソート | 降順ソート} のインデックス値　※1
※ 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(Date)と時刻(Time)

【 参 考 】
・UNIX時刻 とは、1970-01-01 00:00:00 からの経過時間［秒］です。　⇒ スクリプトの計算処理用
・UTC は、世界標準時（協定世界時）です。　⇒ 世界時での入出力用
・LTZ は、Local Time Zone（地方標準時）です。（例：日本標準時）　⇒ 地方時での入出力用

タイムゾーン＆時差

関連定数：LTZ	ローカルタイムゾーンを表す文字列	日本の例："JST"
関連定数：LTZ_OFFSET	UTC時刻を基準としたLTZ時刻の時差 【単位＝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 より取得＆設定されます。LTZ_OFFSET の値は、正の値＝進み時差／負の値＝遅れ時差となります。この値を変更することで、スクリプト内部の時差を任意の値に設定可能です。（なお、この値を変更しても OS には影響を与えません。）

デフォルト変換書式

関連定数：FMT_SDATE	年月日⇔文字列の変換書式（初期値＝"%Y-%m-%d"）	※
関連定数：FMT_STIME	時分秒⇔文字列の変換書式（初期値＝"%H:%M:%S"）	※

※ C言語 {strftime(3)｜strptime(3)} 互換の変換書式（文字列）です。任意の値に設定可能です。

（ a ） 現在時刻の取得

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】	※1
A = ptime()	３種類のプロセス時刻を取得します。	※2

※1 UNIX時刻 は、単一の数値で表現できる世界共通の時刻です。（時差や夏時間の調整・うるう秒などが存在しません。）このため、「日付と時刻」の処理関数の多くが、この UNIX時刻 を処理対象とします。
※2 戻り値 A は構造体。メンバー名と値の意味は以下の通り。【単位＝実数秒】
A.ptime = スクリプト開始 ～ 関数 ptime() 実行までの経過時間 （Process時間）
A.utime = Process時間のうちユーザーモード実行時間 （UsrCPU 時間）
A.stime = Process時間のうちカーネルモード実行時間 （SysCPU 時間）

（ b ） UNIX時刻 ⇔ 数 値（年月日｜時分秒）

システム関数	説明	
(y,m,d,h,m,s) = t2ymdhms#(D)	年,月,日,時,分,秒 ← UNIX時刻	※1
(y,m,d) = t2ymd#(D)	年,月,日 ← UNIX時刻	※1
(h,m,s) = t2hms#(D)	時,分,秒 ← UNIX時刻	※1
D = ymdhms2t#(y,m,d,h,m,s)	UNIX時刻 ← 年,月,日,時,分,秒	※1
D = ymd2t# (y,m,d)	UNIX時刻 ← 年,月,日	※1
D = hms2t# (h,m,s)	UNIX時刻 ← 時,分,秒	※1

※# "0" の時 UTC 変換。"" の時 LTZ 変換。⇒ LTZ_OFFSET 調整有り！
※1 {y｜m｜d｜h｜m} は、年,月,日,時,分を表す整数です。 {s} は、秒を表す実数です。

（ｃ） UNIX時刻 ⇔ 文字列（年月日｜時分秒）

システム関数	説明
S = sdate#(D) S = stime#(D)	"YYYY-MM-DD" ← UNIX時刻 ※1 "HH:MM:SS" ← UNIX時刻 ※1
D = vdate#(S) D = vtime#(S)	UNIX時刻 ← "YYYY-MM-DD" ※1 UNIX時刻 ← "HH:MM:SS" ※1
S = strftime#(FMT,D) D = strptime#(FMT,S)	"任意文字列" ← UNIX時刻 ※2 UNIX時刻 ← "任意文字列" ※3
※# "0" の時 UTC 変換。"" の時 LTZ 変換。⇒ LTZ_OFFSET 調整有り！ ※1 変換書式は、システム変数 {FMT_SDATE｜FMT_STIME} に従います。 ※2 FMT は、Ｃ言語 strftime(3) 互換の変換書式（文字列）です。 ※3 FMT は、Ｃ言語 strptime(3) 互換の変換書式（文字列）です。	

（ｄ） 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）

（ｅ） 時刻構造体

システム関数	説明
A = t2tm#(D) D = tm2t#(A)	時刻構造体 tm ← UNIX時刻 UNIX時刻 ← 時刻構造体 tm
A = tm_norm(A)	時刻構造体 tm の正規化（例：3時65分→4時05分）
※# "0" の時 UTC 変換。"" の時 LTZ 変換。⇒ LTZ_OFFSET 調整有り！	

