

マクロ使用法

目次

プログラミングガイド.....	2
データ型.....	2
変数の使用.....	2
四則演算について	3
マクロの使用	3
プログラムの条件判定(if~endif)	5
プログラムの繰り返し(while~endwhile)	5
点を描画する	6
点を横に10個描画する	6
幾何計算マクロの使用	7
練習問題（放物線）	8
リファレンスマニュアル	12
予約済み変数	12
画面制御系マクロ	12
システム制御系マクロ	14
幾何計算系マクロ	15
データベース操作系マクロ	35
属性変更系マクロ	39
フォント属性変更系マクロ	41
プログラム制御系マクロ	43

プログラミングガイド

データ型

データ型には次のタイプが用意されています。

void	型なし 型の指定がないときに使用されます。
variable	全対応型 すべての型に変化します。
int	整数値 10, 20, 30 などの整数値が格納されます。
bool	真偽 真か偽のいずれかが入ります。真は TRUE、偽は FALSE を使用します。
double	実数値 123.456, 111.222 などの実数値が格納されます。
point	点情報 点の情報を持ちます。
line	線情報 線の情報を持ちます。
circle	円情報 円の情報を持ちます。
object	オブジェクト情報 オブジェクトの情報を持ちます。

変数の使用

変数は英数字と'_' (アンダースコア) が使用でき、文字数に制限はありません。また、大文字と小文字は区別されません。ただし、変数の先頭文字は英字で始まらなければいけません。また、予約済み変数に代入することはできません。予約済み変数についてはリファレンスマニュアルを参照してください。

実数の変数を作成するには、

```
v1 = V(10.0)
```

と、記述します。v マクロは実数を返すことをあらわしますので変数 v1 には 10.0 の値が格納されます。同じように、

```
v2 = V(20.0)
```

とします。すると変数 v2 には 20.0 の値が格納されます。

また、引き数を実数値の場合に限り、変数を混ぜた四則演算が可能です。

```
v3 = V(v1 + v2 + 10.0)
```

とすると、変数 v3 には 40.0 の値が格納されます。

また、マクロ V は省略することが可能です。

```
v3 = v1 + v2 + 10.0
```

となります。

文字列の変数を作成するには、文字列を“(ダブルクォーテーション)”で囲みます。

```
s1 = Str("Sample String")
```

のように記述すると、s1 には“Sample String”が格納されます。

また、マクロ Str は省略することが可能です。

```
s1 = "Sample String"
```

四則演算について

演算子には次のものが用意されています。

演算子	意味	優先順位
*	積 (×)。	1
/	除 (÷)。	1
+	和 (+)。	2
-	差 (−)。	2

計算方法は()で囲まれた範囲が最優先されます。それ以外では、優先順位の低い方が優先されます。また、優先順位が同じものでは、左から順に計算されます。

マクロの使用

マクロの使用方法は、「ウインドウ」メニューから「マクロ」を選択してマクロ入力ウインドウを表示させます (デフォルトではアプリケーションの下側に表示)。そのウインドウ内で1行入力しリターンキーを押した時点で即座に実行されます。直接、ファイル名を指定するとそのファイルの中身が上から順に実行されます。ファイルが実行されているときに限り、if 文、while 文の使用が可能になります。また、ファイル中からファイルを呼び出すことも可能です。そうすると、呼び出したファイルが正常終了するとまた次の行から実行されます。例えば、ファイル1の5行目でファイル2を呼び出した場合、ファイル1は待機してファイル2が実行されます。そして、ファイル2がすべて正常に終了すると、ファイル1に戻り6行目から実行が再開されます。また、ファイルを実行中にエラーが発生したときはエラー行で実行を停止し、エラーが発生したファイル名とその行数とエラーメッセージが表示されます。

パラメトリック的な使い方も可能です。変数を上手く使用することや、プログラムの組み方次第で簡単に形状を変化させることが出来ます。

マクロの文法は基本的に、

`[変数名] = [マクロ名]([引き数], [引き数], [引き数], ...)`

の形をとります。ただし、変数にとる必要がなければ、左辺と '=' は省略できます (例 1)。実数を返すマクロと、文字列を返すマクロは、マクロ名が省略できます (例 2)。引き数が無いマクロに関しては () が省略できます (例 3)。また、英字の大文字と小文字の区別はなく、スペースとタブは無視されます。ただし、ダブルクォーテーション '""' で囲まれた範囲は文字列として認識するため例外です。コメントについては、ダブルスラッシュ '//' を使用します。ダブルスラッシュの位置からその行の最後までは無視されます。

変数の作成は、代入することで作成されます。すでに作成済みの変数に対して代入すると、変数は上書きされ前の値は捨てられます。変数の型はマクロ名により、そのマクロが返す型に自動的に決定、または変更されます (例 4)。変数を参照する場合、未定義変数のデフォルト値は存在しないので、必ず用意しなければなりません。また、変数は 1 度作成されるとアプリケーションが終了するまで保持され、コマンドラインで実行した場合でもファイルから実行した場合でもグローバルに有効です。

(例 1)

```
object = SetLine(0, 0, 10, 10)
```

↓

```
SetLine(0, 0, 10, 10)
```

(例 2)

```
v1 = V(123)
```

↓

```
v1 = 123
```

```
s1 = Str("Test String")
```

↓

```
s1 = "Test String"
```

(例 3)

```
Zoom()
```

↓

```
Zoom
```

(例 4)

```
variable = 100          // variable は実数型で作成される。  
variable = "Test String" // variable は文字列型に変更される。
```

プログラムの条件判定(if~endif)

プログラムをある条件により処理させたり、無視させたりするには if 文を使用します。

[構文]

```
if(bool fFlag)  
    .  
    .  
    .
```

```
endif
```

fFlag が真のときに if から endif までは実行されます。fFlag が偽のときは if から endif までは実行されずにスキップして、endif 以降がまた実行されます。

判定方法は、判定演算子を使用します。判定演算子には次のものが用意されています。

判定演算子	使い方	意味
<	v1 < v2	v1 が v2 よりも小さいとき真になります。
>	v1 > v2	v1 が v2 よりも大きいとき真になります。
<=	v1 <= v2	v1 が v2 よりも小さいか、等しいとき真になります。
>=	v1 >= v2	v1 が v2 よりも大きいのか、等しいとき真になります。
==	v1 == v2	v1 と v2 が等しいとき真になります。
!=	v1 != v2	v1 と v2 が等しくないとき真になります。

if 文の中に if 文を入れ、多重化することも可能です。

```
if(v1 < 10)  
    if(v2 < 20)  
        endif  
    endif  
endif
```

プログラムの繰り返し(while~endwhile)

プログラムをある条件下でループさせるには、while 文を使用します。

[構文]

```
while(bool fFlag)  
    .  
    .
```

•
endwhile

while 行で fFlag を真偽し、fFlag が真であればそれ以降が実行され endwhile に達すると対応する while 行まで戻り、それを fFlag が偽になるまで繰り返されます。fFlag が偽になったとき while から endwhile まだがスキップし、endwhile 以降がまた実行されます。

判定方法は、判定演算子を使用します。判定演算子については条件判定文(if)を参照してください。

while 文の中に while 文を入れ、多重化することも可能です。

```
while(v1 < 10)
    while(v1 < 20)
        endwhile
    endwhile
endwhile
```

点を描画する

点を描画するには、SetPoint マクロを使います。SetPoint マクロには次の2つのマクロが用意されています。

```
object SetPoint(point p)
object SetPoint(double x, double y, double z = 0)
```

object とは、点オブジェクトを返すことをあらわします。1つ目のマクロで、()内の point p を引き数と呼びます。そして、point はデータ型をあらわし、p がそのデータ型の値になります。よって、p が示す座標に点を登録し、点オブジェクトを返します。

```
p1 = Pxy(10.0, 10.0) // 座標(10.0, 10.0)の点作成し、変数 p1 へ代入する。
```

```
obj1 = SetPoint(p1) // p1 をデータベースへ登録し、点を描画する。
```

2つ目のマクロでは、double x, double y, double z = 0 の引き数をとります。x, y, z の座標に点を登録し、点オブジェクトを返します。double z = 0 とは z の値を省略することができ、このときにデフォルトで0になることをあらわします。よって、

```
obj1 = SetPoint(10, 10, 0)
```

と、

```
obj1 = SetPoint(10, 10)
```

は、同じ結果になります。

点を横に10個描画する

変数と while 文を使って点を原点から横方向へ10ずつオフセットした点を10個描画しま

す。

```
i = 0 // カウント用の変数 i を用意。
x = 0 // X座標用のための変数 x を用意。
while(i < 10) // 変数 i が 10 になるまで繰り返す。
    SetPoint(x, 0) // 座標 (x, 0) の位置に点を描画。
    x = x + 10 // X の値を 10 増やす。
    i = i + 1 // 変数 i をインクリメント。
endwhile // while の終了位置。
```

次のプログラムは変数 x を使用しない例です。結果はまったく同じになります。

```
i = 0 // カウント用の変数 i を用意。
while(i < 10) // 変数 i が 10 になるまで繰り返す。
    SetPoint(i * 10, 0) // 座標 (i * 10, 0) の位置に点を描画。
    i = i + 1 // 変数 i をインクリメント。
endwhile // while の終了位置。
```

幾何計算マクロの使用

幾何計算マクロを使用して簡単な図形を書いていきます。ここでは、四角の 4 つの角にコーナー R を付けた形状を書いてみます。

四角の対角となる 2 点とコーナー R の半径を決めます。

```
x1 = 0
y1 = 0
x2 = 50
y2 = 50
rad = 5
```

次に、4 つの頂点を持つ点変数を作成します。

```
pLT = Pxy(x1, y2) // 左上の点を求めます。
pLB = Pxy(x1, y1) // 左下の点を求めます。
pRB = Pxy(x2, y1) // 右下の点を求めます。
pRT = Pxy(x2, y2) // 右上の点を求めます。
```

次に、4 辺の線変数を作成します。

```
lL = Lpp(pLT, pLB) // 左の線を求めます。
lB = Lpp(pLB, pRB) // 下の線を求めます。
lR = Lpp(pRT, pRB) // 右の線を求めます。
lT = Lpp(pLT, pRT) // 上の線を求めます。
```

次に、各コーナー R を作成します。


```
cLT = C1lr(1L, 1T, rad, R, B)    // 左上の円を求めます。
```

```
cLB = C1lr(1L, 1B, rad, R, A)    // 左下の円を求めます。
```

```
cRB = C1lr(1B, 1R, rad, A, L)    // 右下の円を求めます。
```

```
cRT = C1lr(1R, 1T, rad, L, B)    // 右上の円を求めます。
```

次に各接点を求め、実際にデータベースへ登録します。

```
p1 = Plc(1T, cLT, T)             // 上の直線と左上の円の接点を求めます。
```

```
p2 = Plc(1L, cLT, T)             // 左の直線と左上の円の接点を求めます。
```

```
p3 = Plc(1L, cLB, T)             // 左の直線と左下の円の接点を求めます。
```

```
p4 = Plc(1B, cLB, T)             // 下の直線と左下の円の接点を求めます。
```

```
p5 = Plc(1B, cRB, T)             // 下の直線と右下の円の接点を求めます。
```

```
p6 = Plc(1R, cRB, T)             // 右の直線と右下の円の接点を求めます。
```

```
p7 = Plc(1R, cRT, T)             // 右の直線と右上の円の接点を求めます。
```

```
p8 = Plc(1T, cRT, T)             // 上の直線と右上の円の接点を求めます。
```

```
SetLine(p2, p3)                  // 左の直線を登録し、描画します。
```

```
SetLine(p4, p5)                  // 下の直線を登録し、描画します。
```

```
SetLine(p6, p7)                  // 右の直線を登録し、描画します。
```

```
SetLine(p8, p1)                  // 上の直線を登録し、描画します。
```

```
SetArc(CCW, p1, p2, cLT)         // 左上の円弧を登録し、描画します。
```

```
SetArc(CCW, p3, p4, cLB)         // 左下の円弧を登録し、描画します。
```

```
SetArc(CCW, p5, p6, cRB)         // 右下の円弧を登録し、描画します。
```

```
SetArc(CCW, p7, p8, cRT)         // 右上の円弧を登録し、描画します。
```