時刻構造体 tm メンバー（９種）

```
tm.year = 年（西暦4桁）
tm.mon  = 月（01～12）
tm.day  = 日（01～31）
tm.hour = 時（00～23）
tm.min  = 分（00～59）
tm.sec  = 秒（00～60）
```

```
tm.wday  = 曜日（00～06）整数 → 日曜0～土曜6
tm.yday  = 年日（0～365）整数 → 正月0～年末364|365
tm.isdst = 夏時間の有効性（TRUE／FALS）
```

注意１）メンバーの型は、sec のみ実数です。他は全て整数（TRUE／FALSを含む）です。
注意２）wday と yday は、関数出力時に利用可能となります。（関数入力時は設定不要です。）
注意３）黄色の数値は、うるう秒又はうるう年の範囲です。

19 . 数学関数と整数＆実数

基本定数

関連定数：SYS_ANGLE	システム関数の角度単位 (RAD｜DEG)
関連定数：M_PI	円周率　　(M_PI = 3.141593...)
関連定数：M_E	自然対数の底 (M_E = 2.718282...)
関連定数：INT_{MAX｜MIN}	整数の {最大値（正の値）｜最小値（負の値）}
関連定数：DBL_{MAX｜MIN}	実数の {最大値（正の値）｜最小値（負の値）}
関連定数：INT_{QUANT｜EPSILON}	整数の {正の最小値｜イプシロン値}
関連定数：DBL_{QUANT｜EPSILON}	実数の {正の最小値｜イプシロン値}
関連定数：INF	無限大 (INFinity　　)　【参考：実数型】
関連定数：NAN	非　数 (Not A Number)　【参考：実数型】

(a)　三角関数と双曲関数

システム関数	説明
D = sin (D) ｜ cos (D) ｜ tan (D) D = asin (D) ｜ acos (D) ｜ atan (D) D = atan2(Dy,Dx)	{正弦｜余弦｜正接}　関数　※ {正弦｜余弦｜正接} 逆関数　※ {正接} 逆関数（２変数指定）　※
D = sinh(D) ｜ cosh(D) ｜ tanh(D) D = asinh(D) ｜ acosh(D) ｜ atanh(D)	双曲線 {正弦｜余弦｜正接}　関数　※ 双曲線 {正弦｜余弦｜正接} 逆関数　※
※ 角度単位は RAD です。(SYS_ANGLE = RAD｜DEG で設定可能です。)	

(b)　角度変換と座標変換

システム関数	説明
D = deg(D) ｜ rad2deg(D) D = rad(D) ｜ deg2rad(D)	Deg ← Rad Rad ← Deg
(Deg,Min,Sec) = rad2dms(D) (Deg,Min,Sec) = deg2dms(D) D = dms2rad(Deg,Min,Sec) D = dms2deg(Deg,Min,Sec)	度分秒 ← Rad　(Deg と Min は整数 Sec は実数) 度分秒 ← Deg　(〃) Rad ← 度分秒　(Deg と Min は整数 Sec は実数) Deg ← 度分秒　(〃)
(X,Y,Z) = xrot(X,Y,Z,θ) (X,Y,Z) = yrot(X,Y,Z,θ) (X,Y,Z) = zrot(X,Y,Z,θ)	X 座標軸を +θ 回転　※ Y 座標軸を +θ 回転　※ Z 座標軸を +θ 回転　※
※ 角度単位は RAD です。(SYS_ANGLE = RAD｜DEG で設定可能です。)	

(c)　指数関数と対数関数

システム関数	説明
D = exp(D) ｜ exp2(D) ｜ exp10(D) D = log(D) ｜ log2(D) ｜ log10(D)	{　e｜　2｜　10} の累乗【 exp() 別名＝expe() 】 {底ｅ｜底 2｜底10} の対数【 log() 別名＝loge() 】
D = pow(Dx,Dy) ｜ mod(Dx,Dy) D = lgamma(D) ｜ tgamma(D) D = sqrt(D) ｜ cbrt(D) D = lin (D) ｜ dbi (D)	Dx の Dy 乗｜Dx／Dy の余剰（＝演算子%） {loge　　｜True　　} ガンマ関数 平方根　　｜立方根 リニア値　｜デシベル値