これで形状が描画されたので、全体表示します。

```
Zoom()
```

引き数がないマクロは()を省略することができます。従って、

```
Zoom
```

でも同じ結果になります。

以上で完成ですが、最初に指定した x1, y1, x2, y2, rad の値を変更するだけで簡単に形状が変わることがわかります。

練習問題（放物線）

ここでは、while を使って、二次曲線（放物線）を書いてみましょう。

まず、X 軸、Y 軸に目盛りを振ったグラフを描画するプログラムをファイル d:¥Graph.mac に保存します。次に、二次曲線を描画するプログラムをファイル d:¥Curve.mac に保存し、そこから d:¥Graph.mac を呼び出します。プログラムは次の通りです。

Graph.mac の内容

```

length = 50      // 軸の長さを決めます。
grad = 0.5       // 目盛りの長さを決めます。
SetLine(-length, 0, length, 0)
SetLine(0, -length, 0, length)

FontType (FT_TRUETYPE)
FontSelect ("MS 明朝")
FontSize (2, 2, 0)

// X軸に目盛りを振ります。
x = -length
while (x <= length)
    SetLine(x, -grad, x, grad)
    x = x + 5
endwhile

FontAlign (FA_CENTER, FA_BOTTOM)
SetText (-50, grad, "-50")
SetText (-40, grad, "-40")
SetText (-30, grad, "-30")
SetText (-20, grad, "-20")
SetText (-10, grad, "-10")
SetText (10, grad, "10")
SetText (20, grad, "20")
SetText (30, grad, "30")
SetText (40, grad, "40")
SetText (50, grad, "50")

// Y軸に目盛りを振ります。
y = -length
while (y <= length)
    SetLine(-grad, y, grad, y)
    y = y + 5
endwhile

FontAlign (FA_LEFT, FA_CENTER)

```

```

SetText(grad, -50, "-50")
SetText(grad, -40, "-40")
SetText(grad, -30, "-30")
SetText(grad, -20, "-20")
SetText(grad, -10, "-10")
SetText(grad, 10, "10")
SetText(grad, 20, "20")
SetText(grad, 30, "30")
SetText(grad, 40, "40")
SetText(grad, 50, "50")

```

d:¥Curve.mac の内容

```

// グラフを描画するプログラムを呼び出します。
d:¥graph.mac

// 二次曲線  $y = a(x*x) + bx + c$  の式を使用して、
//  $-25 \leq x \leq 25$  の範囲で放物線を描画します。
aVal = 1 / 10 // 式中の a の値を決めます。
bVal = 0      // 式中の b の値を決めます。
cVal = -30    // 式中の c の値を決めます。
min = -25     // x の最小値を決めます。
max = 25      // x の最大値を決めます。
step = 0.5    // x の増分値を決めます。

x1 = 0
y1 = 0
x2 = min
y2 = 0
start = 0

while(x2 <= max)
    y2 = aVal * (x2 * x2) + bVal * x2 + cVal
    if(start != 0)
        SetLine(x1, y1, x2, y2)
    endif
    x1 = x2

```

```
    y1 = y2
    x2 = x2 + step
    start = 1
endwhile

zoom
```

リファレンスマニュアル

予約済み変数

変数名	意味
PI	円周率(π)。
T	指定なし。
R	右。
L	左または大きい。
A	上。
B	下。
I	内接。
O	外接。
S	小さい。
CW	円弧右回り。
CCW	円弧左回り。

画面制御系マクロ

[構文]

```
void EchoOn(void)
```

[解説]

ファイルから実行中のプログラムを表示する。

[構文]

```
void EchoOff(void)
```

[解説]

ファイルから実行中のプログラムを表示しない。

[構文]

```
void Message(string sMessage)
```

[解説]

任意のメッセージを表示する。

[引き数]

sMessage 表示するメッセージ。

[構文]

```
void Zoom()
void Zoom(int x1, int y1, int x2, int y2)
```

[解説]

画面の拡大・縮小を行う。引き数を省略すると全体表示を実行する。
左から右を指定すると拡大し、逆に右から左を指定すると縮小する。

[引き数]

x1	第1点目のX座標を物理座標で指定する。
y1	第1点目のY座標を物理座標で指定する。
x2	第2点目のX座標を物理座標で指定する。
y2	第2点目のY座標を物理座標で指定する。

[構文]

```
void Scroll(int x, int y)
```

[解説]

画面のスクロールを行う。

[引き数]

x	横方向の移動量を物理座標で指定する。
y	縦方向の移動量を物理座標で指定する。

[構文]

```
void Rotate(int iRotate)
```

[解説]

画面の視点回転を行う。

[引き数]

iRotate	回転を行う軸を指定します。この値は次のいずれかになります。
---------	-------------------------------

値	意味
RT_XPLUS	X軸を＋方向へ回転。
RT_XMINUS	X軸を－方向へ回転。
RT_YPLUS	Y軸を＋方向へ回転。
RT_YMINUS	Y軸を－方向へ回転。
RT_ZPLUS	Z軸を＋方向へ回転。
RT_ZMINUS	Z軸を－方向へ回転。
RT_XY	X Y平面を表示するように回転。
RT_ZX	Z X平面を表示するように回転。
RT_YZ	Y Z平面を表示するように回転。

RT_UP	現在の視点を上方向へ回転。
RT_DOWN	現在の視点を下方向へ回転。
RT_LEFT	現在の視点を左方向へ回転。
RT_RIGHT	現在の視点を右方向へ回転。

システム制御系マクロ

[構文]

```
void New()
```

[解説]

新規ファイルを作成する。

[構文]

```
void Open()
```

```
void Open(string sFilePath)
```

[解説]

ファイルを開く。引き数を省略するとファイルダイアログが表示される。

[引き数]

sFilePath 開くファイル名。

[構文]

```
void Save()
```

[解説]

現在のファイルを上書き保存する。

[構文]

```
void SaveAs()
```

[解説]

現在のファイルを名前を付けて保存する。

[構文]

```
void Close()
```

[解説]

現在のファイルを閉じる。

[構文]

```
void Exit()
```

[解説]

アプリケーションを終了する。

幾何計算系マクロ

[構文]

```
void Dump(bool v)
void Dump(int v)
void Dump(double v)
void Dump(string v)
void Dump(point v)
void Dump(line v)
void Dump(circle v)
void Dump(object v)
```

[解説]

変数内容をダンプする。

[引き数]

v ダンプする変数。

[構文]

```
int ValType(variant val)
```

[解説]

変数型を返す。戻り値は次のいずれかになります。

値	意味
VT_BOOL	変数は bool 値。
VT_DOUBLE	変数は double 値。
VT_STRING	変数は string 値。
VT_POINT	変数は point 値。
VT_LINE	変数は line 値。
VT_CIRCLE	変数は circle 値。
VT_OBJECT	変数は object 値。

[引き数]

val 変数型を取得する変数。

[構文]


```
int V(int iValue)
double V(double dValue)
```

[解説]

数値を返す。

[引き数]

iValue 整数。
dValue 実数。

[構文]

```
string Str(string sValue)
```

[解説]

文字列を返す。

[引き数]

sValue 文字列。

[構文]

```
double Sin(double x)
```

[解説]

x のコサインを返します。x が 263 以上か、-263 以下のときは、計算結果の有効桁数が一部失われることがあります。

[引き数]

x ラジアン単位の数値。

[構文]

```
double Cos(double x)
```

[解説]

x のサインを返します。x が 263 以上か、-263 以下のときは、計算結果の有効桁数が一部失われることがあります。

[引き数]

x ラジアン単位の数値。

[構文]

```
double Tan(double x)
```

[解説]

x のタンジェントを返します。x が 263 以上か、-263 以下のときは、計算結果の有効桁数が一部失われることがあります。

[引き数]

x ラジアン単位の角度。

[構文]

```
double ASin(double x)
```

[解説]

$-\pi/2$ ラジアンから $\pi/2$ ラジアンの範囲で x のアークサインを計算します。 x が -1 より小さいか、または 1 より大きい場合には、戻り値は不定になります。

[引き数]

x アークサインを計算する値。

[構文]

```
double ACos(double x)
```

[解説]

0 から π ラジアンの範囲で x のアークコサイン値を返します。 x が -1 より小さいか、または 1 より大きい場合には、戻り値は不定になります。

[引き数]

x アークコサインを計算する値。

[構文]

```
double ATan(double x)
```

[解説]

x のアークタンジェントを返します。 x が 0 の場合、 0 を返します。 $-\pi/2$ ラジアンから $\pi/2$ ラジアンの範囲で値を返します。

[引き数]

x アークタンジェントを計算する値。

[構文]

```
double Sqrt(double x)
```

[解説]

平方根の値を返します。 x を負にすると、戻り値は不定です。

[引き数]

x 平方根を計算する値。

[構文]

```
string StrCat(string s1, string s2)
```

[解説]

文字列と文字列を連結した文字列を返す。

[引き数]

s1 連結される文字列。
s2 連結する文字列。

[構文]

```
string Mid(string sTarget, int iStart, int iCount)
```

[解説]

文字列から中間部分を抽出した文字列を返す。

[引き数]

sTarget 対象となる文字列。
iStart 抽出する文字列の最初の位置を 0 から始まるインデックスで指定します。
iCount 抽出する文字列の文字数を指定します。

[構文]

```
string Left(string sTarget, int iCount)
```

[解説]

文字列から左側部分を抽出した文字列を返す。

[引き数]

sTarget 対象となる文字列。
iCount 抽出する文字列の文字数を指定します。

[構文]

```
string Right(string sTarget, int iCount)
```

[解説]

文字列から右側部分を抽出した文字列を返す。

[引き数]

sTarget 対象となる文字列。
iCount 抽出する文字列の文字数を指定します。

[構文]

```
point Pp(point pTarget)
```

[解説]

点を返す。

[引き数]

pTarget 点。

[構文]

```
point Pxy(double dX, double dY, double dZ = 0)
```

[解説]

直行座標の点を返す。

[引き数]

dX X 値。

dY Y 値。

dZ Z 値。

[構文]

```
point Pra(double dDist, double dAngle)
```

[解説]

極座標の点を返す。

[引き数]

dDist 原点からの距離。

dAngle 原点からの角度。

[構文]

```
point Ppxy(point pTarget, double dX, double dY)
```

[解説]

直行座標で平行移動した点を返す。

[引き数]

pTarget 平行移動する点。

dX X 軸の移動量。

dY Y 軸の移動量。

[構文]

```
point Ppra(point pTarget, double dDist, double dAngle)
```

[解説]

極座標で平行移動した点を返す。

[引き数]

pTarget 平行移動する点。

dDist 点からの移動量。

dAngle 角度。

[構文]

```
point Ppx(point pTarget)
```

[解説]

X 軸に対称な点を返す。

[引き数]

pTarget 対称移動する点。

[構文]

```
point Ppy(point p)
```

[解説]

Y 軸に対称な点を返す。

[引き数]

pTarget 対称移動する点。

[構文]

```
point Ppo(point pTarget)
```

[解説]

原点に対称な点を返す。

[引き数]

pTarget 対称移動する点。

[構文]

```
point Pppa(point pTarget, point pCenter, double dAngle)
```

[解説]

1 点を中心に回転移動した点を返す。

[引き数]

pTarget 対称移動する点。

pCenter 回転の中心点。

dAngle 回転角度。

[構文]

```
point Ppl(point pTarget, line lBase)
```

[解説]

直線に対称な点を返す。

[引き数]

pTarget	対称移動する点。
lBase	基準となる直線。

[構文]

```
point Ppc(point pPass, circle cTouch, double dModifier)
```

[解説]

1 点を通り円に接する直線との接点を返す。

[引き数]

pPass	直線の通過点。
cTouch	接する円。
dModifier	求める点の位置。(R, L, A, B, T)

[構文]

```
point Pl1(line lTarget1, line lTarget2)
```

[解説]

2 直線の交点を返す。

[引き数]

lTarget1	直線 1。
lTarget2	直線 2。

[構文]

```
point Plc(line lTarget, circle cTarget, double dModifier)
```

[解説]

直線と円の交点を返す。

[引き数]

lTarget	直線。
cTarget	円。
dModifier	求める点の位置。(R, L, A, B, T)

[構文]

```
point Pc(circle cTarget)
```

[解説]

円の中心点を返す。

[引き数]

cTarget	円。
---------	----

[構文]

```
point Pcc(circle cTarget1, circle cTarget2, double dModifier)
```

[解説]

2 円の交点を返す。

[引き数]

cTarget1 円 1。

cTarget2 円 2。

dModifier 求める点の位置。(R, L, A, B, T)