（ｄ） ベッセル関数

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

（ｅ） 統計関数と乱数発生

システム関数	説明
D = max(V) | med(V) | min(V) I = argmax(V) | argmed(V) | argmin(V)	{最大値|中央値|最小値} {最大値|中央値|最小値} のインデックス値 ※1 ※1
D = sum (V) | ave (V) D = svar(V) | uvar(V) D = sdev(V) | udev(V) D = erf(D) | erfc(D)	{合計値|平均値} ※1 {標　本|不　偏} 分　散 ※1※2 {標　本|不　偏} 標準偏差 ※1※2 {誤差　|余誤差} 関　数
I = rand(MAX) D = urand() D = nrand([AVE,DEV]) U = seed(I)	整数乱数の発生（戻り値＝ 0　以上 MAX 未満） 一様乱数の発生（戻り値＝ 0.0 以上 1.0 未満） 正規乱数の発生（平均値＝AVE／標準偏差＝DEV） ※3 乱数シード値の設定
※1 パラメーター V は {整数、実数、又は、それらを要素とする配列} の並びです。 ※2 標本値は 1/N の式、不偏値は 1/(N-1) の式です。 ※3 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。	

（ｆ） 整数値と実数値

システム関数	説明
I = int (＊) D = dbl (＊) I|D = atov(＊)	パラメータの整数化【 別名 atoi() 】 パラメータの実数化【 別名 atof() 】 パラメータの数値化（戻り値の型は自動判定）
D = abs (D)	絶対値
D = ceil(D) | floor(D) | trunc(D) D = rint(D) | round(D)	実数の丸め込み {切り上げ|切り下げ|０の方向} 実数の丸め込み {偶数丸め|四捨五入}
(Dint,Dfra) = modf (D) D = Dint + Dfra (Dman,Iexp) = frexp(D) D = ldexp(Dman,Iexp)	実数 D から {整数=Dint＆小数=Dfra} への分解 {整数=Dint＆小数=Dfra} から実数 D への合成 実数 D から {仮数=Dman＆指数=Iexp} への分解 {仮数=Dman＆指数=Iexp} から実数 D への合成

（ｇ） 無限大と非数値

システム関数	説明
I = isinf(D) I = isnan(D)	無限大の判定（戻り値＝TRUE/FALS） 非　数の判定（戻り値＝TRUE/FALS）

20 . プロセスと割り込み

(a) プロセス関数

システム関数	説明
D = sleep(D) U = pause() U = exit(I)	スリープ　の実行（時間＝ D [秒]／戻り値＝残時間[秒]） ポーズ　 の実行（時間＝無制限 ／戻り値＝無し　　） スクリプトの終了（終了値 I ）
I = system(S) S = `S`	外部コマンド S の実行（戻り値＝コマンドの終了値） 外部コマンド S の実行（戻り値＝標準出力）

関連定数：$$ | PID　　自己プロセスのＩＤ値（ $$ == PID ）
関連定数：$?　　外部コマンドの終了値

(b) 割り込み制御

システム関数	説明
I = tx_sig(Isig,Ipid)	シグナル（Isig番）をプロセス（Ipid番）に送信する。 戻り値＝整数値のゼロ（成功）又はNULL（失敗）
P = rx_sig(Isig,Func)	シグナル（Isig番）受信時の処理関数（Func）を登録する。※ 戻り値＝設定前のFunc（成功）又はNULL（失敗）
※ Func は、１つの引数を取る任意のユーザー関数です。シグナル受信時に割り込み実行されます。この時、受信したシグナル番号がパラメータとしてセットされます。なお、Func に {SIG_DFL｜SIG_IGN} を指定すると、シグナル受信時の動作が {既定動作｜受信無視} となります。	

関連定数：SIG_ {DFL｜IGN｜ERR}　　割り込み制御用 {既定動作｜受信無視｜エラー戻り値(NULL)}
関連定数：NSIG　　シグナルの個数（ 0 ～ NSIG-1 が有効）
関連定数：SIG {ABRT｜ALRM｜CHLD｜CONT｜FPE ｜HUP ｜ILL ｜INT ｜KILL
｜PIPE｜QUIT｜SEGV｜STOP｜TERM｜TSTP｜TTIN｜TTOU｜USR1｜USR2}　　POSIX標準シグナル

21 . メモリ管理

(a) 取得と解放

システム関数	説明
P = alloc(I) U = free (P)	I バイトのメモリ領域を確保し、先頭アドレスを戻す。 関数 alloc() で確保したメモリ領域を解放する。

(b) ＧＣ実行

システム関数	説明
U = gc_collect()	ガーベージコレクションを明示的に実行する。　※
※ 極めて大量のメモリを繰り返し使用＆解放する場合等においては、本関数を適宜実行することでメモリ使用量を大幅に削減できる場合があります。（通常は、明示的な実行は不要。）	