[構文]

```
line Lpc(point pPass, circle cTouch, double dModifier)
```

[解説]

1 点を通り円に接する直線を返す。

[引き数]

pPass 通過点。

cTouch 接する円。

dModifier 接する位置(R, L, A, B, T)。

[構文]

```
line Llx(line lTarget)
```

[解説]

X 軸に対称な直線を返す。

[引き数]

lTarget 対称移動する直線。

[構文]

```
line Lly(line lTarget)
```

[解説]

Y 軸に対称な直線を返す。

[引き数]

lTarget 対称移動する直線。

[構文]

```
line Llo(line lTarget)
```

[解説]

原点に対称な直線を返す。

[引き数]

lTarget 対称移動する直線。

[構文]

```
line L1l(line lTarget, line lBase)
```

[解説]

直線に対称な直線を返す。

[引き数]

lTarget 対称移動する直線。

lBase 基準となる直線。

[構文]

```
line L1ln(line lStart, line lEnd)
```

[解説]

2 直線の 2 等分線となる直線を返す。

[引き数]

lStart 始角となる直線。

lEnd 終角となる直線。

[構文]

```
line Lca(circle cTouch, double dAngle, double dModifier)
```

[解説]

ある角度で円に接する直線を返す。

[引き数]

cTouch 接する円。

dAngle 直線の角度。

dModifier 接する位置(R, L, A, B)

[構文]

```
line Lcc(circle cTouch1, circle cTouch2, double dModifer1, double dModifier2)
```

[解説]

2 円に接する直線を返す。

[引き数]

cTouch1 接する第 1 円。

cTouch2 接する第 2 円。

dModifier1 第 1 円に接する位置 (R, L, A, B)

dModifier2 第 2 円に接する位置 (R, L, A, B)

[構文]

```
line Llpa(line lTarget, point pCenter, double dAngle)
```

[解説]

1 点を中心に回転移動した直線を返す。

[引き数]

lTarget 回転移動する直線。

pCenter 回転の中心点。

dAngle 回転角度。

[構文]

```
line Lx(double dX)
```

[解説]

Y 軸に平行な直線を返す。

[引き数]

dX X 軸との接辺。

[構文]

```
line Ly(double dY)
```

[解説]

X 軸に平行な直線を返す。

[引き数]

dY Y 軸との接辺。

[構文]

```
line Lpa(point pPass, double dAngle)
```

[解説]

1 点を通りある角度の直線を返す。

[引き数]

pPass 直線の通過点。

dAngle 直線の角度。

[構文]

```
line Lpp(point pPass1, point pPass2)
```

[解説]

2 点を通る直線を返す。

[引き数]

pPass1 直線の通過点 1。

pPass2 直線の通過点 2。

[構文]

```
line Lppn(point pPass1, point pPass2)
```

[解説]

2 点を通る直線の垂直 2 等分線を返す。

[引き数]

pPass1 直線の通過点 1。

pPass2 直線の通過点 2。

[構文]

```
line Lpl(point pPass, line lTarget)
```

[解説]

1 点を通り 1 直線に平行な直線を返す。

[引き数]

pPass 求める直線の通過点。

lTarget 基準となる直線。

[構文]

```
line Lpln(point pPass, line lTarget)
```

[解説]

1 点を通り 1 直線に垂直な直線を返す。

[引き数]

pPass 求める直線の通過点。

lTarget 基準となる直線。

[構文]

```
line Llr(line lTarget, double dDist, double dModifier)
```

[解説]

平行移動した直線を返す。

[引き数]

lTarget 基準となる直線。

dDist 平行移動する距離。
dModifier 平行移動する方向。(R, L, A, B)

[構文]

```
line Lccn(circle cTarget1, circle cTarget2)
```

[解説]

2 円の交点を通る直線を返す。

[引き数]

cTarget1 円 1。
cTarget2 円 2。

[構文]

```
circle Cxyc(double dX, double dY, double dR)
```

[解説]

中心と半径から円を返す。

[引き数]

dX 中心の X 値。
dY 中心の Y 値。
dR 円の半径。

[構文]

```
circle Cpr(point pCenter, double dR)
```

[解説]

中心と半径から円を返す。

[引き数]

pCenter 中心点。
dR 円の半径。

[構文]

```
circle Cpc(point pCenter, circle cTouch, double dModifier)
```

[解説]

中心と 1 円に接する円を返す。

[引き数]

pCenter 中心点。
cTouch 接する円。
dModifier 求める円の大小(S, L)

[構文]

```
circle Ccxy(circle cTarget, double dX, double dY)
```

[解説]

直行座標で平行移動した円を返す。

[引き数]

cTarget 平行移動する円。

dX X 軸の移動量。

dY Y 軸の移動量。

[構文]

```
circle Ccra(circle cTarget, double dDist, double dAngle)
```

[解説]

極座標で平行移動した円を返す。

[引き数]

cTarget 平行移動する円。

dDist 移動距離。

dAngle 移動角度。

[構文]

```
circle Ccl(circle cTarget, line lBase)
```

[解説]

線対称な円を返す。

[引き数]

cTarget 対称移動する円。

lBase 基準となる直線。

[構文]

```
circle Ccpa(circle cTarget, point pCenter, double dAngle)
```

[解説]

1 点を中心に回転移動した円を返す。

[引き数]

cTarget 回転移動する円。

pCenter 回転の中心点。

dAngle 回転角度。

[構文]

```
circle Ccx(circle cTarget)
```

[解説]

X 軸に対称な円を返す。

[引き数]

cTarget 対称移動する円。

[構文]

```
circle Ccy(circle cTarget)
```

[解説]

Y 軸に対称な円を返す。

[引き数]

cTarget 対称移動する円。

[構文]

```
circle Cco(circle cTarget)
```

[解説]

原点に対称な円を返す。

[引き数]

cTarget 対称移動する円。

[構文]

```
circle Ccd(circle cTarget, double dD, double dModifier)
```

[解説]

半径差から同心円を返す。

[引き数]

cTarget 基準となる円。

dD 半径差。

dModifier 求める円の大小(S, L)

[構文]

```
circle Ccr(circle cTarget, double dR)
```

[解説]

半径から同心円を返す。

[引き数]

cTarget 基準となる円。

dR 半径。

[構文]

```
circle C111(line lTouch1, line lTouch2, line lTouch3, double dModifier1, double  
          Modifier2, double dModifier3)
```

[解説]

3 直線に接する円を返す。

[引き数]

lTouch1	接する直線 1。
lTouch2	接する直線 2。
lTouch3	接する直線 3。
dModifier1	直線 1 から見た求める円の位置。(R, L, A, B)
dModifier2	直線 2 から見た求める円の位置。(R, L, A, B)
dModifier3	直線 3 から見た求める円の位置。(R, L, A, B)

[構文]

```
circle Cpp(point pCenter, point pPass)
```

[解説]

中心と通過点から円を返す。

[引き数]

pCenter	中心点。
pPass	通過点。

[構文]

```
circle Cppr(point pPass1, point pPass2, double dR, double dModifier)
```

[解説]

2 点の通過点と半径から円を返す。

[引き数]

pPass1	通過点 1。
pPass2	通過点 2。
dR	求める円の半径。
dModifier	求める円の位置。(R, L, A, B, T)

[構文]

```
circle Cpl(point pCenter, line lTouch)
```

[解説]

中心と直線に接する円を返す。

[引き数]

pCenter	中心点。
lTouch	接する直線。

[構文]

```
circle Cplr(point pPass, line lTouch, double dR, double dModifier)
```

[解説]

通過点と直線に接する半径指定の円を返す。

[引き数]

pPass	通過点。
lTouch	接する直線。
dR	求める円の半径。
dModifier	求める円の位置。(R, L, A, B, T)

[構文]

```
circle Cpcr(point pPass, circle cTouch, double dR, double dModifier1, double  
dModifier2)
```

[解説]

通過点と円に接する半径指定の円を返す。

[引き数]

pPass	通過点。
cTouch	接する円。
dR	求める円の半径。
dModifier1	求める円の接し方。(I, O)
dModifier2	求める円の位置。(R, L, A, B, T)

[構文]

```
circle Cllr(line lTarget1, line lTarget2, double dR, double dModifier1, double  
dModifier2)
```

[解説]

2 直線に接する半径指定の円を返す。

[引き数]

lTarget1	接する直線 1。
lTarget2	接する直線 2。
dR	求める円の半径。

dModifier1 直線 1 から見た円の位置。(R, L, A, B)

dModifier2 直線 2 から見た円の位置。(R, L, A, B)

[構文]

```
circle Cclr(line lTarget, circle cTarget, double dR, double dModifier1, double  
            dModifier2, double dModifier3)
```

[解説]

直線と円に接する半径指定の円を返す。

[引き数]

lTarget 接する直線。

cTarget 接する円。

dR 求める円の半径。

dModifier1 直線から見た円の位置。(R, L, A, B)

dModifier2 接する円と求める円の接し方。(I, 0)

dModifier3 求める円の位置。(R, L, A, B, T)

[構文]

```
circle Cccr(circle cTarget1, circle cTarget2, double dR, double dModifier1,  
            double dModifier2, double dModifier3)
```

[解説]

2 円に接する半径指定の円を返す。

[引き数]

cTarget1 接する円 1。

cTarget2 接する円 2。

dR 求める円の半径。

dModifier1 円 1 と求める円の接し方。(I, 0)

dModifier2 円 2 と求める円の接し方。(I, 0)

dModifier3 求める円の位置。(R, L, A, B, T)

[構文]

```
circle Cppp(point pPass1, point pPass2, point pPass3)
```

[解説]

3 点を通る円を返す。

[引き数]

pPass1 求める円の通過点 1。

pPass2 求める円の通過点 2。

pPass3 求める円の通過点 3。

[構文]

```
circle Ctpl(point pPass1, point pPass2, line lTouch, double dModifier)
```

[解説]

2 点を通り 1 直線に接する円を返す。

[引き数]

pPass1 通過点 1。
pPass2 通過点 2。
lTouch 接する直線。
dModifier 求める円の位置。(R, L, A, B)

[構文]

```
circle Cppc(point pPass1, point pPass2, circle cTouch, double dModifier1, double  
            dModifier2)
```

[解説]

2 点を通り 1 円に接する円を返す。

[引き数]

pPass1 通過点 1。
pPass2 通過点 2。
cTouch 接する円。
dModifier1 円の接し方。(I, 0)
dModifier2 求める円の位置。(R, L, A, B)

[構文]

```
circle Cplc(point pPass, line lTouch, circle cTouch, double dModifier1, double  
            dModifier2)
```

[解説]

1 点を通り 1 直線と 1 円に接する円を返す。

[引き数]

pPass 通過点。
lTouch 接する直線
cTouch 接する円。
dModifier1 円の接し方。(I, 0)
dModifier2 求める円の位置。(R, L, A, B)

[構文]

```
circle Cpcc(point pPass, circle cTouch1, circle cTouch2, double dModifier1,  
            double dModifier2, double dModifier3)
```

[解説]

1 点を通り 2 円に接する円を返す。

[引き数]

pPass 通過点。
cTouch1 接する円 1。
cTouch2 接する円 2。
dModifier1 円 1 の接し方。(I, 0)
dModifier2 円 2 の接し方。(I, 0)
dModifier3 求める円の位置。(R, L, A, B)

[構文]

```
circle Cllc(line lTouch1, line lTouch2, circle cTouch, double dModifier1, double  
            dModifier2, double dModifier3, double dModifier4)
```

[解説]

2 直線と 1 円に接する円を返す。

[引き数]

lTouch1 接する直線 1。
lTouch2 接する直線 2。
cTouch 接する円。
dModifier1 直線 1 から見た求める円の位置。(R, L, A, B)
dModifier2 直線 2 から見た求める円の位置。(R, L, A, B)
dModifier3 円の接し方。(I, 0)
dModifier4 求める円の位置。(R, L, A, B)

[構文]

```
circle Clcc(line lTouch, circle cTouch1, circle cTouch2, double dModifier1,  
            double dModifier2, double dModifier3, double dModifier4)
```

[解説]

1 直線と 2 円に接する円を返す。

[引き数]

lTouch 接する直線。
cTouch1 接する円 1。
cTouch2 接する円 2。

dModifier1 直線から見た求める円の位置。(R, L, A, B)
dModifier2 円 1 の接し方。(I, 0)
dModifier3 円 2 の接し方。(I, 0)
dModifier4 求める円の位置。(R, L, A, B)

[構文]

```
circle Cccc(circle cTouch1, circle cTouch2, circle cTouch3, double dModifier1,  
             double dModifier2, double dModifier3, double dModifier4)
```

[解説]

3 円に接する円を返す。dModifier1, dModifier2, dModifier3 が全て I で全ての円が交わる時、dModifier4 は S, L で指定する。

[引き数]

cTouch1 接する円 1。
cTouch2 接する円 2。
cTouch3 接する円 3。
dModifier1 円 1 の接し方。(I, 0)
dModifier2 円 2 の接し方。(I, 0)
dModifier3 円 3 の接し方。(I, 0)
dModifier4 求める円の位置。(R, L, A, B, S, L)

[構文]

```
circle Cpll(point pPass, line lTouch1, line lTouch2, double dModifier)
```

[解説]

1 点を通り 2 直線に接する円を返す。

[引き数]

pPass 通過点。
lTouch1 接する直線 1。
lTouch2 接する直線 2。
dModifier 求める円の位置。(R, L, A, B)

[構文]

```
double Dpp(point p1, point p2)
```

[解説]

2 点の距離を返す。

[引き数]

p1 点 1。

p2 点 2。

[構文]

```
double Dpl(point p, line l)
```

[解説]

点と直線の距離を返す。

[引き数]

p 点。

l 直線。

[構文]

```
double App(point pCenter, point pEnd)
```

[解説]

2 点のなす角を $0^{\circ} \sim 360^{\circ}$ で返す。

[引き数]

pCenter 中心点。

pEnd 終了角となる点。

データベース操作系マクロ

[構文]

```
object SetPoint(point p)
```

```
object SetPoint(double x, double y, double z = 0)
```

[解説]

点を登録して点オブジェクトを返す。

[引き数]

p 点。

x X 値。

y Y 値。

z Z 値。

[構文]

```
object SetLine(line l, double z = 0)
```

```
object SetLine(point p1, point p2)
```

```
object SetLine(double x1, double y1, double x2, double y2)
```

```
object SetLine(double x1, double y1, double z1, double x2, double y2, double z2)
```

[解説]

直線を登録して直線オブジェクトを返す。

[引き数]

l	直線。
z	Z 値。
p1	始点。
p2	終点。
x1	始点の X 値。
y1	始点の Y 値。
z1	始点の Z 値。
x2	終点の X 値。
y2	終点の Y 値。
z2	終点の Z 値。

[構文]

```
object SetArc(circle c, double z = 0)
object SetArc(circle c, double sa, double ca, double z = 0)
object SetArc(double x, double y, double r)
object SetArc(double x, double y, double z, double r)
object SetArc(double x, double y, double r, double sa, double ca)
object SetArc(double x, double y, double z, double r, double sa, double ca)
object SetArc(double dVector, point pStart, point pEnd, circle c, double z = 0)
```

[解説]

円弧を登録して円弧オブジェクトを返す。

[引き数]

c	円。
x	中心点の X 値。
y	中心点の Y 値。
z	中心点の Z 値。
r	円の半径。
sa	円弧の開始角度。
ca	円弧の回転角度。
dVector	円弧の回転方向。(CW, CCW)
pStart	円弧上の始点。
pEnd	円弧上の終点。

[構文]

```
object SetText(double x, double y, string sText)
object SetText(double x, double y, double z, string sText)
```

[解説]

文字列を登録して文字列オブジェクトを返す。

[引き数]

x	登録の基準となる X 値。
y	登録の基準となる Y 値。
z	登録の基準となる Z 値。
sText	登録する文字列。

[構文]

```
object SetNurbs(point pt1, point pt2, . . . )
```

[解説]

曲線を登録して曲線オブジェクトを返す。

引き数の個数は 2 ～ 10 点まで。10 点以上は SetNurbsPassPoint と併用し

引数を空にして SetNurbs(空) を実行する

[引き数]

pt1、pt2 曲線の通過点。

[構文]

```
void SetNurbsPassPoint(point pt1)
```

[解説]

曲線の通過点を登録します。

[引き数]

pt1 曲線の通過点。

[構文]

```
object SetLineMsr(int iType, point p1, point p2, point pVal)
```

[解説]

直線寸法を登録して直線寸法要素を返す。

[引き数]

iType 寸法のタイプを指定します。この値は次のいずれかになります。

値	意味
SLM_HORZ	水平寸法
SLM_VERT	垂直寸法

SLM_PARA 平行寸法

p1, p2 引き出し線の頂点を指定します。
 pVal 寸法線上の寸法値位置を指定します。

[構文]

```
object SetArcMsr(int iType, circle cBase, point pPassOn, point pVal)
object SetArcMsr(int iType, circle cBase, point pStart, point pRef, point pVal)
```

[解説]

円寸法を登録して円寸法要素を返す。2つ目の関数では、フリー寸法を表します。

[引き数]

iType 寸法のタイプを指定します。この値は次のいずれかになります。

値	意味
SAM_RAD	半径寸法
SAM_DIA	直径寸法

cBase 基準となる円を指定します。
 pPassOn 基準円上の寸法通過点を指定します。この点と基準円の中心点を結ぶ角度が 円寸法の角度になります。
 pVal 寸法値の位置を指定します。
 pStart 基準円上の開始位置を指定します。
 pRef 屈折位置を指定します。

[構文]

```
object SetAngMsr(point pStart, point pCenter, point pLast, point pVect, point
pVal)
```

[解説]

角度寸法を登録して角度寸法要素を返す。

[引き数]

pStart 開始点を指定します。
 pCenter 角度の中心点を指定します。
 pLast 終了点を指定します。
 pVect 回転方向を指定します。
 pVal 寸法値の位置を指定します。

[構文]

```
object SetPosMsr(point pStart, point pLast, point pBarPos, point pBarAng)
```

[解説]

位置寸法を登録して位置寸法要素を返す。pStart の点から基準十字線の pLast が近い方の線へ基準線上まで伸びます。pLast の点は基準線上の点に自動的に乗ります。また、寸法線は必要に応じて自動的に屈折します。

[引き数]

pStart	開始点を指定します。
pLast	終了点を指定します。
pBarPos	基準十字線の中心位置を指定します。
dBarAng	基準十字線の角度を指定します。

[構文]

```
void Delete(object obj)
```

[解説]

登録要素をデータベースから削除する。

[引き数]

obj	削除する要素。
-----	---------

[構文]

```
void Undo(int iCount = 1)
```

[解説]

アンドゥを実行。

[引き数]

iCount	アンドゥを行う回数。
--------	------------

[構文]

```
void Redo(int iCount = 1)
```

[解説]

リドゥを実行。

[引き数]

iCount	リドゥを行う回数。
--------	-----------

属性変更系マクロ

[構文]

```
void Color(int col)
```


[解説]

色を変更する。

[引き数]

col 新しい色。この値は次のいずれかになります。

値	色
CL_BLACK	黒
CL_BLUE	青
CL_GREEN	緑
CL_CYAN	水色
CL_RED	赤
CL_MAZENDA	紫
CL_YELLOW	黄色
CL_WHITE	白
CL_LT_BLUE	淡い青
CL_LT_GREEN	淡い緑
CL_LT_CYAN	淡い水色
CL_LT_RED	淡い赤
CL_LT_MAZENDA	淡い紫
CL_LT_YELLOW	淡い黄色
CL_LT_WHITE	淡い白

[構文]

```
void LType(int type)
```

[解説]

線種を変更する。

[引き数]

type 新しい線種。この値は次のいずれかになります。

値	意味
LT_SOLID	実線
LT_DASH	破線
LT_DOT	点線
LT_DASHDOT	一点鎖線
LT_DASHDOTDOT	二点鎖線

[構文]

```
void LWidth(int width)
```

[解説]

線幅を変更する。

[引き数]

width 新しい線幅。この値は次のいずれかになります。

値	意味
LW_1	1 ピクセルの幅
LW_2	2 ピクセルの幅
LW_3	3 ピクセルの幅
LW_4	4 ピクセルの幅
LW_5	5 ピクセルの幅
LW_6	6 ピクセルの幅
LW_7	7 ピクセルの幅
LW_8	8 ピクセルの幅
LW_9	9 ピクセルの幅

フォント属性変更系マクロ

[構文]

```
void FontType(int type)
```

[解説]

フォントのタイプを変更する。

[引き数]

type フォントのタイプを指定します。この値は次のいずれかになります。

値	意味
FT_ORIGINAL	オリジナルフォント
FT_TRUETYPE	TrueType フォント
FT_TTOUTLINE	TrueType のアウトラインフォント

[構文]

```
void FontSelect(string sFontName)
```

[解説]

TrueType フォントの種類を変更する。

[引き数]

sFontName TrueType フォントの名前を指定します。

[構文]

```
void FontSize(double width, double height)
void FontSize(double width, double height, double space)
```

[解説]

フォントの大きさを変更する。

[引き数]

width	1 文字の幅。
height	1 文字の高さ。
space	文字間隔。

[構文]

```
void FontAngle(double angle)
```

[解説]

フォントの角度を変更する。SetText() の基準点に対して、angle 度回した角度になります。左回りは正の値を、右回りは負の値を指定します。

[引き数]

angle	文字列の角度。
-------	---------

[構文]

```
void FontAlign(int horz, int vert)
```

[解説]

フォントの位置合わせを変更する。SetText() の基準点に対して、位置合わせを行います。

[引き数]

horz	水平方向の位置合わせ。この値は次のいずれかになります。
------	-----------------------------

値	意味
FA_LEFT	左位置合わせ
FA_CENTER	中央位置合わせ
FA_RIGHT	右位置合わせ

vert	垂直方向の位置合わせ。この値は次のいずれかになります。
------	-----------------------------

値	意味
---	----

FA_TOP	上位置合わせ
FA_CENTER	中央位置合わせ
FA_BOTTOM	下位置合わせ

プログラム制御系マクロ

[構文]

```
void Sleep(double sec)
```

[解説]

スレッドを一定時間停止させます。

[引き数]

sec 停止させる秒数。

[構文]

```
void if(bool fFlag)
```

[解説]

プログラムの条件判定を行います。fFlag が真であれば、対応する endif までは実行され、偽であれば、対応する endif まで無視します。必ず endif とペアで使用しなければいけません。

[引き数]

fFlag 真偽。

[構文]

```
void endif()
```

[解説]

プログラムの条件判定の終了位置をあらわします。必ず if とペアで使用しなければいけません。

[構文]

```
void while(bool fFlag)
```

[解説]

プログラムの繰り返し処理を行います。fFlag が真であれば、対応する endwhile までは実行されて while まで戻ります。これを fFlag が偽になるまで繰り返されます。必ず endwhile とペアで使用しなければいけません。

[引き数]

fFlag 真偽。

[構文]

```
void endwhile()
```

[解説]

プログラムの繰り返し処理の終了位置をあらわします。必ず `while` とペアで使用しなければいけません。

[構文]

```
double Input(string sMessage)
```

```
string Input(string sMessage)
```

[解説]

キーボードからの入力を要求します。ファイルから実行されているときにはプログラムは一旦停止し、入力が終わったら (`E n t e r` キーが押されたら) 再び実行されます。このマクロは実数と文字列の入力を受け付けます。実数が入力された場合は `double` が返し、文字列が入力された場合は `string` が返ります。文字列の入力は “” (ダブルクォーテーション) で囲みます。実数の入力の場合には変数が使用でき、四則演算も使用可能です。

[引き数]

`sMessage` キーボード入力を要求するために表示するメッセージ。

[構文]

```
double InputV(string sMessage)
```

[解説]

キーボードからの入力を要求します。ファイルから実行されているときにはプログラムは一旦停止し、入力が終わったら (`E n t e r` キーが押されたら) 再び実行されます。このマクロは実数の入力しか受け付けません。入力された値 (四則演算した結果) を返します。

[引き数]

`sMessage` キーボード入力を要求するために表示するメッセージ。

[構文]

```
string InputS(string sMessage)
```

[解説]

キーボードからの入力を要求します。ファイルから実行されているときにはプログラムは一旦停止し、入力が終わったら (`E n t e r` キーが押されたら) 再び実行されます。このマクロは文字列の入力しか受け付けません。よって、文字列を “” (ダブルクォーテーション) で囲む必要はありません。入力された文字列を返します。

[引き数]

sMessage キーボード入力を要求するために表示するメッセージ。

[構文]

```
void End()
```

[解説]

現在実行中のプログラムを終了します。